SWE

Generated by Doxygen 1.8.4

Mon May 27 2013 05:17:31

Contents

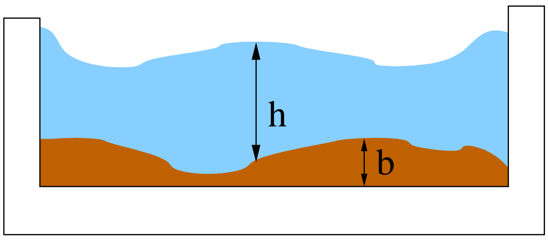
Chapter 1

SWE - A Simple Shallow Water Code

SWE is a teaching code that implements simple Finite Volumes models that solve the shallow water equations - in a problem setting as it would be used for tsunami simulation.

1.1 The Shallow Water Equations

The shallow water equations describe the behaviour of a fluid, in particular water, of a certain (possibly varying) depth h in a two-dimensional domain – imagine, for example, a puddle of water or a shallow pond (and compare the 1D sketch given below). The main modelling assumption is that we can neglect effects of flow in vertical direction. The resulting model proved to be useful for the simulation of tsunami propagation (with appropriate extensions). While an ocean can hardly be considered as "shallow" in the usual sense, tsunami waves (in contrast to regular waves induced by wind, e.g.) affect the entire water column, such that effects of vertical flow can again be neglected. To allow for a non-even sea bottom (as required for accurate modelling of tsunamis), we include the elevation b of the sea floor in our model:



The shallow water equations describe the changes of water depth h and horizontal velocities v_x and v_y (in the resp. coordinate directions) over time, depending on some initial conditions – in the case of tsunami simulation, these initial conditions could, for example, result from an initial elevation of the sea floor caused by an earthquake. The respective changes in time can be described via a system of partial differential equations:

$$\frac{\partial h}{\partial t} + \frac{\partial (v_x h)}{\partial x} + \frac{\partial (v_y h)}{\partial y} = 0$$
$$\frac{\partial (hv_x)}{\partial t} + \frac{\partial (hv_x v_x)}{\partial x} + \frac{\partial (hv_y v_x)}{\partial y} + \frac{1}{2} g \frac{\partial (h^2)}{\partial x} = -gh\frac{\partial b}{\partial x},$$
$$\frac{\partial (hv_y)}{\partial t} + \frac{\partial (hv_x v_y)}{\partial x} + \frac{\partial (hv_y v_y)}{\partial y} + \frac{1}{2} g \frac{\partial (h^2)}{\partial y} = -gh\frac{\partial b}{\partial y},$$

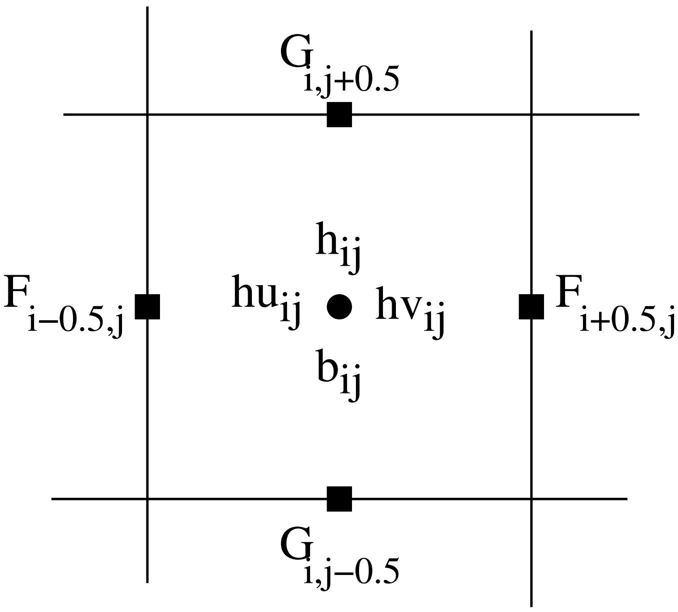
The equation for h is obtained, if we examine the conservation of mass in a control volume. The equations for hv_x and hv_y result from conservation of momentum (note that h is directly related to the volume, and thus the mass of the water – thus hv_x can be interpreted as a momentum).

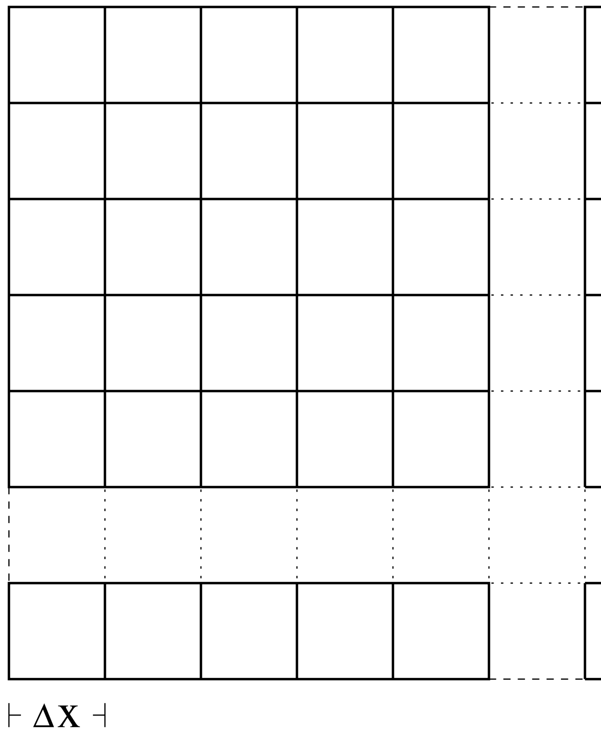
The two terms involving g model a gravity-induced force (g being the constant for the gravitational acceleration, $g = 9.81\ ms^{-2}$), which results from the hydrostatic pressure. The right-hand-side source terms model the effect of an uneven ocean floor (b obtained from respective bathymetry data).

1.1.1 Finite Volume Discretisation

The shallow water equations are usually too difficult to be solved exactly - hence, SWE implements simple discrete models as an approximation. As the applied numerical method (typically a Finite Volume discretization) may vary, we will stick to the basics at this point.

First, SWE assumes that the unknown functions *h(t,x,y)*, *hu(t,x,y) := h(t,x,y)* v_x *(t,x,y)*, *hv(t,x,y) := h(t,x,y)* v_y *(t,x,y)*, as well as the given sea bottom level *b(x,y)*, are approximated on a Cartesian mesh of grid cells, as illustrated below. In each grid cell, with indices *(i,j)*, the unknowns have constant values h_{ij} , hu_{ij} , hv_{ij} , and b_{ij} :





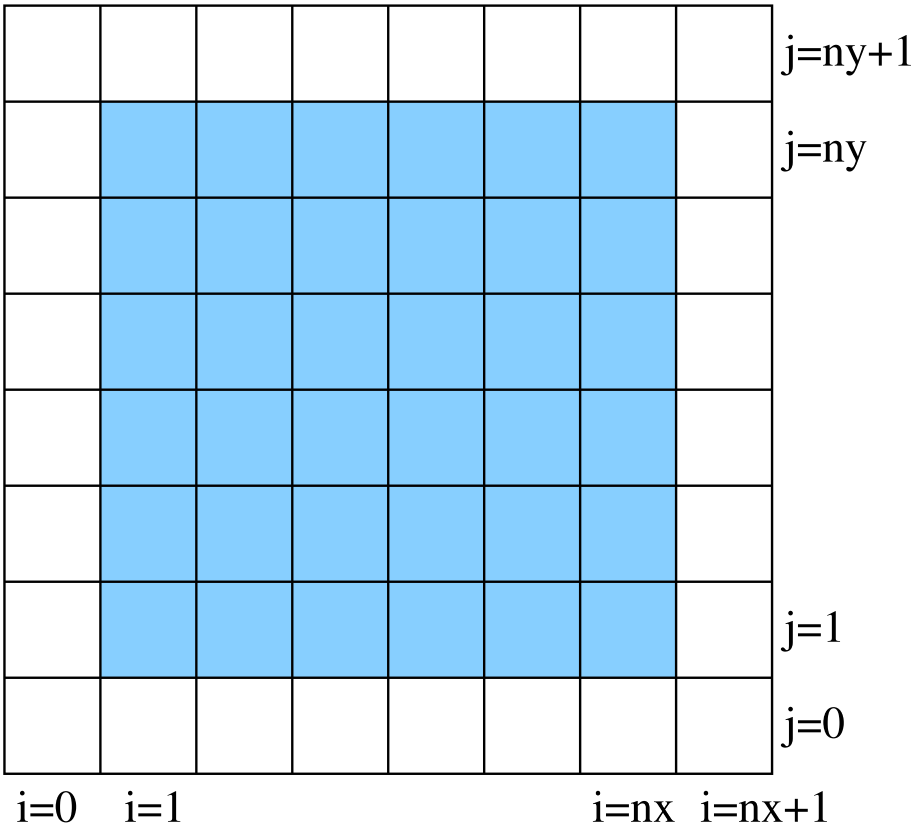
1.1.2 Computing Numerical Fluxes at Edges and Euler Time-Stepping

The details of the numerical schemes are too complicated to be described in this overview. Please refer to the accompanying material. To put it short, we successively perform two main computational steps:

- we compute so-called **numerical fluxes** on each edge of the grid (which approximate the transfer of mass or momentum between grid cells),
- based on these numerical fluxes, we then update the unknowns in each cell.

1.2 Implementation and base class SWE_Block

For the simulation of the shallow water model, we thus require a regular Cartesian grid, where each grid cell carries the respective unknowns - water level, momentum in x- and y-direction, and bathymetry data. The central data structures for Cartesian grid and arrays of unknowns are provided with the abstract base class SWE_Block, which has four 2D arrays SWE_Block::h, SWE_Block::hu, SWE_Block::hv, and SWE_Block::b. To implement the behaviour of the fluid at boundaries, and also to allow the connection of several grid blocks (for parallelization or just to build more complicated computational domains), each array has an additional layer of so-called *ghost cells*, as illustrated in the following figure:



1.2.1 Parallelisation and Different Models

In each time step, our numerical algorithm will compute the flux terms for each edge of the computational domain. To compute the fluxes, we require the values of the unknowns in both adjacent cells. At the boundaries of the fluid domain, the ghost layer makes sure that we also have two adjacent cells for the cell edges on the domain boundary. The values in the ghost layer cells will be set to values depending on the values in the adjacent fluid domain. We will model three different situations: {description} {Outflow:} {h}, {u}, and {v} in the ghost cell are set to the same value as in the adjacent fluid cell. This models the situation that the unknowns do not change across the domain boundary (undisturbed outflow). {Wall:} At a wall, the velocity component normal to the boundary should be 0, such that no fluid can cross the boundary. To model this case, we set the normal velocity, e.g. {u[0]} at the left boundary, to the negative value of the adjacent cell: {-u[1]}. The interpolated value at the boundary edge will then be 0 ({h} is identical in both cells due to the imposed boundary condition). The other two variables are set in the same way as for the outflow situation. {Connect:} With the connect case, we can connect a domain at two boundaries. If we connect the left and right boundary, we will obtain a periodically repeated domain. Here, all ghost values are determined by the values of the unknowns in the fluid cell adjacent to the connected boundary. {description}

To implement the boundary conditions, the class {SWE_Block} contains an array of four enum variables, {boundary[4]} (for left/right/bottom/top boundary), that can take the values `OUTFLOW`, `WALL`, and `CONNECT`.

1.2.2 Multiple Blocks

Via the connect boundary condition, it is also possible to connect several Cartesian grid blocks to build a more complicated domain. Figure fig:connect} illustrates the exchange of ghost values for two connected blocks.

To store the neighbour block in case of a `CONNECT` boundary, `SWE_Block` contains a further array of four pointers, `neighbour[4]` (for left/right/bottom/top boundary), that will store a pointer to the connected adjacent `SWE_Block`.

The respective block approach can also be exploited for parallelisation: the different blocks would then be assigned to the available processors (or processor cores) – each processor (core) works on its share of blocks, while the program has to make sure to keep the values in the ghost cells up to date (which requires explicit communication in

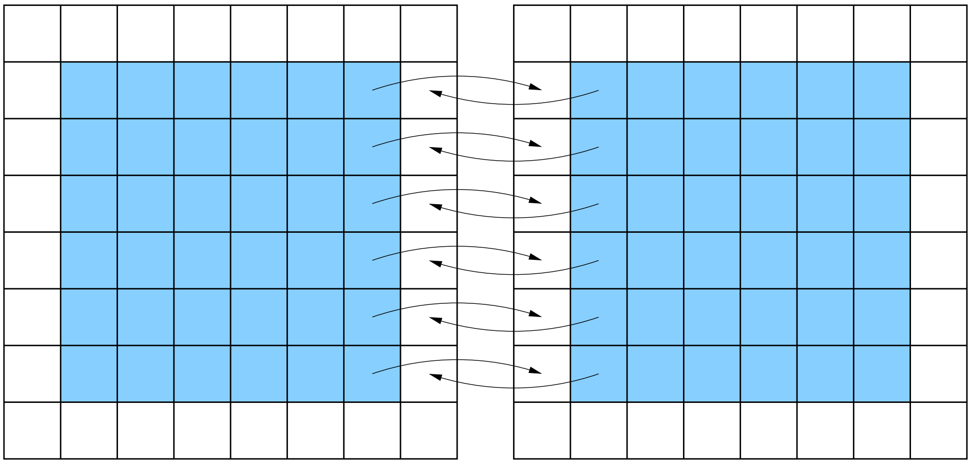


Figure 1.1: Exchange of values in ghost layers between two connected SWE_Blocks.

the case of distributed-memory computers).

1.2.3

For each time step, our solver thus performs the following steps – each step is implemented via a separate member function of the class {SWE_Block}: {enumerate} set the values at the boundaries: {setBoundaryLayer()}; compute the flux terms for all edges: {computeFluxes()}; from the flux terms, compute the in/outflow balance for each cell, and compute the new values of the unknowns for the next time step: {eulerTimestep()}. {enumerate}

1.2.4

The class {SWE_Block} contains further methods that will write the numerical solution into a sequence of files that can be read by the visualisation package ParaView (just enter the respective folder from ParaView – the files will be recognised and displayed as one project). ParaView allows to visualise the computed time-dependent solution (as “movie” or in single-step mode). ParaView is pretty self-explanatory for our purposes, but provides an online help for further instructions.

1.2.5

We also provide a CUDA implementation of the simulation code (requires a computer with a CUDA-capable GPU, together with the respective drivers – visit NVIDIA’s website on CUDA for details on implementation). Apart from the fact that the simulation usually runs a lot faster on the GPU, the program is also capable of plotting the computing solution (water surface) “on the fly”.

Finally: whoever thinks that they can do a better (faster, ...) implementation (visualisation, ...) of the provided code is more than welcome to do so! Feel free to contribute to SWE - for questions, just contact Michael Bader (bader@in.tum.de).

Chapter 2

Todo List

Member io::VtkWriter::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)

This version can only handle a boundary layer of size 1

Chapter 3

Deprecated List

Member generateFileName (std::string baseName, int timeStep)

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

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

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

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

Chapter 7

Class Documentation

7.1 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 ()

7.1.1 Constructor & Destructor Documentation

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

Constructor

Parameters

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

7.1.2 Member Function Documentation

7.1.2.1 void Camera::displayImage ( )

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

7.1.2.2 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

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

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

7.1.2.4 void Camera::setCamera ( )

Set the camera via gluLookAt and set the light position afterwards

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

User starts dragging. Remember the old mouse coordinates.

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

Set the view distance

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

Zoom in

Parameters

scaleFactor	factor which is used for zooming

7.1.2.8 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:

- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/camera.h
- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/camera.cpp

7.2 Controller Class Reference

Public Member Functions

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

7.2.1 Constructor & Destructor Documentation

7.2.1.1 Controller::Controller (Simulation * *sim*, Visualization * *vis*)

Constructor

Parameters

sim	instance of simulation class
vis	instance of visualization class

7.2.2 Member Function Documentation

7.2.2.1 bool Controller::handleEvents ( )

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

7.2.2.2 bool Controller::hasFocus ( )

Returns true, when window has focus

7.2.2.3 bool Controller::isPaused ( )

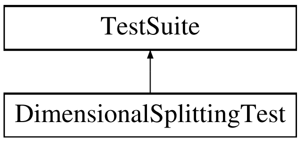
Return whether program is currently paused

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/controller.h
- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/controller.cpp

7.3 DimensionalSplittingTest Class Reference

Inheritance diagram for DimensionalSplittingTest:



Public Member Functions

- void **testCompareNetUpdates** ()
- void **testHorizontalSteadyState** ()

Static Public Attributes

- static const int **row** = 0
- static const float **dt** = 0.01
- static const float **accuracy** = 1.0E-6
- static const int **nx** = 200
- static const int **ny** = 1
- static const float **dx** = 1.f
- static const float **dy** = 1.f

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/testing/DimensionalSplittingTest.t.h

7.4 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

7.4.1 Detailed Description

class Float1D is a proxy class that can represent, for example, a column or row vector of a Float2D 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 (or other "host" data structures).

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/tools/help.hh

7.5 Float2D Class Reference

`#include <help.hh>`

Public Member Functions

- Float2D (int _cols, int _rows)
- 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)

7.5.1 Detailed Description

class Float2D 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].

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Float2D::Float2D (int *_cols*, int *_rows*) `[inline]`

Constructor

Parameters

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

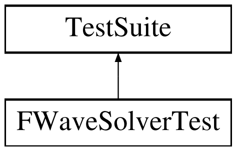
The documentation for this class was generated from the following file:

- /home/raphael/Programmieren/BPraktikum/SWE/src/tools/help.hh

7.6 FWaveSolverTest Class Reference

`#include <FWaveSolverTest.hpp>`

Inheritance diagram for FWaveSolverTest:



Public Member Functions

- void testEigenvalues ()
- void testFlux ()
- void testEigencoefficients ()
- void testSteadyState ()
- void testSupersonic ()
- void testBathymetry ()
- void **testDryCells** ()

7.6.1 Detailed Description

This is the cxxtest test-suite for the FWave template class.

7.6.2 Member Function Documentation

7.6.2.1 void FWaveSolverTest::testBathymetry ( ) `[inline]`

check the handling of bathymetric data

a steady state will be created, but with different water depths

7.6.2.2 void FWaveSolverTest::testEigencoefficients ( ) `[inline]`

testEigencoefficients will do a valueCheck on the function "eigencoeffis" of the template "FWave"

7.6.2.3 void FWaveSolverTest::testEigenvalues ( ) `[inline]`

this function will test the private function roeEigenvals of the template FWave by performing two value checks

7.6.2.4 void FWaveSolverTest::testFlux ( ) [inline]

testFlux will perform a fast value check on the private function "flux" of the template "FWave"

7.6.2.5 void FWaveSolverTest::testSteadyState ( ) [inline]

testSteadyState will calculate the net updates for identical water columns and momentum on both sides, which have to be equal to zero

7.6.2.6 void FWaveSolverTest::testSupersonic ( ) [inline]

testSupersonic will check the function computeNetUpdates for correct behavior in the supersonic case (both eigenvalues greater/less than zero)

- greater than zero: the net-updates on the left have to be zero
- less than zero: the net-updates on the right have to be zero

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/f-wave-solver/src/FWaveSolverTest.hpp

7.7 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]

7.7.1 Detailed Description

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

7.7.2 Member Data Documentation

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

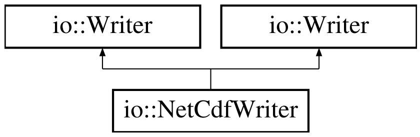
boundarySize[0] == left boundarySize[1] == right boundarySize[2] == bottom boundarySize[3] == top

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/writer/Writer.hh

7.8 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 ETime, bool newfile=false, 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)
- void **writeBoundary** (char ∗up, char ∗bottom, char ∗left, char ∗right)
- **NetCdfWriter** (char ∗CPfile)
- 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 ETime, float i_originX=0., float i_originY=0., unsigned int i_flush=0)
- void writeTimeStep (const Float2D &i_h, const Float2D &i_hu, const Float2D &i_hv, float i_time)

Additional Inherited Members

7.8.1 Constructor & Destructor Documentation

7.8.1.1 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 *ETime*, bool *newfile* = `false`, 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_dynamicBathymetry	

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

Destructor of a netCDF-writer.

7.8.1.3 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 *ETime*, 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	

7.8.2 Member Function Documentation

7.8.2.1 void io::NetCdfWriter::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.

7.8.2.2 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.

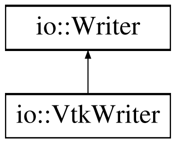
Implements io::Writer.

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/writer/NetCdfWriter.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/writer/NetCdfWriterCP.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/writer/NetCdfWriter.cpp
- /home/raphael/Programmieren/BPraktikum/SWE/src/writer/NetCdfWriterCP.cpp

7.9 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

7.9.1 Constructor & Destructor Documentation

7.9.1.1 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

7.9.2 Member Function Documentation

7.9.2.1 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.

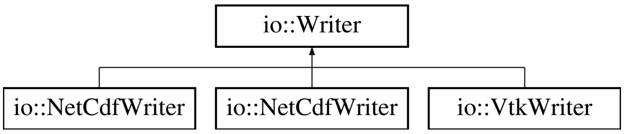
Implements io::Writer.

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/writer/VtkWriter.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/writer/VtkWriter.cpp

7.10 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

7.10.1 Constructor & Destructor Documentation

7.10.1.1 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.

7.10.2 Member Function Documentation

7.10.2.1 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, io::NetCdfWriter, and io::VtkWriter.

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/writer/Writer.hh

7.11 scenarios::ConstantFlow Class Reference

Public Member Functions

- **ConstantFlow** (unsigned int size)
- float getHeight (unsigned int pos)
- float getMomentum (unsigned int pos)
- float getBathymetry (unsigned int pos)
- T getCellSize ()

7.11.1 Member Function Documentation

7.11.1.1 float scenarios::ConstantFlow::getBathymetry (unsigned int *pos*) [inline]

Returns

floor elevation at pos

7.11.1.2 T scenarios::ConstantFlow::getCellSize ( ) [inline]

Returns

Cell size of one cell (= domain size/number of cells)

7.11.1.3 float scenarios::ConstantFlow::getHeight (unsigned int *pos*) [inline]

Returns

Initial water height at pos

7.11.1.4 float scenarios::ConstantFlow::getMomentum (unsigned int *pos*) [inline]

Returns

Initial momentum at pos

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/constant_flow.h

7.12 scenarios::DamBreak Class Reference

Public Member Functions

- **DamBreak** (unsigned int size)
- float getHeight (unsigned int pos)
- float getMomentum (unsigned int pos)
- T getCellSize ()
- **DamBreak** (unsigned int size)
- float getHeight (unsigned int pos)
- float getMomentum (unsigned int pos)
- float getBathymetry (unsigned int pos)
- T getCellSize ()

7.12.1 Member Function Documentation

7.12.1.1 float scenarios::DamBreak::getBathymetry (unsigned int *pos*) [inline]

Returns

floor elevation at pos

7.12.1.2 T scenarios::DamBreak::getCellSize ( ) [inline]

Returns

Cell size of one cell (= domain size/number of cells)

7.12.1.3 T scenarios::DamBreak::getCellSize ( ) [inline]

Returns

Cell size of one cell (= domain size/number of cells)

7.12.1.4 float scenarios::DamBreak::getHeight (unsigned int *pos*) [inline]

Returns

Initial water height at pos

7.12.1.5 float scenarios::DamBreak::getHeight (unsigned int *pos*) [inline]

Returns

Initial water height at pos

7.12.1.6 float scenarios::DamBreak::getMomentum (unsigned int *pos*) [inline]

Returns

Initial momentum at pos

7.12.1.7 float scenarios::DamBreak::getMomentum (unsigned int *pos*) [inline]

Returns

Initial momentum at pos

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/dambreak.h
- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/dambreak_bathy.h

7.13 scenarios::Eisbach Class Reference

Public Member Functions

- **Eisbach** (unsigned int size)
- float getHeight (unsigned int pos)
- float getMomentum (unsigned int pos)
- float getBathymetry (unsigned int pos)
- T getCellSize ()

7.13.1 Member Function Documentation

7.13.1.1 float scenarios::Eisbach::getBathymetry (unsigned int *pos*) [inline]

Returns

floor elevation at pos

7.13.1.2 T scenarios::Eisbach::getCellSize ( ) [inline]

Returns

Cell size of one cell (= domain size/number of cells)

7.13.1.3 float scenarios::Eisbach::getHeight (unsigned int *pos*) [inline]

Returns

Initial water height at pos

7.13.1.4 float scenarios::Eisbach::getMomentum (unsigned int *pos*) [inline]

Returns

Initial momentum at pos

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/eisbach.h

7.14 scenarios::RareRare Class Reference

Public Member Functions

- **RareRare** (unsigned int size)
- unsigned int getHeight (unsigned int pos)
- signed int getMomentum (unsigned int pos)
- T getCellSize ()

7.14.1 Member Function Documentation

7.14.1.1 T scenarios::RareRare::getCellSize ( ) [inline]

Returns

Cell size of one cell (= domain size/number of cells)

7.14.1.2 unsigned int scenarios::RareRare::getHeight (unsigned int *pos*) [inline]

Returns

Initial water height at pos

7.14.1.3 signed int scenarios::RareRare::getMomentum (unsigned int *pos*) [inline]

Returns

Initial momentum at pos

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/rarerare.h

7.15 scenarios::ShockShock Class Reference

Public Member Functions

- **ShockShock** (unsigned int size)
- unsigned int getHeight (unsigned int pos)
- int getMomentum (unsigned int pos)
- T getCellSize ()

7.15.1 Member Function Documentation

7.15.1.1 T scenarios::ShockShock::getCellSize ( ) [inline]

Returns

Cell size of one cell (= domain size/number of cells)

7.15.1.2 unsigned int scenarios::ShockShock::getHeight (unsigned int *pos*) [inline]

Returns

Initial water height at pos

7.15.1.3 int scenarios::ShockShock::getMomentum (unsigned int *pos*) [inline]

Returns

Initial momentum at pos

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/ShockShock.h

7.16 scenarios::Subcrit Class Reference

Public Member Functions

- **Subcrit** (unsigned int size)
- float getHeight (unsigned int pos)
- float getMomentum (unsigned int pos)
- float getBathymetry (unsigned int pos)
- T getCellSize ()

7.16.1 Member Function Documentation

7.16.1.1 float scenarios::Subcrit::getBathymetry (unsigned int *pos*) [inline]

Returns

floor elevation at pos

7.16.1.2 T scenarios::Subcrit::getCellSize ( ) [inline]

Returns

Cell size of one cell (= domain size/number of cells)

7.16.1.3 float scenarios::Subcrit::getHeight (unsigned int *pos*) [inline]

Returns

Initial water height at pos

7.16.1.4 float scenarios::Subcrit::getMomentum (unsigned int *pos*) [inline]

Returns

Initial momentum at pos

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/Subcritical_flow.h

7.17 scenarios::Supercrit Class Reference

Public Member Functions

- **Supercrit** (unsigned int size)
- float getHeight (unsigned int pos)
- float getMomentum (unsigned int pos)
- float getBathymetry (unsigned int pos)
- T getCellSize ()

7.17.1 Member Function Documentation

7.17.1.1 float scenarios::Supercrit::getBathymetry (unsigned int *pos*) [inline]

Returns

floor elevation at pos

7.17.1.2 T scenarios::Supercrit::getCellSize ( ) [inline]

Returns

Cell size of one cell (= domain size/number of cells)

7.17.1.3 float scenarios::Supercrit::getHeight (unsigned int *pos*) [inline]

Returns

Initial water height at pos

7.17.1.4 float scenarios::Supercrit::getMomentum (unsigned int *pos*) [inline]

Returns

Initial momentum at pos

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/Supercritical_flow.h

7.18 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)

7.18.1 Constructor & Destructor Documentation

7.18.1.1 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

7.18.1.2 Shader::∼Shader ( )

Destructor. Unload shaders and free resources.

7.18.2 Member Function Documentation

7.18.2.1 void Shader::disableShader ( )

Restores OpenGL default shaders

7.18.2.2 void Shader::enableShader ( )

Replaces OpenGL shaders by our custom shaders

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

Returns

Location of the uniform variable

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

Set a uniform variable in the shader

7.18.2.5 bool Shader::shadersLoaded ( )

Returns, whether shaders could by loaded successfully

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/shader.h
- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/shader.cpp

7.19 Simulation Class Reference

Public Member Functions

- Simulation ()
- ∼Simulation ()
- 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)

7.19.1 Constructor & Destructor Documentation

7.19.1.1 Simulation::Simulation ( )

Constructor. Initializes SWE_BlockCUDA and creates a new instance of it.

7.19.1.2 Simulation::∼Simulation ( )

Destructor.

7.19.2 Member Function Documentation

7.19.2.1 void Simulation::getScalingApproximation (float & *bScale*, float & *bOffset*, float & *wScale*)

Computes a first approximation of the scaling values needed for visualization. Gets called before simulation starts and determines the average, mininimum and maximum values of the bathymetry and water surface data. Uses latter values to estimate the scaling factors.

7.19.2.2 void Simulation::restart ( )

Restarts the simulation. Restores the initial bondaries.

7.19.2.3 void Simulation::runCuda (struct cudaGraphicsResource ∗∗ *vbo_resource*, struct cudaGraphicsResource ∗∗ *vbo_normals*)

This is the main simulation procedure. Simulates one timestep and computes normals afterwards.

Parameters

vbo_resource	cuda resource holding the vertex positions
vbo_normals	cuda resource holding the normals

7.19.2.4 void Simulation::setBathBuffer (float ∗ *bath*)

Sets the bathymetry buffer. Buffer contains vertex position and vertex normal in sequence.

Parameters

bath	float array in which computed values will be stored

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/simulation.h
- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/simulation.cu

7.20 solver::FWave< T > Class Template Reference

```
#include <FWave.hpp>
```

Public Member Functions

- void computeNetUpdates (const T &hLeft, const T &hRight, const T &huLeft, const T &huRight, const T &bathLeft, const T &bathRight, T &hNetUpdatesLeft, T &hNetUpdatesRight, T &huNetUpdatesLeft, T &huNetUpdatesRight, T &maxEdgeSpeed)

7.20.1 Detailed Description

template<class T>class solver::FWave< T >

This class will approximately solve the initial value problem for the **shallow water equations** over time:

```
/ h  \    /     hu            \
|    | + |                    |   = S(x,t)
\ hu /    \ hu^2 + 1/2*gh^2 /
```

7.20.2 Member Function Documentation

7.20.2.1 template<class T> void FWave::computeNetUpdates (const T & *hLeft*, const T & *hRight*, const T & *huLeft*, const T & *huRight*, const T & *bathLeft*, const T & *bathRight*, T & *hNetUpdatesLeft*, T & *hNetUpdatesRight*, T & *huNetUpdatesLeft*, T & *huNetUpdatesRight*, T & *maxEdgeSpeed*)

calculate the net-updates for a simulation of the flow of water

This implementation will calculate the net-updates for a simulation of flow of water using the height of the water column and its momentum as parameters.

Parameters

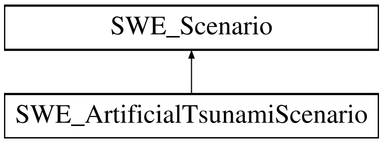
hLeft	water column height on the left side
hRight	water column height on the right side
huLeft	momentum of the water on the left side
huRight	momentum of the water on the right side
bathLeft	elevation of the ocean floor on the left side
bathRight	elevation of the ocean floor on the right side
hNetUpdatesLeft	reference to the variable that will store the update to the height of the water column on the left
hNetUpdatesRight	reference to the variable that will store the update to the height of the water column on the right
huNetUpdatesLeft	reference to the variable that will store the update to the momentum of the water on the left
huNetUpdatesRight	reference to the variable that will store the update to the momentum of the water on the right
maxEdgeSpeed	reference to the variable that will store the maximum edge-speed

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/f-wave-solver/src/FWave.hpp
- /home/raphael/Programmieren/BPraktikum/SWE/submodules/f-wave-solver/src/FWave.cpp

7.21 SWE_ArtificialTsunamiScenario Class Reference

Inheritance diagram for SWE_ArtificialTsunamiScenario:



Public Member Functions

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

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_ArtificialTsunamiScenario.hh

7.22 SWE_AsagiGrid Class Reference

Public Member Functions

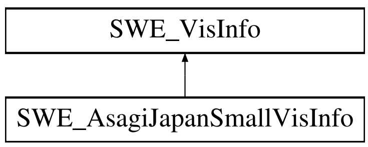
- void **open** (const std::string &i_filename)
- void **close** ()
- asagi::Grid & **grid** ()

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_AsagiScenario.hh

7.23 SWE_AsagiJapanSmallVisInfo Class Reference

Inheritance diagram for SWE_AsagiJapanSmallVisInfo:



Public Member Functions

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

7.23.1 Member Function Documentation

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

Returns

The vertical scaling factor for the bathymetry

Reimplemented from SWE_VisInfo.

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

Returns

The vertical scaling factor of the water

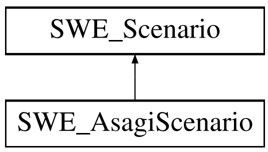
Reimplemented from SWE_VisInfo.

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_AsagiScenario_vis.hh

7.24 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)

7.24.1 Constructor & Destructor Documentation

7.24.1.1 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)

7.24.2 Member Function Documentation

7.24.2.1 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

7.24.2.2 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.

7.24.2.3 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.

7.24.2.4 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))

7.24.2.5 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.

7.24.2.6 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.

7.24.2.7 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.

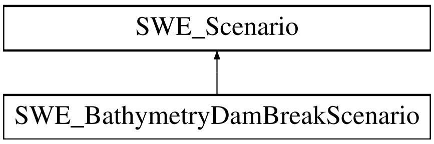
The documentation for this class was generated from the following files:

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_AsagiScenario.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_AsagiScenario.cpp

7.25 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)

7.25.1 Detailed Description

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

7.25.2 Member Function Documentation

7.25.2.1 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.

7.25.2.2 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.

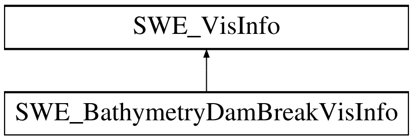
The documentation for this class was generated from the following file:

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_simple_scenarios.hh

7.26 SWE_BathymetryDamBreakVisInfo Class Reference

```
#include <SWE_simple_scenarios_vis.hh>
```

Inheritance diagram for SWE_BathymetryDamBreakVisInfo:



Public Member Functions

- float bathyVerticalOffset ()

7.26.1 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

7.26.2 Member Function Documentation

7.26.2.1 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.

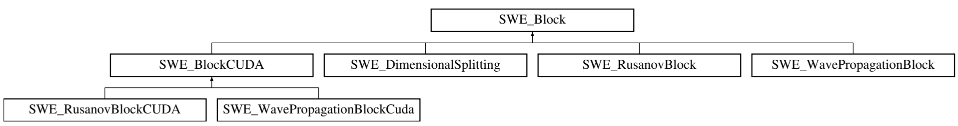
The documentation for this class was generated from the following file:

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_simple_scenarios_vis.hh

7.27 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)=0

    execute a single time step of the simulation
- virtual float simulate (float tStart, float tEnd)=0
- 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, i.e. the grid size in x-direction
- int getNy ()

    returns ny, 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

7.27.1 Detailed Description

SWE_Block is the main data structure to compute our shallow water model on a single Cartesian grid block: SWE_Block 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 by ny, where each grid cell carries three unknowns:

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

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

The mesh sizes of the grid in x- and y-direction are stored in static variables dx and dy. 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 and offsetY.

Ghost layers:

To implement the behaviour of the fluid at boundaries and for using multiple block in serial and parallel settings, SWE_Block 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+1 / ny+1.

Memory Model:

The variables h, hu, hv for water height and momentum will typically be updated by classes derived from SWE_-Block. However, it is not assumed that such and updated will be performed in every time step. Instead, subclasses are welcome to update h, hu, and hv 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 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:

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

Derived Classes

As SWE_Block 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.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 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 can be generated.

7.27.2.2 SWE_Block::∼SWE_Block ( ) [protected], [virtual]

Destructor: de-allocate all variables

7.27.3 Member Function Documentation

7.27.3.1 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.

7.27.3.2 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_WavePropagationBlock, SWE_WavePropagationBlockCuda, SWE_RusanovBlock, SWE_-RusanovBlockCUDA, and SWE_DimensionalSplitting.

7.27.3.3 const Float2D & SWE_Block::getBathymetry ( )

provides read access to the bathymetry data

return reference to bathymetry unknown b

7.27.3.4 const Float2D & SWE_Block::getDischarge_hu ( )

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

return reference to discharge unknown hu

7.27.3.5 const Float2D & SWE_Block::getDischarge_hv ( )

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

return reference to discharge unknown hv

7.27.3.6 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

7.27.3.7 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

7.27.3.8 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 object that contains row variables h, hu, and hv

Reimplemented in SWE_BlockCUDA.

7.27.3.9 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.

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 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?

7.27.3.10 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 object that contains row variables h, hu, and hv

Reimplemented in SWE_BlockCUDA.

7.27.3.11 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

7.27.3.12 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

7.27.3.13 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.

7.27.3.14 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.

7.27.3.15 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, which specifies the inflow (should be NULL for WALL or OUTFLOW boundary)

7.27.3.16 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!

7.27.3.17 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, the values are copied

7.27.3.18 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

7.27.3.19 virtual float SWE_Block::simulate (float *tStart,* float *tEnd*) `[pure 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 that this function can typically only be used, if you only use a single SWE_Block; in particular, simulate 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

Implemented in SWE_WavePropagationBlock, SWE_WavePropagationBlockCuda, SWE_RusanovBlockCUDA, S-WE_RusanovBlock, and SWE_DimensionalSplitting.

7.27.3.20 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.

7.27.3.21 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.

7.27.3.22 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.

7.27.3.23 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.

7.27.3.24 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.

7.27.3.25 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.

7.27.3.26 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.

7.27.3.27 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.

7.27.3.28 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.

7.27.3.29 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.

7.27.3.30 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_WavePropagationBlock, SWE_WavePropagationBlockCuda, SWE_RusanovBlock, SWE_-RusanovBlockCUDA, and SWE_DimensionalSplitting.

7.27.4 Member Data Documentation

7.27.4.1 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:

- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/SWE_Block.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/SWE_Block.cpp

7.28 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**

7.28.1 Detailed Description

SWE_Block1D is a simple struct that can represent a single line or row of SWE_Block unknowns (using the Float1D 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 grids.

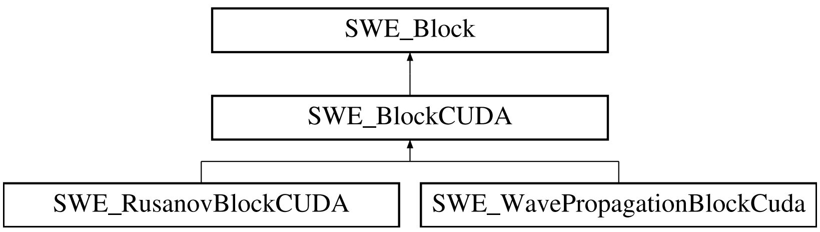
The documentation for this struct was generated from the following file:

- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/SWE_Block.hh

7.29 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_BlockCUDA ()
- virtual SWE_Block1D ∗ registerCopyLayer (BoundaryEdge edge)
- virtual SWE_Block1D ∗ grabGhostLayer (BoundaryEdge edge)
- 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 ()

Protected Attributes

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

Additional Inherited Members

7.29.1 Detailed Description

SWE_BlockCUDA extends the base class SWE_Block 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.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 SWE_BlockCUDA::SWE_BlockCUDA (int *l_nx,* int *l_ny,* float *l_dx,* float *l_dy*)

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]

Parameters

i_cudaDevice	ID of the CUDA-device, which should be used.

7.29.2.2 SWE_BlockCUDA::~SWE_BlockCUDA ( ) [virtual]

Destructor: de-allocate all variables

7.29.3 Member Function Documentation

7.29.3.1 void SWE_BlockCUDA::finalize ( ) [static]

Cleans up the cuda device

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

Returns

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

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

Returns

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

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

"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 object that contains row variables h, hu, and hv

Reimplemented from SWE_Block.

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

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

7.29.3.6 void SWE_BlockCUDA::printDeviceInformation ( ) [static]

Print some available information about the CUDA devices. id of the CUDA device.

total number of CUDA devices on this host.

drive and runtime version

device properties

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

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 object that contains row variables h, hu, and hv

Reimplemented from SWE_Block.

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

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

Reimplemented from SWE_Block.

7.29.3.9 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.

7.29.3.10 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.

7.29.3.11 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.

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

Update the main variables h, hu, hv, and b before an external read access: copies current content of the respective device variables hd, hud, hvd, bd

Reimplemented from SWE_Block.

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

Update (for heterogeneous computing) variables h, hu, hv in copy layers before an external access to the unknowns (only required for PASSIVE and CONNECT boundaries)

- copy (up-to-date) content from device memory into main memory

Reimplemented from SWE_Block.

7.29.3.14 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.

7.29.3.15 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.

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

synchronise the ghost layer content of h, hu, and hv in main memory with device memory and auxiliary data structures, i.e. transfer memory from main/auxiliary memory into device memory

Reimplemented from SWE_Block.

7.29.3.17 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.

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

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

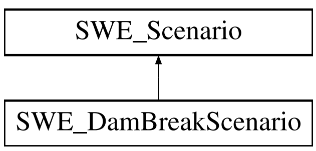
Reimplemented from SWE_Block.

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/cuda/SWE_BlockCUDA.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/cuda/SWE_BlockCUDA.cu

7.30 SWE_DamBreakScenario Class Reference

Inheritance diagram for SWE_DamBreakScenario:



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)

7.30.1 Member Function Documentation

7.30.1.1 float SWE_DamBreakScenario::getBoundaryPos (BoundaryEdge *i_edge*) [inline], [virtual]

Get the boundary positions

Parameters

i_edge	which edge

Returns

value in the corresponding dimension

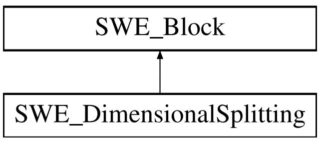
Reimplemented from SWE_Scenario.

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_simple_scenarios.hh

7.31 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 simulateTimestep (float dt)

  execute a single time step of the simulation
- float simulate (float tStart, float tEnd)
- void computeNumericalFluxes ()
- void updateUnknowns (float dt)
- void runTimestep ()

Additional Inherited Members

7.31.1 Member Function Documentation

7.31.1.1 void SWE_DimensionalSplitting::computeNumericalFluxes ( ) [virtual]

calculate the net-updates for the current state of the fluid, that can be applied later by updateUnknowns

Important if you change maxTimestep between this function an #updateUnkowns the accurancy of the calculation is not given any more.

Implements SWE_Block.

7.31.1.2 void SWE_DimensionalSplitting::runTimestep ( )

This funktion calculates and applays all changes for one Timestep

7.31.1.3 float SWE_DimensionalSplitting::simulate (float *tStart*, float *tEnd*) [virtual]

This methode runs a simulation for the time intervall from tStart to tEnd

Implements SWE_Block.

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

apply the net-updates calculated with computeNumericalFluxes

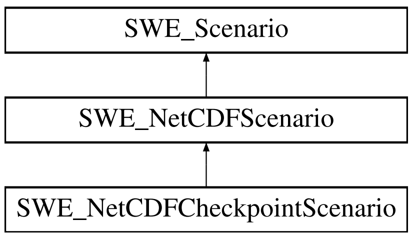
Implements SWE_Block.

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/SWE_DimensionalSplitting.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/SWE_DimensionalSplitting.cpp

7.32 SWE_NetCDFCheckpointScenario Class Reference

Inheritance diagram for SWE_NetCDFCheckpointScenario:



Public Member Functions

- SWE_NetCDFCheckpointScenario ()
- int readNetCDF (const char ∗filename, const char ∗CPFile)
- float **getWaterHeight** (float x, float y)
- float **getBathymetry** (float x, float y)
- float **getVeloc_u** (float x, float y)
- float **getVeloc_v** (float x, float y)
- float **getTime** ()
- float **getBoundaryPos** (BoundaryEdge i_edge)
- float **endSimulation** ()
- BoundaryType **getBoundaryType** (BoundaryEdge edge)

7.32.1 Constructor & Destructor Documentation

7.32.1.1 SWE_NetCDFCheckpointScenario::SWE_NetCDFCheckpointScenario ( ) [inline]

load a scenario from a netCDF file

Parameters

file	the netCDF file to load

7.32.2 Member Function Documentation

7.32.2.1 int SWE_NetCDFCheckpointScenario::readNetCDF (const char * *filename*, const char * *CPFile*) [inline]

readNetCDF will initialize the ids of the nc file and the ids of all the variables which are being used

Parameters

filename	the name of the nc-file to be opened
CPFile	filename of the checkpoint file

Returns

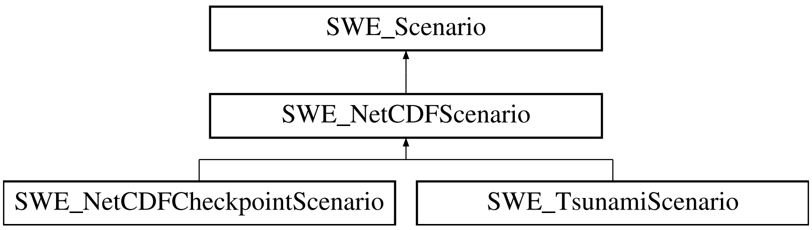
0 if successful, else the error value of the netcdf-library

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_NetCDFCheckpointScenario.hh

7.33 SWE_NetCDFScenario Class Reference

Inheritance diagram for SWE_NetCDFScenario:



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 int **readNetCDF** (char *filename, char *assistfile)
- virtual float **waterHeightAtRest** ()
- virtual float **endSimulation** ()
- virtual BoundaryType **getBoundaryType** (BoundaryEdge edge)
- virtual float **getBoundaryPos** (BoundaryEdge edge)

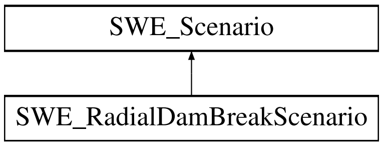
The documentation for this class was generated from the following file:

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_NetCDFScenario.hh

7.34 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)

7.34.1 Detailed Description

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

7.34.2 Member Function Documentation

7.34.2.1 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.

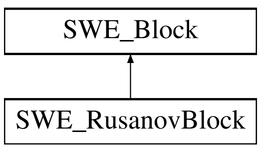
The documentation for this class was generated from the following file:

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_simple_scenarios.hh

7.35 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

7.35.1 Detailed Description

SWE_RusanovBlock is an implementation of the SWE_Block 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.

7.35.2 Constructor & Destructor Documentation

7.35.2.1 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

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

Destructor: de-allocate all variables

7.35.3 Member Function Documentation

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

compute source terms

compute the bathymetry source terms in all cells

7.35.3.2 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

7.35.3.3 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'

7.35.3.4 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.

7.35.3.5 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

Implements SWE_Block.

7.35.3.6 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

Implements SWE_Block.

7.35.3.7 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.

7.35.4 Friends And Related Function Documentation

7.35.4.1 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

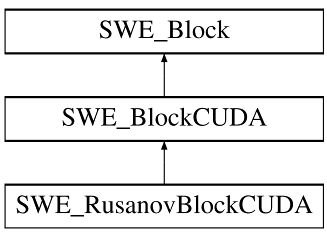
The documentation for this class was generated from the following files:

- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/rusanov/SWE_RusanovBlock.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/rusanov/SWE_RusanovBlock.cpp

7.36 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 ∼SWE_RusanovBlockCUDA ()
- virtual void computeNumericalFluxes ()
- virtual void updateUnknowns (float dt)
- 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

7.36.1 Detailed Description

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

7.36.2 Constructor & Destructor Documentation

7.36.2.1 SWE_RusanovBlockCUDA::SWE_RusanovBlockCUDA (float *_offsetX* = `0`, float *_offsetY* = `0`, const int *i_cudaDevice* = `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]

7.36.2.2 SWE_RusanovBlockCUDA::∼SWE_RusanovBlockCUDA ( ) `[virtual]`

Destructor: de-allocate all variables

7.36.3 Member Function Documentation

7.36.3.1 void SWE_RusanovBlockCUDA::computeNumericalFluxes ( ) `[virtual]`

compute the flux terms on all edges

Implements SWE_Block.

7.36.3.2 __host__ float SWE_RusanovBlockCUDA::simulate (float *tStart*, float *tEnd*) `[virtual]`

perform forward-Euler time steps, starting with simulation time tStart,: until simulation time tEnd is reached; device-global variables hd, hud, hvd are updated; unknowns h, hu, hv in main memory are not updated. Ghost layers and bathymetry sources are updated between timesteps. intended as main simulation loop between two checkpoints

Implements SWE_Block.

7.36.3.3 void SWE_RusanovBlockCUDA::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 CUDA-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

Implements SWE_Block.

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

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. It will force an update of the copy layer in the main memory by calling synchCopyLayerBefore-Read(), and provide an compute the maximum allowed time step size by calling computeMaxTimestepCUDA().

Parameters

dt	size of the time step.

Implements SWE_Block.

7.36.4 Friends And Related Function Documentation

7.36.4.1 ostream& operator<< (ostream & *os*, const SWE_RusanovBlockCUDA & *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

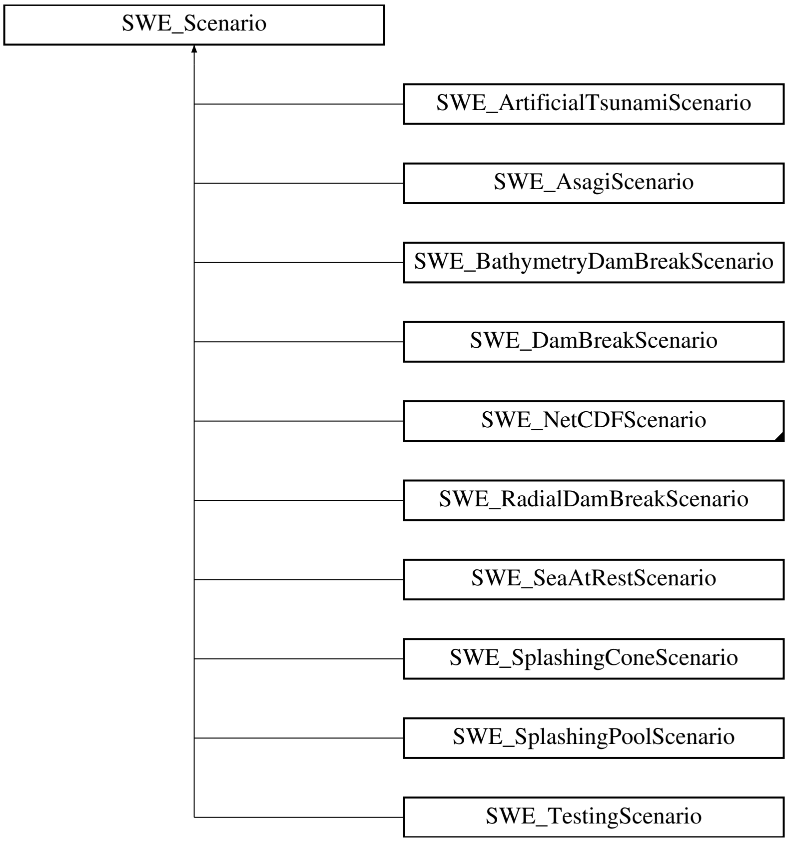
The documentation for this class was generated from the following files:

- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/rusanov/SWE_RusanovBlockCUDA.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/rusanov/SWE_RusanovBlockCUDA.cu

7.37 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)

7.37.1 Detailed Description

SWE_Scenario 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 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 interface for more interesting scenarios.

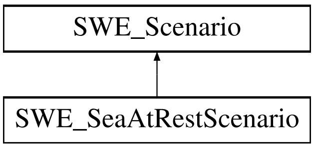
The documentation for this class was generated from the following file:

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_Scenario.hh

7.38 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)

7.38.1 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

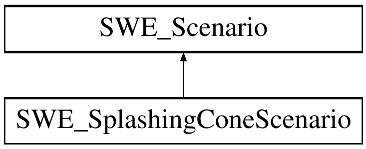
The documentation for this class was generated from the following file:

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_simple_scenarios.hh

7.39 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)

7.39.1 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)

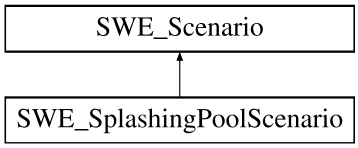
The documentation for this class was generated from the following file:

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_simple_scenarios.hh

7.40 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)

7.40.1 Detailed Description

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

7.40.2 Member Function Documentation

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

Get the boundary positions

Parameters

i_edge	which edge

Returns

value in the corresponding dimension

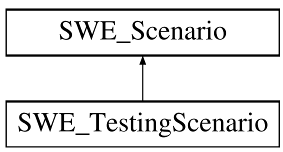
Reimplemented from SWE_Scenario.

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_simple_scenarios.hh

7.41 SWE_TestingScenario Class Reference

Inheritance diagram for SWE_TestingScenario:



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)

7.41.1 Member Function Documentation

7.41.1.1 float SWE_TestingScenario::getBoundaryPos (**BoundaryEdge** *i_edge*) [inline], [virtual]

Get the boundary positions

Parameters

i_edge	which edge

Returns

value in the corresponding dimension

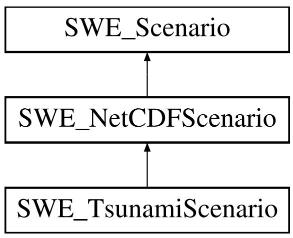
Reimplemented from SWE_Scenario.

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/testing/testing_scenario.hh

7.42 SWE_TsunamiScenario Class Reference

Inheritance diagram for SWE_TsunamiScenario:



Public Member Functions

- SWE_TsunamiScenario ()
- float **getWaterHeight** (float x, float y)
- float **getBathymetry** (float x, float y)
- float **getVeloc_u** (float x, float y)
- float **getVeloc_v** (float x, float y)
- float **getBoundaryPos** (BoundaryEdge i_edge)
- BoundaryType **getBoundaryType** (BoundaryEdge edge)
- int readNetCDF (const char ∗filename, const char ∗d_filename)

7.42.1 Constructor & Destructor Documentation

7.42.1.1 SWE_TsunamiScenario::SWE_TsunamiScenario ( ) [inline]

load a scenario from a netCDF file

Parameters

file	the netCDF file to load

7.42.2 Member Function Documentation

7.42.2.1 int SWE_TsunamiScenario::readNetCDF (const char * *filename*, const char * *d_filename*) [inline]

readNetCDF will initialize the ids of the nc file and the ids of all the variables which are being used

Parameters

filename	the name of the nc-file to be opened

Returns

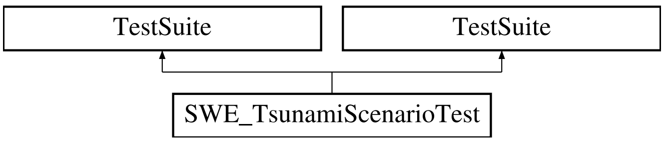
0 if successful, else the error value of the netcdf-library

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_TsunamiScenario.hh

7.43 SWE_TsunamiScenarioTest Class Reference

Inheritance diagram for SWE_TsunamiScenarioTest:



Public Member Functions

- void **testgetBoundaryPos** (void)
- void **testgetBathymetry** (void)
- void **testcornners** (void)
- void **testpossibleScenario** (void)
- void **testgetOriginalBathymetry** (void)
- void **testgetWaterHeight** (void)
- void **testgetVelco** (void)
- void **testtoGridCoordinates** (void)
- void **testgetBoundaryPos** ()

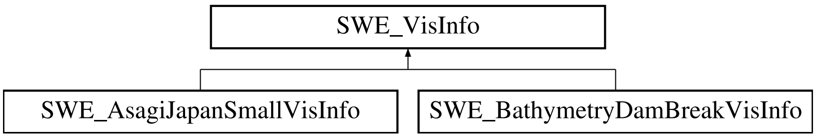
The documentation for this class was generated from the following files:

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_TsunamiScenarioTest.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_TsunamiScenarioTest.t.h

7.44 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 ()

7.44.1 Detailed Description

SWE_VisInfo 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() $*$ simulatedWaterHeight

For bathymetry: displayedBatyhmetry = bathyVerticalScaling() $*$ realBathymetry

- bathyVerticalOffset()

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 for scenario, (water|bathy)VerticalScaling will be guessed form the value initial values. bathyVerticalOffset is always 0 in this case.

7.44.2 Constructor & Destructor Documentation

7.44.2.1 virtual SWE_VisInfo::~SWE_VisInfo ( ) [inline], [virtual]

Empty virtual destructor

7.44.3 Member Function Documentation

7.44.3.1 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.

7.44.3.2 virtual float SWE_VisInfo::bathyVerticalScaling ( ) [inline], [virtual]

Returns

The vertical scaling factor for the bathymetry

Reimplemented in SWE_AsagiJapanSmallVisInfo.

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

Returns

The vertical scaling factor of the water

Reimplemented in SWE_AsagiJapanSmallVisInfo.

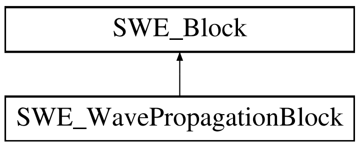
The documentation for this class was generated from the following file:

- /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_VisInfo.hh

7.45 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)
- virtual void simulateTimestep (float dt)
- void computeNumericalFluxes ()
- void updateUnknowns (float dt)
- float simulate (float i_tStart, float i_tEnd)
- virtual ~SWE_WavePropagationBlock ()

Additional Inherited Members

7.45.1 Detailed Description

SWE_WavePropagationBlock is an implementation of the SWE_Block 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)

7.45.2 Constructor & Destructor Documentation

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

Constructor of a SWE_WavePropagationBlock.

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) -> 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) *   *
*         *
```

7.45.2.2 virtual SWE_WavePropagationBlock::∼SWE_WavePropagationBlock ( ) [inline], [virtual]

Destructor of a SWE_WavePropagationBlock.

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

7.45.3 Member Function Documentation

7.45.3.1 void SWE_WavePropagationBlock::computeNumericalFluxes ( ) [virtual]

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

Implements SWE_Block.

7.45.3.2 float SWE_WavePropagationBlock::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

Implements SWE_Block.

7.45.3.3 void SWE_WavePropagationBlock::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

Implements SWE_Block.

7.45.3.4 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.

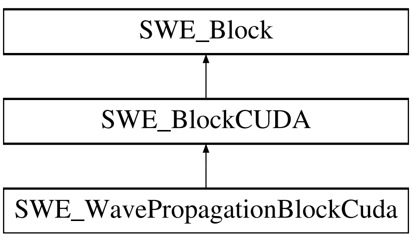
The documentation for this class was generated from the following files:

- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/SWE_WavePropagationBlock.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/SWE_WavePropagationBlock.cpp

7.46 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)
- ~SWE_WavePropagationBlockCuda ()
- void simulateTimestep (float i_dT)
- float simulate (float, float)
- void computeNumericalFluxes ()
- void updateUnknowns (const float i_deltaT)

Additional Inherited Members

7.46.1 Detailed Description

SWE_WavePropagationBlockCuda 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)

7.46.2 Constructor & Destructor Documentation

7.46.2.1 SWE_WavePropagationBlockCuda::SWE_WavePropagationBlockCuda (int *l_nx*, int *l_ny*, float *l_dx*, float *l_dy*)

Constructor of a SWE_WavePropagationBlockCuda.

Allocates the variables for the simulation: Please note: The definition of indices changed in contrast to the CPU-- Implementation.

unknowns hd,hud,hvd,bd stored on the CUDA device are defined for grid indices [0,..,nx+1]*[0,..,ny+1] (-> Abstract class SWE_BlockCUDA) -> computational domain is [1,..,nx]*[1,..,ny] -> plus ghost cell layer

net-updates are defined for edges with indices [0,..,nx]*[0,..,ny] for horizontal and vertical edges for simplicity (one layer is not necessary).

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

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


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

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

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

```
*          *
***********
```

Parameters

i_offsetX	spatial offset of the block in x-direction.
i_offsetY	spatial offset of the offset in y-direction.
i_cudaDevice	ID of the CUDA-device, which should be used.

7.46.2.2 SWE_WavePropagationBlockCuda::~SWE_WavePropagationBlockCuda ( )

Destructor of a SWE_WavePropagationBlockCuda.

Frees all of the memory, which was allocated within the constructor. Resets the CUDA device: Useful if error occured and printf is used on the device (buffer).

7.46.3 Member Function Documentation

7.46.3.1 void SWE_WavePropagationBlockCuda::computeNumericalFluxes ( ) [virtual]

Compute the numerical fluxes (net-update formulation here) on all edges.

The maximum wave speed is computed within the net-updates kernel for each CUDA-block. To finalize the method the Thrust-library is called, which does the reduction over all blockwise maxima. In the wave speed reduction step the actual cell width in x- and y-direction is not taken into account.

TODO: A splitting or direct computation of the time step width might increase the total time step size. Example: dx = 11, dy = 6; max wave speed in x-direction: 10 max wave speed in y-direction: 5.5 max wave speed in both direction: 10

=> maximum time step (current implementation): min(11/10, 6/10) = 0.6 => maximum time step (splitting the dimensions): min(11/10, 6/5.5) = 1.09.. **Row-major vs column-major**

C/C++ arrays are row-major whereas warps are constructed in column-major order from threads/blocks. To get coalesced memory access in CUDA, we can use a 2-dimensional CUDA structure but we have to switch x and y inside a block.

This means, we have to switch threadIdx.x <-> threadIdx.y as well as blockDim.x <-> blockDim.y. Important: blockDim has to be switched for the kernel call as well!

definition of one CUDA-block. Typical size are 8∗8 or 16∗16

Definition of the "main" CUDA-grid. This grid covers only edges 0..#(edges in x-direction)-2 and 0..#(edges in y-direction)-2.

An example with a computational domain of size nx = 24, ny = 16 with a 1 cell ghost layer would result in a grid with (nx+2)∗(ny+2) = (26∗18) cells and (nx+1)∗(ny+1) = (25∗17) edges.

The CUDA-blocks (here 8∗8) mentioned above would cover all edges except the ones lying between the computational domain and the right/top ghost layer:

```
                                                          *
                                                         **        top ghost layer,
                                                        ********   cell ids
                        *******************************  **        = (*, ny+1)
                        *         *         *         *   *
                        *   8*8   *   8*8   *   8*8   *
                        *  block  *  block  *  block  *
                        *         *         *         *
                        *******************************
                        *         *         *         *
                        *   8*8   *   8*8   *   8*8   *
                    *   *  block  *  block  *  block  *
    bottom         **   *         *         *         *
    ghost     ******** *******************************
    layer,         **
```

```
cell ids        *   *                              *
=(*,0)             ***                            ***
                    *                              *
                    *                              *
              left ghost layer,             right ghost layer,
              cell ids = (0,*)             cell ids = (nx+1, *)
```

Implements SWE_Block.

7.46.3.2 __host__ float SWE_WavePropagationBlockCuda::simulate (float *tStart*, float *tEnd*) [virtual]

perform forward-Euler time steps, starting with simulation time tStart,: until simulation time tEnd is reached; device-global variables hd, hud, hvd are updated; unknowns h, hu, hv in main memory are not updated. Ghost layers and bathymetry sources are updated between timesteps. intended as main simulation loop between two checkpoints

Implements SWE_Block.

7.46.3.3 __host__ void SWE_WavePropagationBlockCuda::simulateTimestep (float *i_dT*) [virtual]

Compute a single global time step of a given time step width. Remark: The user has to take care about the time step width. No additional check is done. The time step width typically available after the computation of the numerical fluxes (hidden in this method).

First the net-updates are computed. Then the cells are updated with the net-updates and the given time step width.

Parameters

i_dT	time step width in seconds.

Implements SWE_Block.

7.46.3.4 void SWE_WavePropagationBlockCuda::updateUnknowns (const float *i_deltaT*) [virtual]

Update the cells with a given global time step.

Parameters

i_deltaT	time step size.

definition of one CUDA-block. Typical size are 8∗8 or 16∗16

definition of the CUDA-grid.

Implements SWE_Block.

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/cuda/SWE_WavePropagationBlockCuda.hh
- /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/cuda/SWE_WavePropagationBlockCuda.cu

7.47 Text Class Reference

Public Member Functions

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

7.47.1 Member Function Documentation

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

Returns

True there are more textures

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/text.h
- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/text.cpp

7.48 tools::Args Class Reference

```
#include <args.h>
```

Public Member Functions

- **Args** (int argc, char ∗∗argv)
- unsigned int **size** ()
- unsigned int **timeSteps** ()

7.48.1 Detailed Description

Parse command line arguments

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/tools/args.h

7.49 tools::Logger Class Reference

Public Types

- enum **Level** { **INFO**, **WARNING**, **ERROR** }

Public Member Functions

- void **setOutputStream** (std::ostream &output)
- void **log** (std::string &message, Level level=INFO)
- void **log** (const char ∗message, Level level=INFO)
- void **info** (std::string &message)
- void **info** (const char ∗message)
- std::ostream & **info** ()
- void **warning** (std::string &message)
- void **warning** (const char ∗message)
- std::ostream & **warning** ()
- void **error** (std::string &message)
- void **error** (const char ∗message)
- template<typename T >
  Logger & operator<< (T value)

- Logger & operator<< (std::ostream &(*func)(std::ostream &))
- 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 updateCpuTime ()
- void updateCpuCommunicationTime ()
- void **resetCpuClockToCurrentTime** ()
- void **resetCpuCommunicationClockToCurrentTime** ()
- void initWallClockTime (const double i_wallClockTime)
- void printWallClockTime (const double i_wallClockTime, const std::string i_wallClockTimeMessage="wall clock time")
- void printCpuTime (const std::string i_cpuTimeMessage="CPU time")
- void printCpuCommunicationTime (const std::string i_cpuCommunicationTimeMessage="CPU + communication time")
- void printIterationsDone (unsigned int i_iterations, std::string i_iterationMessage="iterations done")

Static Public Attributes

- static Logger logger

7.49.1 Constructor & Destructor Documentation

7.49.1.1 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" " Depa ://www5.in.tum.de/SWE\n" "\n" "SWE comes with ABSOLUTELY NO WARRANTY.\n" "SWE is f 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).

7.49.1.2 virtual tools::Logger::~Logger ( ) [inline], [virtual]

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

7.49.2 Member Function Documentation

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

Default output stream of the logger.

Returns

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

7.49.2.2 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.

7.49.2.3 template<typename T > Logger& tools::Logger::operator<< (T *value*) [inline]

Can be used to print arbitrary info messages. Does not append std::endl.

7.49.2.4 Logger& tools::Logger::operator<< (std::ostream &(*)(std::ostream &) *func*) [inline]

Allow to print std::endl

7.49.2.5 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.

7.49.2.6 void tools::Logger::printCpuCommunicationTime (const std::string *i_cpuCommunicationTimeMessage* = "CPU + communication time") [inline]

Print elapsed CPU + communication time.

Parameters

i_cpu-Communication-TimeMessage	CPU + communication time message.

7.49.2.7 void tools::Logger::printCpuTime (const std::string *i_cpuTimeMessage* = "CPU time") [inline]

Print elapsed CPU time.

Parameters

i_cpuTime-Message	cpu time message.

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

Print the finish message.

7.49.2.9 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

7.49.2.10 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.

7.49.2.11 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.

7.49.2.12 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.

7.49.2.13 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.

7.49.2.14 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.

7.49.2.15 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.

7.49.2.16 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.

7.49.2.17 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.

7.49.2.18 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)

7.49.2.19 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.

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

Print an arbitrary string.

Parameters

i_string	some string.

7.49.2.21 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.

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

Print the welcome message.

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

Set the process rank.

Parameters

i_processRank	process rank.

7.49.2.24 void tools::Logger::updateCpuCommunicationTime ( ) [inline]

Update the CPU-Communication time.

7.49.2.25 void tools::Logger::updateCpuTime ( ) [inline]

Update the CPU time.

7.49.3 Member Data Documentation

7.49.3.1 static Logger tools::Logger::logger [static]

The logger all classes shoud use

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/tools/logger.h
- /home/raphael/Programmieren/BPraktikum/SWE/src/tools/Logger.hh
- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/tools/logger.cpp
- /home/raphael/Programmieren/BPraktikum/SWE/src/tools/Logger.cpp

7.50 tools::ProgressBar Class Reference

Public Member Functions

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

7.50.1 Member Function Documentation

7.50.1.1 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:

- /home/raphael/Programmieren/BPraktikum/SWE/src/tools/ProgressBar.hh

7.51 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 ()

7.51.1 Member Function Documentation

7.51.1.1 void VBO::finialize ( ) [inline]

Frees all associated memory

7.51.1.2 GLuint VBO::getName ( ) [inline]

Returns

The OpenGL name of the buffer

7.51.1.3 void VBO::init ( )

Initializes the object

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

- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/vbo.h
- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/vbo.cpp

7.52 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**

7.52.1 Constructor & Destructor Documentation

7.52.1.1 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)

7.52.1.2 Visualization::∼Visualization ( )

Destructor (see note below)

7.52.2 Member Function Documentation

7.52.2.1 void Visualization::cleanUp ( )

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

7.52.2.2 cudaGraphicsResource ** Visualization::getCudaNormalsPtr ( )

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

7.52.2.3 cudaGraphicsResource ** Visualization::getCudaWaterSurfacePtr ( )

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

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

Allocates memory for vertices and other geometry data.

Parameters

sim	instance of the simulation class

7.52.2.5 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

7.52.2.6 void Visualization::renderDisplay ( )

Main rendering function. Draws the scene and updates screen

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

Gets called when window gets resized

Parameters

newWidth	new window width in pixels
newHeight	height in pixels

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

Sets current rendering mode

Parameters

mode	rendering mode

7.52.2.9 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:

- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/visualization.h
- /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/visualization.cpp

7.53 WavePropagation Class Reference

```
#include <WavePropagation.h>
```

Public Member Functions

- WavePropagation (T ∗h, T ∗hu, T ∗b, unsigned int size, T cellSize)
- T computeNumericalFluxes ()
- void updateUnknowns (T dt)
- void setOutflowBoundaryConditions ()

7.53.1 Detailed Description

Allocated variables: unknowns h,hu are defined on grid indices [0,..,n+1] (done by the caller) -> computational domain is [1,..,nx] -> plus ghost cell layer

net-updates are defined for edges with indices [0,..,n]

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

```
*   (i-1)   *    (i)    *

      *
     ***
    *****
      *
      *

   NetUpdatesLeft(i-1)
             or
   NetUpdatesRight(i-1)
```

7.53.2 Constructor & Destructor Documentation

7.53.2.1 WavePropagation::WavePropagation (T ∗ *h*, T ∗ *hu*, T ∗ *b*, unsigned int *size*, T *cellSize*) [inline]

Parameters

b	elevation of the ocean floor
size	Domain size (= number of cells) without ghost cells
cellSize	Size of one cell

7.53.3 Member Function Documentation

7.53.3.1 T WavePropagation::computeNumericalFluxes ( )

Computes the net-updates from the unknowns

Returns

The maximum possible time step

7.53.3.2 void WavePropagation::setOutflowBoundaryConditions ( )

Updates h and hu according to the outflow condition to both boundaries

7.53.3.3 void WavePropagation::updateUnknowns (T *dt*)

Update the unknowns with the already computed net-updates

Parameters

dt	Time step size

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/WavePropagation.h
- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/WavePropagation.cpp

7.54 writer::ConsoleWriter Class Reference

```
#include <ConsoleWriter.h>
```

Public Member Functions

- **ConsoleWriter** (std::ostream &ostream=std::cout)
- void write (const T ∗h, const T ∗hu, unsigned int size)

7.54.1 Detailed Description

A simple writer class, that writes h and hu to stdout (or another ostream)

7.54.2 Member Function Documentation

7.54.2.1 void writer::ConsoleWriter::write (const T ∗ *h*, const T ∗ *hu*, unsigned int *size*) [inline]

Writes all values (without boundary values) to the ostream

Parameters

size	Number of cells (without boundary values)

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/writer/ConsoleWriter.h

7.55 writer::VtkWriter Class Reference

```
#include <VtkWriter.h>
```

Public Member Functions

- **VtkWriter** (const std::string &basename="swe1d", const T cellSize=1)
- void write (const T time, const T ∗h, const T ∗hu, const T ∗b, unsigned int size)

7.55.1 Detailed Description

A writer class that generates vtk files

7.55.2 Member Function Documentation

7.55.2.1 void writer::VtkWriter::write (const T *time*, const T $*$ *h*, const T $*$ *hu*, const T $*$ *b*, unsigned int *size*) [inline]

Writes all values to vtk file

Parameters

size	Number of cells (without boundary values)

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

- /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/writer/VtkWriter.h

Chapter 8

File Documentation

8.1 mainpage.txt File Reference

8.1.1 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)

8.1.2 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/.

8.1.3 DESCRIPTION

Main section of the doxygen documentation.

8.2 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/cuda/SWE_BlockCUDA.cu File Reference

```
#include "SWE_BlockCUDA.hh"
#include "SWE_BlockCUDA_kernels.hh"
#include "tools/help.hh"
#include "tools/Logger.hh"
#include <cassert>
#include <cstdlib>
#include <cmath>
```

Functions

- void **checkCUDAError** (const char ∗msg)
- void **tryCUDA** (cudaError_t err, const char ∗msg)

8.2.1 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.)

8.2.2 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/.

8.2.3 DESCRIPTION

TODO

8.3 /home/raphael/Programmieren/BPraktikum/SWE/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

8.3.1 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.)

8.3.2 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/.

8.3.3 DESCRIPTION

TODO

8.3.4 Function Documentation

8.3.4.1 __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)

8.3.4.2 __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)

8.3.4.3 __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)

8.4 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/cuda/SWE_BlockCUDA_-kernels.cu File Reference

```
#include "SWE_BlockCUDA.hh"
#include "SWE_BlockCUDA_kernels.hh"
```

Functions

- __global__ void kernelHdBufferEdges (float *hd, int nx, int ny)
- __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)

8.4.1 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)

8.4.2 LICENSE

8.4.3 DESCRIPTION

TODO

8.4.4 Function Documentation

8.4.4.1 __global__ void kernelBottomBoundary (float ∗ *hd*, float ∗ *hud*, float ∗ *hvd*, int *nx*, int *ny*, BoundaryType *bound*)

CUDA kernel to set bottom boundary layer for conditions WALL & OUTFLOW blockIdx.x and threadIdx.x loop over the boundary elements SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.4.4.2 __global__ void kernelBottomCopyLayer (float ∗ *hd*, float ∗ *hud*, float ∗ *hvd*, float ∗ *bottomCopyLayer*, int *nx*, int *ny*)

CUDA kernel to update bottom copy layer according (for boundary conditions PASSIVE and CONNECT) blockIdx.x and threadIdx.x loop over the boundary elements. Note that diagonal elements are currently not copied! SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.4.4.3 __global__ void kernelBottomGhostBoundary (float ∗ *hd*, float ∗ *hud*, float ∗ *hvd*, float ∗ *bottomGhostLayer*, int *nx*, int *ny*)

CUDA kernel to set bottom boundary layer according to the external ghost layer status (conditions PASSIVE and CONNECT) blockIdx.x and threadIdx.x loop over the boundary elements. Note that diagonal elements are currently not copied! SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.4.4.4 __global__ void kernelHdBufferEdges (float ∗ *hd*, int *nx*, int *ny*)

Sets corner values of hd (only needed for visualization)

Parameters

hd	h-values on device

8.4.4.5 __global__ void kernelLeftBoundary (float ∗ *hd*, float ∗ *hud*, float ∗ *hvd*, int *nx*, int *ny*, BoundaryType *bound*)

CUDA kernel to set left boundary layer for conditions WALL & OUTFLOW blockIdx.y and threadIdx.y loop over the boundary elements SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.4.4.6 __global__ void kernelRightBoundary (float ∗ *hd*, float ∗ *hud*, float ∗ *hvd*, int *nx*, int *ny*, BoundaryType *bound*)

CUDA kernel to set right boundary layer for conditions WALL & OUTFLOW blockIdx.y and threadIdx.y loop over the boundary elements SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.4.4.7 __global__ void kernelTopBoundary (float ∗ *hd*, float ∗ *hud*, float ∗ *hvd*, int *nx*, int *ny*, BoundaryType *bound*)

CUDA kernel to set bottom boundary layer for conditions WALL & OUTFLOW blockIdx.x and threadIdx.x loop over the boundary elements

8.4.4.8 __global__ void kernelTopCopyLayer (float ∗ *hd*, float ∗ *hud*, float ∗ *hvd*, float ∗ *topCopyLayer*, int *nx*, int *ny*)

CUDA kernel to set top boundary layer according to the external ghost layer status (conditions PASSIVE and CONNECT) blockIdx.x and threadIdx.x loop over the boundary elements Note that diagonal elements are currently not copied! SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.4.4.9 __global__ void kernelTopGhostBoundary (float ∗ *hd,* float ∗ *hud,* float ∗ *hvd,* float ∗ *topGhostLayer,* int *nx,* int *ny*)

CUDA kernel to set top boundary layer according to the external ghost layer status (conditions PASSIVE and CONNECT) blockIdx.x and threadIdx.x loop over the boundary elements Note that diagonal elements are currently not copied! SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.5 /home/raphael/Programmieren/BPraktikum/SWE/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)

8.5.1 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`)

8.5.2 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/`.

8.5.3 DESCRIPTION

TODO

8.5.4 Function Documentation

8.5.4.1 __global__ void kernelBottomBoundary (float * *hd*, float * *hud*, float * *hvd*, int *nx*, int *ny*, BoundaryType *bound*)

CUDA kernel to set bottom boundary layer for conditions WALL & OUTFLOW blockIdx.x and threadIdx.x loop over the boundary elements SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.5.4.2 __global__ void kernelBottomCopyLayer (float * *hd*, float * *hud*, float * *hvd*, float * *bottomCopyLayer*, int *nx*, int *ny*)

CUDA kernel to update bottom copy layer according (for boundary conditions PASSIVE and CONNECT) blockIdx.x and threadIdx.x loop over the boundary elements. Note that diagonal elements are currently not copied! SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.5.4.3 __global__ void kernelBottomGhostBoundary (float * *hd*, float * *hud*, float * *hvd*, float * *bottomGhostLayer*, int *nx*, int *ny*)

CUDA kernel to set bottom boundary layer according to the external ghost layer status (conditions PASSIVE and CONNECT) blockIdx.x and threadIdx.x loop over the boundary elements. Note that diagonal elements are currently not copied! SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.5.4.4 __global__ void kernelHdBufferEdges (float * *hd*, int *nx*, int *ny*)

Sets corner values of hd (only needed for visualization)

Parameters

hd	h-values on device

8.5.4.5 __global__ void kernelLeftBoundary (float * *hd*, float * *hud*, float * *hvd*, int *nx*, int *ny*, BoundaryType *bound*)

CUDA kernel to set left boundary layer for conditions WALL & OUTFLOW blockIdx.y and threadIdx.y loop over the boundary elements SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.5.4.6 __global__ void kernelMaximum (float * *maxhd*, float * *maxvd*, int *start*, int *size*)

CUDA kernel for maximum reduction required to compute maximum water height and velocities to determine allow time step

8.5.4.7 __global__ void kernelRightBoundary (float * *hd*, float * *hud*, float * *hvd*, int *nx*, int *ny*, BoundaryType *bound*)

CUDA kernel to set right boundary layer for conditions WALL & OUTFLOW blockIdx.y and threadIdx.y loop over the boundary elements SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.5.4.8 __global__ void kernelTopBoundary (float * *hd*, float * *hud*, float * *hvd*, int *nx*, int *ny*, BoundaryType *bound*)

CUDA kernel to set bottom boundary layer for conditions WALL & OUTFLOW blockIdx.x and threadIdx.x loop over the boundary elements

8.5.4.9 __global__ void kernelTopCopyLayer (float * *hd*, float * *hud*, float * *hvd*, float * *topCopyLayer*, int *nx*, int *ny*)

CUDA kernel to set top boundary layer according to the external ghost layer status (conditions PASSIVE and CO-NNECT) blockIdx.x and threadIdx.x loop over the boundary elements Note that diagonal elements are currently not copied! SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.5.4.10 __global__ void kernelTopGhostBoundary (float * *hd,* float * *hud,* float * *hvd,* float * *topGhostLayer,* int *nx,* int *ny*)

CUDA kernel to set top boundary layer according to the external ghost layer status (conditions PASSIVE and CO-NNECT) blockIdx.x and threadIdx.x loop over the boundary elements Note that diagonal elements are currently not copied! SWE_Block size ny is assumed to be a multiple of the TILE_SIZE

8.6 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/cuda/SWE_WavePropagationBlockCuda.cu File Reference

```
#include "SWE_WavePropagationBlockCuda.hh"
#include "SWE_BlockCUDA.hh"
#include "SWE_WavePropagationBlockCuda_kernels.hh"
#include "tools/Logger.hh"
#include <cassert>
#include <cuda.h>
#include <cuda_runtime_api.h>
#include <thrust/device_vector.h>
```

8.6.1 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.)

8.6.2 LICENSE

8.6.3 DESCRIPTION

SWE_Block in CUDA, which uses solvers in the wave propagation formulation.

8.7 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/cuda/SWE_WavePropagationBlockCuda.hh File Reference

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

Classes

- class SWE_WavePropagationBlockCuda

8.7.1 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.)

8.7.2 LICENSE

8.7.3 DESCRIPTION

SWE_Block in CUDA, which uses solvers in the wave propagation formulation.

8.8 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/cuda/SWE_WavePropagationBlockCuda_kernels.cu File Reference

```
#include "SWE_BlockCUDA.hh"
#include "SWE_WavePropagationBlockCuda_kernels.hh"
#include <cmath>
#include <cstdio>
#include "solvers/FWaveCuda.h"
```

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_huNet-

UpdatesRightD, 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, const int i_offsetY, const int i_blockOffSetX, const int i_blockOffSetY)
- __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_ny)

8.8.1 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.)

8.8.2 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/.

8.8.3 DESCRIPTION

CUDA Kernels for a SWE_Block, which uses solvers in the wave propagation formulation.

8.8.4 Function Documentation

8.8.4.1 __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,* const int *i_offsetY,* const int *i_blockOffSetX,* const int *i_blockOffSetY*)

The compute net-updates kernel calls the solver for a defined CUDA-Block and does a reduction over the computed wave speeds within this block.

Remark: In overall we have nx+1 / ny+1 edges. Therefore the edges "simulation domain"/"top ghost layer" and "simulation domain"/"right ghost layer" will not be computed in a typical call of the function: computeNetUpdatesKernel<<<dimGrid,dimBlock>>>(hd, hud, hvd, bd, hNetUpdatesLeftD, hNetUpdatesRightD, huNetUpdatesLeftD, huNetUpdatesRightD, hNetUpdatesBelowD, hNetUpdatesAboveD, hvNetUpdatesBelowD, hvNetUpdatesAboveD, l_maximumWaveSpeedsD, i_nx, i_ny); To reduce the effect of branch-mispredictions the kernel provides optional offsets, which can be used to compute the missing edges.

SWE_WavePropagationBlockCuda::computeNumericalFluxes() explains the coalesced memory access.

Parameters

i_h	water heights (CUDA-array).
i_hu	momentums in x-direction (CUDA-array).
i_hv	momentums in y-direction (CUDA-array).
i_b	bathymetry values (CUDA-array).
o_hNetUpdatesLeftD	left going net-updates for the water height (CUDA-array).
o_hNetUpdatesRightD	right going net-updates for the water height (CUDA-array).
o_huNetUpdatesLeftD	left going net-updates for the momentum in x-direction (CUDA-array).
o_huNetUpdatesRightD	right going net-updates for the momentum in x-direction (CUDA-array).
o_hNetUpdatesBelowD	downwards going net-updates for the water height (CUDA-array).
o_hNetUpdatesAboveD	upwards going net-updates for the water height (CUDA-array).
o_hvNetUpdatesBelowD	downwards going net-updates for the momentum in y-direction (CUDA-array).
o_hvNetUpdatesAboveD	upwards going net-updates for the momentum in y-direction (CUDA-array).
o_maximumWaveSpeeds	maximum wave speed which occurred within the CUDA-block is written here (CUDA-array).
i_nx	number of cells within the simulation domain in x-direction (excludes ghost layers).
i_ny	number of cells within the simulation domain in y-direction (excludes ghost layers).
i_offsetX	cell/edge offset in x-direction.
i_offsetY	cell/edge offset in y-direction.

array maximum wave speed within this CUDA-block

thread local index in the shared maximum wave speed array

index (l_cellIndexI,l_cellIndexJ) of the cell lying on the right side of the edge/above the edge where the thread works on.

array which holds the thread local net-updates.

location of the thread local cells in the global CUDA-arrays.

reduction partner for a thread

Position of the maximum wave speed in the global device array.

In the 'main' part (e.g. gridDim = nx/TILE_SIZEm ny/TILE_SIZE) the position is simply given by the blockId in x- and y-direction with a stride of gridDim.x + 1. The +1 results from the speeds in the 'boundary' case, see below.

In the 'boundary' case, where the edges lie between the computational domain and the right/top ghost layer, this is more complicated. In this case block offsets in x- and y-direction are used. The offsets define how many blocks in the resp. direction have to be added to get a valid result. Computational domain - right ghost layer: In this case the dimension of the grid in x-direction is 1. Computational domain - top ghost layer: In this case the dimension of the grid in y-direction is 1.

Same Example as in SWE_WavePropagationBlockCuda::computeNumericalFluxes(), assume the CUDA-grid/-blocks has the following layout:

```
                                 *
                                **        top ghost layer,
* block 8 * block 9 * block 10* ********   cell ids
******************************** **
*         *         *         *   *
*  block  *  block  *  block  * b
*    4    *    5    *    6    * 7
*         *         *         *
********************************
```

```
                   *         *         *         *
                   *  block  *  block  *  block  * b
               *   *    0    *    1    *    2    * 3
bottom         **  *         *         *         *
ghost      ******** *******************************
layer          **
               *   *                              *
                  ***                            ***
                   *                              *
                   *                              *
             left ghost layer               right ghost layer
```

This results in a 'main' part containing of (3∗2) blocks and two 'boundary' parts containing of (1∗2) blocks and (3∗1) blocks.

The maximum wave speed array on the device represents therefore logically a (4 ∗ 3)-1 2D-array (-1: no block on the top right). The 'main' part writes into cells 0, 1, 2, 4, 5 and 6. The 'computational domain - right ghost layer' part writes into 3 and 7 with offset in x-direction = 3 The 'computational domain - top ghost layer' part writes into 8, 9, 10 with offset in y-direction = 2

8.8.4.2 __device__ int computeOneDPositionKernel (const int *i_i*, const int *i_j*, const int *i_ny*) [inline]

Compute the position of 2D coordinates in a 1D array. array[i][j] -> i ∗ ny + j

Parameters

i_i	row index.
i_j	column index.
i_ny	#(cells in y-direction).

Returns

1D index.

8.8.4.3 __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*)

The "update unknowns"-kernel updates the unknowns in the cells with precomputed net-updates.

SWE_WavePropagationBlockCuda::computeNumericalFluxes() explains the coalesced memory access.

Parameters

i_hNetUpdates-LeftD	left going net-updates for the water height (CUDA-array).
i_hNetUpdates-RightD	right going net-updates for the water height (CUDA-array).
i_huNetUpdates-LeftD	left going net-updates for the momentum in x-direction (CUDA-array).
i_huNetUpdates-RightD	right going net-updates for the momentum in x-direction (CUDA-array).
i_hNetUpdates-BelowD	downwards going net-updates for the water height (CUDA-array).

i_hNetUpdates-AboveD	upwards going net-updates for the water height (CUDA-array).
i_hvNetUpdates-BelowD	downwards going net-updates for the momentum in y-direction (CUDA-array).
i_hvNetUpdates-AboveD	upwards going net-updates for the momentum in y-direction (CUDA-array).
io_h	water heights (CUDA-array).
io_hu	momentums in x-direction (CUDA-array).
io_hv	momentums in y-direction (CUDA-array).
i_updateWidthX	update width in x-direction.
i_updateWidthY	update width in y-direction.
i_nx	number of cells within the simulation domain in x-direction (excludes ghost layers).
i_ny	number of cells within the simulation domain in y-direction (excludes ghost layers).

cell indices (l_cellIndexI,l_cellIndexJ) of the cell which the thread updates.

location of the thread local cell in the global CUDA-arrays.

positions of the net-updates in the global CUDA-arrays.

Compute the positions of the net updates relative to a given cell

```
                 netUpdateRight(i-1, j)

                          |           |
                          |           |
                          |           |
netUpdateBelow(i,j) -----*************-----
                          *           *
                          * cell(i,j) *
                          *           *
                    -----*************------ netUpdateAbove(i, j-1)
                          |           |
                          |           |
                          |           |
                              netUpdatesLeft(i,j)
```

8.9 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/cuda/SWE_WavePropagation-BlockCuda_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)

8.9.1 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)

8.9.2 LICENSE

8.9.3 DESCRIPTION

CUDA Kernels for a SWE_Block, which uses solvers in the wave propagation formulation.

8.9.4 Function Documentation

8.9.4.1 __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*, const int *i_offsetY*, const int *i_blockOffSetX*, const int *i_blockOffSetY*)

The compute net-updates kernel calls the solver for a defined CUDA-Block and does a reduction over the computed wave speeds within this block.

Remark: In overall we have nx+1 / ny+1 edges. Therefore the edges "simulation domain"/"top ghost layer" and "simulation domain"/"right ghost layer" will not be computed in a typical call of the function: computeNetUpdatesKernel<<<dimGrid,dimBlock>>>(hd, hud, hvd, bd, hNetUpdatesLeftD, hNetUpdatesRightD, huNetUpdatesLeftD, huNetUpdatesRightD, hNetUpdatesBelowD, hNetUpdatesAboveD, hvNetUpdatesBelowD, hvNetUpdatesAboveD, l_maximumWaveSpeedsD, i_nx, i_ny); To reduce the effect of branch-mispredictions the kernel provides optional offsets, which can be used to compute the missing edges.

SWE_WavePropagationBlockCuda::computeNumericalFluxes() explains the coalesced memory access.

Parameters

i_h	water heights (CUDA-array).
i_hu	momentums in x-direction (CUDA-array).
i_hv	momentums in y-direction (CUDA-array).
i_b	bathymetry values (CUDA-array).
o_hNetUpdatesLeftD	left going net-updates for the water height (CUDA-array).
o_hNetUpdatesRightD	right going net-updates for the water height (CUDA-array).
o_huNetUpdatesLeftD	left going net-updates for the momentum in x-direction (CUDA-array).

o_huNetUpdatesRightD	right going net-updates for the momentum in x-direction (CUDA-array).
o_hNetUpdatesBelowD	downwards going net-updates for the water height (CUDA-array).
o_hNetUpdatesAboveD	upwards going net-updates for the water height (CUDA-array).
o_hvNetUpdatesBelowD	downwards going net-updates for the momentum in y-direction (CUDA-array).
o_hvNetUpdatesAboveD	upwards going net-updates for the momentum in y-direction (CUDA-array).
o_maximumWaveSpeeds	maximum wave speed which occurred within the CUDA-block is written here (CUDA-array).
i_nx	number of cells within the simulation domain in x-direction (excludes ghost layers).
i_ny	number of cells within the simulation domain in y-direction (excludes ghost layers).
i_offsetX	cell/edge offset in x-direction.
i_offsetY	cell/edge offset in y-direction.

array maximum wave speed within this CUDA-block

thread local index in the shared maximum wave speed array

index (l_cellIndexI,l_cellIndexJ) of the cell lying on the right side of the edge/above the edge where the thread works on.

array which holds the thread local net-updates.

location of the thread local cells in the global CUDA-arrays.

reduction partner for a thread

Position of the maximum wave speed in the global device array.

In the 'main' part (e.g. gridDim = nx/TILE_SIZEm ny/TILE_SIZE) the position is simply given by the blockId in x- and y-direction with a stride of gridDim.x + 1. The +1 results from the speeds in the 'boundary' case, see below.

In the 'boundary' case, where the edges lie between the computational domain and the right/top ghost layer, this is more complicated. In this case block offsets in x- and y-direction are used. The offsets define how many blocks in the resp. direction have to be added to get a valid result. Computational domain - right ghost layer: In this case the dimension of the grid in x-direction is 1. Computational domain - top ghost layer: In this case the dimension of the grid in y-direction is 1.

Same Example as in SWE_WavePropagationBlockCuda::computeNumericalFluxes(), assume the CUDA-grid/-blocks has the following layout:

```
                                                      *
                                                    **        top ghost layer,
                   * block 8 * block 9 * block 10* ********   cell ids
                   ******************************** **
                   *         *         *         *    *
                   *  block  *  block  *  block  * b
                   *     4   *     5   *     6   * 7
                   *         *         *         *
                   ********************************
                   *         *         *         *
                   *  block  *  block  *  block  * b
               *   *     0   *     1   *     2   * 3
bottom         **  *         *         *         *
ghost     ******** ********************************
layer          **
               *   *                              *
                  ***                            ***
                   *                              *
                   *                              *
             left ghost layer              right ghost layer
```

This results in a 'main' part containing of (3∗2) blocks and two 'boundary' parts containing of (1∗2) blocks and (3∗1) blocks.

The maximum wave speed array on the device represents therefore logically a (4 ∗ 3)-1 2D-array (-1: no block on the top right). The 'main' part writes into cells 0, 1, 2, 4, 5 and 6. The 'computational domain - right ghost layer' part writes into 3 and 7 with offset in x-direction = 3 The 'computational domain - top ghost layer' part writes into 8, 9, 10 with offset in y-direction = 2

8.9.4.2 __device__ int computeOneDPositionKernel (const int *i_i*, const int *i_j*, const int *i_ny*) [inline]

Compute the position of 2D coordinates in a 1D array. array[i][j] -> i ∗ ny + j

Parameters

i_i	row index.
i_j	column index.
i_ny	#(cells in y-direction).

Returns

1D index.

8.9.4.3 __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*)

The "update unknowns"-kernel updates the unknowns in the cells with precomputed net-updates.

SWE_WavePropagationBlockCuda::computeNumericalFluxes() explains the coalesced memory access.

Parameters

i_hNetUpdates-LeftD	left going net-updates for the water height (CUDA-array).
i_hNetUpdates-RightD	right going net-updates for the water height (CUDA-array).
i_huNetUpdates-LeftD	left going net-updates for the momentum in x-direction (CUDA-array).
i_huNetUpdates-RightD	right going net-updates for the momentum in x-direction (CUDA-array).
i_hNetUpdates-BelowD	downwards going net-updates for the water height (CUDA-array).
i_hNetUpdates-AboveD	upwards going net-updates for the water height (CUDA-array).
i_hvNetUpdates-BelowD	downwards going net-updates for the momentum in y-direction (CUDA-array).
i_hvNetUpdates-AboveD	upwards going net-updates for the momentum in y-direction (CUDA-array).
io_h	water heights (CUDA-array).
io_hu	momentums in x-direction (CUDA-array).
io_hv	momentums in y-direction (CUDA-array).
i_updateWidthX	update width in x-direction.

i_updateWidthY	update width in y-direction.
i_nx	number of cells within the simulation domain in x-direction (excludes ghost layers).
i_ny	number of cells within the simulation domain in y-direction (excludes ghost layers).

cell indices (l_cellIndexI,l_cellIndexJ) of the cell which the thread updates.

location of the thread local cell in the global CUDA-arrays.

positions of the net-updates in the global CUDA-arrays.

Compute the positions of the net updates relative to a given cell

```
                  netUpdateRight(i-1, j)

                            |             |
                            |             |
                            |             |
 netUpdateBelow(i,j) -----**************-----
                            *             *
                            * cell(i,j) *
                            *             *
                       -----**************------ netUpdateAbove(i, j-1)
                            |             |
                            |             |
                            |             |
                                 netUpdatesLeft(i,j)
```

8.10 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/rusanov/SWE_RusanovBlock.cpp File Reference

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

Functions

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

8.10.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

8.10.2 LICENSE

8.10.3 DESCRIPTION

TODO

8.10.4 Function Documentation

8.10.4.1 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

8.11 /home/raphael/Programmieren/BPraktikum/SWE/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)

8.11.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

8.11.2 LICENSE

8.11.3 DESCRIPTION

TODO

8.11.4 Function Documentation

8.11.4.1 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

8.12 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/rusanov/SWE_RusanovBlockCUDA.cu File Reference

```
#include <math.h>
#include "tools/help.hh"
#include "SWE_BlockCUDA.hh"
#include "SWE_RusanovBlockCUDA.hh"
#include "SWE_RusanovBlockCUDA_kernels.hh"
```

Functions

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

8.12.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

8.12.2 LICENSE

8.12.3 DESCRIPTION

TODO

8.12.4 Function Documentation

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

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

8.13 /home/raphael/Programmieren/BPraktikum/SWE/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)

8.13.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

8.13.2 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/.

8.13.3 DESCRIPTION

TODO

8.13.4 Function Documentation

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

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

8.14 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/rusanov/SWE_RusanovBlockCUDA_kernels.cu File Reference

```
#include "SWE_BlockCUDA.hh"
#include "SWE_RusanovBlockCUDA_kernels.hh"
```

Functions

- __device__ float **computeFlux** (float fLow, float fHigh, float xiLow, float xiHigh, float llf)
- __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)

8.14.1 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)

8.14.2 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/.

8.14.3 DESCRIPTION

TODO

8.14.4 Function Documentation

8.14.4.1 __global__ void kernelComputeBathymetrySources (float * *hd*, float * *bd*, float * *Bxd*, float * *Byd*, int *ny*, float *g*)

computes the bathymetry source terms for the hu and hv equation for a given cell in the resp. array elements Bxd and Byd

8.14.4.2 __global__ void kernelComputeFluxesF (float ∗ *hd,* float ∗ *hud,* float ∗ *hvd,* float ∗ *Fhd,* float ∗ *Fhud,* float ∗ *Fhvd,* int *ny,* float *g,* float *llf,* int *istart*)

computes the flux vector components Fhd, Fhud and Fhvd for a single edge by calling the function computeFlux

8.14.4.3 __global__ void kernelComputeFluxesG (float ∗ *hd,* float ∗ *hud,* float ∗ *hvd,* float ∗ *Ghd,* float ∗ *Ghud,* float ∗ *Ghvd,* int *ny,* float *g,* float *llf,* int *jstart*)

computes the flux vector components Ghd, Ghud and Ghvd for a single edge by calling the function computeFlux

8.14.4.4 __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*)

CUDA kernel for Euler time step

8.14.4.5 __global__ void kernelMaximum (float ∗ *maxhd,* float ∗ *maxvd,* int *start,* int *size*)

CUDA kernel for maximum reduction required to compute maximum water height and velocities to determine allow time step

8.15 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/rusanov/SWE_Rusanov-BlockCUDA_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)

8.15.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

8.15.2 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.

8.15.3 DESCRIPTION

TODO

8.15.4 Function Documentation

8.15.4.1 __global__ void kernelComputeBathymetrySources (float ∗ *hd*, float ∗ *bd*, float ∗ *Bxd*, float ∗ *Byd*, int *ny*, float *g*)

computes the bathymetry source terms for the hu and hv equation for a given cell in the resp. array elements Bxd and Byd

8.15.4.2 __global__ void kernelComputeFluxesF (float ∗ *hd*, float ∗ *hud*, float ∗ *hvd*, float ∗ *Fhd*, float ∗ *Fhud*, float ∗ *Fhvd*, int *ny*, float *g*, float *llf*, int *istart*)

computes the flux vector components Fhd, Fhud and Fhvd for a single edge by calling the function computeFlux

8.15.4.3 __global__ void kernelComputeFluxesG (float ∗ *hd*, float ∗ *hud*, float ∗ *hvd*, float ∗ *Ghd*, float ∗ *Ghud*, float ∗ *Ghvd*, int *ny*, float *g*, float *llf*, int *jstart*)

computes the flux vector components Ghd, Ghud and Ghvd for a single edge by calling the function computeFlux

8.15.4.4 __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*)

CUDA kernel for Euler time step

8.15.4.5 __global__ void kernelMaximum (float ∗ *maxhd*, float ∗ *maxvd*, int *start*, int *size*)

CUDA kernel for maximum reduction required to compute maximum water height and velocities to determine allow time step

8.16 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/SWE_Block.cpp File Reference

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

8.16.1 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.)

8.16.2 LICENSE

8.16.3 DESCRIPTION

TODO

8.17 /home/raphael/Programmieren/BPraktikum/SWE/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

Variables

- const int **BLOCKS** =4

8.17.1 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.)

8.17.2 LICENSE

8.17.3 DESCRIPTION

TODO

8.18 /home/raphael/Programmieren/BPraktikum/SWE/src/blocks/SWE_WavePropagation-Block.cpp File Reference

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

8.18.1 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)
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/-Sebastian_Rettenberger,_M.Sc.)

8.18.2 LICENSE

8.18.3 DESCRIPTION

SWE_Block, which uses solvers in the wave propagation formulation.

8.19 /home/raphael/Programmieren/BPraktikum/SWE/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

8.19.1 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.)

8.19.2 LICENSE

8.19.3 DESCRIPTION

SWE_Block, which uses solvers in the wave propagation formulation.

8.20 /home/raphael/Programmieren/BPraktikum/SWE/src/examples/swe_DimensionalSplitting.cpp File Reference

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

Functions

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

8.20.1 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)

8.20.2 LICENSE

8.20.3 DESCRIPTION

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

8.20.4 Function Documentation

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

Main program for the simulation on a single SWE_WavePropagationBlock. 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.

simulation time.

Finalize.

8.21 /home/raphael/Programmieren/BPraktikum/SWE/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 "writer/VtkWriter.hh"
#include "scenarios/SWE_simple_scenarios.hh"
#include "tools/help.hh"
#include "tools/Logger.hh"
#include "tools/ProgressBar.hh"
```

Macros

- #define **ARG**(arg_name) getArgByName(vargs, arg_name, argv)

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_bottomNeighborInflow, 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)

8.21.1 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.)

8.21.2 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/.

8.21.3 DESCRIPTION

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

8.21.4 Function Documentation

8.21.4.1 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

8.21.4.2 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_bottomNeighborRank	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.

8.21.4.3 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.

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

Main program for the simulation on a single SWE_WavePropagationBlock. 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.

simulation time.

maximum allowed time step width within a block.

maximum allowed time steps of all blocks

Finalize.

8.22 /home/raphael/Programmieren/BPraktikum/SWE/src/examples/swe_simple.cpp File Reference

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

Functions

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

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

8.22.2 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/.

8.22.3 DESCRIPTION

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

8.22.4 Function Documentation

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

Main program for the simulation on a single SWE_WavePropagationBlock. Initialization.

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

l_baseName of the plots.

true if checkpoint file exists

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.

simulation time.

number of checkpoints that are already passed

maximum allowed time step width.

Finalize.

8.23 /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/vbo.cpp File Reference

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

8.23.1 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.)

8.23.2 LICENSE

8.24 /home/raphael/Programmieren/BPraktikum/SWE/src/opengl/vbo.h File Reference

```
#include "tools/Logger.hh"
```

```
#include <SDL/SDL_opengl.h>
```

Classes

- class VBO

8.24.1 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.)

8.24.2 LICENSE

8.24.3 DESCRIPTION

Handles a VertexBufferObject.

8.25 /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_ArtificialTsunamiScenario.hh File Reference

```
#include <cmath>
```

Classes

- class SWE_ArtificialTsunamiScenario

Macros

- #define **PI** 3.14159265358979323846264338327950

8.25.1 Detailed Description

This file is part of SWE.

Author

Raphael Dümig

8.25.2 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/.

8.25.3 DESCRIPTION

TODO

8.26 /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_AsagiScenario.cpp File Reference

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

8.26.1 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.)

8.26.2 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/.

8.27 /home/raphael/Programmieren/BPraktikum/SWE/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

8.27.1 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.)

8.27.2 LICENSE

8.27.3 DESCRIPTION

Access to bathymetry and displacement files with ASAGI.

8.28 /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_AsagiScenario_vis.hh File Reference

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

Classes

- class SWE_AsagiJapanSmallVisInfo

8.28.1 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.)

8.28.2 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/.

8.28.3 DESCRIPTION

Rescale water height in small Japan scenario

8.29 /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_NetCDFCheckpoint-Scenario.hh File Reference

```
#include "SWE_NetCDFScenario.hh"
#include <netcdf.h>
#include <iostream>
#include <cstdlib>
```

Classes

- class SWE_NetCDFCheckpointScenario

8.29.1 Detailed Description

This file is part of SWE.

Author

Thomas Blocher, Raphael Dümig

8.29.2 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/.

8.29.3 DESCRIPTION

TODO

8.30 /home/raphael/Programmieren/BPraktikum/SWE/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** }

8.30.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

8.30.2 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/.

8.30.3 DESCRIPTION

TODO

8.30.4 Typedef Documentation

8.30.4.1 typedef enum **BoundaryEdge BoundaryEdge**

enum type: numbering of the boundary edges

8.30.4.2 typedef enum **BoundaryType BoundaryType**

enum type: available types of boundary conditions

8.30.5 Enumeration Type Documentation

8.30.5.1 enum **BoundaryEdge**

enum type: numbering of the boundary edges

8.30.5.2 enum **BoundaryType**

enum type: available types of boundary conditions

8.31 /home/raphael/Programmieren/BPraktikum/SWE/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
- class SWE_DamBreakScenario

8.31.1 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.)

8.31.2 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/.

8.31.3 DESCRIPTION

TODO

8.32 /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_TsunamiScenario.hh File Reference

```
#include "SWE_NetCDFScenario.hh"
#include <netcdf.h>
#include <iostream>
#include <cstdlib>
#include <cassert>
```

Classes

- class SWE_TsunamiScenario

8.32.1 Detailed Description

This file is part of SWE.

Author

Thomas Blocher, Raphael Dümig

8.32.2 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/.

8.32.3 DESCRIPTION

TODO

8.33 /home/raphael/Programmieren/BPraktikum/SWE/src/scenarios/SWE_VisInfo.hh File Reference

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

Classes

- class SWE_VisInfo

8.33.1 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.)

8.33.2 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.

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/.

8.33.3 DESCRIPTION

TODO

8.34 /home/raphael/Programmieren/BPraktikum/SWE/src/testing/testing_scenario.hh File Reference

```
#include "scenarios/SWE_Scenario.hh"
```

Classes

- class SWE_TestingScenario

8.34.1 Detailed Description

This file is part of SWE.

Author

Raphael Dümig duemig@in.tum.de

8.34.2 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/.

8.34.3 DESCRIPTION

TODO

8.35 /home/raphael/Programmieren/BPraktikum/SWE/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)

8.35.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema
Sebastian Rettenberger

8.35.2 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/`.

8.35.3 DESCRIPTION

TODO

8.35.4 Function Documentation

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

Generates an output file name for a multiple SWE_Block version based on the ordering of the blocks.

Parameters

i_baseName	base name of the output.
i_blockPositionX	position of the SWE_Block in x-direction.
i_blockPositionY	position of the SWE_Block in y-direction.

Returns

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

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

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

8.35.4.3 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 - one output file is generated per checkpoint)

Deprecated

8.35.4.4 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 version based on the ordering of the blocks.

Parameters

i_baseName	base name of the output.
i_blockPositionX	position of the SWE_Block in x-direction.
i_blockPositionY	position of the SWE_Block in y-direction.
i_fileExtension	file extension of the output file.

Returns

Deprecated

8.35.4.5 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

8.36 /home/raphael/Programmieren/BPraktikum/SWE/src/tools/Logger.cpp File Reference

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

8.36.1 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.)

8.36.2 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/.

8.37 /home/raphael/Programmieren/BPraktikum/SWE/src/tools/Logger.hh File Reference

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

Classes

- class tools::Logger

8.37.1 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.)

8.37.2 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/.

8.37.3 DESCRIPTION

Collection of basic logging routines.

8.38 /home/raphael/Programmieren/BPraktikum/SWE/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

8.38.1 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.)

8.38.2 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/.

8.38.3 DESCRIPTION

A simple progress bar using stdout

8.39 /home/raphael/Programmieren/BPraktikum/SWE/src/writer/NetCdfWriter.cpp File Reference

```
#include "NetCdfWriter.hh"
#include <string>
#include <vector>
#include <iostream>
#include <cassert>
```

8.39.1 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.)

8.39.2 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/.

8.39.3 DESCRIPTION

A writer for the netCDF-format: http://www.unidata.ucar.edu/software/netcdf/

8.40 /home/raphael/Programmieren/BPraktikum/SWE/src/writer/NetCdfWriter.hh File Reference

```
#include <cstring>
#include <string>
#include <vector>
#include <netcdf.h>
#include "writer/Writer.hh"
```

Classes

- class io::NetCdfWriter

8.40.1 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.)

8.40.2 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/.

8.40.3 DESCRIPTION

A writer for the netCDF-format: http://www.unidata.ucar.edu/software/netcdf/

8.41 /home/raphael/Programmieren/BPraktikum/SWE/src/writer/NetCdfWriterCP.cpp File Reference

```
#include "NetCdfWriterCP.hh"
#include <string>
#include <vector>
#include <iostream>
#include <cassert>
```

8.41.1 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.)

8.41.2 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/.

8.41.3 DESCRIPTION

A writer for the netCDF-format: http://www.unidata.ucar.edu/software/netcdf/

8.42 /home/raphael/Programmieren/BPraktikum/SWE/src/writer/NetCdfWriterCP.hh File Reference

```
#include <cstring>
#include <string>
#include <vector>
#include <netcdf.h>
#include "writer/Writer.hh"
```

Classes

- class io::NetCdfWriter

8.42.1 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.)

8.42.2 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/.

8.42.3 DESCRIPTION

A writer for the netCDF-format: http://www.unidata.ucar.edu/software/netcdf/

8.43 /home/raphael/Programmieren/BPraktikum/SWE/src/writer/VtkWriter.cpp File Reference

```
#include <cassert>
#include <fstream>
#include "VtkWriter.hh"
```

8.43.1 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.)

8.43.2 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/.

8.43.3 DESCRIPTION

8.44 /home/raphael/Programmieren/BPraktikum/SWE/src/writer/VtkWriter.hh File Reference

```
#include <sstream>
#include "writer/Writer.hh"
```

Classes

- class io::VtkWriter

8.44.1 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.)

8.44.2 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/.

8.44.3 DESCRIPTION

8.45 /home/raphael/Programmieren/BPraktikum/SWE/src/writer/Writer.hh File Reference

```
#include "tools/help.hh"
```

Classes

- struct io::BoundarySize
- class io::Writer

8.45.1 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.)

8.45.2 LICENSE

8.45.3 DESCRIPTION

8.46 /home/raphael/Programmieren/BPraktikum/SWE/submodules/f-wave-solver/src/F-Wave.cpp File Reference

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

8.46.1 Detailed Description

Implementation of an f-wave solver

Author

Raphael Dümig

8.47 /home/raphael/Programmieren/BPraktikum/SWE/submodules/f-wave-solver/src/F-Wave.hpp File Reference

```
#include "FWave.cpp"
```

Classes

- class solver::FWave< T >

Macros

- #define **G** 9.81

8.47.1 Detailed Description

f-wave solver

Author

Raphael Dümig

8.48 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/main.cpp File Reference

```
#include "types.h"
#include "WavePropagation.h"
#include "scenarios/eisbach.h"
#include "writer/VtkWriter.h"
#include "tools/args.h"
#include <cstring>
```

Functions

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

8.48.1 Detailed Description

This file is part of SWE1D

Diese Datei ist Teil von SWE1D.

Author

Sebastian Rettenberger rettenbs@in.tum.de

8.49 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/constant_flow.h File Reference

```
#include "types.h"
```

Classes

- class scenarios::ConstantFlow

8.49.1 Detailed Description

This file is part of SWE1D

SWE1D 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.

SWE1D 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 SWE1D. If not, see http://www.gnu.org/licenses/.

Diese Datei ist Teil von SWE1D.

SWE1D ist Freie Software: Sie koennen es unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation, Version 3 der Lizenz oder (nach Ihrer Option) jeder spaeteren veroeffentlichten Version, weiterverbreiten und/oder modifizieren.

SWE1D wird in der Hoffnung, dass es nuetzlich sein wird, aber OHNE JEDE GEWAEHELEISTUNG, bereitgestellt; sogar ohne die implizite Gewaehrleistung der MARKTFAEHIGKEIT oder EIGNUNG FUER EINEN BESTIMMTEN ZWECK. Siehe die GNU General Public License fuer weitere Details.

Sie sollten eine Kopie der GNU General Public License zusammen mit diesem Programm erhalten haben. Wenn nicht, siehe http://www.gnu.org/licenses/.

Copyright

2013 Technische Universitaet Muenchen

Author

Sebastian Rettenberger rettenbs@in.tum.de
Raphael Dümig duemig@in.tum.de

8.50 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/dambreak.h File Reference

```
#include "types.h"
```

Classes

- class scenarios::DamBreak

8.50.1 Detailed Description

This file is part of SWE1D

SWE1D 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.

SWE1D 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 SWE1D. If not, see http://www.gnu.org/licenses/.

Diese Datei ist Teil von SWE1D.

SWE1D ist Freie Software: Sie koennen es unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation, Version 3 der Lizenz oder (nach Ihrer Option) jeder spaeteren veroeffentlichten Version, weiterverbreiten und/oder modifizieren.

SWE1D wird in der Hoffnung, dass es nuetzlich sein wird, aber OHNE JEDE GEWAEHELEISTUNG, bereitgestellt; sogar ohne die implizite Gewaehrleistung der MARKTFAEHIGKEIT oder EIGNUNG FUER EINEN BESTIMMTEN ZWECK. Siehe die GNU General Public License fuer weitere Details.

Sie sollten eine Kopie der GNU General Public License zusammen mit diesem Programm erhalten haben. Wenn nicht, siehe http://www.gnu.org/licenses/.

Copyright

2013 Technische Universitaet Muenchen

Author

Sebastian Rettenberger rettenbs@in.tum.de
Raphael Dümig duemig@in.tum.de

8.51 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/dambreak_bathy.h File Reference

```
#include "types.h"
```

Classes

- class scenarios::DamBreak

8.51.1 Detailed Description

This file is part of SWE1D

SWE1D 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.

SWE1D 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 SWE1D. If not, see http://www.gnu.org/licenses/.

Diese Datei ist Teil von SWE1D.

SWE1D ist Freie Software: Sie koennen es unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation, Version 3 der Lizenz oder (nach Ihrer Option) jeder spaeteren veroeffentlichten Version, weiterverbreiten und/oder modifizieren.

Author

Sebastian Rettenberger rettenbs@in.tum.de
Raphael Dümig duemig@in.tum.de

8.52 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/eisbach.h File Reference

```
#include "types.h"
```

Classes

- class scenarios::Eisbach

8.52.1 Detailed Description

This file is part of SWE1D

Diese Datei ist Teil von SWE1D.

SWE1D wird in der Hoffnung, dass es nuetzlich sein wird, aber OHNE JEDE GEWAEHELEISTUNG, bereitgestellt; sogar ohne die implizite Gewaehrleistung der MARKTFAEHIGKEIT oder EIGNUNG FUER EINEN BESTIMMTEN ZWECK. Siehe die GNU General Public License fuer weitere Details.

Sie sollten eine Kopie der GNU General Public License zusammen mit diesem Programm erhalten haben. Wenn nicht, siehe http://www.gnu.org/licenses/.

Copyright

2013 Technische Universitaet Muenchen

Author

Sebastian Rettenberger rettenbs@in.tum.de
Raphael Dümig duemig@in.tum.de

8.53 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/rarerare.h File Reference

```
#include "types.h"
```

Classes

- class scenarios::RareRare

8.53.1 Detailed Description

This file is part of SWE1D

Diese Datei ist Teil von SWE1D.

Author

Raphael Dümig duemig@in.tum.de

8.54 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/-ShockShock.h File Reference

```
#include "types.h"
```

Classes

- class scenarios::ShockShock

8.54.1 Detailed Description

This file is part of SWE1D

Diese Datei ist Teil von SWE1D.

Author

Sebastian Rettenberger rettenbs@in.tum.de

8.55 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/-Subcritical_flow.h File Reference

```
#include "types.h"
```

Classes

- class scenarios::Subcrit

8.55.1 Detailed Description

This file is part of SWE1D

Diese Datei ist Teil von SWE1D.

SWE1D ist Freie Software: Sie koennen es unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation, Version 3 der Lizenz oder (nach Ihrer Option) jeder spaeteren veroeffentlichten Version, weiterverbreiten und/oder modifizieren.

SWE1D wird in der Hoffnung, dass es nuetzlich sein wird, aber OHNE JEDE GEWAEHELEISTUNG, bereitgestellt; sogar ohne die implizite Gewaehrleistung der MARKTFAEHIGKEIT oder EIGNUNG FUER EINEN BESTIMMTEN ZWECK. Siehe die GNU General Public License fuer weitere Details.

Sie sollten eine Kopie der GNU General Public License zusammen mit diesem Programm erhalten haben. Wenn nicht, siehe http://www.gnu.org/licenses/.

Copyright

2013 Technische Universitaet Muenchen

Author

Sebastian Rettenberger rettenbs@in.tum.de
Thomas Blocher blocher@in.tum.de

8.56 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/scenarios/-Supercritical_flow.h File Reference

```
#include "types.h"
```

Classes

- class scenarios::Supercrit

8.56.1 Detailed Description

This file is part of SWE1D

SWE1D 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.

SWE1D 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 SWE1D. If not, see http://www.gnu.org/licenses/.

Diese Datei ist Teil von SWE1D.

SWE1D ist Freie Software: Sie koennen es unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation, Version 3 der Lizenz oder (nach Ihrer Option) jeder spaeteren veroeffentlichten Version, weiterverbreiten und/oder modifizieren.

SWE1D wird in der Hoffnung, dass es nuetzlich sein wird, aber OHNE JEDE GEWAEHELEISTUNG, bereitgestellt; sogar ohne die implizite Gewaehrleistung der MARKTFAEHIGKEIT oder EIGNUNG FUER EINEN BESTIMMTEN ZWECK. Siehe die GNU General Public License fuer weitere Details.

Sie sollten eine Kopie der GNU General Public License zusammen mit diesem Programm erhalten haben. Wenn nicht, siehe http://www.gnu.org/licenses/.

Copyright

2013 Technische Universitaet Muenchen

Author

Sebastian Rettenberger rettenbs@in.tum.de
Thomas Blocher blocher@in.tum.de

8.57 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/tools/args.cpp File Reference

```
#include "args.h"
```

8.57.1 Detailed Description

This file is part of SWE1D

Diese Datei ist Teil von SWE1D.

Copyright

Author

Sebastian Rettenberger rettenbs@in.tum.de

8.58 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/tools/args.h File Reference

```
#include "tools/logger.h"
#include <getopt.h>
#include <cstdlib>
#include <iostream>
#include <sstream>
```

Classes

- class tools::Args

8.58.1 Detailed Description

This file is part of SWE1D

Diese Datei ist Teil von SWE1D.

Copyright

Author

Sebastian Rettenberger rettenbs@in.tum.de

8.59 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/tools/logger.cpp File Reference

```
#include "logger.h"
```

8.59.1 Detailed Description

This file is part of SWE1D

Diese Datei ist Teil von SWE1D.

SWE1D ist Freie Software: Sie koennen es unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation, Version 3 der Lizenz oder (nach Ihrer Option) jeder spaeteren veroeffentlichten Version, weiterverbreiten und/oder modifizieren.

SWE1D wird in der Hoffnung, dass es nuetzlich sein wird, aber OHNE JEDE GEWAEHELEISTUNG, bereitgestellt; sogar ohne die implizite Gewaehrleistung der MARKTFAEHIGKEIT oder EIGNUNG FUER EINEN BESTIMMTEN ZWECK. Siehe die GNU General Public License fuer weitere Details.

Sie sollten eine Kopie der GNU General Public License zusammen mit diesem Programm erhalten haben. Wenn nicht, siehe http://www.gnu.org/licenses/.

Copyright

2013 Technische Universitaet Muenchen

Author

Sebastian Rettenberger rettenbs@in.tum.de

8.60 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/tools/logger.h File Reference

```
#include <cstdlib>
#include <iostream>
```

Classes

- class tools::Logger

8.60.1 Detailed Description

This file is part of SWE1D

SWE1D 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.

SWE1D 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 SWE1D. If not, see http://www.gnu.org/licenses/.

Diese Datei ist Teil von SWE1D.

SWE1D ist Freie Software: Sie koennen es unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation, Version 3 der Lizenz oder (nach Ihrer Option) jeder spaeteren veroeffentlichten Version, weiterverbreiten und/oder modifizieren.

SWE1D wird in der Hoffnung, dass es nuetzlich sein wird, aber OHNE JEDE GEWAEHELEISTUNG, bereitgestellt; sogar ohne die implizite Gewaehrleistung der MARKTFAEHIGKEIT oder EIGNUNG FUER EINEN BESTIMMTEN ZWECK. Siehe die GNU General Public License fuer weitere Details.

Sie sollten eine Kopie der GNU General Public License zusammen mit diesem Programm erhalten haben. Wenn nicht, siehe http://www.gnu.org/licenses/.

Copyright

2013 Technische Universitaet Muenchen

Author

Sebastian Rettenberger rettenbs@in.tum.de

8.61 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/types.h File Reference

Typedefs

- typedef float **T**

8.61.1 Detailed Description

This file is part of SWE1D

SWE1D 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.

SWE1D 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 SWE1D. If not, see http://www.gnu.org/licenses/.

Diese Datei ist Teil von SWE1D.

SWE1D ist Freie Software: Sie koennen es unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation, Version 3 der Lizenz oder (nach Ihrer Option) jeder spaeteren veroeffentlichten Version, weiterverbreiten und/oder modifizieren.

SWE1D wird in der Hoffnung, dass es nuetzlich sein wird, aber OHNE JEDE GEWAEHELEISTUNG, bereitgestellt; sogar ohne die implizite Gewaehrleistung der MARKTFAEHIGKEIT oder EIGNUNG FUER EINEN BESTIMMTEN ZWECK. Siehe die GNU General Public License fuer weitere Details.

Sie sollten eine Kopie der GNU General Public License zusammen mit diesem Programm erhalten haben. Wenn nicht, siehe http://www.gnu.org/licenses/.

Copyright

2013 Technische Universitaet Muenchen

Author

Sebastian Rettenberger rettenbs@in.tum.de

8.62 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/WavePropagation.cpp File Reference

```
#include "WavePropagation.h"
```

8.62.1 Detailed Description

This file is part of SWE1D

SWE1D 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.

Author

Sebastian Rettenberger rettenbs@in.tum.de

8.63 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/WavePropagation.h File Reference

```
#include "types.h"
#include "solvers/FWave.hpp"
```

Classes

- class WavePropagation

8.63.1 Detailed Description

This file is part of SWE1D

SWE1D wird in der Hoffnung, dass es nuetzlich sein wird, aber OHNE JEDE GEWAEHELEISTUNG, bereitgestellt; sogar ohne die implizite Gewaehrleistung der MARKTFAEHIGKEIT oder EIGNUNG FUER EINEN BESTIMMTEN ZWECK. Siehe die GNU General Public License fuer weitere Details.

Sie sollten eine Kopie der GNU General Public License zusammen mit diesem Programm erhalten haben. Wenn nicht, siehe http://www.gnu.org/licenses/.

Copyright

2013 Technische Universitaet Muenchen

Author

Sebastian Rettenberger rettenbs@in.tum.de

8.64 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/writer/-ConsoleWriter.h File Reference

```
#include "types.h"
#include <iostream>
```

Classes

- class writer::ConsoleWriter

8.64.1 Detailed Description

This file is part of SWE1D

SWE1D 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.

SWE1D 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 SWE1D. If not, see http://www.gnu.org/licenses/.

Diese Datei ist Teil von SWE1D.

SWE1D ist Freie Software: Sie koennen es unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation, Version 3 der Lizenz oder (nach Ihrer Option) jeder spaeteren veroeffentlichten Version, weiterverbreiten und/oder modifizieren.

SWE1D wird in der Hoffnung, dass es nuetzlich sein wird, aber OHNE JEDE GEWAEHELEISTUNG, bereitgestellt; sogar ohne die implizite Gewaehrleistung der MARKTFAEHIGKEIT oder EIGNUNG FUER EINEN BESTIMMTEN ZWECK. Siehe die GNU General Public License fuer weitere Details.

Sie sollten eine Kopie der GNU General Public License zusammen mit diesem Programm erhalten haben. Wenn nicht, siehe http://www.gnu.org/licenses/.

Copyright

2013 Technische Universitaet Muenchen

Author

Sebastian Rettenberger rettenbs@in.tum.de

8.65 /home/raphael/Programmieren/BPraktikum/SWE/submodules/SWE1D/src/writer/-VtkWriter.h File Reference

```
#include "types.h"
#include <cassert>
#include <fstream>
#include <sstream>
#include <string>
```

Classes

- class writer::VtkWriter

8.65.1 Detailed Description

This file is part of SWE1D

Diese Datei ist Teil von SWE1D.

Copyright

Author

Sebastian Rettenberger rettenbs@in.tum.de

Index