Interacting with Boids in an Incompressible Fluid Environment

Aytek Aman, Ateş Akaydın, Uğur Güdükbay

Department of Computer Engineering, Bilkent University, Ankara, Turkey

aytek.aman@cs.bilkent.edu.tr, akaydin@cs.bilkent.edu.tr, gudukbay@cs.bilkent.edu.tr

***Abstract* - With the increasing power of computing hardware, computer simulations are being realized in real-time. It is now possible to simulate real-life phenomena on standard personal computers. In this study, we present a school of fish and fluid interaction environment where the school of fish in the environment depict flocking behaviors affected by the flow of an incompressible fluid that can be manipulated interactively by a depth sensor controller. The proposed approach can be used in virtual reality applications and video games to provide immersive underwater scenarios for schools of fish. We developed our application on a commercially available game engine that provides real-time experience for the users.**

1. INTRODUCTION

Crowds are collections of a number of individuals who come together in an unorganized way. With the increasing application of virtual, augmented, and mixed reality technologies in military simulations, safety in architectural design, such as emergency evacuation simulations, entertainment industry, including game development and film production, the simulation of crowd behavior gained significant attention in the research community. The realistic simulation of crowds, including human crowds, schools of fish, flocks of birds, and herds of sheep, is especially important for the entertainment industry, including games and film production. For example, in the popular game Assassin's Creed Unity, the crowd is treated as the main character in the game [1]. The crowds can be modeled and simulated at the micro-scale where the individuals in the crowd are modeled (agent-based) and at the macro-scale where the crowd behavior as a whole (flow-based).

The simulation of animal crowds is important to increase the realism of three-dimensional scenes, including the behavior of schools of fish in underwater simulations, flocks of birds, such as starlings, murmurating while moving from one location to another, and herds of cows moving wildly in the jungle. The realistic simulation of herd behavior is not a trivial task because both psychological and physical factors affect the nature of the herd behavior. One way to simulate the behaviors of the groups of individuals in an organized way is to define some simple behaviors and let the individuals interact with each other with these behaviors. This approach was first proposed by Reynolds [2] and it is called the *Boids Model*.

In this study, we focus on the behavior of a school of fish while fish are interacting with each other and their environment. We incorporate velocity fields into the Boids model to simulate the interaction of boids with the environment. This allows us to experiment with the behavior of a school of fish under the influence of external physical forces like wind and water current. The approach we take allows the real-time simulation of thousands of boids, namely fish. We also provide an interactive method to manipulate the fluid's velocity field using a Microsoft Kinect device [3]. To manipulate the velocity field of the fluid, we use hand gestures that are recognized by the Kinect device. Hand gestures allow the user to change the direction of the school of fish while moving in water.

2. RELATED WORK

Craig W. Reynolds developed a computer model describing the coordinated motion of animals such as flocks of birds, or schools of fish. He named this model the *Boids Model* [2]. Every individual animal in this model is called a boid. In the *Boids Model*, there are three main force components that affect the motion of a boid. These are the *separation*, *alignment*, and *cohesion* forces. Separation describes the force that keeps a boid away from its crowding neighbors so that they do not collide. The alignment force orients the boid towards the average direction of the neighboring boids. Finally, the cohesion force steers the boid towards the average position of the neighboring boids. For each boid in the system, these three driving forces are calculated separately and then a final velocity is calculated. In this way, a completely deterministic yet naturally looking crowd simulation is produced. Reynolds's approach has a number of extensions which incorporate other dynamic effects. For instance, Delgado-Mata et al. [4] study the effects of fear on flock behavior by extending the Reynolds model. Hemerlijk and Hildenbrandt [5] study flocks of birds. They propose a physically accurate model by incorporating the effects of fixed-wing aerodynamics into the basic Reynolds model.

Other than the simulation of herd behaviors, this work also focusses on computational fluid simulation. Almost all of the fluid simulation methods are based on a set of nonlinear differential equations called the Navier-Stokes equations. The analytical solution of this set of equations is available only for several, greatly simplified and constrained (i.e., incompressible fluids) cases. Numerical approximations are more common in the field of Computer Graphics. A number of simplifications are performed on the Navier-Stokes equations to approximate the fluid behavior in real-time. Stam propose a stable animation model for fluid-like objects [6]. Fedkiw et al. propose a method for visual simulation of smoke [7]. Later, Stam extend his previous model to cover real-time applications such as video games [8]. Foster and Fedkiw solve the Navier-Stokes equations efficiently using an adaptation of a semi-Lagrangian method and produce a high-quality implicit surface obtained from the velocity field of the liquid to represent the fluid surface [9]. For a comprehensive survey of common fluid simulation techniques for Computer Graphics, the reader is referred to the book that by Bridson [10].

3. THE PROPOSED APPROACH

Our framework consists of three different components. The first component provides the interface between the user and the system. The second component is a fast and stable fluid solver based on the approach developed by Stam [6]. The third component is the flock simulator and it is responsible for driving the autonomous agents (fish in particular) around. In order to create flocking behavior in a fluid environment, we combine these three components together, as it is seen in Figure 1.

To represent fluid velocity in the environment, we keep a three dimensional grid, where each grid cell has an associated fluid velocity. We call this grid as the cell-wise velocity field and denote it with the function $\vec{v}(\vec{\imath}, t)$. $\vec{\imath}$ is a vector that specifies the grid index in three dimensions and t is the simulation time. For each cell, the associated velocity represents the overall movement of the fluid at the center. The cell-wise velocity field is common to all three components and can be influenced by adding (or injecting) additional velocities. To compute $\vec{v}(\vec{\imath}, t)$, we use a three-dimensional version of the Stam's approach [6]. The extension of the Stam's approach to three dimensions is trivial.

The flow chart given in Figure 1 demonstrates the main dependencies among the three components. For each frame, user input is gathered from the Kinect controller [3], and then appropriate movement information is fed to the fluid simulator. With the user input, fluid simulator updates its state. Then, agents in the simulation environment are simulated with respect to flocking rules and fluid motion. The following sections explain these three components in detail.

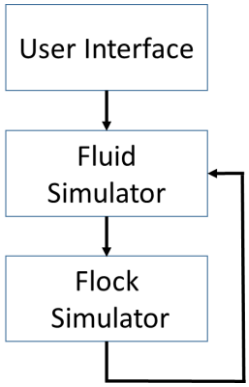


Figure 1. The flow chart demonstrating the system components.

3.1 User Interface

The user interaction in our framework involves the localization of the user's hand position within the simulation environment. For simplicity, we represent the hand as a semi-transparent spherical object centered at the hand's detected position. This object is also called the *Velocity Generator*. We determine the position of the *Velocity Generator* using the functions provided by the Microsoft Kinect API [3]. The *Velocity Generator* tracks its own velocity across successive frames. At each frame, we sample random points inside the boundary of the *Velocity Generator*. Then, we use these points to inject velocity into the cell-wise velocity field. To do so, we determine the grid cells that contain these sampled points. We then add velocities to these corresponding grid cells. After we inject velocities to the fluid, we advance the simulation time by a predetermined time step. Then, the fluid simulator (see Section 3.2) updates the velocity field for the new time step.

3.2 Fluid Simulator

To compute the fluid velocity at an arbitrary position, we keep a continuous velocity field that is separate from the cell-wise velocity field. We denote the continuous velocity field with the vector function $\vec{v}_f(\vec{x}, t)$, where t is the simulation time and $\vec{x}$ is an arbitrary position vector within the field. To compute $\vec{v}_f(\vec{x}, t)$ at some $\vec{x}$, we first determine the cell that contains $\vec{x}$. Then, the index vectors ($\vec{i}$) of the eight local neighbors of this cell are computed separately. For each of the eight neighbors, we retrieve the velocities from the cell-wise velocity field. Finally, we use trilinear interpolation to compute fluid velocity at $\vec{x}$. Having computed the cell-wise velocity field, the control is passed to the Flock Simulator to simulate the flocking behavior.

3.3 Flock Simulator

The flock simulator uses flocking rules proposed by Reynolds [2]. In addition to basic alignment, cohesion and separation forces, we also have boundary forces that make the fish stay in the simulation environment (please note that fish and boid are used interchangeably in the following parts). In order to achieve real-time performance, we keep a secondary grid structure (different from the cell-wise velocity field) to cache the positions of the boids in the simulation space. This grid structure significantly speeds up the neighborhood search calculations. Figure 2 depicts the secondary grid structure that we use for neighborhood search. Additionally, threading is used for the movement of boids. Each boid is assigned to a particular thread for processing.

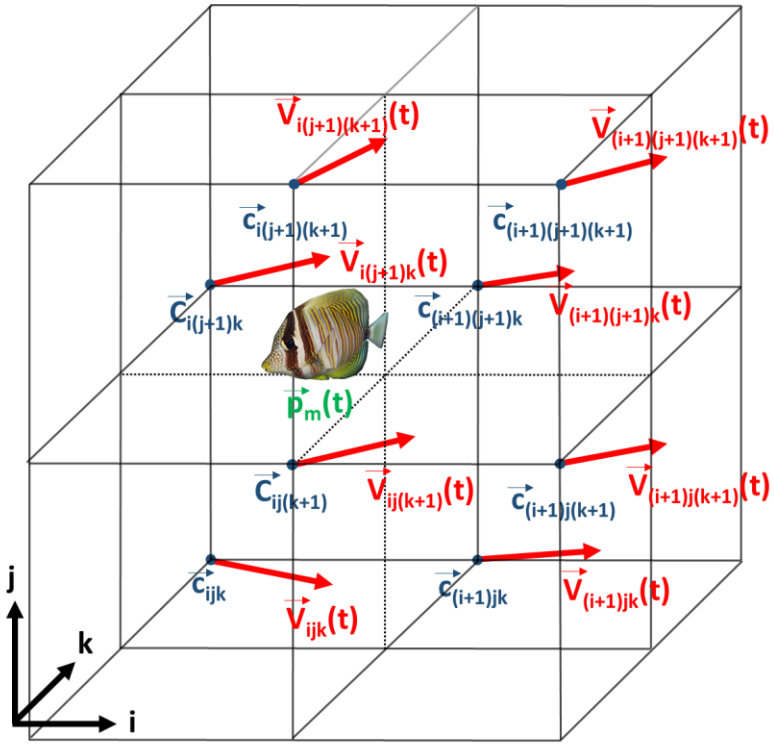


Figure 2. The grid structure used for neighborhood search.

At the beginning, the flock simulator computes the accelerations of the boids with respect to the Reynold's rules. We use a simplified model to compute boid acceleration which is given in Equation 1. For simplicity, we assume that the boids' mass can be omitted. Hence, we drop the mass out of the equation for motion. In Equation 1, the vectors $\vec{f}_a(m)$, $\vec{f}_c(m)$, and $\vec{f}_s(m)$ correspond to the alignment, cohesion and separation forces, respectively. These forces are computed separately for each boid (with index m) using the common formulation proposed by Reynolds [1]. The coefficients ω_a, ω_c, and ω_s control the magnitudes of the three components, respectively. We tune these coefficients empirically to obtain a natural looking simulation.

$$\vec{a}(m,t) = \omega_a \vec{f}_a(m) + \omega_c \vec{f}_c(m) + \omega_s \vec{f}_s(m) + \vec{f}_o(m) \qquad (1)$$

The only additional force we introduce is the obstacle force ($\vec{f}_o(m)$), which keeps the boids away from the aquarium boundaries and the terrain. The obstacle response force is computed using Equation 2.

$$\vec{f}_o(m) = \frac{sgn(\delta_x(m))\vec{e}_x}{|\delta_x(m)| + h} + \frac{sgn(\delta_y(m))\vec{e}_y}{|\delta_y(m)| + h} + \frac{sgn(\delta_z(m))\vec{e}_z}{|\delta_z(m)| + h} \quad (2)$$

In Equation 2, $\vec{e}_x$, $\vec{e}_y$, and $\vec{e}_z$ represent the basis vectors pointing at right (x), up (y) and forward (z) directions, respectively. sgn represents the sign function. A response force is activated only when a boid moves close to any of the six boundaries along the three directions. $\delta_x(m)$, $\delta_y(m)$, and $\delta_z(m)$ are scalars that store the difference between the boid's position and the closest boundary for the respective direction. For any of the six boundaries along the three directions, if the boid's distance to the boundary is greater than a threshold, then the respective direction component is dropped from the equation. Finally, h is the parameter used to set an upper bound for the obstacle avoidance force. The magnitude of the obstacle avoidance force along any direction is never higher than $1/h$.

For simulation environments where we use a terrain at the bottom of the aquarium, we slightly modify the obstacle response force along the up direction. We represent the terrain as a height map. In this case, the upward difference $\delta_y(m)$ represents the difference between the position of the boid and the height map (at the boid's position) along with the up direction.

We then compute an intermediate boid velocity at the current simulation time by summing up the boid's current velocity and acceleration (Equation 3). If the speed of boid exceeds the maximum permissible speed, we clamp the velocity such that its magnitude becomes the maximum speed.

$$\overrightarrow{v'}(m,t) = \vec{v}(m,t) + \vec{a}(m,t) \quad (3)$$

Each boid is affected by the fluid around it. For simplicity, we directly displace the boid with some portion of the fluid velocity vector at its position. The fluid velocity at the boid's position is determined from the continuous velocity field ($\vec{v}_f(\vec{x},t)$). In our test runs, we observed that, just displacing the boids with respect to the fluid velocity around it creates rather synthetic movement. To overcome this problem, we add some other portion of the fluid velocity to the boid itself. Therefore, boid movement under the fluid force can be formulated using Equation 4.

$$\begin{aligned} \vec{p}(m,t+1) &= \vec{p}(m,t) + \overrightarrow{v'}(m,t) + (1-\alpha)\vec{v}_f(\vec{p}(m,t),t) \\ \vec{v}(m,t+1) &= \overrightarrow{v'}(m,t) + \alpha\vec{v}_f(\vec{p}(m,t),t) \end{aligned} \quad (4)$$

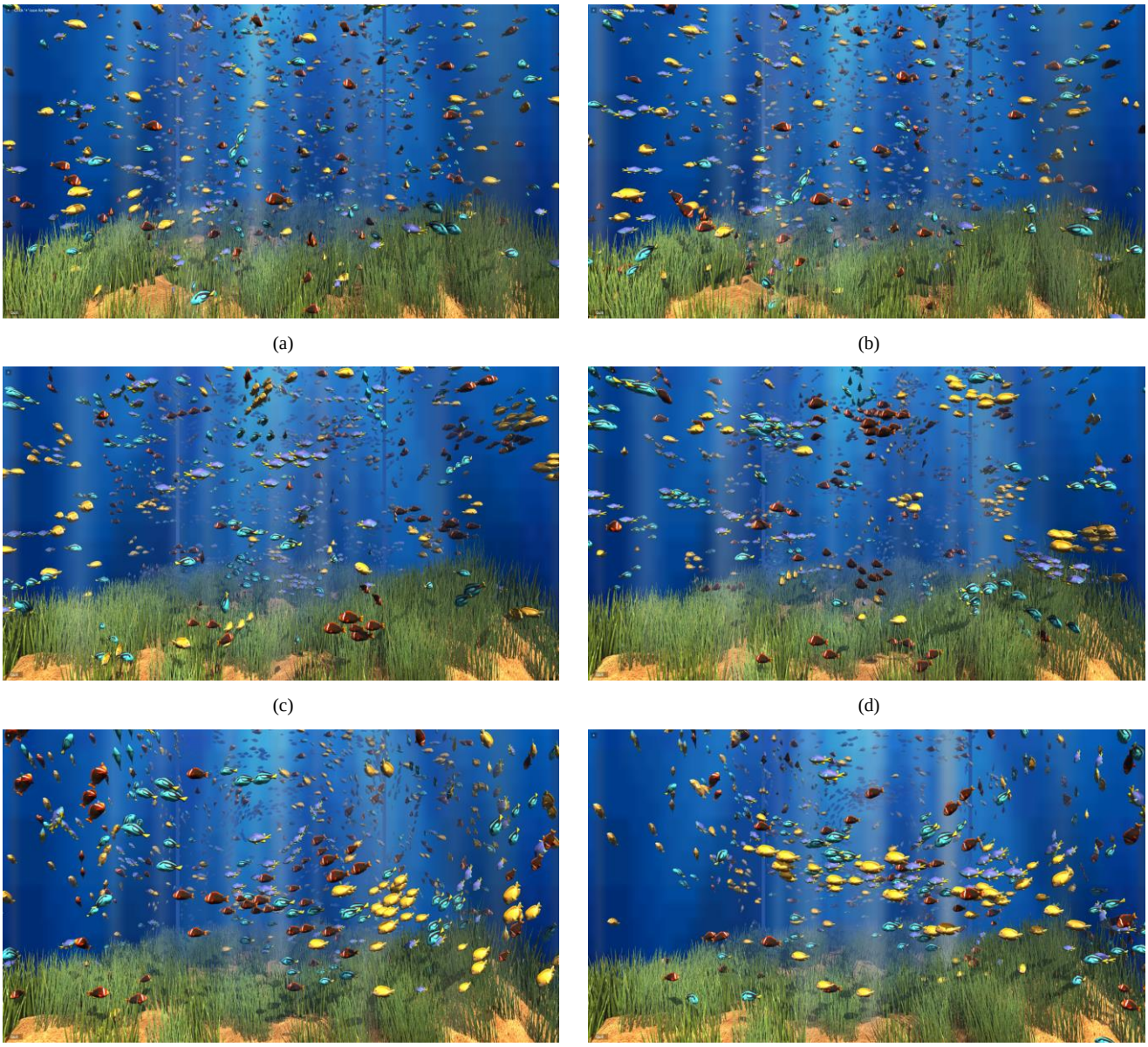
In Equation 4, α is an experimental weight coefficient. Increasing α creates the illusion of stronger fluid behavior where boids are dragged along the velocity field of the fluid. The displacement does not affect the direction of the boids, thus, the dragging effect is more pronounced as it should be. We use spherical linear interpolation [11] to slowly update the directions of the boids in such a way that they are aligned with their velocity vectors.

At any instance, the user can interact with the system by relocating the velocity generator. When such an interrupt occurs, the system schedules a user interface update for the next time step. At the beginning of the next time step, the control is delivered to the user interface component and the new velocities are added to the cell-wise velocity field, as described in Section 3.1.

4. RESULTS

Using the methods described above, we developed a multi-threaded sample application using the Unity 3D Framework [12]. Users can interact with the system via either mouse or the Kinect Controller [3]. Both controllers are used to localize the velocity generator. Several snapshots from the application are presented in Figure 3. Each of the snapshots are taken during run-time and they depict different flocking behaviour, including group formation and vortex movement.

The performance of the system is bound to two parameters. The number of boids in the scene and the number of grid cells of the fluid solver. We tested our application on a workstation with the following hardware configuration: 8 cores, 2.8 GHz, 16 GB of RAM and Quadro K3000 GPU. The graphs given in Figure 4 demonstrate the system performance with respect to the fluid cell resolution and the number of boids in the system.

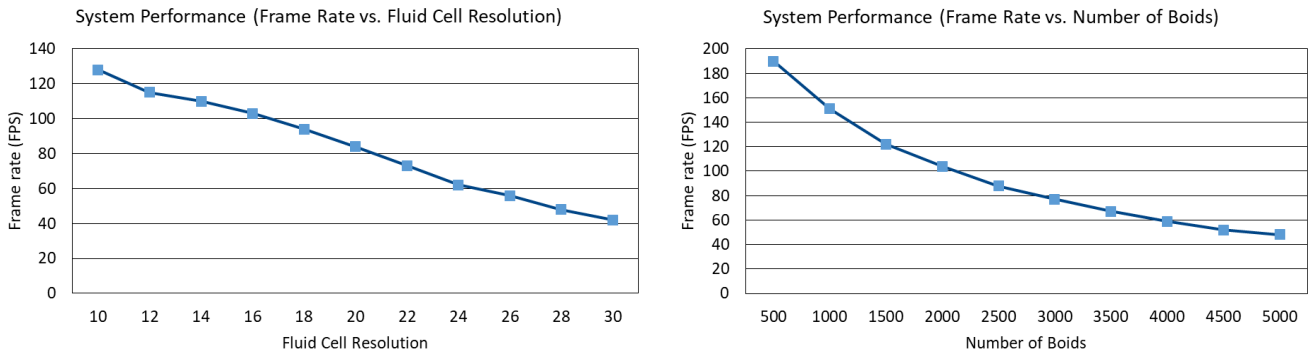


(a) (b)

(c) (d)

(e) (f)

Figure 3. Snapshots taken during run-time from the application. (a) and (b): The first two still frames show the school of fish where the groups are not formed yet and the flocking behaviour is hardly visible. This correspond to low cohesion (ω_c) and alignment (ω_a) and high separation (ω_s) coefficients. As the simulation proceeds, groups are formed and fish start to demonstrate flocking behaviour, which corresponds to high cohesion (ω_c) and alignment (ω_a) and low separation (ω_s) coefficients. The alignment and cohesion forces between fish in different subgroups (shown in different colors) are not taken into account. In this way, separation behavior will be dominant between fish in different subgroups and fish in the same subgroups form their own schools. (e) and (f): still frames demonstrating the influence of fluid motion on the flock. The user created a vortex by doing circular hand motion around the center of the aquarium. The vortex motion also directs the boids accordingly.



(a) (b)

Figure 4. The computational performance of the application in terms of frame rates (frames per second-FPS) with respect to different parameters. (a) The frame rates with respect to the resolution of the fluid velocity field. (b) The frame rates with respect to the number of boids in the system.

5. CONCLUSION

In this study, we demonstrate a framework where boids, specifically fish, and the fluid interact with each other. We have primarily focused on visually appealing school of fish simulation. The physical correctness of the simulation was a secondary concern. Our application runs at real-time frame rates for thousands of fish . The approach we have taken is simple to implement and it is suitable to be used in modern video games or other computer graphics applications. The application provides immersive user experience for the simulations of schools of fish in water where fish are interacting with each other and with the environment. We also provide a hand-gesture based interface to interact with the fluid. The user can manipulate the velocity field of the fluid with the help of a depth camera controller. In this way, hand gestures manipulate the school of fish by changing the directions and velocities of fish in water.

ACKNOWLEDGEMENTS

This work is supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under Grant No. 112E110.

REFERENCES

[1] Jamin Warren, “In Assassin's Creed Unity, the crowd is the main character”, *Kill Screen*, Available at https://killscreen.com/articles/assassins-creed-unity-making-crowd-main-character/, Access date: 24 May 2019.

[2] Craig. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *ACM Computer Graphics (*Proceedings of SIGGRAPH’87*)*, vol. 21, no. 4, pp. 25–34, 1987.

[3] Microsoft, Inc., “Xbox 360 + Kinect, XBOX,” http://www.xbox.com/en-US/kinect, 2014.

[4] Carlos Delgado-Mata, Jesus Ibanez Martinez, Simon Bee, Rocio Ruiz-Rodarte, and Ruth Aylett, “On the use of virtual animals with artificial fear in virtual environments,” *New Generation Computing*, vol. 25, no. 2, pp. 145–169, 2007.

[5] Charlotte K. Hemelrijk and Hanno Hildenbrandt, “Some causes of the variable shape of flocks of birds,” *PloS One*, vol. 6, no. 8, article no. e22479, 13 pages, 2011.

[6] Jos Stam, “Stable fluids,” in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 121–128.

[7] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen, “Visual simulation of smoke,” in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’01. New York, NY, USA: ACM, 2001, pp. 15–22.

[8] Jos Stam, “Real-time fluid dynamics for games,” in *Proceedings of the Game Developer Conference*, vol. 18, 2003, p. 25.

[9] Nick Foster and Ronald Fedkiw, “Practical animation of liquids,” in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*”, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 22–30.

[10] Robert Bridson, *Fluid Simulation for Computer Graphics*. A. K. Peters Ltd., 2008.

[11] Ken Shoemake, “Animating Rotation with Quaternion Curves,” *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*”, ser. SIGGRAPH '85. New York, NY, USA: ACM, 1985, pp. 245-254.

[12] Unity Technologies, “Unity 3d,” http://unity3d.com, 2014.