1

Computational Fluid Dynamics

Victor Udoewa[1] and Vinod Kumar[2]
[1]*George Washington University,*
USAID Development Engineer,
AAAS Science & Technology Policy Fellow, AAAS, 2009-2011,
[2]*Mechanical Engineering,*
University of Texas at El Paso,
USA

1. Introduction

Computational Fluid Dynamics (CFD) is the emerging field of fluid mechanics in which fluid flow problems are solved and analyzed using computational methods and numerical algorithms. In fluid mechanics, there are generally three routes of work in the field, three ways to conduct experiments. The first category is theoretical, or analytical, fluid mechanics. Theoretical fluid mechanics includes theorizing, manipulating and solving equations with pen and paper. The Navier-Stokes equation governing incompressible fluid flow is an example of theoretical fluid mechanics. Secondly, many engineers and physicist work in the area of experimental fluid mechanics. Experimental fluid mechanics involves conducting actual physical experiments and studying the flow and the effect of various disturbances, shapes, and stimuli on the flow. Examples include waves generated by pools, air flow studies in actual wind tunnels, flow through physical pipes, etc. Lastly, a growing number of engineers, mathematicians, computer scientists, and physicists work in the area of computational fluid dynamics (CFD). In CFD, you may still run an experiment of waves across water, an airplane in a wind tunnel, or flow through pipes, but now it is done through the computer Instead of actual, physical, 3D objects. A computer model is created, and computer programmers code the equations representing the physical laws that govern the flow of the molecules of fluid. Then the flow results (such as velocity and pressure) are output into files that can be visualized through pictures or animation so that you see the result just as you do with physical experiments.

In cases where an analytical, or theoretical, solution exists, CFD simulations and the mathematical models, which are coded in the computer program, are corroborated by comparison to the exact solutions. This comparative check is called validation. CFD is not yet to a point where solutions to problems are used without corroboration by existing, known, analytical or exact solutions when available. Validation is not to be confused with verification, however; validation is a check to make sure that the implemented, coded model accurately represents the conceptual, mathematical description and the solution intended to be modeled.

Still, there are many times when there is no analytical solution. In these cases, one often uses a computational approach. In such cases without a known solution, CFD is used to

approximate a solution. Most often, CFD is used when a computational solution is faster, cheaper, or more convenient. Convenience may be due to time or safety or another reason.
If we wanted to create a database of information about 3D flow around a cylinder for different types of fluids at varying speeds, CFD is quite helpful. It would take quite a long time to change the fluid in our flow container and clean the container for every type of fluid we desired. It would also take some time to change the speed of the flow. In this case, it's much faster to simulate it computationally. Then, anytime we wanted to change the speed of the flow, we simply would change one number in a computer input file. Or if we wanted to change viscosity and density for the fluid (representing a different fluid) we would just change the corresponding values in a computer input file. In this case, CFD is faster.
Now imagine if you were doing space defense work for a government, and you were asked to do fluid dynamics simulation of the combustion dynamics during an explosion or when a space shuttle launched (1). It would take immeasurable amounts of money to do test launches over and over as you studied the combustion dynamics of space shuttle thruster ignition. And it would take large amounts of money to test explosive devices, especially considering the damage they cause. In these cases, CFD is, again, quite helpful. The only costs in CFD are the time of creating a computer model, choosing the right mathematical model, coding it, and the power and computer time required to solve the equations. But it is plain to see that CFD is cheaper.
What if you were hired to help design material for the outfits of swimming athletes? Your company gave you the job of studying sharks and their agile ability to swim and maneuver through the water. So you start by trying to study the fluid dynamics around the shark skin (2). How convenient is it to locate sharks and place them in some type of testing container where you have probes and measuring devices located? How convenient is it to place probes on the body of the shark itself? How safe is it to work with the sharks in that manner? No, it's better to create a computer model of a shark and get the information for the shape, design, feel, and density of its skin and to use this information to run simulations. It is clear that CFD is more convenient in this situation. Sometimes a CFD simulation can be all three—safer, cheaper, and more convenient.
Imagine a situation in which two paratroopers, jumping from both side doors of a military cargo aircraft, always crash into each other down below (3). In order to analyze the fluid dynamics of the problem to see what air flow forces are affecting the paratrooper paths, you would need to perform test jumps with paratroopers. However, that is potentially injurious and not safe. You would also have to rent the plane, pay for the rental by hour, hire the test pilot, and pay for all the equipment for the jump. That is expensive. Lastly, the organization of the use of the military aircraft and personnel and equipment takes many months, and it can take from 6 to 12 months to plan the test. In this case a CFD experiment is more convenient: faster, cheaper, and safer.
Usually solving CFD problems involves three stages. First there is the pre-processing stage. In this stage, the geometric boundaries of the problem are defined. In 3D, a volume is created (in 2D, an area) over which the equation will be solved. This volume is broken into smaller units or cells creating a mesh (though there are meshless methods for computing CFD problems). This may be uniform or non-uniform. Along with constitutive equations, the particular equations are chosen for the problem in order to properly physically model the flow. These equations may be manipulated depending on the mathematical method being used. Boundary conditions are prescribed along the boundary. For time dependent problems, initial conditions are prescribed.

Secondly, the problem is solved numerically. At the least, we usually solve for velocity and pressure, but the list of unknowns can be longer depending on the mathematical model (equation) chosen for the physical situation. Other unknowns may include temperature, energy, and density. The numerical solution is usually computed iteratively for steady-state solutions. For time-dependent problems, a step in time is taken, and the equation is numerically solved again, eventually producing a solution for every time step.

The final step in the CFD process is post-processing. In this stage, the solution is analyzed usually with the help of visualization and possibly animation for dynamic, or time-dependent, problems. It is in this stage that CFD results are usually compared to any previous experimental results or known analytical, or theoretical, solutions. This comparison is usually called validation. Today, confidence in CFD is growing, but we have not arrived at the point of trusting CFD solutions without validation. Even if a particular model is validated, we still corroborate the results of a simulation with experimental or analytical results.

Because of the hybrid nature of CFD, advances in CFD are usually made in three areas: computational and applied mathematics, mechanical/chemical engineering, and computer science/electrical engineering. Some researchers work on new theoretical, mathematical models creating new discretization methods (ways to discretize the problem in order to numerically solve it over the discrete units or cells), or turbulence models. They might publish in applied mathematics or computational mathematics journals. Others work on computer architecture (such as different types of supercomputers or computer clusters), coding techniques such as parallel programming, or speeding up the computational processes through faster mesh generation and mesh reordering. They might publish their results in electrical engineering or computer science journals. Lastly others might use CFD to concentrate on new insights in the engineering aspect of the problem such as the mechanics of bird flight or sharkskin-inspired speedo design for less water resistance, or resistance to blood flow inherent to certain veins. They might publish their results in engineering journals next to experimental or theoretical engineering results. Their focus is on the application more so than the math or computer science.

2. Pre-processing

Pre-processing refers to the work that must be done prior to the actual computational experiment or simulation. This work can be reduced to four general areas: geometry definition, volume division, model choice and definition, and boundary condition definition. For the purposes of this article, we will ascribe the work of coding and the choice of computational implementation to the processing stage called simulation. Sometimes researchers refer to mesh generation as pre-processing in general because a mesh is generated when the volume is divided.

2.1 Geometry definition

The first step is to define the computational domain of the problem. The purpose of this definition is to confine the problem to a finite space and limit the computation. It is true that a plane flying in the air has some residual effects on air flow patterns 1,000 miles away, but because of the negligible nature of those effects, we are relatively safe in looking at the effect of the plane on air flow within a reasonable vicinity of the plane thereby limiting our

computational work and making the problem finite. It would be quite a task to compute the effect of the airplane on the air flow at a certain height around the entire earth.
To limit the flow in this way usually requires defining a domain inside which we will compute the flow, outside of which we will not compute the flow. Usually the geometry of the domain is chosen to be a box of some sort, usually a rectangular prism in 3D or a rectangle in 2D. However, any closed shape may be chosen as long as the shape closes off an inside computational domain from an outside space in which computations will not take place.
A closed space does not imply, however, an empty domain box. For example imagine that we are simulating intravenous blood flow (4) (5) around a cancerous growth. The vascular domain is modeled by a 3D cylindrical prism (our domain box), but we still have an object inside. In this case, our object is a semi-spherical cancerous growth on the surface of a wall of the vein. Traditionally, our domain must be totally closed, so the surface of the domain goes from the wall of the vein, joins the surface of the cancerous growth, and continues on the other side rejoining the wall of the vein creating a closed 3D space that does not go under the tumor but continues over the surface of it. Likewise, if we were calculating flow around a sphere (6), the domain box would be the outer half of the domain surface. The inner half of the surface would be the sphere inside the domain box. Just as in the cancer example, we are not calculating the flow inside the sphere, just as we were not calculating the blood flow in the tumor. But the tumor and sphere form part of the boundaries of the domain helping to close off the computational space in which we are interested in the velocity and pressure of the fluid. Remember, domain boxes may contain objects inside which no flow is calculated, but whose surface forms part of the surface of the domain helping to limit the computational space and better define where the fluid flows.

2.2 Volume division

The second step is volume division or mesh generation. Why we must divide the volume is not obvious until one remembers that it is easier to solve a flow problem over a smaller area or volume than a larger one. So dividing the volume into smaller units transforms the large problem over the entire domain into a large number of smaller problems over smaller sub-domains. However, the real reason we divide the volume is because we seek to find, for example, the velocity and pressure of the fluid at various points throughout the domain volume or area. In order to do this, we fill the inner domain volume or area with nodes—points at which we will calculate, in this instance, velocity and pressure. Once we have filled the inner volume or area with nodes, we connect the nodes with edges (and sides in 3D) creating smaller sub-volume or sub-area elements. For example, if our computational box is a rectangular prism and we fill it with nodes, we can connect the nodes to create quadrilaterals or tetrahedrals. If the domain is a rectangle and we fill the rectangle with nodes, we can connect all the nodes to create small rectangles or triangles. This network of rectangles/triangles and quadrilaterals/tetrahedrals creates a mesh of nodes; a mesh has been generated.
Remembering that the purpose of volume or area division is to create more manageable sub-volumes or sub-areas, it behooves us to evenly space out the distribution of nodes. If we do not, we may find that there are large spaces (volumes or areas) with no nodes. This is problematic because it means some of the sub-volume or sub-area elements will still be large; though our goal is to make them small.

Our second goal is to make each sub-element evenly shaped. In 2D, evenness in both directions means that 2D rectangles tend towards squares or that our triangles are equilateral. In 3D, if evenness is desired in all 3 directions, our quadrilaterals tend toward cubes and the tetrahedrals tend to be equilateral. Oblong and unevenly shaped sub-elements also create spaces (areas or volumes) with fewer nodes than parts of the domain with evenly shaped sub-elements. In these cases, our second goal serves a similar purpose as our first goal: to divide the domain into smaller, more manageable sub-domains with evenly distributed nodes.

An important aspect of mesh generation is choosing the appropriate size, or refinement, of the mesh sub-elements, such that important aspects of the flow are properly resolved. The general rule is that no fluid particle should advance through multiple sub-elements, cells, or units in one time step. Therefore our third goal is to increase the refinement (or the number of nodes) in areas or sub-volumes of increased fluid velocity or vorticity or any interesting fluid flow phenomena that you would like to capture computationally. This will allow us to visualize it later.

So far, the realm of geometry definition and mesh generation fall into the computer science side of CFD. When generating the mesh by defining a geometry and dividing that geometric area or volume, one must decide if one will use uniform sub-elements or non-uniform sub-elements. Definitely in parts of the domain with special flow requiring increased refinement, the elements in those parts will not match the refinement of elements elsewhere. But in the general flow one can still choose a uniform, structured mesh or a non-uniform, unstructured mesh. The ability to create non-uniform meshes is important in CFD because of the physical nature of fluids to occupy and fill any void left unoccupied. When dealing with complex geometries and small nooks, crannies, and crevices of an automobile or a model of a city block, it helps to have unstructured, non-uniform meshes that allow for the modeler to create the best shapes to fit the 2D or 3D space (7).

Likewise, the division of the volume or area in mesh generation requires the modeler to choose between quadrilaterals and rectangular prisms or triangles and tetrahedrals (there are other choices of shapes, as well, such as wedges and pyramids in 3D). Generally, quadrilaterals and rectangular prisms have a more accurate solution than triangles and triangular prisms, but there are ways to increase the accuracy of the latter. Because of the non-uniform and sharp geometries found in fluid problems by nature of fluids, triangles and triangular prisms work better geometrically for CFD applications. Normally, CFD researchers will utilize triangles and triangular prisms and then increase the number of interpolation points inside these elements so that no accuracy is lost. Interpolation points are points inside an element at which the solution is calculated. From these interpolation points, we can approximate the solution at any location inside an element.

All of the choices in dividing the volume and discretizing the mesh have the potential to introduce errors. Such errors, due to bad distribution of nodes or parts of the domain where the refinement is too low, are called discretization errors. These are errors that would disappear if we appropriately divided the volume or area or appropriately discretized the mesh.

2.3 Physical model definition

The first two steps dealt with the computer science side of CFD and there are many CFD engineers who work on geometric mesh discretization and mesh partitioning methods. Step three deals with the computational and applied mathematical side of CFD—choosing the

appropriate mathematical model. When preparing to model a certain fluid flow situation, one must decide which equation accurately describes the fluid flow one wishes to simulate. If no equation currently exists, the CFD engineer must do work in the theoretical side of CFD and formulate a new equation or a more specific equation for his or her specific fluid flow situation. If equations do exist, the CFD engineer must simply choose the correct equation for the fluid flow. This is not a trivial step as sometimes the same situation may require different mathematical models at different velocity regimes or different temperature regimes, for instance. So the specific parameters of the flow must be looked at in detail—velocity, viscosity, density, pressure, etc.—so that the correct equation is chosen.

A good example from fluid mechanics is the Navier-Stokes equation which is the basic or fundamental equation for fluid dynamics. If you remove viscosity from the equation, the Navier-Stokes equations become the Euler equations. Since all fluids have some amount of viscosity this approximation is important in flows in which the viscosity is negligible (8) such as sonic flows. A plane flying at sonic speeds will have air sliding past it, relatively, as if it had no viscosity. So the use of the Navier-Stokes equations also depends on the velocity of the flow, or more accurately the Reynolds number which governs the ratio of the kinematic forces to the inertial forces. You can still go further: if you remove vorticity from the Euler equations, you arrive at the full potential equations. The point in this illustration is that choosing the correct mathematical model is important, sometimes difficult, and always specific to the flow situation.

The choice of the mathematical model affects the unknown values you will compute. Some CFD simulations are really computational fluidothermodynamics because temperature and energy are calculated as well (9). For compressible flows, density is an unknown value and we would seek to solve for this value of density in the simulation. So the choice of mathematical modeling affects what unknowns we will compute. More accurately, the unknowns we want to compute in a given situation (along with other details about the flow situation) may help positively affect our choice of a mathematical model or the need to formulate a new one.

Remember that errors can be introduced at this step as well. If a inappropriate or poorly approximating governing equation is chosen, this affects the final solution. If a governing solution is chosen or formulated for which no analytical solution or experimental solution exists, we lose the opportunity for validation to reduce errors. Any simplification in the model or any untrue assumptions the mathematical model uses introduces errors as well. All of these types of errors can be classified as physical approximation errors because they deal with the physical, mathematical model (not the geometric model).

2.4 Boundary condition definition

After a mathematical model is chosen to model the physical phenomenon, usually boundary conditions must be chosen. This is the fourth step of the pre-processing stage, and this step falls on the applied math side of CFD. Usually we deal with boundary-value problems which require values to be assigned along the boundary of the domain of the problem in order to solve the problem throughout the 2D or 3D space.

For instance, in some problems, one may specify the value of the unknown on the boundary. Imagine prescribing the value of the velocity of the fluid on the boundary. Such a boundary condition is called a Dirichlet or a direct boundary condition because you are setting the

value of the unknown. If one specifies the value of a derivative of the unknown, the boundary condition is called a Neumann or natural boundary condition.
For example in hydrodynamic flow around a submerged rock with moss, a CFD researcher would usually place the boundary condition of free-stream velocity on the entrance side and exit side of the domain box assuming that the entrance side is sufficiently upstream from the rock so as to still be undisturbed, and the exit side is sufficiently downstream that the flow conditions have returned to free-stream conditions. The prescription of free-stream velocity would be Dirichlet or direct boundary conditions. The same researcher might assume stress (a derivative of velocity) to be zero in the direction perpendicular to the side surfaces of the domain box. When she prescribes stress in that direction she is setting Neumann or natural boundary conditions in the direction perpendicular to the side surfaces of the domain box.
Such boundaries are spatial boundaries. For time-dependent problems, there are temporal boundaries in a sense. Time dependent problems require an initial condition, prescribed values for the unknowns set at the temporal start of the simulation. In the same hydrodynamic example, let us say we want to simulate the flow when a rock, half the size of the stationary, mossy rock, was thrown into the river passing next to the mossy rock and hitting the riverbed. To start the simulation we need to have the steady flow of the river around the mossy, stationary rock without the 2nd rock thrown in. Once we have computed this flow, we can use the values of velocity and pressure from this flow as initial conditions for a simulation of a moving 2nd rock that is falling to the bottom of the river. From there the simulation will march in time and use the flow results from the previous time step as initial conditions for the next time step.

3. Simulation

The second phase of CFD work is the actual simulation or the "processing" work once the pre-processing work is completed. However, there are still some pre-simulation decisions to be made. CFD work is done through computers which not only decreases the time it takes to perform calculations, but also increases the amount of calculations that can be done in a given time period. As computing power has increased over the years, CFD has been used to solve larger and larger problems.
Large problems, however, were traditionally reserved for supercomputers. Supercomputers are large computers made up of multiple computers or CPUs. A desktop or laptop computer could only handle so many calculations due to hard drive limitations on different types of computer memory. As computers in general become more advanced, not only has the memory capacity of supercomputers increased, but so has the memory of desktops and laptops increased. This has created a cycle where problems solved by supercomputers today are solved by desktop computers and laptop computers tomorrow. And the problems solved by desktop and laptop computers today were only solved by supercomputers yesterday. For example, historically, a simulation of flow past an automobile was done on supercomputers (10) (11). Today one can create a model of an automobile and run a flow simulation of air flow past the automobile with one desktop or laptop machine. This example is one of many indicative of this ever-improving cycle.
Besides memory the other limitation on computing ability in today's world is clock speed. CFD workers are dependent upon computer scientist researchers to continue to increase the clock speed of microprocessors. In general, the faster computers become, the faster is the

speed CFD scientists and engineers can compute solutions to problems. And as the speed of computations increases, the time to do computations decreases, and CFD scientists and engineers can compute larger problems (as long as they have the memory capacity for the calculations and storage for the solution). Currently, as this article goes to press, the fastest machine in the world is the K computer which computes at 10 Petaflops (12). Flops are floating point operations per second, and the prefix peta means 10^15. Therefore, the K computer can compute 10^16 floating point operations per second.

Supercomputers come in varying shapes and sizes. A small supercomputer may have 36-100 CPUs. A larger supercomputer could have 50,000 different nodes, or CPUs. For example, the K computer has 68,544 CPUs, each with 8 cores (octo-core) for a total of 548,352 cores (12). As well, today we have small supercomputing clusters, in which different CPUs are linked together to act as a supercomputer. Often one will find Linux clusters arranged in this way. Each node of a cluster can actually contain multiple processors itself acting as a single computer. A computer or node with 2 processors is called a dual core machine or node. A computer or node with 4 processors is called a quad-core machine or node. Processors in a multicore machine or cluster can use memory in different ways. Some use a shared memory architecture. In this case, all processors can access all the memory because it is completely shared between all processors. Some may have distributed memory where each processor or node has access only to its own memory. Finally there are hybrid machines like the K machine. Each of the 68,544 CPUs has its own distributed memory. But inside each CPUs memory is a system of 8 cores that share memory.

Historically supercomputers used vector-based architecture, but this created a niche market since codes for such machines could not simply be run on non-vector based machines like a desktop computer. Today laptop computers are very similar to supercomputers because many supercomputers use bus-based architecture which is a modified architecture allowing a desktop computer to run more than one processor like a quad core Linux machine (13).

Because most CFD work is done on clusters or supercomputers, CFD programmers often learn parallel programming, a type of computer programming with instructions or directives for communication and transmission of information between processors/cores or nodes on a supercomputer or cluster. Parallel programming is especially important because the purpose of supercomputing is to divide the large problem into smaller pieces given to each node to process. However, in order to solve the larger problem, the nodes must communicate especially and specifically about border regions of the partitioned mesh.

Once the mesh unit and resolution are chosen in the pre-processing stage, the mesh is partitioned and a piece of the mesh is given to each processor or node. However, sometimes researchers will re-order the mesh for large problems. Inefficient mesh partitioning contributes increasing costs of calculations and time for larger and larger problems. To facilitate calculations, mesh re-ordering schemes seek to minimize communication. Communication is minimized most when each processor or node manages a contiguous portion of the mesh. In this case, each processor only shares geometric nodes on the borders of mesh portions with processors that work on neighboring portions of the mesh. Imagine the opposite situation in which mesh elements are randomly distributed. A processor might have to communicate with 8 other processors if 8 of the bordering elements lie on 8 distinct processors! In CFD work, communication can take more than 50% of the computational time depending on the specific problem and its size. So it is very important to minimize this as much as possible, to leave more computational work for actual mathematical computations.

Examples of mesh partitioning and re-ordering methods include MATLAB's MESHPART (14), METIS (15), and PARMETIS (16) (17). Ordering the mesh so that each CPU has access to cells or units that are connected to each other is important. We also re-order and partition to maintain proper load balance so that no CPU has more work than any other.
All of these choices—mesh unit geometry, mesh resolution, partitioning, and order can affect the ability of each processor to solve the resulting algebraic system of equations. Therefore, researchers work on the parallelism of such computer codes. There are many parallel programming language directives such as OpenMP (18), and MPI (19) and languages such as Manticore (20) and NESL (21).
After the mesh partitioning and re-ordering scheme is chosen, the next step a CFD engineer takes is choosing a discretization method. The mesh partitioning lies mostly in the realm of computer science, but this next step of discretization lies in the area of computational and applied mathematics. The mathematical model and governing equation has been chosen, but the CFD scientists or engineer must choose how to discretize the solution of this model over the entire domain.

3.1 Numerical discretization methods

In computational and applied mathematics, there are different numerical discretization methods (22). Three popular methods are finite difference, finite volume, and finite volume methods. Each of these classes of methods contains many variations usually specific to an application area. Other methods include boundary element methods, higher-resolution methods, and meshless methods like spectral methods.

3.2 Finite difference, finite volume, and finite element methods

The finite difference method is probably the oldest of the main three (23). It lends itself quite well to orderly and structured geometries. It is not used as commonly as the finite element method or finite volume method, probably due to the geometric limitations on applications. Still, there are modern finite difference codes that employ overlapping grids and embedded boundaries allowing the use of the finite difference method for difficult or irregular geometries. However, it is the easiest method to code and is often taught first in courses that teach numerical discretization methods.
The finite volume method is a method in which the governing partial differential equation is solved over smaller finite control volumes (24). Since the governing equations are cast in a conservative manner over each control volume, the fluxes across the volumes are conserved
In terms of tests, applications, validation, and literature, the most robust of all methods is the finite element method (25). The finite element method is a type of residual method in which a residual equation is weighted and integrated over the domain. Since the domain is broken into many elements, this integration actually takes places over each element in the mesh. The finite element method requires more memory than the finite volume method but is also more stable than the finite volume method.
There are other methods as well. Boundary elements methods include methods in which the boundary is meshed into separate sub-elements (26) (27). In 3D, a boundary element method domain would be a 2D surface. In 2D, a boundary element method domain would be represented as a 1D surface or edge. There are also immersed boundary methods to deal with situation in which elastic structures interact with fluid flows (28).
There are numerous other discretization methods. Each of the above discretization methods can be used with functions of varying order. Every increase in the order of the functions

used in the discretization method (from linear to quadratic, quadratic to cubic, from cubic to quartic, etc.) carries increased computational costs since higher order functions have more terms and more coefficients. In fact, higher order functions require more sample points or interpolation points to properly resolve them. Accuracy comes at a price. Still, such accuracy is sometimes warranted in cases where there are sharp gradients or shocks in velocity, pressure, density, or temperature, for instance. In these cases, some of the previous discretization schemes fail and introduce what we call "spurious oscillations" which are not actually physical but a byproduct of computational approximation. In these cases, CFD engineers may choose to use higher-order discretization methods or shock capturing methods such as Total Variation Diminishing (TVD) schemes (29), Essentially Non-Oscillating (ENO) schemes (30), and the Piecewise Parabolic Method (PPM) (31).

Spurious oscillations are an example of discretization errors mentioned earlier. They are also called numerical errors because they are not physical. Specifically, such numerical error resulting in spurious oscillations is often called dispersive error or dispersion. Truncation error is the type of numerical error resulting from the difference between the partial differential equation and the finite equation that we actually code.

Additionally, with CFD we can experience a third error—computer error. For example, for one calculation or one floating point operation, computer roundoff is usually negligible. However, when doing repeated calculations over and over in simulations dealing with billions of nodes at which we solve for multiple unknowns at each node, computer roundoff error can build.

3.3 Turbulence models

Another area of CFD modeling is turbulence modeling. Turbulence modeling is a difficult yet still potentially fruitful area of CFD research because of the computational difficulties it poses. Turbulence is a complicated phenomenon that occurs over a wide range of time scales and length scales. This is where the trouble lies making it impossible to fully resolve turbulent phenomena when using most approaches. Because of the wide ranges of time periods over which turbulent periodicity manifests and the wide range of length scales on which turbulence acts, most CFD engineers and scientist choose to resolve a certain length scale and time scale range of turbulence and model the rest. In this case resolving turbulence means that we actually compute and calculate turbulent quantities like vorticity and velocities and pressures in turbulent regions. Modeling turbulence means that we add an expression into our equation, the effect of which is approximately to create the effect of fully resolved turbulence on our unknown values such as velocity and pressure.

As one increases the range of time scales and length scales over which the turbulence is resolved, one must increase the refinement of the simulation both in length (refinement of the mesh) and in time (temporal refinement—the size of the time steps). This increased refinement increases the computational costs (the number of equations to be solved and the time it takes to compute the entire simulation). If one tends toward the other end of the model-resolve spectrum models all turbulence of all length and time scales, the computational costs decrease but one loses accuracy in the simulation. CFD researchers usually tend to resolve a range of turbulent length and time scales and model the rest. Usually the range is related to the range of interest of the flow simulation. For instance, if someone is simulating gas dynamics in the inner-ballistics of a particular weapon, it

would be best to resolve turbulence on the length range of the chamber inside the weapon and model anything much larger and much smaller. Likewise, any turbulence that occurs over a longer period than the time it takes to fire the weapon would have smaller effects on the simulation; it would be best to model turbulence on longer time scales and much shorter time scales.

The most expensive turbulence scheme is Direct Numerical Simulation (DNS) (32). In DNS, all length scales of the turbulence are resolved and little or no modeling is done. It is not used in cases of extremely complex geometries which can create prohibitive expense in resolving the turbulence especially in special geometrically complex portions of the domain.

Less expensive than DNS, the Large Eddy Simulation (LES) resolves turbulence on large length scales as the name suggests (33). A model is used to represent sub-grid scale effects of turbulence. The computational cost of turbulence at small length scales is reduced through modeling.

Reynolds-averaged Navier-Stokes (RANS) is the oldest approach to turbulence modeling and it is cheaper than LES. It involves solving a version of the transport equations with new Reynolds stresses introduced. This addition brings a 2nd order tensor of unknowns to be solved as well. Examples of RANS methods are K-ε methods (34), Mixing Length Model (35), and the Zero Equation model (35).

The Detached-eddy simulation (DES) is a version of the RANS model in which portions of the grid use RANS turbulence modeling and portions of the grid (or mesh) use an LES model (36). Since RANS is usually cheaper to implement than LES, DES is usually more expensive than using RANS throughout the entire domain or grid and usually cheaper than using LES throughout the entire grid. If the turbulent length scales fit within the grid dimensions or the particular portion of the grid is near a boundary or wall, a RANS model is used. However, when the turbulence length scale exceeds the maximum dimension of the grid, DES switches to an LES model. Care must be taken when creating a mesh over which DES will be used to model turbulence due to the switching between RANS and LES. Therefore thought must be given to proper refinement to minimize computation while maximizing accuracy (especially refinement near walls). DES itself does not utilize zonal functions; there is still one smooth function used across the entire domain regardless of the use of RANS or LES in certain regions.

There are many more turbulence models including the coherent vortex simulation which separates the flow field turbulence into a coherent part and a background noise part, somewhat similar to LES (37). Many of the contributions today are coming through different versions of RANS models.

3.4 Linear algebraic equation system

Once the mathematical equation has been chosen in the pre-processing stage, the mesh has been partitioned and distributed, and the numerical discretization is chosen and coded, the last part of the computer program is to solve the resulting algebraic system for the unknowns. Remember in CFD we are calculating the unknowns (velocity and pressure, for instance) at all nodes in the domain. The algebraic system usually looks like $Ax = b$. Usually with many steady problems, Ax is a linear equation system. Therefore we can just invert the matrix A to find the vector x of unknown values. The problem is that for large computations, for instance a computation involving 500 million nodes, inverting the matrix A takes too long. Remember that the number of nodes does not necessarily equal the

number of unknowns. If you're calculating 5 unknowns at every node, than you must multiply the number of nodes by the number of unknowns per node to get the value of actual number of unknowns and the length of the unknown vector x. So for such large problems, we solve the equation system iteratively. These systems are solved iteratively with iterations such as Newton or the Picard iteration.

In many cases with unsteady flows, Ax represents a system of ordinary or partial differential equations. In this case, CFD researchers must choose either an implicit method or an explicit method to deal with the time integration. An explicit method calculates the solution at a later time based on the current solution. Another way to rephrase that is that it calculates the current solution based on an earlier solution in time. An implicit method calculates the solution at a later time based on both the solution at a later time and the current solution. Rephrasing that, an implicit method calculates the current solution based on both the current solution and the solution at an earlier time. Once a method is chosen and the time derivatives have been expanded using an implicit or explicit method, Ax is usually a nonlinear algebraic system.

As stated earlier, for large problems, it may be too time-consuming or too costly (computationally) to compute this directly by inverting A. Usually, then, the linear system is solved iteratively as well (38).

One must then choose which type of iterative solver to employ to find the unknown vector. Depending on the characteristics of this equation system, an appropriate solver is chosen and employed to solve the system. The major implication of all computational research in this area is that there is no one perfect iterative solver for every situation. Rather there are solvers that are better for certain situations and worse for others. When operating and computing with no information about the equation system, there are, however, general solvers that are quite robust at solving many types of problems though may not be the fastest to facilitate the specific problem one may be working on at the moment.

A popular class of iterative solvers for linear systems includes Krylov subspace methods of which GMRES appears to be the most general and robust (38) (39). Its robustness makes it a great choice for a general solver especially when a researcher has no specific information about the equation system she is asked to solve. GMRES minimizes residuals over successively larger subspaces in an effort to find a solution.

Still, in recent years, the Multigrid method has shown better optimal performance than GMRES for the same problems, motivating many to use this method in place of GMRES. The Multigrid method minimizes all frequency components of the residual equally providing an advantage over conventional solvers that tend to minimize high-frequency components of the residual over low-frequency components of the residual. Operating on different scales, the Multigrid method completes in a mesh-independent number of iterations (40) (41).

CFD experts also employ methods to simplify the ability of a solver to solve an algebraic equation system by pre-conditioning the matrix. The point of preconditioning is to transform the matrix closer to the identity matrix which is the easiest matrix to invert. Though we rarely invert matrices for such large problems, matrices that are invertible or closer to being invertible are also easier to solve iteratively.

Preconditioning is an art, involving the correct choice of preconditioner according to the type of linear system (just as in choosing the correct solver or iterative method). Examples include lumped preconditioners, diagonal preconditioners, incomplete LU, Conjugate-Gradient, Cholesky or block preconditioners such as block LU or Schwarz (38) (42) (43). The

best preconditioners are those that take advantage of the natural structure of the algebraic system in order to facilitate the transformation of A to a matrix closer to the identity matrix, thereby facilitating the solving of the system.
The condition number roughly measures at what rate the solution x will change with respect to a change in b. Low conditions numbers imply more ease in inverting and solving the system. Characteristics such as condition number and even definite-ness of systems become important in choosing a preconditioner.

3.5 Libraries

Many CFD programs require thousands of lines of code for one specific simulation. To avoid writing a different program for every application, researchers often write general programs and software packages that can be used for a variety of situations. This generality greatly increases the lines of code even more.
To help, there are many CFD and computational solid dynamics (CSD) libraries and applied mathematics libraries for the solving of such differential equations. They include code libraries such as OFELI (44), GETFEM (45), OOFEM (46), deal.ii (47), fdtl (48), RSL (49), MOUSE (50), OpenFOAM (51), etc. Sometimes a CFD engineer may use a package like ANSYS Fluent (52) to do a simulation, and sometimes a CFD researcher may find that the software does not give him the freedom to do what he would like to do (usually for very specific research applications). In these cases, he can write his own code or piece code together using these libraries. The most general Finite Element Library is deal.ii which is written in C++. It contains great support but currently does not support triangles or tetrahedral.

3.6 Moving problems

Moving problems are another class of CFD challenges. One can either use a body-fitted mesh approach (an approach where the mesh fits around the solid body of interest), an overlapping mesh method, or a meshless method. Body-fitted mesh methods must actual move the mesh elements or cells. Overlapping methods have the option of moving the mesh but capture the movement of interest in the overlapping regions. Meshless methods avoid the need to move the mesh as moving the mesh has the ability to introduce error. Usually when a mesh is moving the aspect ratio of the mesh elements must be checked. If the ratio goes beyond some limit, the mesh must be reordered and remade, and the solution from the old mesh must be projected to the new mesh in order to continue the simulation. Each of those steps has the ability to introduce error. Therefore it is best to limit mesh distortion or concentrate it in elements that have the ability to absorb the deformation without reaching the aspect ratio limit for the computation (7) (53).
There are also numerous methods that avoid meshes. Due to the problems and errors that poor refinement and mesh distortion introduce, some people avoid mesh methods altogether. Some meshless methods (54) include spectral methods (55), the vortex method (meshless method for turbulent flows), and particle dynamics methods (56).

4. Post-processing

This is the final stage when the computations are complete. This involves taking the output files full of velocity, pressure, density, and similar. information at each node for each time step and displaying the information visually, hopefully with color. Many CFD engineers also animate the results so as to better show what happens in time or to follow a fluid particle or a streamline.

4.1 Engineering

Up to this point, all of the work we have spoken about deals with the computer science/electrical engineering part or the computational and applied mathematics part of CFD. The mechanical and chemical engineering part of CFD comes in the post-processing and analysis.

The interpretation of the visualized results is also another point at which error can be introduced. CFD scientists and engineers must therefore take care in not overextending the analysis and simultaneously not missing important implications or conclusions that can be drawn from it. Engineers also make sure to corroborate the results with physical experiments and theoretical analysis. Additionally, the choice of problem and the motivation can come from the engineering side of CFD at the very beginning before any pre-processing work. Finally, the engineering analysis of the phenomenon directs the feedback loop of the research work. At this point the CFD worker must decide what values to change, whether it should be run again, what parameters should be maintained at current values, etc. He must decide which parameters' effect should be tested. He must determine if the results make sense and how to properly communicate those results to other engineers, scientists, scientists outside of the field, lay persons, and policy makers.

CFD work has taken exciting directions and comes to bear in many ways in society. When the fluid is water, CFD workers study hydrodynamics. When the fluid is air, they study aerodynamics. When it is air systems, they study meteorology and climate change. When it is the expanding universe, astronomy and cosmology; blood, medicine; oil and fossil flows, geology and petroleum engineering. CFD workers work with zoologists studying the mechanics of bird flight, paleontologists studying fossils, and chemists studying mixing rates of various gases in chemical reactions and in cycles like the nitrogen cycle.

Because of the importance and presence of fluids everywhere, the interaction of fluids with structures throughout the real world, and the application of mechanics and chemical engineering everywhere, CFD remains important. Moreover, the computational tools CFD engineers use can be utilized to solve and help other computational fields. Since the advent and continual innovation of the computer, all scientific fields have become computational—computational biology, computational chemistry, computational physics, etc. The same computational tools used in CFD can often be applied in other areas (predicting the weather involves a linear algebraic system for instance). This allows CFD engineers and scientists to move in and out of fields and bring their engineering analytical skills and critical reasoning to bear in other situations. CFD engineers and scientists are even used to make video games and animations look more physically realistic instead of simply artistic (57).

5. Closing

There are a host of other interesting areas in CFD such as two-phase or multi-phase flow (58), vorticity confinement techniques (similar to shock capturing methods) (59), probability density function methods (60), and fluid-structure interaction (FSI) (61). Other chapters in this book address those, so we will not talk specifically about them here. It is simply important to understand that as computing capability increases as computer scientists increase the clock speed of the microchip all the time, our ability to solve real-life problems increases. Problems involving two liquids or two phases occur all the time, and there are many instances of fluids interacting with deforming solids—in fact that is the general reality. So combining CFD with computational solid dynamics is an important partnership that lends itself well to solving the major challenges facing us in the 21st century.

6. References

[1] Phing, Anselm Ho Yen. *Simulating Combustion Flow in a Rocket Chamber.* Lulea: Lulea University of Technology, 2008.

[2] *A Study of Sharkskin and its Drag Reducing Mechanism.* Freidmann, Elfriede, Portl, Julia and Richter, Thomas. Berlin: Springer-Verlag, 2010.

[3] *Three-Dimensional Aerodynamic Simulations of Jumping Paratroopers and Falling Cargo Payloads.* Udoewa, Victor. 5, Reston, Virginia: AIAA, 2009, Vol. 46. 0021-8669.

[4] *Blood Flow in Arteries.* Ku, David N. 1, Palo Alto, CA: Annual Reviews, 1997, Vol. 29.

[5] *Outflow boundary conditions for three-dimensional finite element modeling of blood flow and pressure in arteries.* Vignon-Clementel, Irene E., et al., et al. 29-32, Stanford : Elsevier, 2006, Vol. 195.

[6] *Numerical investigation of transitional and weak turbulent flow past a sphere.* Tomboulides, Ananias G. and Orszag, Steven A. Cambridge: Cambridge University Press, 2000, Vol. 416.

[7] *Mesh Generation and Update Techniques for 3D Aerodynamic Simulations.* Udoewa, Victor. 7, s.l.: Wiley, 2009, Vol. 29.

[8] *Slip formulation for numerical simulations of jumping paratroopers.* Udoewa, Victor. 7, Johannesburg: Academic Journals, 2009, Vol. 2.

[9] *Models and Finite Element Techniques for Blood Flow Simulation.* Behr, Marek. s.l.: International Journal for Computational Fluid Dynamics, 2006, Vol. 20.

[10] Sawley, Mark L. Numerical Simulation of the Flow around a Formula 1 Racing Car. *EPFL Supercomputing Review.* [Online] November 1, 1997. [Cited: July 13, 2011.] http://ditwww.epfl.ch/SIC/SA/publications/SCR97/scr9-page11.html.

[11] *Computational study of flow around a simplified car body.* Guilmineau, Emmanuel. 6-7, Ottowa, Canada: Elevier, 2008, Vol. 96.

[12] Top 500. Top 500 Supercomputing Sites. *Top500.org.* [Online] Top 500, June 1, 2011. [Cited: July 13, 2011.] www.top500.org/lists/2011/06.

[13] Salisbury, David F. News Release. *Stanford News Service.* [Online] November 19, 1997. [Cited: July 13, 2011.] http://news.stanford.edu/pr/97/971119supercomp.html.

[14] MESHPART a Matlab Mesh Partitioning and Graph Separator Toolbox. *Cerfacs.* [Online] Cerfacs, February 8, 2002. [Cited: July 13, 2011.] http://www.cerfacs.fr/algor/Softs/MESHPART/.

[15] Karypis, George and Kumar, Vipin. *METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0.* Minneapolis: s.n., 1995.

[16] Karypis, George, Schloegel, Kirk and Kumar, Vipin. *PARMETIS - Parallel Graph Partitioning and Sparse Matrix Ordering.* Minneapolis: University of Minnesota, 2003.

[17] *Applying Parmetis To Structured Remeshing For Industrial CFD Applications.* Laflamme, S., et al., et al. 1, s.l.: Spring: International Journal of High Performance Computing Applications, 2003, Vol. 17.

[18] OpenMP. OpenMP News. *OpenMP.* [Online] OpenMP, July 9, 2011. [Cited: July 13, 2011.] http://openmp.org/wp/.

[19] Pacheco, Peter S. *Parallel Programming with MPI.* San Francisco: Morgan Kaufmann Publishers, Inc., 1997.

[20] *Manticore: A heterogeneous parallel language.* Fluet, Matthew, et al. Nice, France: DAMP 2007, ACM 2007, 2007.

[21] NESL: A Parallel Programming Language. *Computer Science at Carnegie Mellon University.* [Online] Scandal Lab, August 1, 2005. [Cited: July 13, 2011.] http://www.cs.cmu.edu/~scandal/nesl.html.

[22] Kaw, Autar and Kalu, E. Eric. *Numerical Methods with Applications.* s.l.: Lulu Self-published, 2008.

[23] Oliver, Rubenkonig. *The Finite Difference Method (FDM) - An introduction.* Freiburg: Albert Ludwigs University of Freiburg, 2006.

[24] Leveque, Randall. *Finite Volume Methods for Hyperbolic Problems.* Cambridge: Cambridge University Press, 2002.

[25] Huebner, K. H., Thornton, E. A. and Byron, T. D. *The Finite Element Method for Engineers.* s.l.: Wiley Interscience, 1995.

[26] Ang, W. T. *A Beginner's Course in Boundary Element Methods.* s.l.: Universal Publishers, 2007. 978-1581129748.

[27] Beer, Gernot, Smith, Ian and Duenser, Christian. *The Boundary Element Method with Programming: For Engineers and Scientists.* s.l.: Springer, 2008. ISBN 978-3211715741.

[28] *The immersed boundary method.* Peskin, C. S. s.l.: Acta Numerica, 2002, Vol. 11.

[29] Wesseling, P. *Principles of Computational Fluid Dynamics.* s.l.: Springer-Verlag, 2001.

[30] *Uniformly High Order Accurate Essentially Non-Oscillatory Schemes III.* Harten, A., et al., et al. s.l.: Journal of Computational Physics, 1987, Vol. 71.

[31] *The Piecewise parabolic Method (PPM) for Gasdynamical Simulations.* Colella, P. and Woodward, P. s.l.: Journal of Computational Physics, 1984, Vol. 54.

[32] Pope, S. B. *Turbulent Flows.* Cambridge : Cambridge University Press, 2000. ISBN 978-0521598866.

[33] Garnier, E., Adams, N. and Sagaut, P. *Large eddy simulation for compressible flows.* s.l.: Springer, 2009. 978-90-481-2818-1.

[34] *The Numerical Computation of Turbulent Flows.* Launder, B. E. and Spalding, D. B. 2, s.l.: Computer Methods in Applied Mechanics and Engineering, 1974, Vol. 3.

[35] Wilcox, David C. *Turbulence Modeling for CFD (3 ed.).* s.l.: DCW Industries, Inc., 2006. 978-1928729082.

[36] *Comments on the feasibility of LES for wing and on a hybrid RANS/LES approach.* Spalart, P. R. Arlington, TX: 1st ASOSR CONERFENCE on DNS/LES, 1997.

[37] *Coherent Vortex Simulation (CVS), A Semi-Deterministic Turbulence Model Using Wavelets.* Farge, Marie and Schneider, Kai. 4, s.l.: Flow Turbulence and Combustion, 2001, Vol. 66.

[38] Saad, Y. *Iterative Methods for Sparse Linear Systems, 2nd edition.* s.l.: SIAM, 2003. ISBN 978-0-89871-534-7.

[39] *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems.* Saad, Y. and Schultz, M. H. s.l.: SIAM Journal of Scientific and Statistical Computing, 1986, Vol. 7.

[40] Wienands, Roman and Joppich, Wolfgang. *Practical Fourier analysis for multigrid methods.* s.l.: CRC Press, 2005. 1584884924.

[41] Trottenberg, U., Oosterlee, C. W. and Schüller, A. *Multigrid.* s.l.: Academic Press, 2001. ISBN 012701070X.

[42] Trefethen, Lloyd N. and Demmel, James W. *Numerical Linear Algebra.* s.l.: SIAM, 1997. 0898713617.

[43] Golub, Gene H. and Van Loan, Charles F. *Matrix Computations 3rd Edition.* s.l.: John Hopkins University Press, 1996. 0801854148.

[44] OFELI. OFELI An Object Oriented Finite Element Library. *OFELI.* [Online] OFELI, June 1, 2011. [Cited: July 13, 2011.] http://www.ofeli.net/.

[45] Renard, Yves; Pommier, Julien. GETFEM++. *GNA.* [Online] GETFEM, June 1, 2010. [Cited: July 13, 2011.] http://download.gna.org/getfem/html/homepage/.

[46] OOFEM. OOFEM - free object oriented finite element solver. *OOFEM.* [Online] OOFEM, March 29, 2011. [Cited: July 13, 2011.] http://www.oofem.org/en/oofem.html.

[47] Dealii. Dealii Homepage. *Dealii.* [Online] Dealii, January 9, 2011. [Cited: July 13, 2011.] http://www.dealii.org/.

[48] Google. FDTL - Finite Difference Template Library. *FDTL.* [Online] Google, June 2, 2011. [Cited: July 13, 2011.] http://code.google.com/p/fdtl/.

[49] Michalakes, John. A Runtime System Library for Parallel Finite Difference Models with Nesting. *Argonne National Laboratory.* [Online] May 18, 1995. [Cited: July 13, 2011.] http://www.mcs.anl.gov/~michalak/RSL/.

[50] MOUSE team. MOUSE. *USASK.* [Online] University of Duisburg, January 7, 2010. [Cited: July 13, 2011.] http://homepage.usask.ca/~ijm451/finite/fe_resources/node563.html.

[51] OpenFOAM. OpenFOAM - The Open Source CFD Toolbox. *OpenFOAM.* [Online] OpenFOAM, June 2, 2011. [Cited: July 13, 2011.] http://www.openfoam.com/.

[52] Ansys. Features of Ansys Fluent. *Ansys.* [Online] Ansys, June 3, 2011. [Cited: July 13, 2011.] http://www.ansys.com/Products/Simulation+Technology/Fluid+Dynamics/ANSYS+FLUENT/Features.

[53] Huang, Weizhang and Russell, Robert D. *Adaptive Moving Mesh Methods.* s.l.: Springer, 2011. 978-1-4419-7915-5.

[54] Liu, G. R. *Mesh Free Methods, 2nd ed.* s.l.: CRC Press, 2009. 978-1-4200-8209-9.

[55] Hesthaven, J., Gottlieb, S. and Gottlieb, D. *Spectral methods for time-dependent problems.* Cambridge: Cambridge University Press, 2007.

[56] Belytschko, T. and Chen, J.S. *Meshfree and particle methods.* s.l.: John Wiley and Sons, Ltd., 2007. 0-470-84800-6.

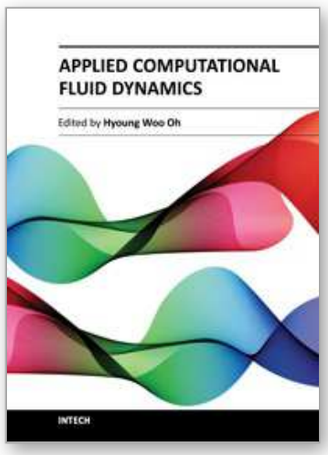
[57] Gourlay, Michael J. "*Fluid Simulation for Video Games*". s.l.: Intel Software Network, 2009.

[58] *Two phase flow in complex systems.* Levy, Salomon. s.l.: Wiley, 1999.

[59] *Numerical Simulation of Vortical Flows Using Vorticity Confinement Coupled with Unstructured Adaptive Grid Refinement.* Murayama, M. and Kato, T. 1, s.l.: Computational Fluid Dynamics Journal, 2001, Vol. 10.

[60] Fox, Rodney. *Computational methods for turbulent reacting flows.* Cambridge: Cambridge University Press, 2003. 978-0-521-65049-6.

[61] *FOIST: Fluid–object interaction subcomputation technique.* Udoewa, V. 9, s.l.: Wiley: Communications in Numerical Methods in Engineering, 2009, Vol. 25.



Applied Computational Fluid Dynamics

Edited by Prof. Hyoung Woo Oh

ISBN 978-953-51-0271-7
Hard cover, 344 pages
Publisher InTech
Published online 14, March, 2012
Published in print edition March, 2012

This book is served as a reference text to meet the needs of advanced scientists and research engineers who seek for their own computational fluid dynamics (CFD) skills to solve a variety of fluid flow problems. Key Features: - Flow Modeling in Sedimentation Tank, - Greenhouse Environment, - Hypersonic Aerodynamics, - Cooling Systems Design, - Photochemical Reaction Engineering, - Atmospheric Reentry Problem, - Fluid-Structure Interaction (FSI), - Atomization, - Hydraulic Component Design, - Air Conditioning System, - Industrial Applications of CFD

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen