DOI: 10.1111/cgf.14013

COMPUTER GRAPHICS *forum*
Volume 39 (2020), number 6 pp. 217–231

Realistic Buoyancy Model for Real-Time Applications

J. M. Bajo,[1,2] G. Patow[3] and C. A. Delrieux[1]

[1]Department of Electric and Computer Engineering, Universidad Nacional del Sur, Bahía Blanca, Argentina
jmb@cs.uns.edu.ar, cad@uns.edu.ar
[2]Department of Computer Science and Engineering, Universidad Nacional del Sur, Bahía Blanca, Argentina
[3]ViRVIG, Universitat de Girona, Girona, Spain
gustavo.patow@udg.edu

Abstract

Following Archimedes' Principle, any object immersed in a fluid is subject to an upward buoyancy force equal to the weight of the fluid displaced by the object. This simple description is the origin of a set of effects that are ubiquitous in nature, and are becoming commonplace in games, simulators and interactive animations. Although there are solutions to the fluid-to-solid coupling problem in some particular cases, to the best of our knowledge, comprehensive and accurate computational buoyancy models adequate in general contexts are still lacking. We propose a real-time Graphics Processing Unit (GPU) based algorithm for realistic computation of the fluid-to-solid coupling problem, which is adequate for a wide generality of cases (solid or hollow objects, with permeable or leak-proof surfaces, and with variable masses). The method incorporates the behaviour of the fluid into which the object is immersed, and decouples the computation of the physical parameters involved in the buoyancy force of the empty object from the mass of contained liquid. The dynamics of this mass of liquid are also computed, in a way such that the relation between the centre of mass of the object and the buoyancy force may vary, leading to complex, realistic beha viours such as the ones arising for instance with a sinking boat.

Keywords: Physically Based Animation, Animation

ACM CCS: • Computing methodologies → Physical simulation

1. Introduction

Solid-to-fluid coupling of submerged objects has been an active research focus in the Computer Graphics field since its beginnings [CL95]. This topic is of particular importance in games [GO16], Virtual Reality, simulators and other interactive applications. Despite this extensive research, a comprehensive interactive model for realistic fluid-to-solid coupling is still a relatively open problem, as most solutions use either extremely simplified proxy geometries [GO16] or resort to complex, accurate off-line solutions [LJF16]. One of the main difficulties arising in this context is the requirement of accurate yet fast computation of the variables involved in the motion dynamics, taking into account all possible situations such as, among other things, water leaking and moving inside the buoyant object. We will refer to fluid, liquid and water as synonyms, as the ideas presented here are adequate to represent buoyancy in other fluids.

We propose a real-time algorithm for an approximate yet realistic computation of the two-way coupling problem. There are recent satisfactory solutions for the solid-to-fluid part of the problem [JSMF*18], hence we will focus on the fluid-to-solid interaction part. The key concept behind our approach is that, with a reasonable parameterization, the textures traditionally used for texture mapping can be readily used to store and compute all needed quantities for the physical simulation. The main contributions of this work can be summarized as follows.

- To the best of our knowledge, this is the first method able to compute the main meaningful quantities needed to describe the buoyancy movement of a 3D model (e.g. centre of gravity, moments of inertia and centre of buoyancy) in real time, see Figure 1. Our method is capable to deal with deformable or shape-varying objects, as it is able to compute the required quantities on the fly.
- We propose a new data structure and algorithm for representing dynamically the liquid inside a submerged 3D object, together with the mechanisms for computing how it enters and moves inside when the object changes position while submerged. This allows to simulate the floating behaviour of a submerged object in a realistic way, which is, to the best of our knowledge, not possible until now in a general way.

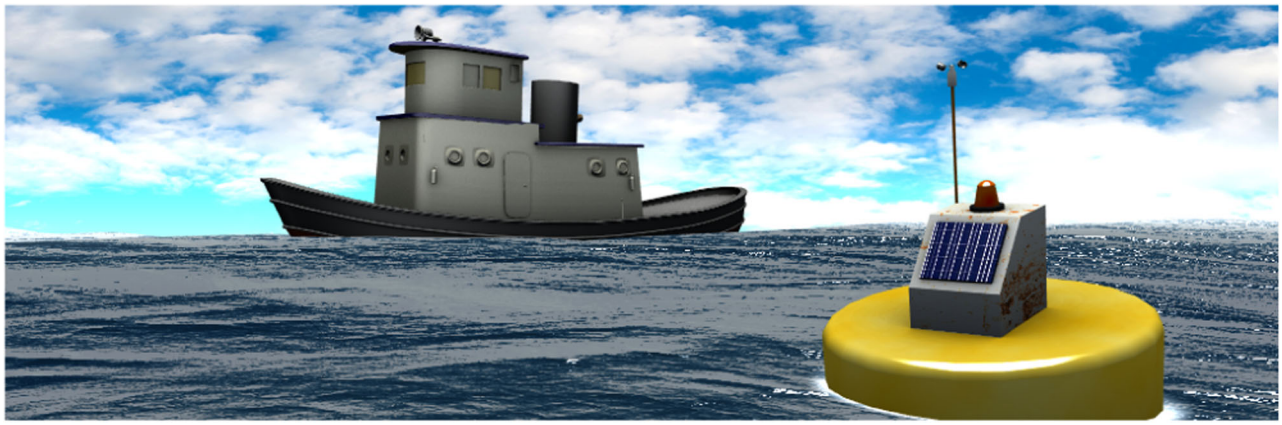


Figure 1: *Simulated boat and buoy in complex dynamic conditions. Our approach leverages GPU power to compute in real time the main physical magnitudes necessary to provide realistic fluid-to-solid coupling. Our technique is also able to model water leaks inside, and the resulting buoyancy behaviour, as is the case with sinking objects.*

- The algorithm does not require manual processing of the input models, as it uses the parameterizations usually employed for texturing objects in the artistic stages of the scene creation process.

2. Previous Work

The effect of liquids on solid objects partially or totally submerged has been a research topic in Computer Graphics from its very beginning [Bri15], always with the objective of achieving visual realism. Offline methods proposed in the literature provide accurate solutions, though not suitable for interactive applications. In this latter context, the proposed methods usually employ simple proxy geometries [GO16, Gou19], which often lack realism.

2.1. Offline methods

[CMT04] proposed a method for solving the two-way coupling by modelling rigid bodies as they were made of fluid. The rigidity of the objects is established by applying constraints on the velocity field. [GSLF05] provided a model for solving solid-water coupling, valid for infinitesimal deformable thin surfaces like cloth or rigid concave shells, using particles. Later, [BBB07] proposed a variational approach that allows robust and accurate solutions on relatively coarse Cartesian grids, restating the classical pressure projection step as a kinetic energy minimization. This idea is roughly similar to other approaches to rigid body contact, where a robust coupling between fluid and arbitrary solid simulations always yields a well-posed symmetric positive semi-definite linear system that can be solved with well-known numerical algorithms. [RMSG*08] presented an algorithm for the accurate computation of two-way coupling of fluids to rigid and deformable solids and shells. Later, [RMEF09] developed an accurate and direct method to compute the tangential velocities for solid-fluid coupling.

[AIA*12] developed a two-way coupling method of Smoothed-particle hydrodynamics (SPH) fluids and arbitrary rigid objects based on hydrodynamic forces. This momentum-conserving method is based on taking samples on the surface of rigid bodies with boundary particles that interact with the fluid. [GB13] presented a method based on the combination of unilateral incompressibility, mass-full FLIP, and blurred boundaries, which proved to be well-suited to the animation of large-scale fluids. In a later contribution, [GKSB13] proposed a model-reduced simulation schema based on basis enrichment for two-way solid–fluid coupling, effectively combining data-driven or artistically derived bases with more general analytic bases derived from Laplacian eigenfunctions. [YSZH13] proposed an efficient framework, based on a unified particle model for fluid–solid coupling, including both rigid and elastic bodies. Their technique applies different coupling schemes for fluid–solid and solid–solid coupling, respectively.

[DHB*16] presented a technique for computing the surface of liquids taking into account scale effects such as surface tension, but their interaction with solid objects is limited to splashing liquids only. [AVW*15] showed a technique for modelling thin liquid films over arbitrary shape surfaces. [TLK16] presented an Eulerian method for solid–fluid coupling. Simultaneously, [LJF16] presented a two-way coupling framework that couples incompressible fluids to reduced deformable bodies. [CMT*16] presented a method to compute wave accelerations by using a dispersion kernel, which uses a spatially variant filter provided by pyramid kernels that compensate for low-frequency truncation errors, and a shadowed convolution operation that accounts for obstacle interactions by modulating the dispersion kernel. [ANZS18] presented an extended partitioned method for two-way solid–fluid coupling, where the fluid and solid solvers are treated as black boxes with limited, exposed interfaces, which facilitates modularity and code reuse. On the other hand, [WW16] presented a design method for the *fabrication* of floating objects. In this proposal, the user chooses the desired orientation and waterline, and then an optimization procedure is used to compute the internal required configurations. When printed in 3D, the final objects float in the desired positions. From another point of view, there are several studies that provide bio-mechanical computational models for articulated bodies moving through fluids [KWC*10, SLST14]. The interaction of water absorbed by some

materials has also been extensively studied, proposing models to describe the change in appearance and to simulate the liquid flow inside the geometry [PC13, LAD08, FBGZ18]. [ZB17] presented an algorithm to simulate two-way coupling between fluid and deformable solids using a cut-cell discretization method in order to calculate boundary conditions at the solid-fluid interface. [TL19] showed a grid-based fluid solver for viscous fluids and solid objects.

None of these methods was designed with real-time, interactive applications in mind, but to be integrated in more accurate, but complex solvers for off-line computations. Instead, the ideas presented in our proposal are intended to be accurate, yet fully usable in real time in interactive environments, such as video-games and VR applications.

2.2. Interactive methods

As already mentioned, in the game industry the use of proxy geometries for water-to-solid interaction is commonplace [GO16, Gou19]. On the other hand, solid-to-water coupling has been extensively studied in the Computer Graphics literature. One example is the work by 2007, who presented Wave Particles, which offers a simple, fast and unconditionally stable approach to wave simulation. The model converts wave particles into a height field surface, which is warped horizontally to account for local wave-induced flow. As a direct extension of that seminal work, [JSMF*18] presented Water Surface Wavelets, a technique for modelling waves using a wavelet transformation that discretizes the liquid motion in terms of amplitude-like functions that vary over space, frequency and direction. This overcomes some limitations of the CFL condition and the Nyquist limit, and allows to simulate highly detailed water waves at very large visual resolutions. Among other simple extensions, the paper presented two-way solid–fluid coupling from solid obstacles like boats and buoys by emitting waves using circular shapes as initial sources. The fluid-to-solid interaction is computed by means of an integration over a simple proxy surface. [MMCK14] developed a method that unifies the dynamics between solids and fluids by sampling the rigid geometry with particles. Although these solutions are intended for real-time applications, the fluid-to-solid coupling in all cases is handled in a generic, proxy-based way.

[WP12] proposed a method for describing the motion of rigid bodies underwater replacing the inertia tensor by the Kirchoff tensor. This method works accurately for completely submerged bodies where the complete surface is in contact with water. [TMFSG07] proposed a method for simulating realistic breaking waves at the shoreline. As already mentioned, our aim here is to go beyond these simple couplings by using the actual geometry and properties of the objects themselves to perform the calculations. Also, we incorporate a simple, yet effective model of the (possibly) dynamic interior of the object, achieving effects which, to the best of our knowledge, were not possible until now.

3. A Comprehensive Buoyancy Model

Our method computes efficiently all the variables involved within buoyancy and solid–liquid interaction of partially submerged objects (including centre of mass, total mass, tensor of inertia, and many others, see Figure 2), considering two main cases. The first one is for homogeneous, dry solid objects interacting with water, whereas the second deals with dynamic behaviour, for instance, such as water entering a sinking boat or a wet plush Teddy bear that has fallen into a pool. For this, our implementation requires the object surface to have a watertight parametrized mesh, which could be done with any parameterization, such as those traditionally used for regular texture mapping. As these computations can be performed interactively, our technique can deal with deformable (or with dynamically variable shape) objects by treating them at each frame as if they were rigid, but recomputing their geometric properties on-the-fly. To this end, we use the information stored in a few specifically tailored textures, see next section.

For the case of non-homogeneous objects, such as a plush Teddy bear partially submerged in water which diffuses through its body, we additionally compute an ancillary structure called *water graph* (WG). This structure is a simple geometrical graph that is used to approximate the local distribution and propagation of water within the object taking into account its internal geometry. Each node of the WG represents the empty space inside the object, which eventually may be filled with liquid. The graph edges represent the interconnections among these empty spaces, modelled as variable radii pipes. The WG is flexible enough to model a wide variety of situations, from hollow objects, where liquid moves freely inside the object, to objects of permeable material, where liquid tends to spread uniformly following a diffusion equation. This allows for an accurate representation of the sinking behaviour in different situations, for instance, the Teddy bear already mentioned, or a sinking boat as seen in many AAA video-games [GO16]. The use of a non-regular data structure, that is the graph, instead of a simple voxelization, tends to reduce the number of elements necessary to accurately describe a given object. The two cases (homogeneous rigid or non-rigid, and non-homogeneous) can be easily superimposed by adding together their respective physical quantities, thus leading to a comprehensive and realistic water-to-solid interaction model that can be integrated into real-time applications. For the solid-to-water part of the simulation, we used the publicly available implementation from [JSMF*18]. This case is discussed in the following section.

3.1. Object physical properties

The method we propose here requires an evaluation of the physical magnitudes corresponding to the dry object being modelled: object mass, inertia tensor and drag coefficients. These are automatically computed, either as a pre-processing stage when the object is not deformed over time, or as part of the set of functions computed at each frame of the simulation.

Mass. The mass of any solid object bounded by a volume $\mathcal{V}$ can be simply calculated as

$$M = \iiint_{\mathcal{V}} \rho \, \mathrm{d}v, \tag{1}$$

that is, the integral of body density at each point over the volume of the body. In homogeneous objects, ρ is constant:

$$M = \iiint_{\mathcal{V}} \rho \, \mathrm{d}v = \rho \iiint_{\mathcal{V}} \mathrm{d}v. \tag{2}$$

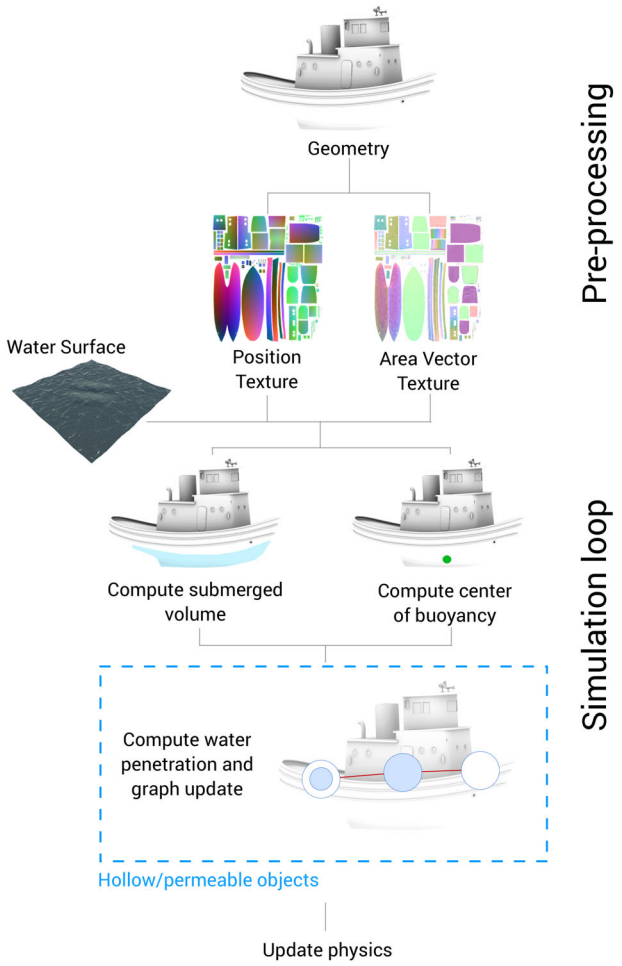


Figure 2: *Overview of our method: from the parameterized input model, we compute two additional textures (i.e. the position and the area-vector textures), and we use them to further compute the parameters of the physical model (e.g. submerged volume, centre of buoyancy, and inertia tensor). Finally, we compute the penetration of water into the geometry of the model, and use an ancillary structure, the water graph, to obtain the relevant physical magnitudes.*

By means of the *Divergence Theorem*, this last expression can be written as an integral over the surface of the object [BBWSH17]. To this end, we identify an arbitrary vector field, s.t. $\nabla \cdot \mathbf{r} = 1$, so we chose $\mathbf{r} = (0, y, 0)$ for our case (in a coordinate system where y points upwards, as in OpenGL and Vulkan), resulting in

$$M = \rho \int\int\int_{\mathcal{V}} \nabla \cdot \mathbf{r} dv = \rho \oiint_{\partial\mathcal{V}} \mathbf{n} \cdot \mathbf{r} ds. \tag{3}$$

If the surface of the object has an adequate parameterization, this integral can be approximated as the sum over the texels of a texture:

$$M = \rho \sum_{\text{texels } i} a_i c_i y_i = \rho \sum_{\text{texels } i} \mathbf{A}_i^y y_i, \tag{4}$$

where $c_i = \mathbf{n_i} \cdot (0, 1, 0)$, y_i is the y world coordinate of the texel and n_i is the unit normal stored for the nth texel, a_i is the texel area and $\mathbf{A}_i^y$ is the texel area vector *projected* in the y direction (so it is a scalar quantity), with $\cdot$ the dot product. We call this texture the *Area Vector* Texture (AVT) and each texel i stores $\mathbf{A}_i = a_i \mathbf{n}_i$. The area projected in any direction $\mathbf{d}$ can be simply computed as $\mathbf{A}_i \cdot \mathbf{d}$.

When the density ρ cannot be considered constant (e.g. inside a hollow boat), we can use a modified per-texel density $\tilde{\rho}_i$ which represents an average of the changing density. However, if the object is not deformable, the exact values could be computed in a preprocessing stage with any method of choice. The approximation presented here can be used only when no extreme accuracy is required, or when approximate real-time computations are adequate. Otherwise, in specific cases like with the hollow boat, we could use a specifically tailored solution, such as using a mass concentrated at the object boundary. In this latter case, we can replace the above equation with

$$M = \rho \sum_{\text{texels } i} \mathbf{A}_i^y t_i, \tag{5}$$

where t_i is the object thickness associated with the ith texel, which can be defined as a per-texel constant in a 2D texture, or by means of a specific mathematical formula. In any case, it is important to note that the mass of the dry object remains constant during the simulation and therefore can be pre-computed.

Centre of mass. The above calculation can be repeated for the object's centre of mass, resulting in

$$\mathbf{C} = \rho \oiint_{\partial\mathcal{V}} \mathbf{n}^T \mathbf{T} \mathrm{d}s, \tag{6}$$

where

$$\mathbf{T} = \begin{bmatrix} x^2/2 & 0 & 0 \\ 0 & y^2/2 & 0 \\ 0 & 0 & z^2/2 \end{bmatrix}. \tag{7}$$

This step is followed by a discrete summation over the texture texels that mimics the previous development.

Inertia tensor. Again, following the same reasoning, for the inertia moments of the object, we obtain the same expression as before, $\mathbf{I} = \rho \oiint_{\partial\mathcal{V}} \mathbf{n}^T \mathbf{T} ds$, but this time T, according to [BBWSH17], is given by

$$\mathbf{T} = \left[\begin{array}{ccc|ccc} x^2y/2 & 0 & 0 & x^3/3 & 0 & 0 \\ 0 & y^2z/2 & 0 & 0 & y^3/3 & 0 \\ 0 & 0 & z^2x/2 & 0 & 0 & z^3/3 \end{array}\right]. \tag{8}$$

3.2. Water-to-object coupling

When an object is submerged in water, its interaction depends on the exact water surface in contact with the object. For the method we present here, the water surface is computed according to any specific method used to model the water behaviour, which can be performed by a simple parameterized function, as commonly done in videogames [GO16], by managing the time steps of a fluid simulation

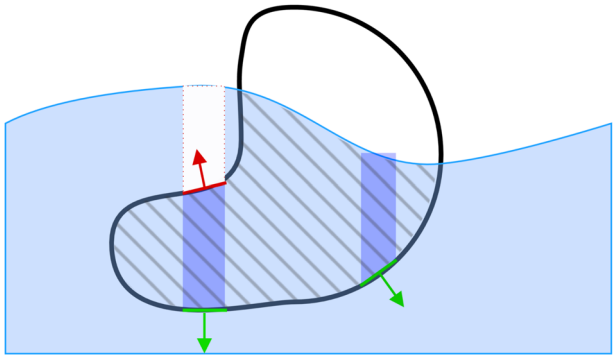


Figure 3: *Submerged volume computation (striped region): Texels whose normal vector has negative vertical component (green texels) contribute to the submerged total volume. On the other hand, texels with normal vector with positive vertical component (red texel) subtract volume from the total.*

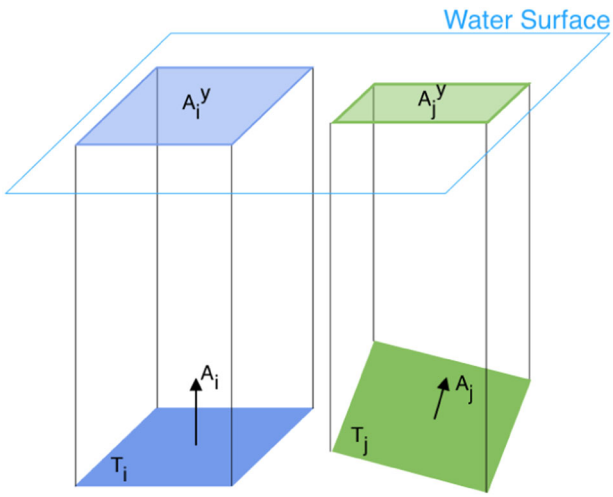


Figure 4: *The volume is computed as the product of the height and projected area A_i^y, which in turn is the product of texel total area and the cosine of the angle between texel normal and vertical axis.*

method over 2D or 3D grids, by Gerstner waves, or even by simpler analytic functions. The only requirement of our algorithm is to be able to retrieve a water level f for any position on the submerged object surface. In our implementation, we use a simple analytic form, plus the technique (and code) by Jeschke *et al.* [JSMF*18].

Submerged volume computation. In order to compute the portion of submerged volume, we apply the following expression, as illustrated in Figure 3,

$$S_V = \iiint_{\mathcal{V}_{\text{sub}}} \mathrm{d}v, = \oiint_{\partial\mathcal{V}_{\text{sub}}} \mathbf{n} \cdot \mathbf{r}\mathrm{d}s, \tag{9}$$

where we have to integrate over the submerged volume $\mathcal{V}_{\text{sub}}$ only, instead of the full object volume $\mathcal{V}$.

Unlike Equation (4), in this case the integration limits should take into account the water surface in order to compute the submerged volume correctly.

It is easy to note that the submerged volume $\mathcal{V}_{\text{sub}}$ is enclosed between the liquid level and the surface of the object located below it, thus $\mathcal{V}_{\text{sub}}$ can be computed integrating between these two limits. This process is depicted in the left liquid column in Figure 3, where the volume between the water surface and the red texel would be subtracted from the water volume between the green texel and the water surface, resulting in the correct volume being computed. Also, observe that we use the AVT to correctly project the texel area in the direction of the y axis (i.e. water coordinates). Figure 4 shows two different texel configurations, whose volume computations take into account their respective orientations. At this point, it is important to remark that the surface normals, encoded in the AVT, already contain the upper/lower surface information, so we could directly compute this as

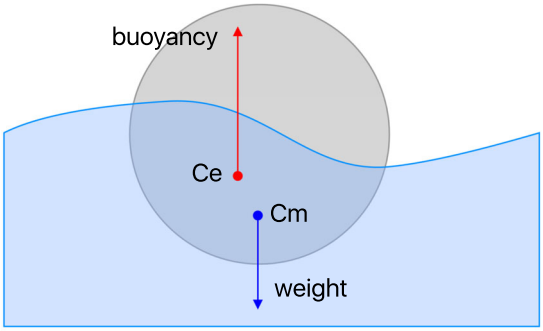
$$S_V = \sum_{\text{texels}_{\text{sub}}\, i} -A_i^y (f_i - y_i), \tag{10}$$

where f_i is the fluid's surface height at texel position; A_i^y is the vertical component of the area vector corresponding to the texel ($A_i^y = A_i \cdot (0, 1, 0)$), with A_i the texel area times its unit normal, stored in the AVT; (x_i, y_i, z_i) is the texel position in world space, and $\text{texels}_{\text{sub}}$ are all the texels that are below the liquid surface. To test this latter situation, we use an additional texture, called *position* texture (PT), which stores the 3D texel positions in object coordinates. The PT computing process consists of taking each of the faces of the object mesh and transforming them into texture space. It is important to note that the UV mapping in the PT must be bijective. Otherwise, we would lose mapping information during the computation of its inverse function, and as a result it may be impossible to determine the world coordinate positions for some texels. Note that situations like the one depicted at the right liquid column in Figure 3 do not require special care, as the upper part of the object will be discarded by the test that checks if the texel is below/above the water surface.

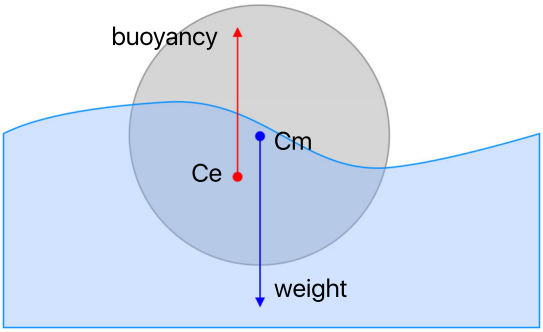
Buoyancy computation. The centre of buoyancy is defined as the centre of mass of the fluid volume displaced by an immersed body. The submerged body receives an upwards buoyancy force applied at the centre of buoyancy. The other force is gravitational pull due to the object weight, which acts downwards through the object's centre of gravity. The comparative strengths of these two forces determine whether the object sinks completely, or only partially until it reaches a stable buoying situation. The relative position of these centres determines whether this buoyancy is stable or not, see Figure 5. This relative position may also introduce a rotational torque τ upon the floating object.

The centre of buoyancy is computed in a similar way as before: given the PT and the fluid model, we compute it for the submerged texels as

$$C_{\text{sub}} = \rho_{\text{water}} \sum_{\text{texels}_{\text{sub}}\, i} \mathbf{n}_i^T \mathbf{T}_i a_i, \tag{11}$$



(a) Stable buoyancy.



(b) Unstable Buoyancy

Figure 5: *Forces acting on a submerged object. The relative position between the centres of buoyancy and mass determines whether a buoyancy condition is stable or not.*

where $\mathbf{r}_i$ is the ith texel position stored in the PT, $\mathbf{T}_i$ is as in Equation (7) for the ith texel and a_i is the corresponding texel area (stored in the AVT).

Drag. The drag force arises opposite to the translation at each point of the object itself. The expression for the drag magnitude by the liquid should be evaluated for all the surface of the object in contact with the liquid, that is the submerged part of the surface:

$$D_i = \frac{1}{2}\rho C_d \mathbf{A}_i^{\mathbf{V}_i} V_i^2, \tag{12}$$

where C_d is the drag coefficient, ρ is the fluid density, $\mathbf{A}_i^{\mathbf{V}} = \mathbf{A}_i \cdot \mathbf{V}/|V|$ is the texel area projected on to $\mathbf{V}_i$, which is the texel relative velocity in world coordinates with respect to the surrounding fluid. Here care must be taken, as this velocity is the sum of the object translational velocity plus the rotational velocity of the point where the ith texel is evaluated, and taking also into account the liquid velocity if necessary (e.g. if the object moves together with the liquid, then no translational drag should be considered). The translational velocity is constant for the whole object. To evaluate these local velocities, we need to know the exact positions of every texel, as provided by the PT.

On the other hand, the rotational velocity of each surface point depends on its distance $\mathbf{d}_i$ to the centre of mass. The velocity of each texel is calculated as $\omega \times \mathbf{d}_i$, where ω is the object angular velocity. The final force is computed as the sum over all the texels involved. In cases where the object density is not constant, or it is permeable and can contain liquid inside, a WG should be determined using an automatic algorithm, or it can be created manually, see Section 4. If the graph is constructed automatically, the number of created nodes depends on the segmentation algorithm. In cases where the graph is manually defined, artists choose the number of nodes according to their own criteria. Clearly, both methods could be used simultaneously.

Physical magnitudes update. The physical simulation of a rigid body involves the resolution of a number of simple first-order differential equations:

$$\frac{d}{dt}\mathbf{Y}(t) = \frac{d}{dt}\begin{bmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ \omega(t) * R(t) \\ F(t) \\ \tau(t) \end{bmatrix}, \tag{13}$$

where x stands for the object position, R is the current rotation that needs to be updated, P is the momentum, F is the total force on the object, τ is the current torque and L is the angular momentum [WB97]. These equations are solved with a straightforward Euler integration scheme, but more general applications would need to use a more careful implementation [ES04, Mil10, BF89]. However, for all our tests, our direct implementation proved to be both stable and accurate enough for real-time requirements.

4. Hollow/Permeable Objects

As already mentioned, we represent the water content in the interior of a solid object with a WG, which models the distribution and movement of liquid within the object. Figure 6 shows different water graphs for the Stanford Bunny model according to alternative mesh segmentation. Please note that different nodes have different sizes, representing their different capacities for liquid collection. When the liquid moves inside the object, flow is modelled as a liquid transfer among the nodes through the graph edges. The edges, in turn, model the amount of liquid that may flow between a pair of nodes at each unit of time. The topology of the graph reflects the description of the object's interior according to user semantics (as typical geometry meshes for real-time applications only describes object's the exterior). Thus, hand-made graphs allow artists to represent specific situations, for example, a sealed room in a damaged boat, or an object in which water is permeating only at specific locations. The following subsections explain all these elements in detail.

4.1. Automatic graph construction

First, we generate a segmentation of the object input triangulated mesh. The segmentation algorithm computes a Shape Diameter Function (SDF) for all facets and applies a graph-cut-based algorithm over these values [YL18], see Figure 6. Then, the algorithm computes, for each sub-mesh, the sphere centred at the sub-mesh centroid that minimizes the quadratic error between the sphere and the sub-mesh vertices. Finally, we connect two nodes with an edge if the respective sub-meshes are adjacent in the original model. Alternatively, as only a relatively sparse and small graph is needed, we also let the user manually provide the graph with the respective node capacities.

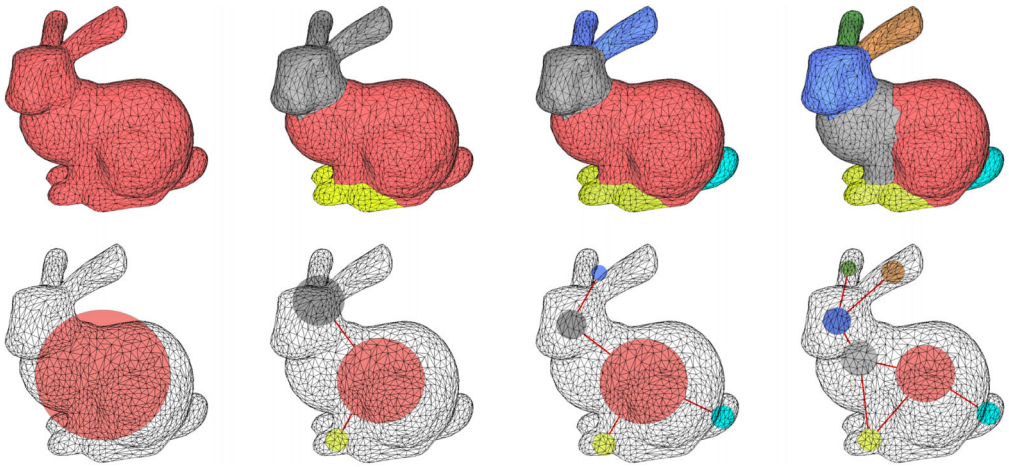


Figure 6: *Water Graph construction: mesh segmentation and node creation for different number of sub-meshes. Same colours refer to the same regions.*

4.2. Water graph update

Once the WG has been created and the node capacities have been set, we can evaluate the (coarse) dynamics of fluid inside the object. The first stage is to compute the amount of water that enters (or permeates) the object. This can be trivially accomplished by adding an arbitrary amount of water to any node in the graph. However, to increase realism and at the same time to let the user have finer control, in our implementation we define an extra *permeability texture* with the same parameterization as before, where the user defines the amount of water that enters at each point (i.e. each texel) of the object. For each texel the user specifies a constant permeability factor, which we multiply by the volume of water in contact with this area if the area is submerged, to get a per-texel water content. These quantities are accumulated in a single GPU pass, and distributed to the corresponding nodes, whose capacities are already known given the prior segmentation done before. The permeability texture can be updated at any time, thus allowing for completely dynamic situations to be simulated (e.g. a naval battle).

For the dynamics of fluid inside the object, at each computation step (e.g. each frame), we use a communicating vessels approximation, which has proven to be effective and practical in most situations. Each node can transfer liquid to any other node with which it is connected to, taking into account the respective node capacities, the edge maximum water flow capacity and the flow direction imposed by gravity. A special case may arise with water and some diffusive materials, as surface tension may pull upwards a certain amount of water.

A straightforward implementation of this idea may lead to a cumbersome algorithm. See, for instance, the situations depicted in Figure 7, where we have three nodes with equal capacity connected as shown, with different amounts of liquid each, and at different heights. The direct implementation of the water flow algorithm would lead to node B sending (some) water down to C, and in a later iteration, A sending water to B. If we think that, in the end, fluid particles are indistinguishable for our purposes, it would be far more practical if A could send water directly to C, avoiding the two-stage

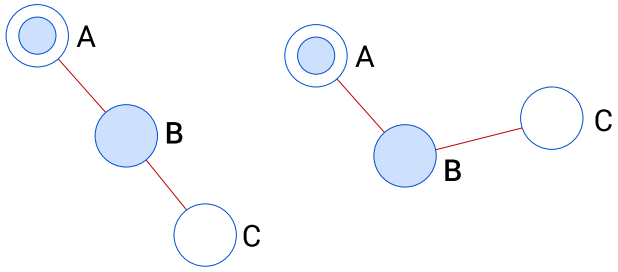


Figure 7: *Two simple graphs, each with three nodes of equal capacity: Node A has some liquid, node B is full while node C is empty. The nodes are connected by regular edges, and located as shown in the figures (upper means higher altitude).*

process described. Function *findDischargeNode*, explained below, is responsible for this implementation, taking into account that a lower node cannot send water upwards to any upper node, or pass through an upper node in its way to a lower node.

The whole process is controlled by an algorithm, which, at each frame, iteratively calls the method *findDischargeNode* for each node in the graph. This method is presented in pseudo-code in Algorithm 1, which, given an initial node n_s for which we want to find a node to transfer water to (called *discharge node*), recursively tries all nodes connected to the current one to find a suitable one. If during the discharge node search we found a node that is higher than the source node, the process must be stopped because that node is an energy barrier, thus water cannot flow through it. This means that liquid present in the source node cannot move forward in that path.

Repeating this process for each node every frame, and searching paths in a random way, ensure a feasible solution to the problem of liquid movement inside the object.

If a node still has some water capacity, it is selected and returned by the function. If no suitable node is found, the algorithm returns

Algorithm 1. *findDischargeNode* - Algorithm to find a node to discharge to

Input:
- n_s: The node whose discharge node we are looking for.
- n_c: The current node being evaluated.
- level: relative level of n_c with respect to n_s.
- $perm_{min}$: Minimal permeability between n_s and n_c.

Output: A discharge node for n_s.

```
if n_c.visited then
    return null;
end
n_c.visited = True
if level > 0 then
    return null; // Energy barrier
end
if (n_c ≠ n_s) && isNotFull(n_c) ) then
    return n_c;
end
for each node n_i adjacent to n_c do
    e = edgeBetweenNodes(n_c, n_i)
    Δ_l = n_i.pos.y − n_c.pos.y
    m = min(perm_min, e.permeability)
    n_d = findDischargeNode(n_s, n_i, level + Δ_l, m);
    if n_d ≠ null then
        return n_d
    end
end
```

the *null* node, meaning that, at the moment, the starting node cannot pass its fluid content to any other node in the graph.

Once a suitable node n_d (or set of nodes) is found, we use a method to actually transfer the corresponding quantity of water from the start node n_s to n_d, using Poiseuille's Law [Bat00], which states:

$$\Phi_{n_s \to n_d} = \frac{\pi \Delta_P r^4}{8 \mu L}, \tag{14}$$

where $\Phi_{n_s \to n_d}$ is the flow rate across a cylindrical pipe, Δp is the pressure drop, r the segment radius, L the path length between $n_s \to n_d$ and μ the liquid viscosity. In the case of a diffusion-like process, Δp should be replaced by the difference in concentration. The algorithm could be easily modified to account for multiple discharge nodes instead of a single one, by simply continuing the recursion and returning a list of suitable nodes instead of a single one. Then, a small algorithm to prioritize these nodes should be implemented, depending on their relative positions with respect to the starting one.

4.3. Computing the physical magnitudes of the graph

As before, it is necessary to compute the physical magnitudes (e.g. mass, centre of mass and tensor of inertia) of the graph, as they will be needed for the final composed system object plus graph. The mass of the graph is given by a discrete version of the previous expression:

$$M_G = \rho_f \sum_n v_n, \tag{15}$$

where the sum is over the graph nodes n, ρ_f is the fluid density and v_n is the occupied volume of the nth node.

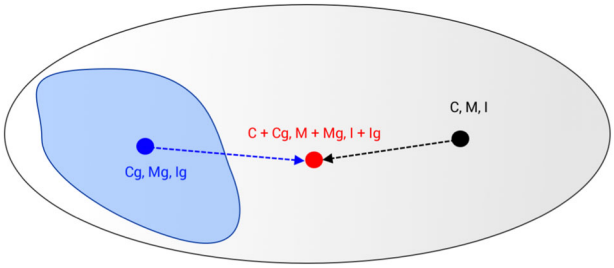


Figure 8: *Computation of the resultant centre of mass (C) and inertia tensor (I) of a non-homogeneous body (in grey) with absorbed liquid (in blue, with C_G and I_G). The liquid magnitudes are computed from the WG. (Please note that the plus sign does not denote the algebraic sum, as the centre of mass and inertia tensors must be composed together as described in the text.)*

The centre of mass of the fluid represented by the graph is

$$\mathbf{C_G} = \frac{\rho_f}{M_G} \sum_n \mathbf{r}_n v_n, \tag{16}$$

where $\mathbf{r}_n = (x_n, y_n, z_n)$ is a node position. Finally, the inertia tensor $\mathbf{I_G}$ is given by the discrete counterpart of the one used above:

$$\begin{aligned} \mathbf{I_G}^{xx} &= \sum (y_n^2 + z_n^2) m_n \quad \mathbf{I_G}^{yy} = \sum (x_n^2 + z_n^2) m_n \\ \mathbf{I_G}^{zz} &= \sum (x_n^2 + y_n^2) m_n \quad \mathbf{I_G}^{xy} = \mathbf{I_G}^{yx} = -\sum x_n y_n m_n \\ \mathbf{I_G}^{yz} &= \mathbf{I_G}^{zy} = -\sum y_n z_n m_n \quad \mathbf{I_G}^{xz} = \mathbf{I_G}^{zx} = -\sum x_n z_n m_n. \end{aligned} \tag{17}$$

4.4. Final Model Computation

To compute the effect of the different physical magnitudes for the submerged object and the water that has entered (e.g. as in a sinking boat), we simply have to combine them. The total mass of the object plus the graph can be simply computed as $M + M_G$; the combined centre of mass is a weighted sum of the previously computed centres of mass, $\mathbf{C_T} = \frac{M\mathbf{C} + M_G \mathbf{C_G}}{M + M_G}$; and the inertia tensor now is $\mathbf{I} + \mathbf{I_G}$. In the latter case, extra care must be taken, as both centres of inertia must be in the same coordinate system before

Table 1: *Computation times (in ms) for different* position *and* area *texture sizes. Impermeable objects (e.g. buoy simulation) do not have a* water graph *defined. On the other side, permeable objects (e.g. drowning ship) have a graph defined, thus the computation times are higher. Computation times neither depend on the number of vertices nor on the number of faces in the geometric mesh. Also, the sizes of the textures used by the method not necessarily should match the textures used for rendering.*

Computation times (ms)		
Texture size	Impermeable	Permeable
512	1.7	2.4
1024	2	3.1
2048	3	4.2

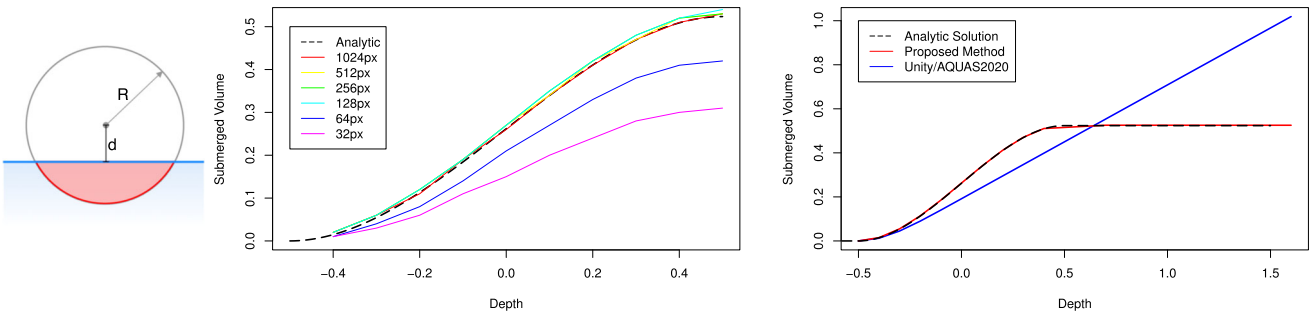


Figure 9: *Submerged volume of a sphere of radius $R = 0.5$ as a function of the immersion depth, from $d = -0.5$ (not submerged) to $d = 0.5$ (totally submerged). Water surface is at depth $d = 0$ (Left). Middle: Analytical value and approximations computed with our method for different AVT and PT resolutions (from 32×32 to 1024×1024 texels). Right: our results with texture size 1024×1024 compared with the analytical solution and the method proposed by the Unity commercial package AQUAS2020.*

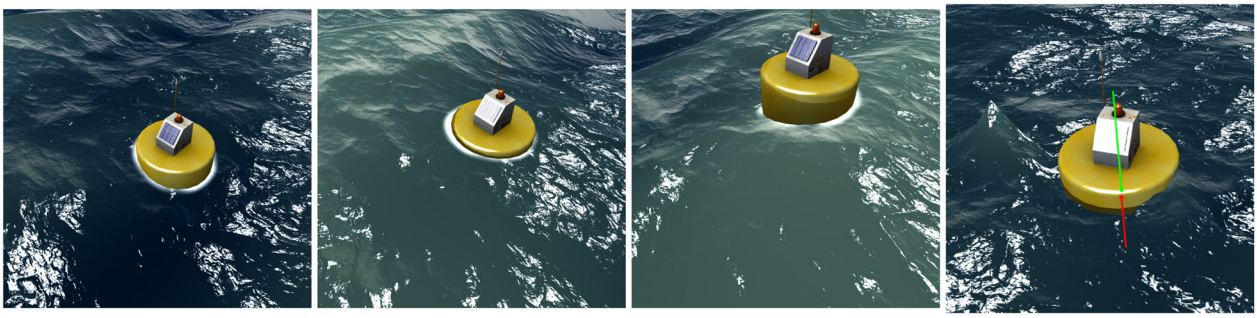


Figure 10: *Frames from an animation, showing the buoyancy dynamics simulation. The buoy is not permeable, thus there is no liquid inside the volume. Using texel positions, the wet portions of the object were rendered accordingly to generate more realistic results. The rightmost frame is generated by exaggerating the effect of the solid-to-fluid coupling technique used [JSMF*18]. It can be seen the perturbation produced by the buoy on the fluid around it.*

the summation can be performed. This can be easily performed by means of the parallel axis theorem [HV28, Abd17], and translating both moments of inertia to the axis that goes though the total centre of mass $\mathbf{C_T}$. Note that this is trivial for the graph, but the calculation for the submerged object is a bit more involved (see Figure 8).

5. Implementation and Results

We implemented our technique using GPGPU techniques based on the OpenCL and OpenGL APIs. Time measures were gathered on an Intel(R) Core(TM) i7 920 at 2.67GHz with a NVidia GeForce GTX 1080 video card. The reduction steps were implemented using a freely available OpenCL demo code provided by Apple [App18]. The accuracy of our method is shown in Figure 9, where we present the submerged volume of a sphere of radius $R = 0.5$ at different depths using multiple PT and AVT texture sizes. We can observe the typical trade-off between required memory and precision: For instance, a set of 128×128 textures can describe a simple volume, such as a sphere. On the rightmost subfigure, we compare our result using the largest texture size (1024×1024) with the analytical solution and the method used in the Unity [Uni19] commercial package called AQUAS2020 [Dom20], which is designed to model and simulate flat water and related effects, such as buoyancy. AQUAS2020 computes the buoyancy force as a resultant of many forces, one for each mesh triangle. The magnitude of each force is calculated as the volume displaced by the corresponding triangle. This process is only applied for triangles facing down, which explains the large error and the linear relation between displaced volume and depth when the sphere is totally submerged. Furthermore, the use of polygons to compute geometry measurements turns an efficient GPU implementation difficult and leads to costly processing times (e.g. a buoyancy simulation of 15k triangles Stanford Bunny takes 22 mS approximately). Apart from this, our texture-based approach also allows to easily implement an LoD technique, simply by changing texture sizes.

As an example of our method, the buoy shown in Figure 10 is simulated using the method proposed in this paper. Note that at the rightmost frame of this sequence, the effect of the solid-to-fluid coupling technique by [JSMF*18] was greatly exaggerated, for comparison purposes only. The same sequence is shown in Figure 11 rendered in orthographic projection and with the force vectors acting on the buoy overlapped to the renders. It is interesting to note how the buoyancy force changes over time according to the submerged geometry. The computation of this force uses the PT and AVT, which are computed directly from the geometry without supervision.

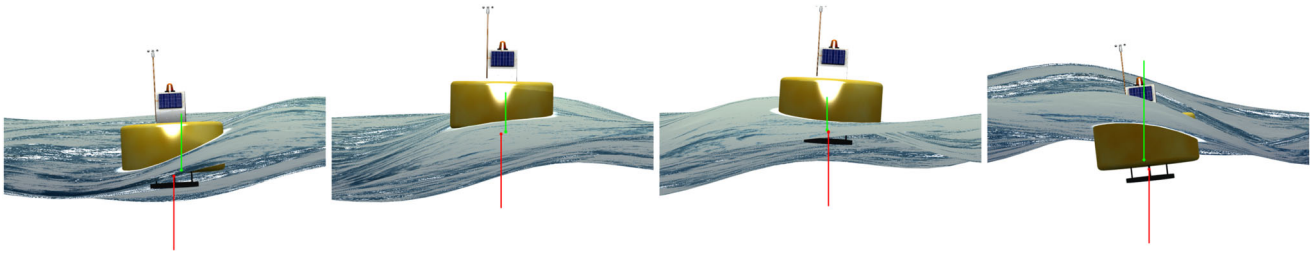


Figure 11: *Frames of the buoy simulation at different times, showing the weight and buoyant forces in orthographic projection. We can see the relative position and magnitude of the buoyancy force (green line) and the weight force (red line) according to the submerged portion of the model. Clearly, the weight force is constant because the object is not permeable, thus its mass does not change over wtime.*

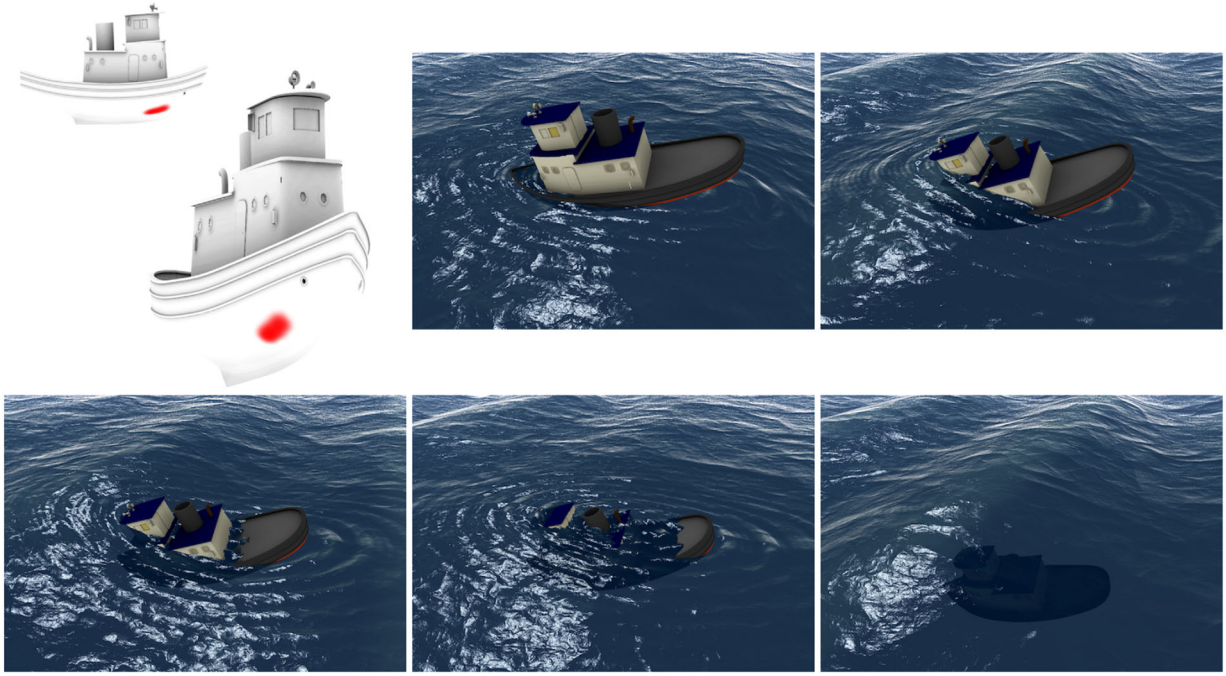


Figure 12: *Frames of the drowning ship simulation, the top-left figure shows the most permeable zone, both from the side and from the front of the ship.*

In Figure 12 several frames of a drowning ship are shown. The initial ship can be seen at Figure 1. The object permeability is defined in a per-texel basis simulating a damaged section, as shown in the top-left inset. The same object is shown in Figure 13 but with a different *Permeability Texture*: in this case, the damaged section is located at the boat stern. The compute times are shown in Table 1. It is important to emphasize that compute times do not depend on the geometry mesh defined by the object, as the method is completely texture-based.

In Figure 14, a set of frames of an unstable buoyancy simulation are shown. As the centre of buoyancy is located below the centre of mass, the object does not recover its initial rotational position. See also Figure 5(b) for a diagram of this effect. On the other hand, the presented method can also be used to simulate objects whose mass distribution changes over time. In these cases, the physical magnitudes needed for the simulation must be computed at each frame of the simulation, as they change continuously. In Figure 15, a ball with variable mass density is depicted. The mass density function is shown as a colour map over the ball, where red zones indicate higher mass density, whereas green zones indicate lower density levels. The distribution is also shown on each frame as insets at the bottom left corners. In this case, unlike the cases of the ship and buoy simulations where the mass density was constant over time, the magnitudes must be re-computed at each frame. This simulation shows that the computing times remain suitable for real-time applications even in cases where physical magnitudes cannot be pre-computed. Furthermore, our model is capable of working with deformable models if position and area textures are computed at each frame.

Figure 16 shows a plush object that tends to absorb liquid through its surface, and water propagation follows a *diffusion* profile. In this

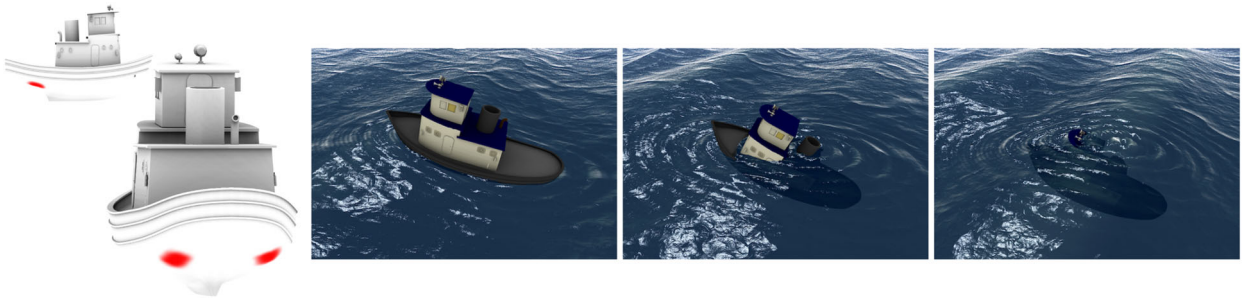


Figure 13: *Frames of the drowning ship simulation, the red spots of the mesh at the figure on the left show the permeable zone, shown both from the side and from the back of the ship.*

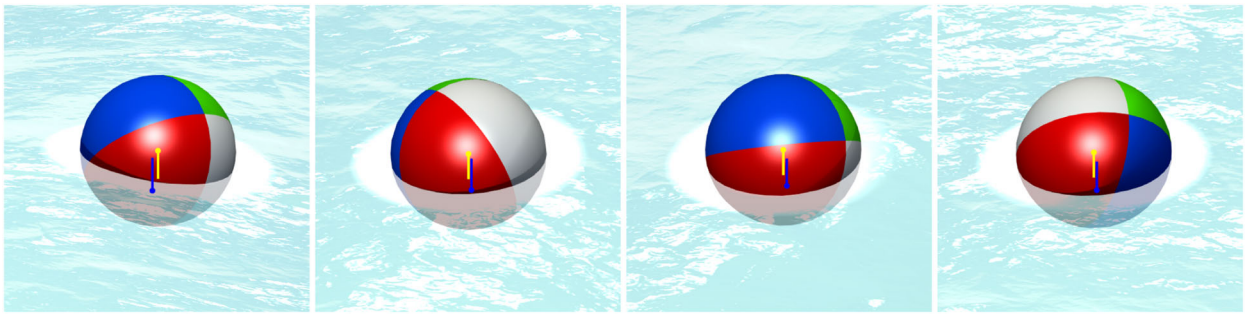


Figure 14: *Different frames of an unstable buoyancy simulation. The ball has a uniform mass density over its surface, thus its centre of mass is located at the geometric centre, but the centre of buoyancy is always located below the centre of mass, generating an unstable movement (i.e. the object does not tend to recover its initial rotation, as opposed to a boat, where the mass is concentrated at its bottom). Compare with the situation in Figure 15.*

case, several water graphs are generated automatically from different mesh segmentation, as previously explained and shown in Figure 6. Figure 17 shows a few frames of the inflatable balloon demo. In this example, a deformable mesh was used to model the geometry changes between a deflated and an inflated balloon. As the volume occupied by the object increases when it is filled with hot air, the lift force also increases.

6. Discussion and Further Work

The computation of all physical values (i.e. mass, centre of mass and inertia tensor) is intimately related, and cannot be assigned manually unless the artist knows the behaviour of the model well. Otherwise the resulting dynamics will be inadequate and, in the end, look unrealistic. The approximations described here were developed with the idea of being used in the context of real-time applications, such as simulators or video-games. If more precision is required, all these values would need to be computed with a more accurate mechanism (e.g. Monte Carlo integration).

If Level-of-Detail techniques are used, the method presented here easily accommodates to this situation: simply by replacing the PT by a lower resolution version would suffice to reduce the amount of computation, which grows linearly with texture resolution. This is empirically confirmed in Table 1. In extreme cases, all the computations could be performed on the polygons themselves, without any textures, moving the cost from the pixel to the geometry pipeline, much in the same way as Mesh Colour Textures work [Yuk17].

Our technique depends on a *reasonable* parameterization and texture resolution. This is not a limitation in itself, as every model is processed and provided a parameterization and texture in modern graphics pipelines. Also, PTs are in a sense similar to Geometry Images (GI) [GGH02], but it must be taken into account that GI are intended as a connectivity-free resampling of an arbitrary shape into a regular 2D grid, requiring a reparameterization of the input object. Instead, PTs simply record, for each texel, its position when projected to 3D space, requiring no reparameterization efforts or any further processing.

Although our technique is presented using a height-field water representation, it could be easily used with a particle-based representation. One alternative could be to use marching cubes to recover the fluid surface, while another possibility could be re-defining the texel depth computation using the particles of the simulation themselves.

The method was developed with the intention of being fully automatic. However, the user can easily intervene to manually edit or define certain parameters (e.g. the permeability texture and the mass distribution along the object or the liquid graph) with the aim to achieve some desired effects, such a specific behaviour. It is

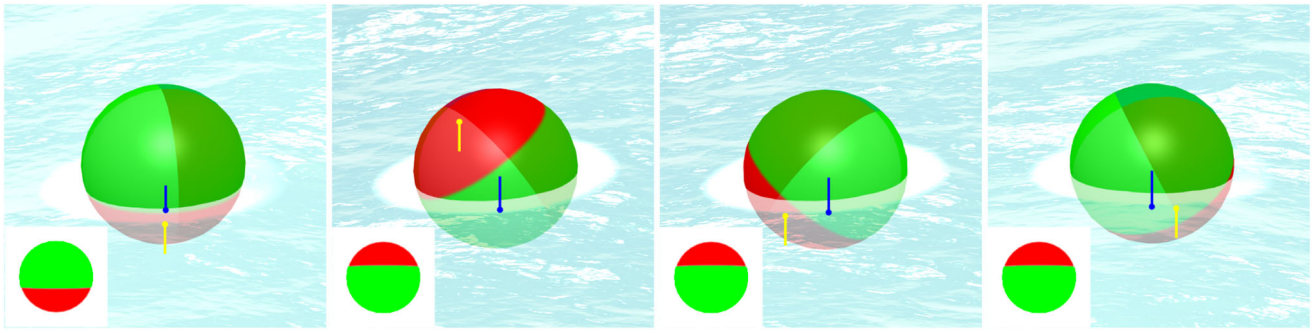


Figure 15: *Frames of a time-dependent mass distribution simulation. Our method can also be used with time-variable mass objects (e.g. a small boat with people moving on it). Textures used as data structures allow the method to compute physical magnitudes in real time (see the insets that show the mass distributions). In the figure, the colour map shows the mass density function at each frame: Red zones show higher mass density and green zones, lower. The yellow line is the weight force and the blue line the buoyant force. In this simulation, the total mass, centre of mass and inertia tensor were computed at each frame using square textures of 1024 × 1024 texels. Average compute time is 3.2 ms.*

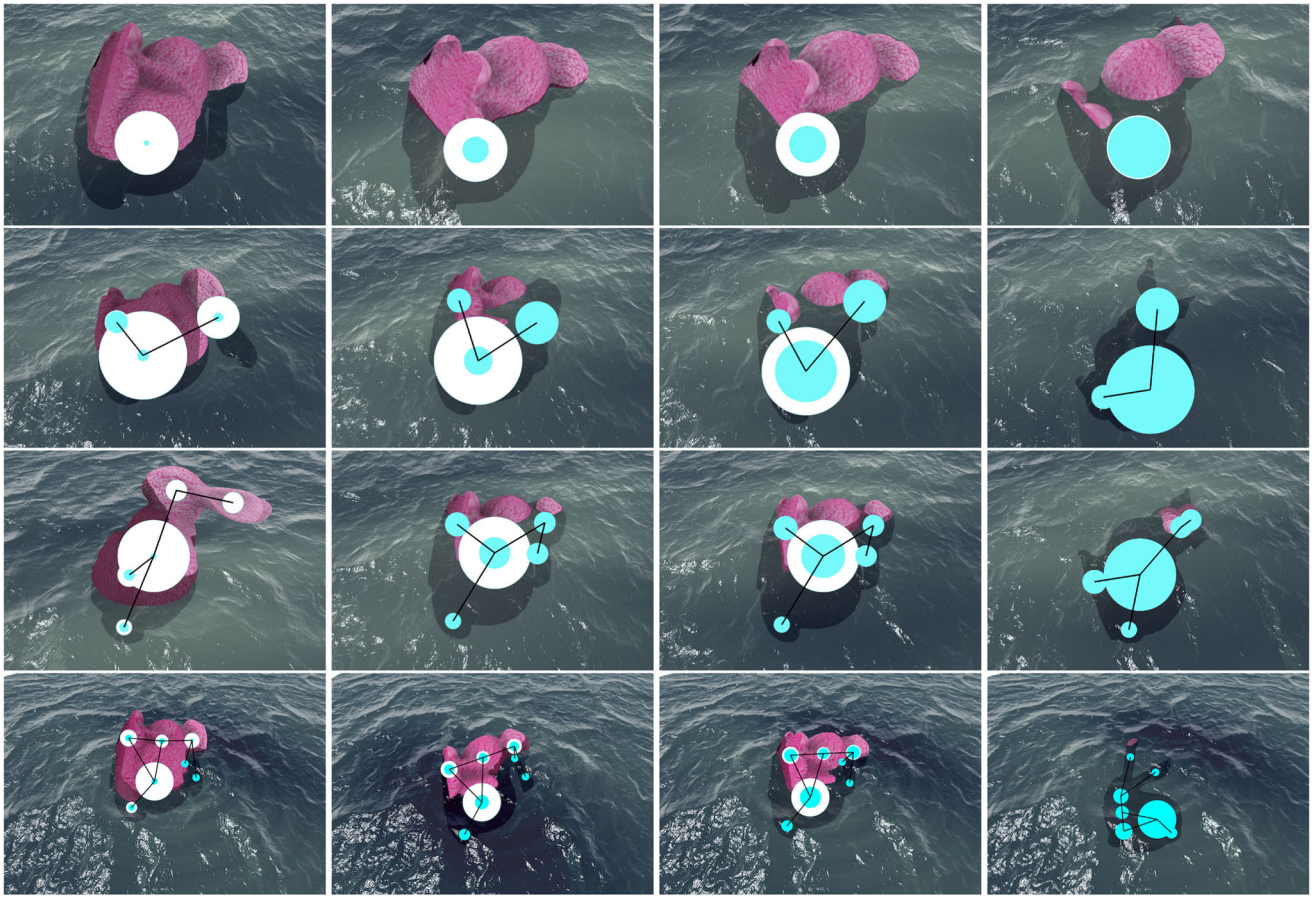


Figure 16: *Frames of a plush bunny simulation using the* water graphs *automatically generated, as shown in Figure 6, with a diffusion profile.*

important to note that the *permeability texture* can be computed interactively, that is allowing dynamic destruction of objects.

With respect to the model processing, it should be noted that the model should be rendered at its outer layer of polygons, so care must be taken not to send the interior polygons to be processed by the algorithm. However, for interior polygons, the graph can be used not only for the liquid that enters the object, but also to model any interior feature that the user might be interested in taking into account in the calculations.

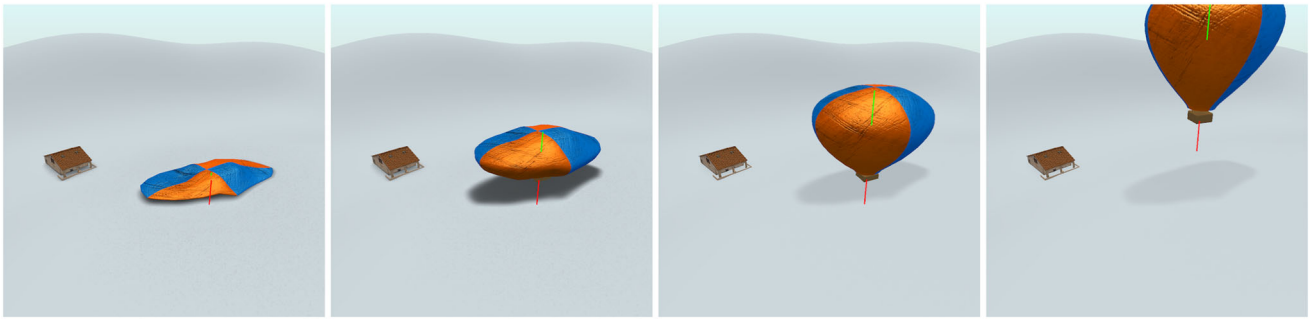


Figure 17: *Frames of a inflatable air balloon simulation showing how the method works for an animated mesh. In this case, the object is submerged in a gaseous medium (i.e. air). The red line shows the* weight *force, whereas the green line shows the* lift *force.*

Our method is not free of drawbacks, though. Perhaps the most important is that computations in texture space may not reflect the actual properties of the object, as they can only compute values at the outer layer of the object, and assume that the interior is homogeneous, which is not always true. One simple and effective way to solve this is to use our graph-based approximation to compute the lack of homogeneity we have not considered so far. Complex density distributions may arise inside an object, requiring more involved graphs which could be difficult to capture. This can be solved with an automatic computation of the graph shape that takes into account the mass distribution within the object in addition to the exterior shape, which is left as future work. For these cases, the only option at the moment is to manually create the associated graph. From an implementation point of view, the mass distribution can be as arbitrary as needed, because if it is constant the different quantities can be pre-computed and the method would work without changes.

Our method also requires closed surfaces. In the case of concave objects (or hollow objects without a side), such as a bucket partially submerged in water, the algorithm as presented would fail because, for the bottom part, our algorithm would only consider two texels, one at each side of the bucket bottom, and not take into account the actual volume of the submerged part. However, this problem could be easily solved by adding an external, invisible polygon that closes the object at the upper side. This would require some extra work. The automatic detection of concave parts will be left as future work. Finally, the graph-based model is practical, efficient and intuitive, clearly useful for real-time applications not concerned with accuracy. However, a more physically realistic model could probably be derived. We think this is a promising venue for further research, which is left as future work.

7. Conclusions

We have presented a technique that allows to simulate interactive and realistic buoyancy behaviours for objects submerged in liquids. This technique is based on two main components: an efficient GPU-based algorithm for the computation in real time of the physical quantities involved in the object motion (e.g. mass, centre of mass, and tensor of inertia), and a graph-based algorithm and data structure for the simulation of permeable objects or with complex, mobile interiors. Together they allow interactive and realistic simulations of both rigid and deformable objects, both permeable and impermeable, and in variable conditions/situations. To the best of our knowledge, it is the first time this is possible in the literature, opening interesting applications for video-games, virtual reality and other interactive applications.

Acknowledgements

We would like to thank the anonymous reviewers for their help proofreading this paper. This work was partially funded by project TIN2017-88515-C2-2-R from Ministerio de Ciencia, Innovación y Universidades, Spain, project 24/K083 from the SECyT, Universidad Nacional del Sur, Argentina, and a scholarship by the National Science and Technology Council of Argentina (CONICET).

References

[Abd17] Abdulghany A. R.: Generalization of parallel axis theorem for rotational inertia. *American Journal of Physics 85*, 10 (2017), 791–795.

[AIA*12] Akinci N., Ihmsen M., Akinci G., Solenthaler B., Teschner M.: Versatile rigid-fluid coupling for incompressible SPH. *ACM Trans. Graph. 31*, 4 (July 2012), 62:1–62:8.

[ANZS18] Akbay M., Nobles N., Zordan V., Shinar T.: An extended partitioned method for conservative solid-fluid coupling. *ACM Transactions on Graphics 37*, 4 (July 2018), 86:1–86:12.

[App18] Apple Developer: Apple developer opencl demo, 2018.

[AVW*15] Azencot O., Vantzos O., Wardetzky M., Rumpf M., Ben-Chen M.: Functional thin films on surfaces. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2015), ACM Press, pp. 137–146.

[Bat00] Batchelor G. K.: *An Introduction to Fluid Dynamics*. Cambridge University Press, Cambridge, 2000.

[BBB07] Batty C., Bertails F., Bridson R.: A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics 26*, 3 (2007), 100.

[BBWSH17] Bächer M., Bickel B., Whiting E., Sorkine-Hornung O.: Spin-it: Optimizing moment of inertia for spinnable objects. *Communications of the ACM 60*, 8 (July 2017), 92–99.

[BF89] Burden R. L., Faires J. D.: *Numerical Analysis*. PWS-Kent Publishing Company, Boston, MA, 1989.

[Bri15] Bridson R.: *Fluid Simulation for Computer Graphics*. Taylor & Francis, Milton, 2015.

[CL95] Chen J. X., Lobo N. d. V.: Toward interactive-rate simulation of fluids with moving obstacles using Navier-Stokes equations. *Graphical Models and Image Processing 57*, 2 (March 1995), 107–116.

[CMT04] Carlson M., Mucha P. J., Turk G.: Rigid fluid: Animating the interplay between rigid bodies and fluid. In *ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM Press, pp. 377–384.

[CMT*16] Canabal J. A., Miraut D., Thuerey N., Kim T., Portilla J., Otaduy M. A.: Dispersion kernels for water wave simulation. *ACM Transactions on Graphics 35*, 6 (November 2016), 202:1–202:10.

[DHB*16] Da F., Hahn D., Batty C., Wojtan C., Grinspun E.: Surface-only liquids. *ACM Transactions on Graphics 35*, 4 (July 2016), 78:1–78:12.

[Dom20] Domatic Games: Aquas 2020, 2020.

[ES04] Eberly D., Shoemake K.: *Game Physics*. Interactive 3D technology series. Taylor & Francis, Milton, 2004.

[FBGZ18] Fei Y. R., Batty C., Grinspun E., Zheng C.: A multi-scale model for simulating liquid-fabric interactions. *ACM Transactions on Graphics 37*, 4 (July 2018), 51:1–51:16.

[GB13] Gerszewski D., Bargteil A. W.: Physics-based animation of large-scale splashing liquids. *ACM Transactions on Graphics 32*, 6 (November 2013), 185:1–185:6.

[GGH02] Gu X., Gortler S. J., Hoppe H.: Geometry images. *ACM Trans. Graph. 21*, 3 (July 2002), 355–361.

[GKSB13] Gerszewski D., Kavan L., Sloan P.-P., Bargteil A. W.: Enhancements to model-reduced fluid simulation. In *Proceedings of Motion on Games* (New York, NY, USA, 2013), ACM Press, pp. 201:223–201:228.

[GO16] Gonzalez-Ochoa C.: Advances in real-time rendering in games: Rendering rapids in uncharted 4. In *SIGGRAPH'16: ACM SIGGRAPH 2016 Courses* (New York, NY, USA, 2016), ACM Press.

[Gou19] Gourlay M. J.: Fluid simulation for video games. https://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-9, 2019. [Accessed 10 January 2019].

[GSLF05] Guendelman E., Selle A., Losasso F., Fedkiw R.: Coupling water and smoke to thin deformable and rigid shells. *ACM Transactions on Graphics 24*, 3 (July 2005), 973–981.

[HV28] Haas A. E., Verschoyle T.: *Introduction to Theoretical Physics*. Constable & Company Ltd, London, 1928.

[JSMF*18] Jeschke S., Skřivan T., Müller-Fischer M., Chentanez N., Macklin M., Wojtan C.: Water surface wavelets. *ACM Transactions on Graphics 37*, 4 (July 2018), 94:1–94:13.

[KWC*10] Kwatra N., Wojtan C., Carlson M., Essa I. A., Mucha P. J., Turk G.: Fluid simulation with articulated bodies. *IEEE Transactions on Visualization and Computer Graphics 16*, 1 (January 2010), 70–80.

[LAD08] Lenaerts T., Adams B., Dutré P.: Porous flow in particle-based fluid simulations. *ACM Transactions on Graphics 27*, 3 (August 2008), 49:1–49:8.

[LJF16] Lu W., Jin N., Fedkiw R.: Two-way coupling of fluids to reduced deformable bodies. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Goslar Germany, Germany, 2016), Eurographics Association, pp. 67–76.

[Mil10] Millington I.: *Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for Your Game*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 2010.

[MMCK14] Macklin M., Müller M., Chentanez N., Kim T.-Y.: Unified particle physics for real-time applications. *ACM Transactions on Graphics 33*, 4 (July 2014), 153:1–153:12.

[PC13] Patkar S., Chaudhuri P.: Wetting of porous solids. *IEEE Transactions on Visualization and Computer Graphics 19*, 9 (September 2013), 1592–1604.

[RMEF09] Robinson-Mosher A., English R. E., Fedkiw R.: Accurate tangential velocities for solid fluid coupling. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), ACM Press, pp. 227–236.

[RMSG*08] Robinson-Mosher A., Shinar T., Gretarsson J., Su J., Fedkiw R.: Two-way coupling of fluids to rigid and deformable solids and shells. *ACM Transactions on Graphics 27*, 3 (August 2008), 46:1–46:9.

[SLST14] Si W., Lee S.-H., Sifakis E., Terzopoulos D.: Realistic biomechanical simulation and control of human swimming. *ACM Transactions on Graphics 34*, 1 (December 2014), 10:1–10:15.

[TL19] Takahashi T., Lin M. C.: A geometrically consistent viscous fluid solver with two-way fluid-solid coupling. *Computer Graphics Forum 38*, 2 (2019), 49–58.

[TLK16] Teng Y., Levin D. I. W., Kim T.: Eulerian solid-fluid coupling. *ACM Transactions on Graphics 35*, 6 (November 2016), 200:1–200:8.

[TMFSG07] Thürey N., Müller-Fischer M., Schirm S., Gross M. H.: Real-time breaking waves for shallow water simulations. *15th Pacific Conference on Computer Graphics and Applications (PG'07)* (2007), pp. 39–46.

[Uni19] Unity Technologies: Unity - game engine, 2019.

[WB97] Witkin A., Baraff D.: Physically based modeling: Principles and practice. In *ACM SIGGRAPH 1997 Courses* (New York, NY, USA, 1997), ACM Press.

[WP12] Weissmann S., Pinkall U.: Underwater rigid body dynamics. *ACM Transactions on Graphics 31*, 4 (July 2012), 104:1–104:7.

[WW16] Wang L., Whiting E.: Buoyancy optimization for computational fabrication. *Computer Graphics Forum 35*, 2 (2016), 49–58.

[YHK07] Yuksel C., House D. H., Keyser J.: Wave particles. *ACM Transactions on Graphics 26*, 3 (July 2007). 99–es.

[YL18] Yaz I. O., Loriot S.: Triangulated surface mesh segmentation. In *CGAL User and Reference Manual*, 4.13 ed. CGAL Editorial Board, 2018.

[YSZH13] Yin X., Shen X., Zhang F., Huang G.: Particle-based simulation of fluid-solid coupling. In *AsiaSim 2013*. G. Tan, G. K. Yeo, S. J. Turner and Y. M. Teo (Eds.). Springer-Verlag, Berlin–Heidelberg, 2013, pp. 373–378.

[Yuk17] Yuksel C.: Mesh color textures. In *Proceedings of High Performance Graphics* (New York, NY, USA, 2017), ACM Press, pp. 17:1–17:11.

[ZB17] Zarifi O., Batty C.: A positive-definite cut-cell method for strong two-way coupling between fluids and deformable bodies. In *SCA '17: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA), ACM Press, pp. 1–11.