Implementation of smoke simulation by BiMocq2 method in Houdini

Di Yang

Masters Project

N.C.C.A Bournemouth University

MSc Computer Animation and Visual Effects 2019-20

August 24, 2020

Abstract

This is an implementation of the smoke fluid settlement based on the $BiMocq^2$(2 levels of Bi-directional mapping of convective quantities) method to obtain a more detailed, more precise and stable fluid simulation. This implementation is done using houdini so that Houdini users can easily apply it in their daily work. This implementation is slightly different from the original paper. The order of the algorithm steps is slightly adjusted, and the final method of affecting the speed variable is changed to better fit Houdini's original smoke simulation.

Acknowledgements

First of all, I would also like to thank Jian Chang. His theoretical guidance and help in analyzing mathematical formulas are valuable contributions to this paper. I would also like to thank Jon Macey for all the useful feedback and suggestions provided throughout the paper. Finally, Many thanks to the class of 2019-20, MSc Computer Animation and Visual Effects, Bournemouth University

Contents

Chapter 1

Introduction

Fluid simulation based on Euler's equations has successfully reproduced various phenomena in computer graphics, such as liquids and fireworks. The framework of solving the Navier-Stokes equations in a time-division manner, especially the key insights of using the Semi-Lagrangian scheme to deal with nonlinear convection terms, brings its most significant advantage, namely stability, and therefore easy to control, however the biggest problem is: numerical diffusion(Bridson, 2016).

Besides, the current popular fluid simulation methods have the problem of insufficient details under the same resource situation, or require more resources and time to satisfy the details. At the same time, the popular Volume Up-res method is difficult to set up, or the added details are incoherent and unreal.

So in order to get a more effective and detailed fluid simulation, a new method called $BiMocq^2$(double levels of Bi-directional mapping of convective quantities), a more stable, accurate, and more detailed fluid simulation method(Qu *et al.*, 2019), has been introduced.

This article will focus on the implementation of the $BiMocq^2$ method in Houdini, which uses the new HDA to replace the original smoke Advect part of the Pyro tool. it could obtain a more detailed, more accurate and stable fluid simulation, simultaneously, keep the way of using simple and artist-friendly.

Chapter 2

Previous Work

Semi-Lagrangian advection method(Stam, 1999) has been widely used in some highly practical simulation for computer graphics and opened a wide range of utilisation for physically-based fluid animation. However, due to the repeated use of field interpolation, the semi-Lagrangian method has a problem of numerical diffusion. Therefore, a large number of algorithms have been proposed to solve this problem.

2.0.1 Higher order Semi-Lagrangian methods

For example, BFECC(Kim *et al.*, 2005) and MacCormack(Selle *et al.*, 2007) are the most important and commonly used fluid settlement methods in Houdini. They are all based on Semi-Lagrangian and restore the numerical accuracy of the flow by measuring the difference between the flow before and after at a given time step. It has been proved that it could reduce dissipation and diffusion encountered in various advection steps in fluid simulation such as velocity, smoke density and image advections(Kim *et al.*, 2005). However, this method lacks the necessary mechanism to avoid accumulated errors, so the process will eventually disappear.

2.0.2 Particle-grid hybrid methods

At present, the most commonly used fluid solution in graphics is Particle-grid hybrid methods(Jiang *et al.*, 2015), which has abandoned the previous pure Eulerian or Lagrangian algorithm (for example, SPH, which has been abandoned after Houdini15). The FLIP(Zhu and Bridson, 2005) algorithm in Houdini is to copy the particle velocity increment to the Euler grid behind it and then calculate the pressure field in the Euler grid, and then calculate the new non-dispersive velocity field based on the calculated pressure field, and then Copy it back to the particles. One of the great benefits of this is that it combines the advantages of the European method that can effectively discretize and solve the pressure equation and the pull method to solve the leading term with small numerical dissipation. APIC(Jiang *et al.*, 2015) and PolyPic(Fu *et al.*, 2017) methods further amend the numerical noise of FLIP by constructing other functions to retain local flow characteristics.

2.0.3 Vorticity modeling

Most visual effect fluid solvers are calculated based on time, where the flow velocity is first advection in the flow and then projected as an incompressible fluid. "Even if a highly accurate advection scheme is used, the self-advection step typically transfers some kinetic energy from divergence-free modes into divergent modes, which are then projected out by pressure, losing energy noticeably for large time steps(Zhang *et al.*, 2015)." Instead of taking smaller time steps or using significantly more complex time integration, many people propose method dealing with the curl form of Navier-Stokes equations, like WeiBmann and Pinkall (2010) introduce a vortex filaments to deal with this curl form.
Recently, Zhang *et al.* (2015) proposed a new scheme called IVOCK (Comprehensive Vorticity of Convective Motion), which can cheaply capture most of the losses in self-convection by identifying self-convection as "a violation of the vorticity equation(Zhang *et al.*, 2015)".

2.0.4 Energy preserving solvers

The common point of existing methods is that energy loss is not only a function of simulation duration, but also depends on the size of the grid and the number of time steps performed during the simulation. And often use complex computer systems to eliminate this effect. Mullen *et al.* (2009) proposes a simple, unconditionally stable, fully Euler integration scheme without numerical viscosity, which can keep the fluid movement active without relying on a correction device. recent research made by Zehnder *et al.* (2018) provides an alternative approach for detail-preserving fluid animation that is surprisingly simple and shows great energy preservation.

Chapter 3

Technical background

3.0.1 Navier-Stokes Equations

In physics, Navier-Stokes equations are a set of partial differential equations describing the movement of viscous fluid materials, and are widely regarded as good fluid flow models. Assuming that the velocity and pressure are known at some initial time t = 0, the evolution of these quantities over time is given by the Navier-Stokes equation:

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{1}{\rho} \nabla p + v \cdot \nabla^2 u + f \tag{3.1}$$

$$\nabla \cdot u = 0$$

where u, p, ρ, v, frepresent velocity, pressure, density, and forces such as gravity. Integrating the above, we can get the following Figure 3.1 to illustrate the factors that cause fluid changes.

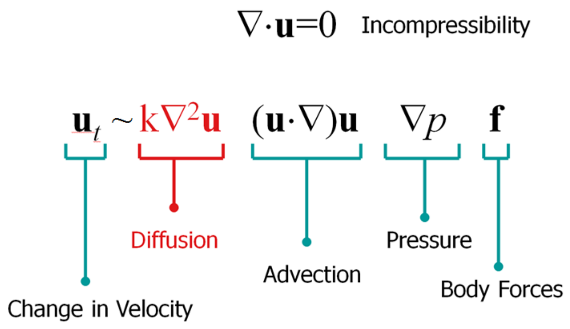

$\nabla \cdot \mathbf{u} = 0$
Incompressibility
$\mathbf{u}_t \sim k\nabla^2\mathbf{u}$ $(\mathbf{u}\cdot\nabla)\mathbf{u}$ ∇p $\mathbf{f}$
Diffusion
Pressure
Advection
Body Forces
Change in Velocity

Figure 3.1: Navier-Stokes Equations: Incompressible Flow

3.0.2 Semi-Lagrangian Advection

The Semi-Lagrangian Scheme (SLS) is a numerical method used to express the equation of fluid motion. this scheme uses the Euler framework, but the discrete equation comes from Lagrange's viewpoint. The idea is to calculate the source of point parcel at each time step. Then, an interpolation scheme is used to estimate the value of the dependent variable at the grid points surrounding the point from which the particles originate.
from Lagrangian view of Navier-Stokes Equations we can get:

$$\frac{d\rho}{dt} + div(\rho\mathbf{v}) = 0 \tag{3.2}$$

In this research, only the advection part should be considered for the entire algorithm,so when in the time division framework, we can get:

$$\frac{\partial v}{\partial t} = -v \cdot \nabla v \tag{3.3}$$

the $\frac{d\rho}{dt}$ could be consided as $\rho_{current}$ - ρ_{back},wet can get a Simple Forward Euler estimation:

$$\rho_{current} = \rho_{back} - v_{back}\Delta t \tag{3.4}$$

So we can understand that Advection means transport quantity from one region to another along the fluid's velocity field(Fedkiw *et al.*, 2001).

3.0.3 VEX Language in Houdini

VEX is the abbreviation of "Vector Expression Language". It is Houdini's embedded scripting language. It is a high-performance C pedigree scripting language for writing custom nodes or shaders. It also absorbs many concepts of Pixar's RenderMan shading language. Houdini automatically compiles VEX scripts before execution, so it can process a large number of data points efficiently. For example, in most other languages, you must traverse all the points in some way to create or modify data, VEX only needs one line @value = 0 can complete the creation or modification of attributes, which is very efficient. At the same time, VEX can import external custom functions (i.e.*.h file) which is extremely convenient and flexible. One disadvantage of coding, especially in vex wrangle editors, is very forgiving when you get things wrong. The error messages are incredible, grammatical errors are easy to cause, and help is not always helpful.

3.0.4 New Pyro Solver(Sparse)

In Houdini18, a new smoke solving system is introduced, called SPARSE PYRO. This solver can run in sparse mode, that is, the required calculations are only performed in the region of interest controlled by the active field, which will greatly save computing time and resources.

Prior to Houdini 18, pyro simulations involving combustion used a fuel field. The subsequent combustion reaction, heat refresh, and smoke generation are all solved in the DOP part. However, the new workflow requires that burns be generated immediately in the SOP and introduced as part of the procurement. The burning volume is created as the source of the initial explosion. Then, this instantaneous combustion merges with the flame field and divergence of the subject.

This method has tremendous advantages of predictability and more control. Its entire structure is more concise and more logical. It is completely based on Navier-Stokes Equations to construct the entire system. The main body can be clearly divided into advection, force, projection, dissipate and other parts, which correspond to the NS equations in Figure3.1.

Chapter 4

Implementation of BiMocq2

In this section, we will first explain the principle of the BiMocq2 algorithm in detail, and then explain its implementation in Houdini, including backward mapping, forward mapping, error correction and our time integration scheme, which uses two levels of Mapping is used to effectively predict the convective velocity field in the later stage. For the sake of brevity, our algorithm discussion and explanation will only focus on the velocity field, but the same derivation is still valid for any other fluid field.

4.0.1 BiMocq2

BiMocq2 was first proposed by Qu *et al.* (2019). In general, the BiMocq2 (n levels of bidirectional flow mapping) method has the following characteristics when used to solve fluids:

- This proposed method is unconditionally stable and can simulate arbitrarily large Δt (even in some cases CFL>30).
- The advection scheme is purely Eulerian, which can be easily parallelized through GPU, increasing performance by 45 times.
- This method track multi-level mapping to improve sharpness and coherence of the simulation simultaneously keeping the computational efficiency.
- Based on the mapping function, the method offers a long-term error correction function to improve the visual quality of the product for further simulation.
- This method has strong compatibility with previous fluid solvers, so it can be integrated into existing pipelines with minor modifications.

The main framework of BiMocq2 can be divided into four-part, given in Figure4.1:

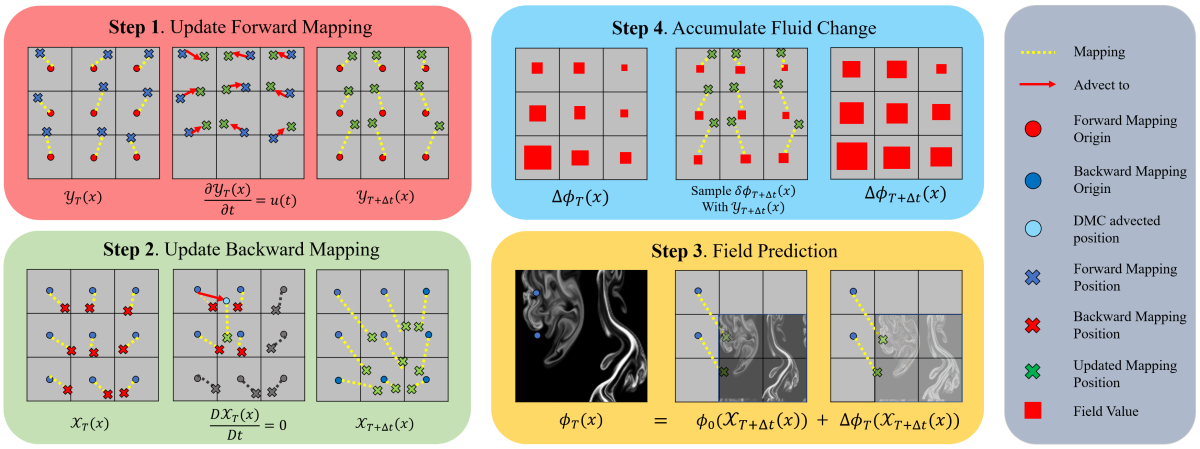

Step 1. Update Forward Mapping
$\mathcal{Y}_T(x)$
$\frac{\partial \mathcal{Y}_T(x)}{\partial t} = u(t)$
$\mathcal{Y}_{T+\Delta t}(x)$
Step 4. Accumulate Fluid Change
$\Delta\phi_T(x)$
Sample $\delta\phi_{T+\Delta t}(x)$ With $\mathcal{Y}_{T+\Delta t}(x)$
$\Delta\phi_{T+\Delta t}(x)$
Step 2. Update Backward Mapping
$\mathcal{X}_T(x)$
$\frac{D\mathcal{X}_T(x)}{Dt} = 0$
$\mathcal{X}_{T+\Delta t}(x)$
Step 3. Field Prediction
$\phi_T(x) = \phi_0(\mathcal{X}_{T+\Delta t}(x)) + \Delta\phi_T(\mathcal{X}_{T+\Delta t}(x))$
Mapping
Advect to
Forward Mapping Origin
Backward Mapping Origin
DMC advected position
Forward Mapping Position
Backward Mapping Position
Updated Mapping Position
Field Value

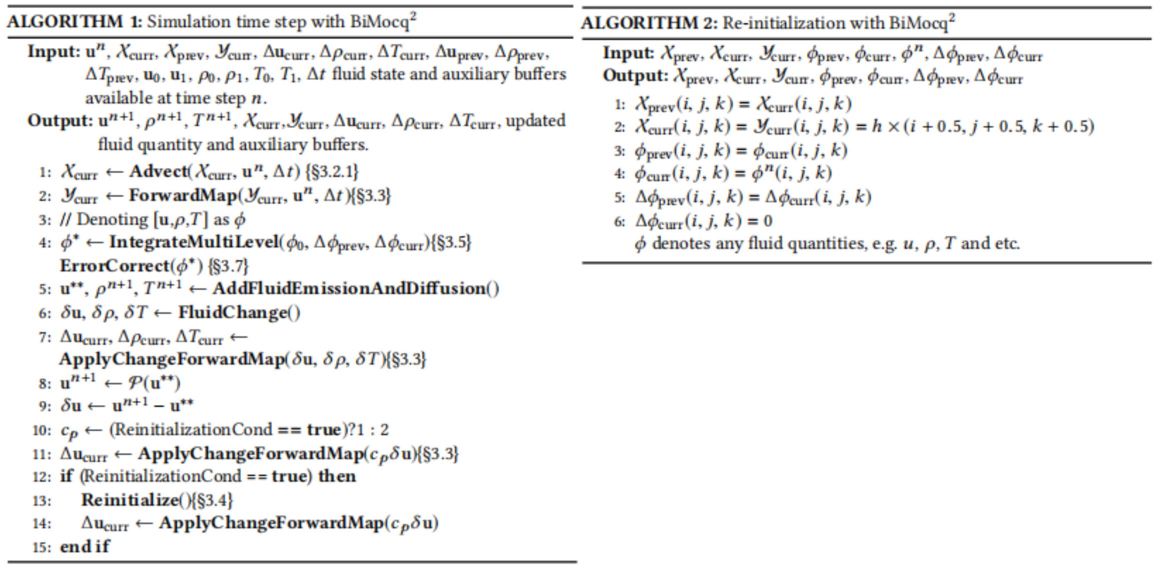
Figure 4.1: Schematic illustration of a one-level BiMocq2

- Step 1: update the forward mapping by solving a partial differential equation of the mapping γ.
- Step 2: update the backward mapping by solving the advection equation of the mapping χ.
- Step 3: any fluid quantity ϕ is obtained by using the updated backward mapping $\chi_{T+\Delta t}$ and interpolating the fluid origin buffer ϕ_0 and accumulation buffer $\Delta\phi_T$.
- Step 4: accumulate any fluid change through the updated forward mapping $\gamma_{T+\Delta t}$ to accumulation buffer $\Delta\phi$ (Qu *et al.*, 2019).

In general, the BiMocq2 method does not have too difficult parts to implement, but it takes a certain amount of time for the DMC algorithm and RK3 algorithm used in the backward and forward parts.

4.0.2 Modification of BiMocq2 in Houdini implementation

Although the original BiMocq2 method is already very clear and organized, some minor modifications have been made to the original method due to the limitations of Houdini's node operation. The process of the original version can be seen in Figure 4.2.The process changes can be seen in Alg1, and the Re-initialization part has not changed.



ALGORITHM 1: Simulation time step with BiMocq^2

```
Input: $\mathbf{u}^n$, $\mathcal{X}_{\text{curr}}$, $\mathcal{X}_{\text{prev}}$, $\mathcal{Y}_{\text{curr}}$, $\Delta\mathbf{u}_{\text{curr}}$, $\Delta\rho_{\text{curr}}$, $\Delta T_{\text{curr}}$, $\Delta\mathbf{u}_{\text{prev}}$, $\Delta\rho_{\text{prev}}$,
       $\Delta T_{\text{prev}}$, $\mathbf{u}_0$, $\mathbf{u}_1$, $\rho_0$, $\rho_1$, $T_0$, $T_1$, $\Delta t$ fluid state and auxiliary buffers
       available at time step $n$.
Output: $\mathbf{u}^{n+1}$, $\rho^{n+1}$, $T^{n+1}$, $\mathcal{X}_{\text{curr}}$,$\mathcal{Y}_{\text{curr}}$, $\Delta\mathbf{u}_{\text{curr}}$, $\Delta\rho_{\text{curr}}$, $\Delta T_{\text{curr}}$, updated
        fluid quantity and auxiliary buffers.
 1: $\mathcal{X}_{\text{curr}} \leftarrow$ Advect($\mathcal{X}_{\text{curr}}$, $\mathbf{u}^n$, $\Delta t$) {§3.2.1}
 2: $\mathcal{Y}_{\text{curr}} \leftarrow$ ForwardMap($\mathcal{Y}_{\text{curr}}$, $\mathbf{u}^n$, $\Delta t$){§3.3}
 3: // Denoting [$\mathbf{u}$,$\rho$,$T$] as $\phi$
 4: $\phi^* \leftarrow$ IntegrateMultiLevel($\phi_0$, $\Delta\phi_{\text{prev}}$, $\Delta\phi_{\text{curr}}$){§3.5}
    ErrorCorrect($\phi^*$) {§3.7}
 5: $\mathbf{u}^{**}$, $\rho^{n+1}$, $T^{n+1} \leftarrow$ AddFluidEmissionAndDiffusion()
 6: $\delta\mathbf{u}$, $\delta\rho$, $\delta T \leftarrow$ FluidChange()
 7: $\Delta\mathbf{u}_{\text{curr}}$, $\Delta\rho_{\text{curr}}$, $\Delta T_{\text{curr}} \leftarrow$
    ApplyChangeForwardMap($\delta\mathbf{u}$, $\delta\rho$, $\delta T$){§3.3}
 8: $\mathbf{u}^{n+1} \leftarrow \mathcal{P}(\mathbf{u}^{**})$
 9: $\delta\mathbf{u} \leftarrow \mathbf{u}^{n+1} - \mathbf{u}^{**}$
10: $c_p \leftarrow$ (ReinitializationCond == true)?1 : 2
11: $\Delta\mathbf{u}_{\text{curr}} \leftarrow$ ApplyChangeForwardMap($c_p\delta\mathbf{u}$){§3.3}
12: if (ReinitializationCond == true) then
13:     Reinitialize(){§3.4}
14:     $\Delta\mathbf{u}_{\text{curr}} \leftarrow$ ApplyChangeForwardMap($c_p\delta\mathbf{u}$)
15: end if
```

ALGORITHM 2: Re-initialization with BiMocq^2

```
Input: $\mathcal{X}_{\text{prev}}$, $\mathcal{X}_{\text{curr}}$, $\mathcal{Y}_{\text{curr}}$, $\phi_{\text{prev}}$, $\phi_{\text{curr}}$, $\phi^n$, $\Delta\phi_{\text{prev}}$, $\Delta\phi_{\text{curr}}$
Output: $\mathcal{X}_{\text{prev}}$, $\mathcal{X}_{\text{curr}}$, $\mathcal{Y}_{\text{curr}}$, $\phi_{\text{prev}}$, $\phi_{\text{curr}}$, $\Delta\phi_{\text{prev}}$, $\Delta\phi_{\text{curr}}$
 1: $\mathcal{X}_{\text{prev}}(i,j,k) = \mathcal{X}_{\text{curr}}(i,j,k)$
 2: $\mathcal{X}_{\text{curr}}(i,j,k) = \mathcal{Y}_{\text{curr}}(i,j,k) = h \times (i+0.5, j+0.5, k+0.5)$
 3: $\phi_{\text{prev}}(i,j,k) = \phi_{\text{curr}}(i,j,k)$
 4: $\phi_{\text{curr}}(i,j,k) = \phi^n(i,j,k)$
 5: $\Delta\phi_{\text{prev}}(i,j,k) = \Delta\phi_{\text{curr}}(i,j,k)$
 6: $\Delta\phi_{\text{curr}}(i,j,k) = 0$
    $\phi$ denotes any fluid quantities, e.g. $u$, $\rho$, $T$ and etc.
```

Figure 4.2: The original design of program pipelines of BiMocq2

Algorithm 1 Implementation pipelines of BiMocq2.

```
Input:
    $u^n$,$\chi_{curr}$,$\chi_{temp}$,$\chi_{init}$,$\gamma_{curr}$,$\chi_{prev}$,$\Delta u_{curr}$,$\Delta t$,
Output:
    $u^{n+1}$,$u_{change}$;
 1: $u^{n+1} \rightarrow$ Semi-advect($u^n$,$\Delta t$);
 2: $\chi_{curr} \rightarrow$ BackwardUpadte($u^n$,$\Delta t$); [Sec4.0.3]
 3: $\gamma_{curr} \rightarrow$ ForwardUpdate($u^n$,$\Delta t$); [Sec4.0.4]
 4: float Distortion = estimateVectorDistortion($\chi_{curr}$,$\gamma_{curr}$);
 5: if(Distortion $\geqslant$ 1.0 or forceinitframe > 10)
 6: Reinitialize(); [Sec4.0.6]
 7: $\chi_{prev} \rightarrow$ Advect($\chi_{init}$, $u^n$, $\Delta t$);
 8: $\chi_{temp} \rightarrow$ Calculationerror($\chi_{prev}$,$\chi_{init}$,$\gamma_{curr}$);
 9: $\chi_{init} \rightarrow$ Cumulateerror($\chi_{temp}$,$\chi_{curr}$); [Sec4.0.5]
10: $u_{change} \rightarrow$ IntegrateMultiLevel($\chi_{prev}$,$\chi_{curr}$);
11: $u^{n+1} \rightarrow$ ApplyChange($u^n$, $u_{change}$);
12: return $u^{n+1}$
```

It is worth concerning that the original method needs to consider the changes to the fluid, made by the force, projection field etc. The current method does not need to consider these, because these will be calculated and recorded in the FieldChange field. This method will Creating errors but will make the implementation of the algorithm easier, all processes can be completed in an HDA.

4.0.3 Backward Mapping Update

The repeated application of semi-Lagrangian convection is one of the main origins of numerical diffusion. MCM is designed to skip all intermediate interpolations by going back to the previous moment and directly interpolating from that moment to obtain a sharper fluid state.

The idea is to find a backward mapping χ, so that given a spatial position x(T), this mapping will calculate its original position x(t0) to satisfy Eqn4.1, showing below:

$$x(t_0) = \chi(x(T)) = x(T) - \int_{t_0}^{T} u(x(\tau), \tau) d\tau \tag{4.1}$$

then we dynamically advect the mapping as:

$$\frac{D\chi}{Dt} = \frac{\partial \chi}{\partial t} + u(t) \cdot \nabla \chi = 0 \tag{4.2}$$

Given a fluid parcel at x(T), the backward mapping of $\chi(x(T))$ returns its original position x(t0) which actually remains fixed as the fluid parcel moves. With Eqn4.2, may acquire a new time step mapping by simply advecting a long time step map. We can use semi-Lagrangian advection to get:

$$\chi^{n+1}(x) = \chi^{n}(\bar{x}) \tag{4.3}$$

where $\bar{x} = x - \Delta t \cdot u^{n}(x)$ in the standard Semi-Lagrangian solvers.

However, Semi-Lagrangian Advection has been proved that in highly rotational velocity fields, this simple strategy drastically loses its accuracy(Qu *et al.*, 2019). therefore, the dual-mesh characteristics (DMC) has been introduced.

Dual mesh characteristic

It is developed based on The particle-mesh method (PMM)(Cho *et al.*, 2018). PMM is a powerful calculation tool that can be used to simulate a convection-based diffusion flow. However, the method in practical applications shows the so-called ”ringing instability(Cho *et al.*, 2018)”, so the second mesh which is formed by tracking back the cells along with the characteristics was imported. The DMC algorithm proves to be conservative in quality, without oscillation, with negligible dissipation, and more effective than conventional schemes. The numerical results show its accuracy and efficiency(Cho *et al.*, 2018). ”DMC solves a final value problem to find the starting point of fluid parcel whose trajectory ends at fixed grid nodes. With piecewise linear velocity field, DMC derives an analytical solution which contains exponentials(Qu *et al.*, 2019).”

The DMC routine can be described as following(explained in 1D, will achieve in 3D): let x be the point position where we store our mapping, the other point position $\hat{x}$ will be:

$$\hat{x} = \begin{cases} x - h & \text{if } v(x) <= 0 \\ x + h & \text{if } v(x) > 0 \end{cases} \tag{4.4}$$

Now assume that the speed can be linearly mixed between x and $\hat{x}$:

$$v(\xi) = v(\hat{x}) + (\xi + \hat{x})a; a = \frac{v(x) - v(\hat{x})}{x - \hat{x}} \tag{4.5}$$

$$a = \frac{v(x) - v(\hat{x})}{x - \hat{x}} \tag{4.6}$$

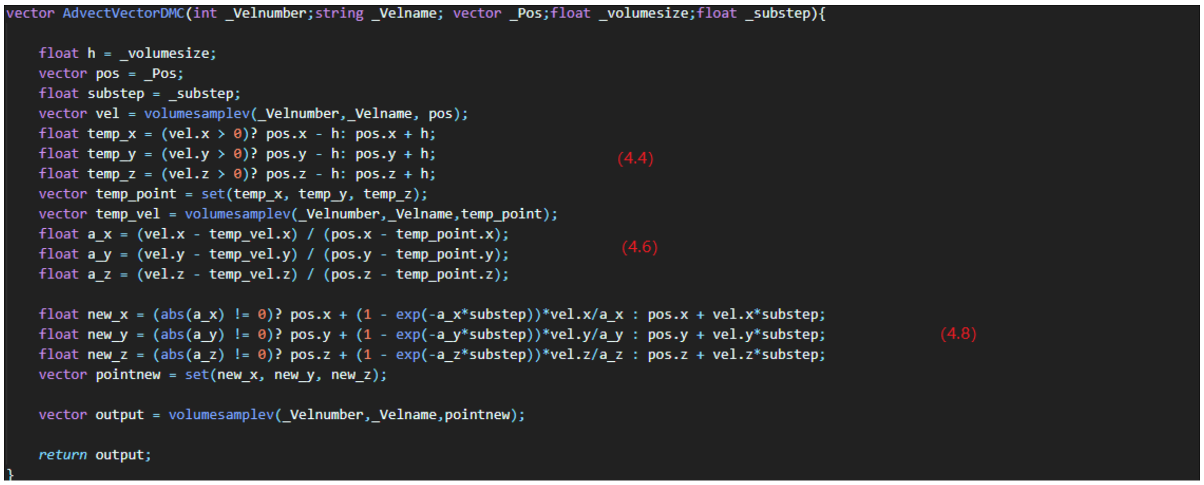
because of $v(x)dt = d\xi$, could get:

$$\Delta t = \int_{t^{n-1}}^{t^n} dt = \int_{\bar{x}}^{x} \frac{1}{v(x)} d\xi \tag{4.7}$$

Combining (4.7) and (4.4) we can get the tracing-back position as:

$$\bar{x} = \begin{cases} x - v(x)\Delta t & \text{if } a = 0; \\ x - \frac{1}{a}(1 - e^{-a\Delta t})v(x) & \text{otherwise;} \end{cases} \tag{4.8}$$

the implementation of BackwardUpdate with DMC can be see below:



```
vector AdvectVectorDMC(int _Velnumber;string _Velname; vector _Pos;float _volumesize;float _substep){

    float h = _volumesize;
    vector pos = _Pos;
    float substep = _substep;
    vector vel = volumesamplev(_Velnumber,_Velname, pos);
    float temp_x = (vel.x > 0)? pos.x - h: pos.x + h;
    float temp_y = (vel.y > 0)? pos.y - h: pos.y + h;                    (4.4)
    float temp_z = (vel.z > 0)? pos.z - h: pos.z + h;
    vector temp_point = set(temp_x, temp_y, temp_z);
    vector temp_vel = volumesamplev(_Velnumber,_Velname,temp_point);
    float a_x = (vel.x - temp_vel.x) / (pos.x - temp_point.x);
    float a_y = (vel.y - temp_vel.y) / (pos.y - temp_point.y);           (4.6)
    float a_z = (vel.z - temp_vel.z) / (pos.z - temp_point.z);

    float new_x = (abs(a_x) != 0)? pos.x + (1 - exp(-a_x*substep))*vel.x/a_x : pos.x + vel.x*substep;
    float new_y = (abs(a_y) != 0)? pos.y + (1 - exp(-a_y*substep))*vel.y/a_y : pos.y + vel.y*substep;          (4.8)
    float new_z = (abs(a_z) != 0)? pos.z + (1 - exp(-a_z*substep))*vel.z/a_z : pos.z + vel.z*substep;
    vector pointnew = set(new_x, new_y, new_z);

    vector output = volumesamplev(_Velnumber,_Velname,pointnew);

    return output;
}
```

Figure 4.3: Implementation of BackwardUpdate with DMC

4.0.4 Forward Mapping Update

After completing the backward mapping, it is time to introduce its corresponding forward mapping, which is used to track accumulated fluid changes and their characteristics. However, when there are external influences such as viscosity and pressure, we should consider the acceleration of the fluid. Mathematically speaking, the relationship between the accumulated fluid change and its original position can be defined as:

$$\Delta u(x(t_0), t) := \int_{t_0}^{t} d\tau \frac{Du}{Dt}(x(\tau), \tau) \tag{4.9}$$

then we define a forward mapping $\gamma_{t_0}^{t}(x) : x(t_0) \mapsto x(t)$,With this forward mapping, the accumulated change can now be updated as:

$$\Delta u(x, t+\Delta t) = \Delta u(x, t) + \delta(\gamma_{t_0}^{t+\Delta t}(x), t+\Delta t) \tag{4.10}$$

This means that to track acceleration, the integral along the time-reversed path should be evaluated to accumulate fluid changes. This will be very time-consuming and labor-intensive.

Fortunately, according to the (Qu *et al.*, 2019), practically all numerical tests can use the third-order accurate Runge-Kutta method to avoid a large number of integral calculations.

Runge-Kutta method

In numerical analysis, Runge-Kutta methods (Runge-Kutta methods) are an important type of implicit or explicit iterative methods for the solution of nonlinear ordinary differential equations. These techniques were invented by mathematicians Karl Runge and Martin Wilhelm Kutta around 1900.

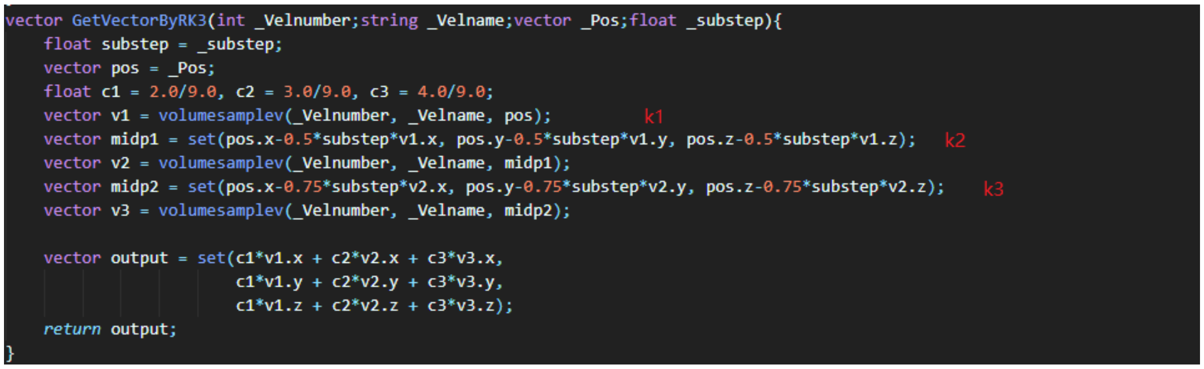
The derivation of the third-order Runge-Kutta is very complicated, and the more detailed derivation process needs to refer to the article of Ralston (1962). The final result of the derivation will be directly given here:

$$y_{n+1} - y_n = \frac{2}{9}k_1 + \frac{1}{3}k_2 + \frac{4}{9}k_3 \tag{4.11}$$

where:

$$\begin{aligned} k_1 &= h_n f(x_n, y_n) \\ k_2 &= h_n f(x_n + \frac{1}{2}h_n, y_n + \frac{1}{2}k_1) \\ k_3 &= h_n f(x_n + \frac{3}{4}h_n, y_n + \frac{3}{4}k_2) \end{aligned}$$

the implementation of ForwardUpdate with RK3 can be see below:



```
vector GetVectorByRK3(int _Velnumber;string _Velname;vector _Pos;float _substep){
    float substep = _substep;
    vector pos = _Pos;
    float c1 = 2.0/9.0, c2 = 3.0/9.0, c3 = 4.0/9.0;
    vector v1 = volumesamplev(_Velnumber, _Velname, pos);          k1
    vector midp1 = set(pos.x-0.5*substep*v1.x, pos.y-0.5*substep*v1.y, pos.z-0.5*substep*v1.z);    k2
    vector v2 = volumesamplev(_Velnumber, _Velname, midp1);
    vector midp2 = set(pos.x-0.75*substep*v2.x, pos.y-0.75*substep*v2.y, pos.z-0.75*substep*v2.z);    k3
    vector v3 = volumesamplev(_Velnumber, _Velname, midp2);

    vector output = set(c1*v1.x + c2*v2.x + c3*v3.x,
                        c1*v1.y + c2*v2.y + c3*v3.y,
                        c1*v1.z + c2*v2.z + c3*v3.z);
    return output;
}
```

Figure 4.4: Implementation of ForwardUpdate with RK3

4.0.5 Error correction and Accumulated Fluid Changes

Currently, there are still two possible sources of numerical diffusion, which may deter getting more visual details in actual simulations. "Any numerical dissipation generated until the reinitialization time will be inherited in future simulations. At the same time, sampling from the previously mapped fluid state to obtain the current flow prediction may also be affected by diffusion.(Qu *et al.*, 2019)" Assume that the numerical spread e between the long-term backward and forward mapping of the estimated fieldϕ is:

$$\phi^*(x_g) = I(\phi_0, \chi(x_g)) \tag{4.12}$$

where I represents a sampling method and ϕ_0 represents the initial state. so that e should be:

$$e(x_g) = \frac{1}{2}(I(\phi^*, \gamma(x_g)) - \phi_0(x_g)) \tag{4.13}$$

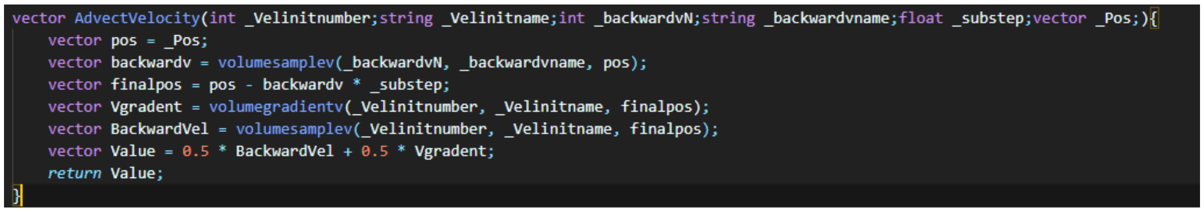
When making long-term corrections, cumulative changes must be considered. Assuming that the long-term error is calculated into, the formula becomes:

$$\phi^*(x_g) = I(\phi_0 + \Delta\phi, \chi(x_g)) \tag{4.14}$$

$$e(x_g) = \frac{1}{2}(I(\phi^*, \gamma(x_g)) - (\phi_0(x_g) + \Delta\phi(x_g))) \tag{4.15}$$

Due to the limited data processed by Houdini Gasfield Wrangle node at one time, the entire process of this part is divided into three parts.

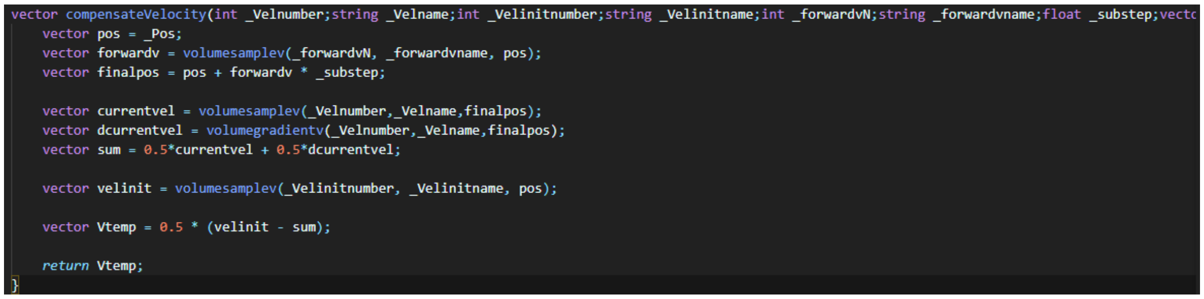
The first part is for $\phi^*(x_g)$:



```
vector AdvectVelocity(int _Velinitnumber;string _Velinitname;int _backwardvN;string _backwardvname;float _substep;vector _Pos;){
    vector pos = _Pos;
    vector backwardv = volumesamplev(_backwardvN, _backwardvname, pos);
    vector finalpos = pos - backwardv * _substep;
    vector Vgradent = volumegradientv(_Velinitnumber, _Velinitname, finalpos);
    vector BackwardVel = volumesamplev(_Velinitnumber, _Velinitname, finalpos);
    vector Value = 0.5 * BackwardVel + 0.5 * Vgradent;
    return Value;
}
```

Figure 4.5: Implementation of Advect Velocity by BackwardVelocity

The second part is for $e(x_g)$:

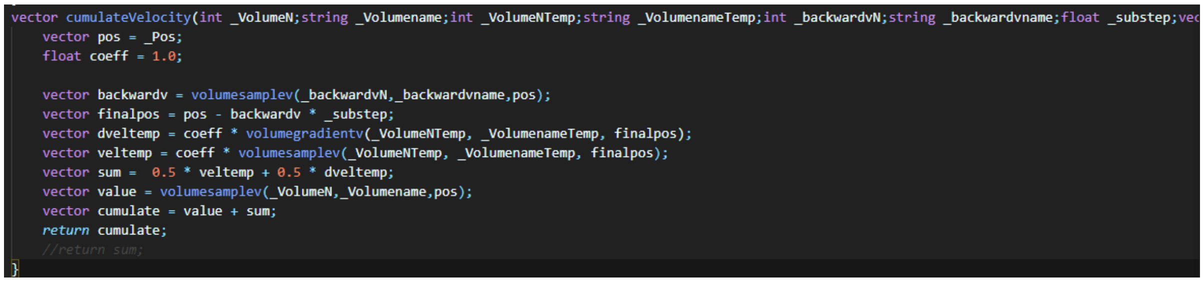


```
vector compensateVelocity(int _Velnumber;string _Velname;int _Velinitnumber;string _Velinitname;int _forwardvN;string _forwardvname;float _substep;vecto
    vector pos = _Pos;
    vector forwardv = volumesamplev(_forwardvN, _forwardvname, pos);
    vector finalpos = pos + forwardv * _substep;

    vector currentvel = volumesamplev(_Velnumber,_Velname,finalpos);
    vector dcurrentvel = volumegradientv(_Velnumber,_Velname,finalpos);
    vector sum = 0.5*currentvel + 0.5*dcurrentvel;

    vector velinit = volumesamplev(_Velinitnumber, _Velinitname, pos);

    vector Vtemp = 0.5 * (velinit - sum);

    return Vtemp;
}
```

Figure 4.6: Implementation of Advect Velocity back by ForwardVelocity

Here, the two functions actually complete a BFECC(Back and Forth Error Compensation and Correction) type error correction.

The third part is used to collect fluid changes, which is to get $\phi_0 + \Delta\phi$:



```
vector cumulateVelocity(int _VolumeN;string _Volumename;int _VolumeNTemp;string _VolumenameTemp;int _backwardvN;string _backwardvname;float _substep;vec
    vector pos = _Pos;
    float coeff = 1.0;

    vector backwardv = volumesamplev(_backwardvN,_backwardvname,pos);
    vector finalpos = pos - backwardv * _substep;
    vector dveltemp = coeff * volumegradientv(_VolumeNTemp, _VolumenameTemp, finalpos);
    vector veltemp = coeff * volumesamplev(_VolumeNTemp, _VolumenameTemp, finalpos);
    vector sum =  0.5 * veltemp + 0.5 * dveltemp;
    vector value = volumesamplev(_VolumeN,_Volumename,pos);
    vector cumulate = value + sum;
    return cumulate;
    //return sum;
}
```

Figure 4.7: Implementation of collect fluid changes

This part requires special attention and is the biggest difference from the original method. Mainly because of the following two reasons:

1. In Houdini, especially the use of GasField wangle node data transmission has great restrictions. When using VEX in these nodes, auto-bind is invalid. You can only manually specify the field to be used, and there is a limit of 4 inputs. This makes the entire section(Error Correction and Accumulated fluid changes) divided into many subsections.

2. In order to make the entire realization process in an HDA more friendly and simple, this part gives

up the monitoring of the original target, and instead only records the changes of the target field, which means giving up monitoring the dissipation of the target field that caused in the process of force and projection etc.

4.0.6 Re-initialization and Estimate the Distortion

When the backward and forward mapping is highly distorted, the indicated mapping function will no longer be accurate. So reinitialization is required to reinitialize the mapping function. During the reinitialization, the mapping will be set to the initial state, and there is mandatory initialization length is set. The specific initialization algorithm is summarized as shown in Figure 4.2.
The method used to detect the high distortion of the backward and forward mapping is also very simple. Ideally, if all calculations are accurate, the backward mapping of the forward mapping of the position will return the position itself. Therefore, the inaccuracy can be measured by looking at the difference between the front and back mappings of the position. The formula is as follows:

$$d_1 = \|\gamma(\chi(x(t))) - x(t)\|$$

$$d_2 = \|\chi(\gamma(x(t_0))) - x(t_0)\|$$

$$d_{max} = max(d_1, d_2) \tag{4.16}$$

the implementation of estimate the distortion can be seen below:

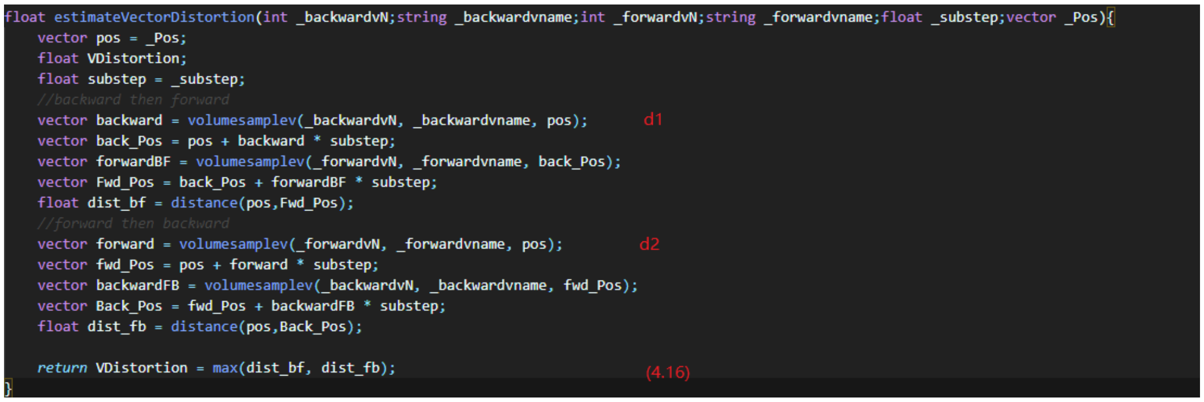
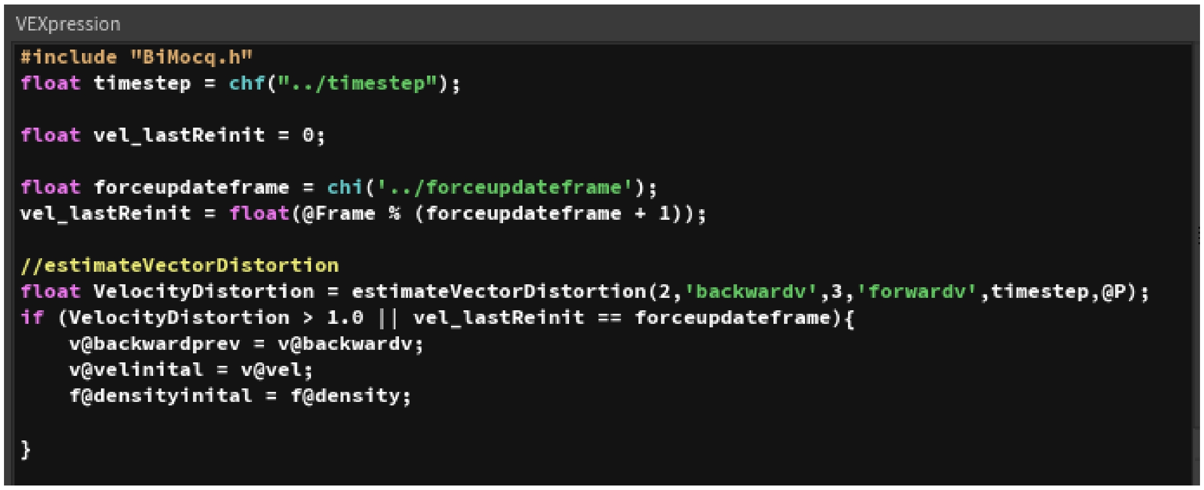

```
float estimateVectorDistortion(int _backwardvN;string _backwardvname;int _forwardvN;string _forwardvname;float _substep;vector _Pos){
    vector pos = _Pos;
    float VDistortion;
    float substep = _substep;
    //backward then forward
    vector backward = volumesamplev(_backwardvN, _backwardvname, pos);       d1
    vector back_Pos = pos + backward * substep;
    vector forwardBF = volumesamplev(_forwardvN, _forwardvname, back_Pos);
    vector Fwd_Pos = back_Pos + forwardBF * substep;
    float dist_bf = distance(pos,Fwd_Pos);
    //forward then backward
    vector forward = volumesamplev(_forwardvN, _forwardvname, pos);          d2
    vector fwd_Pos = pos + forward * substep;
    vector backwardFB = volumesamplev(_backwardvN, _backwardvname, fwd_Pos);
    vector Back_Pos = fwd_Pos + backwardFB * substep;
    float dist_fb = distance(pos,Back_Pos);

    return VDistortion = max(dist_bf, dist_fb);                            (4.16)
}
```

Figure 4.8: Implementation of estimate the distortion

Due to the limitation of the use of VEX in the DOP nodes, Function cannot directly manipulate data with @name. The whole re-initialization method is completed inside Houdini, as shown below:


VEXpression

```
#include "BiMocq.h"
float timestep = chf("../timestep");

float vel_lastReinit = 0;

float forceupdateframe = chi('../forceupdateframe');
vel_lastReinit = float(@Frame % (forceupdateframe + 1));

//estimateVectorDistortion
float VelocityDistortion = estimateVectorDistortion(2,'backwardv',3,'forwardv',timestep,@P);
if (VelocityDistortion > 1.0 || vel_lastReinit == forceupdateframe){
    v@backwardprev = v@backwardv;
    v@velinital = v@vel;
    f@densityinital = f@density;

}
```

Figure 4.9: Implementation of Re-initialization

When reinitializing the double characteristic mapping, the first thing to do is to retain the previous backward mapping which is very important, maintaining the key data for later Multi-level mapping. Then the current target field is reinitialized to the position of the grid unit.

4.0.7 Multi-level Mapping

Although the reinitialization of the mapping may solve the distortion problem, after the reinitialization operation, it will introduce obvious blur and incoherence at the frame. In order to better preserve the visual clarity and temporal continuity of the simulation, two levels of mapping can be used to track the fluid volume.

Multi-level mapping part is the most difficult to understand and confusing part of the whole method. The formula is actually very simple, but it is very difficult to understand how it is used.

Let t_{I-1} be the time when the previous reinitialization of the mapping occurred, simultaneously Let t_I be the time when the most recent mapping reinitialization occurred, get:

$$\chi_0(x(t_I)) : x(t_I) \mapsto x(t_{I-1}) \tag{4.17}$$

$$\chi_1(x(t_I)) : x(t) \mapsto x(t_I) \tag{4.18}$$

where t is the current time step, then,Combine the above two formulas to get:

$$\chi_t(x(t)) : x(t) \mapsto x(t_{I-1}) = \chi_0(\chi_1(x(t))) \tag{4.19}$$

Therefore, with two-level forward mapping, the velocity component at the grid unit XX can be represented as:

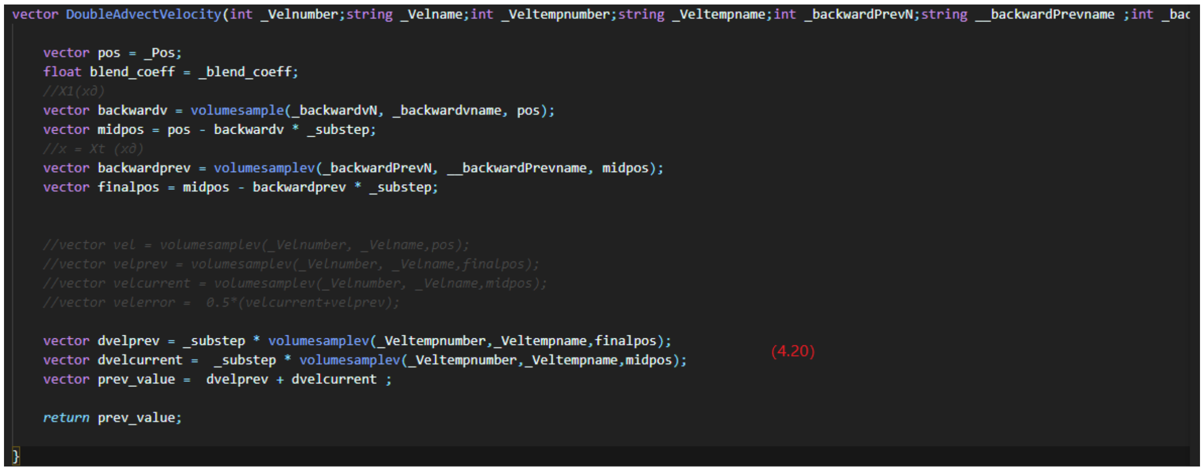
$$x = \chi_t(x_g)$$

$$u(x_g, t) = I(u_{I-1}, x) + I(\Delta u_{I-1}, x) + I(\Delta u_I, \chi_1(x_g)) \tag{4.20}$$

so the vel change can be repersent as:

$$\nabla u = u(x_g, t) - I(u_{I-1}, x) = I(\Delta u_{I-1}, x) + I(\Delta u_I, \chi_1(x_g)) \tag{4.21}$$

where ∇u should be the $e(x_g)$ got from Equ4.15 so the implementation of can be see below:



```
vector DoubleAdvectVelocity(int _Velnumber;string _Velname;int _Veltempnumber;string _Veltempname;int _backwardPrevN;string __backwardPrevname ;int _bac

    vector pos = _Pos;
    float blend_coeff = _blend_coeff;
    //X1(x∂)
    vector backwardv = volumesample(_backwardvN, _backwardvname, pos);
    vector midpos = pos - backwardv * _substep;
    //x = Xt (x∂)
    vector backwardprev = volumesamplev(_backwardPrevN, __backwardPrevname, midpos);
    vector finalpos = midpos - backwardprev * _substep;


    //vector vel = volumesamplev(_Velnumber, _Velname,pos);
    //vector velprev = volumesamplev(_Velnumber, _Velname,finalpos);
    //vector velcurrent = volumesamplev(_Velnumber, _Velname,midpos);
    //vector velerror =  0.5*(velcurrent+velprev);

    vector dvelprev = _substep * volumesamplev(_Veltempnumber,_Veltempname,finalpos);
    vector dvelcurrent =  _substep * volumesamplev(_Veltempnumber,_Veltempname,midpos);          (4.20)
    vector prev_value =  dvelprev + dvelcurrent ;

    return prev_value;

}
```

Figure 4.10: Implementation of Double Advect Velocity Error

In the original paper(Qu *et al.*, 2019), the two processed velocity fields were also mixed to obtain a more accurate velocity field. However, because the process of this algorithm implementation only requires the change of the velocity field, so the implementation stop here. please read the original paper if need.

Chapter 5

Results and Discussion

5.0.1 Result

The BiMocq2 method is implemented in Houdini mainly to get a more accurate solution with more details. From the results, the original setting is reached. The BiMocq2 method not only gets more details under the same conditions, which leads to the better visual richness, but also leads to more turbulence.

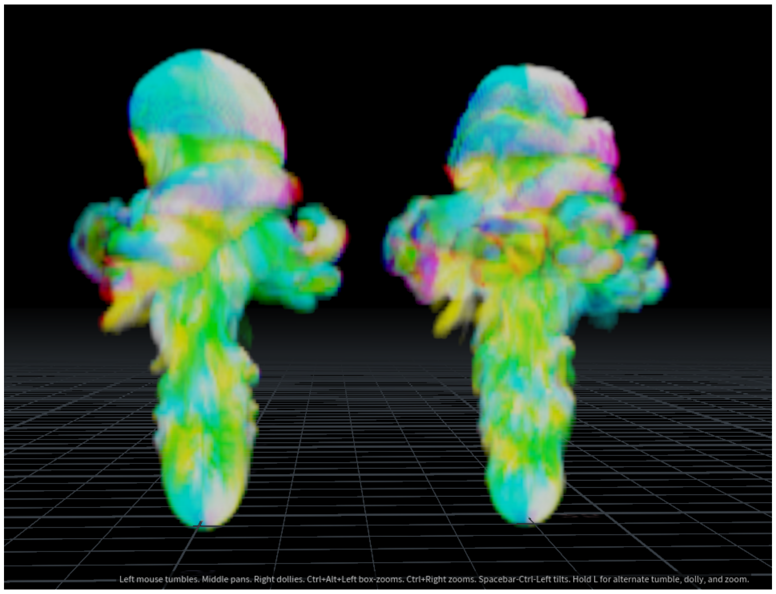

Left mouse tumbles. Middle pans. Right dollies. Ctrl+Alt+Left box-zooms. Ctrl+Right zooms. Spacebar-Ctrl-Left tilts. Hold L for alternate tumble, dolly, and zoom.

Figure 5.1: the left one is the Houdini preset billowy Smoke. The right one is the result of replacing the MacCormack advection in the preset with BiMocq2

simple Semi-Lagrangian

The Figure5.2 below is a comparison between the Semi-Lagrangian method and the Bimocq2 method. It can be found that the difference between the two methods is not obvious without the influence of too many other factors. This is mainly because in this simple case, the fluid dissipation and error are small, and the difference between various algorithms is not too large.

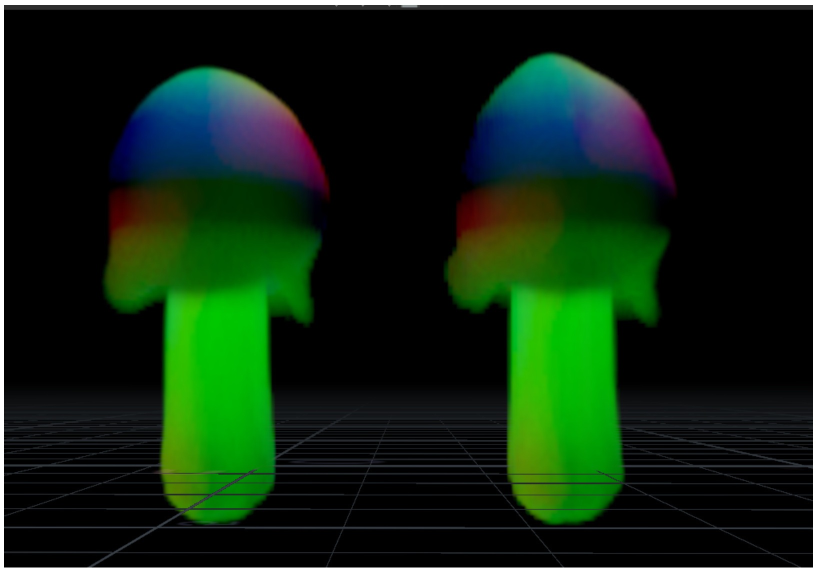


Figure 5.2: The left one is Semi-Lagrangian. The right one is BiMocq2

Addition of other conditions

But the situation is different when other noise parameters are added. The Figure5.3 below is the result of adding some other parameters(disturbance turbulence) on the basis of the Semi-Lagrangian simulation. It can be found that BiMocq2 gives more credible details:

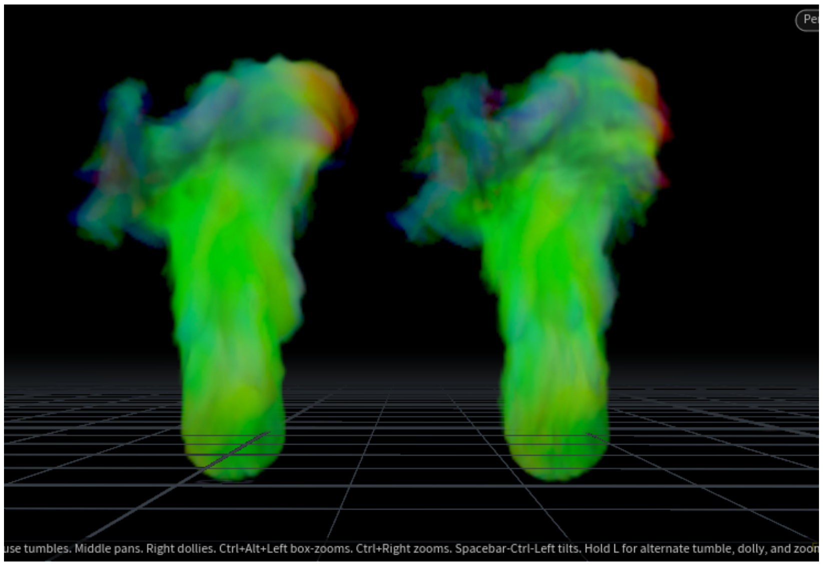

use tumbles. Middle pans. Right dollies. Ctrl+Alt+Left box-zooms. Ctrl+Right zooms. Spacebar-Ctrl-Left tilts. Hold L for alternate tumble, dolly, and zoom

Figure 5.3: The left one is Semi-Lagrangian with Noise. The right one is BiMocq2 with Noise

With MacCormack

In addition, the BiMocq2 method shows strong compatibility and can be applied with Houdini in-built fluid solution to get better results.

From the Figure5.7 below, it can be found that MacCormack obtains better and more detailed calculation results than Semi-Lagrangian without changing all parameters, and BiMocq2 can add more credible details on the basis of MacCormack.

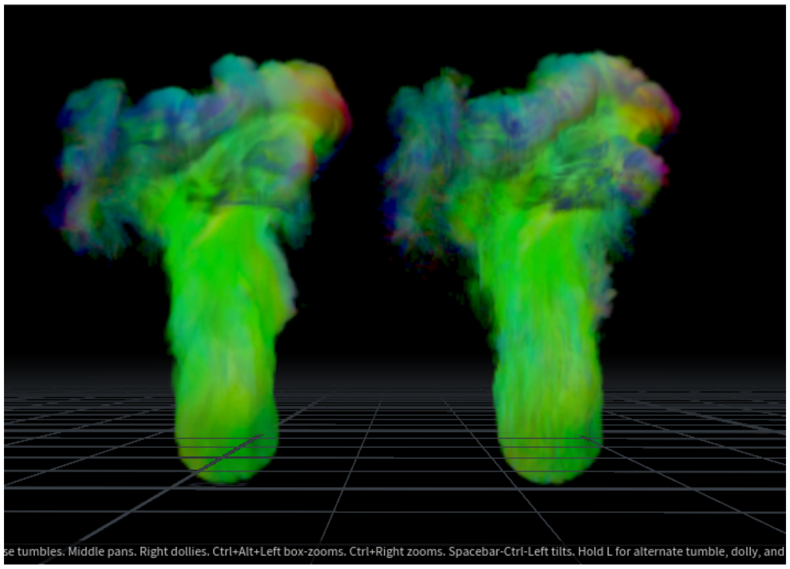

se tumbles. Middle pans. Right dollies. Ctrl+Alt+Left box-zooms. Ctrl+Right zooms. Spacebar-Ctrl-Left tilts. Hold L for alternate tumble, dolly, and

Figure 5.4: The left one is MacCormack with Noise. The right one is MacCormack + BiMocq2 with Noise

With Collision

Also tested the situation with collision, as shown Figure5.6. It can be found that it works well with collision. In this implementation, the abandonment of monitoring the dissipation of the target field caused by the process of force and projection did not cause too much loss.

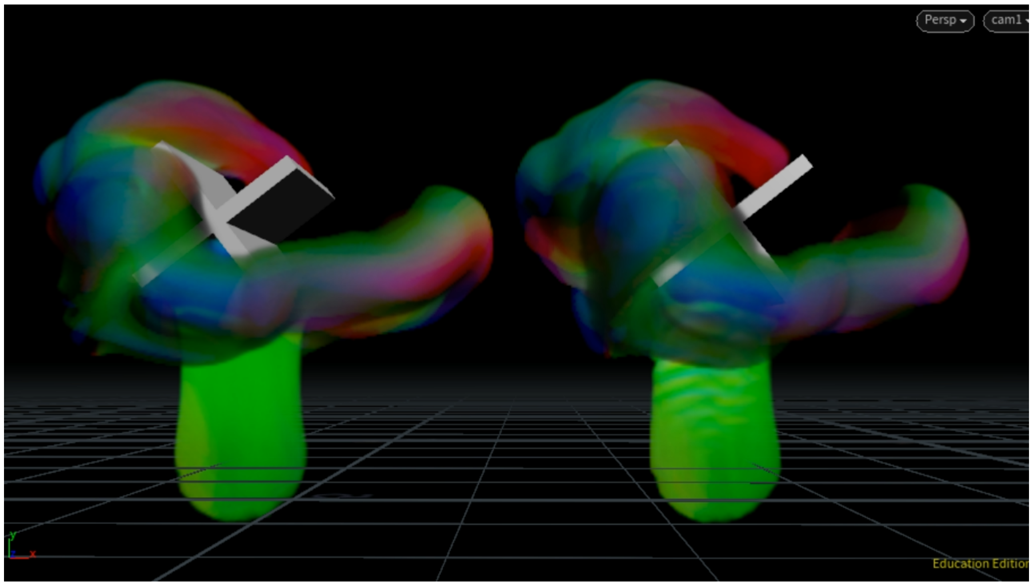


Figure 5.5: The left one is Semi-Lagrangian with Collision. The right one is BiMocq2 with Collision

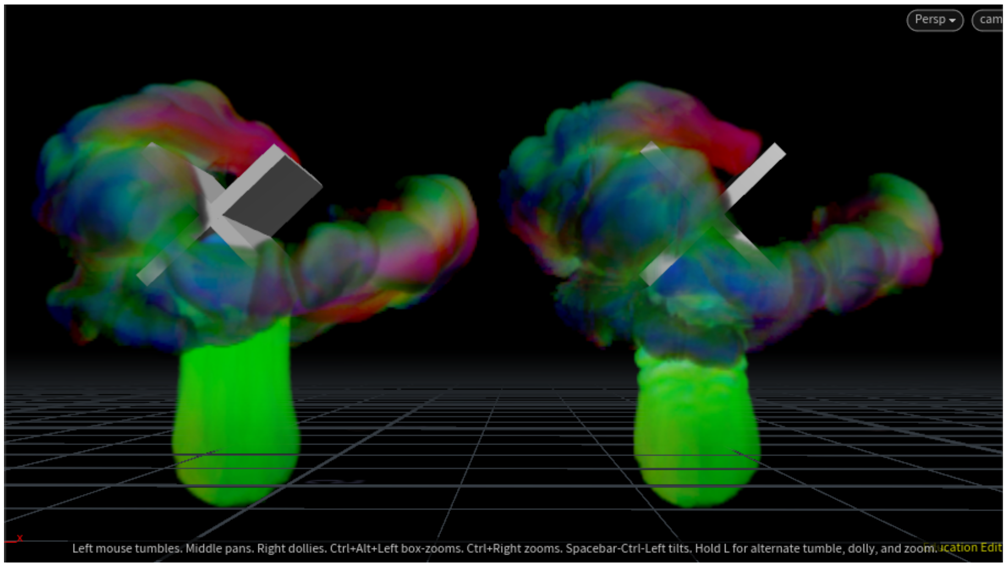

Persp
cam
Left mouse tumbles. Middle pans. Right dollies. Ctrl+Alt+Left box-zooms. Ctrl+Right zooms. Spacebar-Ctrl-Left tilts. Hold L for alternate tumble, dolly, and zoom.
ication Edit

Figure 5.6: The left one is MacCormack with Collision. The right one is MacCormack + BiMocq2 with Collision

5.0.2 Insufficiency

Time-consuming Problem

It has to be said that the BiMocq2 method takes much more time than other in-built fluid solutions and does not reach the speedup of at least 2 to 45 times stated in the original article(Qu *et al.*, 2019).

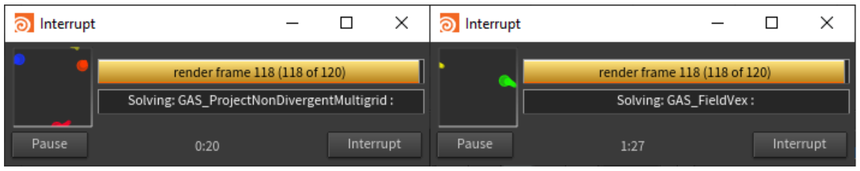

Interrupt
render frame 118 (118 of 120)
Solving: GAS_ProjectNonDivergentMultigrid :
Pause
0:20
Interrupt
Interrupt
render frame 118 (118 of 120)
Solving: GAS_FieldVex :
Pause
1:27
Interrupt

Figure 5.7: The left one is the time using in-built Semi-Lagrangian. The right one is BiMocq2

The main reason should be that too many operations to read and write data drag down the performance of the algorithm. Since the gasfield-wrangle node can only read 4 fields of data at a time, and the data cannot be transmitted between the two nodes, this leads to a lot of unnecessary read/write operations, which leads to the entire algorithm taking more time.

Accuracy Test

At present, only tests related to velocity field have been completed. Although the function of the scalar field has been completed, the effect is not obvious in the current test. Accuracy tests, such as the Vortex Leapfrogging test and Rayleigh-Taylor Instability test, need to set up a specific working environment, plus a long time to solve. Unfortunately, there is no time to do it.

5.0.3 Furure Work

Accuracy Test

First of all, accuracy tests should be completed to determine whether the algorithm is implemented correctly. Although implementation has shown very good characteristics of BiMocq2, because there are some modifications to the original process structure during the algorithm implementation, Accuracy tests can help determine whether these changes are correct.

Optimize the BiMocq2 algorithm

the implementation of BiMocq2 algorithm is far from over, there is still a lot of work to be done. The whole algorithm has great prospects. From the results, the added details of the algorithm are very delicate and real. And the algorithm has strong compatibility and stability, if the settlement time problem is solved, it can be widely used.

GPU Acceleration

In addition, as the Qu *et al.* (2019) said, the BiMocq2 method is a pure Euler method with accuracy and low dissipation. It should be easily parallelized with the help of GPU, achieving a 45X performance improvement. Houdini supports the OpenCL protocol for GPU acceleration, so this will be a good research direction.

Applied to Fluid

The characteristics of BiMocq2 algorithm make it have great potential to be applied to fluid simulation. Specifically, this method also has great potential in numerically predicting the vortex of circulating flow. Imagine that the length of the vortex segment can be determined by forward mapping so that the time vorticity can be determined. This method can avoid the vortex stretching term in the numerically unstable 3D simulation.

Chapter 6

Conclusion

The purpose of this project is to get a more effective and detailed fluid simulation method. The current popular fluid simulation method has the problem of insufficient details under the same resource situation, or require more resources and time to meet the details. At the same time, the popular Volume Up-res method has trouble in setting, or the added details are incoherent and unreal. The BiMocq2 method can get more and more real details under the same resources. Although the implementation still has issues such as Time-consuming, it primarily meets the requirements.

Bibliography

Bridson R., 2016. *Fluid simulation for computer graphics.* CRC Press.

Cho C.-K., Lee B. and Kim S., 01 2018. Dual-mesh characteristics for particle-mesh methods for the simulation of convection-dominated flows. *SIAM Journal on Scientific Computing*, **40**, A1763–A1783.

Fedkiw R., Stam J. and Jensen H. W., 2001. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 15–22.

Fu C., Guo Q., Gast T., Jiang C. and Teran J., 2017. A polynomial particle-in-cell method. *ACM Transactions on Graphics*, **36**(6), 1–12.

Jiang C., Schroeder C., Selle A., Teran J. and Alexey S., 2015. The affine particle-in-cell method. *ACM Transactions on Graphics*, **34**(4), 1–10.

Kim B., Liu Y., Llamas I. and Rossignac J., 2005. Flowfixer: Using bfecc for fluid simulation. *Eurographics AssociationPostfach 2926GoslarGermany.*

Mullen P., Crane K., Pavlov D., Tong Y. and Desbrun M., 2009. Energy-preserving integrators for fluid animation. *ACM Transactions on Graphics*, **28**(3), 1–8.

Qu Z., Zhang X., Gao M., Jiang C. and Chen B., 2019. Efficient and conservative fluids with bidirectional mapping. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2019)*, **38**(4). (*Joint First Authors).

Ralston A., 1962. Runge-kutta methods with minimum error bounds. *Mathematics of Computation*, **16** (80), 431–431.

Selle A., Fedkiw R., Kim B., Liu Y. and Rossignac J., 2007. An unconditionally stable maccormack method. *Journal of Scientific Computing*, **35**(2-3), 350–371.

Stam J., 1999. Stable fluids.

WeiBmann S. and Pinkall U., 2010. Filament-based smoke with vortex shedding and variational reconnection. *ACM Transactions on Graphics*, **29**(4), 1–12.

Zehnder J., Narain R. and Thomaszewski B., 2018. An advection-reflection solver for detail-preserving fluid simulation. *ACM Transactions on Graphics*, **37**(4), 1–8.

Zhang X., Bridson R. and Greif C., 2015. Restoring the missing vorticity in advection-projection fluid solvers. *ACM Transactions on Graphics*, **34**(4), 1–8.

Zhu Y. and Bridson R., 2005. Animating sand as a fluid. *ACM Transactions on Graphics*, **24**(3), 965–972.

Appendix A

Code and Video

All relevant codes are seen in Github

Showreel can be seen in vimeo