ASADES

Acta de la XLII Reunión de Trabajo de la Asociación Argentina de Energías Renovables y Medio Ambiente
Vol. 7, pp 06.119-06.128, 2019, Impreso en Argentina
ISBN 978-987-29873-4-2

UNA APLICACION PARA LA GENERACION DE REDES DE DISCRETIZACION CON blockMesh/OpenFOAM PARA LA SIMULACION DE CONVECCION EN EDIFICIOS

Luis Cardon
INENCO - (UNSA-CONICET) - Facultad de Ciencias Exactas
Universidad Nacional de Salta, Av. Bolivia 5150, 4400, Salta, Argentina
Email: cardonluis2008@gmail.com

Recibido 16/08/19, aceptado 21/10/19

RESUMEN. Se ha redactado un programa en C para facilitar la generación de redes estructuradas de aplicación en la simulación de aspectos térmicos en viviendas y edificios con OpenFOAM. La geometría de viviendas y edificios es tal que en la mayoría de los casos se puede describir mediante bloques prismáticos exahédricos estructurados. OpenFOAM posee un especificación de redes estructuradas basada en bloques prismáticos hexahédricos descriptos por las coordenadas de sus vértices. Estos bloques admiten la especificación de la discretización no uniforme en cada una de las tres direcciones espaciales. El utilitario `blockMesh` genera redes para OpenFOAM a partir de un archivo de datos con la descripción de los vértices y otros datos de la red que usualmente elabora manualmente. El programa desarrollados permite generar el archivo de datos para `blockMesh` para redes de bloques estructurados a partir de muy pocos datos necesarios para describir la planta y la elevación de los dominios típicos de la arquitectura.

Palabras claves: OpenFOAM, red, discretización, MFC.

INTRODUCCION

OpenFOAM (Open-source Field Operation And Manipulation) (OPENFOAM) es un software para el desarrollo de resolutores numéricos, con herramientas adecuadas para el pre y pos procesamiento. Está orientado a la solución de problemas descriptos por ecuaciones diferenciales, en particular de aquellos que surgen del campo de la mecánica del continuo, incluyendo mecánica de fluidos computacional. Los `solvers` de OpenFOAM representan modelos físico-matemáticos descriptos por EDP en términos de un metalenguaje, y define para ellos distintos modelos de discretización y algoritmos de resolución iterativos. El software se distribuye con varios `solvers` para la resolución de distintos problemas, por ejemplo, flujos incompresibles, flujo multifásico, electromagnetismo, mecánica de sólidos, etc.. Estos ofrecen al usuario un punto de partida para desarrollar sus propio modelos.

Nuestro grupo de trabajo ha hecho experiencia en el área de convección natural en cavidades para lo cual OpenFOAM posee desarrollados distintos modelos. Una de las áreas de nuestro interés es el flujo de aire en habitaciones, viviendas o edificios y una parte importante de la tarea de preparación de la simulación es la construcción de la red de discretización, para lo que se dispone varias herramientas.

OpenFOAM acepta redes producidas por generadores con interfaces gráficas, tales como Fluent, Star-CD, GAMBIT, ANSYS, CFX y SALOME, entre otros. Los archivos generados por estos programas tienen su propio formato y deben convertirse al formato de OpenFOAM con utilidades específicas. Debe cuidarse aspectos relacionados con la calidad de la red y las condiciones de borde. Estos generadores de red son adecuados para geometrías complicadas, no obstante sus redes son en alguna medida más difíciles de parametrizar o su modificación requieren operar nuevamente con la GUI.

OpenFOAM posee dos aplicaciones propias para generar redes. El utilitario `snappyHexMesh` generar automáticamente mallas complejas de células hexaédricas que se obtienen a partir de geometrías de superficie trianguladas. `blockMesh` es en cambio una aplicación sencilla para generar redes estructuradas. El programa utilitario `bloquesMesh` lee un archivo denominado `blockMeshDict`, un diccionario en la nomenclatura del programa, en donde el usuario hace la descripción de la red.

En este trabajo se presente un programa utilitario denominado `bloquesMeshH.c` desarrollado para generar el diccionario `blockMeshDict` necesario para discretizar el dominio de cálculo típico de habitaciones, viviendas o edificios. Estos dominios se pueden describir mediante una red estructurada de bloques prismáticos paralelepípedos que pueden representarán habitaciones, o las

paredes y aberturas que dividen o comunican estas habitaciones entre si. Son necesarios muy pocos datos para generar la red: el número de bloques (habitaciones) en cada dirección y el tamaño de cada bloque en cada dirección.

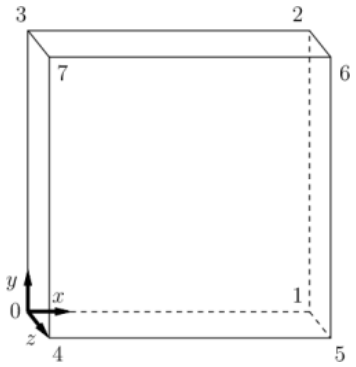
Otras características más complejas de la geometría de las viviendas, tales como techos inclinados, requieren un modificación manual del diccionario generado por `bloquesMeshH.c`. Otras características, tales como bloques edilicios no alineados, protuberantes u ocluidos parcial o totalmente, requieren ser embebidos en una red de bloques que los contenga que se modifica a *psoteriori*.

La aplicación blockMesh y el diccionario blockMeshDict

OpenFOAM provee un utilitario `blockMesh` para la generación de redes estructuradas paramétricas graduadas y con bordes curvos que se guardan en el formato `polyMesh`. El utilitario requiere como datos un diccionario (un archivo escrito con una sintaxis determinada) con las especificaciones necesarias para definir los bloques que componen el dominio de cálculo y su discretización. El utilitario `blockMesh` lee el diccionario `blockMeshDict` y genera la red en una serie de archivos que se guardan en el directorio `/constant/polyMesh`. El diccionario `blockMeshDict` tiene tres secciones.

Una lista ordenada de vértices, cada uno definido por una tríada de coordenadas, que describen la geometría de una red de bloques exahédricos. Los vértices se ingresan en un determinado orden que se describirá luego. El número de orden de ingreso, contado desde cero, identifica a cada vértice y cada bloque quedará definido por ocho vértices.

```
vertices
(
    (0 0 0) // vertice 0
    (1 0 0) // vertice 1
    (1 1 0) // vertice 2
    (0 1 0) // vertice 3
    (0 0 0.1) // vertice 4
    (1 0 0.1) // vertice 5
    (1 1 0.1) // vertice 6
    (0 1 0.1) // vertice 7
);
```



Una lista de bloques, cada uno de ellos descripto por los números de vértices que lo definen. Conjuntamente con la definición de los bloques se especifica la discretización en cada una de las tres direcciones coordenadas, dándose el número de celdas o volúmenes de control. También se especifica aquí los tres coeficientes que establecen el grado de variación del paso de la discretización.

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);
```

La definición de la superficies de contorno. La última parte de la definición de la red permite definir las caras (*patches* en la terminología de OpenFOAM) de aquellos bloques que corresponden a la frontera del dominio de cálculo donde se especificarán las condiciones de borde. El siguiente es un ejemplo de definición de dos de los *parches* de borde

```
 boundary
(
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3) // cara lateral izquierda
            (1 5 4 0) // cara lateral derecha
        );
```

```
    }
);
```

Las superficies de frontera

Un aspecto que requiere cuidado en la generación manual de la red pero más aun en la automática, es la especificación de las caras de bloques o *patches* que constituirán la frontera del dominio de cálculo y en donde se aplicarán las condiciones de borde. Un requisito obvio es que estas caras o *patches* estén sobre la frontera del dominio de cálculo y que sobre ellas esté definida la normal exterior. La determinación de las caras de los bloques sobre la frontera puede hacerse en forma automática y así lo hace el programa `bloquesMeshH.c` como se describe más adelante.

Las caras o *patches* que definen los bloques tienen, como toda área, una orientación dada por su vector normal. Una superficie asociada a un volumen tiene por supuesto una normal exterior y otra interior. La dirección queda definida arbitrariamente por la secuencia en que se listan sus vértices en la definición del *patch*: la circulación antihoraria define un *patch* con normal exterior (Figura 1).

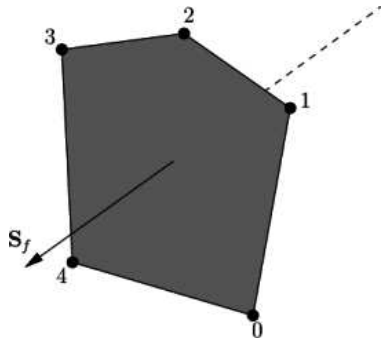


Figura 1: Definición de la normal a una cara.

La especificación de las caras o *patches* que pasarán a constituir la frontera del dominio de cálculo debe hacerse de tal manera que quede determinada como una cara externa. Para ello se sigue la siguiente regla: si el listado de sus vértices se ordena en una secuencia antihoraria, se tiene una cara con normal exterior.

EL PROGRAMA bloquesMeshH.c

El programa `bloquesMeshH.c` desarrollado crea el diccionario `blockMeshDict`; define la red de bloques paralepipédicos estructurados que describen el dominio de cálculo, define la malla de discretización que se aplicará a cada bloque, determina las caras de los bloques en las frontera de cálculo. La especificación de las condiciones de borde sobre los *patches* de borde debe hacerse a posteriori en forma manual. El programa también permite determinar *patches* internos que se utilizarán cuando sea necesario eliminar algunos de los bloques de la red. Finalmente escribe el diccionario compatible con el programa `blockMesh` quien generará la red de discretización del problema.

El programa requiere muy pocos datos, es simple de usar y vuelca el diccionario generado a terminal (Linux) de donde se lo puede redireccionar a un archivo que debe denominarse `blockMeshDict`. A los efectos de demostración, se usará el plano de planta de un pequeño departamento de la ciudad de Salta que se muestra a la izquierda, en el plano de planta de la figura 2.

Para generar la red se divide el plano de planta con líneas paralelas a todas las paredes del edificio. La figura 3, muestra el resultado de hacerlo sobre el plano de planta de la figura 2. La vivienda queda dividida de esta manera en seis bloques. Todos ellos poseen superficie externas donde se aplicarán las condiciones de borde. Note que todos los bloque forman un espacio continuo, salvo el bloque 4, que representa el cuarto de baño, constituirán en el dominio de cálculo para el problema. El bloque 3 será extraído del domino de cálculo y al extraerlo, alguna de las superficies de los bloques 1, 5 y 4 se convertirán en externas y requerirá condiciones de borde. En líneas generales el programa tiene las siguientes partes:

- **Entrada de datos.** El programa requiere como datos el número de bloques que se construirán en cada dimensión (Nx, Ny, Nz.) así como la longitud de cada uno de los bloques

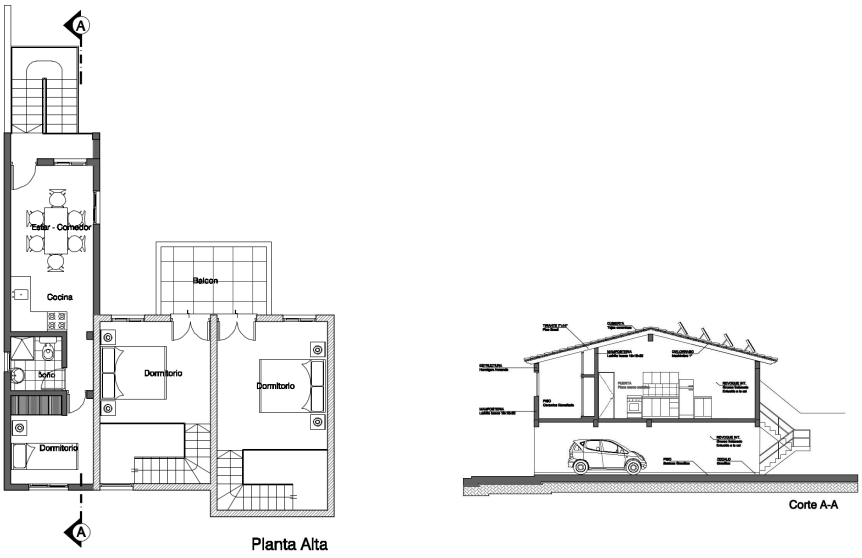


Figura 2: Vista de planta y corte transversal del departamento tomado como ejemplo.

1	3	5
2	4	6

Figura 3: Discretización esquemática en bloques del departamento mostrado en la figura

bloques en cada una del las tres direcciones del espacio. Estas longitudes se almacenan en tres arreglos $dx[]$, $dy[]$, $dz[]$, cuya dimensión es el número de bloques correspondientes a cada dirección. En el caso del ejemplo, será $Nx = 3$, $Ny = 2$, $Nz = 1$. Los arreglos $dx[]$, $dy[]$, $dz[]$ tendrán ahora dimensión 3, 2 y 1 respectivamente. Habrá que asignarles valores a sus elementos según las dimensiones indicadas en el plano.

- **Generación de la lista de vértices.** Los bloques, aun no construidos, se definen por sus vértices. La función `coordenadas_vertices` genera las coordenadas de los vértices ubicando los bloques en una red estructurada. Para ello recorre los planos en profundidad, y cada plano fila por fila. El primer vértice, $(0, 0, 0)$, define el origen de coordenadas. La función imprime las coordenadas de los vértices directamente a pantalla de donde se la puede redireccionar a un archivo.

- **Generación de la lista de bloques.** La estructura implícita de la red estructurada de bloques, numerados consecutivamente, determina los ocho vértices que los definen. Esto último se realiza en la función `numera_vertices`.

Definición de los bloques

Los bloques se numeran recorriendo la red de bloques en el mismo orden en que se recorren los nodos de un bloque. Un triple lazo permite recorrerlos en orden y calcular, el número de vértice correspondiente al origen de coordenadas local del bloque, valor que se registra en la variable `nVerticeGlobal`. La estructura de la red de bloques permite, a partir `nVerticeGlobal`, numerar los restantes siete vértices del bloque, lo que se hace en la función `coordendas_vertices`, que manda los resultados directamente a la terminal.

```
1   void coordendas_vertices(int Nx, int Ny ,int Nz,
2        float dx[], float dy[], float dz[] )
3   {
4     int i,j,k,  ib;
5     float x0,y0,z0;
6   ib=0;
7   z0=0.;
8   for(k=0;k<=Nz;k++)
9      {y0=0.;
10      for(j=0;j<=Ny;j++)
11         {x0=0.;
12          for(i=0;i<=Nx;i++)
13             {printf("(  %3.1f  %3.1f  %3.1f  )  // nodo %d\n",x0,y0,z0, ib);
14              x0=x0+dx[i];
15              ib=ib+1;
16             }
17          y0=y0+dy[j];
18         }
19       z0=z0+dz[k];
20      }
21  }
```

```
1   // ================================================================
2     void numera_vertices(float nVerticeGlobal, int Nx, int Ny)
3     {int v[8];
4     v[0]= nVerticeGlobal;
5     v[1]= v[0] + 1;
6     v[2]= v[0] + (Nx+1) +1;
7     v[3]= v[2] - 1;
8
9     v[4]= v[0] + (Nx+1)*(Ny+1) ;
10    v[5]= v[1] + (Nx+1)*(Ny+1) ;
11    v[6]= v[2] + (Nx+1)*(Ny+1) ;
12    v[7]= v[3] + (Nx+1)*(Ny+1) ;
13  printf(" hex (%d %d %d %d %d %d %d %d) (%d %d %d) simpleGrading
14    (%d %d %d) ", v[0], v[1],v[2],v[3],v[4],v[5],v[6],v[7],10,10,10,1,1,1);
15  return;
16    }
```

Generación de la lista de bloques.

La creación de las listas que definen los bloques se hace de la siguiente manera. Cada vértice de la red es el asiento (nodo origen local) de un bloque. Por ello, se recorren los vértices de la red que darán lugar a los bloques. Este recorrido debe detenerse a $Nx_i - 1$, ya que los nodos sobre los laterales derecho, de fondo, superior, no generan bloques internos a la red. En nuestra descripción de la red de bloques se ha optado por recorrer primero los vértices del plano normal a x_3 que contiene el origen de coordenadas globales, luego los planos paralelos en la dirección positiva del eje x_3. En cada plano se recorren primero los vértices sobre el eje x_1, en la dirección positiva, luego se recorren las filas de vértices en dirección positiva del eje x_2. La numeración de los vértices así

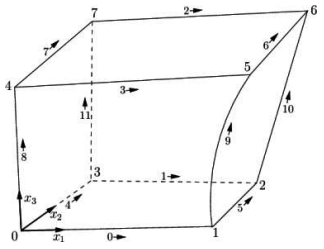


Figura 4: Un bloque simple en OpenFOAM.

recorridos se almacena en la variable `nVerticeGlobal`. Determinado el número del vértice asiento de un bloque en la red global, deben generarse los números de vértices restantes que constituyen el bloque. Para ello se suma al valor del `nVerticeGlobal` una cantidad fija que puede determinarse de la estructura implícita de la red y de las reglas, también predeterminadas, en este caso por la especificación de `blockMeshDict`, de como deben recorrerse los vértices de un bloque. Estas reglas establecen:

- El primer vértice (vértice 0) en la lista de definición de un bloque establece el origen del sistema coordenado local a ese bloque.
- La dirección x_1 corresponde a la dirección del segmento (o arista del bloque) que une el origen con el segundo vértice (vértice 1).
- La dirección de x_2 queda definida por la dirección del segmento entre el segundo y el tercer vértice.
- La dirección de x_3 queda definida por la normal a la superficie que pasa por los tres primeros vértices.
- El cuarto vértice pertenece a la misma superficie que los vértices $0, 1, 2, 3$ y forman un mismo *patch*.

Las reglas precedentes y la estructura implícita de la red permite numerar los vértices de cada bloque en función de los datos globales del la red: Nx, Ny, Nz, lo que se hace en la función `numera_vertices` que se describe a continuación.

La función numera-vertices

La función `numera_vertices` realiza la numeración **global** de los vértices de la siguiente manera, sumando a la numeración local. Para un bloque particular, la numeración **local** se obtiene a partir de la estructura implícita de la red. Dándole el valor 0 al primer vértice, en el origen de coordenadas, que debe coincidir con el sistema coordenado global. El vértice de la derecha sobre el eje x_1, para la misma coordenada x_2 y el mismo plano x_3 debe ser el nodo 1. El tercer vértice debe ser el que define la dirección x_2.

y su numeración local debe ser $1 + Nx \times Ny$. El cuarto es el último vértice sobre vértice sobre la superficie que contiene a los tres precedentes y debe ser $1 + Nx \times Ny - 1$. Los restantes cuatro se construyen de manera similar siguiendo la lógica de las reglas enumeradas en la sección **??** que indica que a la numeración de los cuatro nodos anteriores debe sumársele el número de vértices que tiene el plano normal a x_3, $(Nx + 1) * (Ny + 1)$.

El listado a continuación muestra la implementación en la función `numera_vertices` que genera la lista de definición de bloques.

```
1   // ========================================================
2     void numera_vertices(float nVerticeGlobal, int Nx, int Ny)
3     {int v[8];
4     v[0]= nVerticeGlobal;
5     v[1]= v[0] + 1   ;
6     v[2]= v[0] +  (Nx+1) +1;
7     v[3]= v[2] - 1  ;
8
9     v[4]= v[0] +  (Nx+1)*(Ny+1) ;
10    v[5]= v[1] +  (Nx+1)*(Ny+1) ;
11    v[6]= v[2] +  (Nx+1)*(Ny+1) ;
12    v[7]= v[3] +  (Nx+1)*(Ny+1) ;
13  printf(" hex (%d %d %d %d %d %d %d %d) (%d %d %d) simpleGrading
14        (%d %d %d) ", v[0], v[1],v[2],v[3],v[4],v[5],v[6],v[7],10,10,10,1,1,1);
15  return;
16    }
```

Identificación de los patches de borde

Los *patches* de borde de la red básica se identifican fácilmente a partir de la estructura implícita de la red y de los números de bloques en cada una de las dimensiones.

El cálculo se hace en seis funciones para el piso, techo, el frente y fondo y laterales izquierdo y derecho del dominio de cálculo en sendas funciones de C:

```
void cara_fondo(float Nx, float Ny, float Nz);
void cara_frente(float Nx, float Ny, float Nz);
void cara_piso(float Nx, float Ny, float Nz);
void cara_techo(float Nx, float Ny, float Nz);
void cara_derecha(float Nx, float Ny, float Nz);
void cara_izquierda(float Nx, float Ny, float Nz);
```

Como ejemplo se presenta el listado de la función para el piso:

```
void cara_piso(float Nx, float Ny, float Nz)
{
int i,j,k, nVerticeGlobal;
printf("//piso 25 setiembre ok normal ok---------------\n");
nVerticeGlobal=0;
for(k=0;k<Nz;k++)
{
for(i=0;i<Nx;i++)
{
  piso(nVerticeGlobal,Nx,Ny);
nVerticeGlobal=nVerticeGlobal+1;
}
nVerticeGlobal=nVerticeGlobal+(Nx+1)*(Ny+1)-(Nx);
printf("//---------------\n");
}
printf("//============================--------------\n");
}
```

Anulación de un bloque

La función `bloques` es la encargada de definir los bloques en el formato `exa` requerido por `blockMesh`. Para facilitar la identificación de cada bloque se agregó como comentario al final del mismo su número de orden como se muestra a continuación

```
    hex (64 65 77 76 160 161 173 172) (10 10 10) simpleGrading (1 1 1)   // 59
    hex (65 66 78 77 161 162 174 173) (10 10 10) simpleGrading (1 1 1)   // 60
    hex (66 67 79 78 162 163 175 174) (10 10 10) simpleGrading (1 1 1)   // 61
```

Para producir redes con oclusiones, simplemente se elimina de la lista de bloques él o los bloques que satisfagan la oclusión. Esta operación puede hacerse manualmente, previa identificación numérica del bloque que se quiere eliminar lo cual es bastante engorroso en redes complejas. El programa `bloquesMeshH-3.c` permite hacerlo automáticamente. La misma función `bloques` determina si el bloque debe ser eliminado, en cuyo caso se precede la impresión normal del bloque con el símbolo de comentario de C, //.

Para identificar los bloques automáticamente se requieren datos adicionales que se incorporan como tres arreglos de tipo bandera, uno por cada dimensión, con el mismo número de elementos que el número bloques haya en esa dimensión, todos inicializados en cero. Luego se marcan con 1 los elementos de los arreglos correspondientes a los bloques a ser anulados.

El ejemplo siguiente muestra la especificación de los seis bloques eliminados en la red representada por la Figura 5.

```
int sek,sej,sei;
int BX[XMAX]={0,0,0,0,1,1,1,0,0,0,0};
int BY[XMAX]={0,0,0,0,1,1,1,0,0,0,0};
int BZ[XMAX]={1,0,0,0,0,0,0,0,0,0,0};
```

La función `bloques`, levanta banderas `sek`, `sej` y `sei` si los elementos de los vectores `B*{i}` con `i=x,y,z` son 1 y se anula el bloque si se encuentra las tres banderas levantadas.

La función bloques numera los bloques en el comentario como se muestra para facilitar la verificación de la operación. El listado muestra los bloques anulados

```
// hex (64 65 77 76 160 161 173 172) (10 10 10) simpleGrading (1 1 1)// 59 bloque
// hex (65 66 78 77 161 162 174 173) (10 10 10) simpleGrading (1 1 1)// 60 bloque
// hex (66 67 79 78 162 163 175 174) (10 10 10) simpleGrading (1 1 1)// 61 bloque
```

Anulación de un bloque

La información de los *patches* de borde se da actualmente en una lista denominada `boundary`. Cada *patch* se define en esa lista por su nombre, tipo y por las listas de los vértices de las caras que bloques que integran los *patches*, como se muestra en el siguiente ejemplo

```
frente
 {
     type wall;
     faces
     (
      ( 9 10 13 12 )
      ( 10 11 14 13 )
      ( 12 13 16 15 )
     );
 }
```

El programa `bloquesMeshH-5.c` genera estas listas en forma automática para el caso de una red rectangular de bloques. Genera solo seis *patches*, `frente`, `fondo`, `piso`, `techo`, `izquierda`, `derecha`, uno solo `patch` por cada cara del paralelepípedo. Condiciones de borde localizadas sobre sub *patches* de estas caras deben obtenerse subdividiendo y pos procesando las primeras.

Al eliminar un bloque de la red, se modifican los *patches* de borde, algunos de ellos, los que perteneciendo al bloque eliminado estaban en la frontera del dominio de cálculo deben eliminarse, eliminándose de la lista de `faces` las listas de vértices que lo definen. Simultáneamente al retirar un bloque de la red aparecen nuevos *patches* en el borde del dominio de cálculo reconfigurado. Las listas de vértices que definen estos *patches* deben crearse e incorporarse a las listas de `faces` en el patch correspondiente.

Para facilitar esta tarea, al menos para generar las listas de vértices de las `faces` a incorporar o a eliminar, se ha programa una serie de funciones

```
void cara_izquierda_interna(float Nx,float Ny,float Nz,float Nc, char hacer[]);
void cara_derecha_interna(float Nx, float Ny, float Nz,float Nc, char hacer[]);
void cara_techo_interna(float Nx, float Ny, float Nz, float Nc, char hacer[]);
void cara_piso_interna(float Nx, float Ny, float Nz, float Nc, char hacer[]);
```

Estas funciones identifican y generan las listas de vértices que definen las caras de todos los bloques cuya frontera yace sobre el plano coordenado $x = cte$ o $y = cte$ (falta $z = cte$) que pasa por los vértice número Nc_x o Nc_y, sobre los ejes coordenados. En un pos procesamiento manual, posterior deberá relocalizar estas listas en alguno de los seis *patches* predefinidos: `frente`, `fondo`, `piso`, `techo`, `izquierda`, `derecha`. En la creación de la lista de *patches* que surgen de esta manera debe tenerse en cuenta no solo la ubicación de la cara sino también su orientación. La ubicación del *patch* se realiza identificado el vértice de su esquina inferior izquierda en la numeración global, cuando la cara es mirada desde afuera del dominio de cálculo.

Localizado el vértice, se utiliza la estructura implícita de la red para calcular los demás números de vértices que listados definen el *patch*, según sea la orientación del plano de la cara. Según la orientación, las caras corresponderán a un piso ($\mathbf{n} = \mathbf{e}_2$) o un techo, ($\mathbf{n} = -e_2$), frente y fondo ($\mathbf{n} = \mathbf{e}_3$), ($\mathbf{n} = -e_3$) e izquierda y derecha ($\mathbf{n} = -e_1$), ($\mathbf{n} = \mathbf{e}_2$).

Identificado el nodo vértice y la orientación los demás nodos deben recorrerse en el sentido antihorario para que la lista represente un superficie con normal exterior.

Como ejemplo se lista a continuación la función para un piso interno:

```
void cara_piso_interna(float Nx, float Ny, float Nz, float Nc, char hacer[])
{
int i,j,k,nVerticeGlobal;
printf("     pisoii// %s\n",hacer);
printf("     {\n");
printf("         type wall;\n");
```

```
printf("         faces\n");
printf("         (\n");

nVerticeGlobal=Nc;
printf("// pisoi ---------------, %d\n",nVerticeGlobal);
for(k=0;k<Nz;k++)
{for(i=0;i<Nx;i++)
 {
  piso(nVerticeGlobal,Nx,Ny);
  nVerticeGlobal=nVerticeGlobal+1;
 }
nVerticeGlobal=Nc +(Nx+1)*(Ny+1);
printf("// ---------------\n");
}
printf(");\n");
printf("}\n");
}
```

Cambio de inclinación de las paredes o techos externos Para simular techos inclinados, hace falta modificar la posición de los vértices de algunos bloques. Para hacerlo automáticamente debe especificarse el bloque, la cara que se quiere inclinar y la dirección de la proyección sobre el plano de la cara opuesta de la normal al nuevo plano inclinado.

RESULTADOS

La figura 5 muestra un ejemplo de creación de red con varios bloques en solo dos de las dimensiones. Los bloques se han usado para proporcionar mayor grado de discretización en la adyacencia de las paredes donde habrán capas límites. De la red original se han extraído varios bloques en la parte central superior. A la derecha se muestra la solución con el resolvedor `buoyantBoussinesqSimpleFoam`.

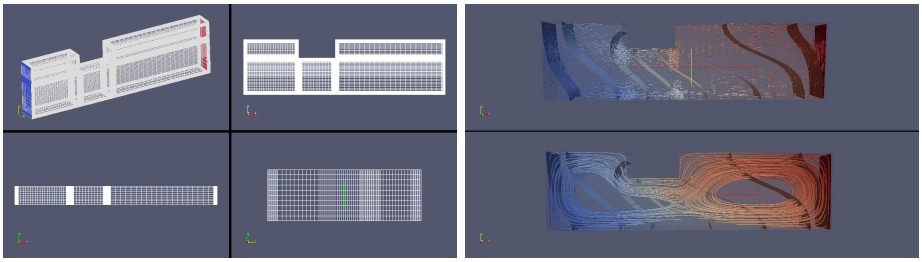


Figura 5: Red de discretización para dos recintos conectados entre recintos generada blockMeshH.c y su resolución con buoyantBoussinesqSimpleFoam.

CONCLUSIONES

Se ha desarrollado una aplicación para la generación de redes de discretización para OpenFOAM orientada y de utilidad en la simulación computacional de la mecánica de fluidos en el interior de habitaciones, viviendas o edificios. En general, puede usarse para discretizar cualquier problema que pueda ser descripto por dominios paralelepipédicos estructurados. La aplicación reduce considerablemente la complejidad de la generación de las redes de discretización necesarias para hacer CFD. La aplicación ha sido probada con éxito para generar las redes que se muestran en el trabajo como se muestra en la simulaciones realizadas para ellas con OpenFOAM.

REFERENCIAS

OPENFOAM. The open source CFD toolbox. [En línea]
Dirección URL: `https://www.openfoam.com/` [Consulta 20 de Agosto de 2019]

AN APPLICATION FOR THE GENERATION OF blockMesh/OpenFOAM DISCRETIZATION MESHES FOR THE SIMULATION OF CONVECTION IN BUILDINGS

ABSTRAC: A C computer program has been written to facilitate the generation of structured meshes for the OpenFOAM software, for application in the simulation of thermal aspects in rooms, houses and buildings. The geometry of homes and buildings is such that in most of the cases they can be described by structured meshes of exahhedral prismatic blocks. OpenFOAM has a specification for structured meshes described by the coordinates of its vertices. These blocks allow the construction of non-uniform grids in each of the three spatial directions. The utility `blockMesh` generates the OpenFOAM meshes from a data file containig the list of the vertices and other data, which the user needs to provides manually. The developed program allows to generate the input data file requiered by `blockMesh` for the construction of meshes from very few data needed to describe the the plant and the elevation of the typical domains of architecture.
Keywords: OpenFOAM, mesh, discretization, CFD.