FAST AND SCALABLE METHODS FOR THE SIMULATION OF INCOMPRESSIBLE FLOW

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Michael Anthony Lentine
May 2012

Abstract

This dissertation presents efficient and scalable algorithms for the simulation of incompressible fluids. Physical simulation of fluids is one of the most interesting and challenging problems because of the amount of small scale details that realistic fluids exhibit. Although a large number of high fidelity simulations can be obtained with existing techniques, the resolution that these techniques can obtain is limited by the amount of computational power available. The simulation of incompressible flow has two main aspects: advection and projection. This thesis addresses performance and scalability issues related to both aspects and demonstrates a number of algorithms that work to massively reduce the computational cost of simulations.

In the first chapters we concentrate on improving the performance and scalability of fluid simulations by investigating new conservative advection methods based off the established semi-Lagrangian method. Applying a conservative limiter to the typical semi-Lagrangian interpolation step can guarantee that the amount of the quantity being advected (e.g. mass, momentum, volume, etc.) does not increase. In addition, a new second step can be utilized that forward advects any of the quantity that was not accounted for in the typical semi-Lagrangian advection. Using this new conservative semi-Lagrangian method, mass and momentum can be conservatively advected in order to improve visual fidelity of smoke simulations at large time steps. In addition to conserving momentum during advection, the commonly used vorticity confinement turbulence model can be modified to exactly conserve momentum as well. It is shown that this new method is amenable to efficient smoke simulation with one time step per frame, whereas the traditional non-conservative semi-Lagrangian

method experiences serious artifacts when run with these large time steps, especially when object interaction is considered.

This method is then extended for water simulation when taking large time steps where, in contrast to smoke, an extrapolated velocity field is required. Inaccuracies with the extrapolated velocity field are alleviated by not using it when it is incorrect, which is determined via conservative advection of a color function that adds forwardly advected semi-Lagrangian rays to maintain conservation when mass is lost. This method is then coupled to the more visually appealing particle levelset method to obtain both a visually appealing and accurate method for simulating water at large time steps.

In the final chapters we discuss improving the performance and scalability of the projection step through the use of faster methods for the pressure solve. This technique coarsens the Eulerian fluid grid during the pressure solve, allowing for a fast implicit update but still maintaining the resolution obtained with a large grid. This allows simulations to run at a fraction of the cost of existing techniques ($\sim$60x faster) while still providing the fine scale structure and details obtained with a full projection. This algorithm scales well to very large grids and large numbers of processors, allowing for high fidelity simulations that would otherwise be intractable.

Acknowledgments

First, I would like to thank my advisor, Ron Fedkiw, for his support and encouragement. He has been an inspiration over the years not only about how to do research but about how to deal with life. I learned a lot from him in the last few years and would not have been able to achieve what I have without his assistance and mentorship. I would especially like to thank him for his commitment to working on high impact and interesting problems even when others thought they were too hard or not worth doing. This made my experience not only extremely fun as I got to work on the things that I wanted to, but it also made me proud of what I was able to accomplish.

I would also like to thank my fellow group members who I have spent a lot of time with over the years. I would like to thank my coauthors Andrew Selle, Jon Gretarsson, Craig Schroeder, Avi Robinson-Mosher, Wen Zheng, Mridul Aanjaneya, Matthew Cong, and Saket Patkar. I would especially like to thank Andy for introducing me to the lab and providing a lot of useful guidance when I was first starting. It was during this time that I realized that I was going to have a lot of fun during my time at Stanford. I would also like to thank the other members of Ron's lab who helped make graduate school an extremely enjoyable experience. These are Tamar Shinar, Jonathan Su, Nipun Kwatra, Elliot English, Linhai Qui, Rahul Sheth, Yue Yu, and Bo Zhu. I would also like to acknowledge the CURIS students who helped our lab out with a number of projects that I worked on including Joyce Pan, Jessica Liu, and Adele Xu.

I also had the privilege of consulting at Industrial Light + Magic would like to thank

those who I worked closely with including Brice Criswell, Frank Losasso Petterson, Kim Libreri, Nick Rasmussen, Zoran Kacic-Alesic, Rick Hankins, Ian Sachs, Chris Twigg, Cliff Ramshaw, Stephen Bowline, and Jeff Smith who all enriched my experience at Industrial Light + Magic. I would especially like to thank Brice and Kim for providing me with unique opportunities that I would not have otherwise had.

I would also like to thank my reading committee consisting of Oussama Khatib and Joseph Teran as well as my other committee members Pat Hanrahan and George Papanicolaou for their time and effort.

I would also like to acknowledge Jessica Hodgins for introducing me to computer graphics and working with me as an undergraduate researcher. Without her help and guidance I wouldn't be where I am today.

Finally, I would like to thank my family and friends especially my parents Anthony and Nancy Lentine for their support as well Yanjing Li for always being there for me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Physical simulation of fluids is one of the most interesting and challenging problems in computer graphics because of the large amount of small scale details that can be obtained. There are two primary approaches to fluid simulation: Lagrangian and Eulerian. Eulerian or grid based methods have been used by many authors including [29, 105, 25] and work by laying down a grid fixed in space in which fluid will move through. Each cell in this grid then tracks various quantities such as smoke density, surface distance and velocity in order to simulate fluid flow. Lagrangian methods in contrast represent the fluid as a set of particles moving through space (see e.g. [96, 17, 85]). Each particle then stores the equivalent quantities such as velocity. However, particles do not store the representation of the surface and thus additional computational costs are incurred in order to keep track of a surface and the re-mesh. Furthermore, it is difficult to enforce incompressibility which leads to either weakly compressible formulations which become computationally expensive near the incompressible limit or the use of grid based methods in order to satisfy incompressibility. For these reasons this thesis focuses on grid based methods for fluid simulation.

In order to achieve small scale details, relatively fine scale resolutions must be used. However, the increased in computational cost often makes this intractable. In fact,

when refining the resolution of a grid by a factor of two, the computational expense increases by a factor of sixteen, two for each spatial dimension and another two for time. This thesis aims to alleviate these costs by presenting fast and scalable algorithms for the simulation of incompressible flow. The simulation of incompressible flow has two main aspects: advection and projection. Advection involves moving the fluid through space and is relatively cheap compared to projection (typically taking around 10% of the total simulation time) and the projection step enforces the incompressibility condition of the fluid and is quite expensive. However, while projection can run at arbitrarily large time steps without any loss of accuracy, the advection step has an imposed time step restriction based on the fluid velocity. Typically this is enforced by restricting the time step so that information cannot travel more than one grid cell in any given time step. While it is possible with existing methods to achieve slightly higher time step sizes, taking large time steps of more than a few grid cells results in large errors in accuracy.

In order to address the loss of accuracy, chapter 2 presents a new conservative advection method based off the established semi-Lagrangian method [105]. Applying a conservative limiter to the typical semi-Lagrangian interpolation step can guarantee that the amount of the quantity being advected (e.g. mass, momentum, volume, etc.) does not increase. In addition, a new second step can be utilized that forward advects any of the quantity that was not accounted for in the typical semi-Lagrangian advection. Chapter 3 then introduces a number of modifications to this algorithm which are used to conservatively advect mass and momentum in order to improve the visual quality of smoke when using large time step sizes. In addition to conserving momentum during advection, the commonly used vorticity confinement turbulence model can be modified to exactly conserve momentum as well. It is shown that this new method is amenable to efficient smoke simulation with one time step per frame, whereas the traditional non-conservative semi-Lagrangian method experiences serious artifacts when run with these large time steps, especially when object interaction is considered. Chapter 4 then extends this method for water simulation when taking large time steps where, in contrast to smoke, an extrapolated velocity field is required.

Inaccuracies with the extrapolated velocity field are alleviated by not using it when it is incorrect, which is determined via conservative advection of a color function that adds forwardly advected semi-Lagrangian rays to maintain conservation when mass is lost. This method is then coupled to the more visually appealing particle levelset method to obtain both a visually appealing and accurate method for simulating water at large time steps.

This thesis also addresses projection which takes the majority of the time in a simulation but can be run with arbitrarily large time steps without loss of accuracy. Chapter 5 introduces a new method which improves the performance of the pressure solve needed to make the velocity field divergence free. This technique uses two separate grids for simulation, the main fluid grid which is used for advection and a coarsened version of the fluid grid which is used as part of the new projection method. Before the pressure is solved for, the velocities are mapped from the main simulation grid to the coarse projection grid. The pressure is then solved for on the coarse grid which updates the coarse velocities. Subsequently, the velocities are then mapped onto the simulation grid creating a incompressible fine scale velocity field. This allows simulations to run at a fraction of the cost of existing techniques ($\sim$60x faster) while still providing the fine scale structure and details obtained with a full projection. This algorithm scales well to very large grids and large numbers of processors, allowing for high fidelity simulations that would otherwise be intractable.

Chapter 2

Unconditionally Stable Conservative Advection

Semi-Lagrangian methods have been around for some time, dating back at least to [15]. Researchers have worked to increase their accuracy, and these schemes have gained newfound interest with the recent widespread use of adaptive grids where the CFL(CourantFriedrichsLewy)-based time step restriction of the smallest cell can be overwhelming. Since these schemes are based on characteristic tracing and interpolation, they do not readily lend themselves to a fully conservative implementation. However, in this chapter, we propose a novel technique that applies a conservative limiter to the typical semi-Lagrangian interpolation step in order to guarantee that the amount of the conservative quantity does not increase during this advection. In addition, we propose a new second step that forward advects any of the conserved quantity that was not accounted for in the typical semi-Lagrangian advection. We show that this new scheme can be used to conserve both mass and momentum for incompressible flows. For incompressible flows, we further explore properly conserving kinetic energy during the advection step, but note that the divergence free projection results in a velocity field which is inconsistent with conservation of kinetic energy

(even for inviscid flows where it should be conserved). While this thesis is mostly concerned with incompressible flows, we also demonstrate that our new method can be applied to compressible flow problems. For compressible flows, we rely on a recently proposed splitting technique that eliminates the acoustic CFL time step restriction via an incompressible-style pressure solve. Then our new method can be applied to conservatively advect mass, momentum and total energy in order to exactly conserve these quantities, and remove the remaining time step restriction based on fluid velocity that the original scheme still had.

2.1 Introduction

The idea of applying the method of characteristics to advect quantities forward in time dates back at least as far as [15] and has gained popularity in many areas, such as atmospheric sciences [107]. Although the simplest schemes trace back along straight line characteristics and use low order interpolation to estimate the data, one can trace back arbitrarily high order curved characteristics and use arbitrarily high order interpolation, see for example [81]. The simplicity of these schemes makes them quite useful for adaptive grids and other data structures, see for example [21, 69, 68, 108, 43]. Recently, authors have considered using semi-Lagrangian methods as building blocks in other schemes, for example [46, 47, 19] showed that the second order accurate BFECC method of [18] can be made unconditionally stable using the first order accurate semi-Lagrangian method as a building block. In addition, [101] showed that the original scheme of MacCormack [75] can be made unconditionally stable in a similar way. A notable feature of the semi-Lagrangian method is that it relieves the time step restriction. This is part of the reason why it has received such interest from the atmospheric sciences community [61, 64, 62, 125], as well as the compressible flow community [63, 98] where the acoustic time step restrictions can be severe. We refer the reader to a particularly interesting body of work that considers a number of methods for making semi-Lagrangian schemes conservative, considering one spatial dimension, multiple spatial dimensions with splitting, multiple

spatial dimensions without splitting, and even obtaining conservation from a non-conservative form [112, 119, 86, 118, 111].

Intuitively, the idea behind a fully conservative semi-Lagrangian scheme is simply to advect the conserved quantities along characteristic paths in a way that is careful to respect conservation. Many numerical methods are based on this principle, for example SPH methods push around chunks of mass, momentum and energy assigned to particles, and have been used to solve both compressible and incompressible flows, including flows with shock waves, see for example [33, 74, 53, 52, 121, 16, 14, 66, 71, 93, 34]. In fact, the idea of pushing around conserved quantities is the basis for volume of fluid methods, which attempt to conserve volume (see for example [92, 97, 65, 39, 67, 126]). In addition, ALE methods also push material around using a moving grid, and some of those methods use a background grid along with a two-step procedure where the material is first advected forward on a moving grid, and then remapped or redistributed to the background mesh in a conservative fashion, see for example [40, 76, 72, 73, 13]. Obviously this idea of pushing around mass in a conservative way respecting propagational characteristics for the sake of consistency has received quite a bit of attention.

Notably, our method is quite simple, both conceptually and as far as implementation is concerned—it requires only a small modification to a standard semi-Lagrangian scheme and utilizes most of the functionality already present. The standard semi-Lagrangian method updates the value at a grid point by tracing a possibly curved characteristic backwards in time to find its point of origin, interpolating the surrounding data to that point, and placing the result of the interpolation at the original grid point. In this manner, a grid point is updated with a linear combination of data from other points. One can view this as placing some fraction of the data from other grid points at this point, and then consider what this means from the point of conservation of this data. Considering the grid as a whole, each grid point traces back some characteristic and obtains some fraction of data stored at other grid points. One can see that the scheme is not conservative, since certain grid points contribute to multiple interpolations and the sum of all the weights from that grid point to all the points

where the interpolations were performed can be larger than one. This means that the data contained at the grid point has been over-depleted, violating conservation. Similarly, some grid points may not be asked for any of their data at all, or the sum of the inquisitive weights may be less than one. This also violates conservation, in the sense that data has been left at that grid point and not advected forward. Of course, we could simply account for this data by leaving it at that grid point, but then the scheme would be inconsistent as this data needs to be advected forward. We note that it is trivial to cure both of these pathologies in the semi-Lagrangian scheme by simply ensuring that the sum of the interpolation weights from every point adds to one, and that any data that wasn't advected forward is pushed forward in our second semi-Lagrangian step.

To summarize, we make the following modifications to a standard semi-Lagrangian method. Each grid point is thought of as a control volume, containing a certain amount of conserved quantity similar to any other conservation law solver. We trace back the potentially curved semi-Lagrangian rays in the usual manner, perform the interpolation in the usual manner, but add the additional step of recording all the interpolation weights for every grid point so that we may check whether or not they are equal to one. Our first correction requires sweeping through the grid, identifying any grid node which has been asked for more information than it contains (i.e. sum of the weights is greater than one), and subsequently scaling down these weights such that their sum is exactly equal to one. Then these corrected weights are used in place of the standard weights in the semi-Lagrangian advection scheme. At this point, the standard semi-Lagrangian scheme is completed, however as mentioned above, we have not advected all of the conserved quantity forward in time. Thus, for each grid point whose weights sum to less than one, we need to advect the remaining conserved data forward in time for consistency. This is done via a second application of the semi-Lagrangian method starting at that grid point and tracing a potentially curved characteristic forward in time, to see where it lands (exactly opposite of the standard semi-Lagrangian method). The remaining data at that point is placed at its new location, however this new location will not lie at a grid point but will instead

lie inside some grid cell. We distribute the remaining data to the surrounding grid points by noting that the transpose of an interpolation operator is a conservative distribution operator. That is, we simply calculate the interpolation weights at the new point, just as one would in a standard semi-Lagrangian interpolation, and use those weights to determine how much of the quantity is distributed to each of the surrounding points. Notably, the building blocks for the second step already exist in most implementations, only the tracing of a characteristic and the computation of interpolation weights are needed in the algorithm.

In this chapter, we consider the application of our method to both incompressible and compressible flow. As far as mass is concerned, treating a variable-density incompressible flow and the density equation in compressible flow requires only straightforward application of the method. As far as momentum and energy are concerned, we take an approach which is similar for both incompressible and compressible flows. In particular we use the method of [54] in order to solve the compressible flow equations in a way that requires an advection step followed by a pressure solve similar to incompressible flow, but which contains an identity term since pressure is based on the time dependent pressure evolution equation. Thus, both methods consist of a conservative advection step, followed by an implicit solve for the pressure, and a final pressure correction step. In the case of incompressible flow, our new semi-Lagrangian method can be used to exactly conserve the momentum of the fluid, and if the pressure correction is viewed as a flux, then one can conserve momentum in that step as well. In addition, we show how to account for stationary walls and potentially moving solid object boundaries. The treatment for compressible flow is similar, except mass, momentum and energy are conservatively advected with our semi-Lagrangian scheme before the pressure is solved for and the correction is applied. We show how to apply the pressure correction in such a way so that both the momentum and total energy are conserved, especially near solid walls and object boundaries. Finally, we note that a conservation style equation can be formulated for the kinetic energy of an incompressible flow. This equation is similar to that for compressible flow, with total energy replaced by kinetic energy along with the appearance of a source term

for losses due to viscosity. Although our scheme can be used to conservatively advect kinetic energy, and accounting for the viscous source term is straight-forward, the pressure projection step is inconsistent with the conservation of kinetic energy and therefore the resulting divergence-free velocity field disagrees with that predicted by kinetic energy conservation. We provide some analysis of this along with quantitative results.

2.2 Conservative semi-Lagrangian method

We begin by discussing the standard semi-Lagrangian method as applied in the simplest case of a passively advected scalar ϕ, in a velocity field $\vec{u}$,

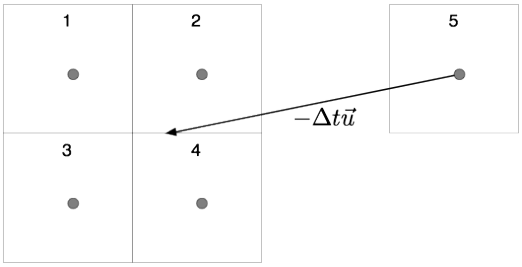
$$\phi_t + \vec{u} \cdot \nabla \phi = 0. \tag{2.1}$$

Combining this equation with conservation of mass, $\rho_t + \nabla \cdot (\rho \vec{u}) = 0$, leads to the conservative form of the same equation:

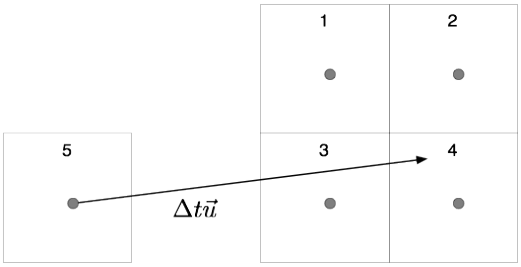
$$(\rho\phi)_t + \nabla \cdot (\phi \rho \vec{u}) = 0. \tag{2.2}$$

For the sake of exposition, we define $\hat{\phi} = \rho\phi$ as the conserved quantity; this allows us to interchangeably talk about ϕ, the passively advected scalar, and $\hat{\phi}$, the conserved quantity. For each grid point $\vec{x}_j$, the semi-Lagrangian method would trace a potentially curved characteristic ray backward in time to some position $\vec{x}$, and use an interpolation kernel to obtain a value of $\hat{\phi}$ at $\vec{x}$. This value is then used as $\hat{\phi}(\vec{x}_j, t^{n+1})$. The first order accurate case is illustrated by Figure 2.1(a), where a straight line characteristic is traced backward in time from cell 5 to find $\vec{x}$ in-between cells 1, 2, 3 and 4. In equation form, this is given by

$$\hat{\phi}(\vec{x}_j, t^{n+1}) = \hat{\phi}(\vec{x}, t^n) = \sum_i w_{ij} \hat{\phi}(\vec{x}_i, t^n), \tag{2.3}$$



(a) Cell 5 casts a ray backward, $-\Delta t\vec{u}$, which lands between cells 1, 2, 3 and 4. Using a standard bilinear interpolation scheme, the weights are calculated to be $w_{15} = .125$, $w_{25} = .375$, $w_{35} = .125$, $w_{45} = .375$.



(b) Cell 5 casts a ray forward, $\Delta t\vec{u}$, which lands between cells 1, 2, 3 and 4. We again use standard bilinear interpolation, giving forward-cast weights $f_{51} = .06$, $f_{52} = .24$, $f_{53} = .14$, $f_{54} = .56$.

Figure 2.1: Standard semi-Lagrangian advection schemes cast rays either forward or backward along characteristic lines in order to determine time t^{n+1} values at cell centers. We take advantage of this in our scheme, making use of the computed weights w_{ij} and f_{ij} as appropriate. The notation w_{ij} and f_{ij} denote the contribution that cell i gives to cell j over a time step.

where w_{ij} are interpolation weights such that $\vec{x} = \sum_i w_{ij}\vec{x}_i$. Dimension-by-dimension linear interpolation yields a first order method. Notably, $\sum_j w_{ij} = 1$ for any consistent interpolation operator, regardless of the size of the stencil or order of accuracy.

After updating $\hat{\phi}$ at every grid point, we can then define the total contribution from cell i to the time t^{n+1} data as $\sigma_i = \sum_i w_{ij}$, noting that this is not expected to sum to 1 due to numerical truncation errors. In fact, since $\hat{\phi}$ is conserved as shown in Equation (2.2), in order to exactly conserve data during the semi-Lagrangian update, σ_i should be exactly 1. Fixing this is the key idea of our numerical method. This is accomplished by visiting each donor grid cell i, examining σ_i, and scaling down the weights w_{ij} to $\hat{w}_{ij} = w_{ij}/\sigma_i$ when $\sigma_i \geq 1$, which guarantees explicitly that we do not artificially create $\hat{\phi}$.

Next we treat the cells for which $\sigma_i < 1$. At these cells we apply a second pass of the standard semi-Lagrangian scheme, casting rays forward, as illustrated for a first order accurate method in Figure 2.1(b), yielding forward-cast weights f_{ij}. Noting that the transpose of an interpolation operator is a conservative distribution operator, we use these weights f_{ij} to distribute the remaining $\hat{\phi}$, i.e. $(1 - \sigma_i)\hat{\phi}_i$, to the cells j used

to perform the interpolation. This can be seen as incrementing the unclamped w_{ij} weights from the first step by an amount equal to $(1-\sigma_i)f_{ij}$, so that the final weights are $\hat{w}_{ij} = w_{ij} + (1-\sigma_i)f_{ij}$. Our update is then given as

$$\hat{\phi}_j^{n+1} = \sum_i \hat{w}_{ij}\hat{\phi}(x_i, t^n). \tag{2.4}$$

At the end of our two applications of the standard semi-Lagrangian steps, we now have modified weights $\hat{w}_{ij}$ to satisfy $\sum_i \hat{w}_{ij} = 1$. That is, every cell on the grid contributes exactly everything it has at time t^n to the time t^{n+1} solution along the characteristic lines which pass through the cell.

To summarize, when $\sigma_i \geq 1$, we clamp the w_{ij} to obtain $\hat{w}_{ij} = w_{ij}/\sigma_i$; using these new $\hat{w}_{ij}$ weights leads to $\sigma_i = 1$. Otherwise if $\sigma_i < 1$, we forward advect the non-advected data at each grid point and use it's placement to calculate the new weights $\hat{w}_{ij}$ which also lead to $\sigma_i = 1$. We note that in the $\sigma_i \geq 1$ case, one could also forward advect and interpolate. In this fashion, one would be advecting negative material to cancel out the excess of positive material that was advected by the first semi-Lagrangian step. However, when this negative material is place at surrounding grid nodes using the f_{ij} weights, it is possible for the target grid node x_j to lose more of the conserved quantity than it originally had. Thus, for now, we only consider the method of clamping even though it seems to limit the method to first order accuracy.

2.2.1 Boundary conditions

In the application of our method, we consider a number of different boundary conditions. For open boundaries, inflow and outflow are treated by adding and filling the appropriate number of ghost cells. For inflow boundary conditions, rays which extend out of the domain are treated in the standard semi-Lagrangian fashion, and the amount of material donated from ghost cells to points interior to the domain is considered to be our inflow. One could modify the inflow scheme to not simply perform semi-Lagrangian interpolation but instead conservatively advect the sum of the

ghost cell data, however this requires careful accounting since, as ghost cells, some of these are not solved for.

Unlike the standard scheme where only interior points need to be updated, our outflow boundaries require evaluation of ghost nodes in the numerical scheme to ensure that they withdraw the correct amount of $\hat{\phi}$ from the interior of the grid. Moreover one needs to ensure that enough ghost cells are updated, such that the information is correctly withdrawn from the interior of the domain. After clamping, one also needs to consistently advect data from interior nodes to ghost cells when $\sigma_i < 1$.

Throughout the chapter, we measure our conservation error at time t^n using the following equation:

$$\text{Error}(t^n) = \Sigma\hat{\phi}(t^n) - \left[\Sigma\hat{\phi}(t^0) + \Sigma_{in} - \Sigma_{out}\right], \tag{2.5}$$

where the first term on the right-hand side represents the current amount of $\hat{\phi}$ on the grid and the second term represents the initial amount. These should only vary through inflow and outflow which are represented by the third and fourth terms. When updating $\hat{\phi}_i^{n+1}$ from $\hat{\phi}^n$, if a semi-Lagrangian ray reaches back to ghost cells and pulls information into the domain, then we track that for the Σ_{in} term. If information is transported from the interior of our grid to the ghost cells, we track that for the Σ_{out} term. That is, $\hat{w}_{ij}$'s which contribute to a ghost cell j are accounted for in Σ_{out}. There is rich literature on treating inflow and outflow boundary conditions for fluid flows, and we imagine that many variations of our method could be designed in such a way that is consistent with our treatment of the interior of the domain. However, we found this sufficient for our examples.

Near solid walls and moving object boundaries, one must be careful not to interpolate across or into the wall or object. All rays that are cast are done in a collision-aware manner, stopping any rays early if they would pass through the interface, similar to the computational geometry approach detailed in the computer graphics literature (see e.g. [37]). Typically, when performing interpolation as in [37] we use information from the solid, such as its velocity. However, that would transfer information

from the solid to the fluid, for example, during momentum advection one would be interpolating momentum from the solid. This is non-physical since advection should not transport conserved quantities across material interfaces. Any transmission of momentum from the solid to the fluid should instead occur when considering the acoustic characteristics, for example when solving for the pressure (which we consider later). Thus, for our scheme we simply set $w_{ij} = 0$ for any interpolation point which is not visible. At this point one could consider scaling up the remaining weights to get interpolation weights such that $\Sigma_i w_{ij} = 1$, although we have not experimented numerically with this option. Finally, in the forward-casting step of the scheme, in order to guarantee conservation we set $f_{ij} = 0$ if cell j is not visible from the interpolation point, and the remaining weights are then scaled up to account for the missing material.

2.2.2 Interpolation

For our new conservative semi-Lagrangian approach we require an interpolation scheme to determine the weights. A simple method uses linear weights between the nearest points as shown in Figure 2.1. While this works rather well for conserving energy as well as converging to the correct solution, the interpolation error can be reduced through the use of higher order interpolation. Consider for example quadratic interpolation. If our point of interest x lies between cells i and $i+1$, then we have available two valid quadratic functions: a left-biased one which interpolates across the range (x_{i-1}, x_i, x_{i+1}), and a right-biased one that interpolates across the range (x_i, x_{i+1}, x_{i+2}). The left-biased quadratic produces weights for an interpolated point x as:

$$\alpha_{i-1,L} = \frac{\bar{x}(\bar{x}-1)}{2}, \qquad \alpha_{i,L} = 1 - \bar{x} - \bar{x}(\bar{x}-1), \qquad \alpha_{i+1,L} = \bar{x} + \frac{\bar{x}(\bar{x}-1)}{2}$$

while the right-biased quadratic produces weights for an interpolated point x as:

$$\alpha_{i,R} = 1 - \bar{x} + \frac{\bar{x}(\bar{x}-1)}{2}, \qquad \alpha_{i+1,R} = \bar{x} - \bar{x}(\bar{x}-1), \qquad \alpha_{i+2,R} = \frac{\bar{x}(\bar{x}-1)}{2}$$

where $\bar{x} = (x - x_i)/\Delta x$. While sufficient for a standard semi-Lagrangian scheme, these interpolations will produce negative weights on the outlying cells ($i-1$ for the left-biased one, $i+2$ for the right-biased one) when $x_i < x < x_{i+1}$. To alleviate these negative weights we instead always zero the weight on the outlying cell and push the missing contribution inward. That is, for the left-biased polynomial the weights would be

$$\tilde{\alpha}_{i-1,L} = 0, \qquad \tilde{\alpha}_{i,L} = 1 - \bar{x} - \frac{\bar{x}(\bar{x}-1)}{2}\left[2 - \frac{\hat{\phi}_{i-1}}{\hat{\phi}_i}\right], \qquad \tilde{\alpha}_{i+1,L} = \bar{x} + \frac{\bar{x}(\bar{x}-1)}{2}$$

Similarly, for the right-biased polynomial the weights would be

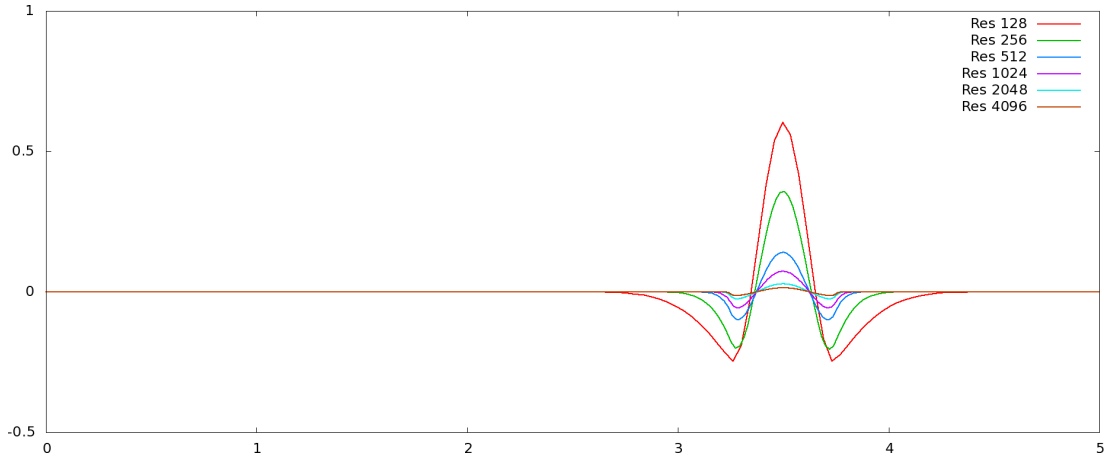
$$\tilde{\alpha}_{i,R} = 1 - \bar{x} + \frac{\bar{x}(\bar{x}-1)}{2}, \qquad \tilde{\alpha}_{i+1,R} = \bar{x} - \frac{\bar{x}(\bar{x}-1)}{2}\left[2 - \frac{\hat{\phi}_{i+2}}{\hat{\phi}_{i+1}}\right], \qquad \tilde{\alpha}_{i+2,R} = 0.$$

This preserves the value given by the higher-order interpolation scheme and significantly reduces the likelihood of a negative weight. Note that both of the quadratic interpolations provide a second order correction to a linear interpolation. If we take both of these interpolation schemes and average them, we get the weights that we use in the quadratic version of our scheme:

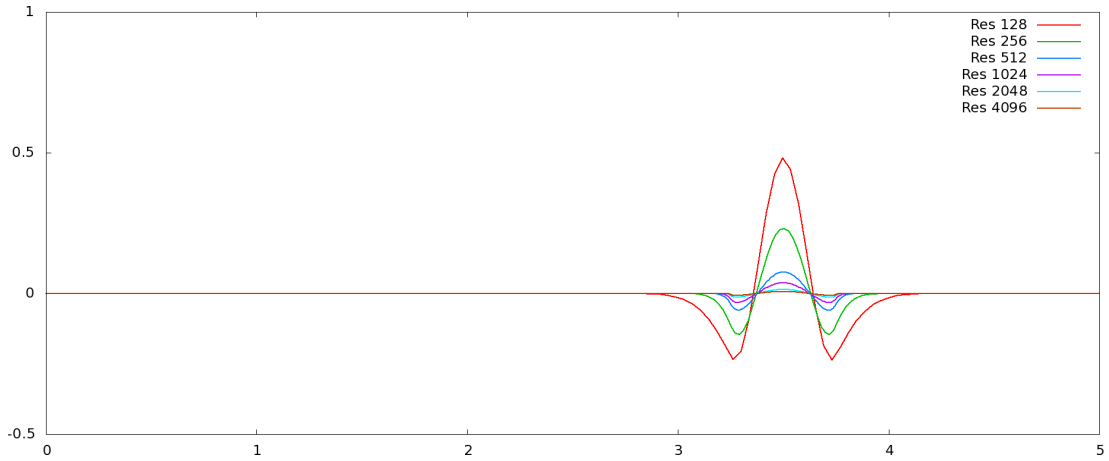
$$\alpha_i = 1 - \bar{x} - \frac{\bar{x}(\bar{x}-1)}{4}\left(1 - \frac{\hat{\phi}_{i-1}}{\hat{\phi}_i}\right), \qquad \alpha_{i+1} = \bar{x} - \frac{\bar{x}(\bar{x}-1)}{4}\left(1 - \frac{\hat{\phi}_{i+2}}{\hat{\phi}_{i+1}}\right).$$

Using these modified weights we can then perform our semi-Lagrangian steps as discussed earlier. Figures 2.2, 2.3 and 2.4 demonstrate the significant error improvement by using this interpolation scheme. Note that in these figures, using second-order Runge-Kutta to trace characteristic lines gives no numerical differences, as the first order approximation is already exact.

Whereas arbitrarily high order characteristics can be traced using our semi-Lagrangian scheme, it is this negativity in the interpolation weights which so far has restricted our method to first order accuracy. Negative weights are not entirely detrimental, and in fact the quadratic version of our scheme admits that to some lesser degree. If

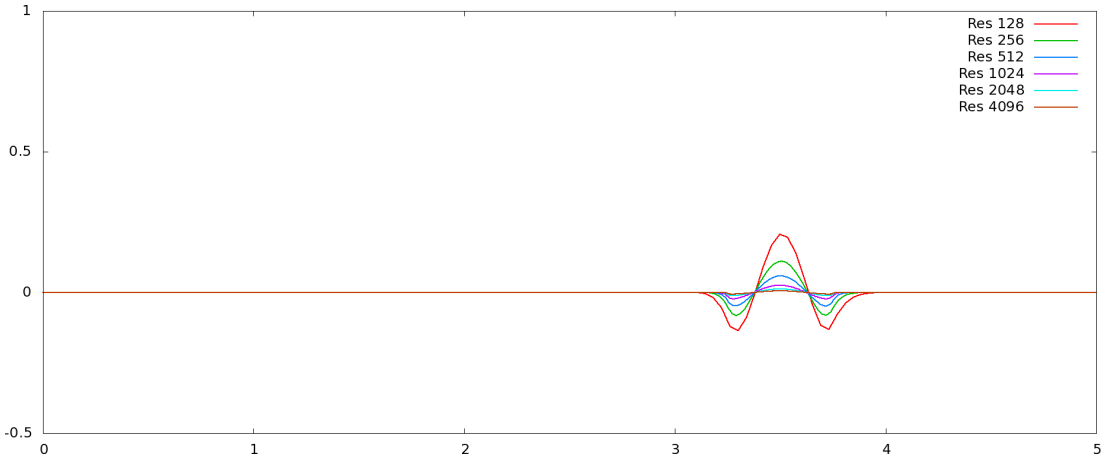


(a) Linear interpolation.

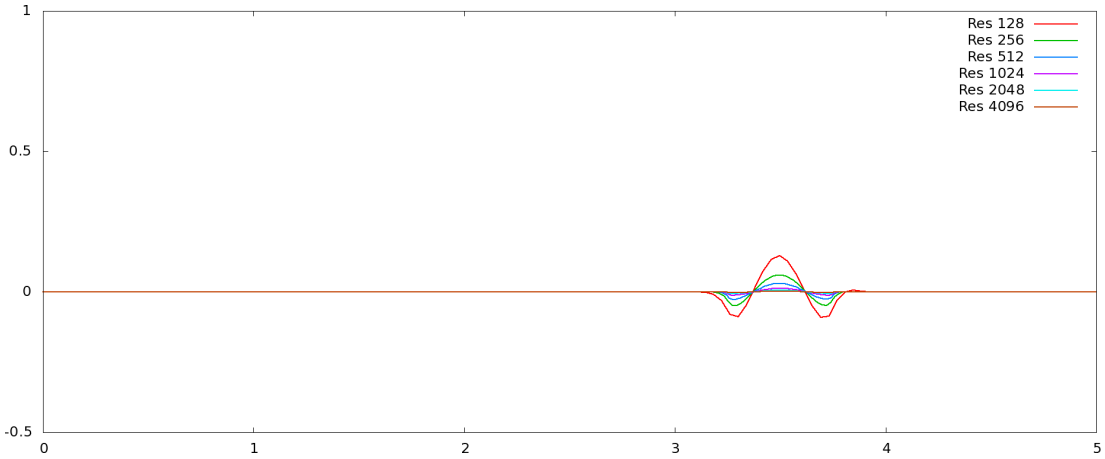


(b) Quadratic interpolation.

Figure 2.2: Error curve for the advected sine-wave "bump" in a constant velocity field $u = 1$ at time $t = 3s$ for linear and quadratic interpolation using our proposed conservative semi-Lagrangian advection scheme, run with a CFL number .9. Using a higher-order interpolation scheme gives noticeably reduced error; for example at $\Delta x = 5/256$ the peak error for the linear interpolation scheme is .111, while the quadratic interpolation scheme has a peak error of .060.

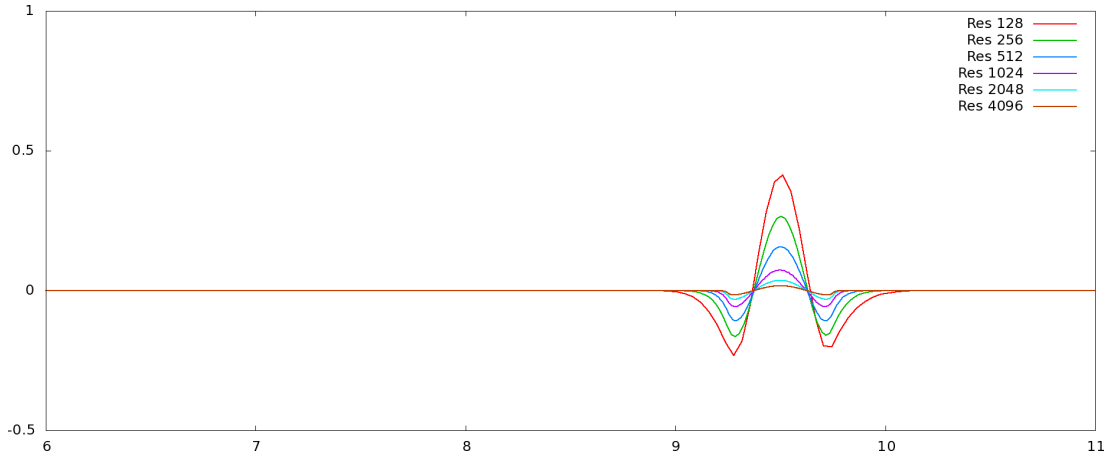


(a) Linear interpolation.

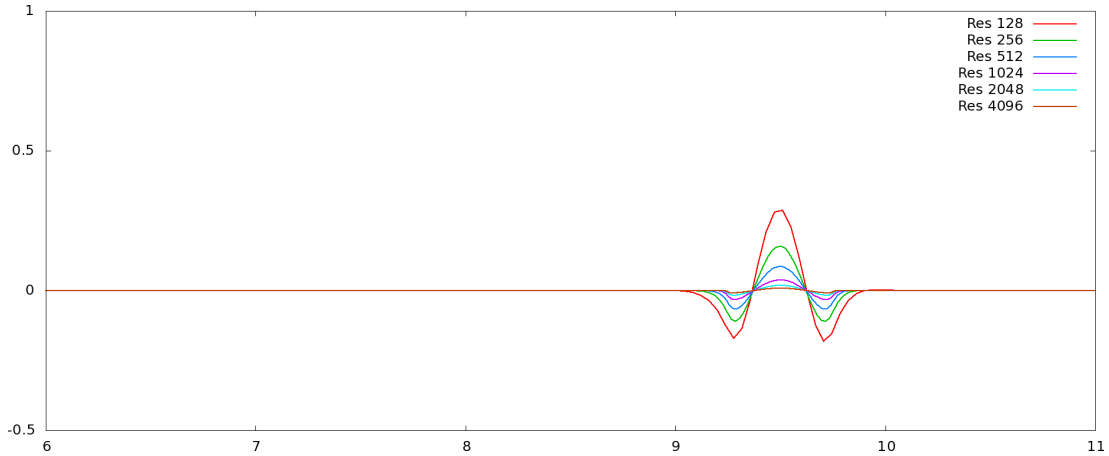


(b) Quadratic interpolation.

Figure 2.3: Error curve for the advected sine-wave "bump" in a constant velocity field $u = 1$ at time $t = 3s$ for linear and quadratic interpolation using our proposed conservative semi-Lagrangian advection scheme, run with a CFL number 2.9. As there are no temporal errors (as any semi-Lagrangian ray exactly captures the characteristic curve), all errors are due to the application of an interpolation scheme. The larger CFL number permits time steps almost three times larger than those taken for Figure 2.2, and so the error introduced by the interpolation scheme are significantly smaller.



(a) Linear interpolation.



(b) Quadratic interpolation.

Figure 2.4: Error curve for the advected sine-wave "bump" in a constant velocity field $u = 1$ at time $t = 9s$ for linear and quadratic interpolation using our proposed conservative semi-Lagrangian advection scheme, run with a CFL number 2.9. As there are no temporal errors (as any semi-Lagrangian ray exactly captures the characteristic curve), all errors are due to the application of an interpolation scheme. As such the number of interpolations needed decreases as the CFL number increases, and the error goes down proportionally. If we run the same simulation with a larger CFL number and a proportionally longer period of time, the errors become similar (see Figure 2.2).

the sum of all the weights at a grid node is equal to some $\epsilon < 0$ at a grid node, then we simply forward-advect $1 + \epsilon$ amount of material. The problem is that a typical quadratic interpolation scheme can have rather large positive weights balancing out rather large negative weights on the side of the interval from which two points are used, and this seems to lead to difficulties. Our process of merging the weights to form $\tilde{\alpha}$ from α tends to cancel out these large positive and negative values making the result more reasonable. Of course one can guarantee that the weights never become negative by simply using standard multi-linear interpolation.

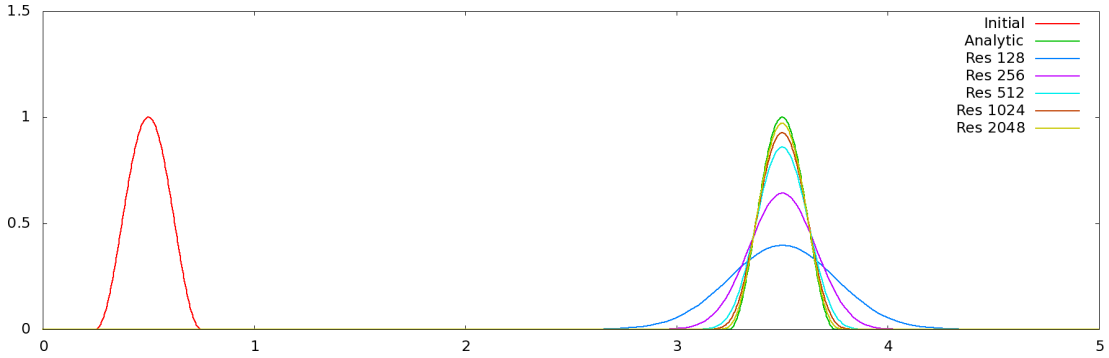
It may be possible to make a second order accurate scheme using only order first order accurate interpolation stencils, as was done in the modified MacCormack scheme of [101] and the modified BFECC scheme of [19]. Another interesting approach would be to apply a second order non-conservative correction to a full conservative first order accurate scheme.

2.2.3 Examples

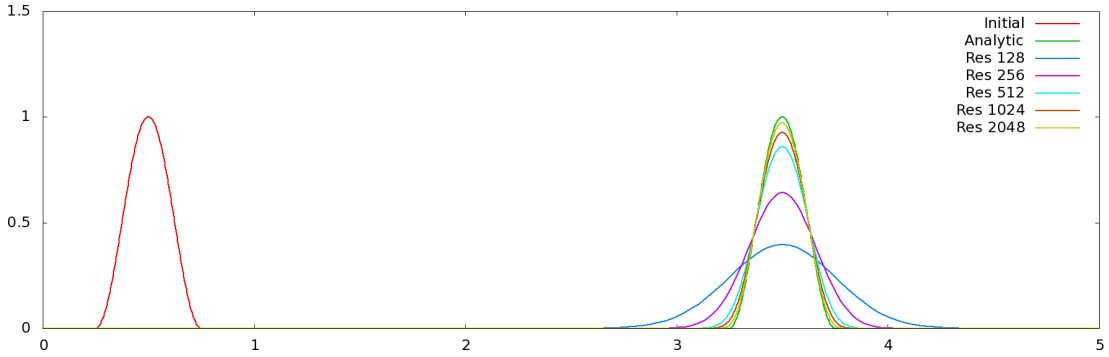
In order to demonstrate the conservation properties of our scheme, we consider an advected sine-wave "bump" using a constant velocity field. That is,

$$\hat{\phi}(x,0) = \begin{cases} \frac{1}{2}\left(1 + \sin\left(4\pi * (x - \frac{3}{8})\right)\right) & .25 \leq x \leq .75 \\ 0 & \text{else} \end{cases} \tag{2.6}$$

with $u = 1$. The problem is discretized over the domain $[0, 5]$, and we solve Equation (2.2) with a CFL number of .9. In Figure 2.5(a), we show the solution as computed by a standard non-conservative semi-Lagrangian advection, while Figure 2.5(b) shows the solution computed by our new scheme. As we expect, the solutions of the two methods agree and both converge to the analytic solution. Figure 2.2 shows a comparison between using linear and quadratic interpolation in our method. Figure 2.3 shows the same comparison using a CFL number of 2.9 instead of 0.9. Note that the errors are much smaller since approximately three times fewer time steps



(a) Standard semi-Lagrangian advection.



(b) Conservative semi-Lagrangian advection.

Figure 2.5: A sine-wave "bump" is advected through a uniform velocity field. Shown is the solution at time $t = 3s$. We apply the first order version of both the standard semi-Lagrangian advection, as well we our proposed conservative semi-Lagrangian advection scheme.

(and thus interpolations) are needed. In Figure 2.4 we run this simulation three times longer with a CFL of 2.9 showing errors more commensurate with Figure 2.2 as expected. We also demonstrate the order of convergence in Table 2.1 which shows that our algorithm gives first order convergence.

Figure 2.6 considers a square wave in the divergent velocity field $u = \sin(\pi x/5)$. Note the marked difference between the conservative and non-conservative method. Figure 3.7 shows dramatic loss of conservation in the standard semi-Lagrangian method as compared to the conservative version which maintains exact value up to round off error. Figure 2.7 shows a convergence analysis for the two schemes using a high resolution full conservative ENO method [104] as a ground truth. As is typical the

Coarse Res	Fine Res	Convergence Order
128	256	0.9384
256	512	1.2062
512	1024	1.1614
1024	2048	1.1260
2048	4096	0.9938

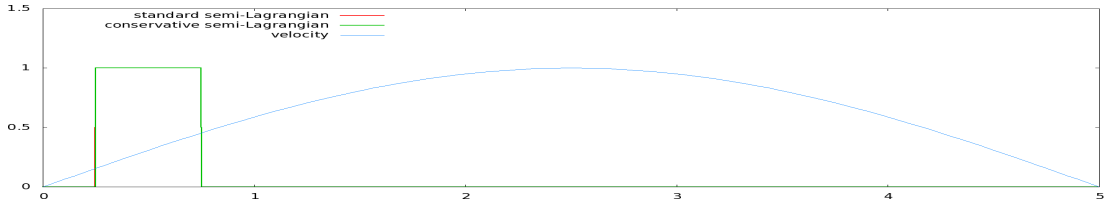
Table 2.1: Convergence order is computed by taking the $log_2(c_e/f_e)$ where c_e is the error in the coarse resolution simualtion and f_e is the error in the fine resoltuion simulation. The order is averaged over all relevant points.

non-conservative method converges to the wrong solution whereas our new method converges to the result obtained via ENO. The reason the non-conservative method converges to the wrong solution in this case is that it solves Equation (3.1) whereas our method solves Equation (2.2). In comparing these two equations, standard semi-lagrangian advection is missing the $\hat{\phi}(\nabla \cdot \vec{u})$ term.

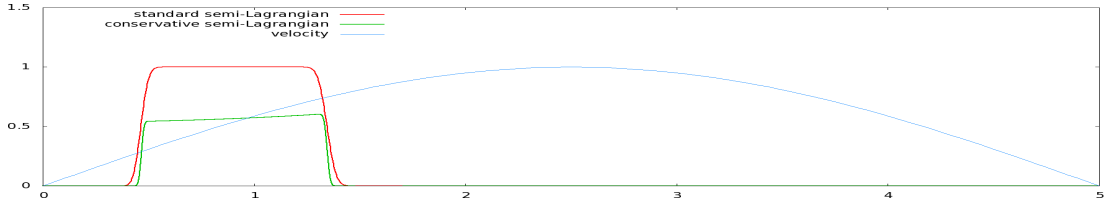
We also consider the Zalesak disc example, discussed in [124]. In this example a notched disk is advected through a velocity field specified by

$$u = (\pi/314)(50 - y)$$
$$v = (\pi/314)(x - 50)$$

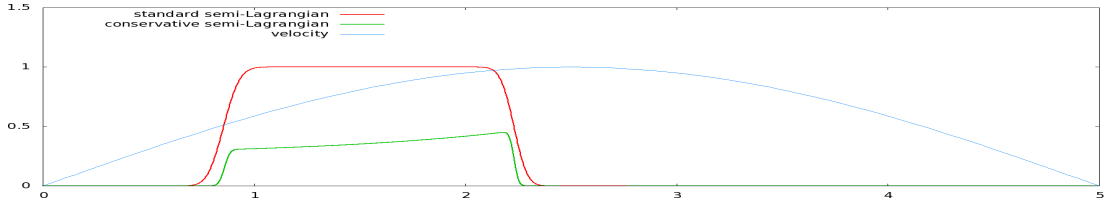
Shown in Figure 2.9 is the disk after one rotation, for a variety of resolutions. We also plot the total mass of the system as a function of time, in Figure 2.10; note that a standard semi-Lagrangian scheme fails to conserve the mass of the disk. The conservative semi-Lagrangian scheme conserves the mass of the disk up to roundoff error.



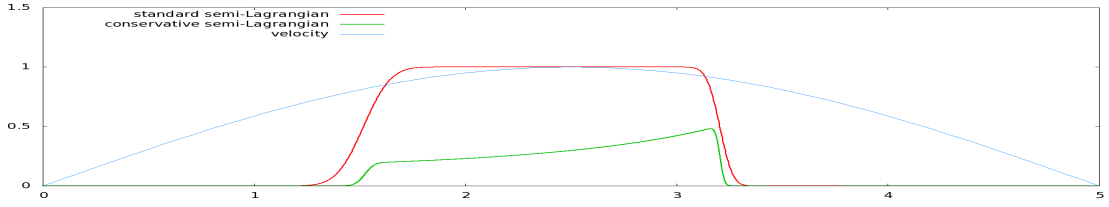
(a) $t = 0s$.



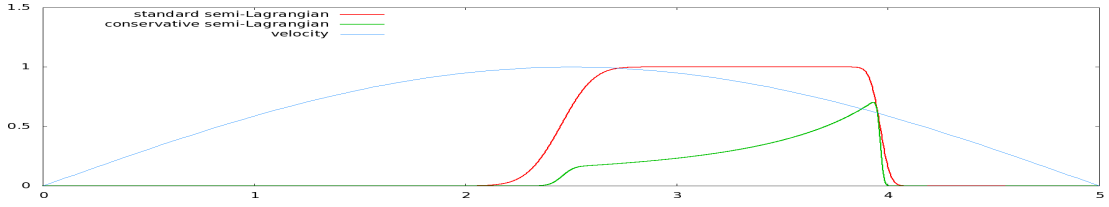
(b) $t = 1s$.



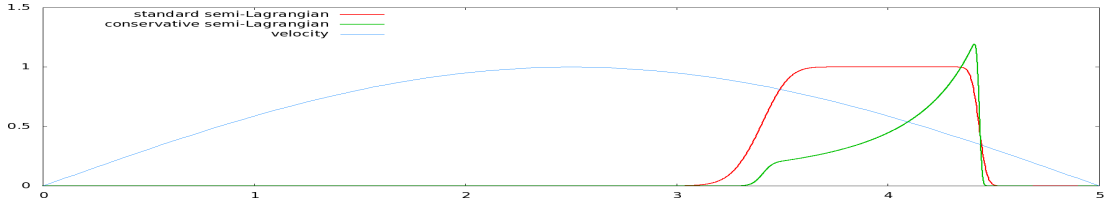
(c) $t = 2s$.



(d) $t = 3s$.

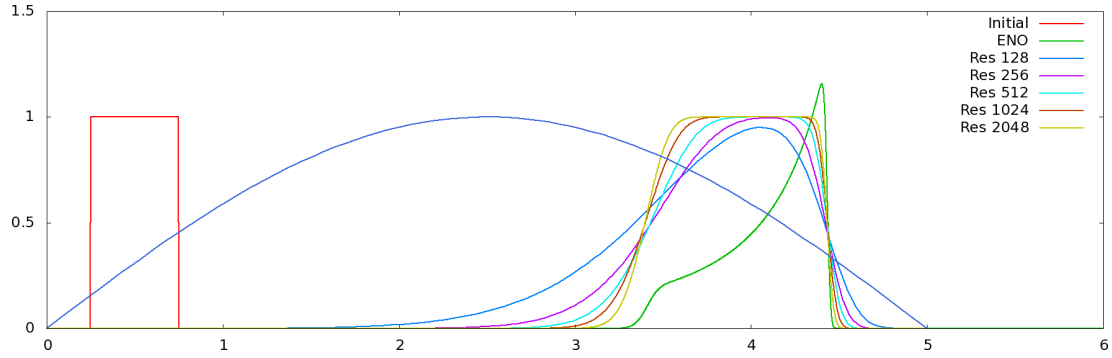


(e) $t = 4s$.

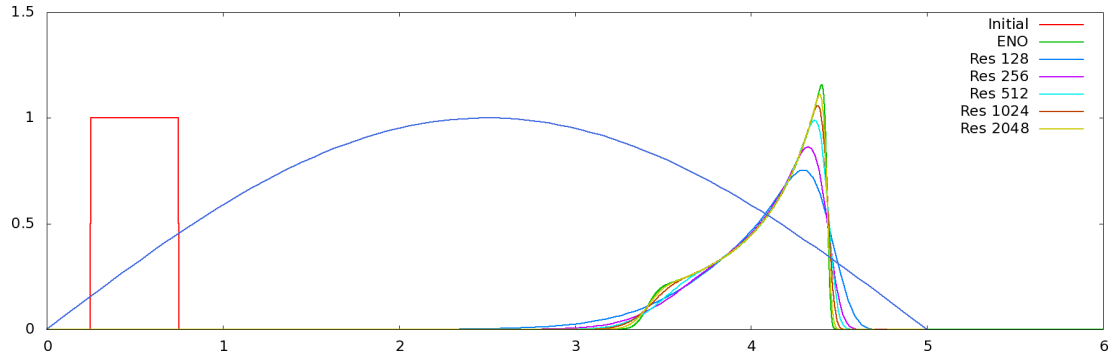


(f) $t = 5s$.

Figure 2.6: We consider the evolution of density in a velocity field that is specified by $u(x) = \sin\left(\pi \frac{x}{5}\right)$. In such a velocity field, the standard semi-Lagrangian approach fails to capture the rarefaction and converges to a non-physical solution. This simulation is run with $\Delta x = 5/8192$.



(a) Standard semi-Lagrangian advection.



(b) Conservative semi-Lagrangian advection.

Figure 2.7: A square wave that evolves with a divergent velocity field $u = \sin\left(\pi\frac{x}{5}\right)$. Shown is the solution at time $t = 3s$. We apply the first order version of both the standard semi-Lagrangian advection, as well as our proposed conservative semi-Lagrangian advection scheme. In this example, we see the standard semi-Lagrangian advection scheme converges to the wrong solution.

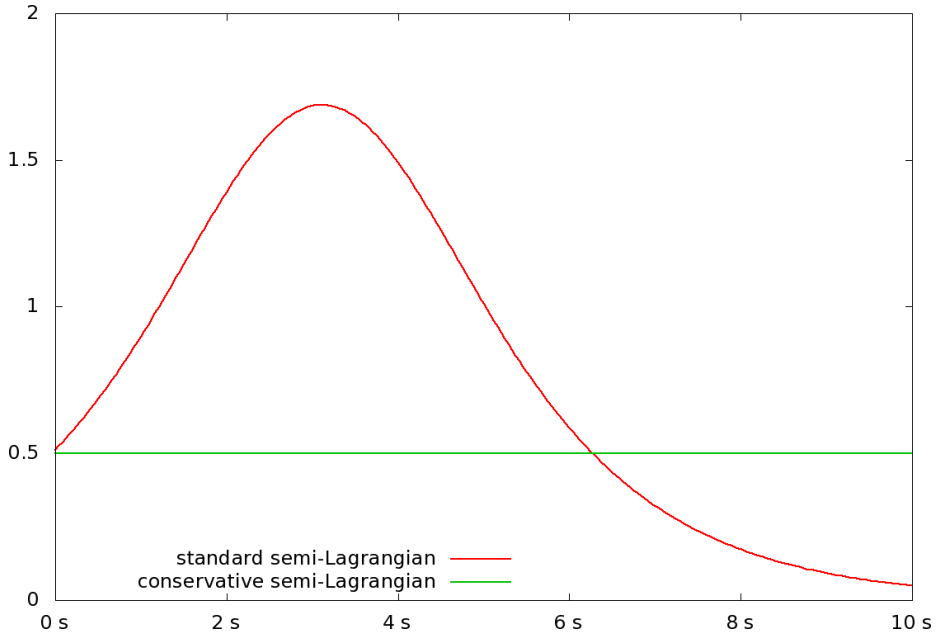


Figure 2.8: Shown is the time history of $\sum_i \Delta x \hat{\phi}_i$ for a square wave that is evolved through a divergent velocity field with $u = \sin\left(\pi \frac{x}{5}\right)$. Solutions for both the standard semi-Lagrangian advection scheme and our proposed conservative semi-Lagrangian advection scheme are shown at high-resolution with $\Delta x = 5/8192$.

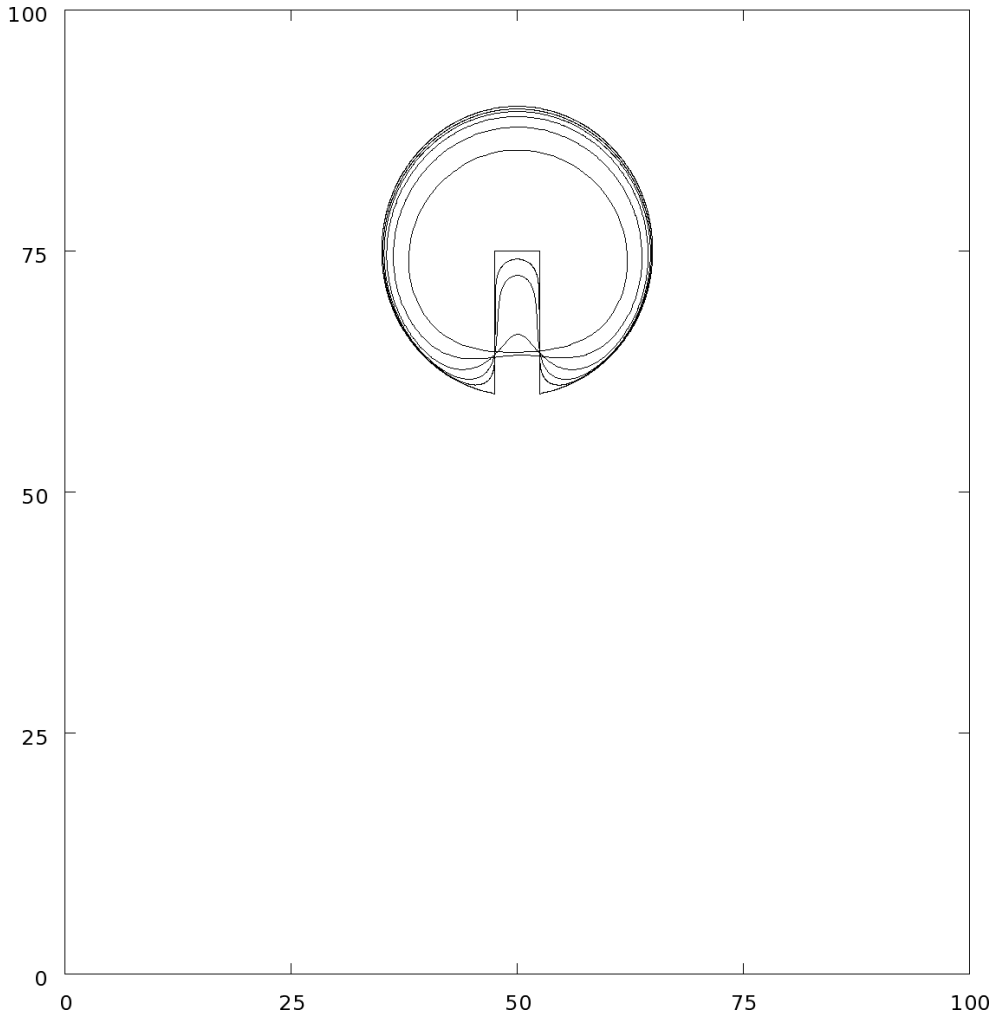


Figure 2.9: After one full rotation of the Zalesak disk [124] using our proposed conservative semi-Lagrangian advection scheme, for a variety of grid resolutions. Shown is the .5 isocontour for grid resolutions $\Delta x = 2^{-7}$, 2^{-8}, 2^{-9}, 2^{-10}, and 2^{-11}, in addition to the analytic solution. The mass of the disk is properly conserved using our method (this is verified in Figure 2.10), while the standard semi-Lagrangian advection scheme loses significant mass. In this light, our scheme can be thought of as the conservative advection of a smeared-out Heaviside color function.

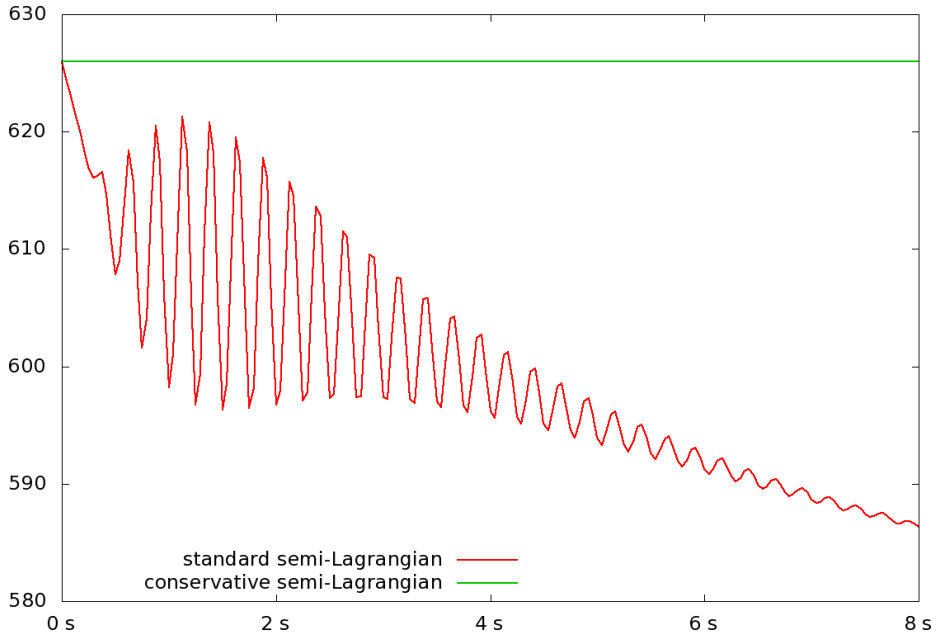


Figure 2.10: Shown is the time history of $\Sigma_i \Delta x \hat{\phi}_i + \Sigma_{out} - \Sigma_{in}$ for Zalesak Disk with $\Delta x = 2^{-7}$. Time history for the standard semi-Lagrangian advection scheme is shown in red, while our proposed conservative semi-Lagrangian advection scheme is shown in green.

2.3 Incompressible flow

We model incompressible flow using the viscous Navier-Stokes equations, given by

$$\begin{cases} \vec{u}_t + \vec{u} \cdot \nabla \vec{u} + \frac{\nabla p}{\rho} = \frac{1}{\rho} \nabla \cdot (\mu \nabla \vec{u}) \\ \nabla \cdot \vec{u} = 0 \end{cases} \tag{2.7}$$

where $\vec{u}$ is the fluid velocity, p is the pressure and μ is the coefficient of viscosity (which is taken to be constant). For the sake of illustration, we use a fairly simple time discretization scheme. First we account for the $\vec{u} \cdot \nabla \vec{u}$ term by advecting $\vec{u}^n$ forward in time using the incompressible velocity field $\vec{u}^n$ with a semi-Lagrangian advection scheme, giving an advected velocity $\vec{u}^\star$. This velocity field is projected and made incompressible by solving

$$\Delta t \nabla \cdot \frac{1}{\rho} \nabla p = \nabla \cdot \vec{u}^\star \tag{2.8}$$

to obtain a pressure, which is then applied via:

$$\vec{u}^{\star\star} = \vec{u}^\star - \frac{\Delta t}{\rho} \nabla p. \tag{2.9}$$

Viscous forces are next implicitly accounted for by solving

$$\tilde{\vec{u}}^{n+1} = \vec{u}^{\star\star} + \frac{\Delta t}{\rho} \nabla \cdot (\mu \nabla \tilde{\vec{u}}^{n+1}), \tag{2.10}$$

after which we project the flow field again by solving Equation (2.8) (replacing $\vec{u}^\star$ with $\tilde{\vec{u}}$), and then finally updating the flow field to time t^{n+1} via

$$\vec{u}^{n+1} = \tilde{\vec{u}}^{n+1} - \frac{\Delta t}{\rho} \nabla p. \tag{2.11}$$

A standard Marker and Cell (MAC, [38]) grid discretization is used, storing fluid velocity in a component-by-component fashion on cell faces. By treating the viscous forces implicitly, we alleviate the viscous time step restriction.

2.3.1 Momentum-conserving scheme

In order to derive a completely conservative scheme for the momentum, we reformulate the incompressible flow equations slightly. First, we multiply Equation (2.7) through by density, giving the following equations in two spatial dimensions:

$$\rho u_t + \rho u u_x + \rho v u_y + p_x = (\mu u_x)_x + (\mu u_y)_y, \tag{2.12}$$

$$\rho v_t + \rho v u_x + \rho v v_y + p_y = (\mu v_x)_x + (\mu v_y)_y. \tag{2.13}$$

Next, we make use of conservation of mass, given in two spatial dimensions as $\rho_t + (\rho u)_x + (\rho v)_y = 0$, noting that for incompressible flow this is identical to $\rho_t + u\rho_x + v\rho_y = 0$. If we combine this with the equations above, we can introduce the momentum $L_u = \rho u$, $L_v = \rho v$ and derive the conservation form of the incompressible flow equations as

$$(L_u)_t + (L_u u)_x + (L_u v)_y + p_x = (\mu u_x)_x + (\mu u_y)_y, \tag{2.14}$$

$$(L_v)_t + (L_v u)_x + (L_v v)_y + p_y = (\mu v_x)_x + (\mu v_y)_y. \tag{2.15}$$

For advection we solve $(L_u)_t + (L_u u)_x + (L_u v)_y = 0$ for $L_u^\star$ using our new conservative semi-Lagrangian scheme. Similarly, $(L_v)_t + (L_v u)_x + (L_v v)_y = 0$ is solved for $L_v^\star$. This small change in form of the equations yields an advection scheme which is robust to the numerical viscosity effects typically seen in a semi-Lagrangian advection solver.

We use the standard pressure update to compute the pressure, where the intermediate velocity field is computed as $u^\star = L_u^\star/\rho$ and $v^\star = L_v^\star/\rho$. Equation (2.9) and (2.12), (2.13), (2.14), (2.15) illustrate that the pressure already acts as a conservative momentum flux between fluid cells. For fluid cells which lie along the fluid-structure interface, pressure acts as a momentum flux from the fluid cell faces to the solid, and vice versa. Thus, after projection we can simply update our x-momentum as $L_u^{\star\star} = \rho u^{\star\star}$ and y-momentum as $L_v^{\star\star} = \rho v^{\star\star}$, after applying the correction defined in Equation (2.9) to the velocity field $\vec{u}^\star$.

The viscous terms are treated implicitly by solving $\frac{\rho\tilde{u}^{n+1} - \rho\tilde{u}^{\star\star}}{\Delta t} = (\mu\tilde{u}_x^{n+1})_x + (\mu\tilde{u}_y^{n+1})_y$

which for constant density and viscosity becomes

$$\tilde{L}_u^{n+1} = L_u^{\star\star} + \Delta t \mu \nabla^2 \tilde{u}^{n+1}, \tag{2.16}$$

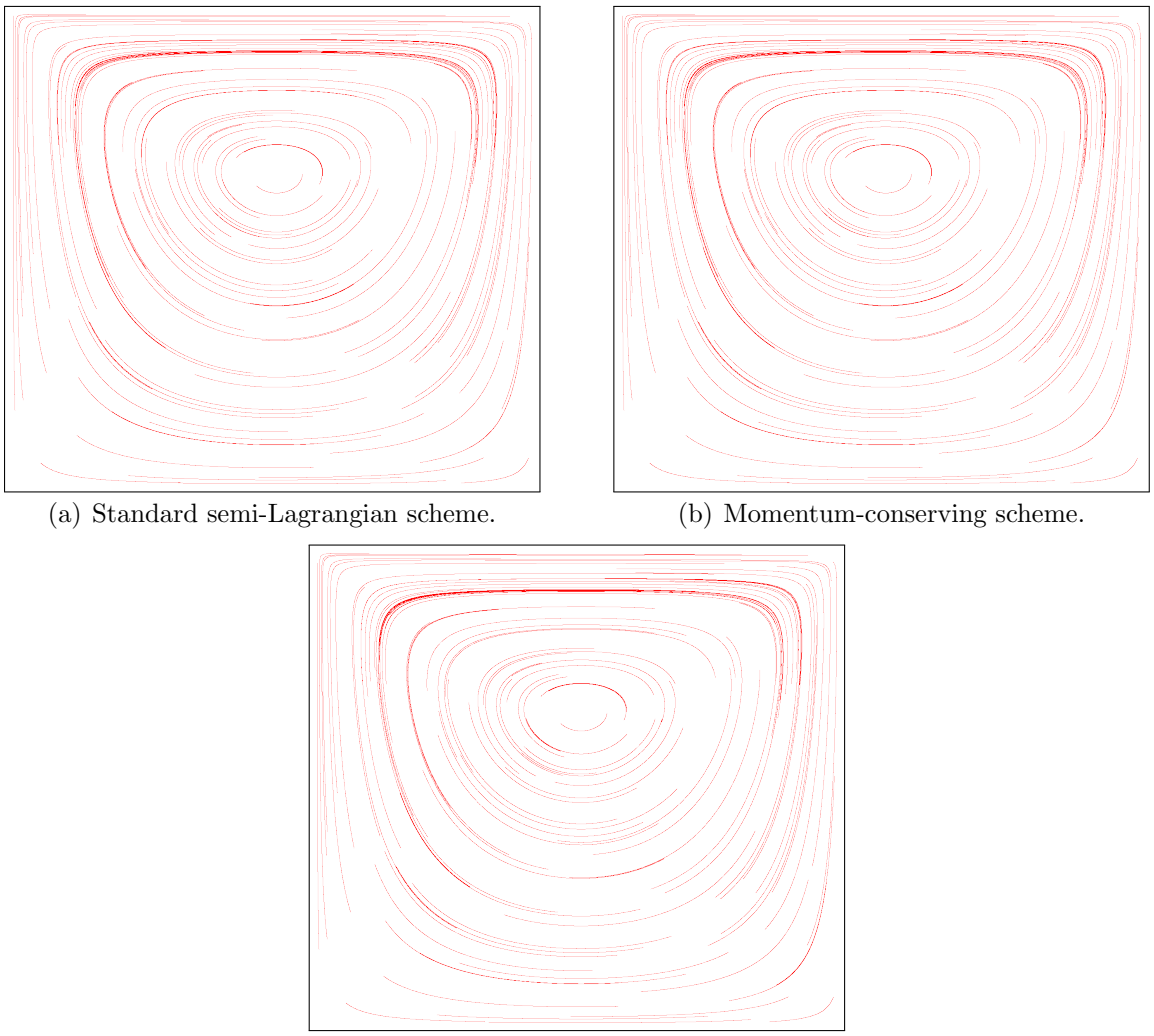
similar to Equation (2.10) above. In order to properly account for momentum transfer during the viscous stage, we are careful to view this viscosity update in a flux-based form. That is, μu_x acts as a momentum flux in between the MAC grid stencil locations of u values, and μu_y acts as a momentum flux in between MAC grid u stencil locations in the other direction. The same approach is used to update v velocities, using μv_x and μv_y as momentum fluxes between MAC grid v stencil locations. This gives

$$\tilde{L}_v^{n+1} = L_v^{\star\star} + \Delta t \mu \nabla^2 \tilde{v}^{n+1}. \tag{2.17}$$

These intermediate quantities are again projected by solving Equation (2.8) (replacing $\vec{u}^{\star}$ with $\tilde{\vec{u}}^{n+1}$). The time t^{n+1} velocity field is computed using Equation (2.11), and momentum is updated as $L_u^{n+1} = \rho u^{n+1}$ and $L_v^{n+1} = \rho v^{n+1}$.

2.3.2 Examples

We consider a cavity with high viscous forces that is driven by a flat, horizontal velocity profile with magnitude 1m/s on the top boundary of the domain. All walls in the domain are closed, the computational domain is $1m \times 1m$ with $\Delta x = .01$, and a driving flow moving at speed 1 m/s. Density is 1, and the viscous forces are determined by $\mu = 100\ Pa \cdot s$. Viscosity causes a vortex to form in the cavity, which quickly settles to steady-state. The resulting steady-state solutions are shown in Figure 2.11 for the standard semi-Lagrangian advection scheme and our momentum-conserving semi-Lagrangian advection scheme. Examining the pressure along the internal boundary, it is interesting to note that both schemes produce 0 net force acting on the solid boundary (i.e. $\sum_{\partial\Omega} p d\vec{A} = 0$), but the magnitude of the force isn't (i.e. $\sum_{\partial\Omega} |p| \neq 0$), suggesting that we properly capture linear momentum but angular momentum remains an issue.



(a) Standard semi-Lagrangian scheme.

(b) Momentum-conserving scheme.

(c) Kinetic energy-conserving scheme.

Figure 2.11: Streamlines for the driven cavity example using standard semi-Lagrangian advection, our proposed momentum-converging method, and our proposed kinetic energy-conserving method. All simulations are run with $\Delta x = 2^{-7}$.

Next we consider the simple case of flow past a sphere with closed walls on the top and bottom of the domain, inflow velocity with magnitude 2 m/s from the left side of the domain and an open outflow boundary on the right side of the domain with $p = 0$. For this example we used a domain of $(0, 0) \times (2, 1)$ (in meters), no viscosity and a grid size defined by $\Delta x = .01$. The solution at time $t = 9s$ is shown in Figure 2.12, using our proposed momentum-conserving scheme. The results of a standard semi-Lagrangian

scheme are qualitatively (but not quantitatively) similar as expected.

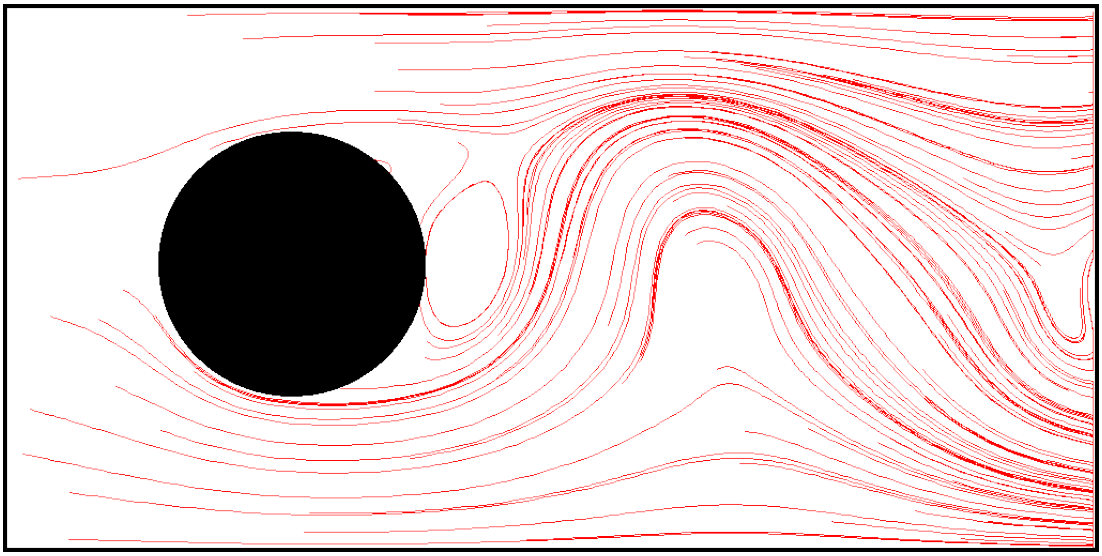


Figure 2.12: Stream-line visualization of flow past a sphere.

We also carry out a detailed study of the momentum, for both our scheme and the standard semi-Lagrangian scheme. The bottom two lines in Figure 2.13 show the cumulative momentum advected into and out of our of the domain for the semi-Lagrangian scheme, while the middle two lines show these same quantities for our momentum-conserving scheme. Since the flow is divergence free, one would generally expect these lines to be commensurate, however, due to numerical errors in interpolation there is some drift, which accumulates as the simulation carries forward. As pointed out above, the pressure acts as a conservative flux between fluid velocity degrees of freedom. Along solid wall boundaries, such as the top and bottom of the domain and around the sphere, the pressure can be scaled by the cell face size and Δt to give an impulse, suggesting that it represents a momentum-preserving collision between the solid and the fluid. However, since these walls are stationary, i.e. they have infinite mass, they remove momentum from the flow. If we sum the momentum lost along the walls we obtain the bottom two lines shown in Figure 2.14 for the semi-Lagrangian scheme and our momentum-conserving scheme respectively. Momentum is also introduced into the domain via pressure at the inflow boundary condition, where an upstream pressure profile is used to maintain a constant inflow

velocity. The gains due to inflow are shown in Figure 2.14 for the semi-Lagrangian method on the top curve, and the second curve from the top is for our momentum-conserving method. Note that since $p = 0$ at the outflow boundary, no momentum is lost. Figure 2.15 shows the result if we sum the previous graphs accounting for all the momentum advected into and out of the domain as well as the pressure fluxes at the walls and inflow. The bottom line, which is 0 to numerical roundoff, shows that our momentum-conserving scheme does indeed conserve momentum during the semi-Lagrangian advection step (which is the only term not accounted for in the previous graphs). In contrast, the standard semi-Lagrangian scheme gains momentum during the advection step. Although we did not compute the momentum gained by carefully looking at that step, we can accurately compute it by accounting for all the remaining terms and seeing what is left over.

2.4 Treating kinetic energy

It is interesting to consider incompressible flow from the standpoint of kinetic energy. Although kinetic energy is not conserved for a viscous fluid, it is conserved for an inviscid fluid. Moreover, it should be conserved during the semi-Lagrangian advection stage, even though it typically is not. We begin by deriving the time derivative of $K = \frac{1}{2}\rho\vec{u}\cdot\vec{u}$ as

$$\begin{aligned} K_t &= \frac{1}{2}\left(\rho\vec{u}\cdot\vec{u}\right)_t = \frac{1}{2}\vec{u}\cdot\vec{u}\rho_t + \rho\vec{u}\cdot\vec{u}_t \\ &= \frac{1}{2}\vec{u}\cdot\vec{u}\left(-\vec{u}\cdot\nabla\rho\right) + \vec{u}\cdot\left(\nabla\cdot\tau - \rho\vec{u}\cdot\nabla\vec{u} - \nabla p\right). \end{aligned}$$

We take advantage of Equation (2.7) from above, and observe that

$$\begin{aligned} \frac{1}{2}(\vec{u}\cdot\vec{u})\vec{u}\cdot\nabla\rho + \rho\vec{u}\cdot(\vec{u}\cdot\nabla\vec{u}) &= \frac{1}{2}(\vec{u}\cdot\vec{u})\vec{u}\cdot\nabla\rho + \frac{1}{2}\rho\vec{u}\cdot\nabla\left[\vec{u}\cdot\vec{u}\right] \\ &= \frac{1}{2}\vec{u}\cdot\nabla\left[\rho\vec{u}\cdot\vec{u}\right] = \vec{u}\cdot\nabla K \\ &= \nabla\cdot(K\vec{u}). \end{aligned}$$

Note that we can freely add in $p\nabla \cdot \vec{u}$ and $K\nabla \cdot \vec{u}$, which are both analytically zero. This gives time evolution of kinetic energy in conservative form as

$$K_t + \nabla \cdot [(K + p)\vec{u}] = \vec{u} \cdot (\nabla \cdot \tau). \tag{2.18}$$

2.4.1 Advection

We compute and store kinetic energy on horizontal u faces as $K_u = \frac{1}{2}\rho u^2$, and at vertical v faces as $K_v = \frac{1}{2}\rho v^2$, and evolve them forward in time separately as they only couple together through pressure fluxes, similar to the advection of the velocity field.

For advection, we solve $(K_u)_t + (K_u u)_x + (K_u v)_y = 0$ for $K_u^\star$ and $(K_v)_t + (K_v u)_x + (K_v v)_y = 0$ for $K_v^\star$, using the time t^n velocity field $\vec{u}^n$. In doing so, we explicitly conserve the kinetic energy of the system during the advection step, which has the effect of relieving the artificial viscosity effects typically seen when using a standard semi-Lagrangian advection scheme.

Once we compute $K^\star$ advected quantities, we use these kinetic energies to determine the magnitudes of the intermediate fluid velocity field $\vec{u}^\star$. That is, $u^\star = \pm\sqrt{2K_u^\star/\rho}$ and $v^\star = \pm\sqrt{2K_v^\star/\rho}$. We also advect fluid velocities forward in time (using either the semi-Lagrangian scheme or the momentum-conserving scheme) and use the sign of the resulting velocity field to determine the sign of $u^\star$ and $v^\star$.

2.4.2 Projection

The modified $\vec{u}^\star$ values are used in Equation (2.8) to compute the fluid pressure. Unlike the momentum update, where the pressure itself acts as a momentum flux and the result of the projection does conserve momentum, for kinetic energy we not only don't have good values for the flux $p\vec{u}$ in Equation (2.18), but the resulting post-velocity projection does not have the same kinetic energy as the pre-projected

velocity. Indeed, the change in kinetic energy due to the projection is

$$\begin{aligned}\Delta K_u &= \frac{\rho}{2}\left((u^{\star\star})^2 - (u^{\star})^2\right) & \Delta K_v &= \frac{\rho}{2}\left((v^{\star\star})^2 - (v^{\star})^2\right)\\ &= \frac{\rho}{2}\left(u^{\star\star} + u^{\star}\right)\left(u^{\star\star} - u^{\star}\right) & &= \frac{\rho}{2}\left(v^{\star\star} + v^{\star}\right)\left(v^{\star\star} - u^{\star}\right)\\ &= \frac{\rho}{2}\left(u^{\star\star} + u^{\star}\right)\left(-\frac{\Delta t}{\rho}p_x\right) & &= \frac{\rho}{2}\left(v^{\star\star} + v^{\star}\right)\left(-\frac{\Delta t}{\rho}p_y\right)\\ &= -\Delta t\hat{u}\left(p_x\right). & &= -\Delta t\hat{v}\left(p_y\right).\end{aligned}$$

where $\hat{u} = \frac{u^{\star\star}+u^{\star}}{2}$ and $\hat{v} = \frac{v^{\star\star}+v^{\star}}{2}$, and we use Equation (2.9) to replace $(u^{\star\star} - u^{\star})$ and $(v^{\star\star} - v^{\star})$ terms respectively. Then ΔK_u and ΔK_v look like $\Delta t\hat{u}p_x$ and $\Delta t\hat{v}p_y$ respectively, overall accounting for the $\vec{u}\cdot\nabla p$ component of $\nabla\cdot(p\vec{u})$. Analytically we would expect this to be sufficient in an incompressible flow, as $p\nabla\cdot\vec{u} = 0$, but examining this update from the discrete standpoint we note that $\nabla\cdot\hat{\vec{u}} = \frac{1}{2}\nabla\cdot\vec{u}^{\star} + \frac{1}{2}\nabla\cdot\vec{u}^{\star\star} = \frac{1}{2}\nabla\cdot\vec{u}^{\star} \neq 0$ in general and some kinetic energy is lost. If we examine the sum of the terms of the update for cell faces $i-1/2$ and $i+1/2$,

$$\hat{u}_{i+1/2}\frac{p_{i+1}-p_i}{\Delta x} + \hat{u}_{i-1/2}\frac{p_i - p_{i-1}}{\Delta x},$$

we can rearrange terms slightly, giving

$$\frac{\hat{u}_{i+1/2}p_{i+1}}{\Delta x} - \boxed{p_i\frac{\hat{u}_{i+1/2}-\hat{u}_{i-1/2}}{\Delta x}} - \frac{\hat{u}_{i-1/2}p_{i-1}}{\Delta x},$$

where the boxed term, when summed over all spatial dimensions for cell i, gives a discrete approximation of $-p\nabla\cdot\hat{\vec{u}}$. That is, by performing an update using ΔK_u and ΔK_v, we are losing exactly this component of the flux. If we view each individual component of the boxed term, $p_i u_{i+1/2}/\Delta x$, these can be thought of as fluxes between cell face $i+1/2$ and cell center i; then the kinetic energy that has been lost in this update is precisely the kinetic energy that accumulates to a cell center, rather than being fully distributed to our degrees of freedom.

Various strategies can be taken to address this, such as taking the accumulated cell-centered kinetic energy and distributing it equally to all of the surrounding cell faces.

We plan on looking into this more in future work [57], but for now we accept the loss of kinetic energy due to projection and incorporate the change in kinetic energy by simply using ΔK_u and ΔK_v as computed above.

If we consider the pressure at a grid cell i and scale it up by the area of a cell face and Δt, we get the impulse $\hat{p}_i$ between dual-cells $i - 1/2$ and $i + 1/2$. In multiple spatial dimensions, this impulse couples together the orthogonal directions, involving every cell face incident to cell i. This impulse exchange can be thought of as a collision between neighboring dual cells. Along this line of reasoning, it is interesting to note that while collisions preserve momentum and total energy, they do not conserve kinetic energy unless the coefficient of restitution is 1. Typically in a collision kinetic energy is lost, and the collisions in this system—with one exception—are no different. In the special case where $(\nabla \cdot \vec{u}^\star)_i = 0$, then the multi-dimensional collision that occurs at cell i does indeed conserve kinetic energy, and can be thought of as a fully elastic collision with a coefficient of restitution equal to 1.

For the momentum update, the application of these collision-based impulses can be done in any order; that is, we can freely iterate over impulses, updating the momentum by applying impulses in a Gauss-Seidel manner. This is not the case for the energy update, as the application of one impulse changes the energy updated by all subsequent impulses due to the cross-terms which arise. If we let $\rho = \frac{m}{\Delta x \Delta y}$, then the update takes the form

$$\hat{u}^{new} = \hat{u}^{old} + \frac{\hat{p}_i}{m}, \tag{2.19}$$

where $\hat{p}_i$ is the impulse defined above. If we consider the change in kinetic energy

after these updates,

$$\begin{aligned}
\Delta KE &= \frac{1}{2}m(\hat{u}^{newer})^2 - \frac{1}{2}m(\hat{u}^{old})^2 \\
&= \frac{1}{2}m\left[\left(\hat{u}^{old} + \frac{\hat{p}_i}{m} - \frac{\hat{p}_{i+1}}{m}\right)^2 - (\hat{u}^{old})^2\right] \\
&= \frac{1}{2}m\left[(\hat{u}^{old})^2 + \hat{u}^{old}\left(\frac{\hat{p}_i}{m} - \frac{\hat{p}_{i+1}}{m}\right) + \left(\frac{\hat{p}_i}{m} - \frac{\hat{p}_{i+1}}{m}\right)^2 - (\hat{u}^{old})^2\right] \\
&= \hat{u}^{old}\left(\frac{\hat{p}_i - \hat{p}_{i+1}}{2}\right) + \frac{1}{2m}\left(\hat{p}_i^2 + \hat{p}_{i+1}^2\right) - \boxed{\frac{\hat{p}_i\hat{p}_{i+1}}{m}},
\end{aligned}$$

then the boxed term is the cross-term which arises from the sequential application of impulse updates to the fluid volume. Note that the result is the same regardless of which impulse $\hat{p}_i$ or $\hat{p}_{i+1}$ is applied first. However, one might misconstrue the gain in kinetic energy due to each impulse depending on the order in which they were applied.

2.4.3 Viscosity

After the projection in Equation (2.9) we compute the viscous forces via Equation (2.10) and then compute the kinetic energy as seen by the fluid velocity field:

$$\begin{aligned}
\Delta K &= \frac{\rho}{2}\left((\tilde{u}^{n+1})^2 - (u^{\star\star})^2\right) \\
&= \frac{\rho}{2}\left(u^{\star\star} + \tilde{u}^{n+1}\right)\left(\tilde{u}^{n+1} - u^{\star\star}\right) \\
&= \frac{\rho}{2}\left(u^{\star\star} + \tilde{u}^{n+1}\right)\left(\frac{\Delta t}{\rho}\nabla\cdot(\mu\nabla u^{\star\star})\right) \\
&= \Delta t\tilde{u}\nabla\cdot(\mu\nabla u^{\star\star}),
\end{aligned}$$

where $\tilde{u} = \frac{u^{\star\star}+\tilde{u}^{n+1}}{2}$ and we use Equation (2.10) to eliminate the $(\tilde{u}^{n+1} - u^{\star\star})$ term. This gives us $K^{\star\star} = \tilde{K}^{n+1} + \Delta K$, the loss of kinetic energy due to viscous effects (noting in the case of inviscid flow that $\Delta K = 0$ and $K^{\star\star} = \tilde{K}^{n+1} = K^{n+1}$). Once $K^{\star\star}$ is computed, it is projected again as discussed above.

2.4.4 Examples

We first reconsider the driven cavity case from Section 2.3.2 using our kinetic energy-conserving semi-Lagrangian advection scheme. For this simple case, we do not attempt to correct for the kinetic energy losses due to the inelastic collisions dictated by the $\hat{p}$ discussed above. That is, kinetic energy is lost during the projection step, even though we know how much is lost to each cell center, as adding this kinetic energy back into the flow field would lead to a divergent velocity field. The simple case of the driven cavity is shown in Figure 2.11, showing that the kinetic energy-conserving scheme compares well with the other two schemes. Unfortunately, for more interesting cases such as the one shown in Figure 2.12, the inability to create a divergence-free velocity field that is consistent with the kinetic energy posses an issue, and the results are qualitatively different.

In spite of that we carry out an analysis for the momentum and kinetic energy in all three schemes: the original semi-Lagrangian scheme, the momentum-conserving scheme, and the kinetic energy-conserving advection scheme, which correctly conserves kinetic energy during advection but fails to account for kinetic energy loses during projection. The reason for this quantitative analysis is to illustrate where the kinetic energy goes, in each of these schemes. We begin by considering the momentum. The top two lines in Figure 2.13 represent the momentum advected into and out of the domain across the inflow and outflow for the kinetic energy-conserving scheme. The middle two lines in Figure 2.14 account for the momentum fluxing through solid wall boundaries due to pressure, as well as the pressure flux at the inflow of the domain. For Figure 2.15, the top line is the sum that represents the momentum loss during advection. Note that this is rather large when compared to the other two schemes, in part because the advected velocity is not consistent with kinetic energy.

Finally, we consider the kinetic energy transfer of all three schemes. Figure 2.16 shows the kinetic energy advected into and out of the domain across inflow and outflow boundaries. Figure 2.17 shows the energy gained due to the pressure interacting with both the solid wall boundaries and pressure flux at the inflow boundary. Note

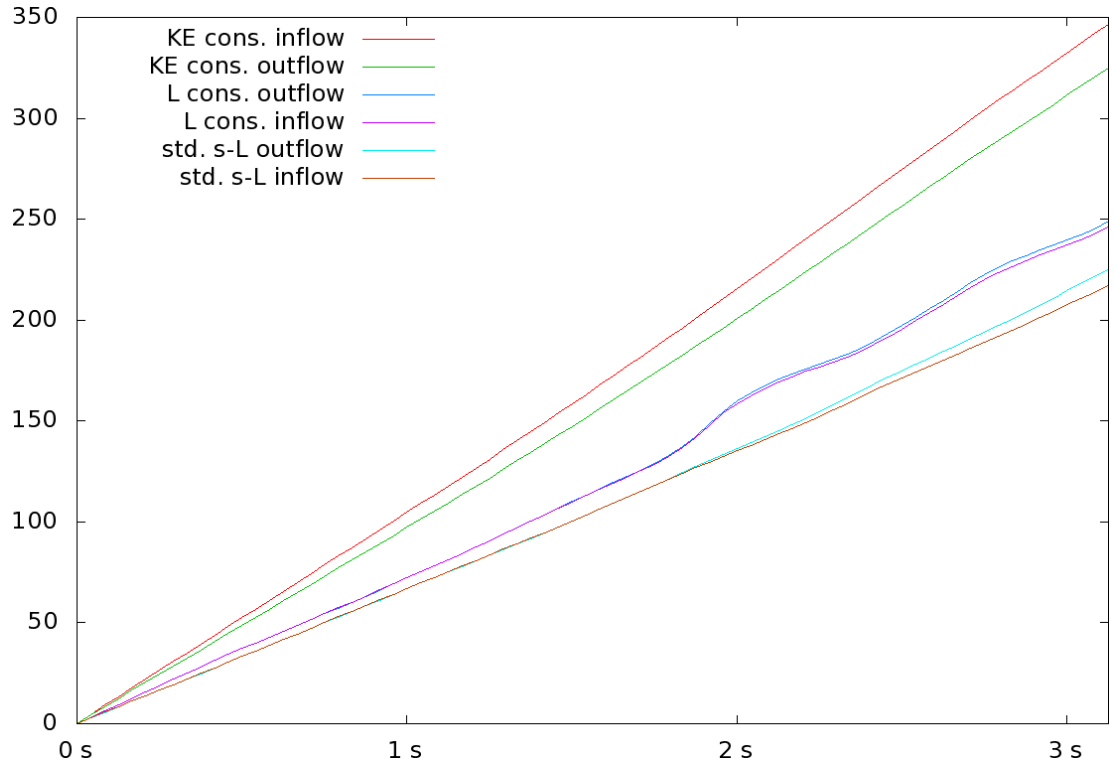


Figure 2.13: Total momentum fluxing into the computational domain and total momentum fluxing out of the computational domain, plotted as a function of time for a standard semi-Lagrangian scheme and our proposed momentum-conserving scheme.

that in this case the pressure acts as a collision between a fluid cell and a solid wall boundary, and that collisions influence both the momentum and the kinetic energy. Similar collisions happen at the inflow, where kinematically moving ghost cells collide with our fluid domain. A new term that we didn't consider for the momentum is the loss of kinetic energy during projection, where fluid cells collide with each other in a partially elastic way, losing kinetic energy; these loses are shown in Figure 2.18 for each of the three schemes. Figure 2.19 shows the sum of all these terms discussed, leaving only losses which occur during the advection stage of our schemes. Note that our kinetic energy-conserving advection scheme does indeed conserve kinetic energy during advection, whereas both the standard semi-Lagrangian scheme as well as the momentum-conserving scheme lose kinetic energy in this step.

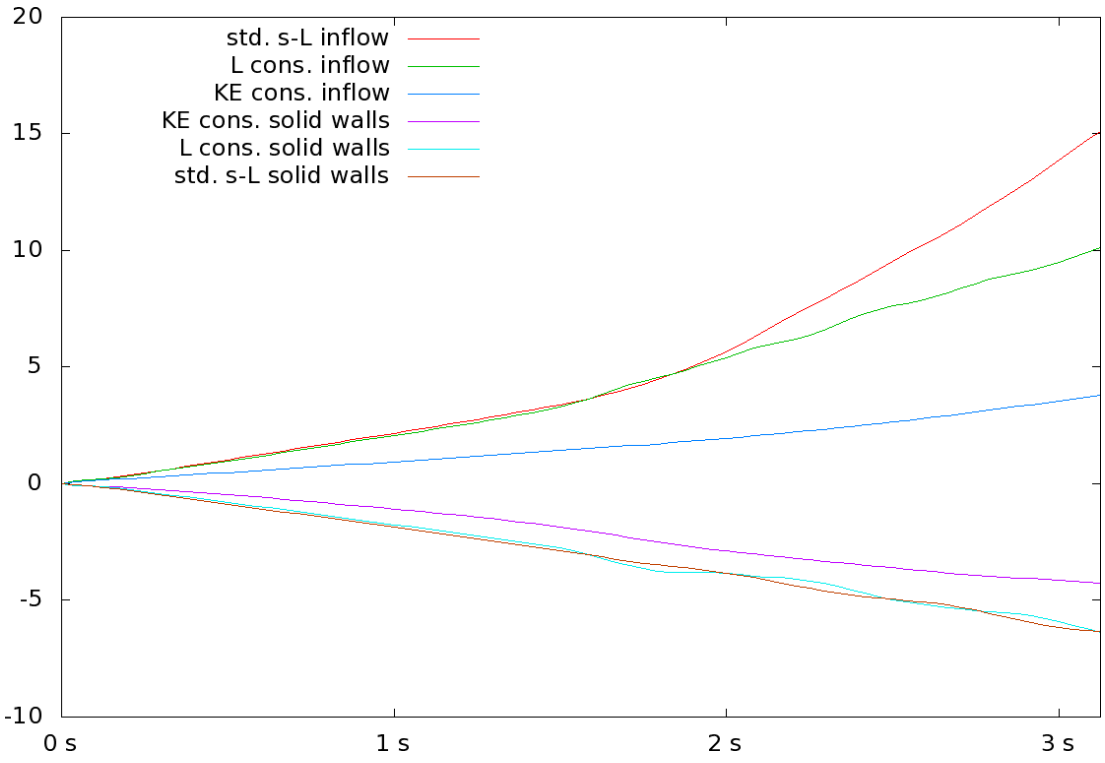


Figure 2.14: Pressure momentum flux into solid wall boundaries, and pressure momentum flux entering the computational domain from the inflow boundary condition, plotted as a function of time for a standard semi-Lagrangian scheme and our proposed momentum-conserving scheme.

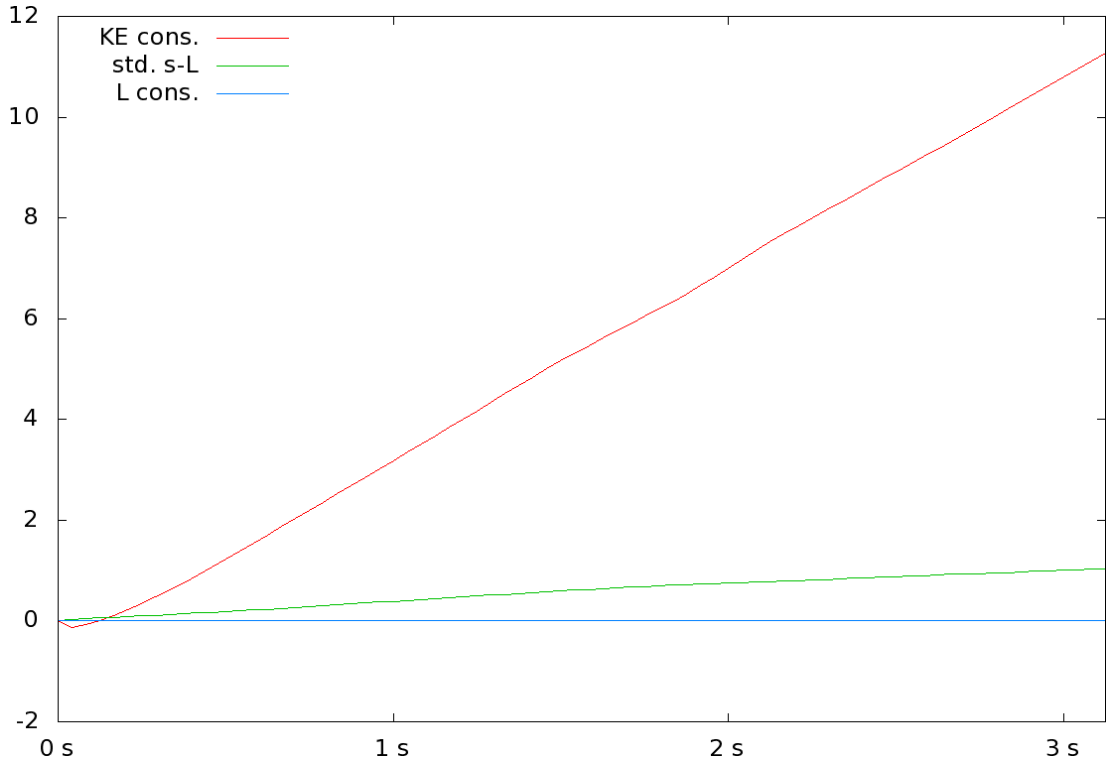


Figure 2.15: Sum total of momentum in the domain, plus momentum fluxed out of the domain (through outflow and solid wall boundaries), minus momentum fluxed into the domain (through inflow), plotted as a function of time for a standard semi-Lagrangian scheme and our proposed momentum-conserving scheme.

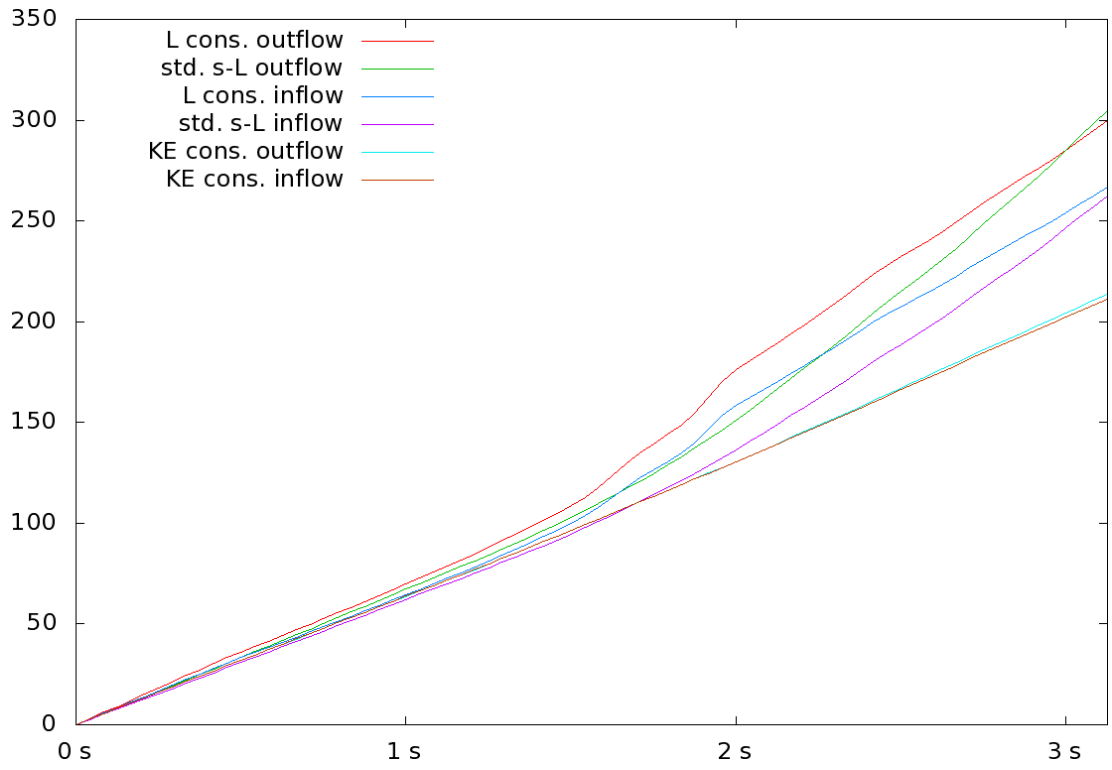


Figure 2.16: Total kinetic energy fluxing into the computational domain and total kinetic energy fluxing out of the computational domain, plotted as a function of time for a standard semi-Lagrangian scheme, our proposed momentum-conserving scheme, and our proposed kinetic energy-conserving scheme.

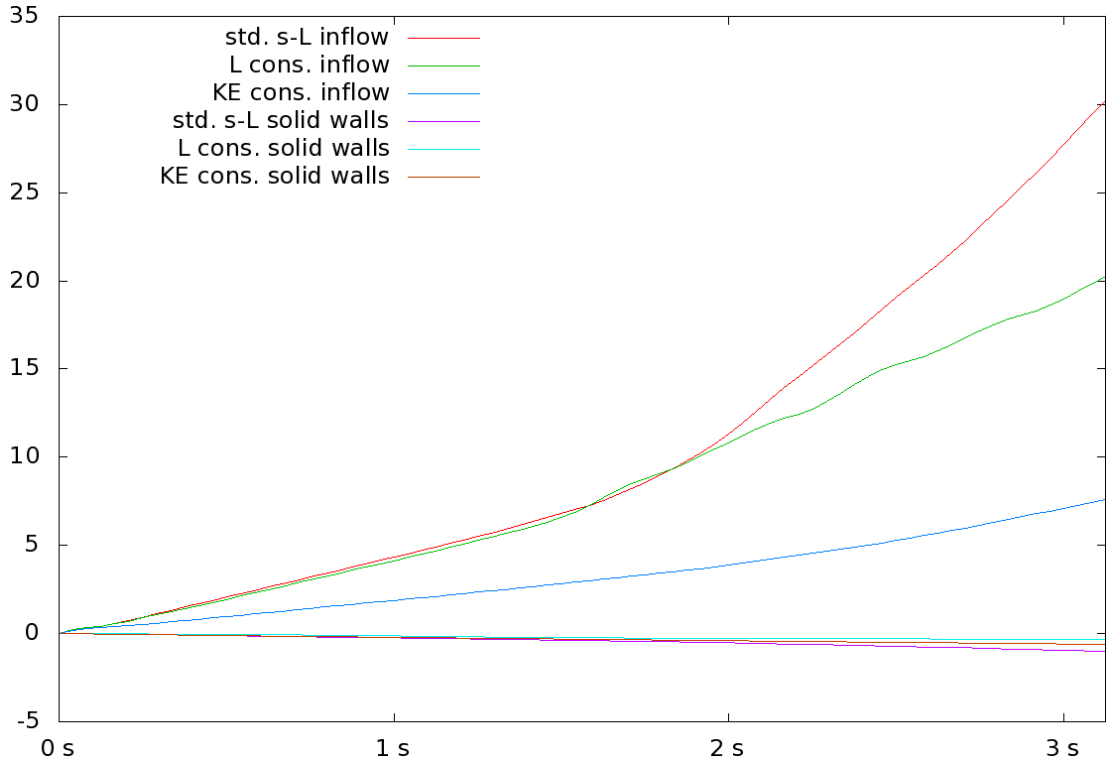


Figure 2.17: Energy flux into solid wall boundaries, and energy flux entering the computational domain from the inflow boundary condition, plotted as a function of time for a standard semi-Lagrangian scheme, our proposed momentum-conserving scheme, and our proposed kinetic energy-conserving scheme.

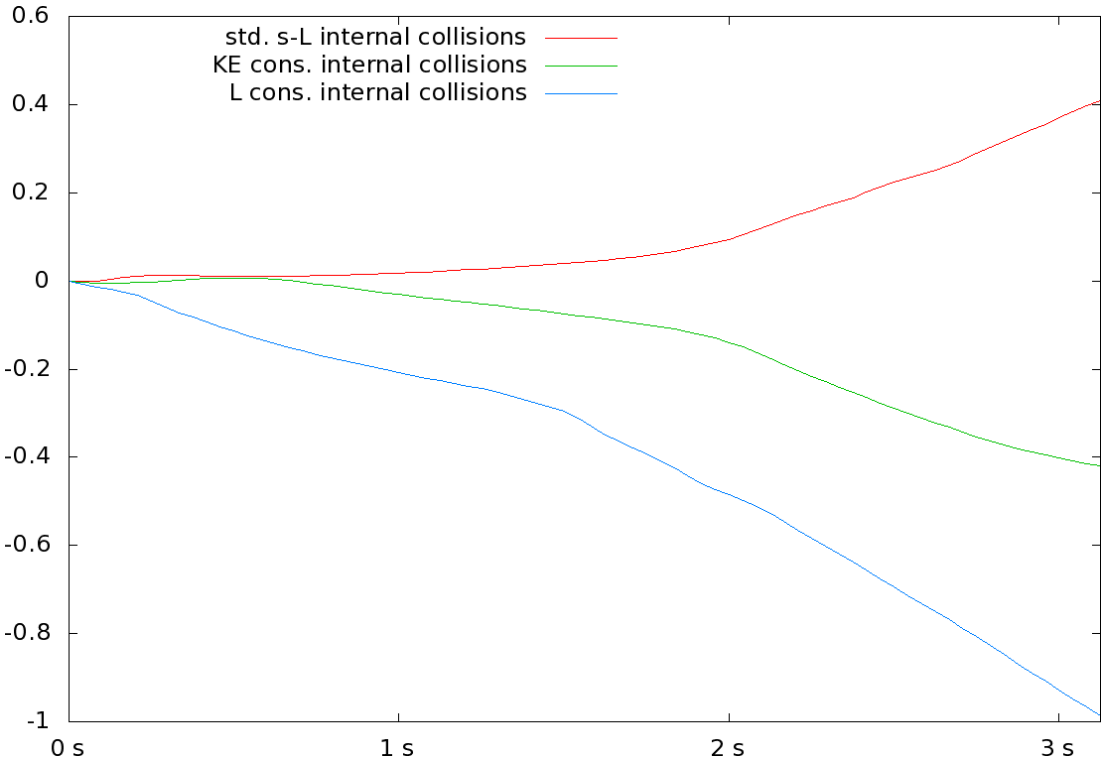


Figure 2.18: Change in kinetic energy due to the pressure projection step away from boundaries, plotted as a function of time for a standard semi-Lagrangian scheme, our proposed momentum-conserving scheme, and our proposed kinetic energy-conserving scheme. Note that in all three schemes the change in momentum due to the pressure projection step away from boundaries is zero.

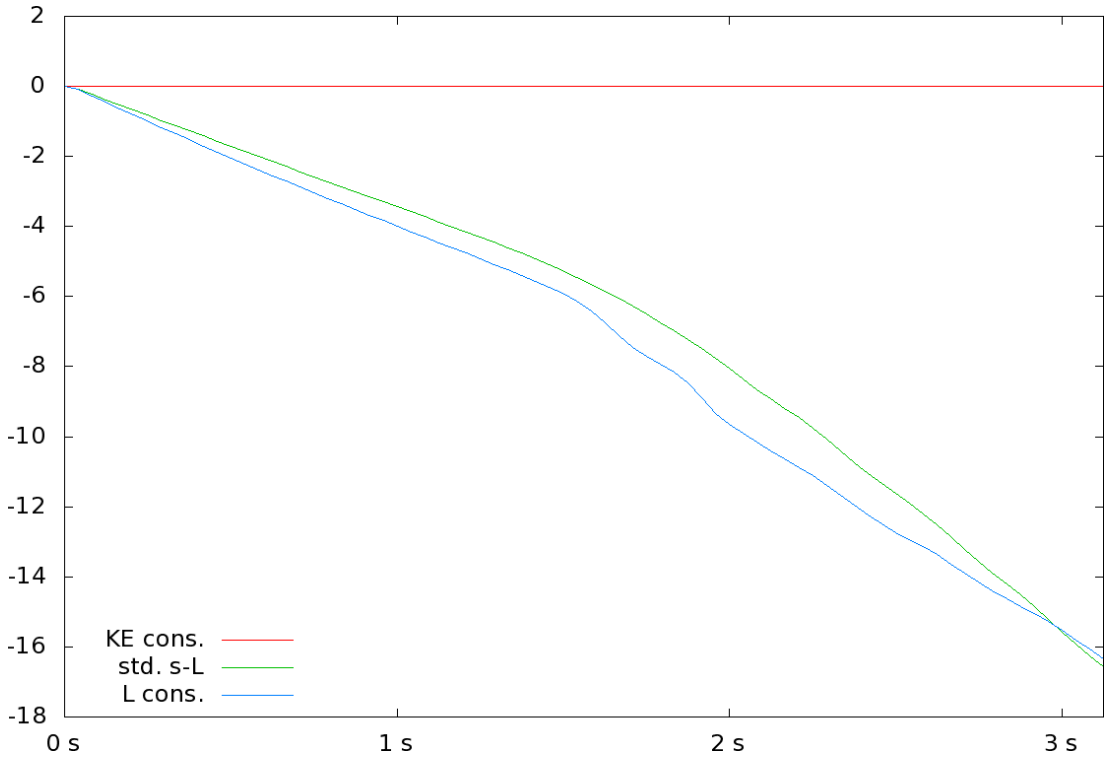


Figure 2.19: Sum total of kinetic energy in the domain, plus kinetic energy fluxed out of the domain (through outflow and solid wall boundaries), minus kinetic energy fluxed into the domain (through inflow), plus kinetic energy lost in the projection step away from boundaries, plotted as a function of time for a standard semi-Lagrangian scheme, our proposed momentum-conserving scheme, and our proposed kinetic energy-conserving scheme.

2.5 Compressible flow

We model compressible flow using the inviscid Euler equations,

$$\begin{pmatrix} \rho \\ \rho\vec{u} \\ E \end{pmatrix}_t + \begin{pmatrix} \nabla \cdot [\rho\vec{u}] \\ \nabla \cdot [\rho\vec{u} \otimes \vec{u}] + \nabla p \\ \nabla \cdot [(E+p)\vec{u}] \end{pmatrix} = 0, \tag{2.20}$$

where ρ is the fluid density, $\rho\vec{u}$ is the momentum, $E = \rho e + \frac{1}{2}\rho\vec{u}\cdot\vec{u}$ is the total energy per unit volume and e is the internal energy per unit mass. These are solved using the splitting proposed in [54]. Defining the state vector as $\vec{U} = (\rho, \rho\vec{u}, E)^T$, the flux is split into its advective component, $F_1(\vec{U})$, and acoustic component $F_2(\vec{U})$:

$$F_1(\vec{U}) = \begin{pmatrix} \nabla \cdot [\rho\vec{u}] \\ \nabla \cdot [\rho\vec{u} \otimes \vec{u}] \\ \nabla \cdot [E\vec{u}] \end{pmatrix}, \qquad F_2(\vec{U}) = \begin{pmatrix} 0 \\ \nabla p \\ \nabla \cdot [p\vec{u}] \end{pmatrix}. \tag{2.21}$$

The method first computes $F_1(\vec{U})$ explicitly with the MENO advection scheme, which uses density-averaged velocities at cell faces, advecting the state variables to an intermediate state $\vec{U}^\star$. That is,

$$\begin{aligned} \rho^\star &= \rho^n - \Delta t \nabla \cdot (\rho\vec{u}) \\ \rho\vec{u}^\star &= \rho\vec{u}^n - \Delta t \nabla \cdot [\rho\vec{u} \otimes \vec{u}] \\ E^\star &= E^n - \Delta t \nabla \cdot (E\vec{u}). \end{aligned}$$

Note that $\rho^\star = \rho^{n+1}$, as the first term in $F_2(\vec{U})$ is zero. Next, we examine the remaining component of the momentum equation,

$$\rho^{n+1}\vec{u}^{n+1} - \rho^{n+1}\vec{u}^\star = -\Delta t \nabla p.$$

We divide through by ρ^{n+1} and take its divergence, yielding an implicit equation for

pressure:

$$\nabla \cdot \vec{u}^{n+1} - \nabla \cdot \vec{u}^{\star} = -\Delta t \nabla \cdot \frac{1}{\rho^{n+1}} \nabla p. \tag{2.22}$$

In order to remain conservative, we discretize $\nabla \cdot \vec{u}^{\star}$ by computing $\vec{u}^{\star}$ at faces. That is, we compute $\nabla \cdot \vec{u}^{\star} = -G^T \hat{\vec{u}}^{\star}$, where $-G^T$ is the discretized divergence operator and $\hat{\vec{u}}^{\star}$ are $\vec{u}^{\star}$ velocities averaged to faces. Then we next eliminate the $\nabla \cdot \vec{u}^{n+1}$ term by considering the pressure evolution equation (see [24]):

$$p_t + \vec{u} \cdot \nabla p + \rho c^2 \nabla \cdot \vec{u} = 0. \tag{2.23}$$

This is discretized as $p^{n+1} = p^a - \Delta t \rho^n (c^n)^2 \nabla \cdot \vec{u}^{n+1}$, where p^a is an advected p^n pressure using the $\vec{u}^n$ velocity field. Plugging this into (2.22), discretizing the gradient ∇ as G and the divergence $\nabla\cdot$ as $-G^T$ gives the following implicit pressure equation:

$$\left[I + \rho^n (c^2)^n \Delta t^2 G^T \left(\frac{1}{\hat{\rho}^{n+1}} G \right) \right] p^{n+1} = p^a + \rho^n (c^2)^n \Delta t G^T \hat{\vec{u}}^{\star}. \tag{2.24}$$

where $\hat{\rho}^{n+1}$ are densities averaged to cell faces.

Finally these pressures are applied to the $\vec{U}^{\star}$ state to get time t^{n+1} quantities. Since pressure values and momentum quantities are collocated, we average pressures to faces as $p^{n+1}_{i+1/2} = \frac{p^{n+1}_{i+1}\rho^{n+1}_i + p^{n+1}_i \rho^{n+1}_{i+1}}{\rho^{n+1}_i + \rho^{n+1}_{i+1}}$, permitting us to evaluate ∇p for the momentum update in a flux-based manner. We also want to evaluate $p\vec{u}$ at cell faces in order to numerically conserve total energy, and so we update the $\hat{\vec{u}}^{\star}$ velocities from Equation (2.22) as $\hat{\vec{u}}^{n+1} = \hat{\vec{u}}^{\star} - \Delta t \frac{G p^{n+1}}{(\rho_i + \rho_{i+1})/2}$. This permits us to write the numerically conservative flux-based update as

$$(\rho\vec{u})^{n+1} = (\rho\vec{u})^{\star} - \Delta t \left(\frac{p^{n+1}_{i+1/2} - p^{n+1}_{i-1/2}}{\Delta x} \right), \qquad E^{n+1} = E^{\star} - \Delta t \left(\frac{(p\hat{u})^{n+1}_{i+1/2} - (p\hat{u})^{n+1}_{i-1/2}}{\Delta x} \right). \tag{2.25}$$

In order to demonstrate our new conservative semi-Lagrangian advection, we use it to replace the MENO advection scheme when solving $F_1(\vec{U})$. Notably the method of [54] was able to stably compute the solutions of compressible flow ignoring the CFL

restriction due to the acoustic wave because of the implicit treatment of pressure in Equation (2.24). However, they were still limited by a CFL restriction based on the fluid velocity. Using our unconditionally stable advection scheme, we are no longer restricted to a fluid velocity-based CFL.

2.5.1 Example

We solve the classic one-dimensional Sod shock tube [117] using the advection-based CFL condition given by

$$\frac{\Delta t}{2}\left(\frac{|u|_{max}}{\Delta x}+\sqrt{\frac{|u|_{max}^2}{\Delta x^2}+4\frac{|p_x|}{\rho\Delta x}}\right)\leq 1.$$

as defined in [54]. The Sod shock tube takes as initial conditions

$$(\rho(x,0),u(x,0),p(x,0))=\begin{cases}(1,0,1) & \text{if } x\leq .5,\\ (.125,0,.1) & \text{if } x>.5.\end{cases} \tag{2.26}$$

This example is solved on a computational domain of $x\in(0,1)$, with $\Delta x=2.5\times10^{-3}$. We compare the results of an approach using MENO and a CFL number of .9 with the results of an approach using our conservative semi-Lagrangian advection scheme with a CFL number of 3. To illustrate convergence, we show a plot of density at time $t=.15s$ for a selection of grid resolutions in Figure 2.20, where conservative semi-Lagrangian advection is used with the semi-implicit compressible flow solver with a CFL number of .5. Figure 2.21 shows convergence when a CFL number of 3 is used. We show the same quantities for $t=.8s$ in Figures 2.23 and 2.24. Each figure also shows the resulting solution when solved using a traditional, fully explicit compressible flow solver with 3rd order accuracy in time and space, for comparison. We stress that the over-shoots near the shock front are a consequence of the semi-implicit discretization of the equations discussed in [54], as the implicit pressure system is centrally biased; to illustrate this point, we show in Figure 2.22 the results

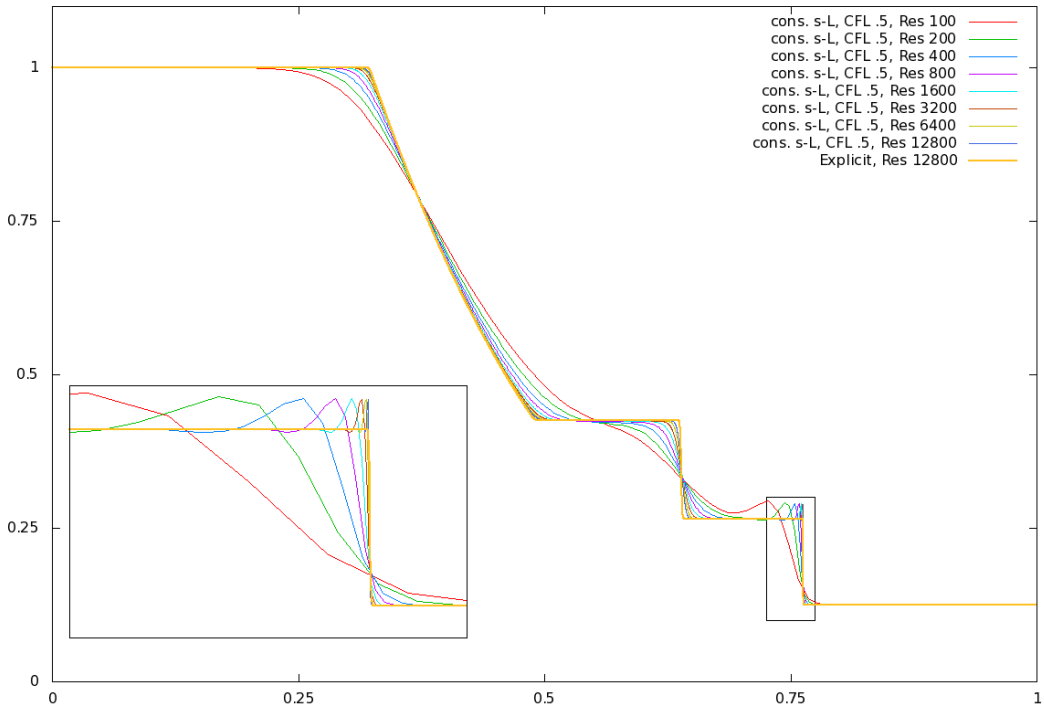


Figure 2.20: Density profile of a SOD shock tube at $t = .15s$, as generated by the scheme detailed in [54], using our new conservative semi-Lagrangian scheme and a CFL number of .5. We zoom in to the box $[.725, .775] \times [.1, .3]$, showing the shock front in greater detail and highlighting convergence at the discontinuity.

generated when a third order MENO advection scheme is used instead, which suffers from these same overshoots.

Currently, in the context of compressible flows, we are working to extend our method in a fashion that hybridizes it with a flux-based scheme such as that of [104]. The goal here would be to apply high order accurate flux-based discretization in most of the domain (albeit with a restrictive CFL condition), yet apply our method near moving solid boundaries and especially thin shells, see [35].

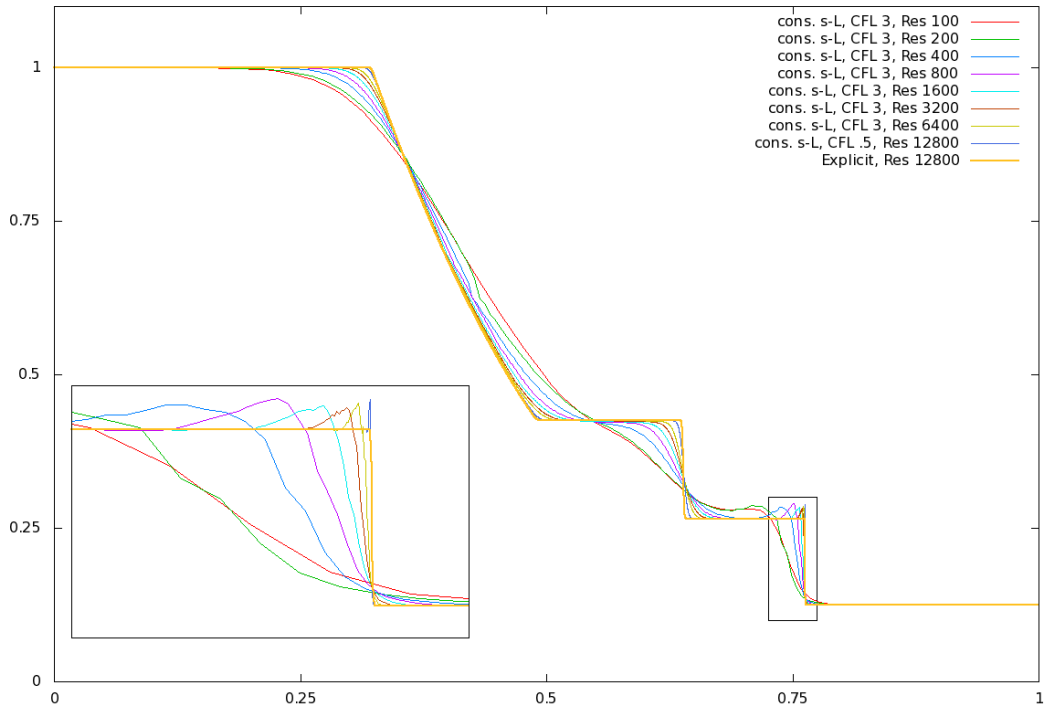


Figure 2.21: Density profile of a SOD shock tube at $t = .15s$, as generated by the scheme detailed in [54], using our new conservative semi-Lagrangian scheme and a CFL number of 3. We zoom in to the box $[.725, .775] \times [.1, .3]$, showing the shock front in greater detail and highlighting convergence at the discontinuity.

2.6 Conclusion

We have presented a conservative, unconditionally stable semi-Lagrangian advection scheme. The method is built from simple, first order semi-Lagrangian building blocks. We show that the method is beneficial in the simulation of both incompressible and compressible flows.

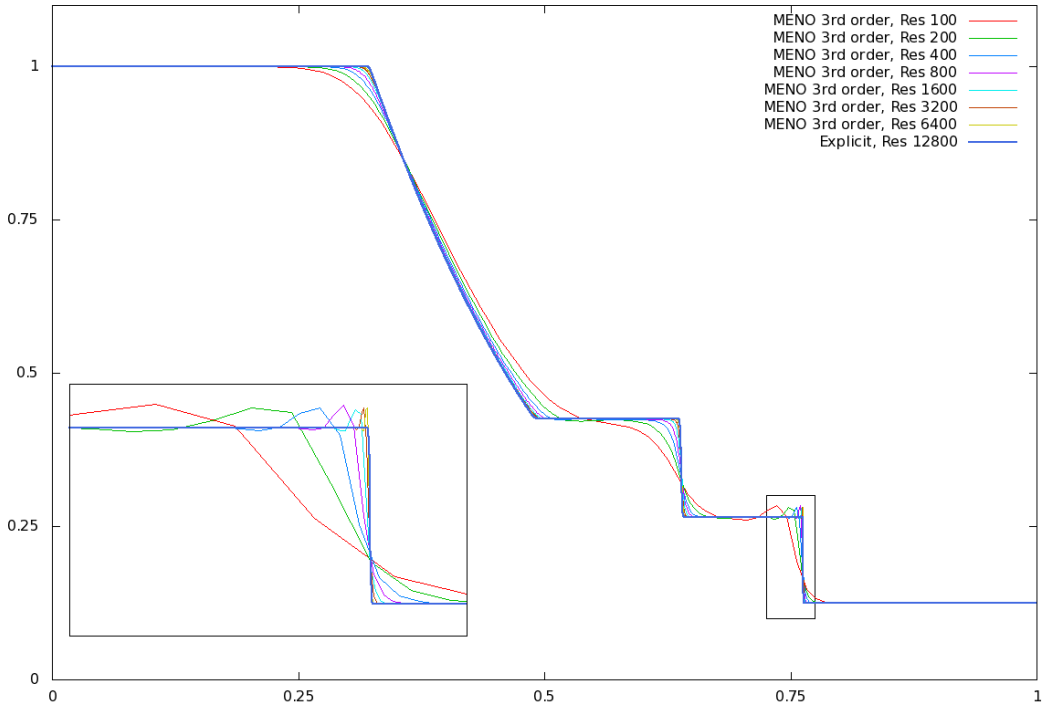


Figure 2.22: Density profile of a SOD shock tube at $t = .15s$, as generated by the scheme detailed in [54], using a third order MENO advection scheme and a CFL number of .5. We zoom in to the box $[.725, .775] \times [.1, .3]$, showing the shock front in greater detail and highlighting convergence at the discontinuity.

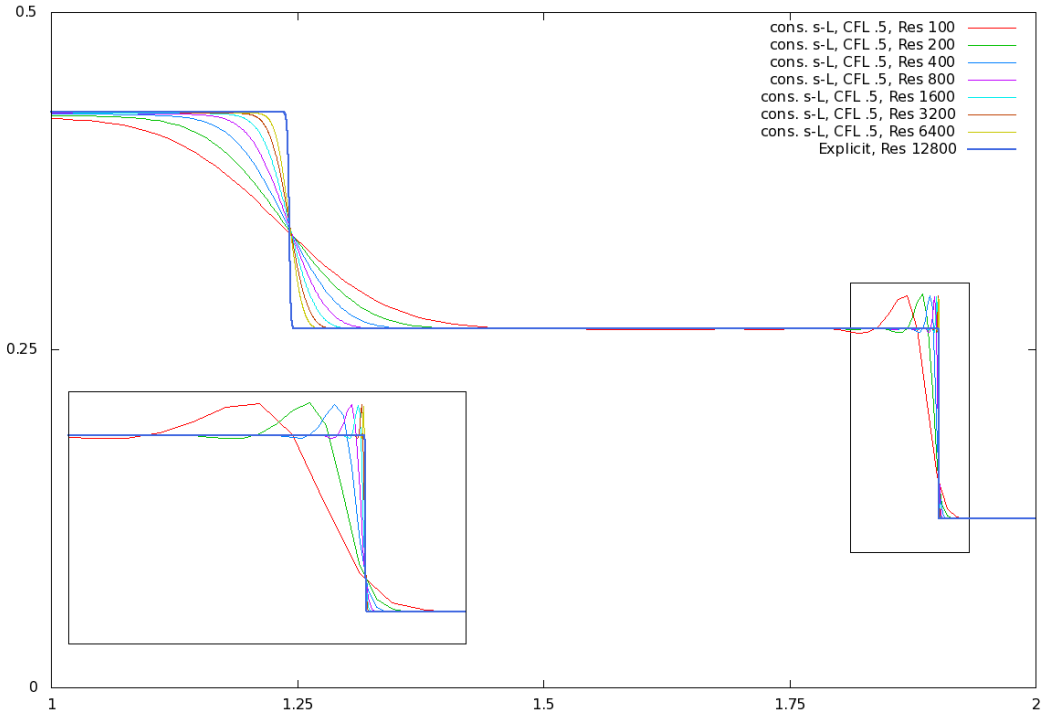


Figure 2.23: Density profile of a SOD shock tube at $t = .8s$, as generated by the scheme detailed in [54], using our new conservative semi-Lagrangian scheme and a CFL number of .5. In order to capture this later time, we extend the computational domain to $x \in (-1, 2)$ and show only $x \in (1, 2)$ to illustrate shock front convergence. We zoom in to the box $[1.812, 1.932] \times [.1, .3]$, showing the shock front in greater detail and highlighting convergence at the discontinuity.

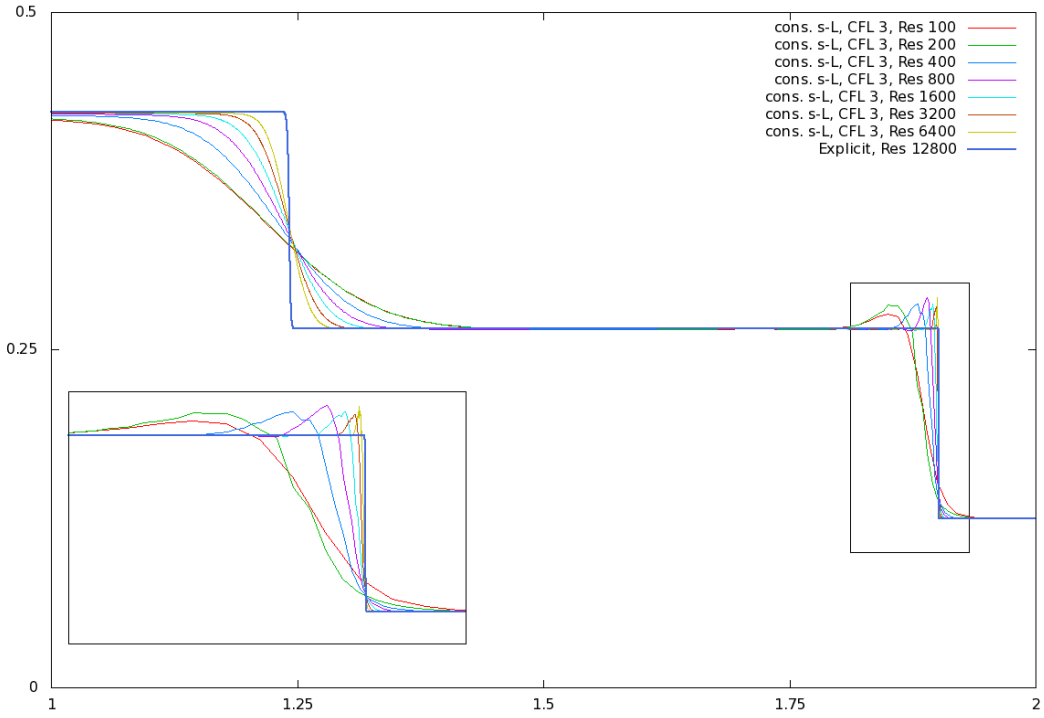


Figure 2.24: Density profile of a SOD shock tube at $t = .8s$, as generated by the scheme detailed in [54], using our new conservative semi-Lagrangian scheme and a CFL number of 3. In order to capture this later time, we extend the computational domain to $x \in (-1, 2)$ and show only $x \in (1, 2)$ to illustrate shock front convergence. We zoom in to the box $[1.812, .932] \times [.1, .3]$, showing the shock front in greater detail and highlighting convergence at the discontinuity.

Chapter 3

Mass and Momentum Conservation

Momentum conservation has long been used as a design principle for solid simulation (e.g. collisions between rigid bodies, mass-spring elastic and damping forces, etc.), yet it has not been widely used for fluid simulation. In fact, semi-Lagrangian advection does not conserve momentum, but is still regularly used as a bread and butter method for fluid simulation. In this chapter, we propose a modification to the semi-Lagrangian method in order to make it fully conserve momentum. While methods of this type have been proposed earlier in the computational physics literature, they are not necessarily appropriate for coarse grids, large time steps or inviscid flows, all of which are common in graphics applications. In addition, we show that the commonly used vorticity confinement turbulence model can be modified to exactly conserve momentum as well. We provide a number of examples that illustrate the benefits of this new approach, both in conserving fluid momentum and passively advected scalars such as smoke density. In particular, we show that our new method is amenable to efficient smoke simulation with one time step per frame, whereas the traditional non-conservative semi-Lagrangian method experiences serious artifacts when run with these large time steps, especially when object interaction is considered.

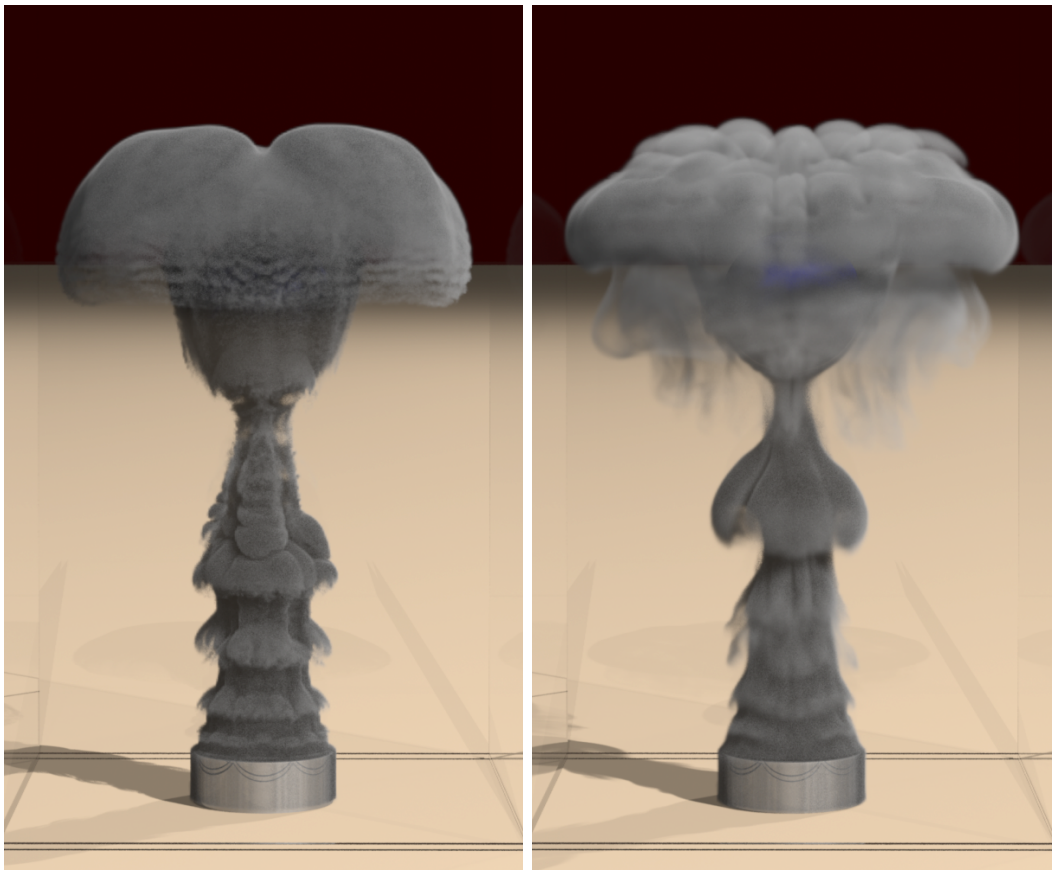


Figure 3.1: A comparison between simulations of (Left) the traditional semi-Lagrangian method and (Right) our method with a very large time step at resolution $256 \times 512 \times 256$. Note how the large time steps cause alternating gaps in the smoke as seen above and below the sphere. Also note the lack of fluid structure resulting from the collision with the sphere. In contrast, our method conserves mass and momentum and produces a highly detailed flow field. Note in particular, the creation of multiple distinct vortex rings that pass through each other using our method.

3.1 Introduction

Momentum conservation is considered to be a fundamental building block for solid simulations (e.g. it is enforced during rigid body collisions, rigid body temporal evolution, elastic and damping forces for mass-spring systems, etc.), however, when simulating incompressible fluids in the graphics community, it has largely been ignored.

The same can be said in the computational physics community, where many methods for simulating incompressible flow typically do not conserve momentum, albeit when simulating compressible flow, momentum is strictly conserved since it is required to get the correct shock speeds. We refer the reader to a few of the papers on compressible flow written in the graphics literature, such as [120, 103, 55]. In contrast, the semi-Lagrangian method, introduced in graphics by Stam [105], does not conserve momentum, yet it has been a staple algorithm for incompressible flow for over a decade.

While many alternative advection methods exist in the computational physics literature, they often come with restrictions on the grid type, grid spacing, or boundary conditions. This makes them rather difficult to use in graphics applications where the geometry tends to be detailed and complex. Most of these schemes also come with restrictions on the size of the time step, but semi-Lagrangian advection is unconditionally stable. In fact, this makes it appealing to build high-order methods out of first-order semi-Lagrangian building blocks, as was done in the BFECC [18, 46] and the MacCormack [101] methods. Semi-Lagrangian methods are useful on octrees because of their hierarchical nature and varying grid cell-size [69, 68], on tetrahedral meshes because of their lack of structure and varying element size [26, 27, 51, 11, 2], and for solid-fluid coupling [8, 9, 1, 99]. It is especially useful when a solid moves through the mesh causing cells to be cut in various sizes or when considering thin shells such as cloth when one does not want leaking (see [37]) across the solid surface.

Some computational physics authors have considered momentum conservation in the context of incompressible flows. However, these researchers are typically interested in obtaining algorithms that converge under refinement in both space and time, necessitating the use of fully viscous flows. In contrast, graphics applications typically use inviscid flows for efficiency (see [25]), and these are known not to converge as the grid cell size and time step size approach zero. In addition, graphics applications typically use large time steps and coarse grids. Hence, it is not clear if these algorithms in the computational physics literature would perform well for graphics applications. We have studied one such paper [58] with a rather simple implementation based on the

Figure 3.2: (Left) using the method from [58], incompressibility is not properly enforced on coarse grids with large time steps and no viscosity. Note the white line down the middle of the image where the smoke splits apart, which occurs because of a lack of incompressibility during the advection. (Right) our new method incorporates incompressibility into advection, keeping the plume from splitting apart.

semi-Lagrangian method conducive to use in graphics applications and found that although this method converged to correct analytical solutions, it performed poorly for applications of interest to graphics. We illustrate this in Figure 3.2 and explain in Section 3.2 why it performs poorly. Furthermore, we propose a novel algorithm which aims to enforce incompressibility during advection at a discrete level, thereby alleviating the problems this scheme has for inviscid flows on coarse grids with large time steps.

The typical projection step which makes the fluid divergence free naturally conserves momentum. Combining this with our new momentum-conserving advection, one has a fully momentum-conserving incompressible flow solver (except for source terms). Note that body forces such as gravity and buoyancy should add momentum to a flow as potential energy is converted into kinetic energy. Conservative algorithms for both rigid and deformable bodies allow for momentum changes based on body forces (these forces would conserve momentum, but the other body, for example the earth, is not modelled). Many graphics applications use a turbulence model such as vorticity confinement (see [25, 102, 122, 44]) to increase the level of detail

in the flow, and we propose a novel modification to this source term which makes it fully conserve momentum. We demonstrate the advantages of using momentum conservation for smoke simulation through several examples. Finally, we show that our method can be adapted to water simulations and present some promising results for energy conservation.

3.2 Advection

A momentum-conserving incompressible flow solver requires a conservative method for advection. Consider a passively advected scalar ϕ in a divergence-free velocity field $\vec{u}$ which is evolved using the equation

$$\phi_t + \vec{u} \cdot \nabla \phi = 0. \tag{3.1}$$

Throughout the chapter, we denote the value of ϕ at position $\vec{x}$ and at time t by $\phi(\vec{x}, t)$. We use $\vec{x}_k$ to denote the location of the center of cell k. When updating ϕ to time t^{n+1}, the traditional semi-Lagrangian method [105] traces a characteristic ray from cell j backwards in time to some position $\vec{x}^\star$ and interpolates ϕ values from the surrounding cells to obtain a ϕ value at $\vec{x}^\star$. This value is then used as the new value of ϕ at cell j, i.e., $\phi(\vec{x}_j, t^{n+1})$. In other words, the semi-Lagrangian method updates ϕ at $\vec{x}_j$ as

$$\phi(\vec{x}_j, t^{n+1}) = \phi(\vec{x}^\star, t^n) = \sum_{i \in N(\vec{x}^\star)} w_{ij} \phi(\vec{x}_i, t^n), \tag{3.2}$$

where $N(\vec{x}^\star)$ denotes the set of cells i near position $\vec{x}^\star$ and w_{ij} are the interpolation weights from cell centers $\vec{x}_i$ to the point $\vec{x}^\star$. Note that cell i could be in the set $N(\vec{x}^\star)$ for several points $\vec{x}^\star$ and several cells j. Also note that the sum $\beta_i = \sum_j w_{ij}$ represents the total amount of ϕ removed from cell i and is typically not equal to 1.

3.2.1 Conservation

[58] proposes two additional steps beyond Equation 3.2 to make this scheme fully conserve ϕ. They note that when $\beta_i > 1$, more ϕ is removed from cell i than exists at time t^n and modify the weights to be $\hat{w}_{ij} = w_{ij}/\beta_i$, for these cells. In addition, when $\beta_i < 1$, some of the ϕ in cell i is not advected with the flow. In this case, they perform a second forward advection step tracing a characteristic ray forward in time from cell i to some new position $\vec{x}^{\star\star}$, and distribute $(1 - \beta_i)\phi(\vec{x}_i, t^n)$ to the cells in $N(\vec{x}^{\star\star})$ using an interpolation stencil. Therefore, if the interpolation weights from cell j to the point $\vec{x}^{\star\star}$ are α_{ij}, they distribute $(1 - \beta_i)\alpha_{ij}\phi(x_i, t^n)$ to each cell j. This means that they modify Equation 3.2 to increment weights w_{ij} by an amount $(1 - \beta_i)\alpha_{ij}$ to get $\hat{w}_{ij}$ for these cells as well as changing $N(\vec{x}^{\star})$ to include all cells from the forward advection step that were not already accounted for during the backward advection step. If we redefine w_{ij} to be $\hat{w}_{ij}$ we note that Equation 3.2 still holds, except that now the weights w_{ij} have been first clamped (to guarantee that no excess amount of ϕ is removed from cell i) and then incremented (to account for any ϕ in cell i that had not yet been advected). These modifications guarantee that $\beta_i = 1$, which implies that for each cell i, the amount of ϕ advected to other cells is exactly equal to the amount of ϕ originally present in cell i, making the scheme fully conservative.

Although the authors show convergence for viscous flows under refinement in space and time, when experimenting with this scheme on coarse grids, with large time steps and inviscid flows, we observed that vortices were able to tear the flow apart, producing gaps such as those illustrated in Figure 3.2 (left) where a white spacing runs down the center line of the flow. We observed that the absence of smoke along the center line was caused because the clamping limits the amount of density that can reach these cells. The quantity $\gamma_j = \sum_i w_{ij}$ gives one an indication of how much of ϕ reaches each cell. For the traditional semi-Lagrangian scheme, $\gamma_j = 1$, meaning that every cell is filled, even though cells are oversampled or undersampled in order to do this. [58] enforces $\beta_i = 1$ but typically produces $\gamma_j \neq 1$. For conservative incompressible flow, all of ϕ should be advected ($\beta_i = 1$) and every destination cell should be exactly filled ($\gamma_j = 1$). This can be viewed as making the weight matrix

W doubly-stochastic.

We propose the following new scheme. First, we make the observation that the clamping and forward advection steps can be flipped, and thus, after the first semi-Lagrangian step we forward advect for all cells with $\beta_i < 1$ before clamping. This advection step ensures that $\beta_i \geq 1$ and $\gamma_j \geq 1$ for all cells. Next, we clamp γ_j to 1 by redefining all w_{ij} to be w_{ij}/γ_j, guaranteeing incompressibility. Unfortunately, to guarantee conservation of ϕ, one still has to rescale β_i to 1, and in general, β_i is not equal to 1 at this point. Simply clamping β_i would guarantee conservation and give improved results over the original scheme but would still give γ_j unequal to 1. To alleviate this problem one could iterate by alternately clamping γ_i and β_i, always ensuring to clamp β_i last to enforce conservation. Doing this will converge such that both γ_i and β_i are 1 but applying a diffusion-based operator is more efficient. For our examples, we use one clamping iteration and then apply our diffusion method (see below).

Diffusing the quantity ϕ would destroy the details of the flow, and so we diffuse the sum γ_j instead. Consider the heat equation $(\gamma_j)_t - \Delta\gamma_j = 0$ discretized with forward Euler in time and central differencing in space, which has an explicit time step restriction of $\Delta t = (\Delta x)^2/2d$, where d is the dimension. This means that between any two cells j and $j+1$ in three dimensions, the flux looks like $(\gamma_{j+1} - \gamma_j)\Delta t/(\Delta x)^2$ or $(\gamma_{j+1} - \gamma_j)/6$ (by substitution). We can then apply this flux to each of the six pieces in the seven point stencil independently. However, this process can be accelerated by considering the heat equation one dimension at a time, where $\Delta t = (\Delta x)^2/2$ and the flux looks like $(\gamma_{j+1} - \gamma_j)/2$. Thus, we sweep through the grid in a dimension by dimension manner considering every flux between adjacent grid points j and $j+1$, updating their values by the amount $(\gamma_{j+1} - \gamma_j)/2$. This is done using Gauss-Jacobi iterations within a dimension, but Gauss-Seidel iterations between dimensional sweeps.

Assuming $\gamma_{j+1} > \gamma_j$, we subtract off the difference $(\gamma_{j+1} - \gamma_j)/2$ from γ_{j+1} and add it to γ_j, making them both equal to $(\gamma_{j+1} + \gamma_j)/2$. This can be viewed as updating

the weights $w_{k,j+1}$ and $w_{k,j}$ to $(w_{k,j+1} + w_{k,j})/2$, for all rows k. Observe that the sum $w_{k,j+1} + w_{k,j}$ does not change in the process implying that the sums β_j remain invariant under diffusion. We also advect ϕ during this update by moving the amount $\phi_{j+1}(\gamma_{j+1} - \gamma_j)/2\gamma_{j+1}$ from cell $j+1$ to cell j. Note that moving ϕ this way does not diffuse ϕ itself as evidenced by the fact that when all γ_j values are equal the ϕ values remain unchanged.

Diffusing the sums γ_j does not affect the sums β_j, so they remain 1. If the heat equation is solved to steady state, then γ_j equals 1 for all cells j. However, this turns out to be expensive and unnecessary. Hence, we actually clamp and diffuse the cumulative weights. We denote the cumulative weight at time t^n by γ_j^n, and initialize $\gamma_j^0 = 1$. These weights are advected forward in time in the same manner as ϕ to get $\tilde{\gamma}_j^{n+1}$. These advected weights are then used to clamp ϕ as described above, after which we apply a few iterations of our diffusion scheme to get γ_j^{n+1} (updating ϕ values in the process). Note that only a few iterations (between 1 and 7) are required in a time step as $\tilde{\gamma}_j^{n+1}$ incorporates the errors in incompressibility from previous iterations. One could alternatively use other existing methods for making the matrix W doubly-stochastic but we found that this method works well in our examples. It is important to note that although we are looking for a doubly-stochastic matrix we want a matrix that is as close as possible to the results from the conservative semi-Lagrangian method prior to adding diffusion.

3.2.2 Collisions

Although this method works well in the absence of solid objects we would also like to simulate examples such as the one shown in Figure 3.4. To conserve ϕ in the presence of objects, we modify both the forward and backward ray casting for interpolation to stop when it hits a solid object. We then use this surface point as $\vec{x}^\star$ (or $\vec{x}^{\star\star}$) for our interpolation weights. Unfortunately, this would still give interpolation weights coming from cells along the surface of the object. In this case, one can simply set those weights to 0, rescaling the remaining weights to sum to 1.

3.3 Incompressible Flow

Having developed a method for conservatively advecting a quantity, we can now apply it to incompressible flow in order to obtain a simulation that conserves both mass and momentum. We use our new advection method to passively advect necessary quantities such as smoke density $\hat{\rho}$ using the equation

$$\hat{\rho}_t + \vec{u} \cdot \nabla \hat{\rho} = 0, \tag{3.3}$$

which conserves the total mass throughout the simulation.

3.3.1 Navier-Stokes Equations

In order to conserve momentum, we solve the inviscid, incompressible Navier-Stokes equations, which are given by

$$\vec{u}_t + (\vec{u} \cdot \nabla)\vec{u} = -\frac{1}{\rho}\nabla p + \vec{f} \tag{3.4}$$

$$\nabla \cdot \vec{u} = 0, \tag{3.5}$$

where $\vec{u}$ is the velocity field of the fluid, ρ is the density of the fluid, $\vec{f}$ are any external forces (such as gravity) scaled by ρ, and p is the fluid pressure. We solve these equations by first calculating an intermediate velocity field $\vec{u}^\star$ via

$$\frac{\vec{u}^\star - \vec{u}^n}{\Delta t} + (\vec{u}^n \cdot \nabla)\vec{u}^n = \vec{f} \tag{3.6}$$

using our conservative advection method on the MAC grid. Since ρ is constant, this conserves momentum as well. We then subsequently add in the pressure forces via the equation

$$\frac{\vec{u}^{n+1} - \vec{u}^\star}{\Delta t} = -\frac{1}{\rho}\nabla p, \tag{3.7}$$

where the pressure is calculated by solving the Poisson equation

$$\nabla \cdot \frac{1}{\rho} \nabla \hat{p} = \nabla \cdot \vec{u}^{\star}, \tag{3.8}$$

where $\hat{p} = p\Delta t$.

We note that the pressure solve conserves momentum because pressure is applied in an equal and opposite manner to each neighboring $\vec{u}^{\star}$. This means that solving this system in the absence of external forces is fully mass and momentum conserving. We also note that since projection naturally conserves momentum and our method only modifies the advection step, other faster projection methods such as [59] and [77] can be used as well.

3.3.2 Vorticity Confinement

Some external forces are meant to add momentum to the system. For example, gravity adds momentum to the system, which is allowed because in reality momentum would be conserved if the earth was simulated. However, some forces such as vorticity confinement [25, 102, 122, 44], which are necessary to make interesting flow fields, are not momentum conserving. Heuristically, vorticity confinement should conserve momentum globally since spinning things in a circle tends to produce equal amounts of linear momentum in each direction. It turns out that it is straightforward to make vorticity confinement conservative; we do this as follows. For each spatial dimension, sum all the forces in that dimension, divide the resultant cumulative force by the number of cells, and decrement the force at every cell by the net force divided by the number of cells. We denote this momentum conserving vorticity confinement force as $\vec{F}^{\star}$. Finally, add the momentum conserving force $\epsilon \vec{F}^{\star}$, where ϵ is a scale parameter.

3.4 Smoke Simulation

Although the traditional semi-Lagrangian method allows us to take bigger time steps than conditionally stable methods, these large time steps result in noticeable errors in the simulation. For conditionally stable simulations, one typically uses the CFL number in order to define stability. A CFL number of 1 means that semi-Lagrangian rays have a maximum length of 1 grid cell, whereas a CFL number of 20 would allow them to trace back as many as 20 grid cells.

We demonstrate our method on several smoke simulation examples ran at 24 frames per second, as shown in Figures 3.3 and 3.4. We simulated two smoke examples, one with a ball and the other without it. Both examples have a density source at the bottom of the domain. For each example, we ran four simulations at resolution $128\times256\times128$: traditional semi-Lagrangian method with a CFL number of 1, our new method with the same CFL number, traditional semi-Lagrangian method at frame rate (1 time step per frame), and our method at frame rate. We also ran simulations at resolution $256 \times 512 \times 256$. Running these simulations using a low CFL number was infeasible, so we ran both examples at frame rate. In the lower resolution case, running at frame rate is equivalent to using a CFL number of 20. For the higher resolution ones the CFL number was around 40. This corresponds to taking 1/40 as many time steps as a CFL 1 simulation.

We compared our method to the traditional semi-Lagrangian method for all these examples. Figure 3.5 shows a comparison of the high resolution examples. Note the large amount of dissipation and artifacts that traditional semi-Lagrangian examples have compared to our method. We also quantitatively compared the two methods, as shown in Figure 3.7. For the lower resolution simulations we also compared our method with the traditional semi-Lagrangian method using a low CFL number (see Figure 3.6), demonstrating that even with low CFL numbers we achieve an increase in visual fidelity. In particular, note the large amount of mass lost both near and above the sphere using the traditional semi-Lagrangian method.

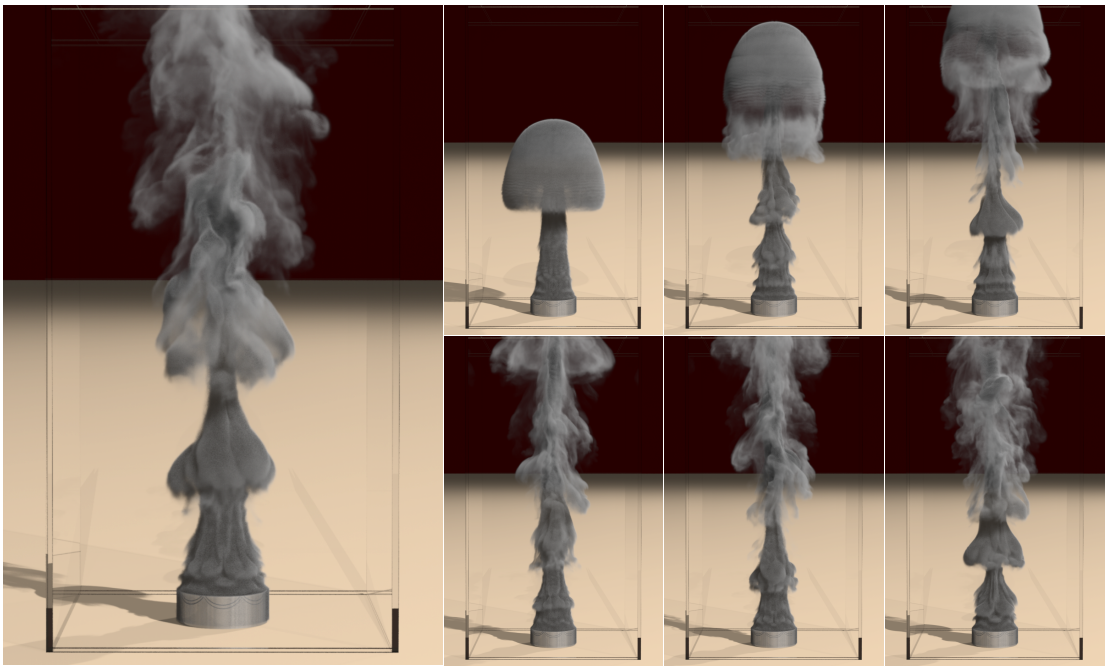


Figure 3.3: An example using our conservative advection method with smoke injected from below simulated at one time step per frame using a high CFL number (approximately 40) at resolution $256 \times 512 \times 256$.

It is important to note that as the grid resolution increases by a factor of 2, the cost increases by a factor of 8 in space and a factor of 2 in time. However, using our method we can reduce this to only a factor of 8 in space as we can generate good results at very large time step sizes. We also note that although our advection method is about three times slower than the traditional semi-Lagrangian method, the projection step dominates the cost for smoke simulations (typically around 90% of the simulation time) and thus our method achieves performance comparable to the traditional semi-Lagrangian method for each time step.

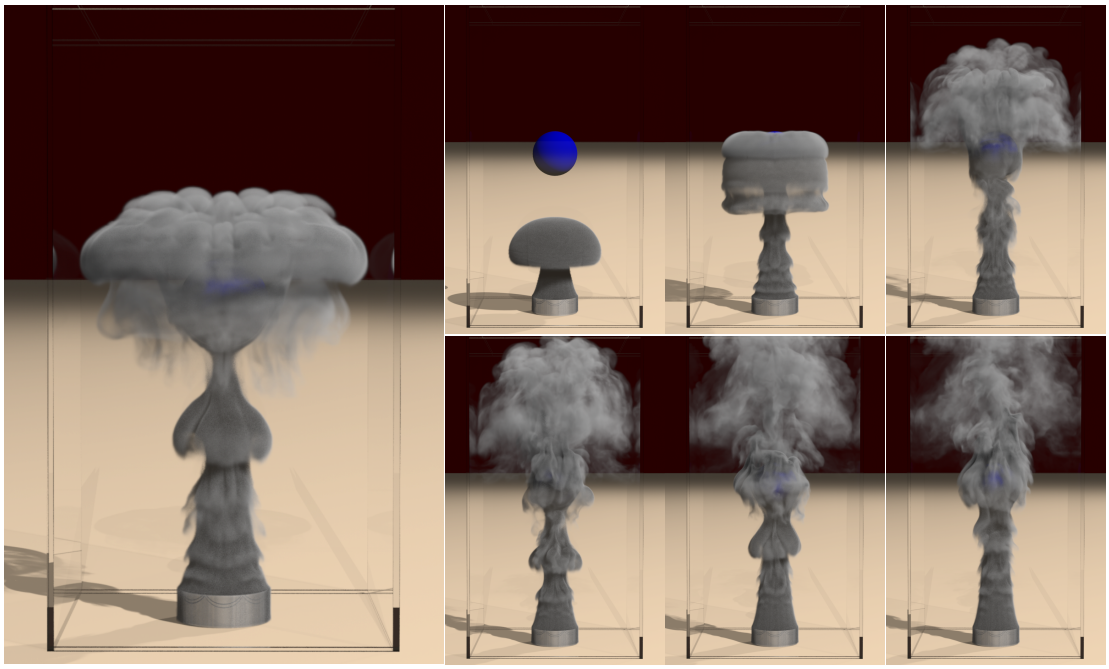


Figure 3.4: An example using our conservative advection method with smoke injected from below and a static sphere simulated at one time step per frame using a high CFL number (approximately 40) at resolution $256 \times 512 \times 256$.

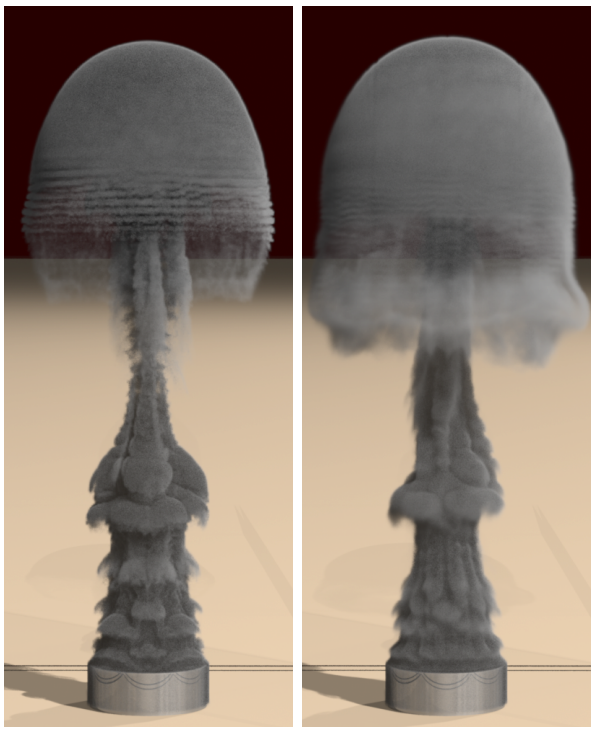


Figure 3.5: A comparison between simulations of (Left) the traditional semi-Lagrangian method and (Right) our method with a very large time step at resolution $256 \times 512 \times 256$. Note how the large time steps yield poor interpolation resulting in alternating gaps in the smoke; this is especially apparent slightly above the ground plane and in the large plume. Conserving the amount of smoke, as done by our method, does not produce these artifacts.

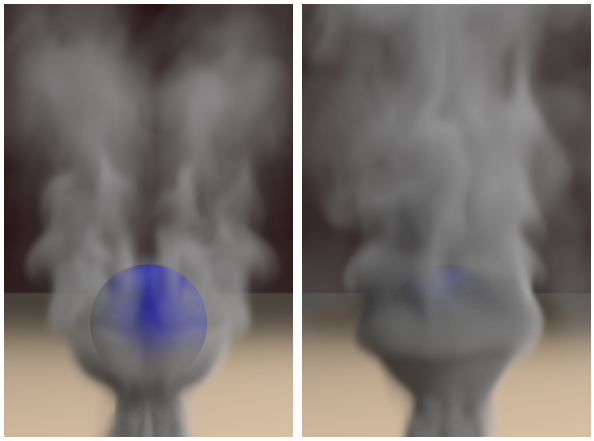


Figure 3.6: A comparison between simulations of (Left) the traditional semi-Lagrangian method and (Right) our method using a typical CFL of 1 at resolution $128 \times 256 \times 128$. Note the large amount of mass lost when the smoke interacts with the sphere as illustrated in Figure 3.7.

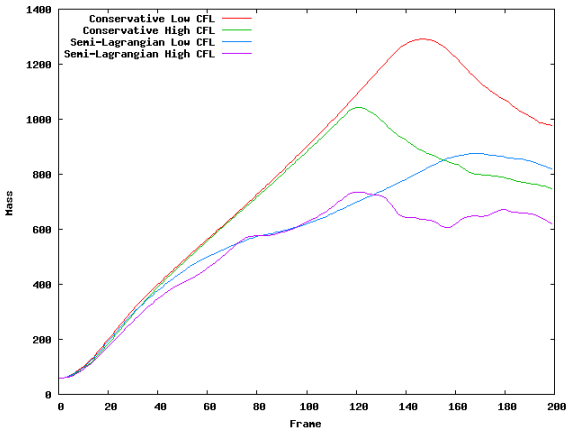


Figure 3.7: A comparison between four simulations at resolution $128 \times 256 \times 128$. The red and green lines are simulations using our conservative scheme. Note that the difference in these at later frames is due to different amounts of smoke exiting the domain as the simulations are different with largely different time steps - but we stress that smoke is fully conserved in both cases. Comparing the blue to the red, or similarly the purple to the green shows the amount of mass loss suffered by the traditional semi-Lagrangian method. Note that an appreciable amount of mass is lost even before large amounts of smoke starts exiting the domain.

3.5 Water Simulation

In addition to smoke, we applied momentum conservation to water. To do this we used the particle level set method [28, 20, 22], although one could also use methods such as coupled level set volume of fluid (CLSVOF) [80], marker level set [79] or front tracking [7, 116]. In order to make water conservative, momentum needs to be conserved during the velocity advection. However, our algorithm, as described in Section 3.2, cannot be directly applied because this would allow momentum to be advected from cells outside the level set (resulting in a gain in momentum), or allow momentum to be advected to cells outside the level set (resulting in a loss in momentum). To deal with these issues we modify both the backward and forward advection steps.

First, the standard semi-Lagrangian method is used to advect the level set from time t^n to t^{n+1}. Next, we consider the velocity advection step of our algorithm. We modify the algorithm described in Section 3.2.1 to use the time t^n level set as a collision body when doing backwards interpolation (see Section 3.2.2), ensuring that we only advect momentum that was part of the water volume. Note that some of the characteristic rays may not land in the level set at time t^n due to numerical errors. With reference to Figure 3.8 as an example, this would mean that faces $2, 3, 4, 5$ would all be outside the level set at time t^n. However, at least one of the two adjacent cells A or B at time t^{n+1} must have had a valid characteristic which landed inside the level set (say, at G), otherwise that particular velocity degree of freedom would not be inside the level set at time t^{n+1}. We then use A and/or B to find valid velocities. However, since these rays end up half of a grid cell away from the velocity value, we find a new velocity by tracing a ray half of a grid cell in the appropriate direction from the base of these characteristics. This would change the position from 6 to 7 which we can then use to update 1. In the case of both A and B being inside the levelset we use the average of the two approximations of 1. Note that when going half of a grid cell in one direction or the other from the base of the characteristic, we trace a ray colliding with the level set at time t^n just as before.

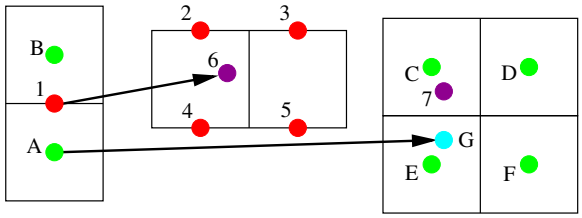


Figure 3.8: Momentum advection during water simulation: shown are the semi-Lagrangian rays used to advect phi values (green) and velocity values (red). Note that when advecting the interpolated velocity value, 6 is used if valid, otherwise 7 may be used.

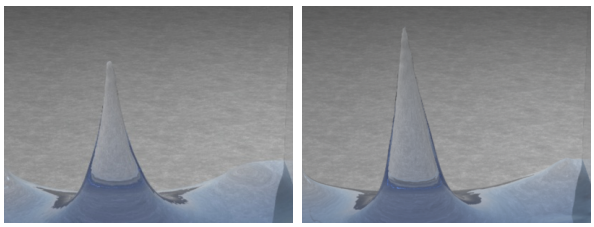


Figure 3.9: A comparison between water simulations using (Left) the traditional semi-Lagrangian method and (Right) our method at resolution $128 \times 256 \times 128$. Note the improvement in momentum seen using our conservative method. In particular, the height of the splash is higher using our method. Note that in this example our method has 25% more momentum than the standard method.

For forward advection we treat the time t^{n+1} level set as a collision body. However, we also need to treat forward advection a bit differently since there is no guarantee that time t^n values of the level set land within the time t^{n+1} level set using the particle level set method. Therefore, in the case when a velocity characteristic does not land within the time t^{n+1} level set, we simply find the nearest point on the surface and allocate the velocity to that point. Our intention here is not to provide an in-depth analysis for water simulation, as the case for our new method is adequately made via smoke simulation. However, we did wish to discuss some of the issues involved in adapting it for water and show some preliminary simulations. Figure 3.9 shows a comparison between our method and the traditional semi-Lagrangian method. In this example, momentum conservation produces a higher splash as expected (see also the video).

3.6 Energy

In addition to mass and momentum conservation, we can also conserve the energy of the simulation. Energy conservation has been researched recently as a way to reduce dissipation (see e.g. [100, 83, 113, 89]). In order to conserve the overall energy, we need to make sure energy is conserved in every step of the simulation. To do this in a visually appealing manner, we choose to add the energy that was lost at every time step with vorticity confinement. The role of vorticity in spinning up or slowing down a flow works to our advantage for adding or removing energy. Moreover, it can be difficult to add energy via a force-based method as projection can remove any force added at this step, but vorticity confinement is by nature incompressible (not exactly, but approximately) and therefore mostly survives the projection step.

Consider adding a momentum conserving vorticity confinement force $\vec{F}$

$$\vec{u}_c = \vec{u}^\star + \epsilon\vec{F} \tag{3.9}$$

to the incremental velocity from Equation 5.7. The increase in energy due to this force can be calculated as

$$E = \frac{1}{2}\left(\sum_i m\vec{u}_c^2 - \sum_i m\vec{u}^{\star 2}\right) \tag{3.10}$$

where m is the mass of the cell. Equation 3.10 simplifies to

$$2E = \epsilon^2 \sum_i m|\vec{F}|^2 + 2\epsilon \sum_i m\vec{u}^\star \cdot \vec{F} \tag{3.11}$$

Solving this quadratic, we get

$$\epsilon = \frac{-\sum_i m\vec{u}^\star \cdot \vec{F} \pm \sqrt{(\sum_i m\vec{u}^\star \cdot \vec{F})^2 + 2E\sum_i m|\vec{F}|^2}}{\sum_i m|\vec{F}|^2} \tag{3.12}$$

We choose the root such that $\epsilon = 0$ when $E = 0$. This means that the $\pm$ is chosen to

be the same sign as $\sum_i m\vec{u}^\star \cdot \vec{F}$. Note that one root is equivalent to vorticity spinning the flow in one direction to add energy and in the other direction to minimize energy, whereas the other root corresponds to vorticity confinement looking to "invert" the entire flow field, which is non-physical.

3.6.1 Tracking Energy

E is updated after every step in order to track how much energy is gained or lost. For advection, we simply advect $KE = m|\vec{u}^n|^2/2$ conservatively using our newly modified version of Equation 3.2 and compare the advected value of $KE^\star$ with $m|\vec{u}^\star|^2/2$ for each cell, where $\vec{u}^\star$ is obtained via conservative momentum advection of $\vec{u}^n$.

The change in energy due to projection can be calculated at each 1-dimensional MAC grid cell as

$$\begin{aligned}
\Delta K &= \frac{\rho}{2}\left((u^{n+1})^2 - (u^\star)^2\right) \\
&= \frac{\rho}{2}\left(u^{n+1} + u^\star\right)\left(u^{n+1} - u^\star\right) \\
&= \frac{\rho}{2}\left(u^{n+1} + u^\star\right)\left(-\frac{\Delta t}{\rho}p_x\right) \\
&= -\Delta t\bar{u}\left(p_x\right)
\end{aligned}$$

where $\bar{u} = (u^{n+1} + u^\star)/2$. ΔKV, where V is the volume of the cell, is added to E. Note that this is corrected during the vorticity confinement application for the next time step.

Figure 3.10 illustrates an example of a closed box with an initial flow field. The flow is then allowed to evolve, and we add fish tracer particles to visualize the flow field. Note that the flow never dies down and always conserves the energy in the system, as shown in Figure 3.11.

If external forces are divergence-free, then this method is fully conservative. Non-divergence free external forces modify the pressure field, and therefore ΔK in some

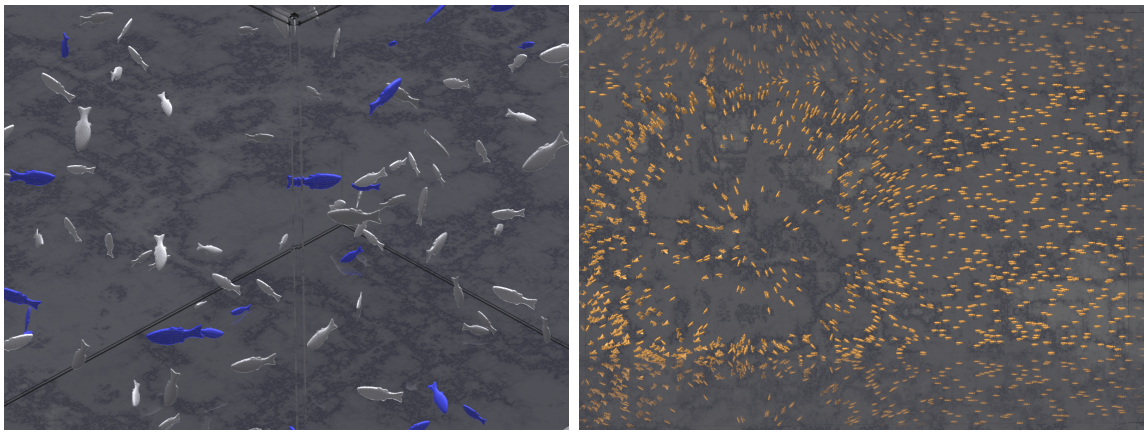


Figure 3.10: (Left) A simulation of energy conserving incompressible flow at resolution $64 \times 64 \times 64$. The initial flow field is created by starting with upwards velocities in the center of the domain and zero elsewhere. The flow is then made divergence free and simulated forward in time, and we note that conserving energy provides a sustainable flow field for long-time simulation as seen in Figure 3.11 (also see the video). Fish models are passively advected to visualize the flow field. (Right) A simulation of energy conserving free surface flow at resolution $64 \times 64 \times 128$. The initial flow field is created by dropping a ball of water into a pool of shallow water (viewed from top down).

sense includes energy added due to these external forces. For example, if we add gravity to the system (with a ground) and project the resulting flow, the projection will lose a large portion of the energy that was added with gravity, and we do not wish to treat this as energy lost. If we did, the energy would constantly increase even for an isolated system. One way to solve this is to simply do two projections, one to make the flow field divergence free before forces are added and another after the forces are added. However, this is a significant increase in cost.

Another way to find the energy lost is to figure out the total energy of the system at the start, keep track of the amount sourced in and out and store that as A. This allows us to calculate $E = A - PE - KE$ for vorticity confinement. For kinetic energy we simply calculate $KE = \sum_i m|\vec{u}^\star|^2/2$ for all cells i. For potential energy it depends on whether we are simulating smoke or water. For water we can calculate $PE = \sum_i mgh$ where g is the gravitational constant and h is the height. For smoke we use buoyancy

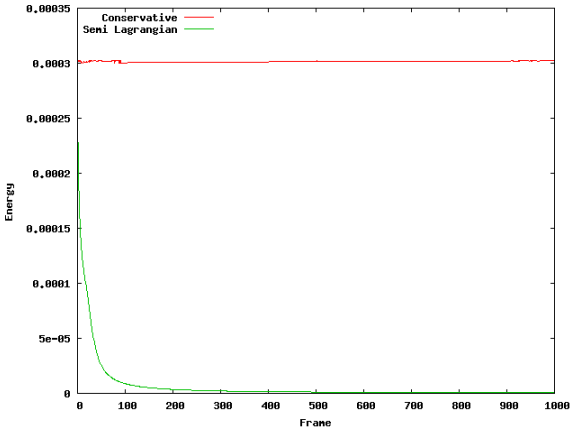


Figure 3.11: A graph of energy as a function of time for the simulation in Figure 3.10. The red line shows that our method conserves energy almost exactly for 1000 frames. Note that we do exactly conserve energy when comparing two velocity fields before projection. However, projection removes or adds a very small amount of energy as can be seen by the wiggles in the red line. For comparison we also plot the results for the same simulation using the traditional semi-Lagrangian method as a green line, and note that the energy quickly dies out.

instead of gravity. We calculate the potential energy as $PE = \sum_i m\hat{\rho}b(h_o - h)$ where b is the buoyancy constant, $\hat{\rho}$ is the smoke density and h_o is a constant. Using this equation, we see that as smoke rises from buoyancy the potential energy decreases and the kinetic energy increases to account for this loss.

Using this formulation we can simulate examples with gravity, such as the one shown in Figure 3.10 right, where we drop a ball of water into a pool of shallow water, which reduces the potential energy and increases the kinetic energy. We can also simulate energy conserving smoke as shown in Figure 3.12. In this case, we did not add any vorticity confinement to the system other than what is added for energy conservation. Note the increased amount of detail obtained using this method. This allows us to provide a method for automatically and dynamically determining how much vorticity should be added into the simulation. However, if energy is lost and there are few areas of vorticity to add energy to, adding in energy can cause undesirable noise. To alleviate this problem we do not add vorticity for these steps, instead accumulating this energy to be added back once sufficient vorticity has developed. Alternatively, one

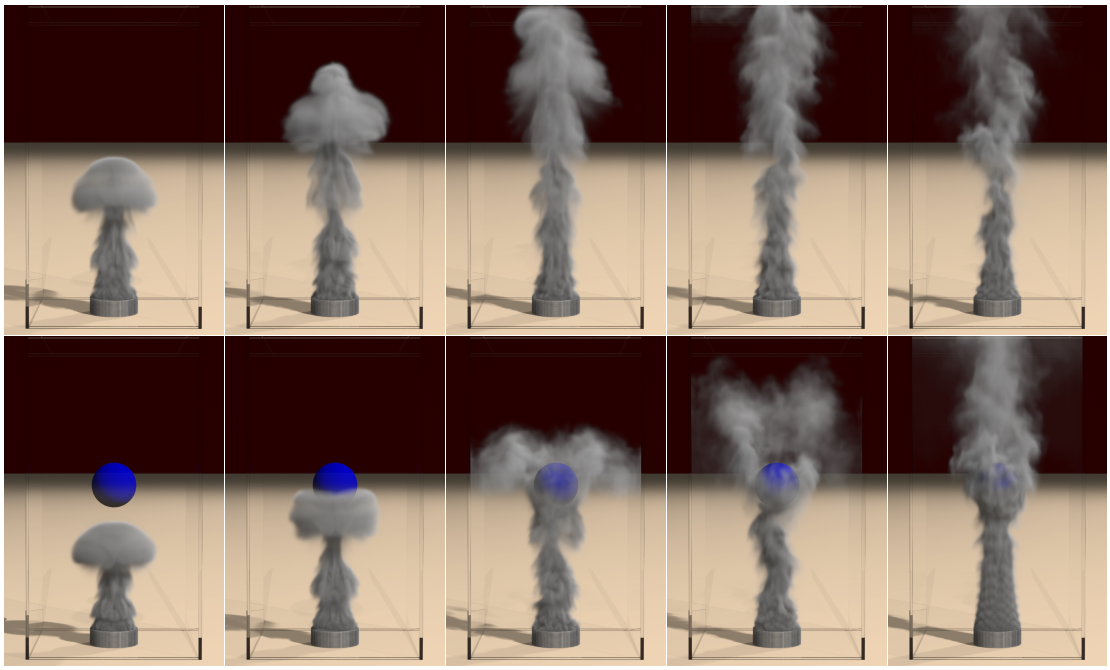


Figure 3.12: Two simulations using our method with energy conservation with smoke injected from below at resolution $128 \times 256 \times 128$. Note that no vorticity confinement was added other than that used to conserve energy. Also note that the resulting density field appears significantly less viscous than the traditional semi-Lagrangian method which explicitly adds vorticity confinement.

could target the correct energy and add the energy lost slowly over time. However, for our examples we found that there was insufficient vorticity only near the very beginning where relatively little energy was lost. Thus, we could simply add in the total energy lost as soon as vortices started appearing.

3.7 Conclusion

We introduced a novel algorithm for mass and momentum conservation in incompressible flow. We designed a new advection method using the basic building blocks used in semi-Lagrangian advection which is known to work well for inviscid flows, coarse grids and large time steps, a scenario common in computer graphics. We also

proposed a modification to the vorticity confinement model which conserves momentum. We have shown that by conserving mass and momentum we are able to run high quality simulations while taking very large time steps (at frame rate). Our technique can also be adapted to conserve momentum for water simulation. Finally, we showed a modification using vorticity confinement that preserves energy as well. By being able to run energy conserving fluid simulations with large time steps, we can create fast, interesting fluid flows that can be potentially used for other applications such as reduced order models or crowd simulations.

Chapter 4

Volume Conservation

This chapter provides a novel simulation method for incompressible free surface flows that allows for large time steps on the order of 10-40 times bigger than the typical explicit time step restriction would allow. Although semi-Lagrangian advection allows for this from the standpoint of stability, large time steps typically produce significant visual errors. This was addressed in chapter 3 for smoke simulation using a mass and momentum conserving version of semi-Lagrangian advection, and while its extension to water for momentum conservation for small time steps was addressed, pronounced issues remained when taking large time steps. The main difference between smoke and water is that smoke has a globally defined velocity field whereas water needs to move in a manner uninfluenced by the surrounding air flow, and this poses real issues in determining an appropriate extrapolated velocity field (see Figure 4.2). We alleviate inaccuracies with the extrapolated velocity field by not using it when it is incorrect, which we determine via conservative advection of a color function which adds forwardly advected semi-Lagrangian rays to maintain conservation when mass is lost. We note that one might also use a more traditional volume-of-fluid method which is more explicitly focused on the geometry of the interface but can be less visually appealing – it is also unclear how to extend volume-of-fluid methods to have larger time steps. Finally, our method utilizes the visual smoothness of a particle level set

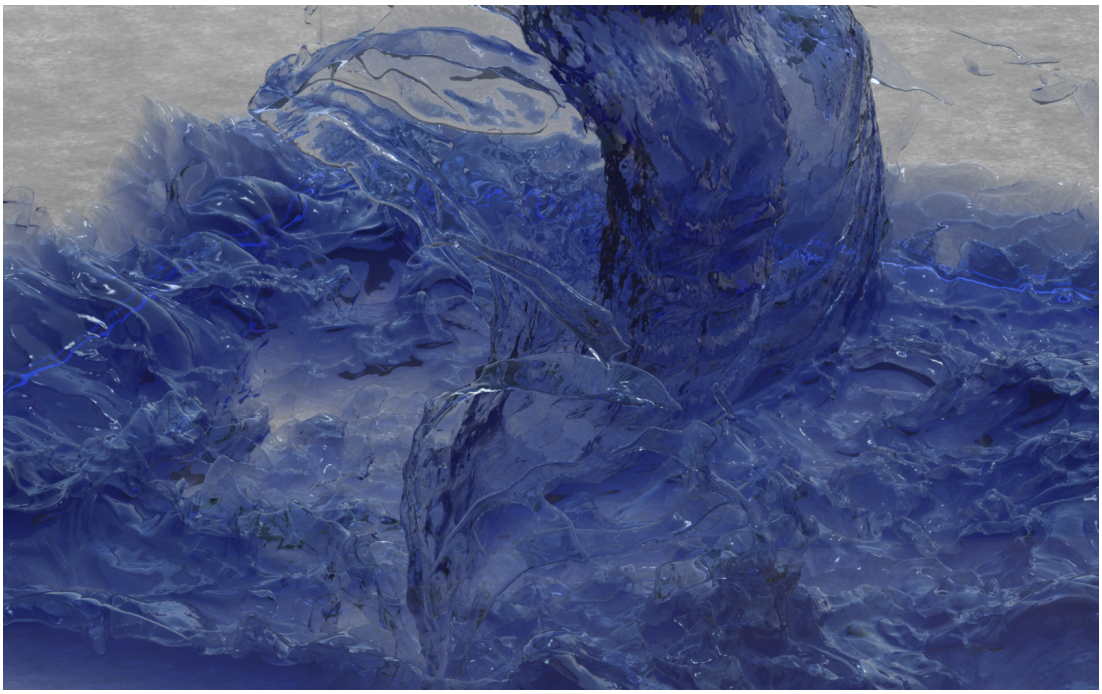
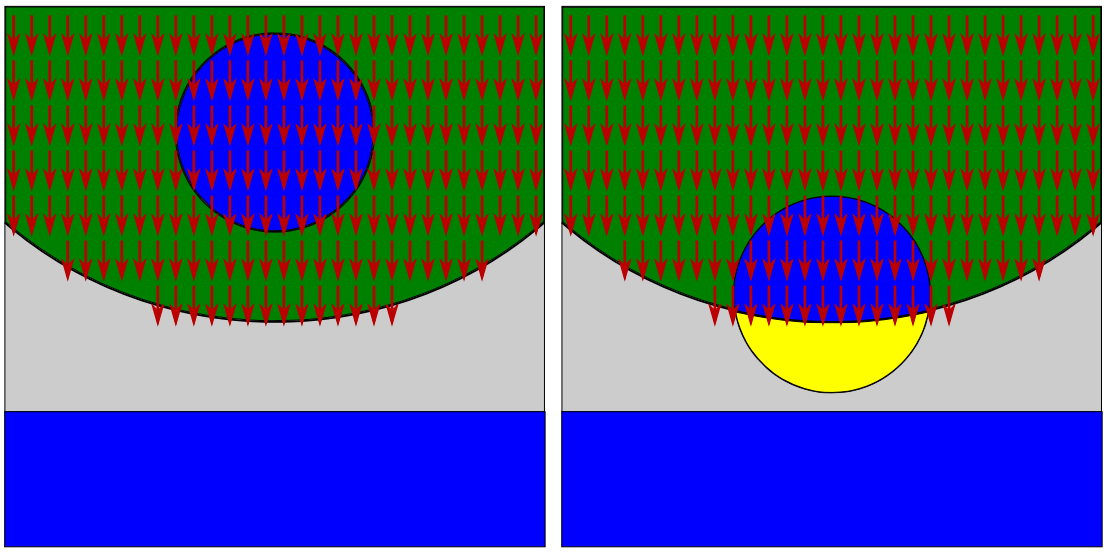


Figure 4.1: Water pouring into a box at a resolution of 512^3. This example ran with a CFL number ranging from 10-60 and demonstrates the large amount of small scale details that can be achieved by using our method.

method coupled to a traditional backward tracing semi-Lagrangian advection where possible, only using our forward traced color function solution in areas of the flow where the particle level set method fails due to the extremely large time steps.

4.1 Introduction

Physically based simulation of water has been one of the most interesting and challenging problems in computer graphics because of the amount of small scale details that can be achieved. Earlier work includes [29, 105, 28]. With the increased availability of low cost memory, multi-core machines and software suitable for MPI and threading, grid sizes can be increased achieving even greater detail by reducing the numerical viscosity that damps out the solution on coarser grids. Unfortunately, the



(a) Extrapolated velocities using closest point extrapolation

(b) Result after taking a large time step

Figure 4.2: Figure 4.2(a) shows the analytic solution for the canonical closest point extrapolation scheme used in free surface flow simulation where the velocity field in the "air" is determined by the closest point in the water surface. This results in a velocity discontinuity along the curved equidistant boundary between the green and grey shaded regions. Everything above this curve has a downward velocity obtained from gravity acceleration of the falling drop whereas everything below this curve has a stationary velocity of 0 obtained from the stationary liquid at the bottom of the figure. For advection, the analytic solution using backwards cast semi-Lagrangian rays gives the result shown in Figure 4.2(b) in blue (not yellow) where everything above the curve moves downward and everything below the curve stays stationary. The actual analytic solution is shown by the union of the blue and yellow regions in the figure and we address the loss of mass depicted by the yellow region by instead forward advecting all of that material. Of course one could forward advect the entire drop but that leads to significantly less accuracy in the blue region where backwards advection works well.

computational cost does not scale linearly in the number of grid cells as the time step size must decrease either due to stability restrictions in explicit schemes or accuracy restrictions for implicit schemes. The traditional semi-Lagrangian advection of [105] is unconditionally stable, and thus, does allow for much larger time steps. However, the visual artifacts that are produced with this method at large time steps make this

approach undesirable. Although this has been addressed in [60] for smoke simulations, there remain a number of issues for water simulation. For example, because water is treated as a free surface, velocities must be extrapolated across the interface which can create artifacts at large time steps. Figure 4.2 demonstrates one such case where a ball with constant downward velocity is falling onto a pool of water. With current methods, the bottom portion of the ball depicted in yellow fails to advect correctly due to inaccurate extrapolated velocities – which could only be accurate if multi-valued. Particle based methods, see e.g. [17, 85, 127, 71] and the references therein, also suffer from accuracy issues and visual artifacts that result from the poor sampling of particles (too sparse or too dense) at these large time steps. There are also additional difficulties in creating surfaces from particle data [123, 4], and we instead focus our efforts on grid based methods – although addressing very large time steps for particle based methods would also be interesting and useful. Note that front tracking methods [7, 116] also rely on an interface representation via particles and thus would suffer from the same sampling problems as particle based methods but would have additional difficulties with self intersection of the surface exacerbated by large time steps.

Since [60] increased the visual accuracy of their simulations by conserving both the mass of the smoke and the momentum of the fluid, our main idea will be to enforce a conservation property for the volume of the liquid. In fact, volume conservation is a well-established idea that researchers have explored for some time. They started by advecting a Heaviside or color function, adding various techniques to recompress the interface representation aiming to keep it sharp in spite of the numerical smearing. One early approach was to treat advection by looking at the volume swept by faces (fluxes) and reconstructing the interface using a simple line interface calculation (SLIC) and later a piecewise linear interface calculation (PLIC) resulting in modern day volume-of-fluid (VOF) schemes (see e.g. [91] and the references therein). In order to avoid overlapping of the flux swept volumes, these methods must be applied one dimension at a time using dimensional splitting and the time step must be restricted to be one half of that allowed by a standard explicit scheme. Various visual

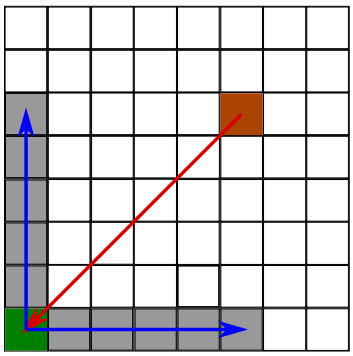


Figure 4.3: Assuming a velocity field directed diagonally downward and to the left as depicted by the orange arrow, the correct volume information for the green cell would be obtained from the brown shaded cell. Any method which looks only in orthogonal directions would be limited to ascertain information only from the grey shaded cells depicted in the picture. One could imagine an alternating dimension by dimension exhaustive approach that scans all 25 cells in the neighborhood in order to eventually find the information in the brown cell, but the semi-Lagrangian method is far more efficient.

artifacts result from this dimensional splitting, and the interface reconstruction ends up being discontinuous across cell boundaries. [110, 109] worked to ameliorate these issues by coupling VOF methods to level set methods where a locally constructed and advected level set function is used to compute surface normals alleviating some of the flotsam and jetsam (see also [78, 80]). However, their improvements did not completely eliminate such visual artifacts which would be exacerbated with very large time steps, especially when dimensional splitting is used. Another possible approach involves enforcing volume preservation globally as demonstrated in [45]. However, this would isotropically expand the blue region in Figure 4.2 as opposed to adding the yellow region to reconstruct the circle. Therefore, we do not take a VOF type or global volume control approach to the problem but instead use a color function type method that relies on semi-Lagrangian advection and sharpening (see Figure 4.3). [84] also proposed an approach to color function advection using a conservative flux based approach which is known to suffer from overshoots and undershoots – the thing VOF methods were created to address. [84] referenced [30] as a method for taking larger time steps; however, this dimension by dimension approach suffers from the

same limitations discussed in Figure 4.3.

Simply utilizing a color function type approach would suffer from all the same problems that originally led to the improved VOF methods or coupled level set VOF methods, and we would prefer to use a more visually pleasing particle level set method such as that proposed in [22, 20] (or even a variant such as Marker Level Set [79]). Therefore, we start with the work of [22, 20] addressing various issues with large time steps, noting that without conservation the issue in Figure 4.2 seems implausible to address (see Section 4.3.1). Thus, we hybridize this method with a color function type approach in order to achieve an approximation of the yellow region in Figure 4.2, and subsequently outline the details for two way coupling of the methods. Our color function approach has many benefits over any dimension by dimension or flux based approach because of the utilization of the semi-Lagrangian advection from [58] which is especially useful for large time steps (again as shown in Figure 4.3). The resulting method enables the simulation of incompressible flow with time steps over an order of magnitude larger than implied by the CFL condition.

4.2 Free Surface Flows

Our fluid solver is based on the particle level set method [22] and proceeds by solving the inviscid incompressible Navier-Stokes equations, which are given by

$$\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho}\nabla p + \mathbf{f}, \tag{4.1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{4.2}$$

where $\mathbf{u}$ is the velocity field of the fluid, ρ is the density of the fluid, $\mathbf{f}$ is the sum of any external forces (such as gravity) scaled by ρ, and p is the fluid pressure. We solve these equations by first calculating an intermediate velocity field $\mathbf{u}^\star$ via

$$\frac{\mathbf{u}^\star - \mathbf{u}^n}{\Delta t} + (\mathbf{u}^n \cdot \nabla)\mathbf{u}^n = \mathbf{f}. \tag{4.3}$$

This equation is typically solved using the standard backward semi-Lagrangian advection scheme [105] which requires an accurate approximation of the velocities in the region traversed by the fluid during a given time step. In the region not occupied by the liquid, one needs to obtain velocities for use in semi-Lagrangian advection, and this is typically done using any closest-point velocity extrapolation scheme within a band near the surface.

We then subsequently apply the pressure forces via

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^{\star}}{\Delta t} = -\frac{1}{\rho}\nabla p, \tag{4.4}$$

where the pressure is calculated by solving the Poisson equation

$$\nabla \cdot \frac{1}{\rho}\nabla \hat{p} = \nabla \cdot \mathbf{u}^{\star}, \tag{4.5}$$

where $\hat{p} = p\Delta t$. We also use the pressure modifications laid out in [32] to achieve second-order accuracy at the surface.

4.3 Taking Large Time Steps

Generally speaking, there are two main steps in the simulation. The first step is to advect all quantities forward including the fluid velocities, level set, and particles after which the second step is the projection. Both of these steps require modifications for large time steps.

4.3.1 Problems With Extrapolation

The standard semi-Lagrangian advection requires looking back along characteristic rays. However, for free surface flows, the air region is not modeled and therefore does not have accurate velocities which need to be approximated. One method of

obtaining these velocities is to add an air flow such as in [41, 70]. The problem with this is that for many graphics simulations, we expect the air to have very little influence on large bodies of water. For example, a falling water drop will deform in a surrounding airflow, and although this can be ameliorated by surface tension, this adds an additional cost and complexity to the simulation.

Instead, as mentioned above, we fill the air region with a pseudo-velocity obtained via the canonical closest point extrapolation scheme which allows the water to move effectively unimpaired by the surrounding air. Typically one extrapolates about three grid cells; however, the extrapolation bandwidth must increase as the size of the time step increases. A previous approach [10] addresses the large time step problem by performing a global extrapolation; however, as seen in Figure 4.2, even with the analytic solution to the global problem, this method will fail. Other approaches such as [12] which uses a multi-valued velocity field in order to handle collisions such as those shown in Figure 4.2, will fail in other cases such as when two spheres collide with each other. In this case, the algorithm would either cause the spheres to incorrectly pass through each other without a collision or overlay the spheres with each other resulting in a violation of volume conservation. The only way to properly handle the collision is to construct an extrapolation field that has knowledge of incompressibility.

This was addressed by [109, 94] who used divergence free extrapolation to alleviate the issue by ensuring that the extrapolated velocity is discretely mass conserving. However, we observed that their proposed solution deforms a falling spherical drop well before merging takes place. This is because divergence free extrapolation solves a second projection in the narrow band around the outside of the drop with a free surface boundary condition applied on the outside of this narrow band. If this boundary condition is enforced by setting the cells to $p = 0$ without considering the interface geometry, the drop deforms. This deformation of the drop can be resolved by using the second order cut cell method from [32] on the outer edge of the band (which we point out for the first time in the paper). While this works to improve merging for the small extrapolation bands required when taking small time steps, when it is applied globally or over the large extrapolation bands required when taking large time steps

to remove the discontinuity shown in Figure 4.2, it will produce a velocity field that deforms the drop far before it gets close to the water surface. Thus, this method, while applicable for improving merging at small time steps, does not alleviate the issues with large time steps.

4.3.2 Advection

As discussed in the previous section, closest-point extrapolation methods are unable to approximate a velocity field far away from the interface that is usable with large time steps. Thus, we only extrapolate within the standard three grid cell band. Then, when advecting the velocity we use the conservative method of [60]. They show that this method works for globally defined velocity fields and large time steps, as well as small time steps with free surfaces, but do not show that it works for large time steps with free surfaces. However, the method works in that case as well requiring no real extensions or modifications. Basically, when cells within the three grid cell band look back to advect velocity forward, any velocities that are missed that would have been updated from a larger band are updated in the second step of forward advection which is required in order to conserve the momentum.

Second order Runge-Kutta particle advection can fail at large time steps if an accurate fluid or extrapolated velocity is not present. We default to advecting the particles with forward Euler if a velocity sampled within an RK2 step is outside the extrapolated velocity band allowing us to maintain the higher-order accuracy of the RK2 method in the presence of accurate velocities while improving the robustness of the method when such velocities are absent. Level set advection also has problems at large time steps. While the particle level set method maintains the shape of the air-water interface at small time steps, it does not typically preserve the amount of volume within the liquid region. At small time steps, these changes in volume are small and are unnoticeable in the animation. However, when the particle level set method is advected for a large time step, this volume loss becomes more apparent and detracts significantly from the visual plausibility of the simulation. Furthermore,

applying a conservative advection scheme to level set itself is insufficient to resolve this volume loss because the conservation of signed distance values does not imply conservation of any physical quantity such as volume. We resolve this by advecting a color function using the conservative method of [60] in conjunction with the particle level set surface as discussed in Section 4.4.

4.3.3 Projection

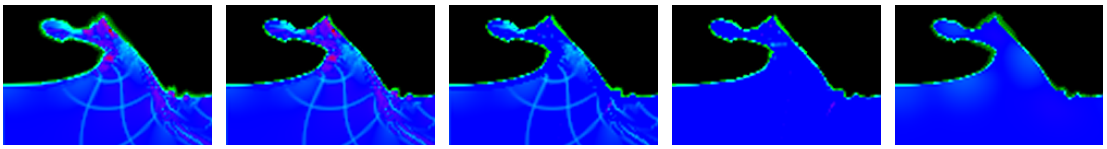
We solve the projection step only in liquid regions while enforcing a free surface boundary condition. As a result, small regions of "air" (not in a liquid region) can create large divergences. While this is acceptable with small time steps due to the restrictions on the motion of the fluid, this results in a large amount of volume loss when taking large time steps. One method of solving this problem is to explicitly fill the "air" region and enforce a target divergence [71] such that the cell is exactly filled in a given time step. However, this method requires the algorithm to detect whether or not the cell will be filled or overfilled for a given velocity field and time step. While simple algorithms, such as thresholding based on the size of the "air" region and the CFL number can be used as an approximate detection scheme, it is difficult to obtain a set of parameters that are suitable for a general fluid flow. We use a threshold that causes the algorithm to only fill small "air" pockets. For larger regions, we instead enforce volume conservation using our conservative color function treatment. This can sometimes lead to cells with $V > 1$, and this is treated along with other advection errors as described in Section 4.4.

4.4 Color Function

In addition to the level set function, we also evolve a color function V which represents the fraction of liquid contained within a region (usually a single cell in the case of a uniform grid). Although the color function can be advected using any method, volume

is conserved only if the method is conservative. However, numerical smearing, even in a conservative method, will cause inaccuracies in V's spatial distribution. One of the main approaches for increasing accuracy are the VOF type methods. Early on, SLIC approximated the interface in each cell using a plane defined by a normal along one of the Cartesian directions, placing it in a location that accurately represented the fraction of volume in a cell. Then, advection was carried out by intersecting the material in a cell with the volume swept out by a cell face along its normal during a time step. In order to be conservative, this needed to be done in a dimension by dimension manner so that the flux-swept volumes did not intersect each other. This leads to a time step restriction corresponding to a CFL number of .5. Additionally, the dimensional splitting and other inaccuracies lead to flotsam, jetsam, and other visual artifacts. SLIC was improved via PLIC, which allows the interface to be represented by any single plane within a cell. However, even these improved methods still have problems with flotsam and jetsam (see [91] and the references within). [110] attempted to improve the VOF method with the utilization of a level set method. The level set is constructed at each time step based on the interface geometry, advected forward in time, used to compute a normal in each cell to define a tangent plane for the fluid, and discarded until the next time step. The surface reconstructions of this CLSVOF method are typically not as smooth as those from the particle level set method as can seen by the examples of [78, 80], and the resulting visual artifacts will be highly exacerbated at large time steps. While previous methods have achieved limited success for CFL numbers slightly larger than 1 [10, 30], the artifacts will be highly exacerbated for CFL numbers in the range of 10-40 as we take in our examples.

Our method using the color function V is conceptually similar to the VOF method. If $V = 1$ or $V = 0$, the region contains either all liquid or all "air" respectively. If $0 < V < 1$, the region contains both liquid and air. First, to compute the initial data, we compute the color function from a piecewise linear rasterization of the level set. Note that when ϕ is changed for fluid sources, the color function also needs to be modified to maintain consistency. Then we advect the color function forward in



(a) (b) (c) (d) (e)

Figure 4.4: Correcting Numerical Dissipation in the Color Function: Figure 4.4(a) shows the color function after advection. Due to numerical dissipation, there are regions inside the level set ($\phi \leq 0$) with a color function value $V = 1$ (blue), $V < 1$ (cyan), and $V > 1$ (magenta) along with regions outside the level set ($\phi > 0$) with a color function value $V = 0$ (black) and $V > 0$ (green). Figures 4.4(b), 4.4(c), and 4.4(d) demonstrate the results after applying our first, second, and third compression step respectively. We then apply a volume-conserving diffusion algorithm which corrects for errors in compression (usually located in regions of high curvature) to obtain a more accurate color function surface representation shown in Figure 4.4(e).

time as per

$$V_t + \mathbf{u} \cdot \nabla V = 0 \tag{4.6}$$

using the conservative method of [58]. Note that we do not use any of the diffusion methods used when advecting smoke density [60]. Our experiments show that diffusion is detrimental in the case of color function advection, since diffusion is based on the heat equation which tends to move the interface faster in regions of higher curvature than lower curvature. This problem is alleviated for momentum advection because diffusion is not applied across the interface, which is well determined before the momentum advection step. The conservative advection method can also result in excess volume accumulating where fluid collides with a rigid body due to the clamping of semi-Lagrangian rays to the surface of the object. This can result in an undesirable viscous fluid appearance which we avoid by performing explicit damped collisions with rigid bodies.

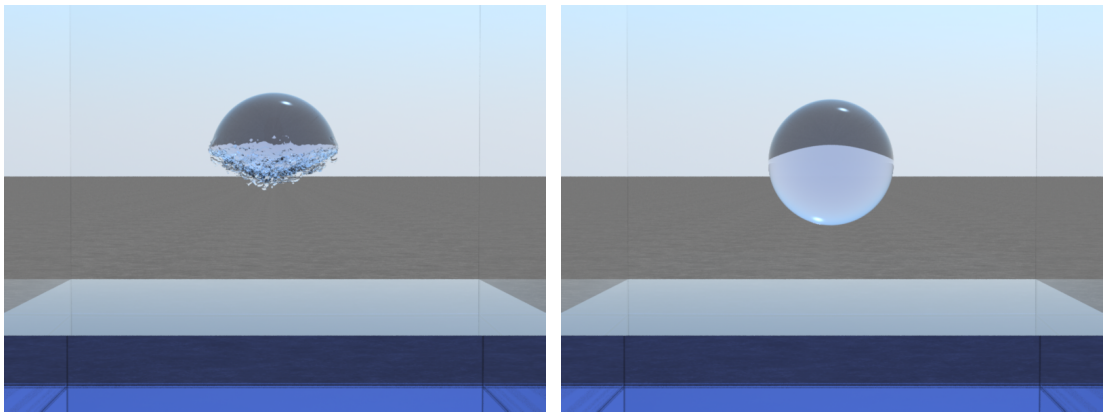
After the color function has been conservatively advected forward in time, it will still smear out due to numerical dissipation. We apply a compression procedure diagrammed in Figure 4.4 to the color function after it has been advected forward in time. First, we compute the .5 isocontour of the color function. Then, we aim to set all cells within the .5 isocontour to have $V = 1$ and all cells outside the .5 isocontour

to have $V = 0$. There are two primary discrepancies that need to be addressed. The first problem is that V values outside the isocontour can be more than 0 due to numerical smearing. The second problem is that V values inside the isocontour are not necessarily 1 (can be either more than or less than 1). Although interior cells with $V > 1$ can result from errors in advection, this can also occur from the collapse of air pockets as discussed in Section 4.3.3. To correct for these errors, we compute differences between the target (0 or 1) and current color function value and then move this difference in the direction of the gradient of V to the interface. For $V > 0$ outside the interface, we move in the direction of the gradient of V until we are at a location $2\Delta x$ interior to the $V = .5$ isocontour. Then, we distribute the difference in color function value to the neighboring cells around this location (excluding cells lying within rigid bodies) in weighted proportions until $V = 1$ in each cell. If all of the difference cannot be distributed at these neighboring cells without overflowing the cells, the remainder of the difference is placed in a location obtained by marching outwards along the gradient of V. For the cells on the interior of the $V = .5$ isocontour with $V > 1$ the difference in color function value $V - 1$ is distributed to the surface in the same way starting from a distance of $2\Delta x$ inside the surface. In the third case, where $V < 1$ for interior cells, we fill V for each cell to 1 using color function values from the surface by distributing a negative value $V - 1$ starting $2\Delta x$ outside the interface and marching inwards in a similar manner in the opposite direction as the first two cases. Both our advection scheme and compression scheme guarantee that values of $V < 0$ cannot occur; however, the compression method can be modified to account for these if a different advection scheme that can result in negative values is used. The compression is done in a Gauss-Seidel fashion using a single iteration (multiple iterations could be used but are not needed since its done at every time step), and thus the order in which the cells are visited matters. One might want to use a method such as Gauss-Jacobi to reduce bias but this will require more iterations, and we have found that Gauss-Seidel is satisfactory for our purposes. This is partially because we couple it with particle level set method for accuracy whenever possible (see Section 4.5).

4.5 Coupling

First, we advect both ϕ and V forward in time using semi-Lagrangian advection for ϕ and conservative semi-Lagrangian advection for V. We then correct the values of ϕ using the particles as in the particle level set method, and subsequently reinitialize ϕ. However, we stress that the second particle correction of the reinitialized level set is not yet applied, and the color function is not yet compressed. At this point, we construct a distance function from the color function in almost the same way that [110] construct it at the beginning of the time step. That is, we first use a VOF construction of the color function, computing a normal from the gradient of V and the placement of the tangent plane in the cell from the value of V itself, and then initialize values of the signed distance function ϕ^V based on the tangent plane in that cell. Nearby cells are initialized with values from the fast marching method. Unlike [110], we compute the normals directly from the gradient of V as opposed to an advected ϕ. This is because an advected ϕ function would be highly inaccurate due to the issue of velocity extrapolation.

While the color function does not represent the interface as accurately as the particle level set method, it is often the only reasonably accurate interface representation available when taking large time steps demonstrated by the yellow region of Figure 4.2. Thus, our aim is to use the particle level set method wherever possible in order to gain visually pleasing surfaces and only to use the color function to represent the surface where the level set representation has lost large amounts of mass such as shown in Figure 4.5. Generally speaking if the interfaces agree we use the particle level set representation of the interface. We say that the two interfaces agree if they do not differ by more than $.5\Delta x$ in distance. At every cell (i, j, k) in the computational domain, we compute a blending parameter $\alpha_{i,j,k} = |\phi_{i,j,k} - \phi^V_{i,j,k}|/\Delta x - .5$ and clamp it in the range $[0, 1]$. Subsequently, we compute the blended level set function ϕ^B via $\phi^B_{i,j,k} = (1 - \alpha_{i,j,k})\phi_{i,j,k} + \alpha_{i,j,k}\phi^V_{i,j,k}$. Note that the minimum pre-clamped value of $\alpha_{i,j,k}$ is $-.5$ and thus when the interface values agree to within half a grid cell, we use the level set version. For $\alpha_{i,j,k} \geq 1$, we use the color function representation, and



(a) PLS method (b) Our method

Figure 4.5: Figure 4.5(a) shows the result we get after taking a large time step using the PLS method. Because closest-point extrapolation discussed in 4.3.1 is unable to provide a good approximation of the velocities in the air region, backward semi-Lagrangian advection fails to accurately advect the level set causing the ball to become clipped at the bottom. Figure 4.5(b) shows the same frame using our method where we are able to reconstruct and subsequently maintain the shape of the ball even when taking a very large time step.

otherwise we interpolate between them. In order to avoid visual kinks in ϕ^B, α is smoothed locally on the grid before linearly interpolating ϕ and ϕ^V to obtain ϕ^B.

Next, ϕ^B is initialized to a signed distance function. In order to prevent spurious particle corrections to the blended level set representation, a particle is deleted if $\alpha > 0$ at the closest surface cell. This signifies that the closest interface in the cell has changed from the using the ϕ obtained from PLS and is instead using the color function to help define the interface, thus rendering the particle correction incorrect in that cell. Particles are then only reseeded in the areas with $\alpha > 0$ in order to preserve sharp features that cannot be captured by the level set alone. Finally, the reinitialized ϕ^B is corrected via particles in the usual fashion of the particle level set method.

In summary, we advect the level set function ϕ, correct it with particles and reinitialize it, advect the color function V, contour it with a VOF method and produce a signed distance function representation ϕ^V, blend ϕ and ϕ^V to obtain ϕ^B, reinitialize the

result, delete particles where the color function representation is being used, reseed particles around the new interface generated by the color function, and correct the combined level set using the particles. We denote the final result once again as ϕ.

At this point, we have the desired level set reconstruction but a smeared out color function. We use the compression method listed in Section 4.4 to compress the color function using the more accurate normals of ϕ instead of the less accurate gradients of the color function. Moreover, we can accelerate the process of finding the zero isocontour and nearby locations using the fact that ϕ is a signed distance function. We stress that this compression does not use isocontours or geometric information from V in any way but instead strives to compress V into the newly generated ϕ.

While this compression method works well, we have noticed that in regions of high and low curvatures compression tends to produce errors near the interface such as $V > 0$ color function values accumulating on the peaks of the waves. (see Figure 4.4). Therefore, we propose a diffusion based algorithm, similar to the ones found in [23, 60] to improve the color function's surface representation. We take the union of the surfaces given by ϕ^V and ϕ and use compression outside this region and diffusion inside. We diffuse the error $e = V - V^\phi$, where V^ϕ is the color function obtained from a piecewise linear rasterization of ϕ, in the color function relative to the level set close to the surface. We have also experimented with solving the Poisson equation with the appropriate target divergences to generate a velocity field inside the union which works as well but is much more expensive than using this sweeping method. After the compression and diffusion steps, the color function yields an improved surface representation (as shown in Figure 4.4) and in order to improve the visual appearance of the final surface, we repeat the blending process using the current V and ϕ.

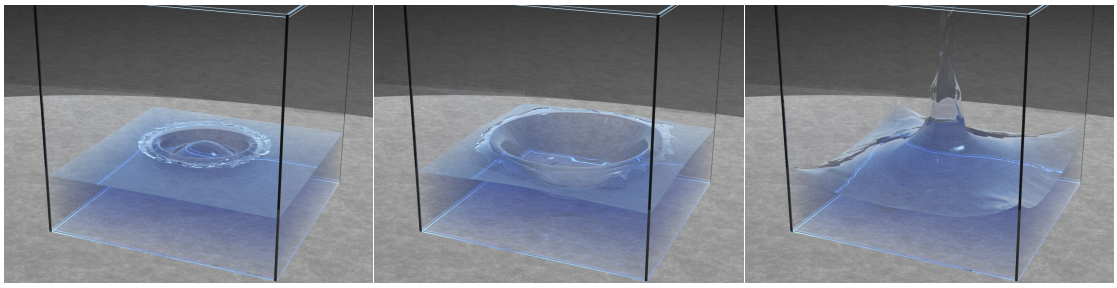


Figure 4.6: Sphere of water dropped onto a stationary flat surface of water at a resolution of 256^3. This example ran with a CFL number ranging from 10-40. The early part of the simulation when the water first starts to fall was simulated with a smaller CFL number. The later parts of the simulation were ran with a higher CFL number.

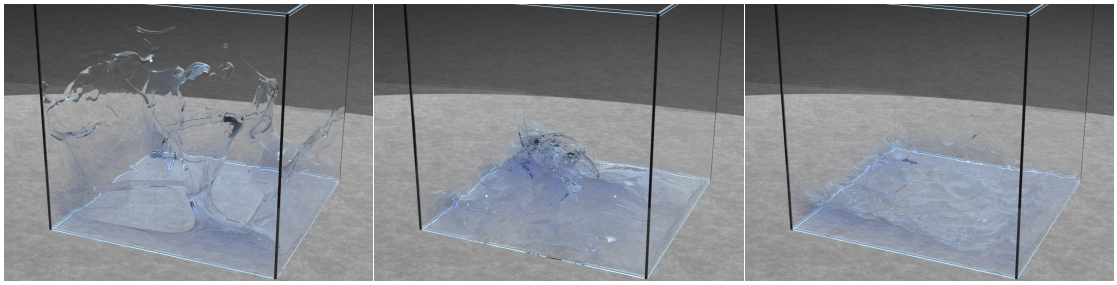


Figure 4.7: Dam break water example at a resolution of 256^3. This example ran with a CFL number ranging from 10-40. The early part of the simulation when the water first starts to fall under the influence of gravity was simulated with a smaller CFL number. The later parts of the simulation were ran with a higher CFL number.

4.6 Results

We demonstrated our method on a number of 3D examples as seen in Figures 4.6, 4.7, 4.9, 4.10, and 4.8. We first ran a baseline simulation for each example using the standard PLS method at a CFL number of 1 at resolutions of 64^3 and 128^3. However, these simulations were too slow to run in a practical amount of time at the higher resolutions such as 256^3. A CFL number of 1 means that advection does not move information farther than 1 grid cell in any time step. Then, we then ran both the

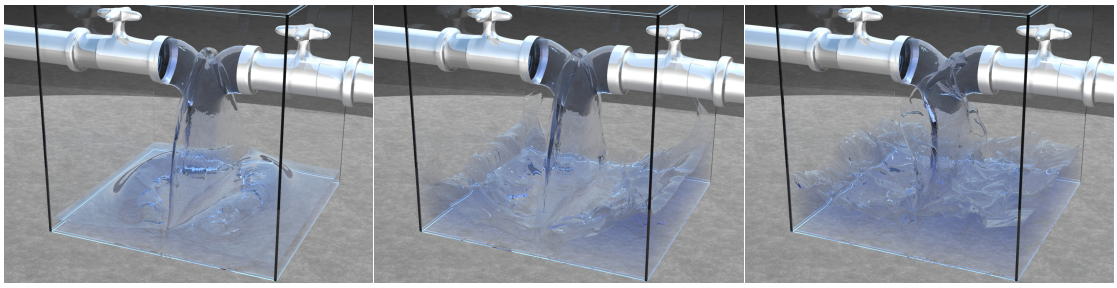


Figure 4.8: Water pouring into a box from two opposite-facing sources at a resolution of 256^3. This example ran with a CFL number ranging from 10-40. The earlier section of the simulation when the sources are first activated until the water first hits the bottom of the container was simulated with a smaller CFL number. The more active parts of the simulation which occur after the water from the sources hits the bottom of the container were ran with a higher CFL number.

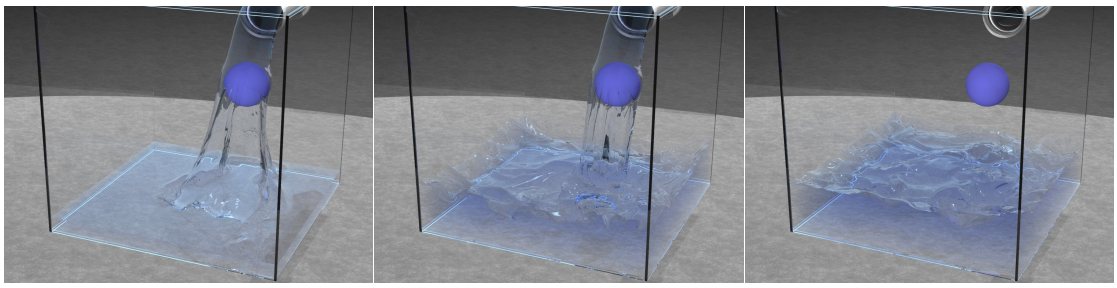


Figure 4.9: Water pouring into a box over a rigid sphere at a resolution of 256^3. This example ran with a CFL number ranging from 10-40. The earlier section of the simulation from when the source is first activated until the water first hits bottom of the container and the later section after the source was turned off and the water starts to calm were simulated with a smaller CFL number. The more active parts of the simulation which occur between the water from the source hitting the bottom of the container and the source turning off were ran with a higher CFL number.

standard PLS method and our method at a high CFL number of 40 at resolutions 64^3, 128^3 and 256^3. For resolutions of 64^3 and 128^3, this equates to running at the frame rate (meaning one time step per frame) since the maximum CFL number was approximately 5 for 64^3 resolution simulations and approximately 20 for 128^3 resolution simulations. We also ran our method at a resolution of 512^3 in order to demonstrate the large resolution simulations that can be obtained by our method at high CFL numbers. For this simulation, we primarily ran with a CFL number of

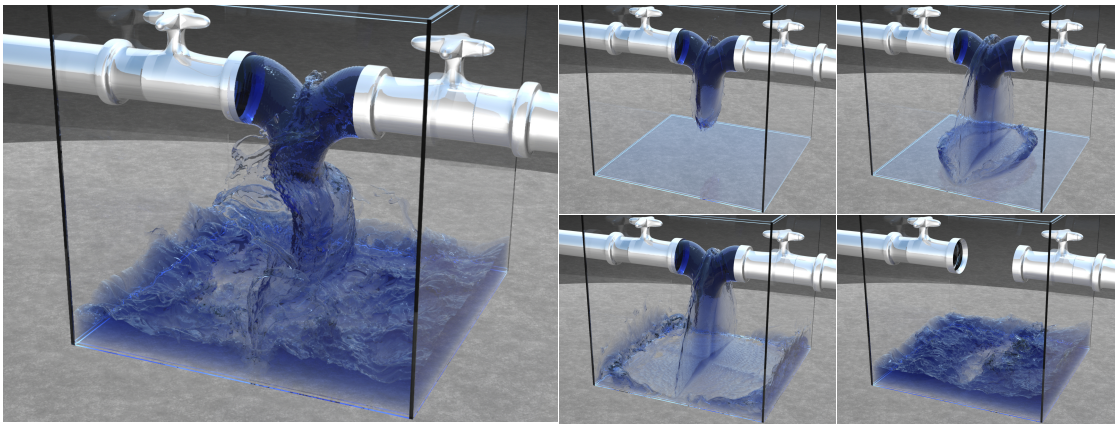


Figure 4.10: Water pouring into a box from two opposite-facing sources at a resolution of 512^3. This example ran with a CFL number ranging from 10-60. The earlier section of the simulation when the sources are first activated until the water from the sources collides was simulated with a smaller CFL number. The remainder of the simulation was ran with a higher CFL number. Note that we increase the amount of absorption during rendering in order to make the fine scale details more apparent.

60. Figure 4.11 shows a comparison between our method and the particle level set method using similar amounts of computation. Figure 4.12 shows a comparison of the volume between the traditional PLS algorithm and our method at a resolution of 128^3. Note the large amount of volume lost without using our method. Compared to the standard particle level set method, the additional steps in our method cause it to execute about twice as slowly for a given time step. However, since our time step is allowed to be 40 times larger, our method approximately achieves a 20 times increase in performance on frames where the CFL number is close to 40 or greater. For frames in specific simulations where the CFL numbers are smaller than 40 (e.g. stationary sphere before it begins falling into water under the influence of gravity), we achieve smaller increases in performance. We emphasize that this is due to the maximum time step size being restricted by the user-specified frame rate for the simulation. Similarly, for the 256^3 resolutions, we ran both the standard PLS method and our method. Note that when running this large of a simulation, our CFL number becomes limited because our current MPI implementation divides the computational domain into a number of smaller domains and we have not generated the code to

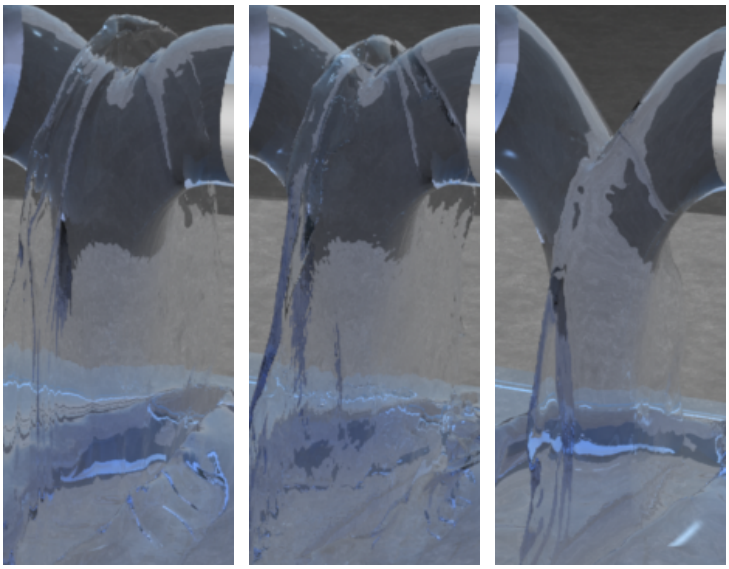


Figure 4.11: Comparisons between our method and the particle level set method using similar amounts of computation. The left figure shows the correct answer obtained by running the particle level set algorithm at a resolution of 256^3 at CFL 1. The middle figure shows the results of running our algorithm at a resolution of 256^3 at CFL 16. This requires a similar amount of computation to running the particle level set method at a resolution of 128^3 at CFL 1 which gives the results shown in the right figure. Our algorithm has the same degree of numerical viscosity as the 256^3 particle level set simulation but requires the same amount of computation as the 128^3 since the resolution is doubled in each dimension and the CFL condition becomes twice as strict.

cross more than one processor boundary (although this can be done). The limitations on the CFL number of simulations due to the size of the MPI domains bring up the important issue that with a CFL number equal to the size of one of these subdomains, a ghost cell implementation requires 27 times more data for ghost cells than the actual simulation (i.e. each grid being replicated in all Cartesian directions and partially replicated in the diagonal directions) which can quickly deplete memory resources. In the standard ghost cell implementation, it might be advantageous for MPI code (although unnecessary for threaded code) to change the boundary process to have each grid request data from the neighboring grids that contain the needed grid cells, sending that request out to other processors which send the information back. Even

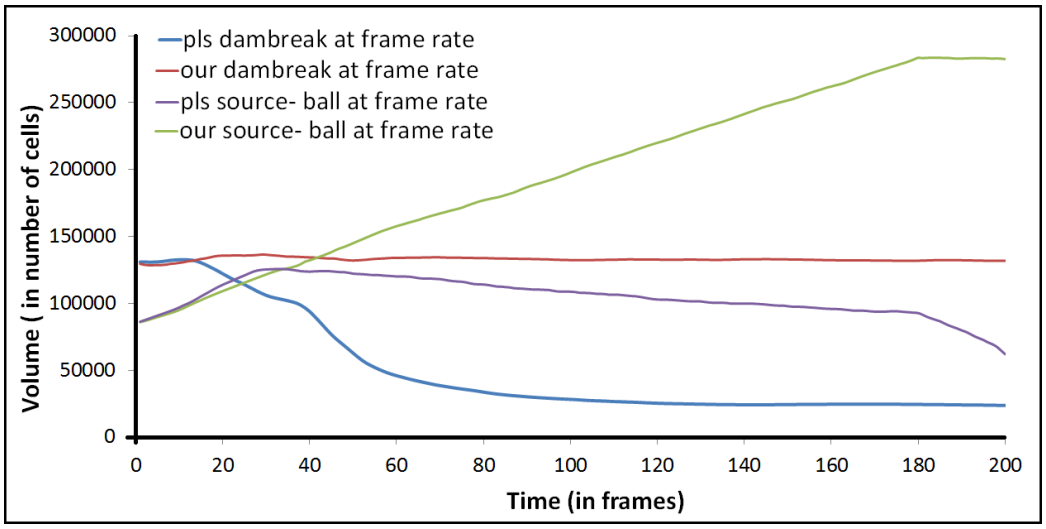


Figure 4.12: Volume vs. time for four different simulations. The red and blue lines represent the liquid volume present in the dam break simulation at frame rate. Notice how our method (red) fully conserves volume while the PLS algorithm (blue) loses a tremendous amount of volume. The green and purple lines represent the volume present in a simulation with water from a source flowing over a ball. Notice how the volume increases linearly until the source is turned off when using our method (green) but decreases slowly when using the PLS algorithm (purple).

so, if every grid cell in the domain is requesting data from another processor, this is equivalent to copying entire grids from one processor to another at every time step, and thus while doubling the CFL number does halve the required number of pressure solves and other steps such as those involved in the particle level set method, it increases the communication cost. Thus, one cannot indefinitely increase the CFL number although we noticed issues in visual accuracy at lower CFL numbers than would lead to concerns with communication bottlenecks. For example, during the first step of a simulation with a source, the velocities are purely horizontal and thus when a large CFL number is taken, the liquid will move a large distance parallel to the ground despite the fact that the fluid should be falling. One alternatively could apply gravity first but that would mean advecting in a divergent flow field which generates problems with objects advecting through solid walls. Instead, these problems are partially resolved by running with a smaller CFL number when such visual artifacts

are present which typically occurs in slow moving fluid flows.

It is important to note that although we can maintain a similar time step size at larger resolutions, the algorithm does scale with the number of grid cells. Therefore, the simulation is a factor of 8 slower when the grid size doubles for a fixed time step (which requires doubling the CFL number of the simulation). However, a number of methods including [59, 77] improve the scalability when the resolution changes. These methods can be used in conjunction with our method to create even faster simulations.

4.7 Conclusion

We have presented a novel method method for accurately simulating incompressible flow even when taking time steps that are more than an order of magnitude larger than implied by the CFL condition. We accomplish this by conservatively advecting a color function representing the fraction of volume contained in each cell along with the particle level set and using the color function to compute an accurate representation of the surface where the particle level set representation fails to suffice. This allows our method to maintain volume and subsequently achieve visually accurate results. Moreover, we have presented a general framework for combining an interface tracking method with a volume tracking method. Investigating the use of other interface tracking methods and volume tracking methods for use within this framework is a potential subject of future work.

Chapter 5

Coarse Grid Projection

Large scale fluid simulation can be difficult using existing techniques due to the high computational cost of using large grids. This chapter presents a novel technique for simulating detailed fluids quickly. Our technique coarsens the Eulerian fluid grid during the pressure solve, allowing for a fast implicit update but still maintaining the resolution obtained with a large grid. This allows our simulations to run at a fraction of the cost of existing techniques while still providing the fine scale structure and details obtained with a full projection. Our algorithm scales well to very large grids and large numbers of processors, allowing for high fidelity simulations that would otherwise be intractable.

5.1 Introduction

Physical simulation of fluids is one of the most interesting and challenging problems because of the amount of small scale details that realistic fluids exhibit. Although many authors such as [29, 105, 25] have used grid-based techniques to produce visually compelling results, the size of the grids that these techniques can use is limited by the amount of computational power available.

As a consequence many authors have developed techniques that add details to these simulations with noise. For example, Kolmolgorov noise (see [106, 56, 95]) and curl noise (see [6]) can be used to enhance the visual fidelity of fluid simulations by coupling the noise to the incompressible Navier-Stokes equations producing a more detailed flow. Alternatively, [50] and [87] determine where to add noise using information from the existing simulation and then add it as a post-process which allows them to add noise where it is best suited. Other techniques such as [100] both determine where to add noise and couple the noise to the Navier-Stokes equations. All of these techniques are successful at adding details but are nonphysical and can produce significantly less realistic results than simply simulating with a higher resolution grid.

Another approach is to improve the baseline simulation on the existing grid. This can be done by using higher order methods in space, such as BFECC, QUICK, and MacCormack methods (see [18, 46, 101, 82]), or in time, such as Runge Kutta. One could also work to maintain certain invariants such as energy (see [83]). Although these methods increase the accuracy and fidelity of the resulting simulation, they are more expensive than traditional fluid simulation and are still limited by the Nyquist frequency of the grid. To increase the grid resolution while keeping the increase in cost to a minimum, adaptive grid techniques were introduced such as AMR [3] and octrees [69]. These techniques are effective at reducing the computational cost in cells where there is not much detailed motion while maintaining details where needed. However, the increased complexity of using these complicated structures both increases computational cost and hinders the ability to design robust numerical methods.

In contrast to grid-based approaches, particle-based methods, which were first introduced in [96] and later expanded on by [17, 85], are not limited by the resolution of the grid. However, these approaches do not store the connectivity of the surface and require additional computational costs to keep track of the surface and to re-mesh. Furthermore, these methods increase in cost as they approach the incompressible limit, and thus many authors use weakly compressible equation-of-state formulations. There has also been work on combining particle and grid-based approaches. [31] uses

a combination of grid and particle-based approaches to produce realistic simulations at variable speeds. [102] introduced a particle-based method to add vorticity to grid-based simulations using particles that are able to accurately track the vorticity of the simulation and [90] extended it to work well with objects. Although these techniques do add details at little cost, they are still limited by the base resolution of the simulation.

In order to handle very high resolution grids, [115] introduces a reduced order model that can handle very large grids at a small cost. However, the use of basis functions lacks the physical realism and details that are achieved through traditional grid-based techniques. Moreover, [49] state that model reduction limits the motion of the model to a pre-specified basis, and any attempted motion outside this basis is artificially suppressed. They further state that since the main advantage of physically based techniques is their ability to automatically capture novel dynamics, such suppression is unacceptable. [95, 42] introduce methods that can run large scale two-dimensional algorithms and then can extend the results to three dimensions. Although these do produce visually compelling results in some instances, the two dimensional simulations are not a very good approximation of the three dimensional behavior for the more general case.

Recently, there has been a large interest in methods for creating a higher resolution result from a lower resolution simulation. [88] introduced a method for increasing the resolution of a simulation to make it more directable, but this also increases the cost. In liquids, authors have worked to track the surface of the liquid on a higher resolution grid than the underlying fluid simulation (see e.g. [48]). [122] uses the vortex particle method in order to increase the resolution of a simulation without the cost of running a simulation on a higher resolution grid. They demonstrated that the vortex particle method of [102] inherently contains additional computational detail which is lost when mapping it onto the underlying grid, and preserve more of it by mapping it on to the higher resolution grid.

In contrast to these methods which rely on a low resolution core simulation, we

propose a method that runs on a high resolution grid but creates a divergence-free velocity field using a coarser grid to speed up the calculations. This allows us to run simulations using higher resolutions while significantly reducing the computational cost, and possibly more importantly, run simulations that scale more linearly on large computational frameworks.

5.2 Performance Analysis

Simulation performance remains a large obstruction to using very large grids. To alleviate this issue we must develop techniques that are not only more efficient but can take advantage of the computational power available. This means that we need to be able to utilize not only all the computational power on one machine but also be able to run simulations across many machines without obtrusive communication costs. Our method improves both the performance on one core and the scalability across many cores.

5.2.1 Scaling

Traditional grid-based algorithms have a cost

$$C = C_l + C_p, \tag{5.1}$$

where C_p is the cost of the projection and C_l is the cost of the remainder of the algorithm. When we increase the resolution of the grid in three dimensions by a factor of k this becomes roughly

$$C_k \approx k^4 C_l + k^4 C_p. \tag{5.2}$$

This is because as we double the size of the grid we have eight times as many cells and each cell is half as wide, requiring twice as many time steps (when assuming a

CFL condition). Note that $k^4 C_p$ is only an approximation of the real cost of the work required to solve the Poisson equation since the matrix inversion does not generally scale linearly with the size of the matrix. One could use multi-grid methods, limit the number of conjugate gradient iterations, etc. in order to make this scale more linearly, but C_p will still scale no better than k^4.

Our goal is to reduce the cost of this system without losing the fine scale details of a higher resolution simulation. To achieve this, we break up the projection into a single projection on a coarser grid, which takes time C_{cp}, and a number of smaller projections that run fully decoupled on individual coarse grid cells, costing C_{fp}. This makes the cost for our algorithm

$$C = C_l + C_{cp} + C_{fp}. \tag{5.3}$$

If we refine our simulation grid, but not our coarse grid, C_{cp} scales only with the number of time steps, and the entire algorithm scales roughly as

$$C_k \approx k^4 C_l + k C_{cp} + k^4 C_{fp}. \tag{5.4}$$

Even if we assume that $C_{cp} + C_{fp} > C_p$, when we increase the resolution, for some k, $kC_{cp} + k^4 C_{fp} < k^4 C_p$. However, we found that $C_{cp} + C_{fp} > C_p$ only for very small resolutions (less than $16 \times 16 \times 16$) meaning that for reasonably sized examples or large examples our method runs significantly faster than traditional grid-based methods. We note that in our experience, about 90% of the time spent in a one-phase smoke simulation is spent during the projection meaning that C_p easily dominates C_l. From a complexity point of view this makes sense since the advection in C_l requires touching every grid point and its neighbors approximately once whereas a conjugate gradient algorithm would require touching every grid point and its neighbors once for every iteration of the conjugate gradient solve.

In spite of the analysis above, in general it seems that one of the k's can be removed from Equation (5.4) instead getting $C_k \approx k^3 C_l + C_{cp} + k^3 C_{fp}$ by keeping the same

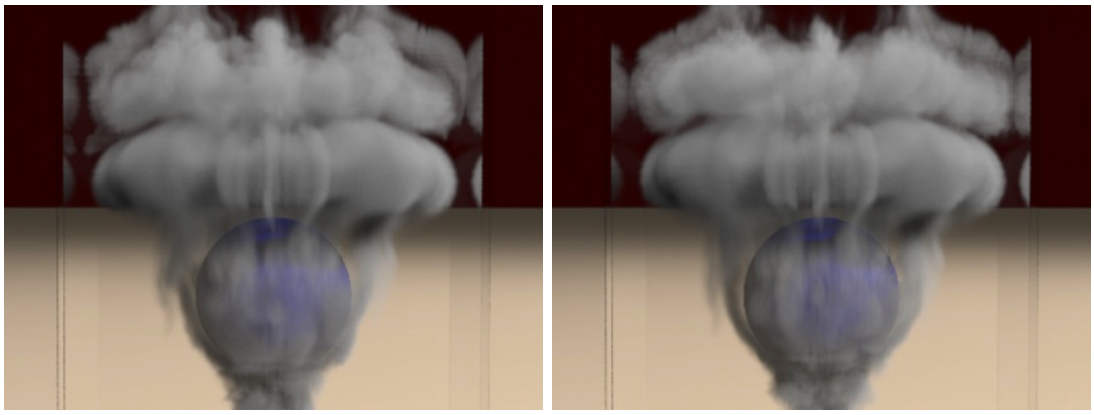


Figure 5.1: Smoke flowing around a moving sphere with on a $512 \times 1024 \times 512$ grid. (Left) uses a CFL number of 0.9 and takes time steps half as large as those generally taken on a $256 \times 512 \times 256$ grid. (Right) uses a CFL number of 2.2 making the time steps around 2.5 times as large.

timestep on the finer grid that was used on the coarser grid. Semi-Lagrangian advection (see [105]) makes this possible. Although we do not expect this to work for an indefinite number of refinements, for one or two refinements, we have achieved good results without halving the time step (see for example Figure 5.1).

5.2.2 Multi-core and Multi-processor Machines

As chip technology continues to advance, the use of multi-core and multi-processor machines and GPUs becomes more and more important, and algorithms such as [5] that can take advantage of this with little cost are able to run much more quickly. Simulating one-phase smoke typically consists of two primary steps, the explicit advection step and the implicit projection step. Looking at Equation (5.1), C_l is the advection step and C_p is the projection step. The advection step easily scales to multiple cores and multiple processors as one can simply break up the grid into several distinct parts and run advection on each part individually. The only issue then becomes how to deal with the boundaries. If the memory is shared, one can devise various strategies such as keeping separate copies of the information to alleviate issues with writing to data that needs to be read. There are also other strategies such as

red/black domain decomposition schemes, locking, etc. Using these strategies, one can usually expect to diminish the computational overhead of the overlapping boundaries. However, for large problems, where many computational nodes are desired, one would need a huge many core shared memory machine with a very large amount of memory – which is quite expensive and can lead to issues with cooling and energy. If algorithms that run on large computers become less tightly coupled, they require less synchronization and thus less shared memory, allowing cluster cores to be physically further apart (even on separate machines).

Thus, a more typical scenario is to switch to a non-shared memory model where memory might be shared by clusters of cores but is not shared between different clusters. Instead, each cluster has its own RAM. Although much more practical, this large non-shared memory infrastructure leads to a significant computational overhead as it requires data communication. That same overhead is also present when trying to utilize graphics cards. For the advection step C_l, boundary data only needs to be copied between clusters of cores once for each time step. Note that to do this a CFL condition is assumed to bound the amount of cells that need to be copied. Projection, on the other hand, requires at least one copy for each iteration of the solver. For a typical conjugate gradient solver this means that we have to copy or exchange data on the boundaries before every iteration of our solver. If we were to use a multi-grid method instead, we would still need many communications for each coarsening and refinement step in a V-cycle and several V-cycles are typically required for convergence.

Practical experience has shown that the computational bottleneck of exchanging boundary information leads to the projection scaling relativity poorly and dominating the cost of the simulation even more so than on a single core. Our algorithm on the other hand uses a far less expensive global solve, C_{cp}, on a coarse grid, which increases its computational demands much slower than usual as the simulation grid is refined (its coefficient is k, not k^4, in Equation (5.4)). An extremely important aspect of this algorithm is that the computational cost of increasing the resolution of the grid is mostly contained in C_l and C_{fp} as can be seen in Equation (5.4), and

both C_l and C_{fp} scale extremely well with the number of processors (almost perfectly linearly). The reason for this is that it is comprised of a large number of independent tasks. During the computations of C_{fp}, no communication of boundary information is required once the boundary conditions are initially set. The result is that on a large number of processors our algorithm is dominated by C_{cp}, which is significantly less expensive than C_p.

5.2.3 Multiphysics

For multiphysics code that is far more complex than a simple one-phase smoke simulation, such as the particle level set method [20], there are many additional costs of the algorithm to consider. For example, a typical PLS simulation contains steps for advecting particles and integrating them to and from the level set. On a single processor this means that C_l is significantly more expensive than in the simple smoke case, and as a result, the speed improvement obtained from converting C_p to $C_{fp} + C_{cp}$ is smaller. However, these additional steps often scale well in multi-processor or multi-core situations as it is simple to assign particles to a certain processor or core and let most of the computations for the particles happen independently on those computational nodes. Doing this only adds a few additional exchanges meaning that in this case C_p still dominates as one refines the grid. Thus, there is still a large benefit to use $C_{cp} + C_{fp}$ instead. In other words, when going from Equation (5.1) to Equation (5.2), the part of the cost, C_p, that scales badly is multiplied by k^4 whereas when going from Equation (5.3) to Equation (5.4) the part of the cost that scales badly, C_{cp}, is only multiplied by k, and as we stated above, it is not clear that a full k is required (see Figure 5.1).

5.3 Making a Divergence Free Flow

In order to use a coarser projection step we must determine how to make a coarser grid from our simulation grid, solve on coarser grid, and then map the results to our simulation grid in a way that maintains the divergence-free property. Our algorithm for one-phase smoke proceeds as in [25], except instead of solving the Poisson equation on the fully resolved fine grid we do the following:

1. Map the velocity field to a coarse uniform grid (5.3.2)
2. Make the resulting field divergence free on that coarse grid (5.3.3)
3. Map these velocities back to our fine simulation grid (5.3.4)
4. Make the resulting field divergence free on the fine grid (5.3.5)

5.3.1 Navier-Stokes Equations

On the fine simulation grid, we solve the inviscid, incompressible Navier-Stokes equations for the conservation of mass and momentum, given by

$$\vec{u}_t + \vec{u} \cdot \nabla \vec{u} = -\frac{1}{\rho}\nabla p + \vec{f} \tag{5.5}$$

$$\nabla \cdot \vec{u} = 0, \tag{5.6}$$

where $\vec{u}$ is the velocity field of the fluid, ρ is the density of the fluid, $\vec{f}$ are any external forces (such as gravity), and p is the fluid pressure. We solve these equations by first calculating an intermediate velocity field $\vec{u}^\star$ using

$$\frac{\vec{u}^\star - \vec{u}^n}{\Delta t} + \vec{u}^n \cdot \nabla \vec{u}^n = \vec{f} \tag{5.7}$$

and subsequently adding in the pressure forces via

$$\frac{\vec{u}^{n+1} - \vec{u}^{\star}}{\Delta t} = -\frac{1}{\rho}\nabla p, \tag{5.8}$$

where the pressure is calculated by solving a Poisson equation of the form

$$\nabla \cdot \frac{1}{\rho}\nabla \hat{p} = \nabla \cdot \vec{u}^{\star} \tag{5.9}$$

and $\hat{p} = p\Delta t$.

Equation (5.7) is solved on the fine simulation grid and represents C_l. The resulting velocities $\vec{u}^{\star}$ are then mapped to coarse grid (step 1). Equations (5.8) and (5.9) representing C_{cp} are then solved on the coarse grid (step 2). After determining $\vec{u}^{n+1}$ on the coarse grid, we then determine $\vec{u}^{n+1}$ on the fine grid, as represented by C_{fp} (steps 3 and 4).

5.3.2 Mapping to the Coarse Grid

Conceptually, our scheme was designed by taking a fine simulation grid and performing binary coarsening up to some level as is done with octrees. This results in both a fine and coarse grid that we could use in our algorithm. However, our framework is actually much more flexible and resembles that proposed in [68]. They started with a base level grid and place an octree in each cell so that the resolution could be increased locally. This was done to lower the computational cost of accessing octree nodes by removing the top of the tree which replaces some number of levels with uniform grid access. Their octrees could be any size including no refinement at all, see figures 5.2(a) and 5.2(b). The deepest levels of their octrees would represent the simulation grid in our framework whose degrees of freedom we would like to upgrade and their highest level represents the uniform coarse grid in which we will carry out the projection C_{cp}. However, our method for C_{fp} is general enough to not only support different levels of octree refinement as shown in figures 5.2(a) and 5.2(b) but to

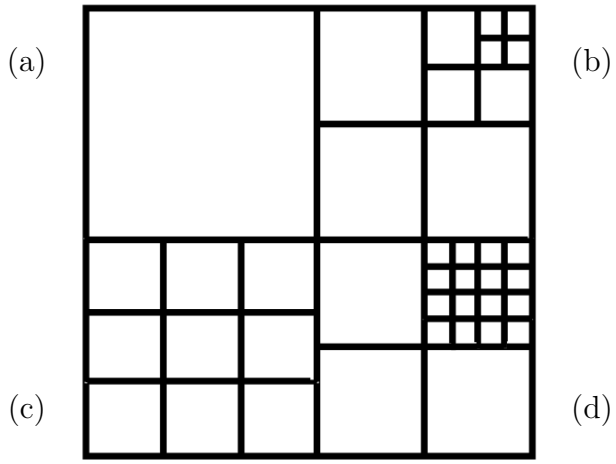


Figure 5.2: A refined grid with four coarse cells. (a) A coarse cell with no refinement. (b) A coarse cell with an octree inside (c) A coarse cell with a uniform grid inside. (d) A coarse cell with multiple levels of uniform grids inside.

also support uniform grids inserted into each coarse grid cell as shown in Figure 5.2(c) or even multiple levels of uniform grids as shown in Figure 5.2(d). This flexibility comes from our method for solving each coarse cell individually. However, we note that for most of our examples we chose to use uniform grids within each coarse cell as in Figure 5.2(c).

From the degree of freedom standpoint, every face on the finest resolution of all grids represents one degree of freedom for velocity whereas each face of the base coarse grid represents a degree of freedom for a much coarser set of velocities. We map the fine grid velocities to the coarser grid by using an area-weighted average of all the fine

grid velocities that are contained by a coarse grid face. In other words,

$$\vec{u}_c^{\star} = \frac{1}{n} \sum_{f \in faces} A_f \vec{u}_f^{\star} \tag{5.10}$$

where $faces$ is the set of fine scale faces that overlap with the coarse face c, A_f is the area of the fine face, and n is the number of elements in $faces$. One could liken this to methods that map particles to a background grid such as the PIC/FLIP method proposed in [127], the vortex particles proposed in [102], or the SPH method proposed in [71]. All these authors had proposed methods for mapping from the velocity degrees of freedom defined on particles to a background coarse grid, performing computation on the coarse grid, and then mapping information back. Contrary to these techniques in which the typical particle to grid mapping aims to have every particle influence the coarse grid, we do not map velocity degrees of freedom that are not incident on the coarse faces, meaning that they have no influence on the coarse grid. We admit that it is desirable to map more degrees of freedom; however, it turns out that our mapping allows for a more efficient handling of boundary conditions and is one of the key ideas that allows C_{fp} to scale linearly with the number of processors.

5.3.3 Coarse Grid Projection

Although we can use the standard Navier-Stokes equations for the coarse projection in the absence of objects, we must modify these equations when dealing with objects. This is because our objects are rasterized on the fine simulation grid, and a coarse grid face can thus contain a fraction of an object. Rasterizing the object on the fine grid allows us to maintain fine scale detail near the objects regardless of the size of the coarse grid. Following [69], we used the volume-weighted Poisson equation given by

$$V_{cell} \nabla \cdot \left(\vec{u}^{\star} - \frac{1}{\rho} \nabla \hat{p} \right) = 0. \tag{5.11}$$

We can then write

$$V_{cell}\nabla\cdot\vec{u}^{\star} = A_{face}\sum_{f\in faces}\left(\vec{u}_f^{\star}\cdot\vec{n}_f\right)\beta_f + \left(\vec{u}_s\cdot\vec{n}_f\right)\left(1-\beta_f\right) \tag{5.12}$$

and

$$V_{cell}\nabla\cdot\left(\frac{1}{\rho}\nabla\hat{p}\right) = A_{face}\sum_{f\in faces}\left(\left(\frac{1}{\rho}\nabla\hat{p}\right)_f\cdot\vec{n}_f\right)\beta_f \tag{5.13}$$

where $\vec{u}_s$ is the velocity of the solid and β_f is the fraction of the face that is not covered by a solid. Using Equation (5.11) with definitions (5.12) and (5.13) instead of Equation (5.9) on the coarse grid, allows for high fidelity modeling of stationary and moving solids as shown in figures 5.7 and 5.6. Note that in equations (5.12) and (5.13) we have factored A_{face} out in front of the sum as opposed to [69]. This is because on coarse grid all of our faces have the same size. As long as this is true, one can omit A_{face} from the equations and still retain symmetry. Otherwise it is more appropriate to include A_{face} inside the summation. We would also like to stress that using the volume-weighted Poisson equation is only necessary in the presence of objects or octree solves.

5.3.4 Mapping to the Fine Grid

After solving for a divergence free field on the coarse grid, we then need to map results of equations (5.8) and (5.9) back to the fine simulation grid. As mentioned earlier, many particle methods do this; however, whereas particle based methods require every particle degree of freedom to receive information from the coarse grid, we instead only map to the fine grid faces that are incident upon the coarse grid faces which we once again stress is a key to our algorithm. As is widely done in PIC/FLIP type methods, one can map either velocities or changes in velocities back to the fine degrees of freedom (see e.g. [127]). In other words, our new velocities for a given fine simulation grid face are:

$$\vec{u}_f^{n+1} = \alpha\vec{u}_c^{n+1} + (1-\alpha)(\vec{u}_f^{\star} + \vec{u}_c^{n+1} - \vec{u}_c^{\star}) \tag{5.14}$$

where α is a constant between 0 and 1, $\vec{u}_c$ are the velocities on our coarsened uniform grid and $\vec{u}_f$ are the velocities on the fine simulation grid. Note that in the case with α equal to 1, every fine grid face is set to have the same velocity as the coarse grid face containing it. In this case, the total flux of material into or out of any coarse grid cell is defined by the local fluxes of the fine grid and is still divergence free, although using α equal to 1 severely dampens the flow field. When α equals 0 the fine grid velocities $\vec{u}_f^\star$ simply acquire a change in velocity equal to the change that the coarse grid experienced. Because we created the coarse grid velocity $\vec{u}_c^\star$ with an area weighted average of the $\vec{u}_f^\star$ incident upon the face, adding this change to every fine grid cell incident on the face creates a net flux of 0 through the cell and thus a divergence free flow field as defined by the fine grid faces. This is much less dissipative because it uses a constant velocity difference to update the fine grid cells as opposed to a constant velocity. Note that when α is not 1 or 0, since the divergence operator is linear, one still obtains a divergence free field on each coarse cell as defined by the fine grid velocities. Moreover, one can blend a degree of dissipation into the numerical method as is typical of a PIC/FLIP scheme. In our examples, we found that lower values for an α create more detailed and dynamic results and thus use a value of 0. Of course if one desires the flow to be more damped a higher value of α could easily be substituted. However, in our examples, we found that additional damping was not needed. After the mapping, we must determine the fine grid velocities interior to each coarse grid cell to make a divergence free velocity field.

5.3.5 Fine Grid Local Projections

The method we used for mapping to the coarse grid, carrying out the coarse grid projection, and mapping back both gives continuity across each coarse grid cell as defined by the fine grid velocity degrees of freedom as well as a divergence-free coarse velocity field. In order to create a divergence-free field on the fine grid, we first consider each coarse grid cell to be its own unique computational domain. $\vec{u}^\star$ is given for every interior degree of freedom as was computed previously by Equation (5.7).

We then solve equations (5.8) and (5.9) for those interior degrees of freedom with fixed velocity boundary conditions for every fine grid velocity incident on the boundaries of this coarse grid cell.

If the coarse grid cell is not refined as in Figure 5.2(a), nothing needs to be done. If the coarse grid cell contains a uniform grid as in Figure 5.2(c), one simply solves equations (5.8) and (5.9) on that uniform grid. If the coarse grid cell contains an octree as in Figure 5.2(b), one can simply use the octree method of [69] in order to find a divergence-free set of velocities for the interior degrees of freedom. We also propose a new method that can be used based on hierarchical subdivision. Consider Figure 5.2(b), looking only at the first level of octree refinement. In this case, four new degrees of freedom are added, two vertical velocities and two horizontal velocities. For this example, Equation (5.8) has four unknowns at the center of each of the four computational cells. However, because there are Neumann boundary conditions, the system has a rank 1 degeneracy and one unknown pressure can be removed. Thus we can set up a 3×3 system of equations and solve with a fast direct method. For faces between cells that are not refined any further, that represents the final velocity degree of freedom. This is true for the bottom and left velocities in Figure 5.2(b). For cells that are refined further, e.g. the top and right velocities in that figure, one can use Equation (5.14) to map either the velocities or change in velocity to those faces. One would then consider each subcell such as the upper right hand corner of Figure 5.2(b) and repeat the process by subdividing once and solving the 3×3 system.

In two spatial dimensions, these 3×3 matrices are very quick to invert and this leads to a very quick hierarchical solver that can be efficiently implemented for example on a GPU. In three dimensions there are eight pressures and seven degrees of freedom, leading to a 7×7 matrix which can also be solved quickly. One might also imagine using the hierarchical subdivision only to do the first level and using the octree solver to solve the upper right hand corner of Figure 5.2(b). Similarly, consider Figure 5.2(d), one can use the first level of the hierarchical approach to get the first four velocities and then apply a uniform grid solver on the upper right hand corner. Note that when

solving these small matrices, one can use direct method, such as Cholesky factorization since these matrices are symmetric positive definite after the extra degree of freedom is eliminated. Because these systems are fairly quick to solve and because there are many independent systems to solve, one can imagine using threads to solve these systems simultaneously. This problem also lends itself well to specialized architectures such as the GPU that can very quickly execute many low cost processes. We note that although we implemented and tested all of the above methods for the fine local projections, for larger subgrids such as $8 \times 8 \times 8$ we primarily used one level of refinement with PCG as we found that this was fastest.

5.4 Discussion

In our examples section, we choose various examples of both smoke and water and ran them using a standard method on a reasonable base grid. We then analyzed two different strategies for using our method. One was to carry out grid refinements and achieve scaling as exemplified by Equation (5.4) where the base grid resolution was used for our projection in all finer grid simulations. This was done to illustrate that very high resolution simulations can be run using our method without suffering the curse of dimensionality on C_{cp}. As was proposed in [25], the vorticity confinement used in the base simulation was reduced linearly proportional to the size of the grid for the finer grid simulations. Even with less vorticity confinement we achieved significantly more computational detail. We also consider whether our base simulation could be accelerated using an even coarser grid for the projection step. As these grids became coarser and coarser, for example sixteen grid cells in a dimension, we did notice artifacts resulting from using our method. However, we stress that a fluid simulation on a very low resolution grid would be enormously simplified as shown in Figure 5.3. To alleviate these artifacts on the very coarse grid, we experimentally verified that feasible but not spectacular results could be obtained by using a Kolmolgorov spectrum similar to the one in [95] in combination with a $16 \times 32 \times 16$ grid but stress that it was only used in this one simulation to demonstrate that even at

resolutions we would not recommend, one could obtain a plausible result.

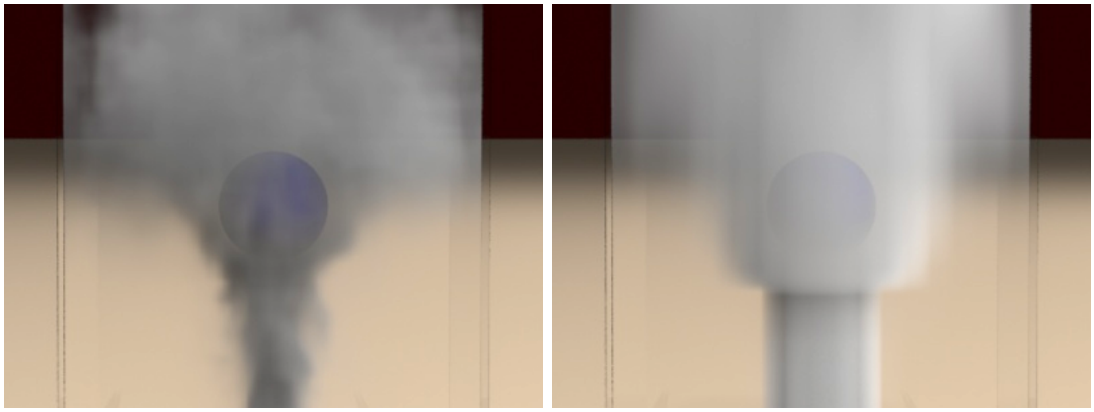


Figure 5.3: A comparison between (Left) a simulation using our method with a fine resolution of $64 \times 128 \times 64$ and a coarse resolution of $16 \times 32 \times 16$ and (Right) without using our method on a $16 \times 32 \times 16$ grid.

Figure 5.4 illustrates that not all algorithms that scale according to Equation (5.4) produce good results. Figure 5.4(c) is a baseline simulation on a 128×128 grid to give a sense one would expect to obtain with a standard method. Figure 5.4(b) shows our method on a 128×128 grid using a grid which was coarsened to 64×64 for the projection step. Note that figures 5.4(b) and (c) are quite similar in detail and structure. Figure 5.4(a) uses our algorithm except that the fine grid projections represented by C_{fp} are replaced with simple interpolation from the coarse grid. This method would also scale as Equation (5.4) but produces inferior results. Not only is detail lost, but more importantly, the structure of the smoke is lost due to added viscosity. This exemplifies the need for the fine grid projections represented by C_{fp} in order to obtain detail and structure that would be present on a finer grid. However, we note that there are other techniques such as [50] and [122] that do not need to run a large simulation. Instead they run a coarse simulation, upsample, and add additional details from noise or particles at the fine resolution which also allows them to also scale well.

We carried out this same comparison for the case of using a finer simulation resolution but using the same resolution for the projection as the base simulation shown in

(a) (b) (c) (d) (e)

Figure 5.4: A 2D smoke simulation run with a 128×128 base grid. (a) is a simulation on a 128×128 fine grid and a 64×64 coarse grid using interpolation. (b) is our method using a 128×128 grid and a 64×64 grid for projection. (c) is a base simulation on a 128×128 grid. (d) is a simulation on a 256×256 fine grid and a 128×128 coarse grid with interpolation. (e) is our method using a 256×256 grid with a 128×128 grid for projection.

Figure 5.4(c). Figure 5.4(e) shows the results obtained using our method. Note the highly increased structure and detail whereas Figure 5.4(d), which is obtained using interpolation, has no obvious added benefits over Figure 5.4(c). In summary, simply using our coarsening methodology with linear interpolation loses large amounts of detail in coarsening (Figure 5.4(c) becomes Figure 5.4(a)) and gains no detail when refining (Figure 5.4(c) becomes Figure 5.4(d)). In contrast, our method preserves detail when coarsening (Figure 5.4(c) becomes Figure 5.4(b)), and gains significantly more detail when refining (Figure 5.4(c) becomes Figure 5.4(e)). Although our results do achieve similar details to the fine grid base simulation, our results do have numerical differences. Figure 5.5 demonstrates the quantitative differences between our method and a standard projection on the fine grid.

While not our original intent, we also explored using our technique for liquids. As for smoke, refinement worked rather well. An added complication with water is in dealing with the free surface where constant pressure Dirichlet boundary conditions are imposed. Since the coarse grid does not contain the pressure degrees of freedom to represent the boundary conditions on the fine grid, one has to either over or under estimate mixed air/water regions. Failing to put constant pressure Dirichlet boundary

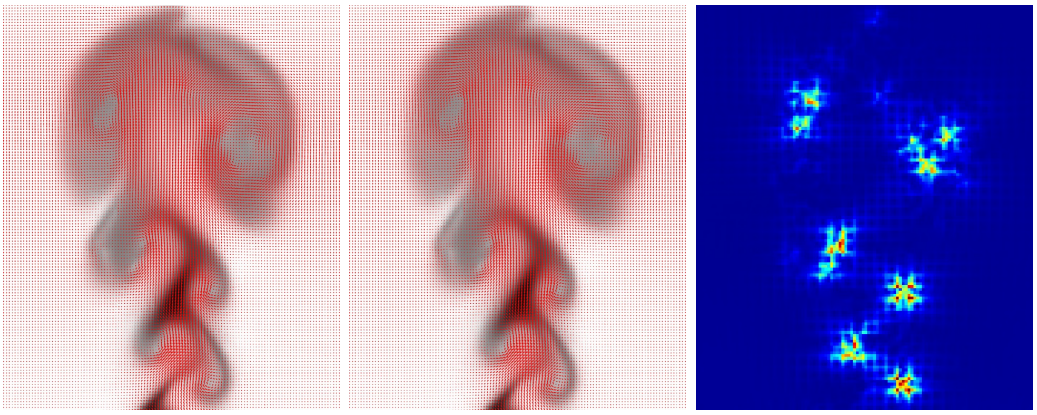


Figure 5.5: A quantitative comparison of our technique and a standard fluid simulation. (Left) Our technique with a 128×256 fine grid and a 32×64 coarse gird. (Center) The base simulation with a 128×256 grid. (Right) A comparison of the velocities between the two techniques. The warmer colors illustrate bigger differences. The maximum velocity error is about 1%.

conditions on the coarse grid cell means that that cell will be solved in a divergence-free manner on the coarse grid, which is preserved by our mapping to the fine grid. This means that any air pockets in that cell would not collapse properly. Moreover, wave type forces that are created by different height columns of water would be lost. If instead, the coarse cell constant pressure boundary condition is set, this treats the whole cell as if it was air and may allow too much or too little flow inside as this cell is not divergence free. This can result in underwater air pockets not only collapsing but losing mass. We tried both of these methods and observed both phenomena to some degree.

Because water waves are generated based on pressure differences one would expect that water would be more sensitive to the coarse grid approximation than smoke. However, we were pleasantly surprised to obtain reasonable simulation results as shown in Figure 5.8. In our experience, refining a reasonable base simulation added detail without noticeable issues until the surface started to become flat and at rest, at which point some minor artifacts could be seen. Under coarsening, however, these artifacts were harder to ignore. Therefore, we decided to use the coarse grid approximation to obtain the flow field only for cells interior to the water. We then

collected all fine scale cells near the free surface for a second Poisson solve that used the velocities from the fine grid projections as Neumann boundary conditions. We defined cells near the free surface to be all fine grid cells inside any coarse cell that contains a mixture of fluid and air. Performing this larger solve removed the artifacts albeit adding to the computation cost. However, since the cells near the surface only represent a lower dimensional set, essentially two spatial dimensions out of three, this additional cost disappears under grid refinement, although on the grids we used there was some overhead. We also note that this overhead tends to be fairly minimal because the surface solve is only needed during times with relatively calm fluid, making the solution quicker to obtain.

5.5 Examples

We demonstrate the effectiveness of our algorithm on a number of smoke and water simulations. All our smoke simulations contained a density source at the bottom of the domain, and our water simulations contained a water source at the top of the domain. Figure 5.4 shows two-dimensional smoke examples for illustration. We then demonstrate our algorithm for three dimensional examples of smoke and water. For each of these examples, we first ran a base simulation without using our method. We then used our method to coarsen the projection resolution, which improves the performance while maintaining similar looking results. We also used our method to refine the base simulation, showing that we can achieve very detailed results in a reasonable amount of time.

5.5.1 Smoke

We ran a number of three-dimensional smoke simulations as shown in figures 5.7 and 5.6. We would like to point out the stark differences in detail resulting from the different structures of the smoke that can be achieved by using our method to refine

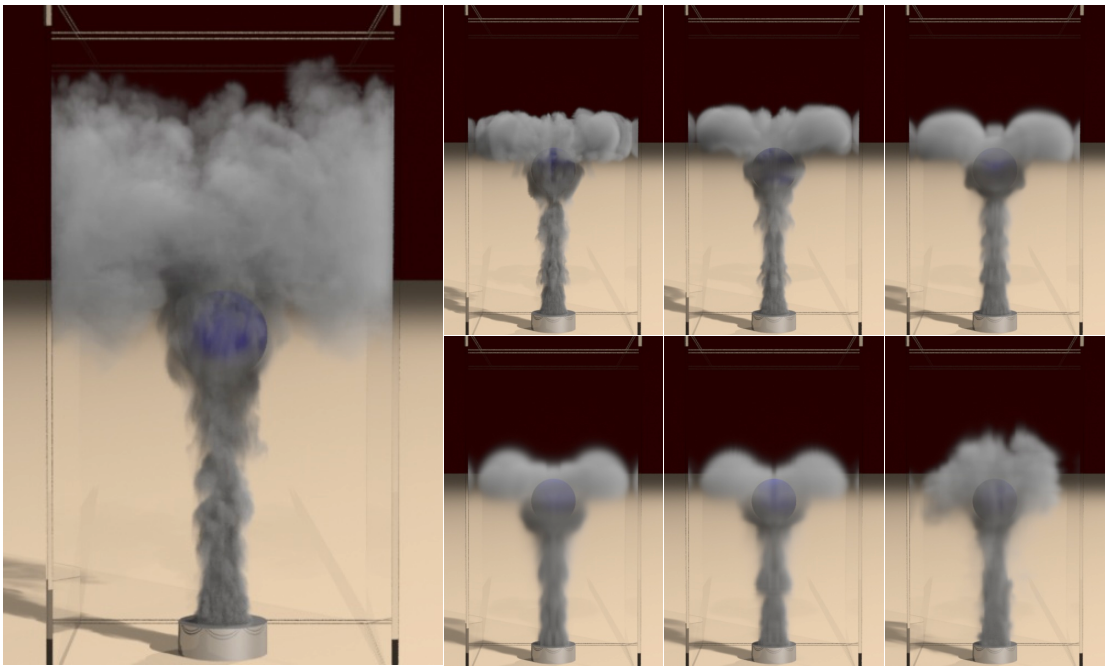


Figure 5.6: An example with smoke flowing around a static sphere. The large figure is a $512 \times 1024 \times 512$ simulation with a $64 \times 128 \times 64$ coarse grid. The smaller figures are comparisons of a simulation using different grid resolutions. The resolutions are starting from the left to right, top to bottom: a $512 \times 1024 \times 512$ simulation with a $64 \times 128 \times 64$ coarse grid, a $256 \times 512 \times 256$ simulation with a $64 \times 128 \times 64$ coarse grid, a $128 \times 256 \times 128$ simulation with a $64 \times 128 \times 64$ coarse grid, a $64 \times 128 \times 64$ base simulation, a $64 \times 128 \times 64$ simulation with a $32 \times 64 \times 32$ coarse grid, and a $64 \times 128 \times 64$ simulation with a $16 \times 32 \times 16$ coarse grid with Kolmolgorov noise.

the grid resolution. For example, when comparing the smoke examples of Figure 5.7, we achieve two distinct vortex rings in the high resolution example but only achieve one with the lower resolutions. Also note the fine scale vorticies around the sphere as shown in Figure 5.6.

5.5.2 Water

For water, we ran a number of simulations as shown in Figure 5.8. For these examples, we used the surface solve towards the end of the simulations when the fluid starts

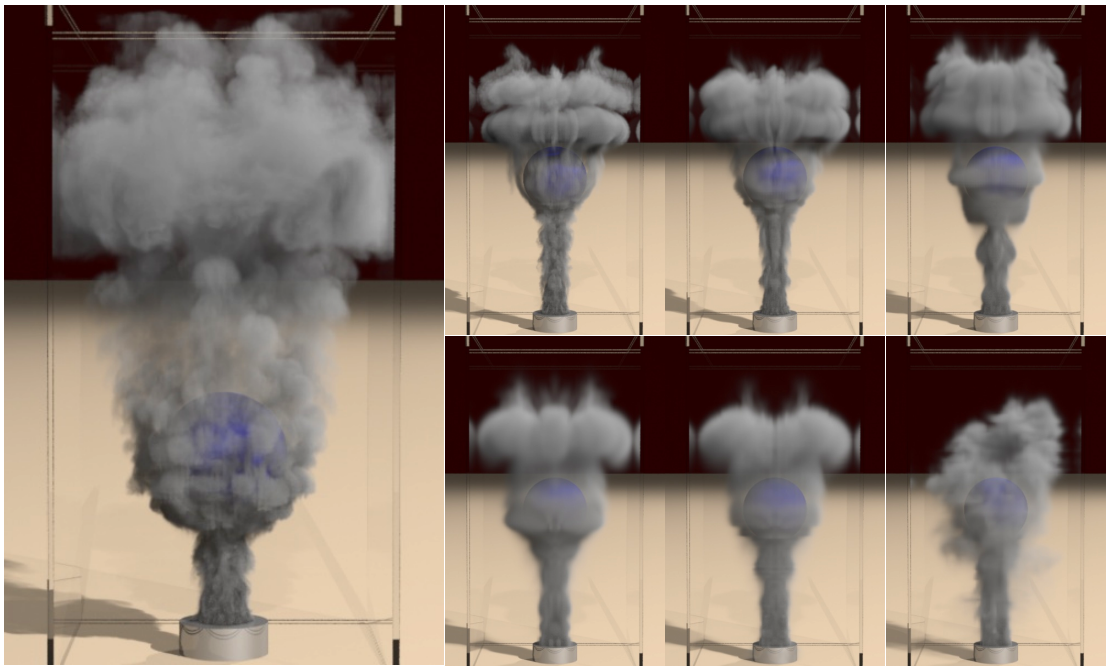


Figure 5.7: An example with smoke flowing around a moving sphere. The Large figure is a $512 \times 1024 \times 512$ simulation with a $64 \times 128 \times 64$ coarse grid. The smaller figures are comparisons of a simulation using different grid resolutions. The resolutions are starting from the left to right, top to bottom: a $512 \times 1024 \times 512$ simulation with a $64 \times 128 \times 64$ coarse grid, a $256 \times 512 \times 256$ simulation with a $64 \times 128 \times 64$ coarse grid, a $128 \times 256 \times 128$ simulation with a $64 \times 128 \times 64$ coarse grid, a $64 \times 128 \times 64$ base simulation, a $64 \times 128 \times 64$ simulation with a $32 \times 64 \times 32$ coarse grid, and a $64 \times 128 \times 64$ simulation with a $16 \times 32 \times 16$ coarse grid with Kolmolgorov noise.

to settle but did not when there is highly turbulent flow near the beginning of the simulation. As with smoke, we can achieve a large amount of additional detail by using our method to increase the resolution of the water.

5.5.3 Timing

The timings for our simulations are shown in Table 5.1. We compare our results with those obtained by running a base simulation on the coarse grid, and a base simulation on the refined grid. Note, for example, that our method runs approximately 67

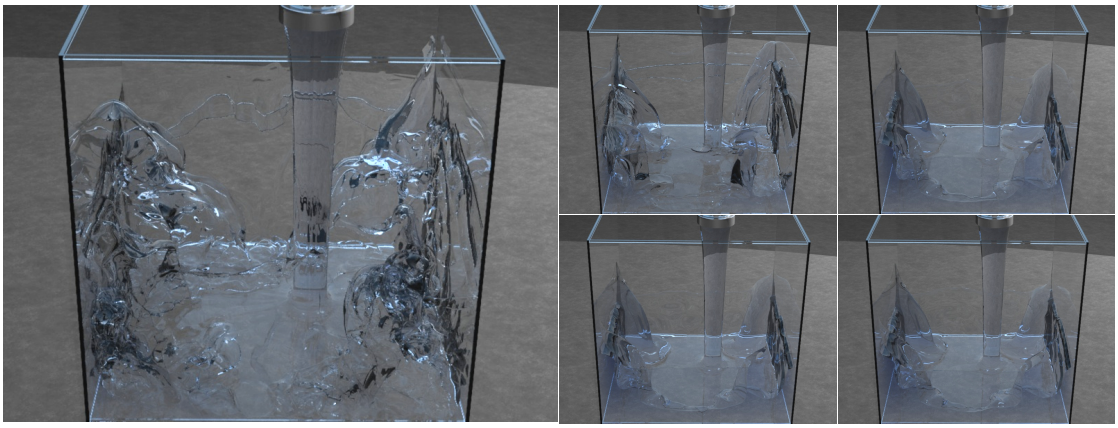


Figure 5.8: An example with water pouring into a box. This is a comparison of a simulation using different grid resolutions. The resolutions are starting from the left to right, top to bottom: a $512 \times 512 \times 512$ simulation with a $128 \times 128 \times 128$ coarse grid, a $256 \times 256 \times 256$ simulation with a $128 \times 128 \times 128$ coarse grid, a $128 \times 128 \times 128$ base simulation, a $128 \times 128 \times 128$ simulation with a $64 \times 64 \times 64$ coarse grid, and a $128 \times 128 \times 128$ simulation with a $32 \times 32 \times 32$ coarse grid.

times faster on the high-resolution static sphere example, shown on line four of the table above. This makes previously infeasible simulations tractable. An important detail to note is that a base high-resolution simulation quickly runs up against hard memory limits on a machine. This is partially alleviated with our method, as there is significantly less memory access during the projection step.

When comparing our method to base simulations on the coarse grid, our method runs slower when using a single processor. This is because a large amount of time is spent in C_{fp}. However, our method scales well with a large number of processors, and we achieve similar timings to the coarse grid base simulations for each time step. This is because the cost of C_{fp} scales linearly and as a result, both algorithms are dominated by the global projection (C_{cp} or C_p). We note that in some cases our method actually runs more quickly per time step than the base simulation because we take smaller time steps for a larger resolution grid.

Example	**Fine**	**Coarse**	**Fine 1 proc**	**Coarse 1 proc**	**Ours 1 proc**
Static Sphere	$64 \times 128 \times 64$	$16 \times 32 \times 16$	$5.6m/34s$	$1.2s/0.4s$	$20s/2s$
Static Sphere	$64 \times 128 \times 64$	$32 \times 64 \times 32$	$5.6m/34s$	$4.5s/0.9s$	$20s/2s$
Static Sphere	$128 \times 256 \times 128$	$64 \times 128 \times 64$	$5.8h/7m$	$5.6m/34s$	$8.3m/25s$
Static Sphere	$256 \times 512 \times 256$	$64 \times 128 \times 64$	$50h/1h$	$5.6m/34s$	$1.6h/2m$
Static Sphere	$512 \times 1024 \times 512$	$64 \times 128 \times 64$	-	$5.6m/34s$	-
Moving Sphere	$64 \times 128 \times 64$	$16 \times 32 \times 16$	$8m/32s$	$0.5s/0.1s$	$20s/2s$
Moving Sphere	$64 \times 128 \times 64$	$32 \times 64 \times 32$	$8m/32s$	$3s/0.6s$	$20s/2s$
Moving Sphere	$128 \times 256 \times 128$	$64 \times 128 \times 64$	$9h/7m$	$8m/32s$	$16m/31s$
Moving Sphere	$256 \times 512 \times 256$	$64 \times 128 \times 64$	$75h/1h$	$8m/32s$	$2.5h/2m$
Moving Sphere	$512 \times 1024 \times 512$	$64 \times 128 \times 64$	-	$8m/32s$	-
Water	$128 \times 128 \times 128$	$32 \times 32 \times 32$	$2m/3s$	$25s/2.5s$	$1m/1.5s$
Water	$128 \times 128 \times 128$	$64 \times 64 \times 64$	$2m/3s$	$1m/3s$	$1m/1.5s$
Water	$256 \times 256 \times 256$	$128 \times 128 \times 128$	$43m/17s$	$2m/3s$	$11m/4.5s$
Water	$512 \times 512 \times 512$	$128 \times 128 \times 128$	$16h/3.2m$	$2m/3s$	$65m/13s$
Example	**Fine**	**Coarse**	**Fine 64 procs**	**Coarse 64 procs**	**Ours 64 procs**
Static Sphere	$64 \times 128 \times 64$	$16 \times 32 \times 16$	$2.2m/13s$	$12s/4s$	$30s/3s$
Static Sphere	$64 \times 128 \times 64$	$32 \times 64 \times 32$	$2.2m/13s$	$30s/6s$	$1m/6s$
Static Sphere	$128 \times 256 \times 128$	$64 \times 128 \times 64$	$16m/19s$	$2.2m/13s$	$2.7m/8s$
Static Sphere	$256 \times 512 \times 256$	$64 \times 128 \times 64$	$2h/2.5m$	$2.2m/13s$	$9m/11s$
Static Sphere	$512 \times 1024 \times 512$	$64 \times 128 \times 64$	$47h/28m$	$2.2m/13s$	$43m/26s$
Moving Sphere	$64 \times 128 \times 64$	$16 \times 32 \times 16$	$2m/12s$	$15s/5s$	$40s/4s$
Moving Sphere	$64 \times 128 \times 64$	$32 \times 64 \times 32$	$2m/12s$	$45s/9s$	$1.1m/7s$
Moving Sphere	$128 \times 256 \times 128$	$64 \times 128 \times 64$	$10m/20s$	$2m/12s$	$6m/12s$
Moving Sphere	$256 \times 512 \times 256$	$64 \times 128 \times 64$	$3h/2.5m$	$2m/12s$	$17m/15s$
Moving Sphere	$512 \times 1024 \times 512$	$64 \times 128 \times 64$	$70h/28m$	$2m/12s$	$70m/28s$
Water	$128 \times 128 \times 128$	$32 \times 32 \times 32$	$2m/3s$	$25s/2.5s$	$1m/1.5s$
Water	$128 \times 128 \times 128$	$64 \times 64 \times 64$	$2m/3s$	$1m/3s$	$1m/1.5s$
Water	$256 \times 256 \times 256$	$128 \times 128 \times 128$	$43m/17s$	$2m/3s$	$11m/4.5s$
Water	$512 \times 512 \times 512$	$128 \times 128 \times 128$	$16h/3.2m$	$2m/3s$	$65m/13s$

Table 5.1: Timing information for our examples as well as base simulations on the fine and coarse grid using both 1 processor and 64 processors. Some large resolution simulations (noted by -) could not be run on a single processor due to RAM restrictions. All of our timings are given in time per frame/time per time step. Note that all examples were run at 24 frames per second and with a CFL number of 0.9.

5.6 Conclusions and Future Work

We have introduced a novel algorithm that improves the performance of existing fluid simulations and can achieve realistic results using large fluid grids. Our algorithm effectively reduces the amount of time required for the Poisson solve by using a coarse

grid projection and then small projections within each coarse grid cell.

Our performance analysis demonstrates that in Equation (5.1) computation schemes can be divided into pieces where one piece scales well with the number of processors, C_l, and the other piece scales poorly with the number of processors, C_p. A typical Newton-Cotes quadrature formula suffers from the curse of dimensionality such that refining the width by a factor of two requires that refinement in every dimension. Thus in Equation (5.2), with three dimensions in space and one dimension in time, the cost of each of these increases by k^4. In section 2, we illustrate a strategy one could use in designing general algorithms. In general, this can be done by writing the code such that more of C_p scales better. The result is essentially putting more computation into C_l. We propose a method that when solving the equations using a Newton-Cotes quadrature style method, instead of refining every term as usual getting k^4 in front of all terms in Equation (5.2), we refine the terms that scale well in Equation (5.4) in the usual manner with k^4 but propose a different method for solving the terms that do not scale well so that k can appear to a lower power. In our specific case, this means dividing C_p into C_{cp} and C_{fp} where C_{cp} scales poorly and C_{fp} scales well. For the remaining terms (such as C_{cp}) that do not scale well, one might imagine using a Monte-Carlo style integration scheme which does not suffer the curse of dimensionality. We would like to stress that simply having this framework does not alone lead to improvements. Cheaper methods for solving C_{cp} do not necessarily lead to good results. For example, linear averaging which fits into the model of Equation (5.4) does scale rather well just like our proposed method but unfortunately gives lack luster results.

Although our method was used to solve the Poisson equation for incompressible flow, Equation (5.8) updates $u^\star$ to u^{n+1} by subtracting $\frac{1}{\rho}\nabla\hat{p}$. This means that at every face of our grid we define $\nabla\hat{p}$, the derivatives of p, on the entire grid. We can think of this as solving a general Poisson equation $\nabla\cdot\left(\frac{1}{\rho}\nabla p\right) = f$ for some f. Our method provides a technique for quickly and efficiently finding an approximation for ∇p on the fine grid. Because many other applications make use of Poisson equations and/or its derivatives, it would be interesting to explore the use of our methodology in those

areas. However, one should be cautious of the fact that we are not doing a formal Hodge decomposition, meaning that the divergence-free vector field that we get is not the unique field that decomposes $u^\star$ into a divergence-free field plus the gradient of a scalar field. That being said, we have found that this approximation works very well for our applications and is likely to do so for others.

One interesting avenue of future work would be to integrate our method with a multigrid solver. Standard multigrid solvers often need special treatment for problems involving free surfaces, object boundaries, special exterior domain boundary conditions, etc. Otherwise, their convergence rate can be a bit slow. The difficulty lies in that multigrid solutions do not obtain a divergence free flow field until they are fully converged (this is also true for cg and all traditional methods). Our method provides an interesting twist in that it can be used as a prolongation operator for multigrid and one might use a multigrid solver, feel that too much time is being spent on convergence, and truncate the process earlier using our technique as a final prolongation producing a divergence-free flow field. In that sense, one can compare to techniques such as [36] where an iterative method is used to solve for contact, and shock-propagation is used at the end in order to clean up the interpenetrations of the geometry providing a visually reasonable result.

Chapter 6

Conclusions

In this dissertation, several algorithms for improving the performance and scalability of fluid simulations have been presented. These algorithms improved both the advection and projection steps within incompressible flow. In Chapter 2 we introduced an unconditionally stable fully conservative advection method based on the traditional semi-Lagrangian method. We then demonstrated how this method can be augmented to conserve mass and momentum in Chapters 3 and 4. In Chapter 5 we introduced an easily parallelizable coarse grid projection method which can improve the performance of the projection step by using two different resolution grids. Combining these methods allows for the use of large time steps and thus can achieve significantly higher resolution simulations using similar amounts of computational power as prior methods.

While these methods allow for the simulation of higher resolutions to achieve high fidelity details, these methods can also be used to approach real time simulation. In this thesis, we have shown multiple orders of magnitudes in performance improvement at high resolutions. However, this benefit is not as large at the resolutions that can be used in real time. As an example, in Chapter 3 we demonstrated the large benefit our method has when running at or near framerate using CFL numbers near 40. However, these examples used high resolution grids which would not be possible in

real time environments. At the resolutions that can be simulated in real time, the CFL numbers that would be needed to achieve one time step per frame are closer to 5. Additionally, the benefits of our projection method described in Chapter 5 depend on the difference in scales between the coarse and fine resolutions that are used. Using the resolutions that can be achieved in real time limits this difference to approximately a factor of two in order to achieve high quality results which reduces the performance benefit obtained. As the amount of computational power increases, the resolutions that can be achieved in real time will also increase allowing these methods to offer larger performance improvements in the real time setting. However, with the current state of hardware, other algorithms must be investigated to increase the performance improvement when running relatively low resolutions that are similar to the ones that can currently be achieved in real time.

In order to address low resolutions, the methods presented can be combined with other performance improving methods such as adaptive grids [69, 43, 10]. It might also be possible to combine this work with more efficient fluid algorithms such as those used in model reduction [114]. However, the loss of visual fidelity caused by these types of methods may be too great for use in real time environments. It would also be interesting to apply this work to additional phenomena such as fire, multiphase flow, and solid fluid coupling in order to achieve improved performance and scalability.

In addition, as we approach real time environments the problem of interactivity becomes important. We are currently investigating novel ways to interact with simulations including dragging objects which can interact with fluids, and using other input devices such as the Microsoft Kinect in order to control fluids through gesture recognition. These algorithms not only require real time simulation but also interaction with users meaning the computational power available to the simulator is less than it would be if simulation was the only task. Investigating other methods of interaction would make interesting future work.

Bibliography

[1] C. Batty, F. Bertails, and R. Bridson. A fast variational framework for accurate solid-fluid coupling. ACM Trans. Graph. (SIGGRAPH Proc.), 26(3):100, 2007.

[2] Christopher Batty, Stefan Xenos, and Ben Houston. Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. In Proceedings of Eurographics, 2010.

[3] M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. J. Comput. Phys., 53:484–512, 1984.

[4] Haimasree Bhatacharya, Yue Gao, and Adam Bargteil. A level-set method for skinning animated particle data. In Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '11, 2011.

[5] J. Bolz, I. Farmer, E. Grinspun, and P. Schroder. Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. ACM Trans. Graph. (SIGGRAPH Proc.), 22(3):917–924, 2003.

[6] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. ACM Trans. Graph., 26(3):46, 2007.

[7] T. Brochu, C. Batty, and R. Bridson. Matching fluid simulation elements to surface geometry and topology. ACM Trans. Graph., 2010.

[8] M. Carlson, P. J. Mucha, and G. Turk. Rigid fluid: Animating the interplay between rigid bodies and fluid. ACM Trans. Graph. (SIGGRAPH Proc.), 23:377–384, 2004.

[9] N. Chentanez, T. G. Goktekin, B. Feldman, and J. O'Brien. Simultaneous coupling of fluids and deformable bodies. In ACM SIGGRAPH/Eurographics Symp. on Comput. Anim., pages 325–333, 2006.

[10] N Chentanez and M. Müller. Real-time Eulerian water simulation using a restricted tall cell grid. In Proc. of ACM SIGGRAPH 2011, pages 82:1–82:10, 2011.

[11] Nuttapong Chentanez, Bryan E. Feldman, François Labelle, James F. O'Brien, and Jonathan R. Shewchuk. Liquid simulation on lattice-based tetrahedral meshes. In ACM SIGGRAPH/Eurographics Symp. on Comput. Anim., pages 219–228, 2007.

[12] D.L. Chopp. Another look at velocity extensions in the level set method. SIAM J. Sci. Comput., 31:3255–3273, 2009.

[13] R. Codina, G. Houzeaux, H. Coppola-Owen, and J. Baiges. The fixed-mesh ALE approach for the numerical approximation of flows in moving domains. J. of Comp. Phys., 228(5):1591–1611, 2009.

[14] F. Colina, R. Egli, and F. Lin. Computing a null divergence velocity field using smoothed particle hydrodynamics. J. Comput. Phys., 217:680–692, 2006.

[15] R. Courant, E. Issacson, and M. Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. Comm. Pure and Applied Math, 5:243–255, 1952.

[16] S. Cummins and M. Rudman. An SPH projection method. J. Comput. Phys., 152(2):584–607, 1999.

[17] M. Desbrun and M.-P. Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. In R. Boulic and G. Hegron, editors, Comput. Anim. and Sim. '96 (Proc. of EG Wrkshp. on Anim. and Sim.), pages 61–76. Springer-Verlag, Aug 1996.

[18] T. Dupont and Y. Liu. Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function. J. Comput. Phys., 190/1:311–324, 2003.

[19] T. Dupont and Y. Liu. Back and forth error compensation and correction methods for semi-Lagrangian schemes with application to level set interface computations. Math. Comp., 76(258):647–668, 2007.

[20] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. J. Comput. Phys., 183:83–116, 2002.

[21] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-Lagrangian particle level set method. Computers and Structures, 83:479–490, 2005.

[22] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. ACM Trans. Graph. (SIGGRAPH Proc.), 21(3):736–744, 2002.

[23] R. Fattal and D. Lischinski. Target-driven smoke animation. ACM Trans. Graph. (SIGGRAPH Proc.), 23:441–448, 2004.

[24] R. Fedkiw, X.-D. Liu, and S. Osher. A general technique for eliminating spurious oscillations in conservative schemes for multiphase and multispecies euler equations. Int. J. Nonlinear Sci. and Numer. Sim., 3:99–106, 2002.

[25] R. Fedkiw, J. Stam, and H. Jensen. Visual simulation of smoke. In Proc. of ACM SIGGRAPH 2001, pages 15–22, 2001.

[26] B. Feldman, J. O'Brien, and B. Klingner. Animating gases with hybrid meshes. ACM Trans. Graph. (SIGGRAPH Proc.), 24(3):904–909, 2005.

[27] B. Feldman, J. O'Brien, B. Klingner, and T. Goktekin. Fluids in deforming meshes. In Proc. of the ACM SIGGRAPH/Eurographics Symp. on Comput. Anim., 2005.

[28] N. Foster and R. Fedkiw. Practical animation of liquids. In Proc. of ACM SIGGRAPH 2001, pages 23–30, 2001.

[29] N. Foster and D. Metaxas. Controlling fluid animation. In Comput. Graph. Int., pages 178–188, 1997.

[30] Peter Frolkovic and Karol Mikula. High-resolution flux-based level set method. SIAM J. Sci. Comput., 29:579–597, 2007.

[31] Yue Gao, Chen-Feng Li, Shi-Min Hu, and Brian A. Barsky. Simulating gaseous fluids with low and high speeds. Comput. Graph. Forum, 28(7):1845–1852, 2009.

[32] F. Gibou, R. Fedkiw, L.-T. Cheng, and M. Kang. A second-order-accurate symmetric discretization of the Poisson equation on irregular domains. J. Comput. Phys., 176:205–227, 2002.

[33] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics-theory and application to nonspherical stars. Mon. Not. R. Astron. Soc., 181:375, 1977.

[34] N. Grenier, M. Antuono, A. Colagrossi, D. Le Touzé, and B. Alessandrini. An Hamiltonian interface SPH formulation for multi-fluid and free surface flows. J. of Comput. Phys., 228(22):8380–8393, 2009.

[35] J.T. Grétarsson and R. Fedkiw. Two-way coupling of compressible flows and thin deforming shells. (In Preparation), 2010.

[36] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. ACM Trans. Graph. (SIGGRAPH Proc.), 22(3):871–878, 2003.

[37] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw. Coupling water and smoke to thin deformable and rigid shells. ACM Trans. Graph. (SIGGRAPH Proc.), 24(3):973–981, 2005.

[38] F. Harlow and J. Welch. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. Phys. Fluids, 8:2182–2189, 1965.

[39] Dalton J. E. Harvie and David F. Fletcher. A new volume of fluid advection algorithm: the stream scheme. J. Comput. Phys., 162(1):1–32, 2000.

[40] C. Hirt, A. Amsden, and J. Cook. An arbitrary Lagrangian-Eulerian computing method for all flow speeds. J. Comput. Phys., 135:227–253, 1974.

[41] J.-M. Hong and C.-H. Kim. Discontinuous fluids. ACM Trans. Graph. (SIGGRAPH Proc.), 24(3):915–920, 2005.

[42] Christopher Horvath and Willi Geiger. Directable, high-resolution simulation of fire on the gpu. ACM Trans. Graph., 28(3):1–8, 2009.

[43] G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. ACM Trans. Graph. (SIGGRAPH Proc.), 25(3):805–811, 2006.

[44] Taekwon Jang, Heeyoung Kim, Jinhyuk Bae, Jaewoo Seo, and Junyong Noh. Multilevel vorticity confinement for water turbulence simulation. Vis. Comput., 26:873–881, 2010.

[45] B. Kim, Y. Liu, I. Llamas, X. Jiao, and J. Rossignac. Simulation of bubbles in foam with the volume control method. In Proc. of ACM SIGGRAPH 2007, pages 98:1–98:10, 2007.

[46] B.-M. Kim, Y. Liu, I. Llamas, and J. Rossignac. Using BFECC for fluid simulation. In Eurographics Workshop on Natural Phenomena 2005, 2005.

[47] B.-M. Kim, Y. Liu, I. Llamas, and J. Rossignac. Advections with significantly reduced dissipation and diffusion. IEEE Trans. on Vis. and Comput. Graph., 13(1):135–144, 2007.

[48] Doyub Kim, Oh-young Song, and Hyeong-Seok Ko. Stretching and wiggling liquids. In SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers, pages 1–7, New York, NY, USA, 2009. ACM.

[49] Theodore Kim and Doug L. James. Skipping steps in deformable simulation with online model reduction. In SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers, pages 1–9, 2009.

[50] Theodore Kim, Nils Thürey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. In SIGGRAPH '08: ACM SIGGRAPH 2008 papers, pages 1–6, 2008.

[51] Bryan M. Klingner, Bryan E. Feldman, Nuttapong Chentanez, and James F. O'Brien. Fluid animation with dynamic meshes. ACM Trans. Graph., 25(3):820–825, 2006.

[52] S. Koshizuka, A. Nobe, and Y. Oka. Numerical analysis of breaking waves using the moving particle semi-implicit method. Int. J. Num. Meth. in Fluids, 26:751–769, 1998.

[53] S. Koshizuka, H. Tamako, and Y. Oka. A particle method for incompressible viscous flows with fluid fragmentation. Comput. Fluid Dyn. J, 1995.

[54] N. Kwatra, J. Su, J.T. Grétarsson, and R. Fedkiw. A method for avoiding the acoustic time step restriction in compressible flow. J. Comput. Phys., 228(11):4146–4161, 2009.

[55] Nipun Kwatra, Jón T. Grétarsson, and Ronald Fedkiw. Practical animation of compressible flow for shock waves and related phenomena. In ACM SIGGRAPH/Eurographics Symp. on Comput. Anim., pages 207–215, 2010.

[56] A. Lamorlette and N. Foster. Structural modeling of flames for a production environment. ACM Trans. Graph. (SIGGRAPH Proc.), 21(3):729–735, 2002.

[57] M. Lentine and R. Fedkiw. Treating kinetic energy in incompressible flows. (In Preparation), 2010.

[58] M. Lentine, J.T. Grétarsson, and R. Fedkiw. An unconditionally stable fully conservative semi-lagrangian method. J. Comput. Phys., 230:2857–2879, 2011.

[59] M. Lentine, W. Zheng, and R. Fedkiw. A novel algorithm for incompressible flow using only a coarse grid projection. ACM Transactions on Graphics, July 2010.

[60] Michael Lentine, Mridul Aanjaneya, and Ronald Fedkiw. Mass and momentum conservation for fluid simulation. In SCA '11: Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 91–100, 2011.

[61] BP Leonard, AP Lock, and MK MacVean. Conservative explicit unrestricted-time-step multidimensional constancy-preserving advection schemes. Monthly Weather Review, 124:2588–2606, 1996.

[62] L.M. Leslie and R.J. Purser. Three-dimensional mass-conserving semi-lagrangian scheme employing forward trajectories. Monthly Weather Review, 123(8), 1995.

[63] R.J. Leveque. A large time step generalization of godunov's method for systems of conservation laws. SIAM J. Num. Analysis, 22(6):1051–1073, 1985.

[64] S.J. Lin and R.B. Rood. Multidimensional flux-form semi-lagrangian transport schemes. Monthly Weather Review, 24(9):2046–2070, 1996.

[65] P. Liovic, M. Rudman, J.L. Liow, D. Lakehal, and D. Kothe. A 3D unsplit-advection volume tracking algorithm with planarity-preserving interface reconstruction. Computers & Fluids, 35(10):1011–1032, 2006.

[66] J. Liu, S. Koshizuka, and Y. Oka. A hybrid particle-mesh method for viscous, incompressible, multiphase flows. J. Comput. Phys., 202(1):65–93, 2005.

[67] J. López, J. Hernández, P. Gómez, and F. Faura. A volume of fluid method based on multidimensional advection and spline interface reconstruction. J. of Comp. Phys., 195(2):718–742, 2004.

[68] F. Losasso, R. Fedkiw, and S. Osher. Spatially adaptive techniques for level set methods and incompressible flow. Computers and Fluids, 35:995–1010, 2006.

[69] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. ACM Trans. Graph. (SIGGRAPH Proc.), 23:457–462, 2004.

[70] F. Losasso, T. Shinar, A. Selle, and R. Fedkiw. Multiple interacting liquids. ACM Trans. Graph. (SIGGRAPH Proc.), 25(3):812–819, 2006.

[71] F. Losasso, J. Talton, N. Kwatra, and R. Fedkiw. Two-way coupled sph and particle level set fluid simulation. IEEE TVCG, 14(4):797–804, 2008.

[72] R. Loubčre and M.J. Shashkov. A subcell remapping method on staggered polygonal grids for arbitrary-Lagrangian-Eulerian methods. J. of Comp. Phys., 209(1):105–138, 2005.

[73] R. Loubčre, M. Staley, and B. Wendroff. The repair paradigm: new algorithms and applications to compressible flow. J. of Comp. Phys., 211(2):385–404, 2006.

[74] L. Lucy. A numerical approach to the testing of the fission hypothesis. Astronomical J., 82:1013–1024, 1977.

[75] R. MacCormack. The effect of viscosity in hypervelocity impact cratering. In AIAA Hypervelocity Impact Conference, 1969. AIAA paper 69-354.

[76] LG Margolin and M. Shashkov. Remapping, recovery and repair on a staggered grid. Computer Methods in Applied Mechanics and Engineering, 193(39-41):4139–4155, 2004.

[77] A. McAdams, E. Sifakis, and J. Teran. A parallel multigrid poisson solver for fluids simulation on large grids. In SCA/Eurographics Symp. on Comput. Anim., pages 1–10, 2010.

[78] V. Mihalef, D. Metaxas, and M. Sussman. Animation and control of breaking waves. In Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim., pages 315–324, 2004.

[79] V. Mihalef, D. N. Metaxas, and Mark Sussman. Simulation of two-phase flow with sub-scale droplet and bubble effects. Comput. Graph. Forum, 2009.

[80] V. Mihalef, B. Unlusu, D. Metaxas, M. Sussman, and M. Hussaini. Physics based boiling simulation. In SCA '06: Proc. of the 2006 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim., pages 317–324, 2006.

[81] C. Min and F. Gibou. A second order accurate projection method for the incompressible Navier-Stokes equation on non-graded adaptive grids. J. Comput. Phys., 219:912–929, 2006.

[82] J. Molemaker, J.M. Cohen, S. Patel, and J. Noh. Low viscosity flow simulations for animation. In SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pages 9–18. Eurographics Association, 2008.

[83] Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. Energy-preserving integrators for fluid animation. In SIGGRAPH '09: ACM SIGGRAPH 2009 papers, pages 1–8, 2009.

[84] Patrick Mullen, Alexander McKenzie, Yiying Tong, and Mathieu Desbrun. A variational approach to eulerian geometry processing. ACM Trans. Graph., pages –1–1, 2007.

[85] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim., pages 154–159, 2003.

[86] T. Nakamura, R. Tanaka, T. Yabe, and K. Takizawa. Exactly conservative semi-Lagrangian scheme for multi-dimensional hyperbolic equations with directional splitting technique. J. of Comp. Phys., 174(1):171–207, 2001.

[87] Rahul Narain, Jason Sewall, Mark Carlson, and Ming C. Lin. Fast animation of turbulence using energy transport and procedural synthesis. In SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers, pages 1–8, New York, NY, USA, 2008. ACM.

[88] Michael B. Nielsen, Brian B. Christensen, Nafees Bin Zafar, Doug Roble, and Ken Museth. Guiding of smoke animations through variational coupling of simulations at different resolutions. In SCA '09: Proc. of the 2009 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim., pages 217–226, 2009.

[89] T. Pfaff, N. Thuerey, J. Cohen, S. Tariq, and M. Gross. Scalable fluid simulation using anisotropic turbulence particles. In ACM SIGGRAPH Asia 2010 papers, SIGGRAPH ASIA '10, pages 174:1–174:8, 2010.

[90] Tobias Pfaff, Nils Thuerey, Andrew Selle, and Markus Gross. Synthetic turbulence using artificial boundary layers. In SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers, pages 1–10, 2009.

[91] J. Pilliod and E. Puckett. Second-order accurate volume-of-fluid algorithms for tracking material interfaces. J. Comput. Phys., 199:465–502, 2004.

[92] J.E. Pilliod et al. Second-order accurate volume-of-fluid algorithms for tracking material interfaces* 1. J. of Comp. Phys., 199(2):465–502, 2004.

[93] D.J. Price. Modelling discontinuities and Kelvin-Helmholtz instabilities in SPH. J. of Comput. Phys., 227(24):10040–10057, 2008.

[94] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable photorealistic liquids. In Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim., pages 193–202, 2004.

[95] N. Rasmussen, D. Nguyen, W. Geiger, and R. Fedkiw. Smoke simulation for large scale phenomena. ACM Trans. Graph. (SIGGRAPH Proc.), 22:703–707, 2003.

[96] W. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. In Comput. Graph. (Proc. of SIGGRAPH 83), volume 17, pages 359–376, 1983.

[97] W. J. Rider and D. B. Kothe. Reconstructing volume tracking. J. Comput. Phys., 141:112–152, 1998.

[98] P.J. Roache. A flux-based modified method of characteristics. Int. J. Num. Meth. in Fluids, 15(11):1259–1275, 1992.

[99] A. Robinson-Mosher, T. Shinar, J.T. Grétarsson, J. Su, and R. Fedkiw. Two-way coupling of fluids to rigid and deformable solids and shells. ACM Trans. on Graphics, 27(3):46:1–46:9, August 2008.

[100] H. Schechter and R. Bridson. Evolving sub-grid turbulence for smoke animation. In SCA '08: Proc. of the 2008 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim., pages 1–7, 2008.

[101] A. Selle, R. Fedkiw, B. Kim, Y. Liu, and J. Rossignac. An Unconditionally Stable MacCormack Method. J. of Sci. Comp., 35(2):350–371, 2008.

[102] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. ACM Trans. Graph. (SIGGRAPH Proc.), 24(3):910–914, 2005.

[103] J. Sewall, N. Galoppo, G. Tsankov, and M. Lin. Visual Simulation of Shockwaves. Graphical Models, 2009.

[104] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock capturing schemes II (two). J. Comput. Phys., 83:32–78, 1989.

[105] J. Stam. Stable fluids. In Proc. of SIGGRAPH 99, pages 121–128, 1999.

[106] J. Stam and E. Fiume. Turbulent wind fields for gaseous phenomena. In Proc. of SIGGRAPH 1993, pages 369–376, 1993.

[107] A. Staniforth and J. Cote. Semi-Lagrangian integration schemes for atmospheric models: A review. Monthly Weather Review, 119:2206–2223, 1991.

[108] J. Strain. Tree methods for moving interfaces. J. Comput. Phys., 151:616–648, 1999.

[109] M. Sussman. A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. J. Comput. Phys., 187:110–136, 2003.

[110] M. Sussman and E. Puckett. A coupled level set and volume-of-fluid method for computing 3D and axisymmetric incompressible two-phase flows. J. Comput. Phys., 162:301–337, 2000.

[111] K. Takizawa, T. Yabe, and T. Nakamura. Multi-dimensional semi-Lagrangian scheme that guarantees exact conservation. Computer Physics Communications, 148(2):137–159, 2002.

[112] R. Tanaka, T. Nakamura, and T. Yabe. Constructing exactly conservative scheme in a non-conservative form. Computer Physics Communications, 126(3):232–243, 2000.

[113] S. Weißmann and U. Pinkall. Filament-based smoke with vortex shedding and variational reconnection. ACM Trans. Graph., 2010.

[114] M Wicke, M Stanton, and A Treuille. Modular bases for fluid dynamics. ACM Trans. Graph., 28(3):1–8, 2009.

[115] Martin Wicke, Matt Stanton, and Adrien Treuille. Modular bases for fluid dynamics. In SIGGRAPH '09: ACM SIGGRAPH 2009 papers, pages 1–8, New York, NY, USA, 2009. ACM.

[116] Christopher Wojtan, N. Thürey, M. H. Gross, and G. Turk. Physics-inspired topology changes for thin fluid features. ACM Trans. Graph., 2010.

[117] P. Woodward and P. Colella. The numerical simulation of two-dimensional fluid flow with strong shocks. J. Comput. Phys., 54:115–173, April 1984.

[118] F. Xiao and T. Yabe. Completely conservative and oscillationless semi-Lagrangian schemes for advection transportation. J. of Comp. Phys., 170(2):498–522, 2001.

[119] T. Yabe, R. Tanaka, T. Nakamura, and F. Xiao. An exactly conservative semi-Lagrangian scheme (CIP–CSL) in one dimension. Monthly Weather Review, 129:332–344, 2001.

[120] G. Yngve, J. O'Brien, and J. Hodgins. Animating explosions. In Proc. of ACM SIGGRAPH 2000, pages 29–36, 2000.

[121] H. Yoon, S. Koshizuka, and Y. Oka. A particle-gridless hybrid method for incompressible flows. Int. J. for Num. Meth. in Fluids, 30:407–424, 1999.

[122] Jong-Chul Yoon, Hyeong Ryeol Kam, Jeong-Mo Hong, Shin-Jin Kang, and Chang-Hun Kim. Procedural synthesis using vortex particle method for fluid simulation. Comput. Graph. Forum, 28(7):1853–1859, 2009.

[123] Jihun Yu and Greg Turk. Reconstructing surfaces of particle-based fluids using anisotropic kernels. In Proc. of the 2010 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim., 2010.

[124] S.T. Zalesak. Fully multidimensional flux-corrected transport algorithms for fluids. J. of Comput. Phys., 31(3):335–362, 1979.

[125] M. Zerroukat, N. Wood, and A. Staniforth. Application of the parabolic spline method (psm) to a multi-dimensional conservative semi-lagrangian transport scheme (slice). J. of Comput. Phys., 225(1):935–948, 2007.

[126] Q. Zhang and P.L.F. Liu. A new interface tracking method: The polygonal area mapping method. J. of Comput. Phys., 227(8):4063–4088, 2008.

[127] Y. Zhu and R. Bridson. Animating sand as a fluid. ACM Trans. Graph. (SIGGRAPH Proc.), 24(3):965–972, 2005.