EMWSD / Wakis

Electromagnetic and Wake Solver Development

Meeting #18

Elena de la Fuente

Carlo Zannini, Lorenzo Giacomel, Giovanni Iadarola

Outline

1. **Where are we?**

2. Main improvements

3. Some (fun) examples

4. Feedback from University

5. Conclusions & Next steps

PhD roadmap

- **2023: Jul – Ago (2 months):**
  - ✓ Physics review FIT
  - ✓ **3D Eqs** in python
  - Test in cube (*PEC BCs not working!*)

- **2023: Sep-Dic (4 months):**
  - PEC BCs
  - Embedded boundaries
  - Add PML/CPML
  - First taste of materials

We are here !

- **2024: Jan–April (4 months):**
  - UNIT TEST

  (…)

Outline

1. Where are we?

2. Main improvements

3. Some (fun) examples

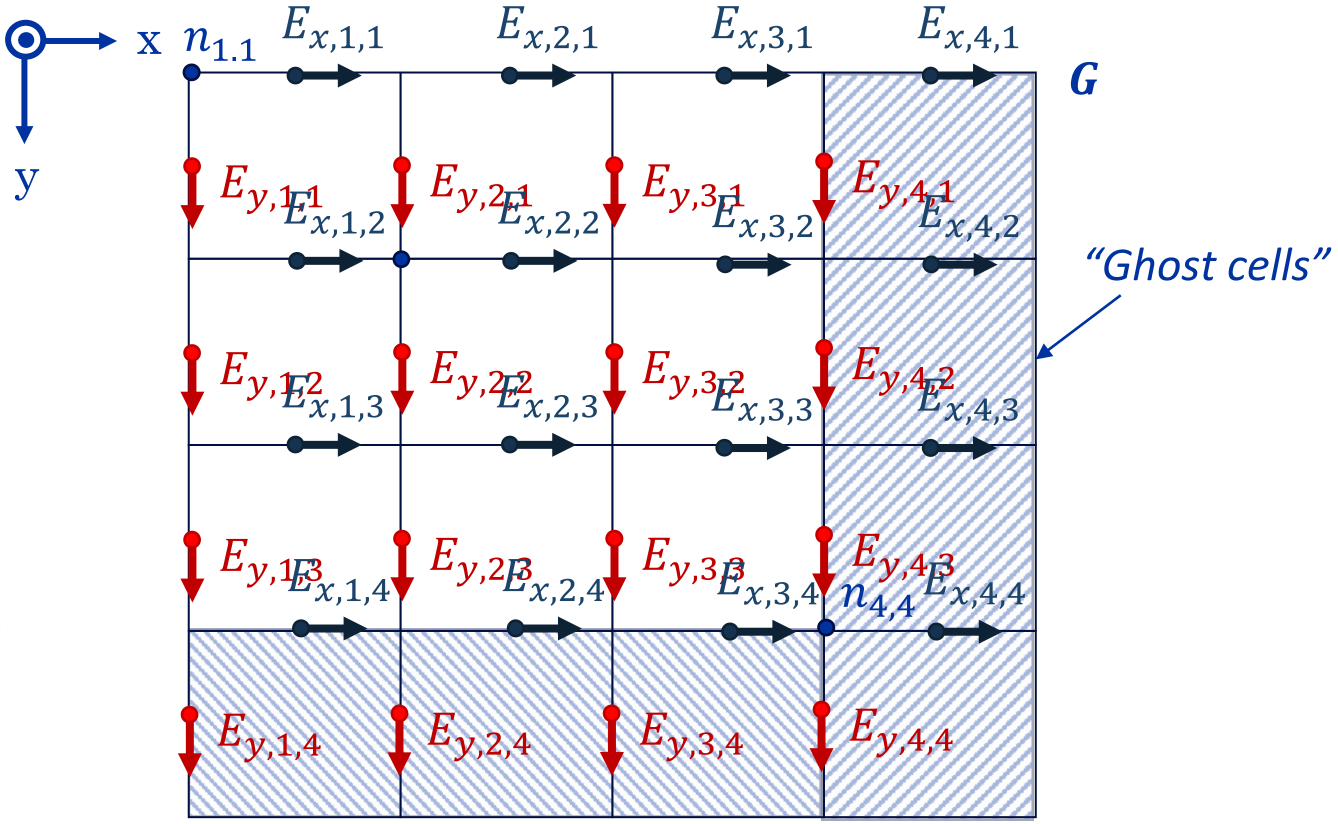
4. Feedback from University

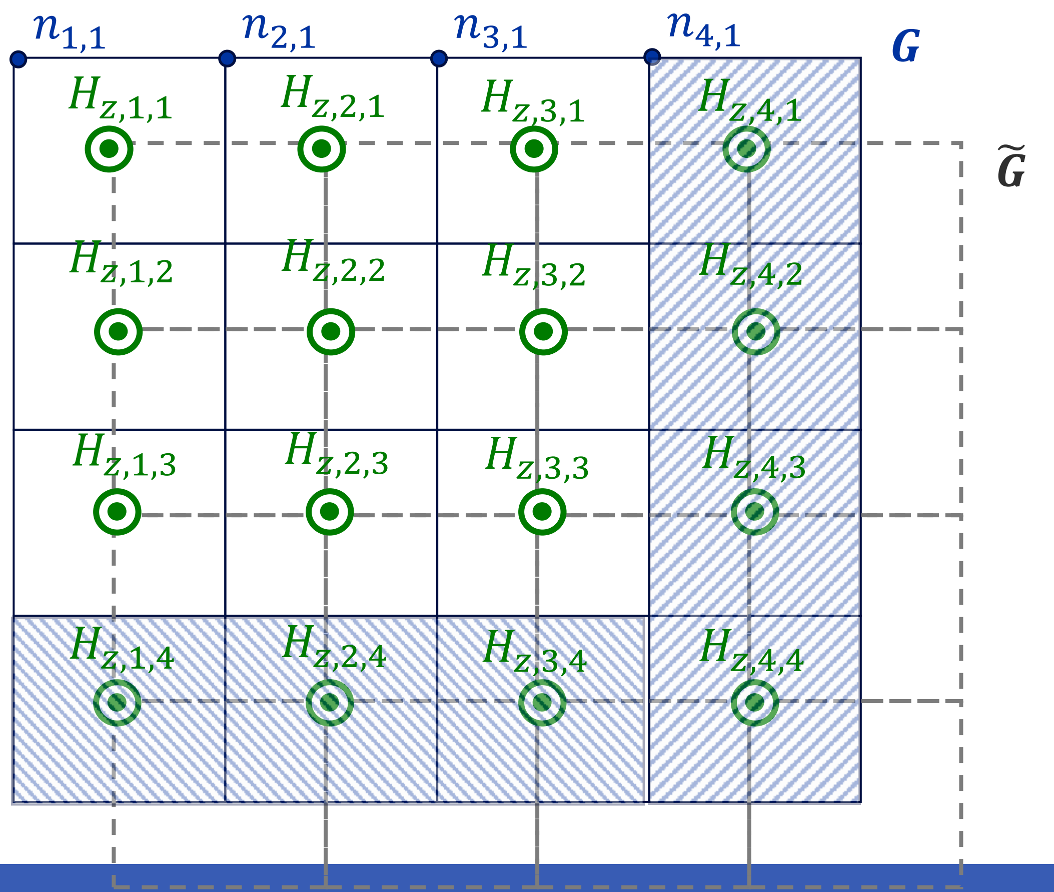
5. Conclusions & Next steps

Primal grid G vs tilde grid $\widetilde{G}$

*For quantities allocated on **primary edges or dual facets** ,such as for $\boldsymbol{E}$ and $\boldsymbol{D}$, these are the components $v_x(N_x, j, k), v_y(i, N_y, k)$ and $v_z(i, j, N_z)$: i.e., *longitudinal*.

In contrast, for vectors gathering quantities allocated on **dual edges or primary facets**, such as $\boldsymbol{H}$ and $\boldsymbol{B}$, these components are $v_x(i, N_y, k), v_x(i, j, N_z), v_y(N_x, i, k), v_y(i, j, N_z), v_z(N_x, j, k)$ and $v_z(i, N_y, k)$: i. e., *tranvserse*





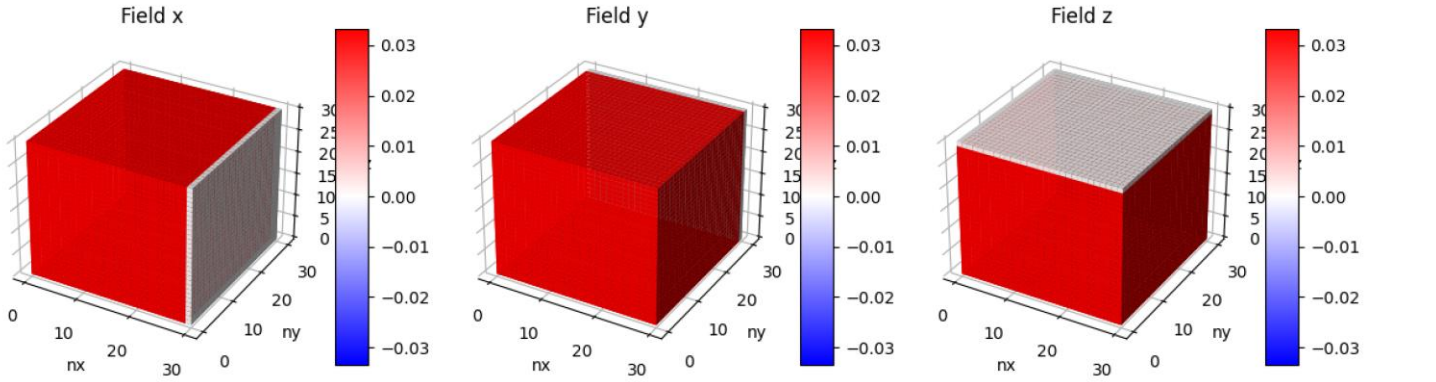
Primal grid G vs tilde grid $\widetilde{G}$ (II)

We can enforce the quantities at the ghost cells to be zero using the ***lengths*** and ***areas*** diagonal matrices $\widetilde{D}_L, \widetilde{D}_A$

- **By definition,** $\tilde{x}_{N_x} = \tilde{x}_{N_x-1}, \tilde{y}_{N_y} = \tilde{y}_{N_y-1}, \tilde{z}_{N_z} = \tilde{z}_{N_z-1}.$
- **This gives the following *lengths*** $\tilde{L}_x = \tilde{x}_{N_x} - \tilde{x}_{N_x-1}$ **and *areas*** $\tilde{A}_x = \tilde{L}_y \cdot \tilde{L}_z$

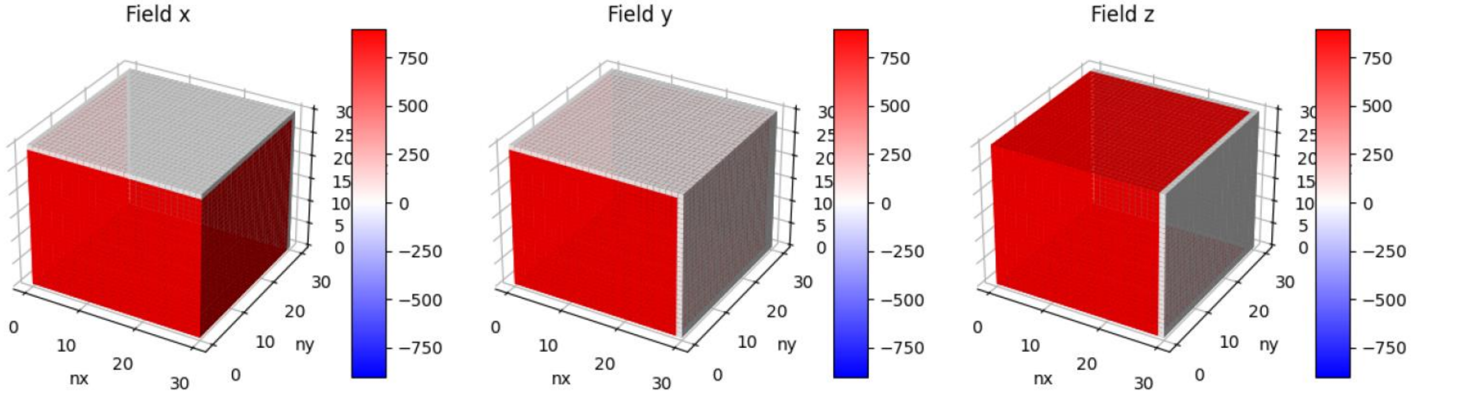
$$\widetilde{D}_L = diag(\tilde{L}_{x,0,0,0}, \dots, \tilde{L}_{x\,N_x,N_y,N_z}, \dots, \tilde{L}_{z,N_x,N_y,N_z})$$

3N x 3N



$$\widetilde{D}_A = diag(\widetilde{A}_{x,0,0,0}, \dots, \widetilde{A}_{x\,N_x,N_y,N_z}, \dots, \widetilde{A}_{z,N_x,N_y,N_z})$$

3N x 3N



**used built-in 3D plotting method

PyFIT [WSL: Ubuntu] - field.py

```python
def inspect3D(self, field='all', xmax=None, ymax=None, zmax=None, cmap='bwr', dpi=100, show=True, handles=False):
    """
    Voxel representation of a 3D array with matplotlib
    """
```

Primal grid G vs tilde grid $\widetilde{G}$ (III)

To use $\widetilde{\mathbf{D}}_{\mathbf{L}}, \widetilde{\mathbf{D}}_{\mathbf{A}}$ correctly, we need to change the ***update equations***:

$$\boldsymbol{h}^{n+1} = \boldsymbol{h}^{n} - \Delta t\, \boldsymbol{D}_{A}^{-1} \boldsymbol{D}_{\mu}^{-1} \boldsymbol{C} \boldsymbol{D}_{s} \boldsymbol{e}^{n+0.5}$$
$$\boldsymbol{e}^{n+1.5} = \boldsymbol{e}^{n+0.5} + \Delta t\, \widetilde{\boldsymbol{D}}_{A}^{-1} \widetilde{\boldsymbol{D}}_{\varepsilon} \widetilde{\boldsymbol{C}} \widetilde{\boldsymbol{D}}_{s} \boldsymbol{h}^{n} - \widetilde{\boldsymbol{D}}_{\varepsilon} \boldsymbol{j}^{n}$$

$$\boldsymbol{h}^{n+1} = \boldsymbol{h}^{n} - \Delta t\, \widetilde{\boldsymbol{D}}_{s} \boldsymbol{D}_{\mu}^{-1} \boldsymbol{D}_{A}^{-1} \boldsymbol{C} \boldsymbol{e}^{n+0.5}$$
$$\boldsymbol{e}^{n+1.5} = \boldsymbol{e}^{n+0.5} + \Delta t\, \boldsymbol{D}_{s} \widetilde{\boldsymbol{D}}_{\varepsilon} \widetilde{\boldsymbol{D}}_{A}^{-1} \widetilde{\boldsymbol{C}} \boldsymbol{h}^{n} - \widetilde{\boldsymbol{D}}_{\varepsilon} \boldsymbol{j}^{n}$$

- New update equations agree with the definition of the Material matrices given in [1]:

$$M_{\mu^{-1}} = \tilde{D}_L D_{\mu^{-1}} D_A^{-1}$$
$$M_{\varepsilon^{-1}} = (\tilde{D}_A D_\varepsilon D_L)^{-1}$$

$\boldsymbol{M}_{\mu^{-1}}$ ($\widetilde{\boldsymbol{D}}_{s}$) acts on $\boldsymbol{E}$ setting longitudinal components to zero (ghosts) and $\boldsymbol{M}_{\varepsilon^{-1}}$ ($\widetilde{\boldsymbol{D}}_{A}^{-1}$) acts on $\boldsymbol{H}$, setting transverse components to zero.

*[1]: Klaus Klopfer PhD thesis [2014]

PEC, PMC & Periodic boundaries (I)

PyFIT [WSL: Ubuntu] - solverFIT3D.py

```
def apply_bc_to_C(self):
    '''
    Modifies rows or columns of C and tDs and itDa matrices
    according to bc_low and bc_high
    '''
    xlo, ylo, zlo = 1., 1., 1.
    xhi, yhi, zhi = 1., 1., 1.

    # Perodic: out == in
    if any(True for x in self.bc_low if x.lower() == 'periodic'):
        if self.bc_low[0].lower() == 'periodic' and self.bc_high[0].lower() == 'periodic':
            self.tL[-1, :, :, 'x'] = self.L[0, :, :, 'x']
            self.itA[-1, :, :, 'y'] = self.iA[0, :, :, 'y']
            self.itA[-1, :, :, 'z'] = self.iA[0, :, :, 'z']

        elif self.bc_low[1].lower() == 'periodic' and self.bc_high[1].lower() == 'periodic':
            self.tL[:, -1, :, 'y'] = self.L[:, 0, :, 'y']
            self.itA[:, -1, :, 'x'] = self.iA[:, 0, :, 'x']
            self.itA[:, -1, :, 'z'] = self.iA[:, 0, :, 'z']

        elif self.bc_low[2].lower() == 'periodic' and self.bc_high[2].lower() == 'periodic':
            self.tL[:, :, -1, 'z'] = self.L[:, :, 0, 'z']
            self.itA[:, :, -1, 'x'] = self.iA[:, :, 0, 'x']
            self.itA[:, :, -1, 'y'] = self.iA[:, :, 0, 'y']
        else:
            raise Exception('Invalid use of periodic boundary condiions')

        self.tDs = diags(self.tL.toarray(), shape=(3*self.N, 3*self.N), dtype=float)
        self.itDa = diags(self.itA.toarray(), shape=(3*self.N, 3*self.N), dtype=float)
```

PERIODIC BCs

- Before we considered tilde matrices $\tilde{\boldsymbol{D}}_{\boldsymbol{L}}, \tilde{\boldsymbol{D}}_{\boldsymbol{A}}$ equal to primal matrices $\boldsymbol{D}_{\boldsymbol{L}}, \boldsymbol{D}_{\boldsymbol{A}}$, periodic boundaries where naturally implemented.
- With the correct values, the *naturally implemented BC's are PMC (transverse H = 0).*
- We implement **periodic boundaries** by modifying $\tilde{\mathbf{D}}_{\mathbf{L}}, \tilde{\mathbf{D}}_{\mathbf{A}}$

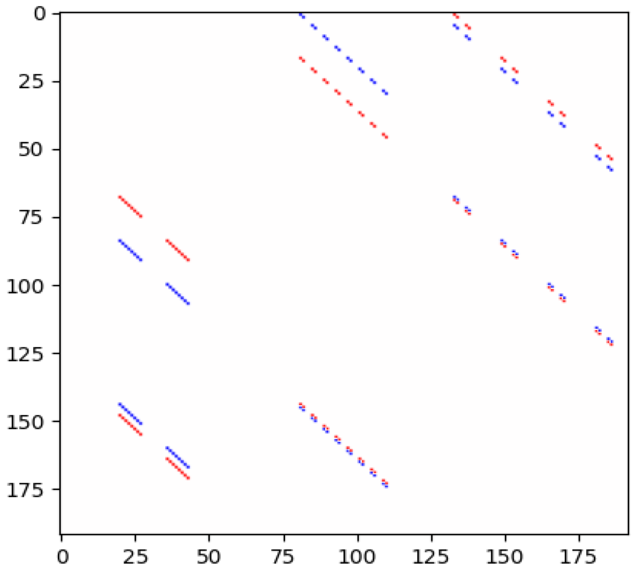
PEC, PMC & Periodic boundaries (II)

PyFIT [WSL: Ubuntu] - solverFIT3D.py

```
# Dirichlet PEC: tangential E field = 0 at boundary
if any(True for x in self.bc_low if x.lower() == 'electric' or x.lower() == 'pec'):

    if self.bc_low[0].lower() == 'electric' or self.bc_low[0].lower() == 'pec':
        xlo = 0
    if self.bc_low[1].lower() == 'electric' or self.bc_low[1].lower() == 'pec':
        ylo = 0
    if self.bc_low[2].lower() == 'electric' or self.bc_low[2].lower() == 'pec':
        zlo = 0
    if self.bc_high[0].lower() == 'electric' or self.bc_high[0].lower() == 'pec':
        xhi = 0
    if self.bc_high[1].lower() == 'electric' or self.bc_high[1].lower() == 'pec':
        yhi = 0
    if self.bc_high[2].lower() == 'electric' or self.bc_high[2].lower() == 'pec':
        zhi = 0

    # Assemble matrix
    self.BC = Field(self.Nx, self.Ny, self.Nz, dtype=np.int8, use_ones=True)

    for d in ['x', 'y', 'z']: #tangential to zero
        if d != 'x':
            self.BC[0, :, :, d] = xlo
            self.BC[-1, :, :, d] = xhi
        if d != 'y':
            self.BC[:, 0, :, d] = ylo
            self.BC[:, -1, :, d] = yhi
        if d != 'z':
            self.BC[:, :, 0, d] = zlo
            self.BC[:, :, -1, d] = zhi

    self.Dbc = diags(self.BC.toarray(),
                    shape=(3*self.N, 3*self.N),
                    dtype=np.int8
                    )

    # Update C (columns)
    self.C = self.C*self.Dbc
```

PEC BCs

- We modify the columns of matrix C by constructing an **auxiliary diagonal matrix Dbc** that will contain **zeros** where the transverse components of the E field should be zero.
- To modify the columns, we multiply this matrix Dbc *after* C



Sparsity of the C matrix after applying PEC BC's in all directions.
Domain with N = 4x4x4.
Size of C = 3N x 3N

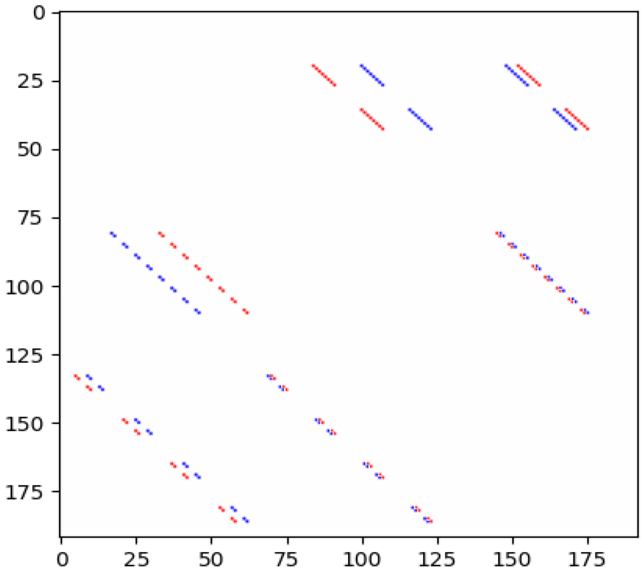
PEC, PMC & Periodic boundaries (III)

PyFIT [WSL: Ubuntu] - solverFIT3D.py

```python
# Dirichlet PMC: tangential H field = 0 at boundary
if any(True for x in self.bc_low if x.lower() == 'magnetic' or x.lower() == 'pmc'):

    if self.bc_low[0].lower() == 'magnetic' or self.bc_low[1] == 'pmc':
        xlo = 0
    if self.bc_low[1].lower() == 'magnetic' or self.bc_low[1] == 'pmc':
        ylo = 0
    if self.bc_low[2].lower() == 'magnetic' or self.bc_low[2] == 'pmc':
        zlo = 0
    if self.bc_high[0].lower() == 'magnetic' or self.bc_high[0] == 'pmc':
        xhi = 0
    if self.bc_high[1].lower() == 'magnetic' or self.bc_high[1] == 'pmc':
        yhi = 0
    if self.bc_high[2].lower() == 'magnetic' or self.bc_high[2] == 'pmc':
        zhi = 0

    # Assemble matrix
    self.BC = Field(self.Nx, self.Ny, self.Nz, dtype=np.int8, use_ones=True)

    for d in ['x', 'y', 'z']: #tangential to zero
        if d != 'x':
            self.BC[0, :, :, d] = xlo
            self.BC[-1, :, :, d] = xhi
        if d != 'y':
            self.BC[:, 0, :, d] = ylo
            self.BC[:, -1, :, d] = yhi
        if d != 'z':
            self.BC[:, :, 0, d] = zlo
            self.BC[:, :, -1, d] = zhi

    self.Dbc = diags(self.BC.toarray(),
                    shape=(3*self.N, 3*self.N),
                    dtype=np.int8
                    )

    # Update C (rows)
    self.C = self.Dbc*self.C
```

PMC BCs

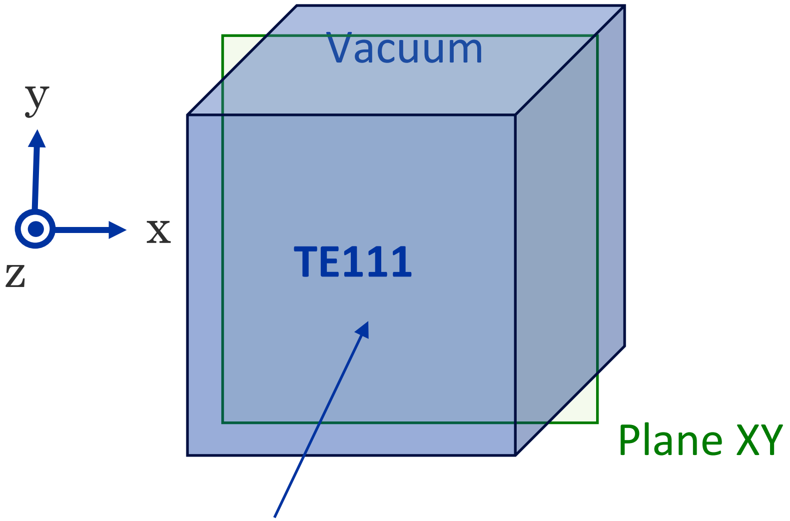
- We use the same Dbc matrix, this time we need to modify the rows of C (columns of $\tilde{C}$)
- To modify the columns, we multiply this matrix Dbc *before* C



Sparsity of the C matrix after applying PMC BC's in all directions.
Domain with N = 4x4x4.
Size of C = 3N x 3N

Resonator test: PEC BC’s

All PEC BCs

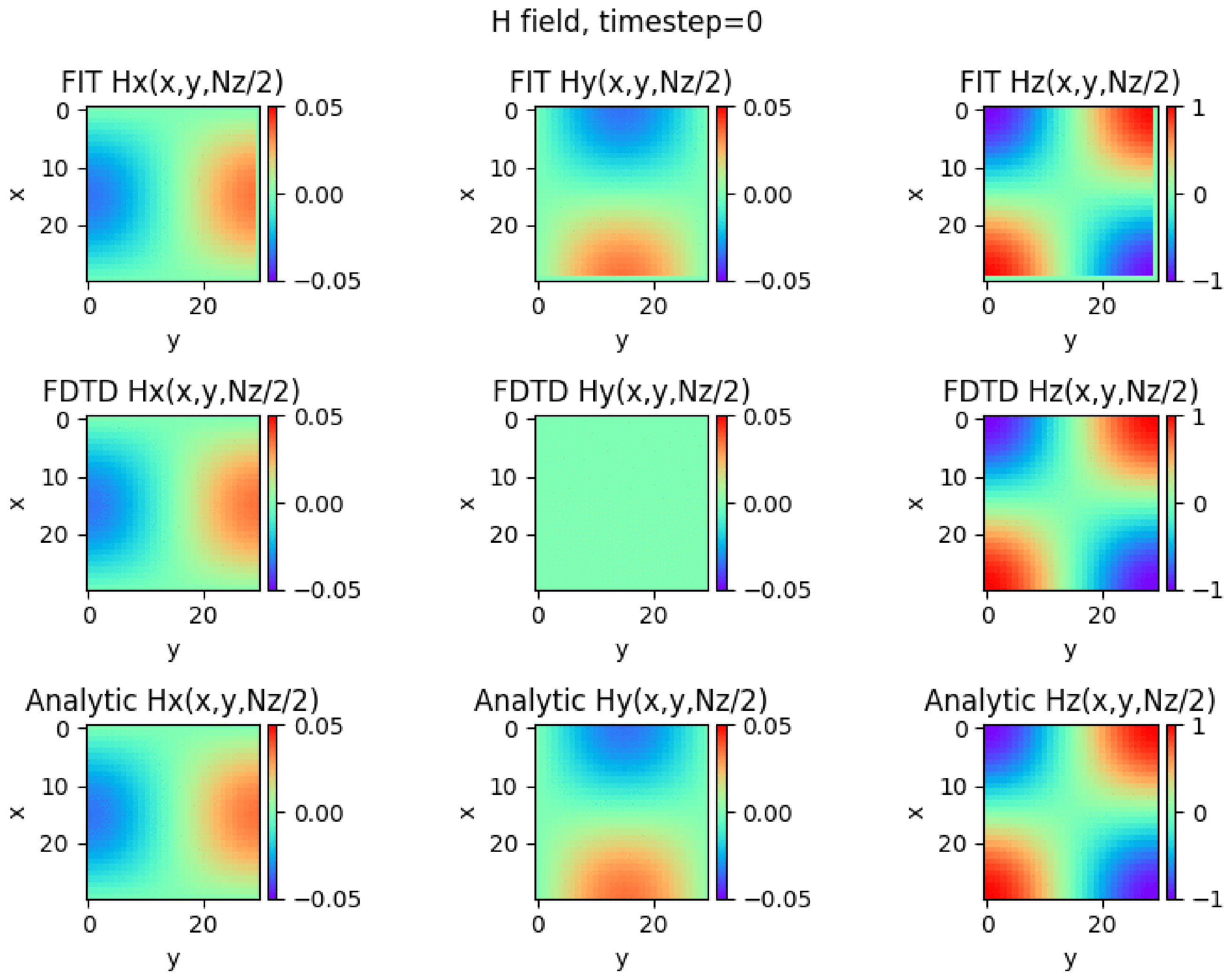


Initial condition: the 3D analytic solution of a cubic resonator mode TE111 for H_x, H_y, H_z

**special routine created for 3D IC, applied before 1st timestep

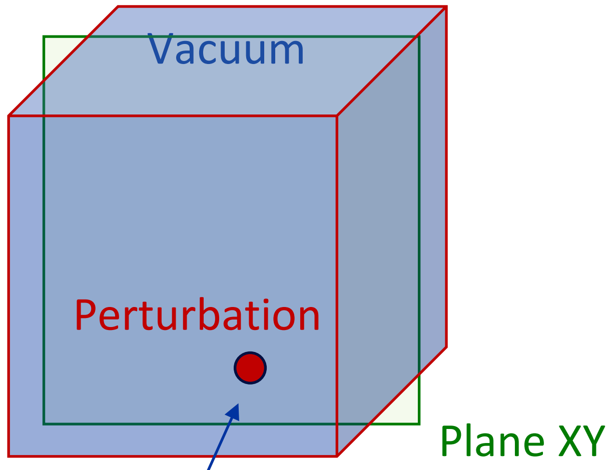
PyFIT [WSL: Ubuntu] - solverFIT3D.py

```
def set_ghosts_to_0(self):
        '''
        Cleanup for initial conditions if they are
        accidentally applied to the ghost cells
        '''
```

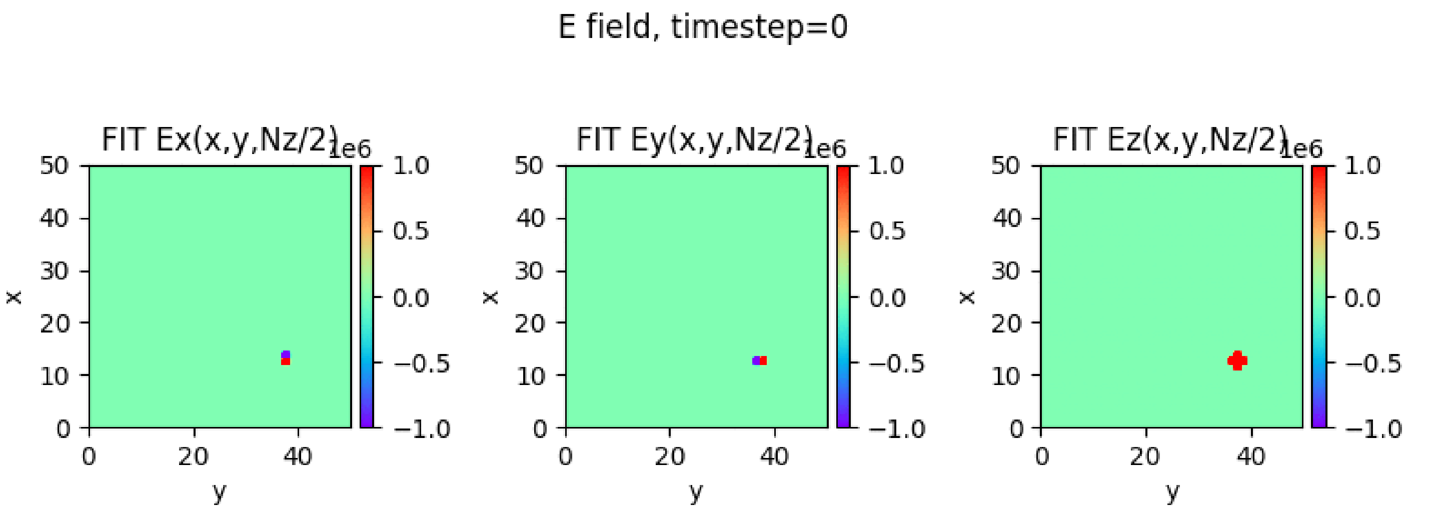


Perturbation test: Periodic BC's

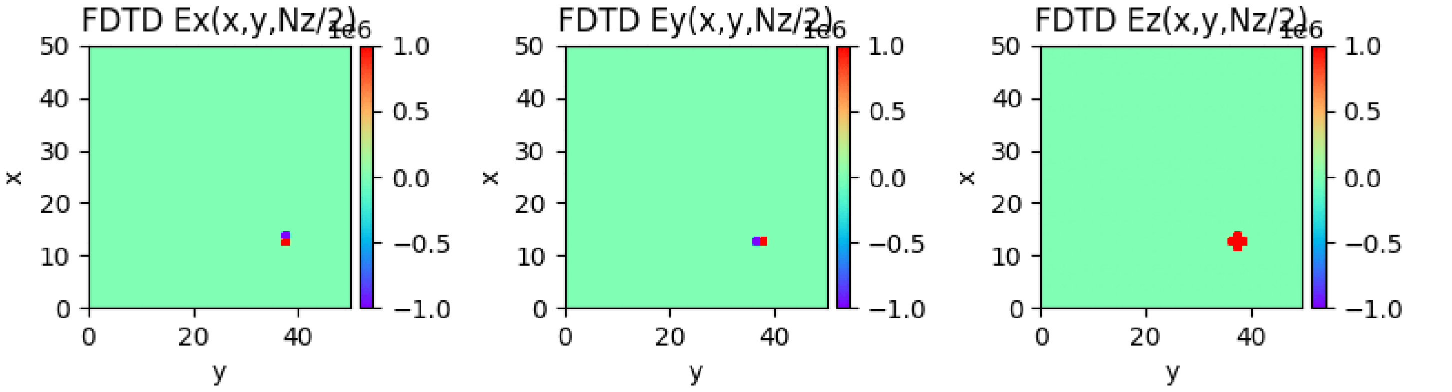
All Periodic BCs



Perturbation at $\frac{3}{4}N_x, \frac{3}{4}N_y, \frac{1}{2}N_z$

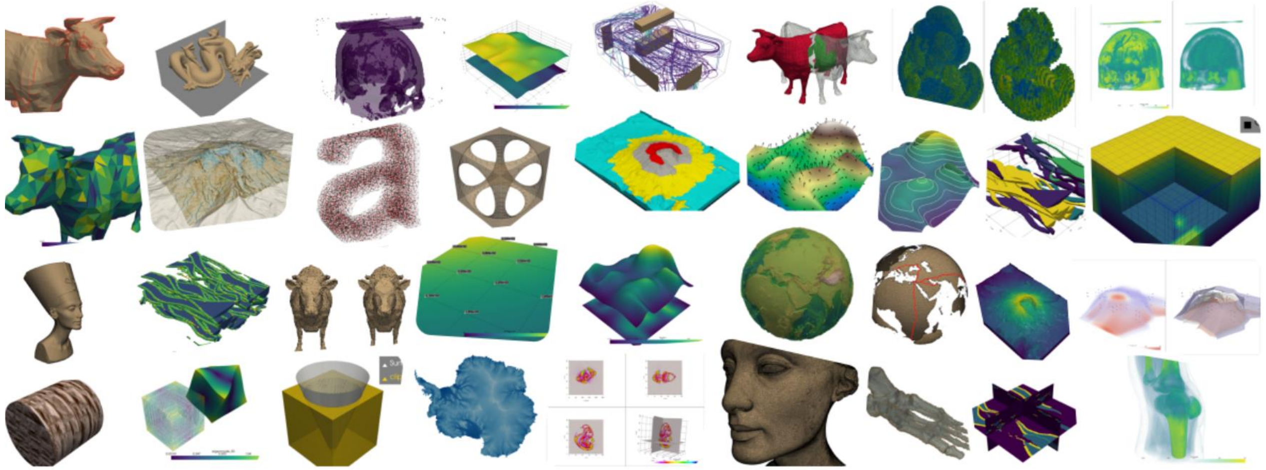


***FDTD does not have Periodic implemented



STL importer with PyVista

3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK)

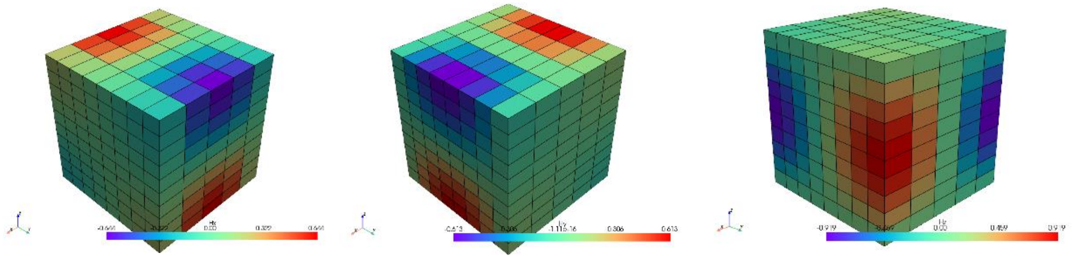


https://pyvista.org/

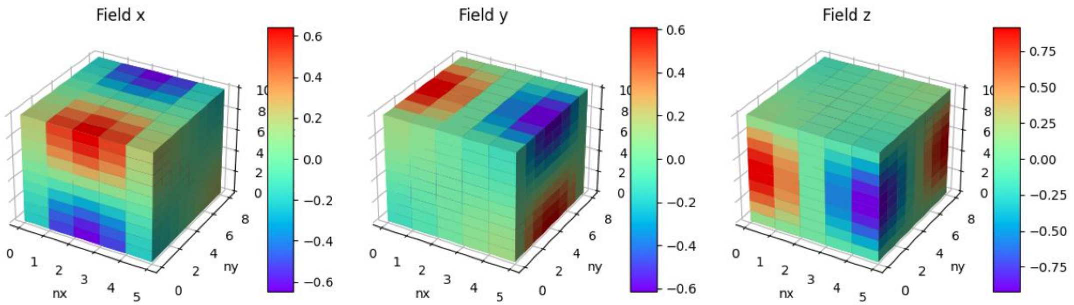
Docs: https://docs.pyvista.org/version/stable/#status

Vtk is x1000 faster than matplotlib for 3D plotting

- PyVista (0s) for 400 cells



- Matplotlib voxel (2' 30s) for 400 cells



STL importer with PyVista (II)

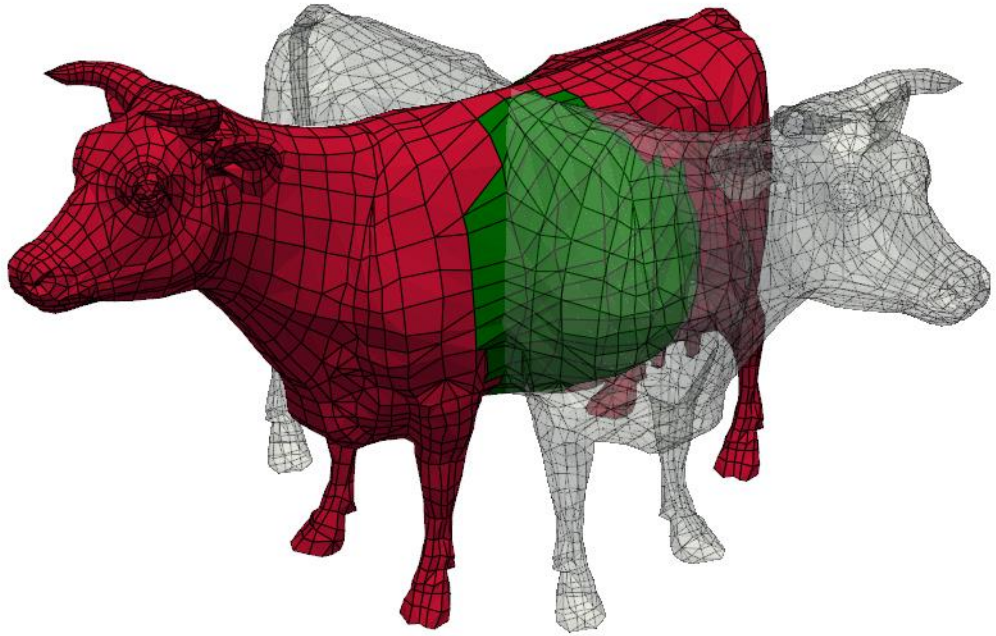
Extract Cells Inside Surface

Extract the cells in a mesh that exist inside or outside a closed surface of another mesh

```
import pyvista as pv
from pyvista import examples

mesh = examples.download_cow()

cpos = [(13.0, 7.6, -13.85), (0.44, -0.4, -0.37), (-0.28, 0.9, 0.3)]

dargs = dict(show_edges=True)
# Rotate the mesh to have a second mesh
rot = mesh.rotate_y(90, inplace=False)

p = pv.Plotter()
p.add_mesh(mesh, color="Crimson", **dargs)
p.add_mesh(rot, color="mintcream", opacity=0.35, **dargs)
p.camera_position = cpos
p.show()
```

Mark points inside with 1 and outside with a 0

```
select = mesh.select_enclosed_points(rot)

select
```



STL importer with PyVista (III)

```
# Plot
pl = pv.Plotter()
pl.add_mesh(grid, show_edges=True, style='wireframe', color='w', opacity=0.15)
pl.add_mesh(grid.extract_cells(cells_inside), scalars='Solid1', cmap='Blues', opacity=0.6)

pl.show()
```

Import Goniometer stl

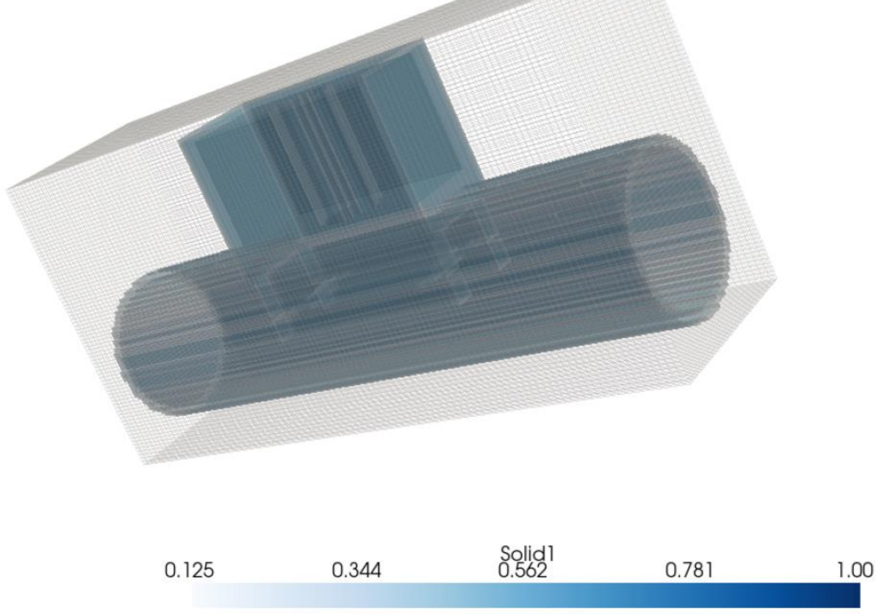
PyFIT [WSL: Ubuntu] - test_stl.py

```
# --- Read stl ----
surf = pv.read('goniometer.stl')
surf = surf.rotate_x(90)    # z axis longitudinal
surf = surf.scale(unit)             # [m]
#surf = surf.subdivide(3, subfilter='linear') #if used, select.threshold() is empty

# --- Domain definition ----

# bounds
xmin, xmax, ymin, ymax, zmin, zmax = surf.bounds
pad = 1.0 * unit

# n cells
Nx = 30*2
Ny = 40*2
Nz = 50*2
N = Nx*Ny*Nz

# cell vertex
x = np.linspace(xmin - pad, xmax + pad, Nx + 1)
y = np.linspace(ymin - pad, ymax + pad, Ny + 1)
z = np.linspace(zmin - pad, zmax + pad, Nz + 1)

# grid
Z, Y, X = np.meshgrid(z, y, x, indexing='ij')
grid = pv.StructuredGrid(X, Y, Z)

```

PyFIT [WSL: Ubuntu] - test_stl.py

```
# ---- Cells inside surface ----
tol = unit*1.e-3
select = grid.select_enclosed_points(surf, tolerance=tol)
points_inside = np.where(select['SelectedPoints'] > 0.1)[0]
cells_inside = np.where(select.point_data_to_cell_data()['SelectedPoints'] > 0.1)[0]

grid['Solid1'] = select.point_data_to_cell_data()['SelectedPoints']
```

- Runtime (2s) for 480.000 cells, included plotting



Adding STL and materials to FIT

PyFIT [WSL: Ubuntu] - gridFIT3D.py

```python
import numpy as np
import pyvista as pv

from field import Field


class GridFIT3D:
    """
    Class holding the grid information and
    stl importing handling using PyVista

    Parameters
    ----------
    xmin, xmax, ymin, ymax, zmin, zmax: float
        extent of the domain.
    Nx, Ny, Nz: int
        number of cells per direction
    stl_solids: dict, optional
        stl files to import in the domain.
        {'Solid 1': stl_1, 'Solid 2': stl_2, ...}
        If stl files are not in the same folder,
        add the path to the file name.
    stl_materials: dict, optional
        Material properties associated with stl
        {'Solid 1': [eps1, mu1],
         'Solid 2': [eps1, mu1],
         ...}
    stl_rotate: list or dict, optional
        Angle of rotation to apply to the stl models: [rot_x, rot_y, rot_z]
        - if list, it will be applied to all stls in `stl_solids`
        - if dict, it must contain the same keys as `stl_solids`,
          indicating the rotation angle per stl
    stl_scale: float or dict, optional
        Scaling value to apply to the stl model to convert to [m]
        - if float, it will be applied to all stl in `stl_solids`
        - if dict, it must contain the same keys as `stl_solids`

    """

    def __init__(self, xmin, xmax, ymin, ymax, zmin, zmax,
                 Nx, Ny, Nz, stl_solids=None, stl_materials=None,
                 stl_rotate=[0., 0., 0.], stl_translate=[0., 0., 0.], stl_scale=0.):
```

PyFIT [WSL: Ubuntu] - gridFIT3D.py

```python
def mark_cells_in_stl(self):

    if type(self.stl_solids) is not dict:
        if type(self.stl_solids) is str:
            self.stl_solids = {'Solid 1' : self.stl_solids}
        else:
            raise Exception('Attribute `stl_solids` must contain a string or a dictionary')

    if type(self.stl_rotate) is not dict:
        # if not a dict, the same values will be applied to all solids
        stl_rotate = {}
        for key in self.stl_solids.keys():
            stl_rotate[key] = self.stl_rotate
        self.stl_rotate = stl_rotate

    if type(self.stl_scale) is not dict:
        # if not a dict, the same values will be applied to all solids
        stl_scale = {}
        for key in self.stl_solids.keys():
            stl_scale[key] = self.stl_scale
        self.stl_scale = stl_scale

    if type(self.stl_translate) is not dict:
        # if not a dict, the same values will be applied to all solids
        stl_translate = {}
        for key in self.stl_solids.keys():
            stl_translate[key] = self.stl_translate
        self.stl_translate = stl_translate

    tol = np.min([self.dx, self.dy, self.dz])*1e-3
    for key in self.stl_solids.keys():

        # import stl
        surf = pv.read(self.stl_solids[key])

        # rotate
        surf = surf.rotate_x(self.stl_rotate[key][0])
        surf = surf.rotate_y(self.stl_rotate[key][1])
        surf = surf.rotate_z(self.stl_rotate[key][2])

        # translate
        surf.translate(self.stl_translate[key])

        # scale
        surf = surf.scale(self.stl_scale[key])

        # mark cells in stl [True == in stl, False == out stl]
        select = self.grid.select_enclosed_points(surf, tolerance=tol)
        self.grid[key] = select.point_data_to_cell_data()['SelectedPoints'] > tol
```

New grid 'GridFIT3D' class:

- Creates grid with a **PyVista Structured Grid** object
- Computes **lengths and areas** for primal and tilde grid diagonal matrices
- **Imports stl** by marking the cells inside the (closed) surface (True/False)
- *Staircased* for now, but potential for pixel smoothing!

Adding STL and materials to FIT (II)

PyFIT [WSL: Ubuntu] - solverFIT3D.py

```python
import numpy as np
from scipy.constants import c as c_light, epsilon_0 as eps_0, mu_0 as mu_0
from scipy.sparse import csc_matrix as sparse_mat
from scipy.sparse import diags, block_diag, hstack, vstack
from scipy.sparse.linalg import inv

from field import Field
from materials import material_lib

class SolverFIT3D:

    def __init__(self, grid, cfln=0.5,
                 bc_low=['Periodic', 'Periodic', 'Periodic'],
                 bc_high=['Periodic', 'Periodic', 'Periodic'],
                 use_conductors=True, use_stl=False,
                 i_s=0, j_s=0, k_s=0, N_pml_low=None, N_pml_high=None):
```

PyFIT [WSL: Ubuntu] - materials.py

```python
'''
Material library dictionary

Format:
{
    'material key' : [eps_r, mu_r],
}

* 'material key' in lower case only
* eps = eps_r*eps_0 and mu = mu_r*mu_0
'''
import numpy as np
from scipy.constants import c as c_light, epsilon_0 as eps_0, mu_0 as mu_0

material_lib = {
    'pec' : [np.inf, 1.],
    'vacuum' : [1.0, 1.0],
    'dielectric' : [10., 1.0],

}
```

PyFIT [WSL: Ubuntu] - solverFIT3D.py

```python
def apply_stl_materials(self):
    '''
    Mask the cells inside the stl and assing the material
    defined by the user

    * Note: stl material should contain **relative** epsilon and mu
    ** Note 2: when assigning the stl material, the default values
               1./eps_0 and 1./mu_0 are substracted
    '''
    grid = self.grid.grid
    self.stl_solids = self.grid.stl_solids
    self.stl_materials = self.grid.stl_materials

    for key in self.stl_solids.keys():

        mask = np.reshape(grid[key], (self.Nx, self.Ny, self.Nz)).astype(int)

        if type(self.stl_materials[key]) is str:
            # Retrieve from material library
            mat_key = self.stl_materials[key].lower()

            eps = material_lib[mat_key][0]*eps_0
            mu = material_lib[mat_key][1]*mu_0

            # Setting to zero
            self.ieps += self.ieps * (-1.0*mask)
            self.imu += self.imu * (-1.0*mask)

            # Adding new values
            self.ieps += mask * 1./eps
            self.imu += mask * 1./mu

        else:
            # From input
            eps = self.stl_materials[key][0]*eps_0
            mu = self.stl_materials[key][1]*mu_0

            # Setting to zero
            self.ieps += self.ieps * (-1.0*mask)
            self.imu += self.imu * (-1.0*mask)

            # Adding new values
            self.ieps += mask * 1./eps
            self.imu += mask * 1./mu
```

- ‘SolverFIT3D’ class **supports EB from stl and from conductors**, through a flag
- **Materials** are applied by modifying the material Fields $ieps, imu$, using the mask calculated by PyVista
- **Every stl imported is applied in order**: last solid will overwrite the previous ones.

Outline

1. Where are we?

2. Main improvements

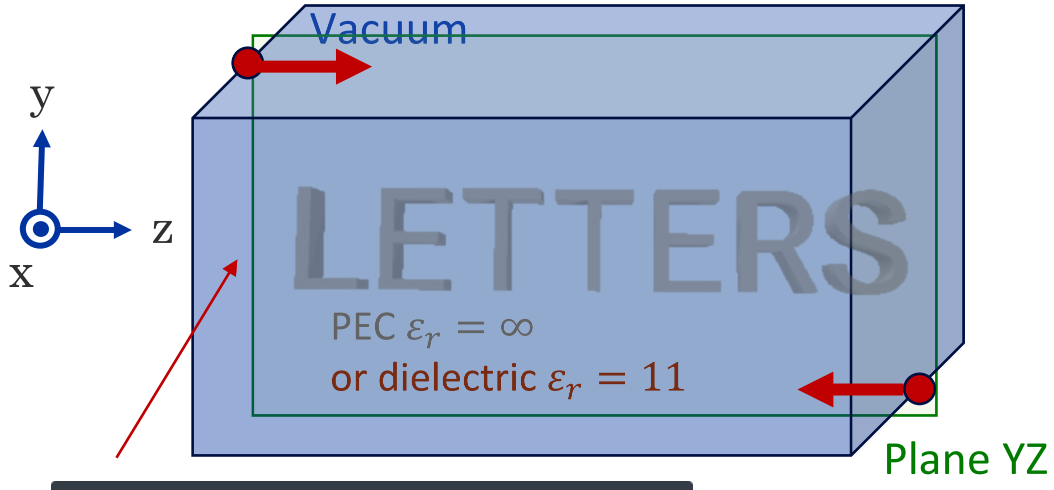
3. Some (fun) examples

4. Feedback from University

5. Conclusions & Next steps

Example(s) I: 1 stl solids made of different materials

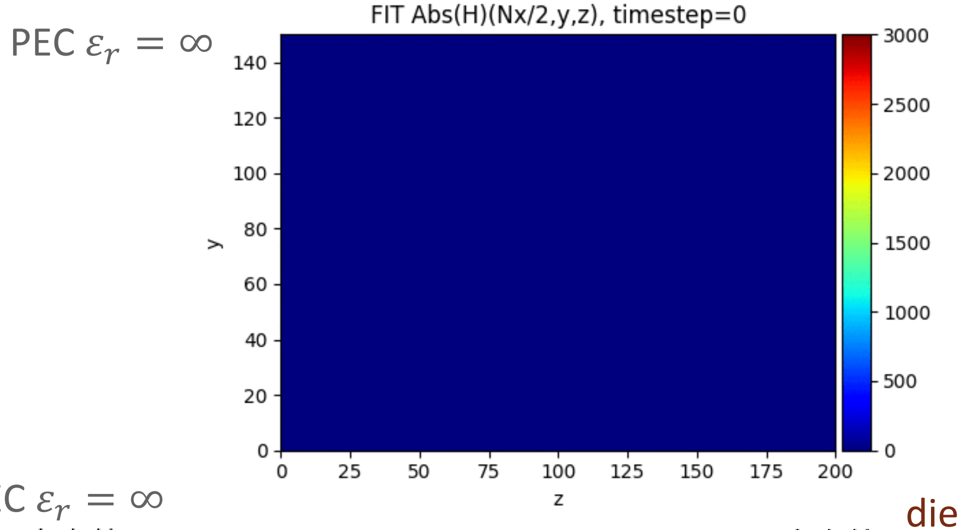
All PEC BCs

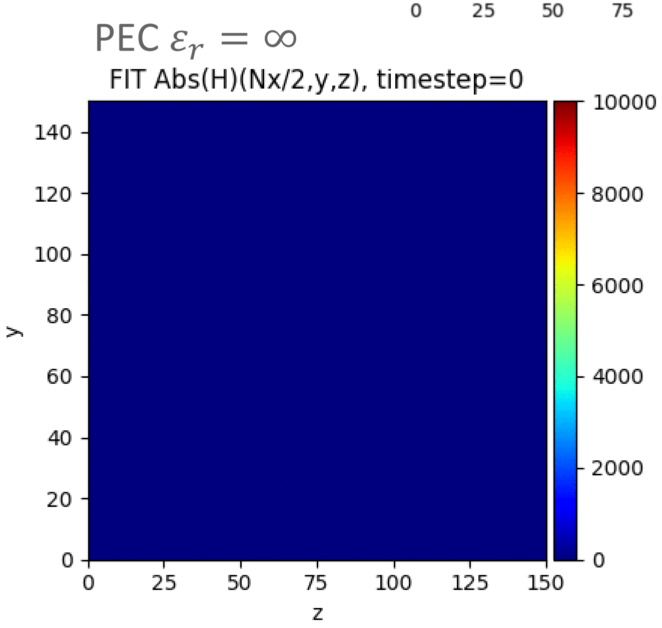


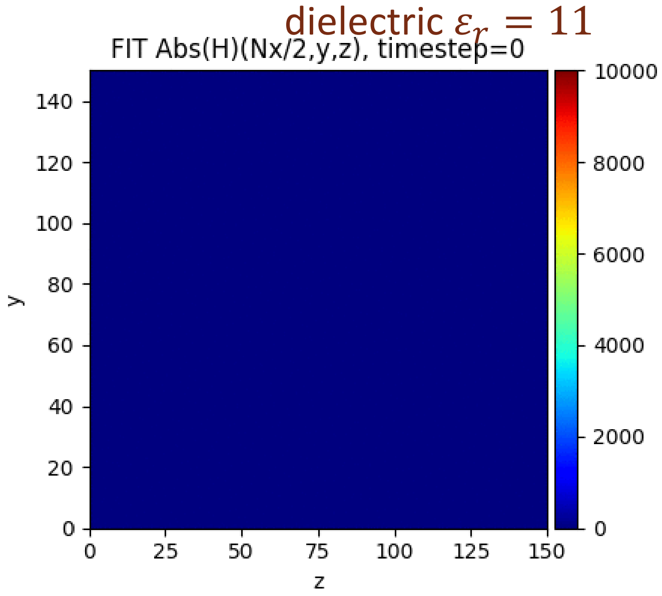
PyFIT [WSL: Ubuntu] - script_letters_fit.py

```
# ----------- Time loop ---------------

Nt = 600
for n in tqdm(range(Nt)):

    # Initial conditions
    zpos = int(Nz*n/(200))
    if zpos < Nz:
        solver.J[int(Nx/2), Ny-10, zpos, 'z'] = 1.0*c_light
        solver.J[int(Nx/2), 10, Nz-1-zpos, 'z'] = 1.0*c_light

    # Advance
    solver.one_step()

    if zpos < Nz:
        solver.J[int(Nx/2), Ny-10, zpos, 'z'] = 0.0
        solver.J[int(Nx/2), 10, Nz-1-zpos, 'z'] = 0.

    # Plot
    if n%5 == 0:
        plot_E_field(solver, n)
        plot_H_field(solver, n)
```

'Moving' perturbations

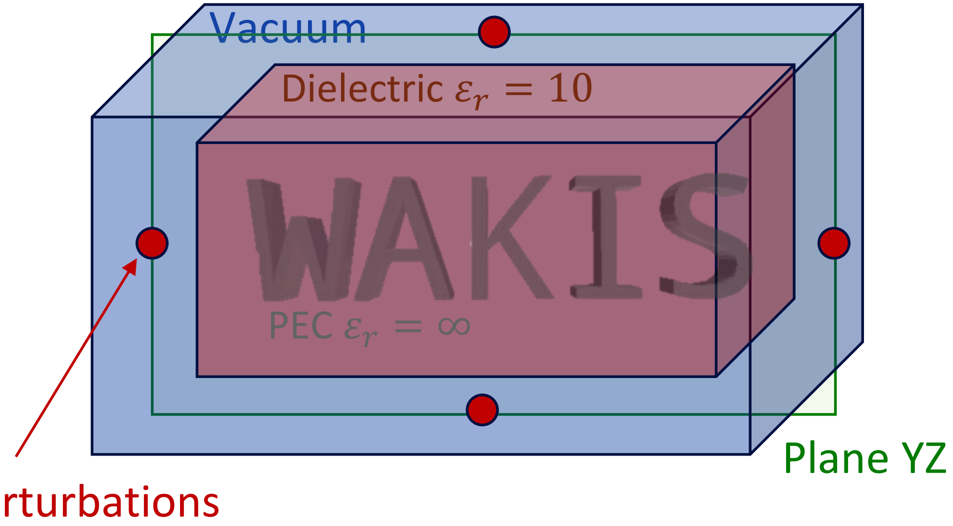






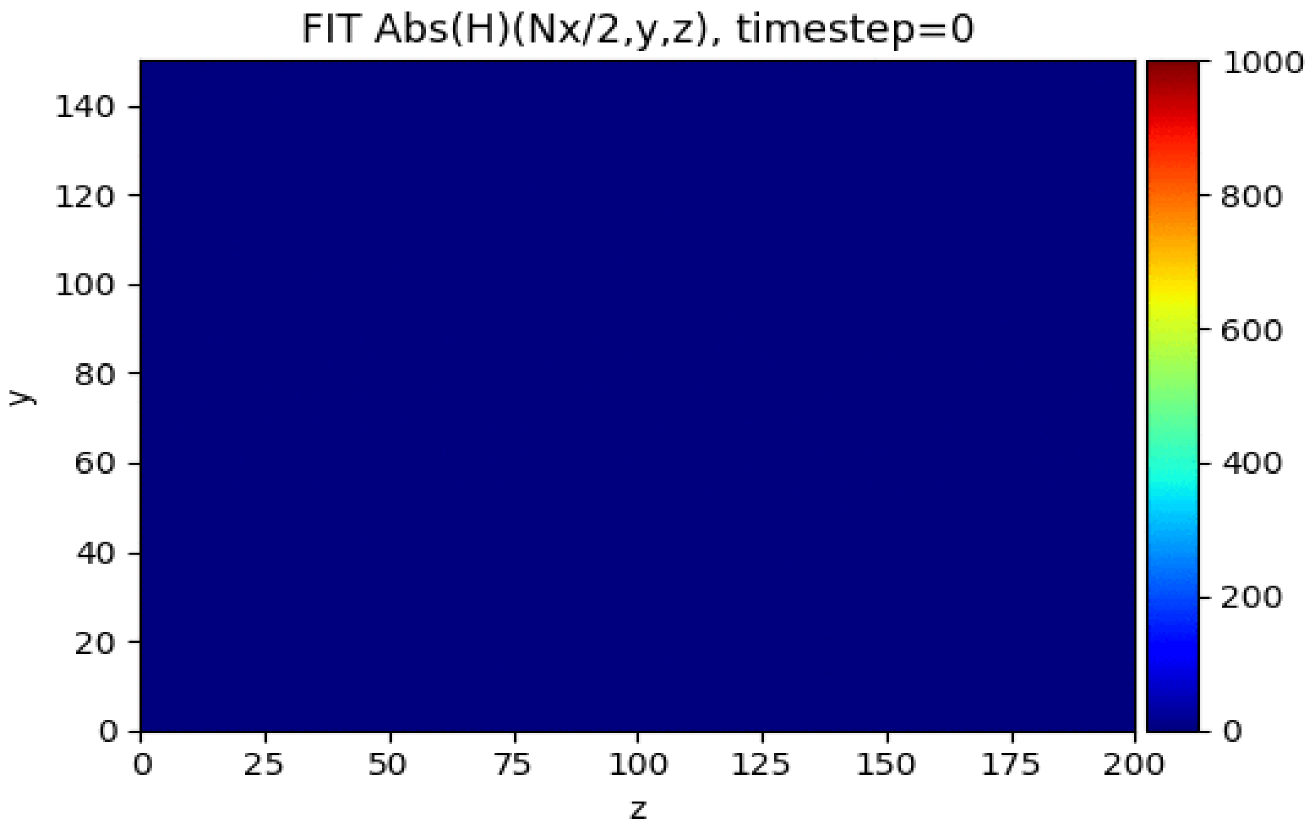
Example II: 2 stl solids made of different materials

All PEC BCs




```
solver.J[int(Nx/2), int(Ny/2), 0, 'z'] = 1.0*c_light
solver.J[int(Nx/2), int(Ny/2), Nz-2, 'z'] = 1.0*c_light
solver.J[int(Nx/2), 1, int(Nz/2), 'z'] = 1.0*c_light
solver.J[int(Nx/2), Ny-2, int(Nz/2), 'z'] = 1.0*c_light
```

https://github.com/elenafuengar/PyFIT/blob/main/examples/script_2solids_fit.py



Ncells = 900,000

Runtime on 8Gb Intel i5-8500 single core: 8 timesteps/s: 2’ 24’’

Outline

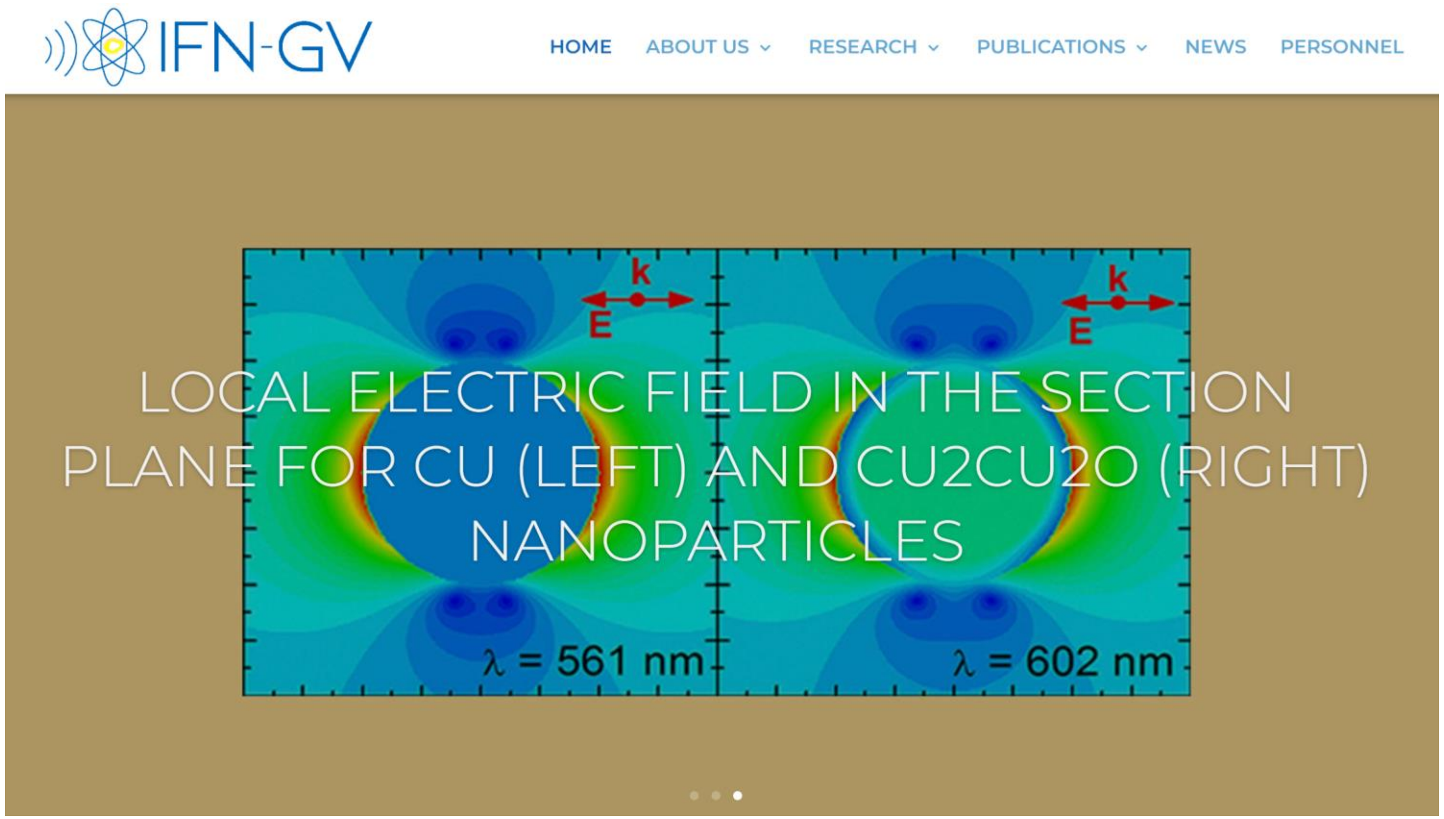
1. Where are we?
2. Main improvements
3. Some (fun) examples
4. **Feedback from University**
5. Conclusions & Next steps

Polytechnic University of Madrid (UPM)

PhD Supervisors: Manuel Cotelo, Eduardo Oliva
Department name: Instituto de Fusion Nuclear Guillermo-Velarde*
Department director: Pedro Velarde

Manuel Cotelo

Eduardo Oliva



Feedback: roadmap until next visit

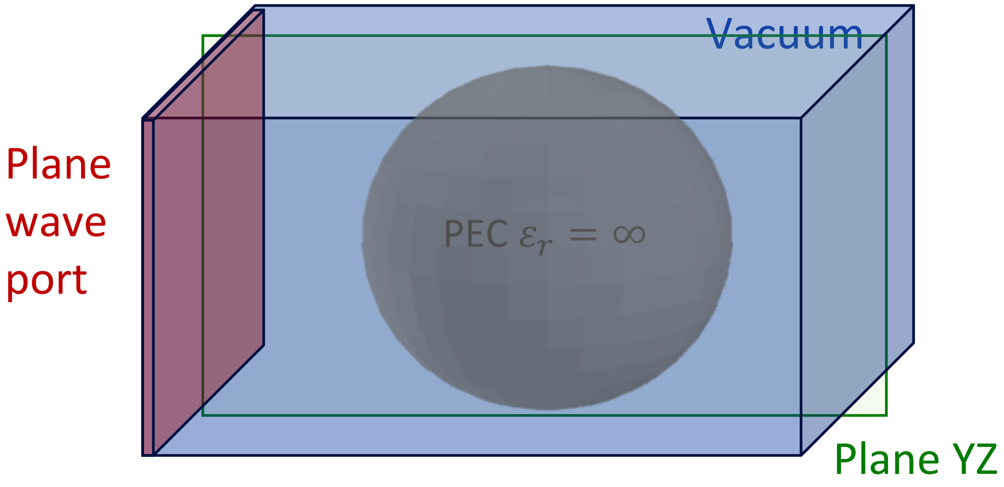
- **Consistency checks:**
  - *Plane wave tests*: Check speed simulation = speed of light. Check frequency at t=0 and t=100 (FFT) to see variation of the main frequency. It can be a 2D plot t vs f. Check interaction with boundaries PEC/Periodic.
  - *Gaussian wave packet tests*: Quantify numerical dispersion of the code -> let gaussian packet propagate and measure the change in sigma. It should increase due to numerical dispersion

- **Next steps (UPM):**
  - Try OUTFLOW absorbing boundary condition (AMReX) instead of PML
  - Further checks on BCs: quantitative measure of reflection, periodicity
  - Divergence correction for current sources is needed?

    i.e., enforce Gauss law: $\nabla \cdot E = \rho / \varepsilon$
  - User defined, time-dependent Dirichlet BC's to input plane waves (can be though as EM ports)
  - Create a class to manage Sources
  - Unit tests with quantitative values for each feature of the code

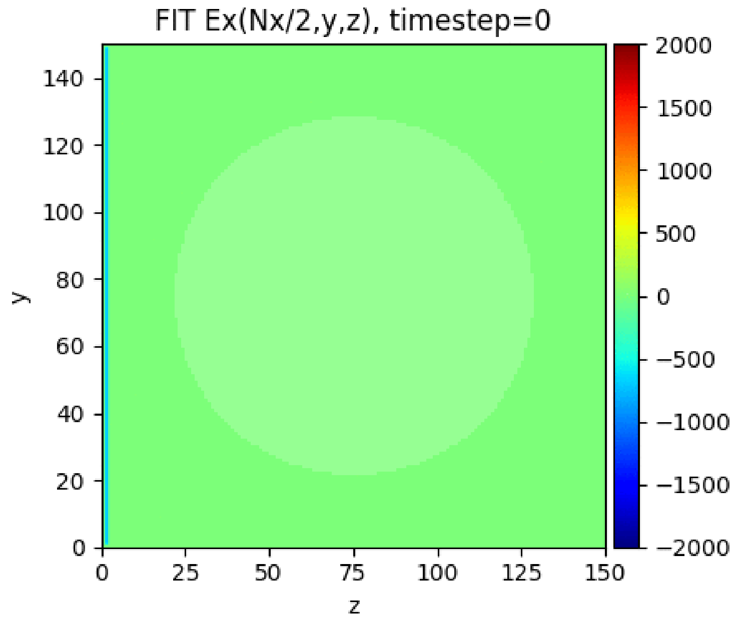
First attempts with Plane wave

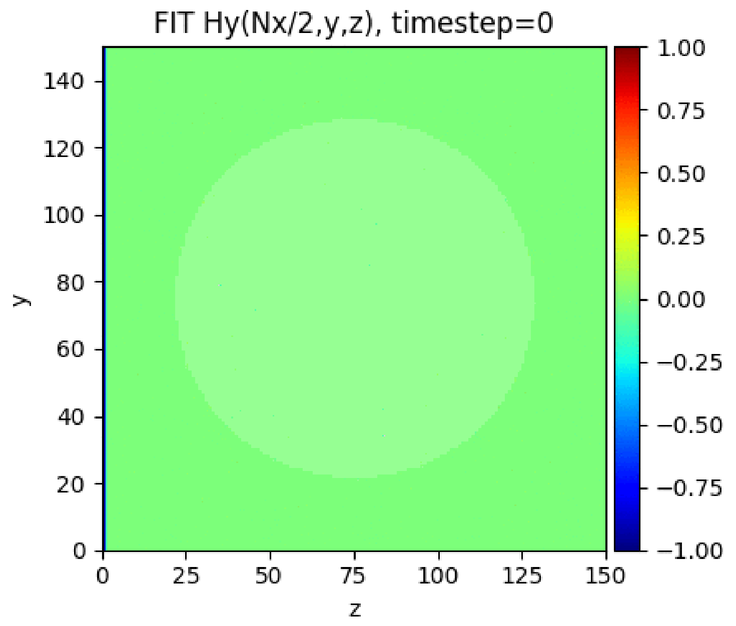
All PEC BCs



PyFIT [WSL: Ubuntu] - script_planewave_fit.py

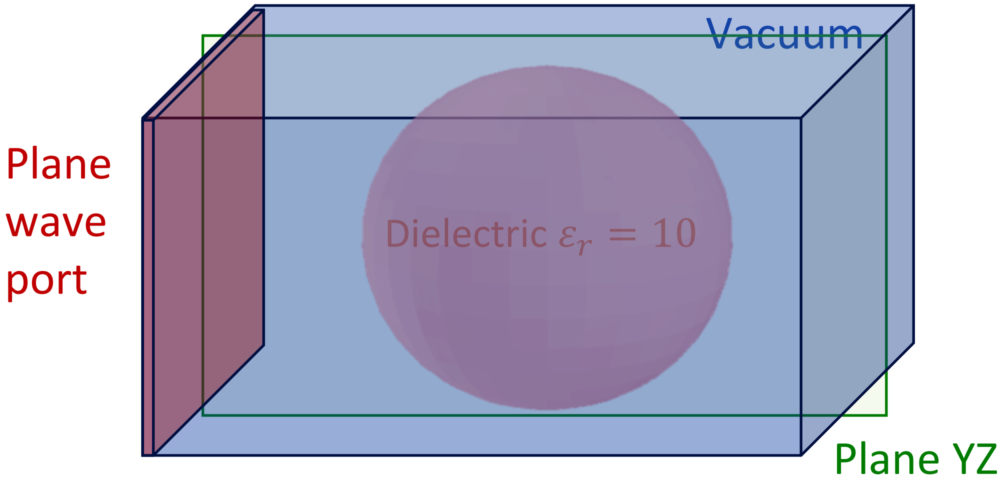
```python
# Initial conditions
def plane_wave(solver,t, Nt,f=None, beta=1.0):

    if f is None:
        f = 15 * 1/(solver.dt*(Nt-1))    # 15 nodes
    vp = beta*c_light         # wavefront velocity beta*c
    w = 2*np.pi*f             # ang. frequency
    kz = w/c_light            # wave number

    solver.H[:,:,0,'y'] = -1.0 * np.cos(w*t)
    solver.E[:,:,0,'x'] = 1.0 * np.cos(w*t) /(kz/(mu_0*vp))
```





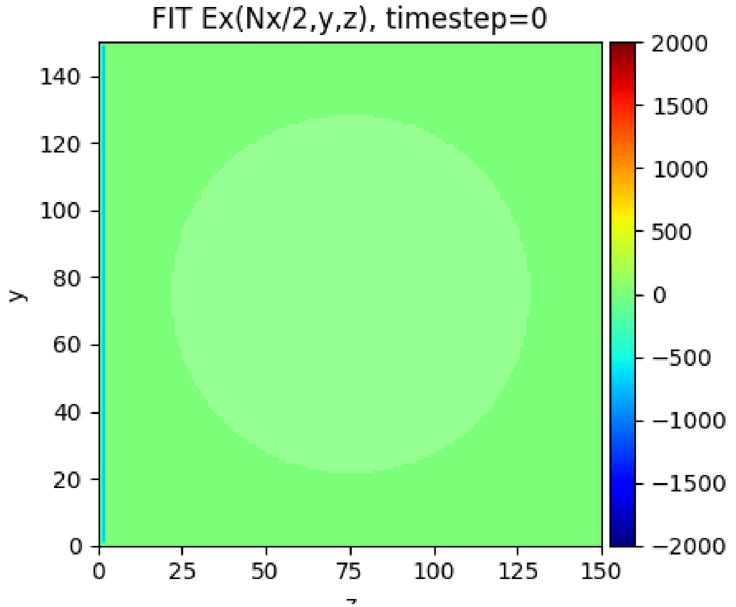
First attempts with Plane wave (II)

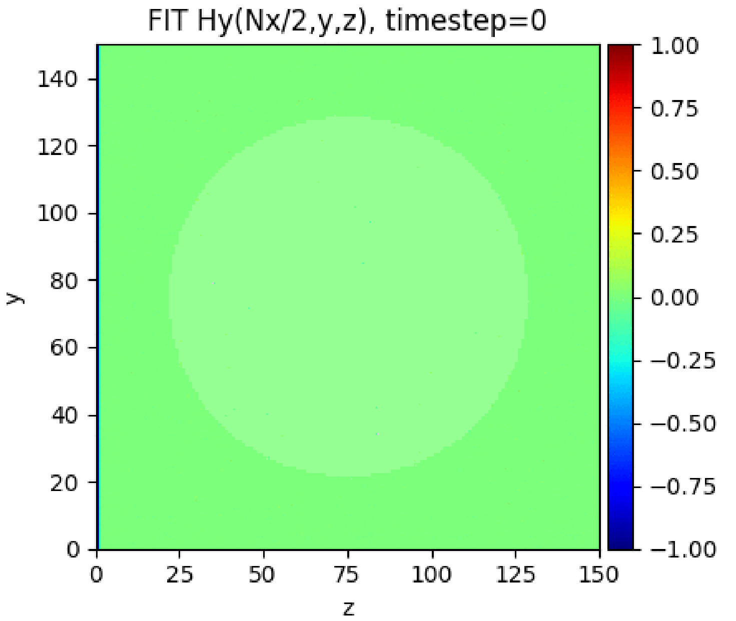
All PEC BCs



PyFIT [WSL: Ubuntu] - script_planewave_fit.py

```python
# Initial conditions
def plane_wave(solver,t, Nt,f=None, beta=1.0):

    if f is None:
        f = 15 * 1/(solver.dt*(Nt-1))    # 15 nodes
    vp = beta*c_light       # wavefront velocity beta*c
    w = 2*np.pi*f           # ang. frequency
    kz = w/c_light          # wave number

    solver.H[:,:,0,'y'] = -1.0 * np.cos(w*t)
    solver.E[:,:,0,'x'] = 1.0 * np.cos(w*t) /(kz/(mu_0*vp))
```





Outline

1. Where are we?

2. Main improvements

3. Some (fun) examples

4. Feedback from University

<u>5. Conclusions & Next steps</u>

Conclusions & Next steps

- ✓ FIT robustness:
  - ✓ Understood difference between primal grid $\boldsymbol{G}$ and dual grid $\widetilde{\boldsymbol{G}}$ quantities
  - ✓ Understood how to correctly implement BCs: PEC, PMC, Periodic
- ✓ New features:
  - ✓ STL importer based on PyVista mesh object and their collision algorithm (*staircased*)
  - ✓ We can associate any materials (ε_r, μ_r non-frequenct dependent yet) to each stl solid
  - – Lorenzo's conductors being adapted from FDTD to FIT
- Feedback from university:
  - Very positive, reviewed all the code together, made some simulations with +4,000,000 cells, new ideas for quantitative analysis based on plane waves and gaussian wave packet.
- Still need to understand
  - **If we can inject a beam** (gaussian current with $J(t)$) without implementing divergence correction
  - How to test the new features of the code (**Unit test**)
  - How to do a **quantitative comparison** of the simulated fields vs CST or analytic calculations

PhD roadmap updated

- **2023: Jul – Ago (2 months):**
  - ✓ Physics review FIT
  - ✓ **3D Eqs** in python
  - Test in cube (*PEC BCs not working!*)
- **2023: Sep-Dic (4 months):**
  - PEC BCs
  - Embedded boundaries
  - Add PML/CPML (feasible??)
  - First taste of materials

We are here !

- **2024: Jan–April (4 months):**
  - UNIT TEST
  - University roadmap

    (next visit ~April 24)

  (...)

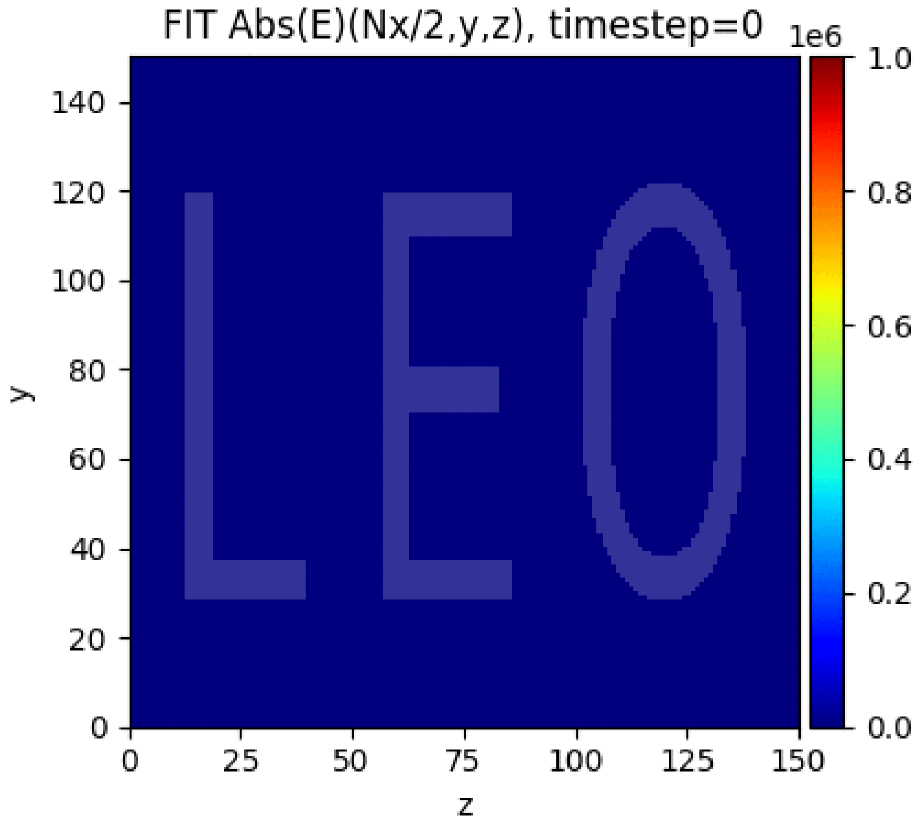
Thank you ☺ !!!

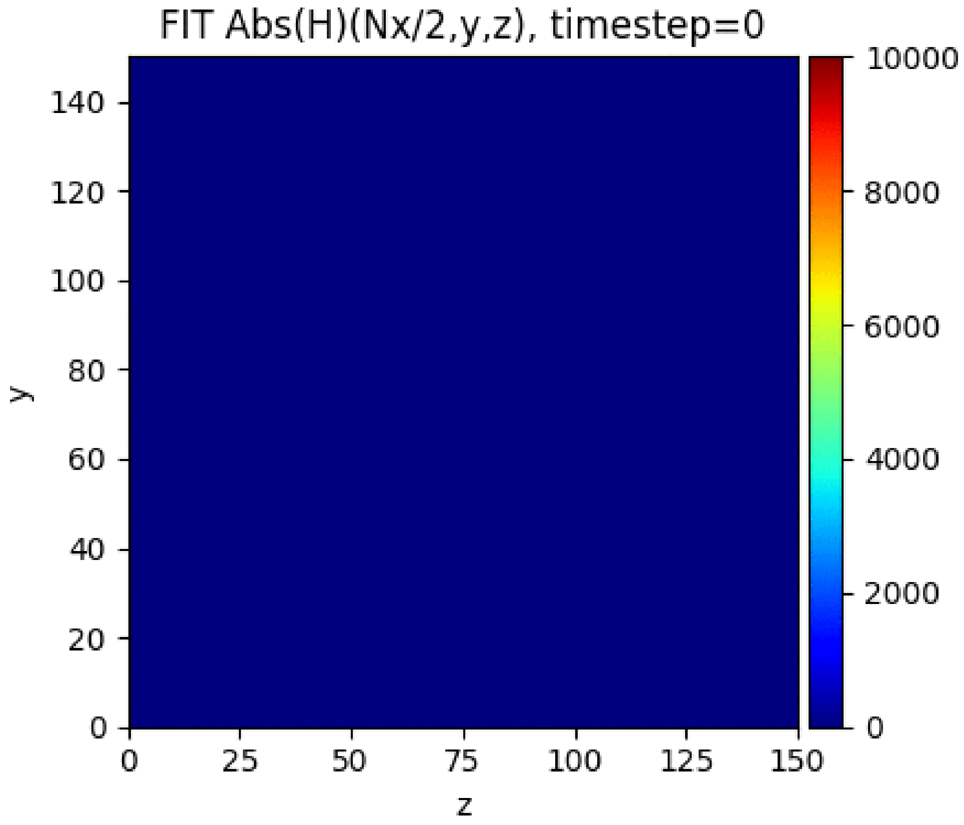
Electromagnetic and Wake Solver Development meeting #18

Elena de la Fuente García (BE–ABP–CEI)

Divergence correction for charges

Image charges appearing in E field due to violation of gauss law (continuity eq)



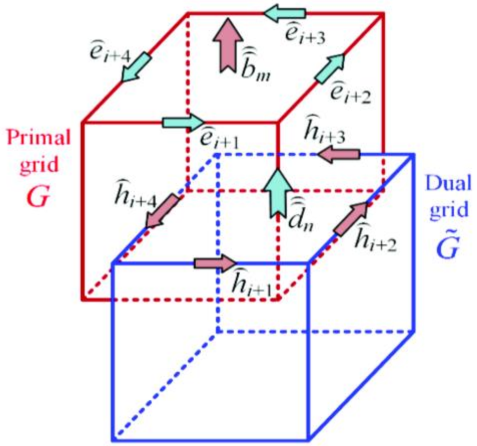


Could be solved applying divergence correction (electrostatic poisson solve)

FIT theory: Grid Maxwell Equations

$$
\begin{cases}
\oint_{\partial A} \boldsymbol{E} \cdot ds = -\iint_A \frac{\partial \boldsymbol{B}}{\partial t} \cdot d\boldsymbol{A} \\
\oint_{\partial A} \boldsymbol{H} \cdot ds = -\iint_A \left(\frac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J}\right) \cdot d\boldsymbol{A} \\
\oiint_{\partial V} \boldsymbol{B} \cdot d\boldsymbol{A} = 0 \\
\oiint_{\partial V} \boldsymbol{D} \cdot d\boldsymbol{A} = \iiint_V \boldsymbol{\rho}\, d\boldsymbol{V} \\
\boldsymbol{D} = \underline{\underline{\varepsilon}}\boldsymbol{E}, \quad \boldsymbol{B} = \underline{\underline{\mu}}\boldsymbol{H}, \quad \mathbf{J} = \underline{\underline{\sigma}}\boldsymbol{E} + \boldsymbol{\rho v}
\end{cases}
$$

FIT



Grid Maxwell Equations

$$\boldsymbol{C D_s e} = -\boldsymbol{D_A} \frac{\partial \boldsymbol{b}}{\partial t}$$

$$\boldsymbol{\tilde{C} \tilde{D}_s h} = \boldsymbol{\tilde{D}_A} \left(\frac{\partial \boldsymbol{d}}{\partial t} + \boldsymbol{j}\right)$$

$$\boldsymbol{S D_A b} = \mathbf{0}$$

$$\boldsymbol{\tilde{S} \tilde{D}_A} \left(\frac{\partial \boldsymbol{d}}{\partial t} + \boldsymbol{j}\right) = \mathbf{0}$$

- Operators
- Spatial matrixes
- Material properties

$$\boldsymbol{d} = \boldsymbol{\tilde{D}_\varepsilon e}, \quad \mathbf{b} = \boldsymbol{D_\mu h}, \quad \mathbf{j} = \boldsymbol{\tilde{D}_\sigma e} + \boldsymbol{D_\rho v}$$

$$\boldsymbol{\varepsilon} = (\boldsymbol{\varepsilon_r} + \frac{\boldsymbol{\sigma}}{\boldsymbol{j\omega}})\boldsymbol{\varepsilon_0}$$

What we care about:
Update equations

$$\boldsymbol{h}^{n+1} = \boldsymbol{h}^n - \Delta t\, \boldsymbol{\tilde{D}_s D_\mu^{-1} D_A^{-1} C e}^{n+0.5}$$

$$\boldsymbol{e}^{n+1.5} = \boldsymbol{e}^{n+0.5} + \Delta t\, \boldsymbol{D_s \tilde{D}_\varepsilon \tilde{D}_A^{-1} \tilde{C} h}^n - \boldsymbol{\tilde{D}_\varepsilon j}^n$$

We need to build all these matrices and then apply these equations every timestep !

Bibliography used:

1990 Joint US-CERN Accelerator Course
Hilton Head, So. Carolina

Wake Fields and Impedances

T. Weiland, R. Wanzenberg

Contents

1

- Wakefields and Impedances [T. Weiland, 1991]

- TE/TM FIT for accelerators [I. Zarg, 2005]

- Open boundaries for FIT [MC. Balk, 2005]

- 2D expansion [I. Zarg, 2015]

- 2D freq. domain [R. SChumann, 2000]

- Frequency domain FIT and FEM Uwe Niedermayer PhD thesis [2015]

- Eigenmode + MPI + Gyrotropic materials Klaus Klopfer PhD thesis [2014]

- Vlasov solver PIC Lukas Hanichen PhD thesis [2016]