There are two main reasons for using grids that are not rectangular with uniform grid spacing:
1. Representing a domain with complex boundaries
2. Put grid points in parts of the domain where high resolution is needed
Frequently, it is necessary to deal with both issues.

Gridlines are straight but unevenly spaced:

\[ \xi = \xi(x) \]
Stretching functions - examples

\[ \xi = \xi(x) \]

Boundary layer
\[ \xi = x^2 \]
\[ \xi = x^n \]

Internal layer
\[ \xi = \frac{x - 0.5}{\gamma(x - 0.5)^2 + \delta} + 1 - x \]

Bilinear Interpolation

Simples grid generation is to break the domain into blocks and use bilinear interpolation within each block

Start by breaking the domain into blocks
Then grid each block separately
Consider an arbitrary shaped quadrilateral block.

1. Select the $\xi$ and $\eta$ direction.

2. Divide the opposite sides evenly with $N$ points in the $\xi$ direction and $M$ in the $\eta$ direction and draw straight lines between the points on the opposite sides.

For a single block, we therefore have:

$$x(\xi, \eta) = \begin{cases} \frac{M-\eta}{M-1} \left( \frac{N-\xi}{N-1} x_1 + \frac{\xi-1}{N-1} x_2 \right) \\ \frac{N-\eta}{N-1} \left( \frac{M-\xi}{M-1} y_1 + \frac{\eta-1}{M-1} y_2 \right) \end{cases}$$

Along the edge between points 0 and 1

$$x(\xi, 0) = \frac{M-\eta}{M-1} \left( \frac{N-\xi}{N-1} x_1 + \frac{\xi-1}{N-1} x_2 \right)$$

Along the edge between points 3 and 2

$$x(3, \eta) = \frac{M-\eta}{M-1} \left( \frac{N-\xi}{N-1} x_1 + \frac{\xi-1}{N-1} x_2 \right)$$

Then interpolate again for points between the edges

$$x(\xi, \eta) = \frac{M-\eta}{M-1} \left( \frac{N-\xi}{N-1} x_1 + \frac{\xi-1}{N-1} x_2 \right)$$

The $y$-coordinate is found in the same way.

For many blocks:

1. we must interpolate for each block and
2. ensure that their boundaries are correct.

Short MATLAB program to generate the grid:

```matlab
N=8; M=5; Nblock=3; NTot=0;

for l=1:Nblock
    for i=Nf:N, for j=1:M
        x0=xp(1,l); x1=xp(1,l+1); x3=xp(2,l); x2=xp(2,l+1);
        N=Nblock(l); M=Mblock(l);
        for i=1:N, for j=1:M
            indice(i,j)=l*(M-1)*(N-1)+i*(M-1)+j;
            % Set appropriate parameters for each block
            if (l==1), Nf=1; end
            if (l==2), Nf=2; end
            if (l==3), Nf=3; end
            % Interpolation formula
            y(ii,j)=((M-j)/(M-1))*(((N-i)/(N-1))*y0+((i-1)/(N-1))*y1)+...((j-1)/(M-1))*(((N-i)/(N-1))*x3+((i-1)/(N-1))*x2);
        end
    end, end;
end
end,
```

$$(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)$$
Sometimes the grid can be improved by smoothing. The simplest smoothing is to replace the coordinate of each grid point by the average of the coordinates around it. This process can be repeated several times to improve the smoothness.

\[
x(i, j) = 0.25 \times \{ x(i+1,j) + x(i-1,j) + x(i,j+1) + x(i,j-1) \}
\]

\[
y(i, j) = 0.25 \times \{ y(i+1,j) + y(i-1,j) + y(i,j+1) + y(i,j-1) \}
\]

Bilinear Interpolation can also be used for curved boundaries, if the points on the boundaries are given.

Higher order interpolation functions can also be used to generate stretched grids for complex boundaries.

The grid generation results in an array of x and y coordinates for each grid point: \( x(\xi, \eta) \), \( y(\xi, \eta) \)

For simple problems we can include the grid generator in the fluid solver and generate the coordinates before we solve for the fluid motion.

For more complex problems the grid generation step is usually separated from the flow solver and the grid points are read from a file.

For commercial codes you can generally use many different grid generators—as long as the data format is consistent.
While bilinear interpolation is often the simplest approach for relatively simple domains, it usually requires a fairly large amount of human input. Thus, there have been major attempts to make the grid generation more automatic.

Elliptic Grid Generation

Elliptic Scheme

"Isotherms" of the conduction equation

\[ \nabla^2 T = 0 \]

has nice properties of smoothness and concentrated contour spacing where solution has a large spatial gradient.

Why not use the solution to the Laplace equation as the new coordinate?

\[ \nabla^2 \xi = 0, \nabla^2 \eta = 0 \]
we obtain

Adding the two equations:

\[ x_1 \nabla^2 \xi + x_2 \nabla^2 \eta = x_1 P(\xi, \eta) + x_2 Q(\xi, \eta) \]
\[ y_1 \nabla^2 \xi + y_2 \nabla^2 \eta = y_2 P(\xi, \eta) + y_2 Q(\xi, \eta) \]

we obtain

\[
q_1 x_{\xi\xi} - 2q_1 x_{\xi\eta} + q_1 x_{\eta\eta} = -J^2(P_{\xi\xi} + Q_{\eta\eta}) \\
q_1 y_{\xi\xi} - 2q_1 y_{\xi\eta} + q_1 y_{\eta\eta} = -J^2(P_{\xi\xi} + Q_{\eta\eta})
\]

where

\[
q_1 = x_1^2 + y_1^2 \\
q_2 = x_1 x_2 + y_1 y_2 \\
q_3 = x_2^2 + y_2^2
\]

Further control can be applied by adding a source term:

\[ \nabla^2 \xi = P(\xi, \eta), \ \nabla^2 \eta = Q(\xi, \eta) \]

Proper choice of \( P, Q \) provides the shape of the mesh.

In practice, instead of solving for \( \xi, \eta \) in terms of \( x, y \), we want to solve for \( x, y \) in terms of \( \xi, \eta \).

Transform the Poisson equation into \( (\xi, \eta) \) space.

In discretized form (x-equation):

\[
\begin{align*}
\alpha & = 0.25 \left[ (x_{i+1,j} - x_{i,j})^2 + (y_{i,j+1} - y_{i,j})^2 \right] \\
\beta & = 0.25 \left[ (x_{i+1,j} - x_{i,j}) (x_{i+1,j} - x_{i,j}) + (y_{i,j+1} - y_{i,j}) (y_{i,j+1} - y_{i,j}) \right] \\
\gamma & = 0.25 \left[ (x_{i,j+1} - x_{i,j})^2 + (y_{i,j+1} - y_{i,j})^2 \right]
\end{align*}
\]

\[
\delta = \frac{1}{16} \left[ (x_{i+1,j} - x_{i,j}) (y_{i,j+1} - y_{i,j}) (y_{i,j+1} - y_{i,j}) (x_{i+1,j} - x_{i,j}) \right]
\]

Selection of \( P, Q \)

\[ P(\xi, \eta) = -a_1 \text{sgn}(\xi - \xi_c) \exp(-c_1 |\xi - \xi_c|) \]

\[ Q(\xi, \eta) = -a_2 \text{sgn}(\eta - \eta_c) \exp(-c_2 |\eta - \eta_c|) \]

\( a_1 \) determines how strongly and \( c \) determines the “reach” of the attraction.
Selection of $P, Q$

$$P(\xi, \eta) = -b_m \text{sgn}(\xi - \xi_m) \exp\left(-d_m \left[(\xi - \xi_m)^2 + (\eta - \eta_m)^2\right]^{1/2}\right)$$

Attracts grid lines to the line $\xi = \xi_m$, near the point $(\xi, \eta) = (\xi_m, \eta_m)$.

$b_m$ determines how strongly and $d_m$ determines the "reach" of the attraction.

Further Recommended Reading:


Unstructured
Hexahedron
Grids
And
Block-Structured Grid

Sometimes it is not possible to maintain a regular layout of the grid. In those cases it is necessary to use UNSTRUCTURED grids where the connectivity of the grid is part of the data structure. In some case it is possible to use hexahedron cells.

Block-structured grids

Use a logically rectangular grid in each block
Use fixed grids with the boundaries represented as lines that

\[
\phi = \begin{cases} 
1 & \text{inside the fluid,} \\
0 & \text{inside the solid.} 
\end{cases}
\]

A smooth transition zone between the different region of \(O(h)\) thickness

The total velocity is given by

\[
u(x) = \phi u_f(x) + (1 - \phi) u_s(x)\]

\[
\nu = \begin{cases} 
1 & \text{inside the fluid,} \\
0 & \text{inside the solid.} 
\end{cases}
\]

Where the fluid velocity if found by a prediction step:

\[
u_f^+ = u^n + \Delta t \left( -u^n \cdot \nabla u^n + \nu \nabla^2 u^n \right)
\]

And a correction step

\[
u_f^{+1} = \nu_f^+ - \Delta t \nabla p
\]

Another way to look at this method is to think of adding a force to the Navier-Stokes equation inside the body:

\[
u^{n+1} = \nu^n + \Delta t \left( -u^n \cdot \nabla u^n + \nu \nabla^2 u^n - \frac{\nabla p}{\rho} - f^n \right)
\]

The force is selected in such a way so that the right hand side gives the correct velocity. Thus, there is no need to explicitly find the force, although some authors have done so.

Flow over a cylinder computed using a colocated stretched grid

\[\text{Re}=60\]
Several authors have recently proposed more accurate methods to include boundaries in simulations using fixed regular grids. Most of these methods rely on finding the velocity next to the solid in a way different from the rest of the fluid. In the simplest implementation, the velocity is interpolated.

**Solid Boundaries**

![Solid Boundaries diagram]

Velocity interpolated at this point

*regular* points

Fluid

Solid

**Chimera grids**

Locally body-fitted grid. Need to interpolate between the grids

**Adaptive Mesh Refinement**

Adaptive mesh refinement (AMR)

Use patches of finer grids in regions of high shear and around the front. The patches are moved as needed.

Patches

Split cells

Self-similar blocks
Grid generation for complex geometries is usually a major task, requiring both large amount of time and money. Structured grids generally lead to fast and accurate flow solvers and are therefore preferred where the cost of generating the grid is not excessive.

For situations where the complexity of the domain is such that grid generation is very expensive and where the user can live with modest accuracy, unstructured grids are generally used.
Unstructured versus structured grids

Structured grids: an ordered layout of grid points.

Unstructured grids: an arbitrary layout of grid points. Information about the layout must be provided.

Since the points are laid out in an irregular unstructured way, the point location and connectivity must be provided.

Unstructured grids can consist of arbitrarily shaped control volumes but tetrahedrons (pyramids) and hexahedrons (boxes) are most common.

For a cell based scheme each cell is a control volume and we solve for the average values in each cell.

For vertex based schemes a control volume is constructed by dividing up the cells next to the vertex. The cells can be divided by drawing a line perpendicular to each edge. The corners of the control volume are where two such lines intersect.

The finite volume method is usually used to derive numerical approximations for unstructured grids. However, finite element methods and "mesh free" methods also use unstructured grids.

Here we will briefly outline a vertex centered method for the advection/diffusion equation.

Data Structure: Indirect addressing

<table>
<thead>
<tr>
<th>Node coordinates</th>
<th>Element corners</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$ $y_1$</td>
<td>$n_{11}$ $n_{12}$ $n_{13}$</td>
</tr>
<tr>
<td>$x_2$ $y_2$</td>
<td>$n_{21}$ $n_{22}$ $n_{23}$</td>
</tr>
<tr>
<td>$x_3$ $y_3$</td>
<td>$n_{31}$ $n_{32}$ $n_{33}$</td>
</tr>
<tr>
<td>$x_4$ $y_4$</td>
<td>$n_{41}$ $n_{42}$ $n_{43}$</td>
</tr>
<tr>
<td>$x_5$ $y_5$</td>
<td>$n_{51}$ $n_{52}$ $n_{53}$</td>
</tr>
</tbody>
</table>

x-coordinate of the first corner point of element $m$: $x(m(1))$

Usually it also helps if the elements know about their neighbors.
Vertex centered advection diffusion equation

\[ \frac{\partial f}{\partial t} + \nabla \cdot uf = \nu \nabla^2 f \]

Integrate over a control volume

\[ \int \frac{\partial f}{\partial t} dv + \int \nabla \cdot uf \ dv = \nu \int \nabla^2 f \ dv \]

Convert volume integrals to surface integrals

\[ \frac{\partial}{\partial t} \int_v f dv + \int_s f u \cdot n ds = \nu \int_v \nabla f \cdot n ds \]

Generation of unstructured triangular grids:

Advancing front: Start at one boundary and keep adding points until you run into another boundary

Delaunay triangulation: generate a coarse grid and refine

Numerical approximation

\[ A_j \frac{d}{dt} f_j = - \sum_{\text{all edges}} f_e \cdot u_e \Delta s_e + \nu \sum_{\text{all edges}} \frac{\Delta s_e}{l_e} \]

\[ f_e = \frac{1}{2} (f_j + f_i) \]

\[ A_e = f_j - f_i \]

\[ u_e = \frac{1}{2} (u_j + u_i) \]

\[ l_e = |x_j - x_i| \]

Generally we loop over the elements.

Originally, triangular grids were developed for the compressible Euler equations. Approximating the viscous fluxes and enforcing incompressibility is more complex. For boundary layer like flow, sometimes hybrid tetrahedron/hexahedron grids have been advocated.

More resources:

http://www.erc.msstate.edu
http://www.cfd-online.com/
http://capella.colorado.edu/~laney/thompson.htm