GMSH

12 de noviembre de 2011

Índice general

Índice de figuras

Capítulo 1

Introducción

GMSH es un programa de descripción, visualización y discretización de entidades geométricas. GMSH se basa en un lenguaje de programación capaz de describir las entidades geométricas básicas y de configurar los parámetros necesarios para mallar tanto sus superficies como sus volúmenes.

Es un programa de código abierto, libre y gratuito que puede ser modificado para ajustar sus características a nuevas necesidades, aunque es lo suficientemente versátil como para definir y mallar la mayoría de geometrías.

GMSH cuenta también con una interfaz gráfica capaz de configurar los casos pero está más orientada hacia la visualización de las geometrías descritas y de los resultados obtenidos. De hecho, desde el propio manual se recomienda el uso del lenguaje de programación para la ejecución de todas las tareas disponibles.

1.1. Mallado.

Se define como mallado la discretización de una línea, superficie o volúmen en porciones de tamaño finito. Las porciones, además de tener un tamaño característico varias veces menor que el espacio discretizado, serán entidades geométricas elementales como los triángulos o cuadriláteros en dos dimensiones o los tetraedros o prismas en tres dimensiones. Esta discretización en estructuras más elementales es esencial para la resolución de ecuaciones en derivadas parciales en dominios arbitrarios por el método de volúmenes finitos (FVM) o el método de elementos finitos (FEM).

El proceso de mallado más habitual es el de discretizar sucesivamente entidades de mayor dimension como si de un problema de contorno se tratara. Por ejemplo, si queremos mallar un cubo primero definiremos los puntos de control en cada uno de sus doce aristas. Luego discretizaremos en porciones elementales cada uno de sus seis lados y finalmente discretizaremos el volúmen.

Para entender mejor este proceso, y con anterioridad a entender la sintaxis de los archivos GMSH presentamos este ejemplo de mallado de un cubo con tetraedros. El primer paso es definir los puntos básicos de la geometría.

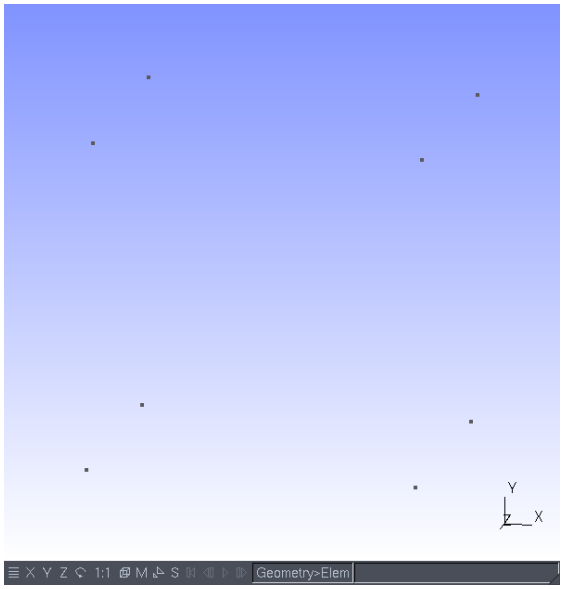

Figura 1.1: Puntos en el espacio que definen un cubo.

El paso siguiente es definir las líneas que servirán de vértices del cubo.

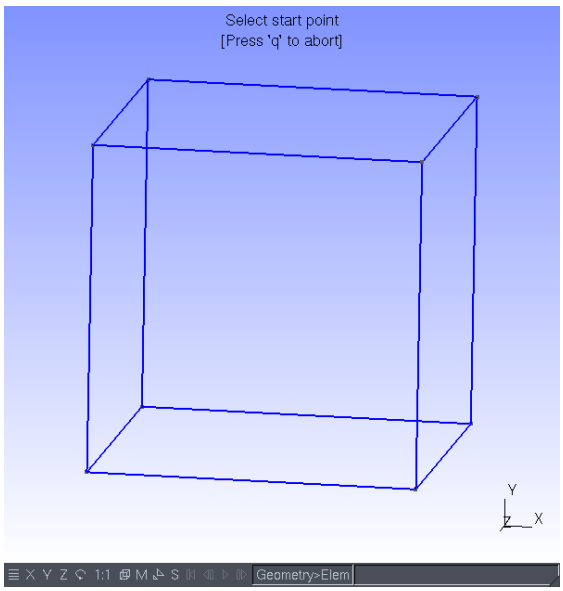

Figura 1.2: Vértices definidos a partir de los puntos.

Pasamos entonces a unir segmentos para crear las seis superficies que definen el cubo y el volumen correspondiente como entidades geométricas

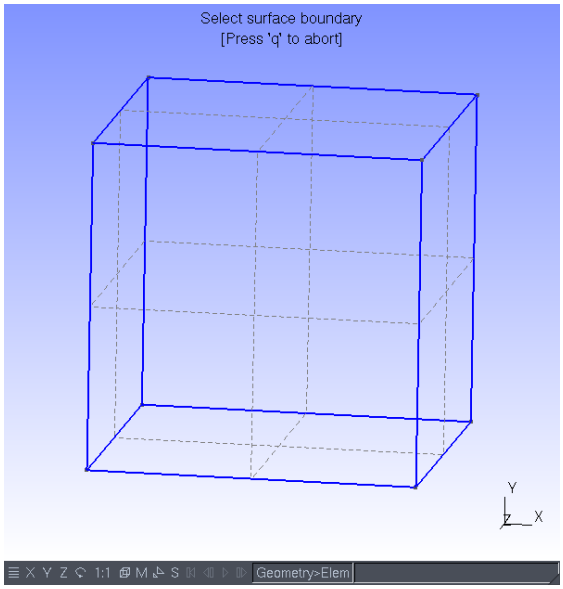

Figura 1.3: Superficies definidas a partir de conjuntos de vértices.

Cuando ya hemos definido todas las entidades pasamos a mallar los segmentos del cubo con el botón Mesh 1D del menú Mesh en la interfaz gráfica para luego mallar las superficies con el botón Mesh2D del menú Mesh en la interfaz gráfica..

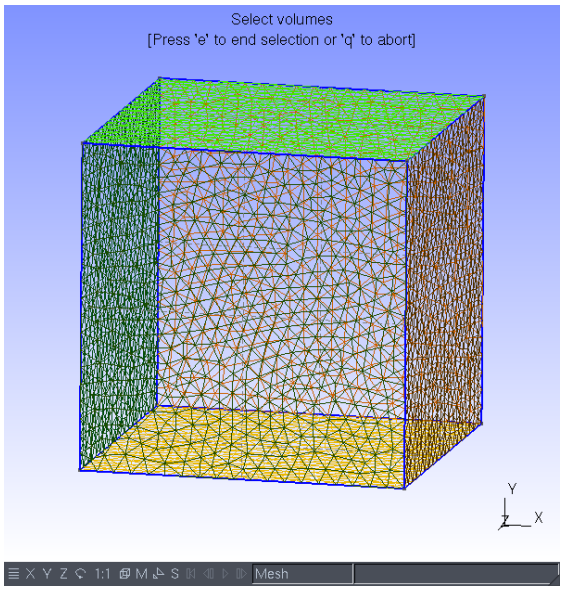

Figura 1.4: Mallado de los vértices y las caras del cubo.

Finalmente mallamos el volumen del cubo que es la entidad tridimensional.

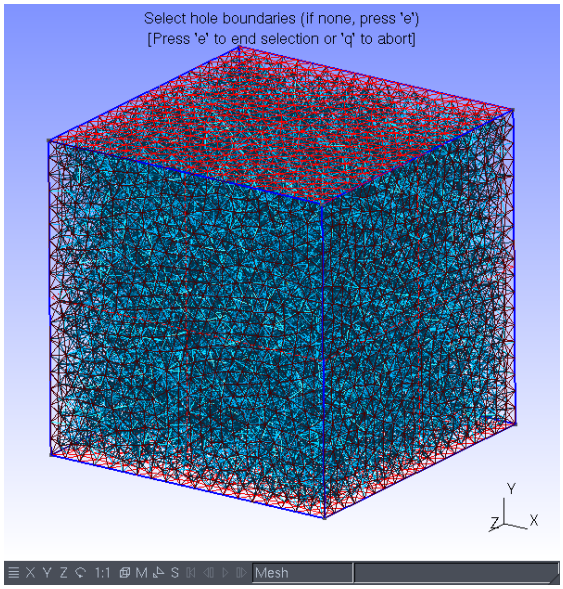

Figura 1.5: Mallado del volumen contenido en el cubo mediante tetraedros a partir de la triangulación de la superficie.

Esta malla tridimensional formada por tetraedros de un volúmen aproximadamente $20^3 \times 8$ veces menor al volumen del cubo (20 puntos por segmento al cubo dividido por el número de tetraedros que caben en un cubo) puede ser ya utilizada por los programas de resolución de ecuaciones en derivadas parciales como problemas fluidodinámicos o estructurales. De hecho, la malla consta de 49580 tetraedros. Esto hace que las previsiones del volumen de elementos resultantes sea acertada puesto que se esperaban unos 64000 tetraedros.

Es importante recalcar que este es el primer paso de todo el ciclo de trabajo de la simulación numérica.

1.2. Tipos de malla.

Existen esencialmente dos tipos de malla, la estructurada y la no estructurada.

1.2.1. Mallas estructuradas

Una malla estructurada es una malla que puede ser transformada, mediante una transformación biyectiva, a una cuadrícula. Esto significa que el problema podría ser resuelto en una malla uniforme y rectangular y devuelto a la geometría original sólo conociendo el jacobiano de la transformación porque este jacobiano se puede invertir.

En GMSH las mallas estructuradas se definen mediante los métodos de interpolación transfinita (TFI).

Si consideramos una región $\bar{\Omega} \in [0,1] \times [0,1]$ y postulamos la existencia de una función F que transforma la región $\bar{\Omega}$ en Ω tales que $F : F \to \bar{\Gamma}$ el método construye una función $U : \bar{\Omega} \to \Omega$

tales que $U = F$ en la frontera de $\bar{\Gamma}$. Utilizando una base polinómica ortonormal $P(\xi)$ esta función puede construirse como el producto tensorial de la base por el valor de F en la frontera de Ω, en dos dimensiones

$$U = P(\xi)F(\eta) + P(\eta)F(\xi) - P(\xi)P(\eta)F$$

y en tres dimensiones

$$U = P(\xi)F(\eta,\zeta) + P(\eta)F(\xi,\zeta) + P(\zeta)F(\xi,\eta) - \\ -P(\xi)P(\eta)F - P(\eta)P(\zeta)F - P(\xi)P(\zeta)F + P(\xi)P(\eta)P(\zeta)F$$

La base polinómica P suelen ser los polinomios cuadráticos de Lagrange. Esto significa que cada uno de los puntos tendrá como posición una interpolación cuadrática entre las fronteras del dominio.

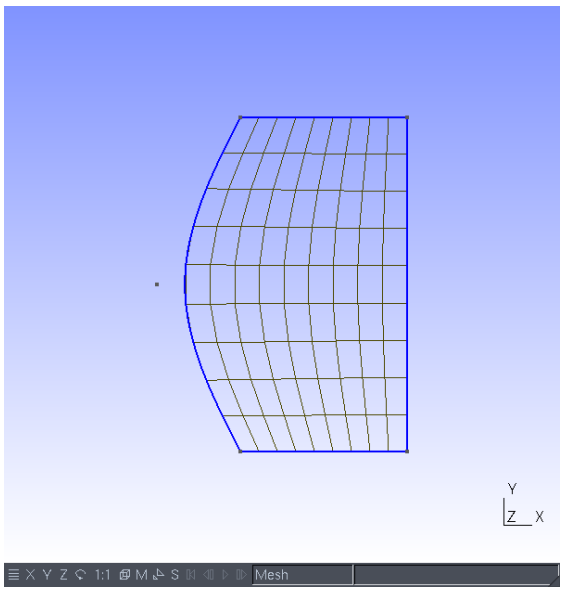

Figura 1.6: Ejemplo de malla estructurada en un dominio simple.

Las mallas estructuradas tienen las siguientes ventajas:

- Tienen una mayor precisión numérica, especialmente en esquemas de volúmenes finitos, puesto que suelen maximizar la relación área/volúmen del elemento
- Permiten un mayor control sobre la geometría, especialmente con mallas multibloque

Y los siguientes inconvenientes:

- Su construcción es más laboriosa
- Pueden ser difíciles de generar en geometrías muy complejas.
- Tienen menos opciones para el refinado en zonas del dominio dadas.

1.2.2. Mallas no estructuradas

Las mallas no estructuradas parten de distribuciones de puntos que cumplen una serie de propiedades dadas, la mayoría de ellas relacionadas con una distribución lo más uniforme posible dentro del contorno. Son adecuadas para geometrías irregulares donde es muy difícil generar una malla estructurada.

GMSH utiliza un algoritmo de creación de malla basado en la triangulación Delaunay. Dado un contorno se demuestra que sólo existe una triangulación óptima en la que, para cada triángulo formado por tres puntos de la malla, no exista ningún punto dentro de la circunferencia que pasa por los tres puntos.

Si bien esta propiedad cierra el problema de la triangulación dados los puntos de la malla, en la mayoría de los casos sólo dispondremos del contorno. GMSH es capaz de encontrar una distribución no estructurada de puntos cuya triangulación resultante tiene suficiente calidad.

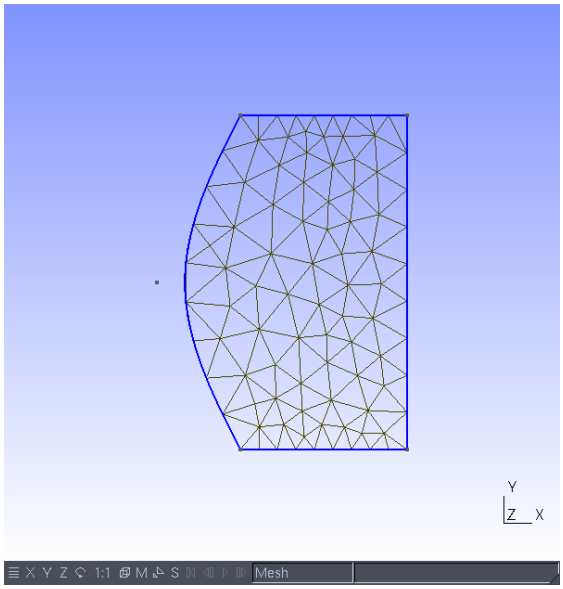

Figura 1.7: Ejemplo de malla no estructurada con la misma geometría y el mismo número de puntos en el contorno que en el caso anterior.

1.3. Discretización y calidad de malla.

Una de las diferencias esenciales entre las mallas estructuradas y no estructuradas es el proceso de refinado. En algunos problemas la malla utilizada no produce suficiente precisión y es necesario rehacerla para aumentarla. En las mallas estructuradas el aumento de precisión no suele suponer un problema más allá de generar demasiados grados de libertad. Sin embargo en las mallas no estructuradas algunos algoritmos de refinado automático pueden funcionar mal.

Uno de los algoritmos más utilizados es el de partir cada uno de los triángulos o tetraedros en dos o más elementos. Este algoritmo recursivo puede refinar puntos arbitrarios en el espacio produciendo una pérdida de precisión el introducir gradientes espúreos debidos a una mala definición de la geometría.

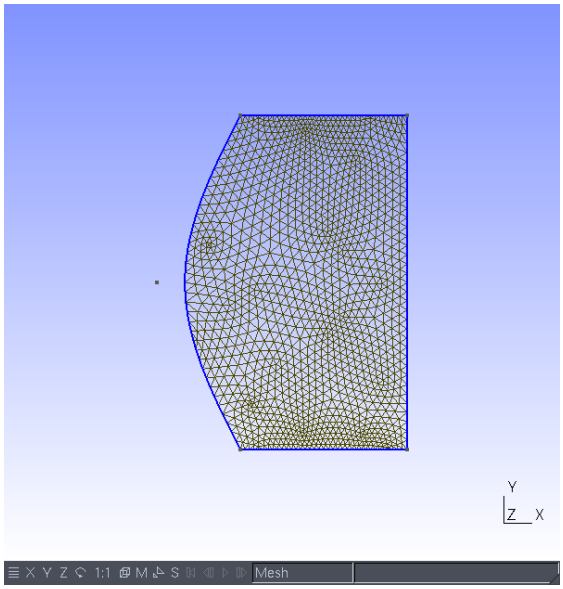

Figura 1.8: Ejemplo de refinado de malla que puede causar problemas. Se pueden apreciar claramente puntos de concentración de triángulos derivados del algoritmo de refinado que pueden afectar significativamente al resultado de la solución de cualquier problema fluidodinámico definido en el dominio mallado.

Por este motivo aumentar la precisión en las mallas no estructuradas puede necesitar una reconfiguración completa de la definición de la geometría para impedir todos estos efectos nocivos.

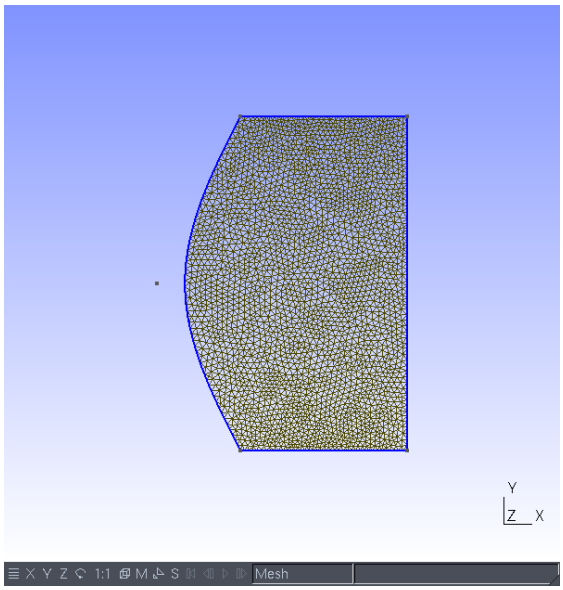

Figura 1.9: Ejemplo de reconfiguración de geometría para obtener un mallado más fino.

1.4. La interfaz gráfica de GMSH

Aunque técnicamente GMSH es un lenguaje de programación cuenta con una interfaz gráfica que, además de ser una herramienta de visualización en sí, permite realizar la mayoría de las operaciones que se describirán en los sucesivos capítulos mediante menús.

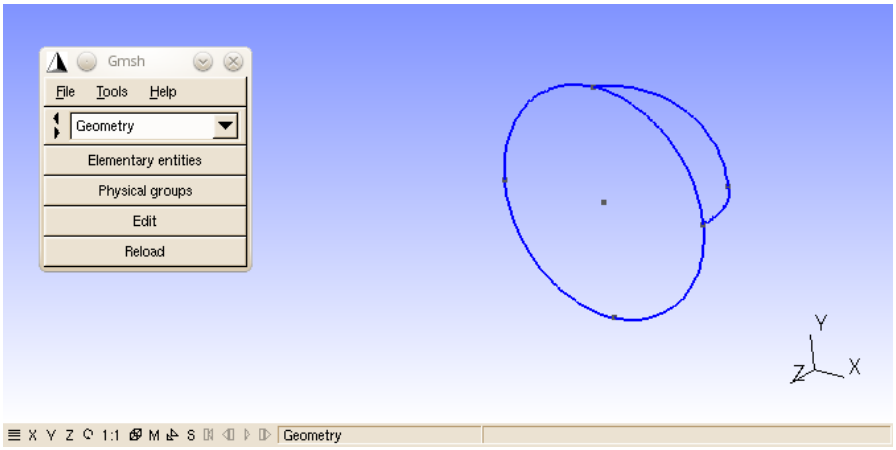

Figura 1.10: Captura de la interfaz gráfica de GMSH

La interfaz gráfica consta de dos ventanas, una dedicada a comandos y otra dedicada a la visualización. Cuenta también con una amplia colección de atajos de teclado para poder cambiar la vista y seleccionar los menús con mayor rapidez.

La ventana de visualización, la que se presenta con un fondo azulado, es muy intuitiva. A parte de permitir rotaciones, movimientos y ampliaciones con los botones y la rueda del ratón podemos cambiar el comportamiento y el tipo de proyección con los botones de la barra inferior. Es de especial utilidad el hecho que cuando se pasa con el ratón por encima de una entidad geométrica el número que la identifica aparece en la barra inferior derecha.

La mayor desventaja del uso de la interfaz gráfica es que cada operación que genera una entidad adicional da números de identificación consecutivos pero arbitrarios lo que hace mucho más complicado escribir un código que dependa de dichas operaciones. La consecuencia directa es que o bien se utiliza la interfaz gráfica como apoyo a la escritura de scripts de definición geométrica o se genera la geometría enteramente con la interfaz gráfica.

La ventana de operaciones consta de una barra de menú, un menú desplegable y botones de operaciones.

De la barra de menú cabe destacar el selector de visibilidad en la entrada de herramientas. Es un gestor que permite cambiar el estado de cada una de las entidades existentes ya sean geométricas o físicas.

El menú desplegable selecciona entre los cuatro estados posibles de GMSH, sólo hablaremos de Geometry y Mesh. Si dentro del estado de Geometry pulsamos el botón *Elementary Entities*, podremos crear, borrar o manipular entidades geométricas. En el menú de de la figura 1.11 vemos las funciones correspondientes a la creación de entidades geométricas. Todas estas funciones disponen de su equivalente en el lenguaje de programación GMSH y se describirán con suficiente precisión en los capítulos siguientes.

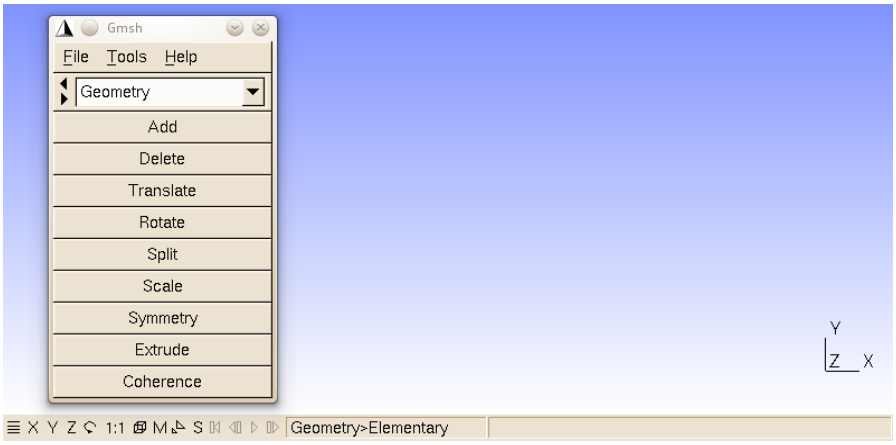

Figura 1.11: Captura de la ventana de menú de creación de entidades geométricas.

La otra entrada importante dentro del menú desplegable es *Mesh*. En él se listan las distintas funciones relacionadas con el mallado y su definición. Destacan los botones correspondientes a la generación y visualización del mallado llamados *1D*, 2D y *3D* que generan las mallas en estas dimensiones respectivamente.

Chapter 2

Entidades geométricas

Las entidades geométricas en gmsh son los puntos, las líneas, las superficies y los volúmenes. Cada una de estas entidades se define mediante entidades de dimensión inferior. Las líneas son definidas por un conjunto de puntos, las superficies por un conjunto de líneas y los volúmenes por un conjunto de superficies.

Cualquier entidad geométrica, sea del tipo que sea, necesita un identificador numérico. A la vez, este identificador puede estár asociado a una variable o a una lista de puntos. Estas variables y estos identificadores deben ser alocateados, bien manualmente o bien con los comandos siguientes.

```
1 cl1 = 0.1;
2
3 p1 = newp; Point(p1) = {-1, -1, 0, cl1};
4 p2 = newp; Point(p2) = {1, -1, 0, cl1};
5 p3 = newp; Point(p3) = {1, 1, 0, cl1};
6 p4 = newp; Point(p4) = {-1, 1, 0, cl1};
7
8 l1 = newl; Line(l1) = {p1, p2};
9 l2 = newl; Line(l2) = {p2, p3};
10 l3 = newl; Line(l3) = {p3, p4};
11 l4 = newl; Line(l4) = {p4, p1};
12
13 ll1 = newll; Line Loop(ll1) = {l1, l2, l3, l4};
14
15 s1 = news; Plane Surface(s1) = {ll1};
```

En el listado de programa anterior vemos cuatro comandos que sirven para generar automáticamente el identificador numérico, cada uno para un tipo distinto de entidad geométrica. Cada punto, línea, superfície, volumen, ciclo de líneas y cíclo de superficies debe tener un identificador único dentro del grupo. Además de `newp`, `newl`, `news` y `newll` para los puntos, líneas, superficies y ciclos de líneas disponemos también de `newv` para volúmenes y `newsl` para ciclos de superficies.

2.1. Puntos

La entidad geométrica más simple es el punto. La sintaxis para generar un punto en el lenguaje GMSH es la siguiente:

```
Point(pid) = {px,py,pz,cl};
```

Además de las tres coordenadas espaciales se proporciona un cuarto argumento que es una longitud característica. Esta longitud es la que determina el tamaño de elemento característico de la malla en el punto correspondiente. Por ejemplo, el listado anterior proporcionaría la malla de la figura 2.1

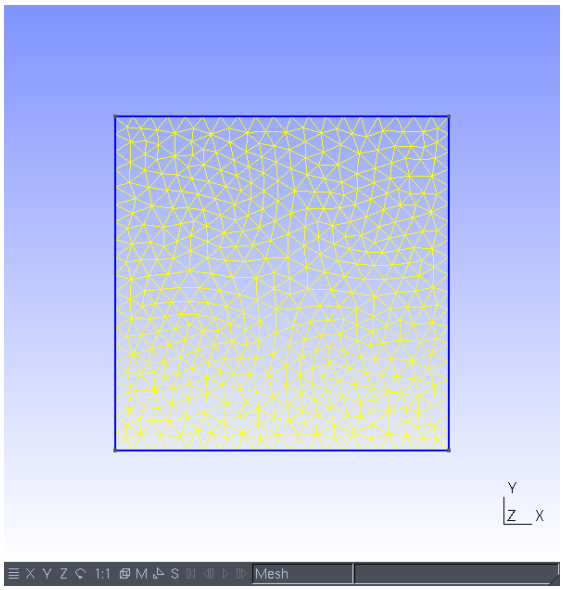

Figura 2.1: Malla generada a partir de un cuadrado de lado 2 y longitud característica 0.1 en todos los puntos. De este modo el número de puntos por lado del cuadrado es exactamente 20.

Mientras que si cambiamos la longitud característica de los puntos superiores con un factor de 5

```
1  cl1 = 0.1;
2
3  p1 = newp; Point(p1) = {-1, -1, 0, cl1};
4  p2 = newp; Point(p2) = {1, -1, 0, cl1};
5  p3 = newp; Point(p3) = {1, 1, 0, 5*cl1};
6  p4 = newp; Point(p4) = {-1, 1, 0, 5*cl1};
7
8  l1 = newl; Line(l1) = {p1, p2};
9  l2 = newl; Line(l2) = {p2, p3};
10 l3 = newl; Line(l3) = {p3, p4};
11 l4 = newl; Line(l4) = {p4, p1};
12
13 ll1 = newll; Line Loop(ll1) = {l1, l2, l3, l4};
14
15 s1 = news; Plane Surface(s1) = {ll1};
16
17 Mesh.Smoothing = 100;
```

Obtenemos

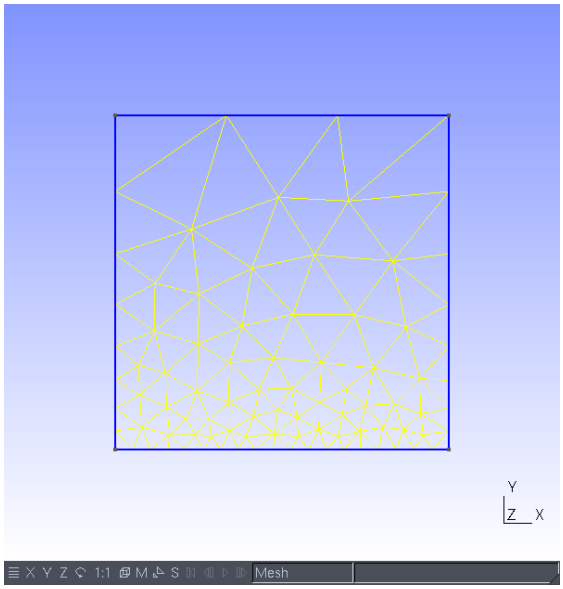

Figura 2.2: Malla obtenida, en un cuadrado de longitud y altura 2, con una longitud característica de 0.1 en los puntos inferiores y una longitud característica de 0.5 en los superiores. Se puede comprobar que el número de puntos en el lado superior es precisamente 4.

2.2. Líneas.

Disponemos de una gran cantidad de tipos de líneas entre las que se cuentan

1. Líneas rectas. `Line{p1,p2}` define una línea recta entre dos puntos
2. Arcos de circunferencia. `Circle{p1,p2,p3}` define un arco de circunferencia desde el punto p1 al punto p3 con centro en p2.
3. Arcos de elipse. `Ellipse{p1,p2,p3,p4}` define un arco de elipse desde p1 a p4 con centro en p2 y siendp p3 cualquier punto del semieje mayor de la elipse.
4. B-splines. `BSpline{p1,p2,p3,p4}` define una b-spline del punto p1 al p4 con puntos de control p2 y p3. Si se introducen sólo 3 argumentos se duplica el punto de control dando una curva simétrica respecto a la mediatriz del segmento p1-p4. Para ser más concretos, esta curva es una aproximación de Schoenberg definida como en Riesenfeld [1973][1] puesto que se pueden introducir tantos puntos intermedios entre el primero y el último como aproximación de una curva pasando obteniendo un spline que sólo pasa entre por el primero y el último. Pronto utilizaremos esta propiedad para la generación automatizada de formas.
5. Líneas compuestas. `Compound Line{list-expr}` define una línea a partir de otras líneas definidas previamente. Los argumentos son los identificadores de línea.
6. Ciclos de líneas. `Line Loop{list-expr}` define un ciclo cerrado de líneas que pueden definir bien una superficie recta o una superfice reglada.

[1] *Applications of B-spline approximation to geometric problems of computer-aided design. UTEC-CSc-73-126, Marzo 1973, Computer Science, Univ. Utah, Salt Lake City.*

2.2.1. Direccionalidad

Muchas entidades como las líneas o las superfícies, que veremos a continuación, tienen una orientación dada. En el caso de las líneas la orientación es dada por la ordenación de los puntos: desde el primero hasta el último. En el caso de las superficies se aplica la regla de la mano derecha para dar una orientación a la normal principal. Esta propiedad, esencial para poder definir condiciones de creación de superfícies y volúmenes, impone una condición adicional a la definición geométrica de los ciclos de líneas y de superfícies, que también se presentarán a continuación.

Para cambiar la orientación de una entidad geométrica basta con cambiar el signo de su número identificador poniendo un signo menos justo delante. En el caso que este número identificador esté contenido en una variable, el efecto de cambiar el signo de la variable será el mismo.

2.3. Superficies

Las superficies en GMSH siempre se definen a partir de un marco formado por líneas. Disponemos de los siguientes tipos de superficie:

1. Superficie plana. `Plane Surface{list-expr}` genera una superficie plana a partir de un ciclo de líneas. El primer argumento es el exterior cerrado de la superficie mientras que el resto de argumentos son los agujeros de la misma.
2. Superficie reglada. `Ruled Surface{list-expr} <In Sphere {p1}>` crea una superficie reglada a partir del ciclo de líneas dado. El argumento adicional impone que se trate de un parche esférico con centro en p1.
3. Superficie compuesta. `Compound Surface{list-expr}` crea una superficie a base de otras superficies más elementales que es reparametrizada como única.
4. Superficie cíclica. `Surface Loop{expr-list}` crea un ciclo de superfícies a base de superficies más elementales para definir el contorno de un volumen.

Por ejemplo, para definir la superficie de una esfera a partir de parches de superfice reglada podemos hacer lo siguiente

```
1 cl1 = 0.3;
2
3 cent = newp; Point(cent) = {0, 0, 0, cl1};
4 north = newp; Point(north) = {0, 0, 1, cl1};
5 south = newp; Point(south) = {0, 0, -1, cl1};
6 p4 = newp; Point(p4) = {1, 0, 0, cl1};
7 p5 = newp; Point(p5) = {-1, 0, 0, cl1};
8 p6 = newp; Point(p6) = {0, 1, 0, cl1};
9 p7 = newp; Point(p7) = {0, -1, 0, cl1};
10
11 l1 = newl; Circle(l1) = {north,cent,p4};
12 l2 = newl; Circle(l2) = {north,cent,p5};
13 l3 = newl; Circle(l3) = {north,cent,p6};
14 l4 = newl; Circle(l4) = {north,cent,p7};
15
16 l5 = newl; Circle(l5) = {south,cent,p4};
17 l6 = newl; Circle(l6) = {south,cent,p5};
```

```
18 l7 = newl; Circle(l7) = {south,cent,p6};
19 l8 = newl; Circle(l8) = {south,cent,p7};
20
21 l9 = newl; Circle(l9) = {p4,cent,p6};
22 l10 = newl; Circle(l10) = {p6,cent,p5};
23 l11 = newl; Circle(l11) = {p5,cent,p7};
24 l12 = newl; Circle(l12) = {p7,cent,p4};
25
26 ll1 = newll; Line Loop(ll1) = {l1,l9,-l3};
27 ll2 = newll; Line Loop(ll2) = {l2,l11,-l4};
28 ll3 = newll; Line Loop(ll3) = {l3,l10,-l2};
29 ll4 = newll; Line Loop(ll4) = {l4,l12,-l1};
30
31 ll5 = newll; Line Loop(ll5) = {l5,l9,-l7};
32 ll6 = newll; Line Loop(ll6) = {l6,l11,-l8};
33 ll7 = newll; Line Loop(ll7) = {l7,l10,-l6};
34 ll8 = newll; Line Loop(ll8) = {l8,l12,-l5};
35
36 s1 = news; Ruled Surface(s1) = {ll1} In Sphere {cent};
37 s2 = news; Ruled Surface(s2) = {ll2} In Sphere {cent};
38 s3 = news; Ruled Surface(s3) = {ll3} In Sphere {cent};
39 s4 = news; Ruled Surface(s4) = {ll4} In Sphere {cent};
40 s5 = news; Ruled Surface(s5) = {ll5} In Sphere {cent};
41 s6 = news; Ruled Surface(s6) = {ll6} In Sphere {cent};
42 s7 = news; Ruled Surface(s7) = {ll7} In Sphere {cent};
43 s8 = news; Ruled Surface(s8) = {ll8} In Sphere {cent};
44
45 sphere = news; Surface Loop(sphere) = {s1,s3,s2,s4,s5,s7,s6,s8};
46
47 svol = newv; Volume(svol) = {sphere};
```

Lo que genera la malla superficial de la figura 2.3

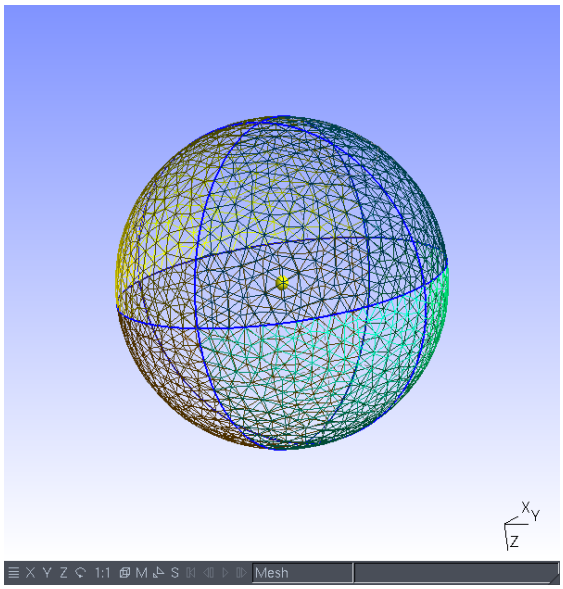

Figura 2.3: Ejemplo de esfera generada a partir de 8 superficies regladas.

Si queremos acumular más puntos sobre el punto situado en al norte de la esfera basta con cambiar la longitud característica asociada a este punto con

```
north = newp; Point(north) = {0,0,1,cl1/10};
```

y obtendremos la malla superficial de la figura 2.4

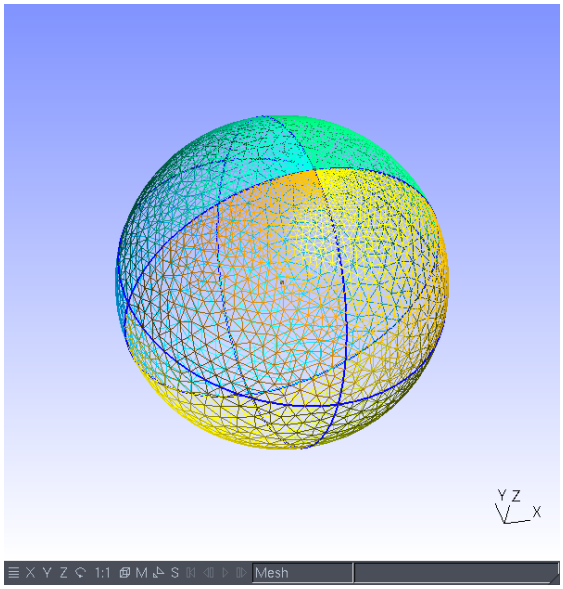

Figura 2.4: Refinado de malla en un punto de la esfera.

2.4. Volúmenes

Los volúmenes se definen siempre a partir de un conjunto convexo y cerrado de superficies. Los tipos de volúmenes disponibles son:

1. Volumen. `Volume{expr-list}` crea un volumen a partir de ciclos de superficies. El primer ciclo especificado representa el contorno exterior del volumen mientras que los siguientes son los agujeros en el mismo.
2. Volumen compuesto. `Compound Volume{expr-list}` crea un volumen compuesto a partir de distintos volumenes.

En el listado de programa anterior también se incluye cómo se puede definir un volumen. Una vez se han mallado los segmentos y las superficies con los botones Mesh 1D y Mesh 2D se puede mallar el volumen definido con el botón Mesh 3D. El resultado es el de la figura 2.5

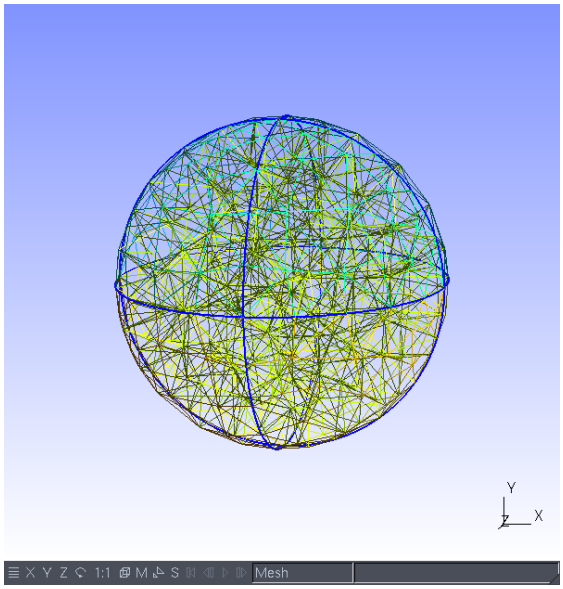

Figura 2.5: Malla del interior de la esfera

Capítulo 3

Transformaciones geométricas.

GMSH define dos tipos de transformaciones geométricas, las transformaciones y las extrusiones. Las transformaciones modifican la geometría de una entidad sin generar ninguna entidad nueva. Las extrusiones generan una entidad de dimensión mayor mediante la operación de extrusión.

3.1. Transformaciones.

Todas las transformaciones operan sobre entidades de la misma manera, aunque algunas con más argumentos que otras. Existen las siguientes transformaciones

Dilate Aplica una homotecia a una entidad o a la copia de la misma. Recibe dos argumentos, el primero es la posición del punto del centro de la homotecia y el segundo es un escalar que representa el factor de escala.

Rotate Rotación. Recibe tres argumentos, el vector del eje de rotación, un punto del eje de rotación y el ángulo de rotación.

Symmetry Simetría respecto a un plano. A parte de los objetos sobre los que se realizará la simetría debe recibir los cuatro coeficientes de la ecucación implícita del plano de simetría como $ax + by + cz + d = 0$.

Translate Traslada cualquier entidad dando también el vector de traslación.

Existen dos comportamientos, transformar la entidad o transformar una copia de la misma. Para ello, en vez de pasar la entidad pasaremos un duplicado con la palabra clave *duplicata*. Por ejemplo tomemos un plano de ecuación z=0 y de lado 2 centrado en el origen utilizando cuatro puntos y trasladaremos una copia según el vector $\vec{v} = (0, 0, 2)$ para crear un plano. Luego rotaremos dos copias de los planos anteriores para crear el resto de lados del cubo. Para finalizar haremos una homotecia para generar un hipercubo.

```
1 p1 = newp; Point(p1) = {1,-1,0};
2 p2 = newp; Point(p2) = {1,1,0};
3 p3 = newp; Point(p3) = {-1,1,0};
4 p4 = newp; Point(p4) = {-1,-1,0};
5
```

```
 6 l1 = newl; Line(l1) = {p1,p2};
 7 l2 = newl; Line(l2) = {p2,p3};
 8 l3 = newl; Line(l3) = {p3,p4};
 9 l4 = newl; Line(l4) = {p4,p1};
10
11 ll1 = newll; Line Loop(ll1) = {l1,l2,l3,l4};
12 s1 = news; Plane Surface(s1) = {ll1};
13
14 // Genero una superficie a partir del duplicado
15 surf[] = Translate {0,0,2}{
16   Duplicata { Surface {s1} ;}
17 };
18
19 s2 = news; s2 = surf[0];
20
21 // Dos rotaciones con copia para generar los otros
22 // lados del cubo.
23
24 surf[] = Rotate{{1,0,0},{0,0,1},Pi/2}{
25   Duplicata { Surface {s1,s2} ;}
26 };
27
28 s3 = news; s4 = news;
29 s3 = surf[0]; s4 = surf[1];
30
31 surf[] = Rotate{{0,1,0},{0,0,1},Pi/2}{
32   Duplicata { Surface {s1,s2} ;}
33 };
34
35 s5 = news; s6 = news;
36 s5 = surf[0]; s6 = surf[1];
37
38 // Homotecia para crear el exterior del hipercubo
39 surf[] = Dilate{ {0,0,1}, 2}{
40   Duplicata { Surface {s1,s2,s3,s4,s5,s6} ;}
41 };
```

Lo más relevante de este listado se ve en las líneas 15, 24, 31 y 39. Ya hemos dicho que las transformaciones pueden modificar una entidad existente o generar una nueva mediante la función *Duplicata*. Como en este caso estamos generando nuevas entidades cualquier función de transformación devuelve una lista con las mismas.

En GMSH, añadir dos corchetes al final de una nueva variable define una lista vacía. En el caso que queramos generar una lista con valores iniciales utilizaremos esta sintaxis:

```
lista[] = {1,2,3,4};
```

En este caso acabamos de generar una lista que contiene 4 elementos que son sucesivamente los números del 1 al 4. En el caso de las rotaciones y las traslaciones la función nos devuelve por orden las copias transformadas de las entidades por el orden en el que las hemos definido en el *Duplicata*.

3.2. Extrusiones

Vamos a generar exactamente la misma figura con extrusiones en vez de con traslaciones y rotaciones. Las extrusiones son las operaciones básicas de la generación de geometría porque,

como veremos en el siguiente capítulo, permiten generar mallas estructuradas a partir de curvas o superficies. El ejemplo siguiente cuenta con dos extrusiones. La primera convierte una recta en un cuadrado dejando la recta como el borde inferior. La segunda convierte el cuadrado en un cubo. La tercera operación es una homotecia y es idéntica al ejemplo anterior.

```
1 cl1 = 0.1;
2
3 p1 = newp; Point(p1) = {-1, -1, 0, cl1};
4 p2 = newp; Point(p2) = {1, -1, 0, cl1};
5 l1 = newl; Line(l1) = {p1, p2};
6
7 // Primera extrusion para crear la tapa inferior
8 ret[] = Extrude {0, 2, 0}{
9   Line{l1};
10 };
11
12 l2 = newl; l2 = ret[0];
13 s1 = news; s1 = ret[1];
14 l3 = newl; l3 = ret[2];
15 l4 = newl; l4 = ret[3];
16
17 // Extruimos la tapa inferior para generar el cubo
18 ret[] = Extrude {0, 0 ,2}{
19   Surface{s1};
20 };
21
22 s2 = news; s2 = ret[0];
23 v1 = newv; v1 = ret[1];
24 s3 = news; s3 = ret[2];
25 s4 = news; s4 = ret[3];
26 s5 = news; s5 = ret[4];
27 s6 = news; s6 = ret[5];
28
29 // Homotecia para crear el exterior del hipercubo
30 surf[] = Dilate{ {0,0,1}, 2}{
31   Duplicata { Surface {s1,s2,s3,s4,s5,s6} ;}
32 };
```

En las líneas 8 y 18 vemos que, como la extrusión también devuelve entidades geométricas nuevas. Para entender la lista de retorno debemos tener en cuenta siempre el mismo orden:

1. El primer argumento es la entidad idéntica a la generadora que quedará al final de la extrusión
2. El segundo argumento es la entidad de dimensión mayor que ha generado la extrusión. En el caso de una curva será una superficie y en el caso de una superficie será un volumen.
3. Además de la entidad generadora y su copia se crearán las entidades correspondientes a los lados de la extrusión. Los siguientes argumentos de salida, y habrá una cantidad arbitraria de ellos, son los lados restantes del cuerpo extruido.

Capítulo 4

Propiedades de la malla.

Con todo lo descrito hasta ahora seremos capaces de generar mallas no estructuradas con GMSH pero nuestro objetivo puede ser generar malas estructuradas par poder controlar tanto la precsión como el coste computacional del problema planteado. Para que GMSH sea capaz de generar mallas estructuradas necesita algo más de información.

Este capítulo se basa en el ejemplo de la generación de una malla estructurada parametrizada alrededor de un ala de envergadura finita. En el caso final del ala de envergadura finita se muestra cómo se proporcionan los datos necesarios para generar una malla estructurada mediante interpolaciones transfinitas y extrusiones.

4.1. Interpolación transfinita

En la introducción se mencionó el hecho que la generación de malla es un proceso en el que se aumenta sucesivamente la dimensión. Primero se mallan las entidades unidimensionales, luego las bidimensionales y finalmente las tridimensionales. La generación de mallas estructuradas sigue un esquema parecido. Todas las mallas unidimensionales son estructuradas porque siempre existe una transformación conforme entre cualquier distribución de puntos en una recta y una malla equiespaciada cartesiana. Como es evidente esta aformación ya no es cierta en el caso bidimensional. Sin embargo GMSH sólo intentará crear una malla bidimensional estructurada si cada uno de los contornos está definido como malla transfinita. Del mismo modo, sólo generará una malla tridimensional estrcutruada por interpolación transfinita si el conjunto convexo de superficies que encierran el volumen han sido definidas como transfinitas.

Para GMSH, transfinito es un atributo que puede asociarse a cualquier entidad geométrica que se pueda mallar, esto es que tenga dimensión mayor a cero menor o igual a tres.

4.1.1. Interpolación transfinita unidimensional.

Para definir una interpolación transfinita en una recta hay que declararla como transfinita utilizando el comando *Transfinite Line*. El uso de este atributo se ejemplifica perfectamente en el sexto ejemplo del tutorial de GMSH

```
/*********************************************************************
 *
 *  Gmsh tutorial 6
 *
 *  Transfinite meshes
 *
 *********************************************************************/

// Let's use the geometry from the first tutorial as a basis for this
// one
Include "t1.geo";

// Delete the left line and create replace it with 3 new ones
Delete{ Surface{6}; Line{4}; }

p1 = newp; Point(p1) = {-0.05, 0.05, 0, lc};
p2 = newp; Point(p2) = {-0.05, 0.1, 0, lc};

l1 = newl; Line(l1) = {1, p1};
l2 = newl; Line(l2) = {p1, p2};
l3 = newl; Line(l3) = {p2, 4};

// Create surface
Line Loop(1) = {2, -1, l1, l2, l3, -3};
Plane Surface(1) = {1};

// Put 20 points with a refinement toward the extremities on curve 2
Transfinite Line{2} = 20 Using Bump 0.05;

// Put 20 points total on combination of curves l1, l2 and l3 (beware
// that the points p1 and p2 are shared by the curves, so we do not
// create 6 + 6 + 10 = 22 points, but 20!)
Transfinite Line{l1} = 6;
Transfinite Line{l2} = 6;
Transfinite Line{l3} = 10;

// Put 30 points following a geometric progression on curve 1
// (reversed) and on curve 3
Transfinite Line{-1,3} = 30 Using Progression 1.2;

// Define the Surface as transfinite, by specifying the four corners
// of the transfinite interpolation
Transfinite Surface{1} = {1,2,3,4};

// (Note that the list on the right hand side refers to points, not
// curves. When the surface has only 3 or 4 points on its boundary the
// list can be omitted. The way triangles are generated can be
// controlled by appending "Left", "Right" or "Alternate" after the
// list.)

// Recombine the triangles into quads
Recombine Surface{1};

// Apply an elliptic smoother to the grid
Mesh.Smoothing = 100;

Physical Surface(1) = 1;
```

En el ejemplo se puede apreciar como hay dos estrategias posibles de concentración de puntos que pueden ayudarnos a mejorar el comportamiento de la malla. El primero es la concentración

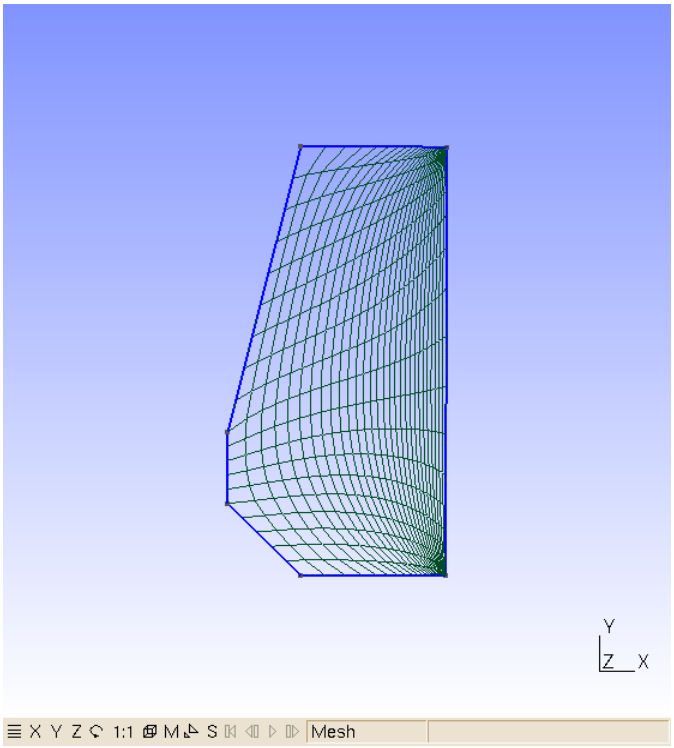

Figura 4.1: Ejemplo de malla producida mediante interpolación transfinita con distintos métodos de concentración de puntos.

de puntos en el centro del segmento o *bump* como en la línea 28. El parámetro que acompaña este atributo va de 0 a infinito; un valor muy grande concentrará todos los puntos en el centro del segmento y un valor muy cercano a cero los acumulará en las esquinas. La otra estrategia es la progresión, mucho más utilizada porque es más controlable. La progresión se define con el parámetro de progresión β que es la distancia entre dos puntos del mallado en función de la distancia de los puntos anteriores. Si la distancia entre el punto p_n y el p_{n-1} es Δx entonces la distancia entre el p_n y el p_{n+1} será de $\beta\Delta x$. Obviamente β puede tomar cualquier valor mientras sea positivo. La orientación de la acumuluación de puntos es la del segmento. Si el segmento va de p_1 a p_2 y si el parámetro $\beta > 1$ entoces los puntos se acumularán cerca de p_1.

Otra característica importante del ejemplo anterior es que se genera una malla transfinita con más de cuatro segmentos. De hecho uno de los conjuntos transfinitos tiene tres segmentos. Esto es posible porque el algoritmo de interpolación transfinita no necesita cuatro segmentos sino cuatro vértices que son precisamente los argumentos de la función *Transfinite Surface* de la línea 43. Una vez definidos los cuatro vértices es importante que el número de puntos colocados en cada segmento que une una pareja de vértices sea idéntico. En caso contrario GMSH dará un error.

4.2. Recombinación

La recombinación es la unión de dos elementos triangulares para formar un paralelepípedo. Aunque esta operación es posible tanto en mallas estructuradas, como en no estructuradas es en las

primeras donde tiene más sentido puesto que se genera una malla parecida a una malla cartesiana. De hecho con una malla recombinada y deshaciendo la interpolación transfinita llegaríamos a una malla cartesiana y equiespaciada.

Por ejemplo la malla de la figura 4.1 ha sido recombinada.

La ventaja de las mallas recombinadas es que reduce el número de elementos de la malla sin reducir el número de nodos ni empeorar el error de la solución. La consecuencia directa es que el sistema de ecuaciones a resolver, aunque tiene el mismo número de variables, tiene una matriz con menos elementos y su resolución requiere menos esfuerzo computacional.

4.3. Extrusión

En el apartado correspondiente ya analizamos la extrusión como una operación que genera entidades geométricas de dimensión mayor. Aunque el uso principal de la extrusión es precisamente el de generar otras entidades geométricas hay casos particulares en los que pueden servir para generar directamente mallas en estas entidades de dimensión mayor.

Conseguirlo es tan sencillo como pasar el comando *Layers* con el número de capas de la extrusón. También se pueden recombinar las celdas para conseguir una extrusión de la malla, de otro modo se haría sólo una extrusión de la geometría y se generarían tetraedros.

```
1 // Mesh.Algorithm = 5;
2 // Mesh.Smoothing = 5;
3
4 // Transfinite Y/N parameter
5
6 fac = 10; // !!!!!DENSITY OF POINTS, THE REST IS TUNED BY HAND
7
8 cord = 1.0; // Cord of the airfoil. Used to define the outer area.
9 cl1 = 0.05;  // Characteristic length of all points of the mesh
10 edge = 0.005;  // Characteristic length of the mesh at any edge.
11 span = 0.01;  // Charecteristic length of the mesh at the maximum width point.
12 width = 5.8;  // Width of the airfoil
13 wake = 3; // Distance of the end of the wake.
14 far = 2; // Distance of the far points.
15 clfar = cl1*far;  // Characteristic length of the mesh at far points.
16 clwake = cl1;
17 np = 4*fac; // Number of points per in the transfinite interpolation. cordwise
18 npa = 2*fac; // Number of divisions of the rotation;
19 npb = 2*fac; // Number of points per int the transfinite interpolation.
       spanwise
20 nph = 3*fac; // Number of points normal to the airfoil
21 npw = np*wake/cord; // Number of points in the wake
22 clus = 1.05; // Clustering parameter
23
24 /* Points and lines that define the airfoil.
25  * Move the points only if you know what you are doing
26  * Because the splines are very sensible to node changes.
27  */
28
29 Point(2) = {0, 0.05, 0, cl1};
30 Point(3) = {0, 0, 0, edge};
31 Point(4) = {0.15, 0.1, 0, cl1};
32 Point(5) = {0.3, 0.1, 0, cl1};
33 Point(7) = {0.45, 0.1, 0, cl1};
34 Point(801) = {cord-0.02, 0.02, 0, edge};
```

```
35 Point(8) = {cord, 0, 0, edge};
36
37 BSpline(1) = {3, 2, 4, 5};
38 BSpline(2) = {5, 7, 801, 8};
39
40 // Points and lines that define the close area.
41 Point(10) = {-0.3, 0, 0, cl1};
42 Point(11) = {0.3, 0.4, 0, cl1};
43 Point(12) = {-0.3, 0.2, 0, cl1};
44 Point(13) = {-0.1, 0.4, 0, cl1};
45 Point(14) = {0.5, 0.4, 0, cl1};
46 Point(15) = {1, 0.3, 0, cl1};
47
48 BSpline(4) = {10, 12, 13, 11};
49 BSpline(5) = {11, 14, 15};
50 Line(6) = {3, 10};
51 Line(7) = {8, 15};
52 Line(8) = {5, 11};
53
54
55 // Points and lines that define the outer area
56
57 Point(16) = {wake, 0, 0, clwake};
58 Point(17) = {wake, 0.3, 0, clfar};
59
60 Line(9) = {15, 17};
61 Line(10) = {8, 16};
62 Point(18) = {-far + 0.3, 0, 0, clfar};
63 Point(19) = {0.3, far, 0, clfar};
64 Point(20) = {wake, far, 0, clfar};
65
66 Line(14) = {18, 10};
67 Line(16) = {19, 11};
68 Point(21) = {0.3, 0, 0, cl1};
69 Circle(17) = {18, 21, 19};
70
71 Point(23) = {1, far, 0, clfar};
72 Line(18) = {19, 23};
73 Line(19) = {23, 20};
74 Line(21) = {23, 15};
75 Line(22) = {20, 17};
76 Line(23) = {17, 16};
77 Line(24) = {8, 16};
78
79 Transfinite Line {1,4,17} = np Using Progression clus;
80 Transfinite Line {-2} = np Using Progression clus;
81 Transfinite Line {-5,-18} = 2*npb;
82
83 // Wake
84 Transfinite Line {10} = npw Using Progression Sqrt(clus);
85 Transfinite Line {9} = npw;
86 Transfinite Line {19} = npw;
87
88 // Normal
89 Transfinite Line {6,8,7} = nph Using Progression clus;
90 Transfinite Line {-23} = nph;
91 Transfinite Line {14,16,21,22} = nph Using Progression 1;
92
93 Line Loop(25) = {14, 4, -16, -17};
94 Plane Surface(26) = {25};
95 Line Loop(27) = {4, -8, -1, 6};
96 Plane Surface(28) = {27};
```

```
97 Line Loop(29) = {16, 5, -21, -18};
98 Plane Surface(30) = {29};
99 Line Loop(31) = {5, -7, -2, 8};
100 Plane Surface(32) = {31};
101 Line Loop(33) = {9, -22, -19, 21};
102 Plane Surface(34) = {33};
103 Line Loop(35) = {10, -23, -9, -7};
104 Plane Surface(36) = {35};
105
106 Transfinite Surface{26};
107 Transfinite Surface{28};
108 Transfinite Surface{30};
109 Transfinite Surface{32};
110 Transfinite Surface{34};
111 Transfinite Surface{36};
112 //Recombine Surface{26,28,30,32,34,36};
113
114 Extrude {{1, 0, 0}, {0, 0, 0}, Pi/2} {
115   Surface{26, 28, 30, 32, 34, 36};
116   Layers{npa};
117   Recombine;
118 }
119
120 Extrude {{1, 0, 0}, {0, 0, 0}, Pi/2} {
121   Surface{70, 112, 151, 134, 92, 53};
122   Layers{npa};
123   Recombine;
124 }
125
126 Extrude {0, 0, -width} {
127   Surface{36, 205, 188, 168, 28, 32, 227, 249, 266, 26, 30, 34};
128   Layers{npb};
129   Recombine;
130 }
131
132 Extrude {{1, 0, 0}, {0, 0, -width}, -Pi/2} {
133   Surface{530, 508, 486, 376, 398, 288};
134   Layers{npa};
135   Recombine;
136 }
137
138 Extrude {{1, 0, 0}, {0, 0, -width}, -Pi/2} {
139   Surface{552, 574, 591, 608, 628, 645};
140   Layers{npa};
141   Recombine;
142 }
143
144 Physical Volume("domain") = {27, 26, 25, 22, 23, 24, 28, 29, 30, 33, 32, 34,
       35, 17, 18, 36, 13, 3, 5, 31, 15, 2, 4, 1, 14, 6, 16, 7, 8, 9, 21, 20, 19,
        12, 11, 10};
145
146 Physical Surface("inflow") = {590, 485, 52, 265, 463, 703};
147 Physical Surface("sides") = {573, 507, 91, 248, 441, 687, 525, 129, 222, 415,
       662, 547};
148 Physical Surface("outflow") = {543, 658, 411, 218, 125, 521, 279, 637, 742,
       301, 197, 143};
149 Physical Surface("airfoil") = {371, 393, 327, 349, 68, 166, 107, 183, 606,
       718, 623, 733};
```

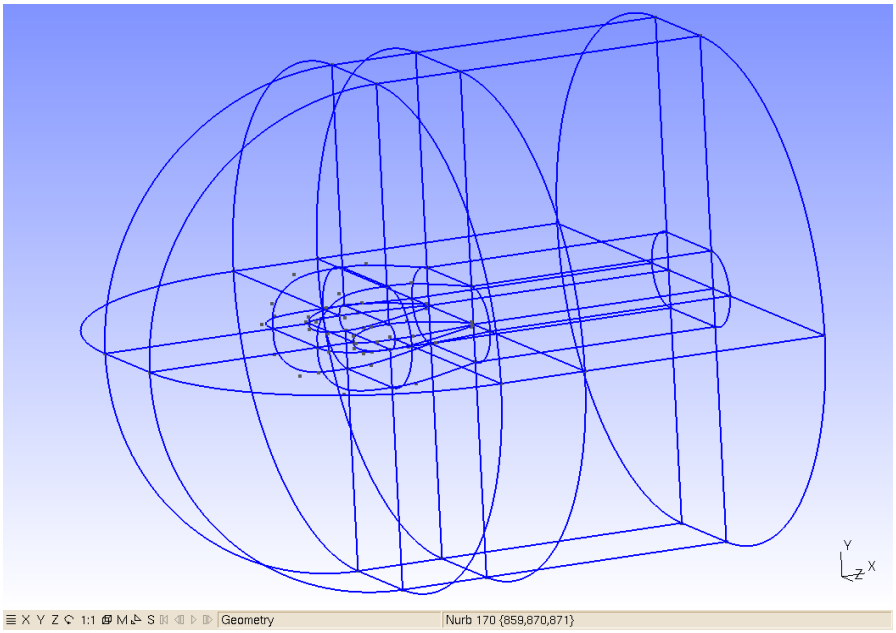

Figura 4.2: Zonas definidas mediante las extrusiones de las superficies elementales.

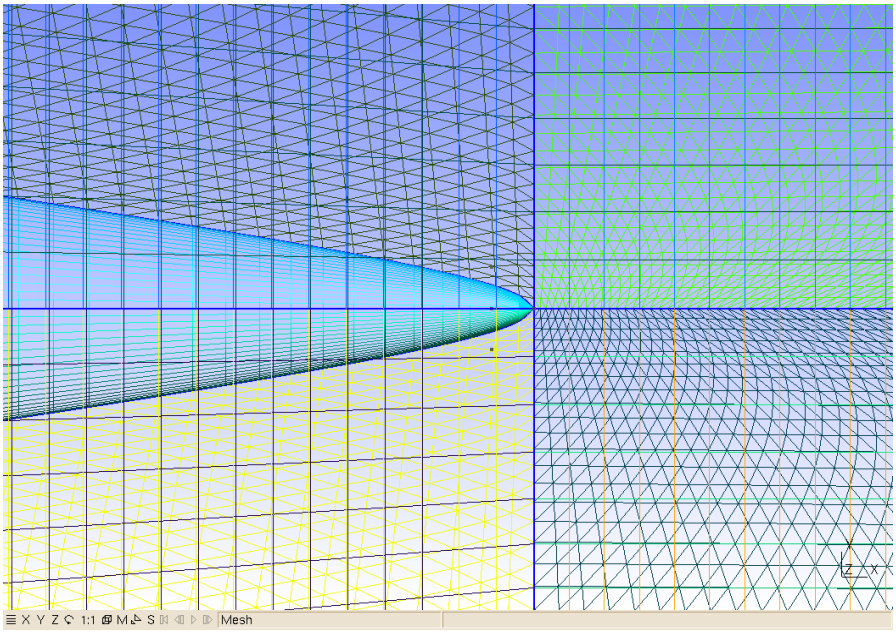

Figura 4.3: Detalle del borde de salida del perfil. Nótese la complejidad en la discretización que produce la acumulación de puntos. Sin este tipo de estrategias el esquema numérico podría desestabilizarse.

Capítulo 5

Entidades físicas.

GMSH diferencia entre la descripción geométrica del caso y la descripción física que luego se exporta como malla. Esta diferenciación es útil porque permite no tener que borrar entidades geométricas auxiliares necesarias para generar la malla. También permite diferenciar los tipos de entidades geométricas de los tipos de entidades físicas puesto que un conjunto convexo de superficies, si bien es útil para definir un volúmen, no tiene ningún sentido desde el punto de vista físico.

Todo lo que sea definido como una entidad física será luego exportado como tal en el momento que se genere una malla. Si deseamos generar una malla volumétrica con las consiguientes condiciones de contorno debemos definir un volúmen físico y las superficies exteriores (de forma única o diferenciando entre ellas) como contorno.

Las entidades físicas y las entidades geométricas no son necesariamente equivalentes, esto es, una entidad física puede estar formada por distintas entidades geométricas con la condición que tengan la misma dimensión. Por ejemplo, un volúmen físico pude contener varios volúmenes geométricos pero no puede contener un volúmen geométrico y una superfície geométrica. No es ni siquiera necesario que las entidades geométricas que forman una entidad física estén conectadas entre sí

Las entidades geométricas posibles son las siguientes

1. Puntos físicos. `Physical Point(expr)={expr-list}`. El primer argumento es bien un número o una cadena de caracteres identificadora del punto físico. La lista del segundo argumento es la lista de los puntos geométricos que forman el punto físico
2. Líneas físicas. `Physical Line(expr)={list-expr}`. El funcionamento es análogo que con el `Physical Point`.
3. Superfícies físicas. `Physical Surface(expr) = {list-expr}`. El funcionamiento es análogo que con el `Physical Point`.
4. Volúmenes físicos. `Physical Volume(expr)={list-expr}`. El funcionamiento es análogo al del `Physical Point`.

El argumento *expr* puede ser bien un número o un nombre entrecomillado. Tanto el nombre como el número asignado se conservará cuando la malla se exporte si el formato lo permite.

Aunque en la descripción geométrica y en la malla se dé un nombre a una entidad geométrica es posible que al traducir de un formato a otro este nombre cambie y se le asigne un número. Es, por ejemplo, el caso de la herramienta ElmerGrid que traduce del formato msh al formato de malla propio del código Elmer.

Por ejemplo, supongamos que queremos mallar una cavidad rectangular y exportar la malla en el formato propio de GMSH, .msh. Para generar la malla y las condiciones de contorno necesitaremos tres entidades físicas.

1. El volumen de la cavidad
2. La tapa superior de la cavidad
3. Las fronteras de la cavidad.

Aunque una cavidad cúbica tiene seis lados, sólo la condición de contorno en la tapa superior es distinta a la de las demás. Por este motivo necesitamos sólo tres entidades geométricas.

Por ejemplo, para genera la malla de una cavidad, debemos especificar las fronteras y el dominio mediante entidades físicas.

```
cl1 = 0.01;
np = 50;

Point(1) = {0, 0, 0, cl1};
Point(2) = {1, 0, 0, cl1};
Line(1) = {1, 2};
Transfinite Line {1} = np Using Progression 1;

Extrude {0, 1, 0}{
  Line{1};
  Layers{np};
  Recombine;
}

Extrude {0, 0 ,1}{
  Surface{5};
  Layers{np};
  Recombine;
}
Physical Surface("top") = {27};
Physical Surface("walls") = {26, 14, 18, 22, 5};
Physical Volume("domain") = {1};
```

Durante esta documentación se han ido presentando varios ejemplos sobre cómo definir entidades físicas para exportar sólo lo que nos interesa de la malla definiendo entidades físicas. Otro ejemplo es el caso en el que hay agujeros en el interior de un volumen. Es posible que el propio agujero sea en sí un volúmen así que nos interesa sólo definir el volúmeno físico que representa nuestro dominio.

En el siguiente listado se ejemplifica como generar un volumen físico que corresponde al encerrado entre los dos cubos.

```
cl1 = 0.2;

p1 = newp; Point(p1) = {-1, -1, 0, cl1};
p2 = newp; Point(p2) = {1, -1, 0, cl1};
```

```
5 l1 = newl; Line(l1) = {p1, p2};
6
7 // Primera extrusion para crear la tapa inferior
8 ret[] = Extrude {0, 2, 0}{
9   Line{l1};
10 };
11
12 l2 = newl; l2 = ret[0];
13 s1 = news; s1 = ret[1];
14 l3 = newl; l3 = ret[2];
15 l4 = newl; l4 = ret[3];
16
17 // Extruimos la tapa inferior para generar el cubo
18 ret[] = Extrude {0, 0 ,2}{
19   Surface{s1};
20 };
21
22 s2 = news; s2 = ret[0];
23 v1 = newv; v1 = ret[1];
24 s3 = news; s3 = ret[2];
25 s4 = news; s4 = ret[3];
26 s5 = news; s5 = ret[4];
27 s6 = news; s6 = ret[5];
28
29 sl1 = newsl;
30 Surface Loop(sl1) = {s1,s2,s3,s4,s5,s6};
31
32 // Homotecia para crear el exterior del hipercubo
33 ret[] = Dilate{ {0,0,1}, 2}{
34   Duplicata { Surface {s1,s2,s3,s4,s5,s6} ;}
35 };
36
37 s11 = news; s11 = ret[0];
38 s12 = news; s12 = ret[1];
39 s13 = news; s13 = ret[2];
40 s14 = news; s14 = ret[3];
41 s15 = news; s15 = ret[4];
42 s16 = news; s16 = ret[5];
43 sl2 = newsl; Surface Loop (sl2) = {s11,s12,s13,s14,s15,s16};
44
45 v2 = newv; Volume(v2) = {sl1,sl2};
46
47 Physical Volume("domain") = {v2};
48 Physical Surface("bottom") = {s11};
49 Physical Surface("top") = {s12};
50 Physical Surface("walls") = {s13,s14,s15,s16};
51 Physical Surface("hole") = {s1,s2,s3,s4,s5,s6};
```

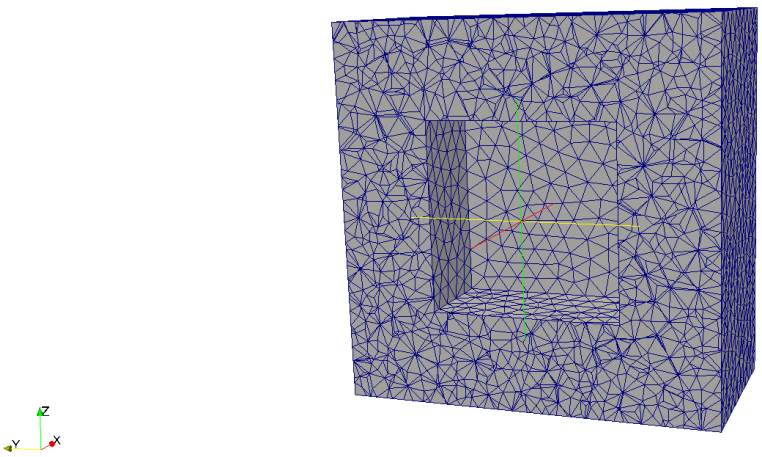

Figura 5.1: Visualización de la malla exportada.

Vemos que el volumen correspondiente es el definido como *v2*, ignorando el volumen generado por la extrusión *v1*. Esto significa que tenemos que identificar las superficies generadas por la extrusión para definir los *surface loops*. Definiremos un *surface loop* para el cubo exterior y otro para el cubo interior y luego los pasaremos como argumentos de la función *Volume* para poder generar el volumen deseado.

GMSH no cuenta con la posibilidad de generar volúmenes mediante operaciones booleanas (intersecciones, uniones, sustracciones) así que todos los cuerpos deben ser definidos mediante sus superficies.

Para generar cuerpos físicos de mayor complejidad se pueden generar con herramientas de CAD e importarlas y remallarlas con GMSH pero este procedimiento queda fuera de los objetivos de este manual.