GPUDEM

1.0

Generated by Doxygen 1.9.1

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 accelerationHandling Namespace Reference

Acceleration handling of particles.

Functions

- __device__ void calculateDefault (int tid, struct registerMemory &rmem, struct particle particles)

  Calculates the acceleration of the particles.
- __device__ void addBodyForces (int tid, struct registerMemory &rmem, struct particle particles, struct bodyForce bodyForces)

  Modifies the acceleration with the body forces terms.

4.1.1 Detailed Description

Acceleration handling of particles.

4.1.2 Function Documentation

4.1.2.1 addBodyForces()

```
__device__ void accelerationHandling::addBodyForces (
            int tid,
            struct registerMemory & rmem,
            struct particle particles,
            struct bodyForce bodyForces )
```

Modifies the acceleration with the body forces terms.

Parameters

tid	Thread index
rmem	Register memory containg all the data about particle with index tid
particles	Contains all the particle data
bodyForces	Contains all the information about the body forces

Returns

Particle accelerations are modified in the register memory rmem

4.1.2.2 calculateDefault()

```
__device__ void accelerationHandling::calculateDefault (
              int tid,
              struct registerMemory & rmem,
              struct particle particles )
```

Calculates the acceleration of the particles.

Parameters

tid	Thread index
rmem	Register memory containg all the data about particle with index tid
particles	Contains all the particle data

Returns

Particle accelerations are written in the register memory rmem

4.2 constant Namespace Reference

Contains all the constant.

Variables

- constexpr var_type PI = 3.14159265358979323846264 3
- constexpr var_type VOLUME_FACTOR = PI ∗ 4.0 / 3.0
- constexpr var_type DAMPING = -1.8257418583505537115
- constexpr var_type ZERO = 0.0
- constexpr var_type NUMBER_04 = 0.4
- constexpr var_type NUMBER_05 = 0.5
- constexpr var_type NUMBER_4o3 = 4.0/3.0
- constexpr var_type NUMBER_1 = 1.0
- constexpr var_type NUMBER_2 = 2.0
- constexpr var_type NUMBER_8 = 8.0
- constexpr var_type AB2C1 = 1.5
- constexpr var_type AB2C2 = 0.5
- constexpr var_type AM2C1 = 0.5

4.2.1 Detailed Description

Contains all the constant.

4.2.2 Variable Documentation

4.2.2.1 AB2C1

```
constexpr var_type constant::AB2C1 = 1.5  [constexpr]
```

4.2.2.2 AB2C2

```
constexpr var_type constant::AB2C2 = 0.5  [constexpr]
```

4.2.2.3 AM2C1

```
constexpr var_type constant::AM2C1 = 0.5  [constexpr]
```

4.2.2.4 DAMPING

```
constexpr var_type constant::DAMPING = -1.8257418583505537115  [constexpr]
```

4.2.2.5 NUMBER_04

```
constexpr var_type constant::NUMBER_04 = 0.4  [constexpr]
```

4.2.2.6 NUMBER_05

```
constexpr var_type constant::NUMBER_05 = 0.5  [constexpr]
```

4.2.2.7 NUMBER_1

```
constexpr var_type constant::NUMBER_1 = 1.0  [constexpr]
```

4.2.2.8 NUMBER_2

```
constexpr var_type constant::NUMBER_2 = 2.0  [constexpr]
```

4.2.2.9 NUMBER_4o3

```
constexpr var_type constant::NUMBER_4o3 = 4.0/3.0  [constexpr]
```

4.2.2.10 NUMBER_8

```
constexpr var_type constant::NUMBER_8 = 8.0  [constexpr]
```

4.2.2.11 PI

```
constexpr var_type constant::PI = 3.14159265358979323846264 3  [constexpr]
```

4.2.2.12 VOLUME_FACTOR

```
constexpr var_type constant::VOLUME_FACTOR = PI * 4.0 / 3.0  [constexpr]
```

4.2.2.13 ZERO

```
constexpr var_type constant::ZERO = 0.0  [constexpr]
```

4.3 contactHandling Namespace Reference

Contact handling of particles.

Functions

- __device__ bool areNeighbours (int cid1, int cid2)

  Checks if two cells are neighbours or not.
- void __device__ CalculateContact (int tid, struct registerMemory &rmem, int i, var_type d, var_type Rs, struct particle particles, struct contact &contacts)

  Calculates the contact parameters between two particles.
- void __device__ ResetContacts (int tid, struct contact &contacts)

  Prepares the contact struct for the next timestep by copying deltat into deltat_last for each contact.
- void __device__ BruteForceContactSearch (int tid, struct registerMemory &rmem, int numberOfActive↩ Particles, struct particle particles, struct contact &contacts)

  Brute force contact search, which goes through all possible combinations and calculates all contacts.
- void __device__ CalculateCellId (int tid, struct registerMemory &rmem, int numberOfActiveParticles, struct particle particles)

  Calculates the cell id.
- void __device__ DecomposedDomainsContactSearch (int tid, struct registerMemory &rmem, int numberOf↩ ActiveParticles, struct particle particles, struct contact &contacts)

  Decomposed domains contact search, which checks if particles are in the same or neighbouring cells and calculates all contacts.
- void __device__ initializeContacts (int tid, struct contact &contacts)

  Initializes the contacts struct at the beginning of the solver kernel.

Variables

- constexpr __device__ int Neighbours [27]

  calculate neighbours on compile time

4.3.1 Detailed Description

Contact handling of particles.

4.3.2 Function Documentation

4.3.2.1 areNeighbours()

```
__device__ bool contactHandling::areNeighbours (
            int cid1,
            int cid2 )  [inline]
```

Checks if two cells are neighbours or not.

Parameters

cid1	Cell id 1
cid2	Cell id 2

Returns

Returns if the cells are neighbours or not

4.3.2.2 BruteForceContactSearch()

```
void __device__ contactHandling::BruteForceContactSearch (
            int tid,
            struct registerMemory & rmem,
            int numberOfActiveParticles,
            struct particle particles,
            struct contact & contacts )
```

Brute force contact search, which goes through all possible combinations and calculates all contacts.

Parameters

tid	Thread index of the particle
rmem	Register memory containing all the data about the particle
numberOfActiveParticles	Number of active parameters
particles	All the particle data
contacts	List of contacts

4.3.2.3 CalculateCellId()

```
void __device__ contactHandling::CalculateCellId (
            int tid,
            struct registerMemory & rmem,
            int numberOfActiveParticles,
            struct particle particles )
```

Calculates the cell id.

Parameters

tid	Thread index of the particle
rmem	Register memory containing all the data about the particle
numberOfActiveParticles	Number of active parameters
particles	All the particle data

4.3.2.4 CalculateContact()

```
void __device__ contactHandling::CalculateContact (
            int tid,
```

```
            struct registerMemory & rmem,
            int i,
            var_type d,
            var_type Rs,
            struct particle particles,
            struct contact & contacts )
```

Calculates the contact parameters between two particles.

Parameters

tid	Thread index of the particle
rmem	Register memory containing all the data about the particle
i	Thread index of the particle, particle tid is in contact with
d	Distance between particles
Rs	Sum of radii of particle i and tid
particles	All the particle data
contacts	List of contacts

Returns

the contact struct is filled up

4.3.2.5 DecomposedDomainsContactSearch()

```
void __device__ contactHandling::DecomposedDomainsContactSearch (
            int tid,
            struct registerMemory & rmem,
            int numberOfActiveParticles,
            struct particle particles,
            struct contact & contacts )
```

Decomposed domains contact search, which checks if particles are in the same or neighbouring cells and calculates all contacts.

Parameters

tid	Thread index of the particle
rmem	Register memory containing all the data about the particle
numberOfActiveParticles	Number of active parameters
particles	All the particle data
contacts	List of contacts

4.3.2.6 initializeContacts()

```
void __device__ contactHandling::initializeContacts (
```

```
            int tid,
            struct contact & contacts )
```

Initializes the contacts struct at the beginning of the solver kernel.

Parameters

tid	Thread index of the particle
contacts	List of particles we are in contact with

Initalizes tid with -1 and deltat with 0, resets contacts.count to 0

4.3.2.7 ResetContacts()

```
void __device__ contactHandling::ResetContacts (
            int tid,
            struct contact & contacts )
```

Prepares the contact struct for the next timestep by copying deltat into deltat_last for each contact.

Parameters

tid	Thread index of the particle
contacts	List of contacts

4.3.3 Variable Documentation

4.3.3.1 Neighbours

```
constexpr __device__ int contactHandling::Neighbours[27]  [constexpr]
```

Initial value:

```
= {
        0, 1, -1, DecomposedDomainsConstants::Nx,    -DecomposedDomainsConstants::Nx,
        DecomposedDomainsConstants::Nx*DecomposedDomainsConstants::Ny,
       -DecomposedDomainsConstants::Nx*DecomposedDomainsConstants::Ny,
        DecomposedDomainsConstants::Nx - 1,         -DecomposedDomainsConstants::Nx - 1,
        DecomposedDomainsConstants::Nx + 1,         -DecomposedDomainsConstants::Nx + 1,
        DecomposedDomainsConstants::Nx*DecomposedDomainsConstants::Ny + 1,
       -DecomposedDomainsConstants::Nx*DecomposedDomainsConstants::Ny + 1,
        DecomposedDomainsConstants::Nx*DecomposedDomainsConstants::Ny - 1,
       -DecomposedDomainsConstants::Nx*DecomposedDomainsConstants::Ny - 1,
        DecomposedDomainsConstants::Nx*(1+DecomposedDomainsConstants::Ny),
       -DecomposedDomainsConstants::Nx*(1+DecomposedDomainsConstants::Ny),
        DecomposedDomainsConstants::Nx*(1+DecomposedDomainsConstants::Ny)+1,
       -DecomposedDomainsConstants::Nx*(1+DecomposedDomainsConstants::Ny)+1,
        DecomposedDomainsConstants::Nx*(1+DecomposedDomainsConstants::Ny)-1,
       -DecomposedDomainsConstants::Nx*(1+DecomposedDomainsConstants::Ny)-1,
        DecomposedDomainsConstants::Nx*(-1+DecomposedDomainsConstants::Ny),
       -DecomposedDomainsConstants::Nx*(-1+DecomposedDomainsConstants::Ny),
        DecomposedDomainsConstants::Nx*(-1+DecomposedDomainsConstants::Ny)+1,
       -DecomposedDomainsConstants::Nx*(-1+DecomposedDomainsConstants::Ny)+1,
        DecomposedDomainsConstants::Nx*(-1+DecomposedDomainsConstants::Ny)-1,
       -DecomposedDomainsConstants::Nx*(-1+DecomposedDomainsConstants::Ny)-1
    }
```

calculate neighbours on compile time

4.4 DecomposedDomainsConstants Namespace Reference

Variables

- constexpr int Nx = 100

  Number of cell in x,y,z direction.
- constexpr int Ny = 100
- constexpr int Nz = 200
- constexpr var_type minx = -1.0

  Min of coordinates.
- constexpr var_type miny = -1.0
- constexpr var_type minz = 0.0
- constexpr var_type maxx = 1.0

  Max of coordinates.
- constexpr var_type maxy = 1.0
- constexpr var_type maxz = 4.0
- constexpr var_type NoverDx = var_type(Nx)/(maxx-minx)

  DO NOT MODIFY - 1/max-min pre-calculated.
- constexpr var_type NoverDy = var_type(Ny)/(maxy-miny)
- constexpr var_type NoverDz = var_type(Nz)/(maxz-minz)

4.4.1 Variable Documentation

4.4.1.1 maxx

```
constexpr var_type DecomposedDomainsConstants::maxx = 1.0  [constexpr]
```

Max of coordinates.

4.4.1.2 maxy

```
constexpr var_type DecomposedDomainsConstants::maxy = 1.0  [constexpr]
```

4.4.1.3 maxz

```
constexpr var_type DecomposedDomainsConstants::maxz = 4.0  [constexpr]
```

4.4.1.4 minx

```
constexpr var_type DecomposedDomainsConstants::minx = -1.0  [constexpr]
```

Min of coordinates.

4.4.1.5 miny

```
constexpr var_type DecomposedDomainsConstants::miny = -1.0  [constexpr]
```

4.4.1.6 minz

```
constexpr var_type DecomposedDomainsConstants::minz = 0.0  [constexpr]
```

4.4.1.7 NoverDx

```
constexpr var_type DecomposedDomainsConstants::NoverDx = var_type(Nx)/(maxx-minx)  [constexpr]
```

DO NOT MODIFY - 1/max-min pre-calculated.

4.4.1.8 NoverDy

```
constexpr var_type DecomposedDomainsConstants::NoverDy = var_type(Ny)/(maxy-miny)  [constexpr]
```

4.4.1.9 NoverDz

```
constexpr var_type DecomposedDomainsConstants::NoverDz = var_type(Nz)/(maxz-minz)  [constexpr]
```

4.4.1.10 Nx

```
constexpr int DecomposedDomainsConstants::Nx = 100  [constexpr]
```

Number of cell in x,y,z direction.

4.4.1.11 Ny

```
constexpr int DecomposedDomainsConstants::Ny = 100  [constexpr]
```

4.4.1.12 Nz

```
constexpr int DecomposedDomainsConstants::Nz = 200  [constexpr]
```

4.5 domainHandling Namespace Reference

Handling of boundary conditions and STL files.

Functions

- __device__ void CalculateOverlap (int tid, struct registerMemory &rmem, int i, var_type d, struct contact &contacts)

  Calculates the contact parameters between a particle and a boundary.
- __device__ void applyBoundaryConditions (int tid, struct registerMemory &rmem, struct particle particles, struct boundaryCondition boundaryConditions, struct contact &contacts, struct materialParameters pars, struct timestepping timestep)

  Calculates the forces based on the boundary constraints and given model.

4.5.1 Detailed Description

Handling of boundary conditions and STL files.

4.5.2 Function Documentation

4.5.2.1 applyBoundaryConditions()

```
__device__ void domainHandling::applyBoundaryConditions (
            int tid,
            struct registerMemory & rmem,
            struct particle particles,
            struct boundaryCondition boundaryConditions,
            struct contact & contacts,
            struct materialParameters pars,
            struct timestepping timestep )  [inline]
```

Calculates the forces based on the boundary constraints and given model.

Parameters

tid	Thread index of the particle
rmem	Register memory containing all the data about the particle
particles	List of all particle data
boundaryConditions	List of all boundary condition data
contacts	List of all contact data
pars	All material parameters
timestep	Timestep specificiations

4.5.2.2 CalculateOverlap()

```
__device__ void domainHandling::CalculateOverlap (
              int tid,
              struct registerMemory & rmem,
              int i,
              var_type d,
              struct contact & contacts )  [inline]
```

Calculates the contact parameters between a particle and a boundary.

Parameters

tid	Thread index of the particle
rmem	Register memory containing all the data about the particle
i	Index of the domain boundary, particle tid is in contact with
d	Distance between particle and domain boundary
contacts	List of contacts

Returns

the contact struct is filled up

4.6 forceHandling Namespace Reference

Contains all the functions to calculate the force between particles.

Functions

- __device__ void calculateForceMindlin (int tid, struct registerMemory &rmem, struct particle particles, struct contact contacts, struct materialParameters pars, struct timestepping timestep)

  Calculates the force acting on the particle in x,y,z system using the Mindlin-Hertz theory.
- var_type calculateTotalKineticEnergy (struct particle particles, int numberOfActiveParticles)

  Calculates the total kinetic energy.
- var_type calculateTotalPotentialEnergy (struct particle particles, struct bodyForce bodyForces, int number↩ OfActiveParticles)

  Calculates the total potential energy.

4.6.1 Detailed Description

Contains all the functions to calculate the force between particles.

4.6.2 Function Documentation

4.6.2.1 calculateForceMindlin()

```
__device__ void forceHandling::calculateForceMindlin (
            int tid,
            struct registerMemory & rmem,
            struct particle particles,
            struct contact contacts,
            struct materialParameters pars,
            struct timestepping timestep )  [inline]
```

Calculates the force acting on the particle in x,y,z system using the Mindlin-Hertz theory.

Parameters

tid	Thread index
rmem	Register memory containing all the data about the particle
particles	The particles struct containing all the data about them
contacts	The struct containing all the contacts
pars	The struct containing all the material parameters
timestep	Timestep settings

Returns

Returns the force in x,y,z coordinate system (adds it to rmem)

FORCES

4.6.2.2 calculateTotalKineticEnergy()

```
var_type forceHandling::calculateTotalKineticEnergy (
            struct particle particles,
            int numberOfActiveParticles )
```

Calculates the total kinetic energy.

Parameters

particles	A list of particles
numberOfActiveParticles	Number of active parameters

4.6.2.3 calculateTotalPotentialEnergy()

```
var_type forceHandling::calculateTotalPotentialEnergy (
            struct particle particles,
            struct bodyForce bodyForces,
            int numberOfActiveParticles )
```

Calculates the total potential energy.

Parameters

particles	A list of particles
bodyForces	Volumetric forces acting on the particles
numberOfActiveParticles	Number of active parameters

4.7 integrators Namespace Reference

Contains the numerical methods and timestepping.

Functions

- __device__ var_type RK1 (var_type dt, var_type x, var_type f)

  Calculates the next point using 1st order Runge-Kutta (Euler)
- __device__ var_type AB2 (var_type dt, var_type x, var_type f, var_type f_old)

  Calculates the next point using 2nd order Adams-Bashfort method.
- __device__ var_type AM2 (var_type dt, var_type x, var_type f, var_type f_old)

  Calculates the next point using 2nd order Adams-Moulton method.
- __device__ void euler (int tid, struct registerMemory &rmem, struct particle particles, struct timestepping timestep)

  Calculates the new vel., angular vel., and position using Euler's method.
- __device__ void exact (int tid, struct registerMemory &rmem, struct particle particles, struct timestepping timestep)

  Calculates the new vel., angular vel., and position exactly from the acceleration.
- __device__ void adams2 (int tid, struct registerMemory &rmem, struct particle particles, struct timestepping timestep, int step)

  Calculates the new vel., angular vel., and position using 2nd order Adams methods.

4.7.1 Detailed Description

Contains the numerical methods and timestepping.

4.7.2 Function Documentation

4.7.2.1 AB2()

```
__device__ var_type integrators::AB2 (
              var_type dt,
              var_type x,
              var_type f,
              var_type f_old )  [inline]
```

Calculates the next point using 2nd order Adams-Bashfort method.

Parameters

dt	Timestep
x	Current point
f	Current derivative
f_old	Previous derivative

4.7.2.2 adams2()

```
__device__ void integrators::adams2 (
              int tid,
              struct registerMemory & rmem,
              struct particle particles,
              struct timestepping timestep,
              int step )  [inline]
```

Calculates the new vel., angular vel., and position using 2nd order Adams methods.

Parameters

tid	Thread index of the particle
rmem	Register memory containing all the data about the particle
particles	All the particle data
timestep	Timestep specificiations
step	Current timestep

4.7.2.3 AM2()

```
__device__ var_type integrators::AM2 (
              var_type dt,
              var_type x,
              var_type f,
              var_type f_old )  [inline]
```

Calculates the next point using 2nd order Adams-Moulton method.

Parameters

dt	Timestep
x	Current point
f	Current derivative
f_old	Previous derivative

4.7.2.4 euler()

```
__device__ void integrators::euler (
            int tid,
            struct registerMemory & rmem,
            struct particle particles,
            struct timestepping timestep )  [inline]
```

Calculates the new vel., angular vel., and position using Euler's method.

Parameters

tid	Thread index of the particle
rmem	Register memory containing all the data about the particle
particles	All the particle data
timestep	Timestep specificiations

4.7.2.5 exact()

```
__device__ void integrators::exact (
            int tid,
            struct registerMemory & rmem,
            struct particle particles,
            struct timestepping timestep )  [inline]
```

Calculates the new vel., angular vel., and position exactly from the acceleration.

Parameters

tid	Thread index of the particle
rmem	Register memory containing all the data about the particle
particles	All the particle data
timestep	Timestep specificiations

4.7.2.6 RK1()

```
__device__ var_type integrators::RK1 (
             var_type dt,
             var_type x,
             var_type f )  [inline]
```

Calculates the next point using 1st order Runge-Kutta (Euler)

Parameters

dt	Timestep
x	Current point
f	Current derivative

4.8 ioHandling Namespace Reference

Contains all the functions for writing and reading data.

Functions

- void saveParticles (int numberOfActiveParticles, struct particle particles, std::string location)

  Save the data of a list of particles in a .particle textfile.
- void saveParticlesVTK (int numberOfActiveParticles, struct particle particles, std::string location)

  Save the data of a list of particles as a vtk compatible .vtu unstructured grid file.
- int readParticlesVTK (struct particle particles, std::string location)

  Reads the particle data from a vtk compatible .vtu unstructured grid file.
- int readParticlesCSV (struct particle particles, std::string location)

  Reads the particle data from a .csv file.

4.8.1 Detailed Description

Contains all the functions for writing and reading data.

4.8.2 Function Documentation

4.8.2.1 readParticlesCSV()

```
int ioHandling::readParticlesCSV (
             struct particle particles,
             std::string location )
```

Reads the particle data from a .csv file.

Parameters

particles	List of particles
location	File location

Returns

Number of particles in the file

4.8.2.2 readParticlesVTK()

```
int ioHandling::readParticlesVTK (
            struct particle particles,
            std::string location )
```

Reads the particle data from a vtk compatible .vtu unstructured grid file.

Parameters

particles	List of particles
location	File location

Returns

Number of particles in the file

4.8.2.3 saveParticles()

```
void ioHandling::saveParticles (
            int numberOfActiveParticles,
            struct particle particles,
            std::string location )
```

Save the data of a list of particles in a .particle textfile.

Parameters

numberOfActiveParticles	Number of active parameters
particles	List of particles
location	File location

4.8.2.4 saveParticlesVTK()

```
void ioHandling::saveParticlesVTK (
            int numberOfActiveParticles,
            struct particle particles,
            std::string location )
```

Save the data of a list of particles as a vtk compatible .vtu unstructured grid file.

Parameters

numberOfActiveParticles	Number of active parameters
particles	List of particles
location	File location

File can be opened in paraview and the particles can be displayed using the Glyph filter

4.9 materialHandling Namespace Reference

Contains all the function for material handling.

Enumerations

- enum methods { Min , Max , HarmonicMean , Mean }

Functions

- void printMaterialInfo (struct materialParameters pars, bool printPairings=false)

  Prints all the materials and material combinations.
- void calculateMaterialContact (struct materialParameters &pars, methods friction, methods elastic, methods damping)

  Calculates all the material pairings and damping.

4.9.1 Detailed Description

Contains all the function for material handling.

4.9.2 Enumeration Type Documentation

4.9.2.1 methods

```
enum materialHandling::methods
```

Enumerator

Min	
Max	
HarmonicMean	
Mean	

4.9.3 Function Documentation

4.9.3.1 calculateMaterialContact()

```
void materialHandling::calculateMaterialContact (
            struct materialParameters & pars,
            methods friction,
            methods elastic,
            methods damping )
```

Calculates all the material pairings and damping.

Parameters

pars	materialParameters struct with ALL material parameters
friction	Method to calculate the friction coefficients
elastic	Method to calculate the elastic coefficients
damping	Method to calculate the damping coefficients

4.9.3.2 printMaterialInfo()

```
void materialHandling::printMaterialInfo (
            struct materialParameters pars,
            bool printPairings = false )
```

Prints all the materials and material combinations.

Parameters

pars	materialParameters struct with ALL material parameters
printPairings	Settings to print the details of material pairings

4.10 memoryHandling Namespace Reference

Memory handling functions which copy between CPU and GPU and allocate memory.

Enumerations

- enum listOfVariables {
  All , Position , Velocity , AngularVelocity ,
  Acceleration , AngularAcceleration , Material , Radius ,
  Force , Torque , CellID }

Functions

- void initializeHostParticles (struct particle &particlesH)

  Fills the host side for the particles with zeros.
- void allocateHostParticles (struct particle &particlesH)

  Allocates the host side for the particles.
- void freeHostParticles (struct particle &particlesH)

  Free the hist side.
- void allocateDeviceParticles (struct particle &particlesD)

  Allocates the device side for the particles.
- void freeDeviceParticles (struct particle &particlesD)

  Allocates the device side for the particles.
- void synchronizeParticles (struct particle dest, struct particle source, listOfVariables vars, cudaMemcpyKind kind)

  Synchronizes the memory between host and device.

4.10.1 Detailed Description

Memory handling functions which copy between CPU and GPU and allocate memory.

4.10.2 Enumeration Type Documentation

4.10.2.1 listOfVariables

```
enum memoryHandling::listOfVariables
```

Enumerator

All	
Position	
Velocity	
AngularVelocity	
Acceleration	
AngularAcceleration	
Material	
Radius	
Force	
Torque	
CellID	

4.10.3 Function Documentation

4.10.3.1 allocateDeviceParticles()

```
void memoryHandling::allocateDeviceParticles (
            struct particle & particlesD )
```

Allocates the device side for the particles.

Parameters

particlesD	The particle struct which needs memory allocation

4.10.3.2 allocateHostParticles()

```
void memoryHandling::allocateHostParticles (
            struct particle & particlesH )
```

Allocates the host side for the particles.

Parameters

particlesH	The particle struct which needs memory allocation

4.10.3.3 freeDeviceParticles()

```
void memoryHandling::freeDeviceParticles (
            struct particle & particlesD )
```

Allocates the device side for the particles.

Parameters

particlesD	The particle struct which needs memory allocation

4.10.3.4 freeHostParticles()

```
void memoryHandling::freeHostParticles (
            struct particle & particlesH )
```

Free the hist side.

Parameters

particlesH	The particle struct which is freed

4.10.3.5 initializeHostParticles()

```
void memoryHandling::initializeHostParticles (
            struct particle & particlesH )
```

Fills the host side for the particles with zeros.

Parameters

particlesH	The particle struct which needs to be initalized

4.10.3.6 synchronizeParticles()

```
void memoryHandling::synchronizeParticles (
            struct particle dest,
            struct particle source,
            listOfVariables vars,
            cudaMemcpyKind kind )
```

Synchronizes the memory between host and device.

Parameters

dest	Destination of the data
source	Source of the data
vars	Variables to copy according to the enum
kind	cudaMemcpyDeviceToHost or HostToDevice

4.11 particleHandling Namespace Reference

Contains all the functions and structs necessary for handling the particles.

Enumerations

- enum class ParticleSizeDistribution { None , Uniform , Gauss }
- enum class ParticleVelocityDistribution { None , Uniform , Gauss }

Functions

- void generateParticleParameters (struct particle p, struct materialParameters pars, int mat_id, int start_id, int end_id)

  Fills up the particle struct p from a given material data.
- void generateParticleLocation (struct particle p, struct particleDistribution pdist, ParticleSizeDistribution psize_dist, ParticleVelocityDistribution pvel_dist)

  Particle generation based on the initial particle distribution.
- void generateParticleLocation (struct particle p, struct coordinates coords, var_type *radii, struct materialParameters pars)

  Particle generation based on coordinate lists.
- void printParticles (struct particle p)

  Print the data of a list of particles.

4.11.1 Detailed Description

Contains all the functions and structs necessary for handling the particles.

4.11.2 Enumeration Type Documentation

4.11.2.1 ParticleSizeDistribution

`enum particleHandling::ParticleSizeDistribution [strong]`

Enumerator

None	
Uniform	
Gauss	

4.11.2.2 ParticleVelocityDistribution

`enum particleHandling::ParticleVelocityDistribution [strong]`

Enumerator

None	
Uniform	
Gauss	

4.11.3 Function Documentation

4.11.3.1 generateParticleLocation() [1/2]

```
void particleHandling::generateParticleLocation (
            struct particle p,
            struct coordinates coords,
            var_type * radii,
            struct materialParameters pars )
```

Particle generation based on coordinate lists.

Parameters

particles	Pre-allocated memory where the particle data is saved
coords	Coordinates of the particles
radii	Radii of the particles
pars	Physical parameters given by the user

Returns

ensamble of particles according to the distribution described in pdist

4.11.3.2 generateParticleLocation() [2/2]

```
void particleHandling::generateParticleLocation (
            struct particle p,
            struct particleDistribution pdist,
            ParticleSizeDistribution psize_dist,
            ParticleVelocityDistribution pvel_dist )
```

Particle generation based on the initial particle distribution.

Parameters

particles	Pre-allocated memory where the particle data is saved
pdist	Particle distribution information based on the particleDistribution struct
psize_dist	Particle size distribuation chosen from the ParticleSizeDistribution enum
pvel_dist	Particle velocity distribuation chosen from the ParticleVelocityDistribution enum

Returns

ensamble of particles according to the distribution described in pdist

4.11.3.3 generateParticleParameters()

```
void particleHandling::generateParticleParameters (
            struct particle p,
            struct materialParameters pars,
            int mat_id,
            int start_id,
            int end_id )
```

Fills up the particle struct p from a given material data.

Parameters

p	Pre-allocated memory where the particle data is saved
pars	Physical parameters given by the user
mat_id	Material ID
start_id	Starting index of this material
end_id	End index of this material

Returns

ensamble of particles according to the distribution described in pdist

4.11.3.4 printParticles()

```
void particleHandling::printParticles (
            struct particle p )
```

Print the data of a list of particles.

Parameters

particles	A list of particles

4.12 RandomGeneration Namespace Reference

Contains everything necessary for random generation.

Functions

- void initializeRandomSeed ()

  Initializes a random seed based on time.
- void initializeRandomSeed (int seed)

  Initializes a random seed based on a given number.
- var_type randomInRange (var_type min, var_type max)

  Generates a random var_type in a range.

Variables

- var_type overRandMax = constant::NUMBER_1/var_type(RAND_MAX)

4.12.1 Detailed Description

Contains everything necessary for random generation.

4.12.2 Function Documentation

4.12.2.1 initializeRandomSeed() [1/2]

```
void RandomGeneration::initializeRandomSeed ( )
```

Initializes a random seed based on time.

4.12.2.2 initializeRandomSeed() [2/2]

```
void RandomGeneration::initializeRandomSeed (
            int seed )
```

Initializes a random seed based on a given number.

Parameters

seed	seed of srand()

4.12.2.3 randomInRange()

```
var_type RandomGeneration::randomInRange (
            var_type min,
            var_type max )
```

Generates a random var_type in a range.

Parameters

min	lower end of the range
max	higher end of the range

4.12.3 Variable Documentation

4.12.3.1 overRandMax

```
var_type RandomGeneration::overRandMax = constant::NUMBER_1/var_type(RAND_MAX)
```

4.13 registerHandling Namespace Reference

Register handling functions.

Functions

- __device__ void fillRegisterMemory (int tid, struct registerMemory &rmem, struct particle particles)
  Copies the data from global memory to the registers.
- __device__ void endOfStepSync (int tid, struct registerMemory &rmem, struct particle particles)
  Saves the necessary data to the registers.
- __device__ void endOfKernelSync (int tid, struct registerMemory &rmem, struct particle particles)
  Saves the necessary data to the registers at the end of the kernel.

4.13.1 Detailed Description

Register handling functions.

4.13.2 Function Documentation

4.13.2.1 endOfKernelSync()

```
__device__ void registerHandling::endOfKernelSync (
            int tid,
            struct registerMemory & rmem,
            struct particle particles )  [inline]
```

Saves the necessary data to the registers at the end of the kernel.

Parameters

tid	Thread index
rmem	Register memory
particles	Data about all the particles

If forces or torques are saved they must be copied back to the global memory

4.13.2.2 endOfStepSync()

```
__device__ void registerHandling::endOfStepSync (
            int tid,
            struct registerMemory & rmem,
            struct particle particles )  [inline]
```

Saves the necessary data to the registers.

Parameters

tid	Thread index
rmem	Register memory
particles	Data about all the particles

Used after each timestep to save position, velocity and angular velocity since these are needed for the next contact calculations

4.13.2.3 fillRegisterMemory()

```
__device__ void registerHandling::fillRegisterMemory (
            int tid,
            struct registerMemory & rmem,
            struct particle particles )  [inline]
```

Copies the data from global memory to the registers.

Parameters

tid	Thread index
rmem	Register memory
particles	Data about all the particles

4.14 timeHandling Namespace Reference

Functions for handling time.

Functions

- void printTimestepSettings (struct timestepping timestep)
  Prints the timestep settings.

4.14.1 Detailed Description

Functions for handling time.

4.14.2 Function Documentation

4.14.2.1 printTimestepSettings()

```
void timeHandling::printTimestepSettings (
            struct timestepping timestep )
```

Prints the timestep settings.

Parameters

timestep	The timestepping struct containg all data about the timesteps

Chapter 5

Class Documentation

5.1 bodyForce Struct Reference

Stores the constant body forces (e.g. gravity) in the different directions.

Public Attributes

- var_type x
  x direction
- var_type y
  y direction
- var_type z
  z direction

5.1.1 Detailed Description

Stores the constant body forces (e.g. gravity) in the different directions.

5.1.2 Member Data Documentation

5.1.2.1 x

```
var_type bodyForce::x
```

x direction

5.1.2.2 y

```
var_type bodyForce::y
```

y direction

5.1.2.3 z

```
var_type bodyForce::z
```

z direction

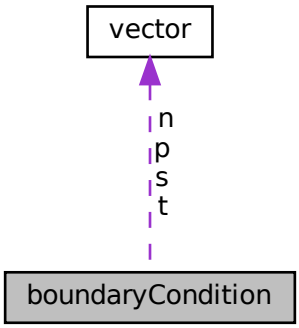
The documentation for this struct was generated from the following file:

- source/forces.cuh

5.2 boundaryCondition Struct Reference

Contains all the data about boundary conditions.

Collaboration diagram for boundaryCondition:



Public Attributes

- vec3D n [NumberOfBoundaries]
  Normal vector, pointing outwards.
- vec3D p [NumberOfBoundaries]
  Point on the plane.
- vec3D s [NumberOfBoundaries]
- vec3D t [NumberOfBoundaries]
- var_type t_scale [NumberOfBoundaries]
- var_type s_scale [NumberOfBoundaries]
- BoundaryConditionType type [NumberOfBoundaries]
  Type of BC.
- var_type alpha [NumberOfBoundaries]
  par1
- var_type beta [NumberOfBoundaries]
  par2
- var_type gamma [NumberOfBoundaries]
  par3
- int material [NumberOfBoundaries]
  parameter set for materials

5.2.1 Detailed Description

Contains all the data about boundary conditions.

5.2.2 Member Data Documentation

5.2.2.1 alpha

```
var_type boundaryCondition::alpha[NumberOfBoundaries]
```

par1

5.2.2.2 beta

```
var_type boundaryCondition::beta[NumberOfBoundaries]
```

par2

5.2.2.3 gamma

```
var_type boundaryCondition::gamma[NumberOfBoundaries]
```

par3

5.2.2.4 material

```
int boundaryCondition::material[NumberOfBoundaries]
```

parameter set for materials

5.2.2.5 n

```
vec3D boundaryCondition::n[NumberOfBoundaries]
```

Normal vector, pointing outwards.

5.2.2.6 p

```
vec3D boundaryCondition::p[NumberOfBoundaries]
```

Point on the plane.

5.2.2.7 s

```
vec3D boundaryCondition::s[NumberOfBoundaries]
```

5.2.2.8 s_scale

```
var_type boundaryCondition::s_scale[NumberOfBoundaries]
```

5.2.2.9 t

```
vec3D boundaryCondition::t[NumberOfBoundaries]
```

5.2.2.10 t_scale

```
var_type boundaryCondition::t_scale[NumberOfBoundaries]
```

5.2.2.11 type

```
BoundaryConditionType boundaryCondition::type[NumberOfBoundaries]
```

Type of BC.

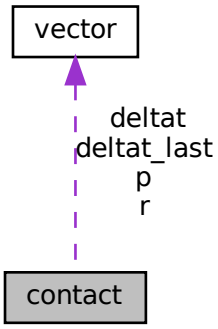
The documentation for this struct was generated from the following file:

- source/domain.cuh

5.3 contact Struct Reference

Contact data between particles, stored in the registers (preferably)

Collaboration diagram for contact:



Public Attributes

- int tid [MaxContactNumber]

  tid of the particle we are in contact with
- int tid_last [MaxContactNumber]

  tid of the particle of last contact
- int material [MaxContactNumber]

  type of other material
- var_type Rstar [MaxContactNumber]

  equivalent radius of contact

- var_type mstar [MaxContactNumber]
  equiavlent mass of contact
- vec3D r [MaxContactNumber]
  unit vector between particles
- vec3D p [MaxContactNumber]
  contact position
- var_type deltan [MaxContactNumber]
  normal overlap
- vec3D deltat [MaxContactNumber]
  tangential overlap
- vec3D deltat_last [MaxContactNumber]
  tangential overlap in the last step
- int count
  number of contacts

5.3.1 Detailed Description

Contact data between particles, stored in the registers (preferably)

5.3.2 Member Data Documentation

5.3.2.1 count

```
int contact::count
```

number of contacts

5.3.2.2 deltan

```
var_type contact::deltan[MaxContactNumber]
```

normal overlap

5.3.2.3 deltat

```
vec3D contact::deltat[MaxContactNumber]
```

tangential overlap

5.3.2.4 deltat_last

```
vec3D contact::deltat_last[MaxContactNumber]
```

tangential overlap in the last step

5.3.2.5 material

```
int contact::material[MaxContactNumber]
```

type of other material

5.3.2.6 mstar

```
var_type contact::mstar[MaxContactNumber]
```

equiavlent mass of contact

5.3.2.7 p

```
vec3D contact::p[MaxContactNumber]
```

contact position

5.3.2.8 r

```
vec3D contact::r[MaxContactNumber]
```

unit vector between particles

5.3.2.9 Rstar

```
var_type contact::Rstar[MaxContactNumber]
```

equivalent radius of contact

5.3.2.10 tid

```
int contact::tid[MaxContactNumber]
```

tid of the particle we are in contact with

5.3.2.11 tid_last

```
int contact::tid_last[MaxContactNumber]
```

tid of the particle of last contact

The documentation for this struct was generated from the following file:

- source/contact.cuh

5.4 coordinate Struct Reference

Cartesian coordinates.

Public Attributes

- var_type x
  x coodinate
- var_type y
  y coodinate
- var_type z
  z coodinate

5.4.1 Detailed Description

Cartesian coordinates.

5.4.2 Member Data Documentation

5.4.2.1 x

```
var_type coordinate::x
```

x coodinate

5.4.2.2 y

```
var_type coordinate::y
```

y coodinate

5.4.2.3 z

```
var_type coordinate::z
```

z coodinate

The documentation for this struct was generated from the following file:

- source/particle.cuh

5.5 coordinates Struct Reference

Cartesian coordinate vectors.

Public Attributes

- var_type * x
  x coodinate
- var_type * y
  y coodinate
- var_type * z
  z coodinate

5.5.1 Detailed Description

Cartesian coordinate vectors.

5.5.2 Member Data Documentation

5.5.2.1 x

```
var_type* coordinates::x
```

x coodinate

5.5.2.2 y

```
var_type* coordinates::y
```

y coodinate

5.5.2.3 z

```
var_type* coordinates::z
```

z coodinate

The documentation for this struct was generated from the following file:

- source/particle.cuh

5.6 materialContact Struct Reference

Struct which contains the reduced quantities for material combinations.

Public Attributes

- var_type mu_star [NumberOfMaterials]
- var_type mu0_star [NumberOfMaterials]
- var_type mur_star [NumberOfMaterials]
- var_type E_star [NumberOfMaterials]
- var_type G_star [NumberOfMaterials]
- var_type beta_star [NumberOfMaterials]

5.6.1 Detailed Description

Struct which contains the reduced quantities for material combinations.

5.6.2 Member Data Documentation

5.6.2.1 beta_star

```
var_type materialContact::beta_star[NumberOfMaterials]
```

5.6.2.2 E_star

```
var_type materialContact::E_star[NumberOfMaterials]
```

5.6.2.3 G_star

```
var_type materialContact::G_star[NumberOfMaterials]
```

5.6.2.4 mu0_star

```
var_type materialContact::mu0_star[NumberOfMaterials]
```

5.6.2.5 mu_star

```
var_type materialContact::mu_star[NumberOfMaterials]
```

5.6.2.6 mur_star

```
var_type materialContact::mur_star[NumberOfMaterials]
```

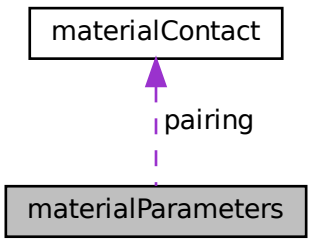
The documentation for this struct was generated from the following file:

- source/material.cuh

5.7 materialParameters Struct Reference

Struct with all the user given material parameters, stored in the shared memory on the device side.

Collaboration diagram for materialParameters:



Public Attributes

- var_type rho [NumberOfMaterials]
  Density.
- var_type E [NumberOfMaterials]
  Young's Modulus.
- var_type G [NumberOfMaterials]
  Shear Modulus.
- var_type nu [NumberOfMaterials]
  Poisson ratio.
- var_type e [NumberOfMaterials]
  Restitution.
- var_type mu [NumberOfMaterials]
  Sliding friction coeff.
- var_type mu0 [NumberOfMaterials]
  Static friction coeff.
- var_type mur [NumberOfMaterials]
  Rolling friction coeff.
- var_type beta [NumberOfMaterials]
  Damping.
- struct materialContact pairing [NumberOfMaterials]
  Lookup table for material pairinga.

5.7.1 Detailed Description

Struct with all the user given material parameters, stored in the shared memory on the device side.

5.7.2 Member Data Documentation

5.7.2.1 beta

```
var_type materialParameters::beta[NumberOfMaterials]
```

Damping.

5.7.2.2 E

```
var_type materialParameters::E[NumberOfMaterials]
```

Young's Modulus.

5.7.2.3 e

```
var_type materialParameters::e[NumberOfMaterials]
```

Restitution.

5.7.2.4 G

```
var_type materialParameters::G[NumberOfMaterials]
```

Shear Modulus.

5.7.2.5 mu

```
var_type materialParameters::mu[NumberOfMaterials]
```

Sliding friction coeff.

5.7.2.6 mu0

```
var_type materialParameters::mu0[NumberOfMaterials]
```

Static friction coeff.

5.7.2.7 mur

```
var_type materialParameters::mur[NumberOfMaterials]
```

Rolling friction coeff.

5.7.2.8 nu

```
var_type materialParameters::nu[NumberOfMaterials]
```

Poisson ratio.

5.7.2.9 pairing

```
struct materialContact materialParameters::pairing[NumberOfMaterials]
```

Lookup table for material pairinga.

5.7.2.10 rho

```
var_type materialParameters::rho[NumberOfMaterials]
```

Density.

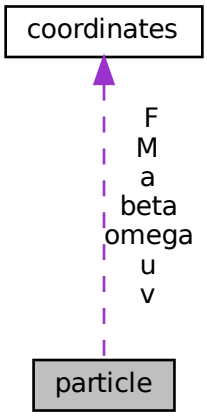
The documentation for this struct was generated from the following file:

- source/material.cuh

5.8 particle Struct Reference

Coordinates, velocity and radius of a particles.

Collaboration diagram for particle:



Public Attributes

- struct coordinates u
  position
- struct coordinates v
  velocity
- struct coordinates a
  acceleration
- struct coordinates omega
  angular velocity
- struct coordinates beta
  angular acceleration
- struct coordinates F
  force acting on the particle
- struct coordinates M
  torque acting on the particle
- var_type * R
  Radius of the particle.
- var_type * m
  Mass of the particle.
- var_type * theta
  Inertia of the particle.
- var_type * R_rec
  Inverse of radius of the particle.
- var_type * m_rec
  Inverse of mass of the particle.
- var_type * theta_rec
  Inverse of inertia of the particle.
- int * material
  Material set.
- int * cid
  Particle cell id.

5.8.1 Detailed Description

Coordinates, velocity and radius of a particles.

5.8.2 Member Data Documentation

5.8.2.1 a

```
struct coordinates particle::a
```

acceleration

5.8.2.2 beta

```
struct coordinates particle::beta
```

angular acceleration

5.8.2.3 cid

```
int* particle::cid
```

Particle cell id.

5.8.2.4 F

```
struct coordinates particle::F
```

force acting on the particle

5.8.2.5 M

```
struct coordinates particle::M
```

torque acting on the particle

5.8.2.6 m

```
var_type* particle::m
```

Mass of the particle.

5.8.2.7 m_rec

```
var_type* particle::m_rec
```

Inverse of mass of the particle.

5.8.2.8 material

```
int* particle::material
```

Material set.

5.8.2.9 omega

```
struct coordinates particle::omega
```

angular velocity

5.8.2.10 R

```
var_type* particle::R
```

Radius of the particle.

5.8.2.11 R_rec

```
var_type* particle::R_rec
```

Inverse of radius of the particle.

5.8.2.12 theta

```
var_type* particle::theta
```

Inertia of the particle.

5.8.2.13 theta_rec

```
var_type* particle::theta_rec
```

Inverse of inertia of the particle.

5.8.2.14 u

`struct coordinates particle::u`

position

5.8.2.15 v

`struct coordinates particle::v`

velocity

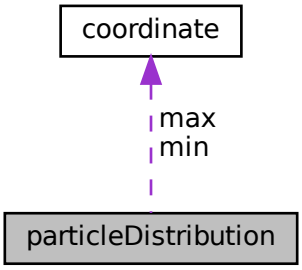
The documentation for this struct was generated from the following file:

- source/particle.cuh

5.9 particleDistribution Struct Reference

All information necessary to generate the initial particles.

Collaboration diagram for particleDistribution:



Public Attributes

- struct coordinate min
  Min. coord. values.
- struct coordinate max
  Max. coord. values.
- var_type vmean
  Average velocity.
- var_type vsigma
  Standard deviation of velocity.
- var_type Rmean
  Mean radius.
- var_type Rsigma
  Standard deviation of the radius.

5.9.1 Detailed Description

All information necessary to generate the initial particles.

5.9.2 Member Data Documentation

5.9.2.1 max

```
struct coordinate particleDistribution::max
```

Max. coord. values.

5.9.2.2 min

```
struct coordinate particleDistribution::min
```

Min. coord. values.

5.9.2.3 Rmean

```
var_type particleDistribution::Rmean
```

Mean radius.

5.9.2.4 Rsigma

```
var_type particleDistribution::Rsigma
```

Standard deviation of the radius.

5.9.2.5 vmean

```
var_type particleDistribution::vmean
```

Average velocity.

5.9.2.6 vsigma

```
var_type particleDistribution::vsigma
```

Standard deviation of velocity.

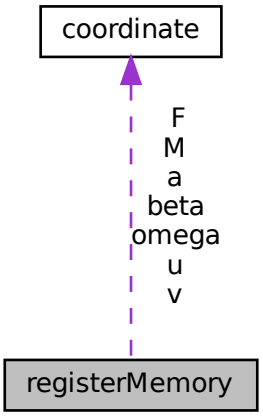
The documentation for this struct was generated from the following file:

- source/particle.cuh

5.10 registerMemory Struct Reference

Register memory, read at the beginning of the kernel and used throughout to store all the particle data locally.

Collaboration diagram for registerMemory:



Public Attributes

- struct coordinate u
- struct coordinate v
- struct coordinate a [AccelerationStored]
- struct coordinate omega
- struct coordinate beta [AccelerationStored]
- struct coordinate F
- struct coordinate M
- var_type R
- var_type m
- var_type m_rec
- var_type theta_rec
- var_type R_rec
- int material
- int cid

5.10.1 Detailed Description

Register memory, read at the beginning of the kernel and used throughout to store all the particle data locally.

5.10.2 Member Data Documentation

5.10.2.1 a

```
struct coordinate registerMemory::a[AccelerationStored]
```

5.10.2.2 beta

```
struct coordinate registerMemory::beta[AccelerationStored]
```

5.10.2.3 cid

```
int registerMemory::cid
```

5.10.2.4 F

```
struct coordinate registerMemory::F
```

5.10.2.5 M

```
struct coordinate registerMemory::M
```

5.10.2.6 m

```
var_type registerMemory::m
```

5.10.2.7 m_rec

```
var_type registerMemory::m_rec
```

5.10.2.8 material

```
int registerMemory::material
```

5.10.2.9 omega

```
struct coordinate registerMemory::omega
```

5.10.2.10 R

```
var_type registerMemory::R
```

5.10.2.11 R_rec

```
var_type registerMemory::R_rec
```

5.10.2.12 theta_rec

```
var_type registerMemory::theta_rec
```

5.10.2.13 u

```
struct coordinate registerMemory::u
```

5.10.2.14 v

```
struct coordinate registerMemory::v
```

The documentation for this struct was generated from the following file:

- source/registers.cuh

5.11 timestepping Struct Reference

Timestep settings.

Public Member Functions

- timestepping (var_type start, var_type end, var_type dt, int saveSteps)
  initialize based on the number of steps between saves
- timestepping (var_type start, var_type end, var_type dt, var_type savetime)
  initialize based on the time between saves

Public Attributes

- var_type starttime
  Start time of the simulation.
- var_type endtime
  End time of the simulation.
- var_type dt
  Timestep size.
- var_type savetime
  Frequency of saves.
- int numberOfSteps
  Number of steps in the simulation.
- int saveSteps
  Number of steps between saves.

5.11.1 Detailed Description

Timestep settings.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 timestepping() [1/2]

```
timestepping::timestepping (
            var_type start,
            var_type end,
            var_type dt,
            int saveSteps )  [inline]
```

initialize based on the number of steps between saves

5.11.2.2 timestepping() [2/2]

```
timestepping::timestepping (
            var_type start,
            var_type end,
            var_type dt,
            var_type savetime )  [inline]
```

initialize based on the time between saves

5.11.3 Member Data Documentation

5.11.3.1 dt

```
var_type timestepping::dt
```

Timestep size.

5.11.3.2 endtime

```
var_type timestepping::endtime
```

End time of the simulation.

5.11.3.3 numberOfSteps

```
int timestepping::numberOfSteps
```

Number of steps in the simulation.

5.11.3.4 saveSteps

```
int timestepping::saveSteps
```

Number of steps between saves.

5.11.3.5 savetime

```
var_type timestepping::savetime
```

Frequency of saves.

5.11.3.6 starttime

```
var_type timestepping::starttime
```

Start time of the simulation.

The documentation for this struct was generated from the following file:

- source/timestep.cuh

5.12 vector Struct Reference

3D vector

Public Member Functions

- __device__ __host__ vector (var_type x=constant::ZERO, var_type y=constant::ZERO, var_type z=constant::ZERO)
- __device__ vector operator+ (const vector &other) const

  addition
- __device__ __host__ vector operator- (const vector &other) const

  subtraction
- __device__ __host__ var_type operator∗ (const vector &other) const

  dot product
- __device__ __host__ vector operator∗ (var_type scalar) const

  multiplication with scalar
- __device__ __host__ vector operator^ (const vector &other) const

  cross product
- __device__ __host__ var_type length () const

  length

Public Attributes

- var_type x
- var_type y
- var_type z

5.12.1 Detailed Description

3D vector

5.12.2 Constructor & Destructor Documentation

5.12.2.1 vector()

```
__device__ __host__ vector::vector (
            var_type x = constant::ZERO,
            var_type y = constant::ZERO,
            var_type z = constant::ZERO )  [inline]
```

5.12.3 Member Function Documentation

5.12.3.1 length()

```
__device__ __host__ var_type vector::length ( ) const  [inline]
```

length

5.12.3.2 operator∗() [1/2]

```
__device__ __host__ var_type vector::operator* (
            const vector & other ) const  [inline]
```

dot product

5.12.3.3 operator*() [2/2]

```
__device__ __host__ vector vector::operator* (
            var_type scalar ) const  [inline]
```

multiplication with scalar

5.12.3.4 operator+()

```
__device__ vector vector::operator+ (
            const vector & other ) const  [inline]
```

addition

5.12.3.5 operator-()

```
__device__ __host__ vector vector::operator- (
            const vector & other ) const  [inline]
```

subtraction

5.12.3.6 operator^()

```
__device__ __host__ vector vector::operator^ (
            const vector & other ) const  [inline]
```

cross product

5.12.4 Member Data Documentation

5.12.4.1 x

```
var_type vector::x
```

5.12.4.2 y

```
var_type vector::y
```

5.12.4.3 z

```
var_type vector::z
```

The documentation for this struct was generated from the following file:

- source/math.cuh

Chapter 6

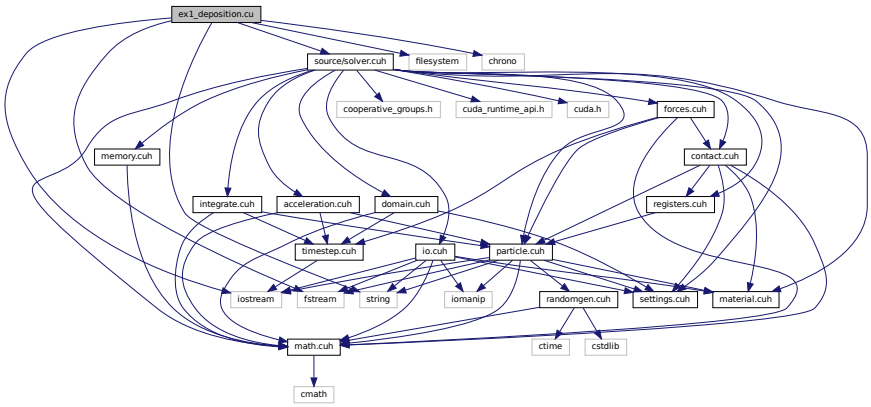
File Documentation

6.1 ex1_deposition.cu File Reference

Gravitational deposition example.

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include <string>
#include <chrono>
#include "source/solver.cuh"
```

Include dependency graph for ex1_deposition.cu:



Functions

- int main (int argc, char const *argv[])

Variables

- constexpr int NumberOfParticles = 2048
- constexpr int NumberOfMaterials = 1
- constexpr int NumberOfBoundaries = 5

6.1.1 Detailed Description

Gravitational deposition example.

Author

Dániel NAGY

Version

1.0

Date

2023.08.04.

This code simulates the deposition of N particles. Material data

- R = 60mm ± 2mm
- E = 2G = 200GPa
- Rho = 2000 kg/m^3
- mu = 0.5, mu0 = 0.7, mur = 0.02
- e = 0.1 Domain
- Layout = 2m x 2m

Author

Dániel NAGY

Version

1.0

Date

2023.08.04.

This code simulates the deposition of 8192 particles. The wall uses the Hertz model too. The particles are stored in the data/ex3_input.vtu input file.

- E = G = 20MPa
- Rho = 1000 kg/m^3
- mu = 0.5, mu0 = 0.7, mur = 0.02
- beta = 1.5 Domain
- Layout = 2m x 2m

Author

Dániel NAGY

Version

1.0

Date

2023.08.04.

This code simulates the deposition of a denser and a lighter material. The dense material has rho=1000kg/m3 and light material has rho=200kg/m3

Domain

- Layout = 2m x 2m

Author

Dániel NAGY

Version

1.0

Date

2023.08.04.

This code simulates the deposition of particles with special STL geometry.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Validation with EDEM

6.1.2 Function Documentation

6.1.2.1 main()

```
int main (
            int argc,
            char const * argv[] )
```

6.1.3 Variable Documentation

6.1.3.1 NumberOfBoundaries

```
constexpr int NumberOfBoundaries = 5  [constexpr]
```

6.1.3.2 NumberOfMaterials

```
constexpr int NumberOfMaterials = 1  [constexpr]
```

6.1.3.3 NumberOfParticles

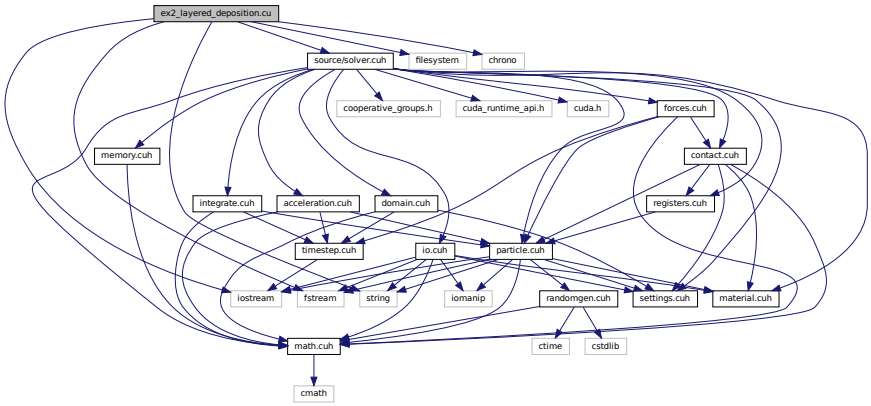
```
constexpr int NumberOfParticles = 2048  [constexpr]
```

6.2 ex2_layered_deposition.cu File Reference

Gravitational deposition in layers example.

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include <string>
#include <chrono>
#include "source/solver.cuh"
```

Include dependency graph for ex2_layered_deposition.cu:



Functions

- int main (int argc, char const ∗argv[ ])

Variables

- constexpr int NumberOfParticles = 8192
- constexpr int NumberOfMaterials = 1
- constexpr int NumberOfBoundaries = 5
- int particlesPerLayer = 1024
- int numberOfLayers = 8 + 1

6.2.1 Detailed Description

Gravitational deposition in layers example.

Author

Dániel NAGY

Version

1.0

Date

2023.08.04.

This code simulates the deposition of N particles. The particles are deposited in layers. Material data

- R = 40mm ± 10mm
- E = 2G = 200GPa
- Rho = 2000 kg/m^3
- mu = 0.5, mu0 = 0.7, mur = 0.02
- e = 0.1 Domain
- Layout = 2m x 2m

6.2.2 Function Documentation

6.2.2.1 main()

```
int main (
            int argc,
            char const * argv[] )
```

6.2.3 Variable Documentation

6.2.3.1 NumberOfBoundaries

```
constexpr int NumberOfBoundaries = 5  [constexpr]
```

6.2.3.2 numberOfLayers

```
int numberOfLayers = 8 + 1
```

6.2.3.3 NumberOfMaterials

```
constexpr int NumberOfMaterials = 1  [constexpr]
```

6.2.3.4 NumberOfParticles

```
constexpr int NumberOfParticles = 8192  [constexpr]
```

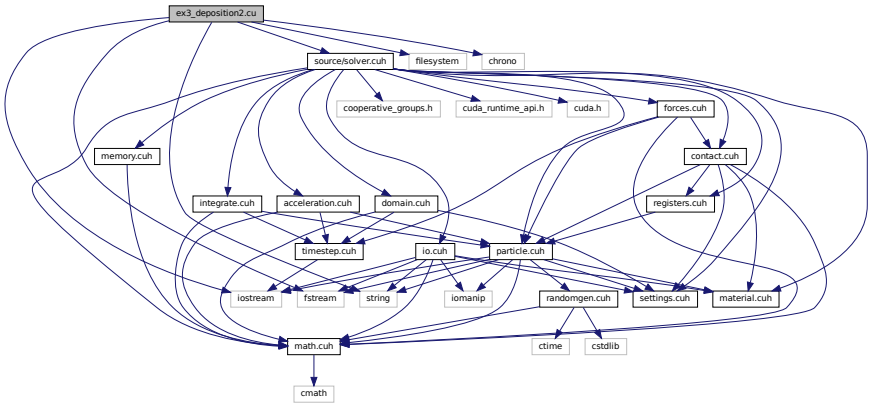
6.2.3.5 particlesPerLayer

```
int particlesPerLayer = 1024
```

6.3 ex3_deposition2.cu File Reference

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include <string>
#include <chrono>
#include "source/solver.cuh"
```

Include dependency graph for ex3_deposition2.cu:



Functions

- int main (int argc, char const *argv[])

Variables

- constexpr int NumberOfParticles = 8192
- constexpr int NumberOfMaterials = 2
- constexpr int NumberOfBoundaries = 5

6.3.1 Function Documentation

6.3.1.1 main()

```
int main (
            int argc,
            char const * argv[] )
```

6.3.2 Variable Documentation

6.3.2.1 NumberOfBoundaries

```
constexpr int NumberOfBoundaries = 5  [constexpr]
```

6.3.2.2 NumberOfMaterials

```
constexpr int NumberOfMaterials = 2  [constexpr]
```

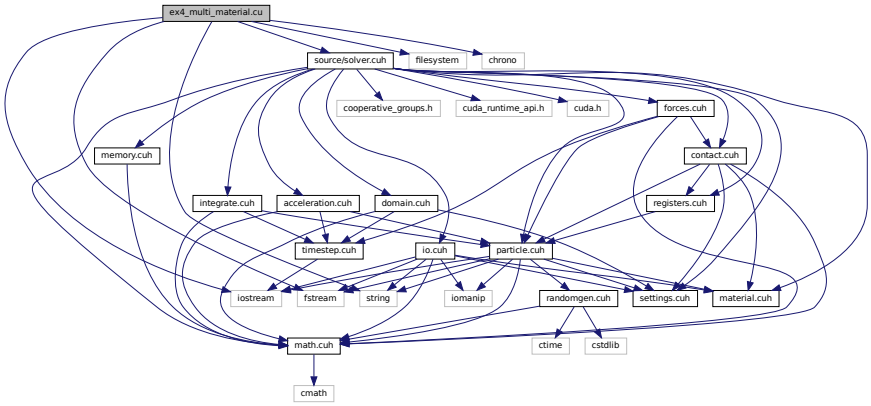
6.3.2.3 NumberOfParticles

```
constexpr int NumberOfParticles = 8192  [constexpr]
```

6.4 ex4_multi_material.cu File Reference

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include <string>
#include <chrono>
#include "source/solver.cuh"
```

Include dependency graph for ex4_multi_material.cu:



Functions

- int main (int argc, char const *argv[])

Variables

- constexpr int NumberOfParticles = 4096
- constexpr int NumberOfMaterials = 3
- constexpr int NumberOfBoundaries = 5

6.4.1 Function Documentation

6.4.1.1 main()

```
int main (
            int argc,
            char const * argv[] )
```

6.4.2 Variable Documentation

6.4.2.1 NumberOfBoundaries

```
constexpr int NumberOfBoundaries = 5  [constexpr]
```

6.4.2.2 NumberOfMaterials

```
constexpr int NumberOfMaterials = 3  [constexpr]
```

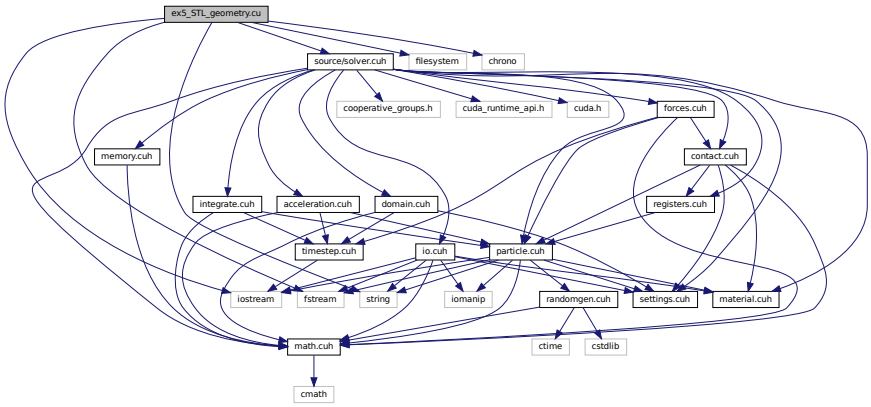
6.4.2.3 NumberOfParticles

```
constexpr int NumberOfParticles = 4096  [constexpr]
```

6.5 ex5_STL_geometry.cu File Reference

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include <string>
#include <chrono>
#include "source/solver.cuh"
```

Include dependency graph for ex5_STL_geometry.cu:



Macros

- #define SQ2 0.7071067812f

Functions

- int main (int argc, char const *argv[])

Variables

- constexpr int NumberOfParticles = 2048
- constexpr int NumberOfMaterials = 2
- constexpr int NumberOfBoundaries = 16

6.5.1 Macro Definition Documentation

6.5.1.1 SQ2

```
#define SQ2 0.7071067812f
```

6.5.2 Function Documentation

6.5.2.1 main()

```
int main (
            int argc,
            char const * argv[] )
```

6.5.3 Variable Documentation

6.5.3.1 NumberOfBoundaries

```
constexpr int NumberOfBoundaries = 16  [constexpr]
```

6.5.3.2 NumberOfMaterials

```
constexpr int NumberOfMaterials = 2  [constexpr]
```

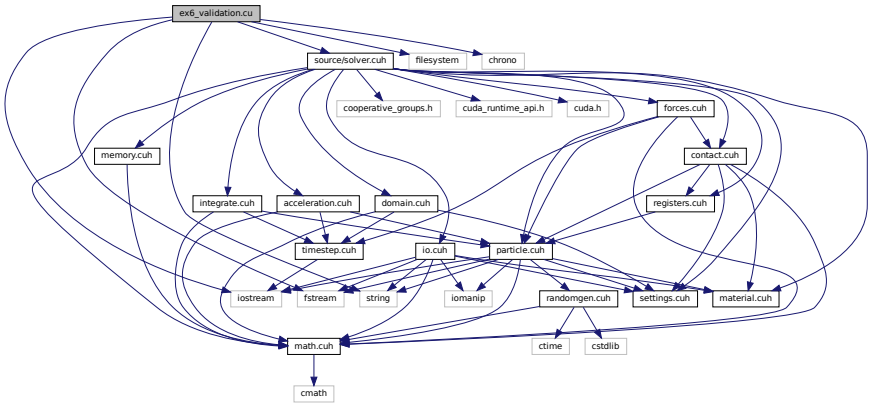
6.5.3.3 NumberOfParticles

```
constexpr int NumberOfParticles = 2048  [constexpr]
```

6.6 ex6_validation.cu File Reference

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include <string>
#include <chrono>
#include "source/solver.cuh"
```

Include dependency graph for ex6_validation.cu:



Functions

- int main (int argc, char const *argv[])

Variables

- constexpr int NumberOfParticles = 8192
- constexpr int NumberOfMaterials = 1
- constexpr int NumberOfBoundaries = 5

6.6.1 Function Documentation

6.6.1.1 main()

```
int main (
            int argc,
            char const * argv[] )
```

6.6.2 Variable Documentation

6.6.2.1 NumberOfBoundaries

```
constexpr int NumberOfBoundaries = 5  [constexpr]
```

6.6.2.2 NumberOfMaterials

```
constexpr int NumberOfMaterials = 1  [constexpr]
```

6.6.2.3 NumberOfParticles

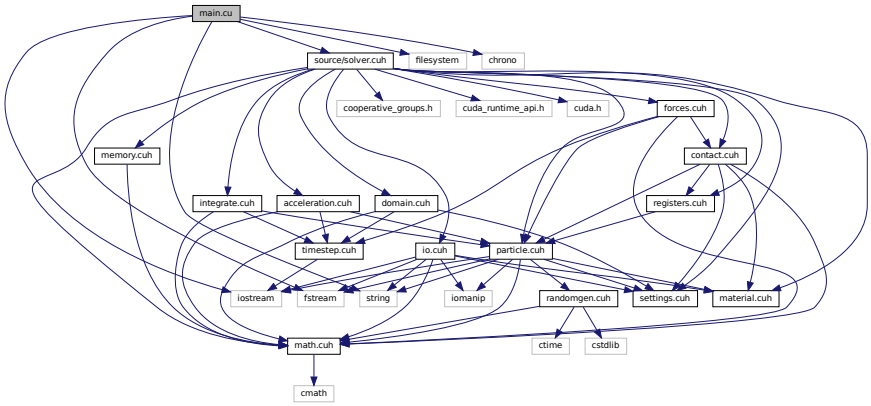
```
constexpr int NumberOfParticles = 8192  [constexpr]
```

6.7 main.cu File Reference

Main part, case definition.

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include <string>
#include <chrono>
#include "source/solver.cuh"
```

Include dependency graph for main.cu:



Functions

- int main (int argc, char const *argv[])

6.7.1 Detailed Description

Main part, case definition.

Author

Dániel NAGY

Version

1.0

Date

2023.07.20.

Handling of I/O, calling functions...

6.7.2 Function Documentation

6.7.2.1 main()

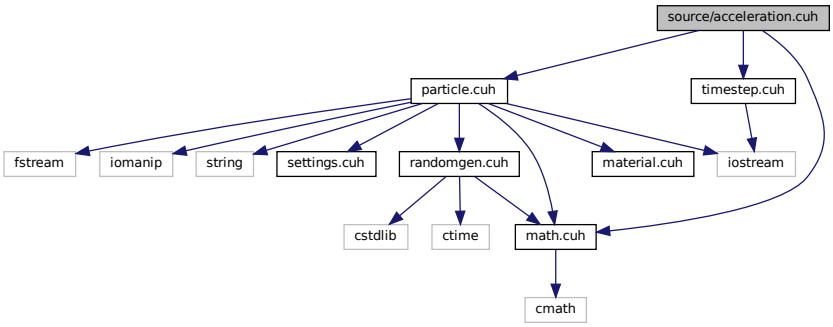
```
int main (
            int argc,
            char const * argv[] )
```

6.8 source/acceleration.cuh File Reference

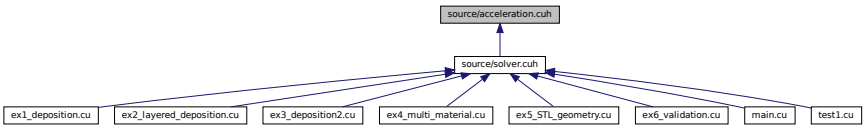
Calculates the acceleration.

```
#include "particle.cuh"
#include "timestep.cuh"
#include "math.cuh"
```

Include dependency graph for acceleration.cuh:



This graph shows which files directly or indirectly include this file:



Namespaces

- accelerationHandling

  Acceleration handling of particles.

Macros

- #define acceleration_H

Functions

- __device__ void accelerationHandling::calculateDefault (int tid, struct registerMemory &rmem, struct particle particles)

  Calculates the acceleration of the particles.
- __device__ void accelerationHandling::addBodyForces (int tid, struct registerMemory &rmem, struct particle particles, struct bodyForce bodyForces)

  Modifies the acceleration with the body forces terms.

6.8.1 Detailed Description

Calculates the acceleration.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Functions to do it

6.8.2 Macro Definition Documentation

6.8.2.1 acceleration_H

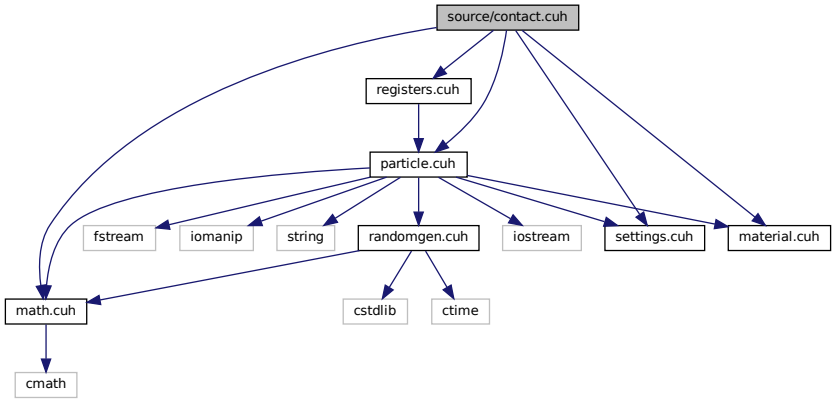
```
#define acceleration_H
```

6.9 source/contact.cuh File Reference

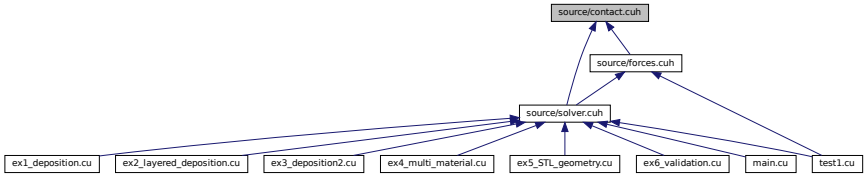
Contact search algorithms.

```
#include "particle.cuh"
#include "material.cuh"
#include "math.cuh"
#include "registers.cuh"
#include "settings.cuh"
```

Include dependency graph for contact.cuh:



This graph shows which files directly or indirectly include this file:



Classes

- struct contact

  Contact data between particles, stored in the registers (preferably)

Namespaces

- contactHandling

  Contact handling of particles.

Macros

- #define contact_H

Functions

- __device__ bool contactHandling::areNeighbours (int cid1, int cid2)

  Checks if two cells are neighbours or not.
- void __device__ contactHandling::CalculateContact (int tid, struct registerMemory &rmem, int i, var_type d, var_type Rs, struct particle particles, struct contact &contacts)

  Calculates the contact parameters between two particles.
- void __device__ contactHandling::ResetContacts (int tid, struct contact &contacts)

  Prepares the contact struct for the next timestep by copying deltat into deltat_last for each contact.
- void __device__ contactHandling::BruteForceContactSearch (int tid, struct registerMemory &rmem, int numberOfActiveParticles, struct particle particles, struct contact &contacts)

  Brute force contact search, which goes through all possible combinations and calculates all contacts.
- void __device__ contactHandling::CalculateCellId (int tid, struct registerMemory &rmem, int numberOf↩ ActiveParticles, struct particle particles)

  Calculates the cell id.
- void __device__ contactHandling::DecomposedDomainsContactSearch (int tid, struct registerMemory &rmem, int numberOfActiveParticles, struct particle particles, struct contact &contacts)

  Decomposed domains contact search, which checks if particles are in the same or neighbouring cells and calculates all contacts.
- void __device__ contactHandling::initializeContacts (int tid, struct contact &contacts)

  Initializes the contacts struct at the beginning of the solver kernel.

Variables

- constexpr __device__ int contactHandling::Neighbours [27]

  calculate neighbours on compile time

6.9.1 Detailed Description

Contact search algorithms.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Contains the contact search algorithms. The following contact search algorithms implemented:

- BruteForce: calculates ALL possible contacts
- DecomposedDomains: only calculates contact if in a neghbouring cell
- DecomposedDomainsFast: EXPERIMENTAL

6.9.2 Macro Definition Documentation

6.9.2.1 contact_H

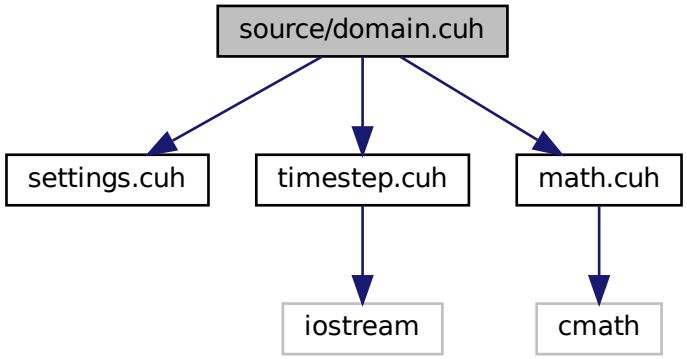
```
#define contact_H
```

6.10 source/domain.cuh File Reference

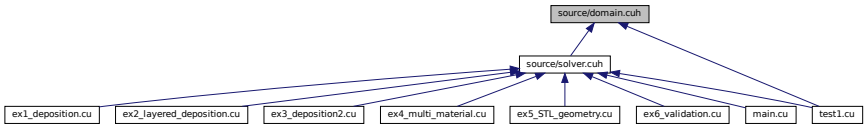
Description of the simulation domain.

```
#include "settings.cuh"
#include "timestep.cuh"
#include "math.cuh"
```

Include dependency graph for domain.cuh:



This graph shows which files directly or indirectly include this file:



Classes

- struct boundaryCondition

  Contains all the data about boundary conditions.

Namespaces

- domainHandling

  Handling of boundary conditions and STL files.

Macros

- #define domain_H

Enumerations

- enum BoundaryConditionType { None , ReflectiveWall , HertzWall }

Functions

- __device__ void domainHandling::CalculateOverlap (int tid, struct registerMemory &rmem, int i, var_type d, struct contact &contacts)

  Calculates the contact parameters between a particle and a boundary.
- __device__ void domainHandling::applyBoundaryConditions (int tid, struct registerMemory &rmem, struct particle particles, struct boundaryCondition boundaryConditions, struct contact &contacts, struct materialParameters pars, struct timestepping timestep)

  Calculates the forces based on the boundary constraints and given model.

6.10.1 Detailed Description

Description of the simulation domain.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Contains the boundary description (STL or Rectangular) and the boundary condition calculations

6.10.2 Macro Definition Documentation

6.10.2.1 domain_H

```
#define domain_H
```

6.10.3 Enumeration Type Documentation

6.10.3.1 BoundaryConditionType

```
enum BoundaryConditionType
```

Enumerator

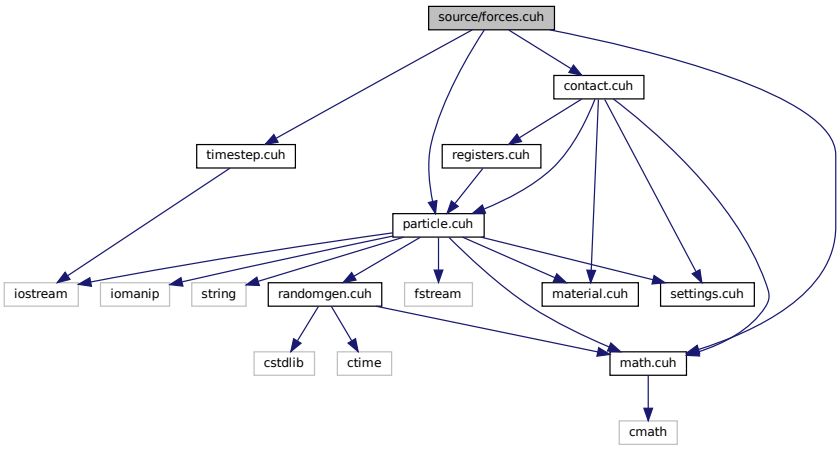
None	
ReflectiveWall	
HertzWall	

6.11 source/forces.cuh File Reference

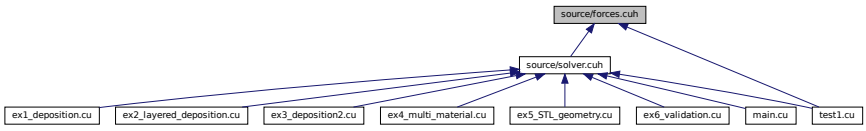
Force calculations.

```
#include "particle.cuh"
#include "math.cuh"
#include "contact.cuh"
#include "timestep.cuh"
```

Include dependency graph for forces.cuh:



This graph shows which files directly or indirectly include this file:



Classes

- struct bodyForce

  Stores the constant body forces (e.g. gravity) in the different directions.

Namespaces

- forceHandling

  Contains all the functions to calculate the force between particles.

Macros

- #define forces_H

Functions

- __device__ void forceHandling::calculateForceMindlin (int tid, struct registerMemory &rmem, struct particle particles, struct contact contacts, struct materialParameters pars, struct timestepping timestep)

  Calculates the force acting on the particle in x,y,z system using the Mindlin-Hertz theory.
- var_type forceHandling::calculateTotalKineticEnergy (struct particle particles, int numberOfActiveParticles)

  Calculates the total kinetic energy.
- var_type forceHandling::calculateTotalPotentialEnergy (struct particle particles, struct bodyForce bodyForces, int numberOfActiveParticles)

  Calculates the total potential energy.

6.11.1 Detailed Description

Force calculations.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Contains methods for force and energy calculations

6.11.2 Macro Definition Documentation

6.11.2.1 forces_H

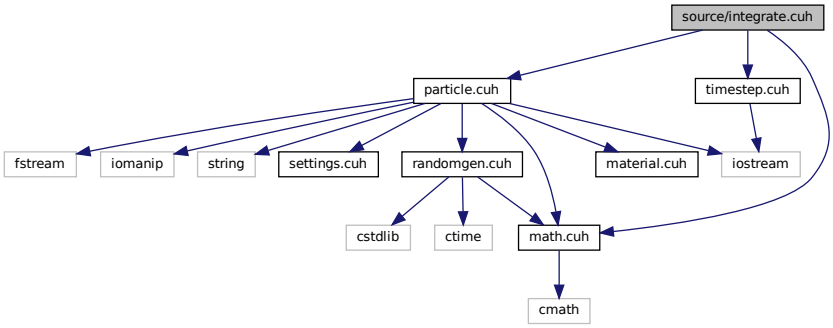
```
#define forces_H
```

6.12 source/integrate.cuh File Reference

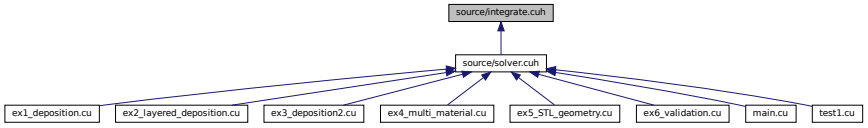
Numerical timestepping schemes.

```
#include "particle.cuh"
#include "timestep.cuh"
#include "math.cuh"
```

Include dependency graph for integrate.cuh:



This graph shows which files directly or indirectly include this file:



Namespaces

- integrators

  Contains the numerical methods and timestepping.

Macros

- #define integrate_H

Functions

- __device__ var_type integrators::RK1 (var_type dt, var_type x, var_type f)

  Calculates the next point using 1st order Runge-Kutta (Euler)
- __device__ var_type integrators::AB2 (var_type dt, var_type x, var_type f, var_type f_old)

  Calculates the next point using 2nd order Adams-Bashfort method.
- __device__ var_type integrators::AM2 (var_type dt, var_type x, var_type f, var_type f_old)

  Calculates the next point using 2nd order Adams-Moulton method.

- __device__ void integrators::euler (int tid, struct registerMemory &rmem, struct particle particles, struct timestepping timestep)

  Calculates the new vel., angular vel., and position using Euler's method.
- __device__ void integrators::exact (int tid, struct registerMemory &rmem, struct particle particles, struct timestepping timestep)

  Calculates the new vel., angular vel., and position exactly from the acceleration.
- __device__ void integrators::adams2 (int tid, struct registerMemory &rmem, struct particle particles, struct timestepping timestep, int step)

  Calculates the new vel., angular vel., and position using 2nd order Adams methods.

6.12.1 Detailed Description

Numerical timestepping schemes.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Contains all the integration methods

6.12.2 Macro Definition Documentation

6.12.2.1 integrate_H

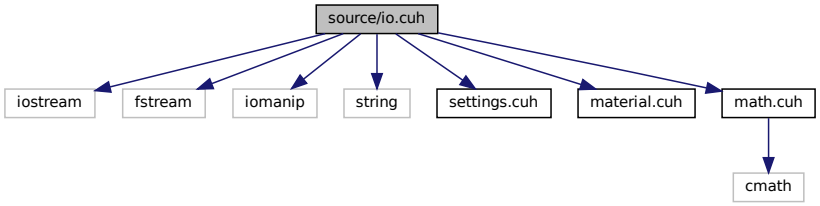
```
#define integrate_H
```

6.13 source/io.cuh File Reference

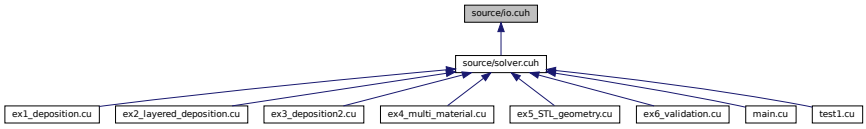
Input-output handling.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include "settings.cuh"
#include "material.cuh"
#include "math.cuh"
```

Include dependency graph for io.cuh:



This graph shows which files directly or indirectly include this file:



Namespaces

- ioHandling

  Contains all the functions for writing and reading data.

Macros

- #define io_H

Functions

- void ioHandling::saveParticles (int numberOfActiveParticles, struct particle particles, std::string location)

  Save the data of a list of particles in a .particle textfile.
- void ioHandling::saveParticlesVTK (int numberOfActiveParticles, struct particle particles, std::string location)

  Save the data of a list of particles as a vtk compatible .vtu unstructured grid file.
- int ioHandling::readParticlesVTK (struct particle particles, std::string location)

  Reads the particle data from a vtk compatible .vtu unstructured grid file.
- int ioHandling::readParticlesCSV (struct particle particles, std::string location)

  Reads the particle data from a .csv file.

6.13.1 Detailed Description

Input-output handling.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Functions to do it

6.13.2 Macro Definition Documentation

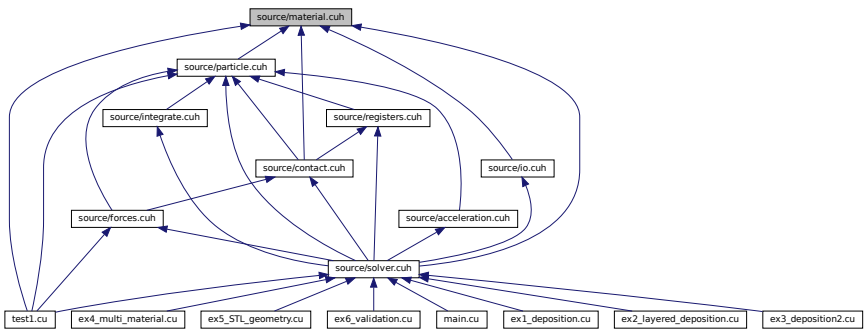
6.13.2.1 io_H

```
#define io_H
```

6.14 source/material.cuh File Reference

Material description.

This graph shows which files directly or indirectly include this file:



Classes

- struct materialContact

  Struct which contains the reduced quantities for material combinations.
- struct materialParameters

  Struct with all the user given material parameters, stored in the shared memory on the device side.

Namespaces

- materialHandling

  Contains all the function for material handling.

Macros

- #define material_H

Enumerations

- enum materialHandling::methods { materialHandling::Min , materialHandling::Max , materialHandling::HarmonicMean , materialHandling::Mean }

Functions

- void materialHandling::printMaterialInfo (struct materialParameters pars, bool printPairings=false)

  Prints all the materials and material combinations.
- void materialHandling::calculateMaterialContact (struct materialParameters &pars, methods friction, methods elastic, methods damping)

  Calculates all the material pairings and damping.

6.14.1 Detailed Description

Material description.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Struct to handle user given material parameters

6.14.2 Macro Definition Documentation

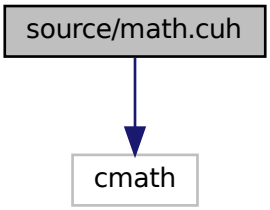
6.14.2.1 material_H

```
#define material_H
```

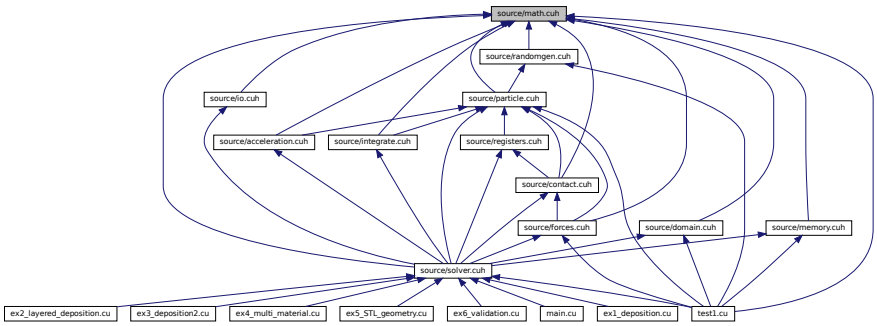
6.15 source/math.cuh File Reference

Math function.

```
#include <cmath>
```
Include dependency graph for math.cuh:



This graph shows which files directly or indirectly include this file:



Classes

- struct vector

  3D vector

Namespaces

- constant

  Contains all the constant.

Macros

- #define math_H
- #define CHECK(call)

  CUDA Check macro.

Typedefs

- typedef struct vector vec3D

Functions

- __device__ var_type calculateDistance (var_type x1, var_type y1, var_type z1, var_type x2, var_type y2, var_type z2)

  Calculate the distance between two points.
- __device__ var_type calculateNormal (var_type v1, var_type v2, var_type v3, var_type n1, var_type n2, var_type n3)

  Calculates the normal component of vector v using the unit vector n (pointing from particle 1 to particle 2)

Variables

- constexpr var_type constant::PI = 3.14159265358979323846 2643
- constexpr var_type constant::VOLUME_FACTOR = PI ∗ 4.0 / 3.0
- constexpr var_type constant::DAMPING = -1.8257418583505537115
- constexpr var_type constant::ZERO = 0.0
- constexpr var_type constant::NUMBER_04 = 0.4
- constexpr var_type constant::NUMBER_05 = 0.5
- constexpr var_type constant::NUMBER_4o3 = 4.0/3.0
- constexpr var_type constant::NUMBER_1 = 1.0
- constexpr var_type constant::NUMBER_2 = 2.0
- constexpr var_type constant::NUMBER_8 = 8.0
- constexpr var_type constant::AB2C1 = 1.5
- constexpr var_type constant::AB2C2 = 0.5
- constexpr var_type constant::AM2C1 = 0.5

6.15.1 Detailed Description

Math function.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Contains common functions

6.15.2 Macro Definition Documentation

6.15.2.1 CHECK

```
#define CHECK(
              call )
```

Value:

```
{ \
  const cudaError_t error = call; \
  if (error != cudaSuccess) \
  { \
  printf("Error: %s:%d, ", __FILE__, __LINE__); \
  printf("code:%d, reason: %s\n", error, cudaGetErrorString(error)); \
  exit(-10*error); \
  } \
}
```

CUDA Check macro.

6.15.2.2 math_H

```
#define math_H
```

6.15.3 Typedef Documentation

6.15.3.1 vec3D

```
typedef struct vector vec3D
```

6.15.4 Function Documentation

6.15.4.1 calculateDistance()

```
__device__ var_type calculateDistance (
             var_type x1,
             var_type y1,
             var_type z1,
             var_type x2,
             var_type y2,
             var_type z2 )  [inline]
```

Calculate the distance between two points.

Parameters

x1	x coord. of point 1
y1	y coord. of point 1
z1	z coord. of point 1
x2	x coord. of point 2
y2	y coord. of point 2
z2	z coord. of point 2

Returns

Distance between points

6.15.4.2 calculateNormal()

```
__device__ var_type calculateNormal (
             var_type v1,
             var_type v2,
             var_type v3,
             var_type n1,
             var_type n2,
             var_type n3 )  [inline]
```

Calculates the normal component of vector v using the unit vector n (pointing from particle 1 to particle 2)

Parameters

v1	1st component of vector v
v2	2nd component of vector v
v3	3rd component of vector v
n1	1st component of vector n
n2	2nd component of vector n
n3	3rd component of vector n

Returns

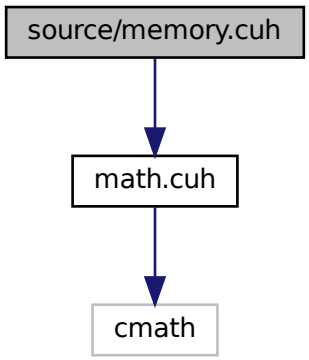
Returns the relative normal velocity

6.16 source/memory.cuh File Reference

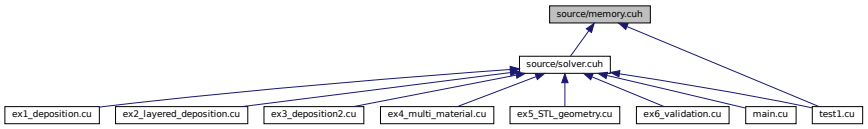
Memory allocation and synchronization of host and device side.

```
#include "math.cuh"
```

Include dependency graph for memory.cuh:



This graph shows which files directly or indirectly include this file:



Namespaces

- memoryHandling

  Memory handling functions which copy between CPU and GPU and allocate memory.

Macros

- #define memory_H

Enumerations

- enum memoryHandling::listOfVariables {
  memoryHandling::All , memoryHandling::Position , memoryHandling::Velocity , memoryHandling::AngularVelocity ,
  memoryHandling::Acceleration , memoryHandling::AngularAcceleration , memoryHandling::Material ,
  memoryHandling::Radius ,
  memoryHandling::Force , memoryHandling::Torque , memoryHandling::CellID }

Functions

- void memoryHandling::initializeHostParticles (struct particle &particlesH)

  Fills the host side for the particles with zeros.
- void memoryHandling::allocateHostParticles (struct particle &particlesH)

  Allocates the host side for the particles.
- void memoryHandling::freeHostParticles (struct particle &particlesH)

  Free the hist side.
- void memoryHandling::allocateDeviceParticles (struct particle &particlesD)

  Allocates the device side for the particles.
- void memoryHandling::freeDeviceParticles (struct particle &particlesD)

  Allocates the device side for the particles.
- void memoryHandling::synchronizeParticles (struct particle dest, struct particle source, listOfVariables vars, cudaMemcpyKind kind)

  Synchronizes the memory between host and device.

6.16.1 Detailed Description

Memory allocation and synchronization of host and device side.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Functions to synchronize data from H2D and D2H

6.16.2 Macro Definition Documentation

6.16.2.1 memory_H

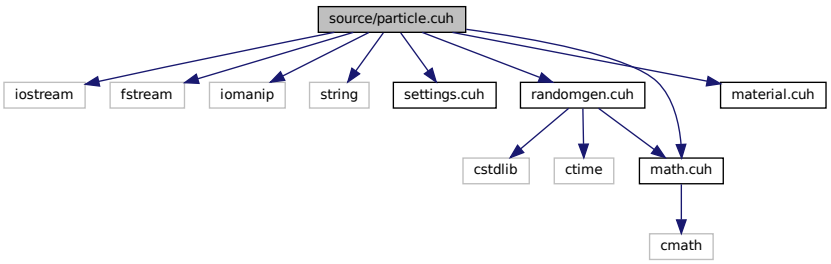
```
#define memory_H
```

6.17 source/particle.cuh File Reference

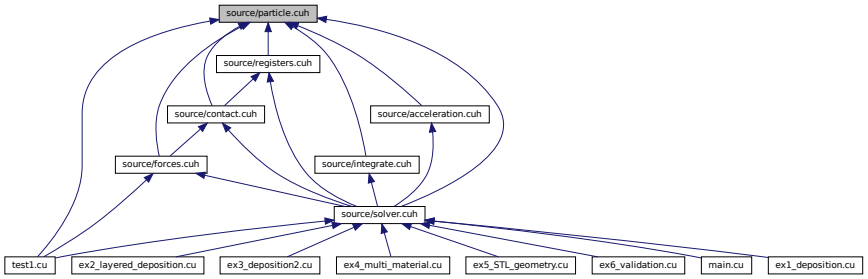
Particle and particle cloud discriptions.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include "settings.cuh"
#include "randomgen.cuh"
#include "material.cuh"
#include "math.cuh"
```

Include dependency graph for particle.cuh:



This graph shows which files directly or indirectly include this file:



Classes

- struct coordinate

  Cartesian coordinates.
- struct coordinates

  Cartesian coordinate vectors.
- struct particle

  Coordinates, velocity and radius of a particles.
- struct particleDistribution

  All information necessary to generate the initial particles.

Namespaces

- particleHandling

  Contains all the functions and structs necessary for handling the particles.

Macros

- #define particle_H

Enumerations

- enum class particleHandling::ParticleSizeDistribution { particleHandling::None , particleHandling::Uniform , particleHandling::Gauss }
- enum class particleHandling::ParticleVelocityDistribution { particleHandling::None , particleHandling::Uniform , particleHandling::Gauss }

Functions

- void particleHandling::generateParticleParameters (struct particle p, struct materialParameters pars, int mat_id, int start_id, int end_id)

  Fills up the particle struct p from a given material data.
- void particleHandling::generateParticleLocation (struct particle p, struct particleDistribution pdist, ParticleSizeDistribution psize_dist, ParticleVelocityDistribution pvel_dist)

  Particle generation based on the initial particle distribution.
- void particleHandling::generateParticleLocation (struct particle p, struct coordinates coords, var_type *radii, struct materialParameters pars)

  Particle generation based on coordinate lists.
- void particleHandling::printParticles (struct particle p)

  Print the data of a list of particles.

6.17.1 Detailed Description

Particle and particle cloud discriptions.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Device and host side instance of a particle

6.17.2 Macro Definition Documentation

6.17.2.1 particle_H

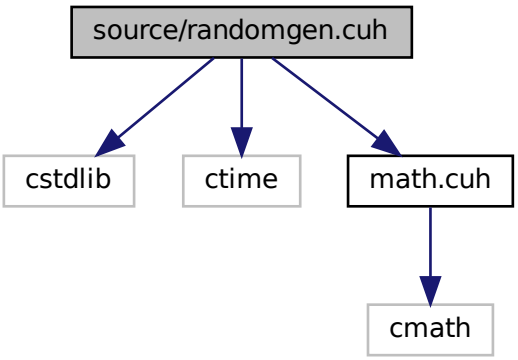
```
#define particle_H
```

6.18 source/randomgen.cuh File Reference

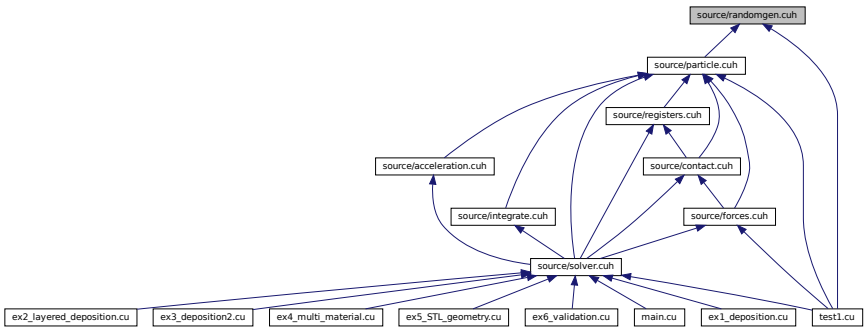
Random number generation.

```
#include <cstdlib>
#include <ctime>
#include "math.cuh"
```

Include dependency graph for randomgen.cuh:



This graph shows which files directly or indirectly include this file:



Namespaces

- RandomGeneration

  Contains everything necessary for random generation.

Macros

- #define random_H

Functions

- void RandomGeneration::initializeRandomSeed ()

  Initializes a random seed based on time.
- void RandomGeneration::initializeRandomSeed (int seed)

  Initializes a random seed based on a given number.
- var_type RandomGeneration::randomInRange (var_type min, var_type max)

  Generates a random var_type in a range.

Variables

- var_type RandomGeneration::overRandMax = constant::NUMBER_1/var_type(RAND_MAX)

6.18.1 Detailed Description

Random number generation.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

Random number generations implemented

6.18.2 Macro Definition Documentation

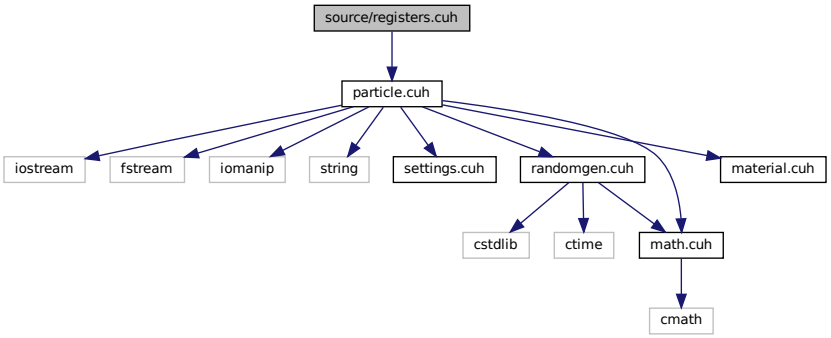
6.18.2.1 random_H

```
#define random_H
```

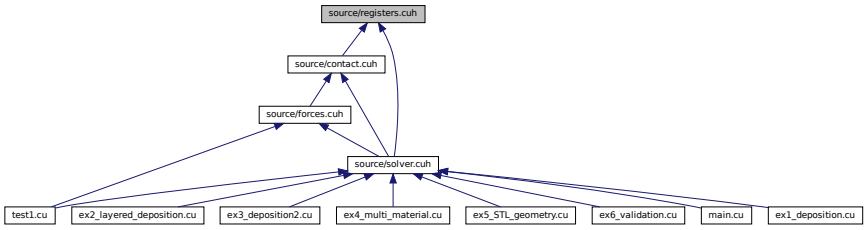
6.19 source/registers.cuh File Reference

Contains the register struct.

```
#include "particle.cuh"
```
Include dependency graph for registers.cuh:



This graph shows which files directly or indirectly include this file:



Classes

- struct registerMemory

  Register memory, read at the beginning of the kernel and used throughout to store all the particle data locally.

Namespaces

- registerHandling

  Register handling functions.

Macros

- #define register_H

Functions

- __device__ void registerHandling::fillRegisterMemory (int tid, struct registerMemory &rmem, struct particle particles)

  Copies the data from global memory to the registers.
- __device__ void registerHandling::endOfStepSync (int tid, struct registerMemory &rmem, struct particle particles)

  Saves the necessary data to the registers.
- __device__ void registerHandling::endOfKernelSync (int tid, struct registerMemory &rmem, struct particle particles)

  Saves the necessary data to the registers at the end of the kernel.

6.19.1 Detailed Description

Contains the register struct.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

This data is stored in the registers!

6.19.2 Macro Definition Documentation

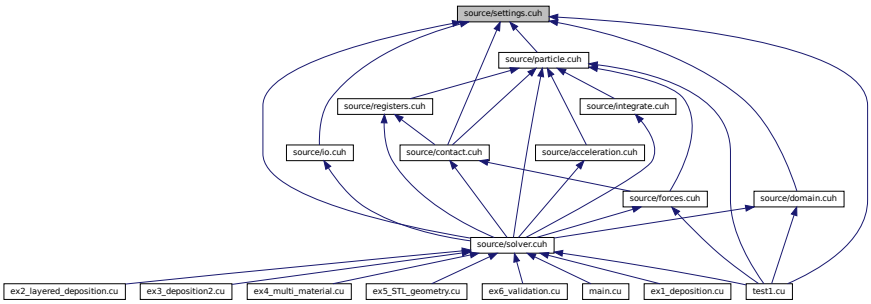
6.19.2.1 register_H

```
#define register_H
```

6.20 source/settings.cuh File Reference

Simulation settings, user given.

This graph shows which files directly or indirectly include this file:



Namespaces

- DecomposedDomainsConstants

Macros

- #define settings_H

Typedefs

- using var_type = float

  Variable type used in the code (float/double)

Enumerations

- enum class DomainType { Rectangular , STL }

  Domain settings.
- enum class ContactModel { Mindlin }

  Contact model.
- enum class ContactSearch { BruteForce , DecomposedDomains , DecomposedDomainsFast , Balanced }

  Contact search algorithm.
- enum class TimeIntegration { Euler , Exact , Adams2 }

  Time integration.
- enum class OutputFormat { ASCII , Binary }

  Time integration.

Variables

- constexpr int Debug = 0

  Debug: 0-Off, 1-Low level, 2-High level.
- constexpr bool UseGPUWideThreadSync = true

  Use cooperative groups for GPU Wide synchronization (required for energy conservation)
- constexpr int BlockSize = 64

  GPU settings.
- constexpr DomainType domainType = DomainType::STL
- constexpr int MaxContactNumber = 16

  Maximum number of contacts.
- constexpr bool BodyForce = true

  Body forces (gravity)
- constexpr bool RollingFriction = true
- constexpr ContactModel contactModel = ContactModel::Mindlin
- constexpr ContactSearch contactSearch = ContactSearch::BruteForce
- constexpr TimeIntegration timeIntegration = TimeIntegration::Exact
- constexpr int AccelerationStored = 1

  Previous accelerations stores, acceleration of particle tid is stored at tid + n∗NumberOfParticles.
- constexpr OutputFormat outputFormat = OutputFormat::ASCII
- constexpr bool SaveVelocity = true

  Save settings.
- constexpr bool SaveAngularVelocity = true
- constexpr bool SaveForce = false
- constexpr bool SaveTorque = false
- constexpr bool SaveId = false
- constexpr bool SaveMaterial = true
- constexpr int DecomposedDomainsConstants::Nx = 100

  Number of cell in x,y,z direction.
- constexpr int DecomposedDomainsConstants::Ny = 100
- constexpr int DecomposedDomainsConstants::Nz = 200
- constexpr var_type DecomposedDomainsConstants::minx = -1.0

  Min of coordinates.
- constexpr var_type DecomposedDomainsConstants::miny = -1.0
- constexpr var_type DecomposedDomainsConstants::minz = 0.0
- constexpr var_type DecomposedDomainsConstants::maxx = 1.0

  Max of coordinates.
- constexpr var_type DecomposedDomainsConstants::maxy = 1.0
- constexpr var_type DecomposedDomainsConstants::maxz = 4.0
- constexpr var_type DecomposedDomainsConstants::NoverDx = var_type(Nx)/(maxx-minx)

  DO NOT MODIFY - 1/max-min pre-calculated.
- constexpr var_type DecomposedDomainsConstants::NoverDy = var_type(Ny)/(maxy-miny)
- constexpr var_type DecomposedDomainsConstants::NoverDz = var_type(Nz)/(maxz-minz)

6.20.1 Detailed Description

Simulation settings, user given.

Author

Dániel NAGY

Version

1.0

Date

2023.09.12.

User given settings

6.20.2 Macro Definition Documentation

6.20.2.1 settings_H

```
#define settings_H
```

6.20.3 Typedef Documentation

6.20.3.1 var_type

```
using var_type = float
```

Variable type used in the code (float/double)

6.20.4 Enumeration Type Documentation

6.20.4.1 ContactModel

```
enum ContactModel  [strong]
```

Contact model.

Enumerator

Mindlin	

6.20.4.2 ContactSearch

enum ContactSearch [strong]

Contact search algorithm.

Enumerator

BruteForce	
DecomposedDomains	
DecomposedDomainsFast	
Balanced	

6.20.4.3 DomainType

enum DomainType [strong]

Domain settings.

Enumerator

Rectangular	
STL	

6.20.4.4 OutputFormat

enum OutputFormat [strong]

Time integration.

Enumerator

ASCII	
Binary	

6.20.4.5 TimeIntegration

enum TimeIntegration [strong]

Time integration.

Enumerator

Euler	
Exact	
Adams2	

6.20.5 Variable Documentation

6.20.5.1 AccelerationStored

```
constexpr int AccelerationStored = 1  [constexpr]
```

Previous accelerations stores, acceleration of particle tid is stored at tid + n∗NumberOfParticles.

6.20.5.2 BlockSize

```
constexpr int BlockSize = 64  [constexpr]
```

GPU settings.

6.20.5.3 BodyForce

```
constexpr bool BodyForce = true  [constexpr]
```

Body forces (gravity)

6.20.5.4 contactModel

```
constexpr ContactModel contactModel = ContactModel::Mindlin  [constexpr]
```

6.20.5.5 contactSearch

```
constexpr ContactSearch contactSearch = ContactSearch::BruteForce  [constexpr]
```

6.20.5.6 Debug

```
constexpr int Debug = 0  [constexpr]
```

Debug: 0-Off, 1-Low level, 2-High level.

6.20.5.7 domainType

```
constexpr DomainType domainType = DomainType::STL  [constexpr]
```

6.20.5.8 MaxContactNumber

```
constexpr int MaxContactNumber = 16  [constexpr]
```

Maximum number of contacts.

6.20.5.9 outputFormat

```
constexpr OutputFormat outputFormat = OutputFormat::ASCII  [constexpr]
```

6.20.5.10 RollingFriction

```
constexpr bool RollingFriction = true  [constexpr]
```

6.20.5.11 SaveAngularVelocity

```
constexpr bool SaveAngularVelocity = true  [constexpr]
```

6.20.5.12 SaveForce

```
constexpr bool SaveForce = false  [constexpr]
```

6.20.5.13 SaveId

```
constexpr bool SaveId = false  [constexpr]
```

6.20.5.14 SaveMaterial

```
constexpr bool SaveMaterial = true  [constexpr]
```

6.20.5.15 SaveTorque

```
constexpr bool SaveTorque = false  [constexpr]
```

6.20.5.16 SaveVelocity

```
constexpr bool SaveVelocity = true  [constexpr]
```

Save settings.

6.20.5.17 timeIntegration

```
constexpr TimeIntegration timeIntegration = TimeIntegration::Exact  [constexpr]
```

6.20.5.18 UseGPUWideThreadSync

```
constexpr bool UseGPUWideThreadSync = true  [constexpr]
```

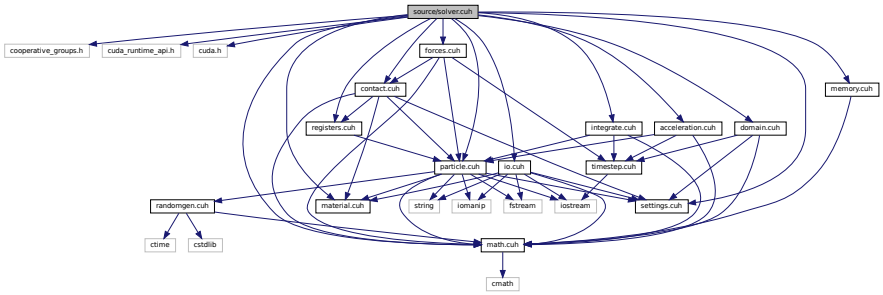
Use cooperative groups for GPU Wide synchronization (required for energy conservation)

6.21 source/solver.cuh File Reference

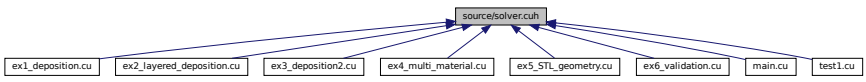
Solver algorithm on the device.

```
#include <cooperative_groups.h>
#include <cuda_runtime_api.h>
#include <cuda.h>
#include "particle.cuh"
#include "settings.cuh"
#include "contact.cuh"
#include "domain.cuh"
#include "forces.cuh"
#include "material.cuh"
#include "memory.cuh"
#include "integrate.cuh"
#include "acceleration.cuh"
#include "math.cuh"
#include "io.cuh"
#include "registers.cuh"
```

Include dependency graph for solver.cuh:



This graph shows which files directly or indirectly include this file:



Macros

- #define solver_H

Functions

- __global__ void solver (struct particle particles, int numberOfActiveParticles, struct materialParameters pars, struct timestepping timestep, struct bodyForce bodyForces, struct boundaryCondition boundaryConditions, int launch)

  Kernel using the perThread approach.

6.21.1 Detailed Description

Solver algorithm on the device.

Author

Dániel NAGY

Version

1.0

Date

2023.07.20.

GPU Code for the perThread calculations for each particle

6.21.2 Macro Definition Documentation

6.21.2.1 solver_H

```
#define solver_H
```

6.21.3 Function Documentation

6.21.3.1 solver()

```
__global__ void solver (
            struct particle particles,
            int numberOfActiveParticles,
            struct materialParameters pars,
            struct timestepping timestep,
            struct bodyForce bodyForces,
            struct boundaryCondition boundaryConditions,
            int launch )
```

Kernel using the perThread approach.

Parameters

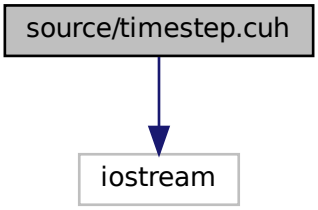
particles	All the particle data
numberOfActiveParticles	Number of currently active particles
pars	Material parameters
timestep	Timestep settings
bodyForces	Body forces (e.g. gravity)
boundaryConditions	Boundary conditions and types
launch	Number of the kernel launch

6.22 source/timestep.cuh File Reference

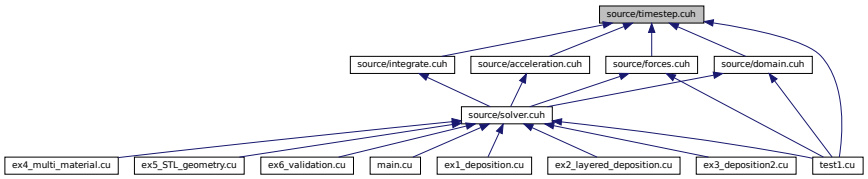
Timestepping settings.

```
#include <iostream>
```

Include dependency graph for timestep.cuh:



This graph shows which files directly or indirectly include this file:



Classes

- struct timestepping

  Timestep settings.

Namespaces

- timeHandling

  Functions for handling time.

Macros

- #define timestep_H

Functions

- void timeHandling::printTimestepSettings (struct timestepping timestep)

  Prints the timestep settings.

6.22.1 Detailed Description

Timestepping settings.

Author

Dániel NAGY

Version

1.0

Date

2023.07.20.

Device side integrator algorithms

6.22.2 Macro Definition Documentation

6.22.2.1 timestep_H

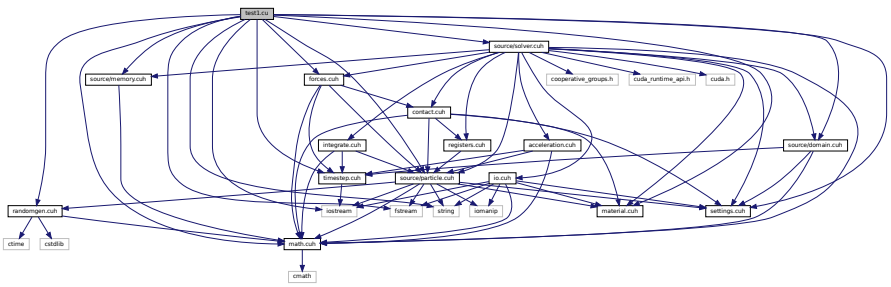
```
#define timestep_H
```

6.23 test1.cu File Reference

Main part, case definition.

```
#include <iostream>
#include <fstream>
#include <string>
#include "source/particle.cuh"
#include "source/domain.cuh"
#include "source/material.cuh"
#include "source/randomgen.cuh"
#include "source/settings.cuh"
#include "source/memory.cuh"
#include "source/solver.cuh"
#include "source/math.cuh"
#include "source/forces.cuh"
#include "source/timestep.cuh"
```

Include dependency graph for test1.cu:



Functions

- int main (int argc, char const ∗argv[ ])

6.23.1 Detailed Description

Main part, case definition.

Author

Dániel NAGY

Version

1.0

Date

2023.07.20.

Handling of I/O, calling functions...

6.23.2 Function Documentation

6.23.2.1 main()

```
int main (
            int argc,
            char const * argv[] )
```

Index