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.

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 but in other cases we need more irregularly shaped control volumes.
Since the points are laid out in an irregular unstructured way, the point location and connectivity must be stored.

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.

While it may seems straight forward to use the polyhedron as a control volume, for time dependent problems this results in a a coupling between the nodal points. The mass in the control volume at \( t + \Delta t \) is, for example:

\[
M_{n+1} = A \left[ \rho_{n} + \rho_{n+1} + \rho_{n+1+1} \right]
\]

\[
A = \frac{1}{2} \left[ x_{B} y_{C} - x_{C} y_{B} + x_{B} y_{A} - x_{A} y_{B} + x_{A} y_{C} - x_{C} y_{A} \right]
\]

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.

Or, we can connect the center of each triangle and the center of the vertex.

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.
Vertex centered advection diffusion equation
\[ \frac{\partial f}{\partial t} + \nabla \cdot uf = \nu \nabla^2 f \]

Integrate over a control volume
\[ \int_v \frac{\partial f}{\partial t} dv + \int_v \nabla \cdot uf dv = v \int_v \nabla^2 f dv \]

Convert volume integrals to surface integrals
\[ \frac{\partial}{\partial t} \int_v f dv + \sum_{e} \int_{S_e} fu \cdot n ds = v \sum_{e} \nabla f \cdot n ds \]

Numerical approximation
\[ A_j \frac{d}{dt} \hat{f}_j = - \sum_{\text{all edges}} f_e u_e \cdot n_e \Delta s_e + \nu \sum_{\text{all edges}} \Delta f_e \]  
\[ f_e = \frac{1}{2}(f_i + f_j) \]  
\[ A_j = \int_v f dv \]  
\[ \Delta f_e = f_j - f_i \]  
\[ u_e = \frac{1}{2}(u_i + u_j) \]  
\[ l_e = |f_j - f_i| \]

Generally we loop over the elements

For the full Euler and the Navier-Stokes equations, the numerical approximations are derived in the same way as for finite volume methods on body fitted grids. While hexahedrons and tetrahedrons are the most common shapes for the control volumes, any shape is possible.

Grid based generation of unstructured meshes
Start with a regular mesh. Split rectangles and move grid points to conform to boundaries.

Octree meshing
Instead of starting with a regular grid, split cells into four cells (in 2D) until grid is refined enough.

Often the size of the triangles is determined by a mesh function that changes from one point in the domain to another.

Advancing front method:
Start at the boundaries and keep adding points until you run into the grid from other boundaries.
A number of algorithms exist to generate Delaunay triangulation. Most are based on starting with a coarse grid and then refine it. Usually points are inserted and the triangles around the new point are reconstructed to be a Delaunay triangulation.

Not a Delaunay triangulation: Points fall within the circle that goes through the points defining the triangle

Delaunay triangulation: By reconnecting the points, we can construct a valid triangulation

Generation of unstructured triangular grids is a well developed field with a number of packages available to do so.

Original triangular grid 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.

Sometimes it is convenient to keep hexahedron control volumes near a boundary to better resolve boundary layers, even if the rest of the domain is gridded using tetrahedrons.

Generation of unstructured triangular grids:
A Delaunay triangulation of a set of points is defined so that no points fall within the circle defined by the three points of a triangle

http://www-2.cs.cmu.edu/~quake/triangle.html


http://www.cs.cmu.edu/~quake/triangle.html

Cartesian Grids for Complex Geometries
No approach beats regular structured grids in terms of efficiency and accuracy. Thus, there have been a number of efforts to use such grids for complex geometries. Usually called Cartesian grid approach.

Two main approaches:

**Cut cells:** Control volumes where the solid surface cuts through are modified in shape to align with the surface. Pioneered for simulations of flows over aerospace structures.

**Immersed Boundaries:** The surface is represented as a lying on "top" of the grid. In older implementations the surface was smoothed onto the grid. In newer versions the solver is modified near the grid. Originally developed for viscous flows.

In cut cell methods the control volume containing a boundary are modified by incorporating the solid boundary into the control surface. The fluxes into the cell are then computed using this new control surface. This approach can result in very small cells and sometimes they are merged with neighboring cells.

The total velocity is given by:

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

Where the fluid velocity is found by a prediction step:

\[ u_f^+ = u^0 + \Delta t \left( -u^0 \cdot \nabla u^0 + \nu \nabla^2 u^0 \right) \]

And a correction step

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

A smooth transition zone between the different region of \( O(h) \) thickness.
The divergence of
\[ \mathbf{u}^{n+1} = \phi \mathbf{u}^{n+1} + (1 - \phi) \mathbf{u}_S^{n+1} \]
but
\[ \nabla \cdot \mathbf{u}^{n+1} = \nabla \cdot (\phi \mathbf{u}^{n+1} + (1 - \phi) \mathbf{u}_S^{n+1}) \]
\[ = \phi \nabla \cdot \mathbf{u}_S^{n+1} + (\mathbf{u}_S^{n+1} - \mathbf{u}_S^{n+1}) \cdot \nabla \phi \]
So it is sufficient to set
\[ \nabla \cdot \mathbf{u}_S^{n+1} = 0 \]
Which result in the standard pressure equation.
Thus, the only thing that needs to be done is to set the velocity inside the body equal to what it should be.

Another way to look at this method is to think of adding a force to the Navier-Stokes equation inside the body:
\[ \mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \left( - \frac{1}{\rho} \nabla p + \mathbf{F} - \frac{1}{\rho} \mathbf{f}_S \right) \]
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.

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.
Computational Fluid Dynamics

Adaptive Mesh Refinement

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

Data structures
- Octrees (Quesdtree in 3D)
- Block structured Cartesian meshes
- Unstructured grid

The fluxes must be matched at the boundaries between the different patches. Interpolating the coarse grid solution to use as boundary conditions for the fine grid generally results in reduced accuracy

PARAMESH
Parallel Adaptive Mesh Refinement

Partial "feature" list from the web site:
- Multidimensional
- Structured Grid Blocks
- Parallelized FORTRAN 90 and C
- Portable
- Uses MPI
- User Tunable Load Balancing
- Support for Conservation Laws
- Distribution contains source code
- Builds upon user's existing codes

Chombo: a Software Framework for Block-Structured AMR

The Chombo package provides a set of tools for implementing finite difference methods for the solution of partial differential equations on block-structured adaptively refined rectangular grids. Both elliptic and time-dependent modules are included. Support for parallel platforms and standardized self-describing file formats are included.

https://seesar.lbl.gov/anag/chombo/index.html

Gerris

Unlike PARAMESH and Chombo, Gerris is a complete code, not a library that you can use with your own code. Features include the ability to track free surfaces and fluid interfaces using the VOF method.

http://gerris.dalembert.upmc.fr/gerris/examples/examples/tangaroa.html

Example of a computation of the flow around a F16XL fighter jet using cut cells and AMR

From:
M. J. Aftosmis, M. J. Berger, J. E. Melton:
Adaptive Cartesian Mesh Generation (downloaded from the web)

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.