Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2004)
R. Boulic, D. K. Pai (Editors)

Extended Galilean Invariance for Adaptive Fluid Simulation

Maurya Shah Jonathan M. Cohen[†] Sanjit Patel Penne Lee Frédéric Pighin

University of Southern California, Institute for Creative Technologies
[†]Rhythm and Hues Studios

Abstract

In an unbounded physical domain, simulating a turbulent fluid on an Eulerian grid is rather tricky. Since it is difficult to predict the motion of the fluid, it is also difficult to guess which computational domain would allow the simulation of the fluid without crossing the computational boundaries. To address this dilemma, we have developed a novel adaptive framework where the simulation grid follows the motion of the flow. Our technique is based on the principle of Galilean Invariance and the culling of simulation cells using a metric derived from continuative boundary conditions. We describe our framework and showcase its advantages over traditional techniques. Timing results and visual comparisons are presented.

1. Introduction

Fluid animations arise in many computer graphics applications such as games, visual effects, and training simulations. Their importance has driven the development of fluid simulation techniques suitable for visual applications, which enable visually compelling simulation of fluid flows in tractable computation time.

Buoyancy-driven flows are of particular interest for the entertainment industry. They include explosions, rising smoke stacks, steam jets, and fires. In such flows, the dominant cause of fluid transport is a buoyancy force which is generated by a temperature differential in the fluid medium. Buoyancy-driven flows tend to rise when unbounded by physical barriers, such as walls or other obstacles, until they cool enough that the driving temperature differential drops too low to further propel the flow. Before cooling, a fluid may rise steadily, swirl, flow around fixed objects, or exhibit other unpredictable behavior.

The unpredictability of such flows, combined with their potential to cover a large physical area before dissipating or cooling, makes them particularly difficult to simulate efficiently. A typical technique for simulating these situations is given by [FSJ01], where the incompressible Navier-Stokes Equations are approximately solved using stable methods on a static uniform grid. This technique requires choosing a grid that is large enough so that the visually important part of the flow will never leave the computational domain, while simultaneously choosing a high-enough grid resolution so that the interesting small-scale detail of the flow is not lost. For large-scale phenomena, these competing requirements may result in extraordinarily large computational domains. Running a simulation on such domains typically requires large amount of memory and computation time.

In this paper, we present an adaptive grid technique for simulating efficiently buoyancy-driven flows. Our technique modifies the spatial location and the shape of the grid, not its resolution. We begin with a small computational domain centered around the source of the flow. The flow velocity field and associated material properties are updated using a standard Navier-Stokes solver [Sta03]. Our algorithm tracks the region of the flow that is "interesting," and dynamically moves the computational domain so that the region of interest of the flow is covered by computational cells. To efficiently implement this tracking procedure, we exploit a property of the Navier-Stokes Equations called *Galilean Invariance*. To our knowledge, this is the first application of Galilean Invariance to accelerate flow calculations.

The contributions of this paper are an algorithm for efficiently moving the computational domain through space, and an algorithm for adaptively culling "unimportant" cells from fluid calculations. Implementing these techniques requires small modifications to an existing fluid solver. The combination of the techniques presented in this paper provides for visually compelling 3D simulation of large-scale flows in a fraction of the computation time required by static grid techniques.

2. Related Work

In computer graphics, buoyancy-driven flows are often simulated using Eulerian (grid-based) methods. Previous work has focused on simulating fire [NFJ02], large-scale explosions [RNGF03], compressible and pseudo-compressible explosions [FOA03, YOH00], and hot gases [FSJ01, FM97, SF93].

Our solver is based on Jos Stam's stable fluid code [Sta99, Sta03, FSJ01] which uses a uniform grid with finite differencing. This method is popular in computer graphics because it is fast and stable for large time steps. This solver has also been extended to solve free-surface flows [FF01, EMF02]. While inaccurate in an engineering sense, the algorithm is easy to implement and produces good visual results.

Adaptive grids are often used in Finite Element, Finite Volume, and Finite Difference methods to concentrate computational nodes in areas of highest detail. There are several varieties of adaptive grid techniques, depending on what type of adaptation is allowed. Moving grids [Pen98] allow the positions of the grid vertices to adapt over time. Dynamically adaptive grids [GKS02] refine a grid over the course of a simulation by creating new computational nodes. Statically adaptive grids [ABB*98] determine *a priori* where to refine the computational domain based on predicted regions of higher turbulence.

A different approach for handling high resolution, taken by [RNGF03], is to simulate only 2D slices of the flow and add 3D detail via randomized turbulence. By limiting the simulation domain to 2D slices, the resolution can be increased dramatically without a large performance penalty. However, this technique only works for flows that have a strong symmetry. We wish to use a full 3D technique that allows for arbitrarily shaped flows in an arbitrarily large physical domain.

Gridless Lagrangian techniques based on Smoothed Particle Hydrodynamics [Mon99] have also been used in computer graphics [PTB*03, MCG03, DG96]. While promising for a number of applications, SPH techniques cannot yet match Eulerian finite-difference techniques in terms of efficiency when the simulated fluid occupies a large portion of the simulation space.

3. Eulerian Fluid Simulation

In this section, we present the basic framework used in our simulator. For more information on this technique, we refer to [Sta03] and [FSJ01].

3.1. Navier-Stokes Equations

The Navier-Stokes Equations that govern incompressible buoyancy-driven fluid flows in a homogeneous medium are

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} &= \nu\nabla^2\boldsymbol{u} - \nabla p + \alpha(T - T_0)\boldsymbol{y} \\ &\quad -\beta\rho\boldsymbol{y} + \varepsilon h(\boldsymbol{N}\times\boldsymbol{\omega}) \qquad (1) \\ \nabla\cdot\boldsymbol{u} &= 0 \qquad (2) \\ \frac{\partial T}{\partial t} + (\boldsymbol{u}\cdot\nabla)T &= k\nabla^2 T, \qquad (3) \end{aligned}$$

where $\boldsymbol{u}$ is the fluid velocity field, p is the pressure field, ν is the kinematic viscosity, T is the temperature field, T_0 is the reference ambient temperature, α is a scalar controlling the amount of thermal advection, $\boldsymbol{y}$ is the unit vector pointing up ($(0,1,0)$), β is a scalar influencing gravity, and k is the coefficient of thermal diffusion. The value ε controls the amount of small scale details added to the flow through vorticity confinement.

For rendering, we also advect a scalar density field ρ, that represents dust or particles in the fluid

$$\frac{\partial \rho}{\partial t} + (\boldsymbol{u}\cdot\nabla)\rho = 0. \qquad (4)$$

The set, $\{\boldsymbol{u}, T, \rho\}$, is a complete solution to the Navier-Stokes Equations.

The above equations are discretized into a uniform grid, which we denote as $\mathbf{G}$. $\mathbf{G}$ consists of an axis-aligned lattice of node points, $\{g_{ijk}\}$, which are spaced Δh apart in the x, y, and z directions. The center of this grid is $\boldsymbol{c}$, and $\mathbf{G}$ contains I, J, and K nodes in the x, y, and z dimensions. In other words, $\mathbf{G}$ occupies the axis-aligned rectangle spanning $\boldsymbol{c} \pm \left(\frac{I\Delta h}{2}, \frac{J\Delta h}{2}, \frac{K\Delta h}{2}\right)$.

Since our solver is based on [Sta03], we only briefly outline the method. First, forces and vorticity confinement are added directly to the flow. The diffusion term for viscosity is handled with a stable implicit technique. The resulting flow, denoted $\tilde{\boldsymbol{u}}$, is projected to be divergence-free using a Conjugate-Gradient solver that implicitly solves for pressure, based on the Poisson equation

$$\Delta t\nabla^2 p = \nabla\cdot\tilde{\boldsymbol{u}}. \qquad (5)$$

The projected fluid field is then computed as $\boldsymbol{u} = \tilde{\boldsymbol{u}} - \nabla p$. The advection step is implemented using a semi-Lagrangian approach. Finally, the divergence free velocity field is obtained by performing the projection again. The density and temperature fields are advected and diffused using identical techniques. Overall, this technique is fast, stable, and produces visually compelling buoyancy-driven flows.

3.2. Boundary Conditions

Traditionally, boundary conditions fall into two categories: closed and open boundary conditions. Closed boundary conditions enforce that the fluid cannot pass through the border

of the domain by setting $\boldsymbol{u} \cdot \boldsymbol{n} = 0$ at the boundaries, where $\boldsymbol{n}$ is the direction normal to the domain wall. This can be used to simulate fluid motion in a fixed domain such as water in a tank. Periodic conditions assume a fluid that flows off one side of the domain returns from the other side. This is useful for generating fluid motion that can be tiled together.

In open boundary conditions, fluids can freely flow off the edge of the computational domain. Open boundaries assume that the physical domain is infinite, but that the flow is only computed over a small region. Open boundaries are appropriate for modeling fluid phenomena that occur in large unbounded spaces, such as an outdoor explosion or steam rising from a smoke stack. The behavior of the fluid outside the computational domain is approximated through boundary conditions.

To simulate open boundaries, we use so-called *continuative* boundary conditions that assume the flow continues unchanged past the boundary, by enforcing

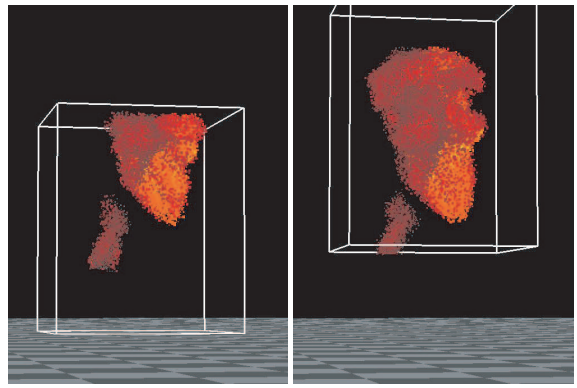
$$(\nabla u_x \cdot n, \nabla u_y \cdot n, \nabla u_z \cdot n) = \boldsymbol{0}. \quad (6)$$

Boundary conditions of this type are considered inaccurate by engineering standards [OS00] because they can generate reflection waves that propagate back from the boundary. A common approach, taken for example by Jiang et al [JSVL99], is to derive more complex boundary conditions designed to cancel these reflective effects. However, for computer graphics applications, simple continuative boundary conditions generate flows that are not affected significantly by the boundaries of the computational domain and therefore suit our purpose.

Continuative boundary conditions are enforced at two stages of the simulation: during the projection step and during the self-advection step. During the projection step, the left-hand side of Equation 5 at a boundary cell is calculated by assuming that the derivative in the direction normal to the boundary is 0. During the advection step, if the semi-Lagrangian back-tracing results in a reference to a location outside the domain, the velocity value of the nearest boundary cell is used.

4. Dynamic Grid Techniques

Because open boundaries approximate an infinite domain, nothing prevents the flow from leaving the extents of the computational domain **G**. As shown in Figure 1 (a), the density field may be advected directly out of the domain, leaving nothing left to render. One remedy is to select a computational domain that is large enough to contain the simulated fluid well within its boundaries. However, this implies either that Δh is large, in which case the solver would be unable to resolve interesting smaller-scale turbulence, or that I, J, and K are large, in which case there would be a large number of nodes in the computational domain, which would make the solver run very slowly. Furthermore, even if we choose a large domain, the unpredictable nature of a turbulent flow makes it very difficult to guess *a priori* just how large the domain needs to be.



(a) (b)

Figure 1: *Fluid simulations: (a) fixed domain, (b) moving domain.*

For these reasons, we are motivated to develop a technique that adapts the location and extents of **G** dynamically based on the characteristics of the evolving flow. Our approach is to combine three techniques: translating the center of the domain to "track" the flow, adjusting the extents to capture the shape of the flow, and culling voxels from **G** which contribute little to the overall flow calculations. The decision as to whether a region must be included in the calculation is based on how close the flow in those voxels meets the continuative boundary conditions. We describe each of these techniques in the following subsections.

4.1. Translating the Domain Using Galilean Invariance

Let us consider a puff of steam rising from a vent. The actual puff is small, but the steam travels a large distance before dissipating. Our idea is to center the computational domain in the middle of the steam puff, and move the center, $\boldsymbol{c}(t)$, as a function of time to keep the puff within **G**. One way to implement this is to remove layers of voxels in regions that the fluid is leaving from and add voxels in regions that the fluid is moving into. Implementing this procedure would require a certain amount of resampling and bookkeeping that could get tricky for a complex flow.

We have developed a simpler approach that exploits a property of the Navier-Stokes Equations called *Galilean Invariance*. Galilean invariance is a principle which states that the fundamental laws of physics are the same in all inertial (uniform-velocity) frames of reference. For fluids, it means that a frame of reference moving at a constant velocity $\boldsymbol{v}$ is equivalent to inducing a uniform flow of $-\boldsymbol{v}$ in the observed flow field $\boldsymbol{u}$. This is the principle behind a wind tunnel, where blowing wind over a wing is equivalent to mov-

ing the wing through motionless air. Figure 2 illustrates this property. For incompressible fluids, Galilean Invariance also holds for an arbitrary rectilinear motion of the frame. In this case, the motion of the frame can be a function of time. This last property is sometime called *Extended Galilean Invariance* [Pop00]. The appendix discusses this property in more details. It is interesting to note that Galilean Invariance no longer holds in a non-inertial rotating frame. In this case, the rotation induces Coriolis and angular acceleration forces [Pop00].

We can exploit Galilean Invariance to efficiently translate the center of the grid by velocity $\boldsymbol{v}(t)$ as follows. If we translate all voxels $\{g_{ijk}\}$ along the velocity $\boldsymbol{v}(t)$, we induce a flow in $\boldsymbol{u}$ equal to $-\boldsymbol{v}(t)$. As a consequence, we can simulate the fluid in a reference frame attached to $\boldsymbol{c}(t)$ by adjusting the velocity at each step by $-\boldsymbol{v}(t)$. We also need to keep track of $\boldsymbol{c}(t)$, the motion of the center of the grid, during the simulation to be able to reconstruct the motion of the fluid in a fixed frame of reference. With this technique, there is no need to add or remove voxels from $\mathbf{G}$, we simply change the location of each cell.

We choose to locate the center of the grid at the center of the set of voxels for which ρ is greater than some threshold. We call the region of space defined by these voxels the *region of interest*. Let $\bar{\rho}(t)$ be the center of the region of interest. To avoid advecting density out of the computational domain, we would like to compensate for the motion of $\bar{\rho}(t)$ before the density advection step. This is not possible since $\bar{\rho}(t)$ can only be computed after the density field has been advected. To address this contradiction, we predict the velocity of $\bar{\rho}(t)$ from the two previous time steps

$$\boldsymbol{v}(t) = \frac{\bar{\rho}(t-\Delta t) - \bar{\rho}(t-2\Delta t)}{\Delta t}, \tag{7}$$

and we define

$$\boldsymbol{c}(t) = \int_0^t \boldsymbol{v}(\tau) d\tau$$

as the positional offset since time 0. Note that $\bar{\rho}(t)$ and $\boldsymbol{c}(t)$ are different points. $\bar{\rho}(t)$ is the center of the region of interest, whereas $\boldsymbol{c}(t)$ is both the center of the grid and the predicted position of $\bar{\rho}(t)$.

According to Galilean Invariance,

$$\{\boldsymbol{u}(\boldsymbol{x},t)), T(\boldsymbol{x},t), \rho(\boldsymbol{x},t)\} \tag{8}$$

and

$$\{\boldsymbol{u}(\boldsymbol{x}-\boldsymbol{c}(t),t) - \boldsymbol{v}(t), T(\boldsymbol{x}-\boldsymbol{c}(t),t), \rho(\boldsymbol{x}-\boldsymbol{c}(t),t)\} \tag{9}$$

are equivalent solutions of Equations 1-4. Expression 8 represents the solution of the Navier-Stokes Equations in a global fixed coordinate system (i.e. "world space"). Expression 9 represents the equivalent solution in a moving frame of reference following the gross motion of the fluid.

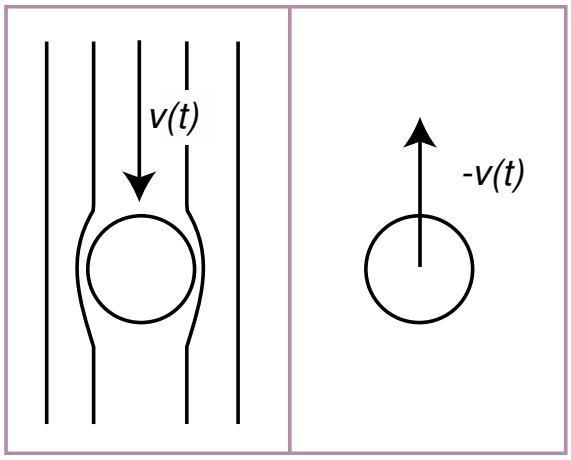


Figure 2: *Galilean Invariance. On the left, a static object is in a uniform flow with velocity $\boldsymbol{v}(t)$. On the right, the object is moving with velocity $\boldsymbol{v}(t)$ in an otherwise static flow. From the point of view of the object, these flows are the same.*

In summary, our algorithm for translating the domain is implemented as follows. At time t, we compute $\boldsymbol{v}(t)$ and $\boldsymbol{c}(t)$. We then store the offset $\boldsymbol{c}(t)$, which is the translation applied to ρ during rendering, and uniformly add $-\boldsymbol{v}(t)$ to the values of $\boldsymbol{u}$ stored at each grid cell g_{ijk}.

As shown in Figure 1 (b), this tracking procedure successfully follows the region of interest.

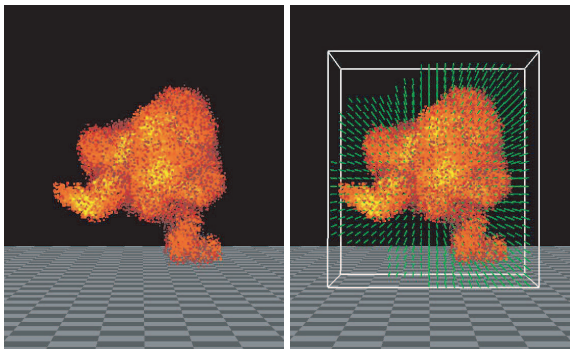
4.2. Reshaping the Domain

We translate the domain to keep the region of interest of the flow centered in the voxel grid. However, the region of interest may change shape significantly over the course of a simulation. To address this issue, we have developed a technique for adjusting the domain to accommodate efficiently the evolving shape of the fluid.

Our method is based on the observation that the boundary conditions as expressed by Equation 6 approximate the behavior of the fluid at the boundaries of the computational volume. In order to adapt the boundaries to the evolution of the flow, we consider the voxels for which this condition is approximatively met. The main idea is that the boundary conditions would not affect the computations if they are already met prior to being enforced.

Equation 6 sets the directional derivative of the flow field to be 0 across the boundaries. We compute the three axis-aligned directional derivatives at each voxel (i,j,k) using central differences

$$\boldsymbol{B}_{i,j,k} = \frac{1}{2\Delta h} \begin{pmatrix} u_x(i+1,j,k) - u_x(i-1,j,k) \\ u_y(i,j+1,k) - u_y(i,j-1,k) \\ u_z(i,j,k+1) - u_z(i,j,k-1) \end{pmatrix}. \tag{10}$$



(a) (b)

Figure 3: *Adaptive grid: (a) rendered fluid, (b) grid visualization (the green dots represent the voxels used in the computations).*

If for a given voxel a component of the vector $\boldsymbol{B}_{i,j,k}$ is close to 0, we predict that placing a boundary at this voxel, perpendicular to the corresponding axis, would have a minimal impact on the simulation. Based on this analysis, we have developed a simple algorithm for adjusting the boundaries of the domain $\mathbf{G}$ at each simulation step. We first compute the directional derivatives, $\boldsymbol{B}_{i,j,k}$, at each voxel. Next, we compute the bounding volume of all the voxels that have at least one of its directional derivative components above a given threshold. This bounding volume becomes the new computational volume. We take this reshaping procedure even further and cull arbitrary voxels from the list of voxels used in the simulation. Our algorithm discards a voxel if it and its neighbors meet one of the boundary conditions. We define the neighborhood of a voxel as being all the voxels within a sphere of radius $6 \times \Delta h$. By changing this radius, we can control how aggressively this procedure culls the computational grid. In our implementation, at the beginning of each new simulation step, we flag the voxels that will be culled and ignore these in the computations. This does not require us to reallocate memory.

Figure 3 illustrates our adaptive grid technique. In Figure 3 (b), the voxels used in the flow computations are represented in green. The white surrounding box visualizes the adaptive boundaries of the simulation grid.

In the next section, we demonstrate through a set of experiments how these techniques can yield considerable speed up over traditional static simulation grids while preserving visual quality suitable for graphics applications.

5. Results

For fast-rising flows, our technique produces good visual results on relatively small computational grids. Figure 5 shows the comparison of simulation 4 (see Table 1), between the motion of the fluid in a static and adaptive grids. All three simulations have similar visual quality.

Table 1 shows timing results for several experiments. All experiments were performed on a 3 GHz PC. Given the same initial conditions, we studied the performances of three algorithms. The first one is a semi-Lagrangian technique using a static grid, the second one uses a moving domain as explained in Section 4.1, and the third one uses a reshaped domain as explained in Section 4.2. Column 2 gives the number of iterations for which each experiment was run. Columns 3, 5, and 8 show the number of voxels included in simulation for the static, translated, and the adaptive grid respectively. For the static and the translated grid, this is a constant number, whereas for the reshaped grid we store the average number of voxels over the entire simulation. Similarly, columns 4, 6 and 9 show the total time in seconds required for the experiment under each grid type. The table shows the speedup obtained using the translated and the reshaped domain for each experiment in columns 7 and 10 respectively. For a given simulation, we decide the size of the static grid by using the displacement of the domain computed using the translated grid. Experiment 4 shows that for a very high-rising flow, our algorithm runs up to 5.3 times faster compared to a static grid.

Our technique is significantly faster than one that uses a large static domain because asymptotically, the advection, force, and vorticity confinement require $O(n)$ time, where n is the number of grid cells in $\mathbf{G}$. However, the cost of the diffusion and projection steps is significantly worse than linear. The sparse Preconditioned Conjugate-Gradient [NW99] solver we are using converges in approximately $O(n^{1.5})$ iterations. Multigrid solvers [WE04] appear to converge in $O(n)$ time, but they are notoriously difficult to implement for irregular domains. Also, the reshaped grid is faster than the translated grid because it adapts the grid to the shape of the fluid, hence a lot of computation time is saved when the bounding volume is much smaller compared to the actual grid size. Because the continuative boundary condition is only an approximation to the true flow, the detail of the computed flow can be different for the adaptive technique than for the static technique. However, this depends on how aggressively the particles are culled in the reshaped grid as discussed in section 4.2. As can be seen from the examples in the video and Figure 5, the overall character and visual fidelity of the flows is maintained. We believe this quality is high enough for many applications such as visual effects, virtual reality simulations, and games.

By limiting the domain, one might argue that our algorithm seriously compromise the accuracy of the simulations. In general, a disadvantage of semi-Lagrangian schemes is that they do not formally conserve integral invariants such as total mass. A posteriori correction is needed to enforce conservation of such quantities. In our case, we use the Hodge decomposition to project the flow onto a mass conserving

Experiments	Iterations	Static Grid		Translated Grid			Reshaped Grid		
		Voxels	Time	Voxels	Time	Speed up	Voxels	Time	Speed up
1	50	270000	1239	150000	612	2.0	119110	488	2.5
2	100	380250	3800	242000	2198	1.7	186257	1691	2.3
3	100	539000	5970	288000	2677	2.2	195583	1761	3.4
4	50	490000	2669	338000	1724	1.5	113074	501	5.3
5	100	490000	5332	288000	2751	1.9	184750	1704	3.1

Table 1: *Timing results.*

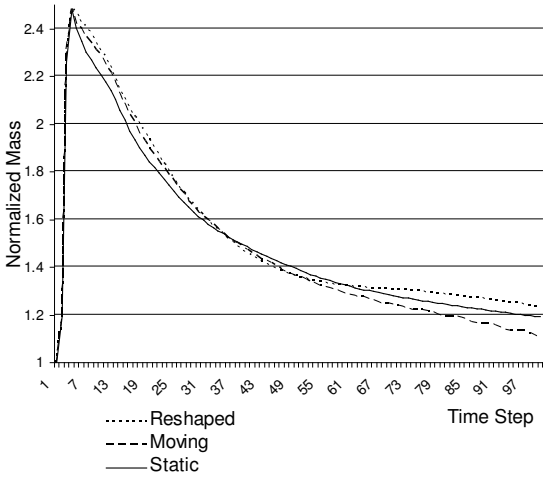


Figure 4: *Mass fluctuations.*

field. However, the solver we are using for the Poisson equation does not preserve mass. In Figure 4, we show the normalized mass (with respect to the original mass) as a function of time for experiment 2 for the static grid, moving grid, and reshaped grid. We observe that the normalized mass fluctuates with all three techniques. The algorithms we propose in this paper do not produce worse mass fluctuations; in fact the three algorithms perform in remarkably similar ways as far as mass preservation is concerned.

The conservative nature of our technique allows us to reduce the number of grid cells, resulting in significant speedups for flows that cover a large physical domain. Furthermore, our adaptive strategy for determining which cells to cull allows a smooth control of speed versus simulation quality. By setting the refinement threshold low, we are less likely to cull important cells, resulting in a more accurate simulation at the cost of greater computation time. Setting a lower value higher threshold will result in a domain that tightly fits the cells which are most turbulent.

6. Conclusion

We have presented a suite of adaptive grid techniques that allows efficient simulation of flows over a large physical domain. These techniques include an efficient and easy to implement method that translates the domain. We also provide a method for adapting the extents of the domain to match the shape of the flow based on the satisfaction of continuative boundary conditions.

As future work, we are interested in further pushing towards real-time fluid flow applications. The metric derived in Equation 10 provides a structured way to rank how important each cell in the domain is to the simulation. We can use this to design a priority scheme that would allow the simulation of flows within a limited computational budget while striving for the best possible simulation quality. We believe that by combining a computational priority scheme such as this with additional adaptive grid techniques or a more efficient solver such as multigrid, we will be able to achieve high quality 3D fluid simulations suitable for games and other interactive applications. Also, we would like to explore better predictors for the velocity, $v(t)$, of the center of the region of interest. For instance, a Kalman filter [GA93] framework might yield better estimates.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. We thank Wen C. Tien for his assistance during the paper submission and Jerry Tessendorf for helpful discussions on Galilean Invariance. This paper was developed with funds of the Department of the Army under contract number DAAD 19-99-D-0046. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the Department of the Army.

Appendix

We justfify the use of Galilean Invariance for a frame of reference under rectilinear acceleration [Pop00]. In what follows, we use Einstein notation: the repetition of an index variable implies a summation over all values of this index.

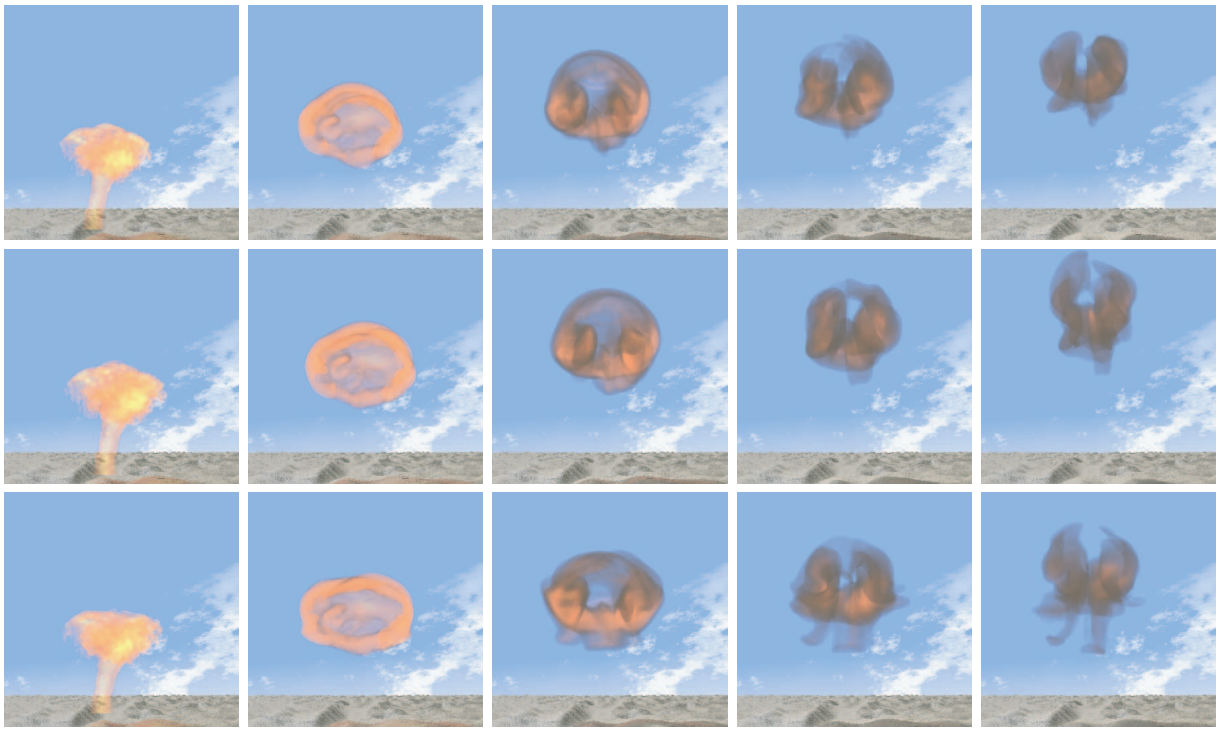


Figure 5: *Comparison between static grid and our adaptive grid algorithms. The top row of images shows the fluid simulated with a static grid. The middle row shows the fluid simulated with a the moving domain technique described in Section 4.1 (using the same initial conditions). The bottom row shows the fluid simulated with the adaptive technique described in Section 4.2.*

Let $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{c}(t)$, where $\boldsymbol{c}(t) = \int_0^t \boldsymbol{v}(\tau) d\tau$, be a change of coordinate system. Let us consider $\boldsymbol{u}(\boldsymbol{x},t)$, a flow, and $\rho(\boldsymbol{x},t)$, a material property of the flow. In other words, ρ is passively advected through the flow

$$\frac{\partial \rho(\boldsymbol{x},t)}{\partial t} + u_i(\boldsymbol{x},t)\frac{\partial \rho(\boldsymbol{x},t)}{\partial x_i} = 0. \tag{11}$$

Let us rewrite this equation using $\boldsymbol{x}'$

$$\frac{\partial \rho(\boldsymbol{x'}-\boldsymbol{c},t)}{\partial t} + u_i(\boldsymbol{x'}-\boldsymbol{c},t)\frac{\partial \rho(\boldsymbol{x'}-\boldsymbol{c},t)}{\partial x_i} = 0.$$

In the rest of this section, we omit the variables to improve readability. We notice that $\frac{\partial .}{\partial x_i} = \frac{\partial .}{\partial x'_i}$ and expand the previous expression further using the chain rule

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho}{\partial x'_i}\frac{\partial (x'_i - c_i)}{\partial t} + u_i \frac{\partial \rho}{\partial x'_i} = 0.$$

Since $\frac{\partial c}{\partial t} = v$ and by redistributing terms, we finally get

$$\frac{\partial \rho}{\partial t} + (u_i - v_i)\frac{\partial \rho}{\partial x'_i} = 0.$$

This equation is the same as Equation 11, if we consider the velocity $\boldsymbol{u}' = \boldsymbol{u} - \boldsymbol{v}$.

Let us now consider the case of self advection. Following Equation 2 we can write

$$\frac{\partial \boldsymbol{u}}{\partial t} + u_i \frac{\partial \boldsymbol{u}}{\partial x_i} = \text{RHS}, \tag{12}$$

where RHS is the right hand side of Equation 2. Following similar steps leads to

$$\frac{\partial \boldsymbol{u}}{\partial t} + u'_i \frac{\partial \boldsymbol{u}}{\partial x'_i} = \text{RHS}.$$

We then derive a PDE on u'

$$\frac{\partial (\boldsymbol{u}' + \boldsymbol{v})}{\partial t} + u'_i \frac{\partial (\boldsymbol{u}' + \boldsymbol{v})}{\partial x'_i} = \text{RHS}.$$

$$\frac{\partial \boldsymbol{u'}}{\partial t} + \frac{\partial \boldsymbol{v}}{\partial t} + u'_i \frac{\partial \boldsymbol{u'}}{\partial x'_i} = \text{RHS}.$$

The term $\frac{\partial \boldsymbol{v}}{\partial t}$ is the acceleration of the moving frame of reference. We can move it to the right hand side where it can be absorbed in a modified pressure term

$$\nabla p + \frac{\partial \boldsymbol{v}}{\partial t} = \nabla(p + \boldsymbol{x}' \cdot \frac{\partial \boldsymbol{v}}{\partial t}).$$

Consequently the Navier-Stokes Equations remain the same under the transformation

$$\begin{aligned} \boldsymbol{x}' &= \boldsymbol{x} + \int_0^t \boldsymbol{v}(\tau) d\tau, \\ p' &= p + \boldsymbol{x}' \cdot \frac{\partial \boldsymbol{v}}{\partial t}. \end{aligned}$$

As a conclusion, the Navier-Stokes Equations for uncompressible fluids are invariant under rectilinear accelerations of the frame of reference.

References

[ABB*98] AHMAD N., BACON D., BOYBEYI Z., DUNN T., HALL M., LEE P., MAYS D., SARMA R. A., TURNER. M.: A solution-adaptive grid generation scheme for atmospheric flow simulations. In *6 th International Conference on Numerical Grid Generation in Computational Field Simulations* (July 1998), pp. 327–335.

[DG96] DESBRUN M., GASCUEL M. P.: Smoothed particles: a new paradigm for animating highly deformable bodies. In *EGCAS '96: Seventh International Workshop on Computer Animation and Simulation* (1996).

[EMF02] ENRIGHT D. P., MARSCHNER S. R., FEDKIW R. P.: Animation and rendering of complex water surfaces. *ACM Transactions on Graphics 21*, 3 (July 2002), 736–744.

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *SIGGRAPH 2001 Conference Proceedings* (August 2001), pp. 23–30.

[FM97] FOSTER N., METAXAS D.: Modeling the motion of a hot, turbulent gas. In *Proceedings of SIGGRAPH 97* (Aug. 1997), pp. 181–188.

[FOA03] FELDMAN B. E., O'BRIEN J. F., ARIKAN O.: Animating suspended particle explosions. *ACM Transactions on Graphics 22*, 3 (July 2003), 708–715.

[FSJ01] FEDKIW R., STAM J., JENSEN H. W.: Visual simulation of smoke. In *SIGGRAPH 2001 Conference Proceedings* (August 2001), pp. 15–22.

[GA93] GREWAL M. S., ANDREWS A. P.: *Kalman filtering: theory and practice*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1993.

[GKS02] GRINSPUN E., KRYSL P., SCHRÖDER P.: Charms: A simple framework for adaptive simulation. *ACM Transactions on Graphics 21*, 3 (July 2002), 281–290.

[JSVL99] JIANG L., SHAN H., VISBAL M., LIU C.: Nonreflecting boundary conditions in curvilinear coordinates. In *Proceedings of 2nd AFSOR International Conference on DNS/LES* (June 1999).

[MCG03] MULLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proceedings of the Symposium on Computer Animation 2003* (2003), pp. 154–159.

[Mon99] MONAGHAN J.: Simulating free surface flows with SPH. *Journal of Computational Physics 110*, 2 (199), 399–406.

[NFJ02] NGUYEN D. Q., FEDKIW R., JENSEN H. W.: Physically based modeling and animation of fire. In *SIGGRAPH 2002 Conference Proceedings* (August 2002), pp. 721–728.

[NW99] NOCEDAL J., WRIGHT S.: *Numerical Optimization*. Springer, New York, 1999.

[OS00] OL'SHANSKII M. A., STAROVEROV V. M.: On simulation of outflow boundary conditions in finite difference calculations for incompressible fluid. *International Journal for Numerical Methods in Fluids 33* (2000), 499–534.

[Pen98] PEN U.-L.: A high-resolution adaptive moving mesh hydrodynamic algorithm. *Astrophys. J. Suppl 115* (1998), 19–34.

[Pop00] POPE S. B.: *Turbulent Flows*. Cambridge University Press, 2000.

[PTB*03] PREMOZE S., TASDIZEN T., BIGLER J., LEFOHN A., WHITAKER R.: Particle based simulation of fluids. In *Proceedings of Eurographics 2003* (September 2003).

[RNGF03] RASMUSSEN N., NGUYEN D. Q., GEIGER W., FEDKIW R. P.: Smoke simulation for large-scale phenomena. *ACM Transactions on Graphics 22*, 3 (July 2003), 703–707.

[SF93] STAM J., FIUME E.: Turbulent wind fields for gaseous phenomena. In *Proceedings of SIGGRAPH 1993* (Aug. 1993), pp. 369–376.

[Sta99] STAM J.: Stable fluids. In *Proceedings of SIGGRAPH 99* (Aug. 1999), Computer Graphics Proceedings, Annual Conference Series, pp. 121–128.

[Sta03] STAM J.: Real-time fluid dynamics for games. In *Proceedings of the Game Developer Conference* (March 2003).