Waveguide Solver

2.1

Generated by Doxygen 1.8.17

Chapter 1

Shape-Optimization of a 3D waveguide using dealii, transformation optics and the finite element method

1.1 Topics of this project

This project began as the implementation used in the thesis for the title of Master of Science by Pascal Kraft at the KIT. It is continued for his PHD studies and possibly as an introduction to dealii for other students in the same research group. This project, apart from mathematical goals, aims at creating a clear and reusable implementation of the the finite element method for Maxwell's equaations in a range of performance values, that enable the inclusion of an optimization-scheme without crippling time- or CPU-time consumption. Therefore the code should fullfill the following criteria:

1. The code should be readable to starters (educational purpose),
2. The code should be maintainable (reusability),
3. The code should be paralellizable via MPI or CUDA (both will be tested as a part of the phd-proceedings),
4. The code should perform well under the given circumstances,
5. The code should give scientific results and not only operate on marginal domains of parameter-values,
6. The code should be portable to other hardware-specifications then those on the given computer at the workspace (i.e. the performance should be usable in large-scale computations for example in Super↩ Computers of the KIT"s SCC.

These demands led to the introduction of a software development scheme for the work on the code based on agile-development and git.

1.2 Prerequisites of this project

In order to be able to work with this code it is important to first achieve a fundamental understanding of the following topics: First and foremost, an understanding of the finite element method is required and completely unreplacable. There exists extensive documentation on this topic and the reader should be aware of the fact, that the mathematical background cannot be understood without this knowledge. However, there are further demands. The programming-language of both this project and dealii itself is C++. This language also forms the backbone id CUDA and manu other, relevant libraries. It is to be considered inevitable in this field. "The vhoice of this language in a wau reduces the importance of the need for a performanc implementation on the code level ∗on the functional or theoretical level this obviously has a very miimal influence on the performanc.(. Also it should be noted that there exists a very large documentation about dealii which might help the reader understand this code. Lastly dealii is basically only available on Linux since it nearly always requires a build-process which would not be possible with out enormous problems on different OS. As far as mathematical knowledge is concerned, a basic education in linear algebra, krylov subspace methods, transformation-optics, functional analysis, optics and optimization theory will further the understanding of both the code and this documentation of it.

Author

Pascal Kraft

Version

2.1

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

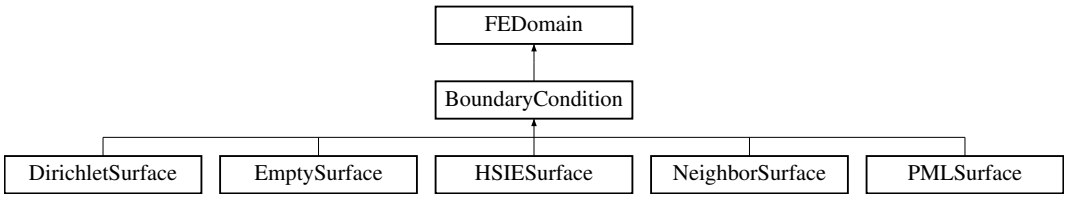
Class Documentation

4.1 BoundaryCondition Class Reference

This is the base type for boundary coniditions. Some implementations are done on this level, some below.

`#include <BoundaryCondition.h>`

Inheritance diagram for BoundaryCondition:



Public Member Functions

- **BoundaryCondition** (unsigned int in_bid, unsigned int in_level, double in_additional_coordinate)
- virtual void **initialize** ()=0
- virtual std::string **output_results** (const dealii::Vector< ComplexNumber > &, std::string)=0
- virtual bool **is_point_at_boundary** (Position2D in_p, BoundaryId in_bid)=0
- void **set_mesh_boundary_ids** ()
- auto **get_boundary_ids** () -> std::vector< BoundaryId >
- virtual auto **get_dof_association** () -> std::vector< InterfaceDofData >=0
- virtual auto **get_dof_association_by_boundary_id** (BoundaryId in_boundary_id) -> std::vector< InterfaceDofData >=0
- virtual auto **get_global_dof_indices_by_boundary_id** (BoundaryId in_boundary_id) -> std::vector< Dof↩ Number >
- virtual void **fill_sparsity_pattern** (dealii::DynamicSparsityPattern *in_dsp, Constraints *constraints)=0
- virtual void **fill_matrix** (dealii::PETScWrappers::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints)=0
- virtual void **fill_matrix** (dealii::SparseMatrix< ComplexNumber > *, Constraints *constraints)=0
- virtual void **fill_matrix** (dealii::PETScWrappers::MPI::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints)=0
- virtual void **fill_matrix** (dealii::PETScWrappers::SparseMatrix *, dealii::PETScWrappers::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints)=0

- virtual void **send_up_inner_dofs** ()
- virtual void **receive_from_below_dofs** ()
- virtual void **finish_dof_index_initialization** ()
- virtual std::vector< DofNumber > **receive_boundary_dofs** (unsigned int other_bid)
- virtual auto **make_constraints** () -> Constraints
- double **boundary_norm** (NumericVectorDistributed *)
- double **boundary_surface_norm** (NumericVectorDistributed *, BoundaryId)
- virtual unsigned int **cells_for_boundary_id** (unsigned int boundary_id)
- void **print_dof_validation** ()

Public Attributes

- const BoundaryId **b_id**
- const unsigned int **level**
- const double **additional_coordinate**
- std::vector< InterfaceDofData > **surface_dofs**
- bool **surface_dof_sorting_done**
- bool **boundary_coordinates_computed** = false
- std::array< double, 6 > **boundary_vertex_coordinates**
- DofCount **dof_counter**
- unsigned int **global_partner_mpi_rank**
- const std::vector< BoundaryId > **adjacent_boundaries**
- std::array< bool, 6 > **are_edge_dofs_owned**
- DofHandler3D **dof_handler**

4.1.1 Detailed Description

This is the base type for boundary coniditions. Some implementations are done on this level, some below.

There are several deriveed classes for this type: Dirichlet, Empty, Hardy, PML and Neighbor. Details about them can be found in the derived classes. To the rest of the code, the most relevant functions are:

- Handling the dofs (number of dofs and association to boundaries)
- Assembly (of sparsity pattern and matrices)
- Building constraints

For the numbering, I always use the scheme 0 = -x, 1 = +x, 2 = -y, 3 = +y, 4 = -z and 5 = +z for all domain types. All domains are cuboid, so there are always 6 surfaces in the coordinate orthogonal directions.

Boundary conditions in this code have three types of surfaces (best visualized with a pml domain, i.e. a FE-domain):

- The surface shared with the inner domain,
- The sufaces shared with other boundary conditions,
- An outward surface, where dofs only couple with the interior of this boundary condition.

Similar to all objects in this code, these objects have an initialize function that is implemented in the derived classes.

Definition at line 30 of file BoundaryCondition.h.

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

- Code/BoundaryCondition/BoundaryCondition.h
- Code/BoundaryCondition/BoundaryCondition.cpp

4.2 BoundaryInformation Struct Reference

Public Member Functions

- **BoundaryInformation** (unsigned int in_coord, bool neg)

Public Attributes

- unsigned int **inner_coordinate**
- bool **negate_value**

4.2.1 Detailed Description

Definition at line 117 of file Types.h.

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

- Code/Core/Types.h

4.3 CellAngelingData Struct Reference

Public Attributes

- EdgeAngelingData **edge_data**
- VertexAngelingData **vertex_data**

4.3.1 Detailed Description

Definition at line 76 of file Types.h.

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

- Code/Core/Types.h

4.4 CellwiseAssemblyData Struct Reference

Public Member Functions

- **CellwiseAssemblyData** (dealii::FE_NedelecSZ< 3 > *fe, DofHandler3D *dof_handler)
- void **prepare_for_current_q_index** (unsigned int q_index)
- Tensor< 1, 3, ComplexNumber > **Conjugate_Vector** (Tensor< 1, 3, ComplexNumber > input)

Public Attributes

- QGauss< 3 > **quadrature_formula**
- FEValues< 3 > **fe_values**
- std::vector< Position > **quadrature_points**
- const unsigned int **dofs_per_cell**
- const unsigned int **n_q_points**
- FullMatrix< ComplexNumber > **cell_mass_matrix**
- FullMatrix< ComplexNumber > **cell_stiffness_matrix**
- dealii::Vector< ComplexNumber > **cell_rhs**
- const double **eps_in**
- const double **eps_out**
- const double **mu_zero**
- MaterialTensor **transformation**
- MaterialTensor **epsilon**
- MaterialTensor **mu**
- std::vector< DofNumber > **local_dof_indices**
- DofHandler3D::active_cell_iterator **cell**
- DofHandler3D::active_cell_iterator **end_cell**
- const FEValuesExtractors::Vector **fe_field**

4.4.1 Detailed Description

Definition at line 166 of file RectangularMode.cpp.

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

- Code/ModalComputations/RectangularMode.cpp

4.5 CellwiseAssemblyDataNP Struct Reference

Public Member Functions

- **CellwiseAssemblyDataNP** (dealii::FE_NedelecSZ< 3 > *fe, DofHandler3D *dof_handler)
- void **prepare_for_current_q_index** (unsigned int q_index)
- Tensor< 1, 3, ComplexNumber > **Conjugate_Vector** (Tensor< 1, 3, ComplexNumber > input)

Public Attributes

- QGauss< 3 > **quadrature_formula**
- FEValues< 3 > **fe_values**
- std::vector< Position > **quadrature_points**
- const unsigned int **dofs_per_cell**
- const unsigned int **n_q_points**
- FullMatrix< ComplexNumber > **cell_matrix**
- const double **eps_in**
- const double **eps_out**
- const double **mu_zero**
- Vector< ComplexNumber > **cell_rhs**
- MaterialTensor **transformation**

- MaterialTensor **epsilon**
- MaterialTensor **mu**
- std::vector< DofNumber > **local_dof_indices**
- DofHandler3D::active_cell_iterator **cell**
- DofHandler3D::active_cell_iterator **end_cell**
- ExactSolutionRamped **exact_solution_ramped**
- bool **has_input_interface** = false
- const FEValuesExtractors::Vector **fe_field**
- Vector< ComplexNumber > **incoming_wave_field**
- IndexSet **constrained_dofs**

4.5.1 Detailed Description

Definition at line 146 of file InnerDomain.cpp.

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

- Code/Core/InnerDomain.cpp

4.6 CellwiseAssemblyDataPML Struct Reference

Public Member Functions

- **CellwiseAssemblyDataPML** (dealii::FE_NedelecSZ< 3 > *fe, DofHandler3D *dof_handler)
- Position **get_position_for_q_index** (unsigned int q_index)
- void **prepare_for_current_q_index** (unsigned int q_index, dealii::Tensor< 2, 3, ComplexNumber > epsilon, dealii::Tensor< 2, 3, ComplexNumber > mu_inverse)
- Tensor< 1, 3, ComplexNumber > **Conjugate_Vector** (Tensor< 1, 3, ComplexNumber > input)

Public Attributes

- QGauss< 3 > **quadrature_formula**
- FEValues< 3 > **fe_values**
- std::vector< Position > **quadrature_points**
- const unsigned int **dofs_per_cell**
- const unsigned int **n_q_points**
- FullMatrix< ComplexNumber > **cell_matrix**
- Vector< ComplexNumber > **cell_rhs**
- std::vector< DofNumber > **local_dof_indices**
- DofHandler3D::active_cell_iterator **cell**
- DofHandler3D::active_cell_iterator **end_cell**
- const FEValuesExtractors::Vector **fe_field**

4.6.1 Detailed Description

Definition at line 354 of file PMLSurface.cpp.

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

- Code/BoundaryCondition/PMLSurface.cpp

4.7 ConstraintPair Struct Reference

Public Attributes

- unsigned int **left**
- unsigned int **right**
- bool **sign**

4.7.1 Detailed Description

Definition at line 201 of file Types.h.

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

- Code/Core/Types.h

4.8 CoreLogger Class Reference

Outputs I want:

```
#include <CoreLogger.h>
```

4.8.1 Detailed Description

Outputs I want:

- Timing output for all solver runs on any level.
- Convergence histories for any solver run on any level (except the lowest one maybe, bc. thats direct).
- Convergence rates
- Dof Numbers on all levels
- Memory Consumption of the direct solver

So this object mainly manages run meta-information. It needs functions that register which run the code is on (which iteration on which level etc.) There will only be one instance of this object and it will be available globally. It should use the FileLogger global instance to create files.

Definition at line 18 of file CoreLogger.h.

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

- Code/OutputGenerators/Console/CoreLogger.h

4.9 DataSeries Struct Reference

Public Attributes

- std::vector< double > **values**
- bool **is_closed**
- std::string **name**

4.9.1 Detailed Description

Definition at line 212 of file Types.h.

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

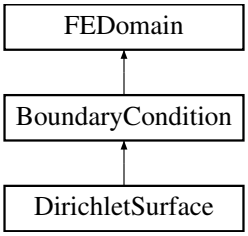
- Code/Core/Types.h

4.10 DirichletSurface Class Reference

This class implements dirichlet data on the given surface.

`#include <DirichletSurface.h>`

Inheritance diagram for DirichletSurface:



Public Member Functions

- **DirichletSurface** (unsigned int in_bid, unsigned int in_level)
- void **fill_matrix** (dealii::PETScWrappers::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_matrix** (dealii::SparseMatrix< ComplexNumber > *, Constraints *constraints) override
- void **fill_matrix** (dealii::PETScWrappers::SparseMatrix *, dealii::PETScWrappers::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_matrix** (dealii::PETScWrappers::MPI::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_sparsity_pattern** (dealii::DynamicSparsityPattern *in_dsp, Constraints *in_constriants) override
- bool **is_point_at_boundary** (Position2D in_p, BoundaryId in_bid) override
- void **initialize** () override
- auto **get_dof_association** () -> std::vector< InterfaceDofData > override
- auto **get_dof_association_by_boundary_id** (BoundaryId in_boundary_id) -> std::vector< InterfaceDofData > override
- std::string **output_results** (const dealii::Vector< ComplexNumber > &, std::string) override
- DofCount **compute_n_locally_owned_dofs** () override
- DofCount **compute_n_locally_active_dofs** () override
- void **determine_non_owned_dofs** () override
- auto **make_constraints** () -> Constraints override

Additional Inherited Members

4.10.1 Detailed Description

This class implements dirichlet data on the given surface.

This class is a simple derived function from the boundary condition base class. Since dirichlet constraints introduce no new degrees of freedom, the functions like fill_matrix don't do anything.

The only relevant function here is the make_constraints function which writes the dirichlet constraints into the given constraints object.

Definition at line 18 of file DirichletSurface.h.

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

- Code/BoundaryCondition/DirichletSurface.h
- Code/BoundaryCondition/DirichletSurface.cpp

4.11 DofAssociation Struct Reference

Public Attributes

- bool **is_edge**
- DofNumber **edge_index**
- std::string **face_index**
- DofNumber **dof_index_on_hsie_surface**
- Position **base_point**
- bool **true_orientation**

4.11.1 Detailed Description

Definition at line 149 of file Types.h.

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

- Code/Core/Types.h

4.12 DofCountsStruct Struct Reference

Public Attributes

- unsigned int **hsie** = 0
- unsigned int **non_hsie** = 0
- unsigned int **total** = 0

4.12.1 Detailed Description

Definition at line 164 of file Types.h.

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

- Code/Core/Types.h

4.13 DofCouplingInformation Struct Reference

Public Attributes

- DofNumber **first_dof**
- DofNumber **second_dof**
- double **coupling_value**

4.13.1 Detailed Description

Definition at line 127 of file Types.h.

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

- Code/Core/Types.h

4.14 DofData Struct Reference

This is only for the storage of basic data.

`#include <DofData.h>`

Public Member Functions

- void **set_base_dof** (unsigned int in_base_dof_index)
- **DofData** (std::string in_id)
- **DofData** (unsigned int in_id)
- auto **update_nodal_basis_flag** () -> void

Public Attributes

- DofType **type**
- int **hsie_order**
- int **inner_order**
- bool **nodal_basis**
- unsigned int **global_index**
- bool **got_base_dof_index**
- unsigned int **base_dof_index**
- std::string **base_structure_id_face**
- unsigned int **base_structure_id_non_face**
- bool **orientation** = true

4.14.1 Detailed Description

This is only for the storage of basic data.

Definition at line 13 of file DofData.h.

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

- Code/BoundaryCondition/DofData.h

4.15 DofIndexData Class Reference

Public Member Functions

- void **communicateSurfaceDofs** ()
- void **initialize** ()
- void **initialize_level** (unsigned int level)

Public Attributes

- bool * **isSurfaceNeighbor**
- std::vector< LevelDofIndexData > **indexCountsByLevel**

4.15.1 Detailed Description

Definition at line 6 of file DofIndexData.h.

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

- Code/Hierarchy/DofIndexData.h
- Code/Hierarchy/DofIndexData.cpp

4.16 DofOwner Struct Reference

Public Attributes

- unsigned int **owner** = 0
- bool **is_boundary_dof** = false
- unsigned int **surface_id** = 0

4.16.1 Detailed Description

Definition at line 81 of file Types.h.

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

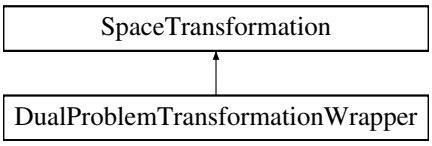
- Code/Core/Types.h

4.17 DualProblemTransformationWrapper Class Reference

If we do an adjoint computation, we need a SpaceTransformation, which has the same properties as the primal one but measures in transformed coordinates. This Wrapper contains the space transformation of the primal version but maps input parameters to their dual equivalent.

`#include <DualProblemTransformationWrapper.h>`

Inheritance diagram for DualProblemTransformationWrapper:



Public Member Functions

- DualProblemTransformationWrapper (SpaceTransformation *non_dual_st)

  Since this object encapsulates another Space Transformation, the construction is straight forward.
- Position math_to_phys (Position coord) const

  One of the core functionalities of a SpaceTransformation is to map a mathematical coordinate to a physical one (so an transformed to an untransformed coordinate).
- Position phys_to_math (Position coord) const

  This function does the same as math_to_phys only in the opposit direction.
- dealii::Tensor< 2, 3, ComplexNumber > **get_Tensor** (Position &coordinate) const
- dealii::Tensor< 2, 3, ComplexNumber > **get_Preconditioner_Tensor** (Position &coordinate, int block) const
- dealii::Tensor< 2, 3, ComplexNumber > **Apply_PML_To_Tensor** (Position &coordinate, dealii::Tensor< 2, 3, double > Tensor_input) const
- dealii::Tensor< 2, 3, ComplexNumber > **Apply_PML_To_Tensor_For_Preconditioner** (Position &coordinate, dealii::Tensor< 2, 3, double > Tensor_input, int block) const
- dealii::Tensor< 2, 3, double > **get_Space_Transformation_Tensor** (Position &coordinate) const
- dealii::Tensor< 2, 3, double > **get_Space_Transformation_Tensor_Homogenized** (Position &coordinate) const
- void estimate_and_initialize ()

  At the beginning (before the first solution of a system) only the boundary conditions for the shape of the waveguide are known.
- double get_Q1 (double z) const

  This member calculates the value of Q1 for a provided z-coordinate.
- double get_Q2 (double z) const

  This member calculates the value of Q2 for a provided z-coordinate.
- double get_Q3 (double z) const

  This member calculates the value of Q3 for a provided z-coordinate.
- double get_dof (int dof) const

  This is a getter for the values of degrees of freedom.
- void set_dof (int dof, double value)

  This function sets the value of the dof provided to the given value.
- double get_free_dof (int dof) const

  This is a getter for the values of degrees of freedom.
- void set_free_dof (int dof, double value)

  This function sets the value of the dof provided to the given value.
- std::pair< int, double > Z_to_Sector_and_local_z (double in_z) const

Using this method unifies the usage of coordinates.

- double System_Length () const

  Returns the complete length of the computational domain.
- double Sector_Length () const

  Returns the length of one sector.
- double Layer_Length () const

  Returns the length of one layer.
- double get_r (double in_z) const

  Returns the radius for a system-coordinate;.
- double get_m (double in_z) const

  Returns the shift for a system-coordinate;.
- double get_v (double in_z) const

  Returns the tilt for a system-coordinate;.
- Vector< double > Dofs () const

  Other objects can use this function to retrieve an array of the current values of the degrees of freedom of the functional we are optimizing.
- unsigned int NFreeDofs () const

  This function returns the number of unrestrained degrees of freedom of the current optimization run.
- unsigned int NDofs () const

  This function returns the total number of DOFs including restrained ones.
- bool IsDofFree (int) const

  Since Dofs() also returns restrained degrees of freedom, this function can be applied to determine if a degree of freedom is indeed free or restrained.
- void Print () const

  Console output of the current Waveguide Structure.

Public Attributes

- std::vector< Sector< 3 > > case_sectors

  This member contains all the Sectors who, as a sum, form the complete Waveguide.
- const double epsilon_K

  The material-property ϵ_r has a different value inside and outside of the waveguides core.
- const double epsilon_M

  The material-property ϵ_r has a different value inside and outside of the waveguides core.
- const int sectors

  Since the computational domain is split into subdomains (called sectors), it is important to keep track of the amount of subdomains.
- const double deltaY

  This value is initialized with the value Delta from the input-file.
- Vector< double > InitialDofs

  This vector of values saves the initial configuration.
- SpaceTransformation * **st**

4.17.1 Detailed Description

If we do an adjoint computation, we need a SpaceTransformation, which has the same properties as the primal one but measures in transformed coordinates. This Wrapper contains the space transformation of the primal version but maps input parameters to their dual equivalent.

Essentially this class enables us to write a waveguide class which is unaware of its being primal or dual. Using this wrapper makes us compute the solution of the inverse order shape parametrization.

Author

Pascal Kraft

Date

1.12.2016

Definition at line 23 of file DualProblemTransformationWrapper.h.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 DualProblemTransformationWrapper()

```
DualProblemTransformationWrapper::DualProblemTransformationWrapper (
            SpaceTransformation * non_dual_st )
```

Since this object encapsulates another Space Transformation, the construction is straight forward.

Parameters

non_dual↩ _st	This pointer points to the actual transformation that is being wrapped.

Definition at line 13 of file DualProblemTransformationWrapper.cpp.

```
15     : SpaceTransformation(3),
16       epsilon_K(GlobalParams.Epsilon_R_in_waveguide),
17       epsilon_M(GlobalParams.Epsilon_R_outside_waveguide),
18       sectors(GlobalParams.Number_of_sectors),
19       deltaY(GlobalParams.Vertical_displacement_of_waveguide) {
20   st = in_st;
21   homogenized = st->homogenized;
22 }
```

4.17.3 Member Function Documentation

4.17.3.1 Dofs()

```
Vector< double > DualProblemTransformationWrapper::Dofs ( ) const  [virtual]
```

Other objects can use this function to retrieve an array of the current values of the degrees of freedom of the functional we are optimizing.

This also includes restrained degrees of freedom and other functions can be used to determine this property. This has to be done because in different cases the number of restrained degrees of freedom can vary and we want no logic about this in other functions.

Implements SpaceTransformation.

Definition at line 127 of file DualProblemTransformationWrapper.cpp.

```
127                                                                 {
128   return st->Dofs();
129 }
```

References SpaceTransformation::Dofs().

4.17.3.2 estimate_and_initialize()

```
void DualProblemTransformationWrapper::estimate_and_initialize ( )  [virtual]
```

At the beginning (before the first solution of a system) only the boundary conditions for the shape of the waveguide are known.

Therefore the values for the degrees of freedom need to be estimated. This function sets all variables to appropiate values and estimates an appropriate shape based on averages and a polynomial interpolation of the boundary conditions on the shape.

Implements SpaceTransformation.

Definition at line 73 of file DualProblemTransformationWrapper.cpp.

```
73                                                                  {
74   st->estimate_and_initialize();
75   return;
76 }
```

References SpaceTransformation::estimate_and_initialize().

4.17.3.3 get_dof()

```
double DualProblemTransformationWrapper::get_dof (
            int dof ) const  [virtual]
```

This is a getter for the values of degrees of freedom.

A getter-setter interface was introduced since the values are estimated automatically during the optimization and non-physical systems should be excluded from the domain of possible cases.

Parameters

dof	The index of the degree of freedom to be retrieved from the structure of the modelled waveguide.

Returns

This function returns the value of the requested degree of freedom. Should this dof not exist, 0 will be returnd.

Implements SpaceTransformation.

Definition at line 90 of file DualProblemTransformationWrapper.cpp.

```
90                                                          {
91   return st->get_dof(dof);
92 }
```

References SpaceTransformation::get_dof().

4.17.3.4 get_free_dof()

```
double DualProblemTransformationWrapper::get_free_dof (
            int dof ) const  [virtual]
```

This is a getter for the values of degrees of freedom.

A getter-setter interface was introduced since the values are estimated automatically during the optimization and non-physical systems should be excluded from the domain of possible cases.

Parameters

dof	The index of the degree of freedom to be retrieved from the structure of the modelled waveguide.

Returns

This function returns the value of the requested degree of freedom. Should this dof not exist, 0 will be returnd.

Implements SpaceTransformation.

Definition at line 98 of file DualProblemTransformationWrapper.cpp.

```
98                                                          {
99   return st->get_free_dof(dof);
100 }
```

References SpaceTransformation::get_free_dof().

4.17.3.5 get_Q1()

```
double DualProblemTransformationWrapper::get_Q1 (
            double z ) const  [virtual]
```

This member calculates the value of Q1 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q1 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implements SpaceTransformation.

Definition at line 78 of file DualProblemTransformationWrapper.cpp.

```
78                                                                  {
79   return st->get_Q1(z);
80 }
```

References SpaceTransformation::get_Q1().

4.17.3.6 get_Q2()

```
double DualProblemTransformationWrapper::get_Q2 (
            double z ) const  [virtual]
```

This member calculates the value of Q2 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q2 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implements SpaceTransformation.

Definition at line 82 of file DualProblemTransformationWrapper.cpp.

```
82                                                                  {
83   return st->get_Q2(z);
84 }
```

References SpaceTransformation::get_Q2().

4.17.3.7 get_Q3()

```
double DualProblemTransformationWrapper::get_Q3 (
            double z ) const  [virtual]
```

This member calculates the value of Q3 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q3 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implements SpaceTransformation.

Definition at line 86 of file DualProblemTransformationWrapper.cpp.

```
86                                                                  {
87   return st->get_Q3(z);
88 }
```

References SpaceTransformation::get_Q3().

4.17.3.8 IsDofFree()

```
bool DualProblemTransformationWrapper::IsDofFree (
            int input ) const  [virtual]
```

Since Dofs() also returns restrained degrees of freedom, this function can be applied to determine if a degree of freedom is indeed free or restrained.

"restrained" means that for example the DOF represents the radius at one of the connectors (input or output) and therefore we forbid the optimization scheme to vary this value.

Implements SpaceTransformation.

Definition at line 139 of file DualProblemTransformationWrapper.cpp.

```
139                                                                 {
140   return st->IsDofFree(input);
141 }
```

References SpaceTransformation::IsDofFree().

4.17.3.9 math_to_phys()

```
Position DualProblemTransformationWrapper::math_to_phys (
            Position coord ) const  [virtual]
```

One of the core functionalities of a SpaceTransformation is to map a mathematical coordinate to a physical one (so an transformed to an untransformed coordinate).

Parameters

coord	the coordinate to be transformed. In this class we simply pass the

Implements SpaceTransformation.

Definition at line 24 of file DualProblemTransformationWrapper.cpp.

```
24                                                                  {
25   return st->math_to_phys(coord);
26 }
```

4.17.3.10 NDofs()

```
unsigned int DualProblemTransformationWrapper::NDofs ( ) const  [virtual]
```

This function returns the total number of DOFs including restrained ones.

This is the lenght of the array returned by Dofs().

Implements SpaceTransformation.

Definition at line 135 of file DualProblemTransformationWrapper.cpp.

```
135                                                                    {
136   return st->NDofs();
137 }
```

References SpaceTransformation::NDofs().

4.17.3.11 phys_to_math()

```
Position DualProblemTransformationWrapper::phys_to_math (
            Position coord ) const  [virtual]
```

This function does the same as math_to_phys only in the opposit direction.

Parameters

coord	the coordinate to be transformed.

Implements SpaceTransformation.

Definition at line 28 of file DualProblemTransformationWrapper.cpp.

```
28                                                                                  {
29   return st->phys_to_math(coord);
30 }
```

4.17.3.12 set_dof()

```
void DualProblemTransformationWrapper::set_dof (
            int dof,
            double value )  [virtual]
```

This function sets the value of the dof provided to the given value.

It is important to consider, that some dofs are non-writable (i.e. the values of the degrees of freedom on the boundary, like the radius of the input-connector cannot be changed).

Parameters

dof	The index of the parameter to be changed.
value	The value, the dof should be set to.

Implements SpaceTransformation.

Definition at line 94 of file DualProblemTransformationWrapper.cpp.

```
94                                                                  {
```

```
95   return st->set_dof(dof, value);
96 }
```

References SpaceTransformation::set_dof().

4.17.3.13 set_free_dof()

```
void DualProblemTransformationWrapper::set_free_dof (
            int dof,
            double value )  [virtual]
```

This function sets the value of the dof provided to the given value.

It is important to consider, that some dofs are non-writable (i.e. the values of the degrees of freedom on the boundary, like the radius of the input-connector cannot be changed).

Parameters

dof	The index of the parameter to be changed.
value	The value, the dof should be set to.

Implements SpaceTransformation.

Definition at line 102 of file DualProblemTransformationWrapper.cpp.

```
102                                                                    {
103   return st->set_free_dof(dof, value);
104 }
```

References SpaceTransformation::set_free_dof().

4.17.3.14 Z_to_Sector_and_local_z()

```
std::pair< int, double > DualProblemTransformationWrapper::Z_to_Sector_and_local_z (
            double in_z ) const  [virtual]
```

Using this method unifies the usage of coordinates.

This function takes a global z coordinate (in the computational domain) and returns both a Sector-Index and an internal z coordinate indicating which sector this coordinate belongs to and how far along in the sector it is located.

Parameters

double	in_z global system z coordinate for the transformation.

Reimplemented from SpaceTransformation.

Definition at line 107 of file DualProblemTransformationWrapper.cpp.

```
107                                                                    {
108   return st->Z_to_Sector_and_local_z(in_z);
```

```
109 }
```

References SpaceTransformation::Z_to_Sector_and_local_z().

4.17.4 Member Data Documentation

4.17.4.1 case_sectors

```
std::vector<Sector<3> > DualProblemTransformationWrapper::case_sectors
```

This member contains all the Sectors who, as a sum, form the complete Waveguide.

These Sectors are a partition of the simulated domain.

Definition at line 73 of file DualProblemTransformationWrapper.h.

4.17.4.2 epsilon_K

```
const double DualProblemTransformationWrapper::epsilon_K
```

The material-property ϵ_r has a different value inside and outside of the waveguides core.

This variable stores its value inside the core.

Definition at line 80 of file DualProblemTransformationWrapper.h.

4.17.4.3 epsilon_M

```
const double DualProblemTransformationWrapper::epsilon_M
```

The material-property ϵ_r has a different value inside and outside of the waveguides core.

This variable stores its value outside the core.

Definition at line 86 of file DualProblemTransformationWrapper.h.

4.17.4.4 sectors

```
const int DualProblemTransformationWrapper::sectors
```

Since the computational domain is split into subdomains (called sectors), it is important to keep track of the amount of subdomains.

This member stores the number of Sectors the computational domain has been split into.

Definition at line 92 of file DualProblemTransformationWrapper.h.

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

- Code/SpaceTransformations/DualProblemTransformationWrapper.h
- Code/SpaceTransformations/DualProblemTransformationWrapper.cpp

4.18 EdgeAngelingData Struct Reference

Public Attributes

- unsigned int **edge_index**
- bool **angled_in_x** = false
- bool **angled_in_y** = false

4.18.1 Detailed Description

Definition at line 64 of file Types.h.

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

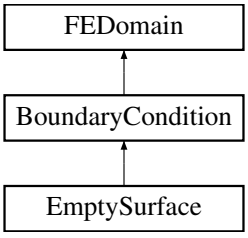
- Code/Core/Types.h

4.19 EmptySurface Class Reference

This boundary condition implements a zero surface. It is required for the sweeping scheme where the upper boundary in sweeping-direction has 0-values.

```
#include <EmptySurface.h>
```

Inheritance diagram for EmptySurface:



Public Member Functions

- **EmptySurface** (unsigned int in_bid, unsigned int in_level)
- void **fill_matrix** (dealii::PETScWrappers::SparseMatrix ∗, NumericVectorDistributed ∗rhs, Constraints ∗constraints) override
- void **fill_matrix** (dealii::SparseMatrix< ComplexNumber > ∗, Constraints ∗constraints) override
- void **fill_matrix** (dealii::PETScWrappers::SparseMatrix ∗, dealii::PETScWrappers::SparseMatrix ∗, NumericVectorDistributed ∗rhs, Constraints ∗constraints) override
- void **fill_matrix** (dealii::PETScWrappers::MPI::SparseMatrix ∗, NumericVectorDistributed ∗rhs, Constraints ∗constraints) override
- void **fill_sparsity_pattern** (dealii::DynamicSparsityPattern ∗in_dsp, Constraints ∗in_constriants) override
- bool **is_point_at_boundary** (Position2D in_p, BoundaryId in_bid) override
- void **initialize** () override
- auto **get_dof_association** () -> std::vector< InterfaceDofData > override
- auto **get_dof_association_by_boundary_id** (BoundaryId in_boundary_id) -> std::vector< InterfaceDofData > override
- std::string **output_results** (const dealii::Vector< ComplexNumber > &, std::string) override
- DofCount **compute_n_locally_owned_dofs** () override
- DofCount **compute_n_locally_active_dofs** () override
- void **determine_non_owned_dofs** () override
- auto **make_constraints** () -> Constraints override

Additional Inherited Members

4.19.1 Detailed Description

This boundary condition implements a zero surface. It is required for the sweeping scheme where the upper boundary in sweeping-direction has 0-values.

The implementation is the same as the dirichlet surface but for a predefined zero function.

Definition at line 17 of file EmptySurface.h.

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

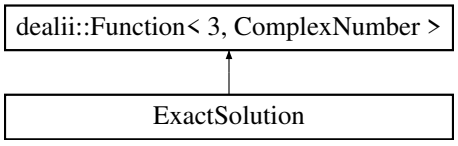
- Code/BoundaryCondition/EmptySurface.h
- Code/BoundaryCondition/EmptySurface.cpp

4.20 ExactSolution Class Reference

This class is derived from the Function class and can be used to estimate the L2-error for a straight waveguide. In the case of a completely cylindrical waveguide, an analytic solution is known (the modes of the input-signal themselves) and this class offers a representation of this analytical solution. If the waveguide has any other shape, this solution does not lose its value completely - it can still be used as a starting-vector for iterative solvers.

`#include <ExactSolution.h>`

Inheritance diagram for ExactSolution:



Public Member Functions

- **ExactSolution** (bool in_rectangular=false, bool in_dual=false)
- ComplexNumber **value** (const Position &p, const unsigned int component) const
- void **vector_value** (const Position &p, dealii::Vector< ComplexNumber > &value) const
- std::vector< std::string > **split** (std::string) const
- dealii::Tensor< 1, 3, ComplexNumber > **curl** (const Position &in_p) const
- dealii::Tensor< 1, 3, ComplexNumber > **val** (const Position &in_p) const

4.20.1 Detailed Description

This class is derived from the Function class and can be used to estimate the L2-error for a straight waveguide. In the case of a completely cylindrical waveguide, an analytic solution is known (the modes of the input-signal themselves) and this class offers a representation of this analytical solution. If the waveguide has any other shape, this solution does not lose its value completely - it can still be used as a starting-vector for iterative solvers.

The structure of this class is defined by the properties of the Function-class meaning that we have two functions:

1. virtual double value (const Point<dim> &p, const unsigned int component) calculates the value for a single component of the vector-valued return-value.
2. virtual void vector_value (const Point<dim> &p, Vector<double> &value) puts these individual components into the parameter value, which is a reference to a vector, handed over to store the result.

Author

Pascal Kraft

Date

23.11.2015

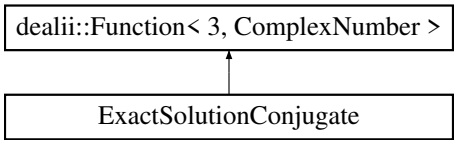
Definition at line 34 of file ExactSolution.h.

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

- Code/Solutions/ExactSolution.h
- Code/Solutions/ExactSolution.cpp

4.21 ExactSolutionConjugate Class Reference

Inheritance diagram for ExactSolutionConjugate:



Public Member Functions

- **ExactSolutionConjugate** (bool in_rectangular=false, bool in_dual=false)
- ComplexNumber **value** (const Position &p, const unsigned int component) const
- void **vector_value** (const Position &p, dealii::Vector< ComplexNumber > &value) const
- dealii::Tensor< 1, 3, ComplexNumber > **curl** (const Position &in_p) const
- dealii::Tensor< 1, 3, ComplexNumber > **val** (const Position &in_p) const

4.21.1 Detailed Description

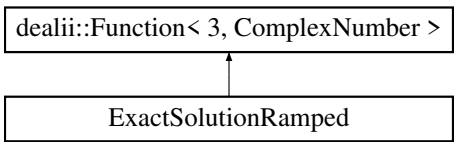
Definition at line 12 of file ExactSolutionConjugate.h.

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

- Code/Solutions/ExactSolutionConjugate.h
- Code/Solutions/ExactSolutionConjugate.cpp

4.22 ExactSolutionRamped Class Reference

Inheritance diagram for ExactSolutionRamped:



Public Member Functions

- **ExactSolutionRamped** (bool in_rectangular=false, bool in_dual=false)
- double **get_ramping_factor_for_position** (const Position &) const
- ComplexNumber **value** (const Position &p, const unsigned int component) const
- void **vector_value** (const Position &p, dealii::Vector< ComplexNumber > &value) const
- dealii::Tensor< 1, 3, ComplexNumber > **curl** (const Position &in_p) const
- dealii::Tensor< 1, 3, ComplexNumber > **val** (const Position &in_p) const
- double **compute_ramp_for_c0** (const Position &in_p) const
- double **compute_ramp_for_c1** (const Position &in_p) const
- double **ramping_delta** (const Position &in_p) const
- double **get_ramping_factor_derivative_for_position** (const Position &in_p) const

4.22.1 Detailed Description

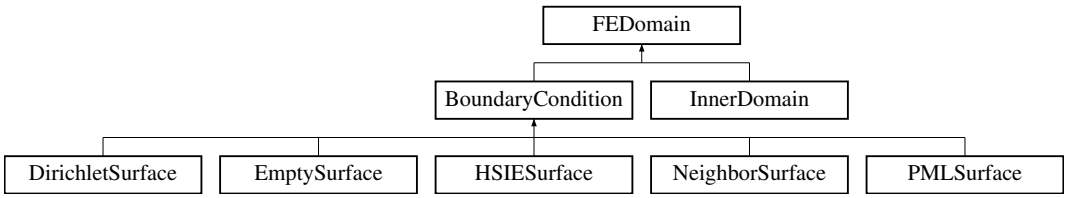
Definition at line 12 of file ExactSolutionRamped.h.

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

- Code/Solutions/ExactSolutionRamped.h
- Code/Solutions/ExactSolutionRamped.cpp

4.23 FEDomain Class Reference

Inheritance diagram for FEDomain:



Public Member Functions

- virtual void **determine_non_owned_dofs** ()=0
- void **initialize_dof_counts** (DofCount n_locally_active_dofs, DofCount n_locally_owned_dofs)
- DofIndexVector **transform_local_to_global_dofs** (DofIndexVector local_indices)
- void **mark_local_dofs_as_non_local** (DofIndexVector)
- virtual bool **finish_initialization** (DofNumber first_own_index)
- void **set_non_local_dof_indices** (DofIndexVector local_indices, DofIndexVector global_indices)
- virtual DofCount **compute_n_locally_owned_dofs** ()=0
- virtual DofCount **compute_n_locally_active_dofs** ()=0
- void **freeze_ownership** ()
- NumericVectorLocal **get_local_vector_from_global** (const NumericVectorDistributed in_vector)
- double **local_norm_of_vector** (NumericVectorDistributed $*$)

Public Attributes

- DofCount **n_locally_active_dofs**
- DofCount **n_locally_owned_dofs**
- dealii::IndexSet **global_dof_indices**
- DofIndexVector **global_index_mapping**
- std::vector$<$ bool $>$ **is_dof_owned**
- bool **is_ownership_ready**

4.23.1 Detailed Description

Definition at line 7 of file FEDomain.h.

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

- Code/Core/FEDomain.h
- Code/Core/FEDomain.cpp

4.24 FileLogger Class Reference

There will be one global instance of this object.

`#include <FileLogger.h>`

4.24.1 Detailed Description

There will be one global instance of this object.

It creates file paths and provides file names. Every IO operation will be piped through this object. The other loggers use it to persist their data.

Definition at line 14 of file FileLogger.h.

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

- Code/OutputGenerators/Files/FileLogger.h

4.25 FileMetaData Struct Reference

Public Attributes

- unsigned int **hsie_level**

4.25.1 Detailed Description

Definition at line 103 of file Types.h.

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

- Code/Core/Types.h

4.26 GeometryManager Class Reference

One object of this type is globally available to handle the geometry of the computation (what is the global computational domain, what is computed locally).

```
#include <GeometryManager.h>
```

Public Member Functions

- void **initialize** ()
- void **initialize_inner_domain** (unsigned int in_level)
- void **initialize_surfaces** ()
- void **perform_initialization** (unsigned int level)
- double **eps_kappa_2** (Position)
- double **kappa_2** ()
- std::pair< double, double > **compute_x_range** ()
- std::pair< double, double > **compute_y_range** ()
- std::pair< double, double > **compute_z_range** ()
- void **set_x_range** (std::pair< double, double >)
- void **set_y_range** (std::pair< double, double >)
- void **set_z_range** (std::pair< double, double >)
- std::pair< bool, unsigned int > **get_global_neighbor_for_interface** (Direction)
- std::pair< bool, unsigned int > **get_level_neighbor_for_interface** (Direction, unsigned int)
- bool **math_coordinate_in_waveguide** (Position) const
- dealii::Tensor< 2, 3 > **get_epsilon_tensor** (const Position &)
- double **get_epsilon_for_point** (const Position &)
- auto **get_boundary_for_direction** (Direction) -> BoundaryId
- auto **get_direction_for_boundary_id** (BoundaryId) -> Direction
- void **validate_global_dof_indices** (unsigned int in_level)
- SurfaceType **get_surface_type** (BoundaryId b_id, unsigned int level)
- void **distribute_dofs_on_level** (unsigned int level)
- void **set_surface_types_and_properties** (unsigned int level)
- void **initialize_surfaces_on_level** (unsigned int level)
- void **initialize_level** (unsigned int level)
- void **print_level_dof_counts** (unsigned int level)

Public Attributes

- double **input_connector_length**
- double **output_connector_length**
- double **shape_sector_length**
- unsigned int **shape_sector_count**
- unsigned int **local_inner_dofs**
- bool **are_surface_meshes_initialized**
- double **h_x**
- double **h_y**
- double **h_z**
- std::array< unsigned int, 6 > **dofs_at_surface**
- std::array< dealii::Triangulation< 2, 2 >, 6 > **surface_meshes**
- std::array< double, 6 > **surface_extremal_coordinate**
- std::pair< double, double > **local_x_range**
- std::pair< double, double > **local_y_range**
- std::pair< double, double > **local_z_range**
- std::pair< double, double > **global_x_range**
- std::pair< double, double > **global_y_range**
- std::pair< double, double > **global_z_range**
- std::array< LevelGeometry, 4 > **levels**

4.26.1 Detailed Description

One object of this type is globally available to handle the geometry of the computation (what is the global computational domain, what is computed locally).

This object is one of the first to be initialized. It contains the coordinate ranges locally and globally. It also has several LevelGeometry objects in a vector. This is the core data behind the sweeping hierarchy. These level objects contain:

- the surface types for all boundaries on this level
- pointers to the boundary condition objects
- dof counting data (how many dofs exist on the level, how many dofs does this process own on this level) and also which dofs are stored where in the dof_distribution member.

This object can also determine if a coordinate is inside or outside of the waveguide and computes kappa squared required for the assembly of Maxwell's equations.

Definition at line 35 of file GeometryManager.h.

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

- Code/GlobalObjects/GeometryManager.h
- Code/GlobalObjects/GeometryManager.cpp

4.27 GradientTable Class Reference

The Gradient Table is an OutputGenerator, intended to write information about the shape gradient to the console upon its computation.

`#include <GradientTable.h>`

Public Member Functions

- **GradientTable** (unsigned int in_step, dealii::Vector< double > in_configuration, double in_quality, dealii::Vector< double > in_last_configuration, double in_last_quality)
- void **SetInitialQuality** (double in_quality)
- void **AddComputationResult** (int in_component, double in_step, double in_quality)
- void **AddFullStepResult** (dealii::Vector< double > in_step, double in_quality)
- void **PrintFullLine** ()
- void **PrintTable** ()
- void **WriteTableToFile** (std::string in_filename)

Public Attributes

- const int **ndofs**
- const int **nfreedofs**
- const unsigned int **GlobalStep**

4.27.1 Detailed Description

The Gradient Table is an OutputGenerator, intended to write information about the shape gradient to the console upon its computation.

Date

28.11.2016

Author

Pascal Kraft

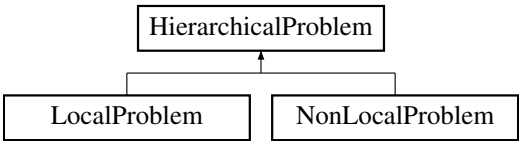
Definition at line 12 of file GradientTable.h.

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

- Code/OutputGenerators/Console/GradientTable.h
- Code/OutputGenerators/Console/GradientTable.cpp

4.28 HierarchicalProblem Class Reference

Inheritance diagram for HierarchicalProblem:



Public Member Functions

- **HierarchicalProblem** (unsigned int level, SweepingDirection direction)
- virtual void **solve** ()=0
- void **solve_with_timers_and_count** ()
- virtual void **initialize** ()=0
- void **make_constraints** ()
- virtual void **assemble** ()=0
- virtual void **initialize_index_sets** ()=0
- void **constrain_identical_dof_sets** (std::vector< unsigned int > *set_one, std::vector< unsigned int > *set_two, Constraints *affine_constraints)
- virtual auto **reinit** () -> void=0
- auto **opposing_site_bid** (BoundaryId) -> BoundaryId
- void **compute_final_rhs_mismatch** ()
- virtual void **compute_solver_factorization** ()=0
- std::string **output_results** (std::string in_fname_part="solution_inner_domain_level")
- virtual void **reinit_rhs** ()=0
- virtual void **make_sparsity_pattern** ()
- void **initialize_dof_counts** ()

Public Attributes

- SweepingDirection **sweeping_direction**
- const SweepingLevel **level**
- Constraints **constraints**
- std::array< dealii::IndexSet, 6 > **surface_index_sets**
- std::array< bool, 6 > **is_hsie_surface**
- std::vector< bool > **is_surface_locked**
- bool **is_dof_manager_set**
- bool **has_child**
- HierarchicalProblem * **child**
- dealii::SparsityPattern **sp**
- DofIndexData **indices**
- NumericVectorDistributed **solution**
- NumericVectorDistributed **direct_solution**
- NumericVectorDistributed **solution_error**
- NumericVectorDistributed **rhs**
- dealii::IndexSet **own_dofs**
- dealii::IndexSet **current_upper_sweeping_dofs**
- dealii::IndexSet **current_lower_sweeping_dofs**
- std::array< std::vector< InterfaceDofData >, 6 > **surface_dof_associations**
- dealii::PETScWrappers::MPI::SparseMatrix * **matrix**
- std::vector< std::string > **filenames**
- ResidualOutputGenerator * **residual_output**
- unsigned int **solve_counter**

4.28.1 Detailed Description

Definition at line 16 of file HierarchicalProblem.h.

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

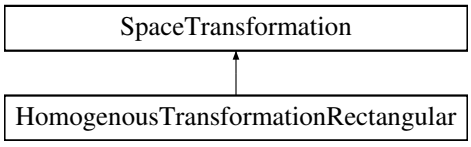
- Code/Hierarchy/HierarchicalProblem.h
- Code/Hierarchy/HierarchicalProblem.cpp

4.29 HomogenousTransformationRectangular Class Reference

For this transformation we try to achieve a situation in which tensorial material properties from the coordinate transformation and PML-regions dont overlap.

`#include <HomogenousTransformationRectangular.h>`

Inheritance diagram for HomogenousTransformationRectangular:



Public Member Functions

- Position **math_to_phys** (Position coord) const
- Position **phys_to_math** (Position coord) const
- dealii::Tensor< 2, 3, ComplexNumber > **get_Tensor** (Position &coordinate) const
- dealii::Tensor< 2, 3, double > **get_Space_Transformation_Tensor** (Position &coordinate) const
- dealii::Tensor< 2, 3, double > **get_Space_Transformation_Tensor_Homogenized** (Position &coordinate) const
- void estimate_and_initialize ()

  At the beginning (before the first solution of a system) only the boundary conditions for the shape of the waveguide are known.
- double get_Q1 (double z) const

  This member calculates the value of Q1 for a provided z-coordinate.
- double get_Q2 (double z) const

  This member calculates the value of Q2 for a provided z-coordinate.
- double get_Q3 (double z) const

  This member calculates the value of Q3 for a provided z-coordinate.
- double get_dof (int dof) const

  This is a getter for the values of degrees of freedom.
- void set_dof (int dof, double value)

  This function sets the value of the dof provided to the given value.
- double get_free_dof (int dof) const

  This is a getter for the values of degrees of freedom.
- void set_free_dof (int dof, double value)

  This function sets the value of the dof provided to the given value.
- double System_Length () const

  Returns the complete length of the computational domain.
- double Sector_Length () const

  Returns the length of one sector.
- double get_r (double in_z) const

  Returns the radius for a system-coordinate;.
- double get_m (double in_z) const

  Returns the shift for a system-coordinate;.
- double get_v (double in_z) const

  Returns the tilt for a system-coordinate;.
- Vector< double > Dofs () const

  Other objects can use this function to retrieve an array of the current values of the degrees of freedom of the functional we are optimizing.
- unsigned int NFreeDofs () const

  This function returns the number of unrestrained degrees of freedom of the current optimization run.
- unsigned int NDofs () const

  This function returns the total number of DOFs including restrained ones.
- bool IsDofFree (int) const

  Since Dofs() also returns restrained degrees of freedom, this function can be applied to determine if a degree of freedom is indeed free or restrained.
- void Print () const

  Console output of the current Waveguide Structure.
- ComplexNumber **evaluate_for_z_with_sum** (double, double, InnerDomain *)

Public Attributes

- std::vector< Sector< 2 > > case_sectors

  This member contains all the Sectors who, as a sum, form the complete Waveguide.

- const double epsilon_K

  The material-property ϵ_r has a different value inside and outside of the waveguides core.

- const double epsilon_M

  The material-property ϵ_r has a different value inside and outside of the waveguides core.

- const int sectors

  Since the computational domain is split into subdomains (called sectors), it is important to keep track of the amount of subdomains.

- const double deltaY

  This value is initialized with the value Delta from the input-file.

- Vector< double > InitialDofs

  This vector of values saves the initial configuration.

4.29.1 Detailed Description

For this transformation we try to achieve a situation in which tensorial material properties from the coordinate transformation and PML-regions dont overlap.

The usage of a coordinate transformation which is identity on the domain containing our PML is a strong restriction however it ensures lower errors since the quality of the PML is harder to estimate otherwise. Also it limits us in how we model the waveguide essentially forcing us to have no bent between the wavguides-connectors.

Author

Pascal Kraft

Date

28.11.2016

Definition at line 27 of file HomogenousTransformationRectangular.h.

4.29.2 Member Function Documentation

4.29.2.1 Dofs()

```
Vector< double > HomogenousTransformationRectangular::Dofs ( ) const  [virtual]
```

Other objects can use this function to retrieve an array of the current values of the degrees of freedom of the functional we are optimizing.

This also includes restrained degrees of freedom and other functions can be used to determine this property. This has to be done because in different cases the number of restrained degrees of freedom can vary and we want no logic about this in other functions.

Implements SpaceTransformation.

Definition at line 293 of file HomogenousTransformationRectangular.cpp.

```
293                                                                    {
294   Vector<double> ret;
295   const int total = NDofs();
296   ret.reinit(total);
297   for (int i = 0; i < total; i++) {
298     ret[i] = get_dof(i);
299   }
300   return ret;
301 }
```

References get_dof(), and NDofs().

4.29.2.2 estimate_and_initialize()

```
void HomogenousTransformationRectangular::estimate_and_initialize ( )  [virtual]
```

At the beginning (before the first solution of a system) only the boundary conditions for the shape of the waveguide are known.

Therefore the values for the degrees of freedom need to be estimated. This function sets all variables to appropiate values and estimates an appropriate shape based on averages and a polynomial interpolation of the boundary conditions on the shape.

Implements SpaceTransformation.

Definition at line 192 of file HomogenousTransformationRectangular.cpp.

```
192                                                                    {
193   if (GlobalParams.Use_Predefined_Shape) {
194     Sector<2> the_first(true, false, GlobalParams.sd.z[0], GlobalParams.sd.z[1]);
195     the_first.set_properties_force(GlobalParams.sd.m[0], GlobalParams.sd.m[1], GlobalParams.sd.v[0],
       GlobalParams.sd.v[1]);
196     case_sectors.push_back(the_first);
197     for (int i = 1; i < GlobalParams.sd.Sectors - 2; i++) {
198       Sector<2> intermediate(false, false, GlobalParams.sd.z[i],
199                              GlobalParams.sd.z[i + 1]);
200       intermediate.set_properties_force(
201           GlobalParams.sd.m[i], GlobalParams.sd.m[i + 1], GlobalParams.sd.v[i],
202           GlobalParams.sd.v[i + 1]);
203       case_sectors.push_back(intermediate);
204     }
205     Sector<2> the_last(false, true,
206                        GlobalParams.sd.z[GlobalParams.sd.Sectors - 2],
207                        GlobalParams.sd.z[GlobalParams.sd.Sectors - 1]);
208     the_last.set_properties_force(
209         GlobalParams.sd.m[GlobalParams.sd.Sectors - 2],
210         GlobalParams.sd.m[GlobalParams.sd.Sectors - 1],
211         GlobalParams.sd.v[GlobalParams.sd.Sectors - 2],
212         GlobalParams.sd.v[GlobalParams.sd.Sectors - 1]);
213     case_sectors.push_back(the_last);
214     for (unsigned int i = 0; i < case_sectors.size(); i++) {
215       deallog << "From z: " << case_sectors[i].z_0
216               << "(m: " << case_sectors[i].get_m(0.0)
```

```
217               « " v: " « case_sectors[i].get_v(0.0) « ")" « std::endl;
218       deallog « "  To z: " « case_sectors[i].z_1
219               « "(m: " « case_sectors[i].get_m(1.0)
220               « " v: " « case_sectors[i].get_v(1.0) « ")" « std::endl;
221     }
222   } else {
223     case_sectors.reserve(sectors);
224     double m_0 = GlobalParams.Vertical_displacement_of_waveguide / 2.0;
225     double m_1 = -GlobalParams.Vertical_displacement_of_waveguide / 2.0;
226     if (sectors == 1) {
227       Sector<2> temp12(true, true, -GlobalParams.Geometry_Size_Z / 2.0,
228                        GlobalParams.Geometry_Size_Z / 2.0);
229       case_sectors.push_back(temp12);
230       case_sectors[0].set_properties_force(
231           GlobalParams.Vertical_displacement_of_waveguide / 2.0,
      -GlobalParams.Vertical_displacement_of_waveguide / 2.0,
232           GlobalParams.Width_of_waveguide, GlobalParams.Width_of_waveguide, 0, 0);
233     } else {
234       double length = Sector_Length();
235       Sector<2> temp(true, false, -GlobalParams.Geometry_Size_Z / (2.0),
236                      -GlobalParams.Geometry_Size_Z / 2.0 + length);
237       case_sectors.push_back(temp);
238       for (int i = 1; i < sectors; i++) {
239         Sector<2> temp2(false, false,
240                         -GlobalParams.Geometry_Size_Z / (2.0) + length * (1.0 * i),
241                         -GlobalParams.Geometry_Size_Z / (2.0) + length * (i + 1.0));
242         case_sectors.push_back(temp2);
243       }
244
245       double length_rel = 1.0 / ((double)(sectors));
246       case_sectors[0].set_properties_force(
247           m_0, InterpolationPolynomialZeroDerivative(length_rel, m_0, m_1), 0,
248           InterpolationPolynomialDerivative(length_rel, m_0, m_1, 0, 0));
249       for (int i = 1; i < sectors; i++) {
250         double z_l = i * length_rel;
251         double z_r = (i + 1) * length_rel;
252         case_sectors[i].set_properties_force(
253             InterpolationPolynomialZeroDerivative(z_l, m_0, m_1),
254             InterpolationPolynomialZeroDerivative(z_r, m_0, m_1),
255             InterpolationPolynomialDerivative(z_l, m_0, m_1, 0, 0),
256             InterpolationPolynomialDerivative(z_r, m_0, m_1, 0, 0));
257       }
258     }
259   }
260 }
```

References case_sectors, Sector_Length(), sectors, and Sector< Dofs_Per_Sector >::set_properties_force().

4.29.2.3 get_dof()

```
double HomogenousTransformationRectangular::get_dof (
            int dof ) const  [virtual]
```

This is a getter for the values of degrees of freedom.

A getter-setter interface was introduced since the values are estimated automatically during the optimization and non-physical systems should be excluded from the domain of possible cases.

Parameters

dof	The index of the degree of freedom to be retrieved from the structure of the modelled waveguide.

Returns

This function returns the value of the requested degree of freedom. Should this dof not exist, 0 will be returnd.

Implements SpaceTransformation.

Definition at line 117 of file HomogenousTransformationRectangular.cpp.

```
117                                                                    {
118   if (dof < (int)NDofs() && dof >= 0) {
119     int sector = floor(dof / 2);
120     if (sector == sectors) {
121       return case_sectors[sector - 1].dofs_r[dof % 2];
122     } else {
123       return case_sectors[sector].dofs_l[dof % 2];
124     }
125   } else {
126     std::cout « "Critical: DOF-index out of bounds in "
127                 "HomogenousTransformationRectangular::get_dof!"
128               « std::endl;
129     return 0.0;
130   }
131 }
```

References case_sectors, NDofs(), and sectors.

Referenced by Dofs().

4.29.2.4 get_free_dof()

```
double HomogenousTransformationRectangular::get_free_dof (
            int dof ) const  [virtual]
```

This is a getter for the values of degrees of freedom.

A getter-setter interface was introduced since the values are estimated automatically during the optimization and non-physical systems should be excluded from the domain of possible cases.

Parameters

dof	The index of the degree of freedom to be retrieved from the structure of the modelled waveguide.

Returns

This function returns the value of the requested degree of freedom. Should this dof not exist, 0 will be returnd.

Implements SpaceTransformation.

Definition at line 133 of file HomogenousTransformationRectangular.cpp.

```
133                                                                    {
134   int dof = in_dof + 2;
135   if (dof < (int)NDofs() - 2 && dof >= 0) {
136     int sector = floor(dof / 2);
137     if (sector == sectors) {
138       return case_sectors[sector - 1].dofs_r[dof % 2];
139     } else {
140       return case_sectors[sector].dofs_l[dof % 2];
141     }
142   } else {
143     std::cout « "Critical: DOF-index out of bounds in "
144                 "HomogenousTransformationRectangular::get_free_dof!"
145               « std::endl;
146     return 0.0;
147   }
148 }
```

References case_sectors, NDofs(), and sectors.

4.29.2.5 get_Q1()

```
double HomogenousTransformationRectangular::get_Q1 (
            double z ) const  [virtual]
```

This member calculates the value of Q1 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q1 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implements SpaceTransformation.

Definition at line 278 of file HomogenousTransformationRectangular.cpp.

```
278                                                                    {
279   std::pair<int, double> two = Z_to_Sector_and_local_z(z_in);
280   return case_sectors[two.first].getQ1(two.second);
281 }
```

References case_sectors, and SpaceTransformation::Z_to_Sector_and_local_z().

4.29.2.6 get_Q2()

```
double HomogenousTransformationRectangular::get_Q2 (
            double z ) const  [virtual]
```

This member calculates the value of Q2 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q2 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implements SpaceTransformation.

Definition at line 283 of file HomogenousTransformationRectangular.cpp.

```
283                                                                    {
284   std::pair<int, double> two = Z_to_Sector_and_local_z(z_in);
285   return case_sectors[two.first].getQ2(two.second);
286 }
```

References case_sectors, and SpaceTransformation::Z_to_Sector_and_local_z().

4.29.2.7 get_Q3()

```
double HomogenousTransformationRectangular::get_Q3 (
            double z ) const  [virtual]
```

This member calculates the value of Q3 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q3 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implements SpaceTransformation.

Definition at line 288 of file HomogenousTransformationRectangular.cpp.

```
288                                                                    {
289   std::pair<int, double> two = Z_to_Sector_and_local_z(z_in);
290   return case_sectors[two.first].getQ3(two.second);
291 }
```

References case_sectors, and SpaceTransformation::Z_to_Sector_and_local_z().

4.29.2.8 IsDofFree()

```
bool HomogenousTransformationRectangular::IsDofFree (
            int index ) const  [virtual]
```

Since Dofs() also returns restrained degrees of freedom, this function can be applied to determine if a degree of freedom is indeed free or restrained.

"restrained" means that for example the DOF represents the radius at one of the connectors (input or output) and therefore we forbid the optimization scheme to vary this value.

Implements SpaceTransformation.

Definition at line 307 of file HomogenousTransformationRectangular.cpp.

```
307                                                                    {
308   return index > 1 && index < (int)NDofs() - 1;
309 }
```

References NDofs().

4.29.2.9 NDofs()

```
unsigned int HomogenousTransformationRectangular::NDofs ( ) const  [virtual]
```

This function returns the total number of DOFs including restrained ones.

This is the lenght of the array returned by Dofs().

Implements SpaceTransformation.

Definition at line 315 of file HomogenousTransformationRectangular.cpp.

```
315                                                                    {
316   return sectors * 2 + 2;
317 }
```

References sectors.

Referenced by Dofs(), get_dof(), get_free_dof(), IsDofFree(), NFreeDofs(), set_dof(), and set_free_dof().

4.29.2.10 set_dof()

```
void HomogenousTransformationRectangular::set_dof (
            int dof,
            double value )  [virtual]
```

This function sets the value of the dof provided to the given value.

It is important to consider, that some dofs are non-writable (i.e. the values of the degrees of freedom on the boundary, like the radius of the input-connector cannot be changed).

Parameters

dof	The index of the parameter to be changed.
value	The value, the dof should be set to.

Implements SpaceTransformation.

Definition at line 150 of file HomogenousTransformationRectangular.cpp.

```
150                                                                    {
151   if (dof < (int)NDofs() && dof >= 0) {
152     int sector = floor(dof / 2);
153     if (sector == sectors) {
154       case_sectors[sector - 1].dofs_r[dof % 2] = in_val;
155     } else if (sector == 0) {
156       case_sectors[0].dofs_l[dof % 2] = in_val;
157     } else {
158       case_sectors[sector].dofs_l[dof % 2] = in_val;
159       case_sectors[sector - 1].dofs_r[dof % 2] = in_val;
160     }
161   } else {
162     std::cout « "Critical: DOF-index out of bounds in "
163                  "HomogenousTransformationRectangular::set_dof!"
164               « std::endl;
165   }
166 }
```

References case_sectors, NDofs(), and sectors.

4.29.2.11 set_free_dof()

```
void HomogenousTransformationRectangular::set_free_dof (
            int dof,
            double value )  [virtual]
```

This function sets the value of the dof provided to the given value.

It is important to consider, that some dofs are non-writable (i.e. the values of the degrees of freedom on the boundary, like the radius of the input-connector cannot be changed).

Parameters

dof	The index of the parameter to be changed.
value	The value, the dof should be set to.

Implements SpaceTransformation.

Definition at line 168 of file HomogenousTransformationRectangular.cpp.

```
169                                                                                {
170   int dof = in_dof + 2;
171   if (dof < (int)NDofs() - 2 && dof >= 0) {
172     int sector = floor(dof / 2);
173     if (sector == sectors) {
174       case_sectors[sector - 1].dofs_r[dof % 2] = in_val;
175     } else if (sector == 0) {
176       case_sectors[0].dofs_l[dof % 2] = in_val;
177     } else {
178       case_sectors[sector].dofs_l[dof % 2] = in_val;
179       case_sectors[sector - 1].dofs_r[dof % 2] = in_val;
180     }
181   } else {
182     std::cout « "Critical: DOF-index out of bounds in "
183                 "HomogenousTransformationRectangular::set_free_dof!"
184               « std::endl;
185   }
186 }
```

References case_sectors, NDofs(), and sectors.

4.29.3 Member Data Documentation

4.29.3.1 case_sectors

```
std::vector<Sector<2> > HomogenousTransformationRectangular::case_sectors
```

This member contains all the Sectors who, as a sum, form the complete Waveguide.

These Sectors are a partition of the simulated domain.

Definition at line 50 of file HomogenousTransformationRectangular.h.

Referenced by estimate_and_initialize(), get_dof(), get_free_dof(), get_m(), get_Q1(), get_Q2(), get_Q3(), get_v(), set_dof(), and set_free_dof().

4.29.3.2 epsilon_K

```
const double HomogenousTransformationRectangular::epsilon_K
```

The material-property ϵ_r has a different value inside and outside of the waveguides core.

This variable stores its value inside the core.

Definition at line 57 of file HomogenousTransformationRectangular.h.

4.29.3.3 epsilon_M

```
const double HomogenousTransformationRectangular::epsilon_M
```

The material-property ϵ_r has a different value inside and outside of the waveguides core.

This variable stores its value outside the core.

Definition at line 63 of file HomogenousTransformationRectangular.h.

4.29.3.4 sectors

```
const int HomogenousTransformationRectangular::sectors
```

Since the computational domain is split into subdomains (called sectors), it is important to keep track of the amount of subdomains.

This member stores the number of Sectors the computational domain has been split into.

Definition at line 69 of file HomogenousTransformationRectangular.h.

Referenced by estimate_and_initialize(), get_dof(), get_free_dof(), NDofs(), set_dof(), and set_free_dof().

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

- Code/SpaceTransformations/HomogenousTransformationRectangular.h
- Code/SpaceTransformations/HomogenousTransformationRectangular.cpp

4.30 HSIEPolynomial Class Reference

This class basically represents a polynomial and its derivative. It is required for the HSIE implementation.

```
#include <HSIEPolynomial.h>
```

Public Member Functions

- ComplexNumber **evaluate** (ComplexNumber x)
- ComplexNumber **evaluate_dx** (ComplexNumber x)
- void **update_derivative** ()
- **HSIEPolynomial** (unsigned int, ComplexNumber)
- **HSIEPolynomial** (DofData &, ComplexNumber)
- **HSIEPolynomial** (std::vector< ComplexNumber > in_a, ComplexNumber k0)
- HSIEPolynomial **applyD** ()
- HSIEPolynomial **applyI** ()
- void **multiplyBy** (ComplexNumber factor)
- void **multiplyBy** (double factor)
- void **applyTplus** (ComplexNumber u_0)
- void **applyTminus** (ComplexNumber u_0)
- void **applyDerivative** ()
- void **add** (HSIEPolynomial b)

Static Public Member Functions

- static void **computeDandI** (unsigned int, ComplexNumber)
- static HSIEPolynomial **PsiMinusOne** (ComplexNumber k0)
- static HSIEPolynomial **PsiJ** (int j, ComplexNumber k0)
- static HSIEPolynomial **ZeroPolynomial** ()
- static HSIEPolynomial **PhiMinusOne** (ComplexNumber k0)
- static HSIEPolynomial **PhiJ** (int j, ComplexNumber k0)

Public Attributes

- std::vector< ComplexNumber > **a**
- std::vector< ComplexNumber > **da**
- ComplexNumber **k0**

Static Public Attributes

- static bool **matricesLoaded** = false
- static dealii::FullMatrix< ComplexNumber > **D**
- static dealii::FullMatrix< ComplexNumber > **I**

4.30.1 Detailed Description

This class basically represents a polynomial and its derivative. It is required for the HSIE implementation.

The core data in this class is a vector a, which stores the coefficients of the polynomials and a vector da, which stores the coefficients of the derivative. Both can be evaluated for a given x with the respective functions. Additionally, there are functions to initialize a polynomial that are required by the hardy space infinite elements and some operators can be applied (like T_plus and T_minus). As an important remark: The value kappa_0 used in HSIE is also kept in these values because we want to be able to apply the operators D and I to one a polynomial. Since they aren't cheap to compute, I precomute them once as static members of this class. If you only intend to use evaluation, evaluation of the derivative, summation and multiplication with constants, then that value is not relevant.

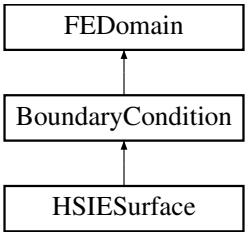
Definition at line 18 of file HSIEPolynomial.h.

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

- Code/BoundaryCondition/HSIEPolynomial.h
- Code/BoundaryCondition/HSIEPolynomial.cpp

4.31 HSIESurface Class Reference

Inheritance diagram for HSIESurface:



Public Member Functions

- **HSIESurface** (unsigned int surface, unsigned int level)
- std::vector< HSIEPolynomial > **build_curl_term_q** (unsigned int, const dealii::Tensor< 1, 2 >)
- std::vector< HSIEPolynomial > **build_curl_term_nedelec** (unsigned int, const dealii::Tensor< 1, 2 >, const dealii::Tensor< 1, 2 >, const double, const double)
- std::vector< HSIEPolynomial > **build_non_curl_term_q** (unsigned int, const double)
- std::vector< HSIEPolynomial > **build_non_curl_term_nedelec** (unsigned int, const double, const double)
- void **set_V0** (Position)
- auto **get_dof_data_for_cell** (CellIterator2D, CellIterator2D) -> DofDataVector
- void **fill_matrix** (dealii::PETScWrappers::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_matrix** (dealii::SparseMatrix< ComplexNumber > *, Constraints *constraints) override
- void **fill_matrix** (dealii::PETScWrappers::SparseMatrix *, dealii::PETScWrappers::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_matrix** (dealii::PETScWrappers::MPI::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_sparsity_pattern** (dealii::DynamicSparsityPattern *in_dsp, Constraints *in_constriants) override
- bool **is_point_at_boundary** (Position2D in_p, BoundaryId in_bid) override
- auto **get_vertices_for_boundary_id** (BoundaryId) -> std::vector< unsigned int >
- auto **get_n_vertices_for_boundary_id** (BoundaryId) -> unsigned int
- auto **get_lines_for_boundary_id** (BoundaryId) -> std::vector< unsigned int >
- auto **get_n_lines_for_boundary_id** (BoundaryId) -> unsigned int
- auto **compute_n_edge_dofs** () -> DofCountsStruct
- auto **compute_n_vertex_dofs** () -> DofCountsStruct
- auto **compute_n_face_dofs** () -> DofCountsStruct
- auto **compute_dofs_per_edge** (bool only_hsie_dofs) -> DofCount
- auto **compute_dofs_per_face** (bool only_hsie_dofs) -> DofCount
- auto **compute_dofs_per_vertex** () -> DofCount
- void **initialize** () override
- void **initialize_dof_handlers_and_fe** ()
- void **update_dof_counts_for_edge** (CellIterator2D cell, unsigned int edge, DofCountsStruct &)
- void **update_dof_counts_for_face** (CellIterator2D cell, DofCountsStruct &)
- void **update_dof_counts_for_vertex** (CellIterator2D cell, unsigned int edge, unsigned int vertex, DofCountsStruct &)
- void **register_new_vertex_dofs** (CellIterator2D cell, unsigned int edge, unsigned int vertex)
- void **register_new_edge_dofs** (CellIterator2D cell, CellIterator2D cell_2, unsigned int edge)
- void **register_new_surface_dofs** (CellIterator2D cell, CellIterator2D cell2)
- auto **register_dof** () -> DofNumber
- void **register_single_dof** (std::string in_id, int in_hsie_order, int in_inner_order, DofType in_dof_type, Dof↩ DataVector &, unsigned int)
- void **register_single_dof** (unsigned int in_id, int in_hsie_order, int in_inner_order, DofType in_dof_type, DofDataVector &, unsigned int, bool orientation=true)
- ComplexNumber **evaluate_a** (std::vector< HSIEPolynomial > &u, std::vector< HSIEPolynomial > &v, dealii::Tensor< 2, 3, double > G)
- void **transform_coordinates_in_place** (std::vector< HSIEPolynomial > *)
- bool **check_dof_assignment_integrity** ()
- bool **check_number_of_dofs_for_cell_integrity** ()
- auto **get_dof_data_for_base_dof_nedelec** (DofNumber base_dof_index) -> DofDataVector
- auto **get_dof_data_for_base_dof_q** (DofNumber base_dof_index) -> DofDataVector
- auto **get_dof_association** () -> std::vector< InterfaceDofData > override
- auto **undo_transform** (dealii::Point< 2 >) -> Position
- auto **undo_transform_for_shape_function** (dealii::Point< 2 >) -> Position
- void **add_surface_relevant_dof** (InterfaceDofData in_index_and_orientation)

- auto **get_dof_association_by_boundary_id** (BoundaryId in_boundary_id) -> std::vector< InterfaceDofData > override
- void **clear_user_flags** ()
- void **set_b_id_uses_hsie** (unsigned int, bool)
- auto **build_fad_for_cell** (CellIterator2D cell) -> FaceAngelingData
- void **compute_extreme_vertex_coordinates** ()
- auto **vertex_positions_for_ids** (std::vector< unsigned int > ids) -> std::vector< Position >
- auto **line_positions_for_ids** (std::vector< unsigned int > ids) -> std::vector< Position >
- std::string **output_results** (const dealii::Vector< ComplexNumber > &, std::string) override
- DofCount **compute_n_locally_owned_dofs** () override
- DofCount **compute_n_locally_active_dofs** () override
- void **finish_dof_index_initialization** () override
- void **determine_non_owned_dofs** () override
- dealii::IndexSet **compute_non_owned_dofs** ()
- bool **finish_initialization** (DofNumber first_own_index) override

Public Attributes

- DofDataVector **face_dof_data**
- DofDataVector **edge_dof_data**
- DofDataVector **vertex_dof_data**
- DofCount **n_edge_dofs**
- DofCount **n_face_dofs**
- DofCount **n_vertex_dofs**

4.31.1 Detailed Description

Definition at line 19 of file HSIESurface.h.

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

- Code/BoundaryCondition/HSIESurface.h
- Code/BoundaryCondition/HSIESurface.cpp

4.32 InhomogenousTransformationRectangle Class Reference

In this case we regard a rectangular waveguide and the effects on the material tensor by the space transformation and the boundary condition PML may overlap (hence inhomogenous space transformation)

`#include <InhomogenousTransformationRectangular.h>`

4.32.1 Detailed Description

In this case we regard a rectangular waveguide and the effects on the material tensor by the space transformation and the boundary condition PML may overlap (hence inhomogenous space transformation)

If this kind of boundary condition works stably we will also be able to deal with more general settings (which might for example incorporate angles in between the output and input connector.

Author

Pascal Kraft

Date

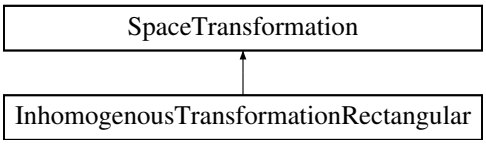
28.11.2016

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

- Code/SpaceTransformations/InhomogenousTransformationRectangular.h

4.33 InhomogenousTransformationRectangular Class Reference

Inheritance diagram for InhomogenousTransformationRectangular:



Public Member Functions

- Position **math_to_phys** (Position coord) const
- Position **phys_to_math** (Position coord) const
- dealii::Tensor< 2, 3, ComplexNumber > **get_Tensor** (Position &coordinate) const
- dealii::Tensor< 2, 3, double > **get_Space_Transformation_Tensor** (Position &coordinate) const
- dealii::Tensor< 2, 3, double > **get_Space_Transformation_Tensor_Homogenized** (Position &coordinate) const
- void estimate_and_initialize ()

  At the beginning (before the first solution of a system) only the boundary conditions for the shape of the waveguide are known.
- double get_Q1 (double z) const

  This member calculates the value of Q1 for a provided z-coordinate.
- double get_Q2 (double z) const

  This member calculates the value of Q2 for a provided z-coordinate.
- double get_Q3 (double z) const

  This member calculates the value of Q3 for a provided z-coordinate.
- double get_dof (int dof) const

  This is a getter for the values of degrees of freedom.

- void set_dof (int dof, double value)

  This function sets the value of the dof provided to the given value.
- double get_free_dof (int dof) const

  This is a getter for the values of degrees of freedom.
- void set_free_dof (int dof, double value)

  This function sets the value of the dof provided to the given value.
- double System_Length () const

  Returns the complete length of the computational domain.
- double Sector_Length () const

  Returns the length of one sector.
- double Layer_Length () const

  Returns the length of one layer.
- double get_r (double in_z) const

  Returns the radius for a system-coordinate;.
- double get_m (double in_z) const

  Returns the shift for a system-coordinate;.
- double get_v (double in_z) const

  Returns the tilt for a system-coordinate;.
- Vector< double > Dofs () const

  Other objects can use this function to retrieve an array of the current values of the degrees of freedom of the functional we are optimizing.
- unsigned int NFreeDofs () const

  This function returns the number of unrestrained degrees of freedom of the current optimization run.
- unsigned int NDofs () const

  This function returns the total number of DOFs including restrained ones.
- bool IsDofFree (int) const

  Since Dofs() also returns restrained degrees of freedom, this function can be applied to determine if a degree of freedom is indeed free or restrained.
- void Print () const

  Console output of the current Waveguide Structure.
- ComplexNumber **evaluate_for_z_with_sum** (double, double, InnerDomain *)
- ComplexNumber **evaluate_for_z** (double z_in, InnerDomain *)

Public Attributes

- std::vector< Sector< 2 > > case_sectors

  This member contains all the Sectors who, as a sum, form the complete Waveguide.
- const double epsilon_K

  The material-property ϵ_r has a different value inside and outside of the waveguides core.
- const double epsilon_M

  The material-property ϵ_r has a different value inside and outside of the waveguides core.
- const int sectors

  Since the computational domain is split into subdomains (called sectors), it is important to keep track of the amount of subdomains.
- const double deltaY

  This value is initialized with the value Delta from the input-file.
- Vector< double > InitialDofs

  This vector of values saves the initial configuration.

4.33.1 Detailed Description

Definition at line 24 of file InhomogenousTransformationRectangular.h.

4.33.2 Member Function Documentation

4.33.2.1 Dofs()

```
Vector< double > InhomogenousTransformationRectangular::Dofs ( ) const  [virtual]
```

Other objects can use this function to retrieve an array of the current values of the degrees of freedom of the functional we are optimizing.

This also includes restrained degrees of freedom and other functions can be used to determine this property. This has to be done because in different cases the number of restrained degrees of freedom can vary and we want no logic about this in other functions.

Implements SpaceTransformation.

Definition at line 224 of file InhomogenousTransformationRectangular.cpp.

```
224                                                                    {
225   Vector<double> ret;
226   const int total = NDofs();
227   ret.reinit(total);
228   for (int i = 0; i < total; i++) {
229     ret[i] = get_dof(i);
230   }
231   return ret;
232 }
```

References get_dof(), and NDofs().

4.33.2.2 estimate_and_initialize()

```
void InhomogenousTransformationRectangular::estimate_and_initialize ( )  [virtual]
```

At the beginning (before the first solution of a system) only the boundary conditions for the shape of the waveguide are known.

Therefore the values for the degrees of freedom need to be estimated. This function sets all variables to appropiate values and estimates an appropriate shape based on averages and a polynomial interpolation of the boundary conditions on the shape.

Implements SpaceTransformation.

Definition at line 124 of file InhomogenousTransformationRectangular.cpp.

```
124                                                                    {
125   if (GlobalParams.Use_Predefined_Shape) {
126     Sector<2> the_first(true, false, GlobalParams.sd.z[0], GlobalParams.sd.z[1]);
127     the_first.set_properties_force(GlobalParams.sd.m[0], GlobalParams.sd.m[1],
128                                    GlobalParams.sd.v[0], GlobalParams.sd.v[1]);
129     case_sectors.push_back(the_first);
130     for (int i = 1; i < GlobalParams.sd.Sectors - 2; i++) {
131       Sector<2> intermediate(false, false, GlobalParams.sd.z[i], GlobalParams.sd.z[i + 1]);
132       intermediate.set_properties_force(
```

```
133           GlobalParams.sd.m[i], GlobalParams.sd.m[i + 1], GlobalParams.sd.v[i],
134           GlobalParams.sd.v[i + 1]);
135       case_sectors.push_back(intermediate);
136     }
137     Sector<2> the_last(false, true,
138                        GlobalParams.sd.z[GlobalParams.sd.Sectors - 2],
139                        GlobalParams.sd.z[GlobalParams.sd.Sectors - 1]);
140     the_last.set_properties_force(
141         GlobalParams.sd.m[GlobalParams.sd.Sectors - 2],
142         GlobalParams.sd.m[GlobalParams.sd.Sectors - 1],
143         GlobalParams.sd.v[GlobalParams.sd.Sectors - 2],
144         GlobalParams.sd.v[GlobalParams.sd.Sectors - 1]);
145     case_sectors.push_back(the_last);
146     for (unsigned int i = 0; i < case_sectors.size(); i++) {
147       deallog « "From z: " « case_sectors[i].z_0
148               « "(m: " « case_sectors[i].get_m(0.0)
149               « " v: " « case_sectors[i].get_v(0.0) « ")" « std::endl;
150       deallog « "  To z: " « case_sectors[i].z_1
151               « "(m: " « case_sectors[i].get_m(1.0)
152               « " v: " « case_sectors[i].get_v(1.0) « ")" « std::endl;
153     }
154   } else {
155     case_sectors.reserve(sectors);
156     double m_0 = GlobalParams.Vertical_displacement_of_waveguide / 2.0;
157     double m_1 = -GlobalParams.Vertical_displacement_of_waveguide / 2.0;
158     if (sectors == 1) {
159       Sector<2> temp12(true, true, -GlobalParams.Geometry_Size_Z / 2.0,
160                        GlobalParams.Geometry_Size_Z / 2.0);
161       case_sectors.push_back(temp12);
162       case_sectors[0].set_properties_force(
163           GlobalParams.Vertical_displacement_of_waveguide / 2.0,
    -GlobalParams.Vertical_displacement_of_waveguide / 2.0,
164           GlobalParams.Width_of_waveguide, GlobalParams.Width_of_waveguide, 0, 0);
165     } else {
166       double length = Sector_Length();
167       Sector<2> temp(true, false, -GlobalParams.Geometry_Size_Z / (2.0),
168                      -GlobalParams.Geometry_Size_Z / 2.0 + length);
169       case_sectors.push_back(temp);
170       for (int i = 1; i < sectors; i++) {
171         Sector<2> temp2(false, false,
172                         -GlobalParams.Geometry_Size_Z / (2.0) + length * (1.0 * i),
173                         -GlobalParams.Geometry_Size_Z / (2.0) + length * (i + 1.0));
174         case_sectors.push_back(temp2);
175       }
176
177       double length_rel = 1.0 / ((double)(sectors));
178       case_sectors[0].set_properties_force(
179           m_0, InterpolationPolynomialZeroDerivative(length_rel, m_0, m_1), 0,
180           InterpolationPolynomialDerivative(length_rel, m_0, m_1, 0, 0));
181       for (int i = 1; i < sectors; i++) {
182         double z_l = i * length_rel;
183         double z_r = (i + 1) * length_rel;
184         case_sectors[i].set_properties_force(
185             InterpolationPolynomialZeroDerivative(z_l, m_0, m_1),
186             InterpolationPolynomialZeroDerivative(z_r, m_0, m_1),
187             InterpolationPolynomialDerivative(z_l, m_0, m_1, 0, 0),
188             InterpolationPolynomialDerivative(z_r, m_0, m_1, 0, 0));
189       }
190     }
191   }
192 }
```

References case_sectors, Sector_Length(), sectors, and Sector< Dofs_Per_Sector >::set_properties_force().

4.33.2.3 get_dof()

```
double InhomogenousTransformationRectangular::get_dof (
            int dof ) const  [virtual]
```

This is a getter for the values of degrees of freedom.

A getter-setter interface was introduced since the values are estimated automatically during the optimization and non-physical systems should be excluded from the domain of possible cases.

Parameters

dof	The index of the degree of freedom to be retrieved from the structure of the modelled waveguide.

Returns

This function returns the value of the requested degree of freedom. Should this dof not exist, 0 will be returnd.

Implements SpaceTransformation.

Definition at line 49 of file InhomogenousTransformationRectangular.cpp.

```
49                                                                          {
50   if (dof < (int)NDofs() && dof >= 0) {
51     int sector = floor(dof / 2);
52     if (sector == sectors) {
53       return case_sectors[sector - 1].dofs_r[dof % 2];
54     } else {
55       return case_sectors[sector].dofs_l[dof % 2];
56     }
57   } else {
58     std::cout « "Critical: DOF-index out of bounds in "
59                 "InhomogenousTransformationRectangular::get_dof!"
60               « std::endl;
61     return 0.0;
62   }
63 }
```

References case_sectors, NDofs(), and sectors.

Referenced by Dofs().

4.33.2.4 get_free_dof()

```
double InhomogenousTransformationRectangular::get_free_dof (
            int dof ) const  [virtual]
```

This is a getter for the values of degrees of freedom.

A getter-setter interface was introduced since the values are estimated automatically during the optimization and non-physical systems should be excluded from the domain of possible cases.

Parameters

dof	The index of the degree of freedom to be retrieved from the structure of the modelled waveguide.

Returns

This function returns the value of the requested degree of freedom. Should this dof not exist, 0 will be returnd.

Implements SpaceTransformation.

Definition at line 65 of file InhomogenousTransformationRectangular.cpp.

```
65                                                                          {
66   int dof = in_dof + 2;
67   if (dof < (int)NDofs() - 2 && dof >= 0) {
68     int sector = floor(dof / 2);
```

```
69     if (sector == sectors) {
70       return case_sectors[sector - 1].dofs_r[dof % 2];
71     } else {
72       return case_sectors[sector].dofs_l[dof % 2];
73     }
74   } else {
75     std::cout « "Critical: DOF-index out of bounds in "
76                  "InhomogenousTransformationRectangular::get_free_dof!"
77               « std::endl;
78     return 0.0;
79   }
80 }
```

References case_sectors, NDofs(), and sectors.

4.33.2.5 get_Q1()

```
double InhomogenousTransformationRectangular::get_Q1 (
            double z ) const  [virtual]
```

This member calculates the value of Q1 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q1 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implements SpaceTransformation.

Definition at line 209 of file InhomogenousTransformationRectangular.cpp.

```
209                                                                    {
210   std::pair<int, double> two = Z_to_Sector_and_local_z(z_in);
211   return case_sectors[two.first].getQ1(two.second);
212 }
```

References case_sectors, and SpaceTransformation::Z_to_Sector_and_local_z().

4.33.2.6 get_Q2()

```
double InhomogenousTransformationRectangular::get_Q2 (
            double z ) const  [virtual]
```

This member calculates the value of Q2 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q2 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implements SpaceTransformation.

Definition at line 214 of file InhomogenousTransformationRectangular.cpp.

```
214                                                                    {
215   std::pair<int, double> two = Z_to_Sector_and_local_z(z_in);
216   return case_sectors[two.first].getQ2(two.second);
217 }
```

References case_sectors, and SpaceTransformation::Z_to_Sector_and_local_z().

4.33.2.7 get_Q3()

```
double InhomogenousTransformationRectangular::get_Q3 (
            double z ) const  [virtual]
```

This member calculates the value of Q3 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q3 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implements SpaceTransformation.

Definition at line 219 of file InhomogenousTransformationRectangular.cpp.

```
219                                                                    {
220   std::pair<int, double> two = Z_to_Sector_and_local_z(z_in);
221   return case_sectors[two.first].getQ3(two.second);
222 }
```

References case_sectors, and SpaceTransformation::Z_to_Sector_and_local_z().

4.33.2.8 IsDofFree()

```
bool InhomogenousTransformationRectangular::IsDofFree (
            int index ) const  [virtual]
```

Since Dofs() also returns restrained degrees of freedom, this function can be applied to determine if a degree of freedom is indeed free or restrained.

"restrained" means that for example the DOF represents the radius at one of the connectors (input or output) and therefore we forbid the optimization scheme to vary this value.

Implements SpaceTransformation.

Definition at line 238 of file InhomogenousTransformationRectangular.cpp.

```
238                                                                {
239   return index > 1 && index < (int)NDofs() - 1;
240 }
```

References NDofs().

4.33.2.9 NDofs()

```
unsigned int InhomogenousTransformationRectangular::NDofs ( ) const  [virtual]
```

This function returns the total number of DOFs including restrained ones.

This is the lenght of the array returned by Dofs().

Implements SpaceTransformation.

Definition at line 246 of file InhomogenousTransformationRectangular.cpp.

```
246                                                                    {
247   return sectors * 2 + 2;
248 }
```

References sectors.

Referenced by Dofs(), get_dof(), get_free_dof(), IsDofFree(), NFreeDofs(), set_dof(), and set_free_dof().

4.33.2.10 set_dof()

```
void InhomogenousTransformationRectangular::set_dof (
            int dof,
            double value )  [virtual]
```

This function sets the value of the dof provided to the given value.

It is important to consider, that some dofs are non-writable (i.e. the values of the degrees of freedom on the boundary, like the radius of the input-connector cannot be changed).

Parameters

dof	The index of the parameter to be changed.
value	The value, the dof should be set to.

Implements SpaceTransformation.

Definition at line 82 of file InhomogenousTransformationRectangular.cpp.

```
82                                                                                {
83   if (dof < (int)NDofs() && dof >= 0) {
84     int sector = floor(dof / 2);
85     if (sector == sectors) {
86       case_sectors[sector - 1].dofs_r[dof % 2] = in_val;
87     } else if (sector == 0) {
88       case_sectors[0].dofs_l[dof % 2] = in_val;
89     } else {
90       case_sectors[sector].dofs_l[dof % 2] = in_val;
91       case_sectors[sector - 1].dofs_r[dof % 2] = in_val;
92     }
93   } else {
94     std::cout « "Critical: DOF-index out of bounds in "
95                  "InhomogenousTransformationRectangular::set_dof!"
96               « std::endl;
97   }
98 }
```

References case_sectors, NDofs(), and sectors.

4.33.2.11 set_free_dof()

```
void InhomogenousTransformationRectangular::set_free_dof (
            int dof,
            double value )  [virtual]
```

This function sets the value of the dof provided to the given value.

It is important to consider, that some dofs are non-writable (i.e. the values of the degrees of freedom on the boundary, like the radius of the input-connector cannot be changed).

Parameters

dof	The index of the parameter to be changed.
value	The value, the dof should be set to.

Implements SpaceTransformation.

Definition at line 100 of file InhomogenousTransformationRectangular.cpp.

```
101                                                                          {
102   int dof = in_dof + 2;
103   if (dof < (int)NDofs() - 2 && dof >= 0) {
104     int sector = floor(dof / 2);
105     if (sector == sectors) {
106       case_sectors[sector - 1].dofs_r[dof % 2] = in_val;
107     } else if (sector == 0) {
108       case_sectors[0].dofs_l[dof % 2] = in_val;
109     } else {
110       case_sectors[sector].dofs_l[dof % 2] = in_val;
111       case_sectors[sector - 1].dofs_r[dof % 2] = in_val;
112     }
113   } else {
114     std::cout « "Critical: DOF-index out of bounds in "
115                  "InhomogenousTransformationRectangular::set_free_dof!"
116               « std::endl;
117   }
118 }
```

References case_sectors, NDofs(), and sectors.

4.33.3 Member Data Documentation

4.33.3.1 case_sectors

```
std::vector<Sector<2> > InhomogenousTransformationRectangular::case_sectors
```

This member contains all the Sectors who, as a sum, form the complete Waveguide.

These Sectors are a partition of the simulated domain.

Definition at line 47 of file InhomogenousTransformationRectangular.h.

Referenced by estimate_and_initialize(), get_dof(), get_free_dof(), get_m(), get_Q1(), get_Q2(), get_Q3(), get_v(), set_dof(), and set_free_dof().

4.33.3.2 epsilon_K

```
const double InhomogenousTransformationRectangular::epsilon_K
```

The material-property ϵ_r has a different value inside and outside of the waveguides core.

This variable stores its value inside the core.

Definition at line 54 of file InhomogenousTransformationRectangular.h.

4.33.3.3 epsilon_M

```
const double InhomogenousTransformationRectangular::epsilon_M
```

The material-property ϵ_r has a different value inside and outside of the waveguides core.

This variable stores its value outside the core.

Definition at line 60 of file InhomogenousTransformationRectangular.h.

4.33.3.4 sectors

```
const int InhomogenousTransformationRectangular::sectors
```

Since the computational domain is split into subdomains (called sectors), it is important to keep track of the amount of subdomains.

This member stores the number of Sectors the computational domain has been split into.

Definition at line 66 of file InhomogenousTransformationRectangular.h.

Referenced by estimate_and_initialize(), get_dof(), get_free_dof(), NDofs(), set_dof(), and set_free_dof().

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

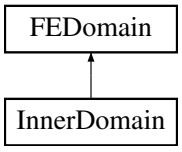
- Code/SpaceTransformations/InhomogenousTransformationRectangular.h
- Code/SpaceTransformations/InhomogenousTransformationRectangular.cpp

4.34 InnerDomain Class Reference

This class encapsulates all important mechanism for solving a FEM problem. In earlier versions this also included space transformation and computation of materials. Now it only includes FEM essentials and solving the system matrix.

```
#include <InnerDomain.h>
```

Inheritance diagram for InnerDomain:



Public Member Functions

- **InnerDomain** (unsigned int level)
- double **evaluate_for_z** (double)
- void **evaluate** ()
- void **store** ()
- void **make_grid** ()
- void **setup_system** ()
- void **assemble_system** (Constraints *constraints, dealii::PETScWrappers::MPI::SparseMatrix *matrix, NumericVectorDistributed *rhs)
- void **assemble_system** (Constraints *constraints, dealii::SparseMatrix< ComplexNumber > *matrix)
- void **assemble_system** (Constraints *constraints, dealii::PETScWrappers::SparseMatrix *matrix, NumericVectorDistributed *rhs)
- void **Compute_Dof_Numbers** ()
- void **solution_evaluation** (Position position, double *solution) const
- void **adjoint_solution_evaluation** (Position position, double *solution) const
- std::vector< InterfaceDofData > **get_surface_dof_vector_for_boundary_id** (BoundaryId b_id)
- void **fill_sparsity_pattern** (dealii::DynamicSparsityPattern *in_pattern, Constraints *constraints)
- void **write_matrix_and_rhs_metrics** (dealii::PETScWrappers::MatrixBase *matrix, NumericVector↩ Distributed *rhs)
- std::string **output_results** (std::string in_filename, NumericVectorLocal in_solution)
- void **fill_rhs_vector** (NumericVectorDistributed in_vec, unsigned int level)
- DofCount **compute_n_locally_owned_dofs** () override
- DofCount **compute_n_locally_active_dofs** () override
- void **determine_non_owned_dofs** () override

Public Attributes

- SquareMeshGenerator **mesh_generator**
- dealii::FE_NedelecSZ< 3 > **fe**
- dealii::Triangulation< 3 > **triangulation**
- DofHandler3D **dof_handler**
- dealii::SparsityPattern **sp**
- dealii::DataOut< 3 > **data_out**
- unsigned int **level**

4.34.1 Detailed Description

This class encapsulates all important mechanism for solving a FEM problem. In earlier versions this also included space transformation and computation of materials. Now it only includes FEM essentials and solving the system matrix.

Upon initialization it requires structural information about the waveguide that will be simulated. The object then continues to initialize the FEM-framework. After allocating space for all objects, the assembly-process of the system-matrix begins. Following this step, the user-selected preconditioner and solver are used to solve the system and generate outputs. This class is the core piece of the implementation.

Author

Pascal Kraft

Date

03.07.2016

Definition at line 80 of file InnerDomain.h.

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

- Code/Core/InnerDomain.h
- Code/Core/InnerDomain.cpp

4.35 InterfaceDofData Struct Reference

Public Member Functions

- **InterfaceDofData** (const DofNumber &in_index, const Position &in_position)

Public Attributes

- DofNumber **index**
- Position **base_point**
- unsigned int **order**

4.35.1 Detailed Description

Definition at line 133 of file Types.h.

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

- Code/Core/Types.h

4.36 JacobianAndTensorData Struct Reference

Public Attributes

- dealii::Tensor< 2, 3, double > **C**
- dealii::Tensor< 2, 3, double > **G**
- dealii::Tensor< 2, 3, double > **J**

4.36.1 Detailed Description

Definition at line 158 of file Types.h.

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

- Code/Core/Types.h

4.37 JacobianForCell Class Reference

Public Member Functions

- **JacobianForCell** (FaceAngelingData &in_fad, const BoundaryId &b_id, double additional_component)
- void **reinit_for_cell** (CellIterator2D)
- void **reinit** (FaceAngelingData &in_fad, const BoundaryId &b_id, double additional_component)
- auto **get_C_G_and_J** (Position2D) -> JacobianAndTensorData
- std::pair< Position2D, double > **split_into_triangulation_and_external_part** (const Position in_point)
- dealii::Tensor< 2, 3, double > **get_J_hat_for_position** (const Position2D &) const
- auto **transform_to_3D_space** (Position2D) -> Position

Static Public Member Functions

- static bool **is_line_in_x_direction** (dealii::internal::DoFHandlerImplementation::Iterators< 2, 2, false >↩ ::line_iterator line)
- static bool **is_line_in_y_direction** (dealii::internal::DoFHandlerImplementation::Iterators< 2, 2, false >↩ ::line_iterator line)

Public Attributes

- dealii::Differentiation::SD::types::substitution_map **surface_wide_substitution_map**
- BoundaryId **boundary_id**
- double **additional_component**
- std::vector< bool > **b_ids_have_hsie**
- MathExpression **x**
- MathExpression **y**
- MathExpression **z**
- MathExpression **z0**
- dealii::Tensor< 1, 3, MathExpression > **F**
- dealii::Tensor< 2, 3, MathExpression > **J**

4.37.1 Detailed Description

Definition at line 8 of file JacobianForCell.h.

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

- Code/BoundaryCondition/JacobianForCell.h
- Code/BoundaryCondition/JacobianForCell.cpp

4.38 LaguerreFunction Class Reference

Static Public Member Functions

- static double **evaluate** (unsigned int n, unsigned int m, double x)
- static double **factorial** (unsigned int n)
- static unsigned int **binomial_coefficient** (unsigned int n, unsigned int k)

4.38.1 Detailed Description

Definition at line 8 of file LaguerreFunction.h.

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

- Code/BoundaryCondition/LaguerreFunction.h
- Code/BoundaryCondition/LaguerreFunction.cpp

4.39 LevelDofIndexData Class Reference

4.39.1 Detailed Description

Definition at line 2 of file LevelDofIndexData.h.

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

- Code/Hierarchy/LevelDofIndexData.h
- Code/Hierarchy/LevelDofIndexData.cpp

4.40 LevelDofOwnershipData Struct Reference

Public Member Functions

- **LevelDofOwnershipData** (unsigned int in_global)

Public Attributes

- unsigned int **global_dofs**
- unsigned int **owned_dofs**
- dealii::IndexSet **locally_owned_dofs**
- dealii::IndexSet **input_dofs**
- dealii::IndexSet **output_dofs**
- dealii::IndexSet **locally_relevant_dofs**

4.40.1 Detailed Description

Definition at line 170 of file Types.h.

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

- Code/Core/Types.h

4.41 LevelGeometry Struct Reference

Public Attributes

- std::array< SurfaceType, 6 > **surface_type**
- CubeSurfaceTruncationState **is_surface_truncated**
- std::array< std::shared_ptr< BoundaryCondition >, 6 > **surfaces**
- std::vector< dealii::IndexSet > **dof_distribution**
- DofNumber **n_local_dofs**
- DofNumber **n_total_level_dofs**
- InnerDomain * **inner_domain**

4.41.1 Detailed Description

Definition at line 25 of file GeometryManager.h.

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

- Code/GlobalObjects/GeometryManager.h

4.42 LocalMatrixPart Struct Reference

Public Attributes

- dealii::AffineConstraints< ComplexNumber > **constraints**
- dealii::SparsityPattern **sp**
- dealii::SparseMatrix< ComplexNumber > **matrix**
- unsigned int **n_dofs**
- dealii::IndexSet **lower_sweeping_dofs**
- dealii::IndexSet **upper_sweeping_dofs**
- dealii::IndexSet **local_dofs**

4.42.1 Detailed Description

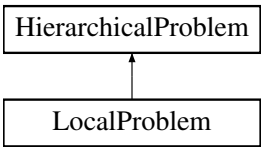
Definition at line 54 of file Types.h.

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

- Code/Core/Types.h

4.43 LocalProblem Class Reference

Inheritance diagram for LocalProblem:



Public Member Functions

- void **solve** () override
- void **initialize** () override
- void **assemble** () override
- void **initialize_index_sets** () override
- void **validate** ()
- auto **reinit** () -> void override
- auto **reinit_rhs** () -> void override
- auto **compare_to_exact_solution** () -> void
- dealii::IndexSet **compute_interface_dof_set** (BoundaryId interface_id)
- void **compute_solver_factorization** () override
- double **compute_L2_error** ()
- double **compute_error** (dealii::VectorTools::NormType, Function< 3, ComplexNumber > *, dealii::Vector< ComplexNumber > &, dealii::DataOut< 3 > *)

Public Attributes

- SolverControl **sc**
- dealii::PETScWrappers::SparseDirectMUMPS **solver**

4.43.1 Detailed Description

Definition at line 13 of file LocalProblem.h.

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

- Code/Hierarchy/LocalProblem.h
- Code/Hierarchy/LocalProblem.cpp

4.44 ModeManager Class Reference

Public Member Functions

- void **prepare_mode_in** ()
- void **prepare_mode_out** ()
- int **number_modes_in** ()
- int **number_modes_out** ()
- double **get_input_component** (int, Position, int)
- double **get_output_component** (int, Position, int)
- void **load** ()

4.44.1 Detailed Description

Definition at line 5 of file ModeManager.h.

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

- Code/GlobalObjects/ModeManager.h
- Code/GlobalObjects/ModeManager.cpp

4.45 MPICommunicator Class Reference

Public Member Functions

- std::pair< bool, unsigned int > **get_neighbor_for_interface** (Direction in_direction)
- void **initialize** ()

Public Attributes

- std::vector< MPI_Comm > **communicators_by_level**
- std::vector< unsigned int > **rank_on_level**

4.45.1 Detailed Description

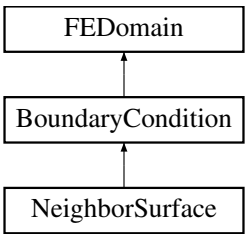
Definition at line 7 of file MPICommunicator.h.

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

- Code/Hierarchy/MPICommunicator.h
- Code/Hierarchy/MPICommunicator.cpp

4.46 NeighborSurface Class Reference

Inheritance diagram for NeighborSurface:



Public Member Functions

- **NeighborSurface** (unsigned int in_bid, unsigned int in_level)
- void **prepare_id_sets_for_boundaries** ()
- bool **is_point_at_boundary** (Position, BoundaryId)
- void **fill_matrix** (dealii::PETScWrappers::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_matrix** (dealii::SparseMatrix< ComplexNumber > *, Constraints *constraints) override
- void **fill_matrix** (dealii::PETScWrappers::SparseMatrix *, dealii::PETScWrappers::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_matrix** (dealii::PETScWrappers::MPI::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_sparsity_pattern** (dealii::DynamicSparsityPattern *in_dsp, Constraints *in_constriants) override
- bool **is_point_at_boundary** (Position2D in_p, BoundaryId in_bid) override
- bool **is_position_at_boundary** (Position in_p, BoundaryId in_bid)
- void **initialize** () override
- void **set_mesh_boundary_ids** ()
- auto **cells_for_boundary_id** (unsigned int boundary_id) -> unsigned int
- void **init_fe** ()
- auto **get_dof_association** () -> std::vector< InterfaceDofData > override
- auto **get_dof_association_by_boundary_id** (BoundaryId in_boundary_id) -> std::vector< InterfaceDofData > override
- void **sort_dofs** ()
- void **compute_coordinate_ranges** ()
- void **set_boundary_ids** ()
- std::string **output_results** (const dealii::Vector< ComplexNumber > &, std::string) override
- DofCount **compute_n_locally_owned_dofs** () override
- DofCount **compute_n_locally_active_dofs** () override
- void **send_up_inner_dofs** () override
- void **receive_from_below_dofs** () override
- void **determine_non_owned_dofs** () override
- void **send_up_boundary_dofs** (unsigned int other_bid)
- std::vector< DofNumber > **receive_boundary_dofs** (unsigned int other_bid) override
- void **finish_dof_index_initialization** () override
- void **distribute_dof_indices** ()

Public Attributes

- const bool **is_lower_interface**
- std::array< std::set< unsigned int >, 6 > **edge_ids_by_boundary_id**
- std::array< std::set< unsigned int >, 6 > **face_ids_by_boundary_id**
- std::array< std::vector< InterfaceDofData >, 6 > **dof_indices_by_boundary_id**
- std::array< std::vector< unsigned int >, 6 > **boundary_dofs**
- std::vector< unsigned int > **inner_dofs**

4.46.1 Detailed Description

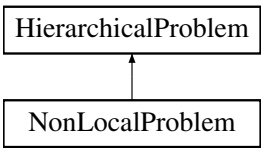
Definition at line 8 of file NeighborSurface.h.

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

- Code/BoundaryCondition/NeighborSurface.h
- Code/BoundaryCondition/NeighborSurface.cpp

4.47 NonLocalProblem Class Reference

Inheritance diagram for NonLocalProblem:



Public Member Functions

- **NonLocalProblem** (unsigned int)
- void **prepare_sweeping_data** ()
- void **assemble** () override
- void **solve** () override
- void **apply_sweep** (Vec x_in, Vec x_out)
- void **init_solver_and_preconditioner** ()
- void **initialize** () override
- void **initialize_index_sets** () override
- void **reinit** () override
- void **compute_solver_factorization** () override
- void **reinit_rhs** () override
- void **reinit_vector** (NumericVectorDistributed *)
- void **S_inv** (NumericVectorDistributed *src, NumericVectorDistributed *dst)
- auto **set_x_out_from_u** (Vec x_out, NumericVectorDistributed *u_in) -> void
- auto **set_u_from_child_solution** (NumericVectorDistributed *u) -> void
- std::string **output_results** ()
- void **write_multifile_output** (const std::string &filename, const NumericVectorDistributed field)
- void **communicate_external_dsp** (DynamicSparsityPattern *in_dsp)
- void **make_sparsity_pattern** () override
- NumericVectorDistributed **vector_from_vec_obj** (Vec in_v)
- void **set_vector_from_child_solution** (NumericVectorDistributed *)
- void **set_child_rhs_from_vector** (NumericVectorDistributed *)
- void **print_vector_norm** (NumericVectorDistributed *, std::string marker)
- void **perform_downward_sweep** (NumericVectorDistributed *)
- void **perform_upward_sweep** (NumericVectorDistributed *)
- void **complex_pml_domain_matching** (BoundaryId in_bid)
- void **register_dof_copy_pair** (DofNumber own_index, DofNumber child_index)

Additional Inherited Members

4.47.1 Detailed Description

Definition at line 14 of file NonLocalProblem.h.

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

- Code/Hierarchy/NonLocalProblem.h
- Code/Hierarchy/NonLocalProblem.cpp

4.48 OutputManager Class Reference

Public Member Functions

- void **initialize** ()
- std::string **get_full_filename** (std::string filename)
- std::string **get_numbered_filename** (std::string filename, unsigned int number, std::string extension)
- void **write_log_ling** (std::string in_line)
- void **write_run_description** ()

Public Attributes

- std::string **base_path**
- unsigned int **run_number**
- std::string **output_folder_path**
- std::ofstream **log_stream**

4.48.1 Detailed Description

Definition at line 8 of file OutputManager.h.

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

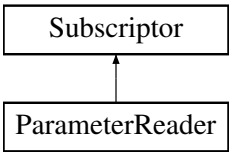
- Code/GlobalObjects/OutputManager.h
- Code/GlobalObjects/OutputManager.cpp

4.49 ParameterReader Class Reference

This class is used to gather all the information from the input file and store it in a static object available to all processes.

`#include <ParameterReader.h>`

Inheritance diagram for ParameterReader:



Public Member Functions

- ParameterReader ()

  Deal Offers the ParameterHandler object wich contains all of the parsing-functionality.
- Parameters read_parameters (const std::string run_file, const std::string case_file)

  This member calls the read_input_from_xml()-function of the contained ParameterHandler and this replaces the default values with the values in the input file.
- void declare_parameters ()

  In this function, we add all values descriptions to the parameter-handler.

4.49.1 Detailed Description

This class is used to gather all the information from the input file and store it in a static object available to all processes.

The ParameterReader is a very useful tool. It uses a deal-function to read a xml-file and parse the contents to specific variables. These variables have default values used in their declaration. The members of this class do two things:

1. declare the variables. This includes setting a data-type for them and a default value should none be provided in the input file. Furthermore there can be restrictions like maximum or minimum values etc.
2. call an external function to parse an input-file.

After creating an object of this type and calling both declare() and read(), this object contains all the information from the input file and can be used in the code without dealing with persistence.

Author

Pascal Kraft

Date

23.11.2015

Definition at line 29 of file ParameterReader.h.

4.49.2 Constructor & Destructor Documentation

4.49.2.1 ParameterReader()

```
ParameterReader::ParameterReader ( )
```

Deal Offers the ParameterHandler object wich contains all of the parsing-functionality.

An object of that type is included in this one. This constructor simply uses a copy-constructor to initialize it.

Definition at line 6 of file ParameterReader.cpp.

```
6 { }
```

4.49.3 Member Function Documentation

4.49.3.1 declare_parameters()

```
void ParameterReader::declare_parameters ( )
```

In this function, we add all values descriptions to the parameter-handler.

This includes

1. a default value,
2. a data-type,
3. possible restrictions (greater than zero etc.),
4. a description, which is displayed in deals ParameterGUI-tool,
5. a hierarchical structure to order the variables.

Deals Parameter-GUI can be installed at build-time of the library and offers a great and easy way to edit the input file. It displays appropriate input-methods depending on the type, so, for example, in case of a selection from three different values (i.e. the name of a solver that has to either be GMRES, MINRES or UMFPACK) it displays a dropdown containing all the options.

Definition at line 8 of file ParameterReader.cpp.

```
8                                       {
9     run_prm.enter_subsection("Run parameters");
10     {
11         run_prm.declare_entry("perform optimization", "false", Patterns::Bool(), "If true, the code will
       perform shape optimization.");
12         run_prm.declare_entry("solver precision" , "1e-6", Patterns::Double(), "Absolute precision for
       solver convergence.");
13         run_prm.declare_entry("GMRES restart after" , "30", Patterns::Integer(), "Number of steps until
       GMRES restarts.");
14         run_prm.declare_entry("GMRES maximum steps" , "30", Patterns::Integer(), "Number of maximum GMRES
       steps until failure.");
15         run_prm.declare_entry("solve directly", "false", Patterns::Bool(), "If this is set to true, GMRES
       will be replaced by a direct solver.");
16         run_prm.declare_entry("kappa angle" , "1.0", Patterns::Double(), "Phase of the complex value
       kappa with norm 1 that is used in HSIEs.");
17         run_prm.declare_entry("processes in x" , "1", Patterns::Integer(), "Number of processes in
       x-direction.");
18         run_prm.declare_entry("processes in y" , "1", Patterns::Integer(), "Number of processes in
       y-direction.");
19         run_prm.declare_entry("processes in z" , "1", Patterns::Integer(), "Number of processes in
       z-direction.");
20         run_prm.declare_entry("sweeping level" , "1", Patterns::Integer(), "Hierarchy level to be used.
       1: normal sweeping. 2: two level hierarchy, i.e sweeping in sweeping. 3: three level sweeping, i.e.
       sweeping in sweeping in swepping.");
21         run_prm.declare_entry("cell count x" , "20", Patterns::Integer(), "Number of cells a single
       process has in x-direction.");
22         run_prm.declare_entry("cell count y" , "20", Patterns::Integer(), "Number of cells a single
       process has in y-direction.");
23         run_prm.declare_entry("cell count z" , "20", Patterns::Integer(), "Number of cells a single
       process has in z-direction.");
24         run_prm.declare_entry("Logging Level", "Production One", Patterns::Selection("Production
       One|Production All|Debug One|Debug All"), "Specifies which messages should be printed and by whom.");
25     }
26     run_prm.leave_subsection();
27
28     case_prm.enter_subsection("Case parameters");
29     {
30         case_prm.declare_entry("source type", "0", Patterns::Integer(), "PointSourceField is 0: empty, 1:
       cos()cos(), 2: Hertz Dipole, 3: Waveguide");
31         case_prm.declare_entry("geometry size x", "5.0", Patterns::Double(), "Size of the computational
       domain in x-direction.");
32         case_prm.declare_entry("geometry size y", "5.0", Patterns::Double(), "Size of the computational
       domain in y-direction.");
```

```
33          case_prm.declare_entry("geometry size z", "5.0", Patterns::Double(), "Size of the computational
        domain in z-direction.");
34          case_prm.declare_entry("epsilon in", "2.3409", Patterns::Double(), "Epsilon r inside the
        material.");
35          case_prm.declare_entry("epsilon out", "1.8496", Patterns::Double(), "Epsilon r outside the
        material.");
36          case_prm.declare_entry("epsilon effective", "2.1588449", Patterns::Double(), "Epsilon r outside
        the material.");
37          case_prm.declare_entry("mu in", "1.0", Patterns::Double(), "Mu r inside the material.");
38          case_prm.declare_entry("mu out", "1.0", Patterns::Double(), "Mu r outside the material.");
39          case_prm.declare_entry("fem order" , "0", Patterns::Integer(), "Degree of nedelec elements in the
        interior.");
40          case_prm.declare_entry("signal amplitude", "1.0", Patterns::Double(), "Amplitude of the input
        signal or PointSourceField");
41          case_prm.declare_entry("width of waveguide", "2.0", Patterns::Double(), "Width of the Waveguide
        core.");
42          case_prm.declare_entry("height of waveguide", "1.8", Patterns::Double(), "Height of the Waveguide
        core.");
43          case_prm.declare_entry("Enable Parameter Run", "false", Patterns::Bool(), "For a series of Local
        solves, this can be set to true");
44          case_prm.declare_entry("Kappa 0 Real", "1", Patterns::Double(), "Real part of kappa_0 for
        HSIE.");
45          case_prm.declare_entry("Kappa 0 Imaginary", "1", Patterns::Double(), "Imaginary part of kappa_0
        for HSIE.");
46          case_prm.declare_entry("PML sigma max", "10.0", Patterns::Double(), "Parameter Sigma Max for all
        PML layers.");
47          case_prm.declare_entry("HSIE polynomial degree" , "4", Patterns::Integer(), "Polynomial degree of
        the Hardy-space polynomials for HSIE surfaces.");
48          case_prm.declare_entry("Min HSIE Order", "1", Patterns::Integer(), "Minimal HSIE Element order
        for parameter run.");
49          case_prm.declare_entry("Max HSIE Order", "21", Patterns::Integer(), "Maximal HSIE Element order
        for parameter run.");
50          case_prm.declare_entry("Boundary Method", "HSIE", Patterns::Selection("HSIE|PML"), "Choose the
        boundary element method (options are PML and HSIE).");
51          case_prm.declare_entry("PML thickness", "1.0", Patterns::Double(), "Thickness of PML layers.");
52          case_prm.declare_entry("PML skaling order", "3", Patterns::Integer(), "PML skaling order is the
        exponent with wich the imaginary part grows towards the outer boundary.");
53          case_prm.declare_entry("PML n layers", "8", Patterns::Integer(), "Number of cell layers used in
        the PML medium.");
54          case_prm.declare_entry("Input Signal Method", "Dirichlet",
        Patterns::Selection("Dirichlet|Taper|Jump"), "Taper uses a tapered exact solution to build a right
        hand side. Dirichlet applies dirichlet boundary values.");
55          case_prm.declare_entry("Signal tapering type", "C1", Patterns::Selection("C0|C1"), "Tapering type
        for signal input");
56          case_prm.declare_entry("Prescribe input zero", "false", Patterns::Bool(), "If this is set to
        true, there will be a dirichlet zero condition enforced on the global input interface (Process index
        z: 0, boundary id: 4).");
57          case_prm.declare_entry("Predefined case number", "1", Patterns::Integer(), "Number in [1,35] that
        describes the predefined shape to use.");
58          case_prm.declare_entry("Use predefined shape", "false", Patterns::Bool(), "If set to true, the
        geometry for the predefined case from 'Predefined case number' will be used.");
59      }
60      case_prm.leave_subsection();
61  }
```

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

- Code/Helpers/ParameterReader.h
- Code/Helpers/ParameterReader.cpp

4.50 Parameters Class Reference

This structure contains all information contained in the input file and some values that can simply be computed from it.

`#include <Parameters.h>`

Public Member Functions

- auto **complete_data** () -> void
- auto **check_validity** () -> bool

Public Attributes

- ShapeDescription **sd**
- double **Solver_Precision** = 1e-6
- unsigned int **GMRES_Steps_before_restart** = 30
- unsigned int **GMRES_max_steps** = 100
- unsigned int **MPI_Rank**
- unsigned int **NumberProcesses**
- double **Amplitude_of_input_signal** = 1.0
- double **Width_of_waveguide** = 2.0
- double **Height_of_waveguide** = 1.8
- double **Horizontal_displacement_of_waveguide** = 0
- double **Vertical_displacement_of_waveguide** = 0
- double **Epsilon_R_in_waveguide** = 2.3409
- double **Epsilon_R_outside_waveguide** = 1.8496
- double **Epsilon_R_effective** = 2.1588449
- double **Mu_R_in_waveguide** = 1.0
- double **Mu_R_outside_waveguide** = 1.0
- unsigned int **HSIE_polynomial_degree** = 5
- bool **Perform_Optimization** = false
- double **kappa_0_angle** = 1.0
- ComplexNumber **kappa_0**
- unsigned int **Nedelec_element_order** = 0
- unsigned int **Blocks_in_z_direction** = 1
- unsigned int **Blocks_in_x_direction** = 1
- unsigned int **Blocks_in_y_direction** = 1
- unsigned int **Index_in_x_direction**
- unsigned int **Index_in_y_direction**
- unsigned int **Index_in_z_direction**
- unsigned int **Cells_in_x** = 20
- unsigned int **Cells_in_y** = 20
- unsigned int **Cells_in_z** = 20
- int **current_run_number** = 0
- double **Geometry_Size_X** = 5
- double **Geometry_Size_Y** = 5
- double **Geometry_Size_Z** = 5
- unsigned int **Number_of_sectors**
- double **Sector_thickness**
- double **Sector_padding**
- double **Pi** = 3.141592653589793238462
- double **Omega** = 1.0
- double **Lambda** = 1.55
- double **Waveguide_value_V** = 1.0
- bool **Use_Predefined_Shape** = false
- unsigned int **Number_of_Predefined_Shape** = 1
- unsigned int **Point_Source_Type** = 0
- unsigned int **Sweeping_Level** = 1
- LoggingLevel **Logging_Level** = LoggingLevel::DEBUG_ALL
- dealii::Function< 3, ComplexNumber > * **source_field**
- bool **Enable_Parameter_Run** = false
- unsigned int **N_Kappa_0_Steps** = 20
- unsigned int **Min_HSIE_Order** = 1
- unsigned int **Max_HSIE_Order** = 10
- double **PML_Sigma_Max** = 5.0
- unsigned int **PML_N_Layers** = 8

- double **PML_thickness** = 1.0
- unsigned int **PML_skaling_order** = 3
- BoundaryConditionType **BoundaryCondition** = BoundaryConditionType::HSIE
- bool **use_tapered_input_signal** = false
- double **tapering_min_z** = 0.0
- double **tapering_max_z** = 1.0
- SignalTaperingType **Signal_tapering_type** = SignalTaperingType::C1
- SignalCouplingMethod **Signal_coupling_method** = SignalCouplingMethod::Jump
- bool **prescribe_0_on_input_side** = false
- bool **solve_directly** = false

4.50.1 Detailed Description

This structure contains all information contained in the input file and some values that can simply be computed from it.

In the application, static Variable of this type makes the input parameters available globally.

Author

: Pascal Kraft

Date

: 28.11.2016

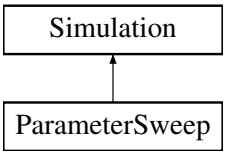
Definition at line 18 of file Parameters.h.

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

- Code/Helpers/Parameters.h
- Code/Helpers/Parameters.cpp

4.51 ParameterSweep Class Reference

Inheritance diagram for ParameterSweep:



Public Member Functions

- void **prepare** () override
- void **run** () override
- void **prepare_transformed_geometry** () override

4.51.1 Detailed Description

Definition at line 9 of file ParameterSweep.h.

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

- Code/Runners/ParameterSweep.h
- Code/Runners/ParameterSweep.cpp

4.52 PMLMeshTransformation Struct Reference

Public Member Functions

- **PMLMeshTransformation** (std::pair< double, double > in_x_range, std::pair< double, double > in_y_range, std::pair< double, double > in_z_range, double in_base_coordinate, unsigned int in_outward_direction, std::array< bool, 6 > in_transform_coordinate)
- Position **operator()** (const Position &in_p) const
- Position **undo_transform** (const Position &in_p)

Public Attributes

- std::pair< double, double > **default_x_range**
- std::pair< double, double > **default_y_range**
- std::pair< double, double > **default_z_range**
- double **base_coordinate_for_transformed_direction**
- unsigned int **outward_direction**
- std::array< bool, 6 > **transform_coordinate**

4.52.1 Detailed Description

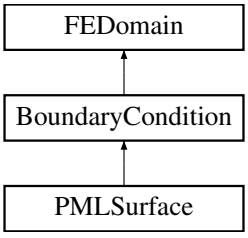
Definition at line 6 of file PMLMeshTransformation.h.

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

- Code/BoundaryCondition/PMLMeshTransformation.h
- Code/BoundaryCondition/PMLMeshTransformation.cpp

4.53 PMLSurface Class Reference

Inheritance diagram for PMLSurface:



Public Member Functions

- **PMLSurface** (unsigned int in_bid, unsigned int in_level)
- bool **is_point_at_boundary** (Position, BoundaryId)
- auto **make_constraints** () -> Constraints override
- void **fill_matrix** (dealii::PETScWrappers::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_matrix** (dealii::SparseMatrix< ComplexNumber > *, Constraints *constraints) override
- void **fill_matrix** (dealii::PETScWrappers::SparseMatrix *, dealii::PETScWrappers::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_matrix** (dealii::PETScWrappers::MPI::SparseMatrix *, NumericVectorDistributed *rhs, Constraints *constraints) override
- void **fill_sparsity_pattern** (dealii::DynamicSparsityPattern *in_dsp, Constraints *in_constriants) override
- bool **is_point_at_boundary** (Position2D in_p, BoundaryId in_bid) override
- bool **is_position_at_boundary** (const Position in_p, const BoundaryId in_bid)
- bool **is_position_at_extended_boundary** (const Position in_p, const BoundaryId in_bid)
- void **initialize** () override
- void **set_mesh_boundary_ids** ()
- void **prepare_mesh** ()
- auto **cells_for_boundary_id** (unsigned int boundary_id) -> unsigned int override
- void **init_fe** ()
- auto **fraction_of_pml_direction** (Position) -> std::array< double, 3 >
- auto **get_pml_tensor_epsilon** (Position) -> dealii::Tensor< 2, 3, ComplexNumber >
- auto **get_pml_tensor_mu** (Position) -> dealii::Tensor< 2, 3, ComplexNumber >
- auto **get_pml_tensor** (Position) -> dealii::Tensor< 2, 3, ComplexNumber >
- auto **get_dof_association** () -> std::vector< InterfaceDofData > override
- auto **get_dof_association_by_boundary_id** (BoundaryId in_boundary_id) -> std::vector< InterfaceDofData > override
- void **compute_coordinate_ranges** (dealii::Triangulation< 3 > *in_tria)
- void **set_boundary_ids** ()
- void **fix_apply_negative_Jacobian_transformation** (dealii::Triangulation< 3 > *in_tria)
- std::string **output_results** (const dealii::Vector< ComplexNumber > &, std::string) override
- void **validate_meshes** ()
- DofCount **compute_n_locally_owned_dofs** () override
- DofCount **compute_n_locally_active_dofs** () override
- void **finish_dof_index_initialization** () override
- void **determine_non_owned_dofs** () override
- dealii::IndexSet **compute_non_owned_dofs** ()
- bool **finish_initialization** (DofNumber first_own_index) override
- bool **mg_process_edge** (dealii::Triangulation< 3 > *current_list, BoundaryId b_id)
- bool **mg_process_corner** (dealii::Triangulation< 3 > *current_list, BoundaryId first_bid, BoundaryId second_bid)
- bool **extend_mesh_in_direction** (BoundaryId in_bid)

Public Attributes

- std::pair< double, double > **x_range**
- std::pair< double, double > **y_range**
- std::pair< double, double > **z_range**
- double **non_pml_layer_thickness**
- dealii::Triangulation< 3 > **triangulation**

4.53.1 Detailed Description

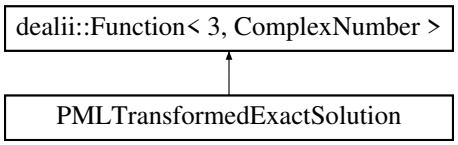
Definition at line 9 of file PMLSurface.h.

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

- Code/BoundaryCondition/PMLSurface.h
- Code/BoundaryCondition/PMLSurface.cpp

4.54 PMLTransformedExactSolution Class Reference

Inheritance diagram for PMLTransformedExactSolution:



Public Member Functions

- **PMLTransformedExactSolution** (BoundaryId in_main_id, double in_additional_coordinate)
- std::vector< std::string > **split** (std::string) const
- ComplexNumber **value** (const Position &p, const unsigned int component) const
- void **vector_value** (const Position &p, dealii::Vector< ComplexNumber > &value) const
- dealii::Tensor< 1, 3, ComplexNumber > **curl** (const Position &in_p) const
- dealii::Tensor< 1, 3, ComplexNumber > **val** (const Position &in_p) const
- std::array< double, 3 > **fraction_of_pml_direction** (const Position &in_p) const
- double **compute_scaling_factor** (const Position &in_p) const

4.54.1 Detailed Description

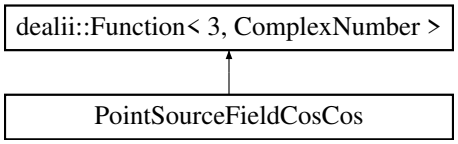
Definition at line 12 of file PMLTransformedExactSolution.h.

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

- Code/Solutions/PMLTransformedExactSolution.h
- Code/Solutions/PMLTransformedExactSolution.cpp

4.55 PointSourceFieldCosCos Class Reference

Inheritance diagram for PointSourceFieldCosCos:



Public Member Functions

- ComplexNumber **value** (const Position &p, const unsigned int component=0) const override
- void **vector_value** (const Position &p, NumericVectorLocal &vec) const override
- void **vector_curl** (const Position &p, NumericVectorLocal &vec)

4.55.1 Detailed Description

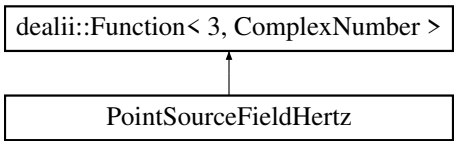
Definition at line 19 of file PointSourceField.h.

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

- Code/Helpers/PointSourceField.h
- Code/Helpers/PointSourceField.cpp

4.56 PointSourceFieldHertz Class Reference

Inheritance diagram for PointSourceFieldHertz:



Public Member Functions

- **PointSourceFieldHertz** (double in_k=1.0)
- void **set_cell_diameter** (double diameter)
- ComplexNumber **value** (const Position &p, const unsigned int component=0) const override
- void **vector_value** (const Position &p, NumericVectorLocal &vec) const override
- void **vector_curl** (const Position &p, NumericVectorLocal &vec)

Public Attributes

- double **k** = 1
- const ComplexNumber **ik**
- double **cell_diameter** = 0.01

4.56.1 Detailed Description

Definition at line 6 of file PointSourceField.h.

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

- Code/Helpers/PointSourceField.h
- Code/Helpers/PointSourceField.cpp

4.57 PointVal Class Reference

Public Member Functions

- **PointVal** (double, double, double, double, double, double)
- void **set** (double, double, double, double, double, double)
- void **rescale** (double)

Public Attributes

- ComplexNumber **Ex**
- ComplexNumber **Ey**
- ComplexNumber **Ez**

4.57.1 Detailed Description

Definition at line 4 of file PointVal.h.

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

- Code/Helpers/PointVal.h
- Code/Helpers/PointVal.cpp

4.58 RayAngelingData Struct Reference

Public Attributes

- bool **is_x_angled** = false
- bool **is_y_angled** = false
- Position2D **position_of_base_point**

4.58.1 Detailed Description

Definition at line 111 of file Types.h.

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

- Code/Core/Types.h

4.59 RectangularMode Class Reference

Public Member Functions

- void **assemble_system** ()
- void **make_mesh** ()
- void **make_boundary_conditions** ()
- void **output_solution** ()
- void **run** ()
- void solve ()
- void **SortDofsDownstream** ()
- IndexSet **get_dofs_for_boundary_id** (types::boundary_id)
- std::vector< InterfaceDofData > **get_surface_dof_vector_for_boundary_id** (unsigned int b_id)

Static Public Member Functions

- static auto **compute_epsilon_for_Position** (Position in_position) -> double

Public Attributes

- double **beta**
- unsigned int **n_dofs_total**
- unsigned int **n_eigenfunctions** = 1
- std::vector< ComplexNumber > **eigenvalues**
- std::vector< PETScWrappers::MPI::Vector > **eigenfunctions**
- std::vector< DofNumber > **surface_first_dofs**
- std::array< std::shared_ptr< HSIESurface >, 4 > **surfaces**
- dealii::FE_NedelecSZ< 3 > **fe**
- Constraints **constraints**
- Constraints **periodic_constraints**
- Triangulation< 3 > **triangulation**
- DoFHandler< 3 > **dof_handler**
- SparsityPattern **sp**
- PETScWrappers::SparseMatrix **mass_matrix**
- PETScWrappers::SparseMatrix **stiffness_matrix**
- NumericVectorDistributed **rhs**
- NumericVectorDistributed **solution**
- const double **layer_thickness**
- const double **lambda**

4.59.1 Detailed Description

Definition at line 47 of file RectangularMode.h.

4.59.2 Member Function Documentation

4.59.2.1 solve()

```
void RectangularMode::solve ( )
```

eigensolver.solve(stiffness_matrix, mass_matrix, eigenvalues, eigenfunctions, n_eigenfunctions);

Definition at line 281 of file RectangularMode.cpp.

```
281                          {
282   print_info("RectangularProblem::solve", "Start");
283   dealii::SolverControl                    solver_control(n_dofs_total, 1e-6);
284   // dealii::SLEPcWrappers::SolverKrylovSchur eigensolver(solver_control);
285   IndexSet own_dofs(n_dofs_total);
286   own_dofs.add_range(0, n_dofs_total);
287   eigenfunctions.resize(n_eigenfunctions);
288   for (unsigned int i = 0; i < n_eigenfunctions; ++i)
289     eigenfunctions[i].reinit(own_dofs, MPI_COMM_SELF);
290   eigenvalues.resize(n_eigenfunctions);
291   // eigensolver.set_which_eigenpairs(EPS_SMALLEST_MAGNITUDE);
292   // eigensolver.set_problem_type(EPS_GNHEP);
293   print_info("RectangularProblem::solve", "Starting solution for a system with " +
       std::to_string(n_dofs_total) + " degrees of freedom.");
294   /**
295   eigensolver.solve(stiffness_matrix,
296                     mass_matrix,
297                     eigenvalues,
298                     eigenfunctions,
299                     n_eigenfunctions);
300                     **/
301   for(unsigned int i =0 ; i < n_eigenfunctions; i++) {
302     // constraints.distribute(eigenfunctions[0]);
303     eigenfunctions[i] /= eigenfunctions[i].linfty_norm();
304   }
305   print_info("RectangularProblem::solve", "End");
306 }
```

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

- Code/ModalComputations/RectangularMode.h
- Code/ModalComputations/RectangularMode.cpp

4.60 ResidualOutputGenerator Class Reference

Public Member Functions

- **ResidualOutputGenerator** (std::string in_name, std::string in_title, bool in_print_output, unsigned int in_↩ level)
- void **push_value** (double value)
- void **close_current_series** ()
- void **new_series** (std::string name)
- void **write_gnuplot_file** ()
- void **run_gnuplot** ()
- void **write_residual_statement_to_console** ()

4.60.1 Detailed Description

Definition at line 5 of file ResidualOutputGenerator.h.

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

- Code/OutputGenerators/Images/ResidualOutputGenerator.h
- Code/OutputGenerators/Images/ResidualOutputGenerator.cpp

4.61 SampleShellPC Struct Reference

Public Attributes

- NonLocalProblem * **parent**

4.61.1 Detailed Description

Definition at line 63 of file HierarchicalProblem.h.

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

- Code/Hierarchy/HierarchicalProblem.h

4.62 Sector< Dofs_Per_Sector > Class Template Reference

Sectors are used, to split the computational domain into chunks, whose degrees of freedom are likely coupled.

`#include <Sector.h>`

Public Member Functions

- Sector (bool in_left, bool in_right, double in_z_0, double in_z_1)

  Constructor of the Sector class, that takes all important properties as an input property.
- dealii::Tensor< 2, 3, double > TransformationTensorInternal (double in_x, double in_y, double in_z) const

  This method gets called from the WaveguideStructure object used in the simulation.
- void set_properties (double, double, double, double)

  This function is used during the optimization-operation to update the properties of the space-transformation.
- void **set_properties** (double, double, double, double, double, double)
- void set_properties_force (double, double, double, double)

  This function is the same as set_properties with the difference of being able to change the values of the input- and output boundary.
- void **set_properties_force** (double, double, double, double, double, double)
- double getQ1 (double) const

  The values of Q1, Q2 and Q3 are needed to compute the solution in real coordinates from the one in trnsformed coordinates.
- double getQ2 (double) const

  The values of Q1, Q2 and Q3 are needed to compute the solution in real coordinates from the one in transformed coordinates.
- double getQ3 (double) const

  The values of Q1, Q2 and Q3 are needed to compute the solution in real coordinates from the one in transformed coordinates.
- unsigned int getLowestDof () const

  This function returns the number of the lowest degree of freedom associated with this Sector.
- unsigned int getNDofs () const

  This function returns the number of dofs which are part of this sector.
- unsigned int getNInternalBoundaryDofs () const

In order to set appropriate boundary conditions it makes sense to determine, which degrees are associated with an edge which is part of an interface to another sector.

- unsigned int getNActiveCells () const

  This function can be used to query the number of cells in a Sector / subdomain.
- void setLowestDof (unsigned int)

  Setter for the value that the getter should return.
- void setNDofs (unsigned int)

  Setter for the value that the getter should return.
- void setNInternalBoundaryDofs (unsigned int)

  Setter for the value that the getter should return.
- void setNActiveCells (unsigned int)

  Setter for the value that the getter should return.
- double get_dof (unsigned int i, double z) const

  This function returns the value of a specified dof at a given internal position.
- double get_r (double z) const

  Get an interpolation of the radius for a coordinate z.
- double get_v (double z) const

  Get an interpolation of the tilt for a coordinate z.
- double get_m (double z) const

  Get an interpolation of the shift for a coordinate z.

Public Attributes

- const bool left

  This value describes, if this Sector is at the left (small z) end of the computational domain.
- const bool right

  This value describes, if this Sector is at the right (large z) end of the computational domain.
- const bool boundary

  This value is true, if either left or right are true.
- const double **z_0**
- const double z_1

  The objects created from this class are supposed to hand back the material properties which include the space-transformation Tensors.
- unsigned int **LowestDof**
- unsigned int **NDofs**
- unsigned int **NInternalBoundaryDofs**
- unsigned int **NActiveCells**
- double * **dofs_l**
- double * **dofs_r**
- unsigned int * **derivative**
- bool * **zero_derivative**

4.62.1 Detailed Description

template<unsigned int Dofs_Per_Sector>
class Sector< Dofs_Per_Sector >

Sectors are used, to split the computational domain into chunks, whose degrees of freedom are likely coupled.

The interfaces between Sectors lie in the xy-plane and they are ordered by their z-value.

Author

Pascal Kraft

Date

17.12.2015

Definition at line 14 of file Sector.h.

4.62.2 Constructor & Destructor Documentation

4.62.2.1 Sector()

```
template<unsigned int Dofs_Per_Sector>
Sector< Dofs_Per_Sector >::Sector (
            bool in_left,
            bool in_right,
            double in_z_0,
            double in_z_1 )
```

Constructor of the Sector class, that takes all important properties as an input property.

Parameters

in_left	stores if the sector is at the left end. It is used to initialize the according variable.
in_right	stores if the sector is at the right end. It is used to initialize the according variable.
in_z_0	stores the z-coordinate of the left surface-plain. It is used to initialize the according variable.
in_z_1	stores the z-coordinate of the right surface-plain. It is used to initialize the according variable.

Definition at line 18 of file Sector.cpp.

```
20     : left(in_left),
21       right(in_right),
22       boundary(in_left && in_right),
23       z_0(in_z_0),
24       z_1(in_z_1) {
25   dofs_l = new double[Dofs_Per_Sector];
26   dofs_r = new double[Dofs_Per_Sector];
27   derivative = new unsigned int[Dofs_Per_Sector];
28   zero_derivative = new bool[Dofs_Per_Sector];
29   if (Dofs_Per_Sector == 3) {
30     zero_derivative[0] = true;
31     zero_derivative[1] = false;
32     zero_derivative[2] = true;
33     derivative     [0] = 0;
34     derivative     [1] = 2;
35     derivative     [2] = 0;
36   }
37   if (Dofs_Per_Sector == 2) {
38     zero_derivative[0] = false;
39     zero_derivative[1] = true;
40     derivative     [0] = 1;
41     derivative     [1] = 0;
42   }
43
44   for (unsigned int i = 0; i < Dofs_Per_Sector; i++) {
```

```
45     dofs_l[i] = 0;
46     dofs_r[i] = 0;
47   }
48   NInternalBoundaryDofs = 0;
49   LowestDof = 0;
50   NActiveCells = 0;
51   NDofs = Dofs_Per_Sector;
52 }
```

4.62.3 Member Function Documentation

4.62.3.1 get_dof()

```
template<unsigned int Dofs_Per_Sector>
double Sector< Dofs_Per_Sector >::get_dof (
            unsigned int i,
            double z ) const
```

This function returns the value of a specified dof at a given internal position.

Parameters

i	index of the dof. This class has a template argument specifying the number of dofs per sector. This argument has to be less or equal.
z	this is a relative value for interpolation with $z \in [0, 1]$. If $z = 0$ the values for the lower end of the sector are returned. If $z = 1$ the values for the upper end of the sector are returned. In between the values are interpolated according to the rules for the specific dof.

Definition at line 152 of file Sector.cpp.

```
152                                                                    {
153   if (i > 0 && i < NDofs) {
154     if (z < 0.0) z = 0.0;
155     if (z > 1.0) z = 1.0;
156     if (zero_derivative[i]) {
157       return InterpolationPolynomialZeroDerivative(z, dofs_l[i], dofs_r[i]);
158     } else {
159       return InterpolationPolynomial(z, dofs_l[i], dofs_r[i],
160                                      dofs_l[derivative[i]],
161                                      dofs_r[derivative[i]]);
162     }
163   } else {
164     print_info("Sector<Dofs_Per_Sector>::get_dof", "There seems to be an error in Sector::get_dof. i > 0
       && i < dofs_per_sector false.", false, LoggingLevel::PRODUCTION_ALL);
165     return 0;
166   }
167 }
```

4.62.3.2 get_m()

```
template<unsigned int Dofs_Per_Sector>
double Sector< Dofs_Per_Sector >::get_m (
            double z ) const
```

Get an interpolation of the shift for a coordinate z.

Parameters

double	z is the $z \in [0, 1]$ coordinate for the interpolation.

Definition at line 181 of file Sector.cpp.

```
181                                                 {
182   if (z < 0.0) z = 0.0;
183   if (z > 1.0) z = 1.0;
184   if (Dofs_Per_Sector == 2) {
185     return InterpolationPolynomial(z, dofs_l[0], dofs_r[0], dofs_l[1],
186                                    dofs_r[1]);
187   } else {
188     return InterpolationPolynomial(z, dofs_l[1], dofs_r[1], dofs_l[2],
189                                    dofs_r[2]);
190   }
191 }
```

4.62.3.3 get_r()

```
template<unsigned int Dofs_Per_Sector>
double Sector< Dofs_Per_Sector >::get_r (
            double z ) const
```

Get an interpolation of the radius for a coordinate z.

Parameters

double	z is the $z \in [0, 1]$ coordinate for the interpolation.

Definition at line 170 of file Sector.cpp.

```
170                                                 {
171   if (z < 0.0) z = 0.0;
172   if (z > 1.0) z = 1.0;
173   if (Dofs_Per_Sector < 3) {
174     print_info("Sector<Dofs_Per_Sector>::get_r", "Error in Sector: Access to radius dof without
      existence.", false, LoggingLevel::PRODUCTION_ALL);
175     return 0;
176   }
177   return InterpolationPolynomialZeroDerivative(z, dofs_l[0], dofs_r[0]);
178 }
```

4.62.3.4 get_v()

```
template<unsigned int Dofs_Per_Sector>
double Sector< Dofs_Per_Sector >::get_v (
            double z ) const
```

Get an interpolation of the tilt for a coordinate z.

Parameters

double	z is the $z \in [0, 1]$ coordinate for the interpolation.

Definition at line 194 of file Sector.cpp.

```
194                                                 {
195   if (z < 0.0) z = 0.0;
196   if (z > 1.0) z = 1.0;
197   if (Dofs_Per_Sector == 2) {
198     return InterpolationPolynomialZeroDerivative(z, dofs_l[1], dofs_r[1]);
199   } else {
200     return InterpolationPolynomialZeroDerivative(z, dofs_l[2], dofs_r[2]);
201   }
202 }
```

4.62.3.5 getLowestDof()

```
template<unsigned int Dofs_Per_Sector>
unsigned int Sector< Dofs_Per_Sector >::getLowestDof
```

This function returns the number of the lowest degree of freedom associated with this Sector.

Keep in mind, that the degrees of freedom associated with edges on the lower (small z) interface are not included since this functionality is supposed to help in the block-structure generation and those dofs are part of the neighboring block.

Definition at line 403 of file Sector.cpp.

```
403                                                 {
404   return LowestDof;
405 }
```

4.62.3.6 getNActiveCells()

```
template<unsigned int Dofs_Per_Sector>
unsigned int Sector< Dofs_Per_Sector >::getNActiveCells
```

This function can be used to query the number of cells in a Sector / subdomain.

In this case there are no problems with interface-dofs. Every cell belongs to exactly one sector (the problem arises from the fact, that one edge can (and most of the time will) belong to more then one cell).

Definition at line 418 of file Sector.cpp.

```
418                                                 {
419   return NActiveCells;
420 }
```

4.62.3.7 getNDofs()

```
template<unsigned int Dofs_Per_Sector>
unsigned int Sector< Dofs_Per_Sector >::getNDofs
```

This function returns the number of dofs which are part of this sector.

The same remarks as for getLowestDof() apply.

Definition at line 408 of file Sector.cpp.

```
408                                                 {
409   return NDofs;
410 }
```

4.62.3.8 getNInternalBoundaryDofs()

```
template<unsigned int Dofs_Per_Sector>
unsigned int Sector< Dofs_Per_Sector >::getNInternalBoundaryDofs
```

In order to set appropriate boundary conditions it makes sense to determine, which degrees are associated with an edge which is part of an interface to another sector.

Due to the reordering of dofs this is especially easy since the dofs on the interface are those in the interval

$$[\,\mathrm{LowestDof} + \mathrm{NDofs} - \mathrm{NInternalBoundaryDofs}\,,\ \mathrm{LowestDof} + \mathrm{NDofs}\,]$$

Definition at line 413 of file Sector.cpp.

```
413                                                                    {
414   return NInternalBoundaryDofs;
415 }
```

4.62.3.9 getQ1()

```
template<unsigned int Dofs_Per_Sector>
double Sector< Dofs_Per_Sector >::getQ1 (
            double z ) const
```

The values of Q1, Q2 and Q3 are needed to compute the solution in real coordinates from the one in trnsformed coordinates.

This function returnes Q1 for a given position and the current transformation.

Definition at line 205 of file Sector.cpp.

```
205                                                  {
206   return 1 / (dofs_l[0] + z * z * z * (2 * dofs_l[0] - 2 * dofs_r[0]) -
207               z * z * (3 * dofs_l[0] - 3 * dofs_r[0]));
208 }
```

4.62.3.10 getQ2()

```
template<unsigned int Dofs_Per_Sector>
double Sector< Dofs_Per_Sector >::getQ2 (
            double z ) const
```

The values of Q1, Q2 and Q3 are needed to compute the solution in real coordinates from the one in transformed coordinates.

This function returnes Q2 for a given position and the current transformation.

Definition at line 211 of file Sector.cpp.

```
211                                                  {
212   return 1 / (dofs_l[0] + z * z * z * (2 * dofs_l[0] - 2 * dofs_r[0]) -
213               z * z * (3 * dofs_l[0] - 3 * dofs_r[0]));
214 }
```

4.62.3.11 getQ3()

```
template<unsigned int Dofs_Per_Sector>
double Sector< Dofs_Per_Sector >::getQ3 (
            double  ) const
```

The values of Q1, Q2 and Q3 are needed to compute the solution in real coordinates from the one in transformed coordinates.

This function returnes Q3 for a given position and the current transformation.

Definition at line 217 of file Sector.cpp.

```
217                                                 {
218   return 0.0;
219 }
```

4.62.3.12 set_properties()

```
template<unsigned int Dofs_Per_Sector>
void Sector< Dofs_Per_Sector >::set_properties (
            double ,
            double ,
            double ,
            double  )
```

This function is used during the optimization-operation to update the properties of the space-transformation.

However, to ensure, that the boundary-conditions remain intact, this function cannot edit the left defrees of freedom if left is true and it cannot edit the right degrees of freedom if right is true

Definition at line 125 of file Sector.cpp.

```
125                                                                 {
126   print_info("Sector<Dofs_Per_Sector>::set_properties", "The code does not work for this number of dofs
       per Sector.", false, LoggingLevel::PRODUCTION_ALL);
127   return;
128 }
```

4.62.3.13 setLowestDof()

```
template<unsigned int Dofs_Per_Sector>
void Sector< Dofs_Per_Sector >::setLowestDof (
            unsigned int inLowestDOF )
```

Setter for the value that the getter should return.

Called after Dof-reordering.

Definition at line 423 of file Sector.cpp.

```
423                                                         {
424   LowestDof = inLowestDOF;
425 }
```

4.62.3.14 setNActiveCells()

```
template<unsigned int Dofs_Per_Sector>
void Sector< Dofs_Per_Sector >::setNActiveCells (
            unsigned int inNumberOfActiveCells )
```

Setter for the value that the getter should return.

Called after Dof-reordering.

Definition at line 439 of file Sector.cpp.

```
440                                         {
441   NActiveCells = inNumberOfActiveCells;
442 }
```

4.62.3.15 setNDofs()

```
template<unsigned int Dofs_Per_Sector>
void Sector< Dofs_Per_Sector >::setNDofs (
            unsigned int inNumberOfDOFs )
```

Setter for the value that the getter should return.

Called after Dof-reordering.

Definition at line 428 of file Sector.cpp.

```
428                                                 {
429   NDofs = inNumberOfDOFs;
430 }
```

4.62.3.16 setNInternalBoundaryDofs()

```
template<unsigned int Dofs_Per_Sector>
void Sector< Dofs_Per_Sector >::setNInternalBoundaryDofs (
            unsigned int in_ninternalboundarydofs )
```

Setter for the value that the getter should return.

Called after Dof-reordering.

Definition at line 433 of file Sector.cpp.

```
434                                             {
435   NInternalBoundaryDofs = in_ninternalboundarydofs;
436 }
```

4.62.3.17 TransformationTensorInternal()

```
template<unsigned int Dimension>
Tensor< 2, 3, double > Sector< Dimension >::TransformationTensorInternal (
            double in_x,
            double in_y,
            double in_z ) const
```

This method gets called from the WaveguideStructure object used in the simulation.

This is where the Waveguide object gets the material Tensors to build the system-matrix. This method returns a complex-values Matrix containing the system-tensors μ^{-1} and ϵ.

Parameters

in_x	x-coordinate of the point, for which the Tensor should be calculated.
in_y	y-coordinate of the point, for which the Tensor should be calculated.
in_z	z-coordinate of the point, for which the Tensor should be calculated.

Definition at line 395 of file Sector.cpp.

```
396                                                 {
397   Tensor<2, 3, double> ret;
398   print_info("Sector<Dimension>::TransformationTensorInternal", "The code does not work for you Sector
       specification." + std::to_string(Dimension), false, LoggingLevel::PRODUCTION_ALL);
399   return ret;
400 }
```

4.62.4 Member Data Documentation

4.62.4.1 z_1

```
template<unsigned int Dofs_Per_Sector>
const double Sector< Dofs_Per_Sector >::z_1
```

The objects created from this class are supposed to hand back the material properties which include the space-transformation Tensors.

For this to be possible, the Sector has to be able to transform from global coordinates to coordinates that are scaled inside the Sector. For this purpose, the z_0 and z_1 variables store the z-coordinate of both, the left and right surface.

Definition at line 55 of file Sector.h.

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

- Code/Core/Sector.h
- Code/Core/Sector.cpp

4.63 ShapeDescription Class Reference

Public Member Functions

- void **SetByString** (std::string)

Public Attributes

- int **Sectors**
- std::vector< double > **m**
- std::vector< double > **v**
- std::vector< double > **z**

4.63.1 Detailed Description

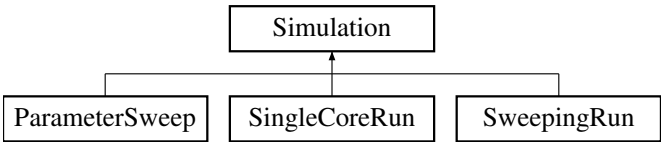
Definition at line 6 of file ShapeDescription.h.

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

- Code/Helpers/ShapeDescription.h
- Code/Helpers/ShapeDescription.cpp

4.64 Simulation Class Reference

Inheritance diagram for Simulation:



Public Member Functions

- virtual void **prepare** ()=0
- virtual void **run** ()=0
- virtual void **prepare_transformed_geometry** ()=0
- void **create_output_directory** ()

4.64.1 Detailed Description

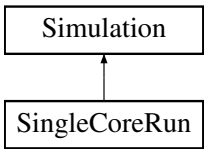
Definition at line 8 of file Simulation.h.

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

- Code/Runners/Simulation.h
- Code/Runners/Simulation.cpp

4.65 SingleCoreRun Class Reference

Inheritance diagram for SingleCoreRun:



Public Member Functions

- void **prepare** () override
- void **run** () override
- void **prepare_transformed_geometry** () override

4.65.1 Detailed Description

Definition at line 8 of file SingleCoreRun.h.

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

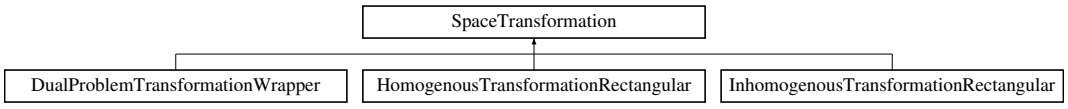
- Code/Runners/SingleCoreRun.h
- Code/Runners/SingleCoreRun.cpp

4.66 SpaceTransformation Class Reference

The SpaceTransformation class encapsulates the coordinate transformation used in the simulation.

`#include <SpaceTransformation.h>`

Inheritance diagram for SpaceTransformation:



Public Member Functions

- **SpaceTransformation** (int)
- virtual Position **math_to_phys** (Position coord) const =0
- virtual Position **phys_to_math** (Position coord) const =0
- bool **is_identity** (Position coord) const
- virtual Tensor< 2, 3, ComplexNumber > **get_Tensor** (Position &coordinate) const =0
- virtual Tensor< 2, 3, double > **get_Space_Transformation_Tensor** (Position &coordinate) const =0
- virtual Tensor< 2, 3, double > **get_Space_Transformation_Tensor_Homogenized** (Position &coordinate) const =0
- virtual Tensor< 2, 3, ComplexNumber > **get_Tensor_for_step** (Position &coordinate, unsigned int dof, double step_width)
- virtual void estimate_and_initialize ()=0

  At the beginning (before the first solution of a system) only the boundary conditions for the shape of the waveguide are known.
- virtual double get_Q1 (double z) const =0

  This member calculates the value of Q1 for a provided z-coordinate.
- virtual double get_Q2 (double z) const =0

  This member calculates the value of Q2 for a provided z-coordinate.
- virtual double get_Q3 (double z) const =0

  This member calculates the value of Q3 for a provided z-coordinate.

- virtual double get_dof (int dof) const =0

  This is a getter for the values of degrees of freedom.
- virtual void set_dof (int dof, double value)=0

  This function sets the value of the dof provided to the given value.
- virtual std::pair< double, double > **dof_support** (unsigned int index) const
- bool **point_in_dof_support** (Position location, unsigned int dof_index) const
- virtual double get_free_dof (int dof) const =0

  This is a getter for the values of degrees of freedom.
- virtual void set_free_dof (int dof, double value)=0

  This function sets the value of the dof provided to the given value.
- virtual std::pair< int, double > Z_to_Sector_and_local_z (double in_z) const

  Using this method unifies the usage of coordinates.
- double Sector_Length () const

  Returns the length of one sector.
- virtual double get_r (double in_z) const =0

  Returns the radius for a system-coordinate;.
- virtual double get_m (double in_z) const =0

  Returns the shift for a system-coordinate;.
- virtual double get_v (double in_z) const =0

  Returns the tilt for a system-coordinate;.
- virtual Vector< double > Dofs () const =0

  Other objects can use this function to retrieve an array of the current values of the degrees of freedom of the functional we are optimizing.
- virtual unsigned int NFreeDofs () const =0

  This function returns the number of unrestrained degrees of freedom of the current optimization run.
- virtual unsigned int NDofs () const =0

  This function returns the total number of DOFs including restrained ones.
- virtual bool IsDofFree (int) const =0

  Since Dofs() also returns restrained degrees of freedom, this function can be applied to determine if a degree of freedom is indeed free or restrained.
- virtual void Print () const =0

  Console output of the current Waveguide Structure.

Public Attributes

- bool **homogenized** = false
- const unsigned int **dofs_per_layer**
- const unsigned int **boundary_dofs_in**
- const unsigned int **boundary_dofs_out**
- const double epsilon_K

  The material-property ϵ_r has a different value inside and outside of the waveguides core.
- const double epsilon_M

  The material-property ϵ_r has a different value inside and outside of the waveguides core.
- const int sectors

  Since the computational domain is split into subdomains (called sectors), it is important to keep track of the amount of subdomains.
- const double deltaY

  This value is initialized with the value Delta from the input-file.
- Vector< double > InitialDofs

  This vector of values saves the initial configuration.
- double InitialQuality

  This vector of values saves the initial configuration.

4.66.1 Detailed Description

The SpaceTransformation class encapsulates the coordinate transformation used in the simulation.

Two important decisions have to be made in the computation: Which shape should be used for the waveguide? This can either be rectangular or tubular. Should the coordinate-transformation always be equal to identity in any domain where PML is applied? (yes or no). However, the space transformation is the only information required to compute the Tensor g which is a 3x3 matrix whilch (multiplied by the material value of the untransfomred coordinate either inside or outside the waveguide) gives us the value of ϵ and μ. From this class we derive several different classes which then specify the interface specified in this class.

Author

Pascal Kraft

Date

17.12.2015

Definition at line 33 of file SpaceTransformation.h.

4.66.2 Member Function Documentation

4.66.2.1 Dofs()

```
virtual Vector<double> SpaceTransformation::Dofs ( ) const  [pure virtual]
```

Other objects can use this function to retrieve an array of the current values of the degrees of freedom of the functional we are optimizing.

This also includes restrained degrees of freedom and other functions can be used to determine this property. This has to be done because in different cases the number of restrained degrees of freedom can vary and we want no logic about this in other functions.

Implemented in DualProblemTransformationWrapper, InhomogenousTransformationRectangular, and HomogenousTransformationRe

Referenced by DualProblemTransformationWrapper::Dofs().

4.66.2.2 estimate_and_initialize()

```
virtual void SpaceTransformation::estimate_and_initialize ( )  [pure virtual]
```

At the beginning (before the first solution of a system) only the boundary conditions for the shape of the waveguide are known.

Therefore the values for the degrees of freedom need to be estimated. This function sets all variables to appropiate values and estimates an appropriate shape based on averages and a polynomial interpolation of the boundary conditions on the shape.

Implemented in DualProblemTransformationWrapper, HomogenousTransformationRectangular, and InhomogenousTransformationRe

Referenced by DualProblemTransformationWrapper::estimate_and_initialize().

4.66.2.3 get_dof()

```
virtual double SpaceTransformation::get_dof (
            int dof ) const  [pure virtual]
```

This is a getter for the values of degrees of freedom.

A getter-setter interface was introduced since the values are estimated automatically during the optimization and non-physical systems should be excluded from the domain of possible cases.

Parameters

dof	The index of the degree of freedom to be retrieved from the structure of the modelled waveguide.

Returns

This function returns the value of the requested degree of freedom. Should this dof not exist, 0 will be returned.

Implemented in DualProblemTransformationWrapper, HomogenousTransformationRectangular, and InhomogenousTransformationRe

Referenced by DualProblemTransformationWrapper::get_dof().

4.66.2.4 get_free_dof()

```
virtual double SpaceTransformation::get_free_dof (
            int dof ) const  [pure virtual]
```

This is a getter for the values of degrees of freedom.

A getter-setter interface was introduced since the values are estimated automatically during the optimization and non-physical systems should be excluded from the domain of possible cases.

Parameters

dof	The index of the degree of freedom to be retrieved from the structure of the modelled waveguide.

Returns

This function returns the value of the requested degree of freedom. Should this dof not exist, 0 will be returnd.

Implemented in DualProblemTransformationWrapper, HomogenousTransformationRectangular, and InhomogenousTransformationRe

Referenced by DualProblemTransformationWrapper::get_free_dof().

4.66.2.5 get_Q1()

```
virtual double SpaceTransformation::get_Q1 (
            double z ) const  [pure virtual]
```

This member calculates the value of Q1 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q1 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implemented in DualProblemTransformationWrapper, HomogenousTransformationRectangular, and InhomogenousTransformationRe

Referenced by DualProblemTransformationWrapper::get_Q1().

4.66.2.6 get_Q2()

```
virtual double SpaceTransformation::get_Q2 (
            double z ) const  [pure virtual]
```

This member calculates the value of Q2 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q2 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implemented in DualProblemTransformationWrapper, HomogenousTransformationRectangular, and InhomogenousTransformationRe

Referenced by DualProblemTransformationWrapper::get_Q2().

4.66.2.7 get_Q3()

```
virtual double SpaceTransformation::get_Q3 (
            double z ) const  [pure virtual]
```

This member calculates the value of Q3 for a provided z-coordinate.

This value is used in the transformation of the solution-vector in transformed coordinates (solution of the system-matrix) to real coordinates (physical field).

Parameters

z	The value of Q3 is independent of x and y. Therefore only a z-coordinate is provided in a call to the function.

Implemented in DualProblemTransformationWrapper, HomogenousTransformationRectangular, and InhomogenousTransformationRe

Referenced by DualProblemTransformationWrapper::get_Q3().

4.66.2.8 IsDofFree()

```
virtual bool SpaceTransformation::IsDofFree (
            int  ) const  [pure virtual]
```

Since Dofs() also returns restrained degrees of freedom, this function can be applied to determine if a degree of freedom is indeed free or restrained.

"restrained" means that for example the DOF represents the radius at one of the connectors (input or output) and therefore we forbid the optimization scheme to vary this value.

Implemented in DualProblemTransformationWrapper, InhomogenousTransformationRectangular, and HomogenousTransformationRe

Referenced by DualProblemTransformationWrapper::IsDofFree().

4.66.2.9 NDofs()

```
virtual unsigned int SpaceTransformation::NDofs ( ) const  [pure virtual]
```

This function returns the total number of DOFs including restrained ones.

This is the lenght of the array returned by Dofs().

Implemented in DualProblemTransformationWrapper, InhomogenousTransformationRectangular, and HomogenousTransformationRe

Referenced by DualProblemTransformationWrapper::NDofs().

4.66.2.10 set_dof()

```
virtual void SpaceTransformation::set_dof (
            int dof,
            double value )  [pure virtual]
```

This function sets the value of the dof provided to the given value.

It is important to consider, that some dofs are non-writable (i.e. the values of the degrees of freedom on the boundary, like the radius of the input-connector cannot be changed).

Parameters

dof	The index of the parameter to be changed.
value	The value, the dof should be set to.

Implemented in DualProblemTransformationWrapper, HomogenousTransformationRectangular, and InhomogenousTransformationRe

Referenced by DualProblemTransformationWrapper::set_dof().

4.66.2.11 set_free_dof()

```
virtual void SpaceTransformation::set_free_dof (
            int dof,
            double value )  [pure virtual]
```

This function sets the value of the dof provided to the given value.

It is important to consider, that some dofs are non-writable (i.e. the values of the degrees of freedom on the boundary, like the radius of the input-connector cannot be changed).

Parameters

dof	The index of the parameter to be changed.
value	The value, the dof should be set to.

Implemented in DualProblemTransformationWrapper, HomogenousTransformationRectangular, and InhomogenousTransformationRe

Referenced by DualProblemTransformationWrapper::set_free_dof().

4.66.2.12 Z_to_Sector_and_local_z()

```
std::pair< int, double > SpaceTransformation::Z_to_Sector_and_local_z (
            double in_z ) const  [virtual]
```

Using this method unifies the usage of coordinates.

This function takes a global z coordinate (in the computational domain) and returns both a Sector-Index and an internal z coordinate indicating which sector this coordinate belongs to and how far along in the sector it is located.

Parameters

double	in_z global system z coordinate for the transformation.

Reimplemented in DualProblemTransformationWrapper.

Definition at line 10 of file SpaceTransformation.cpp.

```
10                                                                                                {
11   std::pair<int, double> ret;
12   ret.first = 0;
13   ret.second = 0.0;
14   if (in_z <= Geometry.global_z_range.first) {
15     ret.first = 0;
16     ret.second = 0.0;
17   } else if (in_z < Geometry.global_z_range.second && in_z > Geometry.global_z_range.first) {
18     ret.first = floor( (in_z + Geometry.global_z_range.first) / (GlobalParams.Sector_thickness));
19     ret.second = (in_z + Geometry.global_z_range.first -  (ret.first * GlobalParams.Sector_thickness)) /
       (GlobalParams.Sector_thickness);
20   } else if (in_z >= Geometry.global_z_range.second) {
21     ret.first = sectors - 1;
22     ret.second = 1.0;
23   }
24
25   if (ret.second < 0 || ret.second > 1){
26     std::cout « "Global ranges: " « Geometry.global_z_range.first « " to " «
       Geometry.global_z_range.second « std::endl;
27     std::cout « "Details " « GlobalParams.Sector_thickness « ", " « floor( (in_z +
       Geometry.global_z_range.first) / (GlobalParams.Sector_thickness)) « " and " « (in_z +
       Geometry.global_z_range.first) / (GlobalParams.Sector_thickness) « std::endl;
28     std::cout « "In an erroneous call: ret.first: " « ret.first « " ret.second: " « ret.second « " and
       in_z: " « in_z « " located in sector " « ret.first « " and " « GlobalParams.Sector_thickness «
       std::endl;
29   }
30   return ret;
31 }
```

References sectors.

Referenced by HomogenousTransformationRectangular::get_m(), InhomogenousTransformationRectangular↩ ::get_m(), InhomogenousTransformationRectangular::get_Q1(), HomogenousTransformationRectangular::get↩ _Q1(), InhomogenousTransformationRectangular::get_Q2(), HomogenousTransformationRectangular::get_↩ Q2(), InhomogenousTransformationRectangular::get_Q3(), HomogenousTransformationRectangular::get_Q3(), HomogenousTransformationRectangular::get_v(), InhomogenousTransformationRectangular::get_v(), and Dual↩ ProblemTransformationWrapper::Z_to_Sector_and_local_z().

4.66.3 Member Data Documentation

4.66.3.1 epsilon_K

```
const double SpaceTransformation::epsilon_K
```

The material-property ϵ_r has a different value inside and outside of the waveguides core.

This variable stores its value inside the core.

Definition at line 69 of file SpaceTransformation.h.

4.66.3.2 epsilon_M

```
const double SpaceTransformation::epsilon_M
```

The material-property ϵ_r has a different value inside and outside of the waveguides core.

This variable stores its value outside the core.

Definition at line 75 of file SpaceTransformation.h.

4.66.3.3 sectors

```
const int SpaceTransformation::sectors
```

Since the computational domain is split into subdomains (called sectors), it is important to keep track of the amount of subdomains.

This member stores the number of Sectors the computational domain has been split into.

Definition at line 81 of file SpaceTransformation.h.

Referenced by Z_to_Sector_and_local_z().

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

- Code/SpaceTransformations/SpaceTransformation.h
- Code/SpaceTransformations/SpaceTransformation.cpp

4.67 SquareMeshGenerator Class Reference

This class generates meshes, that are used to discretize a rectangular Waveguide. It is derived from Mesh↩ Generator.

```
#include <SquareMeshGenerator.h>
```

Public Member Functions

- bool math_coordinate_in_waveguide (Position position) const

  This function checks if the given coordinate is inside the waveguide or not.
- bool phys_coordinate_in_waveguide (Position position) const

  This function checks if the given coordinate is inside the waveguide or not.
- void prepare_triangulation (dealii::Triangulation< 3 > *in_tria)

  This function takes a triangulation object and prepares it for the further computations.
- unsigned int **getDominantComponentAndDirection** (Position in_dir) const
- void **set_boundary_ids** (dealii::Triangulation< 3 > &) const
- void **refine_triangulation_iteratively** (dealii::Triangulation< 3, 3 > *)
- bool **check_and_mark_one_cell_for_refinement** (dealii::Triangulation< 3 >::active_cell_iterator)

Public Attributes

- dealii::Triangulation< 3 >::active_cell_iterator **cell**
- dealii::Triangulation< 3 >::active_cell_iterator **endc**

4.67.1 Detailed Description

This class generates meshes, that are used to discretize a rectangular Waveguide. It is derived from Mesh↩ Generator.

The original intention of this project was to model tubular (or cylindrical) waveguides. The motivation behind this thought was the fact, that for this case the modes are known analytically. In applications however modes can be computed numerically and other shapes are easier to fabricate. For example square or rectangular waveguides can be printed in 3D on the scales we currently compute while tubular waveguides on that scale are not yet feasible.

Author

Pascal Kraft

Date

28.11.2016

Definition at line 23 of file SquareMeshGenerator.h.

4.67.2 Member Function Documentation

4.67.2.1 math_coordinate_in_waveguide()

```
bool SquareMeshGenerator::math_coordinate_in_waveguide (
            Position position ) const
```

This function checks if the given coordinate is inside the waveguide or not.

The naming convention of physical and mathematical system find application. In this version, the waveguide has been transformed and the check for a tubal waveguide for example only checks if the radius of a given vector is below the average of input and output radius. \params position This value gives us the location to check for.

4.67.2.2 phys_coordinate_in_waveguide()

```
bool SquareMeshGenerator::phys_coordinate_in_waveguide (
            Position position ) const
```

This function checks if the given coordinate is inside the waveguide or not.

The naming convention of physical and mathematical system find application. In this version, the waveguide is bent. If we are using a space transformation f then this function is equal to math_coordinate_in_waveguide(f(x,y,z)). \params position This value gives us the location to check for.

4.67.2.3 prepare_triangulation()

```
void SquareMeshGenerator::prepare_triangulation (
            dealii::Triangulation< 3 > * in_tria )
```

This function takes a triangulation object and prepares it for the further computations.

It is intended to encapsulate all related work and is explicitely not const.

Parameters

in_tria	The triangulation that is supposed to be prepared. All further information is derived from the parameter file and not given by parameters.

Definition at line 85 of file SquareMeshGenerator.cpp.

```
85                                                                                {
86   GridGenerator::hyper_cube(*in_tria, -1.0, 1.0, false);
87   GridTools::transform(&Triangulation_Shit_To_Local_Geometry, *in_tria);
88   set_boundary_ids(*in_tria);
89
90   in_tria->signals.post_refinement.connect(
91       std::bind(&SquareMeshGenerator::set_boundary_ids,
92           std::cref(*this), std::ref(*in_tria)));
93
94   refine_triangulation_iteratively(in_tria);
95
96   set_boundary_ids(*in_tria);
97 }
```

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

- Code/MeshGenerators/SquareMeshGenerator.h
- Code/MeshGenerators/SquareMeshGenerator.cpp

4.68 SurfaceCellData Struct Reference

Public Attributes

- std::vector< DofNumber > **dof_numbers**
- Position **surface_face_center**

4.68.1 Detailed Description

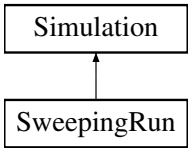
Definition at line 207 of file Types.h.

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

- Code/Core/Types.h

4.69 SweepingRun Class Reference

Inheritance diagram for SweepingRun:



Public Member Functions

- void **prepare** () override
- void **run** () override
- void **prepare_transformed_geometry** () override

4.69.1 Detailed Description

Definition at line 8 of file SweepingRun.h.

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

- Code/Runners/SweepingRun.h
- Code/Runners/SweepingRun.cpp

4.70 tagGSPHERE Struct Reference

Public Attributes

- int **n**
- double * **r**
- double * **t**
- double * **q**
- double * **A**
- double **B**

4.70.1 Detailed Description

Definition at line 23796 of file QuadratureFormulaCircle.cpp.

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

- Code/Helpers/QuadratureFormulaCircle.cpp

4.71 TimerManager Class Reference

Public Member Functions

- void **initialize** ()
- void **switch_context** (std::string context, unsigned int level)
- void **write_output** ()
- void **leave_context** (unsigned int level)

Public Attributes

- std::vector< dealii::TimerOutput > **timer_outputs**
- std::vector< std::string > **filenames**
- std::vector< std::ofstream * > **filestreams**
- unsigned int **level_count**

4.71.1 Detailed Description

Definition at line 6 of file TimerManager.h.

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

- Code/GlobalObjects/TimerManager.h
- Code/GlobalObjects/TimerManager.cpp

4.72 VertexAngelingData Struct Reference

Public Attributes

- unsigned int **vertex_index**
- bool **angled_in_x** = false
- bool **angled_in_y** = false

4.72.1 Detailed Description

Definition at line 70 of file Types.h.

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

- Code/Core/Types.h

Index