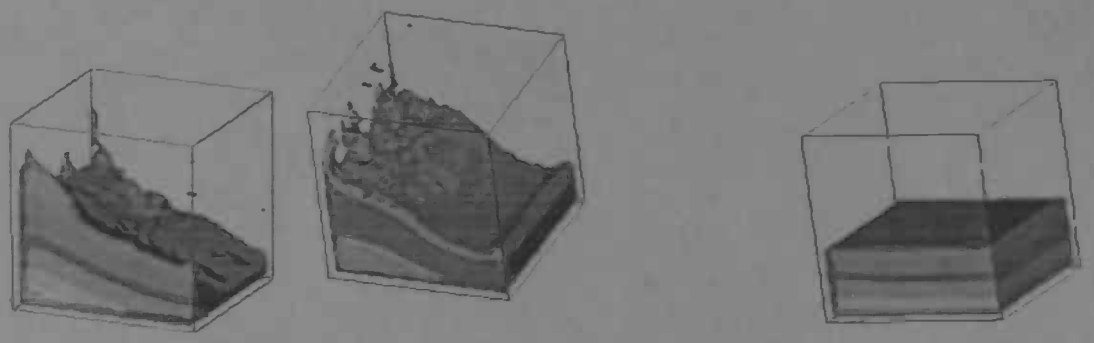
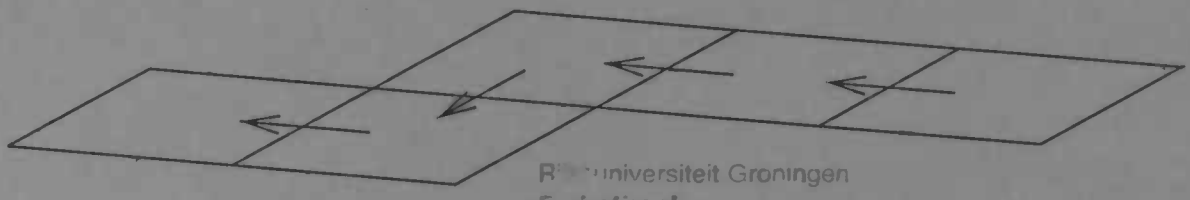
Biebs

Free Surface Flow in 3D Complex Geometries using Enhanced Boundary Treatment

Erwin Loots





Department of
Mathematics

RuG

Master's thesis

Free Surface Flow in 3D Complex Geometries using Enhanced Boundary Treatment

Erwin Loots

University of Groningen
Department of Mathematics
P.O. Box 800
9700 AV Groningen

June 1998

Contents

Chapter 1

Introduction

Few branches of mathematics enable more real-life applications than the field of fluid dynamics.
Although the theoretical basis is rather ancient (the Navier-Stokes equations, which fully describe the evolution of fluid in time and place are already known for 150 years), it is the computer power of our lifetime which enables us to solve the analytical equations numerically, and, consequently, simulate nearly all possible fluid flows. The applications are numerous; we only need to refer to aircraft design and weather prediction.

A lot of the early CFD codes (and, in fact, of the present ones, too) have in common that they use body-fitted coordinates. Although this has advantages in the field of boundary treatment and refinements, the construction of new grids for each separate problem is often more costly than the simulation itself.
The other approach -our approach- is to use a simple, rectilinear (Cartesian) grid. This enables the use of arbitrary geometries, created at very limited costs. The backside, of course, are the measures to be made at the boundaries. A first approach is to describe these boundaries with a staircase geometry, that is, each cell of the grid is either inside or outside the geometry. Although this is a reasonable approach using a high resolution, features such as fluxes and contact angles with the wall are less well obtained in this way. Moreover, especially in 3D, the available resolution is limited.

The R*u*G developed its 3D CFD code, called *ComFlo*, in 1995. Starting as a fully 3D-program, free surface flow (liquid sloshing) was added as early as medio 1996, see [6]. Meanwhile, the standard version was improved using higher-order discretization schemes and features for in- and outflow ([2]). Shortly thereafter, the author participated in further improvements, including large-scale extensions in pre- and postprocessing ([7]). In the autumn of 1997, after a short consolidation process, plans for the near future were determined. This report describes the execution of the first goals: a better boundary treatment (especially in combination with free surfaces), together with the creation of uniform postprocessing tools (supported by MATLAB and AVS).
At this point, in the summer of 1998, *ComFlo* is able to deal with a wide variety of flows, while it has already been used for several industrial applications.

In the following chapters, the theoretical model is explained (Chapter 2). Then the numerical model, as implemented in *ComFlo*, is partly explained with aspects including apertures,

labeling and the pressure Poisson equation (which were already discussed in previous reports) as well as the new and changed features like the boundary treatment and body forces (Chapter 3).
This is followed by some results consisting of test cases (mostly comparisons with earlier situations) and demonstrations of the present capabilities of *ComFlo* (Chapter 4).
Also, in Appendix A, the structure of the main program (i.e. *ComFlo*) is explained at subroutine level. The input file, in combination with postprocessing in Matlab, is described in Appendix B. Finally, Appendix C is dedicated to a short description of AVS and the functions of the *ComFlo* module, thus explaining an important part of the postprocessing. These appendices can help new users of *ComFlo* to get started.

Chapter 2

Mathematical model

2.1 The Navier-Stokes equations

We consider a flow domain of arbitrary form Ω . Part of the domain, Ω_f, consists of fluid: $\Omega_f \subset \Omega$. The fluid is assumed to be viscous and incompressible. For this system, restricted to Ω_f, the unsteady, incompressible Navier-Stokes equations hold (note that the pressure is scaled by the density):

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{2.1}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = -\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) + F_x + f_x \tag{2.2}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} = -\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}\right) + F_y + f_y \tag{2.3}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} = -\frac{\partial p}{\partial z} + \nu\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}\right) + F_z + f_z \tag{2.4}$$

The symbols mean the following:
t is the time; u, v, w are the velocity components in $x-$, $y-$ and $z-$ direction, respectively;
p is the pressure, scaled by density ρ;
ν is the kinematic viscosity, which is equal to $\frac{\mu}{\rho}$, where μ is the dynamic viscosity;
$\boldsymbol{F} = (F_x, F_y, F_z)^T$ is an external body force like gravity;
$\boldsymbol{f} = (f_x, f_y, f_z)^T$ is a virtual body force, working on the fluid, because of a motion of the geometry (like translation or rotation). In section 2.3 more details are explained.

Equation (2.1) expresses conservation of mass in each volume, and equations (2.2), (2.3) and (2.4) denote the conservation of momentum in $x-$, $y-$ and $z-$ direction, respectively. The equations above can also be described in vector form, by setting $u = (u, v, w)^T$ and using the divergence $(\nabla\cdot)$ and grad (∇) operators:

$$\nabla u = 0, \tag{2.5}$$

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = -\nabla p + \nu(\nabla \cdot \nabla)u + F + f \tag{2.6}$$

Here we can replace $(u \cdot \nabla)u$ by $\nabla \cdot (uu^T)$ since the velocity is divergence-free.

2.2 Boundary conditions

Next we describe the conditions at the various boundaries. These can be divided into three classes: Conditions for the solid boundary $\partial\Omega \cap \partial\Omega_f$, the free surface $\partial\Omega_f \backslash (\partial\Omega \cap \partial\Omega_f)$ (which is time-dependent); and, finally, conditions for inflow and outflow areas: $\partial\Omega_i, \partial\Omega_o$.

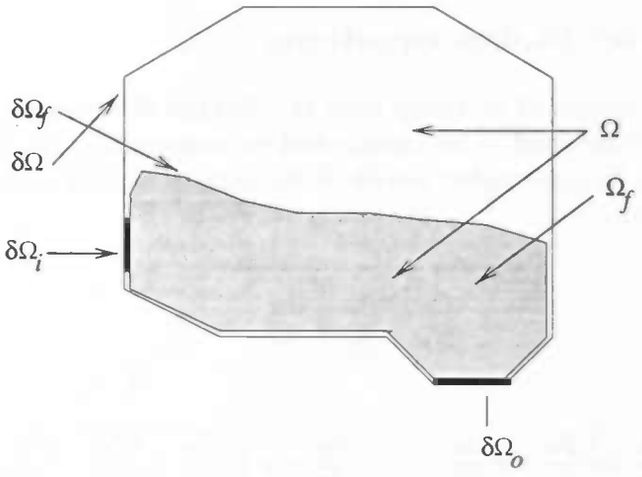


Figure 2.1: *Flow domain and special areas*

2.2.1 Solid boundary

Globally, a division between no-slip walls and free-slip can be made, where both have different conditions:

$$\text{no-slip:} \quad u = 0$$

$$\text{free-slip:} \quad u_n = 0 \quad \text{and} \quad \frac{\partial u_t}{\partial n} = 0$$

Here $u_n = u \cdot n$ is the normal velocity; $u_t = u \cdot t$ denotes the tangential velocity.

Thus, no-slip means that the normal and tangential velocity components are both zero: the fluid does not flow through the wall, while the fluid does not move in tangential directions either: it sticks to the wall. The latter condition does not hold at free-slip walls; here, the presence of the wall does not influence the velocity flowing along (tangential to) the wall.

2.2.2 Free surface

Since we do not solve the Navier-Stokes equations in the entire domain Ω (by using the different fluid characteristics, like density (see [9], for example)), the boundary of the fluid needs appropriate conditions for the pressure and the velocities.
These are:

$$-p + 2\mu \frac{\partial u_n}{\partial n} = -p_0 + 2\gamma H \tag{2.7}$$

$$\mu \left(\frac{\partial u_n}{\partial t} + \frac{\partial u_t}{\partial n} \right) = 0 \tag{2.8}$$

Here p_0 is the pressure outside the fluid (also called the atmospheric pressure). Note that, unlike for flows without free surfaces, the level of the pressure is now determined (by p_0). γ is the surface tension. $2H = \frac{1}{R_1} + \frac{1}{R_2}$ is the total curvature of the surface. This curvature is defined by the two-dimensional curvatures R_1 and R_2; each is defined in one out of two orthogonal planes through the normal of the surface.

2.2.3 In- and outflow

In an inflow region, fluid is pushed into Ω. The amount is controlled by the area of the inflow region and the velocity with which the fluid enters Ω. Therefore, the inflow condition is: $\boldsymbol{u} = \boldsymbol{u}_{in}$. Note that the direction of u_{in} can be outward; in that case, amounts of outflow are controlled.

In outflow regions, the following conditions are mostly used:
$\frac{\partial \boldsymbol{u}}{\partial n} = 0$ and $p = p_0$. Other conditions are possible, but the chosen ones turned out to be practical for our purposes.

2.3 Forces and movement

The coordinate system in which the geometry is contained (called O_g) is relative to an inertial coordinate system O_i. It is necessary, for several reasons to keep track of this relativity.
Assuming that at $t = 0$ the two systems are the same, the geometry system O_g is fully described with respect to O_i by, at first, a rotation vector Φ *or* a set of unit vectors $\boldsymbol{f_1}, \boldsymbol{f_2}, \boldsymbol{f_3}$; and, after that, a translation vector. Then the formula for the virtual body force is

$$\boldsymbol{f} = -\frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times (\boldsymbol{x} - \boldsymbol{x_0})) - \frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t} \times (\boldsymbol{x} - \boldsymbol{x_0}) - 2\boldsymbol{\omega} \times \boldsymbol{u} \tag{2.9}$$

Here $\boldsymbol{q}$ is the translational velocity of O_g (relative to O_i, of course), $\boldsymbol{\omega}$ is the rotation vector in O_g (which does not necessarily intersect the origin: $\boldsymbol{x_0}$ is the centre of rotation; note that this center is not unique since $\boldsymbol{x_0} + \alpha\boldsymbol{\omega}$ (α arbitrary), can also be taken for this centre.), $\boldsymbol{x}$ the position of a fluid particle in O_g and $\boldsymbol{u}$ its velocity (again in O_g).

The rotation vector $\boldsymbol{\omega}$, the rotation center and the translational velocity can be time-dependent. As an example, in figure 2.2 the two systems O_g and O_i are shown; after $t = 0$, the geometry system has moved somewhat to the right, while it is currently rotating around a rotation axis.

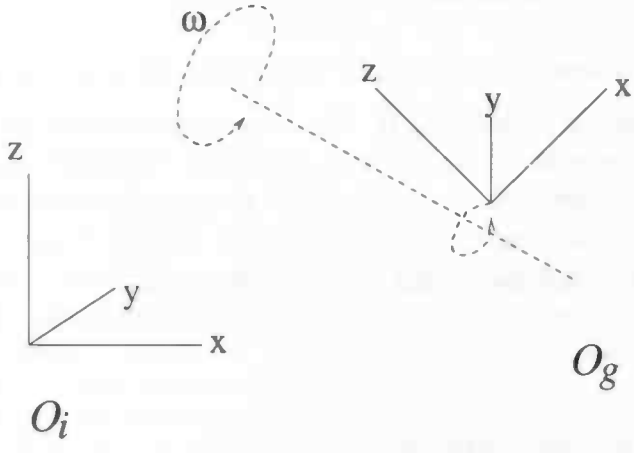


Figure 2.2: *Two coordinate systems*

Chapter 3

Numerical model

In this chapter the numerical model is discussed. This model is implemented in a program called *ComFlo*; sometimes a reference to a certain subroutine is made for advanced users. Most aspects are explained in 2D (mainly because of presentation reasons); extension to 3D is only discussed if it is not straightforward.

3.1 Apertures

In *ComFlo*, the flow domain, Ω, is covered with a rectiliniar grid, which requires a special treatment near the (curved) boundaries. Furthermore, a bookkeeping procedure is required to keep track of the time-dependent fluid configuration $\Omega_f(t)$. Therefore, we introduce *apertures*, i.e. scalars belonging to a cell or a cell face, which contain more detailed information.
The first kind of aperture, the *volume aperture*, is defined in a cell. Firstly, the *geometry* aperture F_b defines which fraction of the cell is contained in Ω: this part of the cell is available for the fluid. Secondly, the *fluid* aperture F_s denotes the cell fraction which is occupied by fluid at a certain time: the part of the cell contained in Ω_f. Because $\Omega_f \subset \Omega$, $\quad 0 \leq F_s \leq F_b \leq 1$.
The simplified case where $F_b \in \{0,1\}$ everywhere, is called a *staircase* geometry, enabling simplified boundary conditions.
The second kind of aperture, the *edge* aperture, is defined on a cell face. The fraction of such a face contained in Ω is denoted by A_x, A_y or A_z, evidently depending on the surface orientation. There are no edge apertures for the fluid.
A lot of information about the orientation of a fluid boundary can be extracted from the described apertures. Moreover, Ω, as described in this way, is source-independent: the original description of the geometry (containing information about angles, curvatures and splines, for example) is not necessary. This approach helps the creation of arbitrary complex forms of geometries, for example by using CSG-trees (see [7]).
In figure 3.1, an example of an aperture distribution is shown.

3.2 Labeling

Before we discretize the equations from Chapter 2, an important issue is the location of the various variables in the Cartesian grid.
The pressure is situated in cell centres. All velocities are staggered: each of the three velocity

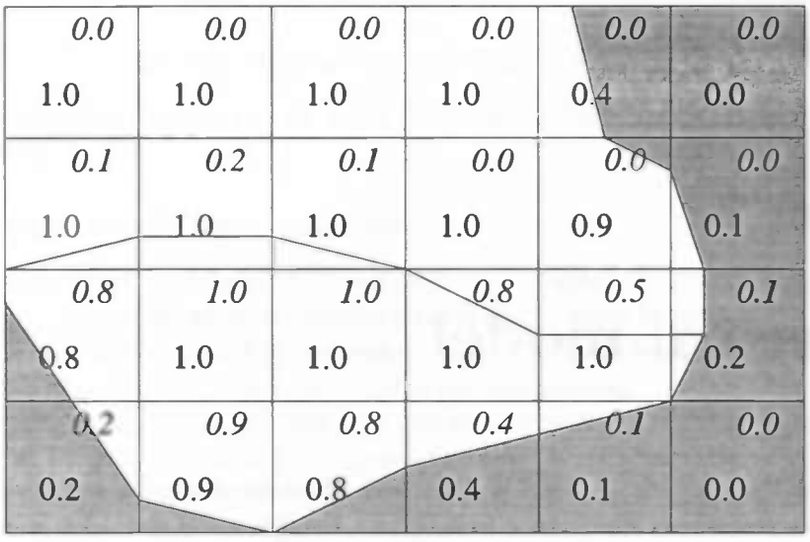


Figure 3.1: *Geometry apertures F_b (lower left) and fluid apertures F_s (upper right) in each cell*

components lies in the centre of cell faces.
Since cells have several functions with respect to solid boundaries and free surfaces, when we solve the equations numerically, they receive labels. Since each cell has a pressure defined in it, these cell labels are called pressure labels.
Based on the geometry only, which is time independent, three labels occur. Cells with $F_b \geq \frac{1}{2}$ are called **F**-(flow) cells. Then, remaining cells which have at least one **F**-cell as neighbour (cells sharing one cell face are called neighbouring cells) are labeled as B-(boundary) cells. All other cells, although possibly having an $F_b > 0$, are not interesting in the discretization process; they are called **X**- (exterior) cells.
Velocity labels are derived from these cell labels; they are called after the cells where these velocities lie between. Obviously these combinations are **FF**, **FB**, **BB**, **BX** and **XX**. The combination **FX** is not possible.
Free surface labels, obviously time-dependent, are a subdivision of the labels described above. First we look again at the pressure labels.
F-labels with $F_s = 0$ contain evidently no fluid, so they are called **E**-(empty) cells. After that, neighbouring **F**-cells, consequently containing fluid, are **S**-(surface) cells. The remaining **F**-cells, situated inside the fluid (possibly having $F_s < 1$), keep their label, but now **F**- is an abbreviation of fluid.

B-cells are, similarly to **F**-cells, divided into three subgroups. These are:
$\mathbf{B}_e$-cells: **B**-cells with $F_b > 0$ and $F_s = 0$;
$\mathbf{B}_s$-cells: **B**-cells with $F_b > 0$ and $F_s > 0$ having at least one **E**- or $\mathbf{B}_e$-neighbour;
$\mathbf{B}_f$-cells: the remaining **B**-cells.
Note that this subdivision does not change the labeling of staircase geometries compared to the old labeling, as described in [7]

A special class of cells has not been accounted for yet: Inflow **I-** and Outflow **O-** cells are separately labeled; usually, they are acquired from B-cells.
As with the geometry labels, free surface velocities are formed out of a combination of cell labels. There are a lot of them:

- **FF**, **FS**, **SS**, the momentum velocities;
- **SE**, **EE**, the free-surface velocities;
- $\mathbf{FB}_f$, $\mathbf{FB}_s$, $\mathbf{SB}_f$, $\mathbf{SB}_s$, $\mathbf{SB}_e$, $\mathbf{EB}_s$, $\mathbf{EB}_e$, the **BF**-velocities ;
- $\mathbf{B}_f\mathbf{B}_f$, $\mathbf{B}_f\mathbf{B}_s$, $\mathbf{B}_s\mathbf{B}_s$, $\mathbf{B}_s\mathbf{B}_e$, $\mathbf{B}_e\mathbf{B}_e$, the **BB**-velocities.

Figure 3.2 shows the labels following from the geometry shown in figure 3.1.

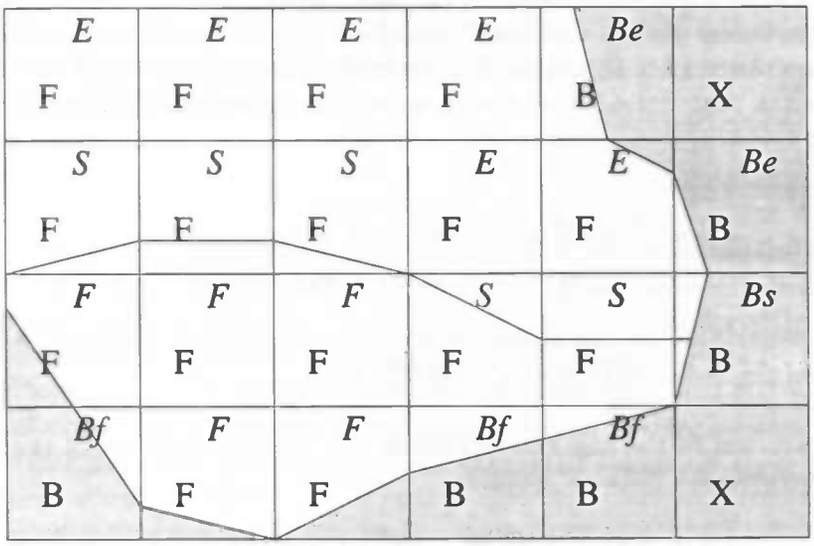


Figure 3.2: *Geometry labels (lower left) and free-surface labels (upper right) in each cell*

Although a lot of alternatives with respect to the labeling of **B**-cells are possible, considering all possible fluid configurations where the free surface meets the solid boundary, the chosen one turned out to be rather convenient.

3.3 Discretizing the Navier-Stokes equations

Having explained the characteristics of the grid and the labels, it is time to use these tools to discretize the Navier-Stokes equations. If we use explicit time integration, and, more specifically, forward Euler, we get the following semi-discrete version of (2.5) and (2.6):

$$\nabla \cdot u^{n+1} = 0 \tag{3.1}$$

$$u^{n+1} - u^n + \delta t \nabla p^{n+1} = \delta t \boldsymbol{R}^n \tag{3.2}$$

Here the term $\boldsymbol{R}^n$ is an abbreviation for the convective and diffusive terms as well as the forces:

$$\boldsymbol{R}^n = \left(-\nabla(u^n u^{nT}) + \nu\nabla\cdot\nabla u^n + \boldsymbol{F}^n + \boldsymbol{f}^n\right)$$

n and $n+1$ are the time levels, and δt is the difference in time between these two levels: the present time step.

3.3.1 The pressure Poisson equation

It is possible to combine equations (3.1) and (3.2) at this stage, but boundary conditions are not available yet. Therefore we first complete the discretization in space.
First, the discrete counterparts of the operators are defined: $\nabla\cdot$ becomes D_h and the discrete version of ∇ will be called G_h, where h denotes the spatial step.
An important step is the following: Because the complete Navier-Stokes equations can only be discretized in **F**-cells, this must be expressed in the $\nabla\cdot$ operator: $D_h = D_h^F + D_h^B$, where the right-hand side terms are defined on the inner domain and the boundary, respectively.
However, a conversion of (3.1) to $D_h^F u^{n+1} + D_h^B u^{n+1}$ is not possible, since the velocity at the boundary is not known; in fact, this velocity should be acquired from internal velocities at the previous time step, i.e. $\boldsymbol{u}_B^{n+1} = f(u_F^n)$, so $\boldsymbol{u}^{n+1} = \boldsymbol{u}^n$ at the solid boundary. The precise value of these boundary velocities is discussed later. Anyway, the complete discretization is

$$\begin{aligned} D_h^F u^{n+1} + D_h^B u^n &= 0 \quad \text{in } \mathbf{F}\text{-cells} && (3.3)\\ u^{n+1} &= u^n + \delta t \boldsymbol{R}_h^n - \delta t G_h p^{n+1} \quad \text{in } \Omega_f && (3.4)\end{aligned}$$

(The right-hand side of the last equation is often written as $\tilde{u} - \delta t G_h p^{n+1}$ since the implementation uses the vector field $\tilde{u} = u^n + \delta t \boldsymbol{R}_h^n$.)

The combination of the last two equations, i.e. the substitution of the second into the first, yields the *pressure Poisson equation*:

$$D_h^F G_h p^{n+1} = D_h^F\left(\frac{\boldsymbol{u}^n}{\delta t} + \boldsymbol{R}_h^n\right) + D_h^B\left(\frac{\boldsymbol{u}^n}{\delta t}\right) \quad \text{in } \mathbf{F}\text{-cells}$$

Now it remains to solve this Poisson equation iteratively. Once solved, the velocity field is acquired by substituting the pressure in equation (3.4). But we will first discuss the actions in two important regions not containing **F**-cells: the solid boundary and the free surface.

3.3.2 Velocities at the solid boundary

As we have seen, conservation of mass in the interior is accounted for by the Navier-Stokes equations. At the boundaries, however, these equations are not used: The pressure is not defined in B-cells, and the velocities are not physical in the sense that they define fluid transport: they are set in the subroutine **BC** in order to fulfill $\boldsymbol{u} = 0$ at the walls; this is done by interpolating and by the use of mirror points.
Although these non-physical velocities are needed for computing the forces and diffusive and convective terms (in subroutine **TILDE**), and in the right-hand side of the pressure Poisson equation, and therefore not unnecessary, they are not suitable for computing fluxes and deriving velocity-related information near the walls. And, above all, they do not ensure mass

conservation.
Of course, B-cells with $F_b = 0$ do not suffer from this problem: effective boundary velocities are zero a priori, and conservation of mass is also fulfilled automatically. Problems only arise if $F_b > 0$.

For these reasons, a new approach was needed to fulfill these aspects:
- Demand conservation of mass in B-cells
- Demand, simultaneously, velocities with more appropriate direction and size (but still use the numerical '**TILDE**'-velocities from (3.4)).

Although the ultimate desire is that these boundary velocities are computed together with internal velocities (thus obtaining all velocities at the same time level), it turns out that there is no such thing as a free lunch.
A first attempt consisted of using the fact that conservation of mass alone can easily be demanded by incorporating it in the pressure Poisson equation (using the pressure gradients afterwards to obtain $\nabla \cdot \boldsymbol{u} = 0$ in **B**-cells). However, although the velocities often approached the new boundary conditions (obtained from the new internal velocities, of course) quite well, this method lacked robustness.

The other approach is to compute all effective boundary velocities (fluxes) at the beginning of the time step, before computing the new velocities. Thus, boundary fluxes are, in this way, known before the interior fluxes.
The idea is to compute the effective **BB**-velocities directly from tangentially orientated internal velocities, somewhat similar to the computation of the numerical **BB**-velocities. The remaining **BF**-velocities are, thereafter, solved from a lineair system containing equations expressing conservation of mass and the prescribed tangential velocity of these **BF**-velocities when extrapolated towards the wall.
In the following section this method is discussed.

3.3.3 Effective boundary velocities

We need to define an initial boundary velocity field u_{eff}, which redefines velocities with labels **FB** and **BB**.
These velocities must represent fluxes; that is, they should immediately determine the flux values between cells. The idea is, therefore, that velocities in partly open cell faces are considered to be placed right between the cell edge and the point where the boundary crosses the cell face, or, in 3D, in the centroid of the triangle determined by the apertures (see figure 3.3).

They are, at the beginning of the new time step, derived from the (internal) momentum velocities of the previous time step. An important aspect of effective velocities is that their sign is always the same as the direction of the internal velocity, thus preventing small recirculation areas.

Since the treatment of **BB**- and **BF**- velocities is different, the configurations of **F**-cells

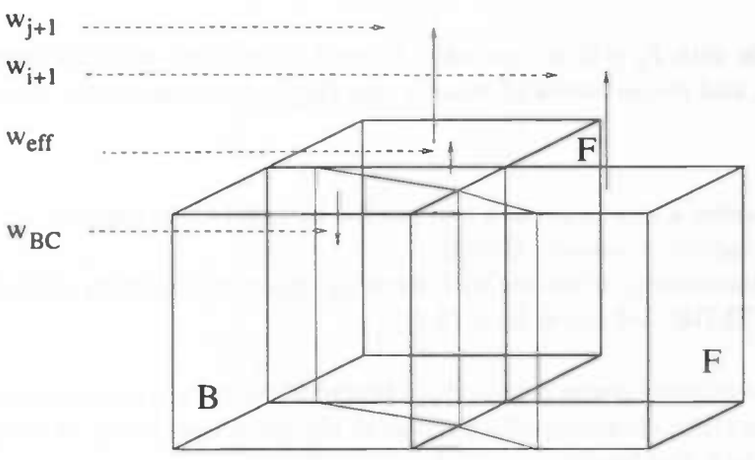


Figure 3.3: *Example of standard and effective BB-velocities*

around a **B**-cell are very important.

A priori, some bad configurations exist which should be recognized first. They have in common that a **B**-cell is situated between two **F**-cells, each belonging to different parts of the geometry; the wall between them is too thin (this is an indication that the user should have used a finer grid or a thicker wall there). These cases are solved in a crude, but very effective way: the geometry aperture F_b is set to zero. This is because it is impossible to determine a unique direction of the wall (i.e. to determine a unique normal per **B**-cell); see figure 3.4.

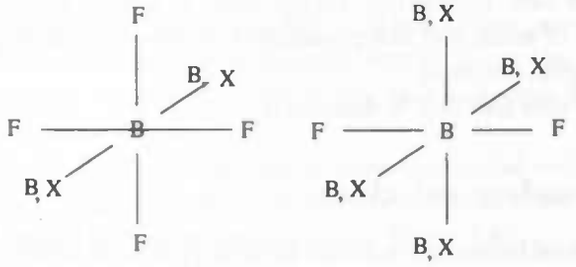


Figure 3.4: *Two examples of bad configurations*

Secondly, some other configurations must be transfered to simpler cases, since, at a maximum, three equations per cell exist: one for conservation of mass and two with respect to the tangential directions (because in each cell only one normal vector is defined, only two independent tangential directions can be obtained). Moreover, the **BF**-velocities should lie next to each other (in 3D: only one edge of the cell should lie between them). Therefore, all situations where a **B**-cell lies *between* two **BF**-velocities are converted to others. This is done by treating those **BF**-velocities as **BB**-velocities; in figure 3.5 an example is shown.

At this point, maximal three actual **BF**-velocities per **B**-cell exist; in addition, they lie next to each other. Therefore we can distinguish three different cases, according to the

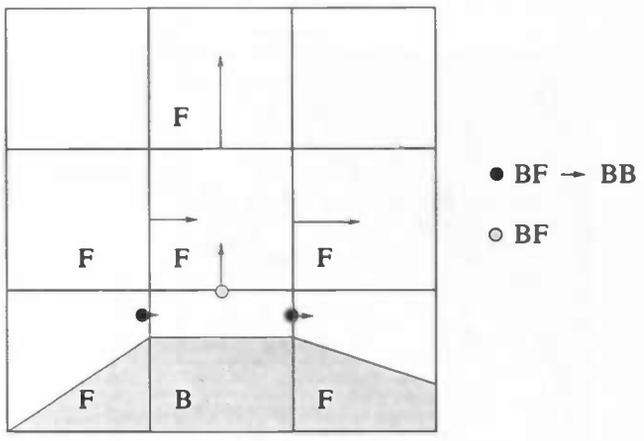


Figure 3.5: *In certain configurations, superfluous BF-velocities are considered effective* **BB** *velocities*

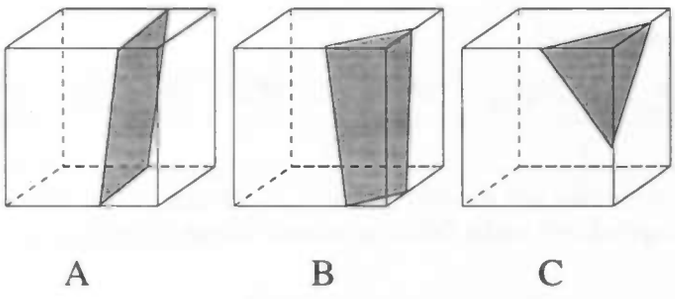


Figure 3.6: *Three different cases with respect to BF-velocities*

amount of actual **BF**-velocities (see also figure 3.6). First, the effective **BB**-velocities, which are considered tangential velocities, are computed. They differ much from the numerical **BB**-velocities since they are not located in the middle of the cell, but between the cell edge and the point where the boundary crosses the cell face; therefore, the interpolation formula uses different distances. For example, in the case of a pure vertical wall through **B**-cells with index i, the **BB**-velocity between them is (see also figure 3.7)

$$w_{BB} = \frac{\frac{1}{2}A_i^z h_i w_{i-1} + \frac{1}{2}(h_{i-1} + A_i^z h_i) w_W}{\frac{1}{2}h_{i-1} + A_i^z h_i}$$

where w_W, the velocity on the wall, is often zero. In the figure, for example, with $w_W = 0$ and $A_i^z = \frac{1}{2}$, the **BB**-velocity becomes $\frac{1}{4}w_{i-1}$, as could be expected.

Now it remains, in each of the three cases, to compute the **BF**-velocities:

- Case A
  Only one **BF**-velocity remains. This velocity is directly solved from $\nabla \cdot \boldsymbol{u} = 0$ (using,

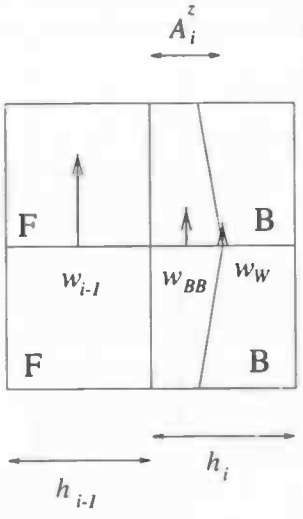


Figure 3.7: *Interpolation for effective BB-velocities*

as always, the apertures):

$$\frac{A_{i-1}^x u_{i-1} - A_i^x u_i}{h_x} + \frac{A_{j-1}^y v_{j-1} - A_j^y v_j}{h_y} + \frac{A_{k-1}^z w_{k-1} - A_k^z w_k}{h_z} = 0 \qquad (3.5)$$

Five of these velocities are already known: They are either effective **BB**-velocities or velocities through closed walls (with aperture values of zero).

- Case B
  There are two **BF**-velocities. This is in fact a 2D-situation, since the only terms in the third direction arise in the mass conservation equation, on the right-hand side. Hence we get a lineair system $A_B x = b_B$, with x the two unknown **BF**-velocities. The first equation is similar to case A.
  The second equation is acquired from the precise tangential velocity at the wall: $\boldsymbol{u} \cdot \boldsymbol{t} = 0$ (or another value). The wall velocity is extrapolated from an interior momentum velocity and the desired boundary velocity (see figure 3.8).

  The extrapolated velocities are not located at the same point, but the tangential vector is constant along the entire wall in the **B**-cell since the apertures contain just that amount of information.
  Let us work out such an example in 2D: Suppose the two unknown velocities are at the left (u) and lower (v) side of the **B**-cell with co-ordinates (i, j, k). Then we start with

$$A_{i-1}^x h_y u_{i-1} + A_{j-1}^y h_x v_{j-1} = A_i^x h_y u_i + A_j^y h_x v_j \qquad (3.6)$$

$$u_W t_x + v_W t_y = 0 \qquad (3.7)$$

  The right-hand side of (3.6) consists of previously computed effective **BB**-velocities or **BX**-velocities (in 3D, two terms with w should be added there). Now the velocities at the wall u_W and v_W must be expressed in the two desired **BF**-velocities u_B and v_B (in

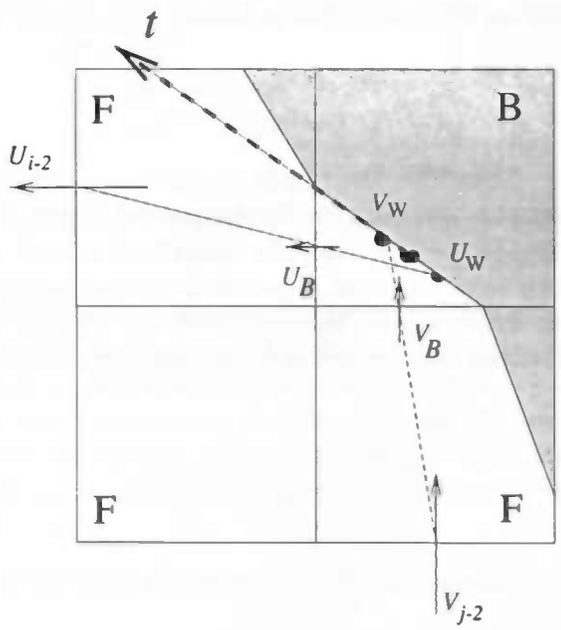


Figure 3.8: *second equation: tangential velocity at the wall*

(3.6): u_{i-1} and v_{j-1}) and two momentum velocities, u_{i-2} and v_{j-2} :
$u_W = \alpha_1 u_{i-2} + \alpha_2 u_B, \quad v_W = \beta_1 v_{j-2} + \beta_2 v_B$.
By extrapolation, it follows that

$$\alpha_1 = -\frac{F_b h_x \mid_i}{F_b h_x \mid_{i-1}}, \quad \beta_1 = -\frac{F_b h_y \mid_j}{F_b h_y \mid_{j-1}}$$

and $\alpha_2 = 1 - \alpha_1, \quad \beta_2 = 1 - \beta_1$.
Consequently, we get the following system:

$$\begin{pmatrix} A^x_{i-1} h_y & A^y_{j-1} h_x \\ \alpha_2 t_x & \beta_2 t_y \end{pmatrix} \begin{pmatrix} u_B \\ v_B \end{pmatrix} = \begin{pmatrix} A^x_i h_y u_i + A^y h_x v_j \\ -\alpha_1 t_x u_{i-2} - \beta_1 t_y v_{j-2} \end{pmatrix} \tag{3.8}$$

- Case C
  In this pure three-dimensional variant, we have three **BF**-velocities in a **B**-cell. The method is almost the same as in case B; now we have a second tangential vector u, giving the third equation, similar to the second. This lineair system $A_C x = b_C$ with $x = (u_B, v_B, w_B)^T$ is also explicitly solved.

An important demand is not discussed yet: the matrices A_B and A_C must be non-singular. Take a look at the general form of A_B:

$$A_B = \begin{pmatrix} I_l \frac{A^x}{h_x} \mid_{ib} & I_d \frac{A^y}{h_y} \mid_{jb} \\ \alpha_2 t_x & \beta_2 t_y \end{pmatrix}$$

Here I_l and I_d (the subscripts denote left and down, respectively) are special indicator functions which determine on which side the unknown **BF**-velocity is located:

$$I_l = \begin{cases} 1 & : \quad u_B = u_{i-1} \\ -1 & : \quad u_B = u_i \end{cases}$$

Since $\alpha_1 \leq 0$, $\alpha_2 \geq 1$. Therefore, the upper row of the matrix is globally determined by the signs of I_l and I_u, and the lower row by the tangential vector $\boldsymbol{t}$. One can image that the vector $(I_l, I_u)^T$ determines, in a crude way, the (negative) normal of the boundary, causing the two rows of matrix A_B to never being dependent (see figure 3.9).
The same argument also holds in the three-dimensional case C: the first row is globally orthogonal to each of the other rows, which roughly describe the two tangential vectors $\boldsymbol{t}$ and $\boldsymbol{u}$.

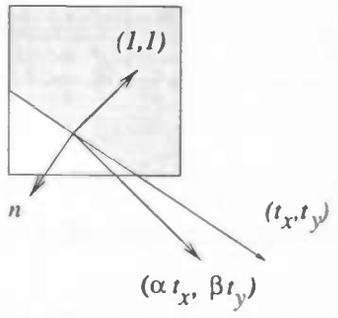


Figure 3.9: *rows of the matrix with* $I_l = I_d = 1$

Once these boundary velocities have been computed, they are used in the right-hand side of the pressure Poisson equation (the part $D_h^B u^n$) and in the fluxes through B-cell faces: $F_x = A_x h_y h_z u_B$, and so on.
The method as described above does also hold for staircase geometries: in that case, the effective **BB**-velocities are always zero, as expected, and the **BF**-velocities are treated like case A: mass conservation forces them to become zero, like the numerical **BF**-velocities. **BB**-velocities are only used for the fluxes, which should indeed be zero.

3.3.4 Free surface

Near the free surface, the boundary conditions (2.7) and (2.8) must be discretized. Remember that **FF**-, **FS**- and **SS**-labels are considered momentum velocities. Therefore a pressure must be defined in an **S**-cell, and the discretization molecules of these velocities involve surrounding **SE**- and **EE**-velocities.

We first discuss these velocities:

- **EE**-velocities. At first, **EE**-velocities which are not surrounded by at least one **SS**-velocity are far away from the fluid: they are set to zero. The other **EE**-velocities are computed using the discrete simplified version of (2.7): the Cartesian orientation of the

surface is determined (x, y or z) and dependent of the configuration of neighbouring **SS**-velocities, some equations of

$$\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} = 0, \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} = 0, \frac{\partial w}{\partial y} + \frac{\partial v}{\partial z} = 0. \tag{3.9}$$

are used.
However, one (or more) of the velocities used here can be **SE**, so they should be known first.

- **SE**-velocities. As we have seen, these velocities appear not only in momentum equations, but may also be needed to compute **EE**-velocities.
  A way to obtain such an **SE**-velocity is demanding $\nabla \cdot u = 0$ in the corresponding **S**- cell. Discretizing mass conservation uses six velocities; the **SE**-velocity must be solved from the other five, which can be **FS**, **SS** and **SE**. In the latter case, with apparently more than one unknown, some other decisions have to be made, such as setting individual derivatives like $\frac{\partial u}{\partial x}$ to zero.

The determination of the free-surface velocities is more extensively treated in [6].

Now the pressure needs to be determined. In **E**-cells the pressure is simply set to its reference (atmospherical) value p_0.
In **S**-cells, a pressure in the centre of the cell is needed. Therefore we use a local height function to interpolate the internal pressure p_F in the neighbouring **F**-cell and the pressure on the surface p_f, where $p_f = p_0 - 2\gamma H$. Here the term $2\mu \frac{\partial u_n}{\partial n}$ is neglected.
The formule for the interpolation is (see also figure 3.10:)

$$p_S + (\eta - 1)p_F = \eta\, p_f, \quad \text{where} \quad \eta = \frac{h}{d} \tag{3.10}$$

Now all measures have been taken to solve the Poisson equation, except for the combination of the former cases: where the free surface meets the solid boundary.

3.3.5 Free surface near the solid boundary

The most difficult situation is the combination of the free surface and solid boundary.
At first, it should be noticed that using the solid boundary conditions (i.e. interpolating momentum velocities towards the wall) is not always handy: for example, if an amount of fluid rushes towards the solid boundary, it is not slowed down until it reaches the wall. Therefore, the velocities of this surface approaching the boundary should not be affected by the solid boundary conditions until, say, the last interior cell is filled.
Thus, the boundary velocities also depend on the fractions F_s/F_b of the concerning cells.
Secondly, the handling of $\mathbf{B}_s$-cells is rather difficult. Since no pressure is defined in these cells and a similar treatment of the computation of **SE**- and **EE**-velocities in the interior fails due to the nearness of the solid wall, some ad hoc measurements had to be made. Further on in this chapter, the consequences for the flux moving algorithm are discussed.

In micro-gravity circumstances, the movement at the wall is mainly determined by the contact angle between the fluid and the wall. Here smooth and staircase walls lead clearly to different results.

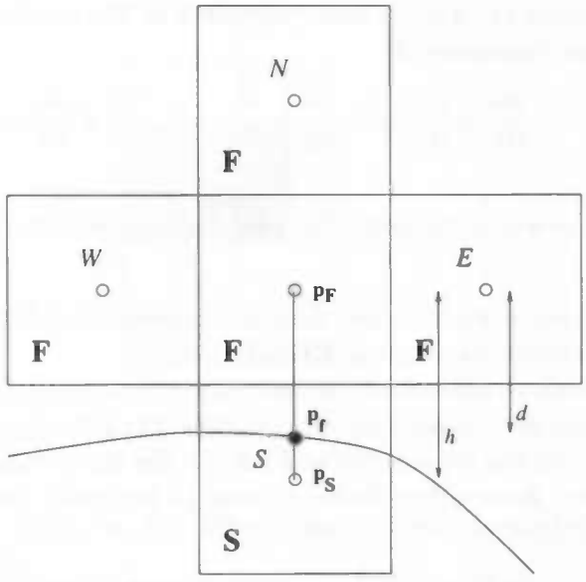


Figure 3.10: *Pressure locations at free surface and in the neighbouring cells*

3.3.6 Recapitulation

As an intermediate summary of the flow of the computation, a short description, as implemented in the computer program, follows. The indicated subroutines are described in Appendix A.

1. Set all internal velocities to zero. **(INIT)**

2. Compute effective boundary velocities out of the internal velocities of the previous time step. The **BB**-velocities are directly computed from tangentially orientated velocities in the interior; the remaining **FB** velocities are obtained from lineair systems. **(EFFBC)**

3. Compute the temporary vector field $u^n + \delta t \boldsymbol{R}^n$. Here $\boldsymbol{R}^n$ contains all convective and diffusive terms and forces (body forces and external forces). u^n is the internal, numerical velocity field of the previous time step. Convective terms are computed using apertures, diffusive terms are not. **(TILDE)**

4. Compute the left-hand side coefficients of the pressure Poisson equation. Here the apertures are used. **(COEFL)**

5. Compute the right-hand side of the pressure Poisson equation: the divergence of the newest vector fields. In the case of internal cells, this is the temporary vector field constructed in **TILDE**. Near the solid boundary, part of these velocities (D_h^B) are the flux velocities. **(COEFR)**

6. Solve the pressure Poisson equation in each **F**- cell. After that, a pressure gradient is added to the temporary internal velocities. **(SOLVEP)**

7. Move the fluid using the Donor-Acceptor algorithm (see the following section). All velocities make sense with respect to direction and size (this was just the aim of introducing effective velocities). The fluxes, therefore, are obtained from them in a straightforward way.**(VFCONV)**

3.4 Adjusted Donor-Acceptor algorithm

Once all velocities are known, the fluid has to be moved. After all changes described above (and for other reasons), the adjusted Donor-Acceptor algorithm as described in [7] needs again some adjustments.

Firstly, the rules now hold for B_f-cells as well: Effective **BF**- and **BB**-velocities lead directly to fluxes.

3.4.1 Restricted fluxes

It is dangerous to move plain fluxes near the free surface. Especially the **SE**-velocities (computed from $\nabla \cdot \boldsymbol{u} = 0$) and fluxes between partly empty cells in general are often too large. [4] suggested the following statement for moving fluxes near the free surface:

$$\delta F = \text{MIN}\left(F_{AD} \mid u \mid \delta t + CF\,, F_D h_x\right)$$

where F is the relative amount of fluid in the cell ($0 \leq F \leq 1$) and

$$CF = \text{MAX}\left(1 - F_{AD} \mid u \mid \delta t - (1 - F_D)h_x, \quad 0\right)$$

Here the subscripts A and D denote acceptor and donor cells, respectively. AD must be replaced by A or D, depending on the surface orientation.
For example, taking $AD = A$, the total flux is more restricted by the fluid amount of the donor cell in the case of fuller acceptor cells; if the acceptor cell is full, the maximum flux is transported, independent of F_D.
In general, $AD = A$ is more useful when the fluid is transported mostly normal to itself; otherwise, $AD = D$ is used.

In our case, taking into account the values of F_b near boundaries, the statement is replaced by

$$\delta F = \text{MIN}\left(\frac{F_s^{AD}}{F_b^{AD}} \mid u \mid \delta t + \text{MAX}\left[\frac{F_b^{AD} - F_s^{AD}}{F_b^{AD}} \mid u \mid \delta t - (F_b^D - F_s^D)h_x, 0\right], F_s^D h_x\right)$$

Despite this, it turned out that it was necessary to take extra measures to prevent the surface region of being 'sprayed' in a too large space. (This phenomenon, the creation of numerical droplets and bubbles, is a well-known disadvantage of the VOF-method and it is sometimes called 'jetsam' and 'flotsam'; see also [8].) These measures involve the use of a local height function at the free surface; the new 'height' is determined in a local row in one of the three main (x, y, or z) directions (see figure 3.11). Thus, the fluid in that column (in the figure, it is actually a row, since the surface is orientated vertically) is summed up from the local 'bottom' and the new height is computed.

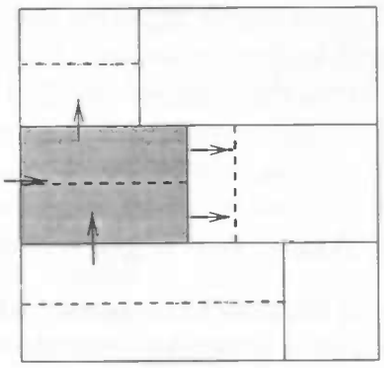


Figure 3.11: *The new 'height' is determined in the local row*

This local height function is also defined near the solid boundary; it therefore depends also on the cell space F_b.
In Chapter 4, an example of the impact of these measures is shown.

3.4.2 Bubble filling

A lot of problems occur where the free surface meets the solid boundary. This is partly due to the labeling: **S**-cells differ only from **F**-cells in having an empty neighbour; at walls, however, this condition is sometimes too severe, thus creating too many **F**-cells there with $F_s < F_b$ (see figure 3.12). The removal of these bubbles, which can be seen as the collapsement of entrapped air, should therefore be considered.

One option is to change those cells in **S**- cells, thus using the fact that empty cells are not found there, and conservation of mass can not be demanded. However, preventing an 'overfill' of these cells, while the extra fluid can not be drained away, is almost impossible.

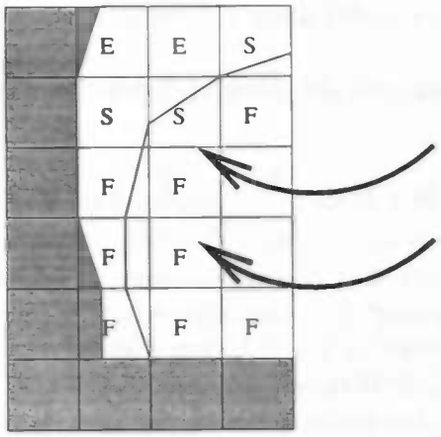


Figure 3.12: *Fluid splashing against a wall*

Another option, which does not change the labeling, is: Demand in such an **F-** cell, which is supposed to be filled (if a fluid gets away from the wall, it does so by moving the free surface along the wall; wall holes are not supposed to bubble up spontaneously), that $\nabla \cdot u^{n+1} = L$, instead of $\nabla \cdot u^{n+1} = 0$, where L is chosen such that the specified cell contains more fluid after moving the fluxes.

By substituting equation (3.2) in this modified divergence equation, as always, we get

$$\nabla \cdot \nabla p^{n+1} = \nabla \cdot \frac{1}{\delta t}(u^n + \delta t \boldsymbol{R}^n - L) \tag{3.11}$$

A choice for L is

$$L = -\frac{1}{\delta t} MIN\,(\alpha F_b, (F_b - F_s)) \tag{3.12}$$

The minus sign denotes a netto increase of fluid in the cell. Furthermore, $L\delta t$ is limited by $\alpha \cdot F_b$ (where α is a positive parameter less than 1) to avoid conflicts with the CFL-condition. The division by δt is evident, since the movement of fluxes in a cell is defined by

$$F_s^{new} = F_s^{old} + \left(-\frac{\Delta F_x}{\Delta x} - \frac{\Delta F_y}{\Delta y} - \frac{\Delta F_z}{\delta z}\right)$$

where F_x, F_y and F_z are fluxes, defined by $F_x = u\delta t$ and so on.

Therefore,

$$F_s^{new} = F_s^{old} + \left(-\frac{\Delta u}{\Delta x} - \frac{\Delta v}{\Delta y} - \frac{\Delta w}{\delta z}\right) \cdot \delta t$$

Normally , the difference between F_s^{new} and F_s^{old} is zero. To increase F_s, a term between the parentheses should be added. An L of size $\frac{1}{\delta t}(F_b - F_s)$ just makes the new F_s exactly equal to F_b.

This technique is also handy when two surfaces in the interior meet: the collision takes place in a few time steps when the distance of the two surfaces is less than one cell. Again, the other way around (the creation of a gap) does not occur spontaneously.
Of course, this method is not always necessary or even desired, but can be useful in a difficult computation of a wildly sloshing fluid, such as the rotating cilinder discussed in subsection 4.5.1. Moreover, the bubbles cause a lot of secondary information to be less exactly computed (the force upon the wall, for example).

3.5 Determining time steps

An important feature of a simulation is the possibility of adjusting time steps: during calm intervals, a larger time step can be permitted; in difficult and unstable situations, a reduced time step is needed.
The tool to achieve this variation in time steps is the Courant-Friedrichs-Levy number (CFL number), defined as:

$$\text{CFL} = \text{MAX}\left(\frac{|\,u\,|\cdot\delta t}{h_x}, \frac{|\,v\,|\cdot\delta t}{h_y}, \frac{|\,w\,|\cdot\delta t}{h_z}\right).$$

Here h_x, h_y and h_z denote distances between the two cell centra between which the velocity is defined.

Numerical analysis shows that the CFL number, in any cell, must not exceed 1. This demand can in the Donor-Acceptor algorithm be seen as the impossibility of fluxes ($u \cdot \delta t$) moving further than one cell. Therefore, a variant of the definition with each velocity multiplied by the edge aperture ($\frac{|u| \cdot \delta t \cdot A_x}{h_x}$, for example) is also possible.
The CFL-condition is implemented in the following way: If the CFL number in any cell exceeds a certain maximum $\mathrm{CFL}_{\max}$ (usually less than 1.0), the time step is immediately, *for the next time cycle*, halved (instead of repeating the current time step, which is awkward because of the resetting of all variables).
On the other hand, if the CFL number of all cells stays smaller than a certain limiter $\mathrm{CFL}_{\min} < \frac{1}{2}\mathrm{CFL}_{\max}$ for a certain amount of cycles, the time step is doubled.

3.6 Forces and changing coordinate systems

As the coordinate system of the geometry O_g stays the same during the simulation, we must keep track of the relation with respect to the inertial coordinate system O_i since some parameters are not changed by rotation (like the gravitation) ; moreover, the total transformation is needed for visualization purposes.
A relative position $(x, y, z)^T$ is related to the inertial position $(x', y', z')^T$ by

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1' \end{pmatrix} = \begin{pmatrix} f_{1_1} & f_{2_1} & f_{3_1} & -t_x \\ f_{1_2} & f_{2_2} & f_{3_2} & -t_y \\ f_{1_3} & f_{2_3} & f_{3_3} & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

where f_1, f_2, f_3 are the unit vectors of O_g relative to O_i, and $(t_x, t_y, t_z)^T$ is the distance between the two origins (the translation vector).

The matrix above changes due to a rotation vector $\omega_g \neq 0$ or due to some translation; the translation is only important for visualization purposes, but rotation influences the simulation directly.
Therefore, the following algorithm has been implemented:

ALGORITHM to keep track of a changing coordinate system

DO (each time when ω_g changes) OR
(each time step, if the body translation or gravitation is defined in O_i):

1. Determine $\theta = \mid \omega_g \mid (T - T_o)$ where T_o was the last time this procedure has been executed. θ is the angle in the plane orthogonal to ω_g which the system rotates with.

2. Compute $\omega_i = (f_1\, f_2\, f_3)\, \omega_g$. This is the rotation vector in the inertial system.

3. Determine a third coordinate system O_{ω_i} given by three unit vectors u_x, u_y, u_z, where $u_z = \frac{\omega_i}{|\omega_i|}$.
   u_y and u_x are then easily constructed, for example by (see also [3]) $u_y = \frac{\omega_i \times e_1}{|\omega_i \times e_1|}$ and $u_x = u_y \times u_z$.

4. The total additional rotation of O_g with respect to O_i is now:

$$R(\theta) = (R_{y,x})^{-1} R_z(\theta) R_{y,x}$$

   where R_z means rotation around the z-axis, and $R_{y,x}$ is the matrix

$$\begin{pmatrix} u_x{}^T & 0 \\ u_y{}^T & 0 \\ u_z{}^T & 0 \\ 0 \;\; 0 \;\; 0 & 1 \end{pmatrix}.$$

   Note that $(R_{y,x})^{-1} = (R_{y,x})^T$ since the u's are orthonormal.

5. The new unit vectors f_i are now simply

$$f_i{}^{new} = R(\theta) f_i{}^{old}, \quad i = 1 \cdots 3.$$

6. Now translation components are included. If the prescribed translation (obtained by integrating q) is supposed to be in the inertial system, multiplication with the new unit vectors is needed first.
   An extra translation is included here if the rotation axis, described by ω_g, does not intersect the origin. Normally, this is taken into account by pre- and postmultiplying with translation matrices, but the total impact is rather trivial. Indeed, the following holds:

$$T_{xyz} = (x_0, y_0, z_0)^T + R(\theta)(-x_0, -y_0, -z_0)^T$$

   where $(x_0, y_0, z_0)^T$ is the center of rotation (or, in fact, an arbitrary point which ω is bound to intersect).

7. The two types of translation described in 6) form the fourth column in the transformation matrix, which can be stored for postprocessing purposes. The simulation can continue; the new unit vectors were already computed in 5).

8. If forces such as gravitation are supposed to be defined in the inertial system, then these vectors should be recomputed as well. Note that if the unit vectors are transformed by a certain rotation vector ω, the gravity vector is rotated by $-\omega$, i.e. the new gravity vector is

$$g_{new} = (R(\theta))^{-1} g_{old}$$

   where $R(\theta)$ is the matrix defined in step 4).

END DO

Although this algorithm is called every time step in some cases (inertial gravity, for example), it hardly influences the computation time. The simulation continues within the same coordinate system; only some external forces change.

Chapter 4

Results

We now present some results of simulations performed with the program *ComFlo*. These results are new in the sense that they could not have been done with the old version from August 1997; so most results show improvements in the fields of forces, boundary velocities and fluxes.

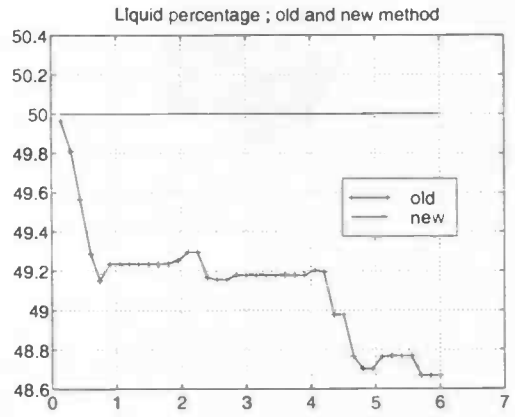


Figure 4.1: *amount of liquid during a simulation*

4.1 Mass conservation

Fluid loss in 3D-dambreak

It should be noted that, in non-staircase geometries, the loss of fluid can be quite high using the old method (in which mass conservation in B-cells did not hold). This is mainly because of an overfill of cells near the boundary (if the fluxes towards B-cells are moved, it concerns the B-cells itself: they will sometimes get an $F_s > F_b$. If not, the F-cells next to those boundary cells suffer of fluid loss).
In staircase geometries, nevertheless, these problems do not occur since the B-cells contain no fluid.

The following graph (4.1) shows a typical fluid loss of a standard problem: a sphere was partly filled with liquid and then rotated. The walls were fairly smooth, although the smoothness does not directly influence the fluid loss as long as it is not a staircase geometry. The computation was done on a 20^3 grid and lasted some 300 cycles. Other simulations show a similar loss, at least with the current CFL-limiters of 0.4 for $\mathrm{CFL}_{\mathrm{MAX}}$, see also section 3.5. In general, higher CFL-limiters make it more difficult to adjust the free surface; this fact directly influences the fluid loss in the old method.

4.2 Flux moving algorithm

We will now give some examples of the progress that has been made in the flux moving algorithm.

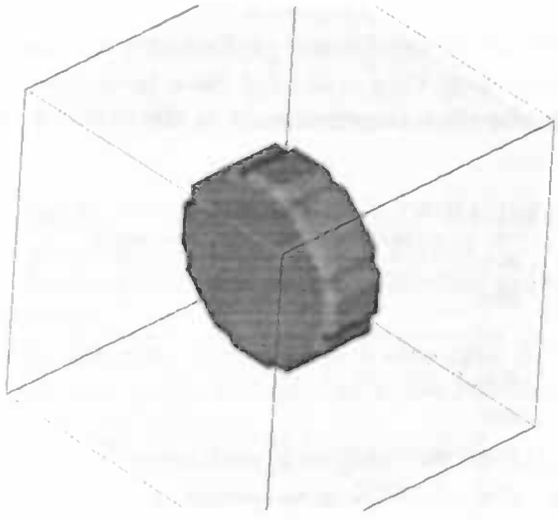


Figure 4.2: *spinning disk: $t = 0$*

4.2.1 Spinning disk

First the well-known testcase of the spinning disk is performed. The disk (see figure 4.2; note that the disk is drawn using an isosurface of the F_s function; hence the nonperfect shape at $t = 0$), with radius $0.3\,m$ and height $0.2\,m$, is spinned using an angular velocity vector $\boldsymbol{\omega} = (1, 1, 0)^T$ and the velocity field $\boldsymbol{\omega} \times \boldsymbol{r}$. This is easily done by overruling the computed velocities by velocities following from this motion, i.e.

$$u = z, \quad v = -z, \quad w = y - x$$

So this is not a standard simulation: the velocities are prescribed a priori. The angular velocity is $\mid \boldsymbol{\omega} \mid = \sqrt{2}\, s^{-1}$. Hence for one full rotation $\sqrt{2\pi}\, s$ are needed. This is simulated in 200 time cycles.
These parameters were also used by [11], to compare another interface tracking method with

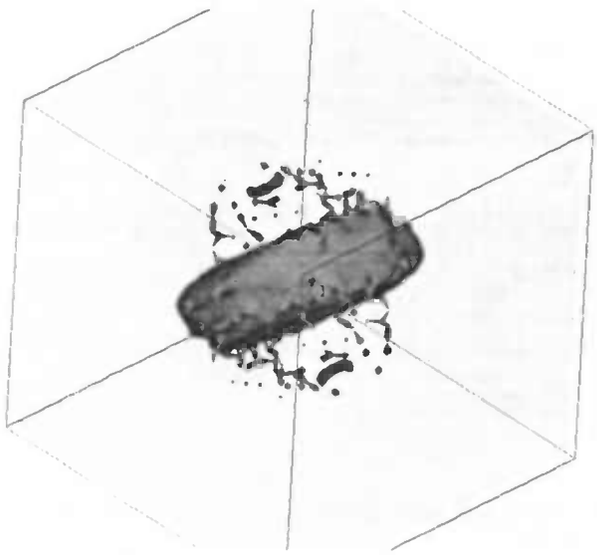


Figure 4.3: *spinning disk, old:* $t = \sqrt{2\pi}/2$

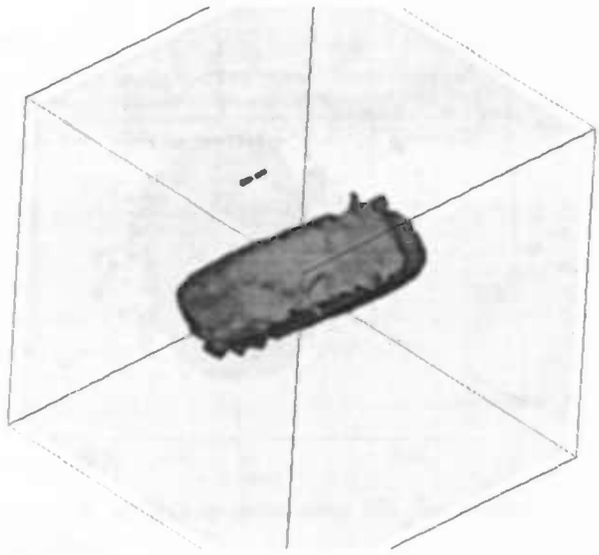


Figure 4.4: *spinning disk, new:* $t = \sqrt{2\pi}/2$

VOF.
The other four figures show the the disk halfway the simulation (4.3, 4.4) and at the end (4.5, 4.6). The problem, apart from keeping the sharp corners (which is almost impossible) is to hold the disk together, and in the right shape. The normal Donor-Acceptor method is known for its bad results here. However, using the local height function mentioned in section 3.4, a lot of improvements can be made. This is especially important because the numerical model is less able to deal with lots of small particles mainly due to the failure in computing orientations there.
It is evident that both methods have problems preserving the original shape, but standard VOF is much less able to keep the fluid together.
It is also clear that the normal Donor-Acceptor method is pure symmetric, despite the standard order of $z-$, $y-$ and $x-$ loops. The local height function, however, seems to be partly dependent of this order. The main problem here, using standard VOF, is the fact that fluid tends to get behind the moving disk: like a comet, volumes of fluid accompany themselves with a tail. In the near future, improvements should be made to prevent these fluid particles from falling behind.
The simulations were done on a $40 \times 40 \times 40$ grid and both lasted some twenty minutes on a workstation; the extra actions involved with the newer method are hardly detectable in computing time.

4.2.2 Boundary collision

When two surfaces (two free surfaces *or* a free surface and a solid boundary) are bound to collide, mass conservation prevents a real 'melting' together at cell level due to the labeling (see also section 3.4), causing a lot of partly empty cells not on the surface. By changing the demands for the divergence in these cases, this problem is mainly removed from our list, as shows the following example of a rather normal sloshing problem.

Here a dambreak in a 2D-cylinder (a disk) is used. The radius is $0.5\,m$, and initially the

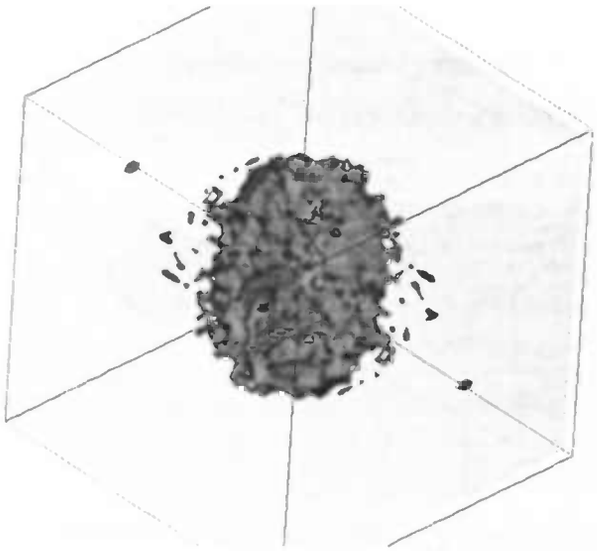


Figure 4.5: *spinning disk, old:* $t = \sqrt{2\pi}$

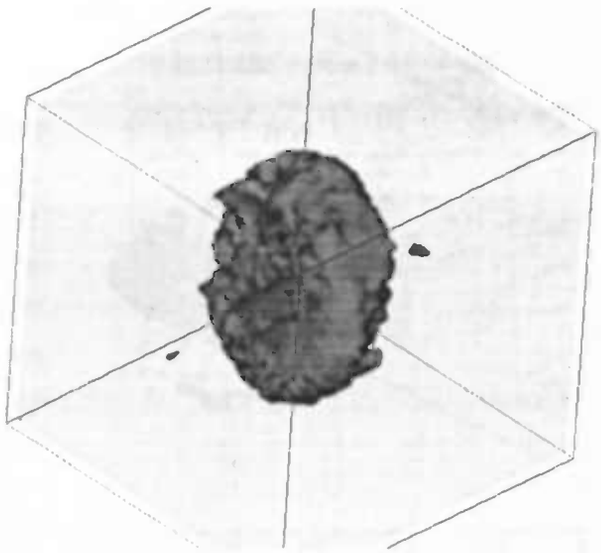


Figure 4.6: *spinning disk, new:* $t = \sqrt{2\pi}$

area $x \geq 0.0$ is filled with fluid of viscosity $\nu = 0.01$. At $t = 0\,s$, the dam is removed and the fluid moves downward due to a gravitational force of $g = 5\,m\,s^{-2}$.
In figures 4.7 and 4.8, the dambreak is simulated in a staircase geometry; in figures 4.9 and 4.10, a smooth geometry is used.
The grid is very coarse: there are only ten cells in each direction. In the old situations, air is entrapped near the walls. Obviously, this strongly influences the forces upon these walls and computations of contact angles.
The progress as shown in the new situations is evident: in the old method, bubbles along the wall occur, and, in the staircase geometry, even in the interior. In the latter case, mass conservation in the cell containing that bubble can be recognized (the arrows represent the real velocities, without having manipulated them).
The method works also for smooth geometries, which contain several nonempty B-cells.
Although one may argue that this is a somewhat crude method to simulate escaping bubbles, the simulation is positively influenced (extra, never-ending fluid transport, from one hole to the other and back, is prevented) while postprocessing results show less 'white noise'.
All these velocity plots are produced by Matlab; the boundaries of the geometry and the free surface are drawn using level lines; this explains the diagonal lines in the figures representing staircase geometries.

4.3 Effective boundary fluxes

The influence of the new velocities near the boundary is directly visible in most velocity plots. As an example, a flow through a pipe, with a slope of 0.3, was simulated. The velocities are shown per component, on the places where they are computed.
The first figure, figure 4.11 shows the old method: all velocities are the old, numerical ones. Because this is a problem without free surfaces, the flux moving algorithm was turned off as usual in situations without free surfaces (if not, since mass conservation does not hold near

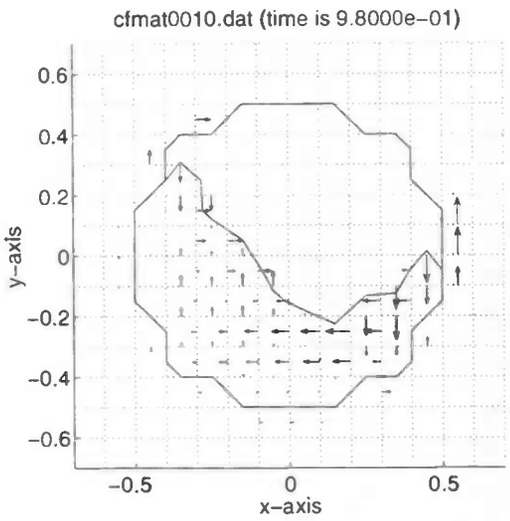


Figure 4.7: *staircase geometry; new flux moving method*

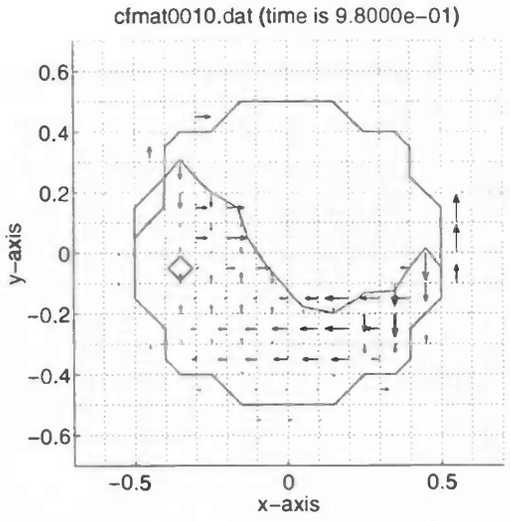


Figure 4.8: *staircase geometry; old flux moving method*

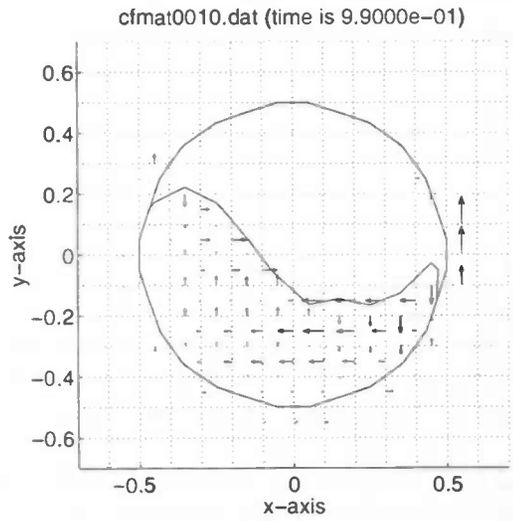


Figure 4.9: *smooth geometry; new flux moving method*

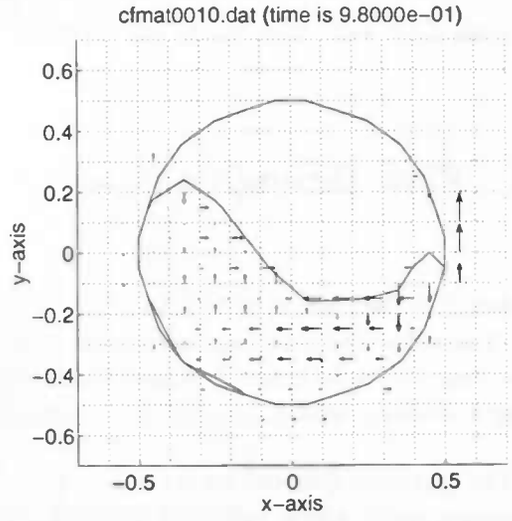


Figure 4.10: *smooth geometry; old flux moving method*

the boundary, a lot of problems would occur). Note that, in this old method, in a lot of B-cells small recirculation areas can be found.
Figure 4.12 shows the new method, with effective velocities. The velocities are zero in various B-cells with closed apertures. In other B-cells, they are plotted in the middle of the cell faces, but they should of course be regarded as situated between the wall and the cell edge. Note that all boundary velocities now have the same direction.
It should be mentioned that a similar figure like 4.11 can still be produced using the numerical velocities which are available at the same time step; the goal of these figures is just to emphasize the difference between numerical and effective velocities.
This simulation was done on a $40 \times 30 \times 1$ grid; the viscosity was 0.01. The computation did not last longer than a quarter of an hour on a common workstation.

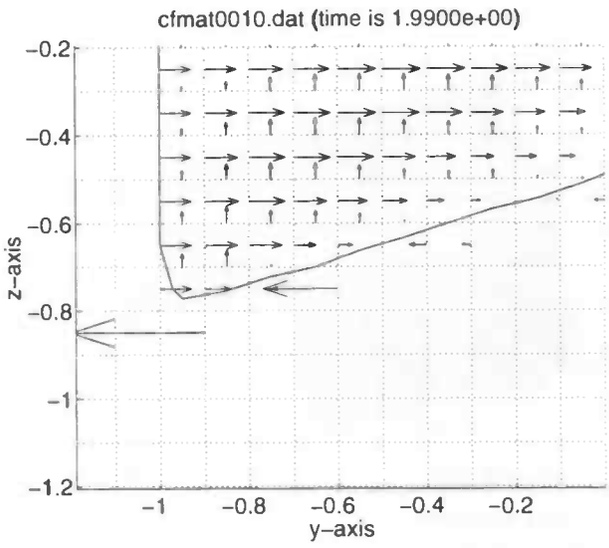


Figure 4.11: *velocities in the old method*

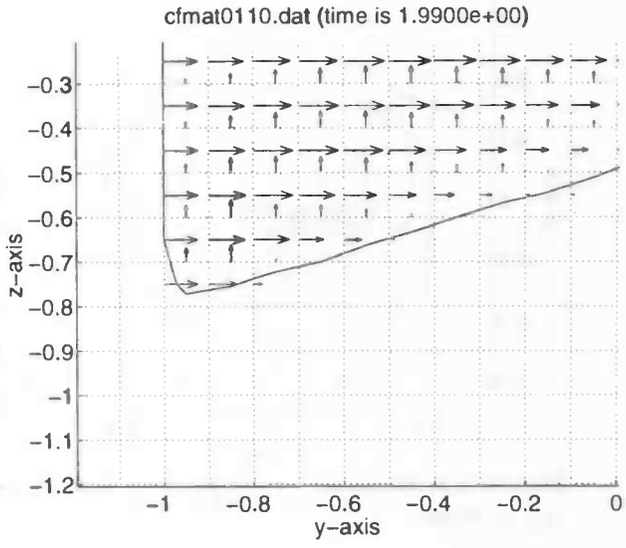


Figure 4.12: *velocities in the new method*

4.4 Flow through a pipe

We consider the motion of fluid in a 2D-cylinder, with inflow at the bottom and outflow at the top. The velocity field along horizontal lines through the pipe will reach a parabolical profile, according to the formula of Poisseuille-Hagen, even if we start with a uniform inflow profile. Using a uniform inflow velocity, the maximum velocity (in the middle) should be $1.5\,u_{in}$.

The physical dimensions are $1\,m \times 8\,m$. The goal is to compare the influence of different boundary cells: along both the vertical walls of the pipe, cells with, for example, $F_b = 1$ (staircase geometry), $F_b = \frac{1}{2}$ (which are **F**-cells) and $0 < F_b < \frac{1}{2}$ ($\mathbf{B}_f$-cells) are placed.
We keep the inflow flux constant at $1\,m^3/s$. This means that in each case the amount of **I**-cells under cells with $F_b = 1$ is measured, and the right value of the inflow velocity of these cells is taken to obtain that total flux.
The grid is in the three cases approximately 15×60. A good method to compare the results is by checking the maximum at the top of the parabola, since the area under it is always the same. The desired parabola is analytically described as $w(y) = 1.5 - 6y^2$ (the pipe is situated in the y, z plane).
The maximum velocity, in the middle, should converge to $1.5\,u_{in}$, in this case 1.5. This convergence is, using this coarse grid, rather slow. However, as we will see later on, these results improve when taking finer grids. In figure 4.15, the evolution of the velocity in a horizontal cross section is shown.
First we keep the resolution approximately fixed at 15×60 cells. This means that the amount of **F**-cells is constant, but the cell apertures of the **B**-cells along the pipe are not.
For several values of F_b the maximum velocity after 10 seconds is measured:

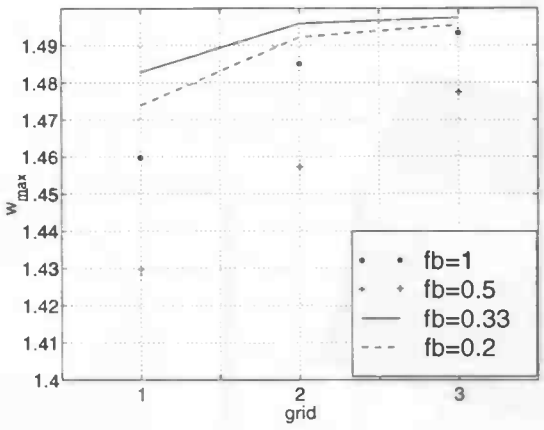


Figure 4.13: *performance on different grids*

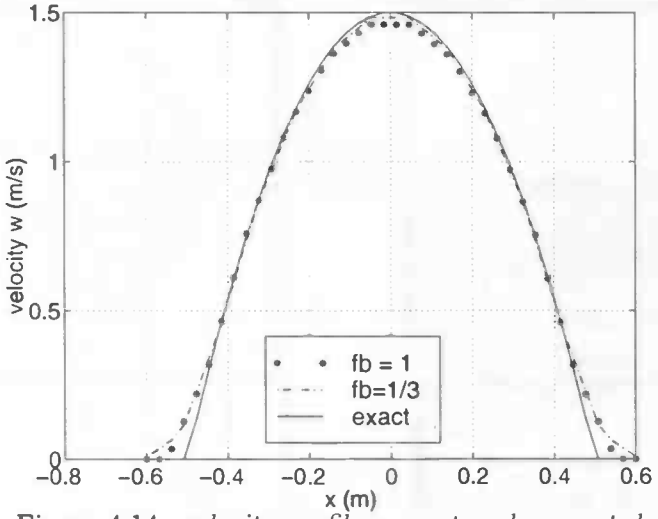


Figure 4.14: *velocity profiles; exact and computed*

$Fb = 0.0$:	1.4760
$Fb = 0.2$:	1.4850
$Fb = 0.25$:	1.4880
$Fb = 0.33$:	1.4910
$Fb = 0.4$:	1.4943

These values differ not too much, and are slightly improving when taking 'larger' cells (with higher values of F_b). In this case, the presence of extra cells, although having the same number of momentum equations, obviously improves the result, in the sense that if the most extremely situated momentum velocities are further from the wall, the parabola gets a better shape.

Now we do a grid refinement study. For different B-cells on both sides, the grid sizes differ. The physical dimensions, as defined in the input file ('*ymax*') and the inflow velocity must be

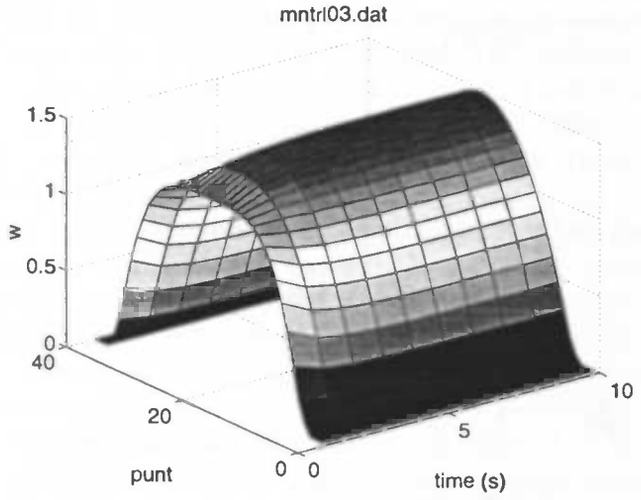


Figure 4.15: *cross section of velocity through a pipe at $z = 7m$*

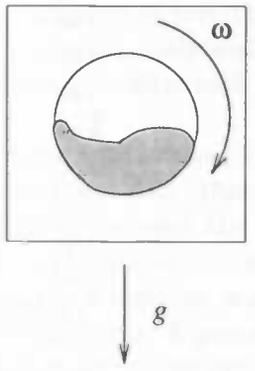


Figure 4.16: *rotating cylinder*

tuned to obtain the same circumstances. The simulations are shown in the following table, together with the maximum velocity in the middle w_{max}.

grid	F_b	$ymax$	w_{in}	w_{max}
10x30	1	0.5	1	1.4597
11x30	1/2	0.55	10/9	1.4297
12x30	1/3	0.5622	10.66/10	1.4828
12x30	1/5	0.5769	10.4/10	1.4739
20x60	1	0.5	1	1.4850
21x60	1/2	0.525	20/19	1.4571
22x60	1/3	0.5324	20.66/20	1.4959
22x60	1/5	0.5392	20.4	1.4922
40x120	1	0.5	1	1.4934
41x120	1/2	0.5125	40/39	1.4774
42x120	1/3	0.5164	40.66/40	1.4975
42x120	1/5	0.5198	40.4/40	1.4956

In figure 4.13 the results are shown for the three main grid sizes: grid 1 (10×30), grid 2 (20×60) and grid 3 (40×120). Figure 4.14 shows the results of two simulations (staircase and $F_b = 1/3$) on the coarsest grid, together with the exact solution.
Note that the velocity remains positive when approaching the wall, since the effective (flux-) velocities are used. However, they are positioned in the middle of the cell. Therefore, and due to interpolation, differences near the wall occur.

The following conclusion can be drawn: B-cells do not dramatically influence the global solution, but local differences due to interpolation of velocities and positions of momentum velocities in relation to the wall can occur.

4.5 Demonstrations

Now two demonstrations follow, giving an indication of the possibilities, computationally as well as in the postprocessing field, of *ComFlo*.

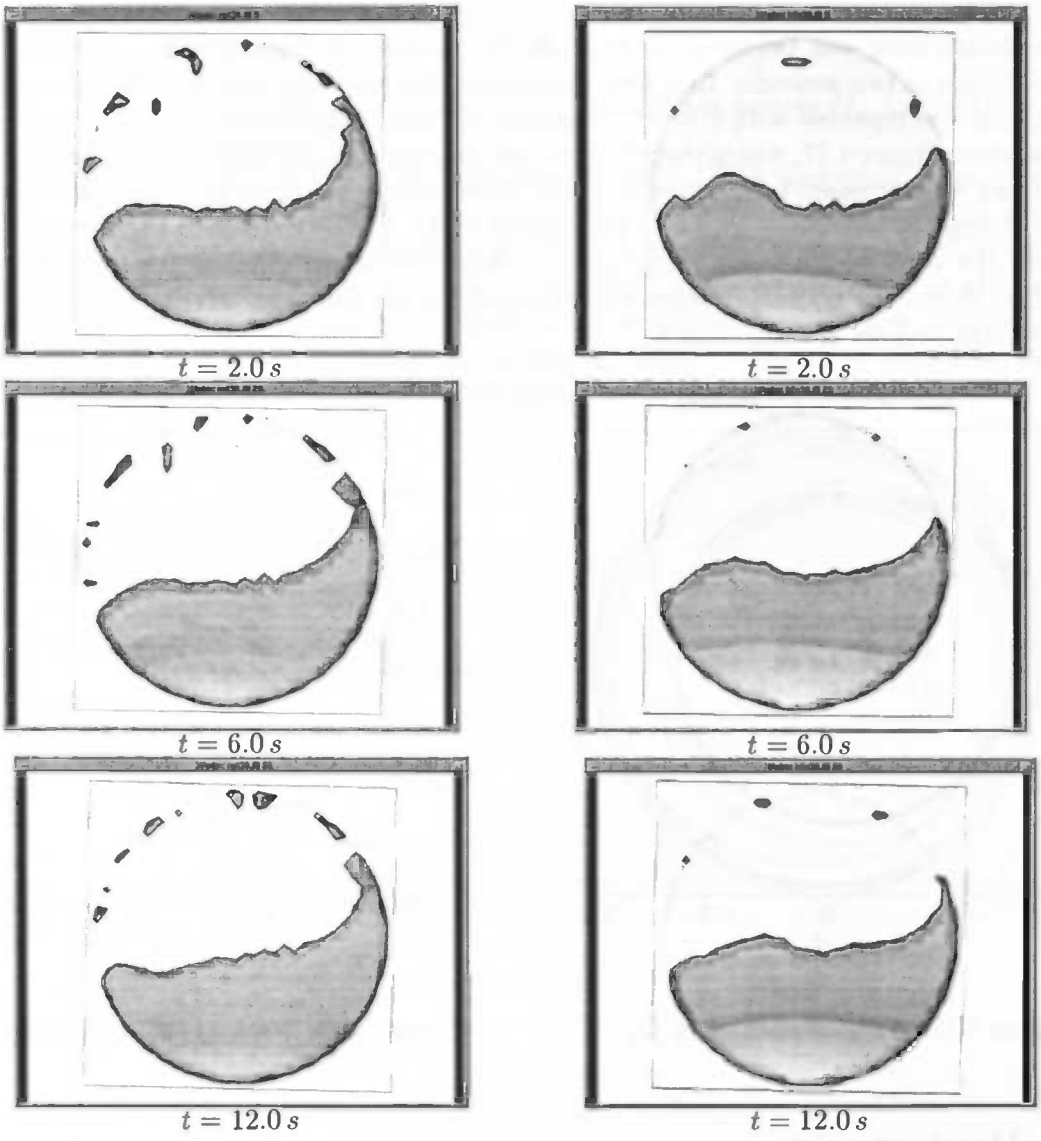


Figure 4.17: *Snapshots of rotating cylinder; left:* $\nu = 10^{-2}$*, right:* $\nu = 10^{-6}$

4.5.1 Rotating cylinder

A first test is the simulation of fluid in a rotating cylinder. [5] described a simulation in which the gravity was orthogonal to the rotation vector, i.e. the gravity had the same direction as the longitudinal axis of the cylinder. In that case, a steady-state solution was reached, which could also analytically be computed by comparing the gravitational and rotational forces.

Now, we take the gravity orthogonal to the longitudinal axis, like a washing machine (see figure 4.16). Initially, the lower half of the cylinder is filled with liquid.

For this simulation, a $30 \times 30 \times 1$ grid was taken, since the problem is two-dimensional. Physically, the radius is $0.5\,m$. It is very difficult to obtain a solution analytically, mainly because that solution cannot be steady-state. However, the configuration after each revolution is fairly similar.

The simulation time was twenty seconds while the rotation vector was set to $(0, 0, \pi)^T$, so each revolution is two seconds. In a first simulation, the viscosity was 0.01; thereafter, the computation was repeated with $\nu = 10^{-6}$. Smaller viscosity numbers decrease the number of drops, as shows figure 4.17, where the left-hand side contain figures of the $\nu = 0.01$ simulation.

Particles were released in the fluid at $t = 0$. Surprisingly, particles starting near the wall seem to change between two tracks: a periodicity of $4\,s$ can be detected. Figures 4.18 and 4.19 show the track of a particle starting $0.1\,m$ above the lowest point of the cylinder. The right figure shows the trajectory when compensated for the rotation [1]. The positions at the end of certain periods are also shown.

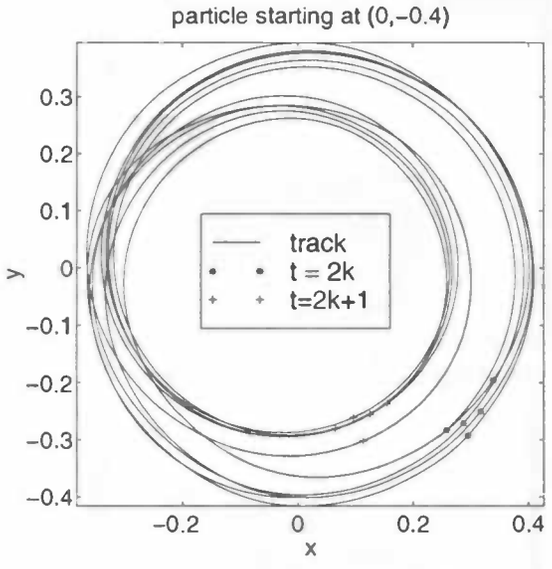


Figure 4.18: *track of particle in* O_g

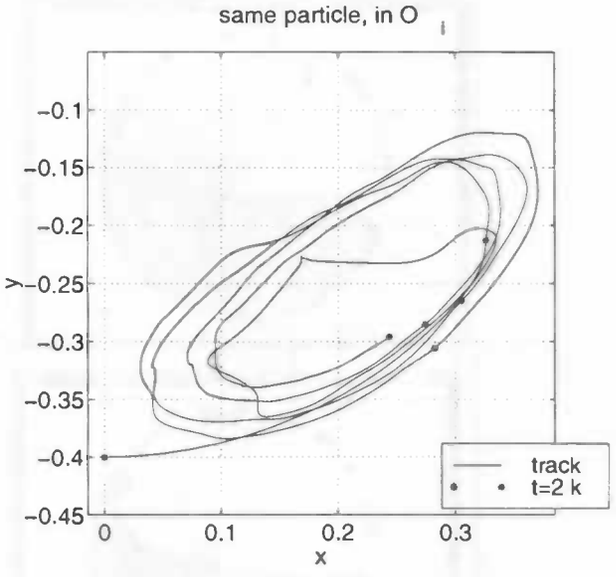


Figure 4.19: *track of particle in* O_i

4.5.2 Marching cube

To express the new visualization properties, a simulation was made of a cube which rolls over one of its edges which is, at that moment, at ground level. The angular velocity each time is $\frac{1}{4}\pi$, meaning that every two seconds one quarter of a turn in a certain direction is made. The movement is described as: two rolls to the west, one to the south, and again one to the west.

If we denote the vertices of the cube by the letters A till H (with $ABCD$ the ground plane), the movement is:

[1] The movement in O_i is obtained by premultiplying the position $\begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$ by the matrix $\begin{pmatrix} \cos(\pi t) & -\sin(\pi t) \\ \sin(\pi t) & \cos(\pi t) \end{pmatrix}$.

Time	Rotation axis	ω
$0-2\,s$	AD	$(0, \frac{\pi}{4}, 0)$
$2-4\,s$	EH	$(0, \frac{\pi}{4}, 0)$
$4-6\,s$	FE	$(\frac{\pi}{4}, 0, 0)$
$6-8\,s$	BF	$(0, 0, -\frac{\pi}{4})$

The rotation axis and ω are described in the coordinate system O_g, i.e. without taking the movements into account.
The viscosity of the fluid was $\nu = 10^{-6}$,the same as water. The computation was performed on a $26 \times 26 \times 26$ grid. Nearly 5900 time cycles were needed to complete the eight seconds; it lasted a few hours on a moderate workstation. The unit vectors in the transformation matrix were slightly altered; after eight seconds, they should again be the same as the standard (inertial) unit vectors. However, they have to be recomputed each cycle according to the algorithm in section 3.6. The difference, in the Euclidean norm, is, nevertheless, less than 10^{-3}. The following snapshots (figure 4.20) indicate the movement of the box.

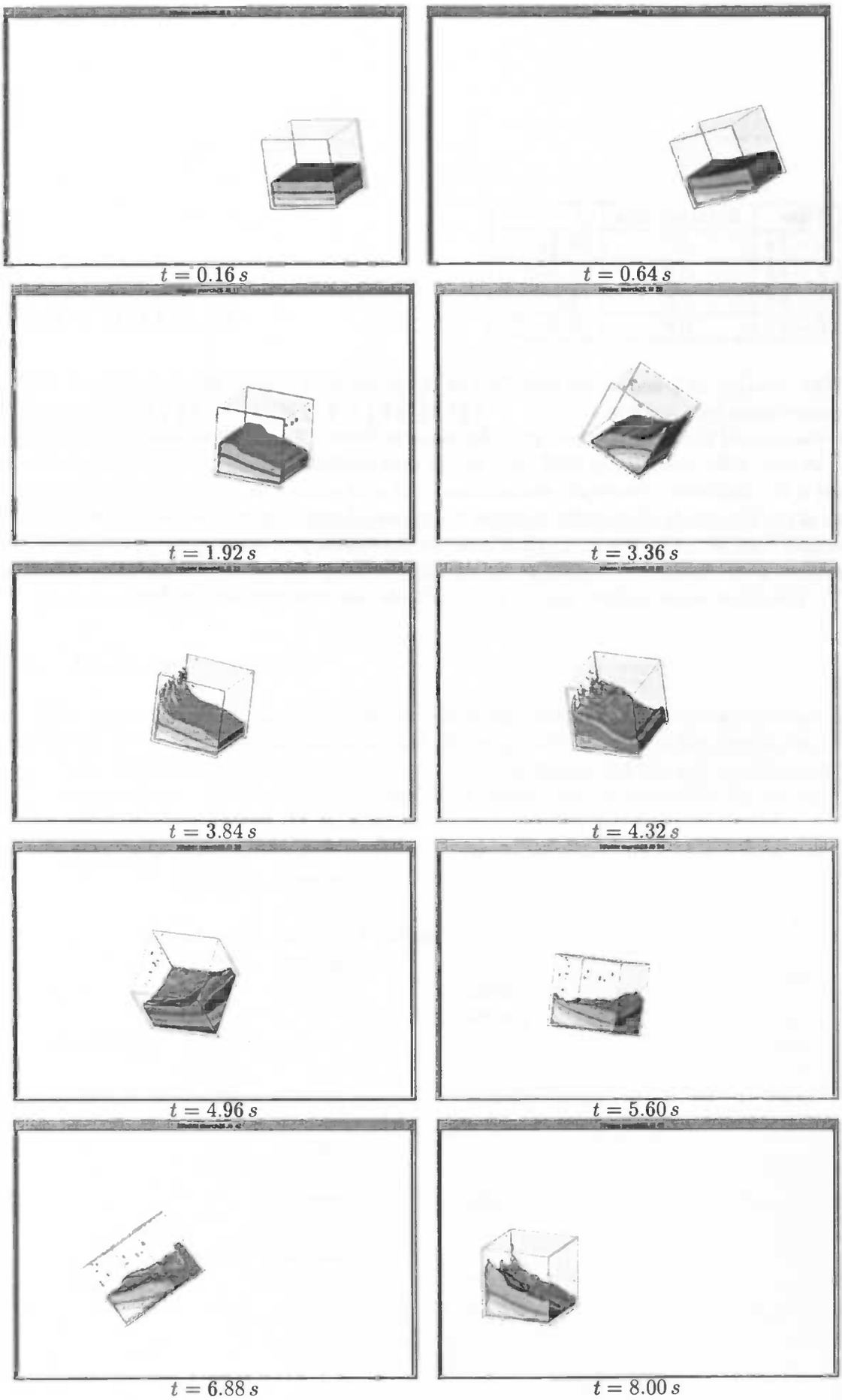


Figure 4.20: *Snapshots of the marching cube*

Chapter 5

Conclusions

In the previous chapters we have shown the features and possibilities of the simulation of free-surface flow in 3D geometries. The solid boundary, previously described with staircase geometries, has been extended towards 'smooth' walls: Apertures in combination with effective velocities near the boundary fulfill the demands of mass conservation and adequate velocities. Together with the possibility of creating arbitrary complex geometries (see [7]), a lot of industrial problems can be handled.
Furthermore on the postprocessing front at lot of improvements have been made. All postprocessing actions, except for AVS, have been combined in a sophisticated MATLAB menu system. Three-dimensional information can be processed with the aid of many state-of-the-art possibilities of AVS, in combination with extensive treatments of forces and movements.

Although a lot has been accomplished now (spring 1998) still a lot can be done. The schedule for the near future is:

- Dynamical interaction: the moving fluid in the geometry excerts forces on the geometry wall. This directly influences the motion of the geometry itself, thus causing a feedback process.
- Moving shape of geometries: Due to the forces as described above, the geometry itself can also change; take for example the pulsing of blood through arteries.
- Local grid refinement, making it possible to pay attention to interesting parts of the flow domain, in first instance near the boundary of Ω.
- Better treatment of the movement of the free surface by making full use of orientation characteristics. Recently, new surface tracking methods (involving normal computation and flux quantifications) have been proposed, some of them also in 3D (see, for example, [8]).
- Extension to a real two-fase flow: The aim is to solve the Navier-Stokes equation in the whole geometry, using different fluid characteristics and a more continuous transition of velocities over the interface.

Appendix A

Program description

The numerical model has been implemented in a FORTRAN program: *ComFlo*. This appendix gives more detailed information about the calling sequence and the several variables and subroutines. This information holds for the summer of 1998; many developments continue to adjust the program structure.

A.1 Calling sequence

The following scheme indicates the order in which the various subroutines are called. The main division is between the initialization and the loop, until the end of the simulation. The loop involves postprocessing routines; they are shown in italics and are not necessary in the actual computation. All choices concerning these aspects can be controlled by setting the right parameters in the input file **Comflo.in**.

```
Initialization  SETPAR   GRID
                BNDLAB   BNDDEF
                         IOLAB
                SETFLD   AUTOSV / SURDEF
                         SURLAB
                         BC                IOBC
                                           VELBC
begin LOOP      INIT
                EFFBC
                TILDE    BDYFRC            TRAFOS
                SOLVEP   COEFL
                         COEFR
                         PRESBC
                         PRESIT            SLAG
                         VELBC
                VFCONV   SURLAB
                         BC                IOBC
                                           VELBC
```

```
            DTADJ
            PRNT
            MATLAB
            COM
            FILLBX
            FLUXBX
            FRCBX
            MNTR     INTERP
            STREAM   INTERP
            AUTOSV
end LOOP
```

A.2 Common block variables

The globally used variables are defined within the *common blocks* structures in Fortran. All but a few uninteresting are listed here with, in the important cases, short descriptions of the variables involved.

- **/APERT/**
  The volume and edge apertures.
  `AX(I,J,K)` Edge aperture between cell (i,j,k) and cell $(i+1,j,k)$
  `AY(I,J,K)` Edge aperture between cell (i,j,k) and cell $(i,j+1,k)$
  `AZ(I,J,K)` Edge aperture between cell (i,j,k) and cell $(i,j,k+1)$
  `FB(I,J,K)` Volume aperture w.r.t. the geometry of cell (i,j,k).
  `FS(I,J,K)`, `FSN(I,J,K)` Volume apertures w.r.t. the free surface of cell (i,j,k), at new and old time level, respectively.

- **/CMPTR/**
  Characteristics of the platform the program runs on .

- **/COEFP/**
  The coefficients in the pressure Poisson equation.
  `DIV(I,J,K)` The right-hand side of the equation in cell (i,j,k)
  `CC(I,J,K)` Coefficient of $p_{i,j,k}$ in left-hand side.
  `CXL(I,J,K)`,`CXR(I,J,K)` Coefficients of $p_{i\pm1,j,k}$ in left-hand side.
  `CYL(I,J,K)`,`CYR(I,J,K)` Coefficients of $p_{i,j\pm1,k}$ in left-hand side.
  `CZL(I,J,K)`,`CZR(I,J,K)` Coefficients of $p_{i,j,k\pm1}$ in left-hand side.

- **/COSYS/**
  Contains variables which keep track of the changing coordinate system.
  `F1,F2,F3` unit vectors of the rotated coordinate system
  `SHISYS(i)` translation vector of the coordinate system

- **/FEAB/**
  Extra variables used for Adams-Bashforth time integration.
  `UNN(I,J,K)`, `VNN(I,J,K)`, `WNN(I,J,K)` These are the old velocities in cell (i,j,k).
  `COEF1,COEF2` These coefficients describe in what way the velocities at the 'intermediate' time level are defined: `WN = COEF1 W + COEF2 WNN`.
  The combination (`COEF1,COEF2`) $= (1,0)$ defines forward Euler, the combination $(1.5,-0.5)$ Adams-Bashforth.

- **/FILLAR/**
  Contains variables for filling degrees in boxes (postprocessing).

- **/FLUID/**
  Characteristics of the fluid.
  `NU` Kinematic viscosity (ν)
  `SIGMA` Surface tension parameter
  `THETA` Contact angle
  `RHO` Density

- **/FLUXAR/**
  Contains variables for fluxes through planes (postprocessing) and fluid transport.
  `XFLUX(I,J,K)`, `YFLUX(I,J,K)`, `ZFLUX(I,J,K)` Flux in each direction leaving cell (i,j,k), as used in **VFCONV**.

- **/FORCE/**
  Characteristics of the external and body forces.
  `AMPLX`, `AMPLY`, `AMPLZ` Amplitude of the oscillation used in the subroutine **BDYFRC**
  `FREQX`, `FREQY`, `FREQZ` Frequencies of this oscillation.
  `GRAVX`, `GRAVY`, `GRAVZ` Gravitational acceleration in each direction.
  `XO`, `YO`, `ZO` Center of rotation
  `OMET(i)` The rotation vector ω in the moving coordinate system.

- **/FRCAR/**
  Contains variables for forces upon the geometry (postprocessing).

- **/GENOUT/**
  Contains infomation about the frequency of postprocessing writing actions.

- **/GRDSTR/**
  Characteristics of an exponentially stretched grid.

- **/GRIDAR/**
  Positioning and distances of the cartesian grid.
  `IMAX, JMAX, KMAX` Number of cells in the three directions
  `X(I), Y(J), Z(K)` Coordinates of grid lines; cell (i,j,k) lies between $x(i-1), x(i), y(j-1), y(j), z(k-1)$ and $z(k)$.
  `DXP(I),DYP(J),DZP(K)` Sizes of each cell: `DXP(I)=X(I)-X(I-1)`
  `DXU(I),DYV(J),DZW(K)` Distances between cell centres:
  `DXU(I) = (DXP(I) + DXP(I+1))*0.5`

- **/LABELS/**
  Cell and velocity labeling.
  `PLABEL(I,J,K)` Cell label based on the geometry only.
  `PLABFS(I,J,K), PLABFSN(I,J,K)` Cell label based on the free surface, at new and old time levels, respectively.
  `ULABEL(I,J,K), ULABFS(I,J,K)` Velocity labels for u based on the geometry and free surface, respectively, between cells (i,j,k) and $(i+1,j,k)$.
  `VLABEL(I,J,K), VLABFS(I,J,K)` Velocity labels for v based on the geometry and free surface, respectively, between cells (i,j,k) and $(i,j+1,k)$.
  `WLABEL(I,J,K), WLABFS(I,J,K)` Velocity labels for w based on the geometry and free surface, respectively, between cells (i,j,k) and $(i,j,k+1)$.

- **/LDSV/**
  Contains information about the autosaving option.

- **/MNTRAR/**
  Contains variables for monitor points, monitor lines and monitor circles: the specified places where velocities and pressure are written to file (postprocessing).

- **/NUMER/**
  Numerical parameters.
  `EPS` Allowed error in pressure Poisson SOR-iteration process.
  `OMSTART` Relaxation factor in SOR-iteration process at $t = 0$
  `OMEGA` Relaxation factor (adjusted to improve convergence).
  `ITER, ITMAX` Amount of SOR-iterations in a time cycle; slow convergence occurs when `ITER` exceeds `ITMAX`.
  `NOM` Counter for amount of time steps with constant `OMEGA`.
  `ITTOT` Cumulative number of iterations.
  `ALPHA` Upwind parameter: `ALPHA`= 1: full upwind .

- **/PHYS/**
  These are the variables characterizing the physics of the system.
  `U(I,J,K),UN(I,J,K)` Velocity u between cells (i,j,k) and $(i+1,j,k)$ at new and old time level, respectively.

V(I,J,K),VN(I,J,K) Velocity v between cells (i,j,k) and $(i,j+1,k)$ at new and old time level, respectively.
W(I,J,K),WN(I,J,K) Velocity w between cells (i,j,k) and $(i,j,k+1)$ at new and old time level, respectively.
P(I,J,K) Pressure p in cell (i,j,k) at new time level.

- **/SPACE/**
  Information about the geometry size and kind of flow domain.
  DOMAIN Type of domain the fluid is set in (cube, cylinder, etc. or a user-specified domain)
  XMIN, XMAX Mesh size in x-direction: XMIN=X(0), XMAX=X(IMAX)
  YMIN, YMAX Mesh size in y-direction: YMIN=Y(0), YMAX=Y(IMAX)
  ZMIN, ZMAX Mesh size in z-direction: ZMIN=Z(0), ZMAX=Z(IMAX)

- **/STRMAR/**
  Contains variables for streamlines (postprocessing).

- **/TIME/**
  Variables about the discrete time.
  T, TMAX Current and maximum simulation time, respectively.
  DT Current time step δt.
  CYCLE Current time cycle.
  CFLON Toggle for variable time steps.
  CFL Current CFL-number
  CFLMIN Low reference CFL-number: if CFL stays under CFLMIN, DT can be doubled.
  CFLMAX High reference CFL-number: if CFL jumps above CFLMAX, DT must be halved.
  CFLCNT Counter for number of cycles with low CFL-numbers

A.3 Subroutines

- **AUTOSV**

  In: T, DT,CYCLE,DTSAVE, NRSAVE,U,V,W,P,FS and all other necessary variables.
  Out: File **auto.sav**
  Description: At equal time intervals (every DTSAVE simulation seconds), the whole state of the computation is saved, which can be restarted after a crash. Moreover, since this routine is called after the regular computation, it can be used to continue after TMAX.

- **AVS**

  In: FB, FS, IMAX, JMAX, KMAX,U, V, W, P, DTAVS, NRAVS, T, DT
  Out: Files **cfavs####.fld** and **cfavs###.dat**

Description: At equal time intervals (after every AVSDT seconds), files for visualizing (3D) data in AVS are created.

- **BC**

In: AX, AY, AZ, FB, IMAX, JMAX, KMAX,
DXP, DYP, DZP, U, ULABEL,V, VLABEL, W, WLABEL
Out: U, V, W
Description: The four solid boundary velocities, namely **BB,FB,SB** and **EB** are computed using the apertures and the boundary conditions (no-slip or free-slip). The velocity labels define the precise computation of these velocities. Finally, the routines **IOBC** and **VELBC** are called to determine the free surface and in- and outflow velocities as well (see section 3.3.4).
Note that boundary velocities use internal velocities from the previous time step; in addition, the amount of fluid is taken into account.

- **BDYFRC**

In: AMPLX, AMPLY, AMPLZ, FREQX, FREQY, FREQZ, DOMAIN, GRAV, T
Out: DQDT, OMET, DOMEDT, GRAV
Description: This routine is called from **TILDE** in order to determine the virtual body force, described by $\frac{\partial q}{\partial t}$ (DQDT), ω (OMET) and $\frac{\partial \omega}{\partial t}$ (DOMEDT). It is also possible to prescribe or adjust the external body force $\boldsymbol{F}$ (GRAV).
If necessary, the subroutine **TRAFOS** is called to compute new unit vectors.

- **BNDDEF**

In: DOMAIN, ILOC, JLOC, KLOC, NRINTP
Out: AX, AY, AZ, FB
Description: For each cell this subroutine is called from **BNDLAB** to compute the apertures in the case that DOMAIN $\neq 0$, i.e. if only a standard geometry is needed. The cell is divided into (NRINTP)^3 points (where each cell wall gets (NRINPT)^2 points) and the fraction of the points in the cell or cell wall that belong to the interior of the geometry is exactly the value of the volume and edge apertures, respectively.

- **BNDLAB**

In: DOMAIN, IMAX, JMAX, KMAX
Out: AX, AY, AZ, PLABEL, ULABEL, VLABEL, WLABEL
Description: First, either **BNDDEF** is called or an input file (**geodata.dat**) is read to obtain the apertures. Second, using these apertures, the cells are labeled **F**,**B** and **X**. Third, the geometry-based velocity labels are set. Fourth, these labels are adjusted to involve free slip and in- and outflow velocity labels.
Fifth, the labels for effective (flux-) velocities, in combination with labels for **B**-cells are determined.

- **COEFL**

  In: AX,AY,AZ, IMAX, JMAX, KMAX, DXP, DYP, DZP, DXU, DYV, DZW, PLABFS, ULABFS, VLABFS, WLABFS
  Out: CC, CXL, CXR, CYL, CYR, CZL, CZR
  Description: The coefficients (namely one central and six neighbour coefficients (city block distanced)) of the matrix on the left-hand side of the Poisson equation are computed in **F**-cells every time cycle.
  These coefficients are computed using the cell distances and the apertures.

- **COEFR**

  In: AX, AY, AZ, IMAX, JMAX, KMAX, DXP, DYP, DZP, PLABFS, U, V, W, DT
  Out: DIV
  Description: The right-hand side of the Poisson equation in **F**-cells is computed every time cycle. Here the apertures are also taken into account. In appropriate cases, the right-hand side (a divergence) is adjusted in order to increase the fluid amount in the cell.

- **DTADJ**

  In: CFLMIN, CFLMAX, DT, U, V, W, DXU, DYV, DZW
  Out: DT , file **dthist.dat**
  Description: Each time step the CFL-value (see section 3.5) is computed. If this value is small enough ($\leq$ CFLMIN) a few consecutive cycles, the time step is doubled. On the other hand, if it is larger than CFLMAX, the time step is immediately halved.
  All information about the changing time steps is written to the file **dthist.dat**.

- **EFFBC**

  In: AX, AY, AZ, FB, FS, IMAX, JMAX, KMAX, PLABEL, PLABFS, DXP, DYP, DZP, UNN, VNN, WNN, ULAB2, VLAB2, WLAB2, PLAB2
  Out: U, V, W
  Description: First, the effective **BB**-velocities are computed. Second, for each **B**-cell a lineair system is solved to determine the normal (**BF**-) velocities. All velocities are acquired from internal velocities of the previous time step.

- **FILLBX**

  In: DXP, DYP, DZP, FB, FS, DTFILL,NRFILL, FCOORD1, NFILLB
  Out: files **fill##.dat**
  Description: This subroutine produces during the computation every DTFILL seconds information about the filling degree of NFILLB boxes specified previously in **SETPAR**.

- **FLUXBX**

  In: DXP,DYP,DZP, XFLUX,YFLUX,ZFLUX,NRFLUX,DTFLUX,NRFLUX,ORIENT
  Out: files **flux##.dat**
  Description: Here the value of the fluxes through planes previously specified in **SETPAR** is written to file every DTFLUX seconds.

- **FRCBX**

  In: DXP, DYP, DZP,FB,FS,P, NFRCB,AX,AY,AZ, NRFRC, DTFRC
  Out: files **frc##.dat**
  Description: Here the forces on the wall in boxes specified in **SETPAR** are written to file every DTFRC simulation seconds.

- **GRID**

  In: IMAX, JMAX, KMAX, XCONC, YCONC, ZCONC, XSTR, YSTR, ZSTR
  Out: X, Y, Z, DXP, DYP, DZP, DXU, DYV, DZW
  Description: A uniform or stretched grid is created. In the latter case, the distances are acquired using a concentration point (XCONC, YCONC, ZCONC) en stretching factors like XSTR.

- **INIT**

  In: COEF1, COEF2, CYCLE, FS, IMAX, JMAX, KMAX, U, V, W
  Out: CYCLE, FSN, U,UN,UNN, V,VN,VNN, W,WN,WNN
  Description: This routine begins a new time cycle. Here the cycle is increased by one, U,V,W are saved in UNN,VNN,WNN and reset; and UN,VN and WN are computed according to the time integration coefficients COEF1 and COEF2.
  After FS is copied to FSN, the subroutine **EFFBC** is called to compute the effective boundary velocities.

- **INTERP**

  In: I,J,K, XPT, YPT, ZPT, DXP,DYP,DZP,DXU,DYV,DZW
  Out: PI, UI, VI, WI
  Description: The pressure and velocities are computed at coordinates (XPT,YPT,ZPT) in cell (i, j, k). This is done by interpolating neighbouring values using the location of the point in the cell.

- **IOBC**

  In: IMAX,JMAX,KMAX, ULABEL,VLABEL,WLABEL
  Out: P, U, V, W
  Description: Boundary conditions with respect to in- and outflow cells are set.

- **IOLAB**

  In: IMAX,JMAX,KMAX,PLABEL
  Out: PLABEL
  Description: In- and outflow cells are labeled by changing boundary cells, i.e. cells with PLABEL=2.

- **LIQPCT**

  In: DXP, DYP, DZP, IMAX,JMAX,KMAX, FB, FS
  Out: LIQUID, VOLUME
  Description: Computes the amount of liquid and writes the liquid percentage to screen.

- **MATLAB**

  In: FB, FS, IMAX,JMAX,KMAX, X,Y,Z, U,V,W,P, NRMATL,DTMATL, T, DT
  Out: files **cfmat####.m**
  Description: Every DTMATL seconds files are created to visualize the geometry, the free surface, the velocity field and the pressure in certain planes in Matlab.

- **MNTR**

  In: U,V,W,P,NMNTRP, NMNTRL, NMNTRC, NRMNTR,,DTMNTR
  Out: files **mntrp##.dat**, **mntrl##.dat**, **mntrc##.dat**
  Description: in **mpoints.dat**, monitor points and monitor lines are defined. This routine writes the pressure and velocities on those positions, as computed by **INTERP** to the specified files every MPDT simulation seconds.

- **PRESBC**

  In: IMAX,JMAX,KMAX, PLABEL, PLABFS, SIGMA, THETA
  Out: P,CXL,CXR,CYL,CYR,CZL,CZR,CC
  Description: Here the free surface condition is applied to S-cells; See section 3.3.4 for further information.
  At the end the coefficients of the pressure Poisson equation are scaled by the central coefficient.

- **PRESIT**

  In: IMAX,JMAX,KMAX, PLABFS, PLABFSN, EPS, OMSTRT, OMEGA,
  CYCLE, ITER, ITMAX, NOM
  Out: P
  Description: This routine solves the pressure Poisson equation using SOR. Starting with OMSTRT, the relaxation parameter ω (OMEGA) is, when possible, changed to obtain the highest convergence ratio. The routine **SLAG** is called for every individual SOR-sweep while a certain error (DELTA) exceeds EPS and the number of iterations ITER stays less than ITMAX. If the second guard is not longer the case, the program terminates because of an apparent non-convergence.

- **PRNT**

  In: IMAX,JMAX,KMAX, U,UN,V,VN,W,WN,PLABEL, PLABFS,DTPRNT,NRPRNT, T,DT
  Out: screen information, file **comflo.out**, file **iter.dat**
  Description: This subroutine handles the more general output to screen and files, which is done every DTPRNT seconds. First, information like time, iterations and changes in velocities and pressure is printed. The file **iter.dat** shows the relation between timesteps and total amount of SOR-iterations versus the simulation time.

- **SETFLD**

  In: IMAX,JMAX,KMAX, LOADQ
  Out: U,V,W, P
  Description: Here the state of the fluid is initialized. If LOADQ equals one (in case of a restart or a continuation) the routine **AUTOSV** is called to obtain the necessary information and a call to **SURLAB** sets the labeling. At this point all data to continue has been acquired.
  Otherwise, in an ordinary startup, **SURDEF** and **SURLAB** are called to set the initial liquid configuration and labels. Also the velocities are initialized (usually to zero) and the atmospheric pressure is defined in the whole grid. At last **BC** is called to obtain the initial boundary conditions.

- **SETPAR**

  In: file **comflo.in**, file **mpoints.dat**
  Out: XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX,IMAX,JMAX,KMAX,TMAX,DT,
  CYCLE, SIGMA,THETA, NU,DOMAIN,AMPLX, AMPLY, AMPLZ,FREQX,FREQY, FREQZ,GRAV,
  ALPHA,EPS,OMSTRT,OMEGA,ITMAX,NOM,CFLON,CFLMIN,CFLMAX
  and all other postprocessing information (more than 50 variables)
  Description: All variables given above are read or initialized. Moreover, all information about the postprocessing (frequency of writing, location of boxes, planes, lines and points) is read. For more information see the input files.

- **SLAG**

  In:IMAX,JMAX,KMAX,DIV,CXL,CXR,CYL,CYR,CZL,CZR,ITER,OMEGA,PLABFS,P
  DELTA,ITER,P
  Out: DELTA, ITER, P
  Description: Here one SOR-sweep is executed using a red-black ordening. The relaxation factor is OMEGA. An adjusted pressure field P is returned, together with the norm DELTA of the difference with the previous P.

- **SOLVEP**

  In: IMAX,JMAX,KMAX,ULABFS,VLABFS,WLABFS,U,V,W,P,DT,DXU,DYV,DZW
  Out: U,V,W,P
  Description: This is the main routine to solve the pressure and obtain new velocities as described in Chapter 3. First, **COEFL** and **COEFR** are called; then the pressure Poisson equation is solved by calling **PRESBC** and **PRESIT**. The new momentum velocities are computed according to formula (3.4).

- **STREAM**

  In:IMAX,JMAX,KMAX,DTSTRM,NRSTRM,NPARTP,NPARTL,NPARTC,DT
  Out: files **partp##.dat**, **partl##.dat**, **partc##.dat**
  Description: This routine produces files containing streamlines. Every cycle, the velocity of each particle (or set of particles, they may initially be set in a circle or on a line) is computed by **INTERP** and the position of the particle is computed using forward Euler. The files are adjusted every DTSTRM simulation seconds.

- **SURDEF**

  In: DOMAIN,FB, IMAX,JMAX,KMAX,X,Y,Z
  Out: FS
  Description: The initial fluid configuration, described by FS, is defined.

- **SURLAB**

  In: IMAX,JMAX,KMAX, PLABEL, PLABFS, FS, ULABEL,VLABEL,WLABEL
  Out: ULABFS, VLABFS, WLABFS, PLABFS, PLABFSN
  Description: Here the free-surface labels (**F**, **S**, and **E** and the various velocity labels) are set at every time step (since the fluid configuration is time-dependent).

- **TILDE**

  In: IMAX,JMAX,KMAX, X,DXP,DXU,Y,DYP,DYV,Z,DZP,DZW,
  GRAV,DQDT,DOMETDT,OMET,DT,UN,VN,WN,ULABFS,VLABFS,WLABFS
  Out: U,V,W
  Description: Here the term $u^n + \delta t \boldsymbol{R}^n$ as described in section 3.3.1 is computed. Since $\boldsymbol{R}^n$ contains also all internal, external and body forces, the routine **BDYFRC** is called to obtain these forces.

- **VELBC**

  In: IMAX,JMAX,KMAX,FS, DXP,DXU,DYP,DYV,DZP,DZW,U,V,W,
  ULABFS,VLABFS,WLABFS
  Out: U,V,W
  Description: In this routine the free-surface velocities **EE** and **SE** are computed using the free-surface labels and the discretized free-surface boundary conditions.

- **VFCONV**

  In: IMAX,JMAX,KMAX, FB,FS,FSN,AX,AY,AZ,DXP,DYP,DZP,
  DT,PLABEL,PLABFS,ULABFS,VLABFS,WLABFS,U,V,W
  Out: FS, XFLUX,YFLUX,ZFLUX
  Description: Here the donor-acceptor algorithm is performed. First the fluxes between **F**-, **S**- and **E**-cells are computed. In these cells the values of F_s are recomputed using those fluxes. Since after that the labeling is obsolete, **SURLAB** is called to set the new labels. Finally, a call to the routine **BC** updates the boundary velocities.

Appendix B

The input file and postprocessing using Matlab

In the following sections we will discuss the main input file, called *Comflo.in*. Here all the characteristics, numerically and physically, are set. Moreover, information about postprocessing must be set in this file. Most of the information produced by the program can be processed using a sophisticated menu system in Matlab.

B.1 Input file

Here an example of the input file is shown:

```
-------------------------------------------------------------------------
dim*    cray
3       0
-------------------------------------------------------------------------
domain  xmin    xmax    ymin    ymax    zmin    zmax    slip*
18      -0.5    0.5    -0.5      0.5    -0.5     0.5     0
-------------------------------------------------------------------------
object  xmin    xmax    ymin    ymax    zmin    zmax    slip*
0       -0.07   0.07    -0.06   0.06    -0.05   0.05    0
-------------------------------------------------------------------------
liqcnf  par1*   par2*   par3*   par4*   par5*   par6*
1       0.0     0.0     0.0     0.0     0.0     0.0
-------------------------------------------------------------------------
rho*    nu      sigma   theta
1.0     1.0E-2  0.0     90.0
-------------------------------------------------------------------------
imax    jmax    kmax    xc      yc      zc      sx      sy      sz
40       40     120     0.0     0.0     0.0     1.0     1.0     1.0
-------------------------------------------------------------------------
eps     omega   itmax   alpha   orde4*  feab1   feab2   nrintp  exact*
1.0E-5  1.3     6000    1.0     0       1.0     0.0     3       0
-------------------------------------------------------------------------
dt      tmax    cfl     cflmin  cflmax
0.01    10.0    1       0.1     0.4
-------------------------------------------------------------------------
```

```
gravx   gravy   gravz   ginrt  finrt
0.0     -0.0    -0.0    0       0
-------------------------------------------------------------------------
amplx   freqx   amply   freqy   amplz   freqz   u0*     v0*     w0*
0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
-------------------------------------------------------------------------
omex    omey    omez    tup*    tdown*  x0      y0      z0
0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
-------------------------------------------------------------------------
load    nsave
0       1
-------------------------------------------------------------------------
npavs   tbavs*  zipavs  npmatl  tbmatl* nprnt   ntcom
10      0.0     0       10      0.0      40     0
-------------------------------------------------------------------------
avs pathname:
/ciwi11/home1/users/csg623/afstu2/comflo/data/
matlab pathname:
data/
-------------------------------------------------------------------------
nfillb  ntfill
1       40
xl      xr      yl      yr      zl      zr
-0.5    0.5     -0.5    0.5     -0.5    0.5
-------------------------------------------------------------------------
nfrcb   ntfrc
1       20
xl      xr      yl      yr      zl      zr
-0.5    0.5     -0.5    0.5     -0.5    0.5
-------------------------------------------------------------------------
nfluxb  ntflux
1       40
xl      xr      yl      yr      zl      zr
-0.5    0.5     -0.5    0.5     0.0     0.0
-------------------------------------------------------------------------
npartp  npartl  npartc  ntpart
0       0       0       10
xpt     ypt     zpt     tstrt                           <- points
xl      xr      yl      yr      zl      zr      tstrt   <- lines
xc      yc      zc      radius  orient  tstrt           <- circles
-------------------------------------------------------------------------
nmntrp  nmntrl  nmntrc  ntmntr
0       3       0       10
xpt     ypt     zpt                                     <- points
xl      xr      yl      yr      zl      zr              <- lines
.0      .0      -0.6    0.6     -3.0    -3.0
.0      .0      -0.6    0.6     0.0     0.0
.0      .0      -0.6    0.6     3.0     3.0
xc      yc      zc      radius  orient  tstrt           <- circles
```

In the first lines, the physical sizes, the grid, the fluid characteristics and the configurations of liquid and geometry are determined. The grid can be exponentially stretched using a concentration point (`xc,yc,zc`) and stretching factors `sx,sy,sz`.
Then numerical parmeters are set: discretization methods in space (upwind, central), and in time (Forward Euler, Adams-Bashforth). For the pressure Poisson iteration, the number of iterations, the relaxation parameter omega and the allowed error eps can be adjusted. Further, CFL checks can be toggled using `cfl`, and, if `cfl` equals one, the limits for doubling and halving the time step are set.
Hereafter, all forces can be determined: gravitation and oscillation, the rotation axis (`omex, omey, omez`) and the rotation centre (`x0,y0,z0`). The oscillation and gravitation can be regarded in the inertial coordinate system or in the moving coordinate system by toggling `ginrt` and `finrt`.
Autosave options are controlled by `load` and `nsave`.
Now the writing frequency of Matlab files (`npavs`), AVS files (`npmat`) and standard information (`nprnt`) is defined.
The rest of the input file is dedicated to secondary data: fillboxes, forceboxes, fluxplanes, streamlines (of individual particles, or particles grouped on lines or circles) and monitor points (or lines, or circles). More details are found in [7].

B.2 Matlab menu system

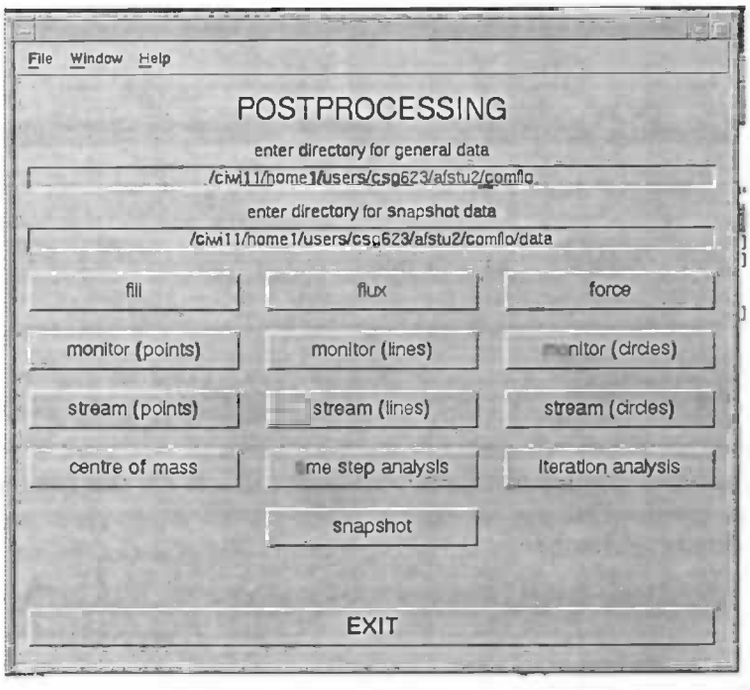


Figure B.1: *Matlab main menu*

The results of the postprocessing tasks as set in the input file can be viewed using the Matlab menu system. The powerful command `uicontrol` enables the creation of several

menus, toggles, radio buttons and so on.
The main menu is divided into several submenus (see figure B.1).

Each of these submenus handles one postprocessing aspect. The simple point-and-click system easily enables beginning users to process all information. Plots of most common combinations of variables can be drawn and printed. Moreover, all variables are available for standard use in Matlab (in the main Matlab window).
A special submenu is the snapshot menu: here the analogons of the AVS datafiles can be viewed. Instead of the AVS files, which contain all important variables on certain time levels, these files contain these variables in the three orthogonal planes which intersect the centre of the geometry. Possible plots which can be drawn are quiver plots of velocities in combination with fluid configurations, or coloured plots of pressure levels. The snapshot submenu is shown in figure B.2.

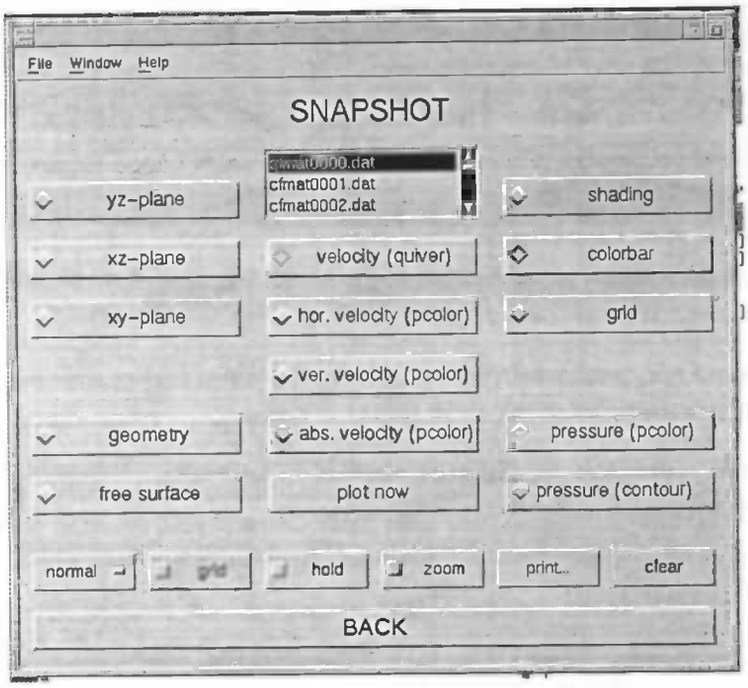


Figure B.2: *Matlab snapshot submenu*

Appendix C

Postprocessing using AVS

AVS, an abbreviation of *Advanced Visualization System* is a package for visualization of large amounts of (3D-) data. Together with MATLAB, which is mainly used for processing derived (2D-) data, AVS handles all postprocessing actions of *ComFlo*.

In the following sections, the interface between *ComFlo* and AVS is explained.

The goals which we would like to reach are:

1. To get a first impression of the fluid configuration, together with pressures and velocities, as soon as new data has been produced, i.e. during the simulation.
2. To be aware of the position (rotation and translation) of the geometry with respect to the starting position, for every data file.
3. To produce a movie, i.e. to obtain a sequence of images, eventually stored in one file, of each desired part of the simulation.

C.1 *ComFlo* module

The three goals are met by using a standard AVS-network, combined with a special module written for this purpose, the *ComFlo* module.
This module, unlike normal modules, directly influences the entire network; it can be considered as a 'master' module. The network itself is not static; it may be changed, saved and loaded by any user, although a few modules of the network may not be removed. In figure C.1 an example of such a network is shown.

The modules "Read field" (for reading the fluid) and "Write Image" (for writing the scene to an *.AVS image) are expected to be found in each network.

A module consists of an initialization part (which is not relevant here) and a computational part, which is executed *each time an input or a parameter of a module is changed.* In the computational part, the value of each parameter can be obtained and changed. A parameter can appear on the dahboard (the left control window) in several forms, constrained by the class to which it belongs (a boolean parameter, for example, can be represented with a toggle

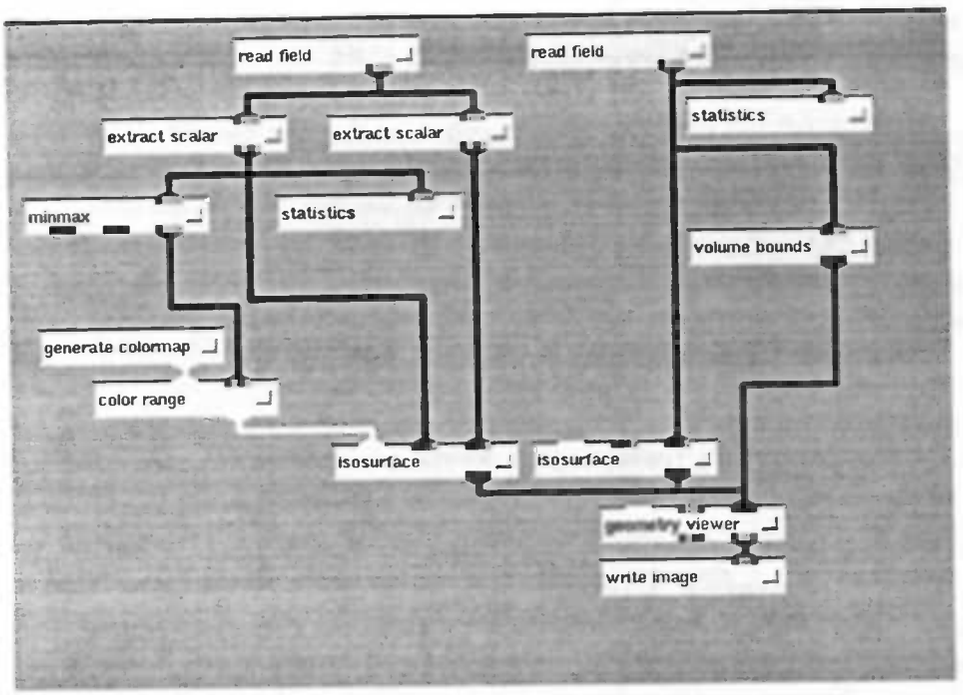


Figure C.1: *Example of an AVS-network*

switch (on/off) or a 'oneshot' push button (default is 'off', pushing the button turns it 'on' just once).).
The *ComFlo* module contains several parameters guiding the movie production process; each function is implemented in the computational part of the module. Some of the parameters influence the whole network in the sense that they can change parameters of other modules. This is accomplished relatively easy with the so-called Command Language Interpreter.

C.2 The Command Language Interpreter

Although all modules are written in Fortran or C, nearly all actions in AVS can be described in a macro language, the Command Language Interpreter (CLI).

To execute AVS in this CLI-mode, type
`unix-prompt > avs -cli`
Now the window in which AVS is started is used as CLI-editing window. For example, reading a network is achieved by typing
`cli > net_read comf.net`

Another possibility is creating a menu system, which enables the user to execute some basic commands. The menu code can be placed in the network (*.net) file. For example:

```
menu "background color" -pulldown
```

```
-add "black" "geom_set_background 0.0 0.0 0.0"
-add "grey"  "geom_set_background 0.5 0.5 0.5"
-add "white" "geom_set_background 1.0 1.0 1.0"
```

adds a pulldown menu consisting of three items, enabling the user to choose a scene background color.

In a module, CLI commands can be used with the FORTRAN-routine **AVScommand(..)** which executes a parameter string as a CLI command. Unfortunately, when used in the computational part of a module, the module does not wait for all the consequences of that CLI-call having been finished; instead, it continues with the next statement (which may be another CLI-call). The best way to avoid problems with the order in which the CLI commands are executed, is:
Construct a (temporary) file containing all desired CLI-commands. Then make one call to the CLI using AVScommand, telling the CLI to run that file as a CLI-script.

C.3 The Geometry Viewer

The geometry Viewer converts the 3D-data field to an image, which is displayed in the Geometry Viewer Window (GVW). The coordinate system in the GVW can be obtained from the ComFlo coordinate system by a rotation of 90 degrees clockwise around the x-axis.
Coordinates range from -5 to 5 by default, making scaling often necessary.
A transformation is set by the CLI-command `geom_set_matrix -mat <mat>` , where `<mat>` is a transposed regular 4×4 transformation matrix.
After each change which influences the GVW, the scene is refreshed by `geom_refresh`.
Once a transformation is executed, additional (concatened) transformations are acquired with `geom_concat_matrix <mat>`. However, to prevent problems due to changing centres of rotation, it is advised to compute the concatenated matrices elsewhere, and present the result to AVS in one `geom_set_matrix` call.

```
sh "uncompress ./comflo/data/cfavs0001.dat "
parm_set ReFi:"Read Field Browser" ./comflo/data/cfavs0001.fld
parm_set WrIm:"Write Image Browser" ./comflo/data/cfim0001.x
sh "mv ./comflo/data/cfim0001.x ./comflo/data/cfim0001.AVS "
sh "convert ./comflo/data/cfim0001.AVS ./comflo/data/cfim0001.pnm "
sh "rm -f ./comflo/data/cfim0001.AVS "
sh "uncompress ./comflo/data/cfavs0002.dat "
parm_set ReFi:"Read Field Browser" ./comflo/data/cfavs0002.fld
parm_set WrIm:"Write Image Browser" ./comflo/data/cfim0002.x
sh "mv ./comflo/data/cfim0002.x ./comflo/data/cfim0002.AVS "
sh "convert ./comflo/data/cfim0002.AVS ./comflo/data/cfim0002.pnm "
sh "rm -f ./comflo/data/cfim0002.AVS "
```

example of a CLI-script

C.4 Making a movie

The module "*ComFlo*" supports the following features for making a movie (see also figure C.2):

- **compress** As each movie frame needs one complete data set, this option saves storage space by immediately compressing the data file when it is no longer necessary, i.e. when another file is read in.
- **begin frame** The number of the first frame of the movie. All AVS-datafiles produced by the program are of the form cfavs####.dat and cfavs####.fld, where #### represents a number (with leading zeroes if necessary) from 0001 to 9999.
- **end frame** The last frame of the desired movie.
- **movement** Toggle for showing the moving geometry in the movie.
- **show network** This toggle shows / hides the network; the user can directly change some additional parameters or change and save the network.
- **start position** This oneshot parameter saves the current position (rotation,scaling and translation) of the geometry in the scene (set by using the mouse, for example) in a 4×4 matrix. The movement of the movie is concatenated with this position. If this parameter has not been used, the movie starts with the normal *ComFlo* position (x-axis to the right, z-axis to above), which is the AVS-position after rotating $-\frac{\pi}{4}$ around the x- axis. Moreover, standard AVS scaling bounds the x- and $y-$ values by ± 5.
- **translation factor** This parameter adjusts the given translation values in order to enable reasonable sizes of the geometry.
- **try movie** While the whole movie making process can take several hours for hundreds of large datafiles, this option is very handy to get an impression of the used area and view angles. When this parameter button is pushed, the currently displayed scene is moved according to the transformation parameters for each frame. Thus, the difference with the real 'MOVIE' parameter is: no new datafiles are read and no images are written.
- **MOVIE.** The actual push button for creating the movie frames. Make sure to push this button only when all precautions (see above) have been made. After the datafiles have been written to *.AVS images, the images are automatically converted to *.PNM files. If the process has finished, an FLI-movie is made by

  unix-prompt> pnm2fli lf <*filename*>

  where lf is a list file which was also automatically created.

Finally, the *ComFlo* module creates a menu, in a separate window (similar to the menu in the Data Viewer application), with the following options:

- set an isosurface level for the fluid.
- set an isosurface level for the geometry.

- determine the fluid representation.
- determine which component colors the fluid.
- determine the background color.
- toggle perspective.
- save current parameters.

It should be noticed that these menu actions can also be accomplished by setting the right parameters or by using the right CLI-commands.

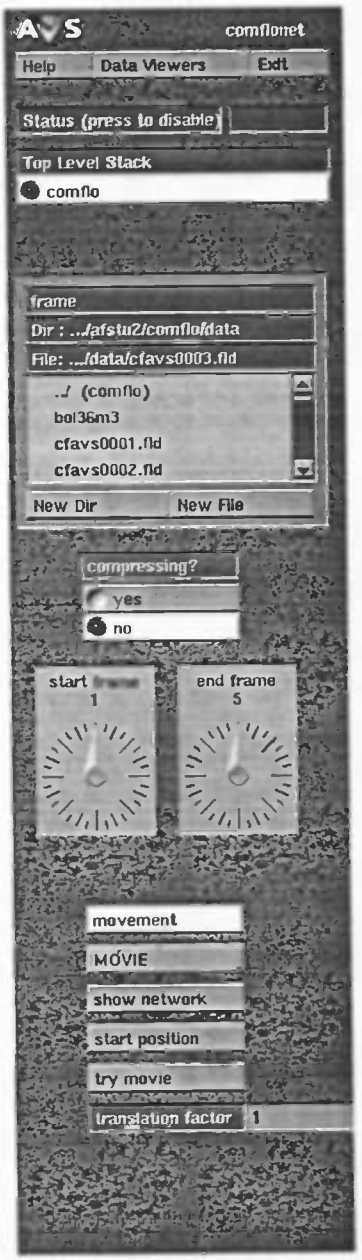


Figure C.2: *dashboard of ComFlo module*

Bibliography

[1] E.F.F Botta (1992) Eindige differentiemethoden, *Lecture notes* RuG.

[2] J. Dijkstra (1997) Simulation of Flow past Complex Geometries using Cartesian Grids, *Master's thesis* RuG.

[3] D. Hearn and P. Baker (1994) Computer Graphics, Prentice Hall.

[4] C.W. Hirt and B.D. Nichols (1981) VOF Method for the Dynamics of Free Boundaries, *J. Comput. Phys*, **39**, 201-225.

[5] J. Gerrits (1996) Fluid Flow in 3-D Complex Geometries – A Cartesian Grid Approach, *Master's thesis* RuG.

[6] J. Gerrits (1996) Three-Dimensional Liquid Sloshing in Complex Geometries, *Master's thesis* RuG.

[7] E. Loots (1997) Free Surface Flow in Three-Dimensional Complex Geometries, *Master's thesis* RuG.

[8] W.J. Rider and D.B. Kothe (1998) Reconstructing Volume Tracking, *J. Comput. Phys*, **141**, 112-152.

[9] G. Trygvasson (1998) Computation of multiphase flows by a finite difference/front tracking method, *Lecture notes on* 29^{th} *Computational fluid dynamics* Von Karman Institute.

[10] A.E.P. Veldman (1994) Numerieke stromingsleer, *Lecture notes* RuG.

[11] D.L. Youngs (1987) An Interface Tracking Method for a 3D Eulerian Hydrodynamics Code, Atomic Weapons Research Establishment.