World Applied Sciences Journal 22 (12): 1685-1694, 2013
ISSN 1818-4952
© IDOSI Publications, 2013
DOI: 10.5829/idosi.wasj.2013.22.12.837

Real-Time Physical Engine for Floating Objects with Two-Way Fluid-Structure Coupling

[1,2]*M. Lazo,* [1,2]*C. Garcia Bauza,* [1,2]*G. Boroni and* [1,3]*A. Clausse*

[1]UNCPBA, Universidad Nacional Del Centro De La Provincia De Buenos Aires, Argentina
[2]CONICET, Consejo Nacional De Investigaciones Científicas Y Técnicas, Argentina
[3]CNEA, Libertador 8250, 1429 Buenos Aires, Argentina

Abstract: A method to simulate graphic animations of objects floating in a water surface in real time is presented. The fluid is simulated by means of the Lattice Boltzmann method for the shallow-waters equations and the movement of the floating objects is calculated with a Newtonian physical engine suitable for the mechanics of rigid bodies. A two-way interaction between the fluid surface and the object structures is achieved by providing inputs to the Newtonian engine representing buoyancy, drag and lift forces calculated from the solution of the Lattice Boltzmann scheme, which in turn is perturbed by displacement forces acting at the objects boundaries. The method is tested in animation scenes of boats and different adrift objects, showing excellent rendering rates in desktop computers.

Key words: Real-time animation · Physics-based modelling · Graphic engines · Physical engines · Lattice Boltzmann · Shallow waters · Fluid-structure coupling

INTRODUCTION

The first graphic animations of fluid surfaces were not based on physical laws but on reduced representations such as Fourier series in two dimensions or similar families of functions [1, 2, 3]. Although it is possible to produce realistic panoramic views with these methods, the animation of effects resembling the interaction with solids is rather limited. On the other hand, liquid surfaces can in principle be modeled by means of the partial differential equations of fluids, but they are limited by stability problems and the computational cost of the numerical solution. In this direction, different methods treating the complete set of three-dimensional Navier-Stokes equations to animate liquid surfaces interacting with solid walls were proposed in [4, 5, 6, 7]. Takashi *et al*. [8] voxelled floating objects to define boundary conditions on the fluid model. Génevaux et al. [9] used an Eulerian formulation of the fluid coupled with floating objects represented as systems of springs and masses. Müller *et al*. [10] proposed a finite-element representation of solid deformable objects interacting with a smoothed-particle hydrodynamics model. Irving *et al*. [11] presented a technique to simulate large bodies of water interacting with solids using a combination of a Navier-Stokes solver and height maps. Lossaso *et al*. [12] developed a graphic representation of multiple interacting fluids. Wendt *et al*. [13] presented a method for 2D and 3D flows using a finite-volumes formulation to capture the boundary conditions avoiding the artifacts of Cartesian grids. Mihalef *et al*. [14] proposed a fluid simulation method that enable one-way interactions with rigid and deformable objects. Robinson-Mosher *et al*. [15] reported a model of interaction between a fluid and rigid shells. The quality of all these results is excellent, but unfortunately the computational cost precludes their application on interactive real-time scenarios.

Kim *et al*. [16] have implemented in GPU an algorithm based in images to calculate buoyancy forces acting in arbitrary models of solids. This approach achieves good performances for real-time applications, although there is no feedback from the solid to the fluid. Harada *et al*. [17] presented a complete model of fluid-structure interaction in real time based in particle tracking and implemented on GPU. In this case, solids are represented as fluids by imposing on them deformable boundary conditions.

Corresponding Author: M. Lazo, UNCPBA, Universidad Nacional Del Centro De La Provincia De Buenos Aires, Argentina.

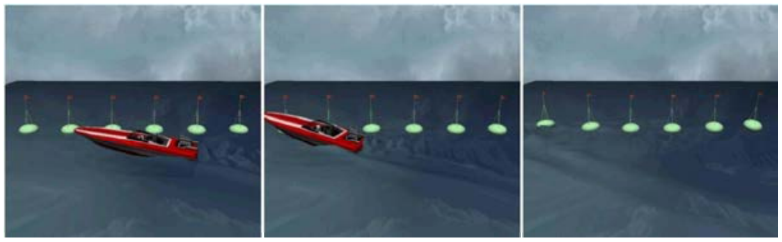
Fig. 1: Frames from a real-time animation of a boat and buoys

Yuksel *et al*. [18] presented a method to simulate the interaction of solids with waves' propagation in an elastic environment, capable of high refreshment rate. Cords [19] proposed a model of solids simulated by particle sets interacting with a bi-dimensional surface-wave equation. In [20] the model was extended to be able to animate large scenarios using a movable grid approach providing greater detail closer to the observer.

A simple approximation that, while maintaining the fluid transport equations, keeps the computational time under control by reducing the mathematical problem to two-dimensions, are the shallow-waters equations [21, 22, 23, 24, 25, 26, 27]. In the present work a Lattice-Boltzmann cellular automata representing the shallow-waters equations [28] is coupled with movable and static solid objects, to produce fast real-time and interactive animations of nautical scenes. Figure 1 shows three snaps of a real-time animation of the passage of a motorboat nearby a line of buoys. The animation depicts all the effects that can be created with the present method. The boat propeller is simulated introducing a force in the stern that push the vessel ahead. It can be seen that the waves created by the passage of the boat affect the position of the buoys.

Model: The animation model proposed here is composed by a water-surface model based in a Lattice-Boltzmann (LB) automaton and models of solid objects represented by triangular meshes moved by a Newtonian physical engine. Both models are coupled, the fluid exerting buoyancy and dynamic forces on the objects and the objects transferring momentum to the fluid.

Water Surface Model: The Shallow Waters equations are a set of 2D partial differential equations describing the movement of free fluid surfaces when the horizontal scale is much larger than the vertical scale. Under this assumption, the spatial average of the Navier-Stokes equations over the vertical coordinate from the solid bottom to the liquid surface, gives the following set [29]:

$$\frac{\partial h}{\partial t}+\frac{\partial (hu_x)}{\partial x}+\frac{\partial (hu_y)}{\partial y}=0$$

$$\frac{\partial (hu_x)}{\partial t}+\frac{\partial \left(hu_x{}^2\right)}{\partial x}+\frac{\partial (hu_x u_y)}{\partial y}=-g\frac{\partial}{\partial x}\left(\frac{h^2}{2}\right)+\nu\frac{\partial^2 (hu_x)}{\partial x^2}+\nu\frac{\partial^2 (hu_x)}{\partial y^2}-gh\frac{\partial z_b}{\partial x}$$

$$\frac{\partial (hu_y)}{\partial t}+\frac{\partial (hu_x u_y)}{\partial x}+\frac{\partial \left(hu_y{}^2\right)}{\partial y}=-g\frac{\partial}{\partial y}\left(\frac{h^2}{2}\right)+\nu\frac{\partial^2 (hu_y)}{\partial x^2}+\nu\frac{\partial^2 (hu_y)}{\partial y^2}-gh\frac{\partial z_b}{\partial y}$$

where h is the fluid height, x and y are the spatial coordinates, u_x and u_y are the velocity components in x and y directions respectively, ν is the viscosity, g is the gravity and z_b is the bed elevation.

The SW equations are solved in the present case by means of the Lattice Boltzmann method (LBM) [29]. Several authors have already produced water animations using LBM for the Navier-Stokes equations [7, 30, 31]. LBM has several advantages over other conventional computational fluid dynamics methods, especially for modeling with complex boundaries, incorporating of microscopic interactions and parallelization of the algorithm. From the numerical perspective, the method can be viewed as an explicit scheme with more state variables than the macroscopic problem require (i.e., density and velocity), which are recovered by means of appropriate averages of the former. LBM ordinarily operates on a regular spatial grid $\vec{x}$ and a set of nine velocity vectors $\vec{v}_\alpha$ that are used in a hidden mesoscopic representation of the fluid, that is:

$$\vec{v}_0=\begin{pmatrix}0\\0\end{pmatrix}$$

$$\vec{v}_1=\begin{pmatrix}v\\0\end{pmatrix}\quad \vec{v}_2=\begin{pmatrix}v\\-v\end{pmatrix}\quad \vec{v}_3=\begin{pmatrix}0\\-v\end{pmatrix}\quad \vec{v}_4=\begin{pmatrix}-v\\-v\end{pmatrix}$$

$$\vec{v}_5=\begin{pmatrix}-v\\0\end{pmatrix}\quad \vec{v}_6=\begin{pmatrix}-v\\v\end{pmatrix}\quad \vec{v}_7=\begin{pmatrix}0\\v\end{pmatrix}\quad \vec{v}_8=\begin{pmatrix}v\\v\end{pmatrix}$$

where $v = \Delta x/\Delta t$ is a characteristic velocity bit given by the ratio between the cell side and the time step associated to each state change of the system. Each cell of the grid is populated by mesoscopic pseudo-particles whose state is given by a density function $f_\alpha = (\vec{x},t)$ representing the number of pseudo-particles in the node $\vec{x}$ at time t moving with velocity $\vec{v}_\alpha$. The physical observables are macroscopic variables generated by moments of $f_\alpha = (\vec{x},t)$ respect to $\vec{v}_\alpha$, namely:

$$h(\vec{x},t) = h_0 \sum_\alpha f_\alpha(\vec{x},t) \quad \text{Water surface height}$$

$$\vec{u}(x,t) = \frac{\sum_\alpha \vec{v}_\alpha f_\alpha(\vec{x},t)}{h(\vec{x},t)} \quad \text{Average velocity}$$

where h_0 is the elementary height of each pseudo particle. The state of each cell change according to a scheme of explicit rules, that is [29]:

$$f_\alpha(\vec{x}+\vec{v}_\alpha\Delta t, t+\Delta t) = f_\alpha(\vec{x},t) - \frac{1}{\tau}\left[f_\alpha(\vec{x},t) - f_\alpha^{eq}(\vec{x},t)\right] + \frac{\Delta t}{6v^2}\vec{v}_\alpha \cdot \overrightarrow{F^f} \quad (1)$$

where $\overrightarrow{F^f}$ is the force acting on the fluid, which will be specified later, τ is a relaxation time that controls the viscosity and $f_\alpha^{eq}(\vec{x},t)$ is an equilibrium function, which for the shallow-waters equations is given by [29]:

$$f_o^{eq} = h\left(1 - \frac{5gh}{6v^2} - \frac{2u^2}{3v^2}\right)$$

$$f_\alpha^{eq} = w_\alpha h\left(\frac{gh}{6v^2} + \frac{\vec{v}_\alpha \cdot \vec{u}}{3v^2} + \frac{(\vec{v}_\alpha \cdot \vec{u})^2}{2v^4} - \frac{u^2}{6v^2}\right), \qquad \alpha = 1,\ldots,8$$

where:

$$w_\alpha = \begin{cases} 1, & \alpha = 1,3,5,7 \\ 1/4, & \alpha = 2,4,6,8 \end{cases} \quad (2)$$

where u is the module of $\vec{u}$.

Fluid-To-Object Coupling: The behavior of a rigid object immersed in a liquid media under gravity conditions is governed by static and dynamic forces (Fig. 2). The hydrostatic force is given by the integral of the local pressure acting normal to the surface of the object in contact with the fluid. In the present model, the external surface of each object is represented by a mesh of triangles and the buoyancy force is then partitioned in individual components acting on each of triangle of the surface, that is:

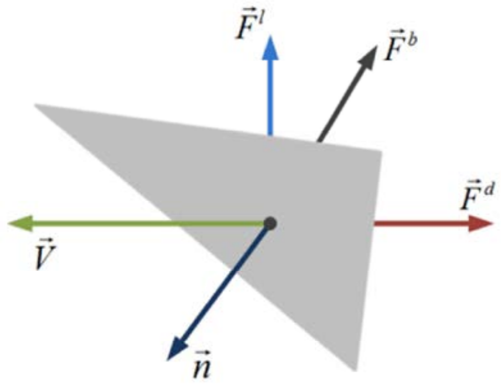

Fig. 2: Force components acting in a triangle of the object surface

$$\overrightarrow{F_i^b} = -\rho g h_i A_i \vec{n}_i \quad (3)$$

where $\overrightarrow{ni}$ and $\overrightarrow{F_i^b}$ are the outward-normal unitary vector of the i-triangle and the buoyancy force acting on its wet area A_i, h_i is the distance between the fluid level and the centroid of the immersed part of the corresponding triangle, ρ is the liquid density and g is the gravity acceleration.

The dynamic components are the drag and lift forces, which depend on the velocity of the object relative to the liquid. The drag force opposes to the object movement whereas the lift force acts in a direction normal to the object velocity. Yuksel *et al.* [18] propose the following models to calculate these forces:

$$\overrightarrow{F_i^d} = -\frac{1}{2}\rho C_d A_i^e \left|\vec{V}\right| \vec{V} \quad (4)$$

$$\overrightarrow{F_i^l} = -\frac{1}{2}\rho C_l A_i^e \left|\vec{V}\right| \vec{V} \times \frac{\vec{n}_i \times \vec{V}}{\left|\vec{n}_i \times \vec{V}\right|} \quad (5)$$

where $\vec{V}$ is the object velocity relative to the liquid, C_d and C_l are the drag and lift coefficients and the effective area A_i^e is given by:

$$A_i^e = \left[\frac{\vec{n}_i \cdot \vec{V}}{\left|\vec{V}\right|}\alpha + (\alpha - 1)\right] A_i \quad (6)$$

α being a user defined parameter ($0 \le \alpha \le 1$) that accounts for object shape and orientation effects. In the numerical experiments presented later $\alpha = 1$. Lower values increase the damping of the object motion.

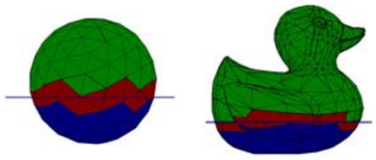

Fig. 3: Triangle classification according to the position respect to the liquid level: totally wet (blue), totally dry (green) and semi-wet (red)

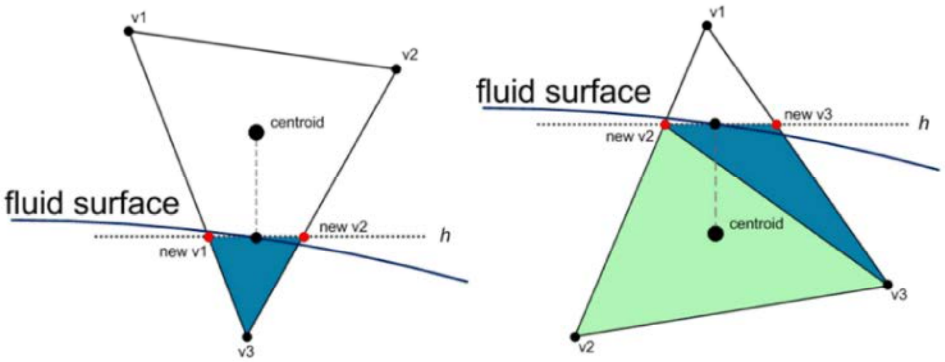

Fig. 4: Processing of semi-wet triangles

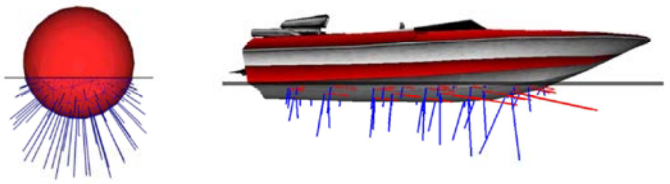

Fig. 5: Forces acting in a still ball and a moving boat (Blue arrows: buoyancy and lift forces, red arrows: drag forces).

Depending on the position respect to the liquid level, each triangle can be classified in three categories: totally wet, totally dry or semi-wet (Fig. 3). The totally dry or totally wet triangles do not present major difficulties in the calculation of the forces. However, the calculation of forces acting on the semi-wet triangles requires some attention. The exact geometrical calculation of the wet area and the corresponding centroid, by intersection of the model mesh and the liquid surface grid, would consume too much calculation time in real time applications. Therefore, in the present implementation an approximate solution is proposed cutting the triangle horizontally at the liquid level of the projected coordinates of the triangle centroid, obtained by interpolation of the closest liquid nodes. As can be seen in Fig. 4, the resulting wet surface can be a triangle or a quadrilateral. In the former case the calculation is performed over the wet triangle as in the totally wet case. If the wet region is a quadrilateral, the area and the centroid can be also calculated, but it is more convenient to preserve the triangle primitive for implementation reasons and compatibility with most of the functionality of existing computer graphic components. Therefore, the quadrilateral is divided in two triangles by cutting through one of its diagonals (Fig. 4). In order to minimize numerical errors in the triangles division, the object mesh should be as regular as possible, with triangles approximately equilateral [32].

Figure 5 shows the forces acting on a floating ball and a moving boat. Only buoyancy forces act on the ball for it is adrift. In turn, since the boat is propelled, drag and lift forces are activated. The forces acting on the objects are implemented by means of the general-purpose physical engine “Newton Game Dynamics” (www.newtondynamics.com), which represents the mechanics of rigid bodies providing functionality to apply a variety of forces and velocities and solves collisions and reaction forces taken into account the properties of the constituent materials. In the present case, each rigid body is represented by a collision mesh that can be described by a simple primitive (e.g., a sphere) or a complex primitive (e.g., a convex hull). Additionally, each

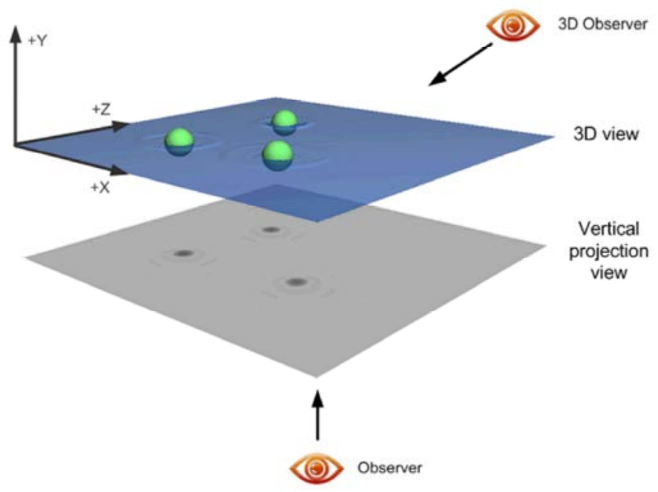

Fig. 6: Configuration of the views using OpenGL depth buffer

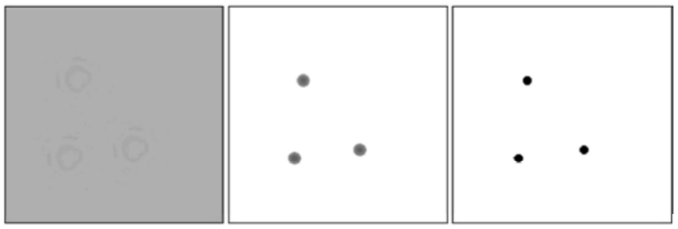

Fig. 7: Depth buffer map of the water surface (left) and the floating objects (center), binary mask showing the wet boundaries (right)

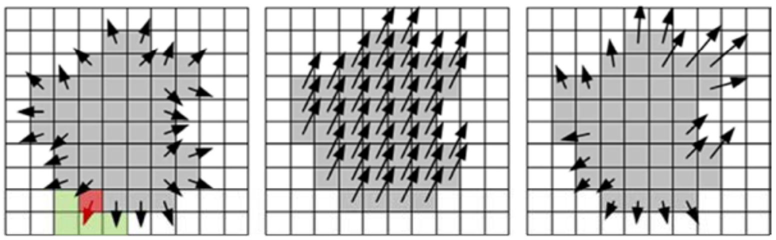

Fig. 8: Unitary vectors $\vec{n}$ representing the normal to the wet perimeter (left), velocity field $\vec{V}$ of the object cells in the surface plane (center), force field $\overrightarrow{Ff}$ acting on the wet perimeter (right)

body has physical attributes as mass, inertial tensor and material properties as friction coefficient and elasticity. The physical engine was implemented by means of an abstraction layer [33], which isolates the user from the specifics of the particular engine, thus facilitating the development of applications. Nevertheless, the model of forces presented here can be implemented using any other similar physical engine e.g., [34].

Object-To-Fluid Coupling: The movement of an object in a liquid surface generates perturbations in the latter through momentum transfer, like wakes spreading behind boats as they move forward or the circular waves expanding outwards from the hitting point of a raindrop in a pond. In the present model, this feedback is simulated by means of forces acting on the LB fluid at the objects boundaries.

The "wet boundary" of the objects is calculated in each time step using the depth buffer of the OpenGL library [35]. The method is similar to the one proposed by Kim *et al*. [16] and Crane *et al*. [36]. The depth buffer stores the free-view length between the observer and the closest point of an object of interest for the ray passing through each pixel. In this way, positioning the observer view perpendicular to the horizontal plane as shown in Fig. 6, the depth map of the water surface (Fig. 7 left) and of the floating objects (Fig. 7 center) can be obtained. By comparing both maps, a binary mask corresponding to the immersed section of the objects can be obtained (Fig. 7 right), which is then used as boundary conditions for the fluid engine to calculate the forces acting on the water. In addition, each object should be labeled in order to identify each border. The position of the observer is important, for if it is too close to the scene some objects can fall out of view, whereas if it is too far the depth-buffer saturation reduces the sensitivity to determine the boundaries.

Once the position of the wet boundary of every object is determined, the forces exerted on the fluid are estimated in each border cell as:

$$\overrightarrow{F^f} = \begin{cases} k_+\left(\vec{V}\cdot\vec{n}\right)\vec{n}, & \left(\vec{V}\cdot\vec{n}\right) > 0 \\ k_-\left(\vec{V}\cdot\vec{n}\right)\vec{n}, & \left(\vec{V}\cdot\vec{n}\right) < 0 \end{cases} \tag{7}$$

where $\vec{V}$ is the velocity of the object in the cell, $\vec{n}$ is an estimation of the vector normal to the border and k_+ and k_- are constant coefficients that can be used to control different animation effects. The physical effect of k_+ and k_- is to displace the fluid with different intensity depending on whether the object is pushing (k_+) o pulling (k_-) the fluid. Realistic effects were obtained by keeping $k_+ / k_- \approx 10$. Eq. (7) provides the expression used in Eq. (1) to move the fluid.

The vector $\vec{n}$ of each border cell is calculated by adding the vectors pointing from the cell center to the adjacent water cells. For example, the normal vector of the red cell in Fig. 8 (left) is estimated as:

$$\vec{n} = (-1,0) + (-1,1) + (0,1) + (1,1) = (-1,3)$$

$$\vec{n} = \left(-1/\sqrt{10}, 3/\sqrt{10}\right)$$

RESULTS

In order to verify the consistency and stability of the proposed method several test cases were tested involving the animation of different scenarios of floating objects. In addition, simulations with sets of different number of floating objects were performed on a standard PC (3.3 GHz Intel i5 with GeForce GTX 560) to assess the performance and scalability of the results.

The first test is a static scene consisting of a boat loaded with a set of heavy objects. The boat is represented by a mesh of 2628 triangles (~500 triangles/m^2). The boat dimensions are 3.23 m long, 0.43 m height and 1.2 m wide. A vertical force acts at the position of the center of each object representing a total weight of 90 kg. Fig. 9 shows how the floating line changes depending on the position of the objects, sinking towards the charged side or balancing the boat if the cargo is centered. In the same picture, the pressure forces exerted by the water on the vessel base are depicted, showing the regions where the structure is more compromised.

The second test case is the interaction of a 29-tons sail boat with a train of 4-meter waves. The boat dimensions are 19 m long, 2.3 m high and 4.6 m wide. The waves are simulated by means of the function:

$$Y(x) = A\left|\sin\frac{2\pi}{L}(x+ct)\right|^n$$

where A is the height of the wave peaks, L is the distance between peaks, x is the corresponding spatial coordinate, c is the module of the wave velocity, n is a shape exponent and t is the time coordinate.

Figure 10 shows snaps of two sequences showing the boat response to the path of a wave. In the first sequence (above) the center of mass is located 0.5 m over the boat base. As the wave passes, the boat sways to one side but recovers the vertical position afterwards. In turn, in the second sequence (below) the center of mass is located 1.65 m above the boat base, which is an unstable configuration. It can be seen that the simulation reproduces how the boat is capsized due to the inclination produced by the path of the wave.

The third case simulates the motion of a motorboat in still water. Fig. 11 shows different snaps of the animation where it can be seen the formation of waves brought about by the movement of the boat leaving a wake behind its path. It is interesting to note how the physical engine is able to recreate the curved form of the wake left by the turns of the motorboat direction. The water surface is represented by 200×200 cells. The boat dimensions are 12.7 m length, 2.4 m stanchion and 3.85 m wide, weighting 16.5 tons and it is represented by a mesh of 460 triangles.

The performance of the engine in simulating scenes with multiple objects was tested animating several sets of objects (balls and pins) floating in a pool (Fig. 12). Each ball is represented by 80 triangles and each bowling pin by a 520 triangles. The fluid surface is modeled with a 175×175 grid, which runs in an execution thread independent from the physical engine. This permits the utilization of large grids without compromising the general

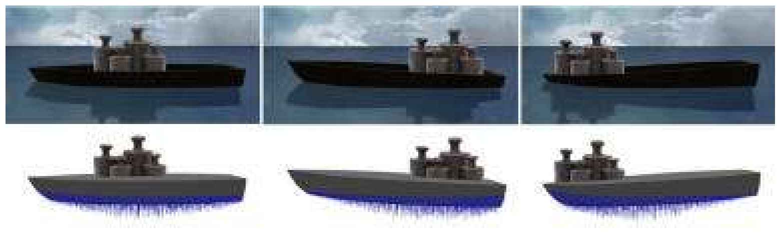

Fig. 9: Animation snaps of a boat with cargo distributed in different positions (above) and the corresponding pressure distribution exerted by the water (below)

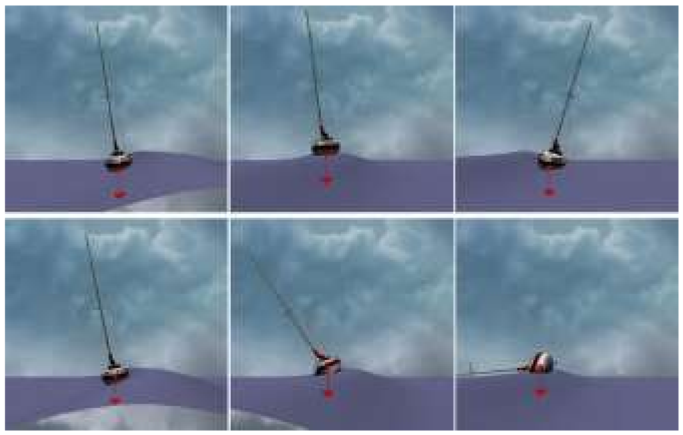

Fig. 10: Animation of the path of a wave under a boat: stable (above) and unstable (below). The wave parameters used in Eq. (10) are $A = 4$ and $n = 8$

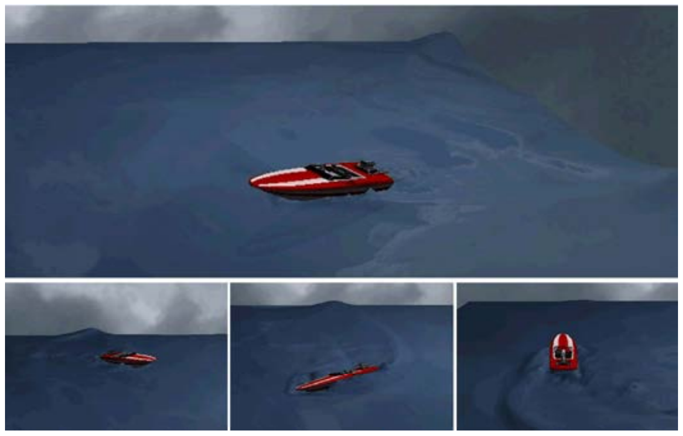

Fig. 11: Animation of a motorboat navigating in still water

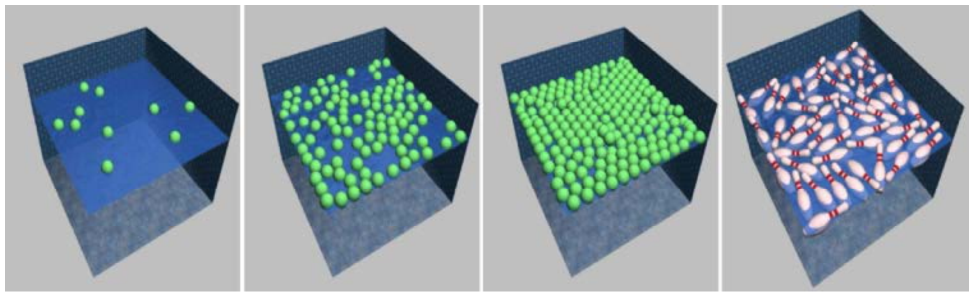

Fig. 12: Snaps of 10, 100 and 200 spherical balls and 50 bowling pins floating in a pool.

Table 1: Performance per frame of animation tests with multiple objects

	10 balls	100 balls	200 balls	50 pins
FPS	150	38	16	11
Forces (ms)	1.84	19.5	39.32	47.3
Positions (ms)	0.067	0.57	15.72	32.4
Obj. Borders (ms)	3.56	3.72	3.92	3.6
Rendering (ms)	1.19	2.08	2.98	4.1

performance of the animation. However, it is important to carefully define synchronization mechanisms in order to avoid inconsistencies during the simultaneous rendering of the water and the floating objects.

Table 1 shows the rendering rate of each scene and the corresponding computational cost of each stage of the algorithm. It can be seen that, for similar objects, the number of objects has a direct impact on the calculation times, which is reasonable since the physical engine acts in each object and, therefore, the increase in calculation time is linear with the size of the objects population. On the other hand, the scene with pins (using 26000 triangles) took a substantially larger calculation time compared with the scenes with balls (using 80, 8000 and 16000 triangles respectively). This suggests that the method works better with many simple objects than with few complex objects. The explanation of this is that with similar objects it is possible to exclude from the calculation cycle objects that do not require actualizations, such as objects that stay still during a certain period of time, objects located far from the observer or out of visual range, etc.

CONCLUSIONS

A physical engine capable of two-way interaction between solid structures and fluid surfaces was presented. The method is useful to produce animations of scenes of arbitrary floating objects in open waters, capturing the perturbation of the surface by the objects and the reaction of the water on the objects. The water is fast, scalable and was tested in several demonstration cases producing realistic animations for interactive applications like training simulators and videogames [37, 38, 39].

An interesting contribution of the present article is the way the fluid-structure forces are calculated using the wet surface of the objects, which save the computational cost of calculating the immersed volume and the corresponding geometrical centers required to apply the Archimedean buoyancy principle. Moreover, it should be noted that although the algorithm of fluid-structure interaction was demonstrated here using a LBM fluid, it can be applied to any other scheme based in grids.

Future extensions of the method include the application of the Lattice-Boltzmann additional data, like local internal energy or distribution entropy, to emulate surface visual effects, as foam or flicking reflections. Also, alternative treatments of the boundary conditions between the objects and the water can be explored using immersed boundary methods [40]. Additionally, GPU implementations can be used to run the Lattice Boltzmann algorithm [41, 42], which greatly reduces the calculation time.

REFERENCES

1. Schachter, B., 1980. Long crested wave models, Computer Graphics and Image Processing, 12: 187-201.
2. Masten, G., P. Watterberg and I. Mareda, 1987. Fourier synthesis of ocean scenes, IEEE Computer Graphics and Application, 7: 16-23.
3. Tso, P. and B. Barsky, 1987. Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping, ACM Transactions on Graphics, 6: 191-214.
4. Foster, N. and D. Metaxas, 1996. Realistic animation of liquids, Graphical Models and Image Processing, 58(5): 471-483.
5. Stam, J., 1999. Stable fluids, SIGGRAPH, Computer Graphics Proceedings, Rockwood A. (Ed.), Addison Wesley Longman, pp: 121-128.
6. Layton, A. and M. Van De Panne, 2002. A numerically efficient and stable algorithm for animating water waves, The Visual Computer, 18: 41-53.
7. Thürey, N., 2007. Physically Based Animation of Free Surface Flows with the Lattice Boltzmann Method, PhD thesis, University of Erlangen-Nuremberg.
8. Takashi, T., U. Heihachi and A. Kunimatsu, 2002. The simulation of FLuid-rigid body interaction. In SIGGRAPH, Sketches and Applications, pp: 226.
9. Génevaux, O., A. Habibi and J. Dischler, 2003. Simulating Fluid-Solid Interaction, Proc. of Graphics Interface 2003, Association for Computing Machinary Inc., pp: 31-38.
10. Müller, M., S. Schirm, M. Teschner, B. Heidelberger and M. Gross, 2004. Interaction of fluids with deformable solids, Research Articles, Computer Animation and Virtual Worlds, 15(3-4): 159-171. [doi>10.1002/cav.v15:3/4].

11. Irving, G., E. Guendelman, F. Lossaso and R. Fedkiw, 2006. Efficient simulation of large bodies of water by coupling two and three dimensional techniques, Proc. of SIGGRAPH ’06; pp: 805-811.
12. Lossaso, F., G. Irving and E. Guendelman, 2006. Melting and burning solids into liquids and gases, IEEE Trans. Vis. Comp. Graph, 12: 343-352.
13. Wendt, J., W. Baxter, I. Oguz and . Lin, 2007. Finite volume flow simulations on arbitrary domains, Graphical Models, 69(1): 19-32.
14. Mihalef, V., S. Kadioglu, M. Sussman, D. Metaxas and V. Hurmusiadis, 2008. Interaction of two-phase flow with animated models, Graphical Models, 70: 33-42.
15. Robinson-Mosher, A., T. Shinar, J. Gretarsson, J. Su and R. Fedkiw, 2008. Two-way Coupling of Fluids to Rigid and Deformable Solids and Shells, ACM Trans. Graph, 27: Article 46.
16. Kim, J., S. Kim, H. Ko and D. Terzopoulos, 2006. Fast GPU computation of the mass properties of a general shape and its application to buoyancy simulation, The Visual Computer, 22: 856-864.
17. Harada, T., S. Koshizuka and Y. Kawaguchi, 2007. Smoothed particle hydrodynamics in complex shapes, Proc. of Spring Conference on Computer Graphics, pp: 26-28.
18. Yuksel, C., D. House and J. Keyser, 2007. Wave Particles, Proceedings of SIGGRAPH; 26, 99, San Diego, California, ACM Press.
19. Cords, H., 2008. Moving with the flow: Wave particles in flowing liquids, 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG '08); 16: 145-152.
20. Cords, H. and O. Staadt, 2009. Real-Time open water environments with interacting objects, Proceedings of Eurographics Workshop on Natural Phenomena (EGWNP '09), Aire-la-Ville, Switzerland, pp: 35-42.
21. Kass, M. and G. Miller, 1990. Rapid stable fluid dynamics for computer graphics, ACM SIGGRAPH Computer Graphics, 24: 49-57.
22. Chen, J. and D. Lobo, 1995. Toward interactive-rate simulation of Fluids with moving obstacles using Navier-Stokes equations, Graph. Models Image Process, 57: 107-116.
23. O'Brien, J. and J. Hodgins, 1995. Dynamic simulation of splashing fluids, Proc. Computer Animation, Geneva, Switzerland, April 19-21, 1995, pp: 198-205.
24. Thon, S. and D. Ghazanfarpour, 2001. A semi-physical model of running waters, Comput. Graph. Forum. Proc. Eurographics, 19: 53-59.
25. Neyret, F. and N. Praizelin, 2001. Phenomenological simulation of brooks, Proc. Eurographics Workshop, pp: 53-64.
26. Foster, N. and R. Fedkiw, 2001. Practical animation of liquids, Proc. of ACM SIGGRAPH, pp: 23-30.
27. Rinaldi, P., E. Dari, M. Vénere and A. Clausse, 2012. A lattice-Boltzmann solver for 3d fluid simulation on GPU, Simulation Modelling Practice and Theory, 25: 163-171.
28. García, Bauza C., G. Boroni, M. Vénere and A. Clausse, 2010. Realtime Interactive Animations of Liquid Surfaces with Lattice-Boltzmann Engines, Australian J. Basic and Applied Sci., 4: 3730-3740.
29. Zhou, J., 2004. Lattice Boltzmann Methods for Shallow Water Flows, Springer-Verlag.
30. Thürey, N. and U. Rüde, 2004. Free Surface Lattice-Boltzmann fluid simulations with and without level sets, Workshop on Vision Modeling and Visualization, pp: 199-208.
31. Li, W., Z. Fan, X. Wei and A. Kaufman, 2005. Flow Simulation with Complex Boundaries, GPU Gems II, Ed. Matt Pharr (Nvidia), Addison Wesley, Chapter, pp: 47.
32. D’Amato, J.P. and P. Lotito, 2011. Mesh optimization with volume preservation using GPU. Latin American Applied Research, 41: 291-297.
33. García, Bauza C., M. Lazo and M. Vénere, 2008. Incorporación de comportamiento físico en motores gráficos, Mecánica Computacional, 27: 3023-3039. (in Spanish).
34. Millington, I., 2010. Game Physics Engine Development, 2nd Ed., How to Build a Robust Commercial-Grade Physics Engine for your Game, Ed. Morgan Kaufmann.
35. Woo, M., J. Neider, T. Davis and D. Shreiner, 1999. The Framebuffer, OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 1.2 (3rd Edition), Chapter 10.
36. Crane, K., I. Llamas and S. Tariq, 2007. Real-Time Simulation and Rendering of 3D Fluids, Chapter 30, GPU Gems 3, Addison-Wesley Professional.
37. Li, G.C., L.Y. Ding and J.T. Wang, 2006. Construction project control in virtual reality: A case study. World Applied Sciences Journal, 6: 2724-2732.

38. Hing, T.H. and M. Musa, 2008. Simulator for control of autonomous nonholonomic wheeled mobile robot. World Applied Sciences Journal, 8: 2534-2543.
39. Boroni, G., C. García Bauza, J.P. D'amato and M. Lazo, 2012. Siper - Virtual reality simulator of periscope, World Applied Sciences Journal, 18(6): 813-817.
40. Boroni, G., J. Dottori, D. Dalponte, P. Rinaldi and A. Clausse, 2012. An improved Immersed-Boundary algorithm for fluid-solid interaction in Lattice-Boltzmann simulations, Latin American Applied Research. ISSN N°: 0327-0793. Accepted 2012.
41. Rinaldi, P., E. Dari, M. Vénere and A. Clausse, 2011. Lattice-Boltzmann Navier-Stokes simulation on graphic processing units, Asian J. Applied Science, 4: 762-770.
42. Rinaldi, P., D. Dalponte, M. Vénere and A. Clausse, 2012. Graph-based cellular automata for simulation of surface flows in large plains, Asian J. Applied Science, 5: 224-231.