2018-12-01

Toward Real-Time FLIP Fluid Simulation through Machine Learning Approximations

Javid Kennon Pack
Brigham Young University

Toward Real-Time FLIP Fluid Simulation Through Machine Learning Approximations

Javid Kennon Pack

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Parris K. Egbert, Chair
Seth R. Holladay
Eric G. Mercer

Department of Computer Science

Brigham Young University

ABSTRACT

Toward Real-Time FLIP Fluid Simulation Through
Machine Learning Approximations

Javid Kennon Pack
Department of Computer Science, BYU
Master of Science

Fluids in computer generated imagery can add an impressive amount of realism to a scene, but are particularly time-consuming to simulate. In an attempt to run fluid simulations in real-time, recent efforts have attempted to simulate fluids by using machine learning techniques to approximate the movement of fluids. We explore utilizing machine learning to simulate fluids while also integrating the Fluid-Implicit-Particle (FLIP) simulation method into machine learning fluid simulation approaches.

Keywords: fluid simulation, machine learning, water

ACKNOWLEDGMENTS

I'd like to give thanks to everyone who has guided and supported me through all my years of school. Special thanks to my wife for bringing home the bacon.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Fluids in computer generated imagery can add an impressive amount of realism to a scene. Fluid simulations are used extensively in modern computer animated films but are noticeably lacking in interactive applications such as computer games. The reason for the lack of fluid simulations in interactive applications is due to the prohibitively expensive computations required by the computer to simulate fluids using current algorithms. A recent approach to simulating fluids in the research community is to use machine learning techniques to approximate the movement of fluids. These machine learning techniques can run faster than the normal algorithms, resulting in real-time simulations. Previous applications of machine learning have integrated regression solvers into either a particle-based or a grid-based simulation method, but a hybrid simulation approach, an approach combining both particle-based and grid-based methods, has yet to be explored. In this thesis we explore several as-of-yet unexplored facets of utilizing machine learning approaches in fluid simulations and find several improvements and optimizations.

Specifically we focus on integrating the hybrid Fluid-Implicit-Particle (FLIP) simulation method into a machine learning solver. To our knowledge this has never been done before. By using FLIP in our machine learning approximate fluid solver, we have a larger selection of potential machine learning features to explore.

We also focus on simulating liquids such as water, whereas previous machine learning approaches focused solely on smoke plume simulations. Liquid simulation requires a much

more accurate simulation when compared to gas or smoke. By showing our approach working with fluids, we can prove that our approach can approximate much more complex materials.

The notable contributions of this research can be summarized as follows:

1. The incorporation of the FLIP method in fluid simulation using machine learning.
2. A comparison of FLIP-based machine learning fluid simulation versus other techniques.
3. The exploration of various design options for using machine learning to perform fluid simulation.

1.1 Motivation

Fluid simulations are used extensively in computer generated imagery. They often appear as visual effects in live action films as well as in fully computer animated movies. Fluid simulations allow film makers to animate the natural movements of various materials without relying on practical effects which for any number of reasons might be costly or dangerous. The accurate motion achieved by modern fluid simulations help viewers to successfully suspend their disbelief and enjoy the movie. For example, a fluid simulation simulating water flowing into a sinking submarine can be believable without putting the actors in danger. These simulations, however, are very time consuming to produce. An effects artist has to setup a scene within the visual effects software and then wait for the scene to simulate the desired frames of movement. Depending on the complexity of the scene, the artist would have to wait hours or days to view the results. After viewing the result, several additional passes might need to be done on the simulation, further increasing the total time needed for a single fluid simulation. Any method to speed up fluid simulations would be a boon to these effects artists and the people who employ them. A real-time fluid simulation that wasn't completely accurate would greatly improve productivity even if it was only used for the first few passes for a given fluid simulation effect. This real-time fluid simulation would likely serve as a pre-visualization tool.

Fluid materials such as water, fire, and smoke are also very commonly seen in video games. Video games, however, must compute all the imagery 60 times a second, or "real time." Because of this time constraint, fluid simulations, which are slow, are almost never seen in video games. As a result, attempted player interaction with these materials is noticeably disappointing. Video games employ various visual shortcuts to attempt to give the impression that the material is behaving correctly, but these shortcuts fall short on closer inspection. If fluid simulation could be achieved in real time, these visual effects could greatly improve the sense of immersion felt by the players leading to a more enjoyable experience. A real-time fluid simulation would also open up completely new gameplay experiences that take advantage of the natural motion of fluids. For example, in recent years many video games, notably "Half Life 2," have employed rigid body physics simulations as a major gameplay element and have enjoyed immense success due to the fun and believable motion of the rigid bodies. This effect has been seen already with games employing two dimensional fluid simulations, as seen in the "Where's My Water?" series, but games with three dimensional fluid simulations are still missing.

As a tool for visual effects artists working on films or a way to achieve real-time fluid simulations in video games, a real-time fluid simulator would open up many new possibilities. By saving money for visual effects studios and by opening up new game design ideas for video game developers, a real-time fluid simulator has the potential of significantly improving areas such as film, animation, and games. We have found that machine learning makes this possible.

Chapter 2

Previous Works

To fully understand this body of work, a basic understanding of fluid simulation and machine learning are needed. The first sections of this chapter will review the current state of the art in these areas. Following that discussion, we will explore current approaches that combine these two.

2.1 Fluid Simulation

"Fluid simulation" refers to simulating the movement of materials that behave like fluids. A popular medium for fluid simulations is water, but other materials such as smoke, fire, sand, snow, and liquids of varying viscosity are also simulated through fluid simulations. These materials are often simulated for movies to add impressive visuals. While common in movies, fluid simulations rarely make it to three dimensional video games and other interactive programs because the computation power required is prohibitively expensive. Interacting with fluids in video games in their current form is disappointing because the lack of simulation means a lack of response to the characters or other objects in the game, breaking the suspension of disbelief many video games strive to achieve. Since most video games aim for 60 frames per second (FPS), a fluid simulation would have to match that speed to avoid visual artifacts such as stuttering. This target of 60 FPS is our goal for the term "real-time." Without a real-time solution for simulating fluids, video games will continue to lack the immersive experience that could be possible with better fluid simulation solutions.

2.1.1 Fluid Equations

Fluid simulations are typically based off of the Navier-Stokes equations. This set of equations has been around for over a hundred years, but with the advent of computers and computer graphics, they have relatively recently found widespread use computing fluid simulations. This pair of differential equations is shown below:

$$\nabla \cdot u = 0 \tag{1}$$

$$\frac{\partial u}{\partial t} = -(\nabla \cdot u)u - \frac{1}{\rho}\nabla p + v\nabla^2 u + F \tag{2}$$

Equation 1 ensures that the fluid is incompressible. It states that the divergence ($\nabla\cdot$) of the velocity field (u) is zero, meaning that the amount of fluid entering and leaving a volume must be equal. Equation 2 accounts for viscosity ($v\nabla^2 u$), pressure ($-\frac{1}{\rho}\nabla p$), convection ($-(\nabla \cdot u)u$), and external forces (F) to calculate the derivative of velocity with respect to time ($\frac{\partial u}{\partial t}$).

Because the Navier-Stokes equations are non-linear partial differential equations, various numerical simulation methods are used to discretize them. The main approaches include the Eulerian, Lagrangian, and hybrid methods. Common to all three approaches is that the most time consuming step is calculating a pressure field that satisfies the Navier-Stokes equations. This step involves solving a large, sparse linear system of equations to satisfy the incompressibility constraint. Despite years of improvements to highly specialized software libraries that solve these types of systems of equations, this step is still very time-consuming.

2.1.2 Fluid Simulation Approaches

There are three main approaches to fluid simulation: Lagrangian particles, Eulerian grids, and hybrid approaches. These three approaches differ in how the fluid is represented and the manner in which fluid movement is calculated.

The Lagrangian approach uses particles to represent the fluid. As such, this approach is also known as the particle-based approach. Lagrangian or particle-based methods perform calculations on actual particles that are free to move about as the simulation progresses. The Lagrangian approach was first detailed in [Gingold and Monaghan, 1977]. The Smooth Particle Hydrodynamics (SPH) approach [Lucy, 1977] is a commonly used particle-based method. SPH was originally developed for solving astrophysics problems, but [Monaghan, 1992] and [Monaghan, 1992] showed that SPH was applicable to fluid simulation. Subsequent work sought to solve various performance and accuracy issues. [Müller et al., 2003] extended SPH to model free surfaces and surface tension. [Solenthaler and Pajarola, 2009] presented Predictive-Corrective Incompressible SPH (PCISPH) which improved simulation speed dramatically. [Müller et al., 2007] detailed a new approach called Position Based Dynamics (PBD) which allowed for larger timesteps while maintaining stable simulations. Recently, the Position Based Fluids (PBF) [Macklin and Müller, 2013] method has shown to be a promising new particle-based approach, further improving simulation speed while preserving fluid stability. Particle-based approaches, however, are hindered by the necessity for very large particle counts compared to other approaches.

The second approach to fluid simulation is the Eulerian or grid-based approach. Grid-based methods incorporate a grid cell structure to facilitate the calculations for the Navier-Stokes equations and don't have actual particles that move about. Instead, each cell keeps track of the presence or absence of fluid in the cell. [Foster and Metaxas, 1996] showed the first 3D water simulation using a grid. [Stam, 1999] detailed new advection approaches that led to faster simulations. [Fedkiw et al., 2001] further improved grid approaches by incorporating higher order interpolation as well as a vorticity confinement term. [Foster and Fedkiw, 2001] simulated liquids and devised methods to prevent mass loss by adding marker particles and level sets. Further improvements to the grid-based approach include fluid-solid interaction [Génevaux et al., 2003], simulating fluids of variable viscosity [Carlson et al., 2002], and better level set calculation approaches [Enright et al., 2002]. Further improvements in

simulation quality have come about by more accurate advection approaches such as those in [Kim et al., 2005] and [Selle et al., 2008]. Grid-based approaches struggle to achieve individual fluid motion such as splashes and droplets. The lack of particles freely flowing in the simulation prevents this type of movement unless a suitably high grid resolution is used.

The final approach is the hybrid approach. This approach uses both particles and grids to simulate fluids. The grid is used to simplify pressure calculations while the particles are advected independently of each other. This approach capitalizes on the strengths of the particle and grid approaches by maintaining both. By using both a grid and particles, hybrid approaches can avoid the drawbacks inherent to particle and grid approaches. The original FLIP method, which uses this hybrid approach, was presented in [Brackbill and Ruppel, 1986]. The technique provided in Houdini, a popular visual effects software package, is called a FLIP-PIC solver and is based on the [Zhu and Bridson, 2005] paper. The FLIP method is faster and less prone to particle instability when compared to traditional particle-based methods. FLIP approaches can make use of many improvements made in both particle-based and grid-based methods, but several FLIP specific approaches are notable. Observing that fluid particles residing far from the surface could be effectively ignored, [Ferstl et al., 2016] proposed narrow band methods that improve simulation efficiency while preserving simulated fluid movement. [Um et al., 2014] and [Ando et al., 2012] improved simulation quality by modifying the distribution of particles within the grid. While not implemented in this project, these FLIP-specific improvements could be used in future work.

With years of advancement in fluid simulation approaches, fluid simulations exhibit remarkably believable movement, yet are still too slow to compute in real time. In section 2.2 we discuss machine learning techniques, then in section 2.3 we explore how machine learning has been applied to fluid simulation, and how it has provided a way to quickly simulate fluids while sacrificing some accuracy.

2.2 Machine Learning

Machine learning is a topic that in recent years has experienced an explosion of interest. One of the reasons for this is the recent advances in graphics card (GPU) technology that has made machine learning approaches such as convolutional neural networks computationally feasible. Convolutional neural networks in particular have now become a viable machine learning tool due to GPU technology.

Machine learning refers to computer algorithms that attempt to provide correct output predictions when given a set of inputs. Typically these algorithms are "trained" with large data sets that map input data with correct output data. The algorithms adapt their internal parameters as they train with the training data in an attempt to yield output predictions as close as possible to the expected results in the training data. The goal is that once the machine learning algorithms have trained for a sufficient amount of time, the predictions they provide when given novel data will be close to what the actual output should be.

A simple example of machine learning is predicting points on a line. Without knowing the equation of a line, a machine learning algorithm can be given hundreds of points on the line. As the machine learning algorithm attempts to predict the y coordinate when given an x coordinate, it can adjust internal parameters slightly in the direction of correcting the error that it observed. The algorithm adjusts itself as it iterates over all the training data and eventually it can accurately match the equation of the secret line, provided it has enough internal parameters and data. In a similar manner, problems much more complicated than predicting points on a line can be solved by more advanced machine learning algorithms.

Recent applications of machine learning have proven very effective in solving historically difficult problems. Computer vision problems such as classification [Krizhevsky et al., 2012] and object detection [Girshick et al., 2014] have become commonplace due to machine learning. The ability for computers to comprehend text documents [Sebastiani, 2002], translate text into new languages [Amodei et al., 2016], and recognize human speech [Hinton et al., 2012] have all become reliant on machine learning techniques.

Machine learning approaches are varied. Support vector machines [Cortes and Vapnik, 1995], clustering [Jain and Dubes, 1988], genetic algorithms [Holland, 1992], decision trees [Breiman, 2001][Breiman, 2017], and neural networks [Haykin, 1994] are just some of the most common machine learning approaches. Different machine learning approaches are suitable for solving different problems.

2.2.1 Convolutional Neural Networks

While there are several different machine learning approaches, the approach that fits well for fluid simulations has been shown to be convolutional neural networks (CNN) [Tompson et al., 2016]. Convolutional networks first proved effective as a machine learning tool in [LeCun et al., 1990], but later exploded in popularity in 2012 when CNNs proved extremely effective at image classification during the ImageNet LSVRC-2012 competition [Krizhevsky et al., 2012]. Since then, CNNs have been shown to be very effective in a wide variety of regression and classification tasks.

CNNs operate by feeding data into the network and then running operations on that data as the data travels from the start to the end. These operations are commonly referred to as layers. Typical operations that make up these layers include convolutions, pooling, and activation layers. By combining these operations and adjusting the various parameters of each layer, a final CNN design is constructed.

2.3 Simulating Fluids with Machine Learning

Integrating machine learning into fluid simulations is quite a recent area of research, with papers starting to appear just a few years ago. Just like the advent of machine learning research a few years prior, the viability of fluid simulations being driven by machine learning has also only recently increased enough to be worth considering due to developments in the compute capabilities of GPUs. The earliest work to take this approach was [Jeong et al., 2015]. [Jeong et al., 2015] used regression forests to predict changes in the position of particles

between each timestep. This technique uses a Lagrangian or particle-based approach where every particle in the simulation is evaluated separately. Various features are formulated for each particle from the properties of nearby particles and then passed into a regression forest solver to predict the acceleration of that particle for the next frame. Since [Jeong et al., 2015] use a particle-based approach, many of the techniques can't be applied directly to an Eulerian or grid-based approach, necessitating development of new techniques specific to grid-based approaches.

However, we can learn key ideas that facilitate machine learning for fluid simulations. We learn that the features given to the machine learning algorithm need to represent the state and context of the fluid well enough that the algorithm has enough information to correctly predict subsequent states of the fluid. Representing the fluid in a format that the machine learning algorithm can learn from is critical for the success of the machine learning model.

The following year [Yang et al., 2016] and [Tompson et al., 2016] both showed success using machine learning with an Eulerian fluid simulation approach. [Yang et al., 2016] used a machine learning algorithm called a multilayer perceptron that took pressure, velocity, and collision data as input and output predicted pressure values. The pressure values were then used in the traditional fluid simulation equations to influence the velocities of the grid cells. This approach of only swapping out the pressure solve step differed from the approach of [Jeong et al., 2015] that sought to use machine learning to drive the complete fluid simulation process. By focusing on the pressure solve, the machine learning algorithm only needs to solve a single step rather than try to predict the full fluid simulation. The pressure solve, as mentioned earlier, is the most time-consuming step and is thus a great target for utilizing fast machine learning approximations.

[Tompson et al., 2016] continued the trend of [Yang et al., 2016] in predicting pressure values on a grid-based fluid simulation using machine learning tools. This work utilized a convolutional neural network that took collision geometry and velocity divergence as inputs

and predicted pressure values. As with [Yang et al., 2016] all the rest of the steps of the fluid simulation were left to their original implementation, showing how the machine learning approximation of the pressure solve step could easily be swapped in for the traditional pressure solve algorithms. The CNN model used in [Tompson et al., 2016] proved much more capable than the multilayer perceptron used in [Yang et al., 2016]. In fact, the multilayer perceptron used in [Yang et al., 2016] could easily be reformulated exactly as a simple CNN with just a couple of layers of convolution. Utilizing several convolution layers and down sampling techniques, the [Tompson et al., 2016] model manages to get impressive results both visually and in speed. The model was able to model a 128x128x128 grid of smoke at 47 FPS.

Other researchers have employed machine learning for tasks other than attempting to simulate fluid motion directly. [Xie et al., 2018] trained a conditional adversarial network to generate high-resolution details for a low-resolution simulation. By utilizing the trained neural network, [Xie et al., 2018] effectively transforms a low-resolution simulation into a high-resolution simulation in real time. [Ma et al., 2018] utilized machine learning to control a fluid source that would in turn affect rigid bodies and have them exhibit desired behaviors. The [Ma et al., 2018] approach successfully learned fluid motion in order to properly drive the fluid source to reach certain goals. [Um et al., 2018] learned to predict splashes from high resolution simulations and applied them to lower resolution simulations to give detail that would otherwise take much longer to simulate.

2.4 Unsolved Problems with Machine Learned Fluid Simulations

While both grid-based and particle-based approaches have been explored, the hybrid FLIP approach has yet to have its suitability for integrating with machine learning tools investigated. Since the FLIP approach can benefit from the capabilities of particle-based and grid-based approaches, it could be even more effective than previous machine learning attempts. Given that the FLIP approach is used often in visual effects software packages, the lack of knowledge pertaining to the suitability of a machine learning driven FLIP approach is something that

needs to be rectified. Previous efforts such as [Jeong et al., 2015] and [Tompson et al., 2016] suggest that a hybrid solver such as a FLIP solution could also benefit from machine learning techniques. Since the FLIP method is typically used in film and video games, this solution has the potential to significantly speed up fluid simulation in these two applications.

Simulating liquids such as water is absent from both [Yang et al., 2016] and [Tompson et al., 2016]. Both papers only show simulated gases such as smoke. Simulating gases is quite a bit more forgiving of errors when compared to liquid simulation. Since results showing successfully simulated liquids would be very impressive, it can be assumed that the absence of these results suggests more work must be done to utilize these two approaches for grid-based simulations of liquids. These holes of knowledge lend themselves to necessitating this body of work which aims to address these issues.

The design of the machine learning model is also something that can be improved upon. [Tompson et al., 2016] presents a single design for the CNN model driving the fluid simulation, but years of machine learning research has shown that improvements to previous designs can always be made. In addition, [Tompson et al., 2016] does not detail the tradeoffs in runtime and accuracy for alternate model designs or parameter adjustment. For fluid simulations to make use of machine learning in real-time applications, an account detailing tradeoffs between runtime, accuracy, and other considerations is necessary. These measurements can help developers wishing to utilize machine learned fluid simulations account for expected performance penalties for their targeted fluid simulation accuracy.

Chapter 3

Thesis Statement

A FLIP fluid simulation with the pressure solve step replaced with a machine learning regression algorithm running on a GPU can achieve real-time results for a large body of fluid in an interactive application. A machine learning approach to FLIP fluid simulation can simulate liquids while continuing to outperform Eulerian grid approaches.

Chapter 4

Solution

This chapter details our approach to fluid simulation using machine learning. Section 4.1 details our changes necessary to fix current state-of-the-art approaches to properly simulate liquid substances. Previous approaches were limited to gas simulations and failed to simulate liquids. Section 4.2 details the benefits of utilizing FLIP rather than a grid-based simulation approach. We explore how our introducing machine learning into FLIP allows for finer details in fluid simulations while keeping the simulation grid size consistent. Section 4.3 explores the design of the machine learning model to optimize accuracy and runtime. Previous work presented a single design without presenting the tradeoffs of alternate designs. This data will help those implementing machine learning driven fluid simulators make informed design decisions. Finally, section 4.4 summarizes the work presented here.

4.1 Fixing Fluids

The [Tompson et al., 2016] paper provided the state-of-the-art basis for our machine learning driven fluid simulator. This approach was shown to work well enough on simple smoke plumes, but lacked any results showing liquid simulation. This omission suggested that the [Tompson et al., 2016] approach does not work correctly with liquids. Our initial attempt to adapt the specifics of this approach to simulate liquids was not successful, confirming this limitation. The simulations attempted with this approach simply collapsed upon themselves, forming a shallow puddle at the bottom of the simulation space. The remainder of this section details

adjustments that need to be made to the [Tompson et al., 2016] approach in order to apply that approach to the simulation of liquids.

In our implementation of their approach, the CNN always predicted positive pressure values. The effect of this faulty behavior prevented the CNN from being able to properly learn from the training data. Unable to learn properly, the models were inaccurate as well. The fix to this problem was found to be adjusting the final layer of the CNN model detailed in [Tompson et al., 2016]. The original model employed a Rectified Linear Unit (ReLU) layer after the very last convolutional layer in the network. A ReLU is an activation function that has become the most commonly used activation function in deep learning models. A ReLU has a very straightforward effect on the data that it is applied to. A ReLU will return 0 if it receives any negative input and will leave positive input unchanged. As an activation function, it serves to help a CNN account for non-linear effects and interaction effects. The impact of activation functions in machine learning is very pronounced and helps them achieve the non-linear results that deep learning is known for. In most deep learning networks, a ReLU layer will follow most blocks of convolution layers. This pattern is extremely common in deep learning research, so much so that it is expected as a foregone conclusion, but in our fluid simulation CNN, it was found to have an adverse effect. In the case of predicting pressure values, the final ReLU in the CNN model had the effect of only allowing predicted pressure values to be positive. Since the training data has both positive and negative pressure values, this limitation severely limited the CNN's ability to train itself. By omitting the final ReLU, the CNN model was finally able to train properly and produced much better results. Since the state-of-the-art model we adapted from [Tompson et al., 2016] was tested on smoke plumes, the adverse effect that the final ReLU layer had easily went unnoticed. Smoke plumes tend to always expand, meaning pressures are generally positive, so the fact that the model could only output positive pressure values instead of both negative and positive pressure values was easy to miss. In summary, skipping the final ReLU function in our new model drastically improved the machine learning model's ability to simulate liquids.

The effect of fixing the CNN model design, while a significant improvement, was not sufficient to accurately model liquids. To further improve the results, we had to adjust the input parameters to allow the model to more accurately account for the state of the fluid simulation. The [Tompson et al., 2016] model utilizes only two fields of data, the geometry field and the velocity divergence field. The geometry field is a boolean occupancy field to specify the presence or absence of solid obstacle geometry. The velocity divergence field is the divergence of the velocity components. Using only these two fields failed to achieve any reasonably realistic results. While they seem sufficient for smoke, the amount of information available to the CNN was insufficient for properly simulating liquid motion. In particular, the fluid simulation struggled to maintain mass and collapsed upon itself as information about nearby fluid cells was absent from the input data. In an attempt to provide more information to the CNN, we tested several extra points of data. The first approach was providing more information about the state of each cell in the grid. The original approach only provides the geometry field to the model. This field only contains information about the presence or absence of solid geometry in a cell. For a liquid simulation, this is insufficient since there are effectively 3 states of matter for cells in the grid: liquid, solid obstacle, or air. Modeling these three states in a single field where -1, 0, and 1 represented different states failed to generalize well, so a different approach was discovered. After experimenting with several approaches, we found that separating this data into different fields rather than attempting to design numerical values that represented each state in a shared field worked quite well.

Additional fields were also discovered that had a moderate effect on accuracy, but to a lesser extent than previous fields. While the original model uses velocity divergence passed in as a field, additionally passing in the individual velocity vector component fields proved fairly effective in increasing accuracy. Taking this idea even further, new fields derived from the product of pairs of velocity components also proved useful, to a lesser extent. The main idea driving the experimentation of different input fields is that the more context that the CNN

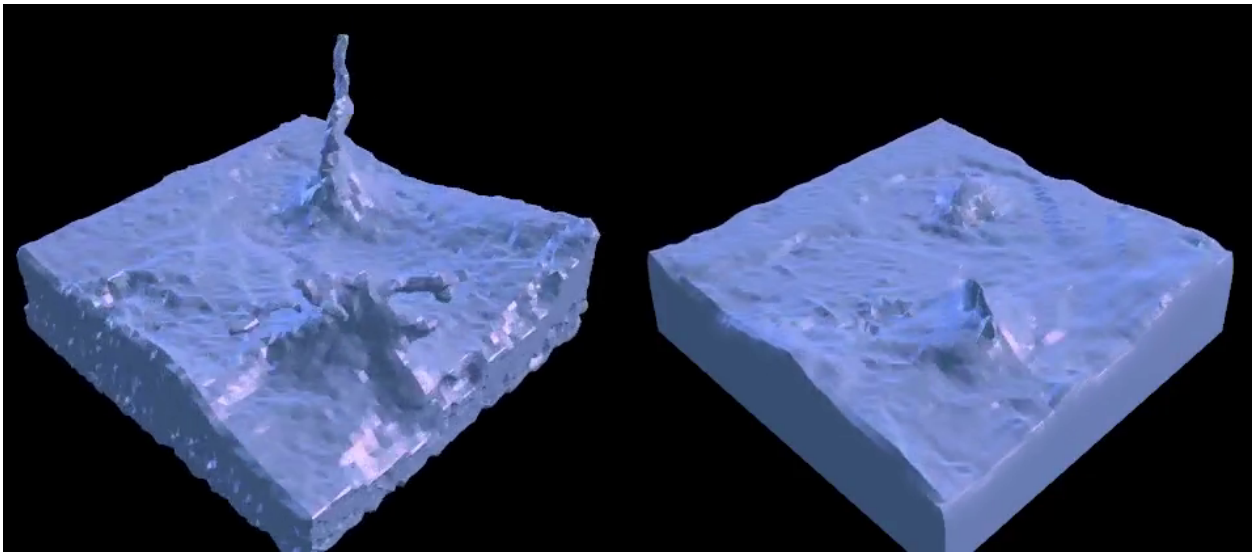

Figure 4.1: The particles of the FLIP approach (Left) enables fluid motion that a Grid approach (Right) cannot achieve. The FLIP approach can more easily result in droplets and spray.

has about the state of the fluid simulation, the more accurately it can model the movement of the fluid. The effect of these new input fields is shown in Table 4.3.

In pursuing accuracy, we discovered several new input fields that increase the accuracy of the resulting CNN model. These input fields, however, come at the cost of computation time. The details of the tradeoffs between simulation time and accuracy will be explored in Section 4.3 along with a summary of other CNN parameters that were analyzed.

4.2 FLIP vs. Grid

The FLIP approach shares many similarities to the Eulerian approach, but also has a few important differences. In terms of the pressure solve step, both the Eulerian approach and the FLIP approach operate the same. This similarity enabled an easy way to compare both approaches when utilizing the machine learned pressure solve step. We ran identical simulations comparing a FLIP approach and a grid approach both using our machine learning model. The differences in the resulting fluid motion can be observed in Figure 4.1.

The overhead of the FLIP approach accounted for 50% more runtime, but resulted in a simulation that featured droplets and splashes forming. For a Grid simulation to form similar levels of varied fluid movement, such as splashes as seen in the FLIP simulation, it needs a much higher grid resolution. This increase in grid resolution would result in much higher runtimes. As such, the extra cost of the FLIP approach is offset by our ability to achieve small details with smaller grid resolutions. This saves both time and memory, both important for real-time applications. Our method confirms our hypothesis that an approximated pressure solve designed for a grid approach will also work in a FLIP approach. For a fluid simulation targeting a desired level of detail, utilizing FLIP instead of a grid approach provides a significant performance gain.

The FLIP formulation of the problem also exposes more data to potentially be utilized in the machine learning model. In particular, the FLIP fluid simulation maintains particles within the grid. These particles are a source of extra information. We tested integrating particle counts within cells and other means of manipulating the particle data, but the results were subpar. In particular, the process of counting the particles took a significant amount of time. This process could have potentially been optimized, but the benefit of adding this data to the machine learning model was quite small. In Section 6.1 we discuss the merits of future works utilizing particle data.

4.3 Optimal Model Parameterization

In previous sections we explored the fixes needed to get fluids working correctly in our CNN model. In this section we will explore the particular parameters and details of our CNN as we attempt to maximize accuracy while making note of the performance tradeoffs. The way the CNN is designed and choices made in the input field are the largest factors affecting the performance and accuracy metrics. In an attempt to discover the best possible choices for these considerations, we discuss the effect of these decisions below. Experimental data presented in this chapter is presented in relation to our baseline approach to show the relative

Filter Size	Runtime (seconds)	Runtime (% of baseline)	Error (% of baseline)
3	0.025	100%	100%
5	0.041	164%	90.4%
7	0.090	360%	150%
9	0.144	576%	380%

Table 4.1: Filter sizes of 3 and 5 work well. Larger filter sizes increase runtime and decrease accuracy.

effect of these parameter choices. Error values are derived from the squared difference of velocity divergence values after predicted pressure values are applied to the velocity. As these error values fluctuate depending on the fluid simulation scene setup, we present error as a percentage to show the relative effectiveness of the chosen experimental parameter.

4.3.1 Filter Size

To begin, we explored the filter size of our convolution layers. The filter size defines the size of the convolution window that operates on the data. A small filter size will only allow convolutions to consider small areas of the input fields, whereas a larger filter allows the convolution operation to act on larger patches of data. The bigger the filter, the more information there is available for the CNN to learn from. While typically a larger filter can lead to a more accurate CNN model, only by conducting tests with various filter sizes will the best option present itself. We explored filter sizes of 3, 5, 7, and 9 and found that 7 and 9 performed very poorly. Larger filters necessitate using more filters to be effective and the amount of filters that a filter size of 7 or 9 would require to be effective exceeded the limitations of our test setup. The filter size of 5 performed best, beating out the filter size of 3. A larger filter size, however, comes at a cost of performance. The error from the model using a filter size of 3 was 11% more than the model using a filter size of 5. This accuracy, however, came at the cost of a 64% longer simulation time. The simulation takes 0.025 seconds for the model using a filter size of 3 compared to 0.041 seconds for the model using a filter size of 5, with all other parameters remaining constant between them. As such, we use

Filters	Runtime (seconds)	Runtime (% of baseline)	Error (% of baseline)
4	0.020	80%	119%
8	0.025	100%	100%
16	0.033	132%	87%
32	0.043	172%	80%

Table 4.2: Increasing the number of filters can have a large effect on accuracy, but adversely affects runtime to a similar degree.

a recommended filter size of 3 for our simulations. A summary of these experiments testing various filter sizes can be seen in Table 4.1.

4.3.2 Number of Filters

Another model parameter is the number of filters. Similarly to the filter size, the number of filters also has an inverse effect on simulation time and simulation accuracy. The more filters allowed in the CNN, the larger the amount of options the network has to transform the data at each layer. The filter count also has a significant impact on memory usage on the GPU. We need to keep the filter count low enough that the model can fit in RAM on modern graphics cards but high enough to allow the model enough accuracy. We experimented with filter counts of 4, 8, 16, and 32 in size. When comparing a model with 8 filters to a model with 32, the model with 8 filters had an error rate of 125% than that of the 32 filter model. On the other hand, the 32 filter model needed 72% longer to compute than the 8 filter model. The simulation takes 0.025 seconds for the model using 8 filters compared to 0.043 seconds for the model using 32, all other parameters being equal. With speed being our main goal, we use 8 filters in our simulations. A summary of experiments testing various filter counts can be seen in Table 4.2.

4.3.3 Input Data

The choice of input data provided to the CNN, in our case, liquid simulations, is another parameter that affects simulation accuracy and runtime. A CNN utilizes input data to make

Input Data	Runtime (seconds)	Runtime (% of baseline)	Error (% of baseline)
Tompson (Geometry and Velocity Divergence)	0.0243	98%	234%
Baseline (Geometry, Fluid, Air, and Velocity Divergence)	0.0247	100%	100%
Baseline and Velocity	0.0251	102%	62%
Baseline, Velocity, and Velocity products)	0.0264	107%	58%

Table 4.3: This table shows how additional input data affects the accuracy of the CNN model at the cost of runtime. Additional input data has a large effect on accuracy but a small effect on runtime.

informed predictions. The more data provided to the CNN, the more accurate the CNN can potentially be. The potential accuracy gains for additional data is dependent on how well that data represents the fluid. With more data, the CNN will also be slower because it must operate on more data. We experimented with various sets of input data. A model provided with just geometry and velocity divergence data as described in [Tompson et al., 2016] had an error rate 134% more than our model provided with geometry, fluid, air, and velocity divergence data. We refer to the set of input data consisting of geometry, fluid, air, and velocity divergence as our baseline as that is the input data choices determined in section 4.1. We also tested adding velocity data and even products of velocity components. Adding velocity data resulted in a model 162% more accurate than our baseline, while only accounting for 2% longer runtime. The choice of input data was found to be an extremely efficient means for increasing accuracy. A summary of the effect of input data choices can be seen in Table 4.3.

4.3.4 Model Layer Design

In addition to testing models of varying filter size, filter count, and input data choices, further tests were conducted to alter the design of the CNN model. The first approach was adding more layers of convolution before the final convolution layer. This approach failed to significantly improve accuracy. Another approach was adding more pooling layers, this

approach also failed to provide any significant improvement. In addition, we attempted to sanitize the input data by normalizing the values or scaling them by various input data values, but these too failed to result in significant improvements. These alternate designs proved to have little impact on simulation time or accuracy. As such, the basic layout detailed in [Tompson et al., 2016] was mostly retained.

4.3.5 Recommended Parameters

From the data presented in tables 4.1, 4.2, and 4.3, we can observe several key results and determine recommended parameters. The most cost effective means to increase accuracy is to provide more input data to the model. Additional input data isn't computationally expensive because it only affects the first layer of the convolutional neural network. The reduction in error is very significant compared to the small increase in runtime. The next most effective method of improving accuracy is increasing the number of filters in the model. Additional filters increase runtime to a greater extent than additional input data. Finally, large filter sizes such as 7 or 9 perform poorly both in accuracy and runtime. A filter size of 5, however, could be a suitable choice if increased accuracy is desired at the cost of runtime. In an effort to maximize accuracy while striving towards real-time simulation runtime, we arrived at the following recommended parameters. The input data for our recommended CNN consists of geometry, fluid, air, velocity divergence, and velocity data. We also recommend using 8 filters and a filter size of 3.

4.4 Summary

This chapter detailed the efforts we made in our goal towards creating a machine learning driven FLIP fluid simulator capable of simulating liquids. In section 4.1 we fixed a state-of-the-art approach to machine learning driven fluid simulation by removing the final ReLU activation layer and providing the CNN with additional input data. These changes enabled the machine learning driven simulator to properly simulate liquids. In section 4.2 we presented

our findings related to utilizing the FLIP approach in our machine learning fluid simulator. We found that the FLIP approach works well with the machine learning fluid simulation approach while also operating at a lower grid resolution. In section 4.3 we explored many design options for our machine learning model. We discovered the parameters that maximize accuracy while keeping runtime close to our goal of real-time.

Chapter 5

Results

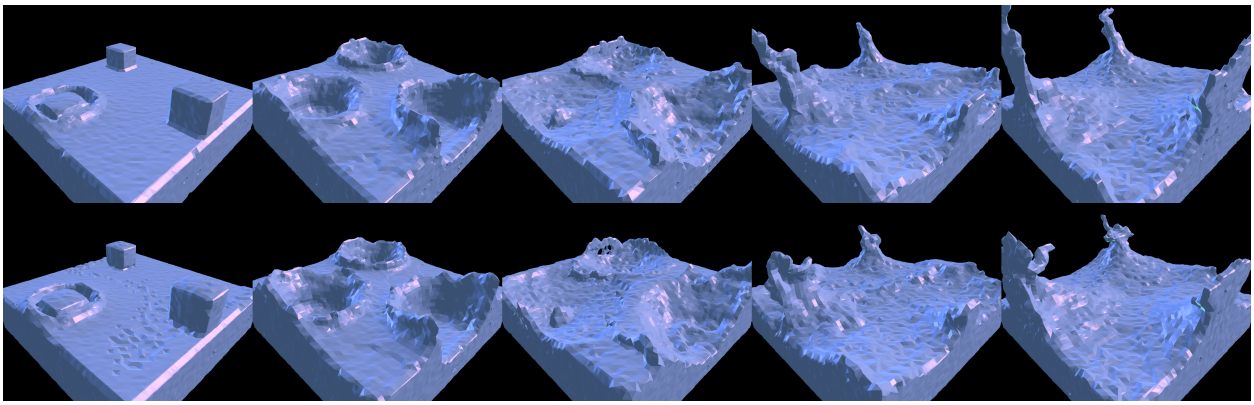

Figure 5.1: Identical simulations run on the original FLIP implementation (Above) vs. our Machine Learning driven FLIP fluid simulation (Below). Fluid motion is not exact, but is visually similar.

The simulation results presented here were generated from a model trained using the recommended parameters presented in section 4.3.5. The model used 8 filters at each step with a filter size of 3. The input data consists of the geometry, fluid, air, velocity divergence, and velocity data. These parameters were chosen as they best approached our goal of "real-time" while maintaining low error. As such, the results presented here represent results that could be expected for a real-time application such as a video game.

Figures 5.1 and 5.2 shows one run from our machine learning driven fluid simulator. The top row shows a fluid simulation using our original FLIP implementation. The second row shows results from the simulation run with our machine learning algorithm. Ideally the results would be exactly the same with the machine learning algorithm perfectly replicating the pressure solve step of original implementation, but small differences can be seen. This

approximated fluid motion closely matches results from the original FLIP implementation. In a real-time application, exact results are not necessary, so slight deviations from the expected results is acceptable. The goal of this machine learning driven simulation approach is believable fluid motion for real-time applications. We believe that since our results closely match the target simulation, this goal has been reached. These results confirm that a machine learning driven FLIP fluid simulation can approximate the unmodified simulation. Figures 5.3, 5.4, and 5.5 show additional results with different scene setups. The fluid motion of the original FLIP implementation and our machine learning implementation will slowly diverge more and more as time progresses and errors from previous timesteps persist. This effect is visible in the figures presented, but is not an issue because the end user will not be able to notice. Figures 5.6 and 5.7 present a realistic rendering of the fluid. With the fluid now transparent, deviations from the target simulations are even less noticeable to the viewer.

These results confirm our hypothesis that a machine learning approach to FLIP fluid simulation can successfully simulate liquids. Our results are believable and have more fluid detail than a comparable Eulerian approach (see section 4.2). Our fluid simulator approaches real-time allowing for on-the-fly liquid simulations in interactive applications such as games (see section 4.3).

While our goal of 60 FPS is still out of reach, approaching 30 FPS by using just 8 filters and a filter size of 3 in our un-optimized implementation on consumer GPU hardware is encouraging. With GPU technology constantly improving and specialized machine learning hardware such as Tensor Processor Units (TPUs) becoming commonplace, we are hopeful that real-time liquid simulation will be possible in the near future.

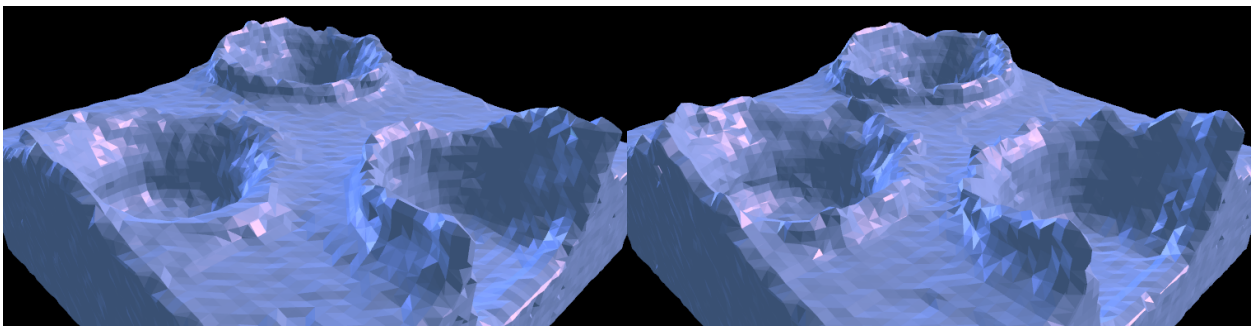

Figure 5.2: A detailed view of a frame from 5.1. Slight differences between the original FLIP implementation (Left) and our machine learning driven FLIP fluid simulation (Right) are noticeable.

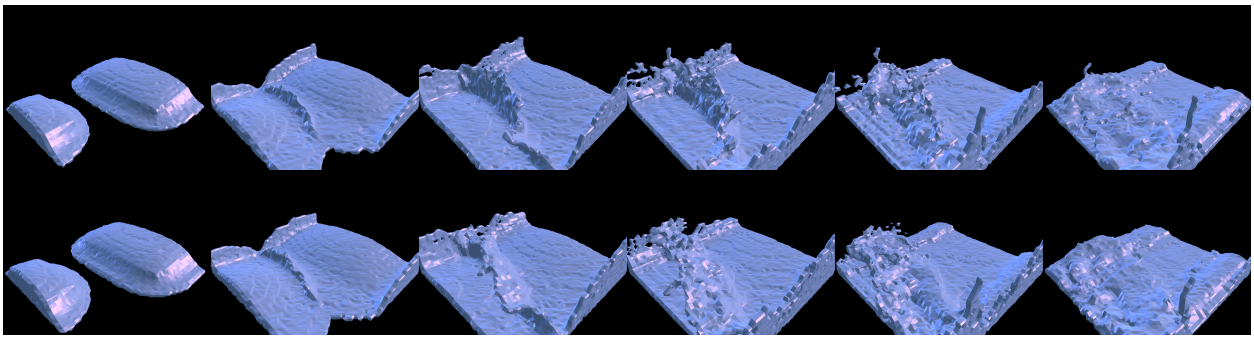

Figure 5.3: An additional result showing a simulation of waves crashing with the original FLIP implementation (Above) vs. our machine learning driven FLIP fluid simulation (Below).

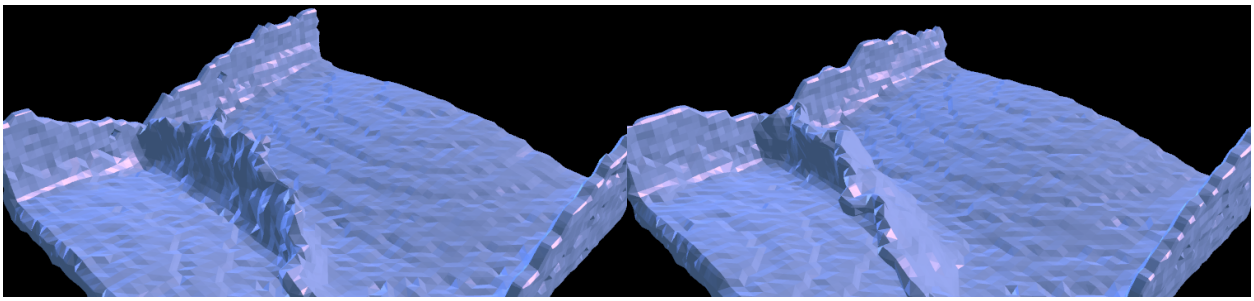

Figure 5.4: A detailed view of a frame from 5.3. Differences between the original FLIP implementation (Left) and our machine learning driven FLIP fluid simulation (Right) are noticeable.

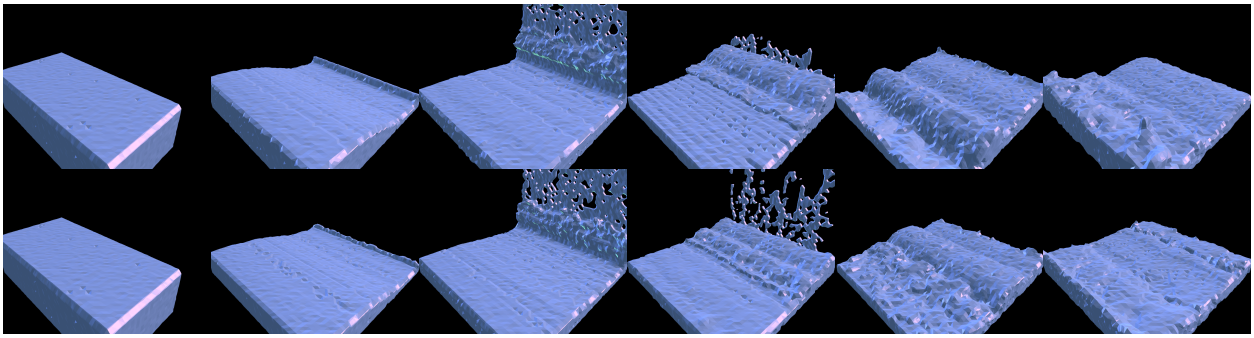

Figure 5.5: An additional result showing a simulation of a dam break with the original FLIP implementation (Above) vs. our machine learning driven FLIP fluid simulation (Below).

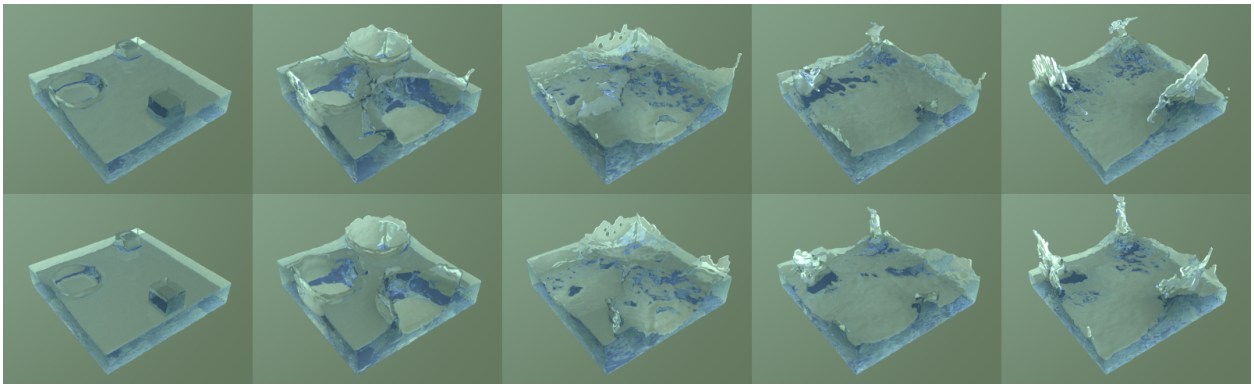

Figure 5.6: An additional result showing a realistic rendering of the fluid simulation with the original FLIP implementation (Above) vs. our machine learning driven FLIP fluid simulation (Below).

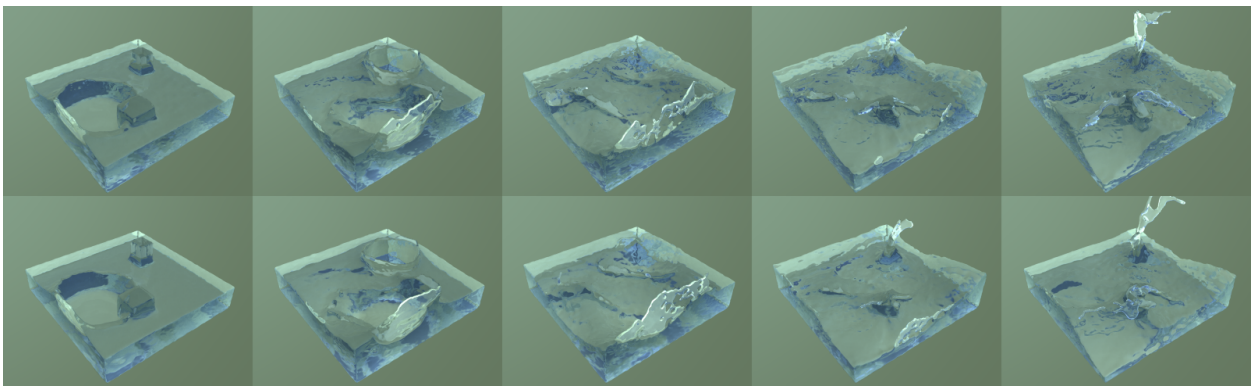

Figure 5.7: An additional result showing a realistic rendering of the fluid simulation with the original FLIP implementation (Above) vs. our machine learning driven FLIP fluid simulation (Below).

Chapter 6

Conclusion

We have presented a technique for using convolutional neural networks for performing fluid simulations. We have implemented this approach using FLIP fluids in an attempt to allow for any arbitrary fluid to be simulated. To sum up the contributions of this body of work, we adapted a state-of-the-art machine learned fluid simulation approach to work with liquids, we explored using the FLIP fluid approach to fluid simulation with the machine learning model, and we quantified performance and accuracy tradeoffs for various parameters of the CNN model. To augment our original CNN to properly simulate liquids, we fixed the model to properly predict both positive and negative pressure values by omitting a particular ReLU layer. In addition, we provided additional input data to the CNN derived from the state of the fluid simulation. These changes enabled the CNN to properly account for and predict the fluid pressure values. By utilizing a FLIP approach to fluid simulation, the simulation can remain low resolution while allowing for fine details such as splash droplets. Finally, we explored various parameters to discover the tradeoff between accuracy and runtime. With real-time fluid simulation at the cost of accuracy being the goal of this research, utilizing a filter size of 3, all effective input fields, and as few filters as possible will result in acceptable fluid movement while approaching real-time simulation times.

6.1 Future Work

This thesis detailed a method of using machine learned fluid simulation to simulate water. Simulating other fluids was not tested and may necessitate further refining of our machine

learned fluid simulation approach. The FLIP approach has been used to simulate fluids of various viscosities in addition to mixed fluid simulations. We anticipate that machine learning-based approaches attempting to approximate these types of simulations will necessitate additional input data. Discovering suitable input data and augmenting the approach detailed in this thesis is left as future work.

References

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in English and Mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.

Ryoichi Ando, Nils Thurey, and Reiji Tsuruno. Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1202–1214, 2012.

JU Brackbill and HM Ruppel. FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational Physics*, 65(2):314–343, 1986.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Leo Breiman. *Classification and Regression Trees*. Routledge, 2017.

Mark Carlson, Peter J Mucha, R Brooks Van Horn III, and Greg Turk. Melting and flowing. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 167–174. ACM, 2002.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3): 273–297, 1995.

Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183(1): 83–116, 2002.

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*, pages 15–22. ACM, 2001.

Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. Narrow band FLIP for liquid simulations. In *Computer Graphics Forum*, volume 35, pages 225–232. Wiley Online Library, 2016.

Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 23–30. ACM, 2001.

Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

Olivier Génevaux, Arash Habibi, and Jean-Michel Dischler. Simulating fluid-solid interaction. In *Graphics Interface*, volume 2003, pages 31–38, 2003.

Robert A Gingold and Joseph J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181 (3):375–389, 1977.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

Simon Haykin. *Neural Networks: a Comprehensive Foundation*. Prentice Hall PTR, 1994.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT press, 1992.

Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. ISBN 0-13-022278-X.

SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, Markus Gross, et al. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)*, 34(6):199, 2015.

ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jaroslaw R Rossignac. Flowfixer: Using BFECC for fluid simulation. Technical report, Georgia Institute of Technology, 2005.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404, 1990.

Leon B Lucy. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82:1013–1024, 1977.

Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. Fluid directed rigid body control using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 37(4):96, 2018.

Miles Macklin and Matthias Müller. Position based fluids. *ACM Transactions on Graphics (TOG)*, 32(4):104, 2013.

Joe J Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30(1):543–574, 1992.

Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 154–159. Eurographics Association, 2003.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)*, 34(1):1–47, 2002.

Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable MacCormack method. *Journal of Scientific Computing*, 35(2-3): 350–371, 2008.

Barbara Solenthaler and Renato Pajarola. Predictive-corrective incompressible SPH. In *ACM Transactions on Graphics (TOG)*, volume 28, page 40. ACM, 2009.

Jos Stam. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. *arXiv preprint arXiv:1607.03597*, 2016.

Kiwon Um, Seungho Baek, and JungHyun Han. Advanced hybrid particle-grid method with sub-grid particle correction. In *Computer Graphics Forum*, volume 33, pages 209–218. Wiley Online Library, 2014.

Kiwon Um, Xiangyu Hu, and Nils Thuerey. Liquid splash modeling with neural networks. In *Computer Graphics Forum*, volume 37, pages 171–182. Wiley Online Library, 2018.

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. *arXiv preprint arXiv:1801.09710*, 2018.

Cheng Yang, Xubo Yang, and Xiangyun Xiao. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424, 2016.

Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Transactions on Graphics (TOG)*, 24(3):965–972, 2005.

Appendix A

Project Setup

To train a machine learning model, we must first obtain training data. Our research requires large amounts of fluid simulation data. We decided on utilizing a fluid simulation software called Mantaflow. Mantaflow is designed for the fluid research community making it a suitable choice for our research setup. Mantaflow is distributed as source code allowing us to modify its behavior as much as we need to facilitate our workflow. This flexibility proved a great asset as it allowed for modifications to be implemented that integrated our machine learning algorithm into the pressure solve equations without the added burden of reimplementing the Navier-Stokes fluid equations. Using Mantaflow we were able to generate hundreds of different fluid simulations each a few seconds long for training data.

We generated a large number of fluid simulations in Mantaflow by randomizing various fluid scene setups. Among the randomization parameters were fluid pools, droplets of water splashing, various timescales, and various simulation space sizes. This randomization serves to provide the machine learning algorithm more variety so it can adapt more generally to novel fluid simulations. Without this randomization or without sufficient randomization, our resulting machine learning model would not perform well when it attempts to simulate a new fluid setup.

Timescale randomization varied in value from 15 frames per second to 60 frames per second. Effectively this randomization attempts to allow our model to generalize to frame rates between 15 and 60 frames per second. This range of frame rates was determined to be a suitable goal. The benefit of being able to adapt to different frame rates as the available end-user processing power varies is very important. This ability to generalize in this manner will greatly improve the viability of real time fluid simulations in real-time applications.

We also randomize the simulation space in an attempt to generalize to various sizes of simulations. A model that only works for a particular size of fluid would be fairly useless, so designing a model that can generalize to the size of the simulation space is extremely important. We tested various grid sizes including 32x32x32, 64x64x64, and 128x128x128. It should be noted that at evaluation time any size grid could be used as the operations in the machine learning model we use can work at any size of simulation space.

We initialize each simulation with various amounts of liquid at various states. Some water sits at rest in a pool at the bottom and some water is suspended in air about to drop into the pool. This setup is a very commonly seen fluid effect and the motion of the resulting splashes is easy to qualitatively judge. This fluid setup was chosen because the results are easy to verify while the motions are complex enough to sufficiently generalize the machine learning model.

To develop a machine learning model, we used the TensorFlow machine learning framework. There are several machine learning frameworks available, but TensorFlow stood out due to its overwhelming popularity and cutting edge development cycle. TensorFlow can run training and inference on the GPU, which is necessary for our goal of real time fluid simulation. Since both TensorFlow and Mantaflow have Python interfaces, integrating both together was also readily possible. Using TensorFlow, we can load in fluid simulation data from disk and use it to train a convolutional neural network (CNN). The goal of this CNN is to take parameters from the current frame of simulation and predict values for the next frame of simulation. Previous works such as [Jeong et al., 2015] predicted the resulting velocity of the fluid particles, but [Tompson et al., 2016] and [Yang et al., 2016] opted to predict only the pressure values and allow the original fluid simulation equations to take those pressure values and apply them to the velocity values. The idea of predicting just the pressure values is that it allows the machine learning model to focus on just the hard step in the fluid simulation equation and allows the algorithm to not have to learn an even broader inference problem. We adopted the [Tompson et al., 2016] approach as our baseline as it seems to allow the machine learning to focus on the part of the problem that most needs a speedup. Taking the resulting pressure values and using them to update the velocity values is computationally inexpensive as well.

Since the machine learning model in [Tompson et al., 2016] has the most impressive results of the papers published to date, we also adopted the CNN layout in our research efforts as a starting point. The [Tompson et al., 2016] model can be seen as the state-of-the-art model for grid-based formulations of machine learned fluid simulations. Since the algorithms guiding both the FLIP approach and normal grid-based approaches are so similar, especially the steps for the pressure solve, using the [Tompson et al., 2016] model for our FLIP approach is a suitable plan. This model was tweaked as we searched for the best performing model for liquid FLIP simulations.

The driving factor in our research efforts is the ability to quickly test and evaluate new models. Our project was designed to allow for easy and quick training of various simulation parameters. To accommodate this goal, we designed our training program to consult a configuration file. This configuration file contained simulation parameters detailing the design

of the CNN model, training data to utilize, and other options. By quickly iterating on model designs and other parameters, we were able to quickly test theories and determine the most effective model design elements.

Our code also takes care of tracking accuracy and evaluation time, which is detailed in Section 4.3. Using this data we were able to arrive at our recommendation of optimal parameters for real-time fluid simulation.