Simple and Fast Fluids

Martin Guay, Fabrice Colin, Richard Egli

Simple and Fast Fluids

Martin Guay Fabrice Colin Richard Egli

24/03/2011

Département d'informatique
Université de Sherbrooke
Sherbrooke (Qc), Canada, J1K 2R1
martin.guay3@usherbrooke.ca

- Author Preprint -

1 Introduction

In this chapter, we present a simple and efficient algorithm for the simulation of fluid flow directly on the GPU using a single pixel shader. By temporarily relaxing the incompressibility condition, we are able to solve the full Navier-Stokes equations over the domain in a single pass.

1.1 Simplicity and Speed

Solving the equations in an explicit finite difference scheme is extremely simple. We believe that anybody who can write the code for a blur post process can implement this algorithm and simulate fluid flow efficiently on the GPU. The code holds in less than 40 lines and is so simple we actually had the solver running in FxComposer (see FxComposer demo). Solving every fluid cell (texel) locally in a

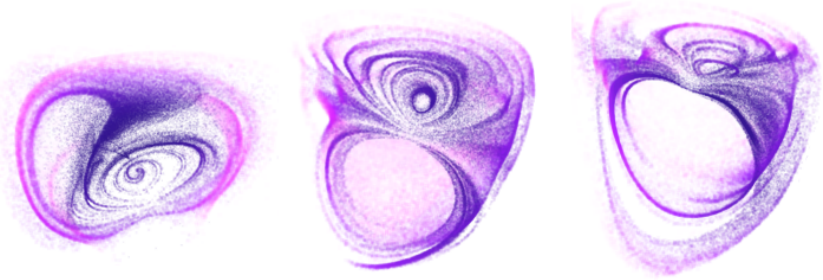

Figure 0.1: 3D simulation with 100k particles visualization.

single pass is not only simple, but also quite fast. In fact, the implementation of this algorithm on the GPU is at least 100 times faster than on the CPU[1]. In this chapter, we show how to couple the two equations of the classical Navier-Stokes equations into a single phase process; then follows a detailed explanation of the algorithm along with example code.

2 Fluid Modeling

The greatest feature of physics-based modeling in computer graphics is the ability for a model to cope with its environment and produce realistic motion and behavior. Attempting to animate fluids non-physically is by experience, a non-trivial task. In order to physically model the motion of fluids, the simplified classical Navier-Stokes equations for incompressible fluids are a good description of such mechanics and a solver based on this model is capable of simulating a large class of fluid-like phenomena. Fortunately a deep understanding of the partial differential equations involved is not required in order to implement such a solver.

2.1 Navier-Stokes Equations

The Navier-Stokes equations, widely used in numerous areas of fluid dynamics, derived from two very simple and intuitive principles of mechanics: the conservation of momentum and the conservation of mass.

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla P + \rho\mathbf{g} + \mu\nabla^2\mathbf{u}, \qquad \text{(momentum conservation eq.)}$$

where $\mathbf{u}$ is a velocity vector, $\mathbf{g}$ is the gravitational acceleration vector, μ the viscosity, P the pressure and where ∇^2 stands for the Laplacian operator $\partial^2/\partial x^2 + \partial^2/\partial y^2 + \partial^2/\partial z^2$.

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0, \qquad \text{(mass conservation eq.)}$$

where $\nabla\cdot$ represents the divergence operator. These equations also have to be supplemented with boundary conditions of Dirichlet, Neumann or even of Robin type. Usually, the incompressibility condition is imposed on the fluid by assuming that its density ρ remains constant over time. Using the latter assumption, Equation **??** simplifies to

$$\nabla\cdot\mathbf{u} = 0. \qquad \text{(incompressibility condition eq.)}$$

Now one of the main features of the formulation of the Navier-Stokes equations illustrated here is the possibility, when working with regular grid domains, to use classical finite difference schemes.

[1]The simulation runs on the CPU at 8.5 fps with 4 threads on an intel Core 2 Quad @ 2.66 GHz simulating only the velocity field over a 256x256 grid. Keeping the same grid size, the simulation of both velocity and density fields, runs at more than 2500 fps on a Geforce 9800 GT using 32-bit floating point render targets. Note that 16-bit floating point is sufficient to represent velocities.

2.2 Density-Invariance Algorithm

Unfortunately, a direct application of Equation **??** results in a non zero divergence field $\mathbf{u}$ i.e. Equation **??** is no longer satisfied by the vector field $\mathbf{u}$. A lot of popular methods for simulating fluids consist in two main steps. First some temporary compressibility is allowed when Equation **??** is solved, then in a second step a correction is applied to the vector field obtained in order to fulfill the incompressibility condition. This correction can be done by considering a projection of the resulting vector $\mathbf{w}$ field onto its divergence free part (see [9]). The latter projection can also be performed in the spirit of Smoothed Particle Hydrodynamics (SPH) method (see for instance [1]). Another way to deal with this incompressibility problem is to take advantage of the relation between the divergence of the vector field and the local density given by Equation **??**, by trying to enforce a density invariance (see among others, [7]). Recently, some techniques combining the two preceding approaches were studied (for an exhaustive study of all the previous techniques in the SPH context, see [10]).

We choose an approach based on density invariance. It is interesting to notice that an algorithm based on the previous approach has proven to be stable for the SPH method [7]. First, observe that the equation Equation **??** can be rewritten as;

$$\frac{\partial \rho}{\partial t} = -\mathbf{u} \cdot \nabla \rho - \rho \nabla \cdot \mathbf{u},$$

clearly illustrating the link between the divergence of the vector field and the variation of the local density. After solving the above equation for density, a corrective pressure field could simply be given as:

$$P = K(\rho^n - \rho_0), \qquad \text{(density-invariant field)}$$

where ρ_0 is the rest (initial) density and where the constant K is often chosen accordingly to the gas state equation (see [3] or [6]).

The corrective field P could be interpreted as an internal pressure whose gradient corrects the original velocity field to get a null divergence vector field. Since we are only interested in its derivative, there is no need to retain the P variable and the corresponding correction to be applied is simply given by

$$\nabla P = K \nabla \rho.$$

Now before jumping directly to code, we first need to discretize the formulation; a topic covered in the next section.

2.3 From Math to Code: Numerical Scheme

One of the main features of the formulation used in this chapter is the ability to use, along a regular grid, simple finite differences. Since the texture is the default data structure on a GPU, a regular grid holding 4 values per cell comes as a natural choice for the spatial discretization of the domain. Also, being an Eulerian formulation, the spatial discretization stays fixed throughout the simulation and the

neighborhood of an element, the elements (in our case, the texels) around it, will always remain the same; therefore greatly simplifying the solver's code.

The simplicity of finite differences along a one phase coupling of both momentum and mass conservation equations through a density-invariant field enables the elaboration of a very simple algorithm to solve fluid flow in a single step. This algorithm is illustrated in the next section; note that other grid-based methods on the GPU exist and the interested reader can refer to [2] for a multi-steps but unconditionally stable simulation or to [5] (also available in GPU Gems 2) for a lattice-based simulation.

3 Solver's Algorithm

A solution to the Navier-Stokes equations is a vector-valued function $\mathbf{u}$ and a scalar-valued function ρ which satisfies the momentum and mass conservation equations. These functions are spatially discretized on a texture where quantities $\mathbf{u}$ and ρ are stored at the texel's center. In order to update a solution $\mathbf{u}$ and ρ from time t_n to time t_{n+1}, we traverse the grid once and solve every texel in the following manner:

1. Solve the mass conservation equation for density by computing the differential operators with central finite differences and integrating the solution with the forward Euler method.
2. Solve the momentum conservation equation for velocity in two conceptual steps:
   (a) Solve the transport equation using the semi-lagrangian scheme.
   (b) Solve the rest of the momentum conservation equation using the same framework as in 1.
3. Impose Neumann boundary conditions.

3.1 Conservation of Mass

In order to compute the right-hand side of the mass conservation equation, we need to evaluate a gradient operator ∇ for a scalar-valued function ρ and a divergence operator $\nabla\cdot$ for a vector-valued function $\mathbf{u}$ both respectively expressed using central finite differences as follows:

$$\nabla\rho^n_{i,j,k} = \left(\frac{\rho^n_{i+1,j,k}-\rho^n_{i-1,j,k}}{2\Delta x}, \frac{\rho^n_{i,j+1,k}-\rho^n_{i,j-1,k}}{2\Delta y}, \frac{\rho^n_{i,j,k+1}-\rho^n_{i,j,k-1}}{2\Delta z}\right),$$
$$\nabla\cdot\mathbf{u}^n_{i,j,k} = \frac{u^n_{i+1,j,k}-u^n_{i-1,j,k}}{2\Delta x} + \frac{v^n_{i,j+1,k}-v^n_{i,j-1,k}}{2\Delta y} + \frac{w^n_{i,j,k+1}-w^n_{i,j,k-1}}{2\Delta z}.$$

And finally, an integration is performed using forward euler over a time step Δt:

$$\rho^{n+1}_{i,j,k} = \rho^n_{i,j,k} + \Delta t(-\mathbf{u}^n_{i,j,k}\cdot\nabla\rho^n_{i,j,k} - \rho^n_{i,j,k}\nabla\cdot\mathbf{u}^n_{i,j,k}).$$

Indeed there exist other finite difference schemes. For instance, one could use upwinding for the transport term or literally semi-lagrangian advection. Unfortunately, the latter results in much numerical dissipation, an issue covered in the next section.

3.2 Conservation of Momentum

We solve the momentum conservation equation for velocity $\mathbf{u}$ in two conceptual steps. The first consists in solving the transport equation $\frac{\partial \mathbf{u}}{\partial t} = -\mathbf{u} \cdot \nabla \mathbf{u}$ with a semi-lagrangian scheme, then solve the rest of the momentum conservation equation ($\frac{\partial \mathbf{u}}{\partial t} = -\frac{\nabla P}{\rho} + \mathbf{g} + \frac{\mu}{\rho}\nabla^2 \mathbf{u}$) with central finite differences and forward Euler integration.

3.2.1 Semi-Lagrangian Scheme

First introduced to computer graphics by Joe Stam in the paper *Stable Fluids* [9], the following scheme is quite useful for solving the generic transport equation given by

$$\frac{\partial \phi}{\partial t} = -\mathbf{u} \cdot \nabla \phi, \tag{3.1}$$

at the current texel's position $\mathbf{x}_{i,j,k}$ with

$$\phi^{n+1}_{i,j,k}(\mathbf{x}_{i,j,k}) = \phi^n(\mathbf{x}_{i,j,k} - \Delta t \mathbf{u}^n_{i,j,k}). \tag{3.2}$$

The idea is to solve the transport equation from a lagrangian viewpoint where the spatial discretization element holding quantities (ex: a particle) moves along the flow of the fluid and answer the following question: where was this element at the previous time step if it has been transported by a field $\mathbf{u}$ and ended up at the current texel's center at the present time? Finally, we use the sampled quantity to set it as the new value of the current texel.

Now when solving the transport equation for velocity, equation Equation **??** becomes $\frac{\partial \mathbf{u}}{\partial t} = -\mathbf{u} \cdot \nabla \mathbf{u}$ and is solved with $\mathbf{u}^{n+1}_{i,j,k}(\mathbf{x}_{i,j,k}) = \mathbf{u}^n(\mathbf{x}_{i,j,k} - \Delta t \mathbf{u}^n_{i,j,k})$.

This method is not only straightforward to implement on the GPU with linear samplers but also unconditionally stable. Unfortunately, quantities advected in this fashion suffer from dramatic numerical diffusion and higher order schemes exist to avoid this issue such as McCormack schemes discussed in [8]. These schemes are especially useful when advecting visual densities as mentioned in the section on fluid visualization.

3.2.2 Density Invariance and Diffusion Forces

After solving the transport term, the rest of the momentum conservation equation is solved with central finite differences. Here is a quick reminder of the part of the Equation **??** which is not yet solved:

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{\nabla P}{\rho} + \mathbf{g} + \frac{\mu}{\rho}\nabla^2 \mathbf{u}.$$

As mentioned earlier, the gradient of the density-invariant field ∇P is equivalent to the density gradient $\nabla \rho$, i.e. $\nabla P \simeq K \nabla \rho$. Since we already computed the density gradient $\nabla \rho^n_{i,j,k}$ when solving the mass conservation equation with (3.1), we need only to scale by K in order to compute the "pressure" gradient ∇P. As for the diffusion term, a Laplacian ∇^2 must be computed. This operator is now expressed using a second order central finite difference scheme:

$$\nabla^2 \mathbf{u}^n_{i,j,k} = (L(u),\, L(v),\, L(w)),$$

where

$$L(f) = \left(\frac{f^n_{i+1,j,k} - 2f^n_{i,j,k} + f^n_{i-1,j,k}}{(\Delta x)^2} + \frac{f^n_{i,j+1,k} - 2f^n_{i,j,k} + f^n_{i,j-1,k}}{(\Delta y)^2} + \frac{f^n_{i,j,k+1} - 2f^n_{i,j,k} + f^n_{i,j,k-1}}{(\Delta z)^2} \right),$$

for any twice continuously differentiable function f. Finally, an integration is performed using forward euler over a time step Δt:

$$\mathbf{u}^{n+1}_{i,j,k} = \mathbf{u}^n_{i,j,k} + \Delta t(-S\nabla \rho^n_{i,j,k} + \mathbf{g} + \nu \nabla^{\mathbf{2}} \mathbf{u}^{\mathbf{n}}_{\mathbf{i,j,k}}).$$

Since the density ρ value should not vary much from ρ_0, we can interpret the $\frac{1}{\rho}$ scale as a constant held by $\nu := \frac{\mu}{\rho_0}$ and $S := K\frac{(\Delta x)^2}{\Delta t \rho_0}$. One can see how we also scale by $\frac{(\Delta x)^2}{\Delta t}$ which seems to give better results (we found $(\Delta x)^2$ while testing over a 2D simulation) and a sound justification has still to be found and will be the subject of future work.

Yet, we solved the equations without bothering about neither boundary conditions (obstacles) nor numerical stability. These two topics are respectively covered in the next two sections.

3.3 Boundary Conditions

Boundary conditions are specified at the boundary (surface) of the fluid in order for the fluid to satisfy specific behaviors. They are essential for the interactions between the fluid and the different types of matter such as solids (obstacles) in the domain. Neumann boundary conditions are considered for fluid-solid interactions. Our method does not provide free surface boundaries for fluid-fluid interactions

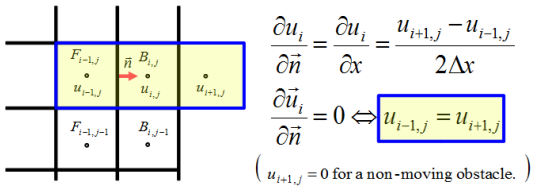

Figure 3.1: Neumann Boundary Conditions.

such as liquid and air interactions necessary for water animations. When simulating smoke or fire, for instance, it is possible to consider the air and gas as a single fluid. Therefore only Neumann boundary conditions are required. Hence in the proposed implementation, computational cells are either tagged as fluid or boundary cells. Note that velocities and densities are defined on all cell types. Before discussing in depth Neumann boundary conditions, it is convenient to first consider the simplest boundary condition which is the Dirichlet boundary condition which means ”to set directly”; hence, one could set the border cells to null velocities and densities to an initial value and make sure the simulation works well before introducing obstacles.

3.3.1 Neumann Boundary Conditions

The Neumann boundary condition specifies the values of the derivative in the direction of the outward normal vector at the boundary. To prevent the fluid from entering obstacles, this condition would come down to having the obstacle's corresponding boundary cells fulfill $\frac{\partial f}{\partial n} = 0$ for every component $f \in \{u,\ v,\ w\}$ of the vector function $\mathbf{u}$, therefore the fluid cells around a boundary cell need to be correctly adjusted in order to satisfy this condition. The derivative in the normal direction at such a boundary cell is quite trivial when assuming that the walls of an obstacle are always coincident with the face of a computational cell (i.e. obstacles would fill completely computational cells). With this supposition, the derivative $\frac{\partial f}{\partial n}$ is either given by $\pm\frac{\partial u}{\partial x}$, $\pm\frac{\partial v}{\partial y}$ or $\pm\frac{\partial w}{\partial z}$ for u, v, and w respectively. As an example, one can see how the fluid cell $F_{i-1,j}$ is adjusted according to the derivative of the boundary cell $B_{i,j}$ in figure 3.1.

The true difficulty is the actual tracking of obstacles, specifically when working with dynamic 3D scenes where objects must first be voxelized in order to be treated by the algorithm. See [2] for a possible voxelization method.

3.4 Stability Conditions

Explicit integration is very simple from both analytical and programmatic points of view, but is only conditionally stable; meaning the time step value Δt has an upper bound defined by a ratio relative to

the spatial resolution $\Delta\mathbf{x}$ over the function's range $\mathbf{u}$:

$$\Delta t < \max\left\{\left|\frac{\Delta x}{u}\right|, \left|\frac{\Delta y}{v}\right|, \left|\frac{\Delta z}{w}\right|\right\}.$$

This condition must be satisfied everywhere in the domain.

4 Code

Short and simple code for the 2D solver is presented in this section. In two dimensions, the first two pixel x, y-components hold velocities while the z-component holds the density. Setting $\Delta x = \Delta y = 1$ greatly simplifies the code. A 3D demo is also available on the CD. $(K \simeq 0.2, \Delta t = 0.15)$

Listing 1: 2D Solver, shader model 2_a

```
///< Central Finite Differences Scale.
float2 CScale = 1.0f/2.0f;

float S=K/dt;

float4 FC = tex2D(FieldSampler,UV);
float3 FR = tex2D(FieldSampler,UV+float2(Step.x,0));
float3 FL = tex2D(FieldSampler,UV-float2(Step.x,0));
float3 FT = tex2D(FieldSampler,UV+float2(0,Step.y));
float3 FD = tex2D(FieldSampler,UV-float2(0,Step.y));

float4x3 FieldMat = {FR,FL,FT,FD};

//du/dx, du/dy
float3 UdX = float3(FieldMat[0]-FieldMat[1])*CScale;
float3 UdY = float3(FieldMat[2]-FieldMat[3])*CScale;

float Udiv = UdX.x+UdY.y;
float2 DdX = float2(UdX.z,UdY.z);

///<
///< Solve for density.
///<
FC.z -=  dt*dot(float3(DdX,Udiv),FC.xyz);

///< Related to stability.
FC.z = clamp(FC.z,0.5f,3.0f);
```

```
///<
///< Solve for Velocity.
///<
float2 PdX = S*DdX;
float2 Laplacian = mul((float4)1,(float4x2)FieldMat)-4.0f*FC.xy;
float2 ViscosityForce = v*Laplacian;

///< Semi-lagrangian advection.
float2 Was  = UV - dt*FC.xy*Step;
FC.xy = tex2D(FieldLinearSampler,Was).xy;

FC.xy += dt*(ViscosityForce - PdX + ExternalForces);

///< Boundary conditions.
for (int i=0; i<4; ++i)
{
if (IsBoundary(UV+Step*Directions[i]))
{
float2 SetToZero = (1-abs(Directions[i]));
FC.xy *= SetToZero;
}
}
return FC;
```

5 Visualization

One of the disadvantages of the Eulerian formulation is the lack of geometric information about the fluid. Up to now we have captured its motion with the velocity field but we still don't know its shape. Nevertheless there are many ways to visualize a fluid. In this section we briefly discuss two simple techniques. The first consists in advecting particles under the computed velocity field and the second, in advecting a scalar density field.

5.1 Particles

Using particles in a one-way interaction with the fluid is by far the most simple and efficient technique for visualizing a fluid. Since the velocity field is computed on the GPU; the whole system can run independently with very few interactions with the CPU. Once the particles are initialized, we only sample the velocity field in order to update their positions as illustrated in code:

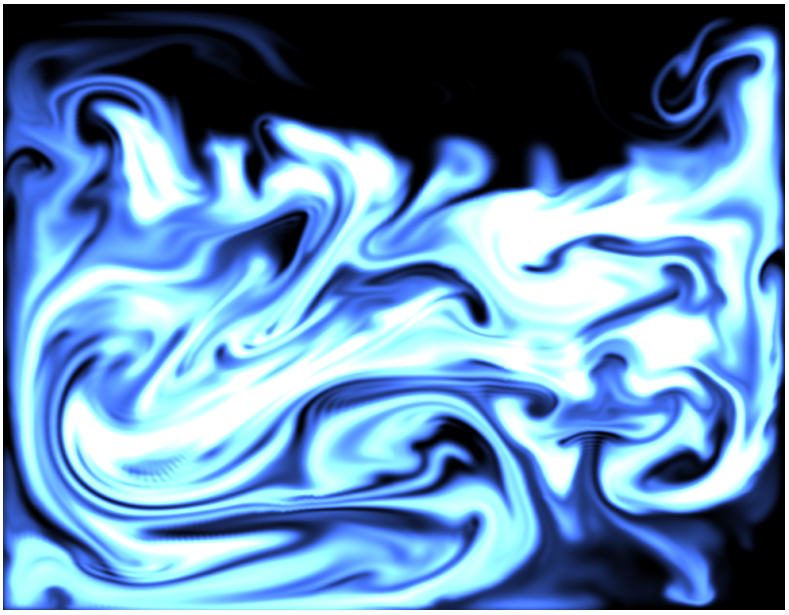

Figure 5.1: Smoke density over a 512 × 512 density and 256 × 256 fluid simulation grid.

Listing 2: Particle advection.

```
v = Field.SampleLevel(LinearSampler,PosToUV(Particle.Pos),0);
Particle.Pos     += dt*v.xyz;
```

5.2 Smoke-Fire Density Field

It is possible to simulate smoke and fire by iteratively solving the convection-diffusion equation for a scalar density field. Rendering such a scalar field in 2D is quite simple as only a texture holding the density field needs to be rendered. As for 3D fields, a volume rendering technique is required. Here are the governing equations for both smoke and fire respectively:

$$\frac{\partial \phi}{\partial t} = \mathbf{u} \cdot \nabla \phi + \mathbf{k} \nabla^{\mathbf{2}} \phi, \qquad \text{(smoke eq.)}$$

$$\frac{\partial \phi}{\partial t} = \mathbf{u} \cdot \nabla \phi + \mathbf{k} \nabla^{\mathbf{2}} \phi - \mathbf{c}, \qquad \text{(fire eq.)}$$

where ϕ is a scalar density, k a diffusion coefficient and c a reaction constant for fire.

Numerical schemes to solve this equation are abundant and the one which maps best to the GPU is the semi-lagrangian method (see equation Equation **??**) but unfortunately, the solution loses much detail as dissipation occurs from this numerical scheme - which in turn enables the omission of the diffusion term from the equation. To address this problem and achieve more compelling visual results,

Figure 5.2: Fire density over a 512 $\times$ 512 density and 256 $\times$ 256 fluid simulation grid.

we strongly suggest using the 3 pass MacCormack method described in [8]. This scheme has second order precision both in space and time therefore keeping the density from losing its small scale features and numerically dissipating its quantity to drastically as with the first order semi-lagrangian method. To add more detail to the simulation one could also amplify the vorticity of the flow (the velocity field) with vorticity confinement, a method discussed in the context of visual smoke simulation in [4].

6 Conclusion

Many algorithms could be generated from this one phase coupling of both equations through a density-invariant field. We hope the one illustrated here serves as a good basis for developers seeking to make use of interactive fluids in their applications.

References

[1] F. Colin, R. Egli, and F.Y. Lin. Computing a null divergence velocity field using smoothed particle hydrodynamics. *Journal of Computational Physics*, 217(2):680–692, 2006.

[2] K. Crane, I. Llamas, and S. Tariq. Real-time simulation and rendering of 3d fluids. *GPU Gems*, 3 (ch. 30), 2007.

[3] M. Desbrun and M.P. Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. *Computer Animation and Simulation (Proceedings of EG Workshop on Animation and Simulation)*, pages 61–76, 1996.

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

[5] W. Li, X. Wei, and A. Kaufman. Implementing lattice boltzmann computation on graphics hardware. *The Visual Computer*, 19(7-8):444–456, 2003.

[6] M. Muller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, 2003.

[7] S. Premože, T. Tasdizen, J. Bigler, A. Lefohn, and R.T. Whitaker. Particle-based simulation of fluids. *Computer Graphics Forum (Eurographics Proceedings)*, 22(3):401–410, 2003.

[8] A. Selle, R. Fedkiw, K. Byungmoon, L. Yingjie, and R. Jarek. An unconditionally stable maccormack method. *Journal of Scientific Computing*, 35(2-3):350–371, 2007.

[9] J. Stam. Stable fluids. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999.

[10] R. Xu, P. Stansby, and D. Laurence. Accuracy and stability in incompressible sph (isph) based on the projection method and a new approach. *Journal of Computational Physics*, 228(18):6703–6725, 2009.