2.5

Exact Buoyancy for Polyhedra

Erin Catto, Crystal Dynamics

erincatto@gphysics.com

Rigid body simulation brings many new capabilities and challenges. For example, imagine an impromptu raft created from the remnants of a collapsed building near a body of water. The player identifies a suitable chunk of wall and expects it to float in a believable manner. For this to happen, the game must simulate buoyancy realistically. It is obvious that increased realism in dynamic simulation naturally leads to increased expectations of emergence. Therefore, buoyancy is an important ingredient in a well-rounded rigid-body simulation system for games that include water in playable levels.

This gem describes an efficient method for computing buoyancy and drag forces on rigid bodies. The algorithm determines an exact buoyancy force for a polyhedron in a water volume. The central equations are in vector form to allow for SIMD optimization.

[Fagerlund] and [Gomez00] have provided similar investigations of real-time buoyancy. Fagerlund uses embedded spheres to approximate the submerged portion of an object. This requires an additional authoring step, and many spheres may be required. Gomez distributes points on the object's surface and attributes a portion of the surface area to each point. He computes vertical columns of displaced water at each surface point. His method also requires an additional authoring step and may require many points to be placed on the surface (e.g., 20 to 30 for a cube).

In contrast, the algorithm presented here requires no additional authoring step. In terms of geometric data, the algorithm only needs the vertices and triangles of the polyhedron; however, it is limited to flat water surfaces.

This algorithm is exact in the hydrostatic sense, because we neglect the inertia of water. This leads to somewhat unrealistic bobbing, but the approach is far simpler than a fully dynamic water simulation.

Buoyancy

Archimedes' principle states that the buoyancy force on a body immersed in water is equal to the weight of the water displaced by the body:

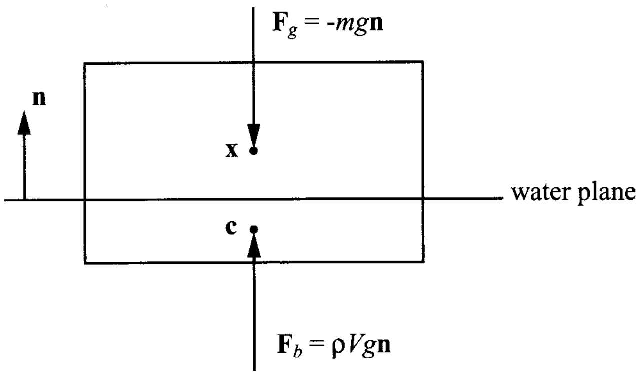
$$\mathbf{F}_b = \rho V g \mathbf{n}, \tag{2.5.1}$$

where ρ is the density of water, V is the volume of the submerged portion of the body, g is the acceleration due to gravity, and $\mathbf{n}$ is the up vector.

As shown in Figure 2.5.1, the buoyancy force counteracts the force due to gravity—for example, the weight of the object:

$$\mathbf{F}_g = -mg\mathbf{n}. \tag{2.5.2}$$

Here, m is the body's mass. The center of mass is $\mathbf{x}$, and the center of buoyancy is $\mathbf{c}$. If the body's average density is larger than the density of water, then the body will sink. On the other hand, if the body's average density is smaller than the density of water, then the body will float.



$\mathbf{F}_g = -mg\mathbf{n}$
$\mathbf{n}$
$\mathbf{x}$
water plane
$\mathbf{c}$
$\mathbf{F}_b = \rho V g\mathbf{n}$

FIGURE 2.5.1 *Buoyancy and gravity forces acting on a body.*

The buoyancy force can also lead to oscillation or bobbing. If a body is dropped into water, its inertia force is added to $\mathbf{F}_g$, and it can displace a volume of water that exceeds the weight of the body. After dipping in too far, the body will accelerate back up, and the process will repeat until the kinetic energy is dissipated.

As shown in Figure 2.5.2, the buoyancy force may produce a torque about the center of mass. This happens because the center of buoyancy is at the center of the displaced volume, which doesn't necessarily coincide with the center of mass. The torque about the center of mass due to buoyancy is:

$$\mathbf{T}_b = \mathbf{r}_b \times \rho V g \mathbf{n}, \tag{2.5.3}$$

where $\times$ is the cross product and $\mathbf{r}_b$ is the radius vector directed from $\mathbf{x}$ to $\mathbf{c}$.

$$\mathbf{r}_b = \mathbf{c} - \mathbf{x} \tag{2.5.4}$$

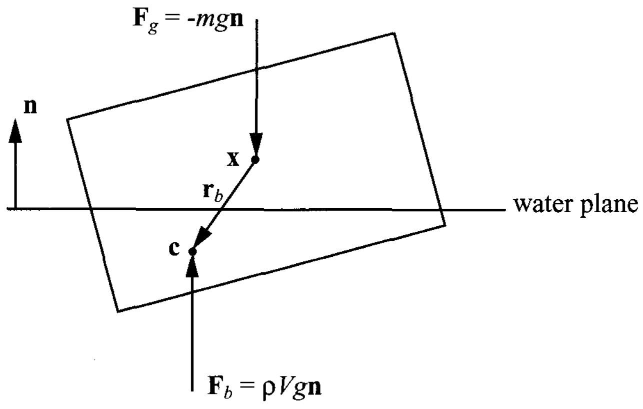

$\mathbf{F}_g = -m g\mathbf{n}$
$\mathbf{n}$
$\mathbf{x}$
$\mathbf{r}_b$
water plane
$\mathbf{c}$
$\mathbf{F}_b = \rho V g\mathbf{n}$

FIGURE 2.5.2 *Buoyancy torque.*

Buoyancy torque is the reason objects have some orientations that are more stable than others. Consider the set of all positions and orientations of the object where the displaced water has a weight equal to the weight of the body. A stable equilibrium configuration is an element of this set where the center of mass is at a (local) minimum height. This is why thin sheets of wood are more stable lying flat on the water than standing up on edge.

Polygon Area

The buoyancy computation for a polyhedron builds from the simpler problem of computing the area of a polygon, which builds on the notion of signed area. Therefore, we will discuss these simpler problems first.

Consider the triangle shown in Figure 2.5.3. The signed area of a triangle has a magnitude equal to the usual triangle area. The ordering of the vertices determines the sign. A Counterclockwise (CCW) order yields a positive sign, while a Clockwise (CW) order yields a negative sign. The edge vectors are defined as $\mathbf{a} = \mathbf{v}_2 - \mathbf{v}_1$ and $\mathbf{b} = \mathbf{v}_3 - \mathbf{v}_1$. Recall that the length of the cross product $\mathbf{a} \times \mathbf{b}$ is the area of the associated parallelogram. The area of the triangle is half the area of the parallelogram. If the plane of the triangle has the unit normal $\mathbf{k}$, then the signed area is:

$$A = \frac{1}{2}\left(\mathbf{a} \times \mathbf{b}\right) \cdot \mathbf{k}. \qquad (2.5.5)$$

Thus, A is positive if the vertices have a CCW order and negative if the vertices have a CW order.

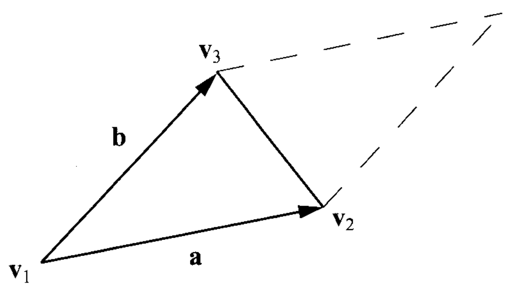


FIGURE 2.5.3 *A triangle.*

According to [O'Rourke98], the area of a polygon is the sum of the signed areas of all the triangles formed from each edge and an arbitrary point **p**. Thus, the area of the quadrilateral in Figure 2.5.4 is the sum:

$$\begin{aligned} A &= A_1 + A_2 + A_3 + A_4 \\ &= A(\mathbf{v}_4, \mathbf{v}_1, \mathbf{p}) + A(\mathbf{v}_3, \mathbf{v}_4, \mathbf{p}) + A(\mathbf{v}_1, \mathbf{v}_2, \mathbf{p}) + A(\mathbf{v}_2, \mathbf{v}_3, \mathbf{p}). \end{aligned} \tag{2.5.6}$$

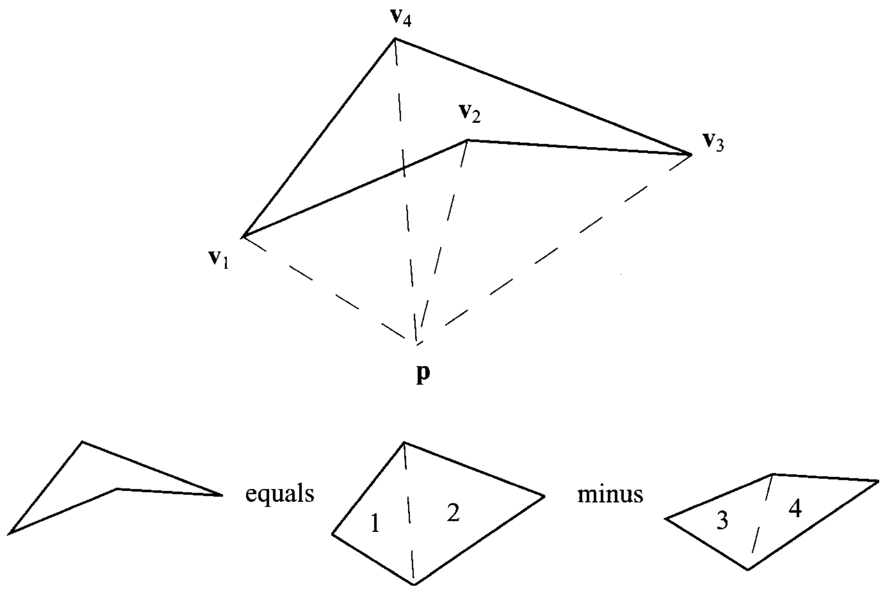


FIGURE 2.5.4 *Area of a polygon as a sum of signed triangle areas. The area of the original polygon is equal to the unsigned areas of triangles 1 and 2, minus the unsigned areas of triangles 3 and 4.*

Note that the first two triangles have positive area (CCW order), while the last two have negative area (CW order), and the overall sum is positive. Also note that each triangle is formed by connecting the CCW ordering of an edge with **p** as the last vertex.

The centroid of a single triangle is simply the average of the vertices, for example:

$$c_1 = \frac{1}{3}\left(v_1 + v_2 + p\right). \tag{2.5.7}$$

The polygon centroid is the area-weighted sum of the component triangles' centroids. Therefore, the centroid of the polygon in Figure 2.5.4 is:

$$c = \frac{1}{A}\left(A_1 c_1 + A_2 c_2 + A_3 c_3 + A_4 c_4\right). \tag{2.5.8}$$

Since we are using signed areas, some of the centroids have a negative weighting.

Polyhedron Volume

Assume for now that the polyhedron is fully submerged. Later, we will deal with the general case of a partially submerged polyhedron. The method for computing a polyhedron's volume is similar to the method for computing a polygon's area. A polygon's area is the sum of signed areas of triangles, while a polyhedron's volume is the sum of the signed volumes of tetrahedron. Each tetrahedron consists of a triangular face of the polyhedron and an arbitrary point **p**. The sign of the volume is positive if **p** is behind the face and negative if **p** is in front of the face, as shown in Figure 2.2.5.

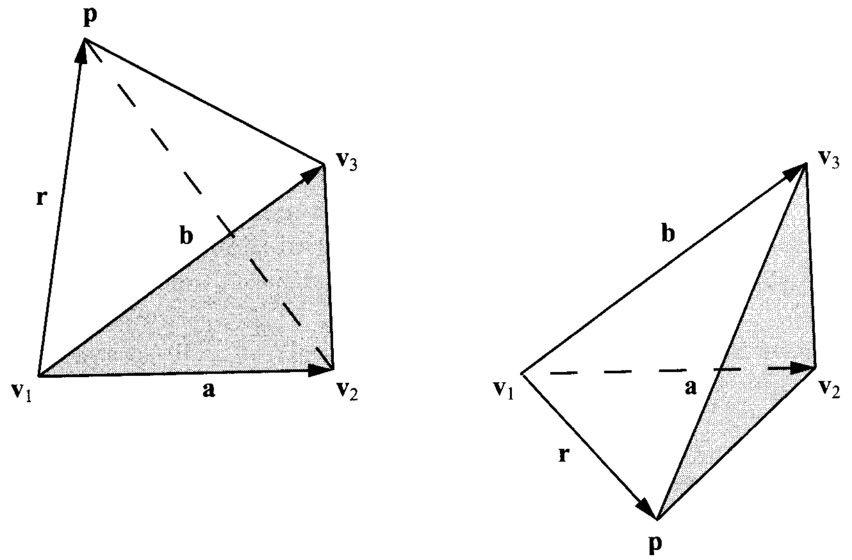


FIGURE 2.5.5 *Two tetrahedra. The tetrahedron on the left has a positive signed volume (**p** in back), while the one on the right has a negative signed volume (**p** in front).*

The formula of [Weisstein] is extended to obtain the signed volume of a tetrahedron:

$$V = \frac{1}{6}(\mathbf{b} \times \mathbf{a}) \cdot \mathbf{r}, \tag{2.5.9}$$

where $\mathbf{a} = \mathbf{v}_2 - \mathbf{v}_1$, $\mathbf{b} = \mathbf{v}_3 - \mathbf{v}_1$, and $\mathbf{r} = \mathbf{p} - \mathbf{v}_1$. Like a triangle, the centroid of a tetrahedron is simply the average of its vertices:

$$\mathbf{c} = \frac{1}{4}(\mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3 + \mathbf{p}). \tag{2.5.10}$$

See [Weisstein] for more details.

Similar to the two-dimensional case, the polyhedron volume is the sum of signed tetrahedron volumes, and the polyhedron centroid is the weighted average of all the tetrahedron centroids.

$$V = \frac{1}{6}\sum_{i=1}^{n}(\mathbf{b}_i \times \mathbf{a}_i) \cdot \mathbf{r}_i \tag{2.5.11}$$

$$\mathbf{c} = \frac{1}{V}\sum_{i=1}^{n} V_i \mathbf{c}_i \tag{2.5.12}$$

Since these formulas are in vector form, they are easy to optimize on SIMD hardware, as shown in Listing 2.5.1.

Listing 2.5.1 Code to Compute the Volume and Centroid of a Tetrahedron

```
float TetrahedronVolume(Vec3& c, Vec3 p, Vec3 v1, Vec3 v2, Vec3 v3)
{
    Vec3 a = v2 - v1;
    Vec3 b = v3 - v1;
    Vec3 r = p - v1;

    float volume = (1.0f/6.0f)*dot(cross(b, a), r);
    c += 0.25f*volume*(v1 + v2 + v3 + p);
    return volume;
}
```

Partial Submersion

Overview

First, consider the two-dimensional case of partial submersion, as shown in Figure 2.5.6. Vertices $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$ are submerged, while vertex $\mathbf{v}_4$ is above the water line. To compute the submerged area, it is necessary to clip the polygon against the water line. Clipping yields the new polygon shown in Figure 2.5.7.

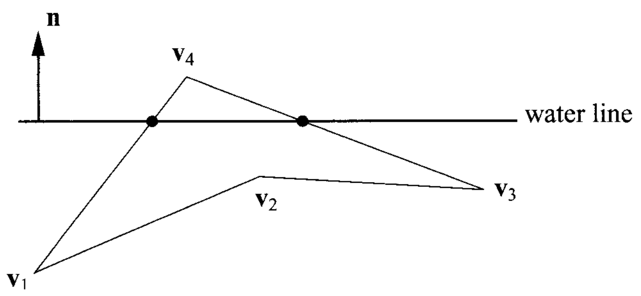


FIGURE 2.5.6 *A partially submerged polygon.*

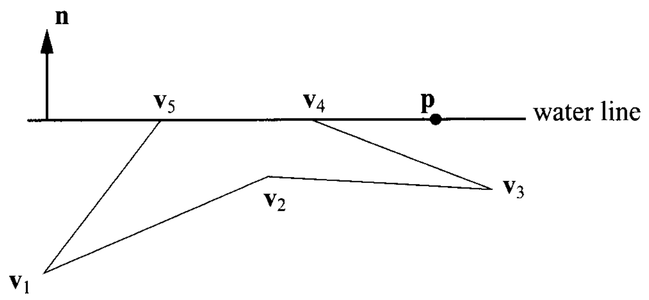


FIGURE 2.5.7 *Clipping the polygon along the water line.*

The area of the polygon in Figure 2.5.7 is the submerged area of the original polygon. To compute the submerged area, choose a point **p** that will form triangles with all the polygon edges. By placing **p** on the water line, the area of triangle ($\mathbf{v}_4$,$\mathbf{v}_5$,$\mathbf{p}$) becomes zero. This eliminates a term from the area sum, making the algorithm more efficient.

Since edges on the water line do not contribute to the submerged area, we can simplify the clipping algorithm. Clip each edge independently. An edge is either above the water line, below the water line, or crosses the water line. Edges above the water line do not contribute to the submerged area. Edges below the water line contribute directly to the submerged area. And finally, edges crossing the water line only contribute the portion that is below the water line.

In the three-dimensional case, clip each triangular face of the polyhedron against the water plane. The clipping process leaves a complex shape on the water plane; it may be several polygons, and the polygons may have holes. By placing **p** on the water line's plane, it is not necessary to consider the faces on the water plane. This greatly simplifies the three-dimensional algorithm.

Clipping

Each triangle belongs to one of three categories:

1. Above the water plane.
2. Below the water plane.
3. Intersecting the water plane.

Category 1 triangles do not contribute to the submerged volume. Category 2 triangles add directly to the volume computation. Category 3 triangles must be clipped against the water plane, resulting in one or two Category 2 triangles.

[Eberly01] presents an algorithm for clipping triangles against a plane. He presents the algorithm in the context of view frustum clipping for graphical rendering, but the algorithm also is well suited for buoyancy calculations. We use a modified version of the algorithm that is optimized for our buoyancy calculation.

Consider a triangle that intersects the water plane. There may be two configurations, as shown in Figure 2.5.8.

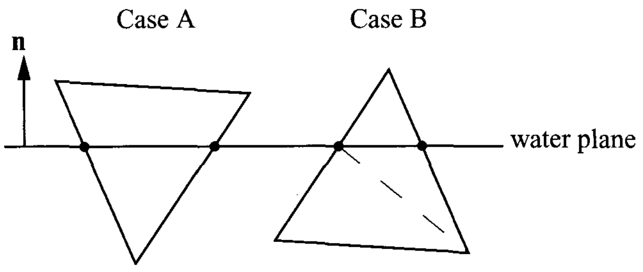


FIGURE 2.5.8 *Clipping configurations Case A and Case B.*

A. One vertex is below the water plane. This results in one Category 2 triangle.
B. Two vertices are below the water plane. This results in a quadrilateral with two Category 2 triangles.

For Case A, identify the two clipping points and process the resulting Category 2 triangle. For Case B, again identify two clipping points and process the resulting quadrilateral as two Category 2 triangles.

A typical graphics pipeline clips triangles against the view frustum and then adds the resulting triangles to the list of rendered triangles. In contrast, the buoyancy algorithm clips the triangles against the water plane and uses the resulting triangles immediately in the volume calculation. The algorithm does not update the list of triangles. This simplifies the code and uses less memory.

At the beginning of the volume calculation, the code computes the depth *d* of each vertex and stores them in an array. The main loop processes each triangle and examines the depths of the triangle vertices. If the code determines that the triangle is Category 2, then the vertices and their depths are passed to the triangle clipper, shown in Listing 2.5.2. The clipper determines the triangle configuration (Case A or B), performs the clipping via linear interpolation, and passes the resulting triangles to the tetrahedron volume function.

Listing 2.5.2 Code for Triangle Clipping

```
float ClipTriangle(Vec3& c, Vec3 p,
    Vec3 v1, Vec3 v2, Vec3 v3,
    float d1, float d2, float d3)
{
    Vec3 vc1 = v1 + (d1/(d1 - d2))*(v2 - v1);
    float volume = 0;

    if (d1 < 0)
    {
        if (d3 < 0)
        {
            Vec3 vc2 = v2 + (d2/(d2 - d3))*(v3 - v2);
            volume += TetrahedronVolume(c, p, vc1, vc2, v1);
            volume += TetrahedronVolume(c, p, vc2, v3, v1);
        }
        else
        {
            Vec3 vc2 = v1 + (d1/(d1 - d3))*(v3 - v1);
            volume += TetrahedronVolume(c, p, vc1, vc2, v1);
        }
    }
    else
    {
        if (d3 < 0)
        {
            Vec3 vc2 = v1 + (d1/(d1 - d3))*(v3 - v1);
            volume += TetrahedronVolume(c, p, vc1, v2, v3);
            volume += TetrahedronVolume(c, p, vc1, v3, vc2);
        }
        else
        {
            Vec3 vc2 = v2 + (d2/(d2 - d3))*(v3 - v2);
            volume += TetrahedronVolume(c, p, vc1, v2, vc2);
        }
    }
    return volume;
}
```

Robustness

Consider the case where a polyhedron is just barely submerged. The triangles produced by clipping may be thin slivers. Due to round-off errors, the sum of the tetrahedron volumes could produce a negative total volume. Also, the centroid may lie outside of the submerged portion of the polyhedron. While these errors may be minor, we consider it good hygiene to avoid spurious results.

The accuracy of the tetrahedron volume formula depends on the quality of the tetrahedrons. High-quality tetrahedrons have balanced interior angles. Low-quality tetrahedrons have interior angles that vary greatly in magnitude.

Consider the two-dimensional case shown in Figure 2.5.9. Notice that the triangle is of low quality, because the interior angles differ greatly. The area formula is:

$$\begin{aligned} A &= \frac{1}{2}(\mathbf{a}\times\mathbf{b})\cdot\mathbf{k} \\ &= \frac{1}{2}\left(a_x b_y - a_y b_x\right) \\ &= \frac{1}{2}\left[\left(L^2 + L\varepsilon\right) - \left(L^2 - L\varepsilon\right)\right] \end{aligned} \tag{2.5.13}$$

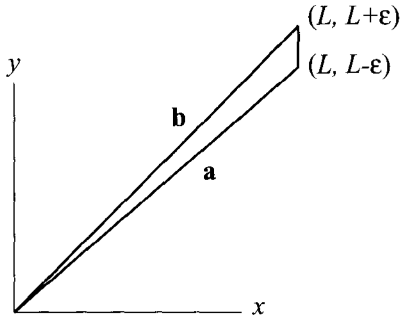


FIGURE 2.5.9 *A slender triangle.*

The exact area is $L\varepsilon$. Now, assume that L is large and ε is small. Then the cross product involves the difference of two large numbers, leading to a relatively small result. This computation is prone to a round-off error, since the L^2 terms dominate.

It is obvious that the tetrahedron quality depends on the placement of **p**. We have already made the decision to place **p** on the water plane, but did not specify where **p** is located on the plane. An improvement in tetrahedron quality is possible by placing **p** directly above the submerged portion of the polyhedron. To achieve this improvement, project a submerged vertex up to the water plane. In our implementation, the projected vertex is chosen arbitrarily from the set of submerged vertices.

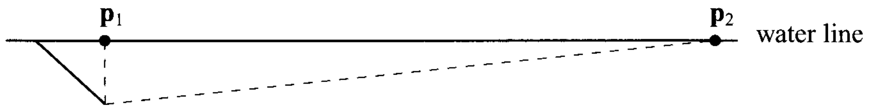


FIGURE 2.5.10 *Example of how the choice of* $\boldsymbol{p}$ *affects triangle quality in two dimensions. The choice* $\boldsymbol{p}_1$ *leads to a high quality triangle because the interior angles are well balanced. The choice* $\boldsymbol{p}_2$ *leads to a lower quality triangle because the interior angles are unbalanced. The choice of* $\boldsymbol{p}$ *in three dimensions has a similar effect on tetrahedron quality.*

Even with an optimal placement of **p**, the tetrahedron volume formula may still be inaccurate. Guards in the code can help to avoid erroneous results. First, abandon the buoyancy calculation if the submerged portion the polyhedron is small. Second, abandon the buoyancy calculation if the total computed volume is negative.

Drag Force

Recall that buoyancy forces coupled with gravity lead to oscillations. In nature, oscillations diminish due to drag forces. Drag forces also allow water currents to move objects.

The drag forces exerted by water on a rigid body are quite complicated. They depend on the surface characteristics and shape of the body. An accurate model of the drag forces requires a full dynamic water simulation. It is questionable whether such accuracy will add significantly to the visual realism, given the computational cost. Therefore, we use an approximate drag-force model that is inexpensive and uses the results of the buoyancy calculation.

Consider the following formula for drag force:

$$\mathbf{F}_d = \beta_l m \frac{V}{V_T}\left(\mathbf{v}_w - \mathbf{v}_c\right) \tag{2.5.14}$$

Here, β_l is a linear drag coefficient in units of one over time, m is the mass of the polyhedron, V is the volume of the submerged portion of the polyhedron, V_T is the total volume of the polyhedron, $\mathbf{v}_w$ is the velocity of the water current, and $\mathbf{v}_c$ is the velocity of the center of buoyancy. For simplicity, $\mathbf{F}_d$ is applied at the center of buoyancy.

The drag force $\mathbf{F}_d$ opposes the motion of the center of buoyancy relative to the water current. Equation 2.5.14 is one of many possible formulas, but it works well in practice.

The drag force $\mathbf{F}_d$ alone is not sufficient to dissipate angular velocity, particularly when the center of buoyancy is directly below the center of mass, and so it is useful to add a drag torque:

$$\mathrm{T}_d = -\beta_a m \frac{V}{V_T} L^2 \boldsymbol{\omega}. \tag{2.5.15}$$

Here, β_a is an angular drag coefficient in units of one over time, and L is the average width of the polyhedron. These parameters ensure that the units of torque are produced.

You can select the drag coefficients β_l and β_a through numerical experiments. First, set the drag coefficients to zero and run a buoyancy simulation with a representative polyhedron, such as a box. The box should oscillate in a stable manner. If not, then reduce the time step in your numerical integrator until the box is stable. Then begin increasing β_l until the bobbing dissipates after a few cycles. Next, drop the box, with an initial angular velocity, about the vertical axis. The box should continue to rotate after the bobbing has ceased. Finally, increase β_a until the rotation dissipates after a few rotations.

Source Code

ON THE CD The source code and a demonstration program are available on the CD-ROM. The code emphasizes clarity over efficiency, so several optimizations are possible. Figure 2.5.11, Color Plate 4, and Color Plate 5 show screenshots from the demo.

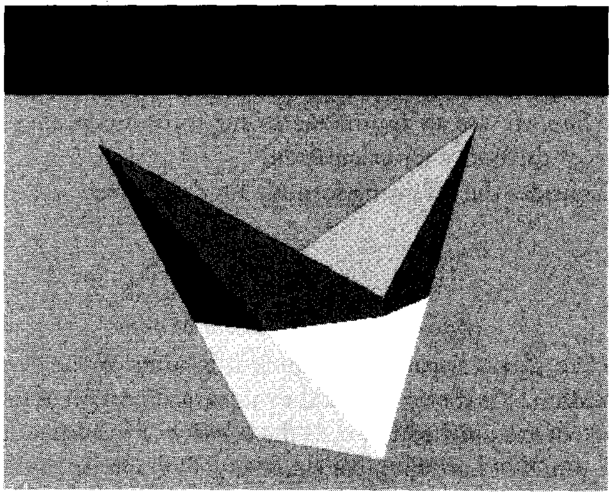


FIGURE 2.5.11 *Buoyancy simulation of a concave polyhedron.*

Conclusion

This article has shown how to compute the exact hydrostatic buoyancy force for a polyhedron submerged in a water volume with a flat surface. Additionally, approximate models of drag force and drag torque were presented to simulate energy dissipation and current coupling. The algorithm is efficient, requires no extra authoring steps, is easy to implement, and integrates well into a physics engine.

Acknowledgment

The author would like to thank the team at Crystal Dynamics for supporting this effort.

References

[Eberly01] Eberly, David H., *3D Game Engine Design.* Morgan Kaufmann, 2001.

[Fagerlund] Fagerlund, Mattias, "Buoyancy Particles or Bobbies." Available online at *http://www.cambrianlabs.com/Mattias/DelphiODE/BuoyancyParticles.asp*.

[Gomez00] Gomez, Miguel, "Interactive Simulation of Water Surfaces." *Game Programming Gems,* Charles River Media, 2000.

[O'Rourke98] O'Rourke, Joseph, *Computational Geometry in C,* 2nd Ed. Cambridge University Press, 1998.

[Weisstein] Weisstein, Eric W., "Tetrahedron." Available online at *http://mathworld.wolfram.com/Tetrahedron.html.*

2.6

Real-Time Particle-Based Fluid Simulation with Rigid Body Interaction

Takashi Amada,
Sony Computer Entertainment, Inc.
taka.am@gmail.com

Realistic, real-time rendering of the motion of fluids is one of the ways to immerse the user into an interactive application, such as a computer game. The interaction of fluids with rigid bodies is important, because in real life, the motion of fluids and rigid bodies is affected by their influences on each other. Fluid simulation based on Computational Fluid Dynamics (CFD) is useful for rendering a visually plausible behavior for the fluid. However, the computational cost of many CFD techniques is often too great for real-time rendering of fluids, which requires fast simulation. Furthermore, many traditional techniques do not enable an easy simulation of fluids interacting with rigid bodies.

This article describes a way to use the smoothed particle hydrodynamics technique to simulate fluids that interact with rigid bodies, and vice versa. We also provide a fast implementation. The proposed method enables real-time simulation of water with rigid body interaction.

Fluid Simulation and Smoothed Particle Hydrodynamics

Basic Approaches to Fluid Simulation

You may have heard of the Naviér-Stokes equation that describes motion for generalized fluid flow. A version of this equation that is valid for incompressible fluids, such as water, is shown in Equation 2.6.1. This partial differential equation describes the conservation of momentum of the incompressible fluid and is equivalent to Newton's second law of motion.