

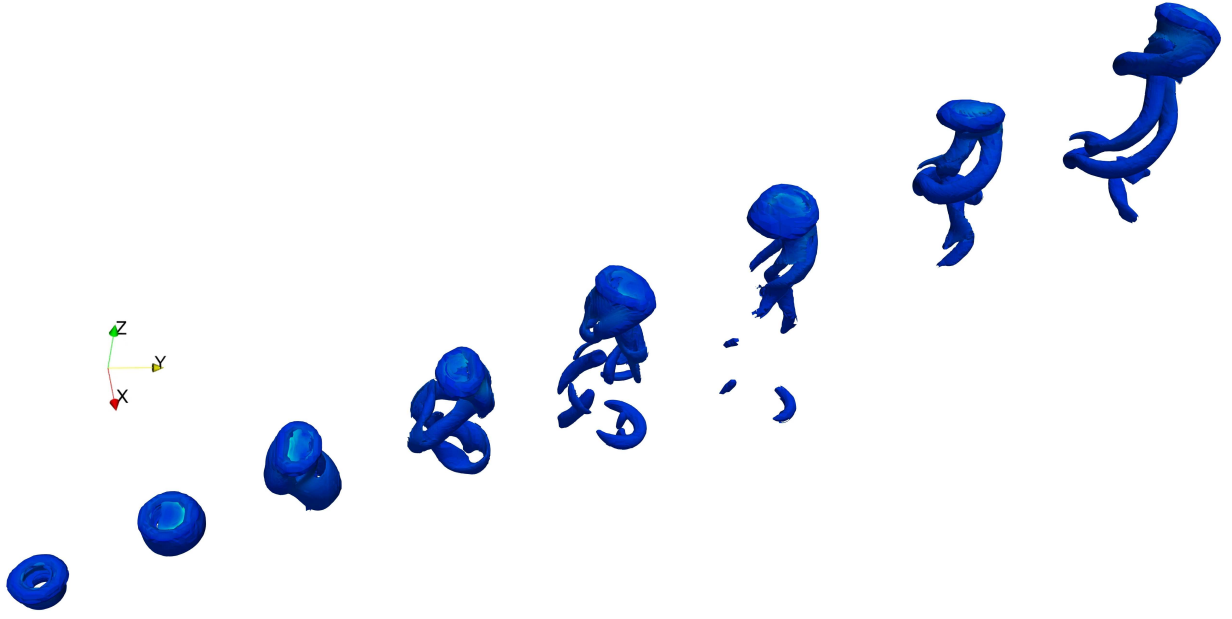


Figure 4.31: Wobbling bubble, $Eo = 10$ and $M = 9.78 \times 10^{-8}$. Time evolution of vortex shedding. Iso-q = 5000.

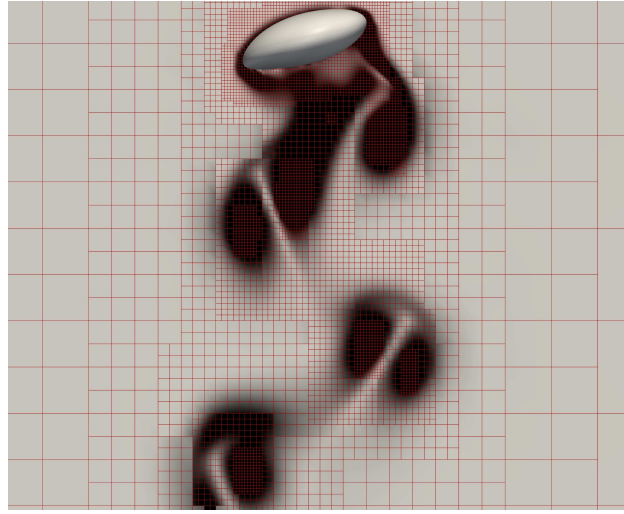


Figure 4.32: Wobbling bubble, $Eo = 10$ and $M = 9.78 \times 10^{-8}$. Example of grid adaptation based on the vorticity gradient.

set to $\lambda_p = 100$. The following physical parameters were used: $\phi = 0.0034m$, $\sigma = 0.01N/m$, $\rho_C = 1000kg/m^3$, $\mu_C = 0.0018Pa \cdot s$ and $g = 9.81m/s^2$ in the $-z$ -direction. The Eulerian domain was set to $(8\phi \times 8\phi \times 80\phi)m$ and six levels of refinement were used.

The pulsing behaviour of the bubble leads to the formation of Kelvin-Helmholtz instabilities in a symmetric varicose mode (LESIEUR, 2008), which is rarely found in nature as well as in numerical simulations. As these instabilities are transported downstream of the bubble, another unstable mode can be identified in the azimuthal direction, as can be seen in Fig. (4.34), which

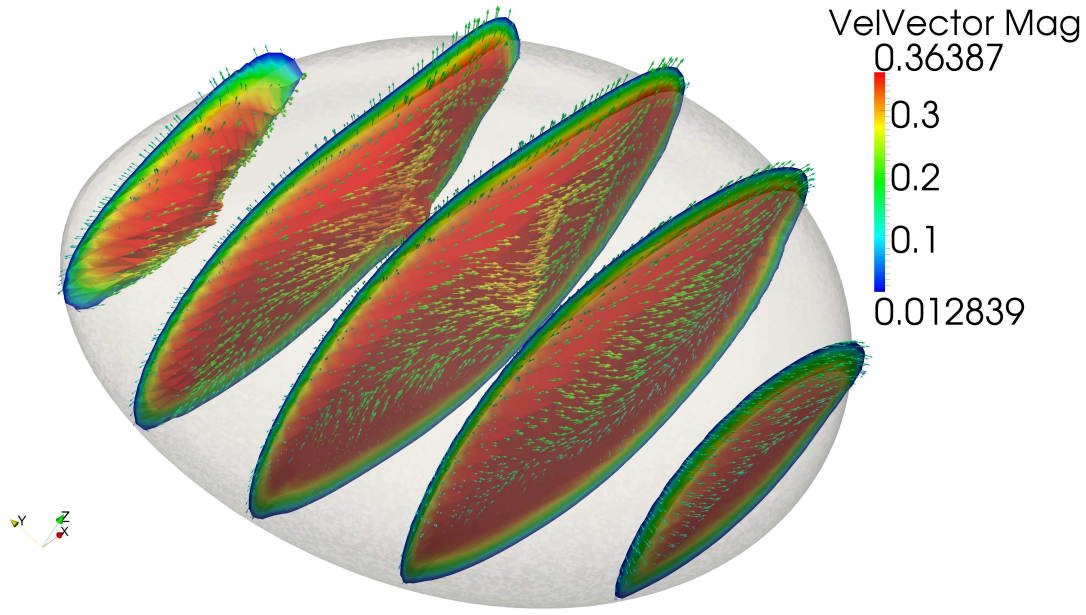


Figure 4.33: Wobbling bubble, $Eo = 10$ and $M = 9.78 \times 10^{-8}$. Velocity field inside the bubble. Planes warped by the velocity vector.

shows an iso-surface of the Q-criterion $Q = 5000$, coloured by the vorticity magnitude. These instabilities give rise to the known hairpin instabilities as shown by the fourth ring downward of the bubble.

Figure (4.35) shows the vortex shedding within a small time interval. The green surfaces show toroidal vortices identified by the Q-criterion with $Q = 5000$, while blue and red surfaces show counter-rotating longitudinal vortices with the z-component of the vorticity. In this case, $\omega_z = \pm 50$ is shown. These counter rotating eddies are non linearly induced by azimuthal modes of the Kelvin-Helmholtz instabilities. The bubble shape at the same time instants is shown at the bottom of the same figure. The pulsing behaviour of this bubble movement is shown in this figure. The varicose instabilities highlighted in the present work are attributed to these pulsations, since they are extremely difficult to sustain experimentally.

The high level of physical details captured by the present simulations enhances the potential of the AMR methodology implemented in the computational code of the present work. This kind of result has a high scientific value for explaining and understanding the process of transition to turbulence in the wakes generated by bubbles in two phase flows.

Figure (4.36) shows unstable modes also inside the bubble. It can be seen that these modes

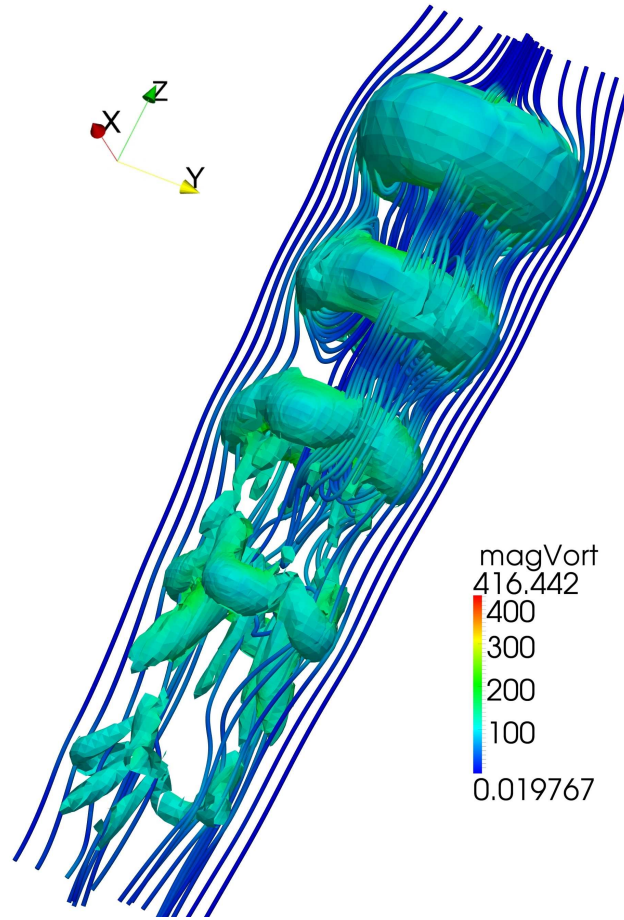


Figure 4.34: Wobbling. $Eo = 3.6$, $M = 2.5 \times 10^{-11}$. Q-criterion = 5000 and streamlines. Colour by vorticity.

are in phase with the modes shown in the Kelvin-Helmholtz instabilities shown in Fig. (4.38), as well as with longitudinal, counter-rotating vortices shown in this same figure.

A comparative analysis between Figures (4.35) and (4.32) yields an interesting interpretation: Notice that the bubble in Fig. (4.35) pulses, but it is not inclined with relation to the z -direction. Figure (4.32), on the other hand, rotates around the y -axis (the axis orthogonal to the cutting plane). As a result, the vortex shedding by the bubble in Fig. (4.35) occurs in the varicose mode, which remains stable while the bubble path remains rectilinear. Fig. (4.32), on the other hand, clearly shows a sinuous mode, instead of varicose, owing to the oscillating bubble path. This kind of result is delivered by what is currently called numerical experimentation. Allied to the laboratory experimentation, this methodology gradually leads to a better understanding of nature and, as a consequence, to control and reproduce natural phenomena.

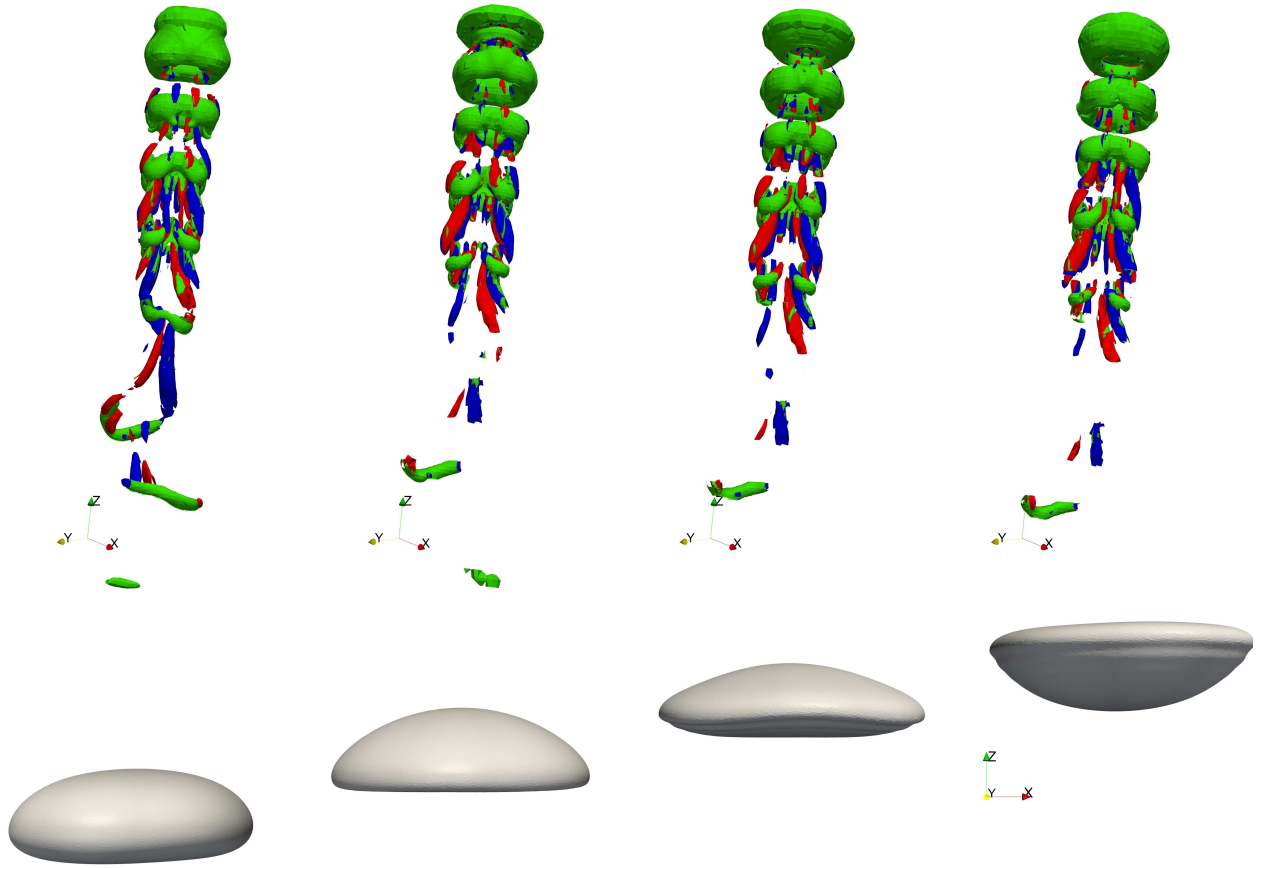


Figure 4.35: Wobbling. $Eo = 3.6$, $M = 2.5 \times 10^{-11}$. Top: Q-criterion $Q = 5000$ in green, z-vorticity in red $\omega_z = +50$ and blue $\omega_z = -50$. Bottom: bubble shape at the respective time instants.

4.2.5 Computational cost of the algorithm

In order to obtain an estimate of the computational cost of the present algorithm, the time spent on each part of the computer code developed in this work was measured during the simulation of the spiral path bubble presented previously. An amount of 5000 time steps were simulated in an Intel Core™i7 PC, 2.4 GHz, with 16 GBRAM 1333MHz running openSUSE Linux 12.1. The time consumption was measured with the Fortran intrinsic `cpu_time` (ADAMS et al., 2009) before and after executing each task. On average, each time step required 121 s of simulation.

The tasks were divided into seven groups, described in Table (4.13).

The first measure taken was the overall time consumed on each task, taking into account the whole simulation. The result is shown in Fig. (4.39). As expected, the most expensive groups were those which are executed at every time step. Notice, however, that although the Eulerian

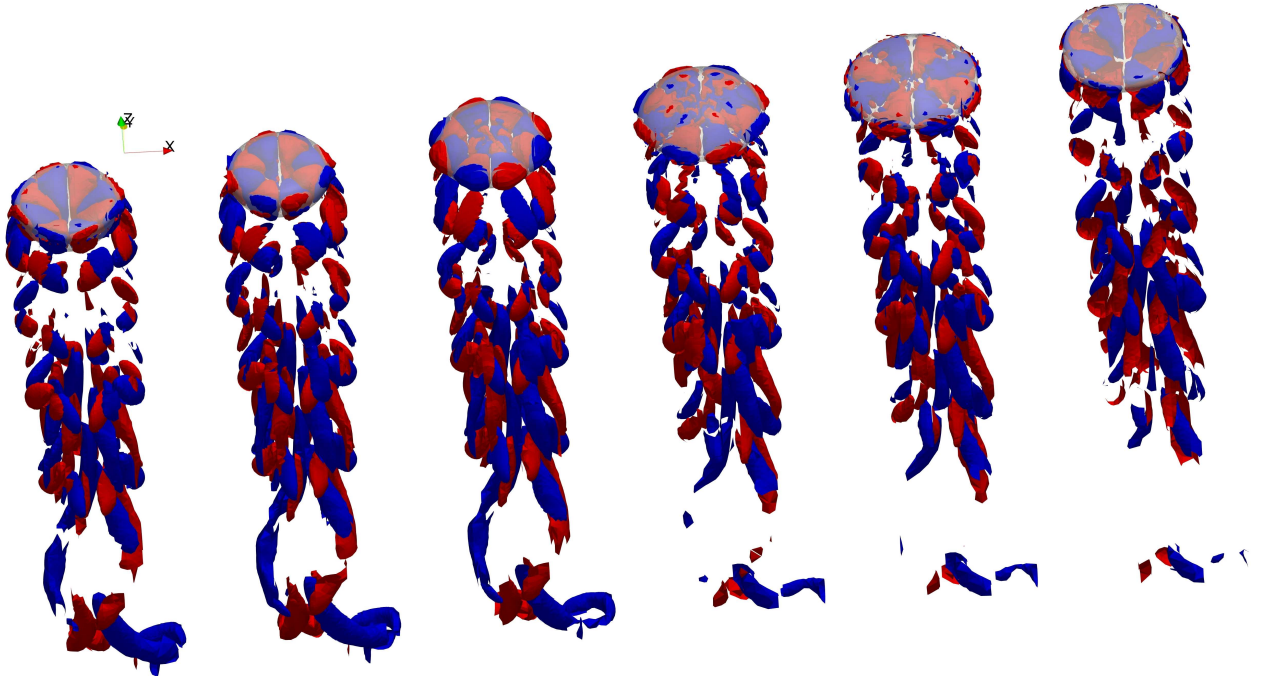


Figure 4.36: Wobbling. $Eo = 3.6$, $M = 2.5 \times 10^{-11}$. Iso surfaces of the z-component of the vorticity. Red=+50, blue = -50.

Table 4.13: Tasks executed at each time step during the simulation, grouped according to the purpose.

Task	Description	Frequency
Pressure	pressure equation solver, pressure correction equation and ghost filling operations	every time step
Velocity	velocity equation solver, velocity update equation and ghost filling operations	every time step
AMR	regridding operation, data transfer from the new grid hierarchy and projection of the velocity field in the new grid onto the divergence-free space	periodically
Lagrangian	surface advection, volume recovery, TSUR3D, surface remesh, indicator function, data transfer between AMR3D and GTS data structures (in both directions)	every time step
Post-Proc	Data writing using HDF5 and legacy-vtk file formats for post processing operations	periodically
Restart Data	Data writing for eventual restart of the case	periodically
Others	calculating the new time step, density and viscosity distribution	every time step

remesh occurs only periodically, it demands approximately the same amount of time as the group related to the velocity solver. The output operations include writing files for post processing (velocity, pressure, density and Lagrangian interface) and also for eventual restart of the case being simulated. While the post processing files contain only data for the velocity, pressure, density and

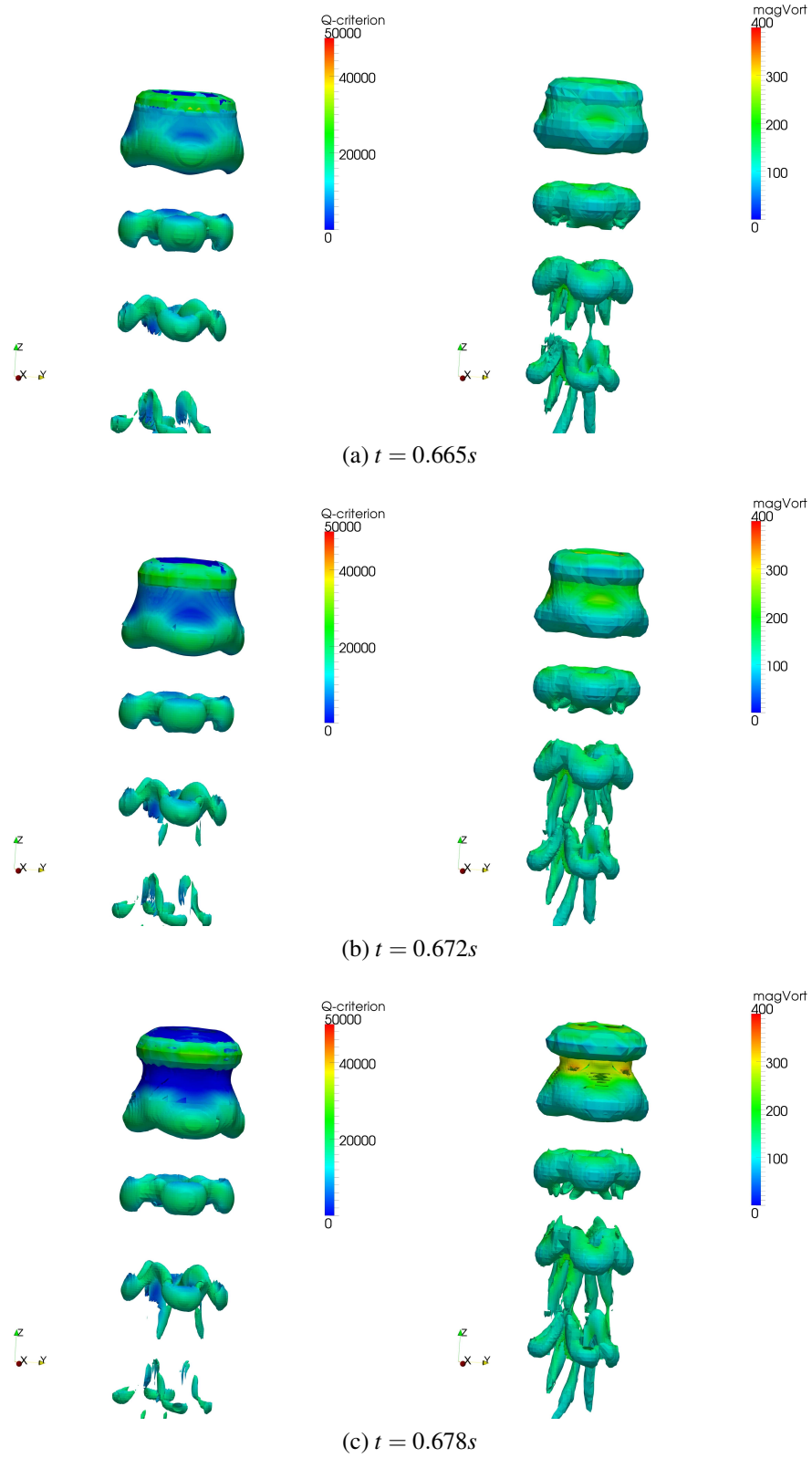


Figure 4.37: Time evolution of the vortex shedding behind the bubble, showing the varicose mode. Left: Iso-surfaces of the vorticity magnitude. Center: Iso-surface of the Q-criterion. Right: bubble shape.

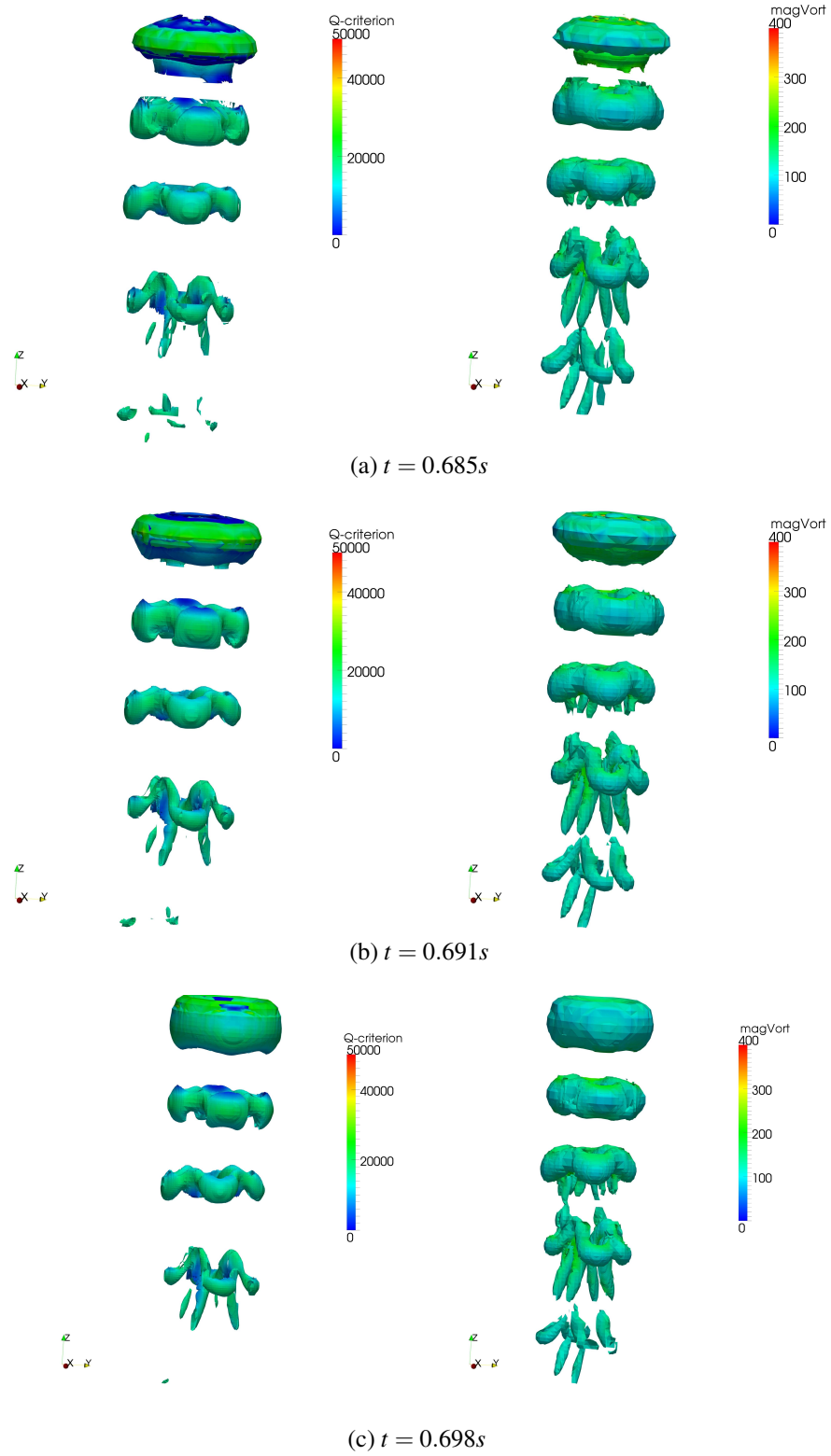


Figure 4.38: Time evolution of the vortex shedding behind the bubble, showing the varicose mode (cont.).

Lagrangian interface at a given time step, the restart files are much more detailed. Data for the last three time steps are written for all physical variables present in the algorithm: velocity, pressure,

density, viscosity and the Lagrangian interface. In the present case, these data were written at every 50 time steps. Nevertheless, Fig. (4.39) shows that less than 0.1% of the total simulation time is spent on these tasks.

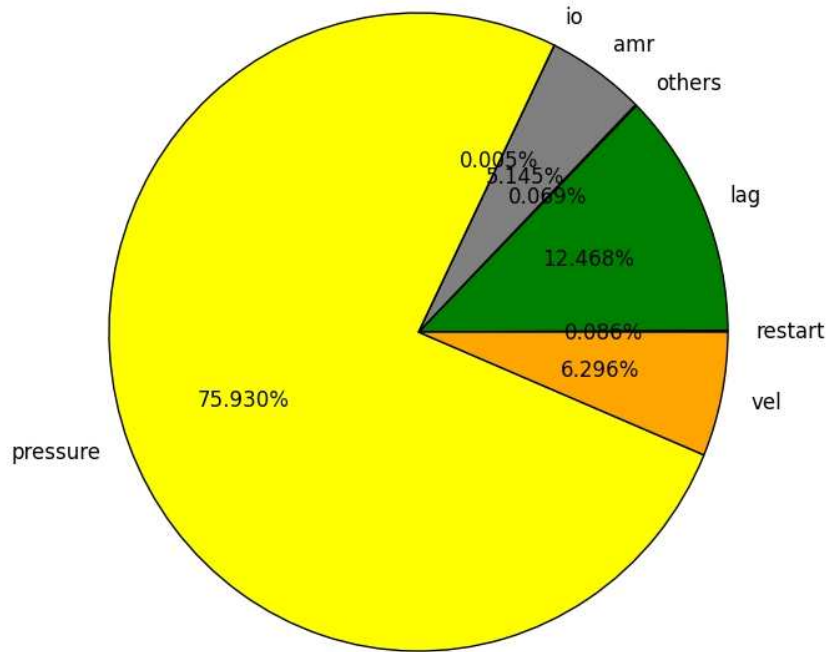


Figure 4.39: Overall time cost of each task group, when taking into account the entire simulation. Pressure: 75.930%; velocity: 6.296%; writing restart files: 0.006%; Lagrangian operations: 12.468%, AMR operations: 5.145%, writing output for post-processing: 0.005%, others: 0.069%.

Figure (4.40) shows the time consumption only when the Eulerian remesh occurs. Notice that, at these specific time steps, most part of the time is spent on AMR-related tasks. As shown in Table (4.13), this group encompasses the tasks for regriding the Eulerian domain, transferring the information from the old mesh to the new one and performing the projection of the velocity field in order to make it solenoidal. Although not reported in the charts, the projection step consumes 99.9% of the time necessary for this task.

The Eulerian remesh is performed under two independent conditions:

1. The Lagrangian interface, which must be completely contained inside the finest level, is less than three cells away from the boundary of this level.
2. The user specifies an interval measured in number of time steps at the end of which the

Eulerian domain will be remeshed.

While the first condition is controlled only by the flow, the second one is specified by the user. In this work, remeshing was forced at every 100 time steps.

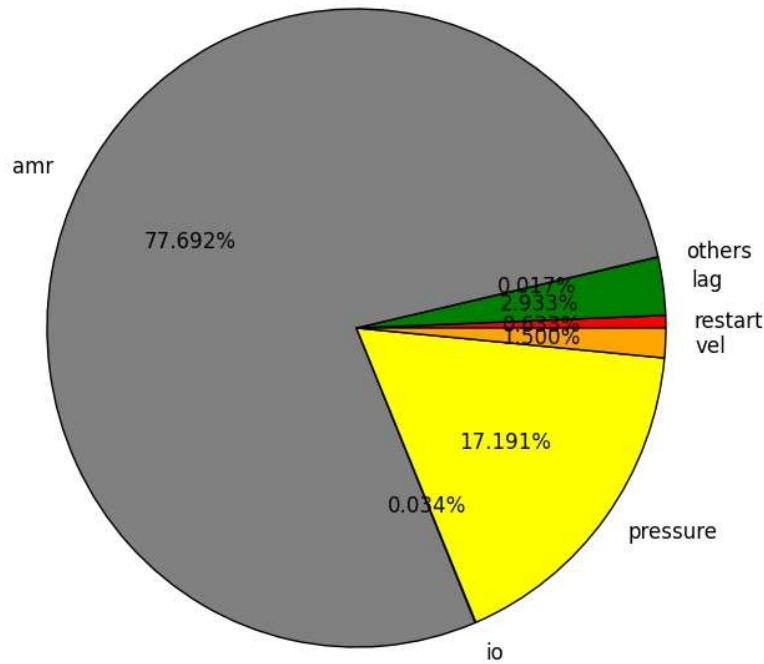


Figure 4.40: Time cost of each task group, when taking into account only the time steps in which the Eulerian domain was remeshed. Pressure: 17.191%; velocity: 1.500%; writing restart files: 0.633%; Lagrangian operations: 2.933%, AMR operations: 77.692%, writing output for post-processing: 0.034%, others: 0.017%.

Finally, Figure (4.41) shows, for the entire simulation, the time spent while executing the Lagrangian operations. The three most expensive tasks in this group are remeshing, the calculation of the Indicator Function and the spread of the interface force on the Eulerian grid. The geometry correction tools, VRA (Volume Recovery Algorithm) and TSUR-3D, demand little time to run, especially VRA, which is executed only if a threshold tolerance is crossed. The task `gts_amr3d` is responsible for transferring the GTS data to the AMR3D structure, using the C-Fortran interface described in section B.2.1. This shows that the only significant overhead generated by such approach is related to memory. Also, the process of assembling the lagrangian maps demands an amount of time which is comparable to the advection of the Lagrangian mesh. However, the elapsed time during the interpolation of the Eulerian velocity field, which is based on the La-

grangian maps, is negligible.

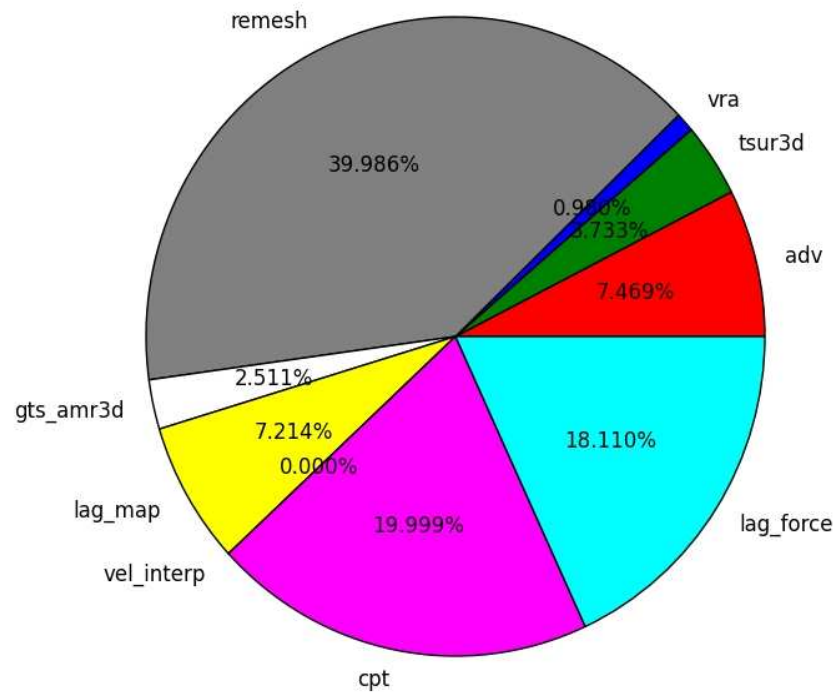


Figure 4.41: Time cost of each task of the Lagrangian group, when taking into account the entire simulation. Surface remesh: 39.986%, transferring information from IB-GTS to AMR3D:2.511; Lagrangian maps: 7.214%, Velocity Interpolation: 0.000%, spreading the interface force on the Eulerian grid: 18.110%, TSUR3D: 3.733%, volume recovery algorithm: 0.980%.

CHAPTER V

Concluding remarks and perspectives

A fully adaptive front-tracking method was presented in this thesis, aiming at the simulation of three-dimensional bubbly flows. The implementation was performed using an in-house built, previously existing framework for the discretization and solution of the Navier-Stokes equations allowing, therefore, that the main focus of this thesis was kept on the implementation of the supporting tools for the Lagrangian framework. In this sense, the GTS library proved to be a very useful tool, since it provided a complete data structure, along with supporting functions for data access, remeshing tools and data output.

The use of an adaptive mesh refinement strategy for the solution of the Navier-Stokes equations enables local detailing of the flow, focusing the grid refinement where it is needed. In fact, a serial algorithm based on uniform grids would be unfeasible if the same grid resolution should be attained. Although the characteristic length of the Lagrangian mesh must be kept constant, the Lagrangian remesh can also be seen as a kind of adaptive strategy, since it sweeps the surface mesh coarsening regions in which the mesh is too refined and refining regions which are too coarse.

Regarding the Lagrangian tools, the remeshing algorithm provided by the GTS library was very useful, since it intrinsically preserves the geometry shape and dimensions, therefore preserving the geometry volume. However, since the interpolation function employed is non-conservative, as demonstrated by the shear flow test, an additional volume recovery algorithm was necessary.

Also, cases simulated during the sensitivity analysis showed the presence of non-physical undulations, a phenomena also reported by other authors (ANNALAND; DEEN; KUIPERS, 2006), (SOUSA et al., 2004). In this sense, the TSUR-3D algorithm yielded good results on preventing such phenomena while still preserving the bubble shape.

As for the density ratio, it has shown little influence on the rise of bubbles at low Reynolds numbers. However, as the Reynolds number increases, its influence grows and interferes on the bubble path, even preventing the transition to an oscillating path in the case of wobbling bubbles.

Regarding the low Reynolds simulations, the terminal bubble shape and Reynolds number were in good agreement when compared to the experimental references. The errors shown in cases simulating skirted bubbles can be attributed to the skirt width at the terminal regime.

The initial bubble shape led to two different behaviours in low and high Re flows. While in the first case all bubbles converged to the same terminal shape and Re number, flows at moderate Re have shown to be very sensitive to the initial bubble shape, leading to toroidal bubbles as the initial bubble aspect increased.

Finally, two cases were simulated for bubbles in the wobbling regime. In the first case, the bubble initially described a zigzag path as reported in the reference work (STENE, 2010). However, using a simulation time larger than the reference work enabled the bubble path to change from zigzag to spiral and then achieve a steady state under this trajectory. In the second case, the bubble ascended describing an almost linear path. The bubble shape oscillated following a pulsatile pattern, and the vortex shedding followed a varicose mode.

The computational framework implemented in the present work consists in the first step for the development of a computational tool for performing numerical experimentation on bubbly flows. While it provides invaluable tools for studying the rise of a single bubble in various flow regimes, some cases still cannot be simulated, due to the lack of a tool providing support for bubble fragmentation. At the present stage, the simulations must be halted when the bubble would start the breakup process.

Such tools are not out of sight at all. The Boolean support supplied by GTS has shown to be a viable tool, even if not the most efficient one, in order to develop such functionality. Also, besides

the embryonic algorithm presented in section A, GTS also provides function for triangulating an iso-surface. Since the indicator function supplies a distance field, it can be used in order to track the breakup site based on the idea that the interpolation and spreading functions require a compact support. The grid spacing is known and also is the minimum number of cells which must be inside the bubble at a given location so that interpolation operations can be done. Therefore, by specifying an iso-value equal to the delta support, the iso-surface triangulation algorithm turns into an automatic tool for finding the breakup site.

Obviously, providing a fragmentation tool automatically requires the ability to handle multiple bubbles, and also the companion ability to perform bubble coalescence. While the algorithm provided in section A can be used, an approach similar to the one proposed for fragmentation also is possible. However, it must be noted that geometrically speaking, both phenomena could be equivalent if a single scalar field were used for more than one bubble.

The simulation of multiple bubbles is possible at the current stage of the computational code, as long as the flow regime prevents the bubble from coalescing. This case is shown in Fig (C.1), which illustrates a simulation replicating the results obtained by Unverdi and Tryggvason (1992). Two bubbles with different diameters are released. The larger bubble is initially below the smaller one, but it is accelerated toward it by its superior buoyancy force. The small bubble is then pushed aside by the larger bubble while both rise without touching each other.

Another crucial step for the long term perspective is the need for a parallel framework in order to implement the above mentioned tools. In this sense, a parallel version of the AMR3D code is under development as part of a Ph.D. thesis in this research group. As for the Lagrangian tools, a parallel natively parallel framework is under development in cooperation with Imperial College London, which at its current stage provides the complete data structure for representing a surface via a triangular mesh. The library, called MFTL (for MultiFlow Triangulation Library) and created by prof. Berend van Wachem, has support for accessing any entity of the mesh and its neighbours, whether they are located at the same processor or not.

Works Cited

- ADAMS, J. C.; BRAINERD, W. S.; HENDRICKSON, R. A.; MAINE, R. E.; MARTIN, J.; SMITH, B. T. *The Fortran 2003 Handbook. The Complete Syntax, Features and Procedures*. USA: Springer Publishing Company, Incorporated, 2009.
- ANNALAND, M. v. S.; DEEN, N.; KUIPERS, J. Numerical simulation of gas bubbles behaviour using a 3d front tracking method. *AiChE J.*, v. 52, p. 99, 2006.
- ASCHER, U. M.; RUUTH, S. J.; WETTON, B. T. Implicit–explicit methods for time–dependent pde’s. *SIAM J. Numer. Anal.*, v. 32, p. 797–823, 1997.
- AZEREDO, D. M. *Simulação Numérica de uma função indicadora de fluidos tridimensional empregando refinamento adaptativo de malhas*. MSc. Dissertation — USP, São Paulo - SP - Brasil, 2007.
- BERGER, M.; COLELLA, P. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, v. 82, p. 64–84, 1989.
- BERGER, M. J. Data structures for adaptive grid generation. *SIAM J. Sci. Stat. Comput.*, v. 7, n. 3, p. 904–916, 1986.
- BERGER, M. J.; JAMESON, A. Automatic adaptive grid refinement for the euler equations. *American Institute of Aeronautics and Astronautics Journal*, v. 23, n. 4, p. 561–568, 1985.
- BERGER, M. J.; OLIGER, J. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, v. 53, n. 3, p. 484–512, 1984.
- BERGER, M. J.; RIGOUTSOS, I. An algorithm for hyperbolic partial differential equations. *IEE Trans. Systems Man and Cybernet*, v. 21, p. 1278–1286, 1991.
- BHAGA, D. *Bubbles in viscous liquids: shapes, wakes and velocities*. (Ph.D. Thesis) — Department of Chemical Engineering, McGill University, Montreal, Canada, September 1976.
- BHAGA, D.; WEBER, M. E. Bubbles in viscous liquids: shapes, wakes and velocities. *Journal of Fluid Mechanics*, v. 105, p. 61–85, 1981.
- BRENNEN, C. E. *Fundamentals of Multiphase Flow*. California: Cambridge University Press, 2005.
- BURTON, J. C.; WALDREP, R.; TABOREK, P. Scaling and instabilities in bubble pinch-off. *Phys. Rev. Lett.*, American Physical Society, v. 94, p. 184502, May 2005.

- CAGNONE, J.; SERMEUS, K.; NADARAJAH, S. K.; LAURENDEAU, E. Implicit multigrid schemes for challenging aerodynamic simulations on block-structured grids. *Computers & Fluids*, v. 44, n. 1, p. 314 – 327, 2011.
- CENICEROS, H.; ROMA, A. A multi-phase flow method with a fast, geometry-based fluid indicator. *Journal of Computational Physics*, v. 205, n. 2, p. 391 – 400, 2005.
- CENICEROS, H. D.; ROMA, A. M.; NETO, A. S.; VILLAR, M. M. A robust, fully adaptive hybrid level-set/front-tracking method for two-phase flows with an accurate surface tension computation. *Commun. Comput. Phys.*, v. 8, n. 1, p. 51–94, jul 2010.
- CLIFT, R.; GRACE, J. R.; WEBER, M. E. *Bubbles, Drops and Particles*. New York: Academic Press, 1978.
- DEEN, N. G.; ANNALAND, M. v. S.; KUIPERS, J. A. M. Multi-scale modeling of dispersed gas-liquid two-phase flow. *Chemical Engineering Science*, v. 59, p. 1853–1861, 2004.
- DURBIN, P.; IACCARINO, G. An approach to local refinement of structured grids. *Journal of Computational Physics*, v. 181, p. 639–653, 2002.
- GAUDLITZ, D.; ADAMS, N. A. Numerical investigation of rising bubble wake and shape variations. *Physics of Fluids*, AIP, v. 21, n. 12, p. 122102, 2009.
- GLIM, J.; ISAACSON, E.; MARCHESIN, D.; MCBRYAN, O. Front tracking for hyperbolic systems. *Adv. Appl. Math.*, v. 2, p. 91–119, 1981.
- GRIFFITH, B. E.; PESKIN, C. S. On the order of accuracy of the immersed boundary method: High order convergence rates for sufficiently smooth problems. *Journal of Computational Physics*, v. 208, n. 1, p. 75–105, 2005.
- HILLS, J. H. Radial non-uniformity of velocity and voidage in a bubble column. *Transactions of the Institution of Chemical Engineering*, v. 52, p. 1–9, 1974.
- HIRT, C. W.; NICHOLS, B. D. Volume of Fluid (VOF) method for the dynamics of free boundaries. *J. Comput. Phys.*, v. 39, p. 201–225, 1981.
- HUA, J.; LOU, J. Numerical simulation of bubble rising in viscous liquid. *Journal of Computational Physics*, v. 222, p. 769–795, 2007.
- IDA, M. An improved unified solver for compressible and incompressible fluids involving free surfaces. part i. convection. *Computer Physics Communications*, v. 132, n. 12, p. 44 – 65, 2000.
- JIN, S.; LEWIS, R. R.; WEST, D. A comparison of algorithms for vertex normal computation. *The Visual Computer*, Springer, v. 21, n. 1-2, p. 71–82, 2005.
- KUMAR, S.; MOSLEMIAN, D.; DUDUKOVIC, M. Gas-holdup measurements in bubble columns using computed tomography. *A.I.Ch.E.*, v. 43, p. 1414–1425, 1997.
- LADD, A. Numerical simulations of particulate suspensions via a discretized boltzmann equation. part 1. theoretical foundation. *Journal of Fluid Mechanics*, v. 271, p. 285–309, 1994.
- LADD, A. Numerical simulations of particulate suspensions via a discretized boltzmann equation. part 2. numerical results. *Journal of Fluid Mechanics*, v. 271, p. 311–339, 1994.

- LAPIN, A.; LUBBERT, A. Numerical simulation of the dynamics of two-phase gas-liquid flows in bubble columns. *Chemical Engineering Science*, v. 49, p. 3661–3674, 1994.
- LEMONNIER, H.; JAMET, D.; LEBAGUE, O. *Validation of advanced computational methods for multiphase flow*. New York: Begell House, Inc., 2005. (Series in thermal & fluid physics & engineering).
- LESIEUR, M. *Turbulence in Fluids*. [S.l.]: Springer, 2008. (Fluid Mechanics and its Applications). ISBN 9781402064357.
- LINDSTROM, P.; TURK, G. Fast and memory efficient polygonal simplification. In: *Proceedings of IEEE Visualization '98*. [S.l.: s.n.], 1998. p. 279–286.
- LINDSTROM, P.; TURK, G. Evaluation of memoryless simplification. *IEEE Trans. Vis. Comp. Graph.*, v. 5, p. 98–115, 1999.
- MAUCH, S. *Efficient algorithms for solving static Hamilton-Jacobi equations*. (Ph.D. Thesis) — Caltech, Pasadena CA, 2003.
- MUDDE, R. F. Gravity-driven bubble flows. *Annual Reviews of Fluid Mechanics*, n. 37, p. 393–423, 2005.
- NOS, R. L. *Simulações de escoamentos tridimensionais bifásicos empregando métodos adaptativos e modelos de campo de fase*. (Ph.D. Thesis) — Universidade de São Paulo, São Paulo - SP - Brasil, 2007.
- PESKIN, C. S. Numerical analysis of blood flow in the heart. *Journal of Computational Physics*, v. 25, n. 1, p. 220–252, 1977.
- PESKIN, C. S. The immersed boundary method. *Acta Numerica*, v. 11, p. 479–517, 2002.
- POPINET, S. *GTS Library Reference Manual*. [S.l.], 2000.
- POPINET, S. Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries. *Journal of Computational Physics*, v. 190, n. 2, p. 572 – 600, 2003.
- PORTELA, L.; OLIEMANS, R. Possibilities and limitations of computer simulations of industrial turbulent dispersed multiphase flows. *Flow, Turbulence and Combustion*, Springer Netherlands, v. 77, p. 381–403, 2006. ISSN 1386-6184.
- QUAN, S. Simulations of multiphase flows with multiple length scales using moving mesh interface tracking with adaptive meshing. *Journal of Computational Physics*, v. 230, n. 13, p. 5430 – 5448, 2011.
- QUAN, S.; LOU, J.; SCHMIDT, D. P. Modeling merging and breakup in the moving mesh interface tracking method for multiphase flow simulations. *Journal of Computational Physics*, v. 228, n. 7, p. 2660 – 2675, 2009.
- QUAN, S.; SCHMIDT, D. P. A moving mesh interface tracking method for 3d incompressible two-phase flows. *Journal of Computational Physics*, v. 221, n. 2, p. 761 – 780, 2007.
- RENAUD, T.; COSTES, M.; PRON, S. Computation of goahead configuration with chimera assembly. *Aerospace Science and Technology*, v. 19, n. 1, p. 50 – 57, 2012.

RICHMYER, R.; MORTON, K. *Differential methods for initial-value problems*. [S.l.]: Interscience publisher, John Wiley & Sons, 1967.

RIDER, W.; KOTHE, D. Stretching and tearing interface tracking methods. In: *12th AIAA Computational Fluid Dynamics Conference and Open Forum*. San Diego - CA - USA: [s.n.], 1995.

ROACHE, P. J. *Verification and Validation in Computational Science and Engineering*. [S.l.]: Hermosa Publishers, 1998.

ROMA, A. M. *A Multilevel self adaptive version of the immersed boundary method*. (Ph.D. Thesis) — New York University, 1996.

ROMA, A. M.; S., P. C.; J., B. M. An adaptive version of the immersed boundary method. *J. Comput. Phys*, v. 153, p. 509–534, 1999.

RUZICKA, M.; DRAHO, J.; FIALOV, M.; THOMAS, N. Effect of bubble column dimensions on flow regime transition. *Chemical Engineering Science*, v. 56, n. 2122, p. 6117 – 6124, 2001. Proceedings of the 5th International Conference on Gas-Liquid and Gas-Liquid-Solid Reactor Engineering.

RUZICKA, M.; ZAHRADNIK, J.; DRAHOS, J.; THOMAS, N. Homogeneous - heterogeneous regime transition in bubble columns. *Chemical Engineering Science*, v. 56, n. 15, p. 4609 – 4626, 2001.

SAMET, H. *Applications of Spatial Data Structures*. Reading, MA, USA: Addison-Wesley, 1989.

SCARDOVELLI, R.; ZALESKI, S. Direct numerical simulation of free-surface and interfacial flow. *Annu. Rev. Fluid Mech*, v. 31, p. 567–603, 1999.

SETHIAN, J. A.; SMEREKA, P. Level set methods for fluid interfaces. *Annual Review of Fluid Mechanics*, v. 35, n. 1, p. 341–372, 2003.

SHIN, S.; JURIC, D. Modeling Three-Dimensional multiphase flow using a level contour reconstruction method for Front Tracking without connectivity. *Journal of Computational Physics*, v. 180, p. 427–470, 2002.

SILVA, H. G.; VILLAR, M. M. Capítulo 2: Verificação e validação de códigos computacionais. In: MANSUR, S.; VIEIRA, E.; NETO, A. (Ed.). *Turbulência*. [S.l.]: ABCM - Associação Brasileira de Ciências Mecânicas, 2010, (Coleção Cadernos de Turbulência, v. 7). p. 51 – 94.

SINGH, R.; N'DRI, N.; UZGOREN, E.; SHYY, W.; GARBEY, M. Three-dimensional adaptive, cartesian grid method for multiphase flow computations. In: *43rd Aerospace Sciences Meeting & Exhibit*. [S.l.: s.n.], 2005. p. Paper No. 2005–1389.

SINGH, R.; SHYY, W. Three-dimensional adaptive grid computation with conservative, marker-based tracking for interfacial fluid dynamics. In: *44rd Aerospace Sciences Meeting*. [S.l.: s.n.], 2006.

SINGH, R.; SHYY, W. Three-dimensional adaptive cartesian grid method with conservative interface restructuring and reconstruction. *Journal of Computational Physics*, v. 224, p. 150–167, 2007.

SOUSA, F.; MANGIAVACCHI, N.; NONATO, L.; CASTELO, A.; TOMÉ, M.; FERREIRA, V.; CUMINATO, J.; MCKEE, S. A front-tracking/front-capturing method for the simulation of 3d multi-fluid flows with free surfaces. *Journal of Computational Physics*, v. 198, n. 2, p. 469–499, 2004. ISSN 0021-9991.

STENE, J. F. *Numerical Simulation of Interfacial and Multiphase Flows using the Front Tracking Method*. (Ph.D. Thesis) — National University of Singapore, 2010.

SUSSMAN, M.; ALMGREN, A. S.; BELL, J. B.; COLELLA, P.; HOWELL, L. H.; WELCOME, M. L. An adaptive Level Set approach for incompressible two-phase flows. *J. Comput. Phys.*, v. 148, p. 81–124, 1999.

SUSSMAN, M.; SMEREKA, P.; OSHER, S. A level set approach for computing solutions in incompressible two-phase flow. *J. Comput. Phys.*, v. 114, p. 146–159, 1994.

TRYGGVASON, G.; BUNNER, B.; ESMAEELI, A.; JURIC, D.; AL-RAWAHI, N.; TAUBER, W.; HAN, J.; NAS, S.; JAN, Y. J. A front-tracking method for the computations of multiphase flow. *Journal of Computational Physics*, v. 169, p. 708–759, 2001.

TRYGGVASON, G.; ESMAEELI, A.; LU, J.; BISWAS, S. Direct numerical simulations of gas/liquid multiphase flows. *Fluid Dynamics Research*, v. 38, p. 660–681, 2006.

TRYGGVASON, G.; SCARDOVELLI, R.; ZALESKI, S. *Direct Numerical Simulations of Gas-Liquid Multiphase Flows*. Cambridge, UK: Cambridge University Press, 2011.

UNVERDI, S. A.; TRYGGVASON, G. A front-tracking method for viscous, incompressible, multi-fluid flows. *Journal of Computational Physics*, v. 100, p. 25–37, 1992.

VILLAR, M. M. *Análise Numérica Detalhada de Escoamentos Bifásicos Bidimensionais*. (Ph.D. Thesis) — Universidade Federal de Uberlândia, Uberlândia, Uberlândia - MG - Brasil, 2007.

VRIES, A. W. G. de. *Path and wake of a Rising Bubble*. (Ph.D. Thesis) — University of Twente, Enschede, February 2001.

WARSITO; OHKAWA, M.; KAWATA, N.; UCHIDA, S. Cross-sectional distributions of gas and solid holdups in slurry bubble column investigated by ultrasonic computed tomography. *Chemical Engineering Science*, v. 54, n. 21, p. 4711 – 4728, 1999.

WELCH, J.; HARLOW, F.; SHANNON, J.; DALY, B. *The MAC method: A computing technique for solving viscous incompressible transient fluid flow problems involving free surfaces*. Emerville, CA - USA, 1965.

WHITE, F. *Viscous Fluid Flow*. New York: McGraw-Hill, 1991.

YEOH, G.; TU, J. *Computational Techniques for Multiphase Flows*. [S.l.]: Butterworth-Heinemann, 2009. (Butterworth Heinemann).

YOKOI, K. A numerical method for free-surface flows and its application to droplet impact on a thin liquid layer. *J. Sci. Comput.*, v. 35, p. 372–396, 2008.

APPENDIX A – Computational Geometry tools

This section presents some preliminary studies about using GTS tools to implement *ad hoc* coalescence and fragmentation algorithms. The studies were carried out in a preliminary stage of this project, in order to assess the ability of the GTS library on providing all the necessary tools for the geometry support of the front-tracking algorithm.

A.1 Coalescence

A coalescence algorithm was developed using the tools provided by GTS for performing boolean operations like intersection, union and difference between surfaces. The algorithm is based on the idea of forcing the intersection between two surfaces and then moving them back to their original positions, creating an intersection. This intersection generates new vertices, edges and elements in the region. Then, both surfaces are taken back to their original position, but the vertices generated by the intersection operation do not move. This generates a connection between both surfaces, performing therefore the coalescence between them. Notice, however, that the mesh in this new region is anything but good, and a remeshing operation is necessary. Algorithm (3) and Fig. (A.1) illustrates it graphically.

Besides physical criteria required for the coalescence to occur, it is necessary to monitor the distance between surfaces. This is done using the kd-tree structure provided by GTS, which yields an efficient, hierarchical way of computing the minimal distance between the two surfaces. An example of the coupling between these two algorithms is shown in Figs. (A.2) and (A.3). Figure (A.2) shows a sphere approaching a dimpled hemisphere and stopping when the surfaces touch each other. The merging algorithm is then applied and the result is shown in Fig. (A.3). These figures show the result of the merging operation, before the application of the remeshing algorithm.

Algorithm 3 Surface coalescence by forced intersection

Input:

- Two surfaces S_1 and S_2
- d distance between S_1 and S_2 .

1. Find $P_1 \in S_1, P_2 \in S_2$, at the smaller distance line between S_1 and S_2
 2. Compute $\mathbf{v} = P_1 P_2$.
 3. Compute $\mathbf{v}_1 = 0.5 * \mathbf{v} + \alpha$ and $\mathbf{v}_2 = -\mathbf{v}_1, \alpha > 0$.
 4. Move S_1 according to $\mathbf{v}_1$ and S_2 according to $\mathbf{v}_2$, forcing intersection
 5. Find the nodes located along the intersection line of S_1 and S_2 .
 6. Move S_1 and S_2 back to their initial position, all nodes but those located at the intersection line.
 7. Perform the Boolean union of S_1 and S_2
-

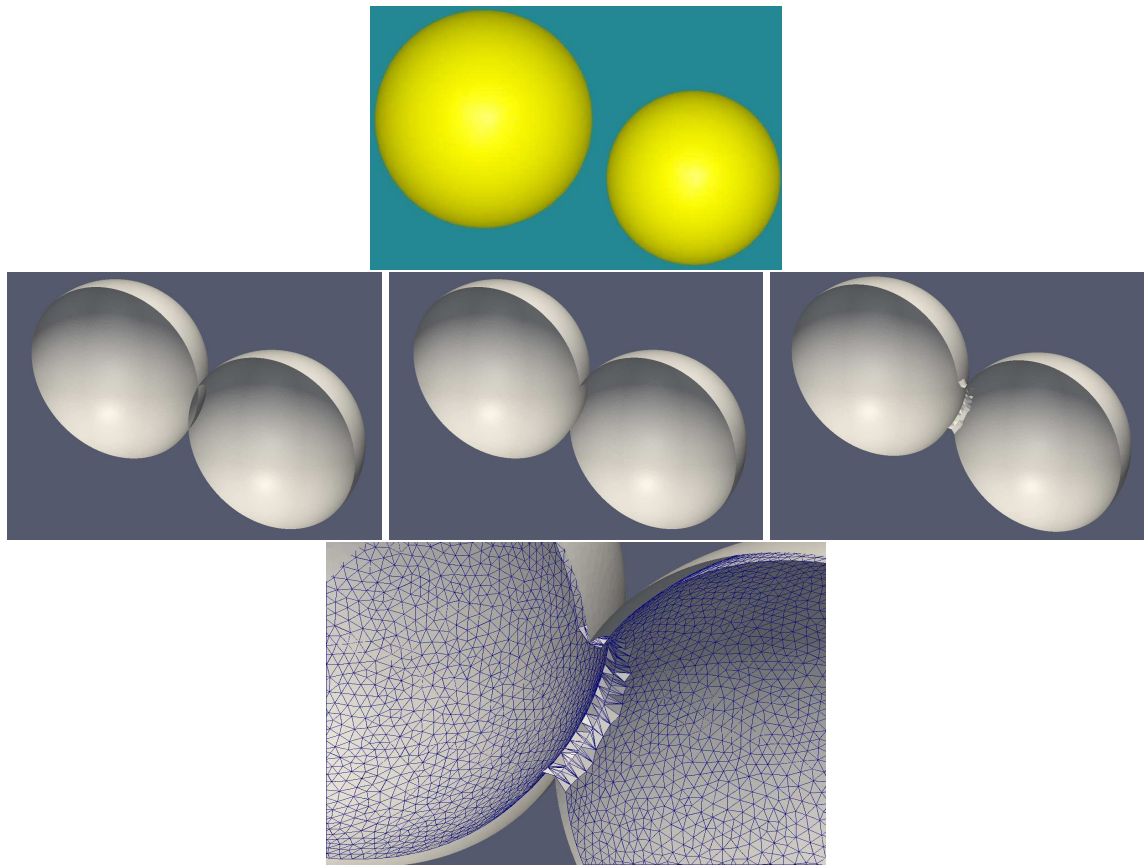


Figure A.1: The coalescence algorithm applied to two spheres. The main sequence of the algorithm is explained in the second row of the figure: the intersection between the two spheres is forced (left), then the irrelevant part of the intersection is dismissed (center) and the surfaces are taken back to their original positions, except for the vertices generated by the intersection process (right). The image on the bottom shows the necking region created by the process, overlapped with the initial stage of the algorithm.

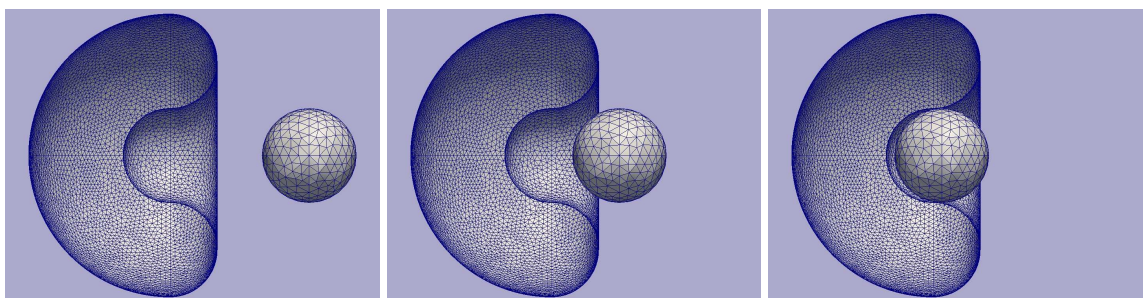


Figure A.2: Identifying the collision of two surfaces. Only a cut of the dimpled hemisphere is showed.

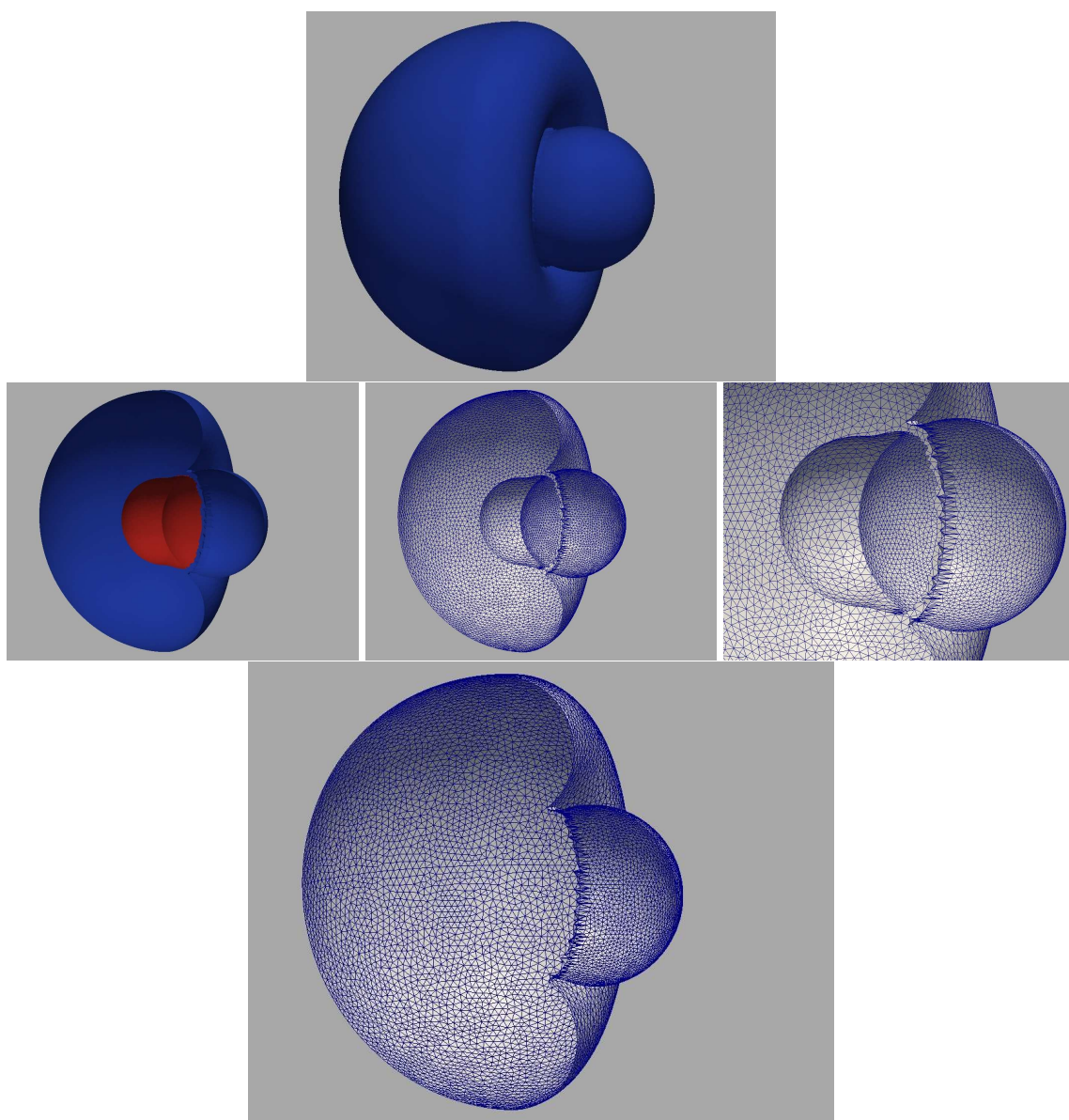


Figure A.3: The merge algorithm applied to a dimpled hemisphere (only half is shown) and a sphere.

A.2 Fragmentation

A naive fragmentation algorithm was also developed for the specific situation of bubble pinching, as illustrated in Fig. (A.4). The algorithm searches for the smallest cross section in the surface and then splits the geometry into two surfaces at this region. The search for the smallest cross section is based on the underlying Eulerian grid. Since the front-tracking algorithm introduces a numerical thickness for the interface in order to mitigate the discontinuity effects, the minimum number of Eulerian cells which must be inside the surface is known. Therefore, the breakup site will be the region in which this constraint is not respected. This procedure is summarized in the algorithm 4, which is illustrated in Fig. (A.5).

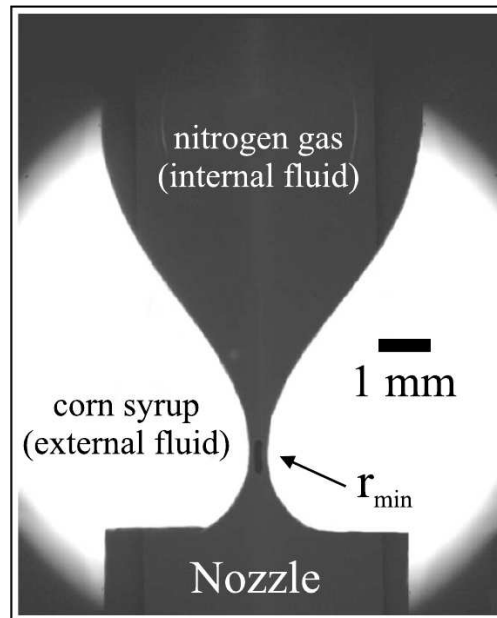


Figure A.4: An example of bubble necking in an experimental setup used for launching bubbles inside a tank filled with liquid (BURTON; WALDREP; TABOREK, 2005).

Algorithm 4 Surface breakup

Input:

- S_1 : surface under breakup
- $[x_1, y_1, z_1], [x_2, y_2, z_2]$: Bounding box of the Eulerian mesh covering S_1
- dx : characteristic length of the Eulerian mesh

1. Identify the patches in the finest level surrounding S_1
 2. Build a connectivity relationship for the grid points of patches identified in 1
 3. Find the regions where Eulerian points have the smallest number of neighbours
 4. Find the triangles located at these regions
 5. Identify the free vertices of these elements
 6. Split these vertices into two groups, G_1 and G_2
 7. Calculate the centroid of each group, C_1 and C_2
 8. $\mathbf{n} = C_1 C_2$ defines the intersection plane.
 9. Calculate the intersection between $\mathbf{n}$ and S_1
-

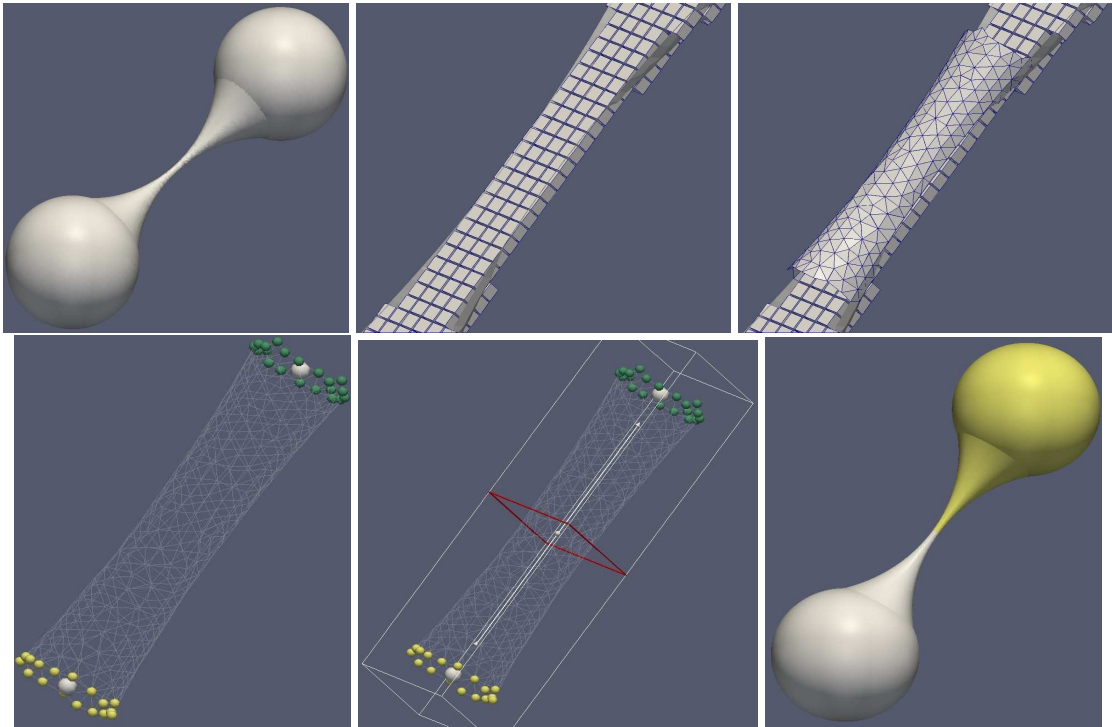


Figure A.5: Breakup algorithm. The necking region (top left) is identified by number of Eulerian cells inside the surface (top center) and the respective surface elements are identified (top right). The centroid of the open ends of this region (bottom left) are used for computing the normal defining the cutting plane (bottom center) and the surface is then split (bottom right).

APPENDIX B – Computational Aspects

B.1 GTS - GNU Triangulated Library

GNU Triangulated Library, or GTS, is a computational library which provides a complete data structure for representing a surface geometry with non-structured, triangulated meshes. The library provides support functions for iterating over the geometric entities (vertices, edges and faces), as well as remeshing functionalities, boolean operations and object location with kd-trees and binary trees.

The implementation is made in C and the data structure resembles an object oriented framework. Its design is based on the GTK+ architecture and Glib provides all supporting containers which are not strictly geometry-related, such as linked lists, hash tables etc. In order to keep the library as generic as possible, no special attribute, such as colour, temperature, etc., is included. However, the inheritance structure implemented allows the user to create new data types to meet specific needs and which are still fully integrated with GTS (POPINET, 2000).

There are seven fundamental data types in GTS: `GtsPoint`, `GtsSegment`, `GtsTriangle`, `GtsVertex`, `GtsEdge`, `GtsFace` and `GtsSurface`. Figure (B.1) shows the three basic types, named geometric types, which are related to the geometric representation of the object. Figure (B.2) shows the types used for defining the mesh and storing information about connectivity and Fig. (B.3) shows the definition of `GtsSurface`. The most fundamental structure is `GtsPoint`, which just stores the coordinates of each point in the mesh. The remaining structures are all linked in some way.

There is an equivalence between the geometric and topology structures. `GtsPoint` and `GtsVertex` represent a point, `GtsSegment` and `GtsEdge` represent an edge and `GtsTriangle`

<pre>typedef struct { GtsObject object; gdouble x,y,z; }GtsPoint;</pre>	<pre>typedef struct { GtsObject object; GtsVertex *v1; GtsVertex *v2; }GtsSegment;</pre>	<pre>typedef struct { GtsObject object; GtsEdge *e1; GtsEdge *e2; GtsEdge *e3; }GtsTriangle;</pre>
---	--	--

Figure B.1: Geometric data types in GTS.

<pre>typedef struct { GtsPoint p; GSList *segments; }GtsVertex;</pre>	<pre>typedef struct { GtsSegment segment; GSList *triangles; }GtsEdge;</pre>	<pre>typedef struct { GtsTriangle triangle; GSList *surfaces; }GtsFace;</pre>
---	--	---

Figure B.2: Mesh-related data types in GTS.

and `GtsFace` represent an area element. Using two `GtsVertex` objects to store its coordinates grants a `GtsSegment` object easy access to its neighbours. The same applies to the definition of `GtsTriangle`. Instead of storing its vertices, `GtsTriangle` stores the edges that define it. The edges are defined in the `GtsEdge` data structure and, similarly to `GtsVertex`, `GtsEdge` stores the list of `GtsTriangle` objects that share a given edge. This information is especially useful for determining whether a surface is open or closed, since in open surfaces the edges on the surface boundaries will be referenced by only one triangle. Finally, a `GtsFace` object is composed by a `GtsTriangle` object and a list of `GtsSurface` object that share it, while the struct `GtsSurface` stores the entire surface by storing its faces. Figure (B.4) shows these relationships graphically.

```
typedef struct {
    GtsObject object;
#ifdef USE_SURFACE_BTREE
    GTree *faces;
#else
    GHashTable *faces;
#endif
    GtsFaceClass *face_class;
    GtsEdgeClass *edge_class;
    GtsVertexClass *vertex_class;
    gboolean keep_faces;
}GtsSurface;
```

Figure B.3: The `GtsSurface` data type in GTS.

The struct `GtsSurface` plays the role of a connectivity matrix, which may be stored in a hash-table or in a kd-tree, as depicted in Fig. (B.3). As pointed out earlier, these supporting data

structures are provided by GLib. Access to the mesh components is provided by a series of function which resembles the use of iterators in C++.

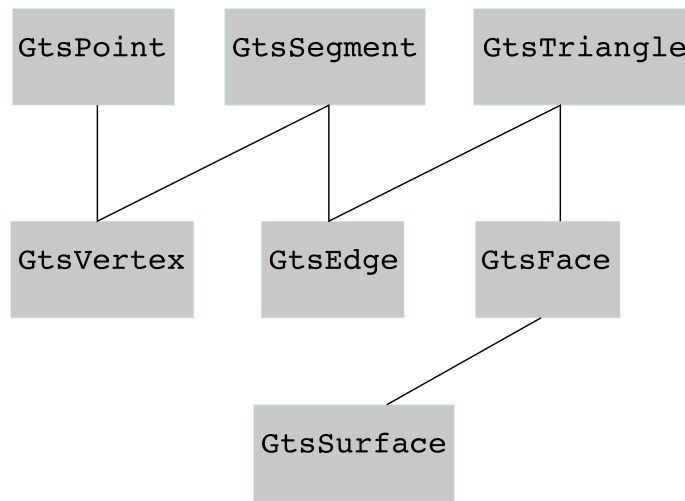


Figure B.4: Relationship between the data types defined in the GTS library.

B.1.1 GTS Functions

GTS functions are grouped according to the kind of object used as input and named accordingly. For example: functions designed to work with `GtsPoint` objects have names that start with `gts_point_`. The same apply for functions named `gts_edge_*`, `gts_triangle_*` and so on. Regarding the kind of functionality provided, GTS functions may be divided into four groups:

1. *geometric functions*: area, volume, centre of mass, distance.
2. *location functions*: point is inside a surface, point belongs to a plane, which is the closest point to some triangle.
3. *mesh-related functions*: mesh refinement, mesh coarsening, boundary edges and vertices
4. *iterator-like functions*: provide access to the mesh components (vertices, edges and faces).

B.2 Coupling GTS and AMR3D codes

This section describes the computational code developed for coupling two parts of the same software: AMR3D, developed for the Eulerian framework, implemented using Fortran (90 and

2003), and IB-GTS, developed in C using the GTS library for the Lagrangian framework. The term *interface*, in this section, refers to the computational interface between AMR3D and IB-GTS. The physical fluid-fluid interface will be called surface.

AMR3D was originally developed by (NOS, 2007) during his doctoral research about the use of phase field formulations and SAMR for modelling and simulating multiphase flows. This code was further adapted according to the requirements of the current project by Ph.D. Millena Martins Villar and currently provides a complete data structure for solving the Navier-Stokes equations based on a one-fluid formulation.

IB-GTS was developed using the GTS library in order to provide the necessary tools for the computational models of the Lagrangian framework for Immersed Boundary-like models, including front-tracking. Its main functionalities are:

- volume- and shape- conservative remeshing
- volume-conservative surface smoothing
- volume recovery by normal expansion of the surface
- generation of a signed distance field based on the Closest Point Transform (CPT)
- interfacial tension force calculation
- mesh quality report
- data conversion between GTS and AMR3D

The interface provides AMR3D with access to the Lagrangian surface data, as well as IB-GTS functions that should be called from the Eulerian framework. Implementation was performed using the Fortran module `iso_c_binding`, described in the next section.

B.2.1 C-Fortran interoperability

Fortran 2003 provides a module named `iso_c_binding`, which establishes a generic interface for interoperability of data and procedures between C and Fortran. Adams et al. (2009)

provide a detailed description about the rules of interoperability. In a nutshell, a C data type is interoperable with a Fortran data type if there is a corresponding data type in Fortran which requires the same amount of memory and their representation have a similar meaning. Therefore, features like union in C or allocatable in Fortran are not interoperable.

Table (B.1) shows some common data types in C and their Fortran corresponding type as defined by the module `iso_c_binding`.

C data type	Fortran equivalent data type
<code>int</code>	<code>integer(c_int)</code>
<code>size_t</code>	<code>integer(c_size_t)</code>
<code>double</code>	<code>real(c_double)</code>
<code>char</code>	<code>character(1,c_char)</code>

Table B.1: Some data types in C and their equivalent types in Fortran, as provided by the module `iso_c_binding` (ADAMS et al., 2009).

The support for C pointers is one of the most useful features of `iso_c_binding`. However, there are two fundamental differences between C pointers and Fortran pointers that have direct impact on the degree of interoperability:

- A pointer in Fortran has the same type as its target. C pointers do not.
- A pointer to an array in Fortran stores information about its shape and index limits, which makes this representation physically incompatible with the plain memory address representation of C pointers.

The physical differences between C pointers and Fortran pointers make it impossible declaring a dynamic variable in one language and allocating it in the other (ADAMS et al., 2009). However, after allocated they become interoperable as long as the data types allocated dynamically are also interoperable.

B.2.2 Module *gts_amr3d_interface*

Since GTS data structure is not directly interoperable with Fortran, a set of data structures and function interfaces was implemented so that the relevant data stored in IB-GTS could be passed

to AMR3D routines, eventually modified and then passed back to IB-GTS, which would use the processed data to update the `GtsSurface` object.

In order to give AMR3D access to the surface geometry, some wrapper data types were developed. Figure (B.5) shows the definition of the types `lag_vertex` and `lag_face`. Notice that the order of the definition of the members within the type must be the same.

<pre> type lag_vertex double precision:: x, y, z double precision:: ul, vl, wl double precision:: rho type (c_ptr)::v type(level_components1),pointer::lagmap end type lag_vertex </pre>	<pre> struct lag_vertex { double x,y,z; double ul,vl,wl; double rho; GtsVertex *v; level_components1 *lagmap; } </pre>
<pre> type lag_face double precision, dimension(3):: centroid double precision:: fx, fy, fz double precision:: ulf, vlf, wlf double precision, dimension(3):: & f_e1, f_e2, f_e3 type (c_ptr)::cell type(level_components1), pointer::lagmap end type lag_face </pre>	<pre> struct lag_face { double centroid[3]; double fx, fy, fz; double ulf,vlf,wlf; double f_e1[3], f_e2[3], f_e3[3]; GtsFace *cell; level_components1 *lagmap; } </pre>

Figure B.5: Wrapper data types as defined inside AMR3D (left) and IB-GTS (right) for granting AMR3D access to the geometric data of the surface.

The module `gts_amr3d_interface` establishes the interface for function call and data transfer between the two parts of the code. These functions are responsible for the interface remeshing, computing the indicator function, updating the interface position and applying the geometric correction tools described in section 3.3.2. Figure (B.6) shows the C declaration of two sample functions and Fig. (B.7) shows the respective sample code extracted from `gts_amr3d_interface`.

```

void surface_remesh(GtsSurface **s, int *remesh_flag,
                   double *edge_threshold, double *angle_threshold,
                   double *max_aspect_ratio, int *mesh_status);

void cpt_face_distance(GtsFace **face, SignedDistance **cpt,
                     double *f, int *f_size);

```

Figure B.6: A snapshot of the headers of the functions used for surface remeshing and calculating the indicator function.

```

module gts_amr3d_interface
use data_front_tracking
use data_types

interface
  subroutine surface_remesh(s, remesh_flag, &
                           edge_threshold, angle_threshold, &
                           max_aspect_ratio, remesh_status) &
    bind(c,name="surface_remesh")

    use iso_c_binding
    implicit none
    type(c_ptr)::s
    integer(c_int)::remesh_flag
    real(c_double)::edge_threshold
    real(c_double)::angle_threshold
    real(c_double)::max_aspect_ratio
    integer(c_int)::remesh_status
  end subroutine surface_remesh
end interface

interface
  subroutine cpt_face_distance(face, cpt, f, f_size) &
    bind(c,name="cpt_face_distance")

    use iso_c_binding
    implicit none
    type(c_ptr)::face, cpt
    integer(c_int)::f_size
    real(c_double),dimension(f_size)::f
  end subroutine cpt_face_distance
end interface

end module gts_amr3d_interface

```

Figure B.7: A snapshot of the module `gts_amr3d_interface` showing the interface for calling functions for surface remeshing and calculating the indicator function.

APPENDIX C – Two non-coalescing rising bubbles

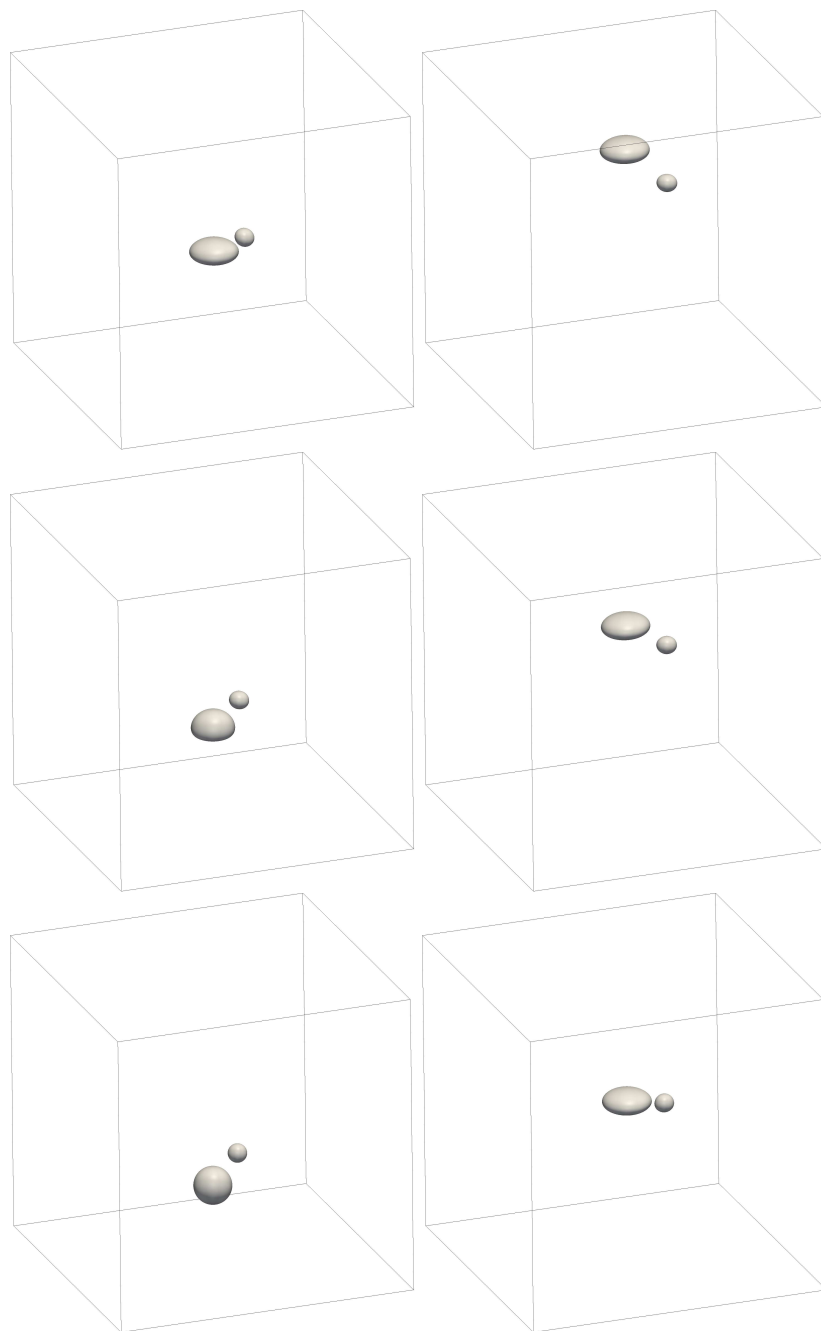


Figure C.1: The rise of two bubbles with different diameter. Temporal sequence is: left column, bottom to top; right column, bottom to top. Case based on (UNVERDI; TRYGGVASON, 1992).