Project:
Real-Time Fluid Simulation

STUDENT 1: JUNSHENG CHEN 2020531020
STUDENT 2: YUEFENG ZHANG 2020531008

ACM Reference Format:
Student 1: Junsheng Chen 2020531020, student 2: Yuefeng Zhang 2020531008, . 2024. Project: Real-Time Fluid Simulation. 1, 1 (January 2024), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In this project, we delve into the complex and intricate world of computer graphics with a particular focus on the simulation of fluid dynamics in real-time, specifically the simulation of smoke. The cornerstone of our approach is the implementation of a more precise and accurate advection solver that significantly enhances the realism of the smoke's behavior and appearance.

Our journey began with the stable fluids solver, a robust foundation for simulating the flow and turbulence of fluids. Recognizing the need for further refinement, we introduced modifications to the advection mechanism through the implementation of a reflection solver. This advanced technique improves the simulation's ability to reflect the intricate and dynamic nature of smoke, capturing the subtle nuances and variable densities that characterize real-world phenomena.

To harness the full power of modern computational hardware, we have translated these complex algorithms into the realm of parallel processing by implementing them on a GPU with the use of the CUDA framework. This strategic move has not only expedited the calculations but also allowed for more detailed and extensive simulations, pushing the boundaries of what can be achieved in real-time graphics.

Further enhancing our simulation's visual output, we have integrated a GPU-based volume renderer for the velocity field, which translates the smoke's motion into visually stunning and highly detailed graphics. This renderer is a pivotal component in visualizing the smoke, offering a depth and clarity that brings the simulated fluid to life.

The results we have obtained so far are promising and demonstrate the potential of these techniques in creating immersive graphical experiences.

Author's address: Student 1: Junsheng Chen 2020531020
student 2: Yuefeng Zhang 2020531008
.

XXXX-XXXX/2024/1-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 IMPLEMENTATION DETAILS

2.1 Implement the stable fluids solver

2.1.1 set boundary values. This function is used to adjust the values of the field at the boundaries, making the fluid interaction with the boundaries more natural and realistic. The boundary we are using here is a square.

2.1.2 lin interp. This function performs three-dimensional linear interpolation for specific points in space.

2.1.3 transport. This function simulates the transport process of particles (using density) in each grid, and then obtains the approximate density of specific spatial locations through linear interpolation(function:lin interp).

2.1.4 lin solve. This function is a linear solver capable of solving systems of linear equations. The specific implementation has been detailed in the "Stable Fluids" paper and will not be elaborated here. This function serves as a tool for the subsequent "diffuse" and "project" functions.

2.1.5 diffuse. This function is designed to simulate the diffusion process of various properties of fluids, using the "lin solve" function to ensure the preservation of some overall characteristics of the field.

2.1.6 project. The purpose of this function is to eliminate certain components of the velocity field to ensure the incompressibility of the fluid. The calculation is also detailed in the paper and will not be overly elaborated on here.

2.1.7 v step and s step. These two functions represent the application of all the previous functions; one is used to update the velocity field, and the other is used to update the density field.

2.2 The reflection solver

The "reflect" is an operator used in fluid animation to replace the traditional energy-dissipating projection operator. Its main purpose is to be applied during intermediate steps of the simulation in order to reduce energy loss in the system and improve the preservation of details. By using the reflect function, a two orders of magnitude reduction in energy loss can be achieved, leading to significantly enhanced preservation of small-scale details in fluid animation. This function's advantage lies in its seamless integration with existing projection-advection solvers and its minimal additional implementation requirements. Its application holds the potential to enhance the visual quality of fluid animation while avoiding the introduction of artificial viscosity and the suppression of small-scale details.

For realizing, the "reflect" algorithm takes a velocity field vel, calculates the pressure field, and then reflects the velocity field about a modified version of itself represented by velDivFree.

2.3 Fluid

"Fluid" part is a total use of "solver" part, we renew velosity and density in "step function". Except that, we also add force and give other setting functions here. The buoyancy we use here is according to the equation(we ignore T part for simplify):

$$\vec{b} = [\alpha s - \beta(T - T_{\text{amb}})]\vec{g}, \tag{1}$$

2.4 Implement on GPU using CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform developed by NVIDIA. It allows developers to program directly on the GPU using specialized programming languages and APIs, enabling efficient parallel computation. This technology is particularly suited for tasks requiring extensive parallel processing, such as scientific computing, deep learning, and image processing. CUDA, with its highly optimized memory management and multi-GPU support, significantly enhances computational speed and efficiency, making GPUs a powerful tool for handling large-scale computational tasks.So sure, it can be used here.

In our project, except “v step” and "s step", we realize all other functions mentioned in 2.1,2.2 and 2.3, and move them all in the "kernels" file.

2.5 GPU-based volume renderer of velocity field

The rendering equation for this body first traverses all directions of the screen, checks for intersections with the previously defined three-dimensional space, and if there is an intersection, it steps in until the maximum value is reached. During each step, it calculates cumulative opacity and cumulative color until either the step is completed or the opacity is large enough to stop the calculation. As required, we provide two methods for opacity calculation here, one based on scalar density fields and one based on velocity fields, which will be shown in the third part later.
For efficiency in computation, in this part, we also utilize GPU acceleration through CUDA, mainly in the intersection of ray blocks and the traversal of each direction for volume rendering.

2.6 Visualization using OpenGL

For visibility and convenience, we have also configured the presentation of the results in OpenGL, just like in the first assignment. We have written shaders, set up VAOs (Vertex Array Objects), VBOs (Vertex Buffer Objects), and added a visualization window with interactive controls. This not only enhances the "3D" effect but also improves playability.

After doing this, what you need to do after downloading is just load cmakelists and then run exe file "opengl".

3 RESULTS

3.1 Begin

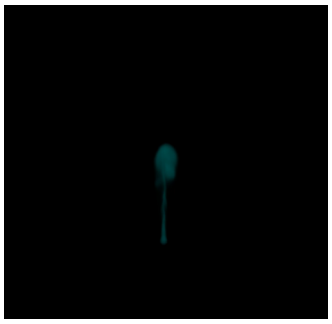

Fig. 1. density denpended volume renderer

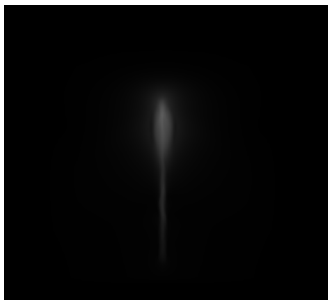

Fig. 2. velosity denpended volume renderer

3.2 Mid

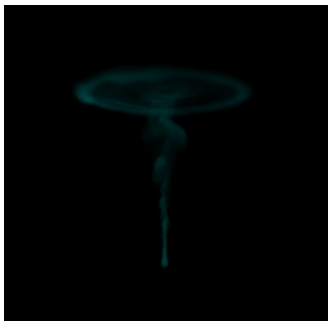

Fig. 3. density denpended volume renderer

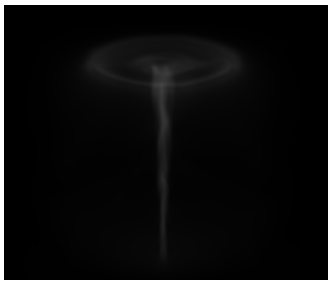

Fig. 4. velosity denpended volume renderer

3.3 End

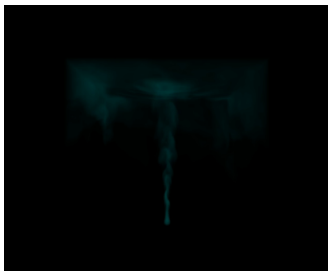

Fig. 5. density denpended volume renderer

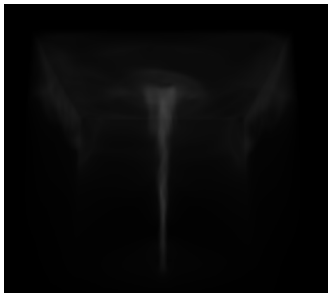

Fig. 6. velosity denpended volume renderer

We also apply two videos, if you are want to know more clearly, please click the link below:
https://epan.shanghaitech.edu.cn/l/LFOblm