

# GPUDEM

1.0

Generated by Doxygen 1.9.1



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Namespace Documentation</b>	<b>7</b>
4.1 accelerationHandling Namespace Reference	7
4.1.1 Detailed Description	7
4.1.2 Function Documentation	7
4.1.2.1 addBodyForces()	7
4.1.2.2 calculateDefault()	8
4.2 constant Namespace Reference	8
4.2.1 Detailed Description	9
4.2.2 Variable Documentation	9
4.2.2.1 AB2C1	9
4.2.2.2 AB2C2	9
4.2.2.3 AM2C1	9
4.2.2.4 DAMPING	9
4.2.2.5 NUMBER_04	9
4.2.2.6 NUMBER_05	9
4.2.2.7 NUMBER_1	10
4.2.2.8 NUMBER_2	10
4.2.2.9 NUMBER_4o3	10
4.2.2.10 NUMBER_8	10
4.2.2.11 PI	10
4.2.2.12 VOLUME_FACTOR	10
4.2.2.13 ZERO	10
4.3 contactHandling Namespace Reference	10
4.3.1 Detailed Description	11
4.3.2 Function Documentation	11
4.3.2.1 areNeighbours()	11
4.3.2.2 BruteForceContactSearch()	12
4.3.2.3 CalculateCellId()	12
4.3.2.4 CalculateContact()	12
4.3.2.5 DecomposedDomainsContactSearch()	13
4.3.2.6 initializeContacts()	13
4.3.2.7 ResetContacts()	14
4.3.3 Variable Documentation	14
4.3.3.1 Neighbours	14

4.4 DecomposedDomainsConstants Namespace Reference	15
4.4.1 Variable Documentation	15
4.4.1.1 maxx	15
4.4.1.2 maxy	15
4.4.1.3 maxz	15
4.4.1.4 minx	16
4.4.1.5 miny	16
4.4.1.6 minz	16
4.4.1.7 NoverDx	16
4.4.1.8 NoverDy	16
4.4.1.9 NoverDz	16
4.4.1.10 Nx	16
4.4.1.11 Ny	17
4.4.1.12 Nz	17
4.5 domainHandling Namespace Reference	17
4.5.1 Detailed Description	17
4.5.2 Function Documentation	17
4.5.2.1 applyBoundaryConditions()	17
4.5.2.2 CalculateOverlap()	18
4.6 forceHandling Namespace Reference	18
4.6.1 Detailed Description	19
4.6.2 Function Documentation	19
4.6.2.1 calculateForceMindlin()	19
4.6.2.2 calculateTotalKineticEnergy()	19
4.6.2.3 calculateTotalPotentialEnergy()	20
4.7 integrators Namespace Reference	20
4.7.1 Detailed Description	20
4.7.2 Function Documentation	20
4.7.2.1 AB2()	21
4.7.2.2 adams2()	21
4.7.2.3 AM2()	21
4.7.2.4 euler()	22
4.7.2.5 exact()	22
4.7.2.6 RK1()	23
4.8 ioHandling Namespace Reference	23
4.8.1 Detailed Description	23
4.8.2 Function Documentation	23
4.8.2.1 readParticlesCSV()	23
4.8.2.2 readParticlesVTK()	24
4.8.2.3 saveParticles()	24
4.8.2.4 saveParticlesVTK()	25
4.9 materialHandling Namespace Reference	25

4.9.1 Detailed Description	25
4.9.2 Enumeration Type Documentation	25
4.9.2.1 methods	25
4.9.3 Function Documentation	26
4.9.3.1 calculateMaterialContact()	26
4.9.3.2 printMaterialInfo()	26
4.10 memoryHandling Namespace Reference	26
4.10.1 Detailed Description	27
4.10.2 Enumeration Type Documentation	27
4.10.2.1 listOfVariables	27
4.10.3 Function Documentation	28
4.10.3.1 allocateDeviceParticles()	28
4.10.3.2 allocateHostParticles()	28
4.10.3.3 freeDeviceParticles()	28
4.10.3.4 freeHostParticles()	28
4.10.3.5 initializeHostParticles()	29
4.10.3.6 synchronizeParticles()	29
4.11 particleHandling Namespace Reference	29
4.11.1 Detailed Description	30
4.11.2 Enumeration Type Documentation	30
4.11.2.1 ParticleSizeDistribution	30
4.11.2.2 ParticleVelocityDistribution	30
4.11.3 Function Documentation	31
4.11.3.1 generateParticleLocation() [1/2]	31
4.11.3.2 generateParticleLocation() [2/2]	31
4.11.3.3 generateParticleParameters()	32
4.11.3.4 printParticles()	32
4.12 RandomGeneration Namespace Reference	32
4.12.1 Detailed Description	33
4.12.2 Function Documentation	33
4.12.2.1 initializeRandomSeed() [1/2]	33
4.12.2.2 initializeRandomSeed() [2/2]	33
4.12.2.3 randomInRange()	33
4.12.3 Variable Documentation	34
4.12.3.1 overRandMax	34
4.13 registerHandling Namespace Reference	34
4.13.1 Detailed Description	34
4.13.2 Function Documentation	34
4.13.2.1 endOfKernelSync()	34
4.13.2.2 endOfStepSync()	35
4.13.2.3 fillRegisterMemory()	35
4.14 timeHandling Namespace Reference	35

---

4.14.1 Detailed Description . . . . .	35
4.14.2 Function Documentation . . . . .	36
4.14.2.1 printTimestepSettings() . . . . .	36
<b>5 Class Documentation</b>	<b>37</b>
5.1 bodyForce Struct Reference . . . . .	37
5.1.1 Detailed Description . . . . .	37
5.1.2 Member Data Documentation . . . . .	37
5.1.2.1 x . . . . .	37
5.1.2.2 y . . . . .	38
5.1.2.3 z . . . . .	38
5.2 boundaryCondition Struct Reference . . . . .	38
5.2.1 Detailed Description . . . . .	39
5.2.2 Member Data Documentation . . . . .	39
5.2.2.1 alpha . . . . .	39
5.2.2.2 beta . . . . .	39
5.2.2.3 gamma . . . . .	40
5.2.2.4 material . . . . .	40
5.2.2.5 n . . . . .	40
5.2.2.6 p . . . . .	40
5.2.2.7 s . . . . .	40
5.2.2.8 s_scale . . . . .	40
5.2.2.9 t . . . . .	40
5.2.2.10 t_scale . . . . .	41
5.2.2.11 type . . . . .	41
5.3 contact Struct Reference . . . . .	41
5.3.1 Detailed Description . . . . .	42
5.3.2 Member Data Documentation . . . . .	42
5.3.2.1 count . . . . .	42
5.3.2.2 deltan . . . . .	42
5.3.2.3 deltat . . . . .	42
5.3.2.4 deltat_last . . . . .	43
5.3.2.5 material . . . . .	43
5.3.2.6 mstar . . . . .	43
5.3.2.7 p . . . . .	43
5.3.2.8 r . . . . .	43
5.3.2.9 Rstar . . . . .	43
5.3.2.10 tid . . . . .	44
5.3.2.11 tid_last . . . . .	44
5.4 coordinate Struct Reference . . . . .	44
5.4.1 Detailed Description . . . . .	44
5.4.2 Member Data Documentation . . . . .	44

---

5.4.2.1 x . . . . .	44
5.4.2.2 y . . . . .	45
5.4.2.3 z . . . . .	45
5.5 coordinates Struct Reference . . . . .	45
5.5.1 Detailed Description . . . . .	45
5.5.2 Member Data Documentation . . . . .	45
5.5.2.1 x . . . . .	45
5.5.2.2 y . . . . .	46
5.5.2.3 z . . . . .	46
5.6 materialContact Struct Reference . . . . .	46
5.6.1 Detailed Description . . . . .	46
5.6.2 Member Data Documentation . . . . .	46
5.6.2.1 beta_star . . . . .	46
5.6.2.2 E_star . . . . .	47
5.6.2.3 G_star . . . . .	47
5.6.2.4 mu0_star . . . . .	47
5.6.2.5 mu_star . . . . .	47
5.6.2.6 mur_star . . . . .	47
5.7 materialParameters Struct Reference . . . . .	47
5.7.1 Detailed Description . . . . .	48
5.7.2 Member Data Documentation . . . . .	48
5.7.2.1 beta . . . . .	48
5.7.2.2 E . . . . .	48
5.7.2.3 e . . . . .	49
5.7.2.4 G . . . . .	49
5.7.2.5 mu . . . . .	49
5.7.2.6 mu0 . . . . .	49
5.7.2.7 mur . . . . .	49
5.7.2.8 nu . . . . .	49
5.7.2.9 pairing . . . . .	50
5.7.2.10 rho . . . . .	50
5.8 particle Struct Reference . . . . .	50
5.8.1 Detailed Description . . . . .	51
5.8.2 Member Data Documentation . . . . .	51
5.8.2.1 a . . . . .	51
5.8.2.2 beta . . . . .	52
5.8.2.3 cid . . . . .	52
5.8.2.4 F . . . . .	52
5.8.2.5 M . . . . .	52
5.8.2.6 m . . . . .	52
5.8.2.7 m_rec . . . . .	52
5.8.2.8 material . . . . .	53

5.8.2.9 omega	53
5.8.2.10 R	53
5.8.2.11 R_rec	53
5.8.2.12 theta	53
5.8.2.13 theta_rec	53
5.8.2.14 u	54
5.8.2.15 v	54
5.9 particleDistribution Struct Reference	54
5.9.1 Detailed Description	55
5.9.2 Member Data Documentation	55
5.9.2.1 max	55
5.9.2.2 min	55
5.9.2.3 Rmean	55
5.9.2.4 Rsigma	55
5.9.2.5 vmean	55
5.9.2.6 vsigma	56
5.10 registerMemory Struct Reference	56
5.10.1 Detailed Description	57
5.10.2 Member Data Documentation	57
5.10.2.1 a	57
5.10.2.2 beta	57
5.10.2.3 cid	57
5.10.2.4 F	57
5.10.2.5 M	57
5.10.2.6 m	57
5.10.2.7 m_rec	58
5.10.2.8 material	58
5.10.2.9 omega	58
5.10.2.10 R	58
5.10.2.11 R_rec	58
5.10.2.12 theta_rec	58
5.10.2.13 u	58
5.10.2.14 v	59
5.11 timestepping Struct Reference	59
5.11.1 Detailed Description	59
5.11.2 Constructor & Destructor Documentation	59
5.11.2.1 timestepping() [1/2]	60
5.11.2.2 timestepping() [2/2]	60
5.11.3 Member Data Documentation	60
5.11.3.1 dt	60
5.11.3.2 endtime	60
5.11.3.3 numberOfSteps	60



5.11.3.4 saveSteps	61
5.11.3.5 savetime	61
5.11.3.6 starttime	61
5.12 vector Struct Reference	61
5.12.1 Detailed Description	62
5.12.2 Constructor & Destructor Documentation	62
5.12.2.1 vector()	62
5.12.3 Member Function Documentation	62
5.12.3.1 length()	62
5.12.3.2 operator*() [1/2]	62
5.12.3.3 operator*() [2/2]	63
5.12.3.4 operator+()	63
5.12.3.5 operator-()	63
5.12.3.6 operator^()	63
5.12.4 Member Data Documentation	63
5.12.4.1 x	63
5.12.4.2 y	63
5.12.4.3 z	63
<b>6 File Documentation</b>	<b>65</b>
6.1 ex1_deposition.cu File Reference	65
6.1.1 Detailed Description	66
6.1.2 Function Documentation	67
6.1.2.1 main()	68
6.1.3 Variable Documentation	68
6.1.3.1 NumberOfBoundaries	68
6.1.3.2 NumberOfMaterials	68
6.1.3.3 NumberOfParticles	68
6.2 ex2_layered_deposition.cu File Reference	68
6.2.1 Detailed Description	69
6.2.2 Function Documentation	69
6.2.2.1 main()	69
6.2.3 Variable Documentation	70
6.2.3.1 NumberOfBoundaries	70
6.2.3.2 numberOfLayers	70
6.2.3.3 NumberOfMaterials	70
6.2.3.4 NumberOfParticles	70
6.2.3.5 particlesPerLayer	70
6.3 ex3_deposition2.cu File Reference	71
6.3.1 Function Documentation	71
6.3.1.1 main()	71
6.3.2 Variable Documentation	71

6.3.2.1 NumberOfBoundaries . . . . .	72
6.3.2.2 NumberOfMaterials . . . . .	72
6.3.2.3 NumberOfParticles . . . . .	72
6.4 ex4_multi_material.cu File Reference . . . . .	72
6.4.1 Function Documentation . . . . .	73
6.4.1.1 main() . . . . .	73
6.4.2 Variable Documentation . . . . .	73
6.4.2.1 NumberOfBoundaries . . . . .	73
6.4.2.2 NumberOfMaterials . . . . .	73
6.4.2.3 NumberOfParticles . . . . .	73
6.5 ex5_STL_geometry.cu File Reference . . . . .	74
6.5.1 Macro Definition Documentation . . . . .	74
6.5.1.1 SQ2 . . . . .	74
6.5.2 Function Documentation . . . . .	75
6.5.2.1 main() . . . . .	75
6.5.3 Variable Documentation . . . . .	75
6.5.3.1 NumberOfBoundaries . . . . .	75
6.5.3.2 NumberOfMaterials . . . . .	75
6.5.3.3 NumberOfParticles . . . . .	75
6.6 ex6_validation.cu File Reference . . . . .	75
6.6.1 Function Documentation . . . . .	76
6.6.1.1 main() . . . . .	76
6.6.2 Variable Documentation . . . . .	76
6.6.2.1 NumberOfBoundaries . . . . .	76
6.6.2.2 NumberOfMaterials . . . . .	76
6.6.2.3 NumberOfParticles . . . . .	76
6.7 main.cu File Reference . . . . .	77
6.7.1 Detailed Description . . . . .	77
6.7.2 Function Documentation . . . . .	78
6.7.2.1 main() . . . . .	78
6.8 source/acceleration.cuh File Reference . . . . .	78
6.8.1 Detailed Description . . . . .	79
6.8.2 Macro Definition Documentation . . . . .	79
6.8.2.1 acceleration_H . . . . .	79
6.9 source/contact.cuh File Reference . . . . .	80
6.9.1 Detailed Description . . . . .	81
6.9.2 Macro Definition Documentation . . . . .	82
6.9.2.1 contact_H . . . . .	82
6.10 source/domain.cuh File Reference . . . . .	82
6.10.1 Detailed Description . . . . .	83
6.10.2 Macro Definition Documentation . . . . .	83
6.10.2.1 domain_H . . . . .	83

6.10.3 Enumeration Type Documentation . . . . .	83
6.10.3.1 BoundaryConditionType . . . . .	83
6.11 source/forces.cuh File Reference . . . . .	84
6.11.1 Detailed Description . . . . .	85
6.11.2 Macro Definition Documentation . . . . .	85
6.11.2.1 forces_H . . . . .	85
6.12 source/integrate.cuh File Reference . . . . .	86
6.12.1 Detailed Description . . . . .	87
6.12.2 Macro Definition Documentation . . . . .	87
6.12.2.1 integrate_H . . . . .	87
6.13 source/io.cuh File Reference . . . . .	88
6.13.1 Detailed Description . . . . .	89
6.13.2 Macro Definition Documentation . . . . .	89
6.13.2.1 io_H . . . . .	89
6.14 source/material.cuh File Reference . . . . .	89
6.14.1 Detailed Description . . . . .	90
6.14.2 Macro Definition Documentation . . . . .	91
6.14.2.1 material_H . . . . .	91
6.15 source/math.cuh File Reference . . . . .	91
6.15.1 Detailed Description . . . . .	92
6.15.2 Macro Definition Documentation . . . . .	93
6.15.2.1 CHECK . . . . .	93
6.15.2.2 math_H . . . . .	93
6.15.3 Typedef Documentation . . . . .	93
6.15.3.1 vec3D . . . . .	93
6.15.4 Function Documentation . . . . .	93
6.15.4.1 calculateDistance() . . . . .	93
6.15.4.2 calculateNormal() . . . . .	94
6.16 source/memory.cuh File Reference . . . . .	94
6.16.1 Detailed Description . . . . .	96
6.16.2 Macro Definition Documentation . . . . .	96
6.16.2.1 memory_H . . . . .	96
6.17 source/particle.cuh File Reference . . . . .	97
6.17.1 Detailed Description . . . . .	98
6.17.2 Macro Definition Documentation . . . . .	99
6.17.2.1 particle_H . . . . .	99
6.18 source/randomgen.cuh File Reference . . . . .	99
6.18.1 Detailed Description . . . . .	100
6.18.2 Macro Definition Documentation . . . . .	100
6.18.2.1 random_H . . . . .	101
6.19 source/registers.cuh File Reference . . . . .	101
6.19.1 Detailed Description . . . . .	102

6.19.2 Macro Definition Documentation	102
6.19.2.1 register_H	102
6.20 source/settings.cuh File Reference	103
6.20.1 Detailed Description	104
6.20.2 Macro Definition Documentation	105
6.20.2.1 settings_H	105
6.20.3 Typedef Documentation	105
6.20.3.1 var_type	105
6.20.4 Enumeration Type Documentation	105
6.20.4.1 ContactModel	105
6.20.4.2 ContactSearch	106
6.20.4.3 DomainType	106
6.20.4.4 OutputFormat	106
6.20.4.5 TimeIntegration	106
6.20.5 Variable Documentation	107
6.20.5.1 AccelerationStored	107
6.20.5.2 BlockSize	107
6.20.5.3 BodyForce	107
6.20.5.4 contactModel	107
6.20.5.5 contactSearch	107
6.20.5.6 Debug	108
6.20.5.7 domainType	108
6.20.5.8 MaxContactNumber	108
6.20.5.9 outputFormat	108
6.20.5.10 RollingFriction	108
6.20.5.11 SaveAngularVelocity	108
6.20.5.12 SaveForce	108
6.20.5.13 Saveld	109
6.20.5.14 SaveMaterial	109
6.20.5.15 SaveTorque	109
6.20.5.16 SaveVelocity	109
6.20.5.17 timeIntegration	109
6.20.5.18 UseGPUWideThreadSync	109
6.21 source/solver.cuh File Reference	110
6.21.1 Detailed Description	111
6.21.2 Macro Definition Documentation	111
6.21.2.1 solver_H	111
6.21.3 Function Documentation	111
6.21.3.1 solver()	111
6.22 source/timestep.cuh File Reference	112
6.22.1 Detailed Description	113
6.22.2 Macro Definition Documentation	113

---

6.22.2.1 timestep_H . . . . .	113
6.23 test1.cu File Reference . . . . .	113
6.23.1 Detailed Description . . . . .	114
6.23.2 Function Documentation . . . . .	114
6.23.2.1 main() . . . . .	114
<b>Index</b>	<b>115</b>



# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">accelerationHandling</a>	Acceleration handling of particles . . . . .	7
<a href="#">constant</a>	Contains all the constant . . . . .	8
<a href="#">contactHandling</a>	Contact handling of particles . . . . .	10
<a href="#">DecomposedDomainsConstants</a>	. . . . .	15
<a href="#">domainHandling</a>	Handling of boundary conditions and STL files . . . . .	17
<a href="#">forceHandling</a>	Contains all the functions to calculate the force between particles . . . . .	18
<a href="#">integrators</a>	Contains the numerical methods and timestepping . . . . .	20
<a href="#">ioHandling</a>	Contains all the functions for writing and reading data . . . . .	23
<a href="#">materialHandling</a>	Contains all the function for material handling . . . . .	25
<a href="#">memoryHandling</a>	Memory handling functions which copy between CPU and GPU and allocate memory . . . . .	26
<a href="#">particleHandling</a>	Contains all the functions and structs necessary for handling the particles . . . . .	29
<a href="#">RandomGeneration</a>	Contains everything necessary for random generation . . . . .	32
<a href="#">registerHandling</a>	Register handling functions . . . . .	34
<a href="#">timeHandling</a>	Functions for handling time . . . . .	35





## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">bodyForce</a>	Stores the constant body forces (e.g. gravity) in the different directions . . . . .	37
<a href="#">boundaryCondition</a>	Contains all the data about boundary conditions . . . . .	38
<a href="#">contact</a>	Contact data between particles, stored in the registers (preferably) . . . . .	41
<a href="#">coordinate</a>	Cartesian coordinates . . . . .	44
<a href="#">coordinates</a>	Cartesian coordinate vectors . . . . .	45
<a href="#">materialContact</a>	Struct which contains the reduced quantities for material combinations . . . . .	46
<a href="#">materialParameters</a>	Struct with all the user given material parameters, stored in the shared memory on the device side . . . . .	47
<a href="#">particle</a>	Coordinates, velocity and radius of a particles . . . . .	50
<a href="#">particleDistribution</a>	All information necessary to generate the initial particles . . . . .	54
<a href="#">registerMemory</a>	Register memory, read at the beginning of the kernel and used throughout to store all the particle data locally . . . . .	56
<a href="#">timestepping</a>	Timestep settings . . . . .	59
<a href="#">vector</a>	3D vector . . . . .	61



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">ex1_deposition.cu</a>	
Gravitational deposition example . . . . .	65
<a href="#">ex2_layered_deposition.cu</a>	
Gravitational deposition in layers example . . . . .	68
<a href="#">ex3_deposition2.cu</a>	71
<a href="#">ex4_multi_material.cu</a>	72
<a href="#">ex5_STL_geometry.cu</a>	74
<a href="#">ex6_validation.cu</a>	75
<a href="#">main.cu</a>	
Main part, case definition . . . . .	77
<a href="#">test1.cu</a>	
Main part, case definition . . . . .	113
<a href="#">source/acceleration.cuh</a>	
Calculates the acceleration . . . . .	78
<a href="#">source/contact.cuh</a>	
Contact search algorithms . . . . .	80
<a href="#">source/domain.cuh</a>	
Description of the simulation domain . . . . .	82
<a href="#">source/forces.cuh</a>	
Force calculations . . . . .	84
<a href="#">source/integrate.cuh</a>	
Numerical timestepping schemes . . . . .	86
<a href="#">source/io.cuh</a>	
Input-output handling . . . . .	88
<a href="#">source/material.cuh</a>	
Material description . . . . .	89
<a href="#">source/math.cuh</a>	
Math function . . . . .	91
<a href="#">source/memory.cuh</a>	
Memory allocation and synchronization of host and device side . . . . .	94
<a href="#">source/particle.cuh</a>	
Particle and particle cloud discriptions . . . . .	97
<a href="#">source/randomgen.cuh</a>	
Random number generation . . . . .	99
<a href="#">source/registers.cuh</a>	
Contains the register struct . . . . .	101

source/ <a href="#">settings.cuh</a>	
Simulation settings, user given . . . . .	103
source/ <a href="#">solver.cuh</a>	
Solver algorithm on the device . . . . .	110
source/ <a href="#">timestep.cuh</a>	
Timestepping settings . . . . .	112

## 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**

<i>tid</i>	Thread index
<i>rmem</i>	Register memory containing all the data about particle with index tid
<i>particles</i>	Contains all the particle data
<i>bodyForces</i>	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**

<i>tid</i>	Thread index
<i>rmem</i>	Register memory containing all the data about particle with index tid
<i>particles</i>	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](#) = 3.141592653589793238462643
- 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 = 3.141592653589793238462643 [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 numberOfActiveParticles, 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 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__ 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

<i>cid1</i>	Cell id 1
<i>cid2</i>	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**

<i>tid</i>	Thread index of the particle
<i>rmem</i>	Register memory containing all the data about the particle
<i>numberOfActiveParticles</i>	Number of active parameters
<i>particles</i>	All the particle data
<i>contacts</i>	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**

<i>tid</i>	Thread index of the particle
<i>rmem</i>	Register memory containing all the data about the particle
<i>numberOfActiveParticles</i>	Number of active parameters
<i>particles</i>	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

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

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

<i>tid</i>	Thread index of the particle
<i>contacts</i>	List of particles we are in contact with

Initializes 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

<i>tid</i>	Thread index of the particle
<i>contacts</i>	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

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

## 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

<i>tid</i>	Thread index of the particle
<i>rmem</i>	Register memory containing all the data about the particle
<i>i</i>	Index of the domain boundary, particle tid is in contact with
<i>d</i>	Distance between particle and domain boundary
<i>contacts</i>	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 numberOfActiveParticles)  
*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

<i>tid</i>	Thread index
<i>rmem</i>	Register memory containing all the data about the particle
<i>particles</i>	The particles struct containing all the data about them
<i>contacts</i>	The struct containing all the contacts
<i>pars</i>	The struct containing all the material parameters
<i>timestep</i>	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

<i>particles</i>	A list of particles
<i>numberOfActiveParticles</i>	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

<i>particles</i>	A list of particles
<i>bodyForces</i>	Volumetric forces acting on the particles
<i>numberOfActiveParticles</i>	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

<i>dt</i>	Timestep
<i>x</i>	Current point
<i>f</i>	Current derivative
<i>f_old</i>	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

<i>tid</i>	Thread index of the particle
<i>rmem</i>	Register memory containing all the data about the particle
<i>particles</i>	All the particle data
<i>timestep</i>	Timestep specifications
<i>step</i>	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

<i>dt</i>	Timestep
<i>x</i>	Current point
<i>f</i>	Current derivative
<i>f_old</i>	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

<i>tid</i>	Thread index of the particle
<i>rmem</i>	Register memory containing all the data about the particle
<i>particles</i>	All the particle data
<i>timestep</i>	Timestep specifications

## 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

<i>tid</i>	Thread index of the particle
<i>rmem</i>	Register memory containing all the data about the particle
<i>particles</i>	All the particle data
<i>timestep</i>	Timestep specifications

#### 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

<i>dt</i>	Timestep
<i>x</i>	Current point
<i>f</i>	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**

<i>particles</i>	List of particles
<i>location</i>	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**

<i>particles</i>	List of particles
<i>location</i>	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**

<i>numberOfActiveParticles</i>	Number of active parameters
<i>particles</i>	List of particles
<i>location</i>	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

<i>numberOfActiveParticles</i>	Number of active parameters
<i>particles</i>	List of particles
<i>location</i>	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

<i>pars</i>	<a href="#">materialParameters</a> struct with ALL material parameters
<i>friction</i>	Method to calculate the friction coefficients
<i>elastic</i>	Method to calculate the elastic coefficients
<i>damping</i>	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

<i>pars</i>	<a href="#">materialParameters</a> struct with ALL material parameters
<i>printPairings</i>	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

<i>particlesD</i>	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

<i>particlesH</i>	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

<i>particlesD</i>	The particle struct which needs memory allocation
-------------------	---

#### 4.10.3.4 freeHostParticles()

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

Free the hist side.

#### Parameters

<i>particlesH</i>	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

<i>particlesH</i>	The particle struct which needs to be initialized
-------------------	---

#### 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

<i>dest</i>	Destination of the data
<i>source</i>	Source of the data
<i>vars</i>	Variables to copy according to the enum
<i>kind</i>	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

<i>particles</i>	Pre-allocated memory where the particle data is saved
<i>coords</i>	Coordinates of the particles
<i>radii</i>	Radii of the particles
<i>pars</i>	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

<i>particles</i>	Pre-allocated memory where the particle data is saved
<i>pdist</i>	Particle distribution information based on the <a href="#">particleDistribution</a> struct
<i>psize_dist</i>	Particle size distribuation chosen from the ParticleSizeDistribution enum
<i>pvel_dist</i>	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

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

##### Returns

ensemble 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

<i>particles</i>	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

<i>seed</i>	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

<i>min</i>	lower end of the range
<i>max</i>	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

<i>tid</i>	Thread index
<i>rmem</i>	Register memory
<i>particles</i>	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

<i>tid</i>	Thread index
<i>rmem</i>	Register memory
<i>particles</i>	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

<i>tid</i>	Thread index
<i>rmem</i>	Register memory
<i>particles</i>	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

<i>timestep</i>	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

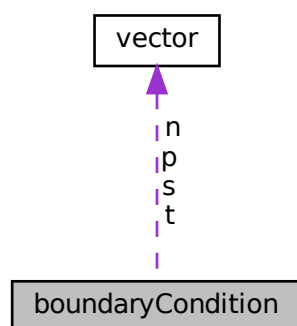
The documentation for this struct was generated from the following file:

- [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.

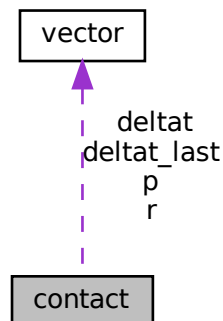
The documentation for this struct was generated from the following file:

- [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`]  
*equivalent 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]`

equivalent 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 coordinate*
- [var\\_type y](#)  
*y coordinate*
- [var\\_type z](#)  
*z coordinate*

### 5.4.1 Detailed Description

Cartesian coordinates.

### 5.4.2 Member Data Documentation

#### 5.4.2.1 x

```
var\_type coordinate::x
```

x coordinate

#### 5.4.2.2 y

`var_type` coordinate::y

y coordinate

#### 5.4.2.3 z

`var_type` coordinate::z

z coordinate

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 coordinate*
- `var_type` \* y  
*y coordinate*
- `var_type` \* z  
*z coordinate*

### 5.5.1 Detailed Description

Cartesian coordinate vectors.

### 5.5.2 Member Data Documentation

#### 5.5.2.1 x

`var_type*` coordinates::x

x coordinate

### 5.5.2.2 y

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

y coordinate

### 5.5.2.3 z

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

z coordinate

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]
```

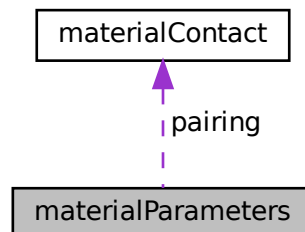
The documentation for this struct was generated from the following file:

- [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 pairing.*

### 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.

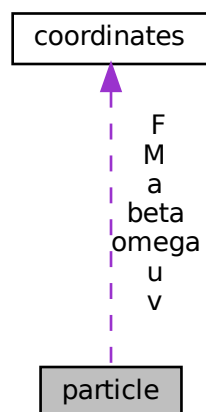
The documentation for this struct was generated from the following file:

- [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

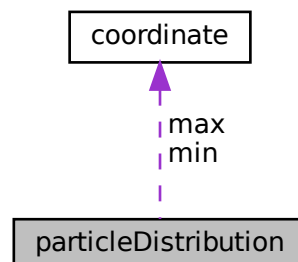
The documentation for this struct was generated from the following file:

- [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.

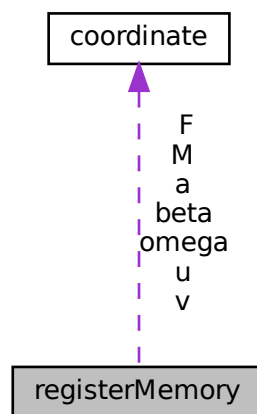
The documentation for this struct was generated from the following file:

- [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](#)



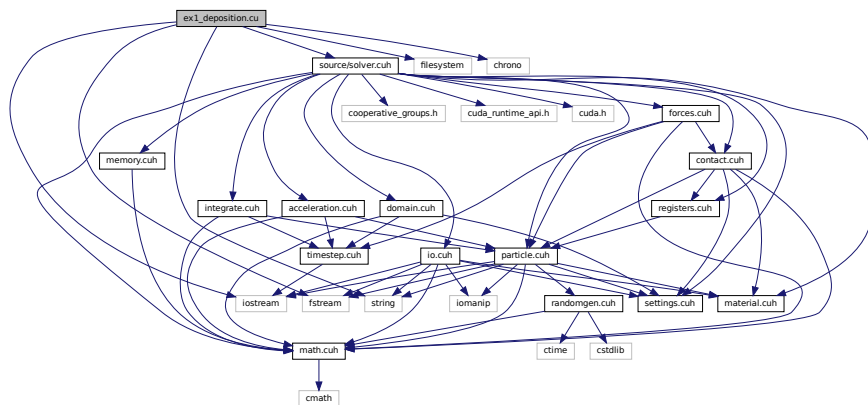
## Chapter 6

# 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

- $R = 60\text{mm} \pm 2\text{mm}$
- $E = 2G = 200\text{GPa}$
- $\rho = 2000 \text{ kg/m}^3$
- $\mu = 0.5, \mu_0 = 0.7, \mu_r = 0.02$
- $e = 0.1$  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.

- $E = G = 20\text{MPa}$
- $\rho = 1000 \text{ kg/m}^3$
- $\mu = 0.5, \mu_0 = 0.7, \mu_r = 0.02$
- $\beta = 1.5$  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  $\rho=1000\text{kg/m}^3$  and light material has  $\rho=200\text{kg/m}^3$

**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

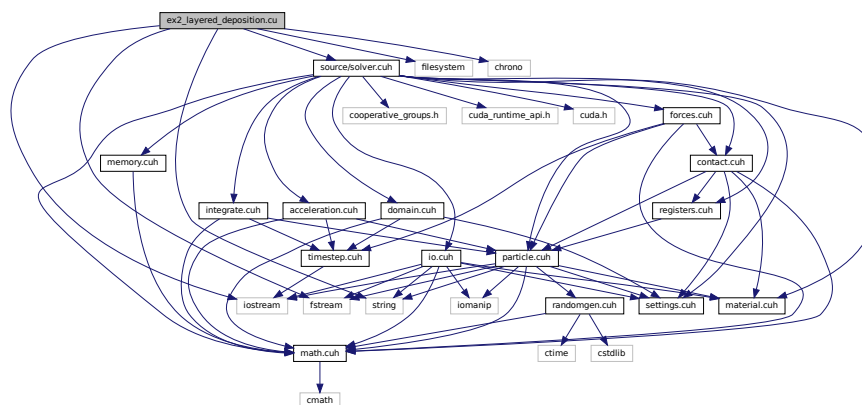
```
constexpr int NumberOfParticles = 2048 [constexpr]
```

## 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

- $R = 40\text{mm} \pm 10\text{mm}$
- $E = 2G = 200\text{GPa}$
- $\rho = 2000 \text{ kg/m}^3$
- $\mu = 0.5$ ,  $\mu_0 = 0.7$ ,  $\mu_r = 0.02$
- $e = 0.1$  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

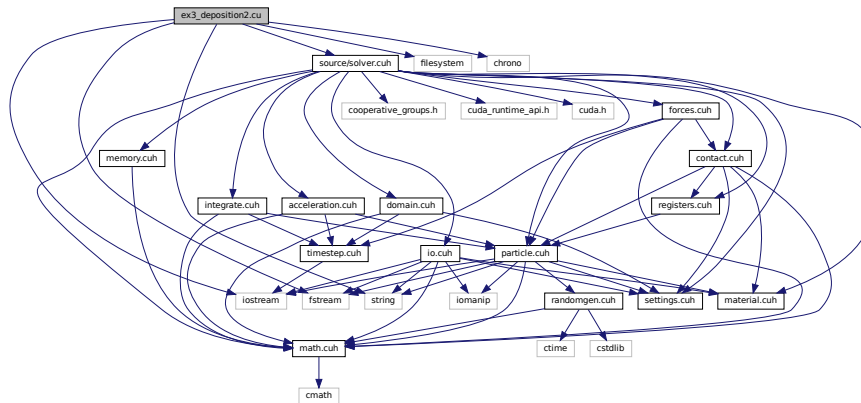
```
constexpr int NumberOfParticles = 8192 [constexpr]
```

### 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

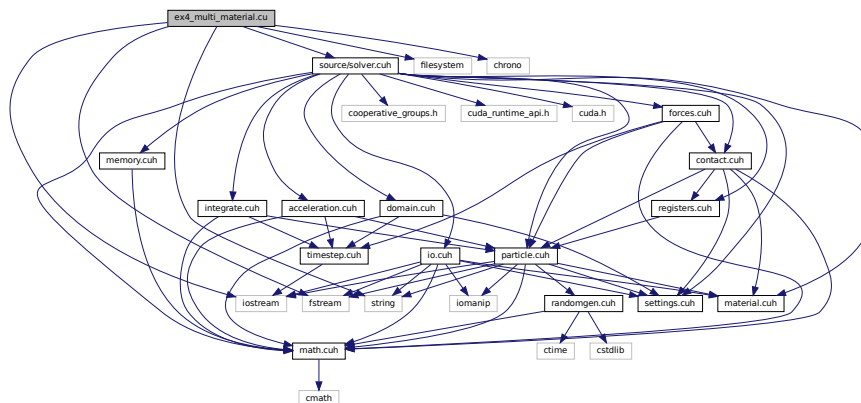
```
constexpr int NumberOfMaterials = 2 [constexpr]
```

### 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

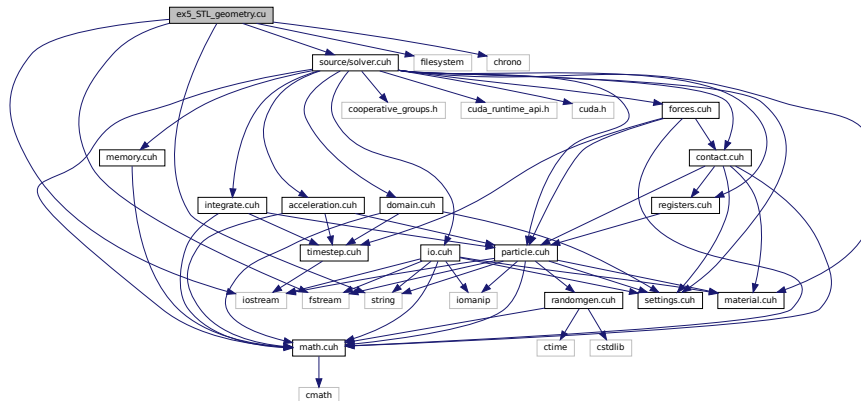
```
constexpr int NumberOfMaterials = 3 [constexpr]
```

#### 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]
```

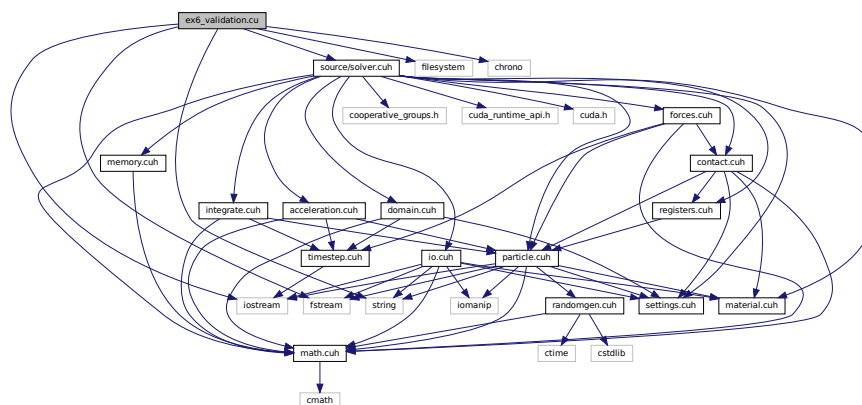
### 6.5.3.3 NumberOfParticles

```
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`

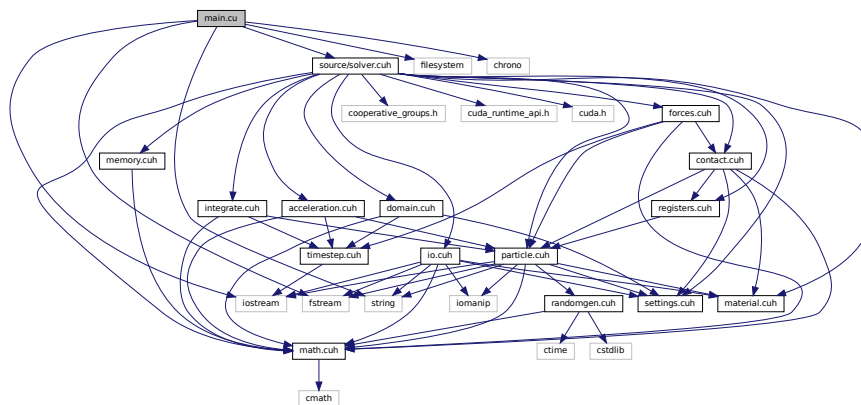
```
constexpr int NumberOfParticles = 8192 [constexpr]
```

## 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

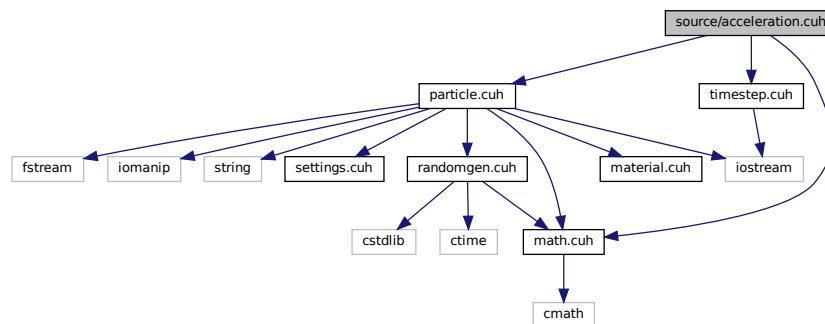
### 6.7.2.1 main()

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

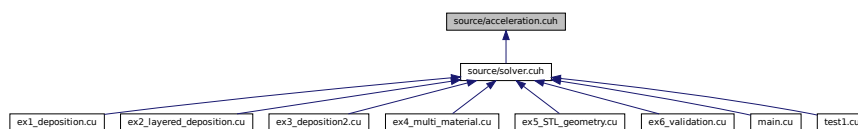
## 6.8 source/acceleration.cuh File Reference

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

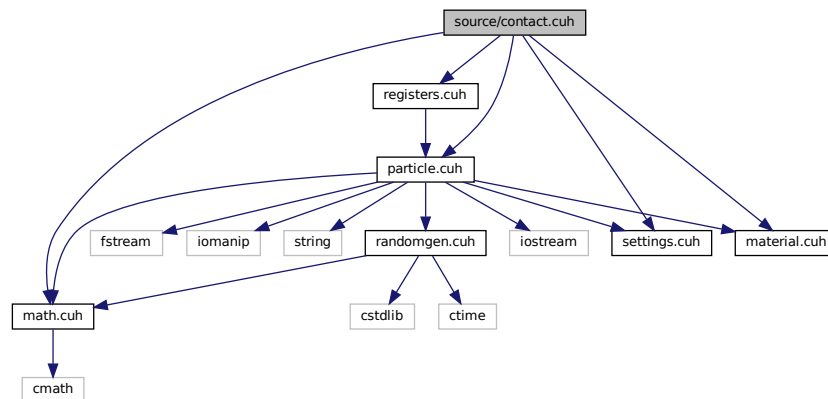
#### 6.8.2.1 acceleration\_H

```
#define acceleration_H
```

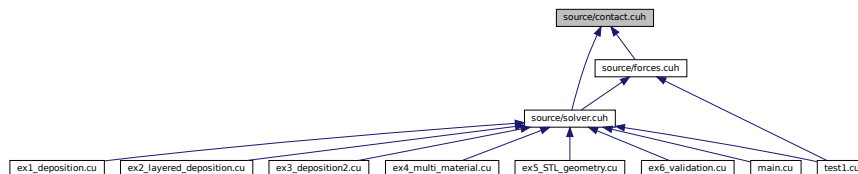
## 6.9 source/contact.cuh File Reference

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 numberOfActiveParticles, 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 neighbouring cell
- DecomposedDomainsFast: EXPERIMENTAL

## 6.9.2 Macro Definition Documentation

### 6.9.2.1 contact\_H

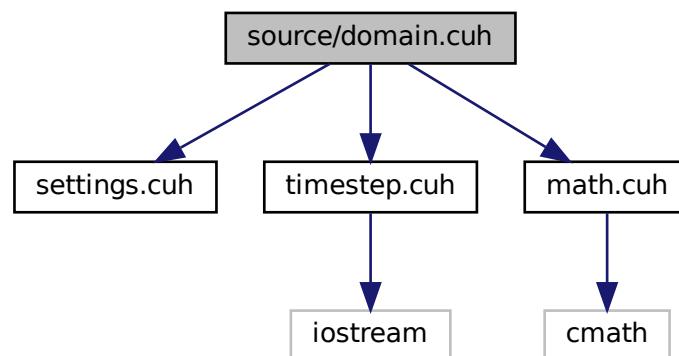
```
#define contact_H
```

## 6.10 source/domain.cuh File Reference

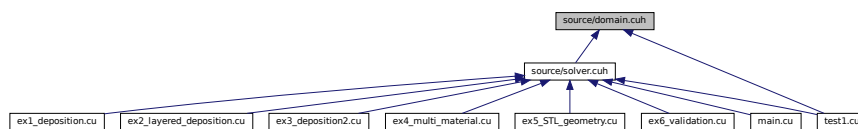
Description of the simulation domain.

```
#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
```

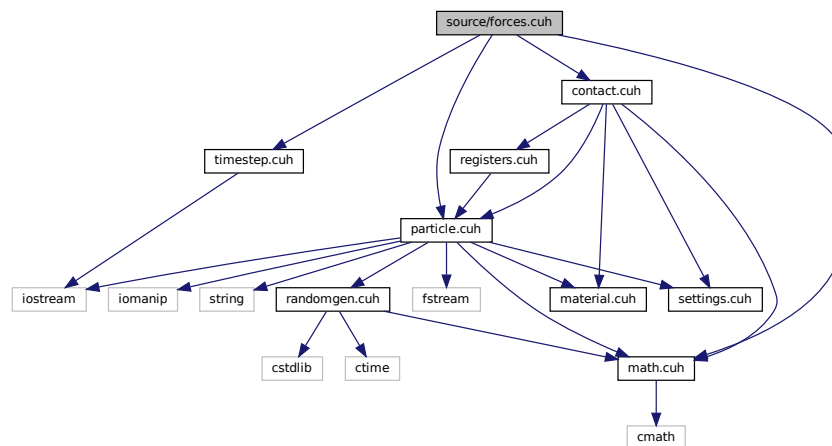
## Enumerator

None	
ReflectiveWall	
HertzWall	

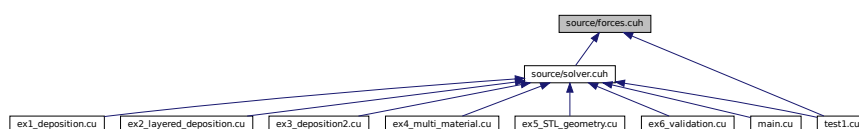
## 6.11 source/forces.cuh File Reference

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`

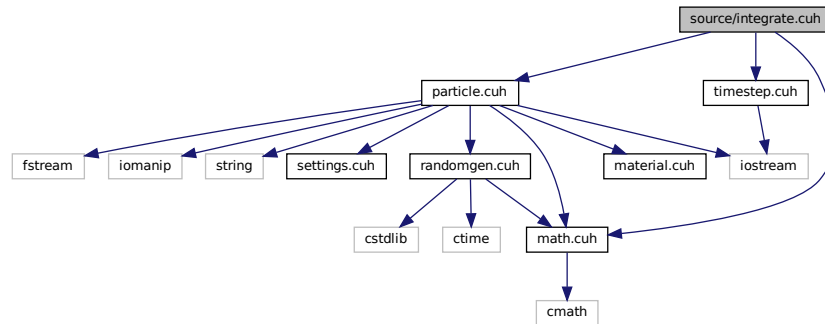
```
#define forces_H
```

## 6.12 source/integrate.cuh File Reference

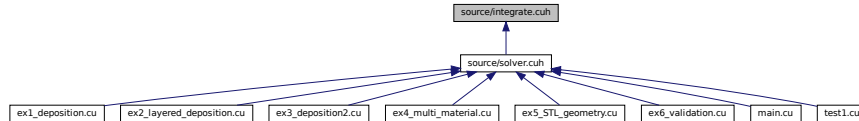
Numerical timestepping schemes.

```
#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`

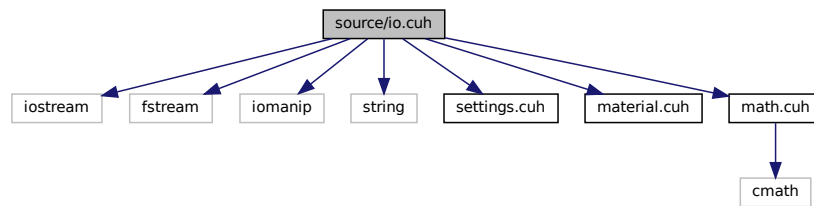
```
#define integrate_H
```

## 6.13 source/io.cuh File Reference

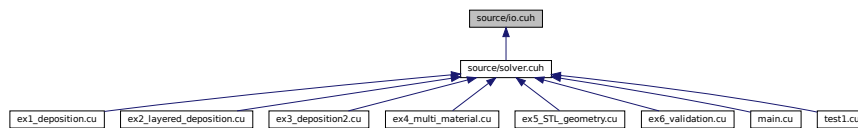
Input-output handling.

```
#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

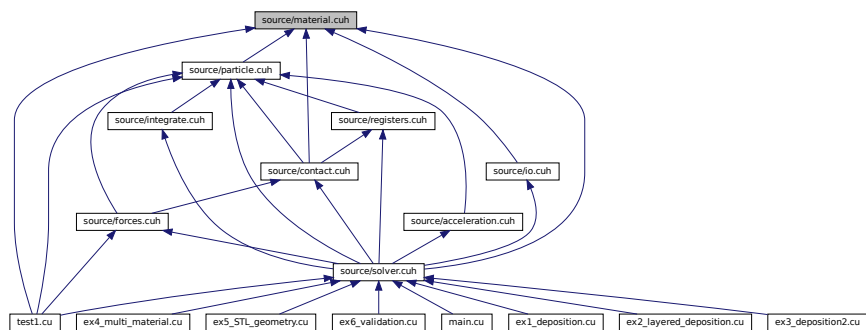
#### 6.13.2.1 io\_H

```
#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

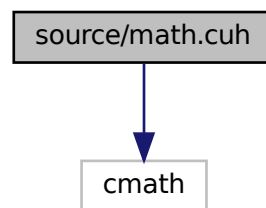
```
#define material_H
```

## 6.15 source/math.cuh File Reference

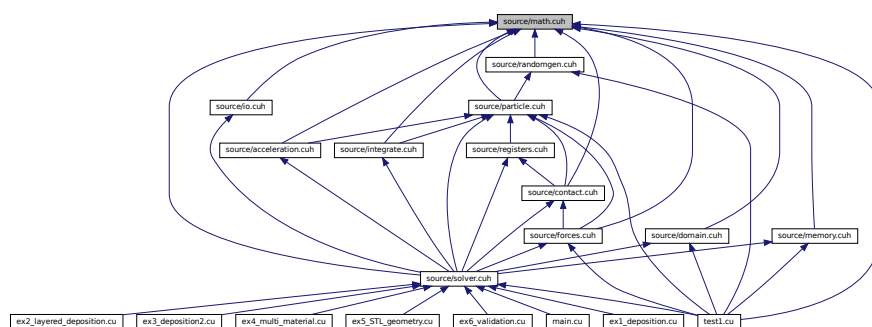
Math function.

```
#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 = 3.141592653589793238462643`
- `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:**

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

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**

<i>x1</i>	x coord. of point 1
<i>y1</i>	y coord. of point 1
<i>z1</i>	z coord. of point 1
<i>x2</i>	x coord. of point 2
<i>y2</i>	y coord. of point 2
<i>z2</i>	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**

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

**Returns**

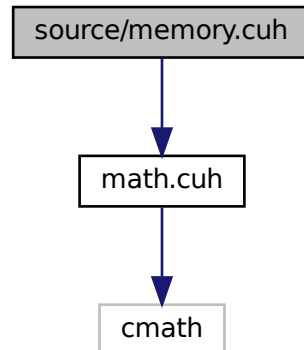
Returns the relative normal velocity

**6.16 source/memory.cuh File Reference**

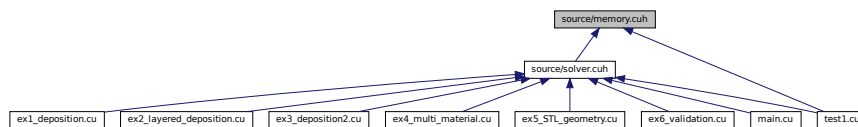
Memory allocation and synchronization of host and device side.

```
#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`

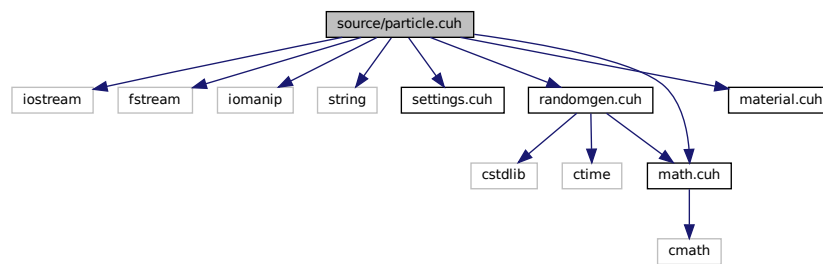
```
#define memory_H
```

## 6.17 source/particle.cuh File Reference

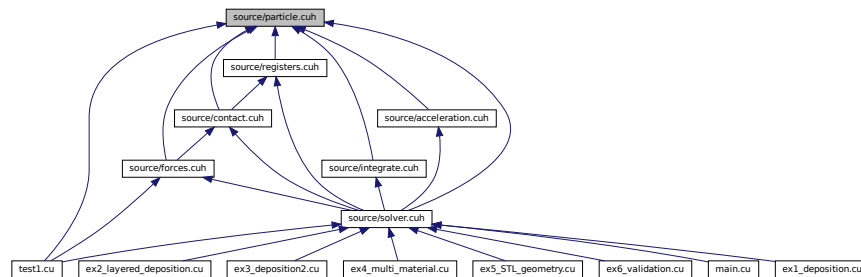
Particle and particle cloud discriptions.

```
#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

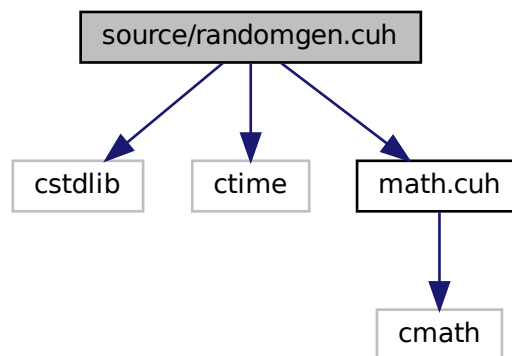
### 6.17.2.1 particle\_H

```
#define particle_H
```

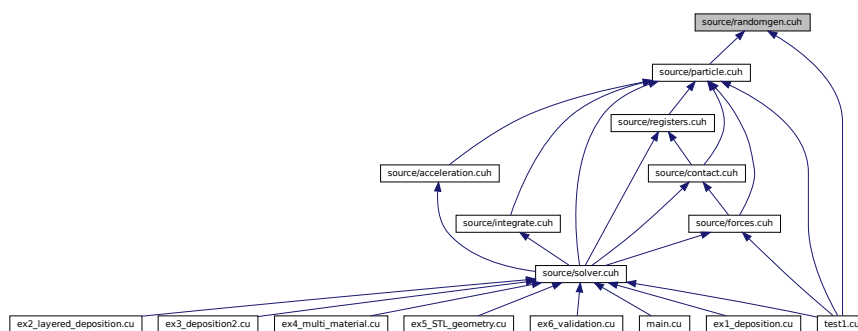
## 6.18 source/randomgen.cuh File Reference

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

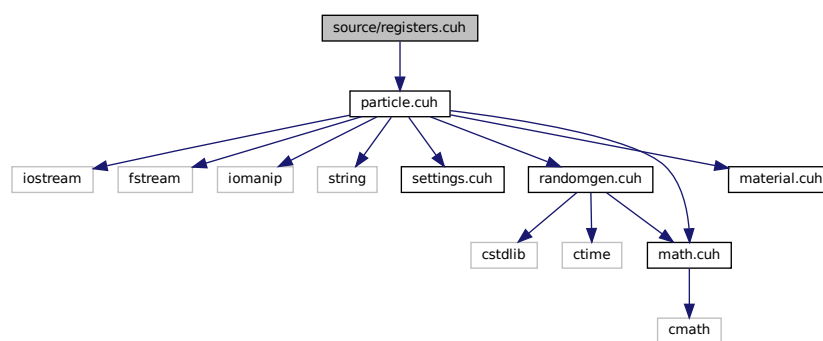
```
#define random_H
```

## 6.19 source/registers.cuh File Reference

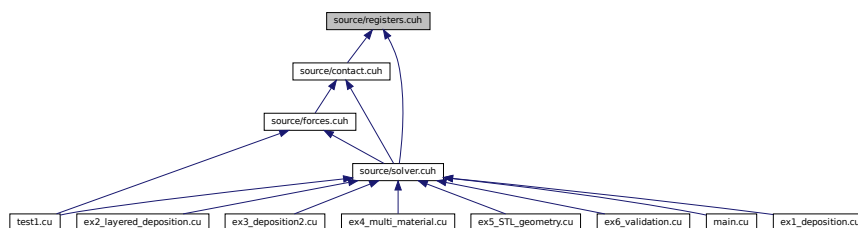
Contains the register struct.

```
#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

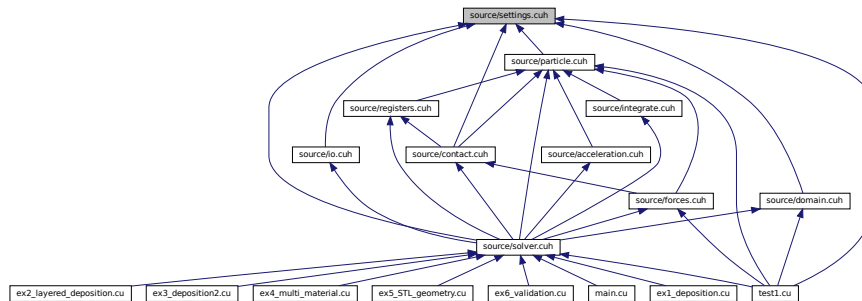
#### 6.19.2.1 register\_H

```
#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)



### 6.21.1 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

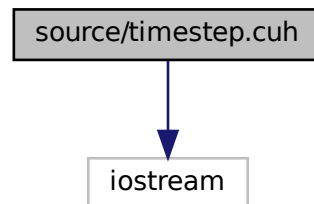
<i>particles</i>	All the particle data
<i>numberOfActiveParticles</i>	Number of currently active particles
<i>pars</i>	Material parameters
<i>timestep</i>	Timestep settings
<i>bodyForces</i>	Body forces (e.g. gravity)
<i>boundaryConditions</i>	Boundary conditions and types
<i>launch</i>	Number of the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

## 6.22 source/timestep.cuh File Reference

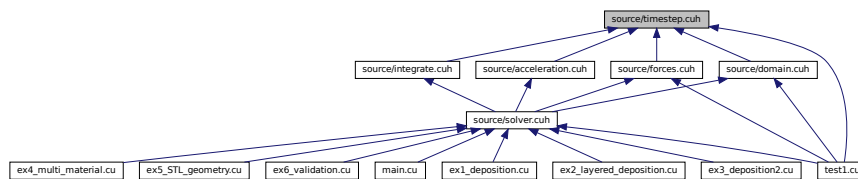
Timestepping settings.

```
#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

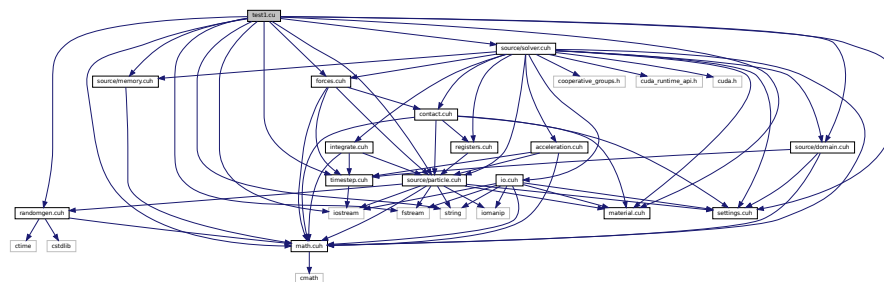
```
#define timestep_H
```

## 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

a

- particle, [51](#)
- registerMemory, [57](#)

AB2

- integrators, [20](#)

AB2C1

- constant, [9](#)

AB2C2

- constant, [9](#)

Acceleration

- memoryHandling, [27](#)

acceleration.cuh

- acceleration\_H, [79](#)

acceleration\_H

- acceleration.cuh, [79](#)

accelerationHandling, [7](#)

- addBodyForces, [7](#)

- calculateDefault, [8](#)

AccelerationStored

- settings.cuh, [107](#)

Adams2

- settings.cuh, [107](#)

adams2

- integrators, [21](#)

addBodyForces

- accelerationHandling, [7](#)

All

- memoryHandling, [27](#)

allocateDeviceParticles

- memoryHandling, [28](#)

allocateHostParticles

- memoryHandling, [28](#)

alpha

- boundaryCondition, [39](#)

AM2

- integrators, [21](#)

AM2C1

- constant, [9](#)

AngularAcceleration

- memoryHandling, [27](#)

AngularVelocity

- memoryHandling, [27](#)

applyBoundaryConditions

- domainHandling, [17](#)

areNeighbours

- contactHandling, [11](#)

ASCII

- settings.cuh, [106](#)

Balanced

- settings.cuh, [106](#)

beta

- boundaryCondition, [39](#)

- materialParameters, [48](#)

- particle, [51](#)

- registerMemory, [57](#)

beta\_star

- materialContact, [46](#)

Binary

- settings.cuh, [106](#)

BlockSize

- settings.cuh, [107](#)

BodyForce

- settings.cuh, [107](#)

bodyForce, [37](#)

- x, [37](#)

- y, [37](#)

- z, [38](#)

boundaryCondition, [38](#)

- alpha, [39](#)

- beta, [39](#)

- gamma, [39](#)

- material, [40](#)

- n, [40](#)

- p, [40](#)

- s, [40](#)

- s\_scale, [40](#)

- t, [40](#)

- t\_scale, [40](#)

- type, [41](#)

BoundaryConditionType

- domain.cuh, [83](#)

BruteForce

- settings.cuh, [106](#)

BruteForceContactSearch

- contactHandling, [12](#)

CalculateCellId

- contactHandling, [12](#)

CalculateContact

- contactHandling, [12](#)

calculateDefault

- accelerationHandling, [8](#)

calculateDistance

- math.cuh, [93](#)

calculateForceMindlin

- forceHandling, [19](#)

calculateMaterialContact

- materialHandling, [26](#)

calculateNormal

- math.cuh, 94
- CalculateOverlap
  - domainHandling, 18
- calculateTotalKineticEnergy
  - forceHandling, 19
- calculateTotalPotentialEnergy
  - forceHandling, 20
- CellID
  - memoryHandling, 27
- CHECK
  - math.cuh, 93
- cid
  - particle, 52
  - registerMemory, 57
- constant, 8
  - AB2C1, 9
  - AB2C2, 9
  - AM2C1, 9
  - DAMPING, 9
  - NUMBER\_04, 9
  - NUMBER\_05, 9
  - NUMBER\_1, 9
  - NUMBER\_2, 10
  - NUMBER\_4o3, 10
  - NUMBER\_8, 10
  - PI, 10
  - VOLUME\_FACTOR, 10
  - ZERO, 10
- contact, 41
  - count, 42
  - deltan, 42
  - deltat, 42
  - deltat\_last, 42
  - material, 43
  - mstar, 43
  - p, 43
  - r, 43
  - Rstar, 43
  - tid, 43
  - tid\_last, 44
- contact.cuh
  - contact\_H, 82
- contact\_H
  - contact.cuh, 82
- contactHandling, 10
  - areNeighbours, 11
  - BruteForceContactSearch, 12
  - CalculateCellId, 12
  - CalculateContact, 12
  - DecomposedDomainsContactSearch, 13
  - initializeContacts, 13
  - Neighbours, 14
  - ResetContacts, 14
- ContactModel
  - settings.cuh, 105
- contactModel
  - settings.cuh, 107
- ContactSearch
  - settings.cuh, 105
- contactSearch
  - settings.cuh, 107
- coordinate, 44
  - x, 44
  - y, 44
  - z, 45
- coordinates, 45
  - x, 45
  - y, 45
  - z, 46
- count
  - contact, 42
- DAMPING
  - constant, 9
- Debug
  - settings.cuh, 107
- DecomposedDomains
  - settings.cuh, 106
- DecomposedDomainsConstants, 15
  - maxx, 15
  - maxy, 15
  - maxz, 15
  - minx, 15
  - miny, 16
  - minz, 16
  - NoverDx, 16
  - NoverDy, 16
  - NoverDz, 16
  - Nx, 16
  - Ny, 16
  - Nz, 17
- DecomposedDomainsContactSearch
  - contactHandling, 13
- DecomposedDomainsFast
  - settings.cuh, 106
- deltan
  - contact, 42
- deltat
  - contact, 42
- deltat\_last
  - contact, 42
- domain.cuh
  - BoundaryConditionType, 83
  - domain\_H, 83
  - HertzWall, 84
  - None, 84
  - ReflectiveWall, 84
- domain\_H
  - domain.cuh, 83
- domainHandling, 17
  - applyBoundaryConditions, 17
  - CalculateOverlap, 18
- DomainType
  - settings.cuh, 106
- domainType
  - settings.cuh, 108
- dt

- timestepping, 60
- E
  - materialParameters, 48
- e
  - materialParameters, 48
- E\_star
  - materialContact, 46
- endOfKernelSync
  - registerHandling, 34
- endOfStepSync
  - registerHandling, 35
- endtime
  - timestepping, 60
- Euler
  - settings.cuh, 107
- euler
  - integrators, 22
- ex1\_deposition.cu, 65
  - main, 67
  - NumberOfBoundaries, 68
  - NumberOfMaterials, 68
  - NumberOfParticles, 68
- ex2\_layered\_deposition.cu, 68
  - main, 69
  - NumberOfBoundaries, 70
  - numberOfLayers, 70
  - NumberOfMaterials, 70
  - NumberOfParticles, 70
  - particlesPerLayer, 70
- ex3\_deposition2.cu, 71
  - main, 71
  - NumberOfBoundaries, 71
  - NumberOfMaterials, 72
  - NumberOfParticles, 72
- ex4\_multi\_material.cu, 72
  - main, 73
  - NumberOfBoundaries, 73
  - NumberOfMaterials, 73
  - NumberOfParticles, 73
- ex5\_STL\_geometry.cu, 74
  - main, 75
  - NumberOfBoundaries, 75
  - NumberOfMaterials, 75
  - NumberOfParticles, 75
  - SQ2, 74
- ex6\_validation.cu, 75
  - main, 76
  - NumberOfBoundaries, 76
  - NumberOfMaterials, 76
  - NumberOfParticles, 76
- Exact
  - settings.cuh, 107
- exact
  - integrators, 22
- F
  - particle, 52
  - registerMemory, 57
- fillRegisterMemory
  - registerHandling, 35
- Force
  - memoryHandling, 27
- forceHandling, 18
  - calculateForceMindlin, 19
  - calculateTotalKineticEnergy, 19
  - calculateTotalPotentialEnergy, 20
- forces.cuh
  - forces\_H, 85
- forces\_H
  - forces.cuh, 85
- freeDeviceParticles
  - memoryHandling, 28
- freeHostParticles
  - memoryHandling, 28
- G
  - materialParameters, 49
- G\_star
  - materialContact, 47
- gamma
  - boundaryCondition, 39
- Gauss
  - particleHandling, 30
- generateParticleLocation
  - particleHandling, 31
- generateParticleParameters
  - particleHandling, 31
- HarmonicMean
  - materialHandling, 26
- HertzWall
  - domain.cuh, 84
- initializeContacts
  - contactHandling, 13
- initializeHostParticles
  - memoryHandling, 29
- initializeRandomSeed
  - RandomGeneration, 33
- integrate.cuh
  - integrate\_H, 87
- integrate\_H
  - integrate.cuh, 87
- integrators, 20
  - AB2, 20
  - adams2, 21
  - AM2, 21
  - euler, 22
  - exact, 22
  - RK1, 22
- io.cuh
  - io\_H, 89
- io\_H
  - io.cuh, 89
- ioHandling, 23
  - readParticlesCSV, 23
  - readParticlesVTK, 24

- saveParticles, 24
  - saveParticlesVTK, 24
- length
  - vector, 62
- listOfVariables
  - memoryHandling, 27
- M
  - particle, 52
  - registerMemory, 57
- m
  - particle, 52
  - registerMemory, 57
- m\_rec
  - particle, 52
  - registerMemory, 57
- main
  - ex1\_deposition.cu, 67
  - ex2\_layered\_deposition.cu, 69
  - ex3\_deposition2.cu, 71
  - ex4\_multi\_material.cu, 73
  - ex5\_STL\_geometry.cu, 75
  - ex6\_validation.cu, 76
  - main.cu, 78
  - test1.cu, 114
- main.cu, 77
  - main, 78
- Material
  - memoryHandling, 27
- material
  - boundaryCondition, 40
  - contact, 43
  - particle, 52
  - registerMemory, 58
- material.cuh
  - material\_H, 91
- material\_H
  - material.cuh, 91
- materialContact, 46
  - beta\_star, 46
  - E\_star, 46
  - G\_star, 47
  - mu0\_star, 47
  - mu\_star, 47
  - mur\_star, 47
- materialHandling, 25
  - calculateMaterialContact, 26
  - HarmonicMean, 26
  - Max, 26
  - Mean, 26
  - methods, 25
  - Min, 26
  - printMaterialInfo, 26
- materialParameters, 47
  - beta, 48
  - E, 48
  - e, 48
  - G, 49
  - mu, 49
  - mu0, 49
  - mur, 49
  - nu, 49
  - pairing, 49
  - rho, 50
- math.cuh
  - calculateDistance, 93
  - calculateNormal, 94
  - CHECK, 93
  - math\_H, 93
  - vec3D, 93
- math\_H
  - math.cuh, 93
- Max
  - materialHandling, 26
- max
  - particleDistribution, 55
- MaxContactNumber
  - settings.cuh, 108
- maxx
  - DecomposedDomainsConstants, 15
- maxy
  - DecomposedDomainsConstants, 15
- maxz
  - DecomposedDomainsConstants, 15
- Mean
  - materialHandling, 26
- memory.cuh
  - memory\_H, 96
- memory\_H
  - memory.cuh, 96
- memoryHandling, 26
  - Acceleration, 27
  - All, 27
  - allocateDeviceParticles, 28
  - allocateHostParticles, 28
  - AngularAcceleration, 27
  - AngularVelocity, 27
  - CellID, 27
  - Force, 27
  - freeDeviceParticles, 28
  - freeHostParticles, 28
  - initializeHostParticles, 29
  - listOfVariables, 27
  - Material, 27
  - Position, 27
  - Radius, 27
  - synchronizeParticles, 29
  - Torque, 27
  - Velocity, 27
- methods
  - materialHandling, 25
- Min
  - materialHandling, 26
- min
  - particleDistribution, 55
- Mindlin

- settings.cuh, 105
- minx
  - DecomposedDomainsConstants, 15
- miny
  - DecomposedDomainsConstants, 16
- minz
  - DecomposedDomainsConstants, 16
- mstar
  - contact, 43
- mu
  - materialParameters, 49
- mu0
  - materialParameters, 49
- mu0\_star
  - materialContact, 47
- mu\_star
  - materialContact, 47
- mur
  - materialParameters, 49
- mur\_star
  - materialContact, 47
- n
  - boundaryCondition, 40
- Neighbours
  - contactHandling, 14
- None
  - domain.cuh, 84
  - particleHandling, 30
- NoverDx
  - DecomposedDomainsConstants, 16
- NoverDy
  - DecomposedDomainsConstants, 16
- NoverDz
  - DecomposedDomainsConstants, 16
- nu
  - materialParameters, 49
- NUMBER\_04
  - constant, 9
- NUMBER\_05
  - constant, 9
- NUMBER\_1
  - constant, 9
- NUMBER\_2
  - constant, 10
- NUMBER\_4o3
  - constant, 10
- NUMBER\_8
  - constant, 10
- NumberOfBoundaries
  - ex1\_deposition.cu, 68
  - ex2\_layered\_deposition.cu, 70
  - ex3\_deposition2.cu, 71
  - ex4\_multi\_material.cu, 73
  - ex5\_STL\_geometry.cu, 75
  - ex6\_validation.cu, 76
- numberOfLayers
  - ex2\_layered\_deposition.cu, 70
- NumberOfMaterials
  - ex1\_deposition.cu, 68
  - ex2\_layered\_deposition.cu, 70
  - ex3\_deposition2.cu, 72
  - ex4\_multi\_material.cu, 73
  - ex5\_STL\_geometry.cu, 75
  - ex6\_validation.cu, 76
- NumberOfParticles
  - ex1\_deposition.cu, 68
  - ex2\_layered\_deposition.cu, 70
  - ex3\_deposition2.cu, 72
  - ex4\_multi\_material.cu, 73
  - ex5\_STL\_geometry.cu, 75
  - ex6\_validation.cu, 76
- numberOfSteps
  - timestepping, 60
- Nx
  - DecomposedDomainsConstants, 16
- Ny
  - DecomposedDomainsConstants, 16
- Nz
  - DecomposedDomainsConstants, 17
- omega
  - particle, 53
  - registerMemory, 58
- operator\*
  - vector, 62
- operator^
  - vector, 63
- operator+
  - vector, 63
- operator-
  - vector, 63
- OutputFormat
  - settings.cuh, 106
- outputFormat
  - settings.cuh, 108
- overRandMax
  - RandomGeneration, 34
- p
  - boundaryCondition, 40
  - contact, 43
- pairing
  - materialParameters, 49
- particle, 50
  - a, 51
  - beta, 51
  - cid, 52
  - F, 52
  - M, 52
  - m, 52
  - m\_rec, 52
  - material, 52
  - omega, 53
  - R, 53
  - R\_rec, 53
  - theta, 53
  - theta\_rec, 53

- u, [53](#)
  - v, [54](#)
- particle.cuh
  - particle\_H, [99](#)
- particle\_H
  - particle.cuh, [99](#)
- particleDistribution, [54](#)
  - max, [55](#)
  - min, [55](#)
  - Rmean, [55](#)
  - Rsigma, [55](#)
  - vmean, [55](#)
  - vsigma, [55](#)
- particleHandling, [29](#)
  - Gauss, [30](#)
  - generateParticleLocation, [31](#)
  - generateParticleParameters, [31](#)
  - None, [30](#)
  - ParticleSizeDistribution, [30](#)
  - ParticleVelocityDistribution, [30](#)
  - printParticles, [32](#)
  - Uniform, [30](#)
- ParticleSizeDistribution
  - particleHandling, [30](#)
- particlesPerLayer
  - ex2\_layered\_deposition.cu, [70](#)
- ParticleVelocityDistribution
  - particleHandling, [30](#)
- PI
  - constant, [10](#)
- Position
  - memoryHandling, [27](#)
- printMaterialInfo
  - materialHandling, [26](#)
- printParticles
  - particleHandling, [32](#)
- printTimestepSettings
  - timeHandling, [36](#)
- R
  - particle, [53](#)
  - registerMemory, [58](#)
- r
  - contact, [43](#)
- R\_rec
  - particle, [53](#)
  - registerMemory, [58](#)
- Radius
  - memoryHandling, [27](#)
- random\_H
  - randomgen.cuh, [100](#)
- randomgen.cuh
  - random\_H, [100](#)
- RandomGeneration, [32](#)
  - initializeRandomSeed, [33](#)
  - overRandMax, [34](#)
  - randomInRange, [33](#)
- randomInRange
  - RandomGeneration, [33](#)
- readParticlesCSV
  - ioHandling, [23](#)
- readParticlesVTK
  - ioHandling, [24](#)
- Rectangular
  - settings.cuh, [106](#)
- ReflectiveWall
  - domain.cuh, [84](#)
- register\_H
  - registers.cuh, [102](#)
- registerHandling, [34](#)
  - endOfKernelSync, [34](#)
  - endOfStepSync, [35](#)
  - fillRegisterMemory, [35](#)
- registerMemory, [56](#)
  - a, [57](#)
  - beta, [57](#)
  - cid, [57](#)
  - F, [57](#)
  - M, [57](#)
  - m, [57](#)
  - m\_rec, [57](#)
  - material, [58](#)
  - omega, [58](#)
  - R, [58](#)
  - R\_rec, [58](#)
  - theta\_rec, [58](#)
  - u, [58](#)
  - v, [58](#)
- registers.cuh
  - register\_H, [102](#)
- ResetContacts
  - contactHandling, [14](#)
- rho
  - materialParameters, [50](#)
- RK1
  - integrators, [22](#)
- Rmean
  - particleDistribution, [55](#)
- RollingFriction
  - settings.cuh, [108](#)
- Rsigma
  - particleDistribution, [55](#)
- Rstar
  - contact, [43](#)
- s
  - boundaryCondition, [40](#)
- s\_scale
  - boundaryCondition, [40](#)
- SaveAngularVelocity
  - settings.cuh, [108](#)
- SaveForce
  - settings.cuh, [108](#)
- Saveld
  - settings.cuh, [108](#)
- SaveMaterial
  - settings.cuh, [109](#)
- saveParticles

- ioHandling, 24
- saveParticlesVTK
  - ioHandling, 24
- saveSteps
  - timestepping, 60
- savetime
  - timestepping, 61
- SaveTorque
  - settings.cuh, 109
- SaveVelocity
  - settings.cuh, 109
- settings.cuh
  - AccelerationStored, 107
  - Adams2, 107
  - ASCII, 106
  - Balanced, 106
  - Binary, 106
  - BlockSize, 107
  - BodyForce, 107
  - BruteForce, 106
  - ContactModel, 105
  - contactModel, 107
  - ContactSearch, 105
  - contactSearch, 107
  - Debug, 107
  - DecomposedDomains, 106
  - DecomposedDomainsFast, 106
  - DomainType, 106
  - domainType, 108
  - Euler, 107
  - Exact, 107
  - MaxContactNumber, 108
  - Mindlin, 105
  - OutputFormat, 106
  - outputFormat, 108
  - Rectangular, 106
  - RollingFriction, 108
  - SaveAngularVelocity, 108
  - SaveForce, 108
  - Saveld, 108
  - SaveMaterial, 109
  - SaveTorque, 109
  - SaveVelocity, 109
  - settings\_H, 105
  - STL, 106
  - TimeIntegration, 106
  - timeIntegration, 109
  - UseGPUWideThreadSync, 109
  - var\_type, 105
- settings\_H
  - settings.cuh, 105
- solver
  - solver.cuh, 111
- solver.cuh
  - solver, 111
  - solver\_H, 111
- solver\_H
  - solver.cuh, 111
- source/acceleration.cuh, 78
- source/contact.cuh, 80
- source/domain.cuh, 82
- source/forces.cuh, 84
- source/integrate.cuh, 86
- source/io.cuh, 88
- source/material.cuh, 89
- source/math.cuh, 91
- source/memory.cuh, 94
- source/particle.cuh, 97
- source/randomgen.cuh, 99
- source/registers.cuh, 101
- source/settings.cuh, 103
- source/solver.cuh, 110
- source/timestep.cuh, 112
- SQ2
  - ex5\_STL\_geometry.cu, 74
- starttime
  - timestepping, 61
- STL
  - settings.cuh, 106
- synchronizeParticles
  - memoryHandling, 29
- t
  - boundaryCondition, 40
- t\_scale
  - boundaryCondition, 40
- test1.cu, 113
  - main, 114
- theta
  - particle, 53
- theta\_rec
  - particle, 53
  - registerMemory, 58
- tid
  - contact, 43
- tid\_last
  - contact, 44
- timeHandling, 35
  - printTimestepSettings, 36
- TimeIntegration
  - settings.cuh, 106
- timeIntegration
  - settings.cuh, 109
- timestep.cuh
  - timestep\_H, 113
- timestep\_H
  - timestep.cuh, 113
- timestepping, 59
  - dt, 60
  - endtime, 60
  - numberOfSteps, 60
  - saveSteps, 60
  - savetime, 61
  - starttime, 61
  - timestepping, 59, 60
- Torque
  - memoryHandling, 27

- type
  - boundaryCondition, [41](#)
- u
  - particle, [53](#)
  - registerMemory, [58](#)
- Uniform
  - particleHandling, [30](#)
- UseGPUWideThreadSync
  - settings.cuh, [109](#)
- v
  - particle, [54](#)
  - registerMemory, [58](#)
- var\_type
  - settings.cuh, [105](#)
- vec3D
  - math.cuh, [93](#)
- vector, [61](#)
  - length, [62](#)
  - operator\*, [62](#)
  - operator<sup>^</sup>, [63](#)
  - operator+, [63](#)
  - operator-, [63](#)
  - vector, [62](#)
  - x, [63](#)
  - y, [63](#)
  - z, [63](#)
- Velocity
  - memoryHandling, [27](#)
- vmean
  - particleDistribution, [55](#)
- VOLUME\_FACTOR
  - constant, [10](#)
- vsigma
  - particleDistribution, [55](#)
- x
  - bodyForce, [37](#)
  - coordinate, [44](#)
  - coordinates, [45](#)
  - vector, [63](#)
- y
  - bodyForce, [37](#)
  - coordinate, [44](#)
  - coordinates, [45](#)
  - vector, [63](#)
- z
  - bodyForce, [38](#)
  - coordinate, [45](#)
  - coordinates, [46](#)
  - vector, [63](#)
- ZERO
  - constant, [10](#)