IDEFIX: USING MULTIFLUIDS AND PASSIVE TRACERS

https://idefix.readthedocs.io/latest/modules/dust.html

IPAG

1

MULTIFLUIDS

Introduction & motivation

AIM: study the evolution of multiple dust species in a gaseous environment.

1

MULTIFLUIDS

Introduction & motivation

AIM: study the evolution of multiple dust species in a gaseous environment.

APPROACHES: Lagrangian particles or pressureless fluids.

1

MULTIFLUIDS

Introduction & motivation

AIM: study the evolution of multiple dust species in a gaseous environment.

APPROACHES: Lagrangian particles or pressureless fluids.

METHOD: solve the continuity and momentum equations, the same way as the gas.

$$\frac{\partial \rho_{\mathrm{i}}}{\partial t} + \vec{\nabla} \cdot \rho_{\mathrm{i}} \vec{\mathrm{v}}_{\mathrm{i}} = 0$$

$$\frac{\partial (\rho_{\mathrm{i}} \vec{\mathrm{v}}_{\mathrm{i}})}{\partial t} + \vec{\nabla} \cdot \rho_{\mathrm{i}} \vec{\mathrm{v}}_{\mathrm{i}} \otimes \vec{\mathrm{v}}_{\mathrm{i}} = -\rho_{\mathrm{i}} \vec{\nabla} \Psi - \vec{\nabla} \mathrm{P}$$

MULTIFLUIDS

Introduction & motivation

AIM: study the evolution of multiple dust species in a gaseous environment.

APPROACHES: Lagrangian particles or pressureless fluids.

METHOD: solve the continuity and momentum equations, the same way as the gas.

BUT: the fluid is **pressureless** and subject to an **aerodynamic drag**.

$$\frac{\partial \rho_\mathrm{i}}{\partial t} + \vec{\nabla} \cdot \rho_\mathrm{i} \vec{\mathrm{v}_\mathrm{i}} = 0$$

$$\frac{\partial (\rho_\mathrm{i} \vec{\mathrm{v}_\mathrm{i}})}{\partial t} + \vec{\nabla} \cdot \rho_\mathrm{i} \vec{\mathrm{v}_\mathrm{i}} \otimes \vec{\mathrm{v}_\mathrm{i}} = -\rho_\mathrm{i} \vec{\nabla} \Psi - \cancel{\vec{\nabla} P} + \underbrace{\gamma_\mathrm{i} \rho_\mathrm{i} \rho (\vec{\mathrm{v}} - \vec{\mathrm{v}_\mathrm{i}})}_{\text{Aerodynamic drag force}}$$

MULTIFLUIDS

Introduction & motivation

AIM: study the evolution of multiple dust species in a gaseous environment.

APPROACHES: Lagrangian particles or pressureless fluids.

METHOD: solve the continuity and momentum equations, the same way as the gas.

BUT: the fluid is **pressureless** and subject to an **aerodynamic drag**.

APPLICATION: gas/dust interactions in a circumstellar disk, varying the dust size.

$$\frac{\partial \rho_i}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i = 0$$

$$\frac{\partial (\rho_i \vec{v}_i)}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i \otimes \vec{v}_i = -\rho_i \vec{\nabla} \Psi - \cancel{\vec{\nabla} P} + \underbrace{\gamma_i \rho_i \rho (\vec{v} - \vec{v}_i)}_{\text{Aerodynamic drag force}}$$

MULTIFLUIDS

Introduction & motivation

AIM: study the evolution of multiple dust species in a gaseous environment.

APPROACHES: Lagrangian particles or pressureless fluids.

METHOD: solve the continuity and momentum equations, the same way as the gas.

BUT: the fluid is **pressureless** and subject to an **aerodynamic drag**.

APPLICATION: gas/dust interactions in a circumstellar disk, varying the dust size.

$$\frac{\partial \rho_i}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i = 0$$

$$\frac{\partial (\rho_i \vec{v}_i)}{\partial t} + \vec{\nabla} \cdot \rho_i \vec{v}_i \otimes \vec{v}_i = -\rho_i \vec{\nabla} \Psi - \cancel{\vec{\nabla} P} + \underbrace{\gamma_i \rho_i \rho (\vec{v} - \vec{v}_i)}_{\text{Aerodynamic drag force}}$$

NOTE: possible to test the non-ideal MHD framework with a bifluid approach

- one fluid of ions/electrons globally neutral (MHD fluid)
- one fluid of neutral particles (purely hydro fluid).

MULTIFLUIDS

Implementation in Idefix

We make use of the template class `Fluid<Phys>`, described by the template parameter `Phys`. `Phys` is `DefaultPhysics` for a gas fluid and `DustPhysics` for a dust fluid.

MULTIFLUIDS

Implementation in Idefix

We make use of the template class `Fluid<Phys>`, described by the template parameter `Phys`. `Phys` is `DefaultPhysics` for a gas fluid and `DustPhysics` for a dust fluid.

```
struct DustPhysics {
  static constexpr bool dust{true};
  static constexpr bool pressure{false};
  static constexpr bool isothermal{false};
  static constexpr bool eos{false};

  static constexpr bool mhd{false};
  static constexpr int nvar{1+COMPONENTS};

  // prefix
  static constexpr std::string_view prefix = "Dust";
};
```

MULTIFLUIDS

Implementation in Idefix

We make use of the template class `Fluid<Phys>`, described by the template parameter `Phys`. `Phys` is `DefaultPhysics` for a gas fluid and `DustPhysics` for a dust fluid.

```
struct DustPhysics {
  static constexpr bool dust{true};
  static constexpr bool pressure{false};
  static constexpr bool isothermal{false};
  static constexpr bool eos{false};

  static constexpr bool mhd{false};
  static constexpr int nvar{1+COMPONENTS};

  // prefix
  static constexpr std::string_view prefix = "Dust";
};
```

Note that *Idefix* defines an alias for the default fluid which is often found in the provided examples:

```
using Hydro = Fluid<DefaultPhysics>;
```

PASSIVE TRACERS

Implementation in Idefix

A passive tracer/scalar allows us to trace how much material of the attached fluid is advected during the simulation, given an initial condition. Idefix can follow such passive tracer by solving the incompressible continuity equation for the scalar $\mathcal{T}$, with $\vec{v}$ being the velocity of the corresponding fluid.

Advection equation $$\frac{\partial \mathcal{T}}{\partial t} + \vec{v} \cdot \vec{\nabla} \mathcal{T} = 0$$

Idefix supports an arbitrary number of tracers per fluid. For example, you can divide radially your domain into N tracers for the gas fluid and check how the tracers are advected during your simulation

4

EXAMPLE OF SETUP

Model: Planet+Dust+Tracers

SETUP: similar to `$IDEFIX_DIR/test/Dust/FargoPlanet`.

4

EXAMPLE OF SETUP

Model: Planet+Dust+Tracers

SETUP: similar to `$IDEFIX_DIR/test/Dust/FargoPlanet`.

GEOMETRY: polar coordinates (x1 = R , x2 = ϕ)

4

EXAMPLE OF SETUP

Model: Planet+Dust+Tracers

SETUP: similar to `$IDEFIX_DIR/test/Dust/FargoPlanet`.

GEOMETRY: polar coordinates (x1 = R , x2 = ϕ)

SYSTEM:

- 2D Keplerian thin disk: $\Sigma_g \propto R^{-1}, \quad h_0 = 0.05$

4

EXAMPLE OF SETUP

Model: Planet+Dust+Tracers

SETUP: similar to `$IDEFIX_DIR/test/Dust/FargoPlanet`.

GEOMETRY: polar coordinates (x1 = R , x2 = ϕ)

SYSTEM:

- 2D Keplerian thin disk: $\Sigma_g \propto \mathrm{R}^{-1}, \;\; h_0 = 0.05$
- locally isothermal e.o.s.: fixed $c_s = h_0 v_\mathrm{K}$

4

EXAMPLE OF SETUP

Model: Planet+Dust+Tracers

SETUP: similar to `$IDEFIX_DIR/test/Dust/FargoPlanet`.

GEOMETRY: polar coordinates (x1 = R , x2 = ϕ)

SYSTEM:

- 2D Keplerian thin disk: $\Sigma_{\rm g} \propto {\rm R}^{-1}$, $h_0 = 0.05$
- locally isothermal e.o.s.: fixed $c_s = h_0 v_{\rm K}$
- 1 Jupiter-mass planet in a fixed circular orbit: $q_{\rm p} = {\rm M_p}/{\rm M_\star} = 10^{-3}$, $R_{\rm p} = 1$

4

EXAMPLE OF SETUP

Model: Planet+Dust+Tracers

SETUP: similar to `$IDEFIX_DIR/test/Dust/FargoPlanet`.

GEOMETRY: polar coordinates (x1 = R , x2 = ϕ)

SYSTEM:

- 2D Keplerian thin disk: $\Sigma_{\rm g} \propto {\rm R}^{-1}$, $h_0 = 0.05$
- locally isothermal e.o.s.: fixed $c_s = h_0 v_{\rm K}$
- 1 Jupiter-mass planet in a fixed circular orbit: $q_{\rm p} = {\rm M_p}/{\rm M_\star} = 10^{-3}$, $R_{\rm p} = 1$
- 1 dust fluid: $\tau_{\rm stop} = 1$ (see Thomas' section), $\Sigma_{\rm d}/\Sigma_{\rm g} = 10^{-2}$

4

EXAMPLE OF SETUP

Model: Planet+Dust+Tracers

SETUP: similar to `$IDEFIX_DIR/test/Dust/FargoPlanet`.

GEOMETRY: polar coordinates (x1 = R , x2 = ϕ)

SYSTEM:

- 2D Keplerian thin disk: $\Sigma_{\rm g} \propto {\rm R}^{-1}$, $h_0 = 0.05$
- locally isothermal e.o.s.: fixed $c_s = h_0 v_{\rm K}$
- 1 Jupiter-mass planet in a fixed circular orbit: $q_{\rm p} = {\rm M_p}/{\rm M_\star} = 10^{-3}$, $R_{\rm p} = 1$
- 1 dust fluid: $\tau_{\rm stop} = 1$ (see Thomas' section), $\Sigma_{\rm d}/\Sigma_{\rm g} = 10^{-2}$
- 2 gas tracers

4

EXAMPLE OF SETUP

Model: Planet+Dust+Tracers

SETUP: similar to `$IDEFIX_DIR/test/Dust/FargoPlanet`.

GEOMETRY: polar coordinates (x1 = R , x2 = ϕ)

SYSTEM:

- 2D Keplerian thin disk: $\Sigma_{\rm g} \propto {\rm R}^{-1}$, $h_0 = 0.05$
- locally isothermal e.o.s.: fixed $c_s = h_0 v_{\rm K}$
- 1 Jupiter-mass planet in a fixed circular orbit: $q_{\rm p} = {\rm M_p}/{\rm M_\star} = 10^{-3}$, $R_{\rm p} = 1$
- 1 dust fluid: $\tau_{\rm stop} = 1$ (see Thomas' section), $\Sigma_{\rm d}/\Sigma_{\rm g} = 10^{-2}$
- 2 gas tracers
- 2 dust tracers

EXAMPLE OF SETUP

Using Idefix

idefix.ini

4

EXAMPLE OF SETUP

Using Idefix

definitions.hpp

idefix.ini

EXAMPLE OF SETUP

Using Idefix

definitions.hpp

setup.cpp

idefix.ini

EXAMPLE OF SETUP

Using Idefix

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

idefix.ini

EXAMPLE OF SETUP

Using Idefix

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1
```

EXAMPLE OF SETUP

Using Idefix

setup.cpp

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1


[Gravity]
potential    central  planet
Mcentral     1.0
```

EXAMPLE OF SETUP

Using Idefix

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1


[Gravity]
potential    central  planet
Mcentral     1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0
```

EXAMPLE OF SETUP

Using Idefix

setup.cpp

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef



[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0



[Setup]
h0             0.05
```

EXAMPLE OF SETUP

Using Idefix

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver       hllc
csiso        userdef



[Gravity]
potential    central  planet
Mcentral     1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0



[Setup]
h0             0.05
```

4

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver       hllc
csiso        userdef



[Gravity]
potential    central  planet
Mcentral     1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0



[Setup]
h0             0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}
```

```
// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{


  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0



[Setup]
h0             0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}
```

```
// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{


  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0


[Setup]
h0              0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}
```

```
// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{


  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef


[Dust]
nSpecies 1
drag    tau  1.0



[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0









[Setup]
h0              0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}
```

```
// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{


  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);
```

```
            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1

[Hydro]
solver          hllc
csiso           userdef

[Dust]
nSpecies 1
drag    tau  1.0

[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Setup]
h0              0.05
```

⚠️ drag_feedback = true by default

→ see Thomas' presentation

EXAMPLE OF SETUP

Using Idefix

Dust parameters

https://idefix.readthedocs.io/latest/modules/dust.html

The dust module can be enabled adding a block *[Dust]* in your input .ini file. The parameters are as follow:

Entry name	Parameter type	Comment
nSpecies	integer	Number of dust species to solve
drag	string, float, ...	The first parameter describe the drag type. Possible values are: `gamma`, `tau`, `size` and `userdef`. The remaining parameters gives the drag parameter β_i for each dust specie. (see below). *Idefix* expect to have as many drag parameters as there are dust species.
drag_feedback	bool	(optionnal) whether the gas feedback is enabled (default true).

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}
```

```
// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{


  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef


[Dust]
nSpecies 1
drag    tau  1.0



[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0









[Setup]
h0              0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}
```

```
// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{


  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);









                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef


[Dust]
nSpecies 1
drag    tau  1.0


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0



[Setup]
h0              0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}
```

```
// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{


  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);









                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef


[Dust]
nSpecies 1
drag   tau  1.0


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg     userdef
X1-end     userdef
X2-beg     periodic
X2-end     periodic
X3-beg     outflow
X3-end     outflow

[Setup]
h0             0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}

// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{


  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);









                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef


[Dust]
nSpecies 1
drag    tau  1.0


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg    userdef
X1-end    userdef
X2-beg    periodic
X2-end    periodic
X3-beg    outflow
X3-end    outflow

[Setup]
h0             0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}

// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  data.hydro->EnrollUserDefBoundary(&BCGas);

  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);









                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef


[Dust]
nSpecies 1
drag    tau  1.0


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg     userdef
X1-end     userdef
X2-beg     periodic
X2-end     periodic
X3-beg     outflow
X3-end     outflow

[Setup]
h0             0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}

// Dust User-defined boundaries
void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = dust->Vc;
    IdefixArray1D<real> x1 = dust->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = dust->data->beg[IDIR];
    } else if (side==right) {
      iref = dust->data->end[IDIR]-1;
    }
    dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}
```

```
// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  data.hydro->EnrollUserDefBoundary(&BCGas);

  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);









                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef


[Dust]
nSpecies 1
drag    tau  1.0


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg     userdef
X1-end     userdef
X2-beg     periodic
X2-end     periodic
X3-beg     outflow
X3-end     outflow

[Setup]
h0              0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}

// Dust User-defined boundaries
void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = dust->Vc;
    IdefixArray1D<real> x1 = dust->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = dust->data->beg[IDIR];
    } else if (side==right) {
      iref = dust->data->end[IDIR]-1;
    }
    dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}
```

```
// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  data.hydro->EnrollUserDefBoundary(&BCGas);
  data.dust[0]->EnrollUserDefBoundary(&BCDust);
  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);









                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef


[Dust]
nSpecies 1
drag    tau  1.0


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg     userdef
X1-end     userdef
X2-beg     periodic
X2-end     periodic
X3-beg     outflow
X3-end     outflow

[Setup]
h0              0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,jref);
        Vc(VX1,k,j,i) = Vc(VX1,k,jref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}

// Dust User-defined boundaries
void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = dust->Vc;
    IdefixArray1D<real> x1 = dust->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = dust->data->beg[IDIR];
    } else if (side==right) {
      iref = dust->data->end[IDIR]-1;
    }
    dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,jref);
        Vc(VX1,k,j,i) = Vc(VX1,k,jref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}

// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  data.hydro->EnrollUserDefBoundary(&BCGas);
  data.dust[0]->EnrollUserDefBoundary(&BCDust);
  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);









                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef
tracer 2

[Dust]
nSpecies 1
drag   tau  1.0


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg     userdef
X1-end     userdef
X2-beg     periodic
X2-end     periodic
X3-beg     outflow
X3-end     outflow

[Setup]
h0              0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}

// Dust User-defined boundaries
void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = dust->Vc;
    IdefixArray1D<real> x1 = dust->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = dust->data->beg[IDIR];
    } else if (side==right) {
      iref = dust->data->end[IDIR]-1;
    }
    dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}

// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  data.hydro->EnrollUserDefBoundary(&BCGas);
  data.dust[0]->EnrollUserDefBoundary(&BCDust);
  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);

                // Tracers
                if(R>1.0) {
                  d.Vc(TRG,k,j,i) = 1.0;
                } else {
                  d.Vc(TRG,k,j,i) = 0.0;
                }

                if(R>1.05) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else if(R<0.95) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else {
                  d.Vc(TRG+1,k,j,i) = 1.0;
                }

                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef
tracer 2

[Dust]
nSpecies 1
drag   tau  1.0


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg     userdef
X1-end     userdef
X2-beg     periodic
X2-end     periodic
X3-beg     outflow
X3-end     outflow

[Setup]
h0              0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;
        Vc(TRG,k,j,i) = Vc(TRG,k,j,iref);
        Vc(TRG+1,k,j,i) = Vc(TRG+1,k,j,iref);
    });
  }
  idfx::popRegion();
}

// Dust User-defined boundaries
void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = dust->Vc;
    IdefixArray1D<real> x1 = dust->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = dust->data->beg[IDIR];
    } else if (side==right) {
      iref = dust->data->end[IDIR]-1;
    }
    dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}

// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  data.hydro->EnrollUserDefBoundary(&BCGas);
  data.dust[0]->EnrollUserDefBoundary(&BCDust);
  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);

                // Tracers
                if(R>1.0) {
                  d.Vc(TRG,k,j,i) = 1.0;
                } else {
                  d.Vc(TRG,k,j,i) = 0.0;
                }

                if(R>1.05) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else if(R<0.95) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else {
                  d.Vc(TRG+1,k,j,i) = 1.0;
                }

                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef
tracer 2

[Dust]
nSpecies 1
drag   tau  1.0


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg     userdef
X1-end     userdef
X2-beg     periodic
X2-end     periodic
X3-beg     outflow
X3-end     outflow

[Setup]
h0              0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;
        Vc(TRG,k,j,i) = Vc(TRG,k,j,iref);
        Vc(TRG+1,k,j,i) = Vc(TRG+1,k,j,iref);
    });
  }
  idfx::popRegion();
}

// Dust User-defined boundaries
void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = dust->Vc;
    IdefixArray1D<real> x1 = dust->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = dust->data->beg[IDIR];
    } else if (side==right) {
      iref = dust->data->end[IDIR]-1;
    }
    dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}

// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  data.hydro->EnrollUserDefBoundary(&BCGas);
  data.dust[0]->EnrollUserDefBoundary(&BCDust);
  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);

                // Tracers
                if(R>1.0) {
                  d.Vc(TRG,k,j,i) = 1.0;
                } else {
                  d.Vc(TRG,k,j,i) = 0.0;
                }

                if(R>1.05) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else if(R<0.95) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else {
                  d.Vc(TRG+1,k,j,i) = 1.0;
                }

                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;









            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef
tracer 2

[Dust]
nSpecies 1
drag   tau  1.0
tracer 2


[Gravity]
potential    central  planet
Mcentral     1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg    userdef
X1-end    userdef
X2-beg    periodic
X2-end    periodic
X3-beg    outflow
X3-end    outflow

[Setup]
h0             0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;
        Vc(TRG,k,j,i) = Vc(TRG,k,j,iref);
        Vc(TRG+1,k,j,i) = Vc(TRG+1,k,j,iref);
    });
  }
  idfx::popRegion();
}

// Dust User-defined boundaries
void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = dust->Vc;
    IdefixArray1D<real> x1 = dust->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = dust->data->beg[IDIR];
    } else if (side==right) {
      iref = dust->data->end[IDIR]-1;
    }
    dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;


    });
  }
  idfx::popRegion();
}

// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  data.hydro->EnrollUserDefBoundary(&BCGas);
  data.dust[0]->EnrollUserDefBoundary(&BCDust);
  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);

                // Tracers
                if(R>1.0) {
                  d.Vc(TRG,k,j,i) = 1.0;
                } else {
                  d.Vc(TRG,k,j,i) = 0.0;
                }

                if(R>1.05) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else if(R<0.95) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else {
                  d.Vc(TRG+1,k,j,i) = 1.0;
                }

                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;

                // Dust Tracers
                if(R>1) {
                  d.dustVc[0](TRD,k,j,i) = 0.0;
                } else {
                  d.dustVc[0](TRD,k,j,i) = 1.0;
                }

                if(phi>0.5) {
                  d.dustVc[0](TRD+1,k,j,i) = 0.0;
                } else if(phi<-0.5) {
                  d.dustVc[0](TRD+1,k,j,i) = 0.0;
                } else {
                  d.dustVc[0](TRD+1,k,j,i) = 1.0;
                }
            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef
tracer 2

[Dust]
nSpecies 1
drag   tau  1.0
tracer 2


[Gravity]
potential     central  planet
Mcentral      1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg     userdef
X1-end     userdef
X2-beg     periodic
X2-end     periodic
X3-beg     outflow
X3-end     outflow

[Setup]
h0              0.05
```

4

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;
        Vc(TRG,k,j,i) = Vc(TRG,k,j,iref);
        Vc(TRG+1,k,j,i) = Vc(TRG+1,k,j,iref);
    });
  }
  idfx::popRegion();
}

// Dust User-defined boundaries
void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = dust->Vc;
    IdefixArray1D<real> x1 = dust->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = dust->data->beg[IDIR];
    } else if (side==right) {
      iref = dust->data->end[IDIR]-1;
    }
    dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;
        Vc(TRD,k,j,i) = Vc(TRD,k,j,iref);
        Vc(TRD+1,k,j,i) = Vc(TRD+1,k,j,iref);
    });
  }
  idfx::popRegion();
}

// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  data.hydro->EnrollUserDefBoundary(&BCGas);
  data.dust[0]->EnrollUserDefBoundary(&BCDust);
  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);

                // Tracers
                if(R>1.0) {
                  d.Vc(TRG,k,j,i) = 1.0;
                } else {
                  d.Vc(TRG,k,j,i) = 0.0;
                }

                if(R>1.05) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else if(R<0.95) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else {
                  d.Vc(TRG+1,k,j,i) = 1.0;
                }

                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;

                // Dust Tracers
                if(R>1) {
                  d.dustVc[0](TRD,k,j,i) = 0.0;
                } else {
                  d.dustVc[0](TRD,k,j,i) = 1.0;
                }

                if(phi>0.5) {
                  d.dustVc[0](TRD+1,k,j,i) = 0.0;
                } else if(phi<-0.5) {
                  d.dustVc[0](TRD+1,k,j,i) = 0.0;
                } else {
                  d.dustVc[0](TRD+1,k,j,i) = 1.0;
                }
            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1



[Hydro]
solver        hllc
csiso         userdef
tracer 2

[Dust]
nSpecies 1
drag   tau  1.0
tracer 2


[Gravity]
potential    central  planet
Mcentral     1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg     userdef
X1-end     userdef
X2-beg     periodic
X2-end     periodic
X3-beg     outflow
X3-end     outflow

[Setup]
h0              0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;
        Vc(TRG,k,j,i) = Vc(TRG,k,j,iref);
        Vc(TRG+1,k,j,i) = Vc(TRG+1,k,j,iref);
    });
  }
  idfx::popRegion();
}

// Dust User-defined boundaries
void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = dust->Vc;
    IdefixArray1D<real> x1 = dust->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = dust->data->beg[IDIR];
    } else if (side==right) {
      iref = dust->data->end[IDIR]-1;
    }
    dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;
        Vc(TRD,k,j,i) = Vc(TRD,k,j,iref);
        Vc(TRD+1,k,j,i) = Vc(TRD+1,k,j,iref);
    });
  }
  idfx::popRegion();
}

// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  data.hydro->EnrollUserDefBoundary(&BCGas);
  data.dust[0]->EnrollUserDefBoundary(&BCDust);
  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);

                // Tracers
                if(R>1.0) {
                  d.Vc(TRG,k,j,i) = 1.0;
                } else {
                  d.Vc(TRG,k,j,i) = 0.0;
                }

                if(R>1.05) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else if(R<0.95) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else {
                  d.Vc(TRG+1,k,j,i) = 1.0;
                }

                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;

                // Dust Tracers
                if(R>1) {
                  d.dustVc[0](TRD,k,j,i) = 0.0;
                } else {
                  d.dustVc[0](TRD,k,j,i) = 1.0;
                }

                if(phi>0.5) {
                  d.dustVc[0](TRD+1,k,j,i) = 0.0;
                } else if(phi<-0.5) {
                  d.dustVc[0](TRD+1,k,j,i) = 0.0;
                } else {
                  d.dustVc[0](TRD+1,k,j,i) = 1.0;
                }
            }
        }
    }
    d.SyncToDevice();
}
```

idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1

[TimeIntegrator]
CFL         0.5
tstop       37.69911184307752 # 6 orbital periods
first_dt    1.e-3
nstages     2

[Hydro]
solver        hllc
csiso         userdef
tracer 2

[Dust]
nSpecies 1
drag   tau  1.0
tracer 2


[Gravity]
potential    central  planet
Mcentral     1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg    userdef
X1-end    userdef
X2-beg    periodic
X2-end    periodic
X3-beg    outflow
X3-end    outflow

[Setup]
h0             0.05
```

EXAMPLE OF SETUP

Using Idefix

https://idefix.readthedocs.io/latest/index.html

definitions.hpp

```
#define     COMPONENTS      2
#define     DIMENSIONS      2

#define     ISOTHERMAL

#define     GEOMETRY        POLAR
```

setup.cpp

```
#include <algorithm>
#include "idefix.hpp"
#include "setup.hpp"

namespace SetupVariables {
real h0;
}

void MySoundSpeed(DataBlock &data, const real t, IdefixArray3D<real> &cs) {
  IdefixArray1D<real> x1 = data.x[IDIR];
  real h0 = SetupVariables::h0;
  idefix_for("MyCS",0,data.np_tot[KDIR],0,data.np_tot[JDIR],0,data.np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      real R = x1(i);
      cs(k,j,i) = h0/sqrt(R);
    });
}

// Gas User-defined boundaries
void BCGas(Fluid<DefaultPhysics> *hydro, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = hydro->Vc;
    IdefixArray1D<real> x1 = hydro->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = hydro->data->beg[IDIR];
    } else if (side==right) {
      iref = hydro->data->end[IDIR]-1;
    }
    hydro->boundary->BoundaryFor("UserDefBoundary_X1", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;
        Vc(TRG,k,j,i) = Vc(TRG,k,j,iref);
        Vc(TRG+1,k,j,i) = Vc(TRG+1,k,j,iref);
    });
  }
  idfx::popRegion();
}

// Dust User-defined boundaries
void BCDust(Fluid<DustPhysics> *dust, int dir, BoundarySide side, real t) {
  idfx::pushRegion("UserDefinedBoundary");
  if(dir==IDIR) {
    IdefixArray4D<real> Vc = dust->Vc;
    IdefixArray1D<real> x1 = dust->data->x[IDIR];
    int iref;
    if(side == left) {
      iref = dust->data->beg[IDIR];
    } else if (side==right) {
      iref = dust->data->end[IDIR]-1;
    }
    dust->boundary->BoundaryFor("UserDefBoundary_X1Dust", dir, side,
      KOKKOS_LAMBDA (int k, int j, int i) {
        real R=x1(i);
        real Vk = 1.0/sqrt(R);

        Vc(RHO,k,j,i) = Vc(RHO,k,j,iref);
        Vc(VX1,k,j,i) = Vc(VX1,k,j,iref);
        Vc(VX2,k,j,i) = Vk;
        Vc(TRD,k,j,i) = Vc(TRD,k,j,iref);
        Vc(TRD+1,k,j,i) = Vc(TRD+1,k,j,iref);
    });
  }
  idfx::popRegion();
}

// Default constructor
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output)
{
  data.hydro->EnrollUserDefBoundary(&BCGas);
  data.dust[0]->EnrollUserDefBoundary(&BCDust);
  data.hydro->EnrollIsoSoundSpeed(&MySoundSpeed);
  SetupVariables::h0 = input.Get<real>("Setup","h0",0);
}

void Setup::InitFlow(DataBlock &data) {
    DataBlockHost d(data);
    real h0 = SetupVariables::h0;

    for(int k = 0; k < d.np_tot[KDIR] ; k++) {
        for(int j = 0; j < d.np_tot[JDIR] ; j++) {
            for(int i = 0; i < d.np_tot[IDIR] ; i++) {
                real R=d.x[IDIR](i);
                real phi = d.x[JDIR](j);
                real Vk=1.0/sqrt(R);
                real cs2=(h0*Vk)*(h0*Vk);

                // Gas initial conditions
                d.Vc(RHO,k,j,i) = 100/R;
                d.Vc(VX1,k,j,i) = 0.0;
                d.Vc(VX2,k,j,i) = Vk*sqrt( 1 - 2.0*h0*h0);

                // Tracers
                if(R>1.0) {
                  d.Vc(TRG,k,j,i) = 1.0;
                } else {
                  d.Vc(TRG,k,j,i) = 0.0;
                }

                if(R>1.05) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else if(R<0.95) {
                  d.Vc(TRG+1,k,j,i) = 0.0;
                } else {
                  d.Vc(TRG+1,k,j,i) = 1.0;
                }

                // Dust initial conditions
                d.dustVc[0](RHO,k,j,i) = d.Vc(RHO,k,j,i) / 100.0;
                d.dustVc[0](VX1,k,j,i) = 0.0;
                d.dustVc[0](VX2,k,j,i) = Vk;

                // Dust Tracers
                if(R>1) {
                  d.dustVc[0](TRD,k,j,i) = 0.0;
                } else {
                  d.dustVc[0](TRD,k,j,i) = 1.0;
                }

                if(phi>0.5) {
                  d.dustVc[0](TRD+1,k,j,i) = 0.0;
                } else if(phi<-0.5) {
                  d.dustVc[0](TRD+1,k,j,i) = 0.0;
                } else {
                  d.dustVc[0](TRD+1,k,j,i) = 1.0;
                }
            }
        }
    }
    d.SyncToDevice();
}
```

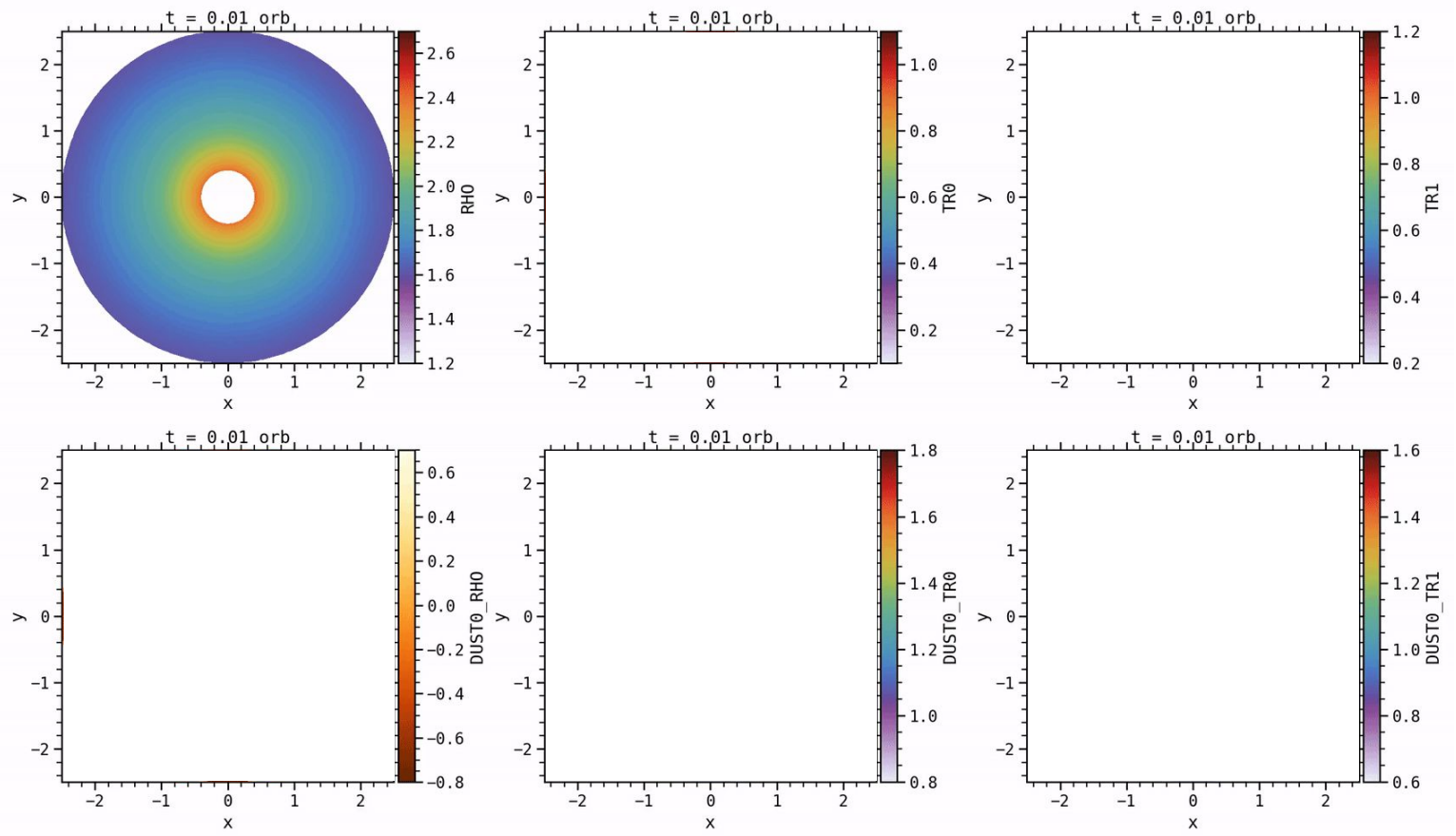
idefix.ini

```
[Grid]
X1-grid    1  0.4      128  l  2.5
X2-grid    1  0.0      256  u  6.283185307179586
X3-grid    1  -1  1    u  1

[TimeIntegrator]
CFL         0.5
tstop       37.69911184307752 # 6 orbital periods
first_dt    1.e-3
nstages     2

[Hydro]
solver        hllc
csiso         userdef
tracer 2

[Dust]
nSpecies 1
drag   tau  1.0
tracer 2


[Gravity]
potential    central  planet
Mcentral     1.0

[Planet]
integrator          analytical
planetToPrimary     1.0e-3
initialDistance     1.0
feelDisk            false
feelPlanets         false
smoothing           plummer     0.03  0.0

[Boundary]
X1-beg    userdef
X1-end    userdef
X2-beg    periodic
X2-end    periodic
X3-beg    outflow
X3-end    outflow

[Setup]
h0             0.05

[Output]
vtk    0.06283185307179586
dmp    6.283185307179586
log    100
```

4

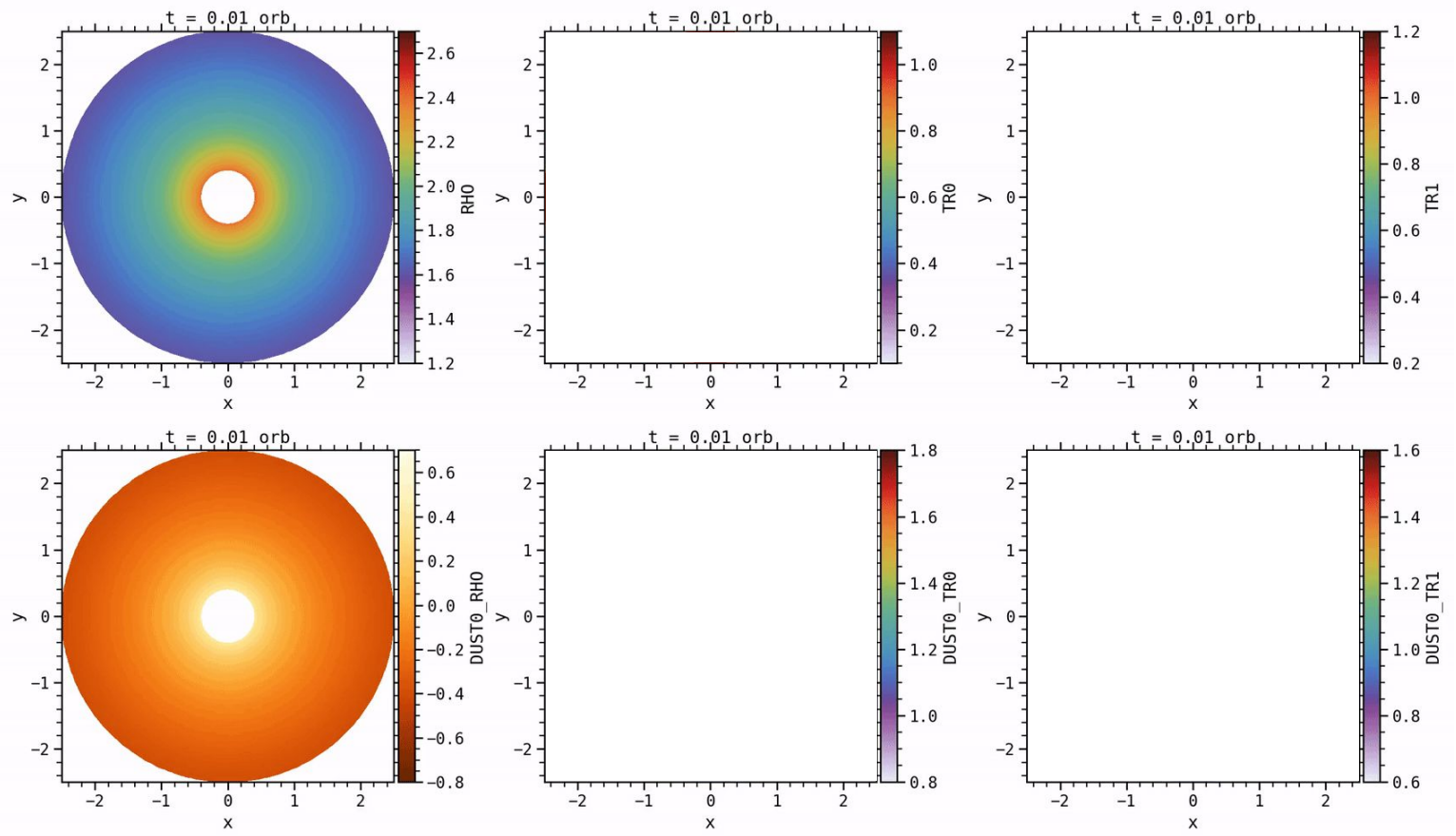
EXAMPLE OF SETUP

Visualization



EXAMPLE OF SETUP

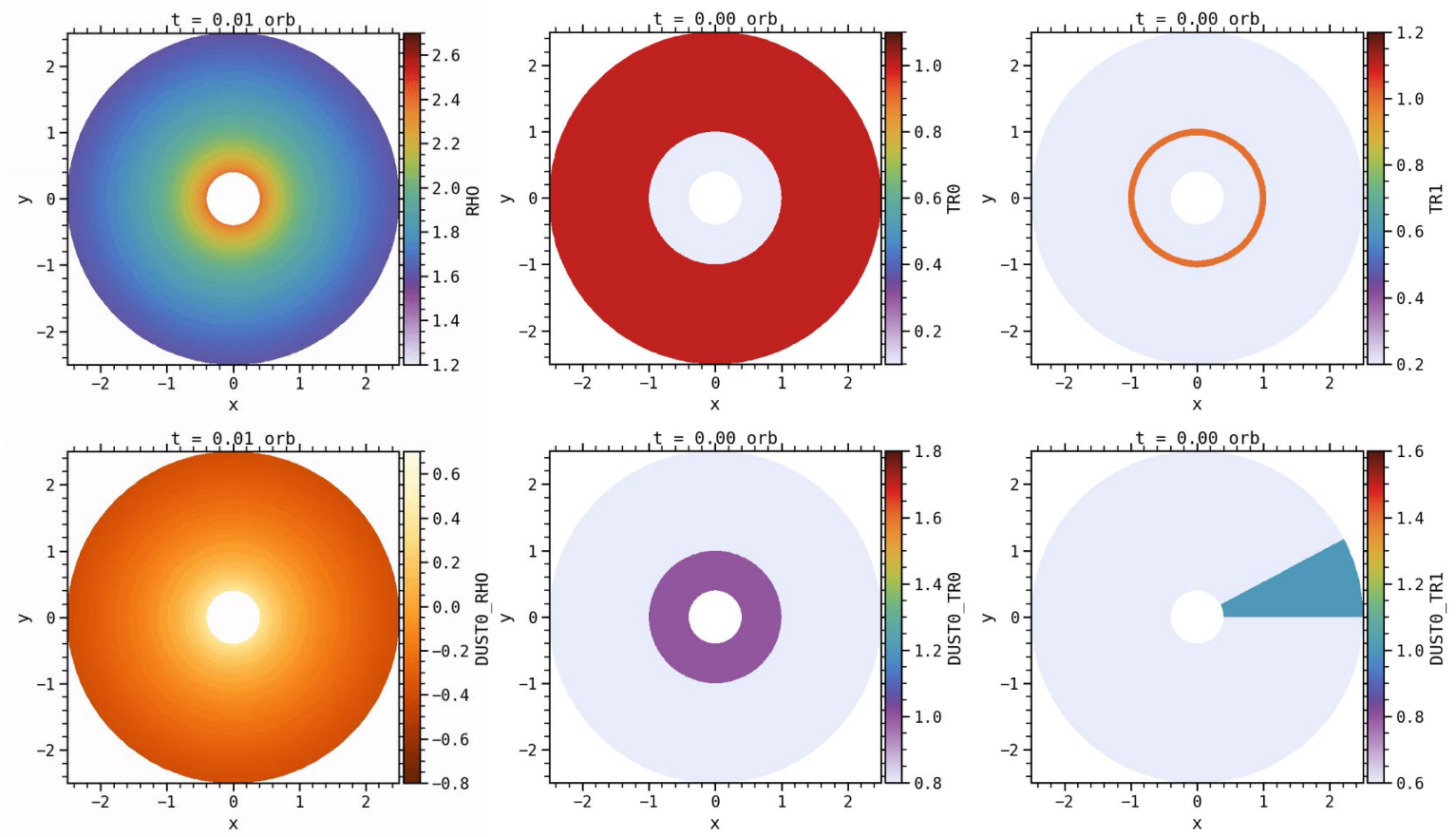
Visualization



4

EXAMPLE OF SETUP

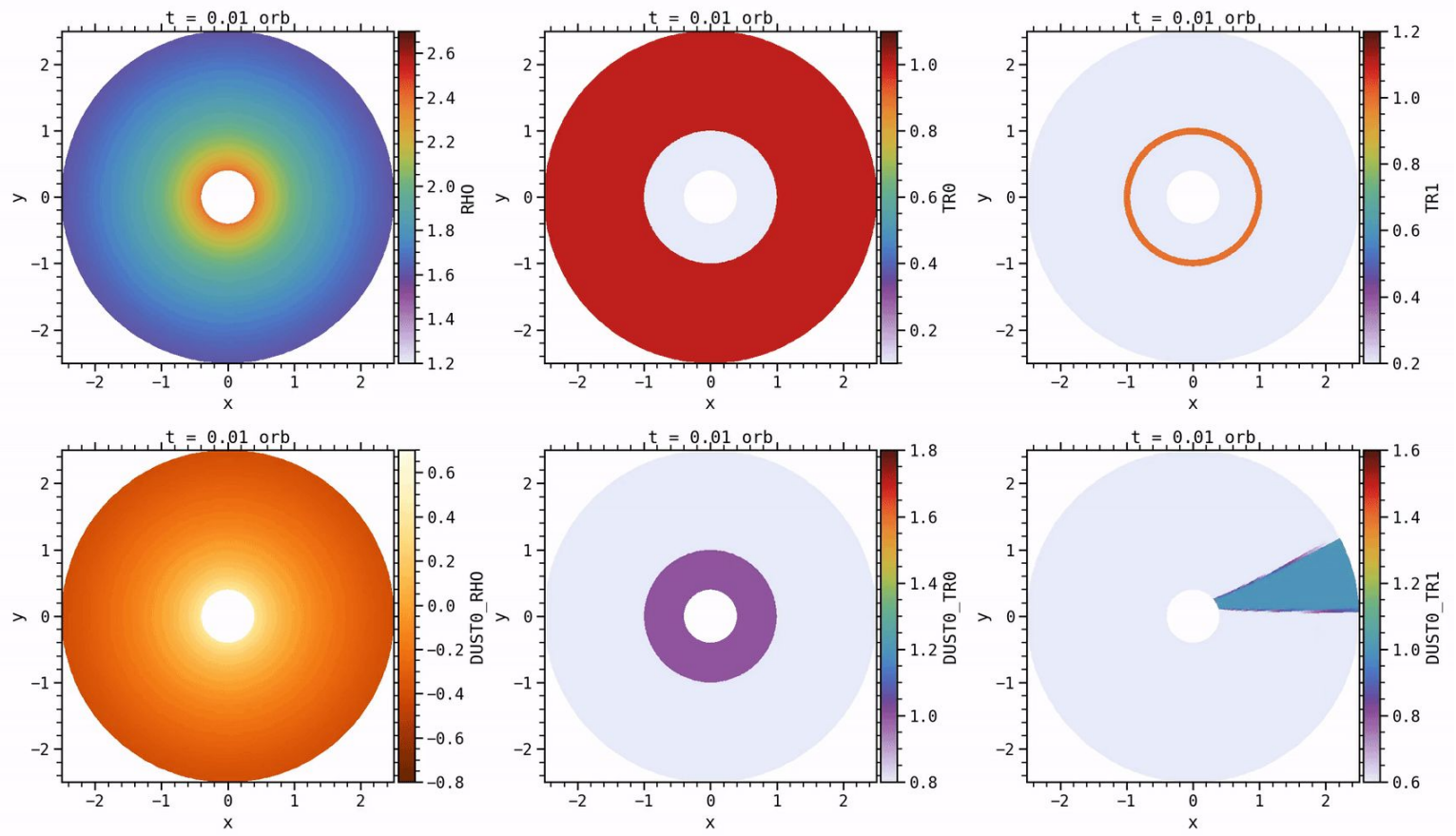
Visualization



EXAMPLE OF SETUP

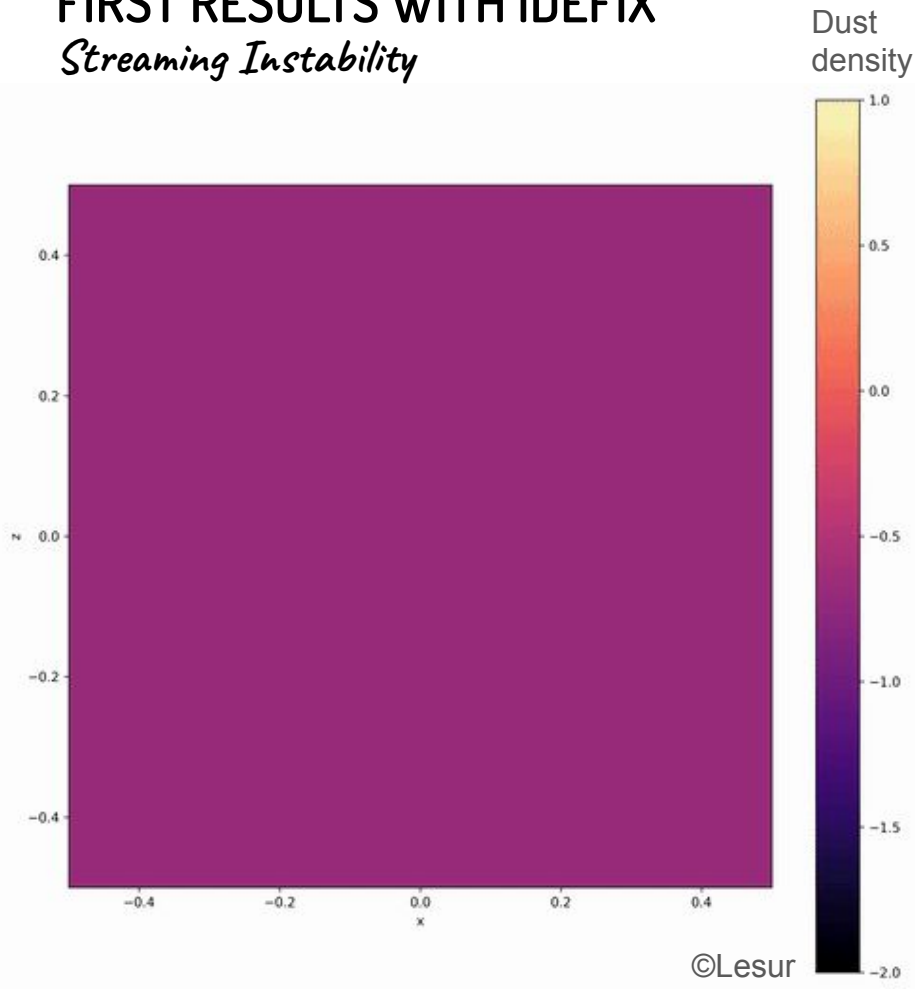
Visualization

$$t_d \simeq 0.8\, t_g$$



FIRST RESULTS WITH IDEFIX

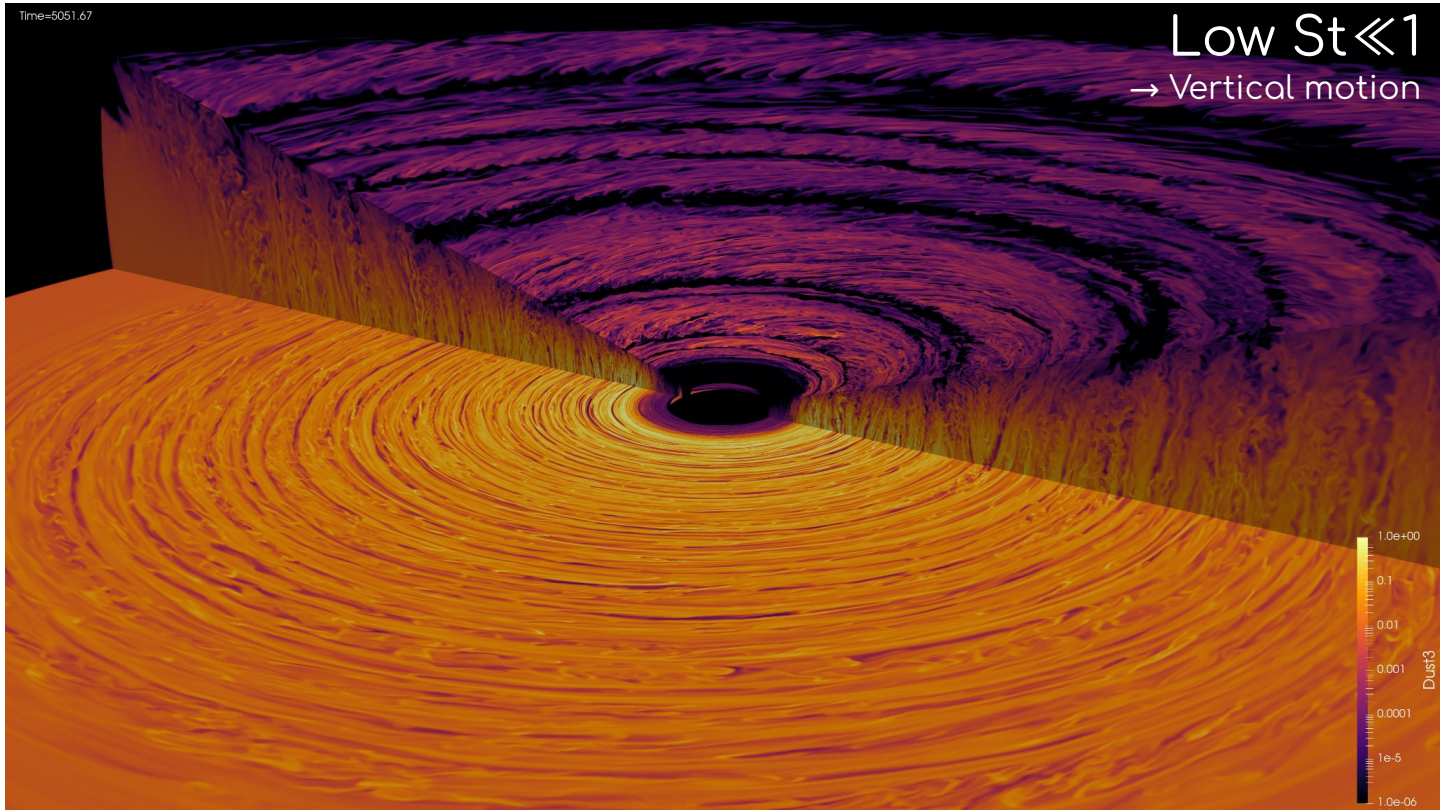
Streaming Instability



See also the low-resolution test in
$IDEFIX_DIR/test/Dust/StreamingInstability

FIRST RESULTS WITH IDEFIX

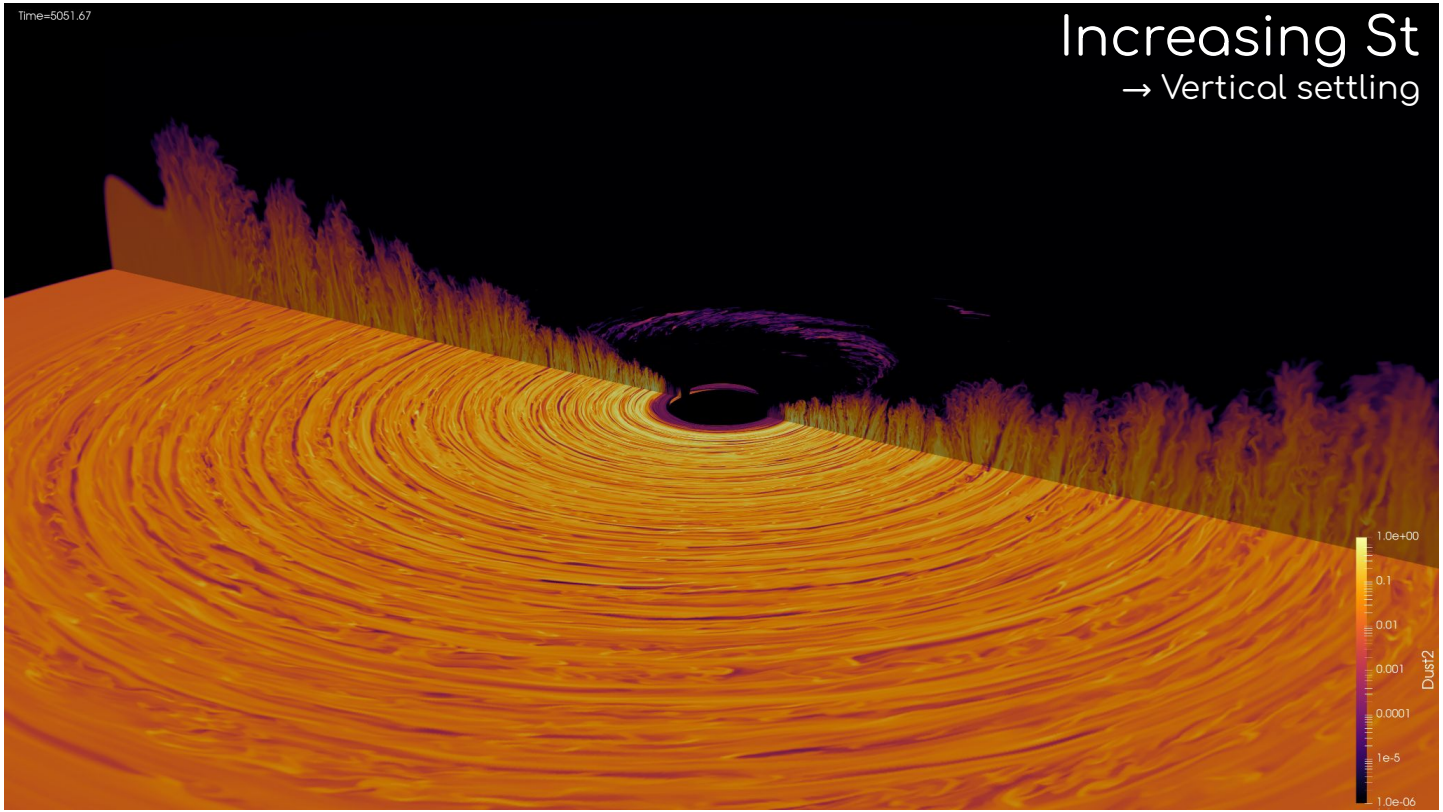
Vertical Shear Instability with dust



©Lesur

FIRST RESULTS WITH IDEFIX

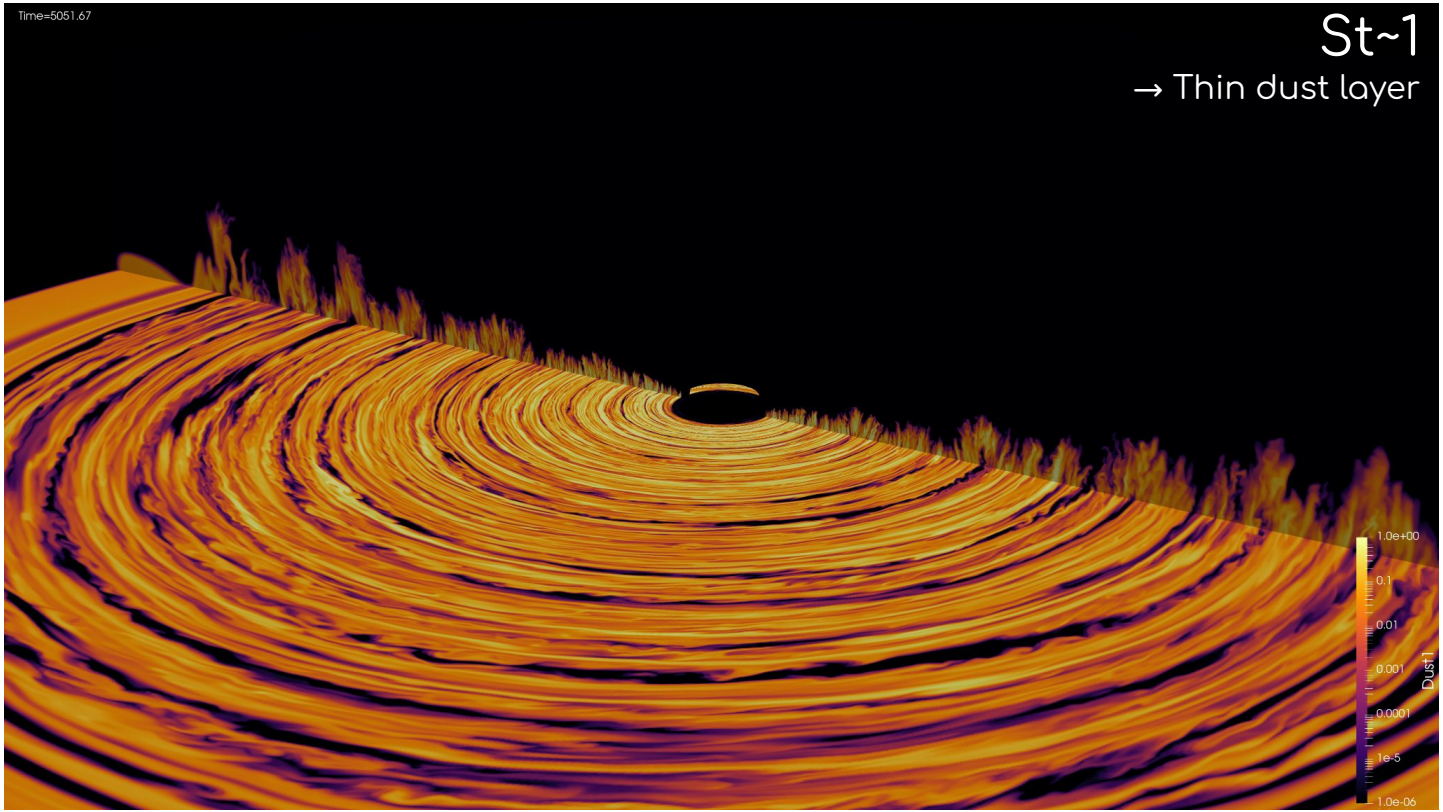
Vertical Shear Instability with dust



©Lesur

FIRST RESULTS WITH IDEFIX

Vertical Shear Instability with dust



©Lesur

ROUND TABLE

Prep

Victor
- userdef restistive simulations + userdef braginskii
- wish: variable gamma OR modify the current Braginskii implementation

Jean
- 3D spherical compressible MHD + Braginskii, varying the thermal diffusivity
- wish: go to the pole ?

François
- ecology class, import matrix for interactions from python package to idefix

Geoff
- eurohpc-ju.europa.eu
- nonos on 2.2 To vtk file ?

GWF/Thomas
- wish: implicit scheme for the drag ?

Mario
- wish: multiple star system, go to the pole ?

Hossam
- wish: dust solver → crashes : add a flat reconstruction when dust_density<0
- wish: dust diffusion, dust particles ?

ROUND TABLE

Prep

Thomas

- wish: more dust tests, what are tests and what are true setups, vtk slices per field

Geoff

- wish: priorities (dev ? doc ? formation ?), AMR, GR, PIC, incompressible method (spectral) ?

GWF: test → vtk_slices (readable with nonos ?) + dust with MHD wind + LookupTable/DumpImage for restarting with .npy file could be possible

Parameters in a dusty run

In your idefix.ini

[Dust]
nSpecies
drag
drag_feedback

Parameters in a dusty run

In your idefix.ini

[Dust]
nSpecies *integer n, number of dust species*
drag
drag_feedback

Parameters in a dusty run

In your idefix.ini

[Dust]

nSpecies	*integer n, number of dust species*
drag	*gas→dust drag law*
drag_feedback	

Parameters in a dusty run

In your idefix.ini

[Dust]

nSpecies	*integer n, number of dust species*
drag	*gas→dust drag law*
drag_feedback	*is there dust→gas drag ?*

Parameters in a dusty run

In your idefix.ini

[Dust]

nSpecies	*integer n, number of dust species*
drag	*gas→dust drag law*
drag_feedback	*is there dust→gas drag ? yes or no*

Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \to d_i}$$

i integer, 1≤i≤n

Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \to d_i}$$

i integer, 1≤i≤n

Where $\vec{f}_{g \to d_i} = \gamma_i \rho_{d_i} \rho \left(\vec{v}_g - \vec{v}_{d_i} \right)$

Four possible drag laws: **gamma** fixes γ_i

fixed drag parameter

Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \to d_i}$$

i integer, 1≤i≤n

$$\gamma_i = \frac{1}{\rho t_i}$$

Where $\vec{f}_{g \to d_i} = \gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) = \frac{\rho_i}{t_i} (\vec{v}_g - \vec{v}_{d_i})$

Four possible drag laws: **gamma** fixes γ_i, **tau** fixes t_i

fixed drag parameter

fixed stopping time

Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \to d_i}$$

i integer, 1≤i≤n

Where $\vec{f}_{g \to d_i} = \gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) = \frac{\rho_i}{t_i} (\vec{v}_g - \vec{v}_{d_i}) = \frac{c_s \rho_{d_i} \rho}{\beta_i} (\vec{v}_g - \vec{v}_{d_i})$

$$\gamma_i = \frac{1}{\rho t_i}$$

$$t_i = \frac{\beta_i}{\rho c_s}$$

Four possible drag laws: **gamma** fixes γ_i, **tau** fixes t_i, **size** fixes β_i

fixed drag parameter

fixed stopping time

Epstein or Stokes drag law with fixed:

- Dust density ρ_s
- Dust size a

Epstein: $\beta_i = (\rho_s a)_i$

Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \to d_i}$$

i integer, 1≤i≤n

Where $\vec{f}_{g \to d_i} = \gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) = \frac{\rho_i}{t_i} (\vec{v}_g - \vec{v}_{d_i}) = \frac{c_s \rho_{d_i} \rho}{\beta_i} (\vec{v}_g - \vec{v}_{d_i})$

$$\gamma_i = \frac{1}{\rho t_i}$$

$$t_i = \frac{\beta_i}{\rho c_s}$$

Four possible drag laws: **gamma** fixes γ_i, **tau** fixes t_i, **size** fixes β_i

fixed drag parameter

fixed stopping time

Epstein or Stokes drag law with fixed:

- Dust density ρ_s
- Dust size a

Epstein: $\beta_i = (\rho_s a)_i$

See Thomas' presentation this afternoon !

Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \to d_i}$$

i integer, 1≤i≤n

Where $\vec{f}_{g \to d_i} = \gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) = \frac{\rho_i}{t_i} (\vec{v}_g - \vec{v}_{d_i}) = \frac{c_s \rho_{d_i} \rho}{\beta_i} (\vec{v}_g - \vec{v}_{d_i})$

$$\gamma_i = \frac{1}{\rho t_i}$$

$$t_i = \frac{\beta_i}{\rho c_s}$$

Four possible drag laws: **gamma** fixes γ_i, **tau** fixes t_i, **size** fixes β_i and **userdef** is whatever you like

fixed drag parameter

fixed stopping time

Epstein or Stokes drag law with fixed:

- Dust density ρ_s
- Dust size a

Epstein: $\beta_i = (\rho_s a)_i$

See Thomas' presentation this afternoon !

Drag laws

$$\frac{\partial(\rho_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i}) = 0$$

$$\frac{\partial(\rho_{d_i} \vec{v}_{d_i})}{\partial t} + \vec{\nabla} \cdot (\rho_{d_i} \vec{v}_{d_i} \otimes \vec{v}_{d_i}) = \rho_{d_i} \vec{g} + \vec{f}_{g \to d_i}$$

i integer, 1≤i≤n

Where $\vec{f}_{g \to d_i} = \gamma_i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i}) = \frac{\rho_i}{t_i} (\vec{v}_g - \vec{v}_{d_i}) = \frac{c_s \rho_{d_i} \rho}{\beta_i} (\vec{v}_g - \vec{v}_{d_i})$

$$\gamma_i = \frac{1}{\rho t_i}$$

$$t_i = \frac{\beta_i}{\rho c_s}$$

Four possible drag laws: **gamma** fixes γ_i, **tau** fixes t_i, **size** fixes β_i and **userdef** is whatever you like

fixed drag parameter

fixed stopping time

Epstein or Stokes drag law with fixed:

- Dust density ρ_s
- Dust size a

Epstein: $\beta_i = (\rho_s a)_i$

See Thomas' presentation this afternoon !

Example for **gamma**

[Dust]

nSpecies	*3*		
drag	*gamma*	*1.0. 2.0 3.0*	sets three dust species (1≤i≤ 3) with drag laws $\vec{f}_{g \to d_i} = i \rho_{d_i} \rho (\vec{v}_g - \vec{v}_{d_i})$
drag_feedback	*yes*		

Example for userdef

In your setup.cpp

```
void MyDrag(DataBlock *data, real beta, IdefixArray3D<real> &gamma) {

  //Compute the drag coefficient gamma from the input beta

  idefix_for("MyDrag",0,data->np_tot[KDIR],0,data->np_tot[JDIR],0,data->np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      gamma(k,j,i) = 1/(beta*data->hydro->Vc(RHO,k,j,i));
        });
}
```

Example for userdef

In your setup.cpp

```
void MyDrag(DataBlock *data, real beta, IdefixArray3D<real> &gamma) {

  //Compute the drag coefficient gamma from the input beta

  idefix_for("MyDrag",0,data->np_tot[KDIR],0,data->np_tot[JDIR],0,data->np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      gamma(k,j,i) = 1/(beta*data->hydro->Vc(RHO,k,j,i));
        });
}
```

Don’t forget to enroll it !

```
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output){
   //(...)
   if(data.haveDust) {
                int nSpecies = data.dust.size();
                        for(int n = 0 ; n < nSpecies ; n++) {
                                data.dust[n]->drag->EnrollUserDrag(&MyDrag);
                        }
                }
}
```

Example for userdef

In your setup.cpp

```
void MyDrag(DataBlock *data, real beta, IdefixArray3D<real> &gamma) {

  //Compute the drag coefficient gamma from the input beta

  idefix_for("MyDrag",0,data->np_tot[KDIR],0,data->np_tot[JDIR],0,data->np_tot[IDIR],
    KOKKOS_LAMBDA (int k, int j, int i) {
      gamma(k,j,i) = 1/(beta*data->hydro->Vc(RHO,k,j,i));
        });
}
```

Don’t forget to enroll it !

```
Setup::Setup(Input &input, Grid &grid, DataBlock &data, Output &output){
   //(...)
   if(data.haveDust) {
                int nSpecies = data.dust.size();
                        for(int n = 0 ; n < nSpecies ; n++) {
                                data.dust[n]->drag->EnrollUserDrag(&MyDrag);
                        }
                }
}
```

With that drag

```
[Dust]
nSpecies        1
drag            userdef   1.0
drag_feedback   yes
```

=

```
[Dust]
nSpecies        1
drag            tau   1.0
drag_feedback   yes
```

Drag feedback and CFL condition

Dust equation of motion: $\frac{\partial(\rho_{d_i}\vec{v}_{d_i})}{\partial t} + \vec{\nabla}\cdot(\rho_{d_i}\vec{v}_{d_i}\otimes\vec{v}_{d_i}) = \rho_{d_i}\vec{g} + \vec{f}_{g\to d_i}$

Drag feedback and CFL condition

Dust equation of motion: $\frac{\partial(\rho_{d_i}\vec{v}_{d_i})}{\partial t} + \vec{\nabla}\cdot(\rho_{d_i}\vec{v}_{d_i}\otimes\vec{v}_{d_i}) = \rho_{d_i}\vec{g} + \vec{f}_{g\to d_i}$

With feedback, additional forces on the gas:

$$\vec{f}_{d_i\to g} = -\vec{f}_{g\to d_i} = -\gamma_i\rho_{d_i}\rho\left(\vec{v}_g - \vec{v}_{d_i}\right)$$ for each dust species i

Drag feedback and CFL condition

Dust equation of motion: $\frac{\partial(\rho_{d_i}\vec{v}_{d_i})}{\partial t} + \vec{\nabla}\cdot(\rho_{d_i}\vec{v}_{d_i}\otimes\vec{v}_{d_i}) = \rho_{d_i}\vec{g} + \vec{f}_{g\to d_i}$

With feedback, additional forces on the gas:

$$\vec{f}_{d_i\to g} = -\vec{f}_{g\to d_i} = -\gamma_i\rho_{d_i}\rho\left(\vec{v}_g - \vec{v}_{d_i}\right)$$ for each dust species i

Without dust or feedback

$$\mathrm{d}t = \sigma_{\mathrm{CFL}}\left(\max_{\mathcal{V}}\left[\sum_d \frac{c_{\max,d}}{\mathrm{d}\ell_d} + \frac{2\eta_{\max}}{\mathrm{d}\ell_d^2}\right]\right)^{-1}$$

Lesur+(2021)

Drag feedback and CFL condition

Dust equation of motion: $\frac{\partial(\rho_{d_i}\vec{v}_{d_i})}{\partial t} + \vec{\nabla}\cdot(\rho_{d_i}\vec{v}_{d_i}\otimes\vec{v}_{d_i}) = \rho_{d_i}\vec{g} + \vec{f}_{g\to d_i}$

With feedback, additional forces on the gas:

$$\vec{f}_{d_i\to g} = -\vec{f}_{g\to d_i} = -\gamma_i\rho_{d_i}\rho\left(\vec{v}_g - \vec{v}_{d_i}\right)$$ for each dust species i

Without dust or feedback

$$\mathrm{d}t = \sigma_{\mathrm{CFL}}\left(\max_{\mathcal{V}}\left[\sum_d \frac{c_{\max,d}}{\mathrm{d}\ell_d} + \frac{2\eta_{\max}}{\mathrm{d}\ell_d^2}\right]\right)^{-1}$$

Lesur+(2021)

With feedback

$$dt < \min\left(\frac{1}{\sum_i \gamma_i(\rho_i+\rho)}\right)$$

Idefix userguide

Thank you