Retiming Smoke Simulation Using Machine Learning

Samuel Charles Gérard Giraud Carrier
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Physical Sciences and Mathematics Commons

BYU ScholarsArchive Citation
Theses and Dissertations. 8106.
https://scholarsarchive.byu.edu/etd/8106

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact ellen_amatangelo@byu.edu.
Retiming Smoke Simulation Using Machine Learning

Samuel Giraud-Carrier

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

Seth Holladay, Chair
Parris Egbert
Ryan Farrell
Chris Archibald

Department of Computer Science
Brigham Young University

Copyright © 2020 Samuel Giraud-Carrier
All Rights Reserved
Retiming Smoke Simulation Using Machine Learning

Samuel Giraud-Carrier
Department of Computer Science, BYU
Master of Science

Art-directability is a crucial aspect of creating aesthetically pleasing visual effects that help tell stories. A particularly common method of art direction is the retiming of a simulation. Unfortunately, the means of retiming an existing simulation sequence which preserves the desired shapes is an ill-defined problem. Naively interpolating values between frames leads to visual artifacts such as choppy frames or jittering intensities. Due to the difficulty in formulating a proper interpolation method we elect to use a machine learning approach to approximate this function. Our model is based on the ODE-net structure and reproduces a set of desired time samples (in our case equivalent to time steps) that achieves the desired new sequence speed, based on training from frames in the original sequence. The flexibility of the updated sequences’ duration provided by the time samples input makes this a visually effective and intuitively directable way to retime a simulation.

Keywords: Retiming, art direction, fluid simulation, machine learning.
ACKNOWLEDGMENTS

Thanks go to my advisor, Seth Holladay, as well as the members of the BYU Graphics lab for their helpful feedback throughout the research process.
# Table of Contents

List of Figures vi

List of Tables ix

1 Introduction 1

2 Background 4

2.1 Time Series Estimation ........................................ 4

2.2 Fluid Simulation ................................................... 5

3 Related Work 7

3.1 Time Series in Machine Learning ................................. 7

3.2 Physically-Based Fluid Simulation ............................... 8

3.3 Machine Learning Application to Fluids ......................... 10

3.4 Art Direction of Fluid Simulation ............................... 12

4 Thesis Statement 16

5 Method 17

5.1 Overview ......................................................... 17

5.2 Encoding .......................................................... 20

5.3 Sequence Learning ............................................... 21

5.4 Decoding .......................................................... 22

5.5 Training ........................................................... 23
5.6 Retiming ................................................................. 25

6 Results and Validation .................................................. 27

7 Discussion ................................................................. 31
  7.1 Contribution ......................................................... 31
  7.2 Limitations ........................................................... 32
  7.3 Future Work ........................................................... 32

References ................................................................. 34
## List of Figures

1.1 An example of a slow motion explosion shot from *Matrix Reloaded* (2003).  

2.1 ODE-net learns the time dependent derivative of the function, $f(x)$ to be approximated.  

2.2 Examples of fields that would be used in a basic two-dimensional smoke simulation. Individual values for each field are stored at every voxel location in the simulation space.  

5.1 Our method encodes the first frame of the original simulation sequence, then (given a new timescale) produces a sequence with the desired retiming. During training, the new timescale ($t_0...t_M$) is identical to the input timescale ($t_0...t_N$).  

5.2 A few example frames of simulation data. Density values are discretized over a 3D grid of voxels (boundaries in light pink).  

5.3 A detailed view of the layers in our encoder network. Every orange block is a convolutional layer with the corresponding kernel size (k) and stride (s). A DownBlock is a combination of several convolutions. Each DownBlock is identical in structure (with the exception of data size and number of channels) and we have expanded one of them in the highlighted region. A hyperbolic tangent activation function is used after every convolution operation except the last one.  

5.4 Overview of the ODE-net framework. The ODESolve learns a time-dependent relationship in the latent space of our simulation data, and provides the latent representations, $z$’s, of the new timesteps to be decoded to volume data.  

---

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>An example of a slow motion explosion shot from <em>Matrix Reloaded</em> (2003).</td>
<td>1</td>
</tr>
<tr>
<td>2.1</td>
<td>ODE-net learns the time dependent derivative of the function, $f(x)$ to be approximated.</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Examples of fields that would be used in a basic two-dimensional smoke simulation. Individual values for each field are stored at every voxel location in the simulation space.</td>
<td>6</td>
</tr>
<tr>
<td>5.1</td>
<td>Our method encodes the first frame of the original simulation sequence, then (given a new timescale) produces a sequence with the desired retiming. During training, the new timescale ($t_0...t_M$) is identical to the input timescale ($t_0...t_N$).</td>
<td>17</td>
</tr>
<tr>
<td>5.2</td>
<td>A few example frames of simulation data. Density values are discretized over a 3D grid of voxels (boundaries in light pink).</td>
<td>18</td>
</tr>
<tr>
<td>5.3</td>
<td>A detailed view of the layers in our encoder network. Every orange block is a convolutional layer with the corresponding kernel size (k) and stride (s). A DownBlock is a combination of several convolutions. Each DownBlock is identical in structure (with the exception of data size and number of channels) and we have expanded one of them in the highlighted region. A hyperbolic tangent activation function is used after every convolution operation except the last one.</td>
<td>21</td>
</tr>
<tr>
<td>5.4</td>
<td>Overview of the ODE-net framework. The ODESolve learns a time-dependent relationship in the latent space of our simulation data, and provides the latent representations, $z$’s, of the new timesteps to be decoded to volume data.</td>
<td>22</td>
</tr>
</tbody>
</table>
5.5 A detailed view of the layers in our decoder network. Every orange block is a convolutional layer with the corresponding kernel size (k) and stride (s). An UpBlock is defined as a transpose convolution operation combined with additional layers of convolution. Each UpBlock is identical in structure (with the exception of data size and number of channels) and we have expanded one of them in the highlighted region. Hyperbolic tangent activation functions are used after every layer except the last convolution operation.

5.6 Definition of the three-dimensional Sobel kernel aligned to the direction of the z-axis.

5.7 Example of retiming. Given an initial frame, \( x(t_0) \), and a set of time samples, \( (t_0, t_1, t_2, \ldots, t_M) \), our method produces resulting frames of data for each timestep.

6.1 Comparisons of retiming to the original sequence. From top to bottom: original smoke simulation with a static source, reconstruction using original timing, retimed to twice as slow, retimed to four times slower. Corresponding frames are highlighted.

6.2 An example of retiming a simulation with a moving source. Corresponding frames in the original and the retimed result are highlighted.

6.3 An example of non-uniform retiming a simulation. Corresponding frames in the original and the retimed result are highlighted.

6.4 An example of retiming emissive volumes. Temperature and heat fields were used during training in addition to density values. From top to bottom: original fire simulation sequence, original timing reconstruction, slowed down by a factor of 2. Corresponding frames are highlighted.

6.5 An example of a retimed explosion. From top to bottom: original simulation sequence, original timing reconstruction, slowed down by a factor of 3. Corresponding frames are highlighted.
7.1 Example of inserting target frames in existing simulation.
List of Tables

6.1 Average absolute error values along with training and evaluation times for all data sets. Training times are the average time per epoch. Evaluation times are an average for an inference of 200 frame values. .......................... 28
Chapter 1

Introduction

Figure 1.1: An example of a slow motion explosion shot from *Matrix Reloaded* (2003).

Fluid simulation is a major part of special effects in modern film production. Complex phenomena such as fire, smoke, and water motion are reproduced using a set of physically-based equations to generate interesting visuals. Ultimately, the purpose of these effects is to support story and artistic choices that make a film experience great. Due to the sequential nature and data complexity of these simulations, art direction for these effects is a difficult task. Directing simulation includes the ability to mold the shape, motion, or speed of a fluid at any point in time. However, each of these aspects of control introduce non-physically-based elements which compete with the original simulation. Implementing the desired direction without disregarding the physics-based simulation parameters requires a great deal of iteration and fine tuning. The objective of research in art directing fluids is to provide artists with the
ability to achieve specific shapes and motions while maintaining the physically-based flow provided by simulation.

Oftentimes a director may wish for specific shapes or varied speed in a fluid simulation effect to achieve a more desirable composition that strengthens the story. For example, a film may require a bonfire to grow into the shape of a fiery bird, or large explosion effects may need to be slowed down to give an illusion of greater scale. If a change is needed to adapt the shape or speed of a sequence, the artist will typically have to reconfigure the initial parameters of the simulation and regenerate all the frames necessary for the shot. This can become a very time and resource intensive iteration process requiring hours or even days. We feel a need to develop more efficient ways of directing fluid simulation.

While we could focus on many aspects of art direction, we will focus on the retiming aspect in this work and discuss how the methods presented here might be applied to other art direction problems. While retiming may not seem like the most important aspect of controlling fluid simulation, it is one of the most frequently used forms of art direction for effects. As mentioned above, changing the timing of a sequence will often help give the impression of greater scale. Another way retiming is often used is to slow down specific sections of a shot in order to emphasize the effect, this is particularly done with destruction effects such as breaking glass or bullet trails. The issue of retiming simulations is not a novel concept and several attempts to handle this task have been developed. These past approaches rely on primarily on interpolating values from nearby frames, but interpolation is limited in the sense that the more removed from an actual data sample the interpolated value is calculated, the less accurate the predicted value becomes. Although these approaches typically provide fairly consistent and visually pleasing results, they rely heavily on the velocity values of a simulation. While the velocities are often the most influential force when dealing with the motion of fluids, approaches that do not take into consideration the sourcing information in the simulation are unable to adapt to situations with moving sources. Other common visual artifacts that can be present in retiming problems include density values
jittering from one frame to the next and densities disappearing too quickly or popping into existence without a valid source. We present a different approach attempting to both reduce potential artifacts and increase the flexibility of retiming, by training a machine learning model on simulation data in a time-dependent manner.

Our method makes use of machine learning to approximate the evolution function of a sequence of simulation data. Rather than perform interpolation directly on the density or velocity values stored in the fields of the simulation, our machine learning model learns a time sensitive derivative function of the original density values. While these values are not numerically equivalent to the actual velocity values but rather an encoded representation of velocity, the function learned by our model makes it very simple to retime a simulation. One simply needs to change the time values at which the learned encoding function is sampled and decode these values back to a density representation. Because we are using machine learning, we can only approximate density values and do not guarantee their numerical accuracy. However, we have not noticed any egregious artifacts in our results. The fact that our approach is faster and more flexible, as well as its intuitive use, make this a good approach to retiming, allowing for more exaggerated retimings than other approaches and mobility of sources in retiming.

Also, due to the way the machine learning model is trained, we believe our method to be extendable to other aspects of art direction for fluid simulation beyond retiming. In particular, one area we would like to explore further is the application of our method to target-based animation for fluids. The goal of this work is to present a machine learning method for retiming volumetric simulation data. Our implementation, which makes use of a time-aware network, lends itself well to intuitive and flexible manipulation of a simulation sequence.
Chapter 2

Background

2.1 Time Series Estimation

Because our method to retime simulation is based on a sequential machine learning model, we feel it is appropriate to briefly describe some of the work related to time-dependent sequence generation. Some of the more common approaches to generating a sequence in machine learning use recurrent neural networks (RNN) [5]. A typical example of these is text generation or translation problems. These networks are designed to “remember” information from previous time steps allowing them to have a more accurate estimation of what should follow. While several different models for this sort of learning exist, they act in a fairly similar manner. The focus of such a learning model is to understand the relationship between \( x_t \) and \( x_{t-1} \), thus allowing them to predict the value of \( x_{t-1} \) directly from \( x_t \). The main drawback to this approach is that the further away from the start state, \( x_0 \), one chooses to extrapolate, the less accurate the prediction will be due to accumulated errors.

Figure 2.1: ODE-net learns the time dependent derivative of the function, \( f(x) \) to be approximated.
Another approach is presented in work done by Chen et al. referred to as ODE-net [3]. Their model strives to learn a time-dependent derivative function rather than a straight mapping. To state this more formally, rather than learn the hidden parameters, $\theta$, that will directly provide $f(x)$ at the next timestep, ODE-net learns the function $f'(x, t)$ allowing the parameters defining the direction of the timestep to be specific to each time sample. Figure 2.1 helps illustrate how this is done. In order to predict the next sample in the time sequence one simply adds $f'(x, t)$ to $f(x)$ resulting in the value of $f(x)$ at time $t + 1$.

2.2 Fluid Simulation

Fluid simulation is a ubiquitous part of special effects today. Ranging from rivers to large scale explosions, simulation is what drives these interesting visuals. Simulating fluid motion is typically achieved by solving the Navier-Stokes equations for fluid flow given in equations 2.1 and 2.2. The first equation defining the incompressible nature of fluids and the second giving the motion based on velocities and other applicable forces.

\begin{align*}
\nabla \cdot \mathbf{u} &= 0 \quad (2.1) \\
\frac{\partial \mathbf{u}}{\partial t} &= -(\nabla \cdot \mathbf{u})\mathbf{u} - \frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{F} \quad (2.2)
\end{align*}

The simulation of these equations is performed on a discretized grid and includes voxel fields of numerical values such as the density and velocity. In some instances, additional fields are necessary to keep track of pressure, viscosity, or temperature, in order to compute the fluid’s motion throughout the volume. A diagram explaining these fields is provided in Figure 2.2. The amount of data being manipulated makes fluid simulation a time intensive and rather expensive process. Beyond being simply computationally difficult, this physically-based system is quite rigid and difficult to control without introducing artificial and inaccurate motion. Allowing the artist to have some control over the simulation is a desired ability, but
Figure 2.2: Examples of fields that would be used in a basic two-dimensional smoke simulation. Individual values for each field are stored at every voxel location in the simulation space.

is difficult given the nature of the Navier-Stokes equations and the complexity in translating artist’s desires into simulation data. Art direction of special effects is still an important area of forefront research.
Chapter 3

Related Work

3.1 Time Series in Machine Learning

Whenever we deal with time dependent problems we run into the problem of coherency. Simply generating a long sequence of independent outputs will result in jittering results lacking the smooth flow that must exist over time. This is not just an issue in generating simulation sequences, but occurs in sequential problems in general.

Coherency is particularly important in speech generation. The long short-term memory (LSTM) model [11] has been used in the context of text generation and has been able to create sequences of relatively coherent output. The primary benefit of this method is the ability to keep some information from input steps in the past as you move forward during generation, but a drawback to this model is the limit on the length of the sequence generated.

Yu et al. used a generative adversarial network (GAN) trained on an entire sequence of frames [35] instead of individual frames to mitigate the lack of coherency in video data. However, because this model is trained with a specific sequence length, this framework limits the length of sequences which can be generated. Vondrick et al. were able to generate full sequences [31] and train the model with particular attention to the temporal coherency of the output, but this was very memory intensive. In the case of fluid simulation, the longer the desired output sequence the more memory would be required to generate results.

Another interesting solution to preserving sequence coherency during generation, as proposed by Saito et al., is to separate time as a variable in its own right and then reincorporate it into the model later on [22]. This GAN model made use of two generators,
one which generated a time continuum and one designed to generate images. The abstract
time sequence generated by the first would be fed as input to the second generator. The
transformation of time to this latent space allowed the model to preserve a higher level of
temporal coherency. We aim to similarly include time as a parameter during training for
retiming simulation data of a fluid sequence.

Of particular interest to our research is an approach to machine learning that strays
from the typical formulation. Where most machine learning models try to approximate some
function, \( f(x) \), the work presented by Chen et al. referred to as ODE-net seeks to learn
differently during training [3]. Instead of learning a direct approximation to the function,
\( f(x) \), ODE-net learns a time-dependent approximation of the derivative of \( f \), formally \( f'(x, t) \).
The most interesting part as related to our work is that the formulation of this network
already includes time and provides a very intuitive approach for varying timesteps. We
adopted this method to train off of our cached fluid volume data over time. Other work
presented by Gholami et al. [10] and Sun et al. [26] build on this idea to make the network
unconditionally stable in areas ODE-net was not previously able to operate. However, due to
the simplicity and availability of ODE-net we have yet to experiment with these more stable
solutions. Based on our experience, our problem is not one in which ODE-net struggles for
stability.

3.2 Physically-Based Fluid Simulation

The first stable implementation of fluid simulation was presented by Jos Stam in 1999 [24].
From there, several improvements were made including more realistic smoke [9] and fire [18].
While all of these methods are physically-based and therefore produce visually accurate results
they can be quite slow. These approaches require solving several equations (typically some
form of equations 2.1 and 2.2) in a discretized space, therefore computation time increases as
the resolution of the solution space increases. Because changes at a specific timestep in a
sequence would require resimulating every frame, starting from the first frame, not just the
localized time steps, several attempts have been made to speed up these algorithms while maintaining their realistic results. One approach focuses on limiting the simulation space, allowing for more computational resources to be devoted to localized regions that require more detail.

The simplest approach to reduce the computational cost of simulation is to control the regions being simulated with high levels of detail. A simple example of when this might be applicable would be in an ocean setting with a boat interacting with the waves. In such a case, the water around the collision object needs to be simulated, but the majority of the ocean surface does not require as many computational resources. The simple approach to this would be to manually describe a bounding region, but bounding boxes or even spheres, while easy to compute, are restrictive in the boundaries they provide. A method for animated boundary regions not restricted to boxes or spheres has also been developed by Stomakhin and Selle [25]. This method does not introduce control of the motion of the fluid itself, but it does allow for a way to limit where simulation occurs. The boundary limitation ensures that only regions that need simulation are considered, and this allows for a more efficient simulation workflow.

Kim and Delaney [14] presented an alternative method for reducing the computational cost of simulation. Their method is of particular value when dealing with a simulation which undergoes a large number of iterations. They proposed to resimulate user specified areas of a motion sequence to enhance detail or slightly modify subregions of an overall effect rather than restarting the entire simulation. While this approach does not increase the solution speed of simulation, it is effective in reducing the time taken in follow up iterations. Each subregion being smaller than the original simulation space will be solved much faster than attempting to resimulate the entirety of the effect.

While these methods are effective in speeding up the process of generating visually pleasing effects, they still require resimulating the effect in order to retime a sequence of frames. Speeding up the simulation of the sequence is insufficient because restarting a simulation
with changed parameters will not guarantee the same shapes in the effect. Specifically when only the speed of the effect needs manipulating, a way to scale the time in a sequence without affecting the shapes generated is needed.

### 3.3 Machine Learning Application to Fluids

Machine learning has been explored as a way not only to speed up fluid simulation but also to provide additional control methods. The most difficult aspect of using machine learning methods to solve simulation problems is the time needed to train on large amounts of data. Obtaining data is fairly simple, as any number of simulations can rather quickly be produced, but training a model on large amounts of data takes several hours to several days. While training a machine learning model often takes a considerable amount of time, evaluating a trained model is typically very fast. As long as the training can be done as an initial step without needing much repetition, the speed of evaluation in machine learning models, allowing for very fast iteration, outweighs the drawback of long training times.

Unfortunately, sometimes the emphasis on lowering computation time reduces the visual quality of the simulation. A system that can achieve real-time results using machine learning [2] has been developed by Bonev et al., but is entirely based on surface deformations and does not take into account other physical processes at play. Because physical elements are not fully incorporated into the algorithm, the results lack realism. A neural network that attempts to recreate physics based motion should not entirely disregard the underlying equations, but rather find a way to incorporate them into the machine-learned model [6]. Wiewel et al. explored the possibility of encoding these physics properties in the latent space of a neural network [32]. A successful example of applying latent encodings to fluids was presented by Kim et al. [12]. In their case, they encoded velocity data to a latent space and trained a network to step forward in the encoded space. These encoded steps are each decoded to actual velocity values, and the resulting sequence maintains strong temporal coherency. However, while the results they generated were impressive, the model was trained
with a specific timestep and would not be able to retime a simulation. Another drawback is that latent spaces are inherently difficult to interpret, thus making this technique a less attractive approach for art direction.

Because machine learning has a fast evaluation time, many attempts have been made to use machine learning to speed up simulation problems. Tompson et al. [28] and Yang et al. [34] both developed machine learning systems which focused just on part of the process, namely the pressure projection step, which allows them to guarantee a certain adherence to the physical properties of the simulation because the remaining portions are still solved using deterministic equations. Other research has elected to replace all aspects of the simulation step with a deep network as proposed by Kim et al. and Ladicky et al. [12, 15]. The complete substitution of simulation with a neural network can drastically decrease computation time, but in many cases (such as the ones referenced above) the machine learned algorithm is restricted to the grid resolution it was trained on. The successes in applying deep learning to physical simulation problems lead us to believe it to be a valuable venue for art directing fluids; however the lack of scalability in these methods is not desirable.

There exists a set of resolution-independent machine learning solutions for fluid simulation, but these are focused primarily on the introduction of detail in a low resolution simulation. One example of this is the addition of droplet detail in liquid splashes [30]. In this paper, Um et al. trained a neural network on samples of droplet splashes, then incorporated this learned data with a fluid solver allowing the solver to preserve details that might otherwise have been lost due to slight inaccuracies in computation. One drawback of this method is that it adds complexity to an already time-intensive process, making it good for enhancing simulations but a less appealing solution for artistic control.

Many techniques have greatly reduced the time and cost of simulating fluids. Additional methods provide evidence that machine learning can be a useful tool for fluid simulation. However, they also have drawbacks. Machine learning techniques have difficulty maintaining long-term temporal coherency. The accumulation of error from frame to frame in a sequential
machine learned model leads to undesirable visual artifacts not present in physically-based methods. As with all machine learning, the possible outputs of a model are greatly constrained by the data used during training, making it difficult to create generalizable models.

3.4 Art Direction of Fluid Simulation

Art direction of fluids has been an area of interest for several years, and various methods of control have been developed. Most methods try to alter the motion behavior of the simulation directly, but often these methods introduce additional forces resulting in noticeably unnatural behavior. Beyond changing shapes and motion, modifying the timing of a simulation is also an important form of art direction.

When it comes to controlling the actual motion of fluids, several methods have been explored. One such form of direction allowed for artistically defined shapes to modify the motion of liquids [21]. However, this method deals only with particle-based simulation, and is therefore not generalizable to other fluid effects such as fire and smoke. We are seeking after a method of control that can be adapted to voxelized simulations as well. Thuerey developed a technique for controlling animation based on interpolation between several simulations [27]. Using multiple smoke simulations he was able to generate new motion results by blending values of different sequences together. While this process does a fairly decent job in maintaining natural behavior, it leads to increased computation time as several simulations are needed for the interpolation.

In all cases of fluid motion direction we want to maximize control while minimizing the impact the added control has on the natural motion of the simulation. One approach given by Treuille et al. uses key frames to define target shapes at specific points in the simulation then uses an optimization method to minimize the strength of the control parameters on the simulation [29]. The optimization step however can become rather computationally impractical as more keyframes are introduced. Pan and Monocha developed another approach which used a similar technique but removed certain constraints of the Navier Stokes equations
While this successfully reduced the resources necessary for control, the removal of these constraints can lead to inaccurate, or unnatural, motion behavior.

Other forms of key frame [16] or target shape [8] control have been developed, but these introduce new and non physically-based equations to the simulation process. An example of these artificial forces is an attraction constraint that drives the fluid toward the target shape. Due to the addition of these external forces, the visual results of these techniques can often seem forced, meaning they have less natural fluid motion overall.

Bangalore and House defined an alternative form of control by defining velocities along curves and driving the simulation with these artist defined shapes [1]. While this method allows for a high level of control, which may be desirable in highly stylized environments, the added control results in visibly unnatural motion. For example, sequences of fluid simulation that lack the curling motion typical of fluids or density values appearing out of thin air. Our goal is to develop a direction method that allows not only for control, but manages to preserve the natural behavior provided by physically based simulation.

An interesting method that provides more intuitive control is to simulate a sequence in reverse. Oborn et al. started from a target shape then ran a simulation in the time-reversed direction such that when it is played in forward time smoke seems to naturally fit to the target shape [19]. This method involved defining a novel set of equations to perform backward simulation of smoke. While this method proves promising, it introduces unnatural equation formulations in the frames leading up to the target frame to deal with problems in reverse simulation. For example, they introduced a force which attempts to reverse entropy. Because backward simulation is an ill-defined problem, with a one to many mapping space, it is difficult to determine the proper frame sequence that should be generated. However, so long as the target shapes are reached and the motion of the effect does not seem unnatural, the idea to start from the end, or at least set key shapes, may lead to useful control methods.

Other forms of direction include changes in resolution or internal shapes of flow. This idea was applied to smoke simulation by Chu and Thuerey [4]. Using machine learning, their
algorithm takes a low-resolution volume and introduces finer level detail to the simulation. While this can make the shapes of a volume more interesting, it is still not intuitively controllable, simply additive to what’s already there. A similar approach by Xie et al. [33] was also developed using deep learning. This system suffers from similar drawbacks as the previous method. Even more recently the idea of style transfer, which has been applied to images in the past, was applied by Kim et al. to smoke simulation [13]. While these algorithms are useful applications of machine learning to fluid simulation, we will focus our efforts on modifying the timescale of a sequence rather than altering details or shapes of the effect.

Direction methods that are most related to our work are attempts to retime a simulation sequence while preserving the original shapes and motion. Several methods exist which can estimate additional frames in order to slow down effects. A group at Weta Digital recently presented an image processing inspired algorithm for retiming [17]. In their process they use bidirectional advection of the velocity values in the simulation followed by blurring and unsharp masking kernels defined in three dimensions over the density values to approximate dissipation as well as its inverse. Another talk given by a group at Blue Sky Studios shared an approach based on the work of Selle et al. [23] which used smoothing over a velocity interpolation function in order to mitigate known flickering artifacts which arise when retiming emissive volumes [7]. While these approaches are both fairly fast and and visually accurate, both assume a static source and focus only on retiming of velocities. Because velocity information in a simulation does not represent the source, attempting these methods on a simulation with a moving source would lead to inaccurate results. Due to this, we turned to machine learning for a solution. To the best of our knowledge no attempt has been made to use machine learning as an approach to retiming fluid simulation.

A recent theme in graphics research has been to apply machine learning techniques to fluid simulation problems. In the case of retiming, we need an approach that is flexible and
fast to iterate on. By adapting elements from time series machine learning algorithms, we present a method that meets these criteria.

The method we propose utilizes a time-aware deep learning network to approximate frames of data at any time step in a simulation. Our work allows for very rapid iteration during a retiming process and for very flexible retiming sample rates. In addition, our method is able to account for moving source. The method we present provides a mapping of volumetric simulation data to a simple time sequence which lends itself well to solving many problems involving the art direction of fluids.
Chapter 4

Thesis Statement

A time-dependent machine learning model can provide an approach to retiming fluid simulation data which allows for more extreme retiming factors and supports non-static sources.
Chapter 5

Method

5.1 Overview

We implemented a deep learning network which provides a mapping of volumetric simulation data to a simple time sequence. In addition to creating this mapping, our method allows for the generation of data at time samples not present in the original sequence. This allows us to retime a simulation by requesting an arbitrary set of time samples. Our method is able to achieve very large retiming scale factors without introducing undesirable visual artifacts. Unlike previous methods, our model is able to account for moving sources in the original simulation. Also, because of the speed of inference of our network, our method allows for very rapid iteration in a production process.

Figure 5.1: Our method encodes the first frame of the original simulation sequence, then (given a new timescale) produces a sequence with the desired retiming. During training, the new timescale \((t_0...t_M)\) is identical to the input timescale \((t_0...t_N)\).
For a simulation to be retimed, we train our machine learning model on that simulation’s original density field data. Although a smoke simulation may involve multiple fields of information, namely density and velocity, we only use the density values in our machine learning model. The use of velocity fields may enhance the results generated by the model, but we limited our model to use only the fields required for rendering purposes (fire simulations require temperature and heat values in addition to density) in order to perform training in a manageable amount of time. In addition to allowing for slightly faster training, we found that using only density values gave comparable results to introducing velocity elements. Figure 5.1 gives an abstracted overview of our method.

![Figure 5.1](image1.png)

**Figure 5.2**: A few example frames of simulation data. Density values are discretized over a 3D grid of voxels (boundaries in light pink).

We obtain density data by running a smoke simulation for 200 frames. Figure 5.2 shows a few sample frames of density data. Density values are stored for each voxel of a discretized 3D space. Each frame of density values is normalized by the maximum value in the entire sequence and then stored in a three-dimensional array. If additional fields are used during training they are normalized in the same way. For example, if temperature values are needed, the fields of temperature values would be scaled by the maximum temperature value in the simulation. This sequence of arrays is what we use during training. The resulting sequence of density (or other fields of data) arrays produced by our algorithm are read in as geometry to a 3D software package frame by frame for rendering purposes.
Our method involves encoding the first frame of a sequence of smoke simulation data to a latent space representation. This latent space is simply a reduced version of the original data. In our case, the latent space is the result of several layers of convolution which greatly reduces the number of variables while maintaining the important spatial relationships in the simulation data.

Once encoded, the ODE solver uses the latent ODE function ("ODEFunc" in Figure 5.1) to learn the evolution through time of the sequence. Because there is no defined ODE function for the time-dependent transitions in the latent space, we approximate this function using a convolutional neural network. Rather than relying on velocity values to determine how the density evolves at each time step, we train our model to develop a latent space representation for each frame in the sequence. We make use of this time-dependent set of latent vectors to approximate how the sequence develops over time. The frame by frame latent space representations provide a smoother and more flexible way to resample time values when velocity values are too far removed from the desired time sample and interpolation methods fail to provide appropriate values. Once we have developed an approximation of the underlying frame to frame evolution of the simulation, we can use a new set of timesteps to evaluate the retiming of the original sequence. Doing so provides a new set of encoded time samples which are then decoded to actual density values.

In summary, the steps for training our method are as follows:

1. Generate a simulation sequence.
2. Encode the first frame of density data.
3. Use ODE solver to generate latent space representations for each timestep.
4. Decode latent vectors to resulting frames of density values.
5. Calculate and backpropagate loss.

Although the work presented in the original development of the ODE-net provided an example of a latent function time series prediction, their example only dealt with points in
2D space [3]. One of our contributions is the extension of the application of the ODE-net structure to a three dimensional problem through time. We also implemented a simple loss function which makes use of ideas from image processing to accentuate regions of detail in our training data. The architecture we present in this thesis is the result of research we have conducted to find an appropriate neural network which would allow for reasonable training times and maximum accuracy. This exploration of model structure included fine tuning of architectural details such as filter size and depth of layers. While we do not claim that the model we present is the optimal solution, we have found that in our experiments this particular setup provided the best results. We will focus this thesis on how to use our workflow to retime simulation, we will discuss further applications of our framework in a later section.

5.2 Encoding

The encoder we designed for our model is fairly simple and follows the general structure used for convolutional encoders. It consists of a series of convolutions which compresses the original data to a simpler representation. The reason for using convolution operations instead of some other neural network architecture is the ability to reduce the data size while maintaining important spatial relationships. Figure 5.3 gives a more detailed description of the encoding architecture that we built. Our method is not dependent on this specific encoder architecture, and its design came through experimentation with the goal of including enough convolutional layers to provide wider spatial awareness and preserve a larger latent space representation. Strided convolutions are used to reduce the size of the data further as it passes through the network. In our retiming workflow, we found that using the density values as input produces good results, so our encoder only deals with a one-channel three-dimensional volume. If other fields, such as temperature and heat (necessary for rendering purposes), were to be included in the learning process they would simply be appended to the input as individual channels.
Figure 5.3: A detailed view of the layers in our encoder network. Every orange block is a convolutional layer with the corresponding kernel size (k) and stride (s). A DownBlock is a combination of several convolutions. Each DownBlock is identical in structure (with the exception of data size and number of channels) and we have expanded one of them in the highlighted region. A hyperbolic tangent activation function is used after every convolution operation except the last one.

Only the first frame of data is encoded before being passed on to the ODE learning portion of the model.

5.3 Sequence Learning

The ODE function portion of our network is defined by a neural network of convolution operations. This neural network learns the relationship between frames of simulation data. In a more mathematical definition, the model learns the $\Delta z$, $z$ being the latent representation of a frame, at each time sample. The network also learns a function that approximates the evolution of these latent space representations through time.

ODE solvers [3] take as input the neural network representing the function $f$ it must learn to approximate, the encoding of the initial frame $z_{t_0}$, and a set of timesteps $(t_0, t_1, ..., t_m)$, for which it will generate individual frames of data. Figure 5.4 shows this in greater detail.
Figure 5.4: Overview of the ODE-net framework. The ODESolve learns a time-dependent relationship in the latent space of our simulation data, and provides the latent representations, $z$'s, of the new timesteps to be decoded to volume data.

Although the figure shows the general case in which the new timesteps can be any arbitrary set of time samples ($t_0...t_M$), the original timing ($t_0...t_N$) is used during training.

Rather than use other sequence approximating machine learning models, we chose to make use of the ODE-net framework. Our reason for doing so is to mitigate accumulation errors present in other learning models. Because we include an ODE solver in our workflow, our algorithm trains to learn the time-dependent derivative of a function, $f$, rather than directly approximating the function itself. When $f(x)$ is approximated directly, accumulated errors between timesteps lead to undesirable results. Having a time-dependent learned model means we can directly request a timestep without having to compute any of the frames leading up to it. In other words, we do not have the risk of accumulating error from previous timesteps. The time dependency of this network is also particularly helpful for building an intuitive way to retime simulations as we will discuss in more detail in a later section.

5.4 Decoding

Our decoder is similar to our encoder with the main variation between the two architectures being the use of transpose convolution operations in order to upscale the data instead
Figure 5.5: A detailed view of the layers in our decoder network. Every orange block is a convolutional layer with the corresponding kernel size \((k)\) and stride \((s)\). An UpBlock is defined as a transpose convolution operation combined with additional layers of convolution. Each UpBlock is identical in structure (with the exception of data size and number of channels) and we have expanded one of them in the highlighted region. Hyperbolic tangent activation functions are used after every layer except the last convolution operation.

of using strided convolutions. Figure 5.5 provides a more detailed view of the decoding architecture. The purpose of the decoder is to expand the latent space representations to full-sized volumetric data for each frame. Unlike the encoder which only operates on one frame of data as input, the decoder takes the entire output sequence of latent space representations provided by the ODE function as input. The result of our decoder is therefore the full predicted simulation sequence.

5.5 Training

We train the model on a 200 frame sequence of volumetric simulation data. Because of memory limitations we were constrained to use a voxel grid resolution of 32x32x32 for our simulations. While it is not necessary to use every frame as training input (random frame samples could also be used), we found that accuracy increased with the number of frames
used. Specifically we use those data fields which are relevant to rendering a simulation. For example, smoke simulations only need density values at render time, but fire or explosions would also need temperature and intensity fields to be included. If additional fields (beyond density) are required we append them as separate channels of our input data. Each field included in the training is first normalized by scaling all data values by the maximum value found in the entire sequence. Examples of fire simulations which require temperature and heat in addition to density will be shown in the results section.

The loss function we developed and use during training is given in Equation 5.1. The loss is calculated as a combination of an mean squared error between the original sequence, \(x\), and the predicted frames, \(\hat{x}\), along with a mean squared error of their gradients.

\[
\text{Loss}(x, \hat{x}) = ||x - \hat{x}||_2^2 + \lambda ||\nabla x - \nabla \hat{x}||_2^2
\]  

(5.1)

Figure 5.6: Definition of the three-dimensional Sobel kernel aligned to the direction of the z-axis.

The reason for including gradient errors in addition to directly comparing the produced sequence to the original data is based on principles of edge detection in image processing. Edges in images can be detected by looking at regions of high gradient magnitude. Similarly, areas of fine detail in three-dimensional volumetric data correspond to areas of high gradient magnitude. We employ this idea in order to preserve areas of high detail in the volume during training. A common approach to edge detection is the use of Sobel kernels, so we use
the three-dimensional definition of these kernels to calculate the gradient magnitudes of our data. An example of the Sobel kernel defined in the z direction is illustrated in Figure 5.6. The kernel is simply rotated to align with the other axes for definitions along the x and y direction. We normalize the magnitudes to avoid overly large numbers and to simplify our masking function.

The model takes in a series of frames of data with their corresponding timestamps and learns to reproduce the full sequence as described above. While it is not necessary to use every frame of data available for training, we found that in practice the model converges faster when as many of the original sequence frames as possible are used.

5.6 Retiming

![Diagram of retiming](image)

Figure 5.7: Example of retiming. Given an initial frame, \(x(t_0)\), and a set of time samples, \((t_0, t_1, t_2, ..., t_M)\), our method produces resulting frames of data for each timestep.

The full process of retiming a sequence involves training the model using all the frames of simulated volumetric data. When the model has learned an approximation to the original sequence, retiming becomes a very intuitive and simple process. In order to produce a sequence matching the shapes of the original with a different timescale we provide a new set of time samples, \((t_0, t_1, t_2, ..., t_M)\), to the model. Evaluating the model with the new timesteps as inputs will generate a new set of frames with the desired retiming. Figure 5.7 shows an abstracted explanation of this idea.
A simple example of selecting appropriate $t$ values for retiming would be to slow down a sequence to half speed. Producing a half speed slow motion effect is achieved by lengthening the sequence to be twice as long, or in other words, we need twice as many time samples for the sequence. Assuming our sequence had data values for 100 frames, we would want to lengthen the sequence to take up 200 frames. With our workflow we simply evaluate the retiming model with half steps ($t = 0.5, 1.0, 1.5, ..., 100.0$) resulting in a 200 frame retimed sequence of data.
Chapter 6

Results and Validation

In order to validate our findings we show side by side comparisons of original sequences with their retimed counterparts. Figure 6.1 shows several frames of an original sequence compared against the matching frames of the reconstructed output as well as examples of retimed results. Along with visually presenting them side by side, we also provide numerical comparisons which are computed as an average absolute error between all frames of input
and output. Table 6.1 lists the error values for several examples. Along with the numerical errors we include the average training time per epoch (the model was trained for 5000 epochs on each dataset) and the approximate evaluation time for each data sequence.

<table>
<thead>
<tr>
<th>Data Sequence</th>
<th>Error</th>
<th>Training Time</th>
<th>Evaluation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Source (Figure 6.1)</td>
<td>0.003</td>
<td>23s</td>
<td>3s</td>
</tr>
<tr>
<td>Wispy Smoke (Figure 6.2)</td>
<td>0.005</td>
<td>25s</td>
<td>5s</td>
</tr>
<tr>
<td>Billowy Smoke (Figure 6.3)</td>
<td>0.005</td>
<td>25s</td>
<td>5s</td>
</tr>
<tr>
<td>Fire (Figure 6.4)</td>
<td>0.012</td>
<td>45s</td>
<td>7s</td>
</tr>
<tr>
<td>Explosion (Figure 6.5)</td>
<td>0.006</td>
<td>45s</td>
<td>7s</td>
</tr>
</tbody>
</table>

Table 6.1: Average absolute error values along with training and evaluation times for all data sets. Training times are the average time per epoch. Evaluation times are an average for an inference of 200 frame values.

Figure 6.2: An example of retiming a simulation with a moving source. Corresponding frames in the original and the retimed result are highlighted.

The method we show allows for smooth approximations for very large retiming factors. This helps illustrate that the model develops a very close estimate of the original sequence and is able to interpolate smoothly between frames. Our model is not restricted to linear retiming and allows the user to arbitrarily manipulate a time sequence to produce a non-uniform retiming. Figure 6.2 shows the retiming of a smoke simulation with a moving source. An example of non-uniform retiming is given in Figure 6.3.

We also see that the interpolation provided by our model does not introduce undesirable artifacts, such as jittering intensity values in emissive volumes. Figure 6.4 and Figure 6.5
show some of our results when training on emissive volumes. We show a retiming factor of 3 on the explosion example to illustrate that our method is not restricted to retiming values which are equidistant from existing frames in the original sequence.
Figure 6.5: An example of a retimed explosion. From top to bottom: original simulation sequence, original timing reconstruction, slowed down by a factor of 3. Corresponding frames are highlighted.
Chapter 7

Discussion

7.1 Contribution

In this paper we have described an approach to retiming fluid simulation that is more flexible than currently available systems. As far as we are aware, this is the first attempt to apply machine learning to the problem of retiming. Although training the machine learning model can take a considerable amount of time, the advantage of machine learning methods is the speed of evaluation once trained. This allows an artist to evaluate various retiming values in mere seconds until the desired timescale is reached. To the best of our knowledge, this is also one of the only methods that allows retiming on simulations with moving sources.

Our method provides more flexibility than previous methods. During training, we are able to use sparser sampling of the fluid simulation. Once trained, our model also handles much larger retiming scale factors while maintaining temporal coherency and avoiding visual artifacts. Of course, these advantages remain within the bounds of the machine learning model’s ability to approximate the sequential evolution of the fluid simulation.

While our algorithm provides a very natural way to perform retiming, the contribution of this work is not limited to this form of art direction. Our framework is simply a way to train a machine learning model which approximates a sequence of simulation data. That model can then be manipulated in a number of ways to allow artistic control. We discuss this further in the Future Works section below.
7.2 Limitations

One of the biggest drawbacks to our method is the need to train a machine learning model. This training brings with it several difficulties not present in other retiming approaches. Training is the most time consuming step of our method, and the fact that the model needs to train for each new simulation sequence makes for a non-generalized solution. As of yet, our model does not generalize across simulations with differing parameters. The temporal and spatial dimensions of our data are also constraints on the model itself. The larger the input data, the longer training will take, which in production is less than ideal.

7.3 Future Work

An additional application of our method we want to explore would be using target shapes to drive a simulation. Consider an existing simulation for which modifying the shapes at specific times in the sequence would be necessary. Rather than resimulating the sequence in its entirety one could use our framework to insert the desired shapes at specific timesteps

Figure 7.1: Example of inserting target frames in existing simulation.
during the training process and use some form of optimization to vary how strictly the resulting frames must adhere to either the directed shapes or the original simulation. A simplistic visualization of such art direction is given in Figure 7.1. This is of course only a theoretical application and is the subject of future work.

Other future work is inspired by the limitations we outlined above. The generalization of our model to handle new simulation sequences without a need to train anew is of particular interest. This generalizability extends also to the size constraint on our current model. We would like to develop a system that is size independent.

What we have presented is a previously unexplored application of machine learning to art direction in fluid simulation. While the work in this paper is not without drawbacks, it opens up a field of additional research for manipulations of time dependent simulation sequences.
References


