EMT: a DGA-based tool for (electromagnetic) simulations

Matteo Cicuttin
aka IV3IWE aka datafl4sh

End summer camp 2k14

30 agosto 2014

Outline

This talk is about EMT, a simulation tool based on the Discrete Geometric Approach. Currently it can solve electromagnetic problems in the frequency domain, but it can be extended to solve various physical problems. Today we will talk about

- Some theory about the problem we want to solve
- Hands on: how to use the software and how it looks
- Something about its internals

As a plus I will annoy you with some boring math!

Maxwell's equations

The whole electromagnetism is described by the Maxwell's equation: *an electronic engineer, when he takes his girlfriend out for dinner, could derive from the Maxwell's equations even the restaurant's menu* (cit.)

- Ampère-Maxwell equation: $\nabla \times \mathbf{h} = i\omega\mathbf{d} + \mathbf{j}_s$
- Faraday-Neumann equation: $\nabla \times \mathbf{e} = -i\omega\mathbf{b}$
- Electric Gauss law: $\nabla \cdot \mathbf{d} = \rho$
- Magnetic Gauss law: $\nabla \cdot \mathbf{b} = 0$

Moreover, there are the constitutive relations

- Electric constitutive relation: $\mathbf{d} = \epsilon\mathbf{e}$
- Magnetic constitutive relation: $\mathbf{b} = \mu\mathbf{h}$
- Ohm's law: $\mathbf{j} = \sigma\mathbf{e}$

Electromagnetic wave propagation (in frequency domain)

From Maxwell equations we can obtain the propagation equation in the frequency domain:

- Take the Ampère-Maxwell equation: $\nabla \times \mathbf{h} = i\omega\mathbf{d} + \mathbf{j}_s$.
- Substitute $\mathbf{h} = \nu\mathbf{b}$: $\nabla \times (\nu\mathbf{b}) = i\omega\epsilon\mathbf{e} + \mathbf{j}_s$
- Using Faraday-Neumann: $\nabla \times \ (\nu\nabla \times \mathbf{e}) = -i\omega(i\omega\epsilon\mathbf{e} + \mathbf{j}_s)$
- Rearrange terms: $\nabla \times \ (\nu\nabla \times \mathbf{e}) - \omega^2\epsilon\mathbf{e} = -i\omega\mathbf{j}_s$

We are not interested in the imposed currents, so we could write the propagation problem as follows:

$$\nabla \times \ (\nu\nabla \times \mathbf{e}) - \omega^2\epsilon\mathbf{e} = 0$$

Swapping the equations and repeating the same procedure we obtain the *complementary formulation*:

$$\nabla \times \ (\xi\nabla \times \mathbf{h}) - \omega^2\mu\mathbf{h} = 0$$

How do we solve that problem?

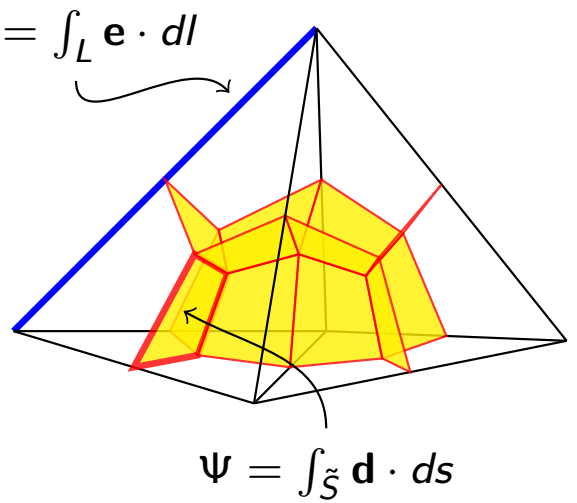
Discrete geometric approach

The discretization of Ω is done by means of two grids:

- The primal grid $\mathcal{G}$, which is tetrahedral;
- The dual grid $\tilde{\mathcal{G}}$ obtained by the barycentric subdivision of $\mathcal{G}$.

Different integral quantities are associated to different geometric entities, on both $\mathcal{G}$ and $\tilde{\mathcal{G}}$.
Tonti's principle states that:

- Configuration variables belong to $\mathcal{G}$
- Source variables belong to $\tilde{\mathcal{G}}$



Discrete geometric equations

We write the Maxwell equations in the discrete domain as follows:

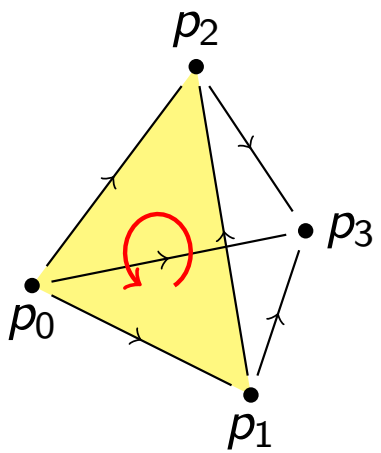
$$\mathbf{C}\mathbf{U} = -i\omega\mathbf{\Phi}$$
$$\mathbf{C}^T\mathbf{F} = i\omega\mathbf{\Psi} + \mathbf{I}_s$$

U: electromotive force
Φ: magnetic flux
F: magnetomotive force
Ψ: electric flux

Note the perfect similarity to the continuous ones:

$$\nabla \times \mathbf{e} = -i\omega\mathbf{b}$$
$$\nabla \times \mathbf{h} = i\omega\mathbf{d} + \mathbf{j}_s$$

C is the face-edge incidence matrix and is the discrete equivalent of the Curl operator



A very interesting fact is that the discrete equations are **exact**: there is no approximation!

Discrete constitutive relations

In addition to the discrete Maxwell equations, we need the discrete constitutive relations:

Discrete form	Continuos form
$\mathbf{F} \approx M_\nu \mathbf{\Phi}$	$\mathbf{h} = \nu \mathbf{b}$
$\mathbf{\Psi} \approx M_\epsilon \mathbf{U}$	$\mathbf{d} = \epsilon \mathbf{e}$

They are the approximate counterparts of ν and ϵ (they are exact *only* for constant fields in the tetrahedron).
The matrices M_ϵ and M_ν are the *constitutive matrices* which relate quantities on the primal grid and quantities on the dual grid.

Discrete geometric equations

Now we are ready to solve the DGA equations to obtain the discrete propagation problem:

- Take the Ampère-Maxwell equation: $\mathbf{C}^T\mathbf{F} = i\omega\mathbf{\Psi} + \mathbf{I}_s$
- Substitute $\mathbf{F} = M_\nu\mathbf{\Phi}$
- Using Faraday-Neumann: $\mathbf{C}^T M_\nu\mathbf{C}\mathbf{U} = -i\omega(i\omega M_\epsilon\mathbf{U} + \mathbf{I}_s)$
- Rearrange terms: $\mathbf{C}^T M_\nu\mathbf{C}\mathbf{U} - \omega^2 M_\epsilon\mathbf{U} = -i\omega\mathbf{I}_s$

We are not interested in the imposed currents, so we could write the propagation problem as follows:

$$\mathbf{C}^T M_\nu\mathbf{C}\mathbf{U} - \omega^2 M_\epsilon\mathbf{U} = 0$$

Swapping the equations and repeating the same procedure we obtain the *complementary formulation*:

$$\mathbf{C}^T M_\xi\mathbf{C}\mathbf{F} - \omega^2 M_\mu\mathbf{F} = 0$$

Boundary conditions

The propagation problem in the form we obtained in the previous slide is not very interesting, we need the *boundary conditions* which, for now, are of four types:

- Perfect Electric Conductor: the electric field on a given boundary is zero (Shorted transmission line)
- Perfect Magnetic Conductor: the magnetic field on a given boundary is zero (Open transmission line)
- Admittance: a surface with a given wave admittance (transmission line closed on a load)
- Port: a surface where an electromagnetic wave can enter (the generator)

But to introduce them we need a slight generalization of the problem!

Admittance boundary conditions

The admittance boundary condition is imposed by considering the contribution of the boundary to the circulation of $\mathbf{h}$:

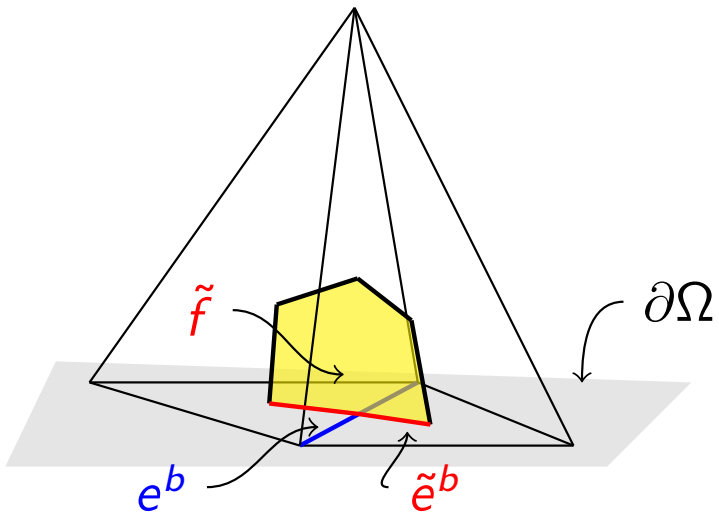
Our aim is achieved by modifying the Ampère-Maxwell law

$$\mathbf{C}^T\mathbf{F} - \mathbf{F}^b = i\omega\mathbf{\Psi} + \mathbf{I}_s$$

Neglecting $\mathbf{I_s}$ our problem becomes

$$\mathbf{C}^T M_\nu \mathbf{C}\mathbf{U} - \omega^2 M_\epsilon \mathbf{U} + i\omega\mathbf{F}^b = 0$$

where "b" remind us that we are dealing with boundary edges.



But we have an equation in $\mathbf{U}$ and $\mathbf{F}^b$! We need to introduce a new matrix, which relates MMFs and EMFs on the boundary edges: $\mathbf{F}^b = M_Y\mathbf{U}^b$, the *admittance matrix*. Our new problem is

$$\mathbf{C}^T M_\nu \mathbf{C}\mathbf{U} - \omega^2 M_\epsilon \mathbf{U} + i\omega M_Y\mathbf{U}^b = 0$$

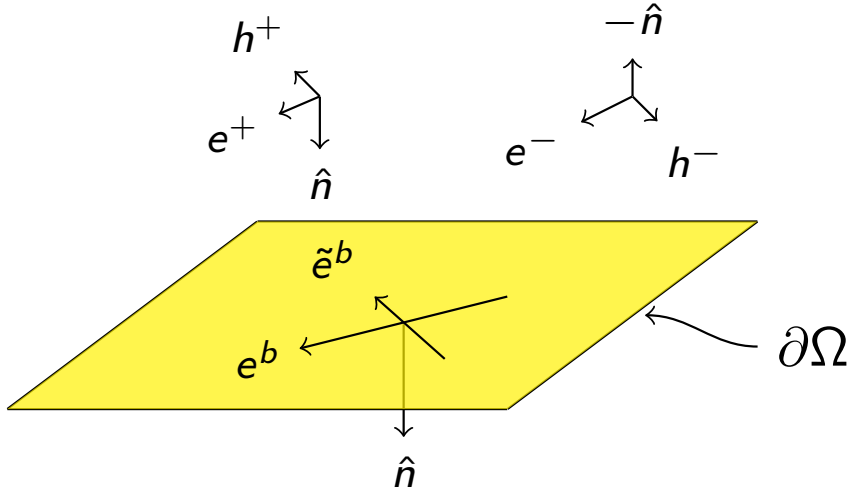
Port boundary conditions

Consider the equation of previous slide:

$$\mathbf{C}^T M_\nu \mathbf{C}\mathbf{U} - \omega^2 M_\epsilon \mathbf{U} + i\omega \mathbf{F}^b = 0$$

instead of writing $\mathbf{F}^b = M_Y \mathbf{U}^b$ we could write $\mathbf{F}^b = M_Y \mathbf{U}^b + 2\mathbf{F}^{b^-}$.

This allows us to separate the components *entering* Ω from the components *leaving* Ω.



The problem then becomes

$$\mathbf{C}^T M_\nu \mathbf{C}\mathbf{U} - \omega^2 M_\epsilon \mathbf{U} + i\omega M_Y \mathbf{U}^b = -2i\omega \mathbf{F}^b$$

which is the *full electromagnetic wave propagation problem in the frequency domain*.

Why a new code?

Goal: study the electromagnetic phenomena inside anechoic chambers
Obstacles:

- matrices resulting from discretization have bad convergence properties (they are indefinite) $\Longrightarrow$ direct solvers
- anechoic chambers are big $\Longrightarrow$ *lots* of elements
- high frequency $\Longrightarrow$ *tons* of elements
- presence of antennas and other objects $\Longrightarrow$ fine geometrical details $\Longrightarrow$ *too many* elements

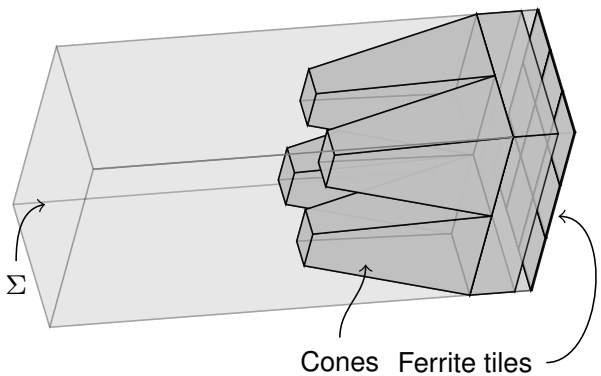
Necessity: we would like to have some equivalent models of the objects inside the chamber, so we can reduce the number of elements.
There isn't a code providing such capabilities.

Equivalent model for anechoic walls

The port boundary condition is crucial to study the *unitary cell* (figure), the basic unit of an anechoic wall.

- 2×2 cones
- 3×3 ferrite tiles

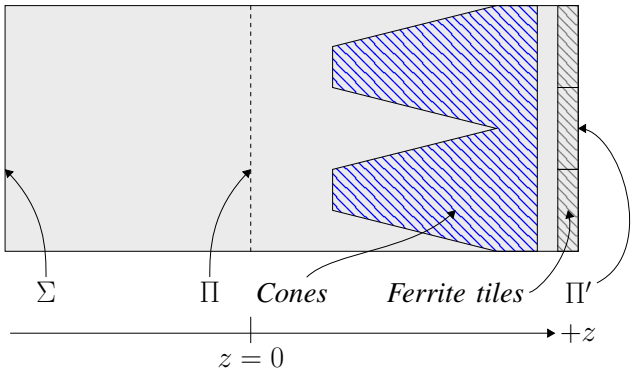
Our goal is to try to substitute the cones and the ferrites with a 2D surface.



Equivalent model for anechoic walls

The port boundary condition is crucial to study the *unitary cell*, to substitute cones and ferrite tiles with a 2D surface. Idea:

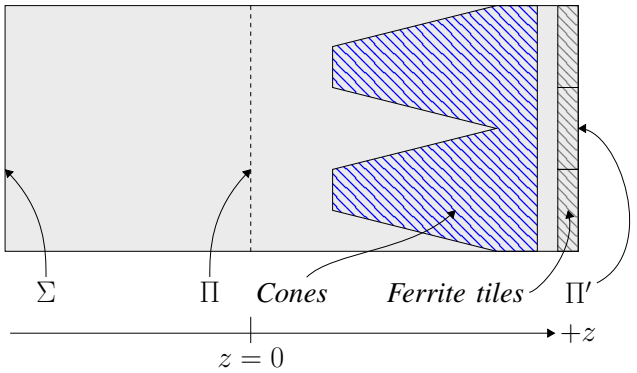
- use the port to apply a plane wave on Σ



Equivalent model for anechoic walls

The port boundary condition is crucial to study the *unitary cell*, to substitute cones and ferrite tiles with a 2D surface. Idea:

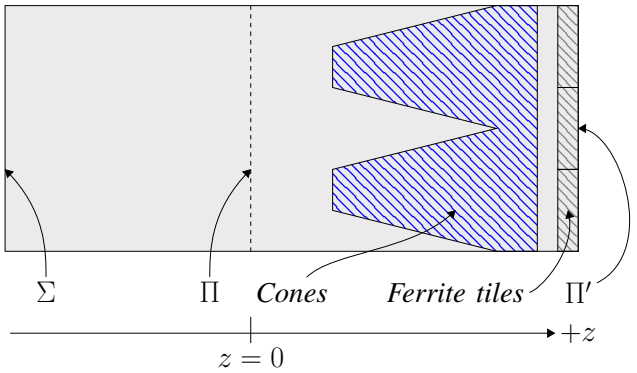
- use the port to apply a plane wave on Σ
- calculate wave impedance on a plane far away from cones



Equivalent model for anechoic walls

The port boundary condition is crucial to study the *unitary cell*, to substitute cones and ferrite tiles with a 2D surface. Idea:

- use the port to apply a plane wave on Σ
- calculate wave impedance on a plane far away from cones
- translate impedance on the rightmost end of the cell

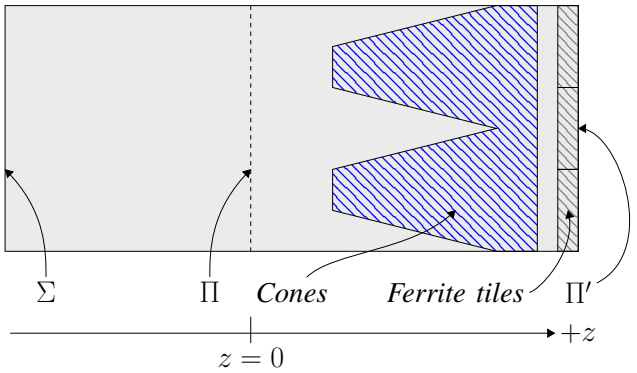


$$Z_{\Pi'}(z) = Z_c \frac{Z_\Pi - iZ_c \tan(\beta z)}{Z_c - iZ_\Pi \tan(\beta z)},$$

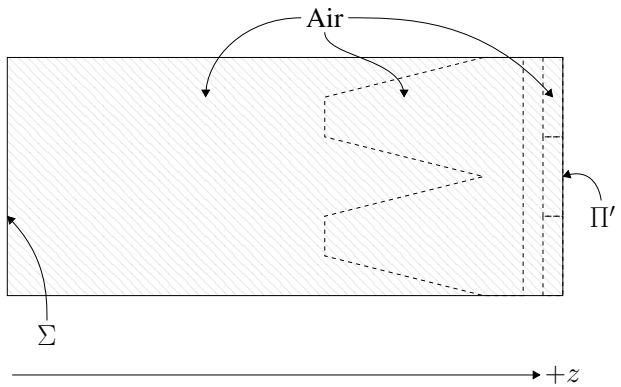
Equivalent model for anechoic walls

The port boundary condition is crucial to study the *unitary cell*, to substitute cones and ferrite tiles with a 2D surface. Idea:

- use the port to apply a plane wave on Σ
- calculate wave impedance on a plane far away from cones
- translate impedance on the rightmost end of the cell
- substitute cones and ferrites with that equivalent impedance



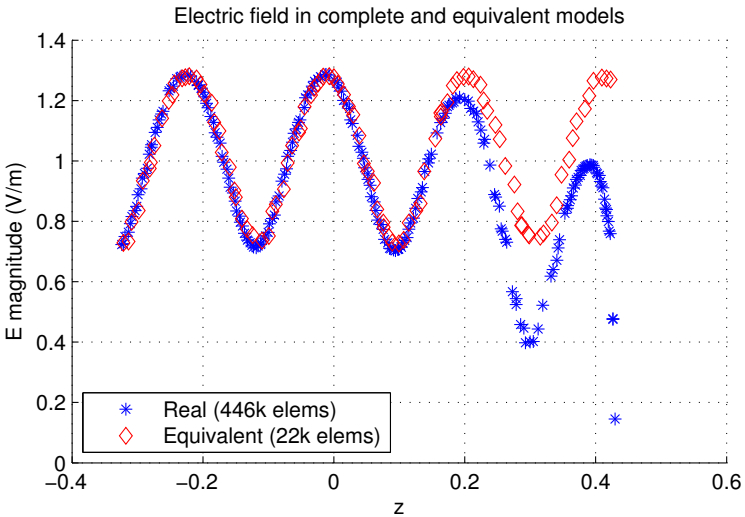
$$Z_{\Pi'}(z) = Z_c \frac{Z_\Pi - iZ_c \tan(\beta z)}{Z_c - iZ_\Pi \tan(\beta z)},$$

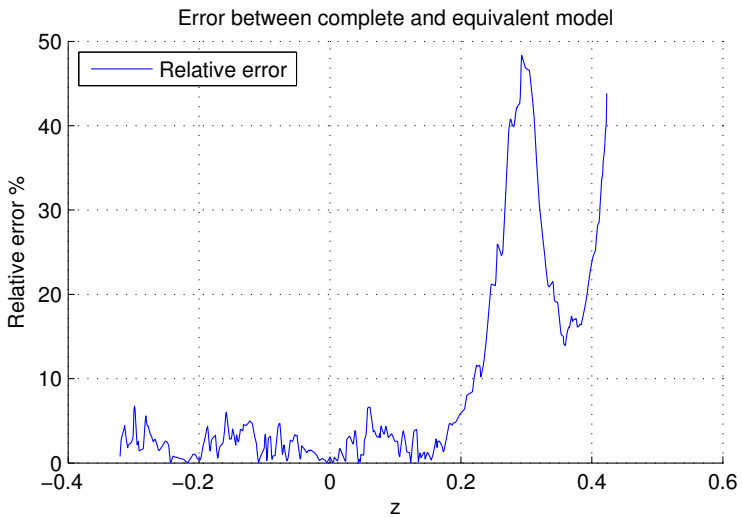


Equivalent model for anechoic walls: results

The proposed equivalent model gave very good results

- it allowed 20x reduction of mesh elements
- it allowed 60x reduction of computation times
- in the whole area of interest the relative error was below 5%





Equivalent radiating elements

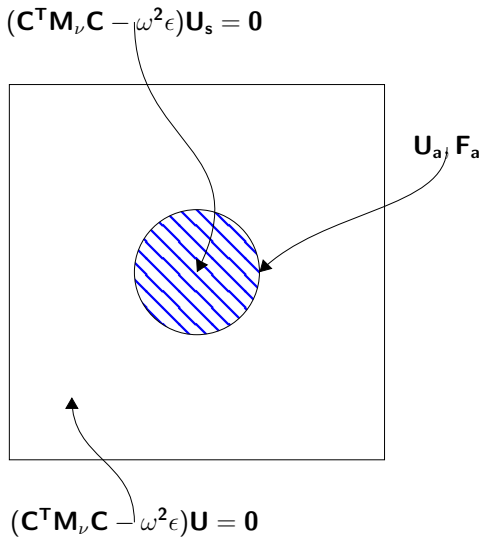
We would like to have a simple object (a sphere) that radiates a field equivalent to the one that is radiated by a more complex antenna

- Simulate the real antenna with NEC/HFSS/FEKO
- Calculate the field on the reference sphere
- Inserisco nell'ambiente la sfera che "irradia" il campo calcolato
- Insert the sphere (that radiates the calculated field) in the simulation environment

Two regions:

- Total field
- Scattering field

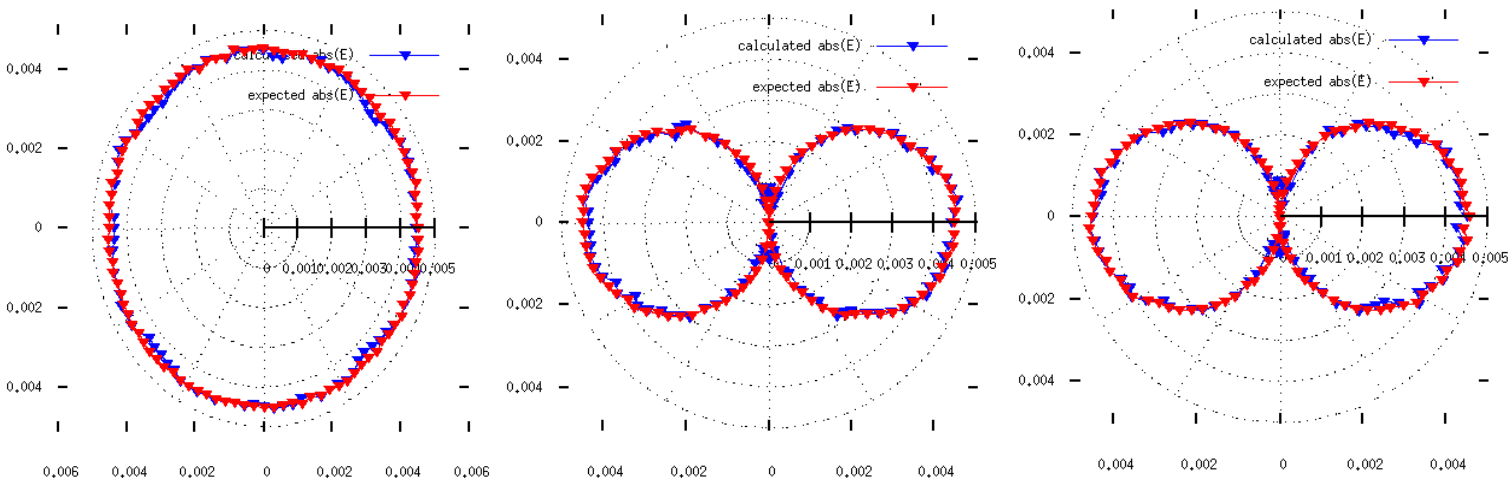
The formulation allows to evaluate (inside the sphere) only the "reaction" of the environment.



Elementi radianti equivalenti: risultati

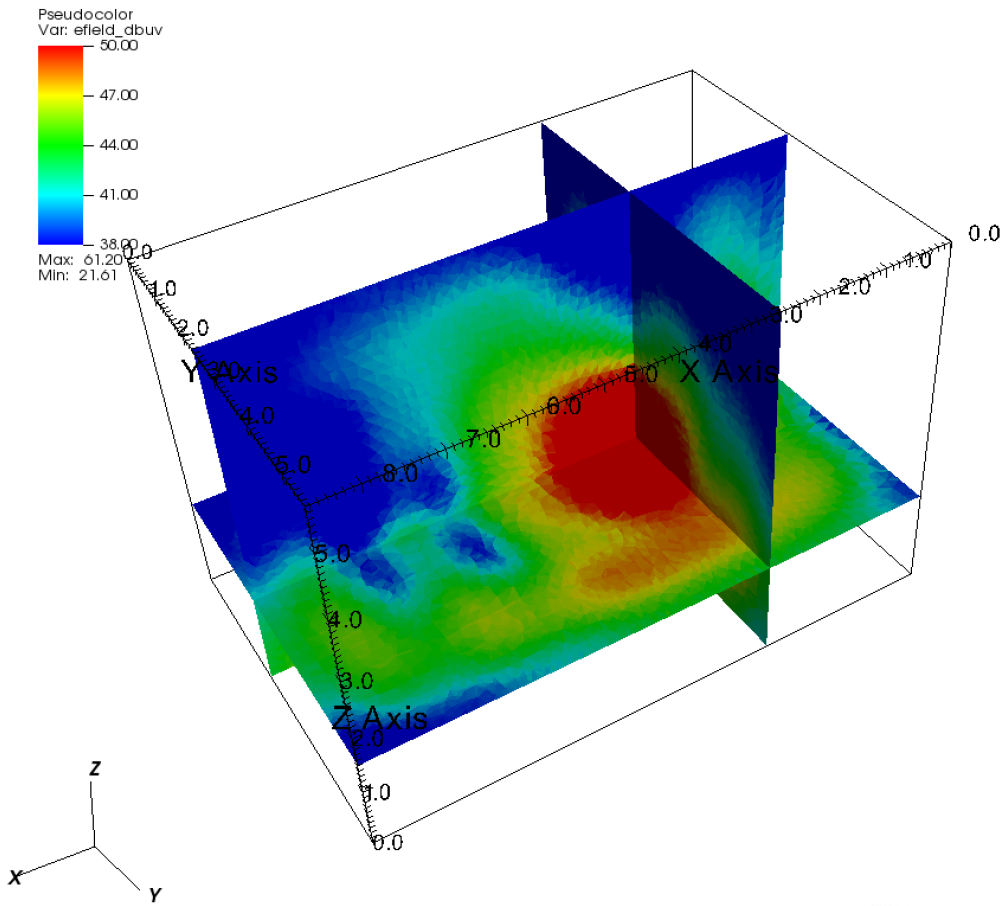
Applicability conditions of the model

- the sphere must have dimensions similar to λ
- the field must be calculated sufficiently far away from the sphere ($> \lambda/2$)



Under these conditions, the model gives very good results!

How do the results look like



First experiments have shown that the predictions of the code are very accurate: the discrepancies between the computed fields and the measured fields are in the order of 1.5 dB!

The code (1)

How do we translate all this theory in code? The *general strategy* to solve a numerical problem with the DGA is

- Read the geometry
- Read simulation parameters (frequency, boundary conditions, materials, ...)
- Assemble the linear system of the discretized problem
- Solve it
- Interpolate quantities (for example to get **E** and **H**)
- Output them in some format

The code (2)

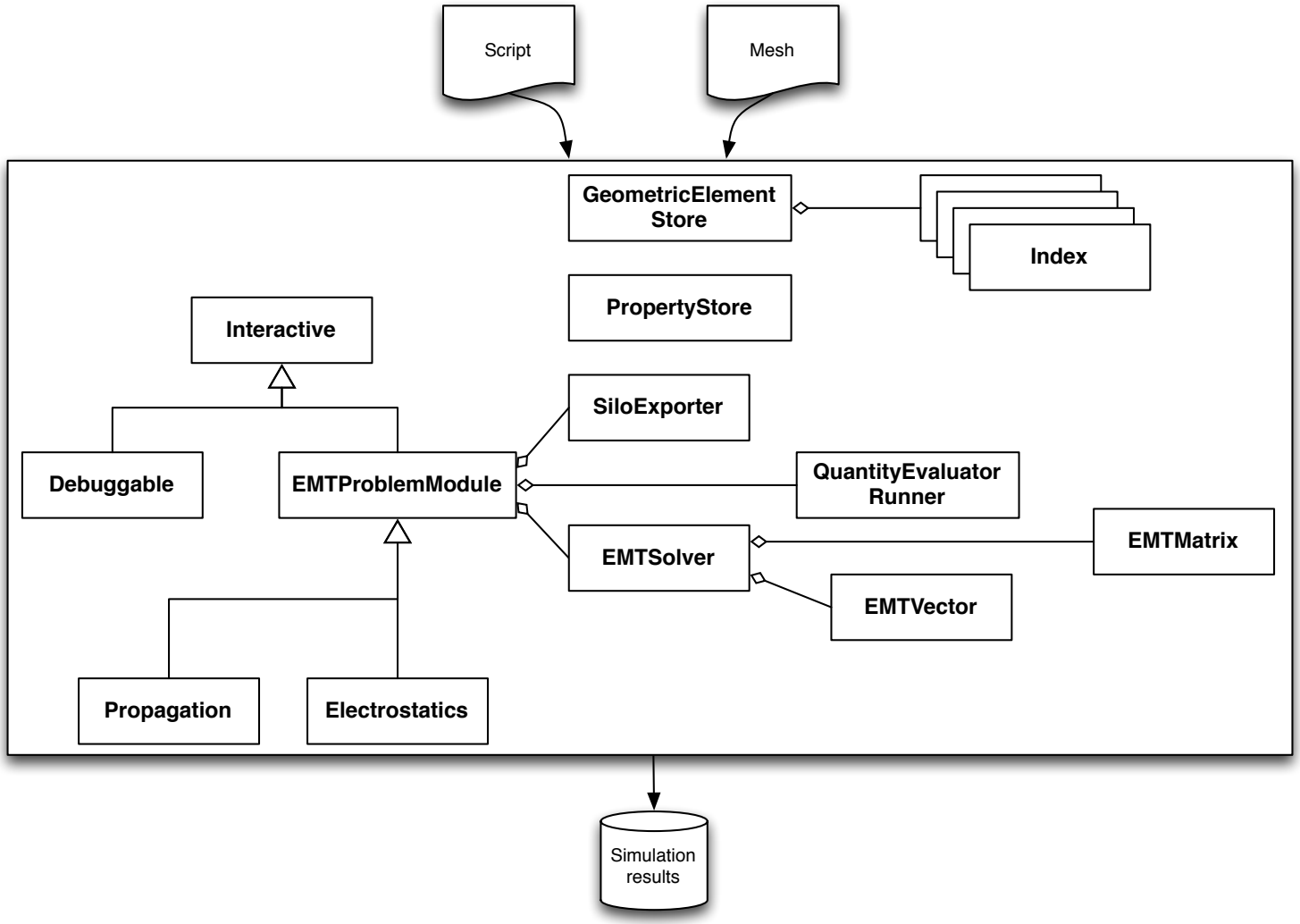
It is not too difficult to write a Matlab script that solves our problem, but we would like to have something more flexible. In particular we want a code that is:

- Expandable and understandable. It should be easy to:
  - code new problems
  - add new numerical solvers
  - add new data import/export procedures
- Modular. If something breaks it must not break surrounding things
- Debuggable and correct.
- Efficient. We want to be able to scale from small problems to very big ones

To achieve these goals some ingredients are needed:

- A serious implementation language: C++11
- A good design: no code is written "on the fly", without thinking about the structure
- Efficient memory usage and parallelism

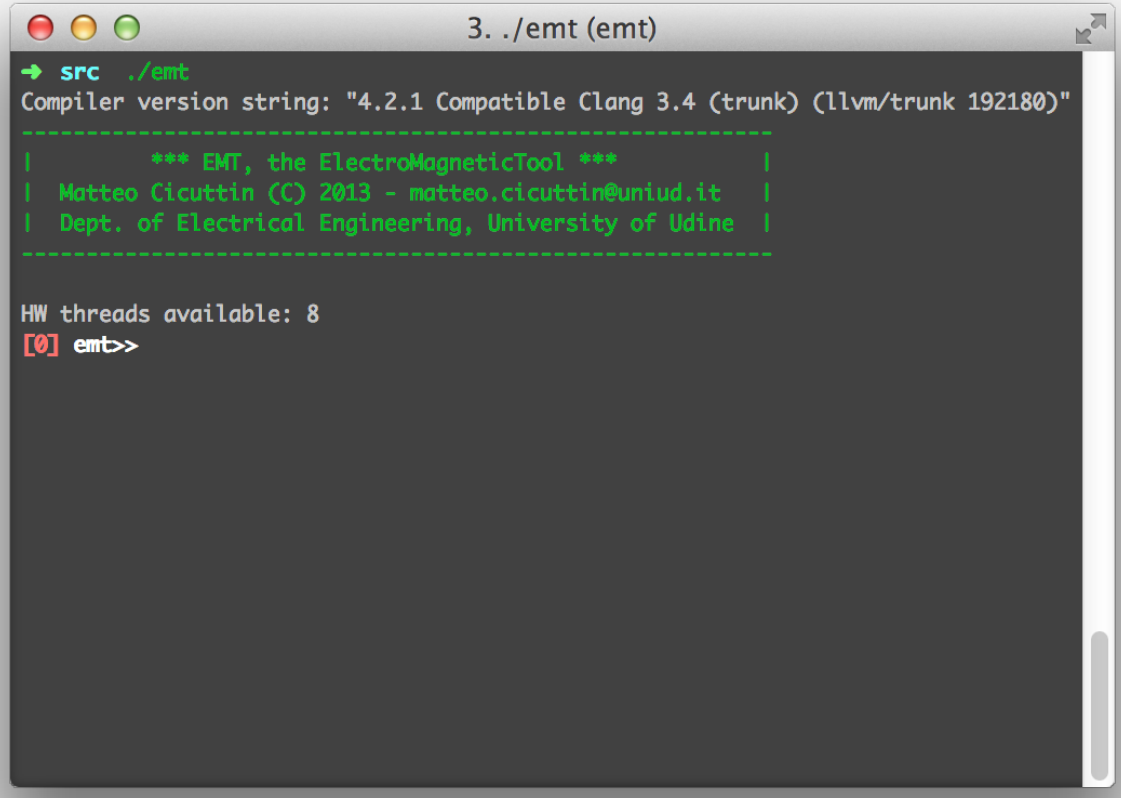
The big picture



How does it look?

Time for a little demo!

the cantenna



EMT for programmers

An important goal of EMT is to be programmer-friendly. I'll show you some features...

- What is a module and how you write one
- The interface with numerical solvers
- The geometry and how EMT deals with parallel execution

The EMT modules

The code is composed by modules: if you want to solve a propagation problem, you enter the propagation module. The modules are one example of the extendability of EMT.

```
➜ src  ./emt
Compiler version string: "4.2.1 Compatible Clang 3.4 (trunk) (llvm/trunk 192180)"
------------------------------------------------------------
|         *** EMT, the ElectroMagneticTool ***             |
|  Matteo Cicuttin (C) 2013 - matteo.cicuttin@uniud.it     |
|  Dept. of Electrical Engineering, University of Udine    |
------------------------------------------------------------

HW threads available: 8
[0] emt>> enter propagation
[0] emt/propagation>>
```

Modules are "templates" (not in the C++ sense!) that allow the programmer to code new problems in a simple way. All the modules (even an empty one) provide access

- to the geometry
- to all the numerical solvers
- to the data I/O facilities

Coding a new problem

Modules encode in some way the *general strategy* we mentioned some slides ago. Coding a new problem is simply matter of

- creating a new module by subclassing `EMTProblemModule` (some 10s LOCs)
- writing a system assembler (the hardest part)
- telling the solver to run (1 LOC)
- choosing which data to export (some 10s LOCs)
- registering your brand new module with the system (1 LOC)

Let's see some code, the `electrostatics` module.

The numerical solvers

Another example of extendability is how the numerical solvers are interfaced. Each problem benefits from specific solvers (BiCGstab vs. multigrid vs...) and there are plenty of libraries implementing them, each with its own interface.

In EMT to call all of them you need only to know that some solver exists.

```
/* Ask the solver to run */
std::cout << "Solving..." << std::endl;
_owner->solver()->solve();
```

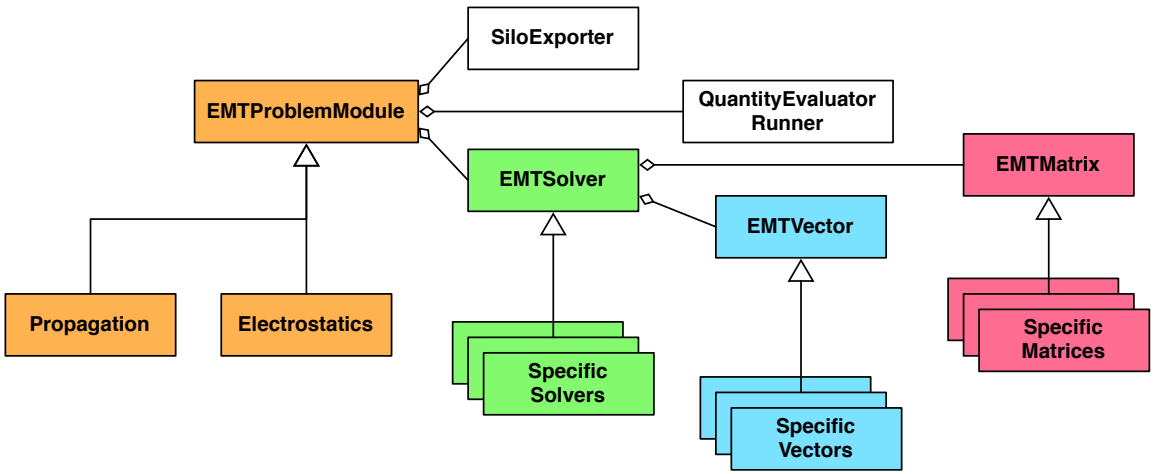
```
    EMTVector *b = _owner->solver()->get_knowns();
    b->set_value(ni_comp, -u, true);
    continue;
}

EMTMatrix *A = _owner->solver()->get_matrix();
nj_comp = _owner->_sc.toCompressed(nj_orig);
A->set_value(ni_comp, nj_comp, mGtEG(i,j), true);
```

This is possible with the *Factory*, an OOP design pattern.

How solvers are interfaced

Whichever solver you get, and whichever matrix/vector format it supports, you only need to

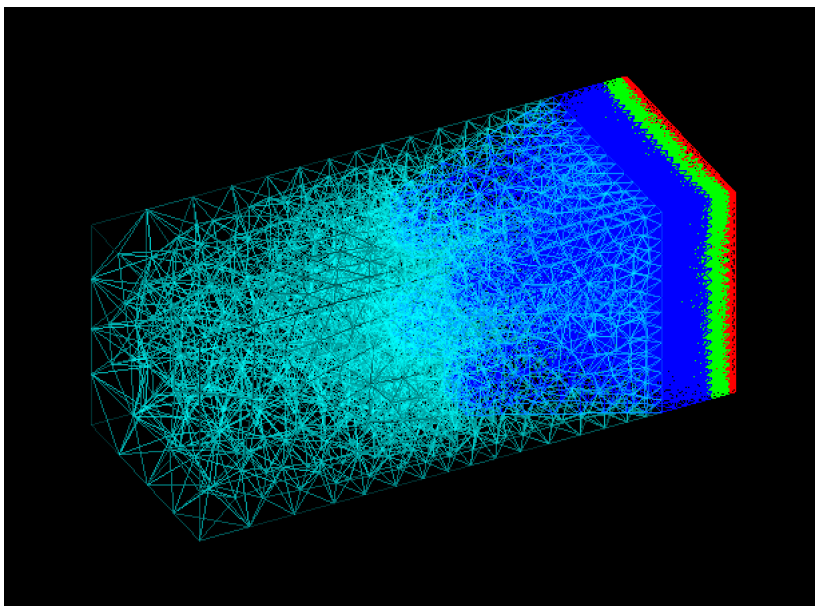
- make it *look like* the generic solver (by subclassing `EMTSolver`, `EMTMatrix` and `EMTVector`
- register it with the system



After this all the modules of EMT are aware of the new solver and able to use it. How? You make your choice at runtime with the `solver` command we already seen.

Representing the geometry (1)

The problem domain Ω, as we already know, is discretized in a tetrahedral grid which could be composed of million of elements. A key element for the scalability of the code is a good representation of the geometry (and the associated materials).



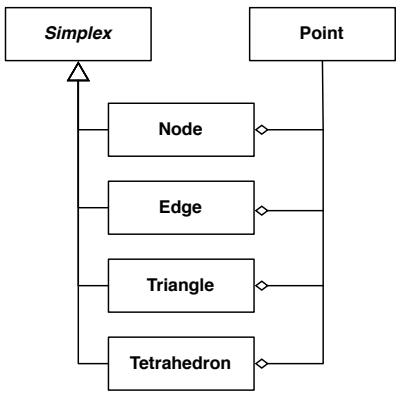
We want:

- Fast lookups
- Optimal memory usage
- Simple way to operate on the elements composing the geometry

Representing the geometry (2)

Each geometric entity of the primal grid is modeled as a class which is a subtype of `Simplex`. There are:

- Nodes (1 point + id + valid bit)
- Edges (2 points + id + valid bit)
- Triangles (3 points + id + valid bit)
- Tetrahedrons (4 points + id + valid bit)



Moreover, there is a table where all the actual `Points` are stored.

Meeting the requirements

- The data structures for the geometry must discard duplicate elements during insertion: red black trees
- We need fast lookups: red-black trees again
- We need also optimal memory usage:
  - red black trees require three pointers per node! What a waste of memory!
  - each node needs some boolean flags: alignment $\Longrightarrow$ 3 bytes of wasted storage per element.

Data structures are rather static: sorted arrays would be ok!

- Implementation with trees would require about 8GB of memory for a 5 million element mesh
- Current implementation requires less than 300MB of memory for the same mesh

What about the boolean flags?

Operations on the geometric elements (1)

Operations on the geometric elements are done by means of *Visitors*. A Visitor is a well-known OOP design pattern which allows us to clearly separate *algorithms* from *structures* on which they operate.

```
1 Tetrahedron t = ges->lookup_tetrahedron(tet_idx);
2
3 BarycenterVisitor bv;
4 t.accept(bv);
5 Point barycenter = bv.getResult();
```

In the code there are tens of Visitors performing lots of tasks, from the simplest ones (calculation of volumes, barycenters, ...) to the most complex ones (system assembly, field interpolation). It looks complicated, but...

Operations on the geometric elements (2)

...an interesting fact about Visitors is that they are designed to *not* share state, so you get parallel execution at no cost (except memory bandwidth):

```
1 std::vector<Point> barycenters;
2 using PVR = ParallelVisitorRunner<BarycenterVisitor, Tetrahedron>;
3 PVR pvr(barycenters);
4 pvr.run();
```

EMT is almost entirely parallel and can use all the cores you throw at it!

Conclusion

We talked about

- some of the theory behind EMT
- the usage of the code
- how the code could be extended

It is only the tip of an iceberg and there is a lot of work to do!

Questions/suggestions at
matteo.cicuttin@uniud.it