Hierarchical

Generated by Doxygen 1.12.0

Chapter 1

Hierarchical (Overview)

Hierarchical contains my attempt to implement serial versions of hierarchical N-body simulation algorithms in C++. This includes the Barnes-Hut(BH) method and the Fast Multipole Method (FMM). A brute-force algorithm is also included for comparison.

1.1 Directory structure

For explanation on the role of each file, see **File Index** in the Doxygen-generated documentation.

- `src`: Where the main body of the project is stored (C++).
- `tests`: C++ test files, depends on `doctest`.
- `build`: Created on calling `make`; Where binaries and object files are stored.
- `pysrc`: Where all Python source code for data processing is stored.
- `data`: Where pre-written experiments dump their test results. This folder is expected to be present for them, or else they will just abort.
- `docs`: Documentation, `Doxyfile`.
- `notes`: Contains report and some other notes in markdown format.
- `scripts`: Very short bash scripts for small tasks.
- `makefiles`: Stores all `make` targets.
- `animations`: Animations to three simulated examples.

1.2 Build Instructions

1.2.1 `hchl`: Main C++ binary

Assuming one is on a Unix-like system, simply navigate to the project root, then type:
```
make
```

This requires that one has `GNU Make` and `clang` installed. Alternatively to use `gcc`, replace occurrences of `clang++` with `g++` for all files in the `makefile` folder.

This will compile all files in `src` and `main.cpp`, which has the `main` function, the entry point of the program. Uncomment some of the lines in `main` then build to see the effect of individual tests.

After compilation finishes, the binary can then be found at `build/hchl`. When benchmarking, I run `./run` to set CPU affinities.

1.2.2 `test`: C++ tests with `doctest`

I have written some tests for parts of the `C++` source. In order to compile this, make sure the `doctest` header is installed. On my system (Arch Linux), I can then write in the test files

```
#include "doctest/doctest.h"
```

This may have to be replaced with your own library location, which one specify by adding `-Ipath/to/doctest/lib` in the `CXX_FLAGS` variable in `makefile.test`.

To then compile the test run

```
make test
```

The binary will be found at `build/test`.

Note that this will be compiled with `-O0` flag and is incompatible with previous object files from the `hchl` target compiled with `-O3`. Therefore `make clean` (see below) must be run first before changing C++ target.

1.2.3 `report`: PDF Generation from markdown

To compile my report, written in `pandoc markdown` at `notes/report.md`, call

```
make report
```

The compiled report will be found at `notes/report.pdf`. All other markdown files found in `notes/` will also be compiled. Requires `pandoc` and the `pandoc-crossref` filter.

1.2.4 `doc`: `Doxygen` documentation generation

I have written `Doxygen`-compatible docstrings in both C++ and Python code. To make the documentation with `doxygen`, run

```
make doc
```

This calls `doxygen` with file options set in `docs/Doxyfile`. For `html` version open `docs/html/index.html` with a browser. For `pdf`, open `docs/latex/refman.pdf`. Requires `doxygen`. PDF compilation requires appropriate `latex` packages.

1.2.5 `clean`: Directory clean-up

Cleans all files resulting from compilation and `make` commands. Please note it will also **remove .plist files in the project root**, since my language servers are generating a lot of garbage in the format.

1.2.6 `Python`

To run Python code, simply edit the `main` function of `main.py` and execute the file. Comment and uncomment lines to see the working of data processing functions, provided that data has been generated previously from `C++`, stored to `data/`, and that the script is executed at the project root.

Some common python libraries such as `matplotlib` and `scipy` will be required. Additionally, parts of the code (animation) depends on `ffmpeg` and a Unix-like shell environment.

1.3 Coding style

The coding style of the project largely follows that outlined in Section 6.6 of *Guide to Scientific Computing in C++, Second Edition* by Pitt-Francis & Whiteley.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

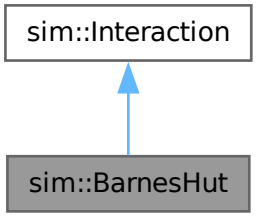
Chapter 5

Class Documentation

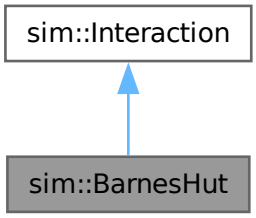
5.1 sim::BarnesHut Class Reference

Calculate forces of a whole grid with the Barnes-Hut method.

Inheritance diagram for sim::BarnesHut:



Collaboration diagram for sim::BarnesHut:



Public Member Functions

- **BarnesHut** (int p, double theta, Kernels ∗mKernels, Force const ∗forceLaw)
- int **GetP** () const
- double **GetTheta** () const
- Grid Calculate (const Grid &g1) const override

  Calculate force for all particles in grid.

Public Member Functions inherited from sim::Interaction

- **Interaction** (Force const ∗forceLaw)

Additional Inherited Members

Protected Member Functions inherited from sim::Interaction

- Vec GetForce (const Particle &p1, const Particle &p2) const

  Get force due to p2 on p1.
- double **GetPot** (const Particle &p1, const Particle &p2) const

5.1.1 Detailed Description

Calculate forces of a whole grid with the Barnes-Hut method.

5.1.2 Member Function Documentation

5.1.2.1 Calculate()

```
Grid sim::BarnesHut::Calculate (
            const Grid & g1) const  [override], [virtual]
```

Calculate force for all particles in grid.

Parameters

in	*g1*	Original grid

Returns

Grid with accel and potential overwritten in particles

Implements sim::Interaction.

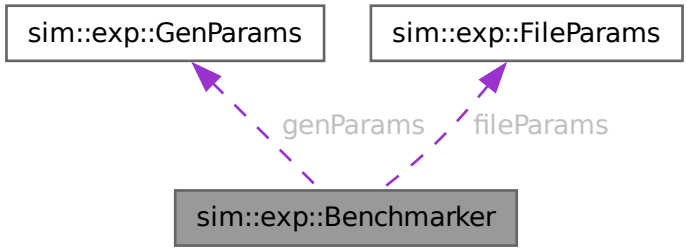
The documentation for this class was generated from the following files:

- Barnes-Hut.hpp
- Barnes-Hut.cpp

5.2 sim::exp::Benchmarker Class Reference

Class to help benchmark and test different algorithms.

Collaboration diagram for sim::exp::Benchmarker:



Public Member Functions

- Benchmarker (GenParams genParams, FileParams fileParams, std::ostream &stream=std::cout)

  Initialised Benchmarker.
- void Run (double param, bool skipBrute) const

  Run the repeats by configuration in genParams and fileParams.
- void Evo (const Grid &grid) const

  Perform time evolution by parameters in genParams and fileParams.

Public Attributes

- std::unique_ptr< LeapFrog > **integrator**
- std::unique_ptr< InvSqForce > **force**
- std::unique_ptr< InvSqKernels > **kernels**
- std::unique_ptr< Interaction > **interactions** [genDefParams.intTypes]
- std::ostream & **stream**
- GenParams **genParams**
- FileParams **fileParams**

5.2.1 Detailed Description

Class to help benchmark and test different algorithms.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Benchmarker()

```
sim::exp::Benchmarker::Benchmarker (
            GenParams genParams,
            FileParams fileParams,
            std::ostream & stream = std::cout)
```

Initialised Benchmarker.

Parameters

in	*genParams*	General parameters for simulations
in	*fileParams*	File-related parameters
in,out	*stream*	Stream to dump finished grids to

5.2.3 Member Function Documentation

5.2.3.1 Evo()

```
void sim::exp::Benchmarker::Evo (
            const Grid & grid) const
```

Perform time evolution by parameters in genParams and fileParams.

Parameters

in	*grid*	Initial condition for simulation.

5.2.3.2 Run()

```
void sim::exp::Benchmarker::Run (
            double param,
            bool skipBrute) const
```

Run the repeats by configuration in genParams and fileParams.

The Benchmarker itself is agnostic to which parameter is actually being varied, but this needs to logged to file so we provide the value (doesn't matter which param it actually is) to the method for it to output.

Parameters

in	*param*	Value of parameter being varied in this run
in	*skipBrute*	Whether to skip run with brute-force

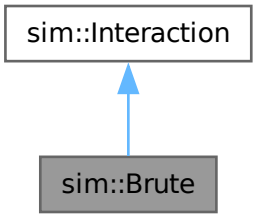
The documentation for this class was generated from the following files:

- Benchmarker.hpp
- Benchmarker.cpp

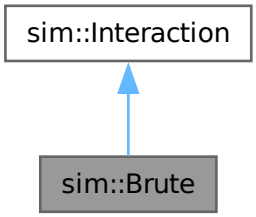
5.3 sim::Brute Class Reference

Brute-force implementation of force calculation for whole grid.

Inheritance diagram for sim::Brute:



Collaboration diagram for sim::Brute:



Public Member Functions

- **Brute** (Force const ∗forceLaw)
- Grid Calculate (const Grid &g1) const override

  Calculate potential and acceleration for each charge in grid.

Public Member Functions inherited from sim::Interaction

- **Interaction** (Force const ∗forceLaw)

Additional Inherited Members

Protected Member Functions inherited from sim::Interaction

- Vec GetForce (const Particle &p1, const Particle &p2) const

  Get force due to p2 on p1.
- double **GetPot** (const Particle &p1, const Particle &p2) const

5.3.1 Detailed Description

Brute-force implementation of force calculation for whole grid.

5.3.2 Member Function Documentation

5.3.2.1 Calculate()

```
Grid sim::Brute::Calculate (
            const Grid & g1) const  [override], [virtual]
```

Calculate potential and acceleration for each charge in grid.

Parameters

in	*g1*	input grid

Returns

Grid containing particles whose accel and potential are updated.

Implements sim::Interaction.

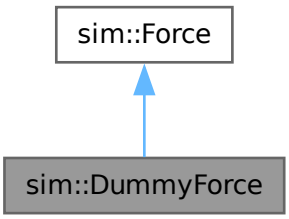
The documentation for this class was generated from the following files:

- Brute.hpp
- Brute.cpp

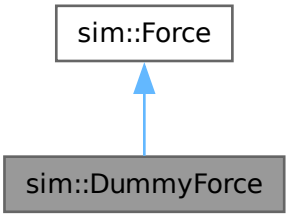
5.4 sim::DummyForce Class Reference

Dummy implementation of Force that does nothing.

Inheritance diagram for sim::DummyForce:



Collaboration diagram for sim::DummyForce:



Public Member Functions

- Vec ForceLaw (const Particle &p1, const Particle &p2) const override
- double PotLaw (const Particle &p1, const Particle &p2) const override

Public Member Functions inherited from sim::Force

- Vec GetForce (const Particle &p1, const Particle &p2) const

  Get force of p2 on p1.
- double GetPot (const Particle &p1, const Particle &p2) const

  Get potential between two particles.

Additional Inherited Members

Protected Member Functions inherited from sim::Force

5.4.1 Detailed Description

Dummy implementation of Force that does nothing.

5.4.2 Member Function Documentation

5.4.2.1 ForceLaw()

```
Vec sim::DummyForce::ForceLaw (
            const Particle & p1,
            const Particle & p2) const  [override], [virtual]
```

Implements sim::Force.

5.4.2.2 PotLaw()

```
double sim::DummyForce::PotLaw (
            const Particle & p1,
            const Particle & p2) const  [override], [virtual]
```

Implements sim::Force.

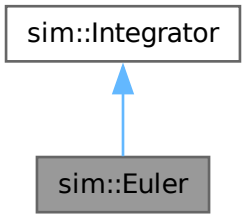
The documentation for this class was generated from the following files:

- Force.hpp
- Force.cpp

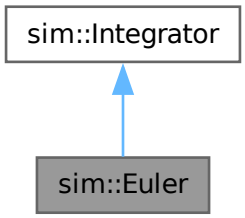
5.5 sim::Euler Class Reference

Euler integrator implementing the Integrator interface.

Inheritance diagram for sim::Euler:



Collaboration diagram for sim::Euler:



Public Member Functions

- **Euler** (double step)
- Particle Evolve (const Particle &p1, const double step) const override
- Particle Evolve (const Particle &p1) const

  Evolve a particle forward by internal step size.
- Grid Evolve (const Grid &g1, const double step) const

  Evolve full grid forward in time by step.
- Grid Evolve (const Grid &g1) const

  Evolve full grid forward in time by internal step size.

Public Member Functions inherited from sim::Integrator

- Integrator (double step)

  Initialise integrator.
- double **GetStep** () const
- Particle Evolve (const Particle &p1) const

  Evolve a particle forward by internal step size.
- Grid Evolve (const Grid &g1, const double step) const

  Evolve full grid forward in time by step.
- Grid Evolve (const Grid &g1) const

  Evolve full grid forward in time by internal step size.

Additional Inherited Members

Protected Attributes inherited from sim::Integrator

- const double **mStep**

5.5.1 Detailed Description

Euler integrator implementing the Integrator interface.

5.5.2 Member Function Documentation

5.5.2.1 Evolve() [1/4]

```
Grid sim::Integrator::Evolve (
            const Grid & g1) const
```

Evolve full grid forward in time by internal step size.

Parameters

in	*g1*	Grid to be evolved in time

Returns

Grid evolved in time

5.5.2.2 Evolve() [2/4]

```
Grid sim::Integrator::Evolve (
            const Grid & g1,
            const double step) const
```

Evolve full grid forward in time by step.

Derived classes must implement Evolve(Particle, double).

Parameters

in	*g1*	Grid to be evolved in time
in	*step*	step size

Returns

Grid evolved in time

5.5.2.3 Evolve() [3/4]

```
Particle sim::Integrator::Evolve (
            const Particle & p1) const
```

Evolve a particle forward by internal step size.

Parameters

in	*p1*	Particle with acceleration precalculated

Returns

Particle after an internal step size.

5.5.2.4 Evolve() [4/4]

```
Particle sim::Euler::Evolve (
            const Particle & p1,
            const double step) const  [override], [virtual]
```

Implements sim::Integrator.

The documentation for this class was generated from the following files:

- Integrator.hpp
- Integrator.cpp

5.6 sim::exp::FileParams Class Reference

Parameters related to file storage in an experiment.

Public Member Functions

- **FileParams** (const std::string runDir)
- std::string **GetCompName** () const

  Get relative path to file where complexity timing should go.
- std::string **GetDumpName** () const

  Get relative path to directory where grids should be dumped.
- std::string **GetRunDirName** () const

  Get relative path to the directory where experiment will run.

Public Attributes

- std::string **runDir** = ""
- std::string **comp** = "complexity.out"
- std::string **dump** = "dump/"

Static Public Attributes

- static const std::string **dataDir** = "./data/"

5.6.1 Detailed Description

Parameters related to file storage in an experiment.

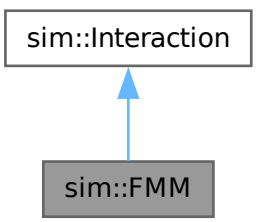
The documentation for this class was generated from the following files:

- Benchmarker.hpp
- Benchmarker.cpp

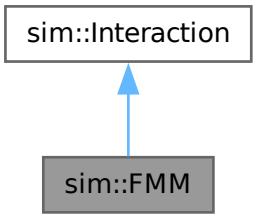
5.7 sim::FMM Class Reference

Fast Multipole Method implementation.

Inheritance diagram for sim::FMM:



Collaboration diagram for sim::FMM:



Public Member Functions

- **FMM** (int p, double theta, int maxPerCell, int maxPairwiseLimit, Kernels ∗kernels, Force const ∗forceLaw)
- int **GetP** () const
- double **GetTheta** () const
- Grid Calculate (const Grid &g1) const override

  Calculate acceleration and potential of all particles in grid g1.

Public Member Functions inherited from sim::Interaction

- **Interaction** (Force const ∗forceLaw)

Additional Inherited Members

Protected Member Functions inherited from sim::Interaction

- Vec GetForce (const Particle &p1, const Particle &p2) const

  Get force due to p2 on p1.
- double **GetPot** (const Particle &p1, const Particle &p2) const

5.7.1 Detailed Description

Fast Multipole Method implementation.

5.7.2 Member Function Documentation

5.7.2.1 Calculate()

```
Grid sim::FMM::Calculate (
            const Grid & g1) const  [override], [virtual]
```

Calculate acceleration and potential of all particles in grid g1.

Parameters

in	*g1*	Initial grid for which acceleration and potential is calculated

Returns

Grid with acceleration and potential information

Implements sim::Interaction.

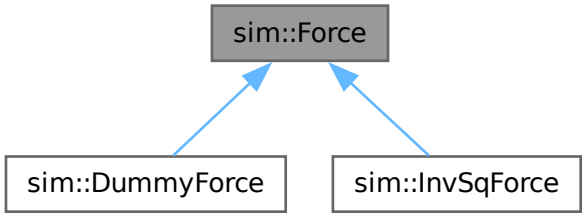
The documentation for this class was generated from the following files:

- FMM.hpp
- FMM.cpp

5.8 sim::Force Class Reference

Virtual class for a general pairwise force.

Inheritance diagram for sim::Force:



Public Member Functions

- Vec GetForce (const Particle &p1, const Particle &p2) const

  Get force of p2 on p1.
- double GetPot (const Particle &p1, const Particle &p2) const

  Get potential between two particles.

Protected Member Functions

- virtual Vec **ForceLaw** (const Particle &p1, const Particle &p2) const =0
- virtual double **PotLaw** (const Particle &p1, const Particle &p2) const =0

5.8.1 Detailed Description

Virtual class for a general pairwise force.

5.8.2 Member Function Documentation

5.8.2.1 GetForce()

```
Vec sim::Force::GetForce (
            const Particle & p1,
            const Particle & p2) const
```

Get force of p2 on p1.

This checks for particle overlap. Derived classes implement instead ForceLaw, which carries out no such checks.

Returns

If two particles do not overlap return force. Otherwise, null vector.

5.8.2.2 GetPot()

```
double sim::Force::GetPot (
            const Particle & p1,
            const Particle & p2) const
```

Get potential between two particles.

This checks for particle overlap. Derived classes implement instead PotLaw, which does not carry out such checks.

Returns

If two particles do not overlap return potential. Otherwise, 0.

The documentation for this class was generated from the following files:

- Force.hpp
- Force.cpp

5.9 sim::exp::GenParams Class Reference

General parameters to be used in an experiment.

Public Types

- enum **intVals** { **brute** , **bh** , **fmm** }

Public Attributes

- int **n** = 1000
- int **p** = 3
- double **theta** = 0.5
- int **repeats** = 10
- int **maxPerCell** = 5
- int **maxPairwiseLimit** = 3
- double **G** = -1
- double **step** = 0.01
- double **evo_time** = 10
- double **scale** = 10
- std::string **intNames** [intTypes] {"brute", "bh", "fmm"}

Static Public Attributes

- static const int **intTypes** = 3

5.9.1 Detailed Description

General parameters to be used in an experiment.

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

- Benchmarker.hpp

5.10 pysrc.Grid.Grid Class Reference

Python side boiled-down implementation of the C++ Grid class.

Public Member Functions

- **__init__** (self)
- **GetSize** (self)
- **__getitem__** (self, int ind)

Public Attributes

- int **mDim** = 3
- npt.ArrayLike **mMaxLim** = np.zeros([self.mDim, 2])
- npt.ArrayLike **mOctant** = np.zeros([self.mDim, 2])
- list **mParticles** = [ ]

5.10.1 Detailed Description

Python side boiled-down implementation of the C++ Grid class.

Represents a list of particles, and the dimensions of the simulation.

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

- Grid.py

5.11 sim::Grid Class Reference

Data structure to describe simulation space and particles within.

Public Member Functions

- Grid (Octant maxLim=Octant())
  Initialise setting maxLim.
- Grid (int size, Octant maxLim=Octant())
  Initialise grid, preallocating space for (size) particles.
- Grid (const std::vector< Particle > &pars, Octant maxLim=Octant())
  Initialise grid with a list of particles.
- int GetSize () const
  Get number of particles in grid.
- const Octant GetLimits () const
  Get Maximum limits imposed on simulation space.
- const Octant GetOctant () const
  Get current simulation space limits.
- Particle & operator[ ] (int index)
  Get particle by index (soul) in grid.
- Particle operator[ ] (int index) const
  Get Particle by index (soul) in grid.
- void AddParticle (const Particle &par)
  Append particle to grid, resizing simulation space if needed.
- void AddParticles (const std::vector< Particle > &par_list)
  Append a list of particles to grid.
- double GetPE () const
  Get total potential energy in grid.
- double GetKE () const
  Get total kinetic energy in grid.
- double GetE () const
  Get total energy in grid.
- Vec GetCOM () const
  Get centre of mass of grid.
- Vec GetL (const Vec centre=Vec({0, 0, 0})) const
  Get angular momentum of grid.
- Vec GetP () const
  Get total momentum of grid.
- void SetOctant (const Octant &octant)
  Force set grid dimensions.
- void Reserve (int size)
  Reserve space for (size) particles.

5.11.1 Detailed Description

Data structure to describe simulation space and particles within.

If maxLim is not initialised, grid expands its simulation space dynamically as particles are appended. if maxLim is initialised however, space can grow only up to maxLim, and particles lying outside are rejected on appending.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Grid() [1/3]

```
sim::Grid::Grid (
            Octant maxLim = Octant())
```

Initialise setting maxLim.

Parameters

in	*maxLim*	Maximum limits for the simulation space

5.11.2.2 Grid() [2/3]

```
sim::Grid::Grid (
            int size,
            Octant maxLim = Octant())
```

Initialise grid, preallocating space for (size) particles.

Parameters

in	*size*	Expected number of particles
in	*maxLim*	Maximum limits for simulation space

5.11.2.3 Grid() [3/3]

```
sim::Grid::Grid (
            const std::vector< Particle > & pars,
            Octant maxLim = Octant())
```

Initialise grid with a list of particles.

Parameters

in	*pars*	List of particles to add to grid
in	*maxLim*	maximum limits for simulation space

5.11.3 Member Function Documentation

5.11.3.1 AddParticle()

```
void sim::Grid::AddParticle (
            const Particle & par)
```

Append particle to grid, resizing simulation space if needed.

If maxLim is initialised for the grid, and the particle is outside that limit, then it is merely ignored and not added to the grid.

Parameters

in	*par*	Particle to be added

5.11.3.2 AddParticles()

```
void sim::Grid::AddParticles (
            const std::vector< Particle > & par_list)
```

Append a list of particles to grid.

Parameters

in	*par_list*	List of particles to be added.

5.11.3.3 GetCOM()

```
Vec sim::Grid::GetCOM () const
```

Get centre of mass of grid.

Returns

Centre of mass of all particles in grid.

5.11.3.4 GetE()

```
double sim::Grid::GetE () const
```

Get total energy in grid.

Returns

Total energy.

5.11.3.5 GetKE()

```
double sim::Grid::GetKE () const
```

Get total kinetic energy in grid.

Returns

Total kinetic energy

5.11.3.6 GetL()

```
Vec sim::Grid::GetL (
            const Vec centre = Vec({0, 0, 0})) const
```

Get angular momentum of grid.

Parameters

in	*centre*	Centre for L calculation. Defaults to the origin.

Returns

Angular momentum about set centre

5.11.3.7 GetLimits()

```
const Octant sim::Grid::GetLimits () const
```

Get Maximum limits imposed on simulation space.

Returns

An Octant object. If uninitialised, grid does not have a size limit.

5.11.3.8 GetOctant()

```
const Octant sim::Grid::GetOctant () const
```

Get current simulation space limits.

Returns

Current simulation space limits.

5.11.3.9 GetP()

```
Vec sim::Grid::GetP () const
```

Get total momentum of grid.

Returns

Sum of momentum of all particles

5.11.3.10 GetPE()

```
double sim::Grid::GetPE () const
```

Get total potential energy in grid.

This assumes the interaction is symmetric and pairwise.

Returns

Total potential energy.

5.11.3.11 GetSize()

```
int sim::Grid::GetSize () const
```

Get number of particles in grid.

Returns

Number of particles in grid.

5.11.3.12 operator[]() [1/2]

```
Particle & sim::Grid::operator[] (
            int index)
```

Get particle by index (soul) in grid.

Returns

Particle at provided index

5.11.3.13 operator[]() [2/2]

```
Particle sim::Grid::operator[] (
            int index) const
```

Get Particle by index (soul) in grid.

Returns

particle at provided index

5.11.3.14 Reserve()

```
void sim::Grid::Reserve (
            int size)
```

Reserve space for (size) particles.

Parameters

in	*size*	Number of particles to reserve space for.

5.11.3.15 SetOctant()

```
void sim::Grid::SetOctant (
            const Octant & octant)
```

Force set grid dimensions.

This is only possible when the grid dimensions are uninitialised, that is when there are no particles in grid.

Parameters

in	*Grid*	dimensions

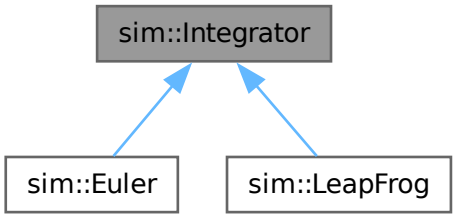
The documentation for this class was generated from the following files:

- Grid.hpp
- Grid.cpp

5.12 sim::Integrator Class Reference

Defines the general integrator interface.

Inheritance diagram for sim::Integrator:



Public Member Functions

- Integrator (double step)
  Initialise integrator.
- double **GetStep** () const
- virtual Particle **Evolve** (const Particle &p1, const double step) const =0
- Particle Evolve (const Particle &p1) const
  Evolve a particle forward by internal step size.
- Grid Evolve (const Grid &g1, const double step) const
  Evolve full grid forward in time by step.
- Grid Evolve (const Grid &g1) const
  Evolve full grid forward in time by internal step size.

Protected Attributes

- const double **mStep**

5.12.1 Detailed Description

Defines the general integrator interface.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Integrator()

```
sim::Integrator::Integrator (
            double step)
```

Initialise integrator.

Parameters

in	*step*	Step size of integrator.

5.12.3 Member Function Documentation

5.12.3.1 Evolve() [1/3]

```
Grid sim::Integrator::Evolve (
            const Grid & g1) const
```

Evolve full grid forward in time by internal step size.

Parameters

in	*g1*	Grid to be evolved in time

Returns

Grid evolved in time

5.12.3.2 Evolve() [2/3]

```
Grid sim::Integrator::Evolve (
            const Grid & g1,
            const double step) const
```

Evolve full grid forward in time by step.

Derived classes must implement Evolve(Particle, double).

Parameters

in	*g1*	Grid to be evolved in time
in	*step*	step size

Returns

Grid evolved in time

5.12.3.3 Evolve() [3/3]

```
Particle sim::Integrator::Evolve (
            const Particle & p1) const
```

Evolve a particle forward by internal step size.

Parameters

in	*p1*	Particle with acceleration precalculated

Returns

Particle after an internal step size.

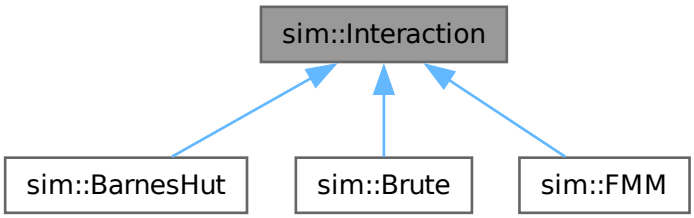
The documentation for this class was generated from the following files:

- Integrator.hpp
- Integrator.cpp

5.13 sim::Interaction Class Reference

Class defining the general Interaction interface.

Inheritance diagram for sim::Interaction:



Public Member Functions

- **Interaction** (Force const ∗forceLaw)
- virtual Grid Calculate (const Grid &g1) const =0

Protected Member Functions

- Vec GetForce (const Particle &p1, const Particle &p2) const

  Get force due to p2 on p1.
- double **GetPot** (const Particle &p1, const Particle &p2) const

5.13.1 Detailed Description

Class defining the general Interaction interface.

5.13.2 Member Function Documentation

5.13.2.1 Calculate()

```
virtual Grid sim::Interaction::Calculate (
            const Grid & g1) const  [pure virtual]
```

Implemented in sim::BarnesHut, sim::Brute, and sim::FMM.

5.13.2.2 GetForce()

```
Vec sim::Interaction::GetForce (
            const Particle & p1,
            const Particle & p2) const  [protected]
```

Get force due to p2 on p1.

Returns

force.

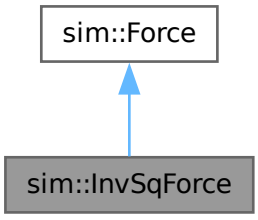
The documentation for this class was generated from the following files:

- Interaction.hpp
- Interaction.cpp

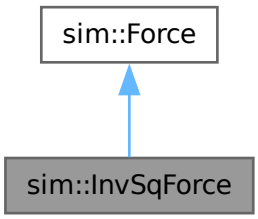
5.14 sim::InvSqForce Class Reference

Implementation of Force that provides the inverse-square law.

Inheritance diagram for sim::InvSqForce:



Collaboration diagram for sim::InvSqForce:



Public Member Functions

- InvSqForce (double G=-1)

  Initialiser for InvSqForce.
- Vec ForceLaw (const Particle &p1, const Particle &p2) const override
- double PotLaw (const Particle &p1, const Particle &p2) const override

Public Member Functions inherited from sim::Force

- Vec GetForce (const Particle &p1, const Particle &p2) const

  Get force of p2 on p1.
- double GetPot (const Particle &p1, const Particle &p2) const

  Get potential between two particles.

Protected Attributes

- double **mG**

Additional Inherited Members

Protected Member Functions inherited from sim::Force

5.14.1 Detailed Description

Implementation of Force that provides the inverse-square law.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 InvSqForce()

```
sim::InvSqForce::InvSqForce (
            double G = -1)
```

Initialiser for InvSqForce.

Parameters

in	*G*	Coupling constant. In our convention G $<$ 0 for gravity.

5.14.3 Member Function Documentation

5.14.3.1 ForceLaw()

```
Vec sim::InvSqForce::ForceLaw (
            const Particle & p1,
            const Particle & p2) const  [override], [virtual]
```

Implements sim::Force.

5.14.3.2 PotLaw()

```
double sim::InvSqForce::PotLaw (
            const Particle & p1,
            const Particle & p2) const  [override], [virtual]
```

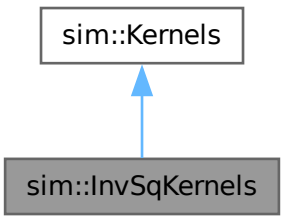
Implements sim::Force.

The documentation for this class was generated from the following files:

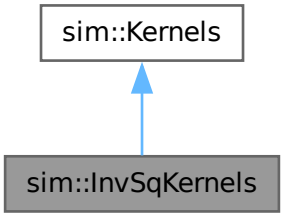
- Force.hpp
- Force.cpp

5.15 sim::InvSqKernels Class Reference

Inheritance diagram for sim::InvSqKernels:



Collaboration diagram for sim::InvSqKernels:



Public Types

- typedef std::complex< double > **cdouble**

Public Member Functions

- InvSqKernels (int p, double G=-1)

  Initialise InvSqKernels.
- cdouble Gamma (const Vec &v, int n, int m) const

  Calculate a specific solid harmonics gamma via recurrence.
- cdouble **Theta** (const Vec &v, int n, int m) const

  Calulate a single solid harmonics theta via recurrence.
- ComplexMatrix GammaCopy (const Vec &v, int n)

  Calculate and return a copy of gamma coefficients of order n.
- ComplexMatrix ThetaCopy (const Vec &v, int n)

  Calculate and return a copy of theta coefficients of order n.
- void P2M (Octree *leaf) override

  P2M kernel. (Eq 3c, Dehnen 2014)
- void M2M (Octree const *child, Octree *parent) override

  M2M kernel (Eq 3d, Dehnen 2014)
- ComplexMatrix M2X (Octree const *source, const Vec &s) override

  Implements M2L, M2P kernels (Eq 3b, Dehnen 2014)
- ComplexMatrix L2X (Octree const *previous, const Vec &sp) override

  Implements L2L, L2P kernels (Eq 3f, Dehnen 2014)

Public Member Functions inherited from sim::Kernels

- Kernels (int p)

  Initialises Kernels.
- void **M2L** (Octree const *source, Octree *sink)
- void **M2P** (Octree const *source, Particle &sinkPar)
- void **L2L** (Octree const *parent, Octree *child)
- void L2P (Octree const *leaf, Particle &containedPar)

  Apply L2P kernel to particle contained in leaf node.
- void CalculateM (Octree *node)

  Calculate M coefficients on a fully-built octree.

Additional Inherited Members

Protected Member Functions inherited from sim::Kernels

Protected Attributes inherited from sim::Kernels

- const int **mP**

5.15.1 Constructor & Destructor Documentation

5.15.1.1 InvSqKernels()

```
sim::InvSqKernels::InvSqKernels (
            int p,
            double G = -1)
```

Initialise InvSqKernels.

Parameters

in	*p*	Order of multipole expansion.
in	*G*	Coupling constant in inverse-square law force.

5.15.2 Member Function Documentation

5.15.2.1 Gamma()

```
InvSqKernels::cdouble sim::InvSqKernels::Gamma (
            const Vec & v,
            int n,
            int m) const
```

Calculate a specific solid harmonics gamma via recurrence.

Kept for completeness, but there is no need for this - in all applications we just use the preprocessed full matrix which is quicker.

5.15.2.2 GammaCopy()

```
ComplexMatrix sim::InvSqKernels::GammaCopy (
            const Vec & v,
            int n)
```

Calculate and return a copy of gamma coefficients of order n.

Returns

solid harmonics gamma of order n.

5.15.2.3 L2X()

```
ComplexMatrix sim::InvSqKernels::L2X (
            Octree const * previous,
            const Vec & sp)  [override], [virtual]
```

Implements L2L, L2P kernels (Eq 3f, Dehnen 2014)

Parameters

in	*previous*	Node with known F coefficients
in	*sp*	Target location

Returns

F coefficients at sp

Implements sim::Kernels.

5.15.2.4 M2M()

```
void sim::InvSqKernels::M2M (
            Octree const * child,
            Octree * parent)  [override], [virtual]
```

M2M kernel (Eq 3d, Dehnen 2014)

Parameters

in	*child*	Child node with known M coefficients
in,out	*parent*	parent node to add shifted M coefficients to

Implements sim::Kernels.

5.15.2.5 M2X()

```
ComplexMatrix sim::InvSqKernels::M2X (
            Octree const * source,
            const Vec & s)  [override], [virtual]
```

Implements M2L, M2P kernels (Eq 3b, Dehnen 2014)

M2L and M2P both finds F coefficients at a target location, so are the same thing.

Parameters

in	*source*	Node with known M coefficients
in	*Target*	position

Returns

F coefficients at s position

Implements sim::Kernels.

5.15.2.6 P2M()

```
void sim::InvSqKernels::P2M (
            Octree * leaf)  [override], [virtual]
```

P2M kernel. (Eq 3c, Dehnen 2014)

Parameters

in,out	*leaf*	Leaf node to apply P2M kernel to.

Implements sim::Kernels.

5.15.2.7 ThetaCopy()

```
ComplexMatrix sim::InvSqKernels::ThetaCopy (
            const Vec & v,
            int n)
```

Calculate and return a copy of theta coefficients of order n.

Returns

solid harmonics theta of order n.

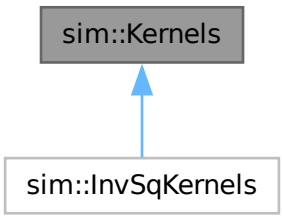
The documentation for this class was generated from the following files:

- InvSqKernels.hpp
- InvSqKernels.cpp

5.16 sim::Kernels Class Reference

Definition of the general Kernels interface.

Inheritance diagram for sim::Kernels:



Public Member Functions

- Kernels (int p)

  Initialises Kernels.
- virtual void P2M (Octree ∗leaf)=0
- virtual void M2M (Octree const ∗child, Octree ∗parent)=0
- virtual ComplexMatrix M2X (Octree const ∗source, const Vec &s)=0
- void **M2L** (Octree const ∗source, Octree ∗sink)
- void **M2P** (Octree const ∗source, Particle &sinkPar)
- virtual ComplexMatrix L2X (Octree const ∗previous, const Vec &sp)=0
- void **L2L** (Octree const ∗parent, Octree ∗child)
- void L2P (Octree const ∗leaf, Particle &containedPar)

  Apply L2P kernel to particle contained in leaf node.
- void CalculateM (Octree ∗node)

  Calculate M coefficients on a fully-built octree.

Protected Member Functions

- virtual void **AddAccel** (Particle &par, const ComplexMatrix &psi) const =0

Protected Attributes

- const int **mP**

5.16.1 Detailed Description

Definition of the general Kernels interface.

Non-const reference or pointer parameters in method signature are usually subject to direct modification, as they are the the "target" of the operation.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 Kernels()

```
sim::Kernels::Kernels (
            int p)
```

Initialises Kernels.

Parameters

in	*p*	order of multipole expansion

5.16.3 Member Function Documentation

5.16.3.1 CalculateM()

```
void sim::Kernels::CalculateM (
            Octree * node)
```

Calculate M coefficients on a fully-built octree.

Parameters

in,out	*node*	Current position in DFS. Start from root.

5.16.3.2 L2P()

```
void sim::Kernels::L2P (
            Octree const * leaf,
            Particle & containedPar)
```

Apply L2P kernel to particle contained in leaf node.

There is a reason we don't enumerate particles on leaf directly. We can make leaf non-const, but Octree stores m↩ Grid as const to prevent side effects. We can make that non-const as well, but when building the Octree sometimes we have only available to us the original grid (We shouldn't need to first create the result grid just to build the octree anyway.) Creating a copy of the grid also won't work, since modifications to it won't go back to the output result grid anyway. All things considered this awkward signature is the best compromise and makes its effects explicit. (What a headache!)

Parameters

in	*leaf*	Leaf node
in,out	*containedPar*	Particle to apply L2P to

5.16.3.3 L2X()

```
virtual ComplexMatrix sim::Kernels::L2X (
            Octree const * previous,
            const Vec & sp)  [pure virtual]
```

Implemented in sim::InvSqKernels.

5.16.3.4 M2M()

```
virtual void sim::Kernels::M2M (
            Octree const * child,
            Octree * parent)  [pure virtual]
```

Implemented in sim::InvSqKernels.

5.16.3.5 M2X()

```
virtual ComplexMatrix sim::Kernels::M2X (
            Octree const * source,
            const Vec & s)  [pure virtual]
```

Implemented in sim::InvSqKernels.

5.16.3.6 P2M()

```
virtual void sim::Kernels::P2M (
            Octree * leaf)  [pure virtual]
```

Implemented in sim::InvSqKernels.

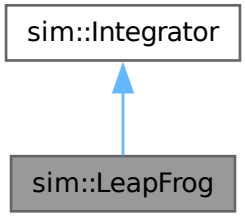
The documentation for this class was generated from the following files:

- Kernels.hpp
- Kernels.cpp

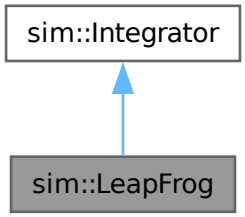
5.17 sim::LeapFrog Class Reference

Leap frog implementation of the Integrator interface.

Inheritance diagram for sim::LeapFrog:



Collaboration diagram for sim::LeapFrog:



Public Member Functions

- **LeapFrog** (double step)
- Particle Evolve (const Particle &p1, const double step) const override
- Particle Evolve (const Particle &p1) const

  Evolve a particle forward by internal step size.
- Grid Evolve (const Grid &g1, const double step) const

  Evolve full grid forward in time by step.
- Grid Evolve (const Grid &g1) const

  Evolve full grid forward in time by internal step size.

Public Member Functions inherited from sim::Integrator

- Integrator (double step)

  Initialise integrator.
- double **GetStep** () const
- Particle Evolve (const Particle &p1) const

  Evolve a particle forward by internal step size.
- Grid Evolve (const Grid &g1, const double step) const

  Evolve full grid forward in time by step.
- Grid Evolve (const Grid &g1) const

  Evolve full grid forward in time by internal step size.

Additional Inherited Members

Protected Attributes inherited from sim::Integrator

- const double **mStep**

5.17.1 Detailed Description

Leap frog implementation of the Integrator interface.

5.17.2 Member Function Documentation

5.17.2.1 Evolve() [1/4]

```
Grid sim::Integrator::Evolve (
            const Grid & g1) const
```

Evolve full grid forward in time by internal step size.

Parameters

in	*g1*	Grid to be evolved in time

Returns

Grid evolved in time

5.17.2.2 Evolve() [2/4]

```
Grid sim::Integrator::Evolve (
            const Grid & g1,
            const double step) const
```

Evolve full grid forward in time by step.

Derived classes must implement Evolve(Particle, double).

Parameters

in	*g1*	Grid to be evolved in time
in	*step*	step size

Returns

Grid evolved in time

5.17.2.3 Evolve() [3/4]

```
Particle sim::Integrator::Evolve (
            const Particle & p1) const
```

Evolve a particle forward by internal step size.

Parameters

in	*p1*	Particle with acceleration precalculated

Returns

Particle after an internal step size.

5.17.2.4 Evolve() [4/4]

```
Particle sim::LeapFrog::Evolve (
            const Particle & p1,
            const double step) const  [override], [virtual]
```

Implements sim::Integrator.

The documentation for this class was generated from the following files:

- Integrator.hpp
- Integrator.cpp

5.18 sim::Matrix< T > Class Template Reference

Public Member Functions

- **Matrix** (int nrows, int ncols)
- void **Resize** (int nrows, int ncols)
- const Row< T > & **operator[ ]** (int rowIndex) const
- Row< T > & **operator[ ]** (int rowIndex)
- Matrix< T > & **operator+=** (const Matrix< T > &otherMatrix)
- int **GetRows** () const
- int **GetCols** () const

The documentation for this class was generated from the following files:

- Matrix.hpp
- Row.hpp

5.19 sim::Octant Class Reference

Public Member Functions

- Octant (const double(&lim)[mDim][2])

  Data structure representing a cuboid box in space.
- **Octant** ()

  Instantiate an Octant without initialising it.
- void Relax (const Vec &vec)

  Ask octant to adjust its boundaries to accommodate new vector.
- bool **IsInitialised** () const
- bool **Within** (const Vec &vec) const

  Test if a vector is within the Octant boundary.
- int GetOctantNumber (const Vec &vec) const

  Get which octant a vector lies within the current Octant.
- Octant GetOctant (int octantNumber) const

  Get one octant of the current Octant, by its octant number.
- double **GetMaxLength** () const

  Get length of the longest side of Octant.
- double const * operator[ ] (int index) const

  Get the limits of the current Octant in the specified dimension.
- bool operator== (const Octant &otherOctant) const

  Two octants are equal if each of their limits agree.

Static Public Member Functions

- static Octant GetOctant (const Octant &octant, int octantNumber)

  Get one octant of the provided octant, by its octant number.

Static Public Attributes

- static const int **mDim** = 3
- static constexpr double **margin** = 2

5.19.1 Constructor & Destructor Documentation

5.19.1.1 Octant()

```
sim::Octant::Octant (
            const double(&) lim[mDim][2])
```

Data structure representing a cuboid box in space.

Octant has capabilities to obtain octant numbers, and find which octant a vector belongs to, therefore its name.

Instantiate an octant from limits directly

Octants intantiated this way are considered "initialised."

Parameters

in	*lim*	limits by dimension; second index corresponds to min and max.

5.19.2 Member Function Documentation

5.19.2.1 GetOctant() [1/2]

```
Octant sim::Octant::GetOctant (
            const Octant & octant,
            int octantNumber)  [static]
```

Get one octant of the provided octant, by its octant number.

This typically called after GetOctantNumber(). See documentation of that function for how octant number is defined.

Parameters

in	*octant*	Octant in which to find the sub-octants.
in	*octantNumber*	A number from 0 to 7 showing which sub-octant it is.

Returns

The sub-octant represented by octantNumber.

5.19.2.2 GetOctant() [2/2]

```
Octant sim::Octant::GetOctant (
            int octantNumber) const
```

Get one octant of the current Octant, by its octant number.

This typically called after GetOctantNumber(). See documentation of that function for how octant number is defined.

Parameters

in	*octantNumber*	A number from 0 to 7 showing which sub-octant it is.

Returns

The sub-octant represented by octantNumber.

5.19.2.3 GetOctantNumber()

```
int sim::Octant::GetOctantNumber (
            const Vec & vec) const
```

Get which octant a vector lies within the current Octant.

The octant number is a three-bit mask. The i-th bit is 1 if vec[i] lies above the midpoint of Octant in that dimension; Else 0.

Returns

Number from 0 to 7 encoding an octant.

5.19.2.4 operator==()

```
bool sim::Octant::operator== (
            const Octant & otherOctant) const
```

Two octants are equal if each of their limits agree.

This operation is not safe against floating point errors and is only meant to check whether two octants are identical copies of each other.

5.19.2.5 operator[]()

```
double const * sim::Octant::operator[] (
            int index) const
```

Get the limits of the current Octant in the specified dimension.

Do not allow modification, for this will break internal tracking of whether octant is initialised or not.

Parameters

in	*index*	Dimension to access

Returns

array of length 2, with lower and upper limits.

5.19.2.6 Relax()

```
void sim::Octant::Relax (
            const Vec & vec)
```

Ask octant to adjust its boundaries to accommodate new vector.

Parameters

in	*vec*	displacement of a vector

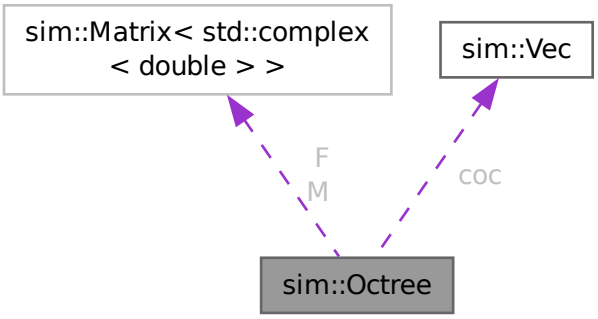
The documentation for this class was generated from the following files:

- Octant.hpp
- Octant.cpp

5.20 sim::Octree Class Reference

Octree (node) implementation.

Collaboration diagram for sim::Octree:



Public Member Functions

- Octree (Octree *parent, const Grid &grid, const Octant &new_oct, int maxParticles, int p)
  Initialise an Octree node.
- Octree * **GetChild** (int index)
  Get raw pointer to child at (index)
- void SetChild (int index, Octree *node)
  Set provided node as child at (index), taking ownership.
- Octree const * **GetChild** (int index) const
  Get raw pointer to constant child at (index)
- Octree * **GetParent** ()
  Get raw pointer to parent.
- Octree const * **GetParent** () const
  Get raw pointer to constant parent.
- double **GetMaxLength** () const
  Get length of longest side of the octant represented by this node.
- bool **IsLeaf** () const

Test if current node is a leaf.

- Particle **GetParticle** (int soul) const

  Get a copy of a particle from the grid by its soul number.
- int **GetMaxParticles** () const
- int **GetP** () const
- Octant **GetOctant** () const

  Get the 3D box/octant that this node represents.
- void AddParticle (int soul)

  Add a particle to this node.
- void SetOctant (const Octant &octant)

  Overwrite the octant that the current node represnets.

Static Public Member Functions

- static std::unique_ptr< Octree > BuildTree (const Grid &grid, int maxParticles, int p)

  Build octree from a grid.

Public Attributes

- std::vector< int > **mOctantSouls** [mBoxes]
- std::vector< int > **mSouls**
- ComplexMatrix **M**
- ComplexMatrix **F**
- Vec **coc** = Vec()
- double **charge** = 0

Static Public Attributes

- static const int **mBoxes** = 8
- static const int **mDim** = 3

5.20.1 Detailed Description

Octree (node) implementation.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 Octree()

```
sim::Octree::Octree (
            Octree * parent,
            const Grid & grid,
            const Octant & new_oct,
            int maxParticles,
            int p)
```

Initialise an Octree node.

Parameters

in	*parent*	pointer to parent node. Take nullptr for root.
in	*grid*	reference to grid the Octree is representing.
in	*new_oct*	Octant of space this node represents.
in	*maxParticles*	Max number of particles in the leaf node.
in	*p*	Order of muiltipole expansion.

5.20.3 Member Function Documentation

5.20.3.1 AddParticle()

```
void sim::Octree::AddParticle (
            int soul)
```

Add a particle to this node.

At a leaf node if limit maxParticles is exceeded then the node is automatically split. This should always be called at the root node, for as the particle is pushed down the relevant COCs are also updated.

Parameters

in	*soul*	Index of the particle in grid

5.20.3.2 BuildTree()

```
std::unique_ptr< Octree > sim::Octree::BuildTree (
            const Grid & grid,
            int maxParticles,
            int p)  [static]
```

Build octree from a grid.

Parameters

in	*grid*	Provides list of particles to construct the octree from
in	*maxParticles*	Max number of particles at leaf node
in	*p*	order of multipole expansion

5.20.3.3 SetChild()

```
void sim::Octree::SetChild (
            int index,
            Octree * octree)
```

Set provided node as child at (index), taking ownership.

The child is saved to a std::unique_ptr and so should not and cannot be deleted elsewhere. Each node has ownership over all their direct children, not the parent. Outside access of nodes should be with raw pointers as they don't take ownership. This way the ownership structure is clear and the tree can be recursively destructed upon destruction of root.

Parameters

in	*index*	index (octantNumber) of the child node
in	*octree*	Pointer to node to be set as child

5.20.3.4 SetOctant()

```
void sim::Octree::SetOctant (
            const Octant & octant)
```

Overwrite the octant that the current node represnets.

This should not be called outside of sim::IO, where this is used to load back the octants dumped to files.

The documentation for this class was generated from the following files:

- Octree.hpp
- Octree.cpp

5.21 pysrc.Particle.Particle Class Reference

Python side boiled-down representation of C++ Particle class.

Public Member Functions

- **__init__** (self, float mass, float charge)

Public Attributes

- **mass** = mass
- **charge** = charge

Static Public Attributes

- int **mDim** = 3
- npt **pos** = np.zeros(mDim)
- npt **vel** = np.zeros(mDim)
- npt **accel** = np.zeros(mDim)

5.21.1 Detailed Description

Python side boiled-down representation of C++ Particle class.

Represents a particle

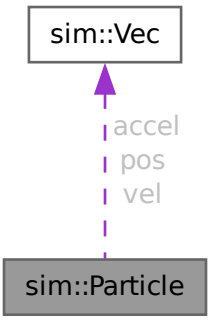
The documentation for this class was generated from the following file:

- Particle.py

5.22 sim::Particle Class Reference

Data structure representing a particle.

Collaboration diagram for sim::Particle:



Public Member Functions

- **Particle** (double mass, double charge)
- double **GetMass** () const
- double **GetCharge** () const
- double GetPE () const

  Get potential energy on the particle.
- double **GetKE** () const

  Get kinetic energy on the particle.
- Vec **GetP** () const

  Get momentum on the particle.

Public Attributes

- Vec **pos** = Vec()
- Vec **vel** = Vec()
- Vec **accel** = Vec()
- double **pot** = 0

5.22.1 Detailed Description

Data structure representing a particle.

5.22.2 Member Function Documentation

5.22.2.1 GetPE()

```
double sim::Particle::GetPE () const
```

Get potential energy on the particle.

Note the risk of double counting, since for pairwise forces potential is really a value shared between a pair, while we store this whole value in a particle.

The documentation for this class was generated from the following files:

- Particle.hpp
- Particle.cpp

5.23 sim::Row< T > Class Template Reference

Public Member Functions

- **Row** (int ncols)
- int **GetSize** () const
- T **operator[ ]** (int colIndex) const
- T & **operator[ ]** (int colIndex)
- Row< T > **operator+** (const Row< T > &otherRow) const
- Row< T > **operator-** (const Row< T > &otherRow) const
- Row< T > & **operator+=** (const Row< T > &otherRow)

Friends

- class **Matrix**< **T** >

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

- Row.hpp

5.24 pysrc.IO.Stats Class Reference

Class storing some statistics for a timestep of time evolution.

Static Public Attributes

- npt **L** .NDArray
- npt **P** .NDArray

5.24.1 Detailed Description

Class storing some statistics for a timestep of time evolution.

No calculation is carried out Python-end, C++ does all the heavy lifting.

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

- IO.py

5.25 sim::Vec Class Reference

Data structure representing a general vector.

Public Member Functions

- **Vec** (const Vec &otherVec)
- **Vec** (const double(&coords)[mDim])
- double & **operator[ ]** (int index)
- double **operator[ ]** (int index) const
- Vec **operator+** () const
- Vec **operator-** () const
- Vec **operator+** (const Vec &otherVec) const
- Vec **operator-** (const Vec &otherVec) const
- Vec **operator**∗ (double factor) const
- Vec **operator**/ (double factor) const
- Vec & **operator=** (const Vec &otherVec)
- Vec & **operator+=** (const Vec &otherVec)
- Vec & **operator-=** (const Vec &otherVec)
- bool **operator==** (const Vec &otherVec) const

  Two vectors are equal when their components agree exactly.
- double **GetNorm** () const
- double GetTheta () const

  Calculate the spherical polar coordinate theta.
- double GetPhi () const

  Calculate the spherical polar coordinate phi.

Static Public Member Functions

- static Vec FromSpherical (double r, double theta, double phi)

  Convert spherical polar coordinates to a Cartesian vector.

Static Public Attributes

- static const int **mDim** = 3

Friends

- double **DotProduct** (const Vec &v1, const Vec &v2)
- Vec **CrossProduct** (const Vec &v1, const Vec &v2)
- std::ostream & **operator**<< (std::ostream &output, const Vec &otherVec)

5.25.1 Detailed Description

Data structure representing a general vector.

5.25.2 Member Function Documentation

5.25.2.1 FromSpherical()

```
Vec sim::Vec::FromSpherical (
            double r,
            double theta,
            double phi)  [static]
```

Convert spherical polar coordinates to a Cartesian vector.

Returns

Cartesian vector the spherical coordinates represent.

5.25.2.2 GetPhi()

```
double sim::Vec::GetPhi () const
```

Calculate the spherical polar coordinate phi.

This conforms to the boost::math::spherical_harmonic conventions, running from 0 to 2pi.

5.25.2.3 GetTheta()

```
double sim::Vec::GetTheta () const
```

Calculate the spherical polar coordinate theta.

This conforms to the boost::math::spherical_harmonic conventions, running from 0 to pi.

The documentation for this class was generated from the following files:

- Vec.hpp
- Vec.cpp

Chapter 6

File Documentation

6.1 Analysis.py File Reference

Python-side processing of C++ produced data.

Functions

- bool pysrc.Analysis.approx (float a, float b)

  Check whether two numbers are approximately equal.
- tuple[list, list] pysrc.Analysis.GetResStats (list paramList, dict res, str int_type)

  Get basic statistics for timing results.
- pysrc.Analysis.AnalyseParam (str fileName, str figDir, tp.Callable loopPlt, tp.Callable finalPlt)

  The master function for time complexity analysis.
- pysrc.Analysis.AnalyseN (str fileName, str figDir)

  Analyse time complexity dependence on number of particles.
- pysrc.Analysis.AnalyseP (str fileName, str figDir)

  Analyse time complexity as function of multipole expansion order.
- pysrc.Analysis.AnalyseTheta (str fileName, str figDir)

  Analyse time complexity as function of opening angle theta.
- list[float] pysrc.Analysis.GetErrors (str fn_brute, str fn_inter)

  Calculate relative errors in calculated acceleration.
- pysrc.Analysis.AnalyseParamError (str folderName, str figDir, str paramName, str xlabel, bool logScale=True)

  Analyse error distribution and change as a parameter is varied.
- pysrc.Analysis.VisualiseGrid (Grid grid, float scale, str title=None, fig=plt.figure())

  Plot particles in a grid on a 3D scatter plot.
- pysrc.Analysis.AnimateGrid (dict[float, Grid] grids, float scale, str figName, str int_type=None)

  Animate the time evolution of a system, saving to disk.
- pysrc.Analysis.GridSnapshots (dict[float, Grid] grids, float scale, str figDir, str int_type=None)

  Save snapshots of time evolution at interval timestamps.
- pysrc.Analysis.AnalyseEvo (str folderName, str figDir)

  For time evolution, generate videos and plots of conserved quantities.

Variables

- dict pysrc.Analysis.intMapping
- str **pysrc.Analysis.combScript** = "scripts/combine-vid.sh"

6.1.1 Detailed Description

Python-side processing of C++ produced data.

6.1.2 Function Documentation

6.1.2.1 AnalyseEvo()

```
pysrc.Analysis.AnalyseEvo (
            str folderName,
            str figDir)
```

For time evolution, generate videos and plots of conserved quantities.

Current plots include energy and components of angular momentum against time

Parameters

folderName	Folder from which to read time evolution results
figDir	Directory to save figures and videos to.

6.1.2.2 AnalyseN()

```
pysrc.Analysis.AnalyseN (
            str fileName,
            str figDir)
```

Analyse time complexity dependence on number of particles.

Parameters

fileName	Path to timing results file
figDir	Directory to save figure to

6.1.2.3 AnalyseP()

```
pysrc.Analysis.AnalyseP (
            str fileName,
            str figDir)
```

Analyse time complexity as function of multipole expansion order.

Parameters

fileName	Path to timing results file
figDir	Directory to save the plots to

6.1.2.4 AnalyseParam()

```
pysrc.Analysis.AnalyseParam (
            str fileName,
            str figDir,
            tp.Callable loopPlt,
            tp.Callable finalPlt)
```

The master function for time complexity analysis.

Parameters

fileName	Path to timing results file.
figDir	Directory to save figures to.
loopPlt	Plotting function to call once for each interaction type.
finalPlt	Plotting function finish up and save the plot.

6.1.2.5 AnalyseParamError()

```
pysrc.Analysis.AnalyseParamError (
            str folderName,
            str figDir,
            str paramName,
            str xlabel,
            bool  logScale = True)
```

Analyse error distribution and change as a parameter is varied.

Plots both one graph with mean errors of all non-brute interaction types compared, and one for each interaction containing change of mean, 5th and 95th percentile errors against parameter. Colours in the first graph is made consistent with timing plots.

If the folder contains fewer brute calculations than the other interactions, it is assumed brute is not affected by parameter varied and everything is compared against brute results of the smallest parameter (for same repeats)

If the recorded param is very close to an integer, it is converted to an integer for better plotting results.

Parameters

folderName	Folder containing dump files with one param varying
figDir	Directory to save figures (created if doesn't exist.)
paramName	Name of parameter varied (for setting text on plots)
xlabel	xlabel to use for plots
logScale	Whether to use log scale for x-axis of plots.

6.1.2.6 AnalyseTheta()

```
pysrc.Analysis.AnalyseTheta (
            str fileName,
            str figDir)
```

Analyse time complexity as function of opening angle theta.

Parameters

fileName	Path to timing results file
figDir	Directory to save the plots to

6.1.2.7 AnimateGrid()

```
pysrc.Analysis.AnimateGrid (
            dict[float, Grid] grids,
            float scale,
            str figName,
            str  int_type = None)
```

Animate the time evolution of a system, saving to disk.

Parameters

grids	Dictionary of system snapshots at each timestamp.
scale	Characteristic scale of system
figName	Name of the exported animation
int_type	Type of interaction for this run

6.1.2.8 approx()

```
bool pysrc.Analysis.approx (
            float a,
            float b)
```

Check whether two numbers are approximately equal.

Can be used to test if a number is very nearly an integer.

6.1.2.9 GetErrors()

```
list[float] pysrc.Analysis.GetErrors (
            str fn_brute,
            str fn_inter)
```

Calculate relative errors in calculated acceleration.

Parameters

fn_brute	Path to brute force file taken as "ground truth".
fn_inter	Path to other interaction file to be compared to brute force

Returns

List of errors (all three components, all particles, all repeats)

6.1.2.10 GetResStats()

```
tuple[list, list] pysrc.Analysis.GetResStats (
            list paramList,
            dict res,
            str int_type)
```

Get basic statistics for timing results.

Parameters

paramList	Order in which to access all param keys in timing results
res	Dictionary containing all timing results
int_type	Interaction type to consider

Returns

lists of average and stdev at each param, ordered as in paramList

6.1.2.11 GridSnapshots()

```
pysrc.Analysis.GridSnapshots (
            dict[float, Grid] grids,
            float scale,
            str figDir,
            str  int_type = None)
```

Save snapshots of time evolution at interval timestamps.

Parameters

grids	Grids at each timestamp of time evolution
scale	Characteristic scale of system
figDir	Directory to save figures to
int_type	Type of interaction.

6.1.2.12 VisualiseGrid()

```
pysrc.Analysis.VisualiseGrid (
            Grid grid,
            float scale,
            str  title = None,
             fig = plt.figure())
```

Plot particles in a grid on a 3D scatter plot.

Scale determines the limits of the plot. It is assumed that the system is centred at the origin.

This is currently used mainly for animations. To plot a single Grid using this function, one might have to do:

fig = VisualiseGrid(...) fig.show() input()

To block the main process.

Parameters

grid	Grid containing all particles of interest
scale	Scale to do the plot at (Some particles might lie outside)
title	Title of plot
fig	Figure object to operate on. Will create one if it's not supplied

Returns

Figure object with a 3D scatter plot

6.1.3 Variable Documentation

6.1.3.1 intMapping

dict pysrc.Analysis.intMapping

Initial value:

```
00001 =  {'fmm':'FMM',
00002                                 'brute':'Brute-force',
00003                                 'bh':'Barnes-Hut'
00004                                 }
```

6.2 Grid.py File Reference

Python side boiled-down implementation of the C++ Grid class.

Classes

- class pysrc.Grid.Grid

  Python side boiled-down implementation of the C++ Grid class.

6.2.1 Detailed Description

Python side boiled-down implementation of the C++ Grid class.

Represents a list of particles, and the dimensions of the simulation.

6.3 IO.py File Reference

IO Library for loading calculations done in C++.

Classes

- class pysrc.IO.Stats

  Class storing some statistics for a timestep of time evolution.

Functions

- **pysrc.IO.MakeDir** (str dir)
- float pysrc.IO.LoadFloat (str floatStr)

  Load a float from string, either in decimal or hexfloat format.
- **pysrc.IO.LoadOctant** (file)
- npt.NDArray **pysrc.IO.LoadVec** (file)
- Particle **pysrc.IO.LoadParticle** (file)
- Grid **pysrc.IO.LoadGrid** (file)
- list[Grid] pysrc.IO.LoadGrids (str fileName)

  Load a series of grids.
- dict pysrc.IO.LoadParamResults (str fileName)

  Load timing results for a certain parameter produced by C++.
- dict pysrc.IO.LoadDumpFolder (str folderName)

  Load and organise fileNames in the dump folder.
- tuple[str, list[Stats], dict[float, Grid], float] pysrc.IO.LoadEvo (str fileName)

  Load a file containing time evolution of a system.

6.3.1 Detailed Description

IO Library for loading calculations done in C++.

6.3.2 Function Documentation

6.3.2.1 LoadDumpFolder()

```
dict pysrc.IO.LoadDumpFolder (
             str folderName)
```

Load and organise fileNames in the dump folder.

Parameters

folderName	Path to folder containing dump files.

Returns

A nested dictionary. First key interaction type, second key param value (needs context for which parameter it is), value is file name.

6.3.2.2 LoadEvo()

```
tuple[str, list[Stats], dict[float, Grid], float] pysrc.IO.LoadEvo (
             str fileName)
```

Load a file containing time evolution of a system.

Returns

interaction type, a list of Stats, grid at each time, and scale.

6.3.2.3 LoadFloat()

```
float pysrc.IO.LoadFloat (
             str floatStr)
```

Load a float from string, either in decimal or hexfloat format.

Parameters

floatStr	String containing a float.

Returns

the float number.

6.3.2.4 LoadGrids()

```
list[Grid] pysrc.IO.LoadGrids (
             str fileName)
```

Load a series of grids.

Returns

A list of the series of grids.

6.3.2.5 LoadParamResults()

```
dict pysrc.IO.LoadParamResults (
             str fileName)
```

Load timing results for a certain parameter produced by C++.

The returned dictinoary is nested. Its outermost key is the parameter value, and the second is interaction type ('brute', 'bh' and 'fmm').

The value to the second key is a list of timing repeats, integers in microseconds.

Returns

A nested dictionary of the result.

6.4 Particle.py File Reference

Python side boiled-down representation of C++ Particle class.

Classes

- class pysrc.Particle.Particle

  Python side boiled-down representation of C++ Particle class.

6.4.1 Detailed Description

Python side boiled-down representation of C++ Particle class.

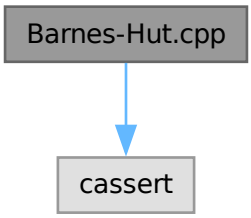
Represents a particle

6.5 Barnes-Hut.cpp File Reference

Implementation of Barnes-Hut method for force calculation.

```
#include <cassert>
#include "Barnes-Hut.hpp"
```

Include dependency graph for Barnes-Hut.cpp:



6.5.1 Detailed Description

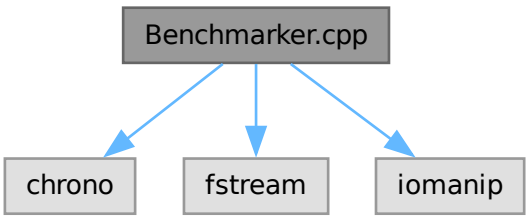
Implementation of Barnes-Hut method for force calculation.

6.6 Benchmarker.cpp File Reference

Implementation of Benchmarker for testing and benchmarking.

```
#include <chrono>
#include <fstream>
#include <iomanip>
#include "Benchmarker.hpp"
#include "Distribution.hpp"
#include "IO.hpp"
#include "Brute.hpp"
#include "Barnes-Hut.hpp"
#include "FMM.hpp"
```

Include dependency graph for Benchmarker.cpp:



6.6.1 Detailed Description

Implementation of Benchmarker for testing and benchmarking.

6.7 Brute.cpp File Reference

Implementation of brute-force method for force calculation.

```
#include "Brute.hpp"
```

6.7.1 Detailed Description

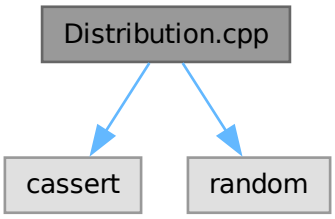
Implementation of brute-force method for force calculation.

6.8 Distribution.cpp File Reference

Generator of Particle distributions for use in experiments.

```
#include <cassert>
#include <random>
#include "Distribution.hpp"
```

Include dependency graph for Distribution.cpp:



Functions

- void **sim::dist::SetSeed** (int newSeed)

  Set random number seed.
- int **sim::dist::GetSeed** ()
- void **sim::dist::SetMassCutoff** (double newCutoff)
- double **sim::dist::GetMassCutoff** ()
- std::vector< Particle > sim::dist::MakeUniformMass (double mean, double spread, int n)

  Make a list of particles whose mass obeys uniform statistics.

- std::vector< Particle > sim::dist::MakeNormalMass (double mean, double sigma, int n)

  Make a list of particles whose mass obeys normal distribution.
- std::vector< Particle > & sim::dist::SetUniformPos (const Vec &posCentre, double(&spread)[Vec::mDim], std::vector< Particle > &list)

  Set positions of particles in a list by a uniform distribution.
- std::vector< Particle > & sim::dist::SetNormalPos (const Vec &posCentre, double(&sigma)[Vec::mDim], std::vector< Particle > &list)

  Set positions of particles in a list by a normal distribution.
- std::vector< Particle > & sim::dist::SetSphericalPos (const Vec &posCentre, double r0, double r1, std::vector< Particle > &list)

  Set position of particles by uniform distribution in a sphere.
- std::vector< Particle > & sim::dist::SetUniformRotVel (const Vec &posCentre, const Vec &omega, std::vector< Particle > &list)

  Set particles velocities by a uniform angular velocity about axis.
- std::vector< Particle > & sim::dist::SetDiskPos (const Vec &posCentre, const Vec &axis, double r0, double r1, double zSigma, std::vector< Particle > &list)

  Set particle positions by random distribution on a disk.
- std::vector< Particle > & sim::dist::SetCircVel (const Vec &centre, const Vec &axis, const double alpha, std::vector< Particle > &list)

  Set particles to velocities required for circular orbits.
- std::vector< Particle > & sim::dist::AddUniformVel (const Vec &boost, std::vector< Particle > &list)

  Boost all particles by velocity.

6.8.1 Detailed Description

Generator of Particle distributions for use in experiments.

6.8.2 Function Documentation

6.8.2.1 AddUniformVel()

```
std::vector< Particle > & sim::dist::AddUniformVel (
            const Vec & boost,
            std::vector< Particle > & list)
```

Boost all particles by velocity.

Parameters

	boost	Velocity to boost particles by
in,out	*list*	List of particles whose positions will be populated

Returns

Modified list of particles

6.8.2.2 MakeNormalMass()

```
std::vector< Particle > sim::dist::MakeNormalMass (
            double mean,
            double sigma,
            int n)
```

Make a list of particles whose mass obeys normal distribution.

Parameters

in	*mean*	Centre of normal distribution
in	*sigma*	Standard deviation
in	*n*	Number of particles to generate masses for

Returns

A list of particles, in which only mass (= charge) are set

6.8.2.3 MakeUniformMass()

```
std::vector< Particle > sim::dist::MakeUniformMass (
            double mean,
            double spread,
            int n)
```

Make a list of particles whose mass obeys uniform statistics.

Parameters

in	*mean*	Centre of uniform distribution
in	*spread*	mean +- spread gives distribution boudaries
in	*n*	Number of particles to generate masses for

Returns

A list of particles, in which only mass (= charge) are set

6.8.2.4 SetCircVel()

```
std::vector< Particle > & sim::dist::SetCircVel (
            const Vec & centre,
            const Vec & axis,
            const double alpha,
            std::vector< Particle > & list)
```

Set particles to velocities required for circular orbits.

This assumes an inverse square law, in which angular velocity at a distance r from the central massive body is $\omega = \sqrt{\alpha / r^3}$.

Parameters

in	*centre*	Centre of circular orbits
in	*axis*	Rotational axis
in	*alpha*	A strengh parameter. For gravity alpha = GM
in,out	*list*	List of particles whose positions will be populated

Returns

Modified list of particles

6.8.2.5 SetDiskPos()

```
std::vector< Particle > & sim::dist::SetDiskPos (
            const Vec & posCentre,
            const Vec & axis,
            double r0,
            double r1,
            double zSigma,
            std::vector< Particle > & list)
```

Set particle positions by random distribution on a disk.

Parameters

in	*posCenter*	Centre of the disk
in	*axis*	Axis perpendicular to plane of disk
in	*r0*	Minimum allowed radius
in	*r1*	Maximum allowed radius
in	*zSigma*	Standard deviation for height fluctuation along axis
in,out	*list*	List of particles whose positions will be populated

Returns

Modified list of particles

6.8.2.6 SetNormalPos()

```
std::vector< Particle > & sim::dist::SetNormalPos (
            const Vec & posCentre,
            double(&) sigma[Vec::mDim],
            std::vector< Particle > & list)
```

Set positions of particles in a list by a normal distribution.

Parameters

in	*posCentre*	Displacement of the centre of the distribution
in	*sigma*	Standard deviation by axis
in,out	*list*	List of particles whose positions will be populated

Returns

Modified list of particles

6.8.2.7 SetSphericalPos()

```
std::vector< Particle > & sim::dist::SetSphericalPos (
            const Vec & posCentre,
            double r0,
            double r1,
            std::vector< Particle > & list)
```

Set position of particles by uniform distribution in a sphere.

Parameters

in	*posCenter*	Centre of the sphere
in	*r0*	Minimum allowed radius for position
in	*r1*	Maximum allowed radius for position
in,out	*list*	List of particles whose positions will be populated

Returns

Modified list of particles

6.8.2.8 SetUniformPos()

```
std::vector< Particle > & sim::dist::SetUniformPos (
            const Vec & posCentre,
            double(&) spread[Vec::mDim],
            std::vector< Particle > & list)
```

Set positions of particles in a list by a uniform distribution.

Parameters

in	*posCentre*	Displacement of the centre of the distribution
in	*spread*	Spread by axis
in,out	*list*	List of particles whose positions will be populated

Returns

Modified list of particles

6.8.2.9 SetUniformRotVel()

```
std::vector< Particle > & sim::dist::SetUniformRotVel (
            const Vec & posCentre,
            const Vec & omega,
            std::vector< Particle > & list)
```

Set particles velocities by a uniform angular velocity about axis.

Parameters

in	*posCenter*	Centre of rotation
in	*omega*	Angular momentum vector
in,out	*list*	List of particles whose positions will be populated

Returns

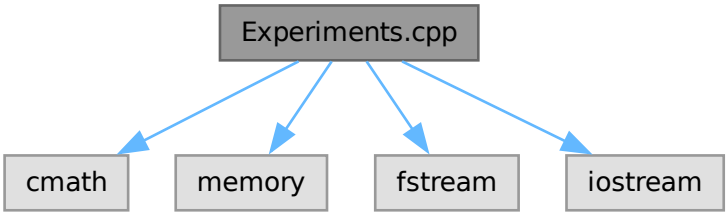
Modified list of particles

6.9 Experiments.cpp File Reference

Collection of functions for running different experiments on the code.

```
#include <cmath>
#include <memory>
#include <fstream>
#include <iostream>
#include "Experiments.hpp"
#include "Brute.hpp"
#include "Distribution.hpp"
#include "Benchmarker.hpp"
#include "IO.hpp"
```

Include dependency graph for Experiments.cpp:



Functions

- void **sim::exp::NComplexity** ()

  Measure complexity of three "Interaction"s against number of particles.
- void **sim::exp::PComplexity** ()

  Measure complexity of algorithms against order of multipole expansion.
- void **sim::exp::ThetaComplexity** ()

  Measure complexity of algorithms against opening angle.
- void **sim::exp::ColdStartSim** ()

  Generate and simulate with a cold start initial condition.
- void sim::exp::ThinDiskSim ()

  Generate and simulate with a thin disk initial condition.
- void sim::exp::GalaxySim ()

  Generate and simulate with a single galaxy initial condition.
- void sim::exp::TwoGalaxiesSim ()

  Generate and simulate with initial conditions with two galaxies.

6.9.1 Detailed Description

Collection of functions for running different experiments on the code.

6.9.2 Function Documentation

6.9.2.1 GalaxySim()

```
void sim::exp::GalaxySim ()
```

Generate and simulate with a single galaxy initial condition.

Thin disk with SMBH at its centre, so velocities are assumed to be just from circular orbits around the SMBH.

6.9.2.2 ThinDiskSim()

```
void sim::exp::ThinDiskSim ()
```

Generate and simulate with a thin disk initial condition.

In this condition, particles of similar masses are uniformly added into a disk, and set to rotate at the same angular velocity, approximated from a use of the Virial theorem.

This is not presented or used in the end as this condition is unstable; The system simply disintegrates, so there's little to see. An earlier attempt to simply model the "thin disk" as a "cylinder" and use angular velocity from that also failed tragically (of course that would...)

It is this that prompted me to use the SMBHs at the centre.

6.9.2.3 TwoGalaxiesSim()

```
void sim::exp::TwoGalaxiesSim ()
```

Generate and simulate with initial conditions with two galaxies.

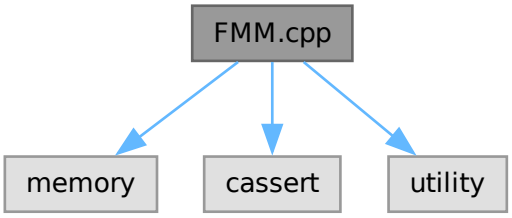
Each galaxy has a SMBH at the centre, and they are orbiting each other.

6.10 FMM.cpp File Reference

Implementation of Fast Multipole Method for force calculation.

```
#include <memory>
#include <cassert>
#include <utility>
#include "FMM.hpp"
```

Include dependency graph for FMM.cpp:



6.10.1 Detailed Description

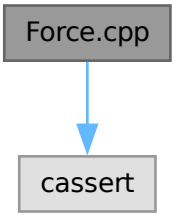
Implementation of Fast Multipole Method for force calculation.

6.11 Force.cpp File Reference

Virtual class for general pairwise force and its implementations.

```
#include <cassert>
#include "Force.hpp"
```

Include dependency graph for Force.cpp:



6.11.1 Detailed Description

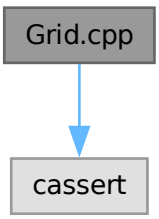
Virtual class for general pairwise force and its implementations.

6.12 Grid.cpp File Reference

Implement Grid to describe simulation space and particles within.

```
#include <cassert>
#include "Grid.hpp"
```

Include dependency graph for Grid.cpp:



6.12.1 Detailed Description

Implement Grid to describe simulation space and particles within.

6.13 Integrator.cpp File Reference

Define general integrator and implement Euler and leapfrog.

```
#include "Integrator.hpp"
#include "Vec.hpp"
```

6.13.1 Detailed Description

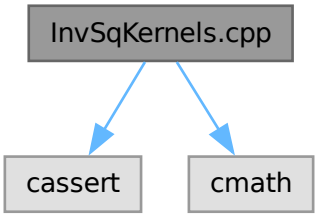
Define general integrator and implement Euler and leapfrog.

6.14 InvSqKernels.cpp File Reference

Implementation of the inverse square multipole expansion kernels.

```
#include <cassert>
#include <cmath>
#include "InvSqKernels.hpp"
```

Include dependency graph for InvSqKernels.cpp:



6.14.1 Detailed Description

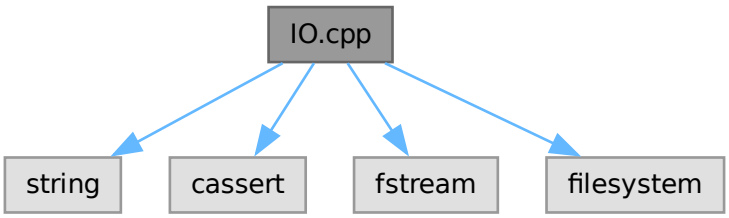
Implementation of the inverse square multipole expansion kernels.

6.15 IO.cpp File Reference

Library for manipulating data into / from text.

```
#include <string>
#include <cassert>
#include <fstream>
#include <filesystem>
#include "IO.hpp"
#include "Octant.hpp"
```

Include dependency graph for IO.cpp:



Functions

- void **sim::IO::Err** (const std::string &errMessage, std::ostream &stream)
- bool **sim::IO::FileExists** (const std::string &fileName)
- bool sim::IO::CheckFile (const std::string &fileName, bool expect)

  Check if file or folder exists.
- void **sim::IO::MakeDir** (const std::string &dirName)
- void sim::IO::SetHexfloatOut (std::ostream &stream)

  Set the ostream to output floating points in hexfloat format.
- Octant **sim::IO::LoadOctant** (std::istream &stream)
- Vec **sim::IO::LoadVec** (std::istream &stream)
- Particle **sim::IO::LoadParticle** (std::istream &stream)
- void **sim::IO::DumpOctant** (const Octant &octant, std::ostream &stream)
- void **sim::IO::DumpVec** (const Vec &vec, std::ostream &stream)
- void **sim::IO::DumpParticle** (const Particle &par, std::ostream &stream)
- void **sim::IO::DumpGrid** (const Grid &grid, std::ostream &stream)
- void **sim::IO::DumpGrid** (const Grid &grid, const std::string &fileName)
- Grid **sim::IO::LoadGrid** (std::istream &stream)
- Grid **sim::IO::LoadGrid** (const Grid &grid, const std::string &fileName)
- void **sim::IO::DumpOctree** (Octree const *root, std::ostream &stream)
- void **sim::IO::DumpOctree** (Octree const *octree, const std::string &fileName)
- std::unique_ptr< Octree > **sim::IO::LoadOctree** (const Grid &grid, std::istream &stream)
- std::unique_ptr< Octree > **sim::IO::LoadOctree** (const Grid &grid, const std::string &fileName)

6.15.1 Detailed Description

Library for manipulating data into / from text.

The function names and parameters should be self explanatory. When an std::istream is not supplied, std::cin is used by default. Similarly if std::ostream is absent, std::cout is used. Alternatively a path to a file can be supplied with the fileName parameter.

Insert remarks about why dumping classes as json files in python is the best human invention ever :)

6.15.2 Function Documentation

6.15.2.1 CheckFile()

```
bool sim::IO::CheckFile (
            const std::string & fileName,
            bool expect)
```

Check if file or folder exists.

If the existence state is different from the expect value, complain in log. The function that receives the unexpected state will likely just abort.

Parameters

in	*fileName*	Path to file or folder.
in	*expect*	Whether we expect the file or folder to be there or not

Returns

Whether the folder or file exists or not.

6.15.2.2 SetHexfloatOut()

```
void sim::IO::SetHexfloatOut (
            std::ostream & stream)
```

Set the ostream to output floating points in hexfloat format.

Note that for istream's, the use of std::hexfloat seems to be less straightforward. There appears to be a bug in g++:

https://gcc.gnu.org/bugzilla/show_bug.cgi?id=81122

However my attemps in clang++ has been equally unsucessful.

Parameters

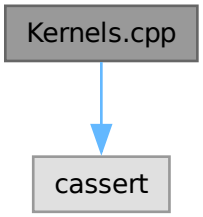
in,out	*stream*	Output stream to be acted on.

6.16 Kernels.cpp File Reference

Definition of the general Kernels interface.

```
#include <cassert>
#include "Kernels.hpp"
```

Include dependency graph for Kernels.cpp:



6.16.1 Detailed Description

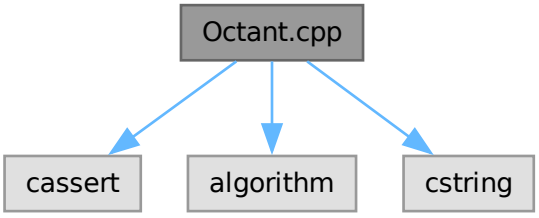
Definition of the general Kernels interface.

6.17 Octant.cpp File Reference

Data structure representing a cuboid box in space.

```
#include <cassert>
#include <algorithm>
#include <cstring>
#include "Octant.hpp"
```

Include dependency graph for Octant.cpp:



6.17.1 Detailed Description

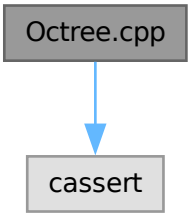
Data structure representing a cuboid box in space.

6.18 Octree.cpp File Reference

Octree implementation.

```
#include "Octree.hpp"
#include <cassert>
```

Include dependency graph for Octree.cpp:



6.18.1 Detailed Description

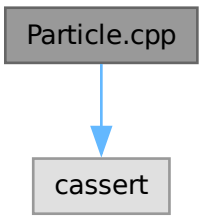
Octree implementation.

6.19 Particle.cpp File Reference

Data structure representing a particle.

```
#include <cassert>
#include "Particle.hpp"
```

Include dependency graph for Particle.cpp:



6.19.1 Detailed Description

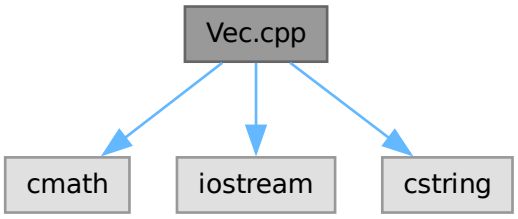
Data structure representing a particle.

6.20 Vec.cpp File Reference

Data structure representing a general vector.

```
#include <cmath>
#include <iostream>
#include <cstring>
#include "Vec.hpp"
```

Include dependency graph for Vec.cpp:



Functions

- double **sim::DotProduct** (const Vec &v1, const Vec &v2)
- Vec **sim::CrossProduct** (const Vec &v1, const Vec &v2)
- std::ostream & **sim::operator<<** (std::ostream &output, const Vec &v1)

6.20.1 Detailed Description

Data structure representing a general vector.

Index