LB2D

Generated by Doxygen 1.7.6.1

Mon Oct 21 2013 12:27:56

Contents

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Namespace Documentation

4.1 lb Namespace Reference

Classes

- struct coordinate

  Coordinate in 2D.
- struct node

  Node representing one lattice site.
- class lattice

  Lattice containing the populations.
- class property_array

  This class allows you to store properties in an array.
- class simulation

  Simulation class implementing LB.
- struct v9

  Lattice parameters for 9 velocity model.

Typedefs

- typedef float float_type
- typedef std::chrono::high_resolution_clock timer_type
- typedef std::chrono::duration < float, std::milli > milliseconds
- typedef timer_type::time_point time_point
- typedef timer_type::duration duration

Functions

- template<typename Node >
  float_type H_root (const Node &n)

Find over-relaxation parameter alpha.

- const v9 & velocity_set ()

  Get a reference single instance of the velocity set.

4.1.1 Detailed Description

Top level namespace

4.1.2 Typedef Documentation

4.1.2.1 typedef float lb::float_type

Floating point type (single or double precision)

Definition at line 19 of file global.hpp.

4.1.2.2 typedef std::chrono::high_resolution_clock lb::timer_type

High resolution clock type

Definition at line 50 of file global.hpp.

4.1.2.3 typedef std::chrono::duration<float, std::milli> lb::milliseconds

High resolution clock units type

Definition at line 52 of file global.hpp.

4.1.2.4 typedef timer_type::time_point lb::time_point

High resolution clock time point type

Definition at line 54 of file global.hpp.

4.1.2.5 typedef timer_type::duration lb::duration

High resolution clock time duration type

Definition at line 56 of file global.hpp.

4.1.3 Function Documentation

4.1.3.1 template<typename Node > float_type lb::H_root (const Node & *n*) [inline]

Find over-relaxation parameter alpha.

This function looks for a state with the same entropy/H-function value as the current state in the hyperplane connecting the current f with the f_eq.

This procedure involves finding the root of the entropy function: H(f + alpha∗(f_eq - f)) - H(f) = 0

You should also account for preventing the populations from becoming negative and for performance reasons not computing alpha for states close to the equilibrium.

For the root finding of the nonlinear equation use Newton-Raphson.

Template Parameters

Node	node type

Parameters

in	*n*	node object

Returns

alpha

Definition at line 36 of file H_root.hpp.

4.1.3.2 const v9 & lb::velocity_set ( ) [inline]

Get a reference single instance of the velocity set.

Returns

9-velocity set

Definition at line 132 of file velocity_set.hpp.

Referenced by lb::simulation::simulation().

Chapter 5

Class Documentation

5.1 lb::coordinate< T > Struct Template Reference

Coordinate in 2D.

`#include <global.hpp>`

Public Member Functions

- coordinate ()

  Default constructor.
- coordinate (T _i, T _j)

  Construct from x and y coordinates.

Public Attributes

- T i

  x coordinate
- T j

  y coordinate

Friends

- std::ostream & operator<< (std::ostream &os, const coordinate &c)

  Print to output stream.

5.1.1 Detailed Description

template<typename T>struct lb::coordinate< T >

Coordinate in 2D.

Template Parameters

Element	type

Definition at line 26 of file global.hpp.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 template<typename T> lb::coordinate< T >::coordinate (T _i, T _j) [inline]

Construct from x and y coordinates.

Parameters

in	*_i*	x coordinate
in	*_j*	y coordinate

Definition at line 36 of file global.hpp.

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

- global.hpp

5.2 lb::lattice Class Reference

Lattice containing the populations.

`#include <lattice.hpp>`

Public Types

- typedef std::vector< node > ::iterator node_iterator

  Iterator type.
- typedef std::vector< node > ::const_iterator const_node_iterator

  Const iterator type.
- typedef std::vector< node > ::reverse_iterator reverse_node_iterator

  Reverse iterator type.
- typedef std::vector< node > ::const_reverse_iterator const_reverse_node_iterator

  Const reverse iterator type.

Public Member Functions

- lattice (unsigned int _nx, unsigned int _ny)

  Construct the lattice with given extent.
- unsigned int index (int i, int j) const

  Convert a coordinate to a unique index.
- node_iterator begin ()

  Iterator pointing to the beginning.
- const_node_iterator begin () const

  Const iterator pointing to the beginning.
- node_iterator end ()

  Iterator pointing to the end.
- const_node_iterator end () const

  Const iterator pointing to the end.
- reverse_node_iterator rbegin ()

  Reverse iterator pointing to the end.
- const_reverse_node_iterator rbegin () const

  Const reverse iterator pointing to the end.
- reverse_node_iterator rend ()

  Reverse iterator pointing to the beginning.
- const_reverse_node_iterator rend () const

  Const reverse iterator pointing to the beginning.
- node & get_node (int i, int j)

  Get node at coordinate (i,j)
- const node & get_node (int i, int j) const

  Get node at coordinate (i,j)
- node & get_node (unsigned int idx)

  Get node at coordinate (i,j)
- const node & get_node (unsigned int idx) const

  Get node at coordinate (i,j)
- void add_wall (coordinate< int > min_coord, coordinate< int > max_coord)

  Add a solid wall.
- void delete_walls ()

  Delete all existing walls.
- void write_fields (std::string file_name)

  Write fields to file.

Public Attributes

- const unsigned int nx

  extent in x direction (excluding buffers)
- const unsigned int ny

  extent in y direction (excluding buffers)

- const unsigned int size

  total number of nodes (excluding buffers)
- const unsigned int buffer_size

  buffer width (equal to one)
- const unsigned int real_nx

  extent in x direction including buffers
- const unsigned int real_ny

  extent in y direction including buffers
- const unsigned int real_size

  total number of nodes including buffers
- const unsigned int n_populations

  number of populations
- std::vector< std::vector < float_type > > f

  population data
- std::vector< float_type > rho

  density data
- std::vector< float_type > u

  flow x-velocity data
- std::vector< float_type > v

  flow y-velocity data
- std::vector< node > nodes

  array holding all node objects
- std::vector< node > wall_nodes

  array holding node objects belonging to a solid wall
- property_array properties

  properties datastructure (can hold many different properties per node)
- const bool periodic_x

  flag whether to use periodicity in x direction
- const bool periodic_y

  flag whether to use periodicity in y direction

Friends

- std::ostream & operator<< (std::ostream &os, const lattice &l)

  print to output stream, useful for debugging only

5.2.1 Detailed Description

Lattice containing the populations.

The lattice is constructed using the function velocity_set() which returns a velocity set object. Hence, the number of populations is defined through that function. Data structures are set up accordingly.

The basic data structure for the population and the macroscopic qunatities are one dimensional arrays (vectors) interpreted as two dimensional planes. The x (i) dimension varies first and the y (j) dimension last.

This class does provide access to the data through node iterators or through direct access of the public members. The node iterators return a node object that provides easy access to all local quantities according to the 2d lattice coordinate.

There are buffer regions (extent is one in all directions) around the data to make the advection procedure easier.

The data is indexed in the range [0, nx-1][0, ny-1]; including buffers indices span the range [-1, nx][-1, ny], repectively.

Definition at line 209 of file lattice.hpp.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 lb::lattice::lattice (unsigned int *_nx,* unsigned int *_ny*)

Construct the lattice with given extent.

Parameters

in	*_nx*	Number of nodes in x direction
in	*_ny*	Number of nodes in y direction

Definition at line 396 of file lattice.hpp.

References buffer_size, index(), nodes, properties, real_nx, real_ny, lb::property_array-::register_flag_property(), and lb::property_array::set_flag_property().

5.2.3 Member Function Documentation

5.2.3.1 unsigned int lb::lattice::index (int *i,* int *j*) const [inline]

Convert a coordinate to a unique index.

Parameters

in	*i*	x coordinate
in	*j*	y coordinate

Returns

unique index

Precondition

Coordinates are in the domain

Definition at line 432 of file lattice.hpp.

References buffer_size, and real_nx.

Referenced by lattice().

5.2.3.2 lattice::node_iterator lb::lattice::begin ( )

Iterator pointing to the beginning.

Returns

iterator

Definition at line 423 of file lattice.hpp.

References nodes.

5.2.3.3 lattice::const_node_iterator lb::lattice::begin ( ) const

Const iterator pointing to the beginning.

Returns

const iterator

Definition at line 424 of file lattice.hpp.

References nodes.

5.2.3.4 lattice::node_iterator lb::lattice::end ( )

Iterator pointing to the end.

Returns

iterator

Definition at line 425 of file lattice.hpp.

References nodes.

5.2.3.5 lattice::const_node_iterator lb::lattice::end ( ) const

Const iterator pointing to the end.

Returns

const iterator

Definition at line 426 of file lattice.hpp.

References nodes.

5.2.3.6 lattice::reverse_node_iterator lb::lattice::rbegin ( )

Reverse iterator pointing to the end.

Returns

reverse iterator

Definition at line 427 of file lattice.hpp.

References nodes.

5.2.3.7 lattice::const_reverse_node_iterator lb::lattice::rbegin ( ) const

Const reverse iterator pointing to the end.

Returns

const reverse iterator

Definition at line 428 of file lattice.hpp.

References nodes.

5.2.3.8 lattice::reverse_node_iterator lb::lattice::rend ( )

Reverse iterator pointing to the beginning.

Returns

reverse iterator

Definition at line 429 of file lattice.hpp.

References nodes.

5.2.3.9 lattice::const_reverse_node_iterator lb::lattice::rend ( ) const

Const reverse iterator pointing to the beginning.

Returns

const reverse iterator

Definition at line 430 of file lattice.hpp.

References nodes.

5.2.3.10 node & lb::lattice::get_node (int *i*, int *j*) [inline]

Get node at coordinate (i,j)

Parameters

in	*i*	x coordinate
in	*j*	y coordinate

Returns

reference to node at coordinate (i,j)

Precondition

coordinates are in domain

Definition at line 434 of file lattice.hpp.

References buffer_size, nodes, and real_nx.

Referenced by add_wall(), and lb::simulation::initialize().

5.2.3.11 const node & lb::lattice::get_node (int *i*, int *j*) const [inline]

Get node at coordinate (i,j)

Parameters

in	*i*	x coordinate
in	*j*	y coordinate

Returns

const reference to node at coordinate (i,j)

Precondition

coordinates are in domain

Definition at line 436 of file lattice.hpp.

References buffer_size, nodes, and real_nx.

5.2.3.12 node & lb::lattice::get_node (unsigned int *idx*) [inline]

Get node at coordinate (i,j)

Parameters

in	*idx*	unique node index

Returns

reference to node at coordinate (i,j)

Precondition

idx is between [0, lattice::real_size)

Definition at line 438 of file lattice.hpp.

References nodes.

5.2.3.13 const node & lb::lattice::get_node (unsigned int *idx*) const [inline]

Get node at coordinate (i,j)

Parameters

in	*idx*	unique node index

Returns

const reference to node at coordinate (i,j)

Precondition

idx is between [0, lattice::real_size)

Definition at line 440 of file lattice.hpp.

References nodes.

5.2.3.14 void lb::lattice::add_wall (coordinate< int > *min_coord*, coordinate< int > *max_coord*)

Add a solid wall.

Creates wall flags in the coordinate rectangle defined by min_coord and max_coord. The corresponding nodes get the flag "wall" and they are also stored in the vector lattice-:wall_nodes for convienience.

Parameters

in	*min_coord*	minimum bounding rectangle corner
in	*max_coord*	maximum bounding rectangle corner

Precondition

(min_coord, max_coord) define a rectangle
Both min_coord and max_coord are in the domain

Definition at line 470 of file lattice.hpp.

References get_node(), lb::coordinate< T >::i, lb::coordinate< T >::j, lb::node::set_flag_property(), and wall_nodes.

5.2.3.15 void lb::lattice::write_fields (std::string *file_name*)

Write fields to file.

Write macroscopic variables to simple ascii file.

Parameters

in	*file_name*	file name

Definition at line 496 of file lattice.hpp.

References buffer_size, nodes, nx, ny, real_nx, rho, u, and v.

Referenced by lb::simulation::write_fields().

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

- lattice.hpp

5.3 lb::node Struct Reference

Node representing one lattice site.

`#include <lattice.hpp>`

Public Member Functions

- node ()
  Default constructor.
- node (lattice *lat, int i, int j)
  Construct from lattice and position.
- void set (lattice *lat, int i, int j)
  Set lattice and position.

- float_type f (unsigned int i) const

  Get population.
- float_type & f (unsigned int i)

  Get/set population.
- float_type rho () const

  Get density.
- float_type & rho ()

  Get/set density.
- float_type u () const

  Get x-velocity.
- float_type & u ()

  Get/set x-velocity.
- float_type v () const

  Get y-velocity.
- float_type & v ()

  Get/set y-velocity.
- bool has_flag_property (std::string name) const

  Query for flag property. Query whether a flag is set for the node.
- bool set_flag_property (std::string name)

  Set a flag Set the flag "name" to true.
- bool unset_flag_property (std::string name)

  Unset a flag Set the flag "name" to false.
- bool has_data_property (std::string name) const

  Query for data property. Query whether data property (object) is stored for the node.
- template<typename T >
  bool set_data_property (std::string name, const T &property)

  Store a data property.
- bool unset_data_property (std::string name)

  Delete a data property.
- template<typename T >
  T & get_data_property (std::string name)

  Get data property.
- template<typename T >
  const T & get_data_property (std::string name) const

  Get data property.

Public Attributes

- lattice * l

  Pointer to a lattice object.
- unsigned int index

  Index for looking up data in the lattice.
- coordinate< int > coord

  Coordinate of node's position.

5.3.1 Detailed Description

Node representing one lattice site.

Easy access to bundled quantities and properties (works as proxy to the lattice class).

Definition at line 26 of file lattice.hpp.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 lb::node::node (lattice * *lat,* int *i,* int *j*)

Construct from lattice and position.

Parameters

in	*lat*	Pointer to the lattice
in	*i*	x coordinate
in	*j*	y coordinate

Precondition

coordinates are in domain

Definition at line 360 of file lattice.hpp.

5.3.3 Member Function Documentation

5.3.3.1 void lb::node::set (lattice * *lat,* int *i,* int *j*)

Set lattice and position.

Parameters

in	*lat*	Pointer to lattice
in	*i*	x coordinate
in	*j*	y coordinate

Precondition

coordinates are in domain

Definition at line 363 of file lattice.hpp.

References lb::lattice::buffer_size, coord, lb::coordinate< T >::i, index, lb::coordinate< T >::j, l, and lb::lattice::real_nx.

5.3.3.2 float_type lb::node::f (unsigned int *i*) const [inline]

Get population.

Parameters

i	Population index

Returns

Value of distribution function

Precondition

population index exists

Definition at line 371 of file lattice.hpp.

References lb::lattice::f, index, and l.

5.3.3.3 float_type & lb::node::f (unsigned int *i*) [inline]

Get/set population.

Parameters

i	Population index

Returns

Reference to value of distribution function

Precondition

population index exists

Definition at line 372 of file lattice.hpp.

References lb::lattice::f, index, and l.

5.3.3.4 float_type lb::node::rho ( ) const [inline]

Get density.

Returns

Local density

Definition at line 373 of file lattice.hpp.

References index, l, and lb::lattice::rho.

Referenced by lb::simulation::initialize().

5.3.3.5 float_type & lb::node::rho ( ) [inline]

Get/set density.

Returns

Reference to local density

Definition at line 374 of file lattice.hpp.

References index, l, and lb::lattice::rho.

5.3.3.6 float_type lb::node::u ( ) const [inline]

Get x-velocity.

Returns

Local flow velocity in x direction

Definition at line 375 of file lattice.hpp.

References index, l, and lb::lattice::u.

Referenced by lb::simulation::initialize().

5.3.3.7 float_type & lb::node::u ( ) [inline]

Get/set x-velocity.

Returns

Reference to local flow velocity in x direction

Definition at line 376 of file lattice.hpp.

References index, l, and lb::lattice::u.

5.3.3.8 float_type lb::node::v ( ) const [inline]

Get y-velocity.

Returns

Local flow velocity in y direction

Definition at line 377 of file lattice.hpp.

References index, l, and lb::lattice::v.

Referenced by lb::simulation::initialize().

5.3.3.9 float_type & lb::node::v ( ) [inline]

Get/set y-velocity.

Returns

Reference to local flow velocity in y direction

Definition at line 378 of file lattice.hpp.

References index, l, and lb::lattice::v.

5.3.3.10 bool lb::node::has_flag_property (std::string *name*) const [inline]

Query for flag property. Query whether a flag is set for the node.

Parameters

in	*name*	Flag name

Returns

True if flag is set, otherwise false

Definition at line 380 of file lattice.hpp.

References lb::property_array::has_flag_property(), index, l, and lb::lattice::properties.

5.3.3.11 bool lb::node::set_flag_property (std::string *name*) [inline]

Set a flag Set the flag "name" to true.

Parameters

in	*name*	Flag name

Returns

True if flag exists, otherwise false

Definition at line 381 of file lattice.hpp.

References index, l, lb::lattice::properties, and lb::property_array::set_flag_property().

Referenced by lb::lattice::add_wall().

5.3.3.12 bool lb::node::unset_flag_property (std::string *name*) [inline]

Unset a flag Set the flag "name" to false.

Parameters

in	*name*	Flag name

Returns

True if flag exists, otherwise false

Definition at line 382 of file lattice.hpp.

References index, l, lb::lattice::properties, and lb::property_array::unset_flag_property().

5.3.3.13 bool lb::node::has_data_property (std::string *name*) const [inline]

Query for data property. Query whether data property (object) is stored for the node.

Parameters

in	*name*	Data property name

Returns

True if thre is such a data property, otherwise false

Definition at line 383 of file lattice.hpp.

References lb::property_array::has_data_property(), index, l, and lb::lattice::properties.

5.3.3.14 template<typename T > bool lb::node::set_data_property (std::string *name,* const T & *property*) [inline]

Store a data property.

Template Parameters

T	Type of the data property

Parameters

in	*name*	Data property name
in	*property*	Data property object

Returns

True if data property exists, otherwise false

Definition at line 385 of file lattice.hpp.

References index, l, lb::lattice::properties, and lb::property_array::set_data_property().

5.3.3.15 bool lb::node::unset_data_property (std::string *name*)

Delete a data property.

Parameters

in	*name*	Data property name

Returns

True if data property exists, otherwise false

Definition at line 386 of file lattice.hpp.

References index, l, lb::lattice::properties, and lb::property_array::unset_data_property().

5.3.3.16 template<typename T > T & lb::node::get_data_property (std::string *name*)

Get data property.

Template Parameters

T	Type of the data property

Parameters

in	*name*	Data property name

Returns

Reference to data property object

Definition at line 388 of file lattice.hpp.

References lb::property_array::get_data_property(), index, l, and lb::lattice::properties.

5.3.3.17 template<typename T > const T & lb::node::get_data_property (std::string *name*) const

Get data property.

Template Parameters

T	Type of the data property

Parameters

in	*name*	Data property name

Returns

Reference to data property object

Definition at line 390 of file lattice.hpp.

References lb::property_array::get_data_property(), index, l, and lb::lattice::properties.

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

- lattice.hpp

5.4 lb::property_array Class Reference

This class allows you to store properties in an array.

`#include <property_array.hpp>`

Public Types

- typedef unsigned long int size_type
  enumeration and size type

Public Member Functions

- property_array (size_type size)
  Construct with array size.
- property_array (const property_array &other)
  Copy construct.
- void swap (property_array &other)
  Swap internal state.
- property_array & operator= (property_array other)
  Assignement.

- ~property_array ()
- bool register_flag_property (std::string name, bool set=false)

  register a new flag property
- template<typename T >
  bool register_data_property (std::string name, bool set=false, const T &value=-T())

  register a new data property
- bool set_flag_property (std::string name, size_type array_index)

  set flag property
- template<typename T >
  bool set_data_property (std::string name, size_type array_index, const T &property)

  set data property
- bool set_flag_property (size_type property_index, size_type array_index)

  set flag property
- template<typename T >
  bool set_data_property (size_type property_index, size_type array_index, const T &property)

  set data property
- bool unset_flag_property (std::string name, size_type array_index)

  unset flag property
- bool unset_data_property (std::string name, size_type array_index)

  unset data property
- bool unset_flag_property (size_type property_index, size_type array_index)

  unset flag property
- bool unset_data_property (size_type property_index, size_type array_index)

  unset data property
- template<typename T >
  T & get_data_property (std::string name, size_type array_index)

  access data property object
- template<typename T >
  const T & get_data_property (std::string name, size_type array_index) const

  access data property object
- template<typename T >
  T & get_data_property (size_type property_index, size_type array_index)

  access data property object
- template<typename T >
  const T & get_data_property (size_type property_index, size_type array_index) const

  access data property object
- bool flag_property_index (std::string name, size_type &property_index) const

  get flag property index from flag property name
- bool data_property_index (std::string name, size_type &property_index) const

  get data property index from data property name
- bool flag_property_name (size_type property_index, std::string &name) const

get flag property name from flag property index

- bool data_property_name (size_type property_index, std::string &name) const

  get data property name from data property index

- size_type size () const

  property array size

- size_type num_flag_properties () const

  number of registered flag properties

- size_type num_data_properties () const

  number of registered data properties

- size_type num_properties () const

  number of registered flag and data properties

- bool has_flag_property (std::string name, size_type array_index) const

  Check whether flag property is set.

- bool has_data_property (std::string name, size_type array_index) const

  Check whether data property is set.

- bool has_flag_property (size_type property_index, size_type array_index) const

  Check whether flag property is set.

- bool has_data_property (size_type property_index, size_type array_index) const

  Check whether data property is set.

- bool count_set_flag_properties (std::string name, size_type &count) const

  count flag properties which are set

- bool count_set_data_properties (std::string name, size_type &count) const

  count data properties which are set

- bool count_set_flag_properties (size_type property_index, size_type &count) const

  count flag properties which are set

- bool count_set_data_properties (size_type property_index, size_type &count) const

  count data properties which are set

- bool exist_data_property (std::string name) const

  check whether data property exists

- bool exist_flag_property (std::string name) const

  check whether flag property exists

Friends

- std::ostream & operator<< (std::ostream &os, const property_array &pa)

  print to output stream

5.4.1 Detailed Description

This class allows you to store properties in an array.

For every index in the property array you can register several flag properties (boolean values) and several custom objects of any type. As long as you do not store a data property for a given index no space will be used.

Example usage for flag property:

```
 property_array pa(5);                                          //
property array of size 5
 pa.register_register_flag_property("test_flag",false);         //
register a flag with name "test_flag", default value false

 pa.set_flag_property("test_flag",2);                           // set
the flag at position 2 to true
 // alternatively you can use the property index
 property_array::size_type idx;
 pa.flag_property_index("test_flag", idx);
 pa.set_flag_property(idx,2);                                   // set
the flag at position 2 to true

 bool value = pa.has_flag_property("test_flag",2);              // get
the value at position 2 for your flag
 // using the index
 value = pa.has_flag_property(idx,2);                           // get
the value at position 2 for your flag

 pa.unset_flag_property("test_flag",2);                         // set
flag at position 2 to false
 // using the index
 pa.unset_flag_property(idx,2);                                 // set
flag at position 2 to false
```

Example usage for data property:

```
 property_array pa(5);
       // property array of size 5
 // assume you want to store a std::vector<float>
 pa.register_register_data_property<std::vector<float> >("some_data",
false);     // register a data property with name "some_data", default value: null

 std::vector<float> test_property(9,1.0);
 pa.set_data_property("some_data", 3, test_property);
       // store data property "some_data" at position 3 to be equal to
variable test_property
 // again you could use an index instead of a string to access the
property
 property_array::size_type idx;
 pa.data_property_index("some_data", idx);
 pa.set_data_property("some_data", 3, test_property);


 if (pa.has_data_property("some_data", 3))
       // check whether there is a data property stored at index 3
 {
        std::vector<float> return_value;
        return_value=pa.get_data_property<std::vector<float> >("
some_data",3);  // retrieve the data property
 }
```

```
pa.unset_data_property("some_data", 3);
       // delete the data property at index 3
```

Definition at line 158 of file property_array.hpp.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 lb::property_array::property_array (size_type *size*) [inline]

Construct with array size.

Parameters

in	*size*	array size

Definition at line 191 of file property_array.hpp.

5.4.2.2 lb::property_array::~property_array ( ) [inline]

Destruct

Definition at line 231 of file property_array.hpp.

References num_data_properties().

5.4.3 Member Function Documentation

5.4.3.1 bool lb::property_array::register_flag_property (std::string *name*, bool *set* = false) [inline]

register a new flag property

Parameters

in	*name*	flag property name
in	*set*	default state

Returns

true if property does not exist yet

Definition at line 251 of file property_array.hpp.

Referenced by lb::lattice::lattice().

5.4.3.2 template<typename T > bool lb::property_array::register_data_property (std::string *name*, bool *set* = false, const T & *value* = T()) [inline]

register a new data property

Template Parameters

T	type for data property

Parameters

in	*name*	data property name
in	*set*	default behaviour: initialized or null
in	*value*	value for default initialization

Returns

true if property does not exist yet

Definition at line 269 of file property_array.hpp.

5.4.3.3 bool lb::property_array::set_flag_property (std::string *name*, size_type *array_index*) [inline]

set flag property

Parameters

in	*name*	flag property name
in	*array_index*	position in array

Returns

true if flag property exists

Precondition

array_index is in range [0, property_array::size())

Definition at line 288 of file property_array.hpp.

Referenced by lb::lattice::lattice(), and lb::node::set_flag_property().

5.4.3.4 template<typename T > bool lb::property_array::set_data_property (std::string *name*, size_type *array_index*, const T & *property*) [inline]

set data property

Template Parameters

T	type for data property

Parameters

in	*name*	data property name
in	*array_index*	position in array
in	*property*	data to store

Returns

true if data property exists

Precondition

array_index is in range [0, property_array::size())

Definition at line 305 of file property_array.hpp.

Referenced by lb::node::set_data_property().

5.4.3.5 bool lb::property_array::set_flag_property (size_type *property_index*, size_type *array_index*) [inline]

set flag property

Parameters

in	*property_index*	flag property index
in	*array_index*	position in array

Returns

true if flag property exists

Precondition

array_index is in range [0, property_array::size())

Definition at line 327 of file property_array.hpp.

References num_flag_properties().

5.4.3.6 template<typename T > bool lb::property_array::set_data_property (size_type *property_index*, size_type *array_index*, const T & *property*) [inline]

set data property

Template Parameters

T	type for data property

Parameters

in	*property_index*	data property index
in	*array_index*	position in array
in	*property*	data to store

Returns

true if data property exists

Precondition

array_index is in range [0, property_array::size())

Definition at line 343 of file property_array.hpp.

References num_data_properties().

5.4.3.7 bool lb::property_array::unset_flag_property (std::string *name*, size_type *array_index*) [inline]

unset flag property

Parameters

in	*name*	flag property name
in	*array_index*	position in array

Returns

true if flag property exists

Precondition

array_index is in range [0, property_array::size())

Definition at line 364 of file property_array.hpp.

Referenced by lb::node::unset_flag_property().

5.4.3.8 bool lb::property_array::unset_data_property (std::string *name*, size_type *array_index*) [inline]

unset data property

Parameters

in	*name*	data property name
in	*array_index*	position in array

Returns

true if data property exists

Precondition

array_index is in range [0, property_array::size())

Definition at line 378 of file property_array.hpp.

Referenced by lb::node::unset_data_property().

5.4.3.9 bool lb::property_array::unset_flag_property (size_type *property_index*, size_type *array_index*) [inline]

unset flag property

Parameters

in	*property_index*	flag property index
in	*array_index*	position in array

Returns

true if flag property exists

Precondition

array_index is in range [0, property_array::size())

Definition at line 400 of file property_array.hpp.

References num_flag_properties().

5.4.3.10 bool lb::property_array::unset_data_property (size_type *property_index*, size_type *array_index*) [inline]

unset data property

Parameters

in	*property_index*	data property index
in	*array_index*	position in array

Returns

true if data property exists

Precondition

array_index is in range [0, property_array::size())

Definition at line 413 of file property_array.hpp.

References num_data_properties().

5.4.3.11 template<typename T > T& lb::property_array::get_data_property (std::string *name,* size_type *array_index*) [inline]

access data property object

Template Parameters

T	data property type

Parameters

in	*name*	data property name
in	*array_index*	position in array

Returns

reference to data property object

Precondition

array_index is in range [0, property_array::size())

Definition at line 438 of file property_array.hpp.

Referenced by lb::node::get_data_property().

5.4.3.12 template<typename T > const T& lb::property_array::get_data_property (std::string *name,* size_type *array_index*) const [inline]

access data property object

Template Parameters

T	data property type

Parameters

in	*name*	data property name
in	*array_index*	position in array

Returns

const reference to data property object

Precondition

array_index is in range [0, property_array::size())

Definition at line 459 of file property_array.hpp.

5.4.3.13 template<typename T > T& lb::property_array::get_data_property (size_type *property_index*, size_type *array_index*) [inline]

access data property object

Template Parameters

T	data property type

Parameters

in	*property_index*	data property index
in	*array_index*	position in array

Returns

reference to data property object

Precondition

array_index is in range [0, property_array::size())

Definition at line 480 of file property_array.hpp.

References num_data_properties().

5.4.3.14 template<typename T > const T& lb::property_array::get_data_property (size_type *property_index*, size_type *array_index*) const [inline]

access data property object

Template Parameters

T	data property type

Parameters

in	*property_index*	data property index
in	*array_index*	position in array

Returns

const reference to data property object

Precondition

array_index is in range [0, property_array::size())

Definition at line 501 of file property_array.hpp.

References num_data_properties().

5.4.3.15 bool lb::property_array::flag_property_index (std::string *name*, size_type & *property_index*) const [inline]

get flag property index from flag property name

Parameters

in	*name*	flag property name
out	*property_index*	flag property index

Returns

true if property exists

Definition at line 521 of file property_array.hpp.

5.4.3.16 bool lb::property_array::data_property_index (std::string *name*, size_type & *property_index*) const [inline]

get data property index from data property name

Parameters

in	*name*	data property name
out	*property_index*	data property index

Returns

true if property exists

Definition at line 534 of file property_array.hpp.

5.4.3.17 bool lb::property_array::flag_property_name (size_type *property_index*, std::string & *name*) const [inline]

get flag property name from flag property index

Parameters

in	*property_index*	flag property index
out	*name*	flag property name

Returns

true if property exists

Definition at line 547 of file property_array.hpp.

References num_flag_properties().

5.4.3.18 bool lb::property_array::data_property_name (size_type *property_index*, std::string & *name*) const [inline]

get data property name from data property index

Parameters

in	*property_index*	data property index
out	*name*	data property name

Returns

true if property exists

Definition at line 559 of file property_array.hpp.

References num_data_properties().

5.4.3.19 bool lb::property_array::has_flag_property (std::string *name*, size_type *array_index*) const [inline]

Check whether flag property is set.

Parameters

in	*name*	flag property name
in	*array_index*	position in array

Returns

true if property is set

Precondition

array_index is in range [0, property_array::size())

Definition at line 586 of file property_array.hpp.

Referenced by lb::node::has_flag_property().

5.4.3.20 bool lb::property_array::has_data_property (std::string *name*, size_type *array_index*) const [inline]

Check whether data property is set.

Parameters

in	*name*	data property name
in	*array_index*	position in array

Returns

true if property is set

Precondition

array_index is in range [0, property_array::size())

Definition at line 600 of file property_array.hpp.

Referenced by lb::node::has_data_property().

5.4.3.21 bool lb::property_array::has_flag_property (size_type *property_index*, size_type *array_index*) const [inline]

Check whether flag property is set.

Parameters

in	*property_index*	flag property index
in	*array_index*	position in array

Returns

true if property is set

Precondition

array_index is in range [0, property_array::size())

Definition at line 614 of file property_array.hpp.

References num_flag_properties().

5.4.3.22 bool lb::property_array::has_data_property (size_type *property_index*, size_type *array_index*) const [inline]

Check whether data property is set.

Parameters

in	*property_index*	data property index
in	*array_index*	position in array

Returns

true if property is set

Precondition

array_index is in range [0, property_array::size())

Definition at line 627 of file property_array.hpp.

References num_data_properties().

5.4.3.23 bool lb::property_array::count_set_flag_properties (std::string *name*, size_type & *count*) const [inline]

count flag properties which are set

Parameters

in	*name*	flag property index
out	*count*	number of flag properties set

Returns

true if flag property exists

Definition at line 639 of file property_array.hpp.

5.4.3.24 bool lb::property_array::count_set_data_properties (std::string *name*, size_type & *count*) const [inline]

count data properties which are set

Parameters

in	*name*	data property name
out	*count*	number of data properties set

Returns

true if data property exists

Definition at line 657 of file property_array.hpp.

5.4.3.25 bool lb::property_array::count_set_flag_properties (size_type *property_index*, size_type & *count*) const [inline]

count flag properties which are set

Parameters

in	*property_index*	flag property index
out	*count*	number of flag properties set

Returns

true if flag property exists

Definition at line 675 of file property_array.hpp.

References num_flag_properties().

5.4.3.26 bool lb::property_array::count_set_data_properties (size_type *property_index*, size_type & *count*) const [inline]

count data properties which are set

Parameters

in	*property_index*	data property index
out	*count*	number of data properties set

Returns

true if data property exists

Definition at line 692 of file property_array.hpp.

References num_data_properties().

5.4.3.27 bool lb::property_array::exist_data_property (std::string *name*) const [inline]

check whether data property exists

Parameters

in	*name*	data property name

Returns

true if exists

Definition at line 708 of file property_array.hpp.

5.4.3.28 bool lb::property_array::exist_flag_property (std::string *name*) const [inline]

check whether flag property exists

Parameters

in	*name*	flag property name

Returns

true if exists

Definition at line 719 of file property_array.hpp.

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

- property_array.hpp

5.5 lb::simulation Class Reference

Simulation class implementing LB.

`#include <simulation.hpp>`

Public Member Functions

- simulation (unsigned int nx, unsigned int ny, float_type _Re, float_type _Vmax)
  Construct from domain size and flow parameters.
- void initialize ()
  Initialize the flow field.
- void advect ()
  advect the populations
- void wall_bc ()
  apply wall boundary conditions
- void collide ()
  collide the populations
- void step ()
  LB step.
- void write_fields ()

Public Attributes

- lattice l
  lattice
- std::vector< int > shift
  amount of nodes to shift each population in data structure during advection
- const float_type Re
  Reynolds number.
- const float_type Vmax
  mean flow velocity
- const float_type visc
  viscosity
- const float_type beta
  LB parameter beta.
- unsigned int time
  simulation time
- bool file_output
  flag whether to write files
- unsigned int output_freq
  file output frequency
- unsigned int output_index
  index for file naming

Friends

- std::ostream & operator<< (std::ostream &os, const simulation &sim)

5.5.1 Detailed Description

Simulation class implementing LB.

This class holds a lattice as member (see simulation::l) and carries out the simulation steps on top of it. The main methods of this class are simulation::advect() and simulation::collide().

Definition at line 24 of file simulation.hpp.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 lb::simulation::simulation (unsigned int *nx*, unsigned int *ny*, float_type *_Re*, float_type *_Vmax*) [inline]

Construct from domain size and flow parameters.

Parameters

in	*nx*	extent in x direction
in	*ny*	extent in y direction
in	*_Re*	Reynolds number
in	*_Vmax*	mean flow velocity

Definition at line 35 of file simulation.hpp.

References shift, lb::v9::size, and lb::velocity_set().

5.5.3 Member Function Documentation

5.5.3.1 void lb::simulation::initialize ( ) [inline]

Initialize the flow field.

Initialization includes defining initial density, velocity and populations. You can use - Taylor-Green vortex flow conditions.

Definition at line 63 of file simulation.hpp.

References lb::lattice::get_node(), l, lb::lattice::nx, lb::lattice::ny, lb::node::rho(), lb::node::u(), and lb::node::v().

5.5.3.2 void lb::simulation::advect ( ) [inline]

advect the populations

Include periodic boundary conditions here also

Definition at line 91 of file simulation.hpp.

Referenced by step().

5.5.3.3 void lb::simulation::write_fields ( ) [inline]

write macroscopic variables to ascii file

Definition at line 140 of file simulation.hpp.

References l, output_index, and lb::lattice::write_fields().

Referenced by step().

5.5.4 Friends And Related Function Documentation

5.5.4.1 std::ostream& operator<< (std::ostream & *os,* const simulation & *sim*) [friend]

print to output stream

Definition at line 150 of file simulation.hpp.

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

- simulation.hpp

5.6 lb::v9 Struct Reference

Lattice parameters for 9 velocity model.

`#include <velocity_set.hpp>`

Public Member Functions

- void f_eq (float_type *f_eq, float_type rho, float_type u, float_type v) const

  Compute equilibrium.

- template<typename Node >
  void equilibrate (Node &n, float_type rho, float_type u, float_type v) const

  Equilibrate a node.

- template<typename Node >
  void equilibrate (Node &n) const

  Equilibrate a node.

Public Attributes

- const std::array< float_type, 9 > W = {{ 16.0/36, 4.0/36, 4.0/36, 4.0/36, 4.0/36, 1.0/36, 1.0/36, 1.0/36, 1.0/36}}

  Lattice weights.
- const std::array< std::array < int, 9 >, 2 > c

  Molecular velocities.
- const float_type cs = 1.0/std::sqrt(3.0)

  Speed of sound.
- const unsigned int size = 9

  Number of velocities.

Friends

- const v9 & lb::velocity_set ()

  Function for instantiating the singleton is a friend.

5.6.1 Detailed Description

Lattice parameters for 9 velocity model.

This class models a the singleton design pattern. That means there exists only one single instance throughout the lifetime of the program. To instantiate and access this object use the free function velocity_set.

This class holds parameters like lattice weights, molecular velocities and speed of sound. It also exposes member functions to compute the equilibrium populations.

Definition at line 31 of file velocity_set.hpp.

5.6.2 Member Function Documentation

5.6.2.1 void lb::v9::f_eq (float_type * *f_eq*, float_type *rho*, float_type *u*, float_type *v*) const [inline]

Compute equilibrium.

Compute f_eq from the locally conserved quantities rho, u and v (see also v9-::equilibrate).

Parameters

in,out	*f_eq*	Pointer to an array of size 9 to store the computed values
in	*rho*	Local density
in	*u*	Local flow velocity in x-direction
in	*v*	Local flow velocity in y-direction

Definition at line 66 of file velocity_set.hpp.

References W.

5.6.2.2 template<typename Node > void lb::v9::equilibrate (Node & *n*, float_type *rho*, float_type *u*, float_type *v*) const [inline]

Equilibrate a node.

Compute f_eq from the locally conserved quantities rho, u and v and set the node's population to that equilibrium (see also v9::f_eq).

Template Parameters

Node	A node type

Parameters

in,out	*n*	Reference to a Node object
in	*rho*	Local density
in	*u*	Local flow velocity in x-direction
in	*v*	Local flow velocity in y-direction

Definition at line 95 of file velocity_set.hpp.

References W.

Referenced by equilibrate().

5.6.2.3 template<typename Node > void lb::v9::equilibrate (Node & *n*) const [inline]

Equilibrate a node.

Compute f_eq from the locally conserved quantities rho, u and v and set the node's population to that equilibrium (see also v9::f_eq and v9::equilibrate). The locally conserved quantities are taken form the node object itself.

Template Parameters

Node	A node type

Parameters

in,out	*n*	Reference to a Node object

Definition at line 122 of file velocity_set.hpp.

References equilibrate().

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

- velocity_set.hpp

Chapter 6

File Documentation

6.1 global.hpp File Reference

global typdefs etc

`#include <iostream> #include <iomanip> #include <chrono>` ×

Classes

- struct lb::coordinate< T >

  Coordinate in 2D.

Namespaces

- namespace lb

Typedefs

- typedef float lb::float_type
- typedef std::chrono::high_resolution_clock lb::timer_type
- typedef std::chrono::duration < float, std::milli > lb::milliseconds
- typedef timer_type::time_point lb::time_point
- typedef timer_type::duration lb::duration

6.1.1 Detailed Description

global typdefs etc

Author

Fabian Bösch

Definition in file global.hpp.

6.2 H_root.hpp File Reference

compute root of H function

`#include "velocity_set.hpp" #include <algorithm>`

Namespaces

- namespace lb

Functions

- template<typename Node >
  float_type lb::H_root (const Node &n)
  Find over-relaxation parameter alpha.

6.2.1 Detailed Description

compute root of H function

Author

Fabian Bösch

Definition in file H_root.hpp.

6.3 lattice.hpp File Reference

lattice and node

`#include "velocity_set.hpp" #include "property_array.hpp" #include <vector> #include <fstream>`

Classes

- struct lb::node
  Node representing one lattice site.
- class lb::lattice
  Lattice containing the populations.

Namespaces

- namespace lb

6.3.1 Detailed Description

lattice and node

Author

Fabian Bösch

Definition in file lattice.hpp.

6.4 property_array.hpp File Reference

property array

```
#include <string> #include <vector> #include <typeinfo>
#include <typeindex> #include <algorithm> #include <iostream> ×
#include <iomanip> #include <stdexcept>
```

Classes

- class lb::property_array

  This class allows you to store properties in an array.

Namespaces

- namespace lb

6.4.1 Detailed Description

property array

Author

Fabian Bösch

Definition in file property_array.hpp.

6.5 simulation.hpp File Reference

simulation

```
#include "H_root.hpp"    #include "lattice.hpp"    #include
<sstream>
```

Classes

- class lb::simulation
  Simulation class implementing LB.

Namespaces

- namespace lb

6.5.1 Detailed Description

simulation

Author

Fabian Bösch

Definition in file simulation.hpp.

6.6 velocity_set.hpp File Reference

velocity set

```
#include "global.hpp" #include <array> #include <cmath>
```

Classes

- struct lb::v9
  Lattice parameters for 9 velocity model.

Namespaces

- namespace lb

Functions

- const v9 & lb::velocity_set ()
  Get a reference single instance of the velocity set.

6.6.1 Detailed Description

velocity set

Author

Fabian Bösch

Definition in file velocity_set.hpp.