Approximate and exact buoyancy calculation for real-time floating simulation of meshes

Gábor Fábián[1]

[1] ELTE Eötvös Loránd University, Department of Numerical Analysis, Budapest, Hungary

Abstract

In this paper, we present methods for simulating floatation of bodies represented by triangular meshes. The primary challenge in creating such a simulation is determining the buoyant force and its reference point. We propose 5 algorithms, 3 approximations and 2 exact methods, that enable the real-time calculation of buoyant forces. Each algorithm is based on rigorous physical and mathematical principles, performing calculations directly on the triangular mesh rather than its approximation. Finally, we test the accuracy and efficiency of these algorithms through simple examples.

CCS Concepts

• ***Mathematics of computing*** → *Integral calculus;* • ***Computing methodologies*** → *Physical simulation; Mesh models;*

1. Introduction

The calculation of buoyant force is a fundamental technique essential for the dynamic simulation of objects floating or submerged in fluids. The interaction between rigid bodies and fluids is called the rigid-body-fluid coupling problem, an extensive review of this topic can be found in [BPD20]. Rigid-body-fluid coupling is an exceptionally challenging problem, involving two main aspects: determining the motion of a body submerged in a fluid and, conversely, evaluating the effect of the moving body on the fluid. In this brief study, we focus exclusively on the first problem. Existing solutions often operate on grids [BR13] [Cla13] or directly on the mesh representing the object [KKKT06] [Roy23]. These methods address more complex problems than ours, but they are computationally intensive. In contrast, we found many implementations that could be computed in real-time on a CPU but relied on highly inaccurate approximations. In this paper, we present simple approximation and exact methods that enables the real-time computation of buoyant forces acting on complex meshes, even without utilizing GPU.

2. Physics of floating motion

Consider a solid 3D body (partially) immersed in a fluid volume. We will assume our body to be rigid and have a uniform density. We also assume that the fluid is incompressible, its surface is at rest and sufficiently large.To further simplify the problem we will ignore all the drag forces both in air and in the fluid. In this case, two forces determine the motion of the object: the gravity force and the buoyant force, as shown in Figure 1. The gravity force can be calculated as

$$\mathbf{F}_G = -mg\mathbf{j},$$

where m is the mass of the object, g is the gravitational constant, and $\mathbf{j} = (0,1,0)$ is the upward pointing unit vector. According to Archimedes' principle, the buoyant force calculated as

$$\mathbf{F}_B = \rho g V_B \mathbf{j},$$

where ρ is the density of the fluid, and V_B is the volume of the submerged part of the object, which equals the volume of the displaced fluid.

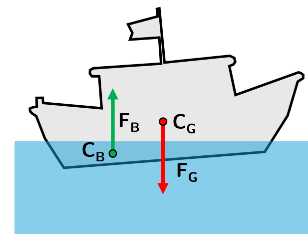

Figure 1: *Floating motion. The gravity force points downwards from the center of gravity, which is the centroid of the whole body. The buoyant force points upwards from the center of buoyancy, which is the centroid of the submerged part of the body.*

It is crucial to note that the reference points of these forces differ. While the gravitational force acts at the object's center of mass $\mathbf{C}_G$, the buoyant force acts at the center of mass of the submerged part, denoted as $\mathbf{C}_B$. The opposing directions of these forces, combined with their different reference points, result in a net force and torque that determine the object's motion.

In a dynamic simulation environment, the object's mass and cen-

© 2025 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.
This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

DOI: 10.2312/egs.20251031

delivered by EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

ter of mass are usually known in advance. These should be specified or precalculated since they remain constant during motion. In contrast, V_B and $\mathbf{C}_B$ vary at every moment. For simulating the object's motion, it is essential to understand how to calculate V_B and $\mathbf{C}_B$ for a given rigid transformation.

3. Mesh volume and centroid calculation

Let us consider the set $\Omega \subset \mathbb{R}^3$, which defines the points of a rigid body in space. The volume and centroid of the body are defined as volume integrals. For instance, the volume of Ω and the x-coordinate of the centroid are

$$\mathrm{vol}(\Omega) = \iiint_\Omega 1 \, dV, \quad \frac{1}{\mathrm{vol}(\Omega)} \iiint_\Omega x \, dV,$$

respectively. According to Gauss's divergence theorem (under appropriate conditions), these volume integrals can be replaced by surface integrals computed over the boundary [MT12], e.g.

$$\mathrm{vol}(\Omega) = \frac{1}{3} \oiint_{\partial\Omega} \mathbf{id} \cdot \, d\mathbf{S}.$$

If the body's surface is represented by a triangular mesh, the integrals can be calculated triangle by triangle. Thus, both the volume and the centroid can be determined by weighted summation of the signed volume elements V_i and centroid elements $\mathbf{C}_i$ for each triangle Δ_i:

$$V = \sum_i V_i(\mathbf{o}), \quad \mathbf{C} = \frac{1}{V} \sum_i V_i(\mathbf{o}) \mathbf{C}_i(\mathbf{o}),$$

The volume and centroid elements can be calculated by formulas

$$V_i(\mathbf{o}) := \frac{1}{6}(\mathbf{a}_i - \mathbf{o}) \cdot \Big((\mathbf{b}_i - \mathbf{o}) \times (\mathbf{c}_i - \mathbf{o}) \Big), \ \mathbf{C}_i(\mathbf{o}) = \frac{\mathbf{a}_i + \mathbf{b}_i + \mathbf{c}_i + \mathbf{o}}{4},$$

as we can read in e.g. [Cat06]. Here, $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i$ denote the three vertices of triangle Δ_i in order. We assumed the triangles have consistent orientation and the outward-facing normal is given by $(\mathbf{b}_i - \mathbf{a}_i) \times (\mathbf{c}_i - \mathbf{a}_i)$. Note that the V_i volume elements are the signed volumes of the tetrahedron defined by $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i$, and an arbitrary chosen reference point $\mathbf{o}$. Similarly, the $\mathbf{C}_i$ centroid element corresponds to the center of mass of the same tetrahedron.

4. Physics simulation

Assume that the current position and orientation of the moving body can be described by the $\mathbf{T}$ rigid transformation. Consider an arbitrary $\mathbf{x}$ vertex of the mesh at rest. For each time step in the simulation there exists an $\mathbf{R}$ rotation and a $\mathbf{t}$ translation, which describe the current location of the $\mathbf{x}$ vertex: $\mathbf{x}' = \mathbf{T}(\mathbf{x}) = \mathbf{R}\mathbf{x} + \mathbf{t}$. At every moment of the simulation, we will execute the following steps.

1. Update the current rigid transformation, $\mathbf{T}$.
2. Classify triangles of the transformed body (non-submerged, partially submerged, totally submerged).
3. Compute the contribution of each triangle to the buoyant force.
4. Pass the force and the torque to the physics simulator.

Our algorithms were implemented using the Unity real-time development platform. Unity itself includes a physics engine, so we relied on it for the rigid body dynamics simulation. Specifically, we utilized the built-in `Rigidbody` component to assign mass to the object. Additionally, in our simulation, this component calculates the effects of gravity and drag forces. Step 1, determining the current rigid transformation, can be achieved using Unity's built-in `Transform.TransformPoint` function. Step 2 can then be carried out straightforwardly. Step 4 can be executed simply using Unity's built-in `Rigidbody.AddForce` and `Rigidbody.AddTorque` functions. The majority of the dynamic simulation is thus handled by Unity, while our custom-developed component modifies the motion of the objects by adding forces amd torques.

The most critical task to address is Step 3: computing the buoyant force. In the following, we present 5 distinct algorithms aiming this problem. The first provides a fast but rough approximation, the second offers a more precise approximation, and the third computes the exact values, the last two offers further development opportunities. The difference among these algorithms lies in which triangles' contributions are considered during their calculations, as illustrated in Figure 2.

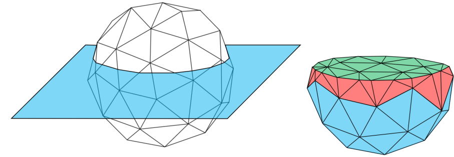

Figure 2: *The fast algorithms only account for totally submerged triangles (blue). The improved algorithm also includes the contribution of the triangles on the fluid surface that enclose the submerged region from above (green). Exact algorithms incorporate, in addition to the above, the contribution of partially submerged triangles (red).*

4.1. Fast approximate algorithm

Classifying triangles can be performed very quickly, even for highly complex meshes; however, calculating the signed volume and centroid elements is computationally expensive. Assume that the set S contains the indices of triangles totally submerged below the fluid surface. Initially, we employed a fast, approximate approach. At the start of the simulation, we precomputed V_i and $\mathbf{C}_i$ for all triangles in the untransformed mesh. At any given simulation step, after classifying the triangles, we calculated the submerged volume and centroid based on the totally submerged triangles. Since the centroid is determined in the object's original local coordinate system, it must be transformed using the object's current transformation to yield the correct result in world space.

4.2. Improved approximate algorithm

Fast algorithm can lead to inaccurate results. In such cases, the contribution of omitted triangles becomes significant. It is important to note that this omission does not only concern partially submerged triangles. The formulas for volume and centroid are based on integrals over *closed* surfaces, meaning that the contributions of polygons formed at the intersection of the fluid surface (plane) and

© 2025 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.

the mesh must also be included in our calculations. A simple approach would involve computing this intersection, triangulating the resulting polygons (with potential holes), and then incorporating the contribution of these surface fragments. However, this procedure would be computationally expensive [Roy23]. The key idea to overcome this problem is as follows. Recall that in the calculations for both volume and centroid elements, the choice of the origin $\mathbf{o}$ is arbitrary. If, at each moment, we adjust the origin so that it lies on the fluid surface, then the volume contribution of all surface elements on the plane becomes zero because the four vertices of any tetrahedra are coplanar. Therefore, let $\mathbf{o}$ be the projection of the object's current centroid onto the $y = 0$ plane, and calculate the volume and centroid elements relative to this reference point.

It is worth mentioning that, unlike the previous algorithm, our calculations in this approach are performed using the vertices of the mesh in the current world coordinate system. As a result, there is no need to transform the centroid after its computation. However, in the case of the current algorithm, pre-computation is not an option, as the reference point $\mathbf{o}$ changes continuously for a moving object. In exchange, this method provides a much more accurate approximation compared to the fast algorithm.

4.3. Exact algorithm

The exact algorithm is a slight refinement of the improved algorithm, where the contributions of triangles intersecting the fluid surface are also taken into account. Excluding edge cases (when

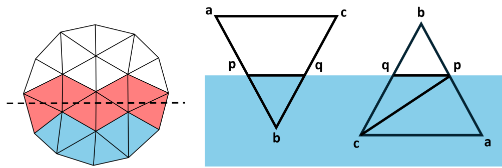

Figure 3: *Non-submerged (white), totally submerged (blue), and partially submerged (red) triangles. Right: two primary cases for partially submerged triangles: if only vertex* **b** *is below the fluid surface, the contribution of the* **pbq** *triangle is considered; if* **c** *and* **a** *are below the surface, the contributions of the* **pqc** *and* **apc** *triangles are also included.*

one or more vertices of the triangle lie exactly on the $y = 0$ plane), two scenarios are possible: the partially submerged triangle either has one vertex or two vertices below the fluid surface. In the first case, we consider the contribution of the subtriangle below the fluid surface. In the second case, the quadrilateral formed below the fluid surface is split into two triangles, and their contributions are added to the existing ones, see Figure 3.

4.4. Surface-based algorithms

In volume-based methods, we applied the divergence theorem to convert volume integrals into surface integrals. However, in this case, integration must be performed over a closed surface, requiring consideration of the contributions from surface elements *inside* the object. These contributions can be eliminated by appropriately choosing the reference point. However, if the water surface is not a perfect plane, this approach is no longer applicable. Furthermore, additional challenges arise, as the surface patch that closes the object on the water surface side is not uniquely defined.

An alternative interpretation of Archimedes' principle, discussed in [Lim11], provides a different perspective. In this formulation, the buoyant force is directly defined as a surface integral, meaning integration is performed over a non-closed surface. The buoyant force is obtained by integrating the pressure over the surface S that is in contact with the water:

$$\mathbf{F}_B = -\iint_\Sigma p \, \mathrm{d}\mathbf{S} = \iint_\Sigma \rho g y \, \mathrm{d}\mathbf{S}$$

Following the previous approach, the calculations can be performed triangle by triangle, summing the contributions of forces and torques for each triangle. After straightforward computation, we get the following formula for force elements: $\mathbf{F}_i = \rho g (\mathbf{C}_i)_y \mathbf{A}_i$, where $(\mathbf{C}_i)_y$ denotes the y-coordinate of the centroid of the triangle, $\mathbf{A}_i$ is the oriented surface element. We developed two algorithms based on this method: the first considers only the contributions of fully submerged triangles (fast surface), while the second also accounts for the contributions of partially submerged triangles (exact surface).

5. Accuracy tests

The exact algorithm considers the contribution of every partially or totally submerged triangle (including those lying on the fluid surface), providing accurate results apart from numerical errors. We validated this accuracy using two simple examples and also assessed the performance of the other two algorithms in terms of precision. For the tests, we examined the submersion of a unit sphere and a unit cube. The centroids of the objects were conceptually placed at point $(0, h, 0)$, and we determined how the magnitude of the buoyant force varied as a function of h. For simplicity, we assumed that $\rho g = 1$. Basic geometric considerations led to straight-

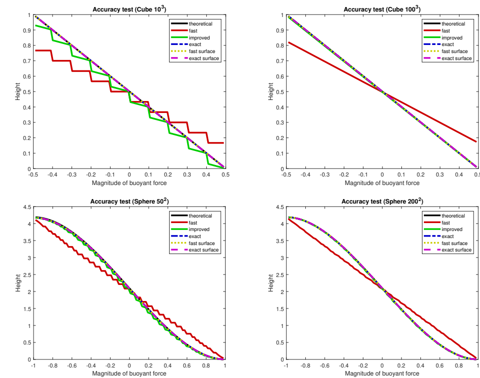

Figure 4: *Accuracy tests for our algorithms in case of submerging a unit cube and a unit sphere.*

forward conclusions. In case of the cube for $h \in (-1/2, 1/2)$ we get

$$F_B^{\text{cube}}(h) = \int_{-1/2}^{-h} 1 \, \mathrm{d}y = -h + \frac{1}{2}.$$

© 2025 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.

	Torusknot	Bunny	Armadillo
Fast	0.09	0.23	0.98
Improved	0.18	0.69	2.96
Exact	0.26	0.75	3.05
Fast Surface	0.22	0.76	3.45
Exact Surface	0.33	0.90	3.94

Table 1: *Performance tests: total time cost for buoyancy calculation for different models (in milliseconds). The values presented in the table are derived from the average of 100 measurements.*

For the sphere, the task was only slightly more complex, requiring the integration of polynomial expressions. In this case for $h \in (-1,1)$ we have

$$F_B^{\text{sphere}}(h) = \int_{-1}^{-h} (1-y^2)\pi \, dy = \frac{\pi}{3}(h-1)^2(h+2).$$

During our tests, we used cube meshes of two different levels of detail. The 10^3 cube contained 1,200 triangles, while the 100^2 cube consisted of 120,000 triangles. Additionally, we tested two sphere-approximating meshes: the sphere approximated with 50^2 segments contained 5,000 triangles, whereas the one with 200^2 segments had 80,000 triangles.

The test results confirmed our expectations. The exact algorithms computed the buoyant force in accordance with the theoretical model, apart from round-off errors. The fast surface algorithm also provided a reasonably accurate approximation. Notably, in both figures, increasing the mesh resolution reduces the error in the improved algorithm, whereas the error in the fast algorithm remains unchanged. This can be explained by the fact that the fast algorithm neglects the contribution of triangles intersecting the water surface. Without an appropriately chosen reference point, this contribution is nonzero, leading to persistent errors regardless of the mesh resolution.

6. Performance tests

We conducted performance tests using several models, three of which are presented here. The torusknot model contains 1760 triangles, the Stanford Bunny model has 5,002, while the Armadillo model has 30,000 triangles. During the measurements, we recorded the response times of the 5 algorithms designed to compute buoyant forces. At the start of the test, the object was almost entirely above the fluid surface, and by the end, it was nearly totally submerged. After performing 100 measurements, we calculated the average cost of computing the buoyant force and torque. The measurements were carried out on a desktop computer with the following configuration: Intel Core i5-9400, 8GB RAM, Intel UHD Graphics 630, running Windows 10 Pro. The results are shown in Table 1.

Even for the detailed Armadillo model, the worst-case measured response time was approximately 10 ms. This enables a simulation speed of 100 FPS on the test computer, ensuring real-time performance for complex models.

7. Conclusion

In this short paper, we presented 5 efficient and easy-to-implement algorithms for determining buoyant force. Our methods are based on mathematical and physical principles. The algorithms have progressively higher computational demands but offer increasing accuracy, with two algorithms providing an exact solution apart from floating point errors. From previous methods [BPD20] deserves special attention. The authors addressed a more complex problem than ours; however, their buoyant force computation is comparable to ours. To achieve this, they approximate the submerged volume, similar to our volume-based approximation methods. A similar measurement was conducted to validate accuracy, confirming that their method only approximates the theoretical curve.

Finally, let us outline the limitations of our algorithm. Throughout the calculations, we assumed that the object is homogeneous and the water surface is a perfect plane. Simulating the motion of inhomogeneous objects would likely require volume-based computations. If the water surface is not planar, the surface-based approach presented here avoids the need to account for contributions from surface patch located inside the object. We aim to generalize our algorithms in this latter direction.

The Unity component implementing the demonstrated algorithms can be found in the [Fab25] GitHub repository.

8. Acknowledgement

Supported by the EKÖP-24 University Excellence Scholarship Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

References

[BPD20] Bajo J. M., Patow G., Delrieux C. A.: Realistic buoyancy model for real-time applications. In *Computer Graphics Forum* (2020), vol. 39, Wiley Online Library, pp. 217–231. 1, 4

[BR13] Bogner S., Rüde U.: Simulation of floating bodies with the lattice boltzmann method. *Computers & Mathematics with Applications 65*, 6 (2013), 901–913. Mesoscopic Methods in Engineering and Science. URL: https://www.sciencedirect.com/science/article/pii/S0898122112006451, doi:https://doi.org/10.1016/j.camwa.2012.09.012. 1

[Cat06] Catto E.: *Exact Buoyancy for Polyhedra*. Charles River Media, 3 2006, pp. 175–188. 2

[Cla13] Clausse A.: Real-time physical engine for floating objects with two-way fluid-structure coupling. *World Applied Sciences Journal 22* (01 2013), 1685–1694. 1

[Fab25] Fabian G.: Floatingmesh – fast approximate and exact buoyancy calculation for meshes, 2025. URL: https://github.com/robagnaibaf/FloatingMesh. 4

[KKKT06] Kim J., Kim S., Ko H., Terzopoulos D.: Fast gpu computation of the mass properties of a general shape and its application to buoyancy simulation. *The Visual Computer 22* (01 2006), 856–864. doi:10.1007/s00371-006-0071-x. 1

[Lim11] Lima F.: Using surface integrals for checking archimedes' law of buoyancy. *European Journal of Physics - EUR J PHYS 33* (10 2011). doi:10.1088/0143-0807/33/1/009. 3

[MT12] Marsden J. E., Tromba A.: *Vector Calculus, 6th Edtition*. W. Freeman and Company, New York, 2012. 2

[Roy23] Royer C.: Buoyancy simulation, 2023. URL: https://github.com/corentin-ryr/Boat-Simulation. 1, 3

© 2025 The Authors.