Proceedings of COBEM 2003 17th International Congress of Mechanical Engineering
COBEM2003 - 1736 Copyright © 2003 by ABCM November 10-14, 2003, São Paulo, SP

A COMPUTATIONAL FRAMEWORK FOR SPEEDING-UP THE DEVELOPMENT OF RESEARCH AND EDUCATIONAL CFD CODES.

Romeu A. Pieritz

Rafael Mendes

Rodrigo F. A. F. da Silva

Clovis R. Maliska

Federal University of Santa Catarina, Mechanical Engineering Department
SINMEC – Computational Fluid Dynamics Laboratory
88040-900 - Florianópolis - SC – Brazil
http://www.sinmec.ufsc.br
rap@sinmec.ufsc.br

Abstract. *The development of Computational Fluid Dynamics codes involves knowledge in two main areas: numerical techniques and computer implementation of the algorithms. It is well recognized that the computer implementation of the algorithm is a key step in developing reusable and easy maintainable codes. The experience shows that when developing CFD codes for research purposes, the time consuming in re-writing pieces of the numerical part of the code, which are already well established, is very large. In addition, if the developer wants to have a reasonable visualization facility for scalar and vector fields, the time consumed in such parts of the code are enormous. If one recognizes that, in general, the researcher do not have strong background in using computer tools, this task is, most of times, unfeasible.The present papers address this crucial question, demonstrating a computer framework designed to simplify the task of developing CFD codes, offering a library with numerical objects and visualization package specially designed to be coupled with the code developed using the CFD-Sinflow Library. This library is an environment specialized in numerical methods for two dimensional fluid flow and heat transfer problems written in C++ standard programming using an object oriented approach. The library is constituted by four different basic modules (classes and pre-defined objects), where: i) Geometric Module (mesh edition and generation, geometry interpolation); ii) Mathematical Module (equations, functions, iterative and non-iterative solvers etc); iii) Numerical Module (the fully implicit time formulation, non-orthogonal boundary-fitted grids, segregate formulation in co-located grids with SIMPLE and SIMPLEC methods etc); and iv) System Module ("actions and events" code implementation methods; points, strings and system call instructions). The main library structure is introduced in three different tutorial cases. Examples of using the library integrated with the visualization package are shown, demonstrating the help that it provides to CFD developers in scientific and educational areas.*

Keywords. *Educational Software, research software, CFD development Library, Finite Volume, boundary-fitted grids.*

1. Introduction

Computational Fluid Dynamics-CFD is a powerful tool for scientific research, education and engineering. The expansion and success of its use depend on the user skills in the numerical area and the knowledge of the physical problem. The CFD code implementation consists in developing specific objects for grid generation, boundary conditions application, coupling of the discrete equations, solution of the linear systems, generation and control of the iterative procedures and post-processing, like scalar visualization, 3D graphics, iso-surfaces, velocity vectors etc. So, the CFD "solution application cycle" can be divided in two different stages: i) the physical problem analysis with the identification of the constituent equations, consequent proposition of the numerical solution, and the ii) computational coding with the algorithmic refinements and necessary tests cases. In this context, the numeric specialist or student spend up a lot of their time with the computational coding.

Two types of professionals grown out in the industry and research: the numeric specialist that has access to a great load software (commercial - minority) and the specialists that are forced to develop his own applications or adapt available codes (i.e.: on the net, most of the times deficient in flexibility and in the physical formulation).

In the universities and research centers (including the companies) is notorious the men/hour cost wasting in the code development that is not reusable and repetitive. In the current context, the necessary evolution to the implementation of new technologies is not generally reached in function of the existence of old codes written in "Fortran" language. Those codes are developed for specific platforms and for well defined problems originating a static base solution. The inflexibility of the language and a structured architecture not allow new solutions development without understanding all the code background. The required work to develop these new solutions can become a hard task to be accomplished by only one numeric specialist, imposing in most of the cases, the total code re-implementation.

New technologies developed by the scientific mathematicians and computer scientists allow starting the algorithms development from an established base problem. This technology is called "Object-Oriented Programming" (OOP), whose main characteristic is the same logical architecture to the human thought organization.

The "Object-Oriented Programming" allied with coding languages like "C" (from UNIX) created a new and improved standard version of this tool called "C++" – Barton (1994). The programming language growing allows today a numerical performance superior as Fortran in super-computers like Cray and Fujitsu - Veldhuizen (1997). This improvement is obtained through modern techniques and extensions introduced in the language by standard ANSI – ANSI (1998) and Ellis (1993) - and for powerful mathematical tools (solvers) – Siek (1998), Tisdale (1998) and Dongarra (1994).

The main objective of this article is to introduce the basic concepts and pedagogical characteristics of the educational CFD Sinflow Library (CSFL Library). This library was developed with the C++ programming language using the object-oriented approach in an open tool, in the format of a "code development library" specialized in CFD numerical methods. In that way, students and professionals can add to the library the mathematical solutions developed for several computational architectures with reduced programming effort. This article introduces the library main modules explaining the code concepts and the mathematical models. The second part of this article presents three study cases to show its pedagogical structure and evaluate the library numerical capabilities and results quality, where comparisons are made with benchmark and analytical solutions. At the end, some conclusions remarks are made.

2. Library Concepts

The CFD Sinflow Library is a classes and objects library to code physical models in Thermal and Fluid Dynamics field. In this way the library allows professionals, educators, and graduation/undergraduation students to access the numerical methods in a simplified way, reducing the computational knowledge necessary to its use. So, the library tries to satisfy these two different requirements: i) the numeric requirement to modelling and solve CFD problems – Maliska (1995); and ii) the computational requirement related to algorithms (software engineering, code implementation techniques, etc), where we have:

- The code reusability: abstraction and implementation - Veldhuizen (1998) - to allow the constant development through the new solution methodologies incorporation without paining the user with constant modifications in the architecture and consequently code re-learning and re-edition;
- Portability between multiple platforms: allows the use of the same source code between different platforms giving the possibility to prototyping the algorithm in small load machines (PCs) and consequent use in supercomputers (CRAY, etc) - Barton (1994) and Pozo (1996);
- The numerical solution methodologies encapsuling: to implement in a transparent way to the user (without computational code knowledge) the pre-processing, processing and post-processing stages;
- Support for different Graphical User Interfaces (GUI): to facilitate the direct integration at the code level with post-processing routines (visualization, plotters and interactive animation generators).

2.1. Basic Framework

The common approach in CFD coding applications in educational and industrial environment is construct the logical algorithm structure around a static iteration looping core, like SIMPLE or SIMPLEC methodologies – Maliska (1995). All routines and functions are developed statically to respond the core characteristics (like physical and boundary conditions inputs) and the code becomes inflexible to add new functionalities at user level. At the end, the code understanding becomes hard to new users and only the author can modify the algorithm.

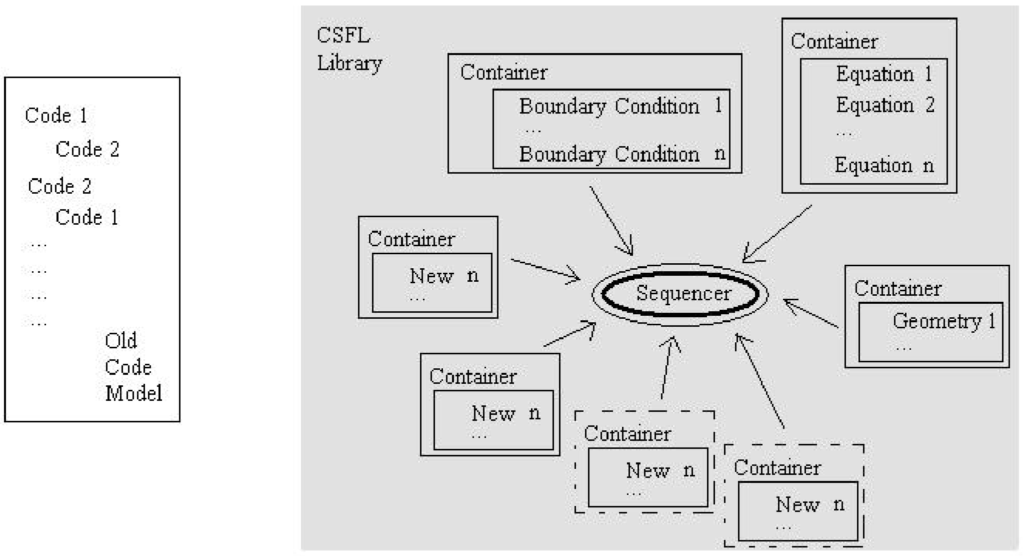


Figure 1. Old code programming architecture (left) and the CSFL Library flexible container set and sequencer (right).

The basic approach to solve this limitation and allow easy code understanding is grouping all different elements in a data set. The basic framework to do it is implemented by the CFD Sinflow Library by working with specialized

“Containers” in a simple architecture: all data and parameters will be grouped in a set of containers (boundary condition container, physical parameter container, equations container, etc) and a flexible specialized iterative “sequencer” will try to solve the CFD problem – Fig. (1). The student starts defining the grid simulation from the problem geometry. The boundary conditions are defined over the grid simulation and added to the boundary condition container. The same logical is used for the other numerical elements and its containers. At the end, the specialized numeric iterative engine (sequencer) receives these data to analyse, identify, perform and control the solution sequence.

These educational steps are reached using modern object-oriented programming techniques allowing easy code understanding. The student can code his numerical solution from a structured algorithm using pre-defined classes and objects, without deep knowledge about C++ and object-oriented programming (like the structured codes described in the tutorial section). This geometric, boundary conditions, physical parameters, mathematical and numeric library elements can be specialized by the user (class and objects hierarchical derivative approach) to solve new or special problems.

3. Library Modules

The current version of the CFD Sinflow Library is constituted of several modules grouping classes and pre-defined objects: i) Geometric Module; ii) Mathematical Module; iii) Numerical Module; and iv) System Module.

3.1. Geometric Module

The geometric module holds the necessary classes for geometry manipulation and its transformations (non-orthogonal mesh - Choi (1993) (1994) (1994)), allowing editing and controlling the simulation grid (base class IGrid and its hierarchy). The main operations are: generate, edit, merge, import and export in ASCII files (format ".sdf" and ".dat”). Figure 2 shows the elemental control volume P and its neighbours volumes in a structured grid framework used by CFD Sinflow Library.

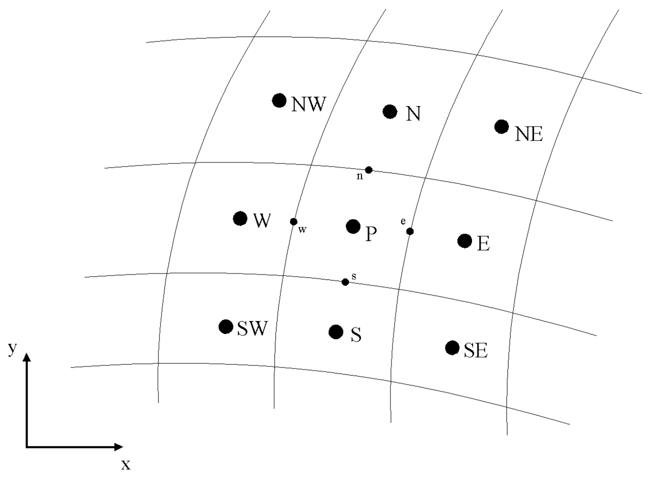


Figure 2. The numerical stencil used in the structured grid.

Others tools available in the CFD Sinflow Library for complex geometries discretization are the widespread curvilinear coordinates system generation routines. In this case the domain boundaries are just specified and the lines coordinates are obtained numerically. In the library (mathematical module) the curvilinear coordinates system generalized can be obtained through methods that use differential elliptic equations or through Lagrange and Hermite of first order interpolation - Maliska (1995).

3.2. Mathematical Module

The math module groups all the mathematical elements for the formulation and solution of the numerical problem, such as: solvers, equations (momentum in x and y directions, energy, mass, curvilinear system, etc), physical parameters, special cells and its boundary conditions.

In the Cartesian coordinate system (x, y), the differential partial equations that govern the elliptic convection-diffusion problems - Bejan (1995), Ferziger (1999) - used in the CFD Sinflow Library and grouped in the mathematical module are: the mass conservation, momentum and energy equations, given by:

$$\frac{\partial r}{\partial t}+\frac{\partial}{\partial x_j}\left(r u_j\right)=0 \qquad (1)$$

$$\frac{\partial}{\partial t}\left(r u_i\right)+\frac{\partial}{\partial x_j}\left(r u_j u_i\right)=-\frac{\partial P}{\partial x_i}+\frac{\partial}{\partial x_j}\left(m\frac{\partial u_i}{\partial x_j}\right)+S^{u_i} \qquad (2)$$

$$\frac{\partial}{\partial t}\left(r T\right)+\frac{\partial}{\partial x_j}\left(r u_j T\right)=\frac{\partial}{\partial x_j}\left(\frac{k}{c_p}\frac{\partial T}{\partial x_j}\right)+S^{T} \qquad (3)$$

where u is the velocity component, ? is the density, k is the thermal conductivity, P is the pressure, cp is the specific heat at constant pressure, µ is the absolute viscosity, T is the temperature and S an appropriate source term.

Equations 1-3 can be written in a general form and write in a general curvilinear coordinate system, as shown in Eq. (4), allowing the numerical procedure to be done in the transformed plane (Maliska (1995), Thompson et al (1985)).

$$\frac{\partial}{\partial t}\left(r\frac{f}{J}\right)+\frac{\partial}{\partial x}\left(rUf\right)+\frac{\partial}{\partial h}\left(rVf\right)=\frac{\partial}{\partial x}\left(\Gamma^f Ja\frac{\partial f}{\partial x}-\Gamma^f Jb\frac{\partial f}{\partial h}\right)+\frac{\partial}{\partial h}\left(\Gamma^f Jg\frac{\partial f}{\partial h}-\Gamma^f Jb\frac{\partial f}{\partial x}\right)+P^f \qquad (4)$$

where

$$U = u\, y_h - v\, x_h \qquad (5)$$

$$V = v\, x_x - u\, y_x \qquad (6)$$

are the contravariant velocities components of the velocity vector and

$$a = \left(x_h^2 + y_h^2\right) \qquad (7)$$

$$g = \left(x_x^2 + y_x^2\right) \qquad (8)$$

$$b = \left(x_x\, x_h + y_x\, y_h\right) \qquad (9)$$

$$J = \left(x_x\, y_h - x_h\, y_x\right)^{-1} \qquad (10)$$

are the metrics and the Jacobian of the transformation, respectively. Table 1 shows the Γ^ϕ and P^ϕ parameters for the different conservation equations being solved.

Table 1. Γ^ϕ and P^ϕ for the different conservation equations

Conservation equation	ϕ	Γ^ϕ	P^ϕ
Global mass	1	0	0
Momentum in x axis	u	µ	$\frac{\partial P}{\partial x} y_h - \frac{\partial P}{\partial h} y_x$
Momentum in y axis	v	µ	$\frac{\partial P}{\partial h} x_x - \frac{\partial P}{\partial x} x_h + \frac{r}{J} g\, b\, \left(T - T_\infty\right)$
Energy	T	k / cp	0

To obtain the approximated equations, balances of the conserved properties are made or the divergence form of the conservation equations are integrated over the control volumes. The time discretization was done using a fully implicit formulation and the CDS, UDS and WUDS interpolation functions are provided to determinate the ϕ values and its derivatives at the interfaces of the control volumes. Boundary conditions are applied using the fictitious volumes concept. A non-staggered grid arrangement was applied with the SIMPLE and SIMPLEC methods for treating the pressure-velocity coupling, as described at Van Doormaal (1984).

The input parameters like boundary conditions and physical parameters are implemented in this module, where we have: i) boundary conditions (temperature, velocity, heat flux, outlet, inlet, convection, and symmetry – Marcondes (1999)) and ii) physical parameters (gravity, specific heat, density, viscosity, etc).

This module also implements classes to “direct” and “iterative” solvers, where we have: i) direct solvers like LU decomposition, Cholesky decomposition and for diagonal systems by band; and ii) iterative solvers like Jacobi, Gauss-Seidel, Conjugated Gradient and TDMA-iterative. The solver choice is an essential step because the direct solvers are much more precise but they need a large memory space allocation. The iterative solvers just work with the non-null elements of the matrix, saving a lot of memory in the computer, but they are subject to fail with unstable linear systems (ill-conditioned matrix).

Special “ISolidBlock” class makes the interaction between solid and fluid and works with complex geometries in the library. Its physical properties and its special characteristics (constant or variable temperature, null velocities in the faces, etc) are hold in an independent way of the common properties objects. This capability allows to setup solids inside of a flow (without the use of sophisticated methods like “multi-block” approach) with special thermal characteristics and to differentiate regions inside of a diffusive phenomenon (like to prescribe a temperature into the physical domain).

3.3. Numeric Module

The numerical module makes the sequence of the CFD iteration classes and objects. The main classes are derived of a "sequencer" (“ISequencer” base class), responsible for the implementation of the iterative looping solution.

The hierarchy of the iterative base sequencer class is organized according to the Fig. (3). The use of this hierarchy classes can be seen as: for the solution of the energy equation is enough to create an object of the “ISequencerEnergy” class, and for the solution of the momentum equations (u and v) and the mass conservation can be used an object of the “ISequencerSimple” or “ISequencerSimplec” classes. In other way, if the problem demands the solution of all equations the object should be of the “ISequencerSimpleEnergy” or “ISequencerSimplecEnergy” type.

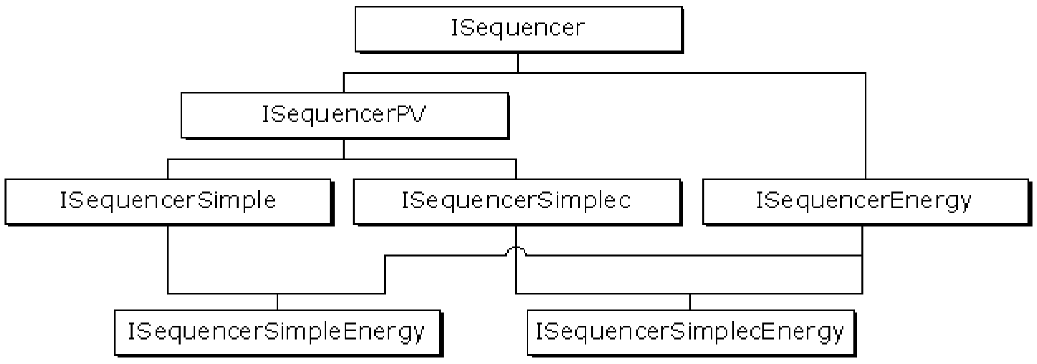


Figure 3. Hierarchical organization of the sequencer classes.

The iterative simulation process is started by calling the execution of the function “ISequencer::Run()”, where it construct, execute and monitor the solution steps. The base iterative cycle steps for some sequencers can be seen in the Fig. (4), where itGlobal, itT and itPV are the respective main iteration control for: maximum number of time iterations, number of temperature evaluation cycles and number of pressure-velocity iterative cycles.
The specialized print base class “ISequencerIO” allows the sequencer class (or its hierarchy) output on screen or in file several attributes like time, matrixes, vectors and simulation residues.

3.4. System Module

The system module has several base classes and objects to the library operation such as: vectors, matrices, points, etc. Other base classes in the system module are the “IString" class (manipulation of characters array for input/output) and the basic geometry manipulation like the coordinated points, lines and polygons. Also, the module groups the base classes for the arrays manipulation of objects and functions, as well as the "containers".

The “IObjectArrayOf” base class (multi-dimensional arrays and containers) allows the creation and the array sequential manipulation of any abstract data type. The main characteristic of these arrays is the automatic memory re-allocation at the moment that the maximum size of the previously allocated memory is reached, that means: once created, it automatically expands its size in function of the new elements addition (completely automatic and user independent). The “IArray” and “IArray2D” classes allow the vectors and matrices manipulation; these are the basic implementation of the vector and matrix classes.

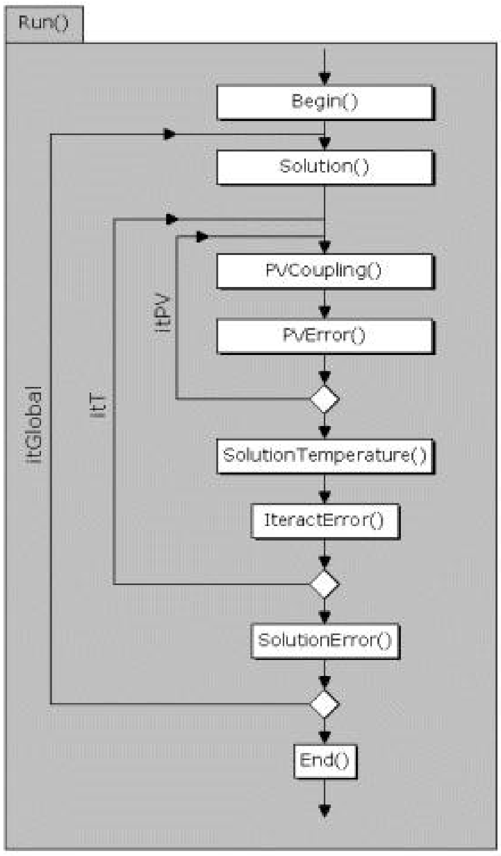


Figure 4. Iterative looping of the “ISequencerSimpleEnergy” and “ISequencerSimplecEnergy” (method “Run”).

The “IContainer” class and its child hierarchy represent the containers of the respective objects indicated in the classes names and they are set in the dynamic arrays, implementing object search functions (for IDs or binary search).

The iteration between custom user routines and the library is reached by the “actions and events”. The main objective of the “IAction" class is to allow the user "to glue" (or to amend) its own methods in routines already defined and implemented by the CSFL library. For example, the iteration between the sequencer class and the user routines (or other CFD Sinflow library objects) is accomplished by the actions during the algorithm execution .

```
…..
class IMonitor :
     public ISFLObject
{
public:
          IMonitor() : ISFLObject(), count( 0 ) {}
          void OutputSolverError( ISFLObject *_object )
          {
                    ISolverIteract *solver = NULL;
                    solver = dynamic_cast<ISolverIteract*>( _object );
                    if ( solver == NULL )  return;
                    count++;
                    ::cerr << count << ": " << solver->SolverError() << ::endl;
          }
private:
          int count;
};
…..
…..
IMonitor *output = new IMonitor();

ISolverIterTdma *solver = new ISolverIterTdma( lsys, 1.0e-05, 1000 );
Solver->ActionSolve = ExecuteMethod(&IMonitor::OutputSolverError, output);
…..
```

Figure 5. User custom class “IMonitor” and its usage to output in the screen the solver errors.

These actions, placed in strategic points inside the iterative looping code, allow user customization of the sequencer procedures. The user can attribute a method for each action that will be executed when the iteration reaches the call of the respective action in the sequencer runtime.

In Fig.(5) the code fragment shows the custom user class "IMonitor" used to output in the screen the solver error for each iteration in the solution of any CFD Sinflow library linear system. The "ISolver::ActionSolve" action (originated from some iterative solver of the library) calls the "IMonitor::OutputSolverError" function (the "lsys" object represents the linear system). By this way, for each solver iteration (in this example the TDMA solver) it will output in the screen the accountant's value (IMonitor::count) and the current error.its hierarchy) output on screen or in file several attributes like time, matrixes, vectors and simulation residues.

4. Educational Tutorials – benchmark comparison

4.1. Lid Driven Algorithm

The main classical CFD educational and benchmark code problem is the "lid driven": a square cavity with mobile cover, shown in the Fig. (6). The problem consists of a square cavity of dimension L, containing a fluid. The cavity is a thermal isolated system and it has a border that moves with constant velocity - Ghia (1982). The Reynolds number is the parameter that specific the problem where the mobile border velocity, the cavity dimension and the kinetic viscosity determine the fluid movement.

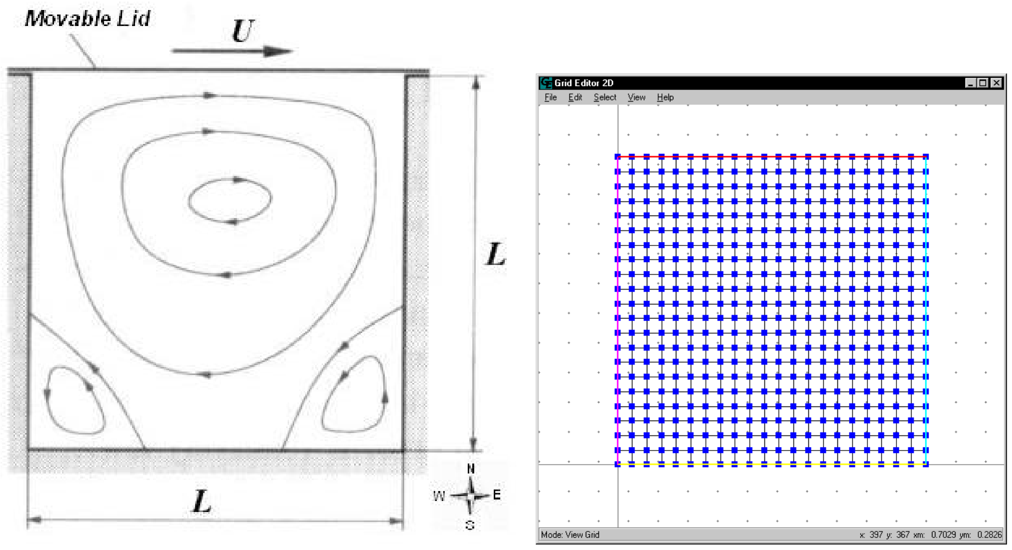

Movable Lid
U
L
L
N
W
E
S
Grid Editor 2D
File Edit Select View Help
Mode: View Grid
x: 397 y: 367 xm: 0.7029 ym: 0.2826

Figure 6. Lid driven geometry and parameters scheme (left) and the simulation grid (right).

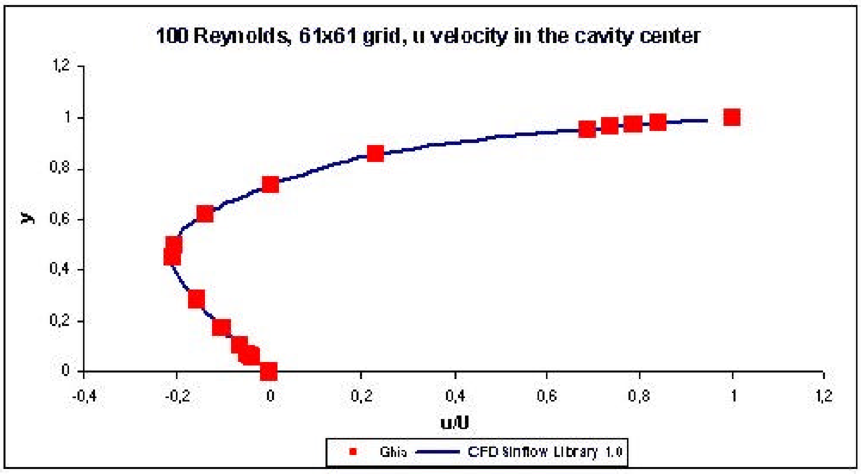
The Fig. (7) presents the code that allows the solution of the lid driven problem. The student starts coding the grid simulation object (IGridCartesian class instantiation – step 1 – Geometric Module). The user can define the boundary conditions over this simulation grid and storing these data in the boundary condition container (step 2 – Numeric Module). The same is done for physical parameters and equations (storing in its respective containers – steps 3 and 4 – Mathematical Module). At the end, the "sequencer" object receives these containers and tries to construct (step 5 – Numeric Module) and execute the iterative solution loop (searching the data and parameters in this container data base – step 6). The intermediary results are stored in file or printed in the screen by specialized objects ("ISequencerIO" class instantiation – step 5). The user can change this solution looping without changing or knowing the code, only by connecting his code by the "action-event" approach (System Module).

The numerical parameters used to solve the problem are: dimension L = 1 m; $u_o = v_o = 0$; U = 100 m/s; viscosity = 1 Pa.s; density = 1 kg/m^3; Cartesian grid 31x31 cells; CDS interpolation; mass and momentum equations in u and v; SIMPLEC; simulation step time = 0.001s; maximum iteration PV (pressure-velocity coupling) = 20; maximum global iteration in the step time = 100; maximum error solver = 10^{-6}; steady state error for mass = pressure = u = v = 10^{-7}, error evaluation points = 4 points in the top and bottom corners; solver TDMA with solver maximum iteration =100.

The main velocity profiles for u and v in the grid central cells line (horizontal and vertical) are show in the Fig. (8). The results are compared with those available in the benchmark - Ghia (1982). A good agreement between the benchmark solution and the library solution can be observed.

```
// Entry Point
void main()
{
// STEP 1: Geometry Definition ---------------------------------------------------
        int nx = 31; int ny = 31;
        IGridCartesian grid( nx, ny, 1.0/double(nx), 1.0/double(ny), 90 );

//STEP 2: Boundary Condition Definition ---------------------------------------
        IBCTPhi bcphi( &IFConstant( 100.0 ), "Velocity Const", bctU );

        // Boundary Condition
        IBCNorth bcnorth( &bcphi, &grid, IPoint( 0, ny -1 ), IPoint( nx -1, ny -1 ) );

        // Boundary Condition Container
        IContainerBCond contbc( "Velocity Boundary Condition", false );
        contbc.Add( & IBCNorth( &bcphi, &grid, IPoint( 0, ny -1 ), IPoint( nx -1, ny -1 ) )  );

// STEP 3: Physical Parameter Definition ----------------------------------------
        // Physical Parameter Container
        IContainerParameter contp("Container Parameters", false);
        contp.Add( &IParamPhysical( &IFieldScalar( "Rho", &grid, 1.0 ), grid.Metric(), ptRho)  );
        contp.Add( &IParamPhysical( &IFieldScalar( "Viscosity", &grid, 1.0 ), grid.Metric(), ptMi)  );
        contp.Add( &IParamInterpolCDS( "CDS - Interpolation" ) );

// STEP 4: Equation definition ------------------------------------------------------
        // Equation Container
        IContainerEquation conteq("Equation Container", false );
        conteq.Add( &IEquationMomentumU( "momentumU", &grid, &contbc, &contp) );
        conteq.Add( &IEquationMomentumV( "momentumV", &grid, &contbc, &contp ) );

        // Solver
        double ersolver = 1.0e-4;  int maxiterator = 100;
        ISolverIterTdma solver( NULL, ersolver, maxiterator, &conte );

// STEP 5: Pressure Velocity Coupling Definition -------------------------------
        double timeinit = 0.0; double timeend  = 1000.0; double timestep = 1.0e-3;

        // Simplec Sequencer Driver
        ISequencerSimplec simplec( "Sequencer Simplec", &conteq, &solver, timeinit, timeend, timestep );
        simplec.SetMaximumIteratorPV( 20 );

        // Output File and Screen Monitor Control Driver – Action Event
        ISequencerIO io( &simplec );

        simplec.ActionRequestIteractPV = ExecuteMethod( &ISequencerIO::OutputScreenMonitor, &io );
        simplec.ActionIteract = ExecuteMethod( &ISequencerIO::OutputFileIteract, &io );
        simplec.ActionSolutionError =ExecuteMethod( &ISequencerIO::OutputScreenSteadyStateError, &io );

// STEP 6: SIMULATION -------------------------------------------------------------
        simplec.Run();

        return (0);
}
```

Figure 7. CFD Sinflow Library basic framework to simulate the lid driven problem - Ghia (1992)



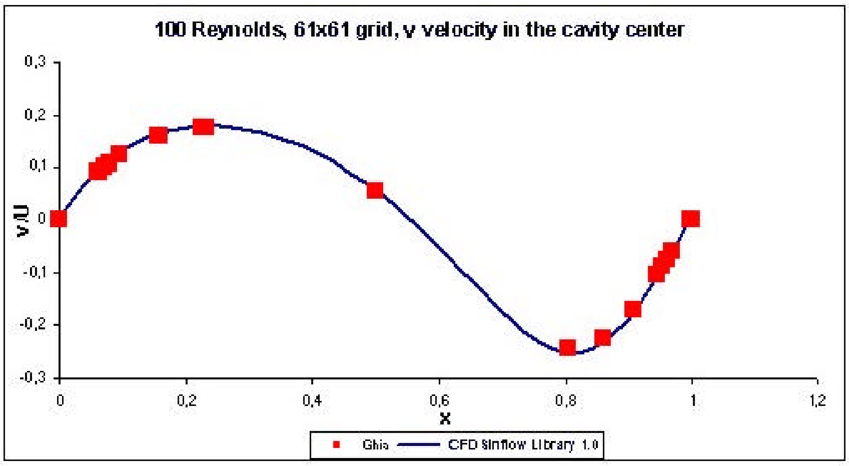


Figure 8. Comparison of the u velocity profile in the half-length vertical line (top), and comparison of the v velocity profile in the half-height horizontal line (bottom) for Re = 100 (U = cover velocity) with the benchmark – Ghia (1992).

4.2. Solid Block Formulation

The problem presented in the previous section is modified by adding the special cells (solid blocks) inside of a cavity with the double size - Patankar (1980) and Pieritz (2000). So, two conjugate square cavities can be constructed with the same dimensions and boundary conditions – Fig. (9). The numeric parameters are the same than the lid driven problem, with a grid simulation of 61x123 cells.

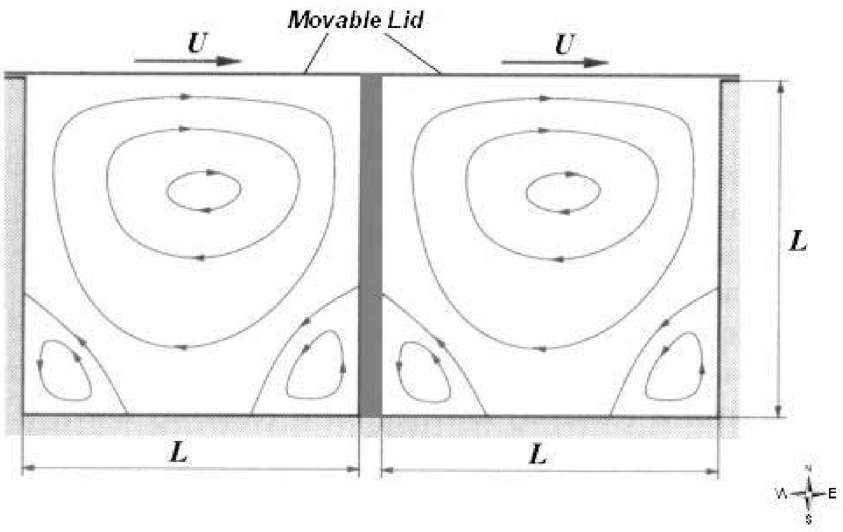


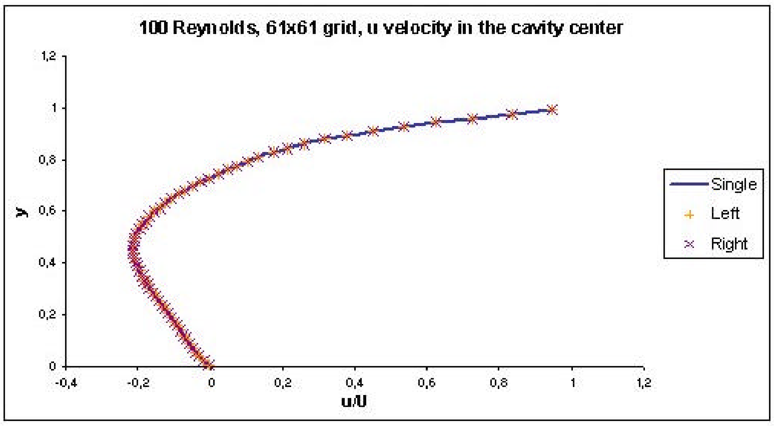
Figure 9. Two conjugate cavities for the Lid driven using the solid blocks formulation.

The code showed in Fig. (7) has the grid size changed in the “step 1” – code fragment in Fig. (10). The solid cells code description to modify the problem is applied in the “Step 3” - Fig. (10).

```
// Entry Point
void main()
{

// STEP 1: Geometry Definition -------------------------------------------------
        int nx = 63;
        int ny = 31;
        IGridCartesian grid( nx,ny,(2.0+1.0/31.0)/double(nx), 1.0/double(ny),90.0 );

        ………

// STEP 3: Physical Parameter Definition ---------------------------------------
        ……....

        //Solid Blocks
        ISolidBlock sblock( IPoint( 31, 0 ), IPoint( 31, ny-1 ) );
        ………
        contp.Add( &sblock  );
        ………
```

Figure 10. Changes in the algorithm of the Fig. (7) for Solid Cells addition.

The main velocity profiles for u and v in the grid simulation central cells line (horizontal and vertical) in both cavities are show in the Fig. (11). The results are compared with those available for the single lid driven cavity. A good agreement between the benchmark solution and the library solution can be observed.



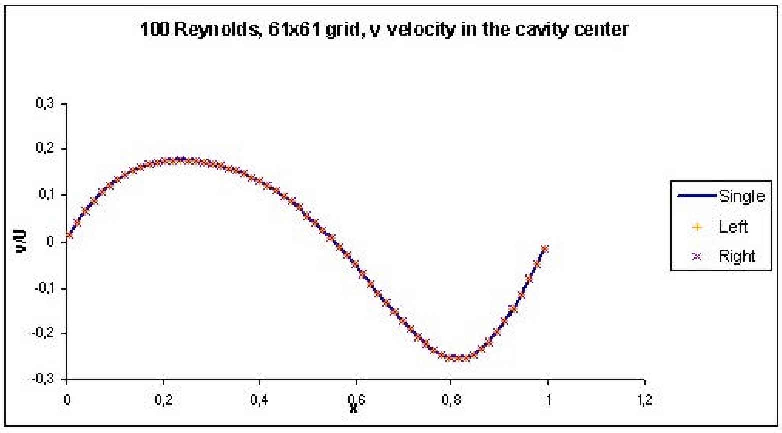


Figure 11. u velocity profiles in the half-length vertical line for the different models, on top, and on bottom the v velocity profiles in the half-height horizontal line for the different models (single and the conjugate cavities, Re = 100 where U is the cover velocity).

4.3. Mass Flow - Parallel plates Algorithm

The objective of this educational problem is to demonstrate the solution of a fluid flow problem with mass flux in/out in the physical domain using the CFD Sinflow Library. The problem-example, in Fig. (12), consists in a flow through parallel plane plates - Maliska (1995), without heat transfer through the walls (plates).

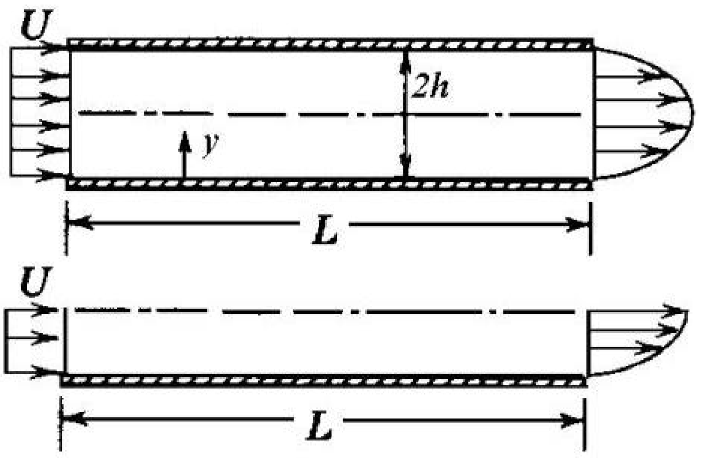


Figure 12. Flow among parallel plane plates, and the geometry for symmetry simulation.

The boundary conditions are: prescribed U velocity in the entrance (west) and outlet condition in the right side (east). With the action of the viscous forces a parabolic velocity profile is formed in the exit that can be checked with the analytic solution of this problem Eq. (11).

As the problem has a symmetry plan (in the direction x) it is possible to solve the same problem just simulating half of the domain. For this is enough to specify the symmetry boundary condition in the area where there is a symmetry plan. The symmetry condition offers a significant reduction in the computational simulations time and memory. This is due to the fact of reducing the calculation domain at least in half of the original domain.

The results obtained for the velocities in the "outlet" of the plates (flow completely developed) can be compared with the analytic solution Eq. (11), as shown in Fig. (14).

$$u = 1{,}5U\left(1 - \left(\frac{y - h}{h}\right)^2\right) \tag{11}$$

where U is the velocity in the plates entrance.

The numerical parameters used to solve the problem are: Reynolds = 50; $u_o = v_o = 0$; viscosity = 1 Pa.s; density = 1 kg/m^3; Cartesian mesh 30x39 cells; CDS interpolation; mass and momentum equations in u and v; SIMPLEC methodology; west boundary condition = U, north and south = wall, and east = outlet; simulation step time = 0.01s; maximum iteration PV (pressure-velocity coupling) = 20; maximum global iteration in the step time = 100; maximum error solver = 10^{-7}; steady state error pressure = u = v = 10^{-7}, steady state error for mass= 0.5; error evaluation points = line y = 0.5 and at the plates exit; solver Jacobi; solver maximum iteration = 500.

The symmetry simulation uses the same numerical parameters, where: symmetry in the north boundary condition (x direction) and grid simulation of 30x20 cells.

The u velocity profiles in the outlet obtained by the total and half of the domain are shown in the Fig. (14). The results are compared with those available for the analytic solution by Eq. (11). A good agreement between the analytic solution and the library solution can be observed.

```
// Entry Point
void main()
{
// STEP 1: Geometry Definition ---------------------------------------------------
       int nx = 30; int ny = 39;
       IGridCartesian grid( nx, ny, 3.0/double(nx), 1.0/double(ny), 90 );
//STEP 2: Boundary Condition Definition ---------------------------------------
       IBCTPhi bctphi( &IFConstant( 50.0 ), "Velocity Const", bctU );
       IBCTOutlet   bctoutlet( "Mass Outlet" );

       IBCWest bcwest( &bctphi, &grid, IPoint( 0, 0 ), IPoint( 0, ny-1 ) );
       IBCEast bceast( &bctoutlet, &grid, IPoint( nx-1, 0 ), IPoint( nx-1, ny-1 ) );

       // Boundary Condition Container
       IContainerBCond contbc( "Speed Boundary Condition", false );
       contbc.Add( &bceast); contbc.Add( &bcwest);
// STEP 3: Physical Parameter Definition ----------------------------------------
       // Fields
       IFieldScalar mi( "Viscosity", &grid, 1.0 );
       IFieldScalar rhoconst( "Rho", &grid, 1.0 );

       // Physycal Parameters
       IParamPhysical phmi ( &mi, grid.Metric(), ptMi );
       IParamPhysical phrho( &rhoconst, grid.Metric(), ptRho );

       // Numerical Interpolation
       IParamInterpolCDS picds( "CDS - Interpolation" );

       // Numerical Error Evaluation Domain
       IErrorBlock eb00( "ErrorBlock 00", IPoint( nx-1, 0 ), IPoint( nx-1, ny-1 ) );
       IErrorBlock eb01( "ErrorBlock 01", IPoint( 0, ny/2 ), IPoint( nx-1, ny/2 ) );

       // Physical Parameter Container
       IContainerParameter contp("Container Parameters", false);
       contp.Add( &phrho );  contp.Add( &phmi  ); contp.Add( &picds );
        contp.Add( &eb00  ); contp.Add( &eb01  );
// STEP 4: Equation definition -----------------------------------------------------
       // Initial Condition
       IFieldlVector initialVel( "initV", &grid, 20.0, 0.0 );

       // Equations
       IEquationMomentumU eqU( "momentumU", &grid, &contbc, &contp, NULL, &initialVel );
       IEquationMomentumV eqV( "momentumV", &grid, &contbc, &contp, NULL, &initialVel );

       // Equation Container
       IContainerEquation conteq("Equation Container", false );
       conteq.Add( &eqU ); conteq.Add( &eqV );

       //Solver Error Domain
       IContainerParameter conte( "Container Solver Error Domain Evaluation", false );
       conte.Add( &eb00  );conte.Add( &eb01  );
       // Solver
       double ersolver = 1.0E-4;  int  maxiterator = 200;
       ISolverIterJacobi solver( NULL, ersolver, maxiterator, &conte );
// STEP 5: Pressure Velocity Coupling Definition -------------------------------
       double timeinit = 0.0,  timeend  = 1000.0, timestep = 1.0E-2;

       // SimplecEnergy
       ISequencerSimplec simplec( "Sequencer Simplec", &conteq, &solver, timeinit, timeend, timestep );
// STEP 6: SIMULATION -------------------------------------------------------------
       simplec.Run();

       return (0);
}
```

Figure 13. Parallel plates algorithm using CFD Sinflow Library.

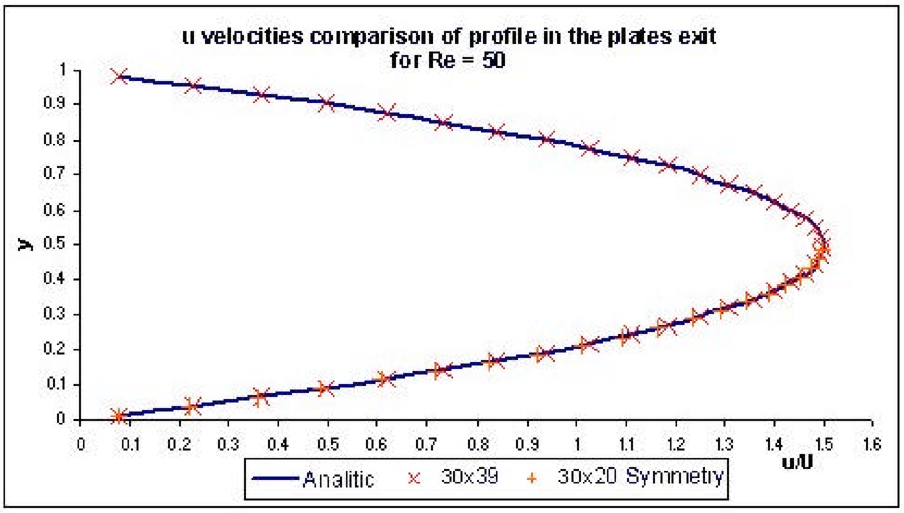


Figure 14. The velocity profile in the exit for Re = 50 for the analytic solution, half grid (with symmetry condition) and the entire geometry (U = entrance velocity).

5. Conclusions

This paper introduced the educational software development library - “CFD Sinflow Library" - specialized in numerical methods in CFD and Thermal Sciences. The library was developed in C++ language, allowing a larger abstraction and encapsulation of the classes and implemented objects. The object-oriented programming also facilitates the reusability of computational codes, allowing the addition of new methodologies and numerical techniques. It is constituted of four different modules to describe the different numerical solution elements.

This library is for research, educational and engineering purposes like an open and platform independent architecture. The tool allows the student: i) time reduction to the numeric solution development cycle; ii) reduce the computational knowledge; iii) the graduation and undergraduation students to access different numerical methods; and iv) its structure is evolutionary: new methodologies and techniques can be added.

The library educational characteristics are introduced in the tutorials, allowing the results comparisons by a numerical standard benchmark and an analytical solution showing an excellent agreement. Other modifications in the tutorial flow problem are made to illustrate the possibilities in the modelling of complex geometries by the solid cells approach. The implementation of these CFD algorithms into teaching modules in undergraduate and graduate programs provides a significant enhancement to the learning of heat transfer and fluid dynamics. Students working in their assignments can develop his own numerical problem and concentrate his efforts in the physical phenomena and numerical methodology understanding instead of in the code implementation.

6. Acknowledgement

The authors are grateful to Axel Dihlmann, Márcio Monteiro, Marcelo Souza, Gerson Bridi, Ewaldo Shubert Jr. and Carlos Donatti for their contribution to this work. The library, the main related documents, examples and tutorials can be downloaded for free from the worldwide web site: http://www.sinmec.ufsc.br/cfd.

7. References

ANSI – Americam National Standarts Institute, 1998, “Information Technology - Programming Languages - C++”, Document Number: ISO/IEC 14882.

Barton, J.J., Nackman, L.R., 1994, “Scientific and Engineering C++, an Introduction with Advanced Techniques”, Addison-Wesley.

Bejan, A., 1995, “Convection Heat Transfer”, John Wiley & Sons.

Choi, S.K., Nam, H.Y., Cho, M., 1993, “Use of the Momentum Interpolation Method for Numerical Solution of Incompressible Flows in Complex Geometries: Choosing Cell Face Velocities”, Numerical Heat Transfer, Part B, Vol. 23, pp. 21-41.

Choi, S.K., Nam, H.Y., Cho, M., 1994, “Use of Staggered and Nonstaggered grid arrangements for Incompressible Flow Calculations on Nonorthogonal Grids”, Numerical Heat Transfer, Part A., Vol. 25, pp. 193-204.

Choi, S.K., Nam, H.Y., Cho, M., 1994, “Systematic Comparision of Finite-Volume Calculation Methods with Staggered and Nonstaggered Grid Arrangements”. Numerical Heat Transfer, Part B., Vol. 25, pp. 205-221.

Dongarra, J.J., Pozo, R., Walker, D.W., 1994, “LAPAC++: A Design Overview of Object-Oriented Extensions for High Performance Linear Algebra”, Technical Report, Oak Ridge National Laboratory, Mathematical Sciences Section.

Ellis, M.A., Stroustrup, B., 1993, “C++ Manual de Referência Comentado – Documento Oficial do ANSI”, Campus.

Ferziger, J.H., Peric, M., 1999, “Computational Methods for Fluid Mechanics”, Springer.

Ghia, U., Ghia, K. N., Shin, C.T., 1982, “High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method”, Journal of Computational Physics, Vol. 48, pp. 387-411.

Maliska, C.R., 1995, “Transferência de Calor e Mecânica dos Fluidos Computacional”, LTC – Livros Técnicos e Científicos Editora.

Marcondes, F., Maliska, C.R., 1999, “Treatment of the Inlet Boundary Conditions in Natural-Convection Flows in Open-ended Channels”, Numerical Heat Transfer, Part B, Vol. 35.

Patankar, S. V., 1980, “Numerical Heat Transfear and Fluid Flow”, McGraw-Hill.

Pieritz, R. A., Kuhnen da Silva, A., Monteiro, M., 2000, “Fundamentos Teóricos da SinFlow Library", Relatório Técnico, SINMEC, Departamento de Engenharia Mecanica, UFSC, Brasil.

Pozo, R., Remington, K.A., 1996, “C++ Programing for Scientists – Lecture Notes”, Nist,

Siek, J. G., Lumsdaine, A.,1998, “The matrix Template Library: A Generic Programming Approach to High Performance Numerical Linear Algebra”, Technical Report, University of Notre Dame, Laboratory for Scientific Computing.

Tisdale, E.R.,1998, “The C++ Scalar, Vector, Matrix and Tensor Classes”, Technical Report.

Thompson, F.F., Warsi, Z.U.A. and Mastin, C.W., 1985, “Numerical Grid Generation – Foundations and Applications”, Elsevier Science Publishing Corporation.

Veldhuizen, T., Jernigan, 1997, “Will C++ be faster than Fortran”, Technical Report, University of Waterloo, Department of Systems Design Engineering.

Van Doormaal, J.P., Raithby, G.D., 1984, “Enhancements of the Simple Method for Predicting Incompressible Fluid Flows", Numerical Heat Transfer, Vol. 7, pp 147-163.

Veldhuizen, T., 1998, “Techniques for Scientific C++”, Technical Report, University of Waterloo, Department of Systems Design Engineering.