# Métodos de Elementos Finitos

Alejandro Alonso Membrilla

Junio de 2021

# Índice

# 1. Introducción

En este proyecto mostraremos una serie de ejercicios relacionados con la teoría e implementación del cálculo numérico de ciertos Métodos de Elementos Finitos (MEFs).

Junto a la presente memoria, se entregan un par de cuadernos de Jupyter con la implementación en Python de los ejercicios asociados a las prácticas. Los ejercicios opcionales de Python entregados han sido realizados en colaboración con Pilar Navarro Ramírez.

# 2. Ejercicios teóricos sobre el MEF multidimensionales

## 2.1. Elementos finitos triangulares

**a)** $P = \mathbb{P}_1[x, y]$ y $\Sigma = \{v_1, v_2, v_3\}$. Teniendo en cuenta que los vectores $v_2 - v_1$ y $v_3 - v_1$ son linealmente independientes, es fácil comprobar que dado cualquier $v \in \mathbb{R}^2$ existen únicos $\lambda_1(v), \lambda_2(v), \lambda_3(v) \in \mathbb{R}$ tales que

$$\sum_{j=1}^{3} v_j \lambda_j(v) = v \quad \text{y} \quad \sum_{j=1}^{3} \lambda_j(v) = 1.$$

A $\lambda_1(v), \lambda_2(v), \lambda_3(v)$ se les denomina coordenadas baricéntricas de $v$ respecto del triángulo $K$. Además, si $v = (x, y)$ las funciones $\lambda_i(x, y) \in \mathbb{P}_1$ para $1 \leq i \leq 3$. Las funciones $\lambda_1, \lambda_2, \lambda_3$ son las funciones base para este elemento finito, ya que $\lambda_i(v_j) = \delta_{ij}$.

*Demostración:* Empezamos viendo que $P$ es unisolvente en $K$. $P = gen(\{1, x, y\})$. Por tanto, dados $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{R}$, encontrar $p \in P$ tal que $p(v_1) = \alpha_1$, $p(v_2) = \alpha_2$, $p(v_3) = \alpha_3$ equivale a resolver el sistema lineal

$$\begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix}$$

donde $a, b$ y $c$ constituirían los coeficientes del polinomio $p$. Este sistema tiene solución única automáticamente por el hecho de haber tomado $v_1, v_2, v_3$ vértices de un triángulo y, por tanto, no alineados.

Ahora vamos a demostrar las propiedades de las coordenadas baricéntricas tal como se explicó en clase. $v_2 - v_1$ y $v_3 - v_1$ son linealmente independientes y, por tanto, son base de $\mathbb{R}^2$. Representamos $v - v1 \in \mathbb{R}^2$ en esta base:

$$v - v_1 = \mu_2(v)(v_2 - v_1) + \mu_3(v)(v_3 - v_1) \implies v = v_1(1 - \mu_2(v) - \mu_3(v)) + v_2\mu_2(v) + v_3\mu_3(v)$$

Tomando $\lambda_1 = 1 - \mu_2(v) - \mu_3(v)$, $\lambda_2 = \mu_2(v)$ y $\lambda_3 = \mu_3(v)$ claramente se cumplen las dos primeras propiedades enunciadas sobre las coordenadas baricéntricas.

Los valores de las $\lambda_i(v) = \lambda_i(x, y)$ pueden calcularse resolviendo el siguiente sistema lineal:

$$\begin{cases} \lambda_1(v) + \lambda_2(v) + \lambda_3(v) = 1 \\ \lambda_1(v)x_1 + \lambda_2(v)x_2 + \lambda_3(v)x_3 = x \\ \lambda_1(v)y_1 + \lambda_2(v)y_2 + \lambda_3(v)y_3 = y \end{cases} \tag{1}$$

Por tanto, sus valores dependerán linealmente de $x$ e $y$, luego $\lambda_i \in \mathbb{P}_1$ para $i = 1, 2, 3$.

Finalmente, si $v = v_1$, entonces

$$v - v_1 = v_1 - v_1 = 0 = 0(v_2 - v_1) + 0(v_3 - v_1)$$

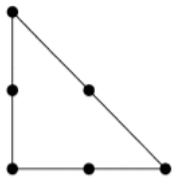
lo que demuestra que $\mu_2(v_1) = \mu_3(v_1) = 0$ y, por tanto, $\lambda_1(v_1) = 1 - \mu_2(v_1) - \mu_3(v_1) = 1$. La propiedad análoga puede demostrarse para $\lambda_2$ y $\lambda_3$, probando que $\lambda_i(v_j) = \delta_{ij}$.

**b)** $P = \mathbb{P}_2$ y $\Sigma = \{v_1, v_2, v_3, v_{12}, v_{13}, v_{23}\}$ donde $v_{ij}$ es el punto medio del segmento que une los vértices $v_i$ y $v_j$ . Las funciones base son

$$p_i = \lambda_i(2\lambda_i - 1) \;\text{ y }\; p_{ij} = 4\lambda_i\lambda_j$$

para cada vértice y cada punto medio respectivamente.

*Demostración:* De nuevo, empezamos viendo la unisolvencia. En este caso, consideraremos suficiente analizar el caso con $v_1 = (0, 0), v_2 = (0, 1), v_3 = (1, 0)$:



Calcularemos el determinante de la matriz del sistema lineal asociado usando cálculo simbólico. Con el siguiente código definimos la matriz correspondiente:

```
import sympy as sp

x1,x2,x3,y1,y2,y3 = sp.symbols('x1,x2,x3,y1,y2,y3')
mat = []
puntos = [(x1,y1),(x2,y2),(x3,y3),
          ((x1+x2)/2,(y1+y2)/2),
          ((x1+x3)/2,(y1+y3)/2),
          ((x2+x3)/2,(y2+y3)/2)]
for x,y in puntos:
        mat.append([1,x,y,x*y,x**2,y**2])
mat = sp.Matrix(mat)
```

Que resulta de la siguiente manera:

$$\begin{bmatrix}
1 & x_1 & y_1 & x_1y_1 & x_1^2 & y_1^2 \\
1 & x_2 & y_2 & x_2y_2 & x_2^2 & y_2^2 \\
1 & x_3 & y_3 & x_3y_3 & x_3^2 & y_3^2 \\
1 & \frac{x_1}{2}+\frac{x_2}{2} & \frac{y_1}{2}+\frac{y_2}{2} & \left(\frac{x_1}{2}+\frac{x_2}{2}\right)\left(\frac{y_1}{2}+\frac{y_2}{2}\right) & \left(\frac{x_1}{2}+\frac{x_2}{2}\right)^2 & \left(\frac{y_1}{2}+\frac{y_2}{2}\right)^2 \\
1 & \frac{x_1}{2}+\frac{x_3}{2} & \frac{y_1}{2}+\frac{y_3}{2} & \left(\frac{x_1}{2}+\frac{x_3}{2}\right)\left(\frac{y_1}{2}+\frac{y_3}{2}\right) & \left(\frac{x_1}{2}+\frac{x_3}{2}\right)^2 & \left(\frac{y_1}{2}+\frac{y_3}{2}\right)^2 \\
1 & \frac{x_2}{2}+\frac{x_3}{2} & \frac{y_2}{2}+\frac{y_3}{2} & \left(\frac{x_2}{2}+\frac{x_3}{2}\right)\left(\frac{y_2}{2}+\frac{y_3}{2}\right) & \left(\frac{x_2}{2}+\frac{x_3}{2}\right)^2 & \left(\frac{y_2}{2}+\frac{y_3}{2}\right)^2
\end{bmatrix}$$

Hemos calculado su determinante, evaluando la matriz en $v_1 = (0,0), v_2 = (0,1), v_3 = (1,0)$, con la línea siguiente:

```
sp.det( mat.subs({x1:0,x2:1,x3:0,y1:0,y2:0,y3:1}) )
```

Obteniendo $\frac{1}{64} \neq 0$. Por tanto el sistema admite solución única.

Para demostrar que las $p_i$ y $p_{ij}$ definidas son funciones base, vamos a comprobar que estas valen 1 en su punto de $K$ correspondiente, y 0 en los demás. Para ello notaremos como $vij$ a cualquier combinación de la forma $\frac{v_i+v_j}{2}$. Nótese que $v_{ii} = v_i$ y $v_{ij} = v_{ji}$.

- $p_i(v_{jk}) = \lambda_i(v_{jk})(2\lambda_i(v_{jk}) - 1) = \frac{1}{2}(\lambda_i(v_j) + \lambda_i(v_k))(\lambda_i(v_j) + \lambda_i(v_k) - 1)$. Observamos que $\lambda_i(v_j) + \lambda_i(v_k)$ solo puede tomar los valores 0, 1 ó 2. $p_i(v_{jk})$ valdrá 0 si $\lambda_i(v_j) + \lambda_i(v_k)$ es distinto de 2, y solo puede valer 2 si $i = j = k$. En este caso $p_i(v_{ii}) = 1$, como se buscaba.
- $p_{ij}(v_{kl}) = 4\lambda_i(v_{kl})\lambda_j(v_{kl}) = 4\frac{1}{2}(\lambda_i(v_k) + \lambda_i(v_l))\frac{1}{2}(\lambda_j(v_k) + \lambda_j(v_l))$. En este caso, si $k = l$, como sucede en los vértices del triángulo, alguna de las componentes del producto se anulará y $p_{ij}$ valdrá 0. Si $k \neq l$, pero al menos uno no coincide con $i$ ni con $j$, también se anulará alguna de las componentes. En conclusión, $p_{ij} = 1 \iff k \neq l \wedge k, l \in \{i, j\}$. En otro caso $p_{ij} = 0$.

# 3. Problema prototipo de cuarto orden

Consideramos la ecuación del problema de la viga:

$$u^{iv)}(x) = f(x), \quad x \in ]0, l[ \tag{2}$$

En principio, asumiremos carga constante, esto es, $f(x) \equiv C$.

**Ejercicio:** a partir del proceso de obtención de la formulación variacional o débil de un problema de cuarto orden como este, obtenga también las correspondientes formulaciones (con las condiciones de contorno de tipo natural adicionales) asociadas a las siguientes variantes o tipos de viga diferentes:

## 3.1. Viga simplemente apoyada

Consideramos condiciones esenciales de contorno de tipo Dirichlet: $u(0) = 0 = u(l)$.

Usaremos el método de Galerkin para transformar la formulación clásica a una variacional equivalente. Tomamos funciones test $v \in \mathcal{H}_0^2$. Es necesario exigir que las funciones test valgan 0 en los extremos puesto que, de momento, la viga tiene ambos extremos fijos. También deben ser al menos dos veces débilmente derivables para poder aplicar los cálculos siguientes. Multiplicando ambos lados de la ecuación 2 por $v$ e integrando obtenemos:

$$\int_0^l u^{iv)}(x)v(x)dx = \int_0^l f(x)v(x)dx = C\int_0^l v(x)dx$$

Aplicando dos veces la fórmula de integración por partes vemos que

$$\int_0^l u^{iv)}(x)v(x)dx = \cancelto{0}{u'''(x)v(x)\Big]_0^l} - \int_0^l u'''(x)v'(x)dx = -u''(x)v'(x)\Big]_0^l + \int_0^l u''(x)v''(x)dx$$

Como, en general, $v'$ puede no anularse en los extremos del intervalo, para que la expresión anterior coincida con una formulación débil debemos imponer que $u''$ se anule en ambos extremos.

El problema variacional asociado queda de la siguiente forma:

$$(V)\begin{cases}\int_0^l u''(x)v''(x)dx = C\int_0^l v(x)dx\\ u(0)=u(l)=u''(0)=u''(l)\end{cases}$$

### 3.2. Viga empotrada en un extremo y simplemente apoyada en el otro

Consideramos las condiciones de contorno $u(0)=u'(0)=0=u(l)$.

Procederemos de igual forma que en el apartado anterior. Los extremos de la viga siguen fijos, así que solo necesitamos imponer sobre las funciones test $v\in\mathcal{H}_0^2$ que, además, cumplan que $v'(0)=0$.

$$\begin{aligned}\int_0^l u^{iv)}(x)v(x)dx =&\cancel{u'''(x)v(x)\Big]_0^l}^{0} - \int_0^l u'''(x)v'(x)dx = -u''(x)v'(x)\Big]_0^l + \int_0^l u''(x)v''(x)dx\\ =&- u''(l)v'(l) + \int_0^l u''(x)v''(x)dx\end{aligned}$$

De esta forma, la única condición adicional que necesitamos imponer sobre $u$ es que $u''(l)=0$. La formulación variacional resulta en:

$$(V)\begin{cases}\int_0^l u''(x)v''(x)dx = C\int_0^l v(x)dx\\ u(0)=u(l)=u'(0)=u''(l)=0\end{cases}$$

### 3.3. Viga en voladizo

Empotrada sólo en uno de los extremos y sin apoyo en el otro, con condiciones: $u(0)=u'(0)=0$.

Tomando $v$ con las mismas condiciones de contorno que $u$, obtenemos:

$$\begin{aligned}\int_0^l u^{iv)}(x)v(x)dx =&u'''(x)v(x)\Big]_0^l - \int_0^l u'''(x)v'(x)dx = u'''(l)v(l) - u''(x)v'(x)\Big]_0^l + \int_0^l u''(x)v''(x)dx\\ =&u'''(l)v(l) - u''(l)v'(l) + \int_0^l u''(x)v''(x)dx\end{aligned}$$

Como $v(l)$ y $v'(l)$ no son necesariamente nulos, $u'''(l)v(l)-u''(l)v'(l)=0$ para cualquier $v$ dentro del espacio de perturbaciones utilizado si y solo si $u'''(l)=u''(l)=0$. La formulación variacional resulta en:

$$(V)\begin{cases}\int_0^l u''(x)v''(x)dx = C\int_0^l v(x)dx\\ u(0)=u'(0)=u''(l)=u'''(l)=0\end{cases}$$

## 4. FreeFEM++

### 4.1. Ejercicio propuesto

El ejercicio obligatorio propuesto consiste en deducir la formulación fuerte de un problema de ecuaciones en derivadas parciales con restricciones en la frontera a partir de un código de ejemplo de

FreeFEM++ que implementa su correspondiente versión variacional. El código en cuestión es el siguiente:

```
mesh Th=square(10,10);
fespace Vh(Th,P1);      // P1 FE space
Vh uh,vh;               // unkown and test function.
func f=1;                  //  right hand side function
func g=0;                  //  boundary condition function

problem laplace(uh,vh,solver=GMRES,tgv=1e5) =    //  definion of  the problem
        int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) ) //  bilinear form
        + int1d(Th,1)( uh*vh)
        - int1d(Th,1)( vh)
        - int2d(Th)( f*vh )                          //  linear form
        + on(2,3,4,uh=g) ;                       //  boundary condition form

laplace; // solve the problem plot(uh); // to see the result
plot(uh,ps="LaplaceP1.eps",value=true);
```

El dominio del problema es el cuadrado $[0,1]^2$ que notaremos por $\Omega$. Destacamos que la llamada a la función `square(10,10)` no crea un cuadrado de $10 \times 10$, sino uno de $1 \times 1$ con una resolución de 10 nodos en cada lado. Los lados del cuadrado los notaremos por $\omega_1$, $\omega_2$, $\omega_3$, $\omega_4$ en el mismo orden que los numera FreeFEM (abajo, derecha, arriba, izquierda).

En forma analítica, el problema variacional anterior puede representarse como encontrar una función $u$ tal que:

$$\begin{aligned} \mathcal{A}(u,v) &= \int_\Omega \nabla u \cdot \nabla v \, d\Omega + \int_{\omega_1} u(x,0)v(x,0)\, dx \\ \mathcal{B}(v) &= \int_\Omega f v \, d\Omega + \int_{\omega_1} v(x,0)\, dx \\ \mathcal{A}(u,v) &= \mathcal{B}(v) \end{aligned} \tag{3}$$

bajo las restricciones de que $f = 1$ en $\Omega$ y $v = u = g = 0$ en $\omega_2, \omega_3$ y $\omega_4$.

Gracias a la primera identidad de Green, podemos usar que

$$\int_\Omega \nabla u \cdot \nabla v \, d\Omega = -\int_\Omega \Delta u \, v \, d\Omega + \int_{\partial\Omega} v \, \nabla u \cdot \hat{n} \, d\gamma = -\int_\Omega \Delta u \, v \, d\Omega - \int_{\omega_1} v \, \frac{\partial u}{\partial y} \, d\gamma \tag{4}$$

donde hemos usado, en la última igualdad, que $v = 0$ en $\omega_2$, $\omega_3$, $\omega_4$, y que el vector normal exterior unitario del borde de $\Omega$, $\hat{n}$, es igual a $(0,-1)$ en todo $\omega_1$. Sustituyendo (4) en (3), obtenemos:

$$-\int_\Omega \Delta u \, v \, d\Omega - \int_{\omega_1} v(x,0) \, \frac{\partial u}{\partial y}(x,0)\, dx + \int_{\omega_1} u(x,0)v(x,0)\, dx = \int_\Omega f v \, d\Omega + \int_{\omega_1} v(x,0)\, dx \tag{5}$$

o, equivalentemente,

$$\int_\Omega (-\Delta u - f)\, v \, d\Omega + \int_{\omega_1} (u(x,0) - \frac{\partial u}{\partial y}(x,0) - 1) v \, dx = 0$$

donde la integral de la izquierda representa la ecuación que debe cumplirse en todo $\Omega$, $\Delta u = f = 1$, y la integral de la derecha supone una condición de contorno adicional. Para que esta última se anule

para toda función test $v$, por el lema fundamental del cálculo de variaciones, es necesario imponer la condición de contorno siguiente: $u(x,0) - \frac{\partial u}{\partial y}(x,0) = 1$. Esta condición es, en particular, una de tipo Robin puesto que relaciona el valor de $u$ en $\omega_1$ con una de sus derivadas parciales.

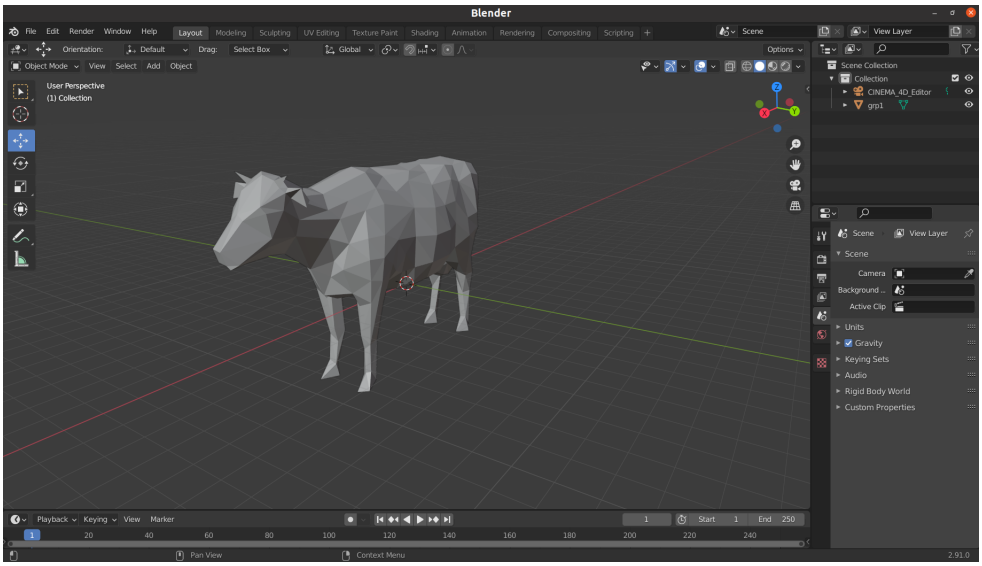
La formulación clásica asociada al problema variacional en (3) es la siguiente:

$$(P)\begin{cases} -\Delta u(x,y) = 1 & \forall (x,y) \in [0,1]^2 \\ u(x,1) = u(0,y) = u(1,y) = 0 & x,y \in [0,1] \\ u(x,0) - \frac{\partial u}{\partial y}(x,0) = 1 & x \in [0,1] \end{cases}$$

## 4.2. EXTRA: carga de modelos 3D en FreeFEM++

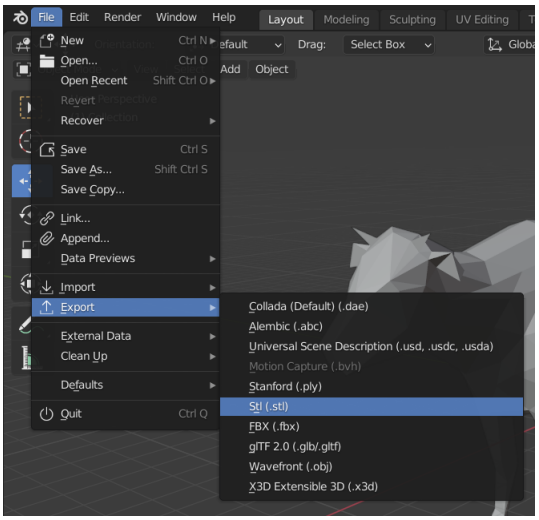
FreeFEM++ cuenta con múltiples herramientas para generar mallados 2D y 3D mediante código. Sin embargo, en la práctica es habitual querer analizar como podrían comportarse ciertos objetos en el mundo real bajo ciertas circunstancias que pueden simularse con software como FreeFEM++. Esos objetos incluso podrían no existir aún, y nuestra tarea podría ser querer encontrar un diseño adecuado para el mismo. Hacer esto solo a través de código es una tarea laboriosa, especialmente si se trabaja en colaboración con diseñadores, con menos experiencia en programación.

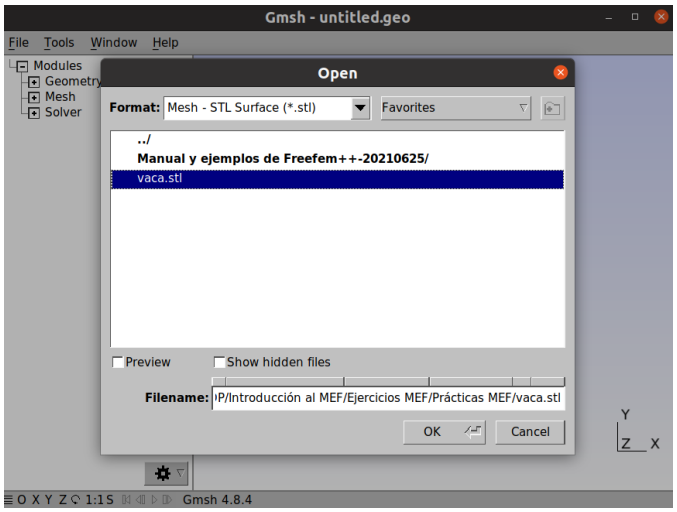
Una alternativa podría ser usar los métodos que FreeFEM proporciona para cargar modelos externos. Blender es una herramienta de software libre para, entre otras cosas, modelado 3D. Es muy completo, ampliamente utilizado y, a día de hoy, en intenso desarrollo. Empezaremos cargando/creando un modelo 3D en Blender (descargado de este enlace):



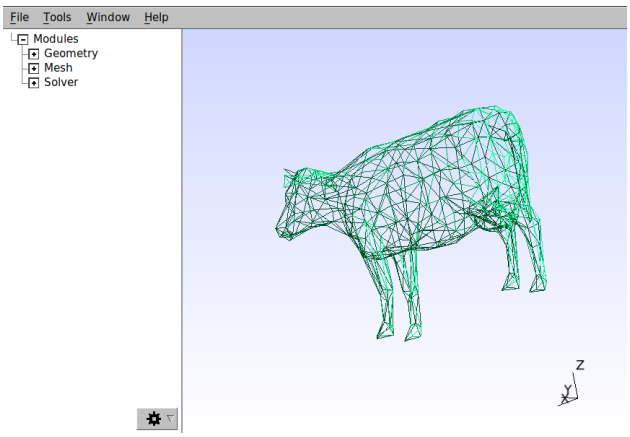

Modelo 3D de una vaca cargado en Blender.

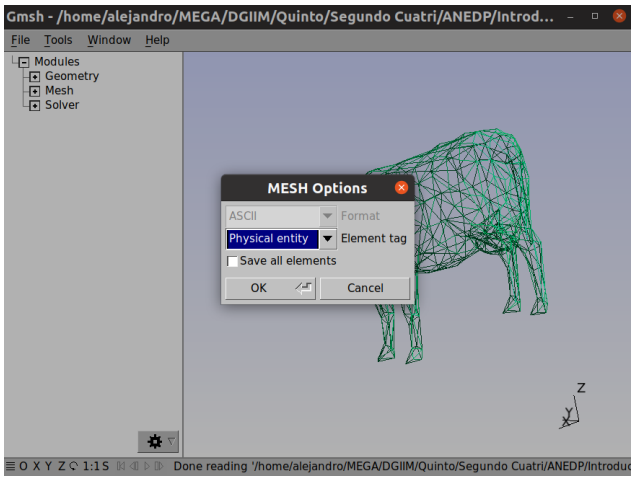
Blender permite importar y exportar modelos 3D en varios formatos (entre otros, FBX, STL, glTF...). Ninguno de ellos es compatible con FreeFEM++. Para salvar esta incompatibilidad usaremos el software Gmsh que permite exportar al formato `.mesh` que usa FreeFEM++ de forma nativa. Gmsh admite mallados en formato `.stl`, al cual es posible exportar mediante Blender.




Exportamos la vaca a STL y la cargamos desde Gmsh. Ahora podemos exportar nuestra vaca a formato `.mesh`. Es importante hacerlo como entidad física para que FreeFEM++ lo reconozca.




El siguiente código mínimo en FreeFEM++ permite visualizar el modelo de la vaca usando la orden `medit`:

```
load "medit"
meshS Th=readmeshS("vaca.mesh");
medit("vaca", Th);
```

con el siguiente resultado:

```
-- FreeFem++ v4.6 (Thu Apr  2 15:47:38 CEST 2020 - git v4.6)
Load: lg_fem lg_mesh lg_mesh3 eigenvalue
1 : load "medit"
2 : meshS Th=readmeshS("vaca.mesh");
3 : medit("vaca", Th);
4 :  sizestack + 1024 =1272  ( 248 )

read meshS ok 0surface Mesh,
```

```
        num Triangles:= 718, num Vertice:= 362 num boundary Edges:= 0
--- Warning manifold obj nb:2 adj 2 of  dim =2
-- MeshS : vaca.mesh, space dimension 3,
        num Triangle elts 718, num Vertice 362 num Bordary elts 0
version de medit ffmedit -popen -filebin 1  vaca
--- Warning manifold obj nb:2 adj 2 of  dim =2
--- Warning manifold obj nb:2 adj 2 of  dim =2
--- Warning manifold obj nb:2 adj 2 of  dim =2
-- Medit,  Release 3.0a (Nov. 30, 2007)
Copyright (c) LJLL, 1999-2007.
compiled:  Thu Apr 2 15:52:17 CEST 2020 (with ff++ 4.6).

medit with binary version of popen : mesh(es)
mesh_name= vaca
Loading data file(s)
End of mesh
Input seconds:     0.00

medit1()

Building scene(s)
Creating scene 1
Loading default options
Scene seconds:     0.07

Rendering scene(s)
```