

# **INFORME DE INVESTIGACIÓN**

Softwares de simulación para Tecnología electrónica  
y otras cátedras de ingeniería

**Compatibilidad electromagnética**

**Gonzalo Nahuel Vaca**



Universidad Nacional de la Matanza  
Argentina  
9 de diciembre de 2022

## Resumen

Este informe es una memoria breve del trabajo realizado durante el año 2022. En el marco de la investigación se evaluaron distintos programas para el análisis de compatibilidad electromagnética.

### 1. Compatibilidad electromagnética

Esta sección es una introducción a la temática que abordan los programas evaluados; su naturaliza e importancia.

El diseño de circuitos electrónicos que manejan señales de alta frecuencia se someten a efectos no deseados. Esto se debe a que las trazas del impreso son fuentes de radiación electromagnética. Luego, se inducen corrientes en las pistas cercanas que pueden alterar o destruir parte del sistema. Además, las ecuaciones que rigen el comportamiento de la radiación en el circuito impreso son de gran complejidad.

Es posible construir guías de onda en un circuito impreso, como así también filtros y blindajes. También es posible dibujar adaptadores de impedancia para conectar una antena. En la figura 1 se puede observar un ejemplo de un circuito que implementa estos componentes como parte de sus trazas.

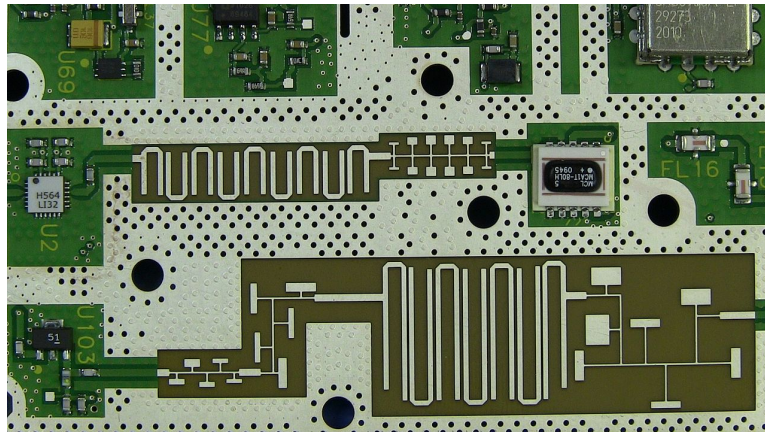


Figura 1: Guías de ondas, filtros, adaptadores y blindajes impresos.

Los componentes modernos tienen un alto grado de integración y a menudo manejan señales de alta frecuencia. Los encapsulados tienen un gran número de terminales que en algunos casos obligan a crear un circuito impreso donde las pistas de alta frecuencia se encuentran en proximidad. En la figura 2 se puede ver una *field-programable gate array* (FPGA) y se puede apreciar la cercanía de sus terminales. Además, este tipo de componente se suele utilizar en una cadena de procesamiento de datos de alta frecuencia.

Finalmente, se pone de manifiesto la necesidad de utilizar una herramienta informática para asistir en el cálculo y simulación de la física involucrada.

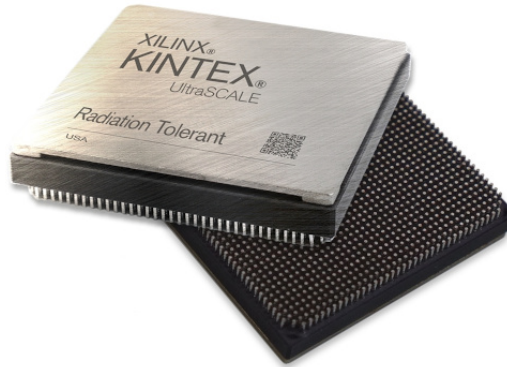


Figura 2: Componente de alto grado de integración.

Los programas evaluados utilizan el método de elementos finitos (FEM). El cuál permite resolver numéricamente ecuaciones diferenciales que de la geometría del circuito y de las señales que lo estimulan. Esta estrategia se utiliza en distintos tipos de simulación como: análisis estructural, transferencia de calor, flujo de fluidos, transporte de masa y potencial electromagnético.

El FEM es un método numérico general para resolver ecuaciones diferenciales parciales en dos o tres variables espaciales (es decir, algunos problemas de valores en la frontera). Para resolver un problema, el FEM subdivide un sistema grande en partes más pequeñas y simples que se denominan elementos finitos. Esto se logra mediante una discretización espacial particular en las dimensiones del espacio, que se implementa mediante la construcción de una malla del objeto: el dominio numérico para la solución, que tiene un número finito de puntos. La formulación del método de elementos finitos de un problema de valores en la frontera finalmente da como resultado un sistema de ecuaciones algebraicas. El método aproxima la función desconocida sobre el dominio. Las ecuaciones simples que modelan estos elementos finitos se ensamblan luego en un sistema de ecuaciones más grande que modela todo el problema. Luego, el FEM aproxima una solución minimizando una función de error asociada a través del cálculo de variaciones.

En la figura 3 se puede observar el resultado de una simulación por el método FEM.

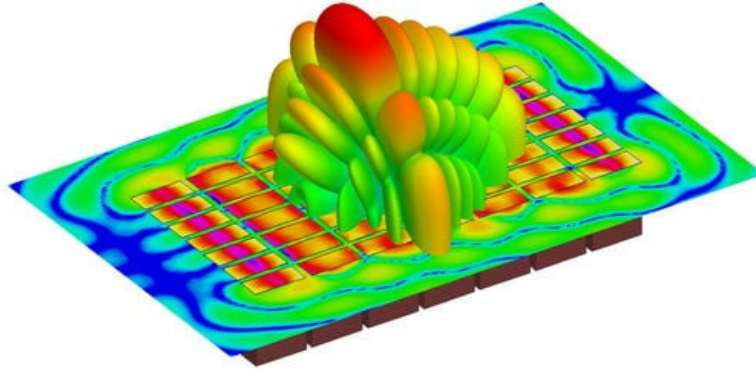


Figura 3: Simulación de un circuito impreso.

## 2. OpenEMS

En esta sección se encuentran los detalles de la evaluación del programa OpenEMS.

### 2.1. Introducción

El programa OpenEMS fue escrito en el lenguaje C++ y utiliza el método de las diferencias finitas en el dominio temporal (FDTD) para resolver simulaciones electromagnéticas.

El programa fue pensado originalmente para realizar simulaciones de equipos médicos de resonancia magnética (MRI). Sin embargo, se desarrolló un analizador léxico que permite transformar el formato *Hyperlynx* que utiliza KiCad al esquema de geometría de OpenEMS.

El modelo geométrico de OpenEMS es *CSXCAD* y es una biblioteca propia del programa. Además, permite su visualización y trabajo en coordenadas cartesianas y cilíndricas. Finalmente, provee una interfaz para Matlab/Octave y Python.

Las características más relevantes son:

- Diseño y simulación en coordenadas cartesianas.
- Diseño y simulación en coordenadas cilíndricas.
- Soporta modelos *voxel* para simulaciones MRI.
- Multi-hilos y SIMD.
- Intefaz Matlab/Octave.
- Definición de materiales en el espacio.
- Definición de excitaciones en el espacio.

- Generación de archivos en formato *vtk* y *hdf5*
- Materiales dispersivos(Drude/Lorentz/Debye)
- Rutinas de post-procesamiento configurables (Matlab/Octave).
- Sub-espacios de simulación para optimizar densidades de cálculo.
- Simulaciones remotas por medio de *ssh*.
- Es libre y gratuito.

Se creó un entorno de trabajo en Ubuntu 2020 y en contenedores de *docker* para las versiones de Ubuntu 2018 y 2016. Además se estudió como utilizarlo con Octave y se obtuvieron los siguientes resultados:

- El analizador léxico se encuentra abandonado y no es posible hacerlo funcionar
- KiCad 6 cambió su formato *Hyperlynx* y el analizador léxico es por lo tanto obsoleto.
- Las geometrías de los circuitos impresos se deben construir a mano.
- Los errores de geometría no son útiles para depurar los datos ingresados.

Dado el estado actual de este programa se hace imposible utilizarlo como herramienta de las cátedras si antes no se repara y actualiza el analizador léxico.

En la figura 4 se puede observar una simulación realizada en OpenEMS.

## 2.2. Instalación

La instalación del programa no es trivial y a continuación se detallan los pasos a seguir para tener el programa en funcionamiento. Esto tiene la finalidad de demostrar la dificultad que enfrentaría el cuerpo de estudiantes y docentes.

A continuación se detallan los pasos a seguir:

1. Satisfacer las dependencias del programa, a continuación se muestran los comandos para el sistema Ubuntu 22.04 LTS:
  - Dependencias mínimas obligatorias:
 

```
sudo apt-get install
build-essential cmake git libhdf5-dev libvtk7-dev
libboost-all-dev libcgall-dev libtinyxml-dev
qtbase5-dev libvtk7-qt-dev
```

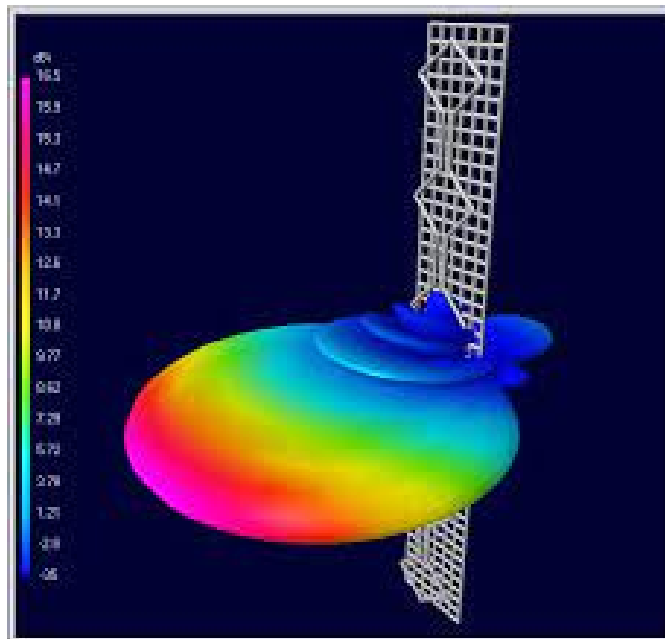


Figura 4: Simulación una antena en OpenEMS.

- GNU Octave (equivalente GNU de Matlab, opcional):  
`sudo apt-get install octave liboctave-dev`
- Interfaz Python (opcional):  
`sudo pip3 install matplotlib cython h5py`

## 2. Clonar e instalar

- Clonar el repositorio y sus submódulos:  
`git clone --recursive`  
`https://github.com/thliebig/openEMS-Project.git`
- Dentro del repositorio ejecutar:  
`./update_openEMS.sh ~/opt/openEMS --python`

## 2.3. Verificación

La prueba más simple para verificar la instalación es ejecutar el binario de openEMS. Para lograrlo debe navegar hasta la carpeta de la instalación, la cuál normalmente es `opt/openEMS/bin` dentro de su usuario; luego se debe ejecutar el comando `./openEMS` y deberá observar una salida como la siguiente:

-----

```
| openEMS 64bit -- version v0.0.35-76-gd4448fa  
| (C) 2010-2018 Thorsten Liebig <thorsten.liebig@gmx.de>  
| GPL license
```

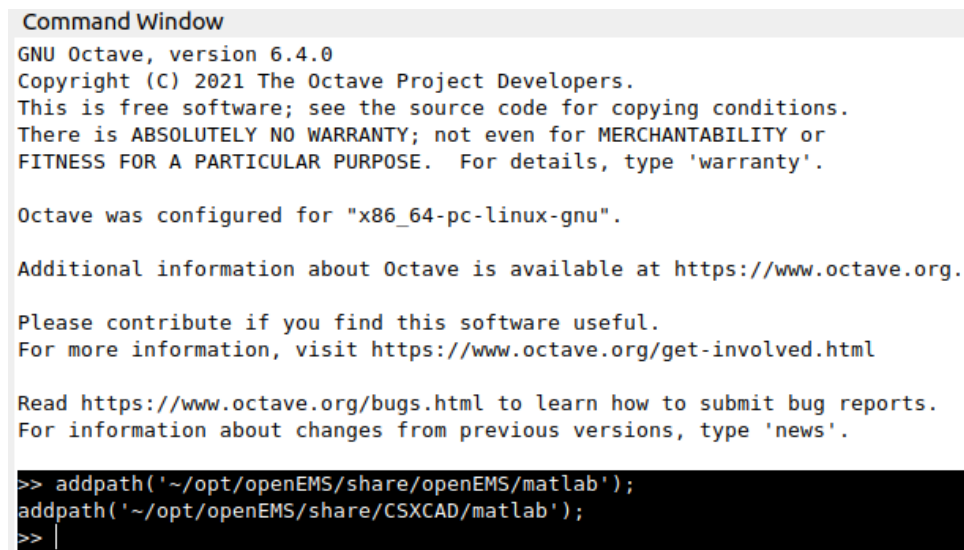
```
-----  
Used external libraries:  
CSXCAD -- Version: v0.6.2-109-gcd9decb  
hdf5    -- Version: 1.10.7  
compiled against: HDF5 library version: 1.10.7  
tinyxml -- compiled against: 2.6.2  
fparser  
boost   -- compiled against: 1_74  
vtk     -- Version: 7.1.1  
compiled against: 7.1.1
```

A continuación se detallan los pasos a seguir para verificar la correcta instalación de openEMS con GNU Octave.

Como se muestra en la figura 5 se debe colocar el *path* de openEMS en la ventana de comandos.

Los comandos para agregar los *path* son:

```
addpath('~/opt/openEMS/share/openEMS/matlab');  
addpath('~/opt/openEMS/share/CSXCAD/matlab');
```



```
Command Window  
GNU Octave, version 6.4.0  
Copyright (C) 2021 The Octave Project Developers.  
This is free software; see the source code for copying conditions.  
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or  
FITNESS FOR A PARTICULAR PURPOSE. For details, type 'warranty'.  
  
Octave was configured for "x86_64-pc-linux-gnu".  
  
Additional information about Octave is available at https://www.octave.org.  
  
Please contribute if you find this software useful.  
For more information, visit https://www.octave.org/get-involved.html  
  
Read https://www.octave.org/bugs.html to learn how to submit bug reports.  
For information about changes from previous versions, type 'news'.  
  
>> addpath('~/opt/openEMS/share/openEMS/matlab');  
addpath('~/opt/openEMS/share/CSXCAD/matlab');  
>>
```

Figura 5: Configuración de *path* en GNU Octave.

El siguiente paso es verificar la interfaz CXCAD, a continuación se muestra el comando y respuesta en la ventana de comandos de GNU Octave:

```
>> InitCSX
ans =
Properties: []
```

Con la interfaz inicializada se puede realizar una verificación de sus funciones como se muestra a continuación:

```
>> InitFDTD('NrTS', 0, 'EndCriteria', 0)
ans =
    ATTRIBUTE: [1x1 struct]
```

El siguiente paso es verificar el correcto funcionamiento del simulador como se muestra a continuación:

```
>> RunOpenEMS( '.', 'nonexistant.xml', '' )
[...]
Read openEMS xml file: nonexistent.xml ...
openEMS: Error File-Loading failed!!! File: nonexistent.xml
```

El error que se muestra significa que el simulador funciona correctamente ya que su argumento es un archivo inexistente. Sin embargo, el binario del simulador se invocó sin problemas.

Es importante verificar que el programa de visualización de definiciones de geometría funcione de forma correcta, esto se logra con el siguiente comando:

```
CSXGeomPlot('nonexistant.xml')
```

La ventana de comandos debe arrojar un error ya que el archivo de geometría no existe, sin embargo se debe ejecutar el visualizador como se muestra en la figura 6.

Llegado a este punto se puede concluir que openEMS fue instalado de forma correcta.

Se recomienda instalar el programa ParaView para tener una herramienta para visualizar las simulaciones. Los ejemplos de este documento utilizan ParaView para representar de forma gráfica los resultados.

## 2.4. Ejemplos de uso

### 2.4.1. Guía de onda superficial

Las instrucciones para generar la geometría, su excitación y la definición de las variables a observar son las siguientes:



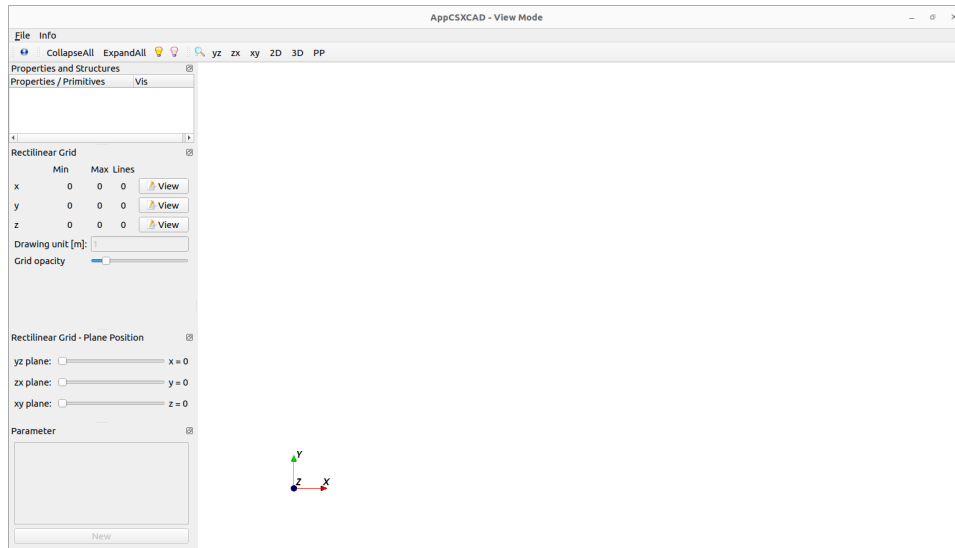


Figura 6: Programa de visualización de geometrías.

```
close all
clear
clc
FDTD = InitFDTD('NrTS',100, 'EndCriteria',0, 'OverSampling',50);
FDTD = SetSinusExcite(FDTD,10e6);
FDTD = SetBoundaryCond(FDTD,{'PMC' 'PMC' 'PEC' 'PEC' 'MUR' 'MUR'});
CSX = InitCSX();
mesh.x = -10:10;
mesh.y = -10:10;
mesh.z = -10:30;
CSX = DefineRectGrid(CSX, 1, mesh);
CSX = AddExcitation(CSX,'excitation',0,[0 1 0]);
CSX = AddBox(CSX,'excitation',0,[-10 -10 0],[10 10 0]);
CSX = AddDump(CSX,'Et');
CSX = AddBox(CSX,'Et',0,[-10 0 -10],[10 0 30]);
mkdir('tmp');
WriteOpenEMS('tmp/tmp.xml',FDTD,CSX);
```

Antes de lanzar la simulación se procede a verificar la geometría generada con los siguientes comandos:

```
CSXGeomPlot( 'tmp/tmp.xml' );
```

En la figura 7 se puede observar el plano que simula una guía de onda.

Una vez satisfecho con la geometría, se procede a lanzar la simulación. Para eso, primero debe cerrar el visualizador de geometría y escribir el siguiente comando:

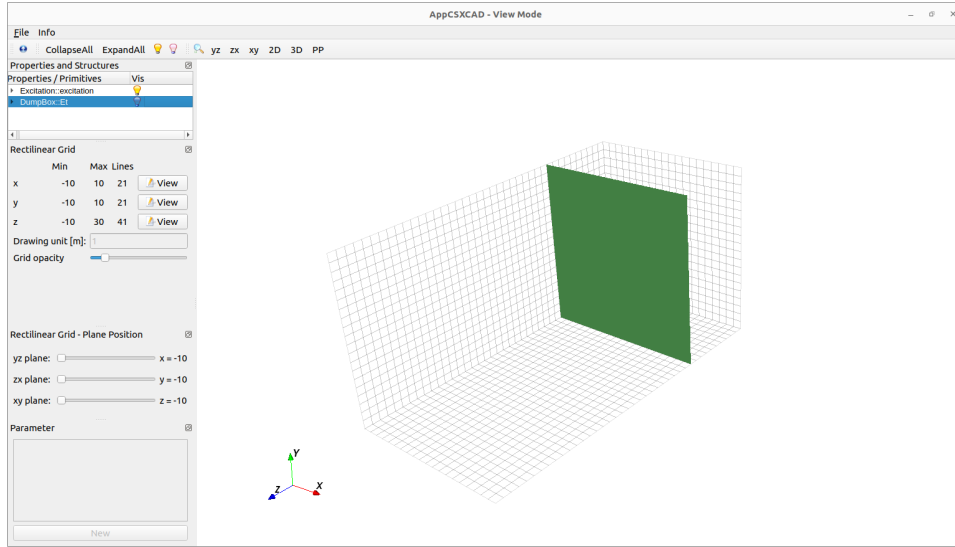


Figura 7: Geometría del ejemplo 1.

```
RunOpenEMS('tmp', 'tmp.xml', '');
```

Para visualizar la simulación se puede usar el programa ParaView. Para eso se debe abrir el archivo `Et.vtr`, luego debe ingresar al inspector de objetos y seleccionar que el color se asocie al  $E\text{-Field}$  además de indicar el eje  $Y$ . Seguidamente, puede hacer click en el boton de *play*. Se verá una animación con la variación del potencial eléctrico en el tiempo, sin embargo los colores se encuentran sobre el plano y no se puede apreciar con facilidad la onda.

Para mejorar la representación se debe colocar el filtro *warp-by-vector* que deformará la representación según la intensidad del campo, esto genera una representación en donde se puede observar con facilidad la onda en la guía.

En la figura 8 se puede observar la guía de onda simulada en un plano.

#### 2.4.2. Esfera metálica dispersa un frente de onda

En este ejemplo se simula la dispersión que produce una esfera metálica cuando una onda incide sobre ella. Para tal fin se generó una geometría esférica con la propiedad física de ser un conductor eléctrico. Se la puede ver en la figura 9.

Luego se definió un cubo como volumen de simulación lo suficientemente grande para que el cálculo numérico sea lo más preciso posible pero que los tiempos de cómputo sean aceptables. El volumen se puede observar en la figura 10.

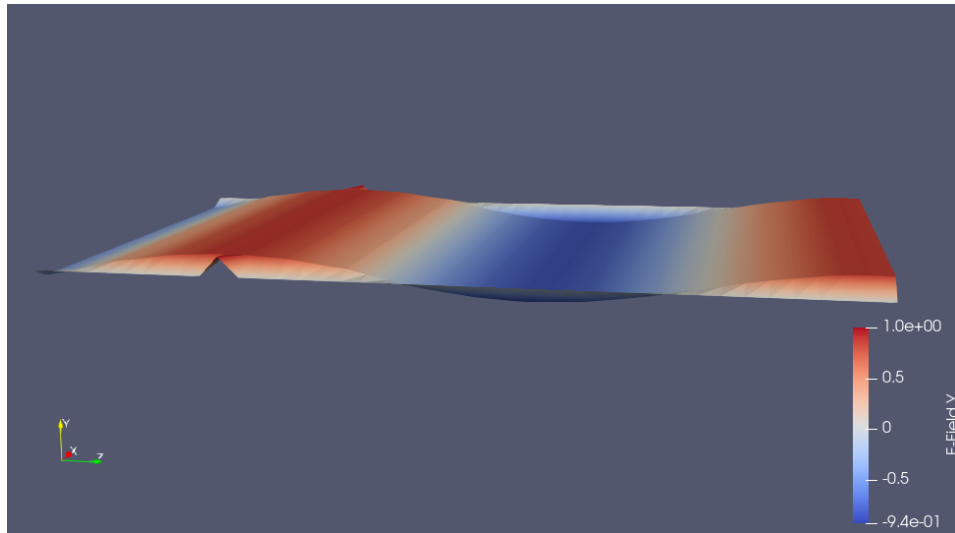


Figura 8: Simulación del ejemplo 1.

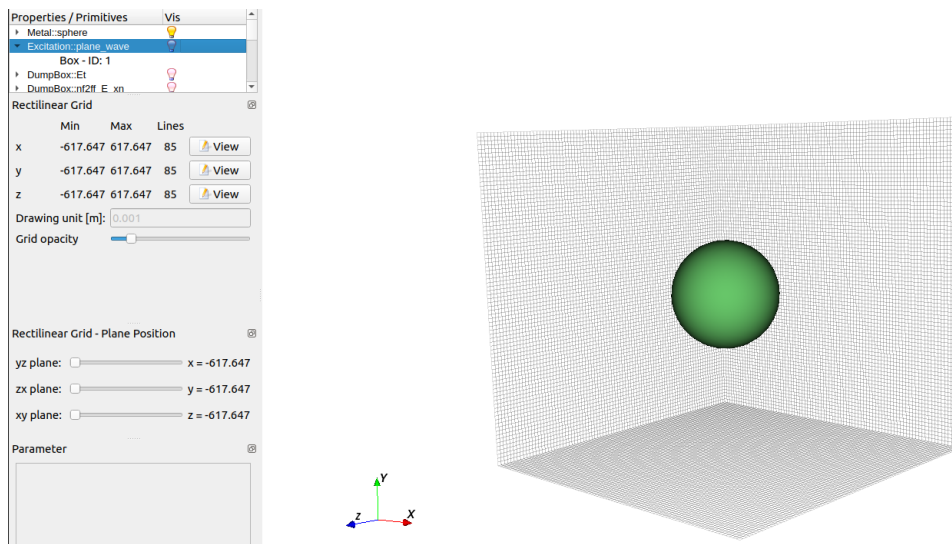


Figura 9: Esfera metálica del ejemplo 2.

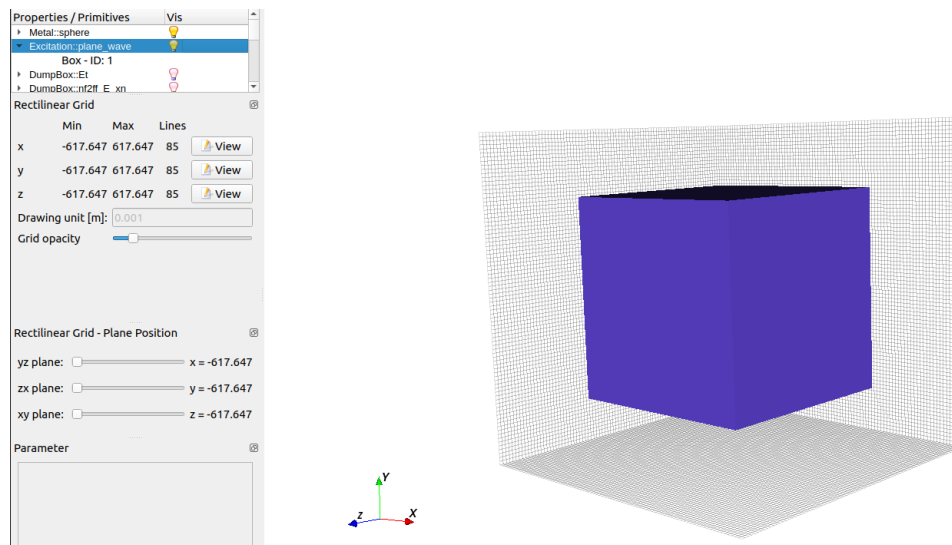


Figura 10: Volumen de simulación del ejemplo 2.

Luego se corrió la simulación y se la visualizó con ParaView. Se tomaron capturas en 3 momentos distintos y se los muestran en 2 y 3 dimensiones. Se pueden observar en las figuras 11,12,13,14,15 y 16.

A continuación se describe el código para generar el ejemplo:

```
% Si es la primera ejecución de octave de la sesión de trabajo
% agregar openEMS al path, sino physical_constants no va a estar
% definido
addpath('~/opt/openEMS/share/openEMS/matlab');
addpath('~/opt/openEMS/share/CSXCAD/matlab');

close all
clear
clc

%% Se configura la simulación
physical_constants;
unit = 1e-3; % longitudes en mm

sphere.rad = 200;

inc_angle = 0 /180*pi; % ángulo incidente (x-axis) en rad

% Tamaño de la caja de simulación
SimBox = 1000;
PW_Box = 750;
```

```

%% configuración FDTD parámetros y función de excitación
f_start = 50e6; % frecuencia de inicio
f_stop = 1000e6; % frecuencia de fin
f0 = 500e6;

FDTD = InitFDTD( );
FDTD = SetGaussExcite( FDTD, 0.5*(f_start+f_stop), 0.5*(f_stop-f_start) );
BC = [1 1 1 1 1 1]*3; % set boundary conditions
FDTD = SetBoundaryCond( FDTD, BC );

%% configurar CSXCAD geometría y malla
max_res = c0 / f_stop / unit / 20; % cell size: lambda/20
CSX = InitCSX();

% creación de malla
smooth_mesh = SmoothMeshLines([0 SimBox/2], max_res);
mesh.x = unique([-smooth_mesh smooth_mesh]);
mesh.y = mesh.x;
mesh.z = mesh.x;

%% creación de esfera de metal
CSX = AddMetal( CSX, 'sphere' ); % se crea un conductor perfecto (PEC)
CSX = AddSphere(CSX, 'sphere', 10, [0 0 0], sphere.rad);

%% excitación de onda planar
k_dir = [cos(inc_angle) sin(inc_angle) 0]; % dirección de la onda planar
E_dir = [0 0 1]; % polarización de la onda planar --> E_z

CSX = AddPlaneWaveExcite(CSX, 'plane_wave', k_dir, E_dir, f0);
start = [-PW_Box/2 -PW_Box/2 -PW_Box/2];
stop = -start;
CSX = AddBox(CSX, 'plane_wave', 0, start, stop);

CSX = AddDump(CSX, 'Et');
start = [mesh.x(1) mesh.y(1) 0];
stop = [mesh.x(end) mesh.y(end) 0];
CSX = AddBox(CSX, 'Et', 0, start, stop);

%% cálculo nf2ff
start = [mesh.x(1) mesh.y(1) mesh.z(1)];
stop = [mesh.x(end) mesh.y(end) mesh.z(end)];
[CSX nf2ff] = CreateNF2FFBox(CSX, 'nf2ff', start, stop);

```

```

mesh = AddPML(mesh,8);

CSX = DefineRectGrid( CSX, unit, mesh );

%% carpeta de simulación
Sim_Path = 'Sphere_RCS';
Sim_CSX = 'Sphere_RCS.xml';

[status, message, messageid] = rmdir( Sim_Path, 's' );
[status, message, messageid] = mkdir( Sim_Path );

WriteOpenEMS( [Sim_Path '/' Sim_CSX], FDTD, CSX );

%% Se muestra la geometría
CSXGeomPlot( [Sim_Path '/' Sim_CSX] );

%% se corre la simulación
RunOpenEMS( Sim_Path, Sim_CSX);

```

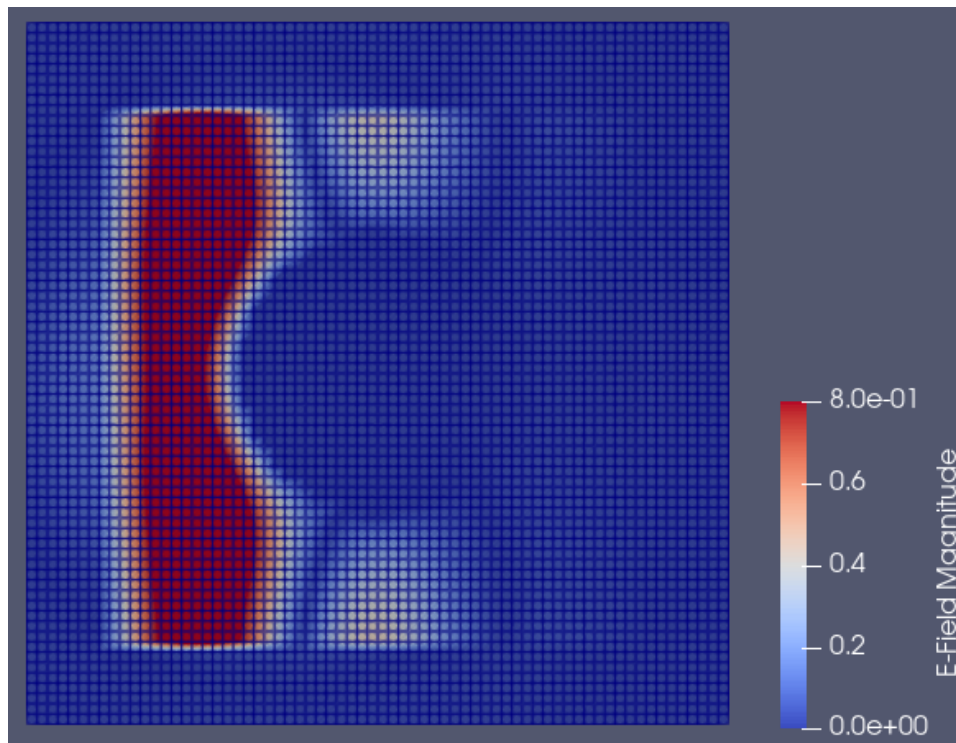


Figura 11: Simulación momento 1.

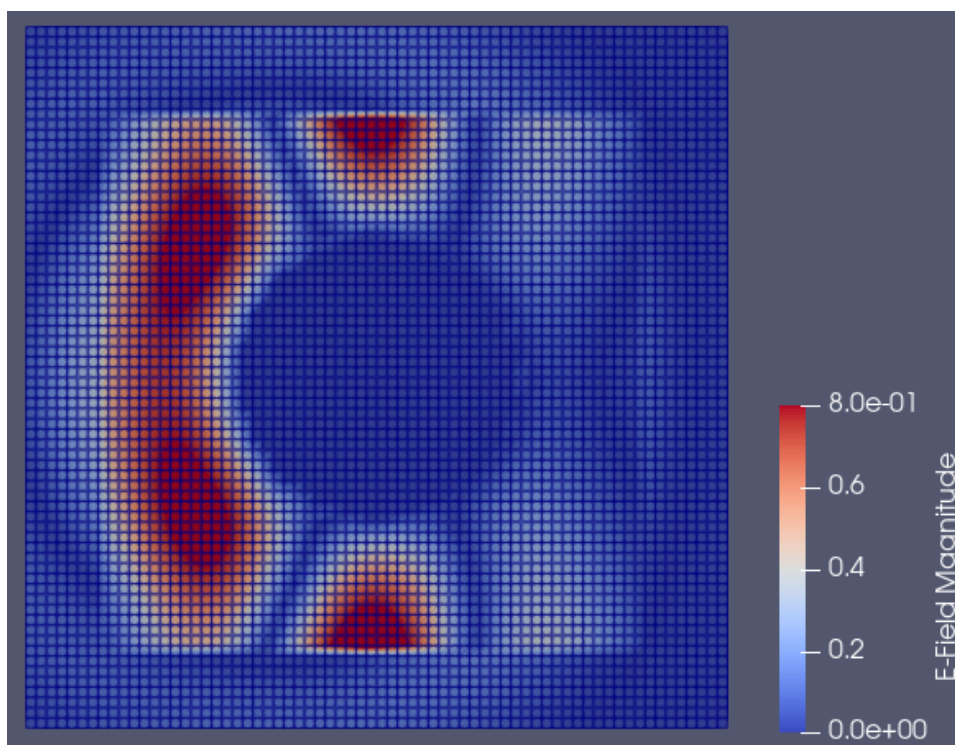


Figura 12: Simulación momento 2.

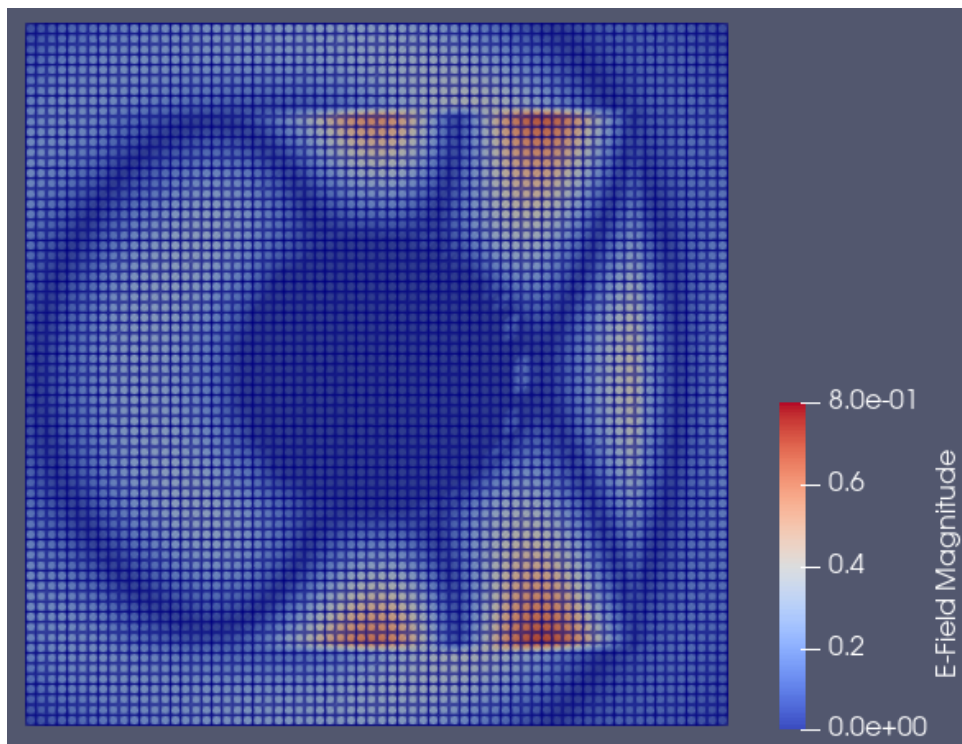


Figura 13: Simulación momento 3.



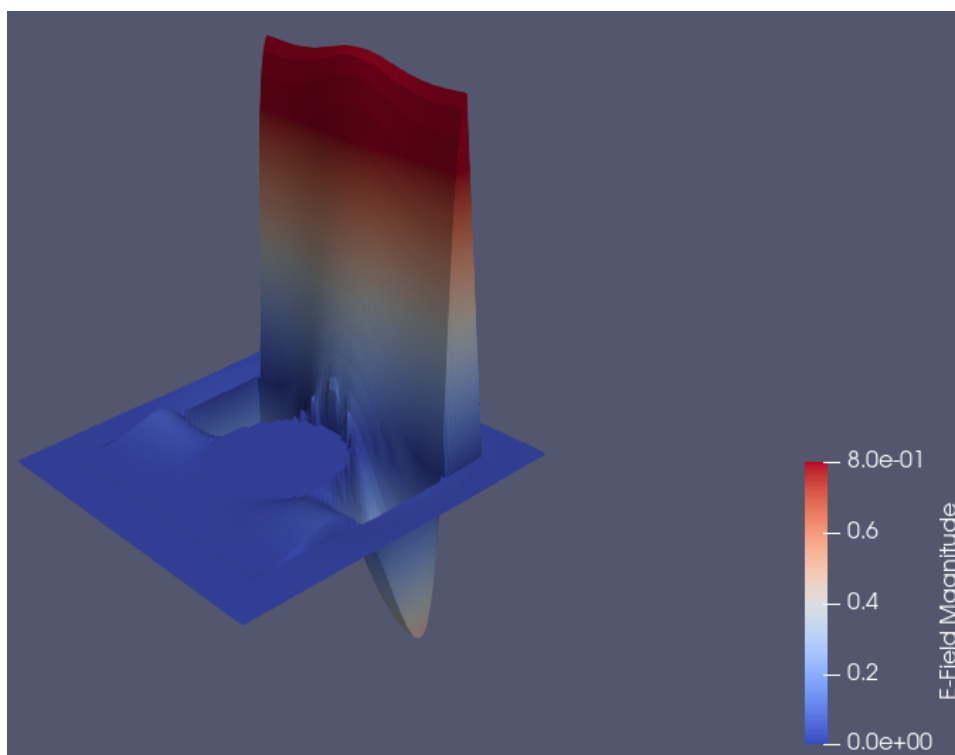


Figura 14: Simulación momento 1 en 3D.

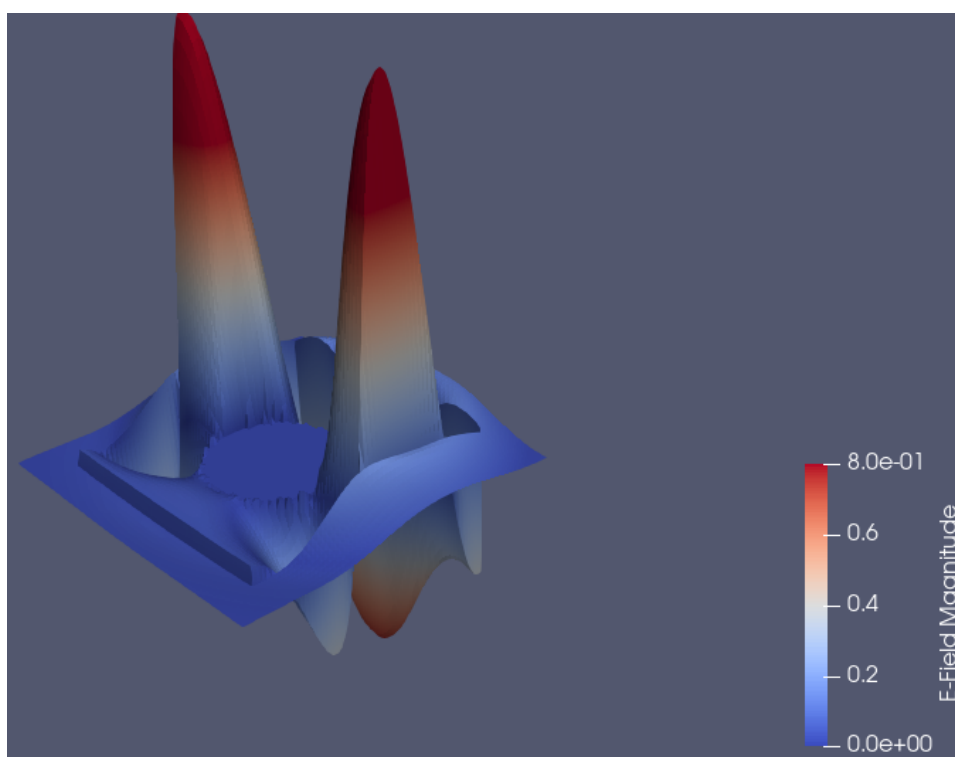


Figura 15: Simulación momento 2 en 3D.

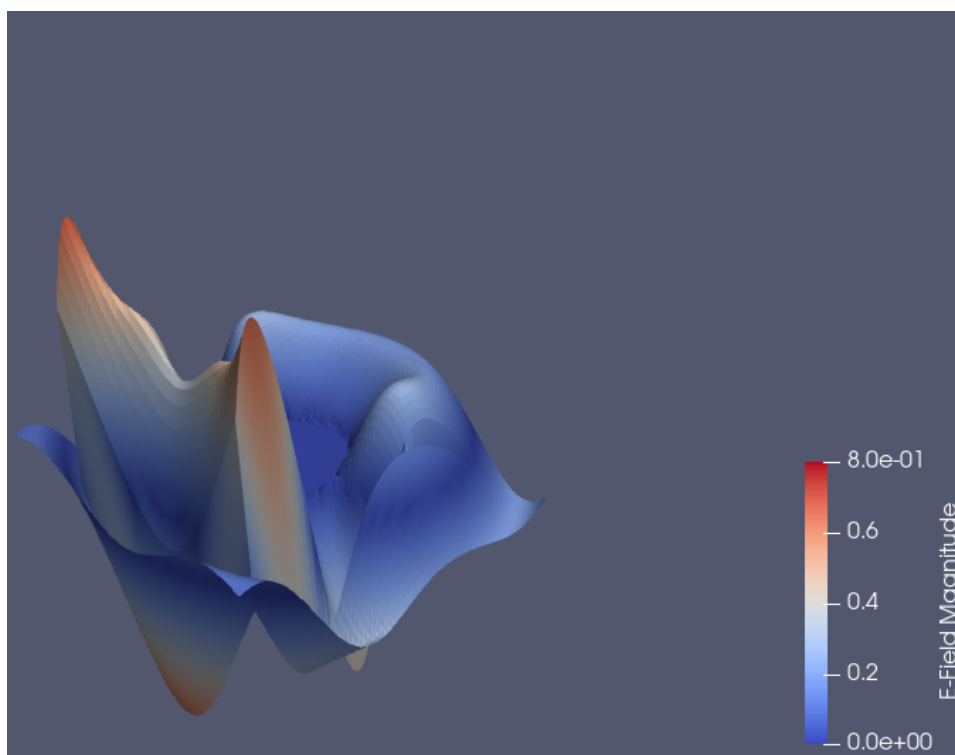


Figura 16: Simulación momento 3 en 3D.

## 2.5. Primeros intentos de simular circuitos impresos

OpenEMS utiliza un *parser* para transformar los archivos *HyperLynx* que exportan programas como KiCad e Eagle en un formato que el motor de matlab pueda entender. Sin embargo, el *parser* hyp2mat es un programa cuyo único autor dió por finalizado y quedaron deprecadas las recetas de Autotools y las bibliotecas que utiliza para *linkear*.

Existen tres posibilidades en este punto:

- Crear un dockerfile que genere una imagen con Octave y OpenEMS en base a una versión de Ubuntu que soporte la última versión de hyp2mat que se publicó.
- Utilizar un método alternativo para importar los circuitos impresos.
- Abandonar OpenEMS como herramienta viable.

## 2.6. Montaje de GNU/Octave y OpenEMS en docker

Se pudo crear una serie de recetas que generan una imagen y contenedor con Octave y OpenEMS.

Dockerfile:

```
# Universidad Nacional de la Matanza.

# Los Dockerfiles comienzan a partir de una imagen base
FROM ubuntu:focal

# En esta sección se actualiza la base de datos del gestor
# de paquetes (apt) y se procede a instalar las herramientas
# y sus dependencias
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -y -q \
    octave-common \
    openems \
    octave-openems \
    libcsxcad0

# Se genera un usuario de desarrollo (dev) con permisos de
# administrador y sin necesidad de ingresar su contraseña
RUN adduser --disabled-password --gecos '' dev && \
    usermod -aG sudo dev && \
    echo "dev ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

# Se hace /bin/sh symlink a bash en vez de dash:
RUN echo "dash dash/sh boolean false" | debconf-set-selections
RUN DEBIAN_FRONTEND=noninteractive dpkg-reconfigure dash
```

```
# Cuando se ingrese al contenedor se ingresara como dev
USER dev
ENV HOME /home/dev
ENV LANG en_US.UTF-8
```

```
# Se crea una carpeta de trabajo y se la configura como
# workdir
RUN mkdir /home/dev/workspace
WORKDIR /home/dev/workspace
```

Receta de creación:

```
#!/bin/bash
docker build \
    -t openems .
```

Receta de lanzamiento del contenedor:

```
#!/bin/bash

# El comentario que comienza con un signo de exclamacion
# tiene la finalidad de indicarle al sistema con que binario
# debe ejecutar este archivo, en nuestro caso el intérprete de
# BASH

# Se define el directorio del host como "path to working directory"
# (PWD), esto es, la carpeta desde donde lanzamos este script.
DIRECTORIO_DE_MI_MAQUINA=PWD

docker run -ti \
    -e DISPLAY=$DISPLAY \
    --net="host" \
    --name="UNLaM" \
    --hostname="UNLAM" \
    -v $DIRECTORIO_DE_MI_MAQUINA:/home/dev/workspace \
    openems
/bin/bash
```

```
Step 9/10 : RUN mkdir /home/dev/workspace  
--> Running in ff216454d37b  
Removing intermediate container ff216454d37b  
--> e2e99e68430f  
Step 10/10 : WORKDIR /home/dev/workspace  
--> Running in b9d1405df162  
Removing intermediate container b9d1405df162  
--> 2ca4330c923b  
Successfully built 2ca4330c923b  
Successfully tagged openems:latest
```

Figura 17: Construcción de imagen

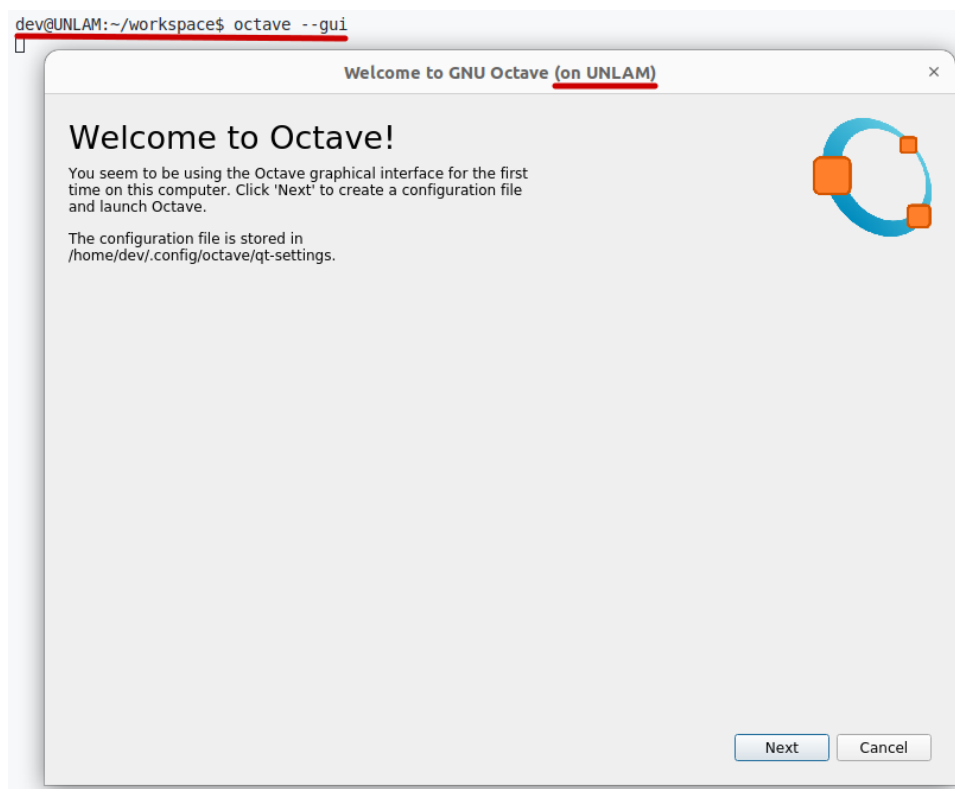


Figura 18: Ejecución del contenedor

### 3. Ansys

Ansys HFSS (simulador de estructura de alta frecuencia) es un solucionador comercial de métodos de elementos finitos para estructuras electromagnéticas (EM) de Ansys que ofrece varias tecnologías de resolución de última generación. Cada solucionador en ANSYS HFSS es un procesador de solución automatizado para el cual el usuario dicta la geometría, las propiedades del material y el rango requerido de frecuencias de solución.

Los ingenieros utilizan Ansys HFSS principalmente para diseñar y simular componentes electrónicos de alta velocidad y alta frecuencia en sistemas de radar, sistemas de comunicación, satélites, ADAS, microchips, placas de circuito impreso, productos IoT y otros dispositivos digitales y dispositivos RF. El solucionador también se ha utilizado para simular el comportamiento electromagnético de objetos como automóviles y aviones. ANSYS HFSS permite a los diseñadores de sistemas y circuitos simular problemas de EM, como pérdidas por atenuación, acoplamiento, radiación y reflexión.

HFSS captura y simula objetos en 3D, teniendo en cuenta la composición de los materiales y las formas/geometrías de cada objeto. Además, está preparada para ser utilizadas en el diseño de antenas y elementos complejos de circuitos electrónicos de radiofrecuencia, incluidos filtros, líneas de transmisión y empaques.

Durante el transcurso del 2022 se logró un contacto comercial que realizó una propuesta de trabajo a la UNLaM a cambio de licencias y capacitación. Se dió una capacitación sobre la funcionalidad y sinergia entre los distintos motores de físicas.

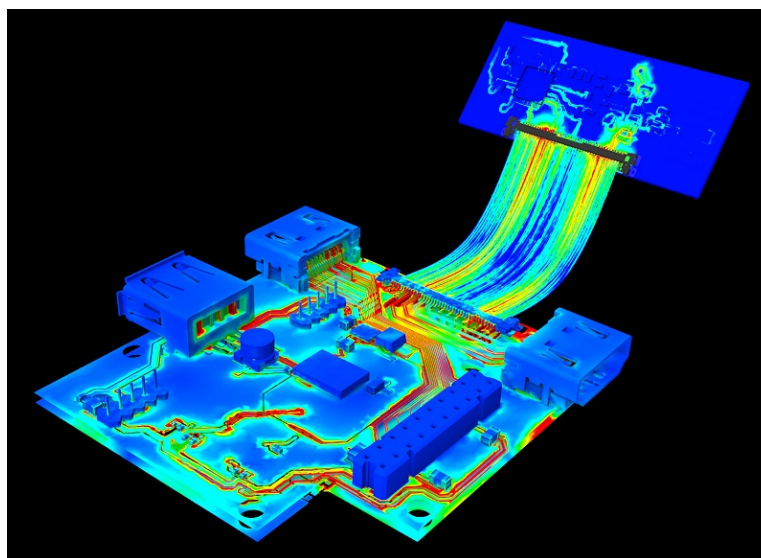


Figura 19: Simulación de PCB flexible en Ansys.

## 4. Elmer

Elmer es un software de simulación multi-física de código abierto desarrollado principalmente por *IT Center for Science (CSC)*. El desarrollo de Elmer se inició como una colaboración nacional con las universidades finlandesas, los institutos de investigación y la industria.

Elmer incluye modelos físicos de dinámica de fluidos, mecánica estructural, electromagnetismo, transferencia de calor y acústica. Estos se describen mediante ecuaciones diferenciales parciales que Elmer resuelve mediante el método de elementos finitos (FEM). Elmer admite computación paralela.

Actualmente los campos de uso más destacados son la glaciología computacional y el electromagnetismo computacional.

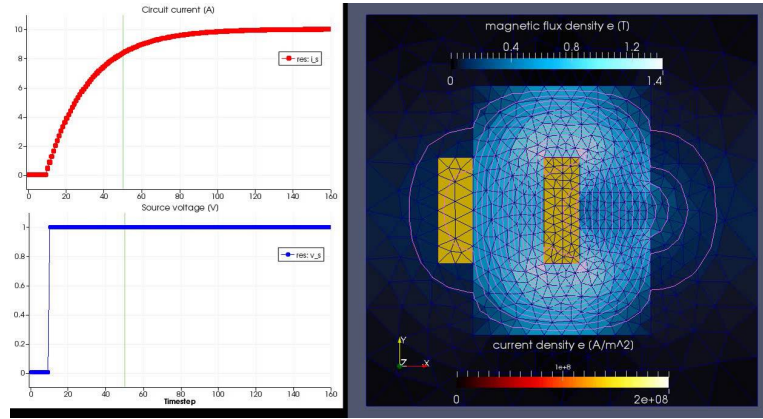


Figura 20: Simulación en Elmer.



## 5. Conclusiones

- OpenEMS: el programa no logró cumplir las expectativas y no se ajusta a las necesidades académicas. No se recomienda para su uso ni continuar su investigación.
- Ansys: a pesar que el programa es propietario y demandaría un costo económico para la universidad, se ajusta a las necesidades académicas. Se recomienda continuar con su investigación.
- Elmer: se debe comenzar su investigación. Esto significa comenzar con sus pruebas de simulación y flujo de trabajo.