Final Report

FYP Number: fyp23001

Project Title: Physical-based Character Animation in Fluids

Group Member:
Sinan Wang (3035770599)
Zebin Guo (3035770915)

Supervisor:
Professor Taku Komura

ABSTRACT

This report presents the progress and developments made in our final year project, which focuses on training the world model in physical simulation using neural networks. The project comprises two main components: physical simulation through a neural network-based world model for fluid dynamics simulation, and character animation within these simulated fluids to achieve realistic animations. A long-term goal of the project is to extend the control policies of robots from virtual environments to real-world physical domains.

Since our previous interim report, notable progress has been achieved in our research. We have successfully identified the supervised approach as the preferred method for constructing the world model. While an alternative approach involving a differentiable fluid solver holds promise for greater accuracy in the output, significant research gaps persist if we intend to adopt the two-way coupling divergence-free stream function solver. In contrast, the neural network approach offers distinct advantages by reducing the reliance on extensive simulation knowledge. The neural network handling the timestep minimizes the need for specialized simulation expertise. Further, we adopt two datasets generated by a self-developed fluid solver into this model, investigating the outputs, corresponding influencing factors, and how the change of hyperparameters and loss function will lead to a change in reconstructed output.

By leveraging neural networks and incorporating cutting-edge research in differentiable fluid simulation and self-supervised learning, our project seeks to enhance character animation's realism and control capabilities within simulated fluid environments. The findings and insights presented in this report contribute to the ongoing progress in physical simulation and character animation, opening new avenues for future research and development.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to my supervisor, Taku Komura, for his invaluable guidance and support to our final year report. His expertise and mentorship have been instrumental in shaping the outcome of this project. Thank you for your unwavering dedication and belief in our abilities.

Table of Contents

List of Figures

1. Introduction

With the advance of the Metaverse, physical simulation and character animation are becoming increasingly popular. Many prior works associated with fluid simulation have diligently sought to emulate the physical rules in virtual environments. Noteworthy among these are endeavors employing fluid solvers, such as utilizing the stream function (Ando, 2015) to mimic fluid behavior, guaranteeing that the fluid is divergence-free with no numerical errors. Despite the precision and comprehension of physical laws in these simulations, solving partial differential equations iteratively within physical solvers has revealed inherent drawbacks. For example, numerical approaches to solving PDEs may introduce unneglectable numerical errors. Some may be accumulated in iteration, leading to incorrect results. Further, numerical methods discard the continuity and differentiability of original equations. It prohibits non-differentiable fluid solvers from extendable works, like adopting the fluid solver to train agents using a deep learning approach, as no gradient information can be used for backpropagation and initial posture optimization.

Fortunately, the emergence of the world model paradigm has opened up promising avenues in physical simulation. This paradigm represents a novel approach that integrates neural networks to understand virtual environments, as demonstrated by notable achievements such as PlaNet (Hafner, 2019) and DayDreamer (Wu, 2023) conducted by Google Research. However, it is essential to note that these accomplishments have primarily focused on training control policies within simplified environments such as planes and vacuum, which are not concerned with other real-world conditions such as fluids. Hence, we plan to research the feasibility of developing the world model using deep learning approach in the context of fluids.

Motivated by the concept of the world model, this project shifts its focus toward the realization of dynamic robot simulations within complex fluid environments. Specifically, this environment revolves around the concept of vorticity, enhancing the authenticity of the fluid simulation. Consequently, the project, titled "Physical-Based Character Animation in Fluids," aims to develop a comprehensive world model for fluids and leverage it to design a control policy for rigid-body agents. The objective is to demonstrate the effectiveness of differentiable world models and their potential to replace traditional fluid simulation techniques.

The project is planned to progress in two stages. Firstly, a differentiable world model will be implemented using neural networks. This model takes the previous states and actions as input and predicts the current state. Furthermore, the model retains hidden states, enabling the preservation of information from states preceding the previous one. Secondly, a control policy will be formulated for the robots, which accepts the current and hidden states as input and generates corresponding actions. These actions will be iteratively fed into the world model to generate future movement sequences. An additional reward function based on states will be manually defined as the optimization objective. Consequently, the deliverables of this project will include a specialized world model and a tailored control policy specifically designed for fluid environments based on vorticity. These outcomes will contribute to advancing the understanding and application of differentiable world models in the context of fluid simulations. Due to time limitations, we can only work out the world model, leaving the control policy as future work.

The subsequent sections of this report are organized as follows. Firstly, the remaining portion of **Section 1** will introduce foundational concepts such as supervised learning and physical-based animation. It will be followed by a discussion of the study's objectives and deliverables. Subsequently, **Section 2** will present an initial outline of the training process for the world model in fluid environments, accompanied by an operational framework for training the control policy. The final architecture of the world model will be examined as a solution to the encountered data scale problem, along with a detailed explanation of the rationale behind certain choices made in this project. In **Section 3**, the issue of the original results will be addressed, with a comprehensive explanation of the experiments conducted and the world model employed provided. Furthermore, **Section 4** will present a set of overarching conclusions derived from the current methodology. In **Section 5**, future work will be elaborated.

1.1. Background

1.1.1. Physical Simulation

Physical simulation involves using computers to model the real physical world. It originated from applied mathematics and physics and has expanded to include the simulation of fluids and their interactions with solids. This project focuses on fluid simulation and addresses the challenges associated with solid-fluid coupling.

Fluid simulation is a well-researched field that draws interest from mathematicians, physicists, and computer scientists. At its core are the Navier-Stokes equations, which describe fluid flow through partial differential equations. Solving these equations computationally enables the simulation of fluid behavior in virtual environments.

One critical aspect of fluid simulation is accurately representing the interactions between fluids and solid objects. This interaction requires addressing the solid boundary conditions and considering various phenomena such as viscosity, turbulence, and pressure forces. Achieving realistic simulation results necessitates the development of sophisticated algorithms and numerical methods capable of capturing these complex interactions. The accurate simulation of solid-fluid coupling poses challenges in accurately modeling the behavior of both fluids and solid objects. Researchers must develop efficient computational techniques to handle the computational demands of the coupled system.

1.1.2. Character Animation

Character animation is a fascinating and specialized field within computer graphics that focuses on breathing life into virtual characters through movement. It combines artistic creativity with technological tools to create the illusion of thought, emotion, and personality in digitally rendered characters. Over the years, character animation has undergone significant transformations, thanks to technological advancements.

In computer graphics, character animation employs various methods to achieve realistic and engaging performances. Keyframe animation, a traditional approach, involves manually setting specific frames where the character assumes distinct poses. This technique gives animators precise control over the character's movements, expressions, and gestures. On the other hand, motion capture has revolutionized character animation by capturing the movements of real actors and transferring them onto digital characters. This approach adds an extra layer of authenticity and believability to the animation, as it incorporates the nuances of human motion. Furthermore, procedural techniques, such as simulations for cloth or hair dynamics, contribute to the realism of characters, especially when they interact with the virtual environment.

Character animation in computer graphics goes beyond mere movement. It encompasses creating characters that can believably interact with their virtual surroundings and other characters, which requires a deep understanding of anatomy, physics, and storytelling. Animators must consider how characters' bodies and limbs move naturally, how they respond to external forces, and how they emote and express themselves. By weaving these elements together, animators can breathe life into characters and make them relatable and engaging to audiences.

Character animation is a multidisciplinary endeavor that draws upon various fields of expertise. Artists and animators collaborate with technical experts to create characters that move realistically and possess depth and personality. They must have a solid grasp of anatomy, studying how muscles, joints, and skeletal structures work together to produce fluid motion. Additionally, they delve into the principles of physics to ensure that characters interact with their environment in a physically plausible manner.

The ultimate aim of character animation is to create virtual characters that resonate with audiences. Whether in video games, films, or virtual reality experiences, animated characters play a crucial role in storytelling and immersing viewers in the digital medium. When done skillfully, character animation can evoke emotions, convey narratives, and forge connections between the audience and the virtual world.

1.2. Motivation

A significant portion of character animation research predominantly focuses on the interaction with solids, with relatively few forays into the dynamics involving fluids. Yet, fluids—from the air we breathe to the water we interact with—play an indispensable role in our everyday lives. This discrepancy highlights a crucial area of potential exploration and innovation in character animation. The integration of machine learning techniques, as exemplified by implementing reinforcement learning in projects like DeepMimic (Peng, 2018), demonstrates promising advancements in this field. Furthermore, the progressive developments in fluid simulation within computer graphics, coupled with the emergence of world models that try to simulate the world by neural networks, present a compelling foundation for our project. These advancements not only signify the maturing of the field but also provide a starting point. Further, the

differentiability in fluid simulation is another heating topic, targeting to extend the built virtual environment to various tasks, like training robots to move in the simulated environment. Therefore, we investigate building the "world model", a differentiable neural network model capable of predicting the next frame's state given current states and previous information.

1.3. Objectives

The project aims to advance fluid simulation and character animation within fluid environments through four key objectives. The first objective focuses on developing an efficient and robust fluid solver capable of accurately representing fluid behavior and handling fluid-solid interactions. The second objective involves training a differentiable world model to simulate the fluid environment and predict its behavior using neural networks and machine learning techniques. The third objective is to train control policies for characters within fluid environments, enabling physically realistic animation and interactions. The fourth objective explores transferring virtual control techniques to real-world applications, contributing to robotics, automation, and physical character animation advancements.

1.4. Deliverables

Our final deliverables include a comprehensive program that generates physically realistic character animations within a virtual environment. The program takes inputs such as a virtual character, a virtual environment, and commands to guide the character's movements. It utilizes the trained differentiable world model to predict future states and ensure realistic interactions with the fluid environment. If feasible, we aim to produce a technical paper documenting our methodologies and findings. However, the extensive work is impossible within the one-year final-year project session. Due to time limitations, this project can now provide a trained world model capable of predicting future states as the result of final year project. The remaining deliverables, such as optimizing the world model and training control policy, are left as future work.

1.5. Literature Review

This section mainly focuses on the literature review for the physical simulation aspect.

Solving Navier-Stokes (NS) equations for fluid simulation in computer graphics gained popularity, starting with the works of Foster and Metaxas (1997) and the introduction of 'Stable Fluids' by Stam (1999). These methods predominantly employed pressure projection to ensure the velocity vector field's solenoidality, meaning the field remains divergence-free. However, it was observed that these methods often struggled to maintain the persistence of fluid vortices (which would dissipate quickly) and sometimes failed to enforce the divergence-free condition thoroughly. In response, Elcott et al. (2007) suggested using the vorticity formulation of the NS equations, with vorticity as the primary variable for fluid simulation. Subsequently, numerous variants of vorticity-based formulations emerged, such as those by Angelidis and Neyret (2005), based on Lagrangian vortex filaments, Gamito et al. (1995), focusing on vortex particles, and Pfaff et al. (2012), centered on vortex sheets. Ando (2015) also made significant contributions by utilizing the stream function to simulate multi-phase flows, incorporating solid-fluid coupling solutions.

The methods previously mentioned are both accurate and physically correct. However, their limitations lie in speed and, crucially, the lack of differentiability, which is a crucial requirement for training agents in the character animation segment of our project. Addressing this challenge, Ha (2018) introduced the concept of the 'world model,' an innovative approach that leverages neural networks to simulate the real world, which is both rapid and differentiable. As artificial intelligence and machine learning grow, similar groundbreaking works have emerged, including projects like DayDreamer (Wu, 2023).

2. Methodology

This section elaborates on our approach and is divided into four distinct steps. The initial step involves constructing the fluid solver. Following this, we deploy robots to interact with the solver, gathering data over time to create a comprehensive dataset. The third step is utilizing this dataset to train our world model. Because of the performance limitation of the supervised learning approach, as discussed in **Section 2.3**, another self-supervised learning approach utilizing the differentiable feature of the Navier-Strokes Equation will be discussed and regarded as future work to train the world model. Ultimately, the trained world model trains our control policy for controlling the characters. The first three steps belong to the physical

simulation part, while the last step is the character animation part. These four steps are combined and presented in Figure 1.

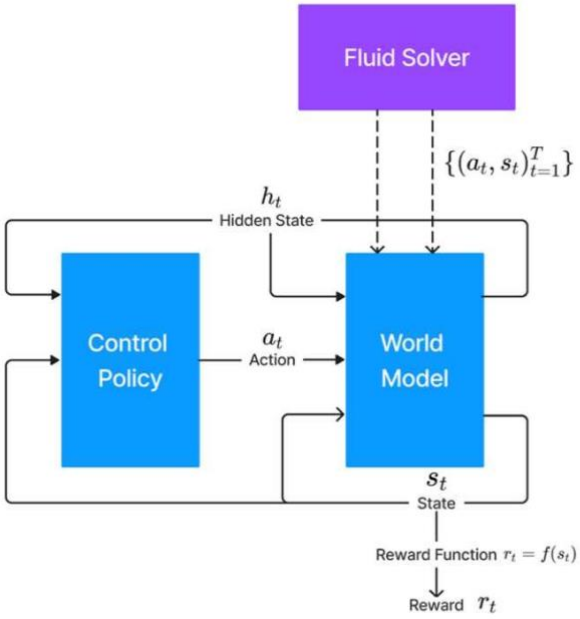


Figure 1. The framework of this project. The fluid solver provides action-state pairs for the world model. The world model then uses the data to learn a general transition model $p(s_t|s_{t-1}, a_{t-1})$. Then, with a learned world model, a control policy can be trained using the traits of neural network backpropagation from the user-defined energy function.

2.1. Vorticity-based Fluid Solver

In this section, a brief introduction about a general fluid solver will be first given, followed by the details of our method. A general fluid solver refers to a numerical solver that is able to compute the velocity at every point in the fluid domain for the next time step, given the current velocities at every point. Unlike how they are defined continuously in the physics, these velocities usually become discrete in the context of computational methods, either on a grid in an Eulerian framew ork or carried by particles in a Lagrangian framework. Here Eulerian and Lagrangian are two different perspectives of how can we describe, measure or view the fluids. Eulerian methods refer to those methods that measure the fluid quantities at fixed points inside domain, while Lagrangian methods refer to those measure the fluid quantities at points moving alongside with the fluids. Since we are using an Eulerian framework in our project, details of the Lagrangian view will not be given here. In the Eulerian framework, velocities or other quantities are stored on a grid, where the grid may be a regular grid which stores the physical quantities at the center of every cell, or a MAC grid where physical quantities from different

directions (x, y or z components) may be stored at different positions on every cell. Then, in order to represent the continuous velocity field at every point inside the fluid domain, interpolation is then used.

Having discussed the discrete storage, we then choose a physical quantity as the primary simulation variable. It is usually the velocity field itself, but it can also be the vorticity or recently, the impulse. When the vorticity is used as the primary simulation variable, the method is called the vortex method. Similarly, if the impulse is used, it is called the impulse method. For the general method when the velocity is used as the primary simulation variable.

A classic Eulerian method is composed of two parts: Advection and Projection. In the advection step, we use the current velocity field to advent the current velocity field to obtain the velocity field at the next time step. A commonly seen advection scheme is the Semi-Lagarangian Advection, where for each position where we want to know the velocity at the next time step, we do a time integration, and trace back to its current position using the current velocity field. That is, given a grid position A where we want to know the velocity at the next time step, we need to find its current position B. That means if a particle is at B now, it will move to A at the next step. Then the velocity at A at the next time step will be the same as the velocity at B at the current step. Then we only need to compute the current velocity at position B, which is computed by interpolation because B is usually not exactly on the position where we store the velocities. This process is done for every grid position that we store the velocities and is called Advection. After the advection, we need to do a projection step. The reason is that an incompressible fluid must have a divergence-free velocity field, while the velocity field after the advection step is usually not divergence-free. In this step, a variable called “pressure” is introduced. Here the pressure is not the exact real pressure, but is only used for projection. What we want to achieve here is that after the subtraction of the gradient of the pressure, the velocity field becomes divergence-free. Then, a linear system can be established by this relationship to solve for the pressure. Note that from the description, it can be seen that the pressure is a scalar field, instead of a vector field. Here some well-known methods like Conjugate Gradient (CG) method, or Gauss-Seidle (GS) methods may be applied to solve for the system, equipped with some pre-conditioners like multi-grid preconditioners for faster convergence of the system. After solving for the pressure, we subtract the velocity from the gradient of the pressure and obtain the divergence-free velocity field at the next time step. Then we again use this velocity

field to advect itself for the next time step, and do projection again. By repeatedly doing these two processes, we can obtain all the velocity fields at different time steps after the initial time and the simulation is finished when it reaches the maximum time that we set. Above is a brief description of how a general fluid solver works and then we will explain the details of our solver.

For discrete storage, we adopt the well-known discrete storage scheme (MAC grid) which stores all the horizontal velocities on the center of the vertical edges of the grids and stores all the vertical velocities on the center of the horizontal edges of the grids for the 2D case. A vortex method is used for our fluid solver, for better vortex structure preserving. That is we choose the vorticity as the primary simulation quantity. In the advection step, we use the current velocity field to advect the vorticity field to obtain the vorticity field at the next time step, and then we solve the velocity field at the next time step from the vorticity field at the next time step. Then the only problem left is that how can we reconstruct the velocity field from the vorticity field. The vorticity is defined as the curl of the velocity. Therefore, the problem is equivalent to find a velocity field whose curl is the "closest" to the known vorticity field. "Closest" is defined in the sense of L2 norm. Note that if we only impose this condition, the reconstructed velocity field is not unique because we may add any constant vector field to reconstructed result and the final result is still the same. Therefore, we needs to impose another condition, that is the velocity field is divergence free. Equipped with this condition, we are able to construct a well-posed Poisson equation: The Lagrangian of the velocity if the negative of the curl of the vorticity. This Poisson can be solved similarly as we mentioned before, by a conjugate gradient method with a multigrid preconditioner.

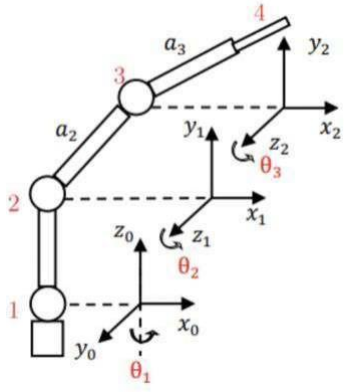
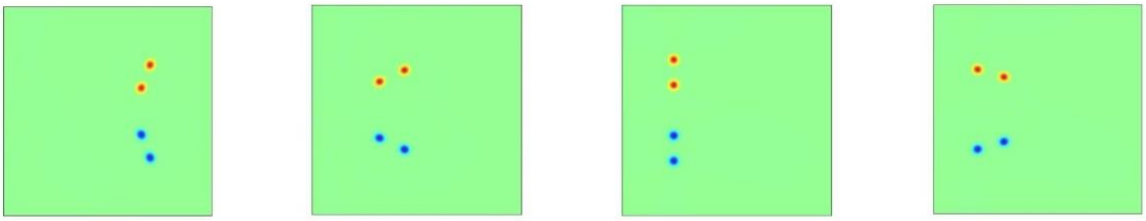


Figure 2. Example of joint representation of robots. The position and orientation represent each joint. For each joint, the orientation is represented by the respective angles in 3 domains of freedoms. (Zhou, 2023)

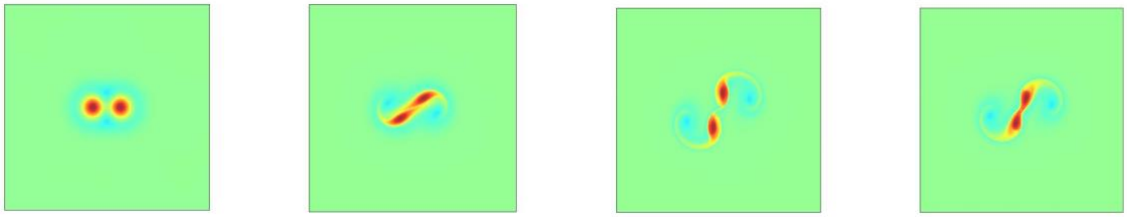
2.2. Dataset Creation

After the fluid solver is built, we use our solver to do fluid simulation for different initial velocity field and store the velocity field at every time step. In this project, we plan to use several different datasets listed as follows: smoke ejection (from the original paper), 2D leapfrogging vortices, 2D Taylor vortices, 2D Karmen vortex streets, and 3D leapfrogging vortices. Up to now, only the first two datasets are used for training our world model, but we plan to include more in the future.

The provided dataset by the original paper Deep Fluids (Kim, 2019) utilize the density of the smoke to visualize the fuilds. In 2D space, a source of the smoke is moving and smoke is ejected from the source at every time step. The smoke source position is used for transitional parameters.

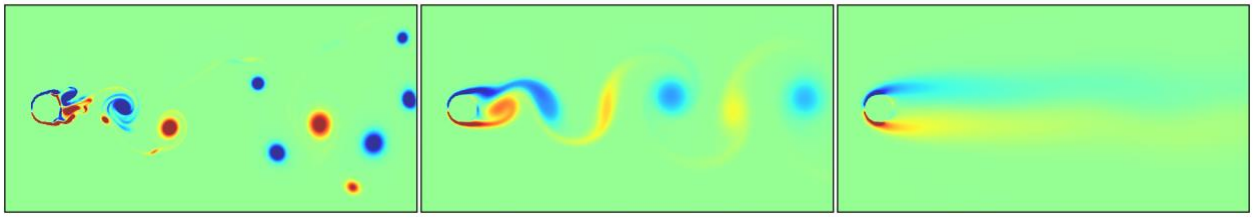


For the example of 2D leapfrogging vortices in a box (above is four key frames of the example), two pairs of vortices of the same magnitude of vorticity but with reverse signs (one is positive and the other is negative) are released from the left side of the box and they start to rotate and move to the right side of box. Upon getting close to the right side of the box, two pairs of the vortices separate and rotate back to the left side along the top and bottom sides of the box respectively. This process is repeated around 3 times. Note that for our own dataset, we did not introduce additional transitional parameter.



For the example of 2D Taylor vortex (above is four key frames of the example), one pair of vortices of the same magnitude of vorticity and the signs is released at the center of the fluid

domain. After the release, the right vortex moves up and the left vortex moves down. At some time step, they separate and become two individual vortices.



For the example of 2D Karmen vortex street (above is an illustration of the karman vortex street effect using three different viscosity, the left is the smallest viscosity), it is an example of solid-fluid coupling. Since finally we want the world model to be able to handle solid-fluid coupling cases, it is necessary to include some solid-fluid coupling examples in the training set. In this example, we put a disc at the left side of the fluid domain, and set an inflow from the left side. Once the inflow reaches the disc, vortices start to shed off fom the disc. For different viscosities, we observe different behaviors and patterns of vortex shedding. When the viscosity is small, the vortices tend to shed randomly. When the viscosity increases, the vortices start to become follow a fixed pattern and vortices with different signs appear in turn. Finally, when the viscosity increases to a certain value, there will be no vorticities appearing. This phenomenon conforms to the real world situation very well.

For the 3D leapfrogging vortices, it is the 3D extension of the 2D case. The only difference is that in 3D, the vortices become vortex rings and the original rotation movement becomes jumping movements of two vortex rings.

2.3. Large Dataset Size Issue in World Model Training

The size of a standard velocity field dataset is determined by three factors. The first factor is S, which denotes the number of examples (leapfrog is one such example); The second is F, which denotes the number of frames inside each example; The third is D, which denotes the dimensionality of each frame. It is important to note that each frame is accompanied by a transitional parameter list, which is implemented as a queue. The size of this parameter list is $1 * F$, indicating that each frame's parameter, treated as a scalar value, is sequentially

appended to the queue, starting from the tail.

Despite the seemingly flawless nature of training the world model through supervised learning in the overall stream task, one major obstacle arises: the dataset size for the supervised learning approach. As discussed above, it is necessary to store the state of fluid particles and transitional parameters in the dataset. However, as we progress to the actual experimentation phase, the issue of dataset size becomes apparent and poses significant challenges in neural network optimization.

For instance, let's examine a 3D example with a resolution of $256 * 256 * 256$. In each frame, the three components (x, y and z) of the velocity of every pixel needs to be stored. In the standard configuration, we have 10 examples, and in average each example has around 2000 frames. By a simple calculation, suppose we use float number to store the velocities, then we will have to store in total $10 * 2000 * 256 * 256 * 256 * 3 * 4 = 15GB$ Consequently, the dataset becomes massive, requiring substantial storage capacity and computational resources to handle it effectively.

To exacerbate matters, when this voluminous dataset is directly fed into a 10-layer neural network for optimization, with the number of neurons per hidden layer being 10 times the input size of the vector field, an astonishing number of over 6.5 billion hidden variables are created. This overwhelming amount of hidden variables surpasses the computational limitations of personal computing resources, making it impractical to process and train the network efficiently.

The implications of such a predicament are two-fold. Firstly, the computational burden imposed by the sheer magnitude of over 6.5 billion hidden variables can strain personal computing resources. This can result in sluggish performance, extended computation times, and even system instability, hindering the training process and impeding further analysis. Secondly, the excessive number of hidden variables engenders a heightened risk of overfitting. Overfitting occurs when a model becomes excessively specialized to the training data, leading to reduced generalization ability on unseen data. With such an enormous number of hidden

variables, the model becomes highly sensitive to noise and minor variations present in the training dataset, compromising its ability to make accurate predictions on new data instances.

2.4. Ultimate World Model Training with Autoencoder

Our solution to train the neural network with a large dataset size is to compress the data using the auto-encoder method. The idea from Deep Fluids (Kim et al., 2019) shows that using an auto-encoder can compress the state of fluids like smoke to a latent vector space. The representation of the auto-encoder is shown in Figure 4 below. The state of smoke can be well compressed using a loss function that combines the unsupervised and supervised parts. The unsupervised part z is regarded as the encoded result of the velocity field, and the supervised part p is the encoded result of the transitional parameter. As our project simulates water, another kind of fluid, this idea is apt for our work.

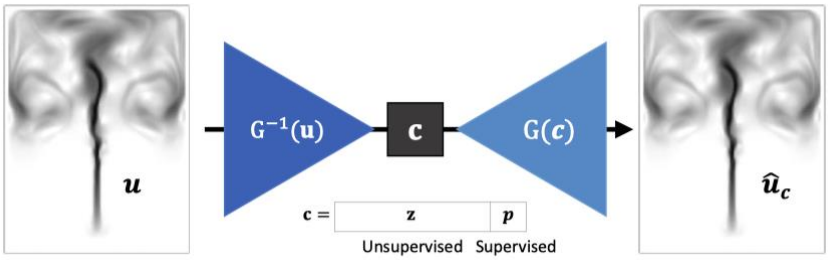


Figure 2. Representation of auto-encoder. The decoder is used to examine the effect of information preservation by reconstructing the original smoke from the compressed data. (Kim et al., 2019)

More concretely, the pipeline of training the world model can be found in Figure 3 below. The input velocity field has dimension $H * W * D$, where H is the height, W is the width, D is the depth with 1 for $2D$. In addition to the velocity field, transition parameters queue together to make the dataset. The model has one autoencoder with an encoder and a decoder, and a multi-layer neural network.

The encoder consists of multiple convolutional layers using the same padding. These layers are connected with skip connections through concatenation. The number of layers in the encoder is determined by the logarithm base 2 of the input size scale in arithmetic terms. The output of the last convolutional layer is then connected to a fully connected layer, which maps the output to the size of the latent code.

The latent code comprises a supervised part (c) and an unsupervised part (p). The unsupervised part is the same size as the input data's transitional parameter. In simulation, the latent code of the current timestep is fed into a neural network, which consists of three fully connected layers connected by batch normalization and dropout layers. The output of this neural network is the transitional difference, where the latent code of the next frame can be easily computed by adding the output to the latent code of the previous frame. Following this process, we can predict a sequence of latent codes of any length given an initial code. To improve prediction accuracy, we replace the predicted unsupervised segment with the ground truth unsupervised segment from the corresponding timestep. After predicting the sequence of latent codes, a decoder reconstructs the velocity fields using these codes, with each convolutional layer output being upscaled by a factor of 2. Further, we employ an upscaling technique with interpolation to prevent the "chessboard effect" in reconstructed images by introducing smoothness in the upsampled feature map. This upscaling mitigates the visual artifact known as the chessboard effect, as described by Odena et al. (2016).

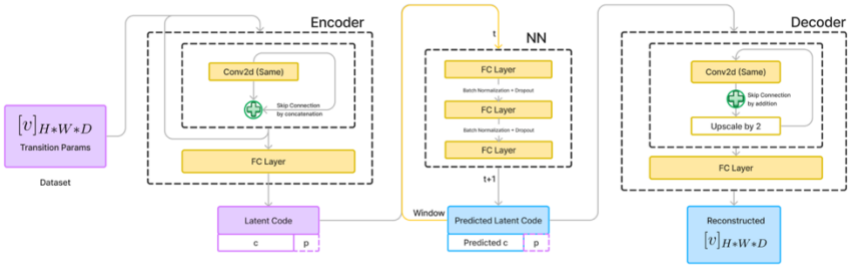


Figure 3. The architecture of world model. In training, we first train the encoder and decoder together as "Autoencoder" and use latent codes generated by the encoder as "ground truth" in training the neural network.

During the training process, we primarily use the loss function proposed in the Deep Fluids paper by Kim (2019) to guide our optimization efforts. Additionally, we introduce a window of sequences within the loss function optimization. In the loss function, we calculate the gradient of certain sequences of latent codes, instead of just one latent code. In this manner, future trends can be included, facilitating the neural network to envision further future states. Furthermore, we define a novel loss function specifically tailored for training the neural network, which is extensively discussed in Section 3.2 of this report.

2.5. Control Policy Training

In this section, we propose a method to train the agents, stemming from the idea from Li et al. (2023). As in the simulation process in the fluid solver, the fluid state is updated iteratively, with the process of fluid advection, pressure projection, and coupling of fluid and rigid particles. As the stepping process can be viewed as the transmission of a neural network, we can carefully define an energy function that serves as the objective function to minimize and calculate the gradient of variables in every simulation process. Compared to generating thousands of sequences of samples to generate a dataset for supervised learning, we only require producing several samples of frames as "ground truth". After the initial states of fluid and rigid particles are set, they are simulated to the target frames and generate the output using a predefined energy function, which usually constitutes the displacement and velocity difference of simulated particles and target particles. The difference is then backpropagated to optimize particles' trajectory and initial state.

In this approach, the problem can be formulated as an optimization problem involving design variables, namely the initial states of both agents and fluids. We randomly select the initial states of rigid and fluids and subject them to simulation using the vorticity method for multiple steps. Following the predefined simulation duration, the obtained simulation result is compared to the ground truth using a specific energy function. Subsequently, the partial derivatives for the initial states, denoted as $\frac{\mathrm{d}s^{n+1}}{\mathrm{d}s^{0}}$, are calculated using the chain rule. The backpropagation technique can be employed to optimize the initial states. The differentiation procedures of PDEs should be carefully designed and require experiments to check if potential problems exist, like gradient explosion, as mentioned in DiffFR (Li et al., 2023).

3. Experiment and Results

3.1. Experiment Settings

Our work builds upon Kim's implementation of the Deep Fluids paper (2019). The original code employed TensorFlow 1.15 for training the world model, while the Manta library facilitated data generation. We upgraded the training code to TensorFlow 2 to enhance efficiency and ensure compatibility. For the coding platform, we utilized Ubuntu 20.04 on the Windows Subsystem for Linux 2 (WSL2). The computational resources employed for the world model training consisted of

an RTX4080 Laptop GPU and an i9-13980 HX CPU. These choices were made to optimize the training process and leverage the computational capabilities of the available resources.

3.2. Issues of Original Implementation

In the original implementation, the methodology leading to the example results shown in the paper does not adopt its main idea. Instead, it discards the involvement of autoencoder, and the training of neural network creates billions of weights and biases for optimization, requiring more than 72 hours for 2000 frames using RTX 4080 Laptop GPU, which matches the inference of data size problem in neural network training in **Section 2.3.** Although the result is better, this method is not suggestable, as the need for computation is extensive for personal computing power.

Furthermore, a coding error was identified when executing the original command in the running script provided by the author. The observed outcome, depicted in Figure 4, exhibits peculiar blocks in the reconstructed result, indicative of repetitive and abrupt fluctuations in the norm of entries in reconstructed velocity fields. Moreover, the ground truth representation on the right predominantly appears as a homogeneous gray region, lacking discernible visual patterns. The underlying cause of strange fluctuations lies in the improper configuration of the supervisor, wherein the original code wrongly set up the model saver for restoring the trained model from the checkpoint. Consequently, the decoder utilized for velocity field reconstruction consistently operates with initial weight initialization, leading to meaningless results. The grey image of the ground truth result is caused by a falsely chosen scaling function, which maps the norm of velocity entries to the corresponding pixel value. The result is corrected by dividing the original range by 100 and re-did the scaling process.

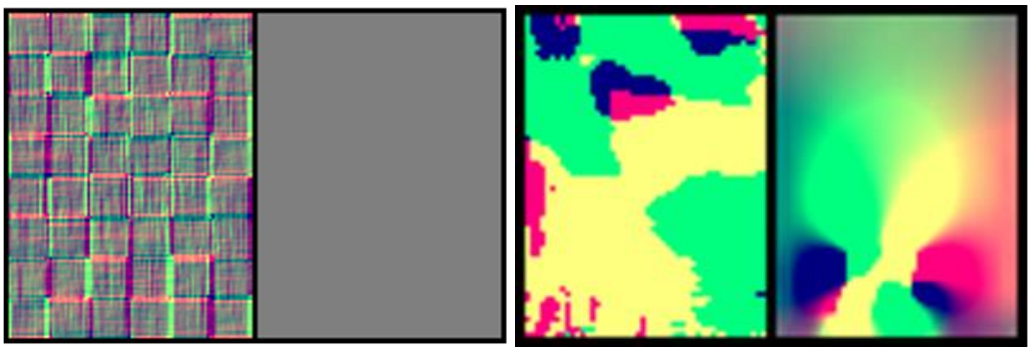


Figure 4. Original result of reconstructed velocity field with smoke. The left column is the reconstructed result, while the right column is the ground truth. The left image is a sample of erroneous results, and the right image is a sample of results after error correction.

After rectifying the issue, the corrected sample result is presented in the right image of Figure 4, demonstrating a noticeably improved reconstruction of the velocity field characterized by smoothness and coherence. The coding error, which previously hindered the proper initialization of the decoder weights, has been addressed through the manual configuration of the model saver to restore the target model from the checkpoint.

3.3. Dataset Selection and Novel Loss Function Adoption

We employ two distinct datasets in this project to train the world model.

The first dataset is constructed based on the script outlined in the original paper, utilizing the Manta framework. This dataset leverages smoke for visualization purposes, capturing velocity information solely at the positions overlapped by smoke particles. The dimensions of this dataset are specified as (100, 400, 64, 48, 2), i.e., this dataset has 100 sequences, each sequence has 400 frames, each frame is a 2-D velocity matrix of size (64, 48), and each entry of the matrix is a 2-D velocity vector. Another component of the dataset is queues of transitional parameters. The transitional parameter is defined as the list of source positions for smoke ejection, facilitating fluid dynamics modeling over time.

Our project's vortex solver generates the second dataset with dimensions of (5, 400, 256, 256, 2). Notably, this dataset does not incorporate explicit transitional parameters. Instead, we calculate the difference between consecutive states' averages as a rudimentary measure.

The original loss function for training the neural network is defined as $L_T(\boldsymbol{x_t}, \dots, \boldsymbol{x_{t+w-1}}) = \frac{1}{w}\sum_{i=t}^{t+w-1}|\Delta \boldsymbol{z_i} - T_i|_2^2$, where the adoption of window size can minimize errors accumulated in multiple consecutive steps, $\Delta \boldsymbol{z_i}$ represents the difference between two successive latent codes, and T_i is the output of the neural network at time T. However, the goal of the neural network should be learning the velocity field of training, instead of the transitional difference of training. The error minimized should be the current predicted velocity field gap against the original velocity field.

Therefore, the new loss function is defined as $L'_T(\boldsymbol{x_t}, \dots, \boldsymbol{x_{t+w-1}}) = \frac{1}{w}\sum_{i=t}^{t+w-2}|\boldsymbol{z_i} - \boldsymbol{z'_i}|_2^2$, where $\boldsymbol{z'_i}$ is defined as the sum of T_i and $\boldsymbol{z'_{i-1}}$.Using this loss function, the deviation of velocity fields can be corrected in the training process.

3.4. Performance Analysis

In this section, we analyze the performance of our approach and examine the factors that influence it. It is important to note that, to the best of the author's knowledge, there is no established standard metric for evaluating the performance of reconstructed velocity fields using machine learning techniques. Therefore, we assess performance based on a comparison between the reconstructed results and the ground truth and by measuring the deviation of the predicted latent codes. In our experiment, the ground truth velocity fields are directly encoded and decoded from velocity fields in the dataset. We will further explore the impact of factors such as the sequence length, the fitting condition of the neural network, and the effect of the newly proposed loss function. By thoroughly examining these factors, we aim to gain insights into the performance of our approach and identify critical elements that contribute to its effectiveness.

The original implementation described in the Deep Fluids paper (Kim, 2019) involves the utilization of multiple datasets to train the world model. However, the frame settings vary among different scenes, which led the author to explore the influence of data size. As a result, our initial analysis concentrates on evaluating the performance of the smoke dataset under two different configurations: 100 sequences times 400 frames and 10 sequences times 400 frames. The results obtained from the autoencoder and neural network are visually presented in Figure 5 and Figure 6, respectively. These figures provide an overview of the outcomes achieved through our approach, enabling a comparative assessment of the performance under different data size settings.

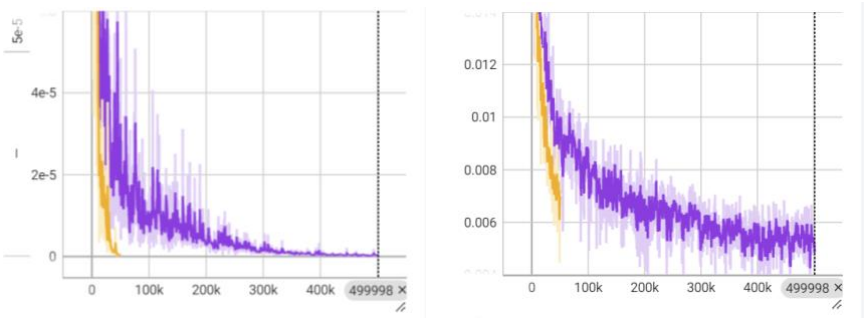


Figure 5. The training result of the autoencoder. The left figure is the loss of the supervised part (p), and the right figure is the total loss. The curve in purple represents the training with 100 sequences, and the curve in yellow represents the training with 10 sequences.

The results demonstrate that the supervised parameter, which is the encoded result of transitional parameters, tends to converge easily. In contrast, the overall loss of the autoencoder is smaller with larger data sizes. Optimizing the training process of the autoencoder is important since the encoded latent code serves as the reference and ground truth for training the neural network. Due to the distinct performance of training the velocity field and transitional parameters, we may consider training the two parts separately to facilitate the training performance, which is listed as future work.

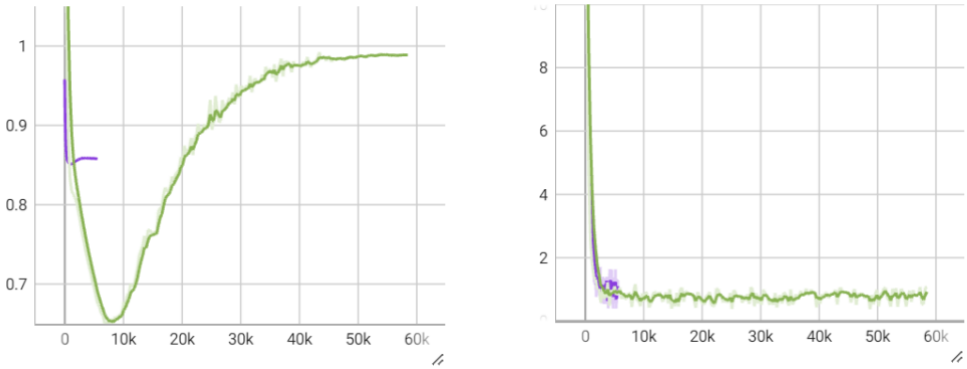


Figure 6. The testing and training result of the neural network. The curve in green represents the training with 100 sequences, and the curve in purple represents one with 10 sequences.

Conversely, in the context of neural network training, the data size does not significantly affect the training outcome of the neural network, as it approaches convergence in fewer than 10,000 steps. However, the testing result indicates the superiority of scaling up the data size, particularly within the first 10,000 steps. Moreover, the escalating testing loss suggests potential overfitting concerns when training the neural network for an excessive number of steps.

The neural network plays a crucial role in predicting future states within the world model. The

training process is visualized through the loss curve presented in Figure 7, which exhibits significant fluctuations that hinder the analysis of the fitting condition. As a result, we directly examine the differences between the predicted latent codes and the corresponding ground truth, as depicted in Figure 7. Despite the observed oscillations in the training curve, the L2 difference in latent code between the training and testing sets remains relatively consistent. This result suggests that, although the loss function does not exhibit a continuous decrease, the neural network is likely underfitted. Consequently, further optimization efforts are required, which should be considered an important area for future work.

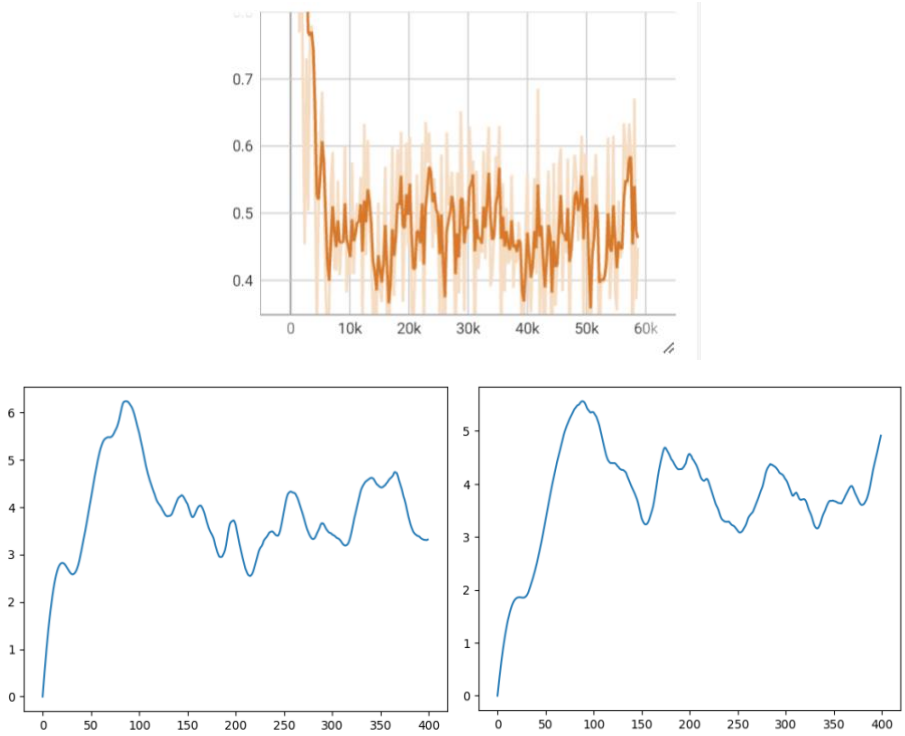


Figure 7.(Up) the training curve of fitting the neural network. (Down) The L2 difference between the predicted latent code and the ground truth is depicted in the following curves. The left curve corresponds to the training set, whereas the right represents the testing set. The horizontal axis represents the step, and the vertical axis is the difference in the L2 difference value.

Introducing the newly proposed loss function for neural network optimization is a breakpoint in our project. This is because, unlike the traditional approach that measures the transitional difference, our loss function focuses on evaluating the difference in velocity fields. This shift in metric allows us to capture the dynamics of reconstructed velocity fields without relying on rational transitional parameters. Figure 8 shows that the neural network trained using the original loss function exhibits minimal output differences. In contrast, the network trained with our loss function demonstrates significant changes in the prediction of latent code differences. While the

long-term prediction of velocity fields may exhibit some disarray, the predicted fields within the first 50 steps show remarkable similarity to the ground truth.

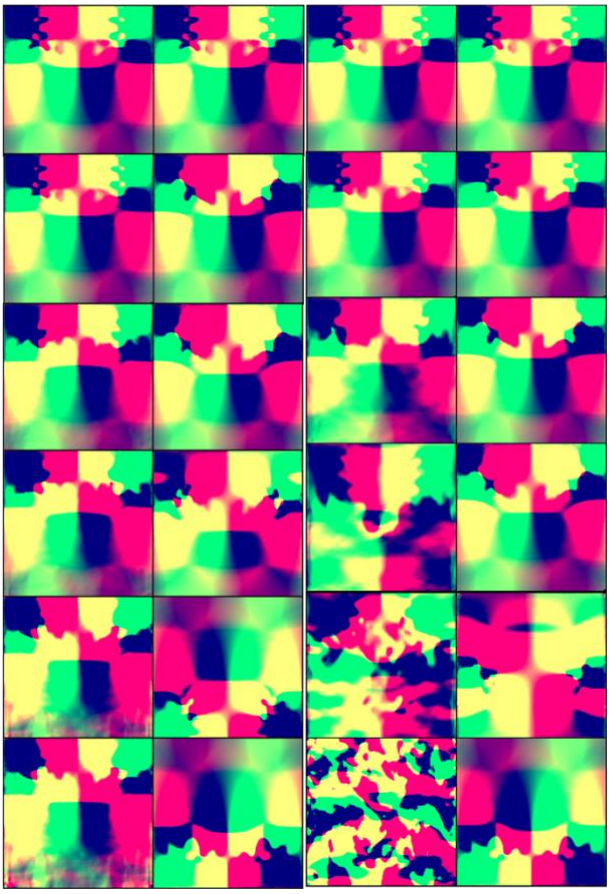


Figure 8. The reconstructed result with the neural network trained by the original loss function (left) and proposed novel loss function (right). The left column is the reconstructed result, and the right column is the reconstructed field directly from the autoencoder without the involvement of the neural network. The frames are at the time step of 0, 2, 10, 50, 200, 400, respectively.

The introduction of this novel loss function enhances our understanding of the dynamics of reconstructed velocity fields, and lays the foundation for further exploration of transitional relationships without rational transitional parameters.

3.5. Hyperparameter Analysis

As the world model heavily relies on machine learning techniques, it is crucial to fine-tune the hyperparameters and analyze their impact on performance. This section focuses on two critical hyperparameters: the window size for neural network training and the transitional parameter.

We begin by examining the effect of the window size on the neural network's performance using the original loss function. The testing and training curves of the neural network are presented in Figure 9 (see below). Interestingly, contrary to the conclusions drawn in the original paper, it is evident that the training and testing loss prevail when using a smaller window size. We also observe that lower window sizes mitigate overfitting issues as the number of timesteps increases. Conversely, the test error for the largest window size proliferates after approximately 6,000 steps.

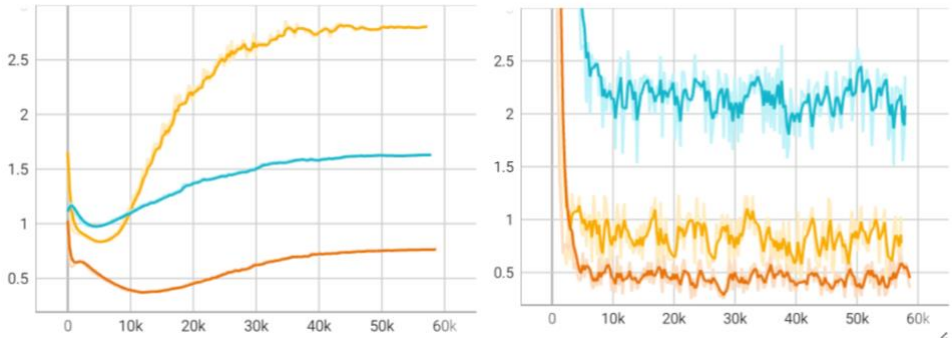


Figure 9. The testing (left) and training (right) results of the neural network using different window sizes. The light yellow curve has a window size of 15, the blue curve has a window size of 10, and the orange curve has a window size of 5.

A choice of reasonable transitional parameters is a major difference between the smoke and leapfrog datasets. Therefore, an investigation into the effect of the transitional parameters is needed. As two distinct datasets with different data sizes are compared, judging the loss in training and testing is meaningless. Instead, we visualize the gap between the latent code difference, similar to the one in analyzing the neural network training performance, shown in Figure 10 below. It reveals that introducing transitional parameters can stabilize the long-term prediction, as the two curves from the smoke dataset tend to have a bound of error in prediction. In comparison, the other two curves reveal a tendency for error accumulation. Meanwhile, the latent code difference of the leapfrog dataset corroborates the conclusion of the novel loss function made in **Section 3.3**. Although the latent code difference is larger in the neural network with the novel loss function, we cannot deduce the supriority of the original loss function, because the reconstructed result tends to have slight variation for different timesteps, as shown in Figure 8 above.

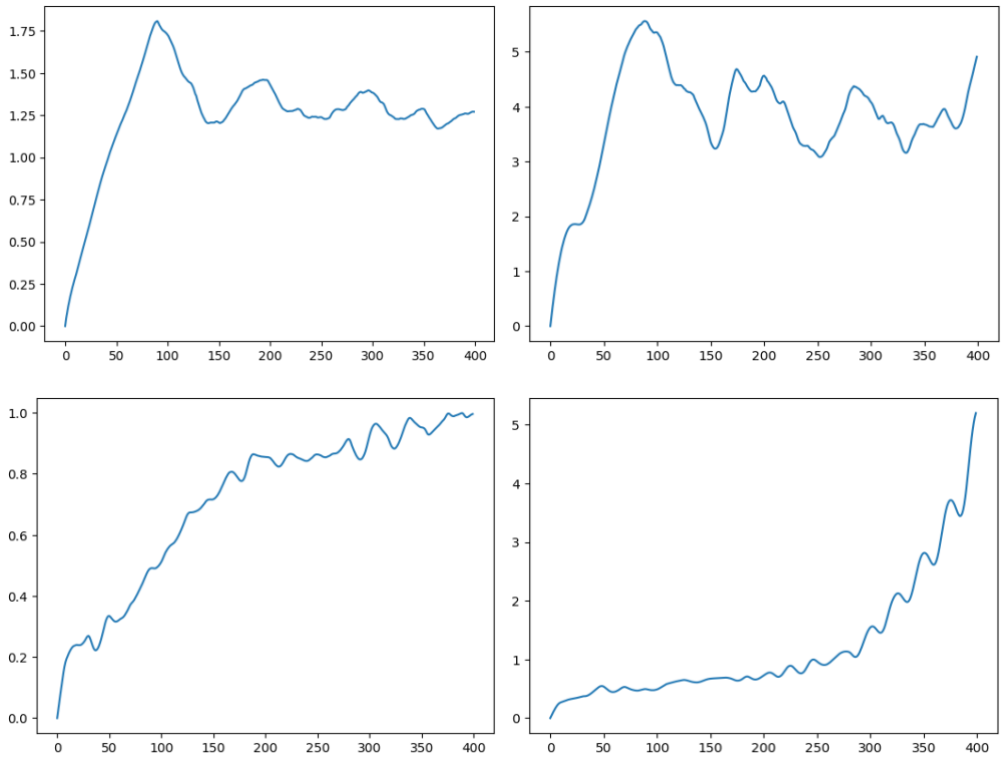


Figure 10. The latent code difference of the smoke dataset (with reasonable transitional parameter) with original loss function (upper left), with novel loss function (upper right), generated leapfrog dataset (without reasonable transitional parameter) with original loss function (lower left), and with novel loss function (lower right)

4. Conclusion

In conclusion, this paper has comprehensively explored building a virtual world model in a computer graphics context, employing advanced machine learning techniques. Our journey began with developing a fluid solver, optimized for handling solid-fluid interactions. We then shifted our focus towards training a world model, adapting to challenges such as storage limitations, applying new architecture of the world model, and analyzing the result.

Based on the current result, this architecture of training the world model requires a rational transitional parameter to help guide the simulation step for unseen input. Our newly proposed novel function may help to improve the diversity of neural network output. Still, it fails to handle long-term prediction, which can be stabilized by the involvement of rational transitional

parameters in the dataset. Further, as we require accurate prediction of future states to mimic the states in the real world for control policy training, this method requires more investigation to improve its accuracy.

Our immediate focus is a thorough review of literature in differentiable fluid simulations and self-supervised learning for control policy, which will guide our decision on whether a strategic shift is necessary. This step is crucial for the continued development and training of the world model and the control policy. Our ultimate goal is to achieve a harmonious blend of physical realism and computational efficiency in character animation in fluid environments. This endeavor not only advances the field of computer graphics but also opens new avenues for realistic simulations in various applications, from entertainment to scientific visualization.

5. Future Work

Apart from most of the work presented in the previous section, we may investigate the feasibility of another method, which develops a differentiable world model as the world model. This method requires more investigation of the traditional construction of fluids solver and the inference of numerical form of differentiation formulas. However, this method is based on numerical solvers. Hence the accuracy might be guaranteed, which is a more promising approach. Multiple excellent work has been done, like DiffFR (Li et al., 2023). However, if we need to build the differentiable world model based on the stream function solver to prevent the numerical error introduced in solving the pressure project, more research is required regarding the two-way coupling stream function solver.

References

Ando, R., Thuerey, N., & Wojtan, C. (2015). A stream function solver for liquid simulations. *ACM Transactions on Graphics*, *34*(4), 1–9. https://doi.org/10.1145/2766935

Angelidis, A., & Neyret, F. (2005). Simulation of smoke based on vortex filament primitives. *Proceedings of the 2005 ACM SIGGRAPH*. https://doi.org/10.1145/1073368.1073380

Das, K., & Behera, R. N. (2017). A survey on Machine Learning: Concept,Algorithms and Applications. *International Journal of Innovative Research in Computer and Communication Engineering*, *5*(2). https://www.rroij.com/open-access/a-survey-on-machine-learning-conceptalgorithms-and-applications-.pdf

Elcott, S., Tong, Y., Kanso, E., Schröder, P., & Desbrun, M. (2007). Stable, circulation-preserving, simplicial fluids. *ACM Transactions on Graphics*, *26*(1), 4. https://doi.org/10.1145/1189762.1189766

Gamito, P., Lopes, P. F., & Gomes, M. R. (1995). Two-dimensional simulation of gaseous phenomena using vortex particles. In *Eurographics (Wien)* (pp. 3–15). https://doi.org/10.1007/978-3-7091-9435-5_1

Ha, D., & Schmidhuber, J. (2018). World models. *arXiv (Cornell University)*. https://doi.org/10.5281/zenodo.1207631

Li, Z., Xu, Q., Ye, X., Ren, B., & Liu, L. (2023). DIFFFR: Differentiable SPH-Based Fluid-Rigid coupling for rigid body control. *ACM Transactions on Graphics*, *42*(6), 1–17. https://doi.org/10.1145/3618318

Odena, A., Dumoulin, V., & Olah, C. (2016). Deconvolution and Checkerboard artifacts. *Distill*, *1*(10). https://doi.org/10.23915/distill.00003

Peng, X. B., Abbeel, P., Levine, S., & Van De Panne, M. (2018). DeepMimic. *ACM Transactions on Graphics*, *37*(4), 1–14. https://doi.org/10.1145/3197517.3201311

Pfaff, T., Thuerey, N., & Groß, M. (2012). Lagrangian vortex sheets for animating fluids. *ACM Transactions on Graphics*, *31*(4), 1–8. https://doi.org/10.1145/2185520.2185608

Stam, J. (1999). Stable fluids. *Proceedings of SIGGRAPH '99*. https://doi.org/10.1145/311535.311548

Wu, P., Escontrela, A., Hafner, D., Goldberg, K., & Abbeel, P. (2022). DayDreamer: World Models for Physical Robot Learning. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2206.14176

Zhou, Z., Zhang, Y., & Li, Y. (2023). Model Predictive Control Design of a 3-DOF Robot Arm Based on Recognition of Spatial Coordinates. *2023 9th International Conference on Mechatronics and Robotics Engineering (ICMRE)*. https://doi.org/10.1109/icmre56789.2023.10106581