SWE

Generated by Doxygen 1.8.6

Sun Nov 9 2014 11:43:21

Contents

Chapter 1

SWE

The Shallow Water Equations teaching code.

Documentation

The documentation is available in the `Wiki`

License

SWE is release unter GPLv3 (see `gpl.txt`)

Chapter 2

SWE/src/examples

Contains example programs that use SWE.

- **swe_simple.cpp (**p. **91)** A "simple" example that only runs on one core. Instead of the CPU it can also use the GPU for wave propagation.
- **swe_mpi.cpp (**p. **88)** Similar to the example above, but it can run on more the one node using MPI. If used with CUDA it requires one GPU per MPI task.
- **swe_opengl.cpp** An example program that uses the OpenGL visualization.

Chapter 3

README

Chapter 4

Todo List

Member io::VtkWriter::VtkWriter (p. **72**) **(const std::string &i_fileName, const Float2D** (p. **21**) **&i_b, const BoundarySize &i_boundarySize, int i_nX, int i_nY, float i_dX, float i_dY, int i_offsetX=0, int i_offsetY=0)**

This version can only handle a boundary layer of size 1

Chapter 5

Deprecated List

Member generateFileName (p. **100**) **(std::string baseName, int timeStep)**

Member generateFileName (p. **100**) **(std::string baseName, int timeStep, int block_X, int block_Y, std::string i_fileExtension=".vts")**

Member generateFileName (p. **100**) **(std::string i_baseName, int i_blockPositionX, int i_blockPositionY, std::string i_fileExtension=".nc")**

Chapter 6

Hierarchical Index

6.1 Class Hierarchy

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

Chapter 7

Class Index

7.1 Class List

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

Chapter 8

File Index

8.1 File List

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

Chapter 9

Class Documentation

9.1 tools::Args Class Reference

```
#include <args.hh>
```

Public Types

- enum **Argument** { **Required** = required_argument, **No** = no_argument, **Optional** = optional_argument }
- enum **Result** { **Success** = 0, **Error**, **Help** }

Public Member Functions

- **Args** (const std::string &description="", bool addHelp=true)
- void **addOption** (const std::string &longOption, char shortOption=0, const std::string &description="", Argument argument=Required, bool required=true)
- **Result parse** (int argc, char ∗const ∗argv, bool printHelp=true)
- bool **isSet** (const std::string &option)
- template<typename T >
  T **getArgument** (const std::string &option)
- template<typename T >
  T **getArgument** (const std::string &option, T defaultArgument)
- void **helpMessage** (const char ∗prog, std::ostream &out=std::cout)
- template<>
  std::string **getArgument** (const std::string &option)

9.1.1 Detailed Description

Parses command line arguments

9.1.2 Member Enumeration Documentation

9.1.2.1 enum tools::Args::Result

Enumerator

Help Help message printed

9.1.3 Member Function Documentation

9.1.3.1 Result tools::Args::parse (int *argc,* char ∗const ∗ *argv,* bool *printHelp* = true) [inline]

Returns

True of options are successfully parsed, false otherwise

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

- src/tools/**args.hh**

9.2 io::BoundarySize Struct Reference

```
#include <Writer.hh>
```

Public Member Functions

- int & **operator[ ]** (unsigned int i)
- int **operator[ ]** (unsigned int i) const

Public Attributes

- int **boundarySize** [4]

9.2.1 Detailed Description

This struct is used so we can initialize this array in the constructor.

9.2.2 Member Data Documentation

9.2.2.1 int io::BoundarySize::boundarySize[4]

boundarySize[0] == left boundarySize[1] == right boundarySize[2] == bottom boundarySize[3] == top

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

- src/writer/**Writer.hh**

9.3 Camera Class Reference

Public Member Functions

- **Camera** (const char ∗window_title)
- void **setCamera** ()
- void **reset** ()
- void **viewDistance** (float viewDistance)
- void **orient** (float angX, float angY)
- void **zoomIn** (float scaleFactor)
- void **zoomOut** (float scaleFactor)
- void **startPanning** (int xPos, int yPos)
- void **panning** (int newX, int newY)
- void **displayImage** ()

9.3.1 Constructor & Destructor Documentation

9.3.1.1 Camera::Camera (const char * *window_title*)

Constructor

Parameters

view_distance	initial view distance from the origin
window_title	title of the current window

9.3.2 Member Function Documentation

9.3.2.1 void Camera::displayImage ( )

Calculates the current framerate, updates the window title and swaps framebuffers to display the new image

9.3.2.2 void Camera::orient (float *angle_dX*, float *angle_dY*)

Increment viewing orientation of the camera

Parameters

angle_dX	angle relative to the x-axis
angle_dY	angle relative to the rotated y-axis

9.3.2.3 void Camera::panning (int *newX*, int *newY*)

User drags our object. Transform screen coordinates into world coordinates and update the objects position

9.3.2.4 void Camera::setCamera ( )

Set the camera via gluLookAt and set the light position afterwards

9.3.2.5 void Camera::startPanning (int *xPos*, int *yPos*)

User starts dragging. Remember the old mouse coordinates.

9.3.2.6 void Camera::viewDistance (float *viewDistance*)

Set the view distance

9.3.2.7 void Camera::zoomIn (float *scaleFactor*)

Zoom in

Parameters

scaleFactor	factor which is used for zooming

9.3.2.8 void Camera::zoomOut (float *scaleFactor*)

Zoom out

Parameters

scaleFactor	factor which is used for zooming

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

- src/opengl/camera.h
- src/opengl/camera.cpp

9.4 Controller Class Reference

Public Member Functions

- **Controller** (**Simulation** ∗sim, **Visualization** ∗vis)
- bool **handleEvents** ()
- bool **hasFocus** ()
- bool **isPaused** ()

9.4.1 Constructor & Destructor Documentation

9.4.1.1 Controller::Controller (Simulation ∗ *sim*, Visualization ∗ *vis*)

Constructor

Parameters

sim	instance of simulation class
vis	instance of visualization class

9.4.2 Member Function Documentation

9.4.2.1 bool Controller::handleEvents ( )

Process all user events in a loop Returns true, when user wants to quit

9.4.2.2 bool Controller::hasFocus ( )

Returns true, when window has focus

9.4.2.3 bool Controller::isPaused ( )

Return whether program is currently paused

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

- src/opengl/controller.h
- src/opengl/controller.cpp

9.5 Float1D Class Reference

```
#include <help.hh>
```

Public Member Functions

- **Float1D** (float ∗_elem, int _rows, int _stride=1)
- float & **operator[ ]** (int i)
- const float & **operator[ ]** (int i) const
- float ∗ **elemVector** ()
- int **getSize** () const

9.5.1 Detailed Description

class **Float1D** (p. 20) is a proxy class that can represent, for example, a column or row vector of a **Float2D** (p. 21) array, where row (sub-)arrays are stored with a respective stride. Besides constructor/deconstructor, the class provides overloading of the []-operator, such that elements can be accessed as v[i] (independent of the stride). The class will never allocate separate memory for the vectors, but point to the interior data structure of **Float2D** (p. 21) (or other "host" data structures).

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

- src/tools/**help.hh**

9.6 Float2D Class Reference

`#include <help.hh>`

Public Member Functions

- **Float2D** (int _cols, int _rows, bool _allocateMemory=true)
- **Float2D** (int _cols, int _rows, float ∗_elem)
- **Float2D** (**Float2D** &_elem, bool shallowCopy)
- float ∗ **operator[ ]** (int i)
- float const ∗ **operator[ ]** (int i) const
- float ∗ **elemVector** ()
- int **getRows** () const
- int **getCols** () const
- **Float1D getColProxy** (int i)
- **Float1D getRowProxy** (int j)

9.6.1 Detailed Description

class **Float2D** (p. 21) is a very basic helper class to deal with 2D float arrays: indices represent columns (1st index, "horizontal"/x-coordinate) and rows (2nd index, "vertical"/y-coordinate) of a 2D grid; values are sequentially ordered in memory using "column major" order. Besides constructor/deconstructor, the class provides overloading of the []-operator, such that elements can be accessed as a[i][j].

9.6.2 Constructor & Destructor Documentation

9.6.2.1 Float2D::Float2D (int *_cols*, int *_rows*, bool *_allocateMemory* = `true`) `[inline]`

Constructor: takes size of the 2D array as parameters and creates a respective **Float2D** (p. 21) object; allocates memory for the array, but does not initialise value.

Parameters

_cols	number of columns (i.e., elements in horizontal direction)
_rows	rumber of rows (i.e., elements in vertical directions)

9.6.2.2 Float2D::Float2D (int *_cols,* int *_rows,* float ∗ *_elem*) [inline]

Constructor: takes size of the 2D array as parameters and creates a respective **Float2D** (p. 21) object; this constructor does not allocate memory for the array, but uses the allocated memory provided via the respective variable #_elem

Parameters

_cols	number of columns (i.e., elements in horizontal direction)
_rows	rumber of rows (i.e., elements in vertical directions)
_elem	pointer to a suitably allocated region of memory to be used for thew array elements

9.6.2.3 Float2D::Float2D (Float2D & *_elem,* bool *shallowCopy*) [inline]

Constructor: takes size of the 2D array as parameters and creates a respective **Float2D** (p. 21) object; this constructor does not allocate memory for the array, but uses the allocated memory provided via the respective variable #_elem

Parameters

_cols	number of columns (i.e., elements in horizontal direction)
_rows	rumber of rows (i.e., elements in vertical directions)
_elem	pointer to a suitably allocated region of memory to be used for thew array elements

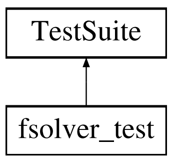
The documentation for this class was generated from the following file:

- src/tools/**help.hh**

9.7 fsolver_test Class Reference

```
#include <fsolver_test.h>
```

Inheritance diagram for fsolver_test:



Public Member Functions

- void **testFsolver_eigenvalue** (void)
- void **testFsolver_sameInput** (void)
- void **testFsolver_supersonicproblem** (void)
- void **testFsolver_NetupdateLeft** (void)
- void **testFsolver_overloadFunction** (void)

9.7.1 Detailed Description

Test class for fsolver implementation. This file contains serveral tests for task 1.4 and 2.3

9.7.2 Member Function Documentation

9.7.2.1 void fsolver_test::testFsolver_eigenvalue (void) [inline]

Testing fsolver with calculated values

9.7.2.2 void fsolver_test::testFsolver_NetupdateLeft (void) [inline]

Testing fsolver for Task 2 Testing netupdate and maxEdgeSpeed with calculated values.

9.7.2.3 void fsolver_test::testFsolver_overloadFunction (void) [inline]

Cxxtests for overloaded function

9.7.2.4 void fsolver_test::testFsolver_sameInput (void) [inline]

Testing fsolver with same inputs. left_h == right_h left_hu == right_hu

9.7.2.5 void fsolver_test::testFsolver_supersonicproblem (void) [inline]

Supersonic problem

lambda1_roe && lambda2_roe < 0 => left_wavespeed = lambda1_roe right_wavespeed = 0

lambda2_roe && lambda2_roe > 0 => left_wavespeed = 0 right_wavespeed = lambda2_roe

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

- src/solvers/fsolver_test.h

9.8 tools::Logger Class Reference

Public Member Functions

- **Logger** (const int i_processRank=0, const std::string i_programName="SWE", const std::string i_welcomeMessage="Welcome to", const std::string i_copyRights="\n\nSWE Copyright (C) 2012-2013\n"" Technische Universitaet Muenchen\n"" Department of Informatics\n"" Chair of Scientific Computing\n"" http://www5.in.tum.de/SWE\n""\n""SWE comes with ABSOLUTELY NO WARRANTY.\n""SWE is free software, and you are welcome to redistribute it\n""under certain conditions.\n""Details can be found in the file \'gpl.txt\'.", const std::string i_finishMessage="finished successfully.", const std::string i_midDelimiter="\n--\n", const std::string i_largeDelimiter="\n***\n", const std::string i_indentation="\t")
- virtual ∼**Logger** ()
- void **printWelcomeMessage** ()
- void **printFinishMessage** ()
- std::ostream & **cout** ()
- void **setProcessRank** (const int i_processRank)
- void **printString** (const std::string i_string)

- void **printNumberOfProcesses** (const int i_numberOfProcesses, const std::string i_processesName="MPI processes")
- void **printNumberOfCells** (const int i_nX, const int i_nY, const std::string i_cellMessage="cells")
- void **printNumberOfCellsPerProcess** (const int i_nX, const int i_nY)
- void **printCellSize** (const float i_dX, const float i_dY, const std::string i_unit="m")
- void **printNumberOfBlocks** (const int i_nX, const int i_nY)
- void **printStartMessage** (const std::string i_startMessage="Everything is set up, starting the simulation.")
- void **printSimulationTime** (const float i_time, const std::string i_simulationTimeMessage="Simulation at time")
- void **printOutputFileCreation** (const std::string i_fileName, const int i_blockX, const int i_blockY, const std::string i_fileType="netCDF")
- void **printOutputTime** (const float i_time, const std::string i_outputTimeMessage="Writing output file at time")
- void **printStatisticsMessage** (const std::string i_statisticsMessage="Simulation finished. Printing statistics for each process.")
- void **printSolverStatistics** (const long i_firstSolverCounter, const long i_secondSolverCounter, const int i_blockX=0, const int i_blockY=0, const std::string i_firstSolverName="f-Wave solver", const std::string i_secondSolverName="Augemented Riemann solver")
- void **updateTime** (const std::string &i_name)
- void **resetClockToCurrentTime** (const std::string &i_name)
- void **initWallClockTime** (const double i_wallClockTime)
- void **printWallClockTime** (const double i_wallClockTime, const std::string i_wallClockTimeMessage="wall clock time")
- void **printTime** (const std::string &i_name, const std::string &i_message)
- double **getTime** (const std::string &i_name)
- void **printIterationsDone** (unsigned int i_iterations, std::string i_iterationMessage="iterations done")
- void **printElementUpdatesDone** (unsigned int i_iterations, const int i_nX, const int i_nY, const std::string &i_name, const std::string i_iterationMessage="element updates per second done")

Static Public Attributes

- static **Logger logger**

9.8.1 Constructor & Destructor Documentation

9.8.1.1 tools::Logger::Logger (const int *i_processRank* = `0`, const std::string *i_programName* = `"SWE"`, const std::string *i_welcomeMessage* = `"Welcome to"`, const std::string *i_copyRights* = `"\n\nSWE Copyright (C) 2012-2013\n" "  Technische Universitaet Muenchen\n" "  Depar` `://www5.in.tum.de/SWE\n" "\n" "SWE comes with ABSOLUTELY NO WARRANTY.\n" "SWE is fr`, const std::string *i_finishMessage* = `"finished successfully."`, const std::string *i_midDelimiter* = `"\n------------------------------------------------------------------\n"`, const std::string *i_largeDelimiter* = `"\n*************************************************************\n"`, const std::string *i_indentation* = `"\t"`) `[inline]`

The Constructor. Prints the welcome message (process rank 0 only).

Parameters

i_processRank	rank of the constructing process.
i_programName	definition of the program name.
i_welcome-Message	definition of the welcome message.

i_startMessage	definition of the start message.
i_simulation-TimeMessage	definition of the simulation time message.
i_executionTime-Message	definition of the execution time message.
i_cpuTime-Message	definition of the CPU time message.
i_finishMessage	definition of the finish message.
i_midDelimiter	definition of the mid-size delimiter.
i_largeDelimiter	definition of the large delimiter.
i_indentation	definition of the indentation (used in all messages, except welcome, start and finish).

9.8.1.2 virtual tools::Logger::∼Logger ( ) `[inline]`, `[virtual]`

The Destructor. Prints the finish message (process rank 0 only).

9.8.2 Member Function Documentation

9.8.2.1 std::ostream& tools::Logger::cout ( ) `[inline]`

Default output stream of the logger.

Returns

extended (time + indentation) std::cout stream.

9.8.2.2 double tools::Logger::getTime (const std::string & *i_name*) `[inline]`

Get elapsed time

Parameters

i_name	Name of the time

Returns

elapsed time

9.8.2.3 void tools::Logger::initWallClockTime (const double *i_wallClockTime*) `[inline]`

Initialize the wall clock time.

Parameters

i_wallClockTime	value the wall block time will be set to.

9.8.2.4 void tools::Logger::printCellSize (const float *i_dX*, const float *i_dY*, const std::string *i_unit* = `"m"`) `[inline]`

Print the size of a cell

Parameters

i_dX	size in x-direction.
i_dY	size in y-direction.
i_unit	measurement unit.

9.8.2.5 void tools::Logger::printElementUpdatesDone (unsigned int *i_iterations*, const int *i_nX*, const int *i_nY*, const std::string & *i_name*, const std::string *i_iterationMessage* = "element updates per second done") [inline]

Print number of element updates done

Parameters

i_iterations	Number of iterations done
i_interation-Message	Iterations done message

9.8.2.6 void tools::Logger::printFinishMessage ( ) [inline]

Print the finish message.

9.8.2.7 void tools::Logger::printIterationsDone (unsigned int *i_iterations*, std::string *i_iterationMessage* = "iterations done") [inline]

Print number of iterations done

Parameters

i_iterations	Number of iterations done
i_interation-Message	Iterations done message

9.8.2.8 void tools::Logger::printNumberOfBlocks (const int *i_nX*, const int *i_nY*) [inline]

Print the number of defined blocks. (process rank 0 only)

Parameters

i_nX	number of blocks in x-direction.
i_nY	number of blocks in y-direction.

9.8.2.9 void tools::Logger::printNumberOfCells (const int *i_nX*, const int *i_nY*, const std::string *i_cellMessage* = "cells") [inline]

Print the number of cells. (process rank 0 only)

Parameters

i_nX	number of cells in x-direction.
i_nY	number of cells in y-direction.

i_cellMessage	cell message.

9.8.2.10 void tools::Logger::printNumberOfCellsPerProcess (const int *i_nX*, const int *i_nY*) [inline]

Print the number of cells per Process.

Parameters

i_nX	number of cells in x-direction.
i_nY	number of cells in y-direction.

9.8.2.11 void tools::Logger::printNumberOfProcesses (const int *i_numberOfProcesses*, const std::string *i_processesName* = "MPI processes") [inline]

Print the number of processes. (process rank 0 only)

Parameters

i_numberOf-Processes	number of processes.
i_processes-Name	name of the processes.

9.8.2.12 void tools::Logger::printOutputFileCreation (const std::string *i_fileName*, const int *i_blockX*, const int *i_blockY*, const std::string *i_fileType* = "netCDF") [inline]

Print the creation of an output file.

Parameters

i_fileName	name of the file.
i_blockX	block position in x-direction.
i_blockY	block position in y-direction.
i_fileType	type of the output file.

9.8.2.13 void tools::Logger::printOutputTime (const float *i_time*, const std::string *i_outputTimeMessage* = "Writing output file at time") [inline]

Print the current output time.

Parameters

i_time	time in seconds.
i_outputTime-Message	output message.

9.8.2.14 void tools::Logger::printSimulationTime (const float *i_time*, const std::string *i_simulationTimeMessage* = "Simulation at time") [inline]

Print current simulation time. (process rank 0 only)

Parameters

i_time	time in seconds.

9.8.2.15 void tools::Logger::printSolverStatistics (const long *i_firstSolverCounter*, const long *i_secondSolverCounter*, const int *i_blockX =* `0`, const int *i_blockY =* `0`, const std::string *i_firstSolverName =* `"f-Wave solver"`, const std::string *i_secondSolverName =* `"Augemented Riemann solver"`) `[inline]`

Print solver statistics

Parameters

i_firstSolver-Counter	times the first solver was used.
i_secondSolver-Counter	times the second solver was used.
i_blockX	position of the block in x-direction
i_blockY	position of the block in y-direction
i_firstSolver-Name	name of the first solver.
i_secondSolver-Name	name of the second solver.

9.8.2.16 void tools::Logger::printStartMessage (const std::string *i_startMessage =* `"Everything is set up, starting the simulation."`) `[inline]`

Print the start message. (process rank 0 only)

9.8.2.17 void tools::Logger::printStatisticsMessage (const std::string *i_statisticsMessage =* `"Simulation finished. Printing statistics for each process."`) `[inline]`

Print the statics message.

Parameters

i_statistics-Message	statistics message.

9.8.2.18 void tools::Logger::printString (const std::string *i_string*) `[inline]`

Print an arbitrary string.

Parameters

i_string	some string.

9.8.2.19 void tools::Logger::printTime (const std::string & *i_name*, const std::string & *i_message*) `[inline]`

Print elapsed time.

Parameters

i_name	Name of the timer
i_message	time message.

9.8.2.20 void tools::Logger::printWallClockTime (const double *i_wallClockTime*, const std::string *i_wallClockTimeMessage* = "wall clock time") [inline]

Print the elapsed wall clock time.

Parameters

i_wallClockTime	wall clock time message.

9.8.2.21 void tools::Logger::printWelcomeMessage ( ) [inline]

Print the welcome message.

9.8.2.22 void tools::Logger::resetClockToCurrentTime (const std::string & *i_name*) [inline]

Reset a clock to the current time

Parameters

i_name	Name of timer/clock

9.8.2.23 void tools::Logger::setProcessRank (const int *i_processRank*) [inline]

Set the process rank.

Parameters

i_processRank	process rank.

9.8.2.24 void tools::Logger::updateTime (const std::string & *i_name*) [inline]

Update a timer

Parameters

i_name	Name of timer

9.8.3 Member Data Documentation

9.8.3.1 tools::Logger tools::Logger::logger [static]

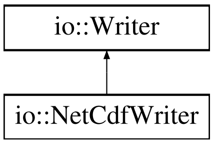
The logger all classes should use

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

- src/tools/**Logger.hh**
- src/tools/**Logger.cpp**

9.9 io::NetCdfWriter Class Reference

Inheritance diagram for io::NetCdfWriter:



Public Member Functions

- **NetCdfWriter** (const std::string &i_fileName, const **Float2D** &i_b, const **BoundarySize** &i_boundarySize, int i_nX, int i_nY, float i_dX, float i_dY, float i_originX=0., float i_originY=0., unsigned int i_flush=0)
- virtual ∼**NetCdfWriter** ()
- void **writeTimeStep** (const **Float2D** &i_h, const **Float2D** &i_hu, const **Float2D** &i_hv, float i_time)

Additional Inherited Members

9.9.1 Constructor & Destructor Documentation

9.9.1.1 io::NetCdfWriter::NetCdfWriter (const std::string & *i_baseName*, const Float2D & *i_b*, const BoundarySize & *i_boundarySize*, int *i_nX*, int *i_nY*, float *i_dX*, float *i_dY*, float *i_originX* = 0., float *i_originY* = 0., unsigned int *i_flush* = 0)

Create a netCdf-file Any existing file will be replaced.

Parameters

i_baseName	base name of the netCDF-file to which the data will be written to.
i_nX	number of cells in the horizontal direction.
i_nY	number of cells in the vertical direction.
i_dX	cell size in x-direction.
i_dY	cell size in y-direction.
i_originX	
i_originY	
i_flush	If $>$ 0, flush data to disk every i_flush write operation
i_dynamic-Bathymetry	

9.9.1.2 io::NetCdfWriter::∼NetCdfWriter ( ) [virtual]

Destructor of a netCDF-writer.

9.9.2 Member Function Documentation

9.9.2.1 void io::NetCdfWriter::writeTimeStep (const Float2D & *i_h*, const Float2D & *i_hu*, const Float2D & *i_hv*, float *i_time*) [virtual]

Writes the unknwons to a netCDF-file (-$>$ constructor) with respect to the boundary sizes.

boundarySize[0] == left boundarySize[1] == right boundarySize[2] == bottom boundarySize[3] == top

Parameters

i_h	water heights at a given time step.
i_hu	momentums in x-direction at a given time step.
i_hv	momentums in y-direction at a given time step.
i_boundarySize	size of the boundaries.
i_time	simulation time of the time step.

Implements **io::Writer** (p. 74).

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

- src/writer/**NetCdfWriter.hh**
- src/writer/**NetCdfWriter.cpp**

9.10 tools::ProgressBar Class Reference

Public Member Functions

- **ProgressBar** (float totalWork=1., int rank=0)
- void **update** (float done)
- void **clear** ()

9.10.1 Member Function Documentation

9.10.1.1 void tools::ProgressBar::update (float *done*) [inline]

Parameters

done	The amount of work already done

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

- src/tools/**ProgressBar.hh**

9.11 Shader Class Reference

Public Member Functions

- **Shader** (char const ∗vertexShaderFile, char const ∗fragmentShaderFile)
- ∼**Shader** ()
- bool **shadersLoaded** ()
- void **enableShader** ()
- void **disableShader** ()
- GLint **getUniformLocation** (const char ∗name)
- void **setUniform** (GLint location, GLfloat value)

9.11.1 Constructor & Destructor Documentation

9.11.1.1 Shader::Shader (char const ∗ *vertexShaderFile,* char const ∗ *fragmentShaderFile*)

Constructor. Check whether shaders are supported. If yes, load vertex and fragment shader from textfile into memory and compile

Parameters

vertexShaderFile	name of the text file containing the vertex shader code
fragmentShader-File	name of the text file containing the fragment shader code

9.11.1.2 Shader::∼Shader ( )

Destructor. Unload shaders and free resources.

9.11.2 Member Function Documentation

9.11.2.1 void Shader::disableShader ( )

Restores OpenGL default shaders

9.11.2.2 void Shader::enableShader ( )

Replaces OpenGL shaders by our custom shaders

9.11.2.3 GLint Shader::getUniformLocation (const char ∗ *name*) [inline]

Returns

Location of the uniform variable

9.11.2.4 void Shader::setUniform (GLint *location,* GLfloat *value*) [inline]

Set a uniform variable in the shader

9.11.2.5 bool Shader::shadersLoaded ( )

Returns, whether shaders could by loaded successfully

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

- src/opengl/shader.h
- src/opengl/shader.cpp

9.12 Simulation Class Reference

Public Member Functions

- void **restart** ()
- void **loadNewScenario** (**SWE_Scenario** ∗scene)
- void **resize** (float factor)
- void **setBathBuffer** (float ∗output)
- void **runCuda** (struct cudaGraphicsResource ∗∗vbo_resource, struct cudaGraphicsResource ∗∗vbo_-normals)
- int **getNx** ()
- int **getNy** ()

- const **Float2D** & **getBathymetry** ()
- void **getScalingApproximation** (float &bScale, float &bOffset, float &wScale)
- void **toogleLoop** ()

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

- src/opengl/simulation.h

9.13 SWE_AsagiGrid Class Reference

Public Member Functions

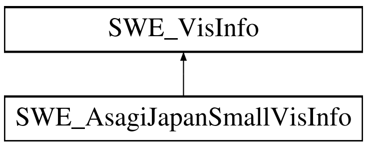
- void **open** (const std::string &i_filename)
- void **close** ()
- asagi::Grid & **grid** ()

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

- src/scenarios/**SWE_AsagiScenario.hh**

9.14 SWE_AsagiJapanSmallVisInfo Class Reference

Inheritance diagram for SWE_AsagiJapanSmallVisInfo:



Public Member Functions

- virtual float **waterVerticalScaling** ()
- virtual float **bathyVerticalScaling** ()

9.14.1 Member Function Documentation

9.14.1.1 virtual float SWE_AsagiJapanSmallVisInfo::bathyVerticalScaling ( ) [inline], [virtual]

Returns

The vertical scaling factor for the bathymetry

Reimplemented from **SWE_VisInfo** (p. 61).

9.14.1.2 virtual float SWE_AsagiJapanSmallVisInfo::waterVerticalScaling ( ) [inline], [virtual]

Returns

The vertical scaling factor of the water

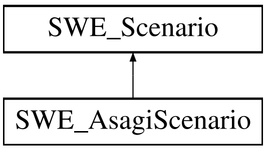
Reimplemented from **SWE_VisInfo** (p. 62).

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

- src/scenarios/**SWE_AsagiScenario_vis.hh**

9.15 SWE_AsagiScenario Class Reference

Inheritance diagram for SWE_AsagiScenario:



Public Member Functions

- **SWE_AsagiScenario** (const std::string i_bathymetryFile, const std::string i_displacementFile, const float i_-duration, const float i_simulationArea[4], const bool i_dynamicDisplacement=false)
- void **deleteGrids** ()
- float **getWaterHeight** (float i_positionX, float i_positionY)
- float **getBathymetry** (const float i_positionX, const float i_positionY)
- float **getBathymetryAndDynamicDisplacement** (const float i_positionX, const float i_positionY, const float i_time)
- bool **dynamicDisplacementAvailable** (const float i_time)
- float **endSimulation** ()
- **BoundaryType getBoundaryType** (**BoundaryEdge** i_edge)
- float **getBoundaryPos** (**BoundaryEdge** i_edge)

9.15.1 Constructor & Destructor Documentation

9.15.1.1 SWE_AsagiScenario::SWE_AsagiScenario (const std::string *i_bathymetryFile*, const std::string *i_displacementFile*, const float *i_duration*, const float *i_simulationArea[4]*, const bool *i_dynamicDisplacement* = `false`) `[inline]`

Constructor of an Asagi Scenario, which initializes the corresponding Asagi grids.

Parameters

i_originX	origin of the simulation area (x-direction)
i_originY	origin of the simulation area (y-direction)
i_bathymetryFile	path to the netCDF-bathymetry file
i_displacement-File	path to the netCDF-displacement file
i_duration	time the simulation runs (in seconds)

9.15.2 Member Function Documentation

9.15.2.1 bool SWE_AsagiScenario::dynamicDisplacementAvailable (const float *i_time*) `[inline]`

Check if there is an dynamic displacement is available for the corresponding time.

Parameters

i_time	current simulation time

Returns

true if there is data available, false else

9.15.2.2 float SWE_AsagiScenario::endSimulation ( ) [inline], [virtual]

Get the number of seconds, the simulation should run.

Returns

number of seconds, the simulation should run

Reimplemented from **SWE_Scenario** (p. 58).

9.15.2.3 float SWE_AsagiScenario::getBathymetry (const float *i_positionX*, const float *i_positionY*) [inline], [virtual]

Get the bathymetry including static displacement at a specific location

Parameters

i_positionX	position relative to the origin of the displacement grid in x-direction
i_positionY	position relative to the origin of the displacement grid in y-direction

Returns

bathymetry (after the initial displacement (static displacement)

Reimplemented from **SWE_Scenario** (p. 58).

9.15.2.4 float SWE_AsagiScenario::getBathymetryAndDynamicDisplacement (const float *i_positionX*, const float *i_positionY*, const float *i_time*) [inline]

Get the bathymetry including dynamic displacement at a specific location

Parameters

i_positionX	position relative to the origin of the displacement grid in x-direction
i_positionY	position relative to the origin of the displacement grid in y-direction
i_time	time relative to the origin of the dynamic displacement

Returns

bathymetry (after the initial displacement (static displacement), after the specified amount of time (dynamic displacement))

9.15.2.5 float SWE_AsagiScenario::getBoundaryPos (BoundaryEdge *i_edge*) [inline], [virtual]

Get the boundary positions

Parameters

i_edge	which edge

Returns

value in the corresponding dimension

Reimplemented from **SWE_Scenario** (p. 58).

9.15.2.6 BoundaryType SWE_AsagiScenario::getBoundaryType (BoundaryEdge *i_edge*) [inline], [virtual]

Get the boundary types of the simulation

Parameters

edge	specific edge

Returns

type of the edge

Reimplemented from **SWE_Scenario** (p. 58).

9.15.2.7 float SWE_AsagiScenario::getWaterHeight (float *i_positionX*, float *i_positionY*) [inline], [virtual]

Get the water height at a specific location (before the initial displacement).

Parameters

i_positionX	position relative to the origin of the bathymetry grid in x-direction
i_positionY	position relative to the origin of the bathymetry grid in y-direction

Returns

water height (before the initial displacement)

Reimplemented from **SWE_Scenario** (p. 58).

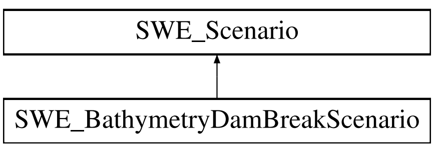
The documentation for this class was generated from the following files:

- src/scenarios/**SWE_AsagiScenario.hh**
- src/scenarios/**SWE_AsagiScenario.cpp**

9.16 SWE_BathymetryDamBreakScenario Class Reference

```
#include <SWE_simple_scenarios.hh>
```

Inheritance diagram for SWE_BathymetryDamBreakScenario:



Public Member Functions

- float **getBathymetry** (float x, float y)
- virtual float **endSimulation** ()
- virtual **BoundaryType getBoundaryType** (**BoundaryEdge** edge)
- float **getBoundaryPos** (**BoundaryEdge** i_edge)
- float **getWaterHeight** (float i_positionX, float i_positionY)

9.16.1 Detailed Description

Scenario "Bathymetry Dam Break": uniform water depth, but elevated bathymetry in the centre of the domain

9.16.2 Member Function Documentation

9.16.2.1 float SWE_BathymetryDamBreakScenario::getBoundaryPos (BoundaryEdge *i_edge*) [inline], [virtual]

Get the boundary positions

Parameters

i_edge	which edge

Returns

value in the corresponding dimension

Reimplemented from **SWE_Scenario** (p. 58).

9.16.2.2 float SWE_BathymetryDamBreakScenario::getWaterHeight (float *i_positionX,* float *i_positionY*) [inline], [virtual]

Get the water height at a specific location.

Parameters

i_positionX	position relative to the origin of the bathymetry grid in x-direction
i_positionY	position relative to the origin of the bathymetry grid in y-direction

Returns

water height (before the initial displacement)

Reimplemented from **SWE_Scenario** (p. 58).

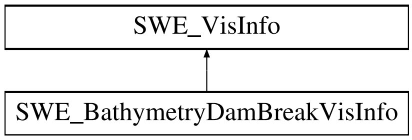
The documentation for this class was generated from the following file:

- src/scenarios/**SWE_simple_scenarios.hh**

9.17 SWE_BathymetryDamBreakVisInfo Class Reference

`#include <SWE_simple_scenarios_vis.hh>`

Inheritance diagram for SWE_BathymetryDamBreakVisInfo:



Public Member Functions

- float **bathyVerticalOffset** ()

9.17.1 Detailed Description

VisInfo "Bathymetry Dam Break": uniform water depth, but elevated bathymetry in the center of the domain Set bathymetry offset hence it is visible in the screen

9.17.2 Member Function Documentation

9.17.2.1 float SWE_BathymetryDamBreakVisInfo::bathyVerticalOffset ( ) [inline], [virtual]

Returns

The vertical offset for the bathymetry. Should be 0 for "real" scenarios (scenarios with dry areas)

Reimplemented from **SWE_VisInfo** (p. 61).

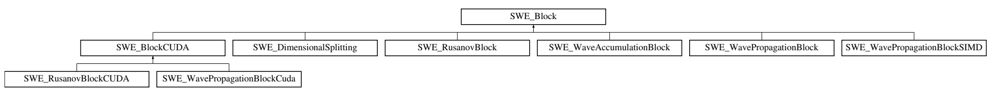
The documentation for this class was generated from the following file:

- src/scenarios/SWE_simple_scenarios_vis.hh

9.18 SWE_Block Class Reference

#include <SWE_Block.hh>

Inheritance diagram for SWE_Block:



Public Member Functions

- void **initScenario** (float _offsetX, float _offsetY, **SWE_Scenario** &i_scenario, const bool i_multipleBlocks=false)

  initialise unknowns to a specific scenario:
- void **setWaterHeight** (float(∗_h)(float, float))

  set the water height according to a given function
- void **setDischarge** (float(∗_u)(float, float), float(∗_v)(float, float))

  set the momentum/discharge according to the provided functions
- void **setBathymetry** (float _b)

  set the bathymetry to a uniform value
- void **setBathymetry** (float(∗_b)(float, float))

  set the bathymetry according to a given function

- const **Float2D** & **getWaterHeight** ()
    provides read access to the water height array
- const **Float2D** & **getDischarge_hu** ()
    provides read access to the momentum/discharge array (x-component)
- const **Float2D** & **getDischarge_hv** ()
    provides read access to the momentum/discharge array (y-component)
- const **Float2D** & **getBathymetry** ()
    provides read access to the bathymetry data
- void **setBoundaryType** (**BoundaryEdge** edge, **BoundaryType** boundtype, const **SWE_Block1D** ∗inflow=NULL)
    set type of boundary condition for the specified boundary
- virtual **SWE_Block1D** ∗ **registerCopyLayer** (**BoundaryEdge** edge)
    return a pointer to proxy class to access the copy layer
- virtual **SWE_Block1D** ∗ **grabGhostLayer** (**BoundaryEdge** edge)
    "grab" the ghost layer in order to set these values externally
- void **setGhostLayer** ()
    set values in ghost layers
- float **getMaxTimestep** ()
    return maximum size of the time step to ensure stability of the method
- void **computeMaxTimestep** (const float i_dryTol=0.1, const float i_cflNumber=0.4)
- virtual void **simulateTimestep** (float dt)
    execute a single time step (with fixed time step size) of the simulation
- virtual float **simulate** (float tStart, float tEnd)
- virtual void **computeNumericalFluxes** ()=0
    compute the numerical fluxes for each edge of the Cartesian grid
- virtual void **updateUnknowns** (float dt)=0
    compute the new values of the unknowns h, hu, and hv in all grid cells
- int **getNx** ()
    *returns **nx** (*p. *40), i.e. the grid size in x-direction*
- int **getNy** ()
    *returns **ny** (*p. *40), i.e. the grid size in y-direction*

Static Public Attributes

- static const float **g** = 9.81f
    static variable that holds the gravity constant (g = 9.81 m/s^2):

Protected Member Functions

- **SWE_Block** (int l_nx, int l_ny, float l_dx, float l_dy)
- virtual ∼**SWE_Block** ()
- void **setBoundaryBathymetry** ()
- virtual void **synchAfterWrite** ()
- virtual void **synchWaterHeightAfterWrite** ()
- virtual void **synchDischargeAfterWrite** ()
- virtual void **synchBathymetryAfterWrite** ()
- virtual void **synchGhostLayerAfterWrite** ()
- virtual void **synchBeforeRead** ()
- virtual void **synchWaterHeightBeforeRead** ()
- virtual void **synchDischargeBeforeRead** ()
- virtual void **synchBathymetryBeforeRead** ()
- virtual void **synchCopyLayerBeforeRead** ()
- virtual void **setBoundaryConditions** ()
    set boundary conditions in ghost layers (set boundary conditions)

Protected Attributes

- int **nx**

  size of Cartesian arrays in x-direction
- int **ny**

  size of Cartesian arrays in y-direction
- float **dx**

  mesh size of the Cartesian grid in x-direction
- float **dy**

  mesh size of the Cartesian grid in y-direction
- **Float2D h**

  array that holds the water height for each element
- **Float2D hu**

  array that holds the x-component of the momentum for each element (water height h multiplied by velocity in x-direction)
- **Float2D hv**

  array that holds the y-component of the momentum for each element (water height h multiplied by velocity in y-direction)
- **Float2D b**

  array that holds the bathymetry data (sea floor elevation) for each element
- **BoundaryType boundary** [4]

  type of boundary conditions at LEFT, RIGHT, TOP, and BOTTOM boundary
- const **SWE_Block1D** ∗ **neighbour** [4]

  for CONNECT boundaries: pointer to connected neighbour block
- float **maxTimestep**

  maximum time step allowed to ensure stability of the method
- float **offsetX**

  x-coordinate of the origin (left-bottom corner) of the Cartesian grid
- float **offsetY**

  y-coordinate of the origin (left-bottom corner) of the Cartesian grid

9.18.1 Detailed Description

SWE_Block (p. 38) is the main data structure to compute our shallow water model on a single Cartesian grid block-: **SWE_Block** (p. 38) is an abstract class (and interface) that should be extended by respective implementation classes.

Cartesian Grid for Discretization:

SWE_Blocks uses a regular Cartesian grid of size **nx** (p. 40) by **ny** (p. 40), where each grid cell carries three unknowns:

- the water level **h** (p. 40)
- the momentum components **hu** (p. 40) and **hv** (p. 40) (in x- and y- direction, resp.)
- the bathymetry **b** (p. 40)

Each of the components is stored as a 2D array, implemented as a **Float2D** (p. 21) object, and are defined on grid indices [0,..,**nx** (p. 40)+1]∗[0,..,**ny** (p. 40)+1]. The computational domain is indexed with [1,..,**nx** (p. 40)]∗[1,..,**ny** (p. 40)].

The mesh sizes of the grid in x- and y-direction are stored in static variables **dx** (p. 40) and **dy** (p. 40). The position of the Cartesian grid in space is stored via the coordinates of the left-bottom corner of the grid, in the variables **offsetX** (p. 40) and **offsetY** (p. 40).

Ghost layers:

To implement the behaviour of the fluid at boundaries and for using multiple block in serial and parallel settings, **SW-E_Block** (p. 38) adds an additional layer of so-called ghost cells to the Cartesian grid, as illustrated in the following figure. Cells in the ghost layer have indices 0 or **nx** (p. 40)+1 / **ny** (p. 40)+1.

Memory Model:

The variables **h** (p. 40), **hu** (p. 40), **hv** (p. 40) for water height and momentum will typically be updated by classes derived from **SWE_Block** (p. 38). However, it is not assumed that such and updated will be performed in every time step. Instead, subclasses are welcome to update **h** (p. 40), **hu** (p. 40), and **hv** (p. 40) in a lazy fashion, and keep data in faster memory (incl. local memory of acceleration hardware, such as GPGPUs), instead.

It is assumed that the bathymetry data **b** (p. 40) is not changed during the algorithm (up to the exceptions mentioned in the following).

To force a synchronization of the respective data structures, the following methods are provided as part of **SWE_-Block** (p. 38):

- **synchAfterWrite()** (p. 46) to synchronize **h** (p. 40), **hu** (p. 40), **hv** (p. 40), and **b** (p. 40) after an external update (reading a file, e.g.);
- **synchWaterHeightAfterWrite()** (p. 47), **synchDischargeAfterWrite()** (p. 47), **synchBathymetryAfter-Write()** (p. 46): to synchronize only **h** (p. 40) or momentum (**hu** (p. 40) and **hv** (p. 40)) or bathymetry **b** (p. 40);
- **synchGhostLayerAfterWrite()** (p. 47) to synchronize only the ghost layers
- **synchBeforeRead()** (p. 46) to synchronize **h** (p. 40), **hu** (p. 40), **hv** (p. 40), and **b** (p. 40) before an output of the variables (writing a visualization file, e.g.)
- **synchWaterHeightBeforeRead()** (p. 47), **synchDischargeBeforeRead()** (p. 47), **synchBathymetryBefore-Read()** (p. 46): as **synchBeforeRead()** (p. 46), but only for the specified variables
- **synchCopyLayerBeforeRead()** (p. 47): synchronizes the copy layer only (i.e., a layer that is to be replicated in a neighbouring **SWE_Block** (p. 38).

Derived Classes

As **SWE_Block** (p. 38) just provides an abstract base class together with the most important data structures, the implementation of concrete models is the job of respective derived classes (see the class diagram at the top of this page). Similar, parallel implementations that are based on a specific parallel programming model (such as OpenMP) or parallel architecture (such as GPU/CUDA) should form subclasses of their own. Please refer to the documentation of these classes for more details on the model and on the parallelisation approach.

9.18.2 Constructor & Destructor Documentation

9.18.2.1 SWE_Block::SWE_Block (int *l_nx,* int *l_ny,* float *l_dx,* float *l_dy*) `[protected]`

Constructor: allocate variables for simulation

unknowns h (water height), hu,hv (discharge in x- and y-direction), and b (bathymetry) are defined on grid indices [0,..,nx+1]∗[0,..,ny+1] -> computational domain is [1,..,nx]∗[1,..,ny] -> plus ghost cell layer

The constructor is protected: no instances of **SWE_Block** (p. 38) can be generated.

9.18.2.2 SWE_Block::∼SWE_Block ( ) `[protected]`, `[virtual]`

Destructor: de-allocate all variables

9.18.3 Member Function Documentation

9.18.3.1 void SWE_Block::computeMaxTimestep (const float *i_dryTol =* 0.1, const float *i_cflNumber =* 0.4)

Compute the largest allowed time step for the current grid block (reference implementation) depending on the current values of variables h, hu, and hv, and store this time step size in member variable maxTimestep.

Parameters

i_dryTol	dry tolerance (dry cells do not affect the time step).
i_cflNumber	CFL number of the used method.

9.18.3.2 virtual void SWE_Block::computeNumericalFluxes ( ) [pure virtual]

compute the numerical fluxes for each edge of the Cartesian grid

The computation of fluxes strongly depends on the chosen numerical method. Hence, this purely virtual function has to be implemented in the respective derived classes.

Implemented in **SWE_DimensionalSplitting** (p. 52), **SWE_WavePropagationBlockSIMD** (p. 68), **SWE_WavePropagationBlock** (p. 64), **SWE_WavePropagationBlockCuda** (p. 65), **SWE_WaveAccumulationBlock** (p. 63), **SWE_RusanovBlock** (p. 55), and **SWE_RusanovBlockCUDA** (p. 57).

9.18.3.3 const Float2D & SWE_Block::getBathymetry ( )

provides read access to the bathymetry data

return reference to bathymetry unknown b

9.18.3.4 const Float2D & SWE_Block::getDischarge_hu ( )

provides read access to the momentum/discharge array (x-component)

return reference to discharge unknown hu

9.18.3.5 const Float2D & SWE_Block::getDischarge_hv ( )

provides read access to the momentum/discharge array (y-component)

return reference to discharge unknown hv

9.18.3.6 float SWE_Block::getMaxTimestep ( ) [inline]

return maximum size of the time step to ensure stability of the method

Returns

current value of the member variable **maxTimestep** (p. 48)

9.18.3.7 const Float2D & SWE_Block::getWaterHeight ( )

provides read access to the water height array

Restores values for h, v, and u from file data

Parameters

_b	array holding b-values in sequence return reference to water height unknown h

9.18.3.8 SWE_Block1D ∗ SWE_Block::grabGhostLayer (BoundaryEdge *edge*) [virtual]

"grab" the ghost layer in order to set these values externally "grab" the ghost layer at the specific boundary in order to set boundary values in this ghost layer externally. The boundary conditions at the respective ghost layer is set to PASSIVE, such that the grabbing program component is responsible to provide correct values in the ghost layer, for example by receiving data from a remote copy layer via MPI communication.

Parameters

specified	edge

Returns

a **SWE_Block1D** (p. 48) object that contains row variables h, hu, and hv

Reimplemented in **SWE_BlockCUDA** (p. 50).

9.18.3.9 void SWE_Block::initScenario (float *_offsetX,* float *_offsetY,* SWE_Scenario & *i_scenario,* const bool *i_multipleBlocks* = `false`)

initialise unknowns to a specific scenario:

Initializes the unknowns and bathymetry in all grid cells according to the given **SWE_Scenario** (p. 58).

In the case of multiple SWE_Blocks at this point, it is not clear how the boundary conditions should be set. This is because an isolated **SWE_Block** (p. 38) doesn't have any in information about the grid. Therefore the calling routine, which has the information about multiple blocks, has to take care about setting the right boundary conditions.

Parameters

i_scenario	scenario, which is used during the setup.
i_multipleBlocks	are the multiple SWE_blocks?

9.18.3.10 SWE_Block1D * SWE_Block::registerCopyLayer (BoundaryEdge *edge*) `[virtual]`

return a pointer to proxy class to access the copy layer

register the row or column layer next to a boundary as a "copy layer", from which values will be copied into the ghost layer or a neighbour;

Returns

a **SWE_Block1D** (p. 48) object that contains row variables h, hu, and hv

Reimplemented in **SWE_BlockCUDA** (p. 50).

9.18.3.11 void SWE_Block::setBathymetry (float *_b*)

set the bathymetry to a uniform value

set Bathymetry b in all grid cells (incl. ghost/boundary layers) to a uniform value bathymetry source terms are re-computed

9.18.3.12 void SWE_Block::setBathymetry (float(*)(float, float) *_b*)

set the bathymetry according to a given function

set Bathymetry b in all grid cells (incl. ghost/boundary layers) using the specified bathymetry function; bathymetry source terms are re-computed

9.18.3.13 void SWE_Block::setBoundaryBathymetry ( ) [protected]

Sets the bathymetry on OUTFLOW or WALL boundaries. Should be called very time a boundary is changed to a OUTFLOW or WALL boundary **or** the bathymetry changes.

9.18.3.14 void SWE_Block::setBoundaryConditions ( ) [protected], [virtual]

set boundary conditions in ghost layers (set boundary conditions)

set the values of all ghost cells depending on the specifed boundary conditions

- set boundary conditions for typs WALL and OUTFLOW
- derived classes need to transfer ghost layers

Reimplemented in **SWE_BlockCUDA** (p. 50).

9.18.3.15 void SWE_Block::setBoundaryType (BoundaryEdge *edge,* BoundaryType *boundtype,* const SWE_Block1D ∗ *i_inflow* = NULL)

set type of boundary condition for the specified boundary

Set the boundary type for specific block boundary.

Parameters

i_edge	location of the edge relative to the SWE_block.
i_boundaryType	type of the boundary condition.
i_inflow	pointer to an **SWE_Block1D** (p. 48), which specifies the inflow (should be NULL for WALL or OUTFLOW boundary)

9.18.3.16 void SWE_Block::setDischarge (float(∗)(float, float) *_u,* float(∗)(float, float) *_v*)

set the momentum/discharge according to the provided functions

set discharge in all interior grid cells (i.e. except ghost layer) to values specified by parameter functions Note: unknowns hu and hv represent momentum, while parameters u and v are velocities!

9.18.3.17 void SWE_Block::setGhostLayer ( )

set values in ghost layers

set the values of all ghost cells depending on the specifed boundary conditions; if the ghost layer replicates the variables of a remote **SWE_Block** (p. 38), the values are copied

9.18.3.18 void SWE_Block::setWaterHeight (float(∗)(float, float) *_h*)

set the water height according to a given function

set water height h in all interior grid cells (i.e. except ghost layer) to values specified by parameter function _h

9.18.3.19 float SWE_Block::simulate (float *i_tStart,* float *i_tEnd*) [virtual]

perform the simulation starting with simulation time tStart, until simulation time tEnd is reached

simulate implements the main simulation loop between two checkpoints; Note: this implementation can only be used, if you only use a single **SWE_Block** (p. 38) and only apply simple boundary conditions! In particular, **SWE_Block::simulate** (p. 45) can not trigger calls to exchange values of copy and ghost layers between blocks!

Parameters

tStart	time where the simulation is started
tEnd	time of the next checkpoint

Returns

actual end time reached

Reimplemented in **SWE_WavePropagationBlockSIMD** (p. 68), **SWE_WavePropagationBlockCuda** (p. 66), **SW-E_RusanovBlockCUDA** (p. 57), and **SWE_RusanovBlock** (p. 56).

9.18.3.20 void SWE_Block::simulateTimestep (float *dt*) [virtual]

execute a single time step (with fixed time step size) of the simulation

Executes a single timestep with fixed time step size

- compute net updates for every edge
- update cell values with the net updates

Parameters

dt	time step width of the update

Reimplemented in **SWE_WavePropagationBlockSIMD** (p. 68), **SWE_WavePropagationBlockCuda** (p. 66), **SW-E_RusanovBlockCUDA** (p. 57), and **SWE_RusanovBlock** (p. 56).

9.18.3.21 void SWE_Block::synchAfterWrite ( ) [protected], [virtual]

Update all temporary and non-local (for heterogeneous computing) variables after an external update of the main variables h, hu, hv, and b.

Reimplemented in **SWE_BlockCUDA** (p. 51).

9.18.3.22 void SWE_Block::synchBathymetryAfterWrite ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables after an external update of the bathymetry b

Reimplemented in **SWE_BlockCUDA** (p. 51).

9.18.3.23 void SWE_Block::synchBathymetryBeforeRead ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables before an external access to the bathymetry b

Reimplemented in **SWE_BlockCUDA** (p. 51).

9.18.3.24 void SWE_Block::synchBeforeRead ( ) [protected], [virtual]

Update all temporary and non-local (for heterogeneous computing) variables before an external access to the main variables h, hu, hv, and b.

Reimplemented in **SWE_BlockCUDA** (p. 51).

9.18.3.25 void SWE_Block::synchCopyLayerBeforeRead ( ) [protected], [virtual]

Update (for heterogeneous computing) variables in copy layers before an external access to the unknowns

Reimplemented in **SWE_BlockCUDA** (p. 51).

9.18.3.26 void SWE_Block::synchDischargeAfterWrite ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables after an external update of the discharge variables hu and hv

Reimplemented in **SWE_BlockCUDA** (p. 51).

9.18.3.27 void SWE_Block::synchDischargeBeforeRead ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables before an external access to the discharge variables hu and hv

Reimplemented in **SWE_BlockCUDA** (p. 51).

9.18.3.28 void SWE_Block::synchGhostLayerAfterWrite ( ) [protected], [virtual]

Update the ghost layers (only for CONNECT and PASSIVE boundary conditions) after an external update of the main variables h, hu, hv, and b in the ghost layer.

Reimplemented in **SWE_BlockCUDA** (p. 51).

9.18.3.29 void SWE_Block::synchWaterHeightAfterWrite ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables after an external update of the water height h

Reimplemented in **SWE_BlockCUDA** (p. 52).

9.18.3.30 void SWE_Block::synchWaterHeightBeforeRead ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables before an external access to the water height h

Reimplemented in **SWE_BlockCUDA** (p. 52).

9.18.3.31 virtual void SWE_Block::updateUnknowns (float *dt*) [pure virtual]

compute the new values of the unknowns h, hu, and hv in all grid cells

based on the numerical fluxes (computed by computeNumericalFluxes) and the specified time step size dt, an Euler time step is executed. As the computational fluxes will depend on the numerical method, this purely virtual function has to be implemented separately for each specific numerical model (and parallelisation approach).

Parameters

dt	size of the time step

Implemented in **SWE_DimensionalSplitting** (p. 53), **SWE_WavePropagationBlockSIMD** (p. 69), **SWE_WavePropagationBlock** (p. 65), **SWE_WavePropagationBlockCuda** (p. 66), **SWE_WaveAccumulationBlock** (p. 63), **SWE_RusanovBlock** (p. 56), and **SWE_RusanovBlockCUDA** (p. 58).

9.18.4 Member Data Documentation

9.18.4.1 float SWE_Block::maxTimestep [protected]

maximum time step allowed to ensure stability of the method

maxTimestep can be updated as part of the methods computeNumericalFluxes and updateUnknowns (depending on the numerical method)

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

- src/blocks/**SWE_Block.hh**
- src/blocks/**SWE_Block.cpp**

9.19 SWE_Block1D Struct Reference

```
#include <SWE_Block.hh>
```

Public Member Functions

- **SWE_Block1D** (const **Float1D** &_h, const **Float1D** &_hu, const **Float1D** &_hv)
- **SWE_Block1D** (float ∗_h, float ∗_hu, float ∗_hv, int _size, int _stride=1)

Public Attributes

- **Float1D h**
- **Float1D hu**
- **Float1D hv**

9.19.1 Detailed Description

SWE_Block1D (p. 48) is a simple struct that can represent a single line or row of **SWE_Block** (p. 38) unknowns (using the **Float1D** (p. 20) proxy class). It is intended to unify the implementation of inflow and periodic boundary conditions, as well as the ghost/copy-layer connection between several **SWE_Block** (p. 38) grids.

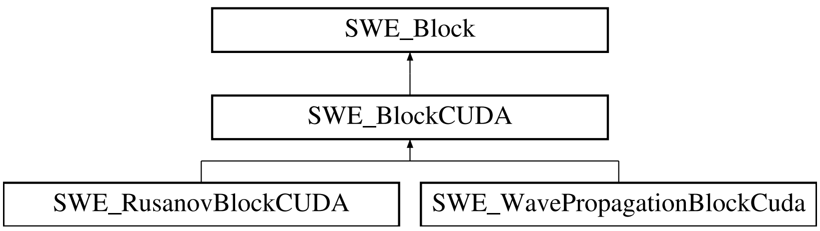
The documentation for this struct was generated from the following file:

- src/blocks/**SWE_Block.hh**

9.20 SWE_BlockCUDA Class Reference

```
#include <SWE_BlockCUDA.hh>
```

Inheritance diagram for SWE_BlockCUDA:



Public Member Functions

- **SWE_BlockCUDA** (int l_nx, int l_ny, float l_dx, float l_dy)
- virtual **SWE_Block1D** ∗ **registerCopyLayer** (**BoundaryEdge** edge)

  return a pointer to proxy class to access the copy layer
- virtual **SWE_Block1D** ∗ **grabGhostLayer** (**BoundaryEdge** edge)

  "grab" the ghost layer in order to set these values externally
- const float ∗ **getCUDA_waterHeight** ()
- const float ∗ **getCUDA_bathymetry** ()

Static Public Member Functions

- static void **printDeviceInformation** ()
- static void **init** (int i_cudaDevice=0)
- static void **finalize** ()

Protected Member Functions

- virtual void **synchAfterWrite** ()
- virtual void **synchWaterHeightAfterWrite** ()
- virtual void **synchDischargeAfterWrite** ()
- virtual void **synchBathymetryAfterWrite** ()
- virtual void **synchGhostLayerAfterWrite** ()
- virtual void **synchBeforeRead** ()
- virtual void **synchWaterHeightBeforeRead** ()
- virtual void **synchDischargeBeforeRead** ()
- virtual void **synchBathymetryBeforeRead** ()
- virtual void **synchCopyLayerBeforeRead** ()
- virtual void **setBoundaryConditions** ()

  set boundary conditions in ghost layers (set boundary conditions)

Protected Attributes

- float ∗ **hd**
- float ∗ **hud**
- float ∗ **hvd**
- float ∗ **bd**

Additional Inherited Members

9.20.1 Detailed Description

SWE_BlockCUDA (p. 48) extends the base class **SWE_Block** (p. 38) towards a base class for a CUDA implementation of the shallow water equations. It adds the respective variables in GPU memory, and provides methods for data transfer between main and GPU memory.

9.20.2 Member Function Documentation

9.20.2.1 static void SWE_BlockCUDA::finalize ( ) [static]

Cleans up the cuda device

9.20.2.2 const float* SWE_BlockCUDA::getCUDA_bathymetry ( ) [inline]

Returns

pointer to the array #hb (bathymetry) in device memory

9.20.2.3 const float* SWE_BlockCUDA::getCUDA_waterHeight ( ) [inline]

Returns

pointer to the array #hd (water height) in device memory

9.20.2.4 virtual SWE_Block1D* SWE_BlockCUDA::grabGhostLayer (BoundaryEdge *edge*) [virtual]

"grab" the ghost layer in order to set these values externally

"grab" the ghost layer at the specific boundary in order to set boundary values in this ghost layer externally. The boundary conditions at the respective ghost layer is set to PASSIVE, such that the grabbing program component is responsible to provide correct values in the ghost layer, for example by receiving data from a remote copy layer via MPI communication.

Parameters

specified	edge

Returns

a **SWE_Block1D** (p. 48) object that contains row variables h, hu, and hv

Reimplemented from **SWE_Block** (p. 43).

9.20.2.5 static void SWE_BlockCUDA::init (int *i_cudaDevice =* 0) [static]

Initializes the cuda device Has to be called once at the beginning.

9.20.2.6 virtual SWE_Block1D* SWE_BlockCUDA::registerCopyLayer (BoundaryEdge *edge*) [virtual]

return a pointer to proxy class to access the copy layer

register the row or column layer next to a boundary as a "copy layer", from which values will be copied into the ghost layer or a neighbour;

Returns

a **SWE_Block1D** (p. 48) object that contains row variables h, hu, and hv

Reimplemented from **SWE_Block** (p. 44).

9.20.2.7 virtual void SWE_BlockCUDA::setBoundaryConditions ( ) [protected], [virtual]

set boundary conditions in ghost layers (set boundary conditions)

set the values of all ghost cells depending on the specifed boundary conditions

- set boundary conditions for typs WALL and OUTFLOW
- derived classes need to transfer ghost layers

Reimplemented from **SWE_Block** (p. 45).

9.20.2.8 virtual void SWE_BlockCUDA::synchAfterWrite ( ) [protected], [virtual]

Update all temporary and non-local (for heterogeneous computing) variables after an external update of the main variables h, hu, hv, and b.

Reimplemented from **SWE_Block** (p. 46).

9.20.2.9 virtual void SWE_BlockCUDA::synchBathymetryAfterWrite ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables after an external update of the bathymetry b

Reimplemented from **SWE_Block** (p. 46).

9.20.2.10 virtual void SWE_BlockCUDA::synchBathymetryBeforeRead ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables before an external access to the bathymetry b

Reimplemented from **SWE_Block** (p. 46).

9.20.2.11 virtual void SWE_BlockCUDA::synchBeforeRead ( ) [protected], [virtual]

Update all temporary and non-local (for heterogeneous computing) variables before an external access to the main variables h, hu, hv, and b.

Reimplemented from **SWE_Block** (p. 46).

9.20.2.12 virtual void SWE_BlockCUDA::synchCopyLayerBeforeRead ( ) [protected], [virtual]

Update (for heterogeneous computing) variables in copy layers before an external access to the unknowns

Reimplemented from **SWE_Block** (p. 47).

9.20.2.13 virtual void SWE_BlockCUDA::synchDischargeAfterWrite ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables after an external update of the discharge variables hu and hv

Reimplemented from **SWE_Block** (p. 47).

9.20.2.14 virtual void SWE_BlockCUDA::synchDischargeBeforeRead ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables before an external access to the discharge variables hu and hv

Reimplemented from **SWE_Block** (p. 47).

9.20.2.15 virtual void SWE_BlockCUDA::synchGhostLayerAfterWrite ( ) [protected], [virtual]

Update the ghost layers (only for CONNECT and PASSIVE boundary conditions) after an external update of the main variables h, hu, hv, and b in the ghost layer.

Reimplemented from **SWE_Block** (p. 47).

9.20.2.16 virtual void SWE_BlockCUDA::synchWaterHeightAfterWrite ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables after an external update of the water height h

Reimplemented from **SWE_Block** (p. 47).

9.20.2.17 virtual void SWE_BlockCUDA::synchWaterHeightBeforeRead ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables before an external access to the water height h

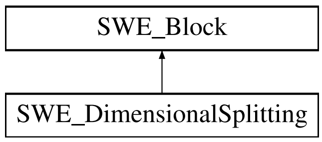
Reimplemented from **SWE_Block** (p. 47).

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

- src/blocks/cuda/**SWE_BlockCUDA.hh**

9.21 SWE_DimensionalSplitting Class Reference

Inheritance diagram for SWE_DimensionalSplitting:



Public Member Functions

- **SWE_DimensionalSplitting** (int l_nx, int l_ny, float l_dx, float l_dy)
- void **computeNumericalFluxes** ()
- void **updateUnknowns** (float dt)
- virtual ∼**SWE_DimensionalSplitting** ()

Additional Inherited Members

9.21.1 Constructor & Destructor Documentation

9.21.1.1 SWE_DimensionalSplitting::SWE_DimensionalSplitting (int *l_nx*, int *l_ny*, float *l_dx*, float *l_dy*)

Constructor for **SWE_DimensionalSplitting** (p. 52) Declaring arrays and variables

9.21.1.2 SWE_DimensionalSplitting::∼SWE_DimensionalSplitting ( ) [virtual]

Delete arrays and variables

9.21.2 Member Function Documentation

9.21.2.1 void SWE_DimensionalSplitting::computeNumericalFluxes ( ) [virtual]

Compute numerical fluxes with x and y sweep for Task 3.2

Implements **SWE_Block** (p. 43).

9.21.2.2 void SWE_DimensionalSplitting::updateUnknowns (float *dt*) [virtual]

Updateing height and momentum in x and y direction

Implements **SWE_Block** (p. 47).

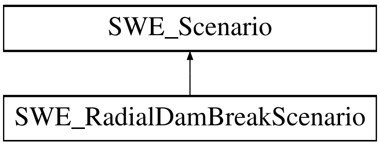
The documentation for this class was generated from the following files:

- src/blocks/**SWE_Block.hh**
- src/blocks/**SWE_Block.cpp**

9.22 SWE_RadialDamBreakScenario Class Reference

```
#include <SWE_simple_scenarios.hh>
```

Inheritance diagram for SWE_RadialDamBreakScenario:



Public Member Functions

- float **getBathymetry** (float x, float y)
- float **getWaterHeight** (float x, float y)
- virtual float **endSimulation** ()
- virtual **BoundaryType getBoundaryType** (**BoundaryEdge** edge)
- float **getBoundaryPos** (**BoundaryEdge** i_edge)

9.22.1 Detailed Description

Scenario "Radial Dam Break": elevated water in the center of the domain

9.22.2 Member Function Documentation

9.22.2.1 float SWE_RadialDamBreakScenario::getBoundaryPos (BoundaryEdge *i_edge*) [inline], [virtual]

Get the boundary positions

Parameters

i_edge	which edge

Returns

value in the corresponding dimension

Reimplemented from **SWE_Scenario** (p. 58).

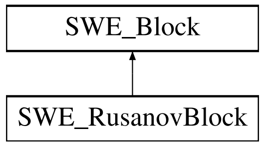
The documentation for this class was generated from the following file:

- src/scenarios/**SWE_simple_scenarios.hh**

9.23 SWE_RusanovBlock Class Reference

```
#include <SWE_RusanovBlock.hh>
```

Inheritance diagram for SWE_RusanovBlock:



Public Member Functions

- **SWE_RusanovBlock** (float _offsetX=0, float _offsetY=0)
- virtual ~**SWE_RusanovBlock** ()
- virtual void **simulateTimestep** (float dt)

    execute a single time step of the simulation
- virtual float **simulate** (float tStart, float tEnd)

    compute simulate from specified start to end time
- virtual void **computeNumericalFluxes** ()

    compute flux terms on edges
- virtual void **updateUnknowns** (float dt)

    update unknowns according to fluxes (Euler time step)

Protected Member Functions

- virtual void **computeBathymetrySources** ()

    compute source terms
- float **computeLocalSV** (int i, int j, char dir)
- virtual void **computeMaxTimestep** ()

Static Protected Member Functions

- static float **computeFlux** (float fLoc, float fNeigh, float xiLoc, float xiNeigh, float llf)

Protected Attributes

- **Float2D Fh**
- **Float2D Fhu**
- **Float2D Fhv**
- **Float2D Gh**
- **Float2D Ghu**
- **Float2D Ghv**
- **Float2D Bx**
- **Float2D By**

Friends

- ostream & **operator**<< (ostream &os, const **SWE_RusanovBlock** &swe)

Additional Inherited Members

9.23.1 Detailed Description

SWE_RusanovBlock (p. 54) is an implementation of the **SWE_Block** (p. 38) abstract class. It uses a simple Rusanov flux (aka local Lax-Friedrich) in the model, with some simple modifications to obtain a well-balanced scheme.

9.23.2 Constructor & Destructor Documentation

9.23.2.1 SWE_RusanovBlock::SWE_RusanovBlock (float *_offsetX* = 0, float *_offsetY* = 0)

Constructor: allocate variables for simulation

unknowns h,hu,hv,b are defined on grid indices [0,..,nx+1]∗[0,..,ny+1] -> computational domain is [1,..,nx]∗[1,..,ny] -> plus ghost cell layer

flux terms are defined for edges with indices [0,..,nx]∗[1,..,ny] or [1,..,nx]∗`0,..,ny` Flux term with index (i,j) is located on the edge between cells with index (i,j) and (i+1,j) or (i,j+1)

bathymetry source terms are defined for cells with indices [1,..,nx]∗[1,..,ny]

@ param _offsetX x coordinate of block origin @ param _offsetY y coordinate of block origin

9.23.2.2 SWE_RusanovBlock::∼SWE_RusanovBlock ( ) `[virtual]`

Destructor: de-allocate all variables

9.23.3 Member Function Documentation

9.23.3.1 void SWE_RusanovBlock::computeBathymetrySources ( ) `[protected]`, `[virtual]`

compute source terms

compute the bathymetry source terms in all cells

9.23.3.2 float SWE_RusanovBlock::computeFlux (float *fLow*, float *fHigh*, float *xiLow*, float *xiHigh*, float *llf*) `[static]`, `[protected]`

compute the flux term on a given edge (acc. to local Lax-Friedrich method aka Rusanov flux): fLow and fHigh contain the values of the flux function in the two adjacent grid cells xiLow and xiHigh are the values of the unknowns in the two adjacent grid cells "Low" represents the cell with lower i/j index ("High" for larger index). llf should contain the local signal velocity (as compute by computeLocalSV) for llf=dx/dt (or dy/dt), we obtain the standard Lax Friedrich method

9.23.3.3 float SWE_RusanovBlock::computeLocalSV (int *i*, int *j*, char *dir*) `[protected]`

computes the local signal velocity in x- or y-direction for two adjacent cells with indices (i,j) and (i+1,j) (if dir='x') or (i,j+1) (if dir='y'

9.23.3.4 void SWE_RusanovBlock::computeNumericalFluxes ( ) `[virtual]`

compute flux terms on edges

compute the flux terms on all edges; before the computation, computeBathymetrySources is called

Implements **SWE_Block** (p. 43).

9.23.3.5 float SWE_RusanovBlock::simulate (float *tStart,* float *tEnd*) [virtual]

compute simulate from specified start to end time

implements interface function simulate: perform forward-Euler time steps, starting with simulation time tStart,: until simulation time tEnd is reached; boundary conditions and bathymetry source terms are computed for each timestep as required - intended as main simulation loop between two checkpoints

Reimplemented from **SWE_Block** (p. 45).

9.23.3.6 void SWE_RusanovBlock::simulateTimestep (float *dt*) [virtual]

execute a single time step of the simulation

Depending on the current values of h, hu, hv (incl. ghost layers) update these unknowns in each grid cell (ghost layers and bathymetry are not updated). The Rusanov implementation of simulateTimestep subsequently calls the functions computeNumericalFluxes (to compute all fluxes on grid edges), and updateUnknowns (to update the variables according to flux values, typically according to an Euler time step).

Parameters

dt	size of the time step

Reimplemented from **SWE_Block** (p. 46).

9.23.3.7 void SWE_RusanovBlock::updateUnknowns (float *dt*) [virtual]

update unknowns according to fluxes (Euler time step)

implements interface function updateUnknowns: based on the (Rusanov) fluxes computed on each edge (and stored in the variables Fh, Gh, etc.); compute the balance terms for each cell, and update the unknowns according to an Euler time step.

Parameters

dt	size of the time step.

Implements **SWE_Block** (p. 47).

9.23.4 Friends And Related Function Documentation

9.23.4.1 ostream& operator<< (ostream & *os,* const SWE_RusanovBlock & *swe*) [friend]

overload operator<< such that data can be written via cout << -> needs to be declared as friend to be allowed to access private data

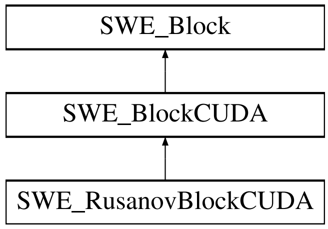
The documentation for this class was generated from the following files:

- src/blocks/rusanov/**SWE_RusanovBlock.hh**
- src/blocks/rusanov/**SWE_RusanovBlock.cpp**

9.24 SWE_RusanovBlockCUDA Class Reference

`#include <SWE_RusanovBlockCUDA.hh>`

Inheritance diagram for SWE_RusanovBlockCUDA:



Public Member Functions

- **SWE_RusanovBlockCUDA** (float _offsetX=0, float _offsetY=0, const int i_cudaDevice=0)
- virtual void **computeNumericalFluxes** ()

    compute the numerical fluxes for each edge of the Cartesian grid
- virtual void **updateUnknowns** (float dt)

    compute the new values of the unknowns h, hu, and hv in all grid cells
- virtual void **simulateTimestep** (float dt)

    execute a single time step of the simulation
- virtual float **simulate** (float tStart, float tEnd)

Friends

- ostream & **operator**<< (ostream &os, const **SWE_RusanovBlockCUDA** &swe)

Additional Inherited Members

9.24.1 Detailed Description

SWE_RusanovBlockCUDA (p. 56) extends the base class **SWE_BlockCUDA** (p. 48), and provides a concrete CUDA implementation of a simple shallow water model based on Rusanov Flux computation on the edges and explicit time stepping.

9.24.2 Member Function Documentation

9.24.2.1 virtual void SWE_RusanovBlockCUDA::computeNumericalFluxes ( ) [virtual]

compute the numerical fluxes for each edge of the Cartesian grid

The computation of fluxes strongly depends on the chosen numerical method. Hence, this purely virtual function has to be implemented in the respective derived classes.

Implements **SWE_Block** (p. 43).

9.24.2.2 virtual float SWE_RusanovBlockCUDA::simulate (float *i_tStart,* float *i_tEnd*) [virtual]

perform the simulation starting with simulation time tStart, until simulation time tEnd is reached

simulate implements the main simulation loop between two checkpoints; Note: this implementation can only be used, if you only use a single **SWE_Block** (p. 38) and only apply simple boundary conditions! In particular, **SWE_Block::simulate** (p. 45) can not trigger calls to exchange values of copy and ghost layers between blocks!

Parameters

tStart	time where the simulation is started
tEnd	time of the next checkpoint

Returns

actual end time reached

Reimplemented from **SWE_Block** (p. 45).

9.24.2.3 virtual void SWE_RusanovBlockCUDA::updateUnknowns (float *dt*) [virtual]

compute the new values of the unknowns h, hu, and hv in all grid cells

based on the numerical fluxes (computed by computeNumericalFluxes) and the specified time step size dt, an Euler time step is executed. As the computational fluxes will depend on the numerical method, this purely virtual function has to be implemented separately for each specific numerical model (and parallelisation approach).

Parameters

dt	size of the time step

Implements **SWE_Block** (p. 47).

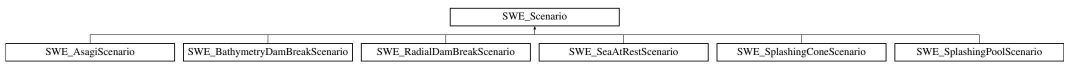
The documentation for this class was generated from the following file:

- src/blocks/rusanov/**SWE_RusanovBlockCUDA.hh**

9.25 SWE_Scenario Class Reference

```
#include <SWE_Scenario.hh>
```

Inheritance diagram for SWE_Scenario:



Public Member Functions

- virtual float **getWaterHeight** (float x, float y)
- virtual float **getVeloc_u** (float x, float y)
- virtual float **getVeloc_v** (float x, float y)
- virtual float **getBathymetry** (float x, float y)
- virtual float **waterHeightAtRest** ()
- virtual float **endSimulation** ()
- virtual **BoundaryType getBoundaryType** (**BoundaryEdge** edge)
- virtual float **getBoundaryPos** (**BoundaryEdge** edge)

9.25.1 Detailed Description

SWE_Scenario (p. 58) defines an interface to initialise the unknowns of a shallow water simulation - i.e. to initialise water height, velocities, and bathymatry according to certain scenarios. **SWE_Scenario** (p. 58) can act as stand-alone scenario class, providing a very basic scenario (all functions are constant); however, the idea is to provide derived classes that implement the **SWE_Scenario** (p. 58) interface for more interesting scenarios.

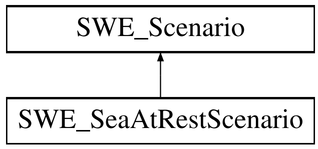
The documentation for this class was generated from the following file:

- src/scenarios/**SWE_Scenario.hh**

9.26 SWE_SeaAtRestScenario Class Reference

```
#include <SWE_simple_scenarios.hh>
```

Inheritance diagram for SWE_SeaAtRestScenario:



Public Member Functions

- float **getWaterHeight** (float x, float y)
- float **getBathymetry** (float x, float y)

9.26.1 Detailed Description

Scenario "Sea at Rest": flat water surface ("sea at rest"), but non-uniform bathymetry (id. to "Bathymetry Dam Break") test scenario for "sea at rest"-solution

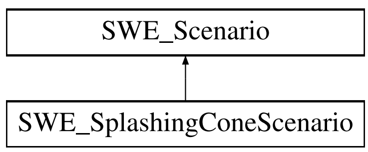
The documentation for this class was generated from the following file:

- src/scenarios/**SWE_simple_scenarios.hh**

9.27 SWE_SplashingConeScenario Class Reference

```
#include <SWE_simple_scenarios.hh>
```

Inheritance diagram for SWE_SplashingConeScenario:



Public Member Functions

- float **getWaterHeight** (float x, float y)
- float **getBathymetry** (float x, float y)
- float **waterHeightAtRest** ()
- float **endSimulation** ()
- virtual **BoundaryType getBoundaryType** (**BoundaryEdge** edge)

9.27.1 Detailed Description

Scenario "Splashing Cone": bathymetry forms a circular cone intial water surface designed to form "sea at rest" but: elevated water region in the centre (similar to radial dam break)

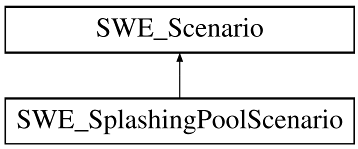
The documentation for this class was generated from the following file:

- src/scenarios/**SWE_simple_scenarios.hh**

9.28 SWE_SplashingPoolScenario Class Reference

#include <SWE_simple_scenarios.hh>

Inheritance diagram for SWE_SplashingPoolScenario:



Public Member Functions

- float **getBathymetry** (float x, float y)
- float **getWaterHeight** (float x, float y)
- virtual float **endSimulation** ()
- float **getBoundaryPos** (**BoundaryEdge** i_edge)

9.28.1 Detailed Description

Scenario "Splashing Pool": intial water surface has a fixed slope (diagonal to x,y)

9.28.2 Member Function Documentation

9.28.2.1 float SWE_SplashingPoolScenario::getBoundaryPos (BoundaryEdge *i_edge*) [inline], [virtual]

Get the boundary positions

Parameters

i_edge	which edge

Returns

value in the corresponding dimension

Reimplemented from **SWE_Scenario** (p. 58).

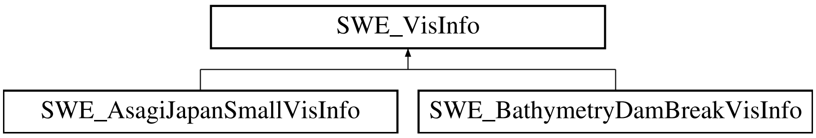
The documentation for this class was generated from the following file:

- src/scenarios/**SWE_simple_scenarios.hh**

9.29 SWE_VisInfo Class Reference

#include <SWE_VisInfo.hh>

Inheritance diagram for SWE_VisInfo:



Public Member Functions

- virtual ~**SWE_VisInfo** ()
- virtual float **waterVerticalScaling** ()
- virtual float **bathyVerticalOffset** ()
- virtual float **bathyVerticalScaling** ()

9.29.1 Detailed Description

SWE_VisInfo (p. 61) defines an interface that can be used for online visualization of a shallow water simulation. In particular, it provides information required for proper scaling of the involved variables.

For water height: displayedWaterHeight = **waterVerticalScaling()** (p. 62) $*$ simulatedWaterHeight

For bathymetry: displayedBatyhmetry = **bathyVerticalScaling()** (p. 61) $*$ realBathymetry

- **bathyVerticalOffset()** (p. 61)

The default water height should be 0. In this case a bathymetry value smaller than 0 means water and a value greater than 0 is land. Therefore bathyVerticalOffset should 0 for all real scenarios.

If you do not not provide an **SWE_VisInfo** (p. 61) for scenario, (water|bathy)VerticalScaling will be guessed form the value initial values. bathyVerticalOffset is always 0 in this case.

9.29.2 Constructor & Destructor Documentation

9.29.2.1 virtual SWE_VisInfo::~SWE_VisInfo ( ) `[inline]`, `[virtual]`

Empty virtual destructor

9.29.3 Member Function Documentation

9.29.3.1 virtual float SWE_VisInfo::bathyVerticalOffset ( ) `[inline]`, `[virtual]`

Returns

The vertical offset for the bathymetry. Should be 0 for "real" scenarios (scenarios with dry areas)

Reimplemented in **SWE_BathymetryDamBreakVisInfo** (p. 38).

9.29.3.2 virtual float SWE_VisInfo::bathyVerticalScaling ( ) `[inline]`, `[virtual]`

Returns

The vertical scaling factor for the bathymetry

Reimplemented in **SWE_AsagiJapanSmallVisInfo** (p. 33).

9.29.3.3 virtual float SWE_VisInfo::waterVerticalScaling ( ) [inline], [virtual]

Returns

The vertical scaling factor of the water

Reimplemented in **SWE_AsagiJapanSmallVisInfo** (p. 33).

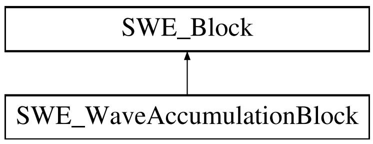
The documentation for this class was generated from the following file:

- src/scenarios/**SWE_VisInfo.hh**

9.30 SWE_WaveAccumulationBlock Class Reference

#include <SWE_WaveAccumulationBlock.hh>

Inheritance diagram for SWE_WaveAccumulationBlock:



Public Member Functions

- **SWE_WaveAccumulationBlock** (int l_nx, int l_ny, float l_dx, float l_dy)
- void **computeNumericalFluxes** ()
- void **updateUnknowns** (float dt)

Additional Inherited Members

9.30.1 Detailed Description

SWE_WaveAccumulationBlock (p. 62) is an implementation of the **SWE_Block** (p. 38) abstract class. It uses a wave propagation solver which is defined with the pre-compiler flag WAVE_PROPAGATION_SOLVER (see above).

Possible wave propagation solvers are: F-Wave, Apprximate Augmented Riemann, Hybrid (f-wave + augmented). (details can be found in the corresponding source files)

9.30.2 Constructor & Destructor Documentation

9.30.2.1 SWE_WaveAccumulationBlock::SWE_WaveAccumulationBlock (int *l_nx*, int *l_ny*, float *l_dx*, float *l_dy*)

Constructor of a **SWE_WaveAccumulationBlock** (p. 62).

Allocates the variables for the simulation: unknowns h,hu,hv,b are defined on grid indices [0,..,nx+1]∗[0,..,ny+1] (-> Abstract class **SWE_Block** (p. 38)) -> computational domain is [1,..,nx]∗[1,..,ny] -> plus ghost cell layer

Similar, all net-updates are defined as cell-local variables with indices [0,..,nx+1]∗[0,..,ny+1], however, only values on [1,..,nx]∗[1,..,ny] are used (i.e., ghost layers are not accessed). Net updates are intended to hold the accumulated(!) net updates computed on the edges.

9.30.3 Member Function Documentation

9.30.3.1 void SWE_WaveAccumulationBlock::computeNumericalFluxes ( ) `[virtual]`

Compute net updates for the block. The member variable **maxTimestep** (p. 48) will be updated with the maximum allowed time step size

Implements **SWE_Block** (p. 43).

9.30.3.2 void SWE_WaveAccumulationBlock::updateUnknowns (float *dt*) `[virtual]`

Updates the unknowns with the already computed net-updates.

Parameters

dt	time step width used in the update.

Implements **SWE_Block** (p. 47).

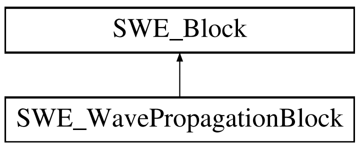
The documentation for this class was generated from the following files:

- src/blocks/**SWE_WaveAccumulationBlock.hh**
- src/blocks/**SWE_WaveAccumulationBlock.cpp**

9.31 SWE_WavePropagationBlock Class Reference

```
#include <SWE_WavePropagationBlock.hh>
```

Inheritance diagram for SWE_WavePropagationBlock:



Public Member Functions

- **SWE_WavePropagationBlock** (int l_nx, int l_ny, float l_dx, float l_dy)
- void **computeNumericalFluxes** ()
- void **updateUnknowns** (float dt)
- void **updateUnknownsRow** (float dt, int i)
- virtual ∼**SWE_WavePropagationBlock** ()

Additional Inherited Members

9.31.1 Detailed Description

SWE_WavePropagationBlock (p. 63) is an implementation of the **SWE_Block** (p. 38) abstract class. It uses a wave propagation solver which is defined with the pre-compiler flag WAVE_PROPAGATION_SOLVER (see above).

Possible wave propagation solvers are: F-Wave, Apprximate Augmented Riemann, Hybrid (f-wave + augmented). (details can be found in the corresponding source files)

9.31.2 Constructor & Destructor Documentation

9.31.2.1 SWE_WavePropagationBlock::SWE_WavePropagationBlock (int *l_nx,* int *l_ny,* float *l_dx,* float *l_dy*)

Constructor of a **SWE_WavePropagationBlock** (p. 63).

Allocates the variables for the simulation: unknowns h,hu,hv,b are defined on grid indices [0,..,nx+1]∗[0,..,ny+1] (-> Abstract class **SWE_Block** (p. 38)) -> computational domain is [1,..,nx]∗[1,..,ny] -> plus ghost cell layer

net-updates are defined for edges with indices [0,..,nx]∗[0,..,ny-1] or [0,..,nx-1]∗`0,..,ny`

A left/right net update with index (i-1,j-1) is located on the edge between cells with index (i-1,j) and (i,j):

```
*********************
*         *         *
* (i-1,j) *  (i,j)  *
*         *         *
*********************

          *
         ***
        *****
          *
          *
NetUpdatesLeft(i-1,j-1)
          or
NetUpdatesRight(i-1,j-1)
```

A below/above net update with index (i-1, j-1) is located on the edge between cells with index (i, j-1) and (i,j):

```
*         *
* (i, j)  *   *
*         *  **  NetUpdatesBelow(i-1,j-1)
*********** *****         or
*         *  **  NetUpdatesAbove(i-1,j-1)
* (i,j-1) *   *
*         *
```

9.31.2.2 virtual SWE_WavePropagationBlock::∼SWE_WavePropagationBlock ( ) `[inline]`, `[virtual]`

Destructor of a **SWE_WavePropagationBlock** (p. 63).

In the case of a hybrid solver (NDEBUG not defined) information about the used solvers will be printed.

9.31.3 Member Function Documentation

9.31.3.1 void SWE_WavePropagationBlock::computeNumericalFluxes ( ) `[virtual]`

Compute net updates for the block. The member variable **maxTimestep** (p. 48) will be updated with the maximum allowed time step size

Implements **SWE_Block** (p. 43).

9.31.3.2 void SWE_WavePropagationBlock::updateUnknowns (float *dt*) [virtual]

Updates the unknowns with the already computed net-updates.

Parameters

dt	time step width used in the update.

Implements **SWE_Block** (p. 47).

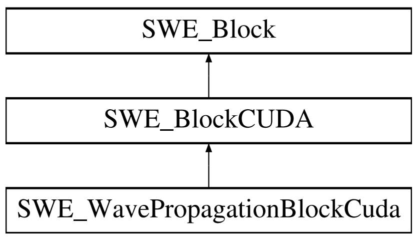
The documentation for this class was generated from the following files:

- src/blocks/**SWE_WavePropagationBlock.hh**
- src/blocks/**SWE_WavePropagationBlock.cpp**

9.32 SWE_WavePropagationBlockCuda Class Reference

`#include <SWE_WavePropagationBlockCuda.hh>`

Inheritance diagram for SWE_WavePropagationBlockCuda:



Public Member Functions

- **SWE_WavePropagationBlockCuda** (int l_nx, int l_ny, float l_dx, float l_dy)
- void **simulateTimestep** (float i_dT)

    execute a single time step (with fixed time step size) of the simulation
- float **simulate** (float, float)
- void **computeNumericalFluxes** ()

    compute the numerical fluxes for each edge of the Cartesian grid
- void **updateUnknowns** (const float i_deltaT)

    compute the new values of the unknowns h, hu, and hv in all grid cells

Additional Inherited Members

9.32.1 Detailed Description

SWE_WavePropagationBlockCuda (p. 65) is an implementation of the SWE_BlockCuda abstract class. It uses a wave propagation solver which is defined with the pre-compiler flag WAVE_PROPAGATION_SOLVER (see above).

Possible wave propagation solvers are: F-Wave, <strike>Approximate Augmented Riemann, Hybrid (f-wave + augmented).</strike> (details can be found in the corresponding source files)

9.32.2 Member Function Documentation

9.32.2.1 void SWE_WavePropagationBlockCuda::computeNumericalFluxes ( ) [virtual]

compute the numerical fluxes for each edge of the Cartesian grid

The computation of fluxes strongly depends on the chosen numerical method. Hence, this purely virtual function has to be implemented in the respective derived classes.

Implements **SWE_Block** (p. 43).

9.32.2.2 float SWE_WavePropagationBlockCuda::simulate (float *i_tStart*, float *i_tEnd*) [virtual]

perform the simulation starting with simulation time tStart, until simulation time tEnd is reached

simulate implements the main simulation loop between two checkpoints; Note: this implementation can only be used, if you only use a single **SWE_Block** (p. 38) and only apply simple boundary conditions! In particular, **SWE_Block::simulate** (p. 45) can not trigger calls to exchange values of copy and ghost layers between blocks!

Parameters

tStart	time where the simulation is started
tEnd	time of the next checkpoint

Returns

actual end time reached

Reimplemented from **SWE_Block** (p. 45).

9.32.2.3 void SWE_WavePropagationBlockCuda::simulateTimestep (float *dt*) [virtual]

execute a single time step (with fixed time step size) of the simulation

Executes a single timestep with fixed time step size

- compute net updates for every edge
- update cell values with the net updates

Parameters

dt	time step width of the update

Reimplemented from **SWE_Block** (p. 46).

9.32.2.4 void SWE_WavePropagationBlockCuda::updateUnknowns (const float *dt*) [virtual]

compute the new values of the unknowns h, hu, and hv in all grid cells

based on the numerical fluxes (computed by computeNumericalFluxes) and the specified time step size dt, an Euler time step is executed. As the computational fluxes will depend on the numerical method, this purely virtual function has to be implemented separately for each specific numerical model (and parallelisation approach).

Parameters

dt	size of the time step

Implements **SWE_Block** (p. 47).

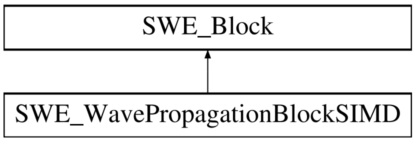
The documentation for this class was generated from the following file:

- src/blocks/cuda/**SWE_WavePropagationBlockCuda.hh**

9.33 SWE_WavePropagationBlockSIMD Class Reference

```
#include <SWE_WavePropagationBlockSIMD.hh>
```

Inheritance diagram for SWE_WavePropagationBlockSIMD:



Public Member Functions

- **SWE_WavePropagationBlockSIMD** (int l_nx, int l_ny, float l_dx, float l_dy)
- virtual void **simulateTimestep** (float dt)
- void **computeNumericalFluxes** ()
- void **updateUnknowns** (float dt)
- float **simulate** (float i_tStart, float i_tEnd)
- virtual ∼**SWE_WavePropagationBlockSIMD** ()

Additional Inherited Members

9.33.1 Detailed Description

SWE_WavePropagationBlockSIMD (p. 67) is an implementation of the **SWE_Block** (p. 38) abstract class. It uses a wave propagation solver which is defined with the pre-compiler flag WAVE_PROPAGATION_SOLVER (see above).

Possible wave propagation solvers are: F-Wave, Apprximate Augmented Riemann, Hybrid (f-wave + augmented). (details can be found in the corresponding source files)

9.33.2 Constructor & Destructor Documentation

9.33.2.1 SWE_WavePropagationBlockSIMD::SWE_WavePropagationBlockSIMD (int *l_nx*, int *l_ny*, float *l_dx*, float *l_dy*)

Constructor of a **SWE_WavePropagationBlockSIMD** (p. 67).

Allocates the variables for the simulation: unknowns h,hu,hv,b are defined on grid indices [0,..,nx+1]∗[0,..,ny+1] (-> Abstract class **SWE_Block** (p. 38)) -> computational domain is [1,..,nx]∗[1,..,ny] -> plus ghost cell layer

net-updates are defined for edges with indices [0,..,nx]∗[0,..,ny-1] or [0,..,nx-1]∗0,..,ny

A left/right net update with index (i-1,j-1) is located on the edge between cells with index (i-1,j) and (i,j):

```
********************
*         *        *
* (i-1,j) *  (i,j) *
*         *        *
********************

          *
         ***
        *****
          *
          *
NetUpdatesLeft(i-1,j-1)
          or
NetUpdatesRight(i-1,j-1)
```

A below/above net update with index (i-1, j-1) is located on the edge between cells with index (i, j-1) and (i,j):

```
*          *
* (i, j)   *   *
*          *  **  NetUpdatesBelow(i-1,j-1)
*********** *****         or
*          *  **  NetUpdatesAbove(i-1,j-1)
* (i,j-1)  *   *
*          *
```

9.33.2.2 virtual SWE_WavePropagationBlockSIMD::∼SWE_WavePropagationBlockSIMD ( ) [inline], [virtual]

Destructor of a **SWE_WavePropagationBlockSIMD** (p. 67).

In the case of a hybrid solver (NDEBUG not defined) information about the used solvers will be printed.

9.33.3 Member Function Documentation

9.33.3.1 void SWE_WavePropagationBlockSIMD::computeNumericalFluxes ( ) [virtual]

Compute net updates for the block. The member variable **maxTimestep** (p. 48) will be updated with the maximum allowed time step size

Implements **SWE_Block** (p. 43).

9.33.3.2 float SWE_WavePropagationBlockSIMD::simulate (float *i_tStart*, float *i_tEnd*) [virtual]

Runs the simulation until i_tEnd is reached.

Parameters

i_tStart	time when the simulation starts
i_tEnd	time when the simulation should end

Returns

time we reached after the last update step, in general a bit later than i_tEnd

Reimplemented from **SWE_Block** (p. 45).

9.33.3.3 void SWE_WavePropagationBlockSIMD::simulateTimestep (float *dt*) [virtual]

Update the bathymetry values with the displacement corresponding to the current time step.

Parameters

i_asagiScenario	the corresponding ASAGI-scenario Executes a single timestep. • compute net updates for every edge • update cell values with the net updates
dt	time step width of the update

Reimplemented from **SWE_Block** (p. 46).

9.33.3.4 void SWE_WavePropagationBlockSIMD::updateUnknowns (float *dt*) [virtual]

Updates the unknowns with the already computed net-updates.

Parameters

dt	time step width used in the update.

Implements **SWE_Block** (p. 47).

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

- src/blocks/**SWE_WavePropagationBlockSIMD.hh**
- src/blocks/**SWE_WavePropagationBlockSIMD.cpp**

9.34 Text Class Reference

Public Member Functions

- void **addText** (const char ∗text)
- void **startTextMode** ()
- bool **showNextText** (SDL_Rect &location)
- void **endTextMode** ()

9.34.1 Member Function Documentation

9.34.1.1 bool Text::showNextText (SDL_Rect & *location*) [inline]

Returns

True there are more textures

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

- src/opengl/text.h
- src/opengl/text.cpp

9.35 VBO Class Reference

Public Member Functions

- void **init** ()
- GLuint **getName** ()
- void **setBufferData** (GLsizei size, const void ∗data, GLenum target=GL_ARRAY_BUFFER, GLenum usage=GL_STATIC_DRAW)
- void **bindBuffer** (GLenum target=GL_ARRAY_BUFFER)
- void **finialize** ()

9.35.1 Member Function Documentation

9.35.1.1 void VBO::finialize ( ) [inline]

Frees all associated memory

9.35.1.2 GLuint VBO::getName ( ) [inline]

Returns

The OpenGL name of the buffer

9.35.1.3 void VBO::init ( )

Initializes the object

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

- src/opengl/**vbo.h**
- src/opengl/**vbo.cpp**

9.36 Visualization Class Reference

Public Member Functions

- **Visualization** (int windowWidth, int windowHeight, const char ∗window_title)
- ∼**Visualization** ()
- void **init** (**Simulation** &sim, **SWE_VisInfo** ∗visInfo=0L)
- void **cleanUp** ()
- cudaGraphicsResource ∗∗ **getCudaNormalsPtr** ()
- cudaGraphicsResource ∗∗ **getCudaWaterSurfacePtr** ()
- void **renderDisplay** ()
- void **modifyWaterScaling** (float factor)
- void **setRenderingMode** (RenderMode mode)
- void **toggleRenderingMode** ()
- int **resizeWindow** (int newWidth, int newHeight)

Static Public Member Functions

- static bool **isExtensionSupported** (const char ∗szTargetExtension)

Public Attributes

- **Camera** ∗ **camera**

9.36.1 Constructor & Destructor Documentation

9.36.1.1 Visualization::Visualization (int *windowWidth,* int *windowHeight,* const char ∗ *window_title*)

Constructor. All dimensions are node-based, this means a grid consisting of 2x2 cells would have 3x3 nodes.

Parameters

	window_title title of the window created
	_grid_x_size number of nodes of the grid (in x-direction)

	_grid_y_size number of nodes of the grid (in y-direction)

9.36.1.2 Visualization::∼Visualization ( )

Destructor (see note below)

9.36.2 Member Function Documentation

9.36.2.1 void Visualization::cleanUp ( )

Frees all memory we used for geometry data Needs to be called before destructor gets called in order to work correctly

9.36.2.2 cudaGraphicsResource ∗∗ Visualization::getCudaNormalsPtr ( )

Returns a pointer to the cuda memory object holding the vertex normals

9.36.2.3 cudaGraphicsResource ∗∗ Visualization::getCudaWaterSurfacePtr ( )

Returns a pointer to the cuda memory object holding the vertex positions

9.36.2.4 void Visualization::init (Simulation & *sim*, SWE_VisInfo ∗ *visInfo* = `0L`)

Allocates memory for vertices and other geometry data.

Parameters

sim	instance of the simulation class

9.36.2.5 bool Visualization::isExtensionSupported (const char ∗ *szTargetExtension*) `[static]`

Returns, whether a special extension is supported by the current graphics card

Parameters

szTarget-Extention	string describing the extension to look for

9.36.2.6 void Visualization::renderDisplay ( )

Main rendering function. Draws the scene and updates screen

9.36.2.7 int Visualization::resizeWindow (int *newWidth*, int *newHeight*)

Gets called when window gets resized

Parameters

newWidth	new window width in pixels
newHeight	height in pixels

9.36.2.8 void Visualization::setRenderingMode (RenderMode *mode*)

Sets current rendering mode

Parameters

mode	rendering mode

9.36.2.9 void Visualization::toggleRenderingMode ( )

Switches between 3 different rendering modes:

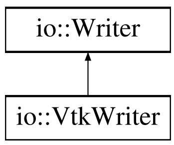
- Shaded: Use OpenGL shading
- Wireframe: Only render edges of each triangle
- Watershader: Use custom GLSL shader for water surface

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

- src/opengl/visualization.h
- src/opengl/visualization.cpp

9.37 io::VtkWriter Class Reference

Inheritance diagram for io::VtkWriter:



Public Member Functions

- **VtkWriter** (const std::string &i_fileName, const **Float2D** &i_b, const **BoundarySize** &i_boundarySize, int i_nX, int i_nY, float i_dX, float i_dY, int i_offsetX=0, int i_offsetY=0)
- void **writeTimeStep** (const **Float2D** &i_h, const **Float2D** &i_hu, const **Float2D** &i_hv, float i_time)

Additional Inherited Members

9.37.1 Constructor & Destructor Documentation

9.37.1.1 io::VtkWriter::VtkWriter (const std::string & *i_baseName*, const Float2D & *i_b*, const BoundarySize & *i_boundarySize*, int *i_nX*, int *i_nY*, float *i_dX*, float *i_dY*, int *i_offsetX* = 0, int *i_offsetY* = 0)

Creates a vtk file for each time step. Any existing file will be replaced.

Parameters

i_baseName	base name of the netCDF-file to which the data will be written to.
i_nX	number of cells in the horizontal direction.
i_nY	number of cells in the vertical direction.

i_dX	cell size in x-direction.
i_dY	cell size in y-direction.
i_offsetX	x-offset of the block
i_offsetY	y-offset of the block
i_dynamic-Bathymetry	

Todo This version can only handle a boundary layer of size 1

9.37.2 Member Function Documentation

9.37.2.1 void io::VtkWriter::writeTimeStep (const Float2D & *i_h,* const Float2D & *i_hu,* const Float2D & *i_hv,* float *i_time*) [virtual]

Writes one time step

Parameters

i_h	water heights at a given time step.
i_hu	momentums in x-direction at a given time step.
i_hv	momentums in y-direction at a given time step.
i_time	simulation time of the time step.

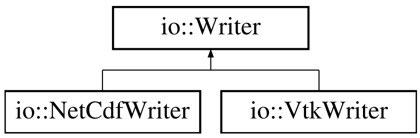
Implements **io::Writer** (p. 74).

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

- src/writer/**VtkWriter.hh**
- src/writer/**VtkWriter.cpp**

9.38 io::Writer Class Reference

Inheritance diagram for io::Writer:



Public Member Functions

- **Writer** (const std::string &i_fileName, const **Float2D** &i_b, const **BoundarySize** &i_boundarySize, int i_nX, int i_nY)
- virtual void **writeTimeStep** (const **Float2D** &i_h, const **Float2D** &i_hu, const **Float2D** &i_hv, float i_time)=0

Protected Attributes

- const std::string **fileName**

  file name of the data file
- const **Float2D** & **b**

  (Reference) to bathymetry data
- const **BoundarySize boundarySize**

Boundary layer size.

- const unsigned int **nX**

  dimensions of the grid in x- and y-direction.

- const unsigned int **nY**
- size_t **timeStep**

  current time step

9.38.1 Constructor & Destructor Documentation

9.38.1.1 io::Writer::Writer (const std::string & *i_fileName*, const **Float2D** & *i_b*, const **BoundarySize** & *i_boundarySize*, int *i_nX*, int *i_nY*) [inline]

Parameters

i_boundarySize	size of the boundaries.

9.38.2 Member Function Documentation

9.38.2.1 virtual void io::Writer::writeTimeStep (const **Float2D** & *i_h*, const **Float2D** & *i_hu*, const **Float2D** & *i_hv*, float *i_time*) [pure virtual]

Writes one time step

Parameters

i_h	water heights at a given time step.
i_hu	momentums in x-direction at a given time step.
i_hv	momentums in y-direction at a given time step.
i_time	simulation time of the time step.

Implemented in **io::NetCdfWriter** (p. 30), and **io::VtkWriter** (p. 73).

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

- src/writer/**Writer.hh**

Chapter 10

File Documentation

10.1 src/blocks/cuda/SWE_BlockCUDA.hh File Reference

```
#include "blocks/SWE_Block.hh"
#include "tools/help.hh"
#include <iostream>
#include <fstream>
#include <cuda_runtime.h>
```

Classes

- class **SWE_BlockCUDA**

Functions

- void **checkCUDAError** (const char *msg)
- void **tryCUDA** (cudaError_t err, const char *msg)
- __device__ int **getCellCoord** (int x, int y, int ny)
- __device__ int **getEdgeCoord** (int x, int y, int ny)
- __device__ int **getBathyCoord** (int x, int y, int ny)

Variables

- const int **TILE_SIZE** =16

10.1.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel
Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.`)

10.1.2 LICENSE

10.1.3 DESCRIPTION

TODO

10.1.4 Function Documentation

10.1.4.1 __device__ int getBathyCoord (int *x*, int *y*, int *ny*) `[inline]`

Return index of a specific element in the arrays of bathymetry source terms

Parameters

i,j	x- and y-coordinate of grid cell
ny	grid size in y-direction (without ghost layers)

10.1.4.2 __device__ int getCellCoord (int *x*, int *y*, int *ny*) `[inline]`

Return index of hd[i][j] in linearised array

Parameters

i,j	x- and y-coordinate of grid cell
ny	grid size in y-direction (without ghost layers)

10.1.4.3 __device__ int getEdgeCoord (int *x*, int *y*, int *ny*) `[inline]`

Return index of edge-data Fhd[i][j] or Ghd[i][j] in linearised array

Parameters

i,j	x- and y-coordinate of grid cell
ny	grid size in y-direction (without ghost layers)

10.2 src/blocks/cuda/SWE_BlockCUDA_kernels.hh File Reference

Functions

- __global__ void **kernelHdBufferEdges** (float ∗hd, int nx, int ny)
- __global__ void **kernelMaximum** (float ∗maxhd, float ∗maxvd, int start, int size)
- __global__ void **kernelLeftBoundary** (float ∗hd, float ∗hud, float ∗hvd, int nx, int ny, **BoundaryType** bound)
- __global__ void **kernelRightBoundary** (float ∗hd, float ∗hud, float ∗hvd, int nx, int ny, **BoundaryType** bound)

- __global__ void **kernelBottomBoundary** (float ∗hd, float ∗hud, float ∗hvd, int nx, int ny, **BoundaryType** bound)
- __global__ void **kernelTopBoundary** (float ∗hd, float ∗hud, float ∗hvd, int nx, int ny, **BoundaryType** bound)
- __global__ void **kernelBottomGhostBoundary** (float ∗hd, float ∗hud, float ∗hvd, float ∗bottomGhostLayer, int nx, int ny)
- __global__ void **kernelTopGhostBoundary** (float ∗hd, float ∗hud, float ∗hvd, float ∗topGhostLayer, int nx, int ny)
- __global__ void **kernelBottomCopyLayer** (float ∗hd, float ∗hud, float ∗hvd, float ∗bottomCopyLayer, int nx, int ny)
- __global__ void **kernelTopCopyLayer** (float ∗hd, float ∗hud, float ∗hvd, float ∗topCopyLayer, int nx, int ny)

10.2.1 Detailed Description

This file is part of SWE.

Author

Michael Bader (bader AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Univ.--Prof._Dr._Michael_Bader`)

10.2.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see `http://www.gnu.org/licenses/`.

10.2.3 DESCRIPTION

TODO

10.3 src/blocks/cuda/SWE_WavePropagationBlockCuda.hh File Reference

```
#include <cassert>
#include "SWE_BlockCUDA.hh"
```

Classes

- class **SWE_WavePropagationBlockCuda**

10.3.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer)
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)

10.3.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see http://www.-gnu.org/licenses/.

10.3.3 DESCRIPTION

SWE_Block (p. 38) in CUDA, which uses solvers in the wave propagation formulation.

10.4 src/blocks/cuda/SWE_WavePropagationBlockCuda_kernels.hh File Reference

Functions

- __global__ void **computeNetUpdatesKernel** (const float ∗i_h, const float ∗i_hu, const float ∗i_hv, const float ∗i_b, float ∗o_hNetUpdatesLeftD, float ∗o_hNetUpdatesRightD, float ∗o_huNetUpdatesLeftD, float ∗o_huNetUpdatesRightD, float ∗o_hNetUpdatesBelowD, float ∗o_hNetUpdatesAboveD, float ∗o_hvNetUpdatesBelowD, float ∗o_hvNetUpdatesAboveD, float ∗o_maximumWaveSpeeds, const int i_nx, const int i_ny, const int i_offsetX=0, const int i_offsetY=0, const int i_blockOffSetX=0, const int i_blockOffSetY=0)
- __global__ void **updateUnknownsKernel** (const float ∗i_hNetUpdatesLeftD, const float ∗i_hNetUpdatesRightD, const float ∗i_huNetUpdatesLeftD, const float ∗i_huNetUpdatesRightD, const float ∗i_hNetUpdatesBelowD, const float ∗i_hNetUpdatesAboveD, const float ∗i_hvNetUpdatesBelowD, const float ∗i_hvNetUpdatesAboveD, float ∗io_h, float ∗io_hu, float ∗io_hv, const float i_updateWidthX, const float i_updateWidthY, const int i_nx, const int i_ny)
- __device__ int **computeOneDPositionKernel** (const int i_i, const int i_j, const int i_nx)

10.4.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer)

10.4.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

10.4.3 DESCRIPTION

CUDA Kernels for a **SWE_Block** (p. 38), which uses solvers in the wave propagation formulation.

10.5 src/blocks/rusanov/SWE_RusanovBlock.cpp File Reference

```
#include "SWE_RusanovBlock.hh"
#include <math.h>
```

Functions

- ostream & **operator**<< (ostream &os, const **SWE_RusanovBlock** &swe)

10.5.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

10.5.2 LICENSE

10.5.3 DESCRIPTION

TODO

10.5.4 Function Documentation

10.5.4.1 ostream& operator<< (ostream & *os*, const SWE_RusanovBlock & *swe*)

overload operator<< such that data can be written via cout << -> needs to be declared as friend to be allowed to access private data

10.6 src/blocks/rusanov/SWE_RusanovBlock.hh File Reference

```
#include <iostream>
#include <stdio.h>
#include <fstream>
#include "tools/help.hh"
#include "SWE_Block.hh"
```

Classes

- class **SWE_RusanovBlock**

Functions

- ostream & **operator**<< (ostream &os, const **SWE_RusanovBlock** &swe)

10.6.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

10.6.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see `http://www.-gnu.org/licenses/`.

10.6.3 DESCRIPTION

TODO

10.6.4 Function Documentation

10.6.4.1 ostream& operator<< (ostream & *os*, const SWE_RusanovBlock & *swe*)

overload operator<< such that data can be written via cout << -> needs to be declared as friend to be allowed to access private data

10.7 src/blocks/rusanov/SWE_RusanovBlockCUDA.hh File Reference

```
#include <iostream>
#include <stdio.h>
#include <fstream>
#include <cuda_runtime.h>
#include "tools/help.hh"
#include "SWE_Block.hh"
#include "SWE_BlockCUDA.hh"
```

Classes

- class **SWE_RusanovBlockCUDA**

Functions

- ostream & **operator**<< (ostream &os, const **SWE_RusanovBlockCUDA** &swe)

10.7.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

10.7.2 LICENSE

10.7.3 DESCRIPTION

TODO

10.8 src/blocks/rusanov/SWE_RusanovBlockCUDA_kernels.hh File Reference

Functions

- __global__ void **kernelComputeFluxesF** (float ∗hd, float ∗hud, float ∗hvd, float ∗Fhd, float ∗Fhud, float ∗Fhvd, int ny, float g, float llf, int istart)
- __global__ void **kernelComputeFluxesG** (float ∗hd, float ∗hud, float ∗hvd, float ∗Ghd, float ∗Ghud, float ∗Ghvd, int ny, float g, float llf, int jstart)
- __global__ void **kernelComputeBathymetrySources** (float ∗hd, float ∗bd, float ∗Bxd, float ∗Byd, int ny, float g)

- __global__ void **kernelEulerTimestep** (float ∗hd, float ∗hud, float ∗hvd, float ∗Fhd, float ∗Fhud, float ∗Fhvd, float ∗Ghd, float ∗Ghud, float ∗Ghvd, float ∗Bxd, float ∗Byd, float ∗maxhd, float ∗maxvd, int nx, int ny, float dt, float dxi, float dyi)
- __global__ void **kernelMaximum** (float ∗maxhd, float ∗maxvd, int start, int size)

10.8.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

10.8.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see `http://www.gnu.org/licenses/`.

10.8.3 DESCRIPTION

TODO

10.9 src/blocks/SWE_Block.cpp File Reference

```
#include "SWE_Block.hh"
#include "tools/help.hh"
#include "solvers/fsolver.cpp"
#include <cmath>
#include <iostream>
#include <cassert>
#include <limits>
```

10.9.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel
Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Sebastian_Rettenberger,_M.Sc.`)

10.9.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

10.9.3 DESCRIPTION

TODO

10.10 src/blocks/SWE_Block.hh File Reference

```
#include "tools/help.hh"
#include "scenarios/SWE_Scenario.hh"
#include <iostream>
#include <fstream>
```

Classes

- class **SWE_Block**
- struct **SWE_Block1D**
- class **SWE_DimensionalSplitting**

10.10.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Sebastian_Rettenberger,_M.Sc.)

10.10.2 LICENSE

10.10.3 DESCRIPTION

TODO

10.11 src/blocks/SWE_WaveAccumulationBlock.cpp File Reference

```
#include "SWE_WaveAccumulationBlock.hh"
#include <cassert>
#include <string>
#include <limits>
```

10.11.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer)
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)
Michael Bader (bader AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Michael_-Bader)

10.11.2 LICENSE

10.11.3 DESCRIPTION

SWE_Block (p. 38), which uses solvers in the wave propagation formulation.

10.12 src/blocks/SWE_WaveAccumulationBlock.hh File Reference

```
#include "blocks/SWE_Block.hh"
#include "tools/help.hh"
#include <string>
```

Classes

- class **SWE_WaveAccumulationBlock**

10.12.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer)
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)
Michael Bader (bader AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Michael_-Bader)

10.12.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see http://www.-gnu.org/licenses/.

10.12.3 DESCRIPTION

SWE_Block (p. 38), which uses solvers in the wave propagation formulation.

10.13 src/blocks/SWE_WavePropagationBlock.cpp File Reference

```
#include "SWE_WavePropagationBlock.hh"
#include <cassert>
#include <string>
#include <limits>
```

10.13.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer)
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)
Michael Bader (bader AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Michael_-Bader)

10.13.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see http://www.-gnu.org/licenses/.

10.13.3 DESCRIPTION

Implementation of **SWE_Block** (p. 38) that uses solvers in the wave propagation formulation.

10.14 src/blocks/SWE_WavePropagationBlock.hh File Reference

```
#include "blocks/SWE_Block.hh"
#include "tools/help.hh"
#include <string>
#include "solvers/Hybrid.hpp"
```

Classes

- class **SWE_WavePropagationBlock**

10.14.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer)
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)
Michael Bader (bader AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Michael_-Bader)

10.14.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see http://www.-gnu.org/licenses/.

10.14.3 DESCRIPTION

Implementation of **SWE_Block** (p. 38) that uses solvers in the wave propagation formulation.

10.15 src/blocks/SWE_WavePropagationBlockSIMD.cpp File Reference

```
#include "SWE_WavePropagationBlockSIMD.hh"
#include <cassert>
#include <string>
#include <limits>
```

10.15.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer`)
Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.`)

10.15.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see `http://www.-gnu.org/licenses/`.

10.15.3 DESCRIPTION

SWE_Block (p. 38), which uses solvers in the wave propagation formulation.

10.16 src/blocks/SWE_WavePropagationBlockSIMD.hh File Reference

```
#include "blocks/SWE_Block.hh"
#include "tools/help.hh"
#include <string>
#include "solvers/Hybrid.hpp"
```

Classes

- class **SWE_WavePropagationBlockSIMD**

10.16.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer`)
Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.`)

10.16.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

10.16.3 DESCRIPTION

SWE_Block (p. 38), which uses solvers in the wave propagation formulation.

10.17 src/examples/swe_mpi.cpp File Reference

```
#include <algorithm>
#include <cassert>
#include <cmath>
#include <cstdlib>
#include <mpi.h>
#include <string>
#include <vector>
#include "blocks/SWE_WavePropagationBlock.hh"
#include "blocks/SWE_WaveAccumulationBlock.hh"
#include "writer/VtkWriter.hh"
#include "scenarios/SWE_simple_scenarios.hh"
#include "tools/args.hh"
#include "tools/help.hh"
#include "tools/Logger.hh"
#include "tools/ProgressBar.hh"
```

Functions

- int **computeNumberOfBlockRows** (int i_numberOfProcesses)
- void **exchangeLeftRightGhostLayers** (const int i_leftNeighborRank, **SWE_Block1D** ∗o_leftInflow, **SWE_-Block1D** ∗i_leftOutflow, const int i_rightNeighborRank, **SWE_Block1D** ∗o_rightInflow, **SWE_Block1D** ∗i_-rightOutflow, MPI_Datatype i_mpiCol)
- void **exchangeBottomTopGhostLayers** (const int i_bottomNeighborRank, **SWE_Block1D** ∗o_bottom-NeighborInflow, **SWE_Block1D** ∗i_bottomNeighborOutflow, const int i_topNeighborRank, **SWE_Block1D** ∗o_topNeighborInflow, **SWE_Block1D** ∗i_topNeighborOutflow, const MPI_Datatype i_mpiRow)
- int **main** (int argc, char ∗∗argv)

10.17.1 Detailed Description

This file is part of SWE.

Author

Michael Bader (bader AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Univ.--Prof._Dr._Michael_Bader)
Alexander Breuer (breuera AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer)
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)

10.17.2 LICENSE

10.17.3 DESCRIPTION

Setting of SWE, which uses a wave propagation solver and an artificial or ASAGI scenario on multiple blocks.

10.17.4 Function Documentation

10.17.4.1 int computeNumberOfBlockRows (int *i_numberOfProcesses*)

Compute the number of block rows from the total number of processes.

The number of rows is determined as the square root of the number of processes, if this is a square number; otherwise, we use the largest number that is smaller than the square root and still a divisor of the number of processes.

Parameters

numProcs	number of process.

Returns

number of block rows

10.17.4.2 void exchangeBottomTopGhostLayers (const int *i_bottomNeighborRank,* SWE_Block1D * *o_bottomNeighborInflow,* SWE_Block1D * *i_bottomNeighborOutflow,* const int *i_topNeighborRank,* SWE_Block1D * *o_topNeighborInflow,* SWE_Block1D * *i_topNeighborOutflow,* const MPI_Datatype *i_mpiRow*)

Exchanges the bottom and top ghost layers with MPI's SendReceive.

Parameters

i_bottom-NeighborRank	MPI rank of the bottom neighbor.
o_bottom-NeighborInflow	ghost layer, where the bottom neighbor writes into.
i_bottom-NeighborOutflow	host layer, where the bottom neighbor reads from.
i_topNeighbor-Rank	MPI rank of the top neighbor.
o_topNeighbor-Inflow	ghost layer, where the top neighbor writes into.
i_topNeighbor-Outflow	ghost layer, where the top neighbor reads from.

i_mpiRow	MPI data type for the horizontal ghost layers.

10.17.4.3 void exchangeLeftRightGhostLayers (const int *i_leftNeighborRank*, SWE_Block1D ∗ *o_leftInflow*, SWE_Block1D ∗ *i_leftOutflow*, const int *i_rightNeighborRank*, SWE_Block1D ∗ *o_rightInflow*, SWE_Block1D ∗ *i_rightOutflow*, MPI_Datatype *i_mpiCol*)

Exchanges the left and right ghost layers with MPI's SendReceive.

Parameters

i_leftNeighbor-Rank	MPI rank of the left neighbor.
o_leftInflow	ghost layer, where the left neighbor writes into.
i_leftOutflow	layer where the left neighbor reads from.
i_rightNeighbor-Rank	MPI rank of the right neighbor.
o_rightInflow	ghost layer, where the right neighbor writes into.
i_rightOutflow	layer, where the right neighbor reads form.
i_mpiCol	MPI data type for the vertical gost layers.

10.17.4.4 int main (int *argc*, char ∗∗ *argv*)

Main program for the simulation on a single **SWE_WavePropagationBlock** (p. 63) or **SWE_WaveAccumulation-Block** (p. 62). Initialization.

MPI Rank of a process.

number of MPI processes.

total number of grid cell in x- and y-direction.

l_baseName of the plots.

number of SWE_Blocks in x- and y-direction.

local position of each MPI process in x- and y-direction.

number of checkpoints for visualization (at each checkpoint in time, an output file is written).

number of grid cells in x- and y-direction per process.

size of a single cell in x- and y-direction

origin of the simulation domain in x- and y-direction

time when the simulation ends.

checkpoints when output files are written.

MPI row-vector: l_nXLocal+2 blocks, 1 element per block, stride of l_nYLocal+2

MPI row-vector: 1 block, l_nYLocal+2 elements per block, stride of 1

MPI ranks of the neighbors

Simulation (p. 32).

simulation time.

maximum allowed time step width within a block.

maximum allowed time steps of all blocks

Finalize.

10.18 src/examples/swe_simple.cpp File Reference

```
#include <cassert>
#include <cstdlib>
#include <string>
#include <iostream>
#include "blocks/SWE_WavePropagationBlock.hh"
#include "writer/VtkWriter.hh"
#include "scenarios/SWE_simple_scenarios.hh"
#include "tools/args.hh"
#include "tools/help.hh"
#include "tools/Logger.hh"
#include "tools/ProgressBar.hh"
```

Functions

- int **main** (int argc, char **argv)

10.18.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer) Michael Bader (bader AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Univ.-Prof._Dr._Michael_Bader)

10.18.2 LICENSE

10.18.3 DESCRIPTION

Basic setting of SWE, which uses a wave propagation solver and an artificial or ASAGI scenario on a single block.

10.18.4 Function Documentation

10.18.4.1 int main (int *argc*, char ** *argv*)

Main program for the simulation on a single **SWE_WavePropagationBlock** (p. 63). Initialization.

number of grid cells in x- and y-direction.

l_baseName of the plots.

number of checkpoints for visualization (at each checkpoint in time, an output file is written).

size of a single cell in x- and y-direction

origin of the simulation domain in x- and y-direction

time when the simulation ends.

checkpoints when output files are written.

Simulation (p. 32).

simulation time.

maximum allowed time step width.

Finalize.

10.19 src/opengl/vbo.cpp File Reference

```
#include "vbo.h"
#include "visualization.h"
```

10.19.1 Detailed Description

This file is part of SWE.

Author

Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)

10.19.2 LICENSE

10.20 src/opengl/vbo.h File Reference

```
#include "tools/Logger.hh"
#include <SDL/SDL_opengl.h>
```

Classes

- class **VBO**

10.20.1 Detailed Description

This file is part of SWE.

Author

Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)

10.20.2 LICENSE

10.20.3 DESCRIPTION

Handles a VertexBufferObject.

10.21 src/scenarios/SWE_AsagiScenario.cpp File Reference

```
#include "SWE_AsagiScenario.hh"
```

10.21.1 Detailed Description

This file is part of SWE.

Author

Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)

10.21.2 LICENSE

10.22 src/scenarios/SWE_AsagiScenario.hh File Reference

```
#include <cassert>
```

```
#include <cstring>
#include <string>
#include <iostream>
#include <map>
#include <asagi.h>
#include "SWE_Scenario.hh"
```

Classes

- class **SWE_AsagiGrid**
- class **SWE_AsagiScenario**

10.22.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Dipl.-Math._Alexander_Breuer`)
Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Sebastian_Rettenberger,_M.Sc.`)

10.22.2 LICENSE

10.22.3 DESCRIPTION

Access to bathymetry and displacement files with ASAGI.

10.23 src/scenarios/SWE_AsagiScenario_vis.hh File Reference

```
#include "SWE_VisInfo.hh"
```

Classes

- class **SWE_AsagiJapanSmallVisInfo**

10.23.1 Detailed Description

This file is part of SWE.

Author

Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)

10.23.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see http://www.-gnu.org/licenses/.

10.23.3 DESCRIPTION

Rescale water height in small Japan scenario

10.24 src/scenarios/SWE_Scenario.hh File Reference

Classes

- class **SWE_Scenario**

Typedefs

- typedef enum **BoundaryType BoundaryType**
- typedef enum **BoundaryEdge BoundaryEdge**

Enumerations

- enum **BoundaryType** {
  OUTFLOW, **WALL**, **INFLOW**, **CONNECT**,
  PASSIVE }
- enum **BoundaryEdge** { **BND_LEFT**, **BND_RIGHT**, **BND_BOTTOM**, **BND_TOP** }

10.24.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

10.24.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

10.24.3 DESCRIPTION

TODO

10.24.4 Typedef Documentation

10.24.4.1 typedef enum BoundaryEdge BoundaryEdge

enum type: numbering of the boundary edges

10.24.4.2 typedef enum BoundaryType BoundaryType

enum type: available types of boundary conditions

10.24.5 Enumeration Type Documentation

10.24.5.1 enum BoundaryEdge

enum type: numbering of the boundary edges

10.24.5.2 enum BoundaryType

enum type: available types of boundary conditions

10.25 src/scenarios/SWE_simple_scenarios.hh File Reference

```
#include <cmath>
#include "SWE_Scenario.hh"
```

Classes

- class **SWE_RadialDamBreakScenario**
- class **SWE_BathymetryDamBreakScenario**
- class **SWE_SeaAtRestScenario**
- class **SWE_SplashingPoolScenario**
- class **SWE_SplashingConeScenario**

10.25.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel
Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Sebastian_Rettenberger,_M.Sc.`)

10.25.2 LICENSE

10.25.3 DESCRIPTION

TODO

10.26 src/scenarios/SWE_VisInfo.hh File Reference

```
#include "SWE_Scenario.hh"
```

Classes

- class **SWE_VisInfo**

10.26.1 Detailed Description

This file is part of SWE.

Author

Michael Bader
Kaveh Rahnema
Tobias Schnabel
Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Sebastian_Rettenberger,_M.Sc.`)

10.26.2 LICENSE

10.26.3 DESCRIPTION

TODO

10.27 src/tools/args.hh File Reference

```
#include <getopt.h>
#include <algorithm>
#include <map>
#include <iomanip>
#include <string>
#include <sstream>
#include <vector>
```

Classes

- class **tools::Args**

10.27.1 Detailed Description

This file is part of SWE.

Author

Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)

10.27.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see http://www.-gnu.org/licenses/.

10.27.3 DESCRIPTION

Command line argument parser

10.28 src/tools/help.hh File Reference

```
#include <cstring>
#include <iostream>
#include <fstream>
#include <sstream>
```

Classes

- class **Float1D**
- class **Float2D**

Functions

- std::string **generateFileName** (std::string baseName, int timeStep)
- std::string **generateFileName** (std::string i_baseName, int i_blockPositionX, int i_blockPositionY, std::string i_fileExtension=".nc")
- std::string **generateFileName** (std::string baseName, int timeStep, int block_X, int block_Y, std::string i_file-Extension=".vts")
- std::string **generateBaseFileName** (std::string &i_baseName, int i_blockPositionX, int i_blockPositionY)
- std::string **generateContainerFileName** (std::string baseName, int timeStep)

10.28.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema
Sebastian Rettenberger

10.28.2 LICENSE

10.28.3 DESCRIPTION

TODO

10.28.4 Function Documentation

10.28.4.1 std::string generateBaseFileName (std::string & *i_baseName,* int *i_blockPositionX,* int *i_blockPositionY*) `[inline]`

Generates an output file name for a multiple **SWE_Block** (p. 38) version based on the ordering of the blocks.

Parameters

i_baseName	base name of the output.
i_blockPositionX	position of the **SWE_Block** (p. 38) in x-direction.

i_blockPositionY	position of the **SWE_Block** (p. 38) in y-direction.

Returns

the output filename **without** timestep information and file extension

10.28.4.2 std::string generateContainerFileName (std::string *baseName*, int *timeStep*) [inline]

generate output filename for the ParaView-Container-File (to visualize multiple SWE_Blocks per checkpoint)

10.28.4.3 std::string generateFileName (std::string *baseName*, int *timeStep*) [inline]

generate output filenames for the single-SWE_Block version (for serial and OpenMP-parallelised versions that use only a single **SWE_Block** (p. 38) - one output file is generated per checkpoint)

Deprecated

10.28.4.4 std::string generateFileName (std::string *i_baseName*, int *i_blockPositionX*, int *i_blockPositionY*, std::string *i_fileExtension* = ".nc") [inline]

Generates an output file name for a multiple **SWE_Block** (p. 38) version based on the ordering of the blocks.

Parameters

i_baseName	base name of the output.
i_blockPositionX	position of the **SWE_Block** (p. 38) in x-direction.
i_blockPositionY	position of the **SWE_Block** (p. 38) in y-direction.
i_fileExtension	file extension of the output file.

Returns

Deprecated

10.28.4.5 std::string generateFileName (std::string *baseName*, int *timeStep*, int *block_X*, int *block_Y*, std::string *i_fileExtension* = ".vts") [inline]

generate output filename for the multiple-SWE_Block version (for serial and parallel (OpenMP and MPI) versions that use multiple SWE_Blocks - for each block, one output file is generated per checkpoint)

Deprecated

10.29 src/tools/Logger.cpp File Reference

```
#include "Logger.hh"
```

10.29.1 Detailed Description

This file is part of SWE.

Author

Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.`)

10.29.2 LICENSE

10.30 src/tools/Logger.hh File Reference

```
#include <map>
#include <string>
#include <iostream>
#include <ctime>
```

Classes

- class **tools::Logger**

10.30.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer`)
Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.`)

10.30.2 LICENSE

10.30.3 DESCRIPTION

Collection of basic logging routines.

10.31 src/tools/ProgressBar.hh File Reference

```
#include <cassert>
#include <cmath>
#include <ctime>
#include <algorithm>
#include <iostream>
#include <limits>
#include <unistd.h>
#include <sys/ioctl.h>
```

Classes

- class **tools::ProgressBar**

10.31.1 Detailed Description

This file is part of SWE.

Author

Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.`)

10.31.2 LICENSE

10.31.3 DESCRIPTION

A simple progress bar using stdout

10.32 src/writer/NetCdfWriter.cpp File Reference

```
#include "NetCdfWriter.hh"
#include <string>
#include <vector>
#include <iostream>
#include <cassert>
```

10.32.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer`)
Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.`)

10.32.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see `http://www.-gnu.org/licenses/`.

10.32.3 DESCRIPTION

A writer for the netCDF-format: `http://www.unidata.ucar.edu/software/netcdf/`

10.33 src/writer/NetCdfWriter.hh File Reference

```
#include <cstring>
#include <string>
#include <vector>
#include <netcdf.h>
#include "writer/Writer.hh"
```

Classes

- class **io::NetCdfWriter**

10.33.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Dipl.--Math._Alexander_Breuer`)
Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.`)

10.33.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

10.33.3 DESCRIPTION

A writer for the netCDF-format: `http://www.unidata.ucar.edu/software/netcdf/`

10.34 src/writer/VtkWriter.cpp File Reference

```
#include <cassert>
#include <fstream>
#include "VtkWriter.hh"
```

10.34.1 Detailed Description

This file is part of SWE.

Author

Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/Sebastian_Rettenberger,_M.Sc.`)

10.34.2 LICENSE

10.34.3 DESCRIPTION

10.35 src/writer/VtkWriter.hh File Reference

```
#include <sstream>
#include "writer/Writer.hh"
```

Classes

- class **io::VtkWriter**

10.35.1 Detailed Description

This file is part of SWE.

Author

Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.`)

10.35.2 LICENSE

10.35.3 DESCRIPTION

10.36 src/writer/Writer.hh File Reference

```
#include "tools/help.hh"
```

Classes

- struct **io::BoundarySize**
- class **io::Writer**

10.36.1 Detailed Description

This file is part of SWE.

Author

Sebastian Rettenberger (rettenbs AT in.tum.de, `http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.`)

10.36.2 LICENSE

10.36.3 DESCRIPTION

Index