

# Herramientas Computacionales para Matemática Aplicada

Curso 2020



Introducción al uso de  
herramientas para CFD

# Que es “CFD”:

Este acrónimo —adoptado directamente del inglés— hace referencia a la rama de la Mecánica de Fluidos denominada *Computational Fluid Dynamics*, traducida normalmente al castellano como *Fluidodinámica Computacional* o *Dinámica de Fluidos Computacional*, y que consiste en el empleo de computadores y de técnicas numéricas para resolver todos aquellos problemas físicos que están relacionados con el movimiento de los fluidos y, en ocasiones, de otros fenómenos asociados como la transferencia de calor, las reacciones químicas, el arrastre de sólidos, etc.

En general, el CFD comprende un amplio abanico de disciplinas científicas, entre las que cabe destacar a las matemáticas, la programación, las ciencias físicas y la ingeniería, que deben aunarse para dar lugar el desarrollo de un **código** que sea capaz de resolver las ecuaciones del flujo de manera satisfactoria.

Por tanto, el objetivo final es la creación de un software (programa numérico) que proporcione el cálculo detallado del movimiento de los fluidos por medio del empleo del ordenador (capaz de ejecutar una gran cantidad de cálculos por unidad de tiempo) para la resolución de las ecuaciones matemáticas que expresan las leyes por las que se rigen los fluidos.

## 1.2 Reseña histórica sobre el CFD

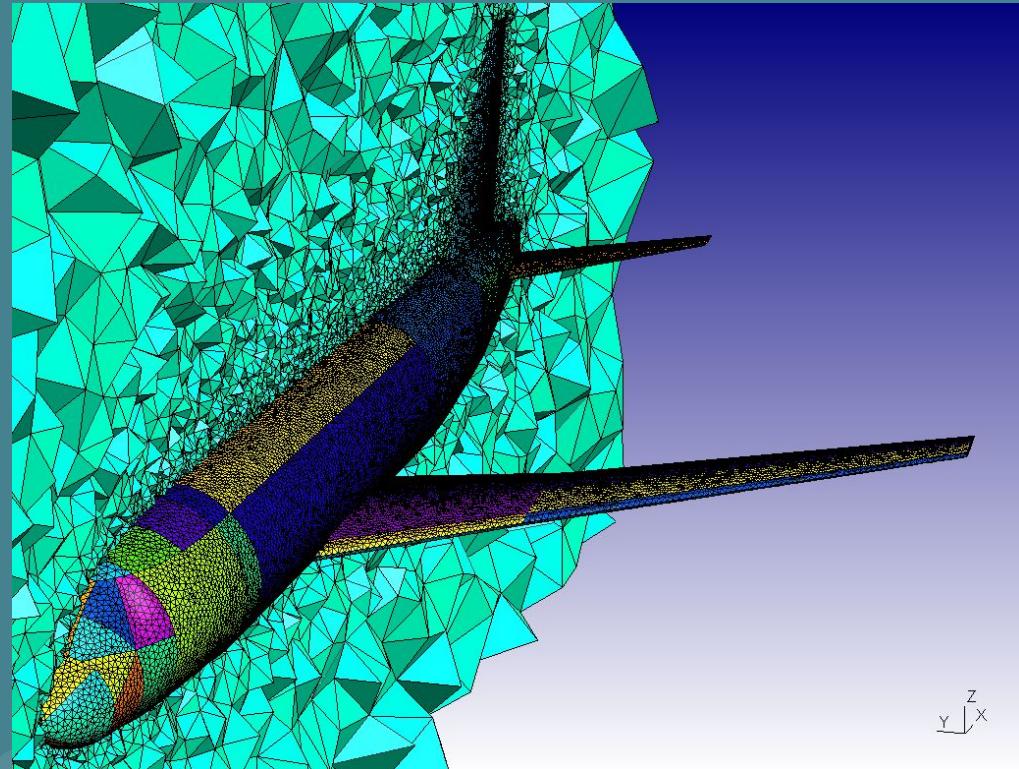
Lógicamente, la consolidación de estas técnicas ha sido una consecuencia más del progresivo desarrollo de los computadores desde los años 1950-1960. Hasta entonces, los métodos computacionales para resolver numéricamente las ecuaciones del flujo no eran viables, al no disponerse de máquinas capaces de ejecutar un gran número de operaciones de cálculo por unidad de tiempo.

Hace ya casi dos siglos desde que las ecuaciones de gobierno de la Mecánica de Fluidos quedaron definitivamente formuladas por Claude **Navier** (1785-1836) y George **Stokes** (1819-1903) cuando introdujeron los términos de transporte viscoso a las ecuaciones de Euler (1707-1783), dando lugar a las famosas ecuaciones de Navier-Stokes:

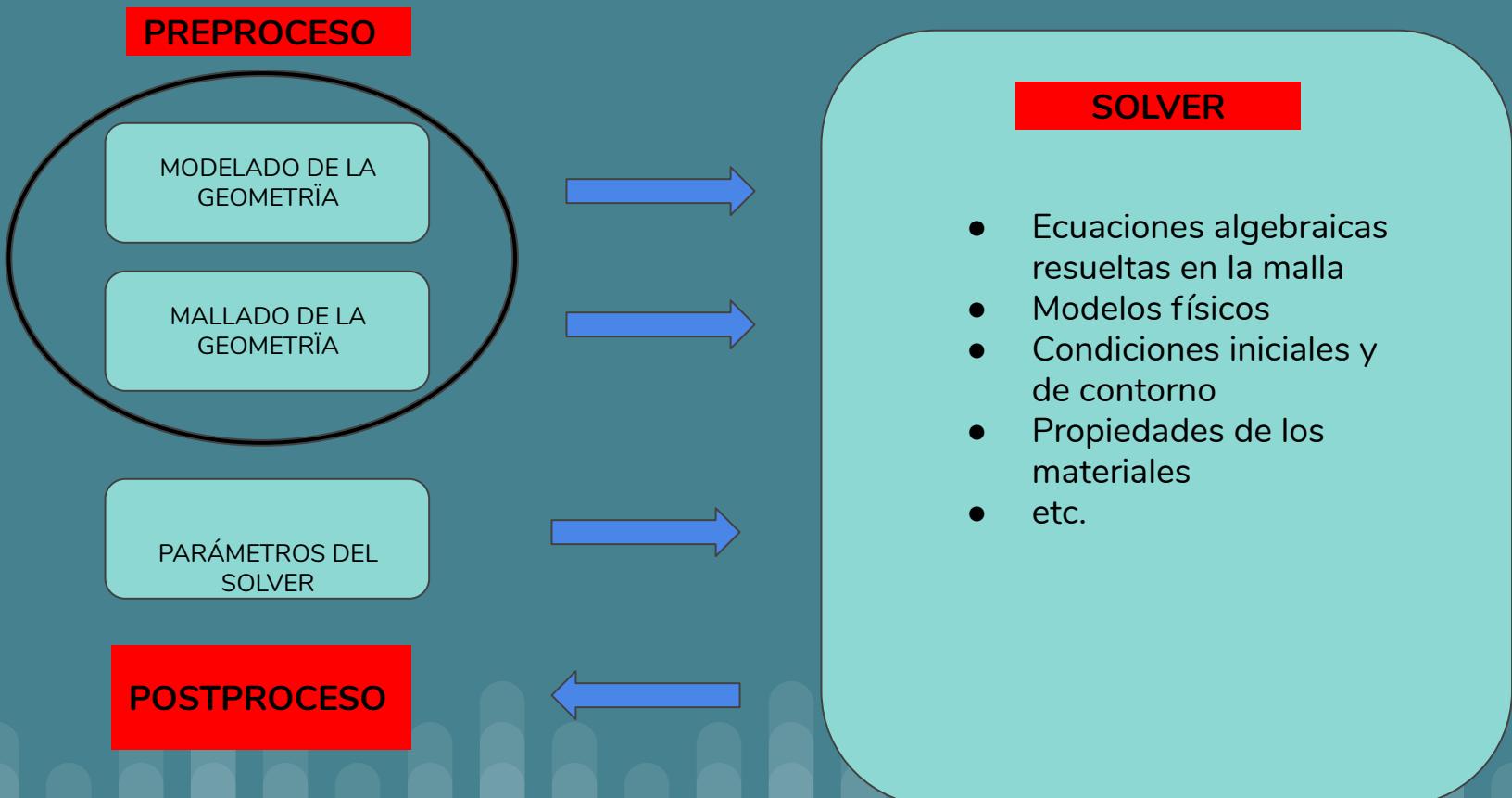
$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \quad [1.1]$$

$$\rho \frac{\partial \vec{v}}{\partial t} + \rho (\vec{v} \cdot \nabla) \vec{v} = -\nabla p + \rho \vec{g} + \nabla \cdot \tau_{ij} \quad [1.2]$$

$$\rho \frac{\partial E}{\partial t} + \rho \nabla \cdot (\vec{v} E) = \nabla \cdot (k \nabla T) + \rho \vec{g} + \nabla \cdot (\bar{\sigma} \cdot \vec{v}) + \dot{W}_f + \dot{q}_H \quad [1.3]$$



# Estructura de un código de CFD típico



# Introducción al método de Diferencias Finitas:

## Método de Diferencias Finitas

Aproxima las derivadas en las ecuaciones diferenciales por su expresión en forma de serie de Taylor truncadas. Por ejemplo, consideremos el problema de valores iniciales

$$\begin{cases} \frac{dy}{dx} + 3y = 0 & 0 \leq x \leq 1 \\ y(0) = 5 \end{cases}$$

Por separación de variables, es fácil resolver el problema de manera analítica:

$$\frac{dy}{y} = -3 dx \rightarrow \ln|y| = -3x + K$$

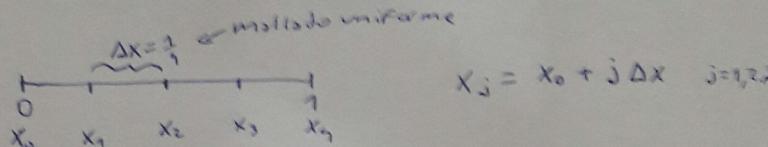
$$y = C e^{-3x} \quad y(0) = C = 5$$
$$y(x) = 5e^{-3x}$$

Aunque en este caso pudimos resolver de manera exacta el problema, consideremos como sería una solución al mismo usando el método de diferencias finitas.

# Introducción al método de Diferencias Finitas:

Como primer paso, debemos discretizar el dominio de interés. En este caso 1D podemos pensar en una partición del intervalo  $[0,1]$ . Pero en casos 2D y 3D uno en general se refiere a esto como "hacer un mallaado del dominio".

Ej:



Los puntos  $x_0, x_1, \dots$  son los nodos del mallaado.

Llámese  $y_j = y(x_j)$ , representando el valor de  $y$  sobre los nodos.

A partir del desarrollo en serie de Taylor de  $y(x)$  obtenemos

$$y_{i+1} = y_i - \Delta x \left( \frac{dy}{dx} \right)_i + \frac{(\Delta x)^2}{2} \left( \frac{d^2y}{dx^2} \right)_i - \frac{(\Delta x)^3}{3!} \left( \frac{d^3y}{dx^3} \right)_i + \dots$$

En este caso, hemos empleado un esquema de

# Introducción al método de Diferencias Finitas:

diferencias "hacia atrás", aunque análogamente podríamos haber empleado un esquema de "diferencias hacia adelante" o de "diferencias centradas". Despreciando términos de orden superior y reordenando la expresión se obtiene

$$\left(\frac{dy}{dx}\right)_i = \frac{y_i - y_{i-1}}{\Delta x} + O(\Delta x)$$

El error que se comete por despreciar los términos de orden superior en la serie de Taylor se denomina "error de truncamiento". En este caso, puesto que el error de truncamiento es de orden  $O(\Delta x)$ , se dice que la representación discreta tiene una precisión de "primer orden". Combinando esta expresión con la ED original obtenemos el sistema de ecuaciones algebraicas (una ecuación por cada nodo)

$$\frac{y_i - y_{i-1}}{\Delta x} + \beta y_i = 0$$

La obtención de la solución aproximada pasa por resolver este sistema de ecuaciones algebraicas.

# Introducción al método de Diferencias Finitas:

El sistema de ecuaciones puede escribirse como

$$-y_{i+1} + (1 + \alpha)x_i y_i = 0$$

$i = 1, 2, \dots, N-1$

Número de nodos

$$y_0 = 5 \leftarrow c_i$$

Por ejemplo, si  $N=5$ , el sistema de ecuaciones puede escribirse matricialmente como

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1+\alpha & 0 & 0 & 0 \\ 0 & -1 & 1+\alpha & 0 & 0 \\ 0 & 0 & -1 & 1+\alpha & 0 \\ 0 & 0 & 0 & -1 & 1+\alpha \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

La inversión de matrices en sistemas con millones de nodos puede ser inabordable. Típicamente, el número de operaciones necesarias para invertir una matriz de  $\mathbb{R}^{n \times n}$  escala con  $n^3$ .

No obstante, para matrices "rals" ("sparse matrices") como la anterior, existen algoritmos más eficientes (orden  $O(n)$ ). Por ejemplo, el sistema anterior puede resolverse de modo sencillo mediante la relación de recurrencia

$$\begin{cases} y_i = \frac{y_{i-1}}{1+\alpha} & i \geq 1 \\ y_0 = 5 & \end{cases}$$

# Solución numérica de ejemplo simple por el método de Diferencias Finitas:

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <string>

using namespace std;

int main(int numarg,char **cargs)
{
ofstream Outfile;
string Encabezado("# x_node y(x_node) ");

Outfile.open ("Datos.dat",ios_base::out);
Outfile.precision(6);

Outfile << Encabezado << endl;

int NumIntervalos = 10;
double DeltaX = 1.0/NumIntervalos;
std::vector<double> y(NumIntervalos+1); //el número de nodos es NumIntervalos+1
y[0] = 5; //condición inicial

Outfile << 0 << "    " << y[0] << endl;
```

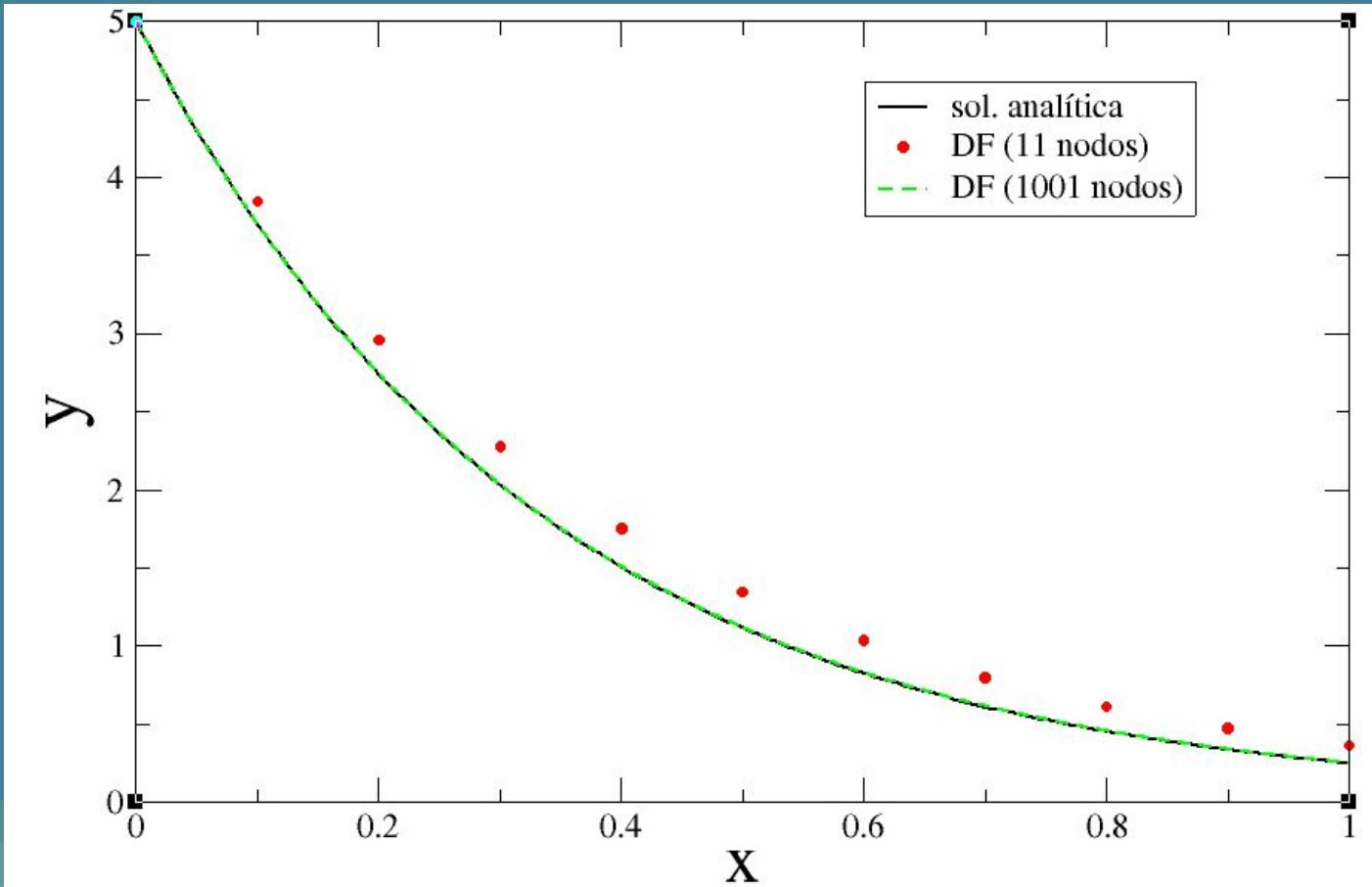
# Solución numérica de ejemplo simple por el método de Diferencias Finitas:

```
for (int i = 1; i < NumIntervalos+1; ++i){ //sol. iterativa para el resto de los nodos  
    y[i] = y[i-1]/(1+3*DeltaX);  
    Outfile << i*DeltaX << " " << y[i] << endl;  
}  
  
cout << "Listo, solución por diferencias finitas guardadas en el archivo " << "Datos.dat" << endl;  
return(0);  
}
```

```
$ g++ EjDiffFinitas.cpp -lm  
$ ./a.out  
Listo, solución por diferencias finitas guardadas en el archivo Datos.dat  
$ gedit Datos.dat &
```

#	x_node	y(x_node)
0	5	
0.1	3.84615	
0.2	2.95858	
0.3	2.27583	
0.4	1.75064	
0.5	1.34665	
0.6	1.03588	
0.7	0.796832	
0.8	0.612947	
0.9	0.471498	
1	0.362691	

# Solución numérica de ejemplo simple por el método de Diferencias Finitas:



# El Método de los Elementos Finitos (FEM):

Este método se basa en una representación funcional de la solución numérica. En lugar de obtener una representación discreta de la solución (en un conjunto de puntos previamente definidos), las variables se resuelven como combinación lineal de varias funciones continuas denominadas **base**  $v_i$ , de forma que se cumple:

$$\phi = \sum_{i=1}^N \phi_i v_i \quad [2.6]$$

En este caso la solución no será exacta, sino que dará lugar a un **residuo**, cuya minimización por medio de funciones peso caracterizará el método. La definición de la base de funciones es la que da nombre al método, elementos finitos, en contraposición con otra familia de métodos residuales, denominada espectral.

# Lista parcial de software disponible para FEM:

Elmer	Open source multiphysical simulation software developed by Finnish Ministry of Education's CSC, written primarily in Fortran (written in Fortran90, C and C++)	CSC	8.2	2016-03-15	GPL	Free	Linux, Mac OS X, Windows
Abaqus	Advanced Franco-USA software from <a href="#">SIMULIA</a> , owned by Dassault Systemes	Abaqus Inc.	2019	2019-12	Proprietary commercial software		Linux, Windows
CONSELF	CAE simulation from your browser	CONSELF SRL	2.9	2015-10	SaaS	Freemium	Web browser
FreeCAD	Parametric 3D modeler with a FEM workbench allowing it to use external solvers like CalculiX, Z88, Elmer, and OpenFoam	FreeCAD Team	0.18	12 March 2019	LGPL 2	Free	Linux, Windows, Mac OS X
ANSYS	US-based and -developed full CAE software package	Ansys Inc.	19.2	2018-09-18	Proprietary commercial software	Free student version available, up to 32,000 nodes/elements <sup>[8]</sup>	Windows, Linux
COMSOL Multiphysics	COMSOL Multiphysics Finite Element Analysis Software (formerly FEMLAB)	COMSOL Inc.	5.4	2018-10-03	Proprietary EULA		Linux, Mac OS X, Windows, Web browser

# El Método de los Volúmenes Finitos (FVM):

The **finite volume method (FVM)** is a method for representing and evaluating partial differential equations in the form of algebraic equations.<sup>[1]</sup> In the finite volume method, volume integrals in a partial differential equation that contain a **divergence** term are converted to surface integrals, using the **divergence theorem**. These terms are then evaluated as fluxes at the surfaces of each finite volume. Because the flux entering a given volume is identical to that leaving the adjacent volume, these methods are **conservative**. Another advantage of the finite volume method is that it is easily formulated to allow for unstructured meshes. The method is used in many computational fluid dynamics packages. "Finite volume" refers to the small volume surrounding each node point on a mesh.

Alternative to the **finite difference method** or **finite element method**, values are calculated at discrete places on a meshed geometry.<sup>[clarification needed]</sup>

# Métodos de resolución iterativos:

Como ya mencionamos, en sistemas grandes (digamos millones de nodos), la inversión de matrices por técnicas usuales es inviable por involucrar algoritmos  $O(N^3)$ , salvo cuando las matrices son de ciertas formas especiales, (“ very sparse matrices”) tales como las tridiagonales que pueden invertirse con algoritmos  $O(N)$

Además, los sistemas algebraicos resultantes frecuentemente son NO LINEALES. Por todo esto, suele ser conveniente (desde un punto de vista de recursos computacionales) optar por algoritmos iterativos de solución, en los cuales en vez de la solución exacta al problema de la inversión de matrices se obtiene una solución aproximada, dentro de cierta tolerancia determinada por el usuario.

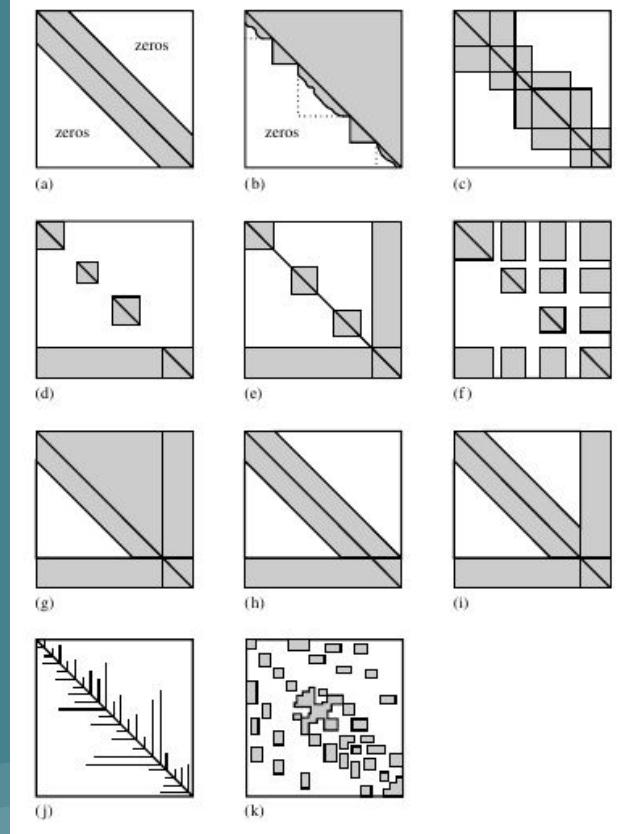


Figure 2.7.1. Some standard forms for sparse matrices. (a) Band diagonal; (b) block triangular; (c) block tridiagonal; (d) singly bordered block diagonal; (e) doubly bordered block diagonal; (f) singly bordered block triangular; (g) bordered band-triangular; (h) and (i) singly and doubly bordered band diagonal; (j) and (k) other! (after Tewarson [1]).

# Aspectos comunes a DF, FEM y FVM:

A pesar de sus diferencias, los tres métodos comparten las siguientes características:

- Identificación de los fenómenos físicos y químicos que deben modelarse.
- Especificación de las condiciones iniciales y de contorno del problema.
- Definición de la geometría a modelizar: el dominio computacional.
- Generación de la malla que partitiona el dominio en un número suficientemente grande de elementos.
- Las ecuaciones que sea necesario resolver se discretizan y linealizan para obtener un sistema algebraico de ecuaciones.

# Software CFD Open Source: OpenFoam

## OpenFOAM

Este software nació en el [Imperial College de Londres](#), una institución con una fuerte tradición en investigación en el campo de la [mecánica de fluidos](#), el cual ha sido centro investigador de técnicas relacionadas con el CFD ([Dinámica de Fluidos Computacional](#)) desde el año [1960](#).

**OpenFOAM** (Open Field Operation and Manipulation) es un software CFD gratuito y de código abierto, lanzado y desarrollado principalmente por [OpenFoam Ltd](#) desde 2004, siendo distribuido más tarde por la [Fundación OpenFoam](#). Cuenta con una amplia base de usuarios en la mayoría de las áreas de ingeniería y ciencias, tanto de organizaciones comerciales como académicas. OpenFoam está organizado en un conjunto de módulos [C++](#) que posibilitan la resolución de problemas, desde complejos flujos de fluidos que involucran [reacciones químicas](#), [turbulencias](#) y [transferencia de calor](#), hasta [acústica](#), [mecánica sólida](#) y [electromagnética](#).<sup>2</sup>

OpenFoam se actualiza profesionalmente cada seis meses para incluir desarrollos patrocinados por los clientes y contribuciones de la comunidad. Es probado de forma independiente por especialistas en aplicaciones del tipo [OpenCFD](#) del [Grupo ESI \(Engineering System International\)](#), los socios de desarrollo y clientes seleccionados, y respaldado por la infraestructura, los valores y el compromiso del grupo ESI a nivel mundial.

La garantía de calidad se basa en pruebas rigurosas diseñadas para evaluar el comportamiento de regresión, el uso de memoria, el rendimiento del código y la escalabilidad, antes de publicar las nuevas versiones.

El equipo de desarrolladores, especialistas en aplicaciones, formadores y evaluadores están ubicados globalmente en el este de [Asia](#), [India](#), [Europa](#) y [América del Norte](#).



OpenFoam funcionando en el gnome-terminal.

Tipo de programa	programa informático
Desarrollador	CFD Direct. <sup>1</sup>
Lanzamiento	2004
Última versión	v5.0
estable	30 de octubre de 2017
Género	Dinámica de Fluidos Computacional, Simulación Software
Programado en	C++
Sistema operativo	Unix/Linux
Licencia	GPLv3
[editar datos en Wikidata]	

# Ramas principales del proyecto:

openfoam.org/version/7/

Apps tools\_mfc\_bo... misc\_mfc\_bo... admin\_mfc\_bo... sci\_mfc\_book... oil\_mfc\_book...

## OpenFOAM

The OpenFOAM Foundation

Download Development Recent

### OpenFOAM v7 | Patch Releases

For Ubuntu 16.04LTS, 18.04LTS, 19.04, 19.10, Windows 10 and Docker images for other Linux and macOS

⌚ 2nd September 2019

[Read More](#)

### OpenFOAM 7 Released

The OpenFOAM Foundation is pleased to announce the release of version 7 of the OpenFOAM open source CFD toolbox. Version 7 is a snapshot of the OpenFOAM development version which, through

# OpenVFOAM

The open source CFD toolbox

Follow us on [Twitter](#)

Home Products Services Download Code Documentation Community Governance News

About us Contact Jobs Legal

**Get OpenFOAM**

**OpenFOAM v1912 Released**  
Download OpenFOAM  
Subscribe to the newsletter

**Our Services**

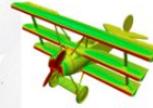
OpenFOAM Training  
OpenFOAM Support  
OpenFOAM Development  
Engineering Services

**More information**

OpenFOAM History  
Official Documentation  
OpenFOAM Governance  
OpenFOAM Community  
Tutorial Wiki

**OpenFOAM core trainings in 2020**

16-19 MAR - Bracknell (UK)  
20-24 APR - Frankfurt (DE)  
11-14 MAY - Stockholm (SW)  
22-25 JUN - Toulouse (FR)  
13-16 JUL - Munich (DE)



>Main News

**ESI OpenCFD releases OpenFOAM-v1912**  
ESI-OpenCFD is pleased to announce the release of OpenFOAM v1912 of the OpenFOAM open source CFD toolbox

Dec 23rd 2019

**ESI OpenCFD releases OpenFOAM-v1906**  
ESI-OpenCFD is pleased to announce the release of OpenFOAM v1906 of the OpenFOAM open source CFD toolbox

Jun 27th 2019

**ESI OpenCFD releases OpenFOAM-v1812**  
ESI-OpenCFD is pleased to announce the release of OpenFOAM v1812 of the OpenFOAM open source CFD toolbox

Dec 20th 2018

[Main News Archive...](#)

# Documentación online:

## 1 Introduction

## 2 Tutorials

### 2.1 Lid-driven cavity flow

2.1.1 Pre-processing

2.1.2 Viewing the mesh

2.1.3 Running an application

2.1.4 Post-processing

2.1.5 Increasing the mesh resolution

2.1.6 Introducing mesh grading

2.1.7 Increasing the Reynolds number

2.1.8 High Reynolds number flow

2.1.9 Changing the case geometry

2.1.10 Post-processing the modified geometry

### 2.2 Stress analysis of a plate with a hole

2.2.1 Mesh generation

2.2.2 Running the code

2.2.3 Post-processing

2.2.4 Exercises

### 2.3 Breaking of a dam

2.3.1 Mesh generation

2.3.2 Boundary conditions

2.3.3 Setting initial field

2.3.4 Fluid properties

2.3.5 Turbulence modelling

2.3.6 Time step control

## 3.5 Standard solvers

3.5.1 'Basic' CFD codes

3.5.2 Incompressible flow

3.5.3 Compressible flow

3.5.4 Multiphase flow

3.5.5 Direct numerical simulation (DNS)

3.5.6 Combustion

3.5.7 Heat transfer and buoyancy-driven flows

3.5.8 Particle-tracking flows

3.5.9 Discrete methods

3.5.10 Electromagnetics

3.5.11 Stress analysis of solids

3.5.12 Finance

## 3.6 Standard utilities

3.6.1 Pre-processing

3.6.2 Mesh generation

3.6.3 Mesh conversion

3.6.4 Mesh manipulation

3.6.5 Other mesh tools

3.6.6 Post-processing

3.6.7 Post-processing data converters

3.6.8 Surface mesh (e.g. OBJ/STL) tools

3.6.9 Parallel processing

3.6.10 Thermophysical-related utilities

3.6.11 Miscellaneous utilities

# Estructura de OpenFoam:

OpenFOAM is first and foremost a *C++ library*, used primarily to create executables known as *applications*. The applications fall into two categories: *solvers*, that are each designed to solve a specific problem in continuum mechanics; and *utilities*, that are designed to perform tasks that involve data manipulation. New solvers and utilities can be created by its users with some pre-requisite knowledge of the underlying method, physics and programming techniques involved.

OpenFOAM is supplied with pre- and post-processing environments. The interfaces to the pre- and post-processing are themselves OpenFOAM utilities, thereby ensuring consistent data handling across all environments. The overall structure of OpenFOAM is shown in Figure 1.1.

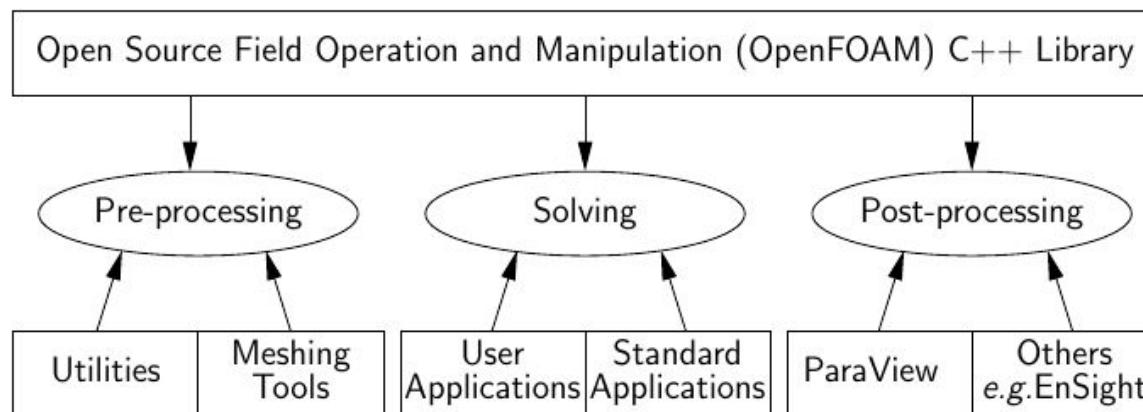


Figure 1.1: Overview of OpenFOAM structure.

# Algunos de los solvers incluídos en OpenFoam:

$$\frac{\partial}{\partial t}(T) - \nabla \cdot (D_T \nabla T) = S_T$$

## 'Basic' CFD codes

laplacianFoam

Laplace equation solver for a scalar quantity

potentialFoam

Potential flow solver which solves for the velocity potential, to calculate the flux-field, from which the velocity field is obtained by reconstructing the flux

scalarTransportFoam

Passive scalar transport equation solver

## Incompressible flow

adjointShape-OptimizationFoam

Steady-state solver for incompressible, turbulent flow of non-Newtonian fluids with optimisation of duct shape by applying "blockage" in regions causing pressure loss as estimated using an adjoint formulation

boundaryFoam

Steady-state solver for incompressible, 1D turbulent flow, typically to generate boundary layer conditions at an inlet

icoFoam

Transient solver for incompressible, laminar flow of Newtonian fluids

# Algunos de los solvers incluídos en OpenFoam:

## Electromagnetics

---

`electrostaticFoam`

Solver for electrostatics

`magneticFoam`

Solver for the magnetic field generated by permanent magnets

`mhdFoam`

Solver for magnetohydrodynamics (MHD): incompressible, laminar flow of a conducting fluid under the influence of a magnetic field

## Stress analysis of solids

---

`solidDisplacementFoam`

Transient segregated finite-volume solver of linear-elastic, small-strain deformation of a solid body, with optional thermal diffusion and thermal stresses

`solidEquilibriumDisplacementFoam`

Steady-state segregated finite-volume solver of linear-elastic, small-strain deformation of a solid body, with optional thermal diffusion and thermal stresses

## Finance

---

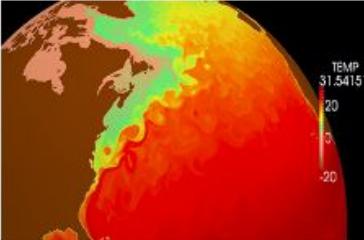
`financialFoam`

Solves the Black-Scholes equation to price commodities

# Una herramienta de post-procesamiento avanzado: Paraview

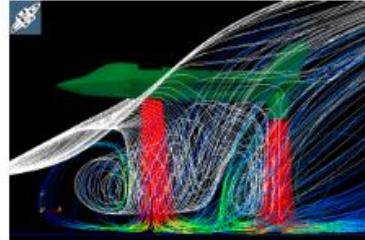
 [About](#) [Flavors](#) [Domains](#) [Resources](#) [Developer Tools](#) [Download](#) 





**Climate Research**

The Climate Data Analysis Tools (CDAT) project leverages ParaView with other open-source tools to enable analysts to track, monitor, and predict climate changes.



**CFD Simulations**

Computational fluid dynamics (CFD) simulations with ParaView enable aviation teams to study lift and drag, and thereby improve design efficiency.

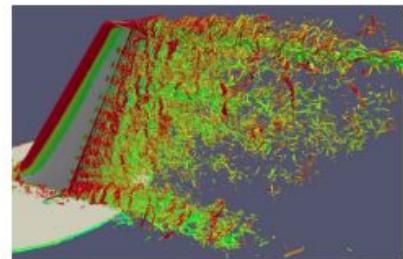


**Immersive Data**

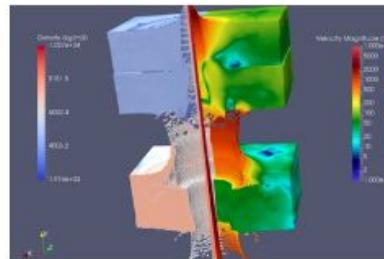
By immersing themselves in the data with ParaView, researchers can interactively explore data in a more intuitive manner.



**Cloud-resolving simulation**



**PHASTA and ParaView Catalyst**



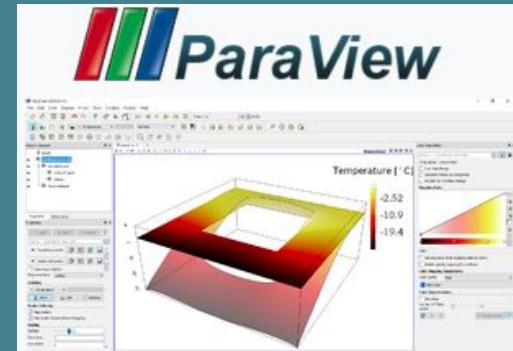
**Shaped Charge Jet Penetration**

# Una herramienta de post-procesamiento avanzada: Paraview

**ParaView** is an open-source multiple-platform application for interactive, scientific visualization. It has a client–server architecture to facilitate remote visualization of datasets, and generates level of detail (LOD) models to maintain interactive frame rates for large datasets. It is an application built on top of the Visualization Toolkit (VTK) libraries. ParaView is an application designed for data parallelism on shared-memory or distributed-memory multicompilers and clusters. It can also be run as a single-computer application.

**ParaView** is an open-source, multi-platform data analysis and visualization application. Paraview is known and used in many different communities to analyze and visualize scientific data sets.<sup>[2]</sup> It can be used to build visualizations to analyze data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities.<sup>[3]</sup>

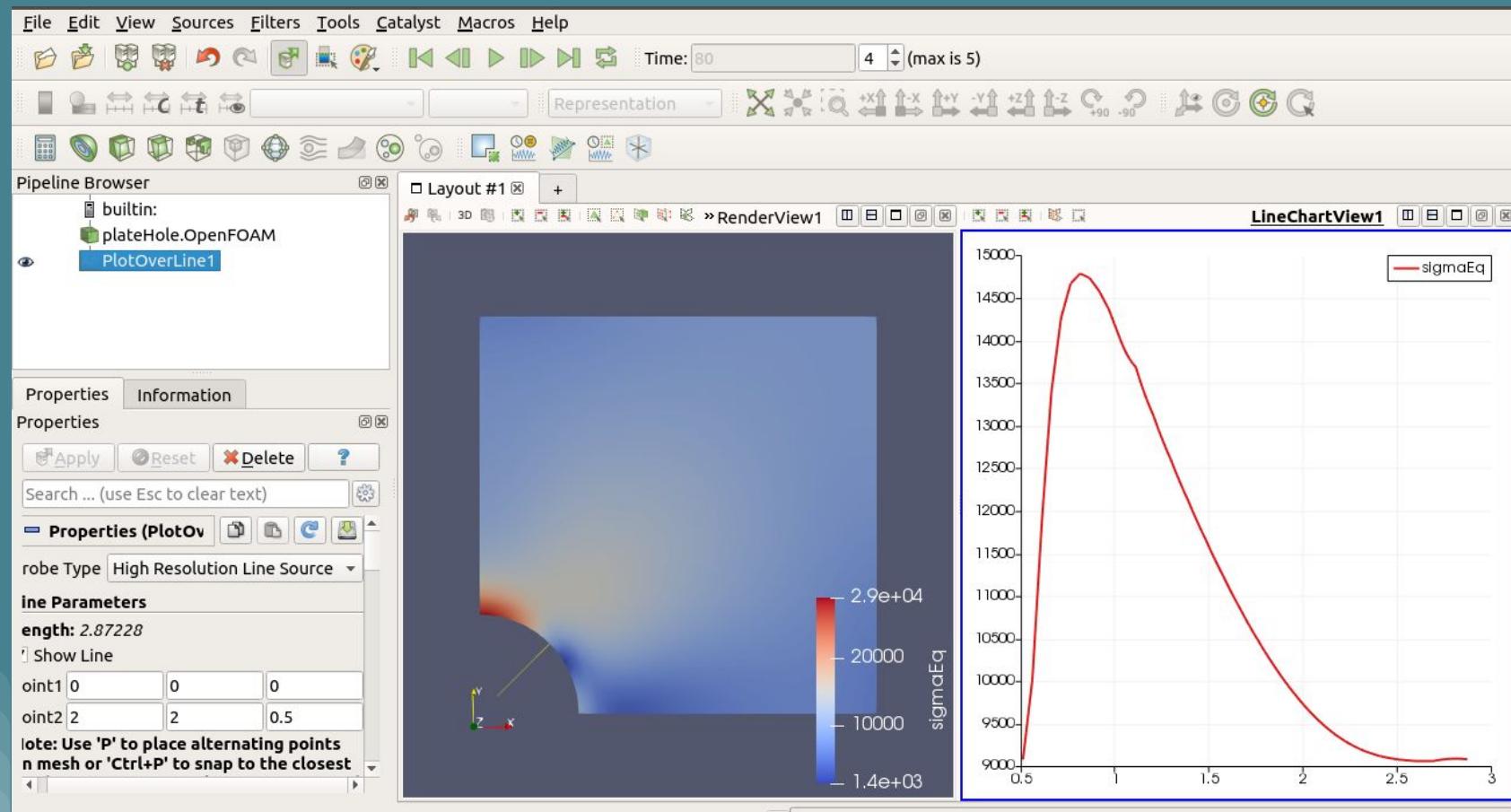
ParaView was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of terascale as well as on laptops for smaller data.<sup>[3]</sup>



Paraview 5.0

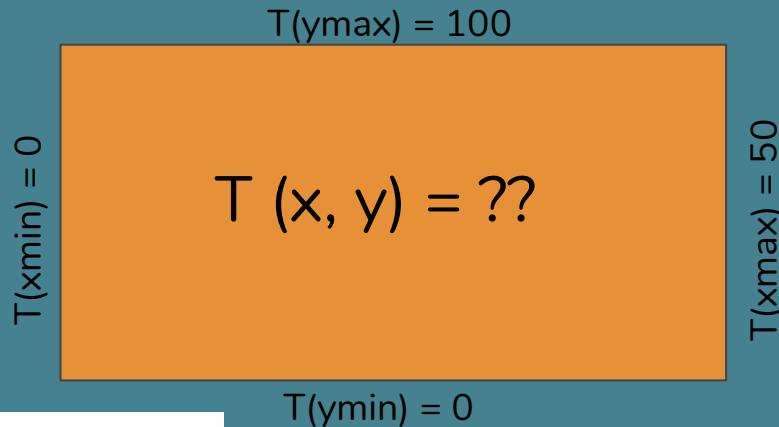
<b>Developer(s)</b>	Sandia National Laboratories, Kitware Inc, Los Alamos National Laboratory
<b>Stable release</b>	5.8.0 / February 18, 2020; 43 days ago <sup>[1]</sup>
<b>Preview release</b>	5.8.0-RC3 / February 11, 2020; 50 days ago <sup>[1]</sup>
<b>Repository</b>	<a href="#">Paraview Repository</a> ↗
<b>Written in</b>	C, C++, Fortran, Python
<b>Operating system</b>	Unix/Linux, macOS, Microsoft Windows
<b>Type</b>	Scientific visualization, Interactive visualization
<b>License</b>	3-clause BSD
<b>Website</b>	<a href="http://www.paraview.org">www.paraview.org</a> ↗

# Una herramienta de post-procesamiento avanzada: Paraview



# Caso de ejemplo: Ecuación de Laplace en un dominio 2D

Problema 1 de la práctica:



## 3.5 Standard solvers

The solvers with the OpenFOAM distribution are in the `$FOAM_SOLVERS` directory, reached quickly by typing `sol` at the command line. This directory is further subdivided into several directories by category of continuum mechanics, e.g. incompressible flow, combustion and solid body stress analysis. Each solver is given a name that is reasonably descriptive, e.g. `icoFoam` solves incompressible, laminar flow. The current list of solvers distributed with OpenFOAM is given in the following Sections.

### 3.5.1 'Basic' CFD codes

#### `laplacianFoam`

Solves a simple Laplace equation, e.g. for thermal diffusion in a solid.

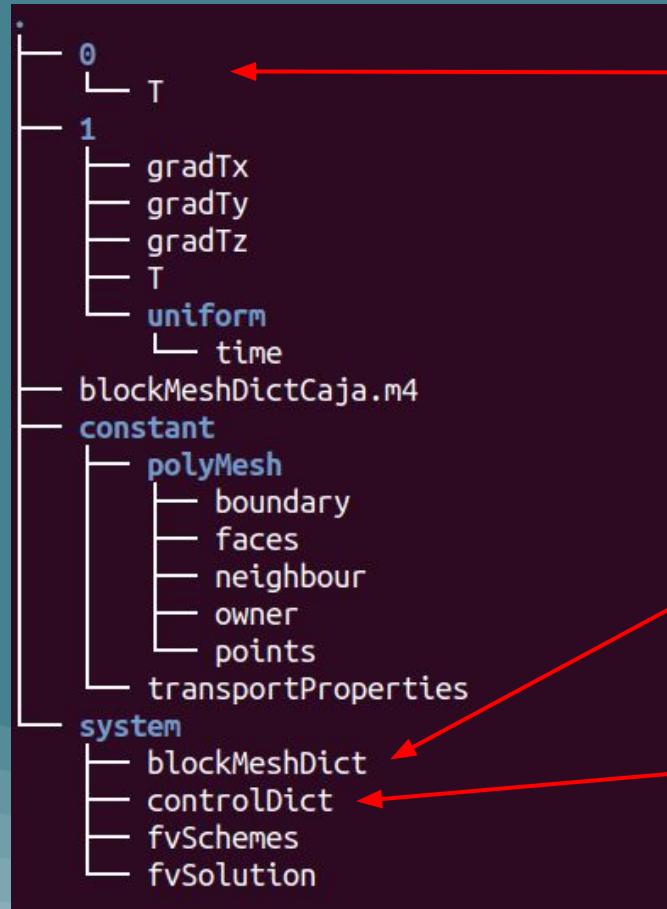
#### `potentialFoam`

Potential flow solver which solves for the velocity potential, to calculate the flux-field, from which the velocity field is obtained by reconstructing the flux.

#### `scalarTransportFoam`

Solves the steady or transient transport equation for a passive scalar.

# Estructura de directorios típica en un caso de OF



En este archivo se puede modificar la condición inicial y de contorno impuesta sobre el campo T

Archivo que lee el generador de mallas interno de OF, llamado **blockMesh**

Archivo que controla diversos aspectos generales de la simulación

# blockMeshDict

```
vertices
(
    (0 0 -0.5)
    (20.0 0 -0.5)
    (20.0 10.0 -0.5)
    (0 10.0 -0.5)
    (0 0 0.5)
    (20.0 0 0.5)
    (20.0 10.0 0.5)
    (0 10.0 0.5)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (60 30 1) simpleGrading (1 1 1)
);
edges
();
boundary
(
    xmin
    {
        type wall;
        faces
        (
            (0 4 7 3)
        );
    }
    xmax
    {
        type wall;
    }
);
```

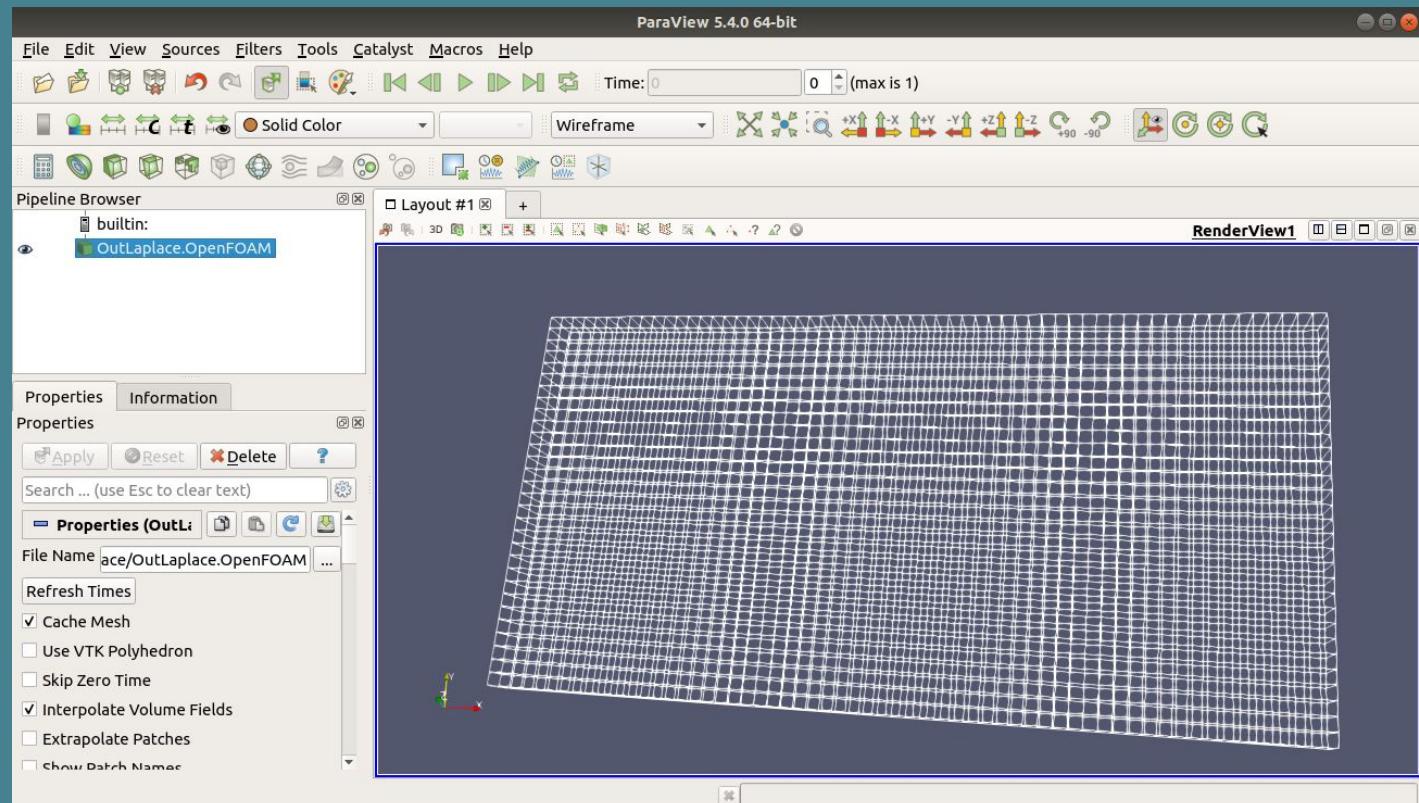
Coordenadas relevantes  
a la caja de simulación

Controlamos lo fino del  
mallado en cada dirección

Definimos las caras que  
están en el contorno

# Visualizando la malla generada con paraFoam

Una vez que terminamos de editar el archivo `blockMeshDict`, generamos la malla tipeando  
`blockMesh`  
luego conviene hacer un chequeo mediante  
`checkMesh`  
si todo está OK, podemos visualizar la malla generada tipeando  
`paraFoam`



# Caso de ejemplo: Ecuación de Laplace en un dominio 2D

```
/*
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      T;
}
// ****
dimensions [0 0 0 1 0 0 0];
internalField uniform 273;
boundaryField
{
    xmin
    {
        type      value
        type      fixedValue;
        value     uniform 0;
    }
    xmax
    {
        type      value
        type      fixedValue;
        value     uniform 50;
    }
    ymin
    {
        type      value
        type      fixedValue;
        value     uniform 0;
    }
    ymax
    {
```

Archivo 0/T

Cantidades con dimensiones en OF:

No.	Property	SI unit
1	Mass	kilogram (kg)
2	Length	metre (m)
3	Time	second (s)
4	Temperature	Kelvin (K)
5	Quantity	mole (mol)
6	Current	ampere (A)
7	Luminous intensity	candela (cd)

Definimos las condiciones de contorno

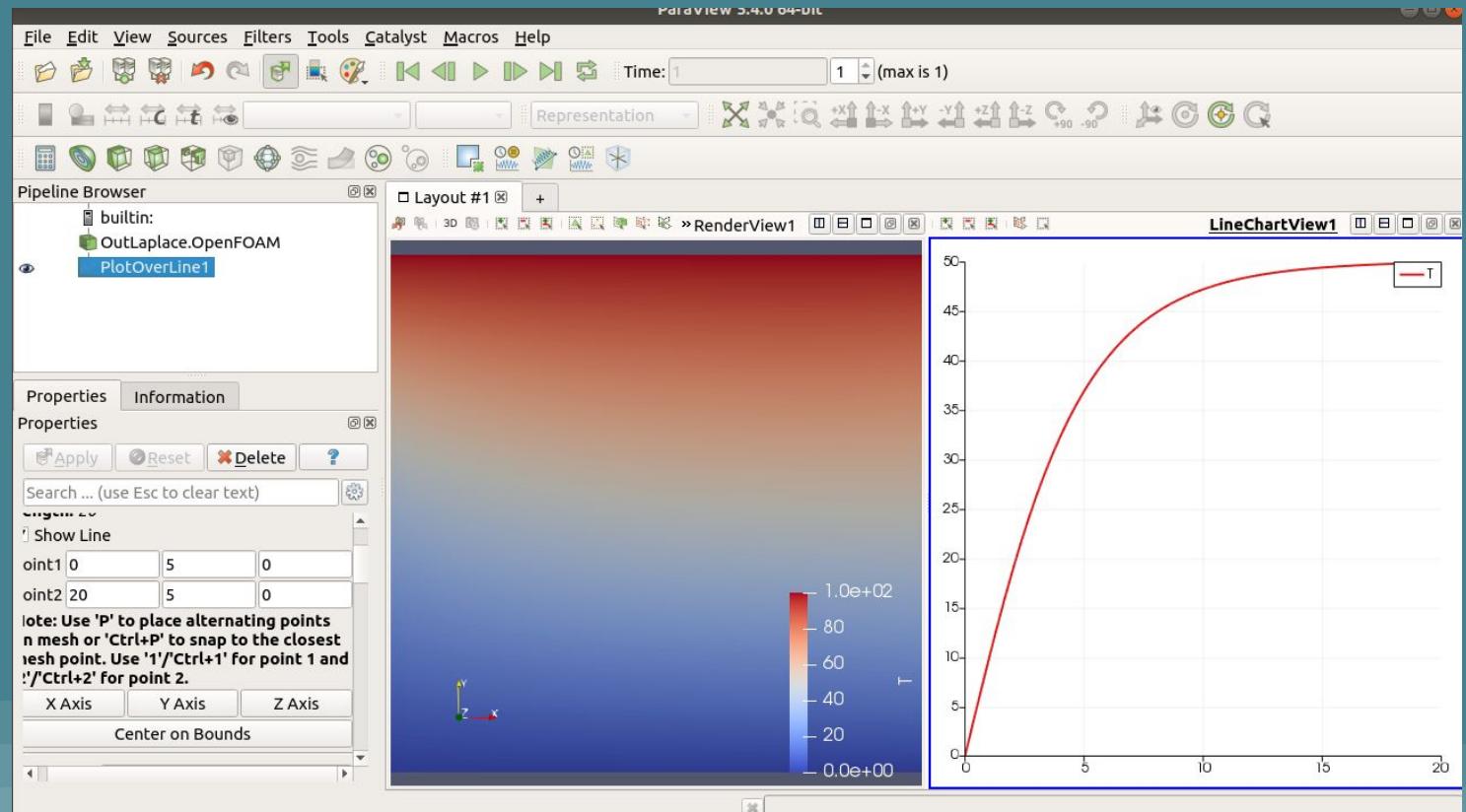
# Postprocesando los resultados

Una vez que terminamos de editar todos los archivos de configuración, y que ya generamos la malla, podemos correr el solver, tipeando

`laplacianFoam`

Una vez que termina la simulación, visualizamos los resultados con

`paraFoam`



# Caso de ejemplo: Difusión en 2D

Problema 2 de la práctica:



# Caso de ejemplo: Difusión en 2D

## Configurando la condición inicial con la utilidad setFields

```
/*----- C++ -----*/
| ====== | Field          | OpenFOAM: The Open Source CFD Toolbox
| \ \ / | Operation      | Version: 4.x
| \ \ / | And            | Web:     www.OpenFOAM.org
| \ \ / | Manipulation |
\-----*/
Foamfile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "system";
    object       setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue T 0
);

regions
(
    boxToCell
    {
        box (0 0 -0.5) (10 10 0.5);
        fieldValues
        (
            volScalarFieldValue T 1
        );
    }
);
```

# Caso de ejemplo: Difusión en 2D

Especificación de la constante de difusión:

```
/*----- C++ -----*/
| ====== | Field          | OpenFOAM: The Open Source CFD Toolbox
| \ \ / | Operation      | Version: 2.2.1
| \ \ / | And            | Web:      www.OpenFOAM.org
| \ \ / | Manipulation   |
\*----- */
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "constant";
    object       transportProperties;
}
// * * * * *
DT [ 0 2 -1 0 0 0 0 ] 4.0;
```

# Caso de ejemplo: Difusión en 2D

Control de la simulación  
mediante el archivo  
`controlDict` del directorio  
`system`

```
application      scalarTransportFoam;  
  
startFrom       startTime;  
  
startTime        0;  
  
stopAt          endTime;  
  
endTime          20.0;  
  
deltaT          $mfc_DeltaT;  
  
writeControl    runTime;  
  
numDumps        20;  
  
writeInterval   #calc "($endTime-$startTime)/$numDumps";  
  
  
purgeWrite      0;  
  
writeFormat     ascii;  
  
writePrecision  6;  
  
writeCompression off;  
  
timeFormat      general;
```

# Caso de ejemplo: Difusión en 2D

Si queremos correr el caso en paralelo, debemos hacer  
`decomposePar`

y luego

```
mpirun -np ${NumProcesadores} scalarTransportFoam -parallel
```

Cuando finaliza la simulación debemos reconstruir el sistema desde los datos generados en los distintos procesadores mediante

```
reconstructPar
```

Luego ya estamos en condiciones de postprocesar la simulación

# Ejemplo de uso de las herramientas de postprocesado

Podemos generar un perfil donde se muestrea la solución mediante un archivo en el directorio `system` llamado por ejemplo `perfil_axial_T` y luego, desde línea de comandos tipeando

```
postProcess -func perfil_axial_T
```

Generará los perfiles de la solución en el directorio

```
postProcessing
```

# Ejemplo de uso de las herramientas de postprocesado

```
/*----- C++ -----*/
=====
 \ \ / F ield      | OpenFOAM: The Open Source CFD Toolbox
  \ \ / O peration  | Website: https://openfoam.org
   \ \ / A nd        | Version: 6
    \ \ M anipulation |
-----  
Description
  Writes graph data for specified fields along a line, specified by start
  and end points.  
*-----*/  
  
start  (0 5 0);
end    (20 5 0);
fields  (T);  
  
// Sampling and I/O settings
#includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"  
  
// Override settings here, e.g.
/*
setConfig
{
    type lineCell;
    axis x;          // y, z, xyz
}
*/  
  
// Must be last entry
#includeEtc "caseDicts/postProcessing/graphs/graph.cfg"  
// *****
```

# Archivos fvSchemes y fvSolution

```
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "system";
    object       fvSchemes;
}
// * * * * *

ddtSchemes
{
    default      Euler;
}

gradSchemes
{
    default      Gauss linear;
}

divSchemes
{
    default      none;
    div(phi,T)   Gauss limitedLinear 1;
}

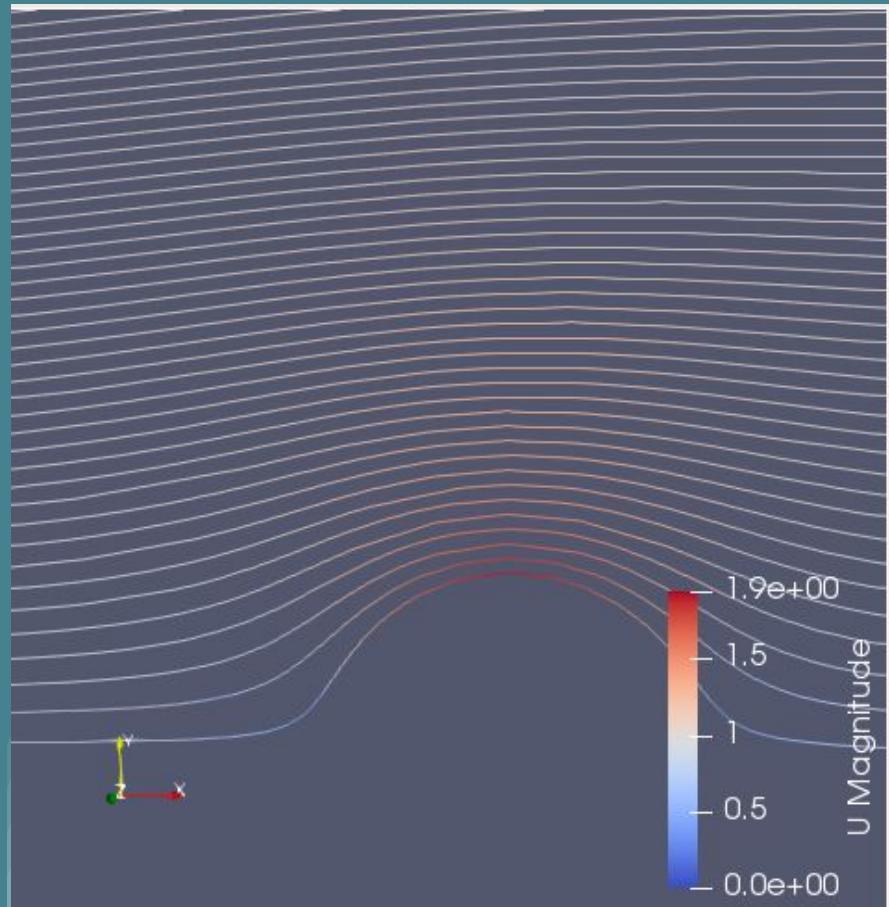
laplacianSchemes
{
    default      none;
    laplacian(DT,T) Gauss linear corrected;
}
```

```
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "system";
    object       fvSolution;
}
// * * * * *

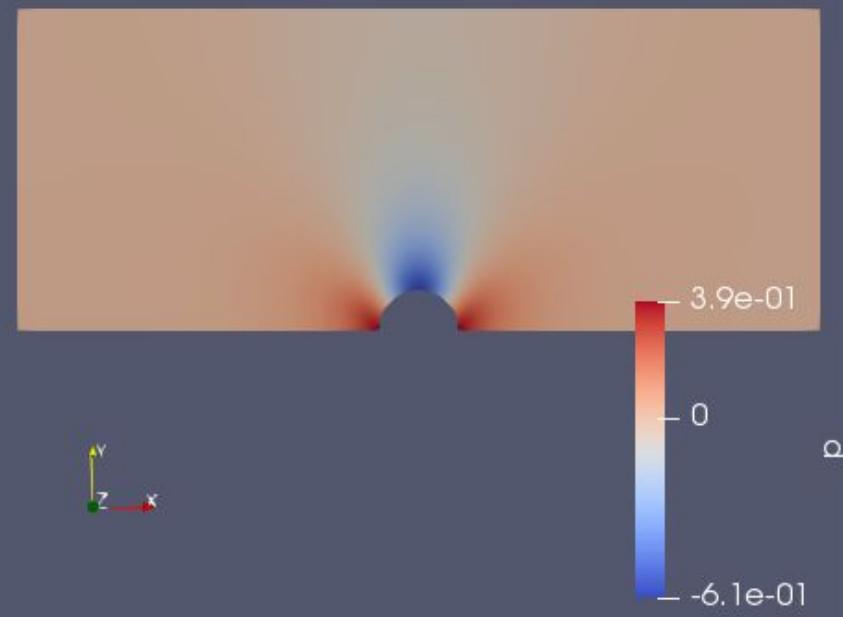
solvers
{
    T
    {
        solver          PBiCG;
        preconditioner DILU;
        tolerance       1e-06;
        relTol          0;
    }
}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
}
```

# Caso de ejemplo: Flujo ideal alrededor de un cilindro



Problema 3 de la práctica:



# Flujo ideal alrededor de un cilindro: potencial complejo

Para el caso particular de un flujo irrotacional y no viscoso, la velocidad del fluido puede escribirse (asumimos un sistema efectivamente 2D)  $\bar{V} = \bar{\nabla} \phi$   $\phi$ : "potencial de velocidad"

Si, además, el flujo es incompresible ( $\bar{\nabla} \cdot \bar{V} = 0$ ), resulta  $\nabla^2 \phi = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$

En mecánica de fluidos se demuestra que, para que la condición  $\bar{\nabla} \times \bar{V} = 0$  se mantenga en el tiempo, es necesario que el fluido sea no-viscoso (ver P. Kundu, "Fluid Mechanics")

# Flujo ideal alrededor de un cilindro: potencial complejo

Dado el potencial de velocidades, podemos formar su armónica conjugada  $\psi$  la función analítica  $F(z) = \phi(x, y) + i\psi(x, y)$

Las líneas  $\psi = \text{cte}$  son trayectorias ortogonales a las líneas equipotenciales y son llamadas "líneas de corriente".

Es fácil ver que  $\vec{V} = (\text{Re}[F'(z)], \text{Im}[F'(z)])$

(es usual definir la "velocidad compleja"

$$q = F'(z)$$

con lo cual  $|\vec{V}| = |F'(z)| = |F'(z)|$ .

R: radio del cilindro  
U<sub>0</sub>: vel en el inlet (x)

Para el flujo irrotacional alrededor de un cilindro el potencial complejo es  $F(z) = U_0(z + R^2/z)$  : (\*)

# Herramientas para generación de mallas

## Mesh generators [ edit ]

Many commercial product descriptions emphasize simulation rather than the meshing technology that enables simulation

- Lists of mesh generators (external):

- [Free/open source mesh generators](#)
- [Public domain and commercial mesh generators](#)

- [ANSYS](#)

- [CD-adapco](#) and Siemens

- [Comet Solutions](#)

- [CGAL Computational Geometry Algorithms Library](#)

- [Mesh generation](#)

- [2D Conforming Triangulations and Meshes](#)

- [3D Mesh Generation](#)



- [CUBIT](#)
- [Gmsh](#)
- [Hextreme meshes](#)
- [MeshLab](#)
- [MSC Software](#)
- [Omega\\_h Tri/Tet Adaptivity](#)
- [Open FOAM Mesh generation and conversion](#)
- [Salome Mesh module](#)
- [TetGen](#)
- [TetWild](#)
- [TRIANGLE Mesh generation and Delaunay triangulation](#)

# Generando mallas no estructuradas con gmsh:

Gmsh is a [finite-element mesh generator](#) developed by Christophe Geuzaine and Jean-François Remacle. Released under the [GNU General Public License](#), Gmsh is [free software](#).

Gmsh contains 4 modules: for geometry description, meshing, solving and post-processing. Gmsh supports [parametric input](#) and has advanced visualization mechanisms. Since version 3.0, Gmsh supports full [constructive solid geometry](#) features, based on [Open Cascade Technology](#).<sup>[1][2]</sup>

A modified version of Gmsh is integrated with SwiftComp, a general-purpose multiscale modeling software. The modified version, called [Gmsh4SC](#), is compiled and deployed on the Composites Design and Manufacturing HUB ([cdmHUB](#)).

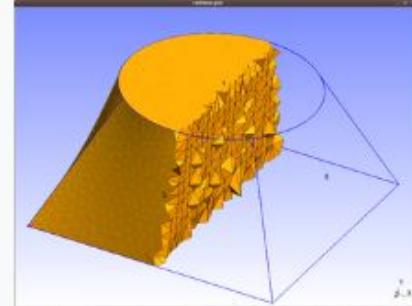
## Contents [hide]

- 1 [Interfaces](#)
- 2 [See also](#)
- 3 [References](#)
- 4 [External links](#)

## Interfaces [edit]

Various graphical user interfaces exist that integrate Gmsh into their workflow:

- A Matlab interface available with [FEATool Multiphysics](#).
- The FEM Workbench of [FreeCAD](#) supports Gmsh for meshing inside the program, along with other meshers like [Netgen](#).<sup>[3]</sup>



Clipped view of Tetrahedral mesh; generated and viewed in Gmsh 2.3.1

<b>Developer(s)</b>	Christophe Geuzaine and Jean-François Remacle
<b>Stable release</b>	4.0.0 / August 22, 2018; 19 months ago
<b>Repository</b>	<a href="https://gitlab.onelab.info/gmsh/gmsh">gitlab.onelab.info/gmsh/gmsh</a>
<b>Written in</b>	C++
<b>Operating system</b>	Unix/Linux, macOS, Windows
<b>License</b>	GNU General Public License
<b>Website</b>	<a href="http://gmsh.info">gmsh.info</a>

# Generando mallas no estructuradas con gmsh:

## Gmsh

A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities ▲

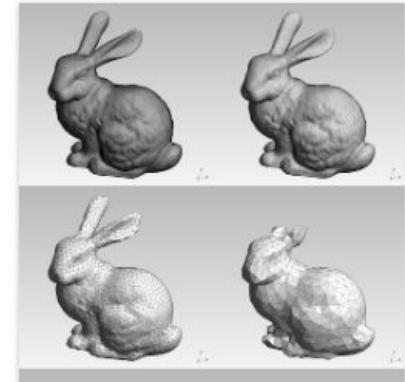
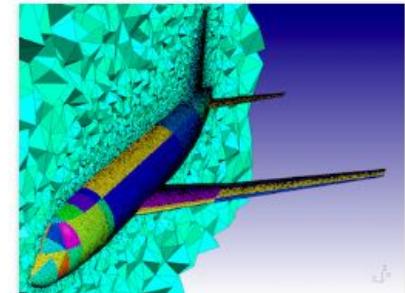
Christophe Geuzaine and Jean-François Remacle

[Download](#) | [Documentation](#) | [Licensing](#) | [Screenshots](#) | [Links](#) | [References](#)  | 

Gmsh is an open source 3D finite element mesh generator with a built-in CAD engine and post-processor. Its design goal is to provide a fast, light and user-friendly meshing tool with parametric input and advanced visualization capabilities. Gmsh is built around four modules: geometry, mesh, solver and post-processing. The specification of any input to these modules is done either interactively using the graphical user interface, in ASCII text files using Gmsh's own scripting language (.geo files), or using the C++, C, Python or Julia Application Programming Interface (API).

See this [general presentation](#) for a high-level overview of Gmsh and recent developments, the [screencasts](#) for a quick tour of Gmsh's graphical user interface, and the [reference manual](#) for a more thorough overview of Gmsh's capabilities, some [frequently asked questions](#) and the documentation of the [C++](#), [C](#), [Python](#) and [Julia API](#).

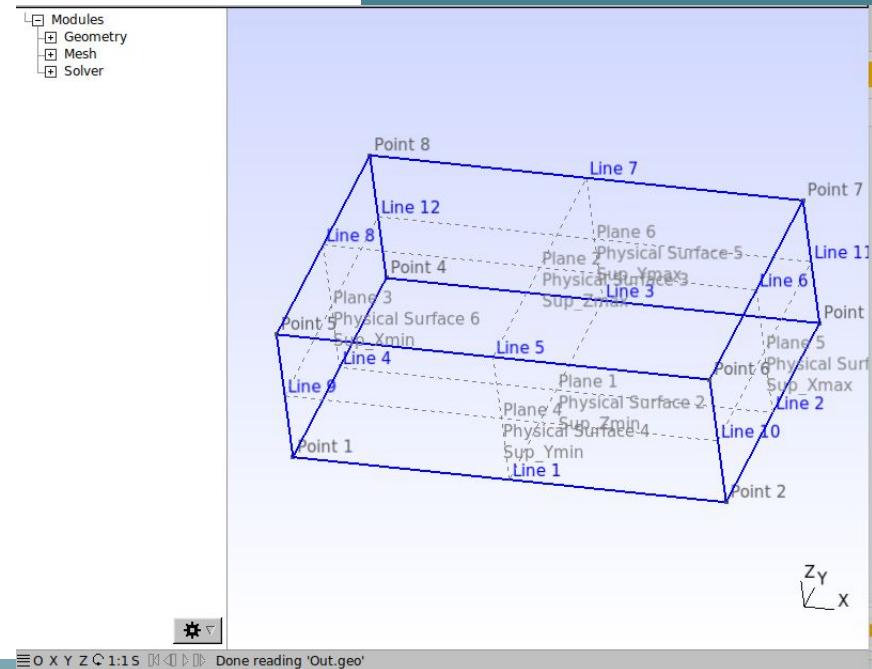
The [source code repository](#) contains many examples written using both the built-in script language and the API (see e.g. the [tutorials](#) and the [demos](#)).



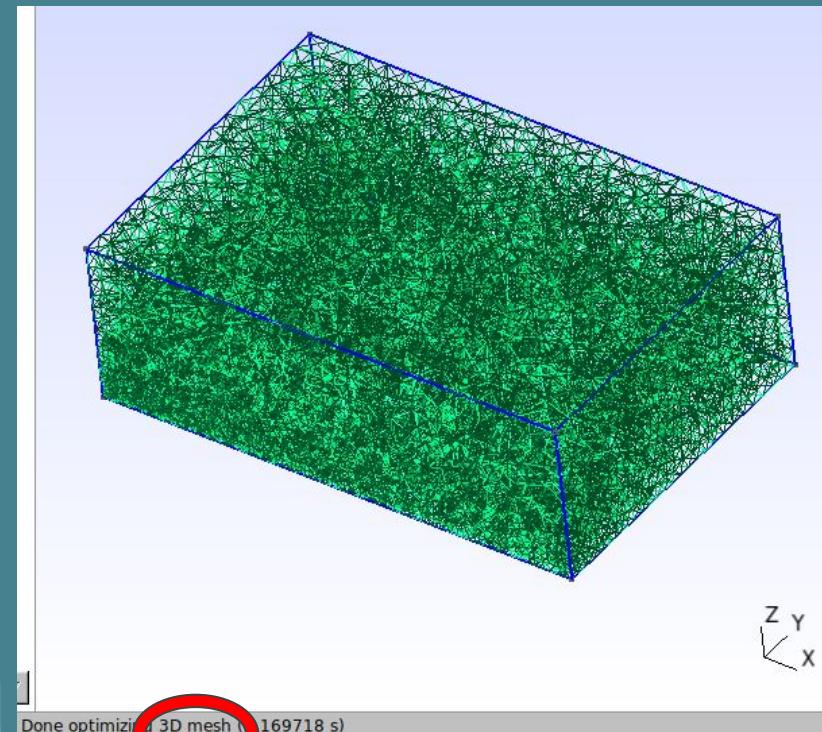
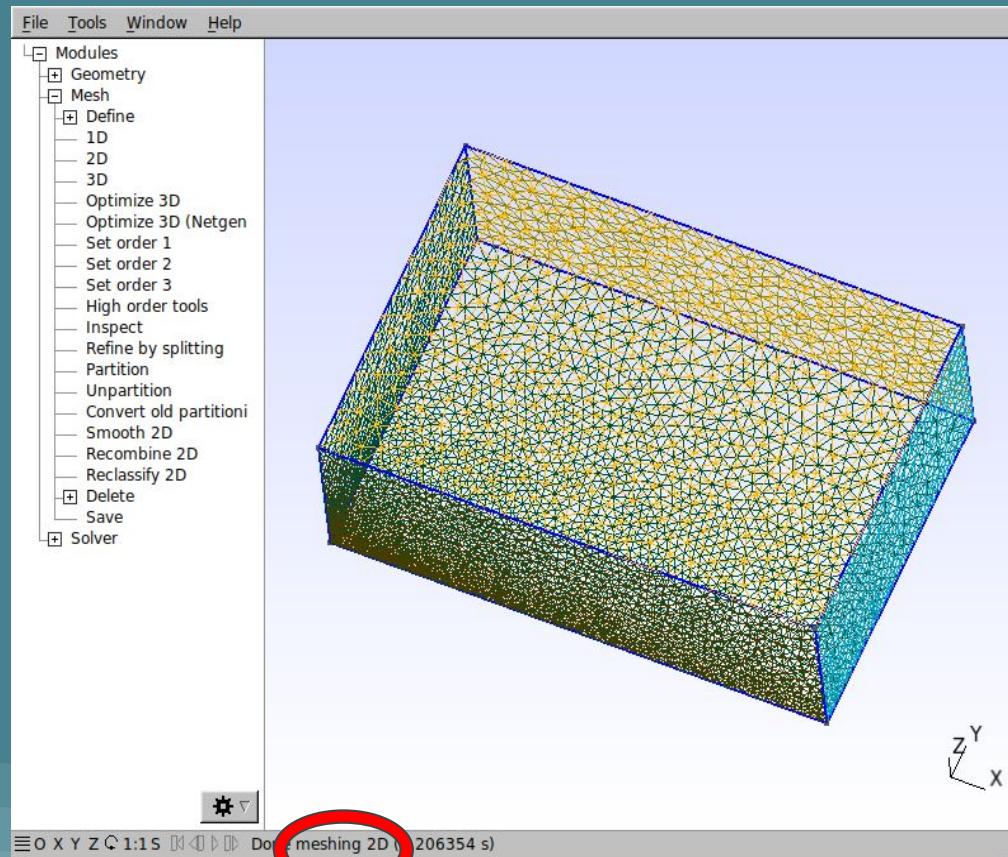
# Generando mallas no estructuradas con gmsh:

```
Point(1) = {0.000000000000e+00, 0.000000000000e+00, 0.000000000000e+00, 0.400000000000e+00};  
Point(2) = {6.000000000000e+01, 0.000000000000e+00, 0.000000000000e+00, 1.000000000000e+00};  
Point(3) = {6.000000000000e+01, 4.000000000000e+01, 0.000000000000e+00, 2.000000000000e+00};  
Point(4) = {0.000000000000e+00, 4.000000000000e+01, 0.000000000000e+00, 2.000000000000e+00};  
Point(5) = {0.000000000000e+00, 0.000000000000e+00, 2.000000000000e+01, 2.500000000000e+00};  
Point(6) = {6.000000000000e+01, 0.000000000000e+00, 2.000000000000e+01, 2.500000000000e+00};  
Point(7) = {6.000000000000e+01, 4.000000000000e+01, 2.000000000000e+01, 2.500000000000e+00};  
Point(8) = {0.000000000000e+00, 4.000000000000e+01, 2.000000000000e+01, 2.500000000000e+00};  
Line(1) = {1, 2};  
Line(2) = {2, 3};  
Line(3) = {3, 4};  
Line(4) = {4, 1};  
Line(5) = {5, 6};  
Line(6) = {6, 7};  
Line(7) = {7, 8};  
Line(8) = {8, 5};  
Line(9) = {1, 5};  
Line(10) = {2, 6};  
Line(11) = {3, 7};  
Line(12) = {4, 8};  
Line Loop(1) = {-1, -2, -3, -4};  
Plane Surface(1) = {1};  
Line Loop(2) = {5, 6, 7, 8};  
Plane Surface(2) = {2};  
Line Loop(3) = {4, 9, -8, -12};  
Plane Surface(3) = {3};  
Line Loop(4) = {1, 10, -5, -9};  
Plane Surface(4) = {4};  
Line Loop(5) = {2, 11, -6, -10};  
Plane Surface(5) = {5};  
Line Loop(6) = {3, 12, -7, -11};  
Plane Surface(6) = {6};  
Surface Loop(1) = {1, 2, 3, 4, 5, 6};  
Volume(1) = {1};
```

Modules  
Geometry  
Mesh  
Solver



# Generando mallas no estructuradas con gmsh:



# Enlaces en donde profundizar:

- The Open Foam User Guide [cfd.direct.openfoam/user-guide/](http://cfd.direct.openfoam/user-guide/)
- Técnicas numéricas en ingeniería de fluidos: introducción a la dinámica de fluidos computacional (CFD) por el método de volúmenes finitos J. M. Fernández Oro, Barcelona: Reverté, 2012.
- Primer Curso de Ecuaciones en Derivadas Parciales I. Peral Alonso, Addison-Wesley Iberoamericana.
- <https://openfoam.org/>
- <https://www.openfoam.com/>
- <https://www.paraview.org/>
- <http://qmesh.info/>