Dynamic Simulation of Fluids

Hashem Alayed, Sean Boocock, and Mohannad Aldughaim

***Abstract*—**

Adopting a method for the simulation of fluid volumes outlined in Dynamic Simulation of Splashing Fluids[1], we designed a system to simulate the flow of fluid in a rectangular space of arbitrary size. Additionally, we simulated the interaction between this fluid volume and falling objects. Lastly, we designed a scalable simulation of rain generation and falling with realistic interaction between the fluid volume and individual water droplets. Final results were rendered in Maya and implemented in Python from a C++ prototype we created earlier.

I. Introduction

In this research project, we explored approximate ways in which to model the dynamic flow of fluids and that fluid's interaction with objects in an environment. The paper that provided the main reference for our work was a 1995 article, *Dynamic Simulation of Splashing Fluids.* The authors of this article implemented an algorithm that had previously been presented in the literature before[2] to simulate the flow of fluid in $O(tn^2)$ time with n being the resolution of the model and t the number of time steps involved. While approximate, the model is an accurate replication of the movement of fluids when impacted by objects or when objects are floating on its surface, without any need to explicitly model surface level behavior. The original study compared the computerized model to actual splashing of fluids in video frames as a means of substantiating the accuracy of the model and determining crucial constants, a task we found ourselves tackling as well.

At a basic level, the motivation for our work in this particular area is rooted in curiosity in the simple elegance of physical phenomena. Nature is filled with fascinating movements and chief among them is the movement of water. Such beauty and elegance has served as inspiration for those of us in the realm of computer graphics to try to represent the movements of fluids in an algorithmic form. The means by which this could be achieved seem obvious at first – perhaps a brute force simulation capturing known physical laws of nature – but become problematic when meted with constraints of current technology. The simple appearance of flowing water belies the extremely complex, and computationally taxing, task of animating water movement. Even something as simple as the splashing of water needs be described by animators as series of complicated movements from the moment an object hit the surface of the water, followed by displacement of water, and consequent actions up to the generation of foam and bubbles. In other words, it is beyond the scope of computer artists to model nature's movements that occurs in mere seconds and physically accurate dynamically based simulations tend to be very computationally expensive.

In the *Dynamic Simulation of Splashing Fluids* article that constituted the starting point for our work, the researchers provide detailed descriptions of the means used to replicate the dynamic behavior or movement of fluid. Their dynamic method of simulation permits animation of water movements as fluid is impacted by

objects producing splashes and waves as well as the impact of floating object on the water's surface. In doing so they and we have sacrificed some degree of realism of animation of natural phenomenon, in exchange for an order of magnitude less computational time than competing methods.

Using a model of fluid as a series of interconnected columns, we simulated the flow of water in response to various disturbances as well as its interaction with falling objects and a novel rain simulation. We hope to demonstrate the verisimilitude of this approach to natural phenomenon and potential real time performance.

II. Additional Background

There are many techniques to simulate fluids in the literature dating back more[6] than five[8] decades[9] and having their root in the work of Lewis Richardson[10] and other early 20th century mathematicians. Many of these, and often regarded as having the potential for the most accuracy, involve solutions to the 3D Navier-Stokes equations. In their most general form, the Navier-Stokes equations are "a formidable set of nonlinear partial differential equations whose general properties are still not known".[3] In fact, their full description is the subject of an outstanding Clay Mathematics Institute Millennium prize, one of six remaining problems identified as being of paramount importance in the field of mathematics.[4] For the more restricted case of a 3d, incompressible fluid the equations involves the division of space occupied by the fluid into cell network and the measurement of the fluid behavior as it moves across or through the network. However, the technique is not practical for interactive computer animation because of the relatively steep computational cost, with such cost increasing as the cube of the model's resolution.[1]

Another technique is particle systems involving the use of waterfalls or falling water models. With particle systems, dynamic equations are solved to replicate particles that fall due to gravitational factor. The technique generally does not account for the interaction of water particles. The technique is practically applicable to animation of spray or loosely packed water particles or small volume of water particles. With large volume of particles, such as a large pool of water, the technique is not recommendable since simulation becomes computationally untenable for the number of particles needed to serve as a good model.

Particle systems[5] is a technique for modeling of explicit or clear wave movement on the surface of fluid using functions to separately describe wave forms displacing the representation of the fluid's free surface. However, this limited use of particles systems would be inappropriate if it were possible to achieve its results – accurate depictions of traveling waves – via the same methods use for the main fluid volume. The ideal solution at least for simulation of large volumes of water is a technique that replicates a simplified subset of fluid dynamics. Such a technique would ideally not entail costly computations of the aforementioned methods while still serving as a good approximation and modeling most of a fluid's visually apparent phenomena like travelling waves.

The technique that satisfies these requirements, owing first to Kass and Miller[2] and later to O'Brien and Hodgins[1] permit a broad range of movements that includes the response resulting from impacts and the movements of floating objects. This more granular method of modeling fluid volumes as a series of

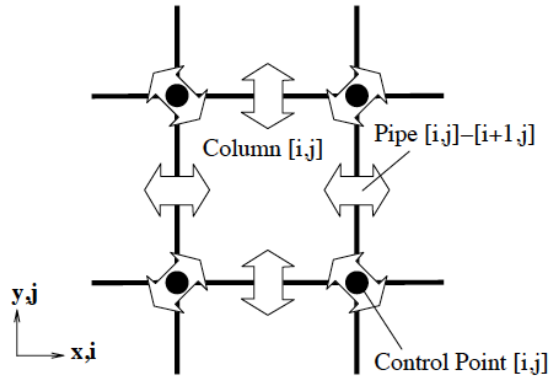
interconnected columns connected by virtual pipes naturally gives rise to phenomena like surface waves without having to explicitly account for the vertical flow of fluids. Additionally as we will show, it is amenable to simulations involving external objects including non-porous spheres and rain.

III. Simulation Model

The simulation of the fluid volume used in our study involves the employment of two-part system namely the main volume and the surface of the fluid. As a whole, the two subsystems serve as the basis of the model.

A. Main Volume model

The main volume model entails formula for dividing the body volume into rectilinear grid of connected columns. In these columns, the characteristics of fluids are considered constant. The movement in these columns goes though series of virtual pipes connecting nearby columns.



In order to simulate the flow of fluid in the fluid volume, one must first calculate the individual flow through each of the virtual pipes connecting a column to its adjacent partners.

$$h_{ij} = \frac{V_{ij}}{d_x d_y} \quad \textbf{(1)}$$

$$P_{ij} = h_{ij}\rho g + p_0 + E_{ij}. \quad \textbf{(2)}$$

$$a_{ij\to kl} = \frac{c(\Delta P_{ij\to kl})}{m} \quad \textbf{(3)}$$

$$a_{ij\to kl} = \frac{\rho g(h_{ij} - h_{kl}) + E_{ij} - E_{kl}}{\rho l}. \quad \textbf{(4)}$$

$$Q^{t+\Delta t}_{ij\to kl} = Q^{t}_{ij\to kl} + \Delta t(ca_{ij\to kl}) \quad \textbf{(5)}$$

The above equations form the basis for calculating those flows. Equation (2) gives the pressure on the (i,j) column in the fluid volume as a function of its height, fluid density, initial pressure and external pressure. How fast the fluid will move between adjacent columns is naturally a function of the difference in total pressure between those columns as well as how big of a connection those columns have (how wide the "virtual" pipe is in this case). Equations (3) and (4) detail the acceleration of fluid along a pipe connecting the (i,j) and (k,l) columns in the fluid volume. The flow for individual pipes is a function of how much (and therefore how fast) fluid can move between columns in a given delta t (5). It is only a short summation more to calculate the change in volume per column subject to conservation constraints (6) – (8).

$$\Delta V_{ij} = \Delta t \sum_{kl\in\eta_{ij}} \left[\frac{Q^{t+\Delta t}_{ij\to kl} + Q^{t}_{ij\to kl}}{2} \right]. \quad \textbf{(6)}$$

$$\forall ij : \forall kl \in \eta_{ij} : Q_{ij\to kl} = -Q_{kl\to ij}. \quad \textbf{(7)}$$

$$V^{t+\Delta t}_{ij} \geq 0 \iff V^{t}_{ij} \geq -\Delta V^{t}_{ij}. \quad \textbf{(8)}$$

As the height of each column is a simple function of volume (1), we get the change in surface behavior essentially for free without the need to explicitly model the upward motion of the fluid within the columns.

B. Surface Model

If the main volume involves columns arranged vertically and pipes arranged horizontally, surface model is that subsystem for permitting extraneous objects to relate with the fluid makeup. The objects hitting or floating on the surface have force applied on the surface model creating external pressure to the main volume subsystem. It is the volume of the columns that have something to do with the vertical positioning of the

$$z_{ij} = \frac{h_{i,j} + h_{i,j+1} + h_{i+1,j} + h_{i+1,j+1}}{4}. \quad \textbf{(9)}$$

surface elements. In the surface model is a rectilinear grid of control points defining a mesh which is mapped onto the columns of the volume model with the control points interpolated between nearby columns, eq. 9 above. The grid points position is found by the taking the average height of the four columns that are around the grid point.

IV. External Objects

In the event of collision of objects with the fluid surface, we simulated the movement of the fluid and the object through computation of forces rising from the collision and their application to the fluid surface and the object. For this, we decided to take an alternative approach than that used by O'Brien and Hodgins in their work modeling the same interaction.

$$\Delta E = 0 \quad PE_{at\ initial\ height} = KE_{at\ surface} \quad \textbf{(10)}$$

We started from first principles[7] and used the conservation of energy to first derive the kinetic energy of the falling object at impact with the surface of the water (10) and then the work-energy equation to derive the average impact force over the distance it would take for the falling object to come to rest in the fluid before rising again.

$$\Delta Energy = Work = F \cdot d_{stopping\ distance} \quad \textbf{(11)}$$

$$\Delta Energy = KE_{at\ surface} - 0 = F \cdot d_{stopping\ distance} \quad \textbf{(12)}$$

$$F_{avg.\ impact} = {KE_{at\ surface}}/{d_{stopping\ distance}} \quad \textbf{(13)}$$

Since the timescales we were using to simulate were on the scale of .01-.05 seconds, we approximated that the impact force would be exerted during one timestep. The stopping distance that appears in (13)-(15) is a measure of how far the object will descend below the surface until the kinetic energy at the surface has dissipated. Since we were trying to simulate water for the most part, we felt that this distance should decrease exponentially as the crossectional area of the falling object increased. Though approximate, this method led to good results without a more elaborate minimization process suggested in the literature.

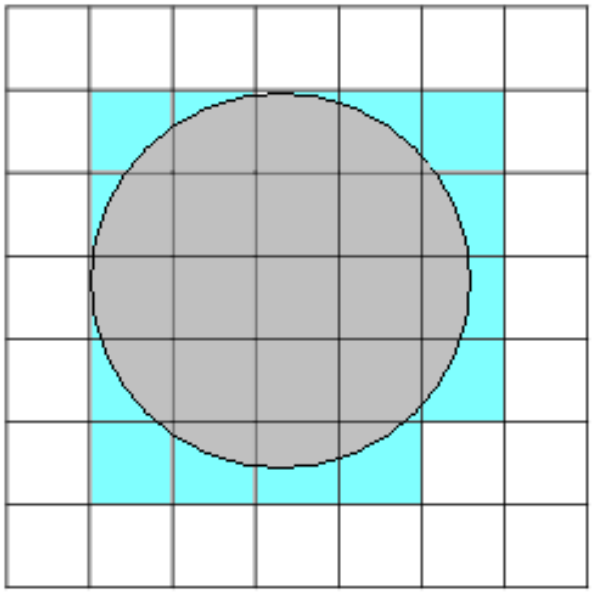


Figure 1: The crossectional area of the falling sphere.

The last component in modeling falling objects interacting with the fluid volume was calculating the pressure on individual columns. Since we had decided to model falling spheres, the calculations for how much pressure would be exerted on each column were more involved. We treated the situation, as shown in figure 1, as if the force from the falling (and resting) object were exerted equally over the whole of its crossectional area. Areas in which the crossectional area only partially overlapped a column received proportionately less pressure as determined by hand calculations of the exact area overlapping each column for a few grid sizes relative to the radius of our falling spheres.

After impact the objects' weights were modeled and continually applied to the fluid volume. The height of the object became a function of the interpolation of the overlapped columns with a factor accounting for how much into the fluid the object would rest given its buoyancy.

V. Rain

In the course of work we latched onto the idea of simulating rain with our fluid volume. Rain seemed like an interesting choice as, given its nature, it would interact with the fluid in a unique way versus other falling objects.

Our final implementation was fairly elegant, allowing us to easily change the number of simultaneously falling droplets, the time at which the rain would start, how fast/long it would ramp up to its peak, and corresponding time steps for its gradual decline. We spawned the rain at randomized locations subject to the constraint that rain would fall in the center of each column (the positioning calculated automatically based on grid size of the fluid volume). During the fall of the droplets under gravity, the vertical axis of the droplets was stretched to simulate the distortion real rain drops suffer as they fall from clouds. This was done using a linear interpolation of scales, updating for every time step between spawning and impact at the surface.

The droplet-surface interaction was handled with a basic collision detection check on every timestep. If a rain drop collided with the surface, the volume of the corresponding column that it hit was adjusted to account for the added volume of the droplet. Although simple in concept, this model produced nice, scalable results for anything between a slight drizzle to a torrential downpour.

VI. Implementation

Initial implementation of the simulation was done in C++. C++ afforded us the ability to test our implementation of the algorithm to a much higher degree than we were able to using Python in our final implementation. It also allowed us lower level control

of data structures such as having Column objects store pointers to adjacent columns that allowed for easier updating of pipe flow.

We found in these early implementation steps that the update step for each column, while at first appearing to need to update each of the 8 virtual pipes connected to it, only needed to calculate flows for half of the pipes. We adopted a "scanline" approach to updating the columns, moving across the grid of columns and exploiting the conservation constraint (7) to only calculate the right, bottom right, bottom and bottom left virtual pipes for every column. This worked even at the first time step as boundary conditions necessitated that the (0,0) column would always have zero flow along the left, top left, top and top right virtual pipes. We made other optimizations in this first step, including updating a columns data structure to reflect values for the next time step as we were calculating values for this time step further along the grid.

Although C++ afforded us an easy way to test our code, and showcased the potential real-time performance of our implementation, we decided to move to Python in order to construct our scene (section VII below) and render our results. The falling object and rain simulation portions of our project were also implemented exclusively in Python. As Python is a higher level language, we had to simplify and change some of our data structures to make the implementation possible. One change for instance was dropping the Column and ColumnGrid data structures we had used in C++ in favor of storing values in multi-dimensional Python lists. Another change was working exclusively in height instead of the volume of columns (though the two are equivalent (1)). The performance of the final implementation suffered from the move to Maya's Python interpreter but the algorithmic performance was still O(tn^2).

VII. THE SCENE

The whole scene was implemented using Maya. The basic idea of the scene was to create a Christmas-like atmosphere, in which we have snow and some Christmas trees around.

The scene consists of three main objects and several additional objects that give the scene a nice look, as you can see in figure 2.

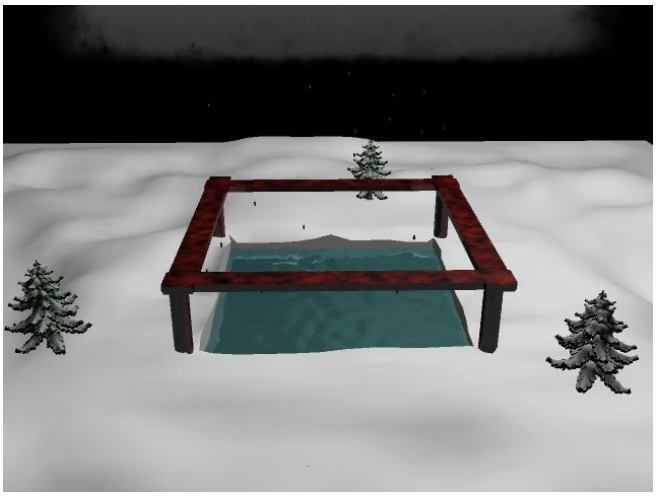


Figure 2: The whole Scene.

The main objects are:

A. The pool

The pool was created using Maya's PolyCube model, which is a polygon that consists of a grid of columns. The texture we used is called a Blinn using the Hypershade tool. We used refraction to make it reflect objects around it. The grid size was 30x30.

B. The Snow

The snow was created as a 50x50 grid of Maya's PolyPlane model. Then the portion of the pool were cut using the Boolean's "Difference" command. The texture used for this one was a Blinn too. The bumping was added using the Sculpt Geometry Tool.

C. The Cloud

The cloud was created using a 3D fluid container with a 70x70 resolution and size slightly larger than the pool. Several attributes were adjusted to make it look more like a real cloud, such as, turning off the density and velocity, color, opacity, texture opacity, and texture amplitude and ratio.

The additional objects to the scene were:

- The trees, which were imported from Turbosquid.com, but textured by us.
- The balls on the trees were created and textured by us.
- The red marble poles also were created and textured by us.
- The light is an ambient light, which is located right above the cloud and pointed into the middle of the pool.
- Rain drops are simple sphere that change shape - in the Python code - with the change if time.
- The falling ball is a simple sphere that has a picture texture, as you can see in figure 3.

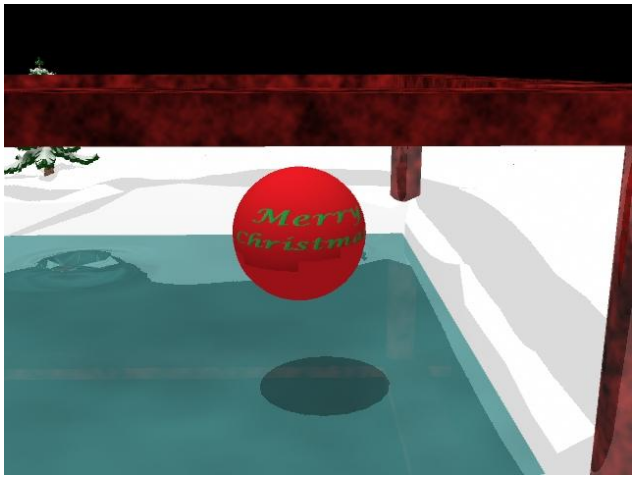


Figure 3: The Falling ball.

Finally, the animation of the scene was made though using the Maya command "SetKeyFrame". We set each keyframe in the Python code for each change of the water and rain drops (or the falling ball).

VIII. CONCLUSION

We believe we demonstrated the utility of these algorithms as good approximations for fluid dynamics in lesser computation time than other approaches. Our implementation of falling objects as well as rain follow in this vein, serving as good approximations of their physical counterparts. While our results were not in all cases wholly faithful to physical phenomena, we think the results demonstrate a strong verisimilitude with the natural phenomena we hoped to capture. Moreover, any error in the simulation model is also justified by the complexities in the movement or behavior of the fluid itself. This is an open problem with much potential for future work at both ends of the spectrum (better lower cost approximations and more complete descriptions of

the physical phenomena regardless of computational cost or mathematical complexity). Future work using this same approach might explore modeling more physical phenomena like foam or modeling the displacement of fluid from sub-surface objects (including an uneven container). Another potential area would be to explore the accuracy of moving to a hexagonal grid of columns each with twelve pipes. Overall we found this project to be a satisfying endeavor that produced aesthetically pleasing, accurate, and sometimes unexpected results.

VIDEOS

In this section, the "youtube" links for the simulation videos are shown along with a simple description.

- Falling ball scene:

http://www.youtube.com/watch?v=kYTUL56_ch8

- Single wave with no background:

http://www.youtube.com/watch?v=XutpF3PoSeA

- Multiple waves with no background:

http://www.youtube.com/watch?v=VWLKt__XKLw

- Full raining scene (without shadows):

http://www.youtube.com/watch?v=pndcBEf_S74

- Full raining scene (with shadows):

http://www.youtube.com/watch?v=OKH4ERTiUfU

- slow motion rain fall/ripples:

http://www.youtube.com/watch?v=_RZq8UX57r0

REFERENCES

[1] J.F. O'Brien and J.K. Hodgins, "Dynamic Simulation of Splashing Fluids," *Proc. Computer Animation 95,* IEEE CS Press, Los Alamitos, Calif., Apr. 1995, pp. 198-205.

[2] Kass, M. ,and Miller, G., "Rapid, Stable Fluid Dynamics for Computer Graphics," Proceedings of SIGGRAPH`90,in *Computer Graphics*,Vol.24, No.4,pp.49-57,1990.

[3] Schetz, Joseph A. and Allen E. Fuhs. *Fundamentals of fluid mechanics*.
New York: John Wiley & Sons, 1999. pp. 66-67

[4] http://www.claymath.org/millennium/Navier-Stokes_Equations/

[5]Reeves, W.T. "Particle System – a Technique for Modeling a Class of Fuzzy Objects." ACM transactions on graphics 2. 2(1983): 91-108

[6] Harlow, F. H. and J. E. Welch. "Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface." The Physics of Fluids 8(1965): 2182–2189

[7] Chabay, Ruth W. and Bruce A. Sherwood. *Matter and Interactions*. Hoboken: John Wiley & Sons, 2011. pp. 227-236

[8] "The Legacy of Group T-3" http://www.lanl.gov/orgs/t/t3/history.shtml

[9] Harlow, F.H. and J. E. Fromm. "Computer Experiments in Fluid Dynamics." Scientific American 212.3(1965): 104

[10] Hunt, J.C.R. "Lewis Fry Richardson and His Contributions to Mathematics, Meteorology, and Models of Conflict." Annual Review of Fluid Mechanics 30(1998): xiii-xxxvi