HVDC

Generated by Doxygen 1.9.4

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

File Documentation

2.1 test_3.h File Reference

Test case with oil and paper layers and oil filled cubic butt gaps.

```
#include <tmesh_3d.h>
#include <simple_connectivity_3d_thin.h>
```

Functions

- static int uniform_refinement (tmesh_3d::quadrant_iterator q)

  Uniform refinement function.
- static int refinement (tmesh_3d::quadrant_iterator quadrant)

  Performs local refinement at the interfaces.
- static int coarsening (tmesh_3d::quadrant_iterator quadrant)

  Performs local coarsening, leaves interfaces refined.
- double epsilon_fun (const double &x, const double &y, const double &z)

  Return the value of epsilon at the point (x,y,z)
- double sigma_fun (const double &x, const double &y, const double &z)

  Return the value of sigma at the point (x,y,z)
- std::vector< size_t > find_idx (tmesh_3d &tmsh, std::vector< std::vector< double > > &points, std::vector< std::vector< double > > &tols, const size_t &N_rhos)

  Find the global index of the points given by the vector 'points'.

Variables

- constexpr int NUM_REFINEMENTS = 5

  Level for global refinement.
- constexpr int maxlevel = 6

  Level for local refinement.
- constexpr int minlevel = 4

  Level for global coarsening.
- constexpr double DELTAT = 50.0

  Temporal time step.

- constexpr double T = 5000

  Final time of simulation.
- constexpr double tau = 50.0

  Time constant for boundary conditions.
- constexpr bool save_sol = true

  If set to 'true' saves data for Paraview visualization.
- constexpr double epsilon_0 = 8.8542e-12

  Permittivity vacuum.
- constexpr double epsilon_r_1 = 2.0

  Permittivity oil.
- constexpr double epsilon_r_2 = 4.0

  Permittivity paper.
- constexpr double sigma_ = 3.21e-14

  Conducivity.
- constexpr double z_oil = 5e-5

  Thickness oil layer.
- constexpr double z_paper = 3e-4

  Thickness paper layer and butt gaps side length.
- constexpr double tol = 1e-5

  Tolerance value for refinement and point selection.
- constexpr size_t N_rhos = 6

  Number of poiunts to select for output.
- std::vector< size_t > rho_idx
- std::vector< std::vector< double > > points {{5e-4,5e-4,0.0},{5e-4,5e-4,z_paper},{5e-4,5e-4,z_oil+z_paper},{5e-4,5e-4,z_oil+2∗z_paper},{5e-4,5e-4,2∗z_oil+2∗z_paper},{5e-4,5e-4,1e-3}}

  Coordinates of the selected points.
- std::vector< std::vector< double > > tols {{1e-4,1e-4,tol},{1e-4,1e-4,tol},{1e-4,1e-4,tol},{1e-4,1e-4,tol},{1e-4,1e-4,tol},{1e-4,1e-4,tol}}

  Tolerance around the selected points.
- bool extra_refinement = true

  true for all cases except Test 1

2.1.1 Detailed Description

Test case with oil and paper layers and oil filled cubic butt gaps.

2.1.2 Function Documentation

2.1.2.1 coarsening()

```
static int coarsening (
            tmesh_3d::quadrant_iterator quadrant )  [static]
```

Performs local coarsening, leaves interfaces refined.

2.1.2.2 epsilon_fun()

```
double epsilon_fun (
            const double & x,
            const double & y,
            const double & z )
```

Return the value of epsilon at the point (x,y,z)

2.1.2.3 find_idx()

```
std::vector< size_t > find_idx (
            tmesh_3d & tmsh,
            std::vector< std::vector< double > > & points,
            std::vector< std::vector< double > > & tols,
            const size_t & N_rhos )
```

Find the global index of the points given by the vector 'points'.

2.1.2.4 refinement()

```
static int refinement (
            tmesh_3d::quadrant_iterator quadrant )  [static]
```

Performs local refinement at the interfaces.

2.1.2.5 sigma_fun()

```
double sigma_fun (
            const double & x,
            const double & y,
            const double & z )
```

Return the value of sigma at the point (x,y,z)

2.1.2.6 uniform_refinement()

```
static int uniform_refinement (
            tmesh_3d::quadrant_iterator q )  [static]
```

Uniform refinement function.

2.1.3 Variable Documentation

2.1.3.1 DELTAT

```
constexpr double DELTAT = 50.0  [constexpr]
```

Temporal time step.

2.1.3.2 epsilon_0

```
constexpr double epsilon_0 = 8.8542e-12  [constexpr]
```

Permittivity vacuum.

2.1.3.3 epsilon_r_1

```
constexpr double epsilon_r_1 = 2.0  [constexpr]
```

Permittivity oil.

2.1.3.4 epsilon_r_2

```
constexpr double epsilon_r_2 = 4.0  [constexpr]
```

Permittivity paper.

2.1.3.5 extra_refinement

```
bool extra_refinement = true
```

true for all cases except Test 1

2.1.3.6 maxlevel

```
constexpr int maxlevel = 6  [constexpr]
```

Level for local refinement.

2.1.3.7 minlevel

```
constexpr int minlevel = 4  [constexpr]
```

Level for global coarsening.

2.1.3.8 N_rhos

```
constexpr size_t N_rhos = 6  [constexpr]
```

Number of poiunts to select for output.

2.1.3.9 NUM_REFINEMENTS

```
constexpr int NUM_REFINEMENTS = 5  [constexpr]
```

Level for global refinement.

2.1.3.10 points

```
std::vector<std::vector<double> > points {{5e-4,5e-4,0.0},{5e-4,5e-4,z_paper},{5e-4,5e-4,z_oil+z_paper},{5e-
```

Coordinates of the selected points.

2.1.3.11 rho_idx

```
std::vector<size_t> rho_idx
```

2.1.3.12 save_sol

```
constexpr bool save_sol = true  [constexpr]
```

If set to 'true' saves data for Paraview visualization.

2.1.3.13 sigma_

```
constexpr double sigma_ = 3.21e-14  [constexpr]
```

Conducivity.

2.1.3.14 T

```
constexpr double T = 5000  [constexpr]
```

Final time of simulation.

2.1.3.15 tau

```
constexpr double tau = 50.0  [constexpr]
```

Time constant for boundary conditions.

2.1.3.16 tol

```
constexpr double tol = 1e-5  [constexpr]
```

Tolerance value for refinement and point selection.

2.1.3.17 tols

```
std::vector<std::vector<double> > tols {{1e-4,1e-4,tol},{1e-4,1e-4,tol},{1e-4,1e-4,tol},{1e-4,1e-4,tol},{1e-
```

Tolerance around the selected points.

2.1.3.18 z_oil

```
constexpr double z_oil = 5e-5  [constexpr]
```

Thickness oil layer.

2.1.3.19 z_paper

```
constexpr double z_paper = 3e-4  [constexpr]
```

Thickness paper layer and butt gaps side length.

2.2 test_3.h

Go to the documentation of this file.

```
1
5 #include <tmesh_3d.h>
6 #include <simple_connectivity_3d_thin.h> // Loading library with domain geometry
7
8 constexpr int NUM_REFINEMENTS = 5;
9 constexpr int maxlevel = 6;
10 constexpr int minlevel = 4;
12 constexpr double DELTAT = 50.0;
13 constexpr double T = 5000;
14 constexpr double tau = 50.0;
15 constexpr bool save_sol = true;
17 // Problem parameters
18 constexpr double epsilon_0 = 8.8542e-12;
19 constexpr double epsilon_r_1 = 2.0;
20 constexpr double epsilon_r_2 = 4.0;
21 constexpr double sigma_ = 3.21e-14;
23 constexpr double z_oil = 5e-5;
24 constexpr double z_paper = 3e-4;
25 constexpr double tol = 1e-5;
27 constexpr size_t N_rhos = 6;
28 std::vector<size_t> rho_idx;
29 std::vector<std::vector<double>>
     points{{5e-4,5e-4,0.0},{5e-4,5e-4,z_paper},{5e-4,5e-4,z_oil+z_paper},{5e-4,5e-4,z_oil+2*z_paper},{5e-4,5e-4,2*z_oil+2*z_
30 std::vector<std::vector<double>>
     tols{{1e-4,1e-4,tol},{1e-4,1e-4,tol},{1e-4,1e-4,tol},{1e-4,1e-4,tol},{1e-4,1e-4,tol},{1e-4,1e-4,tol}};
31 bool extra_refinement = true;
33 static int
34 uniform_refinement (tmesh_3d::quadrant_iterator q)
35 { return NUM_REFINEMENTS; }
37 static int
38 refinement (tmesh_3d::quadrant_iterator quadrant)
39 {
40   int currentlevel = static_cast<int> (quadrant->the_quadrant->level);
41   double zcoord;
42   int retval = 0;
43   for (int ii = 0; ii < 8; ++ii)
44     {
45       zcoord = quadrant->p(2, ii);
46
47       if (zcoord > z_paper - tol || zcoord < 2*z_paper+2*z_oil+tol)
48         {
49           retval = maxlevel - currentlevel;
50           break;
51         }
52     }
53
54   if (currentlevel >= maxlevel)
55     retval = 0;
56
57   return retval;
58 }
60 static int
61 coarsening (tmesh_3d::quadrant_iterator quadrant)
62 {
63   int currentlevel = static_cast<int> (quadrant->the_quadrant->level);
```

```
64   double xcoord,ycoord,zcoord;
65   int retval = currentlevel - minlevel;
66   for (int ii = 0; ii < 8; ++ii)
67     {
68       xcoord = quadrant->p(0, ii);
69       ycoord = quadrant->p(1, ii);
70       zcoord = quadrant->p(2, ii);
71
72       if (fabs(zcoord - z_paper) < tol || fabs(zcoord - z_paper-z_oil) < tol || fabs(zcoord -
     2*z_paper-z_oil) < tol || fabs(zcoord - 2*z_paper-2*z_oil) < tol ||
73           (xcoord<z_paper && fabs(ycoord-5e-4+z_paper/2)<tol && zcoord>5e-4-z_paper/2 &&
     zcoord<5e-4+z_paper/2) ||
74           (xcoord<z_paper && fabs(ycoord-5e-4-z_paper/2)<tol && zcoord>5e-4-z_paper/2 &&
     zcoord<5e-4+z_paper/2) ||
75           (fabs(xcoord-z_paper)<tol && ycoord>5e-4-z_paper/2 && ycoord<5e-4+z_paper/2 &&
     zcoord>5e-4-z_paper/2 && zcoord<5e-4+z_paper/2))
76         {
77           retval = 0;
78           break;
79         }
80     }
81
82   if (currentlevel <= minlevel)
83     retval = 0;
84
85   return (retval);
86 }
88 double epsilon_fun(const double & x, const double & y, const double & z)
89 {
90     if((z > z_paper && z<z_paper+z_oil) || (z>2*z_paper+z_oil && z<2*(z_paper+z_oil)))
91         return  epsilon_0 * epsilon_r_1;
92
93     if((z > z_paper+z_oil && z<2*z_paper+z_oil) && x<z_paper && (y>5e-4-z_paper/2 && y<5e-4+z_paper/2))
94         return  epsilon_0 * epsilon_r_1;
95
96     return epsilon_0 * epsilon_r_2;
97 }
99 double sigma_fun(const double & x, const double & y, const double & z)
100 {return sigma_ * DELTAT;}
102 std::vector<size_t> find_idx(tmesh_3d &tmsh,std::vector<std::vector<double>
      &points,std::vector<std::vector<double> &tols, const size_t &N_rhos)
103 {
104   std::vector<size_t> id(N_rhos,0);
105
106   for (size_t i = 0; i < N_rhos; i++) {
107     bool found = false;
108
109     for (auto quadrant = tmsh.begin_quadrant_sweep ();
110        quadrant != tmsh.end_quadrant_sweep ();
111        ++quadrant){
112
113         for (int ii = 0; ii < 8; ++ii) {
114
115           if (fabs(quadrant->p(0,ii)-points[i][0])<tols[i][0] &&
     fabs(quadrant->p(1,ii)-points[i][1])<tols[i][1] && fabs(quadrant->p(2,ii)-points[i][2])<tols[i][2]) {
116             id[i] = quadrant->t(ii);
117             found = true;
118             std::cout « "Point " « i+1 « ":  x= " « quadrant->p(0,ii) « ", y= " « quadrant->p(1,ii) « ",
     z= " « quadrant->p(2,ii) « std::endl;
119             break;
120           }
121         }
122
123         if (found)
124           break;
125     }
126     if(!found)
127         std::cout « "Node " « i+1 « " not found in current rank" « std::endl;
128   }
129   return id;
130 }
```

2.3 HVDC_main.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <algorithm>
#include <octave_file_io.h>
```

```
#include <bim_distributed_vector.h>
#include <bim_sparse_distributed.h>
#include <bim_timing.h>
#include <mumps_class.h>
#include <quad_operators_3d.h>
#include <test_4.h>
```

Functions

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

2.3.1 Detailed Description

Author

Alessandro Lombardi

Version

0.1

2.3.2 LICENSE

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details at https://www.gnu.org/copyleft/gpl.html

2.3.3 Function Documentation

2.3.3.1 main()

```
int main (
              int argc,
              char ** argv )
40 {
41
42   using q1_vec  = q1_vec<distributed_vector>;
43
44   /*
45 Manegement of solutions ordering:  ord0-> phi                ord1->rho
46 Equation ordering:  ord0->diffusion-reaction equation       ord1->continuity equation
47 */
48   ordering
49     ord0 = [] (tmesh_3d::idx_t gt) -> size_t { return dof_ordering<2, 0> (gt); },
50     ord1 = [] (tmesh_3d::idx_t gt) -> size_t { return dof_ordering<2, 1> (gt); };
51
52   // Initialize MPI
```

```
53   MPI_Init (&argc, &argv);
54   int rank, size;
55   MPI_Comm_rank (MPI_COMM_WORLD, &rank);
56   MPI_Comm_size (MPI_COMM_WORLD, &size);
57
58   // Generate the mesh in 3d
59   tmesh_3d tmsh;
60   tmsh.read_connectivity (simple_conn_p, simple_conn_num_vertices,
61                           simple_conn_t, simple_conn_num_trees);
62
63   //Uniform refinement
64   int recursive = 1;
65   tmsh.set_refine_marker (uniform_refinement);
66   tmsh.refine (recursive);
67
68   //In test 1 we only have uniform refinement, in all other cases we perform additional refinement
69   if (extra_refinement)
70   {
71     tmsh.set_refine_marker(refinement);
72     tmsh.refine (recursive);
73
74     tmsh.set_coarsen_marker(coarsening);
75     tmsh.coarsen(recursive);
76   }
77
78   tmesh_3d::idx_t gn_nodes = tmsh.num_global_nodes ();
79   tmesh_3d::idx_t ln_nodes = tmsh.num_owned_nodes ();
80   tmesh_3d::idx_t ln_elements = tmsh.num_local_quadrants ();
81
82   // Allocate linear solver
83   mumps *lin_solver = new mumps ();
84
85  // Allocate initial data container
86   q1_vec sold (ln_nodes * 2);
87   sold.get_owned_data ().assign (sold.get_owned_data ().size (), 0.0);
88
89   q1_vec sol (ln_nodes * 2);
90   sol.get_owned_data ().assign (sol.get_owned_data ().size (), 0.0);
91
92   std::vector<double> xa;
93   std::vector<int> ir, jc;
94
95   // Declare system matrix
96   distributed_sparse_matrix A;
97   A.set_ranges (ln_nodes * 2);
98
99   // Buffer for export filename
100   char filename[255]="";
101
102   //Output rho vector
103   size_t N_timesteps = (size_t) (ceil(T/DELTAT)+1);
104   std::vector<std::vector<double» rho_out(N_timesteps,std::vector<double>(N_rhos+1));
105
106   // Compute coefficients
107
108   // diffusion
109   std::vector<double> epsilon (ln_elements, 0.);
110   std::vector<double> sigma (ln_elements, 0.);
111   q1_vec zero (ln_nodes);
112
113   // reaction
114   std::vector<double> delta0 (ln_elements, 0.);
115   std::vector<double> delta1 (ln_elements, 0.);
116   q1_vec zeta0 (ln_nodes);
117   q1_vec zeta1 (ln_nodes);
118
119   // rhs
120   std::vector<double> f0 (ln_elements, 0.);
121   std::vector<double> f1 (ln_elements, 0.);
122   q1_vec g0 (ln_nodes);
123   q1_vec g1 (ln_nodes);
124
125   // Initialize constant (in time) parameters and initial data
126   for (auto quadrant = tmsh.begin_quadrant_sweep ();
127        quadrant != tmsh.end_quadrant_sweep ();
128        ++quadrant)
129     {
130       double xx{quadrant->centroid(0)}, yy{quadrant->centroid(1)}, zz{quadrant->centroid(2)};
131
132       epsilon[quadrant->get_forest_quad_idx ()] = epsilon_fun(xx,yy,zz);
133       sigma[quadrant->get_forest_quad_idx ()] = sigma_fun(xx,yy,zz);
134
135       delta0[quadrant->get_forest_quad_idx ()] = -1.0;
136       delta1[quadrant->get_forest_quad_idx ()] = 1.0;
137       f0[quadrant->get_forest_quad_idx ()] = 0.0;
138       f1[quadrant->get_forest_quad_idx ()] = 1.0;
139
```

```
140       for (int ii = 0; ii < 8; ++ii)
141         {
142           if (!  quadrant->is_hanging (ii))
143           {
144             zero[quadrant->gt (ii)] = 0.;
145             zeta0[quadrant->gt (ii)] = 1.0;
146             zeta1[quadrant->gt (ii)] = 1.0;
147             g0[quadrant->gt (ii)] = 0.;
148 
149             double zz=quadrant->p(2,ii);
150             sold[ord0(quadrant->gt (ii))] = 0.0;
151             sold[ord1(quadrant->gt (ii))] = 0.0;
152             sol[ord0(quadrant->gt (ii))] = 0.0;
153             sol[ord1(quadrant->gt (ii))] = 0.0;
154           }
155           else
156             for (int jj = 0; jj < quadrant->num_parents (ii); ++jj)
157               {
158                 zero[quadrant->gparent (jj, ii)] += 0.;
159                 zeta0[quadrant->gparent (jj, ii)] += 0.;
160                 zeta1[quadrant->gparent (jj, ii)] += 0.;
161                 g0[quadrant->gparent (jj, ii)] += 0.;
162 
163                 sold[ord0(quadrant->gparent (jj, ii))] += 0.;
164                 sold[ord1(quadrant->gparent (jj, ii))] += 0.;
165               }
166         }
167     }
168 
169   bim3a_solution_with_ghosts (tmsh, sold, replace_op, ord0, false);
170   bim3a_solution_with_ghosts (tmsh, sold, replace_op, ord1);
171 
172   zero.assemble (replace_op);
173   zeta0.assemble (replace_op);
174   zeta1.assemble (replace_op);
175   g0.assemble (replace_op);
176 
177   // Save inital conditions
178   sprintf(filename, "model_0_u_0000");
179   tmsh.octbin_export (filename, sold, ord0);
180   sprintf(filename, "model_0_v_0000");
181   tmsh.octbin_export (filename, sold, ord1);
182 
183   int count = 0;
184 
185   //Choosing the indices for the nodes corresponding to the vales of rho of interest
186   if (rank == 0) {
187     rho_out[0]=std::vector<double>(N_rhos+1,0.0);
188     rho_idx = find_idx(tmsh,points,tols,N_rhos);
189   }
190 
191   // Time cycle
192   for( double time = DELTAT; time <= T; time += DELTAT)
193   {
194     count++;
195 
196     // Define boundary conditions
197     dirichlet_bcs3 bcs0, bcs1;
198     bcs0.push_back (std::make_tuple (0, 4, [](double x, double y, double z){return 0.0;})); //bottom
199     bcs0.push_back (std::make_tuple (0, 5, [time](double x, double y, double z){return 1.5e4 * (1 -
    exp(-time/tau));})); //top
200 
201     // Print curent time
202     if(rank==0)
203       std::cout«"TIME= "«time«std::endl;
204 
205     // Reset containers
206     A.reset ();
207     sol.get_owned_data ().assign (sol.get_owned_data ().size (), 0.0);
208     sol.assemble (replace_op);
209 
210     // Initialize non constant (in time) parameters
211     for (auto quadrant = tmsh.begin_quadrant_sweep ();
212     quadrant != tmsh.end_quadrant_sweep ();
213     ++quadrant)
214     {
215       for (int ii = 0; ii < 8; ++ii)
216         if (!  quadrant->is_hanging (ii))
217           g1[quadrant->gt (ii)] = sold[ord1(quadrant->gt (ii))];
218 
219         else
220           for (int jj = 0; jj < quadrant->num_parents (ii); ++jj)
221             g1[quadrant->gparent (jj, ii)] += 0.;
222     }
223 
224     g1.assemble(replace_op);
225 
```

```
226     // advection_diffusion
227     bim3a_advection_diffusion (tmsh, epsilon, zero, A, true, ord0, ord0);
228     bim3a_advection_diffusion (tmsh, sigma, zero, A, true, ord1, ord0);
229
230     // reaction
231     bim3a_reaction (tmsh, delta0, zeta0, A, ord0, ord1);
232     bim3a_reaction (tmsh, delta1, zeta1, A, ord1, ord1);
233
234     //rhs
235     bim3a_rhs (tmsh, f0, g0, sol, ord0);
236     bim3a_rhs (tmsh, f1, g1, sol, ord1);
237
238     //boundary conditions
239     bim3a_dirichlet_bc (tmsh, bcs0, A, sol, ord1, ord0, false);
240
241     // Communicate matrix and RHS
242     A.assemble ();
243     sol.assemble ();
244
245     // Solver analysis
246     lin_solver->set_lhs_distributed ();
247     A.aij (xa, ir, jc, lin_solver->get_index_base ());
248     lin_solver->set_distributed_lhs_structure (A.rows (), ir, jc);
249     std::cout « "lin_solver->analyze () return value = "« lin_solver->analyze () « std::endl;
250
251     // Matrix update
252     A.aij_update (xa, ir, jc, lin_solver->get_index_base ());
253     lin_solver->set_distributed_lhs_data (xa);
254
255     // Factorization
256     std::cout « "lin_solver->factorize () = " « lin_solver->factorize () « std::endl;
257
258     // Set RHS data
259     lin_solver->set_rhs_distributed (sol);
260
261     // Solution
262     std::cout « "lin_solver->solve () = " « lin_solver->solve () « std::endl;
263
264     // Copy solution
265     q1_vec result = lin_solver->get_distributed_solution ();
266     for (int idx = sold.get_range_start (); idx < sold.get_range_end (); ++idx)
267       sold (idx) = result (idx);
268     sold.assemble (replace_op);
269
270     // Save solution
271     if (save_sol == true)
272     {
273       sprintf(filename, "model_0_u_%4.4d",count);
274       tmsh.octbin_export (filename, sold, ord0);
275       sprintf(filename, "model_0_v_%4.4d",count);
276       tmsh.octbin_export (filename, sold, ord1);
277     }
278
279     // Save rho values
280     if (rank == 0)
281     {
282       std::vector<double> temp(N_rhos+1);
283       temp[0] = time;
284       for (size_t i=1; i < N_rhos+1; i++)
285         temp[i] = sold[ord1(rho_idx[i-1])];
286
287       rho_out[count] = temp;
288     }
289   }
290
291   // Print file with rho values
292   if (rank == 0)
293   {
294     std::ofstream outFile("Arho.txt");
295     outFile « N_rhos+1 « "\t" « N_timesteps « "\n";
296
297     for (const auto &e :  rho_out) {
298       for (size_t i=0; i < N_rhos+1; i++)
299         outFile « e[i] « "\t";
300       outFile « "\n";
301     }
302   }
303
304   // Close MPI and print report
305   MPI_Barrier (MPI_COMM_WORLD);
306
307   // Clean linear solver
308   lin_solver->cleanup ();
309
310   MPI_Finalize ();
311
312   return 0;
```

```
313 }
```

Index