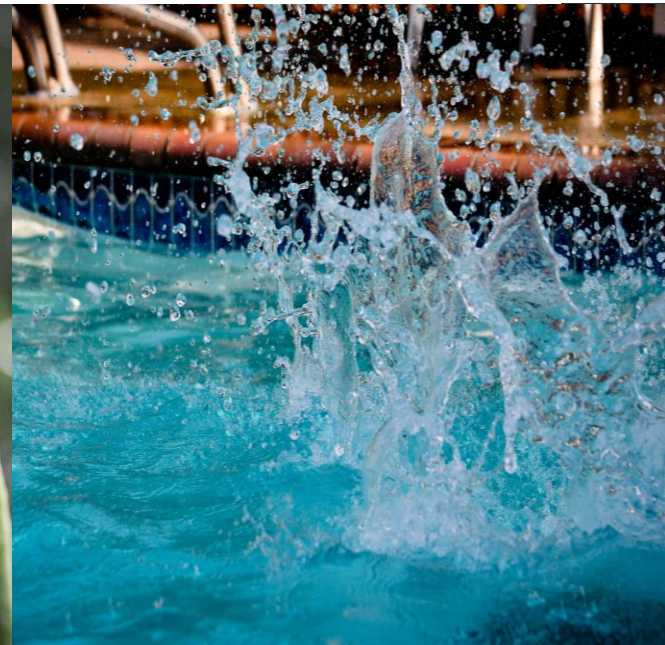
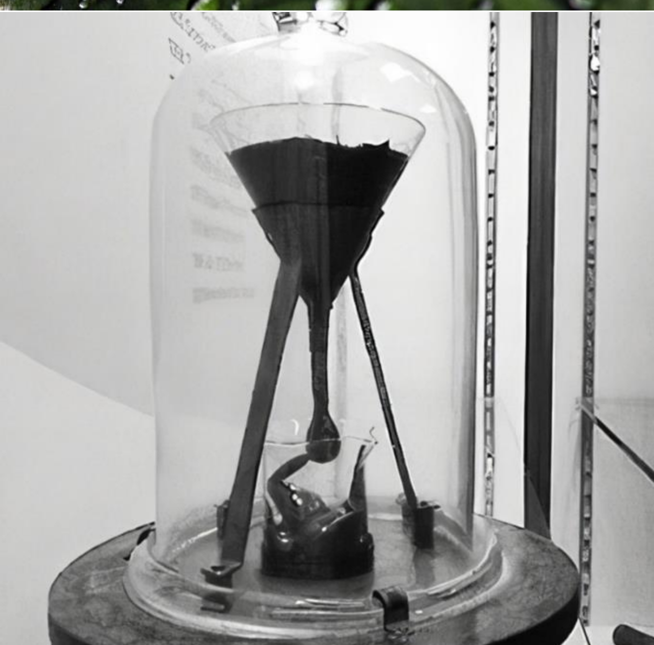
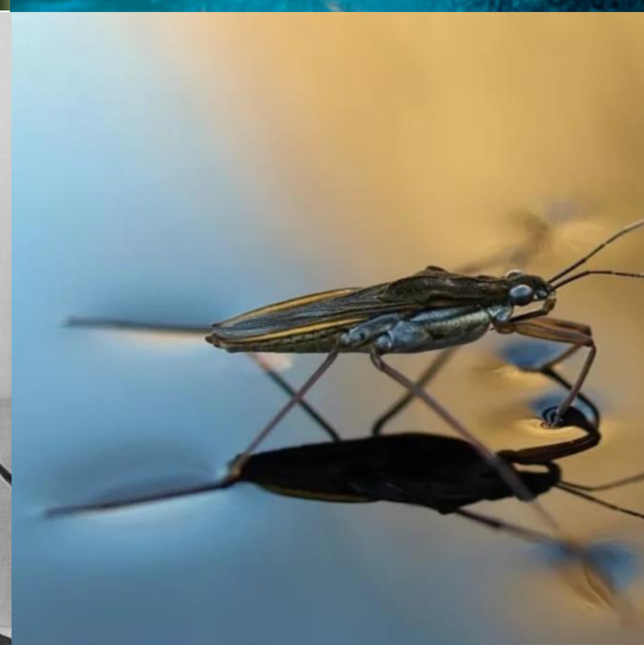
Fluid Simulation

Libin Liu

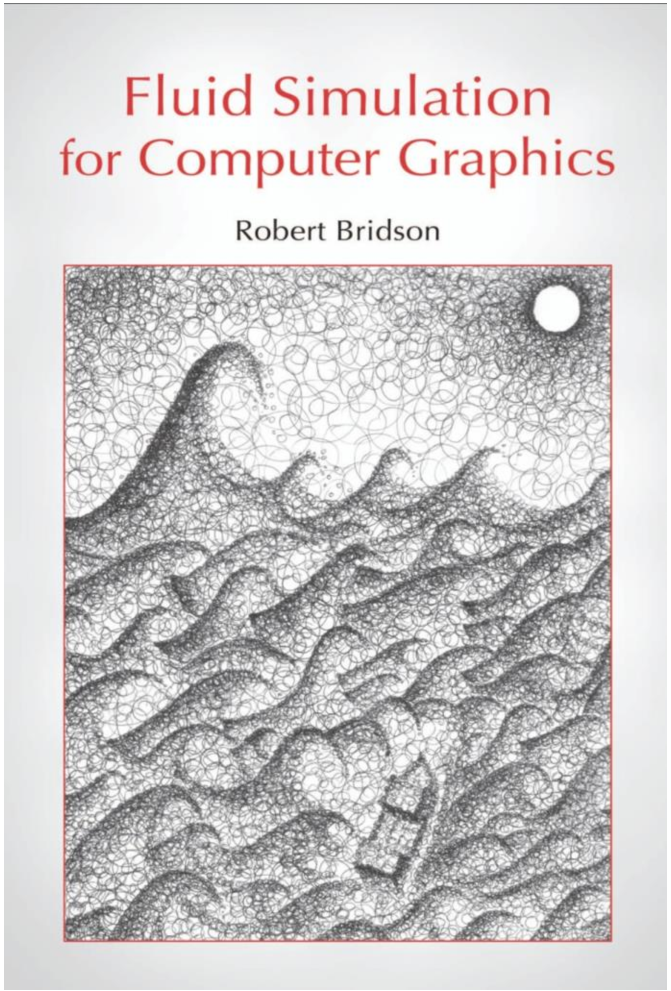
CFCS, Peking University


Outline

- Navier-Stokes Equations
- Lagrangian and Eulerian Viewpoints
- Numerical methods
- Eulerian methods
  - Smoke
  - Water
- Lagrangian methods
  - Smoothed Particle Hydrodynamics (SPH) method

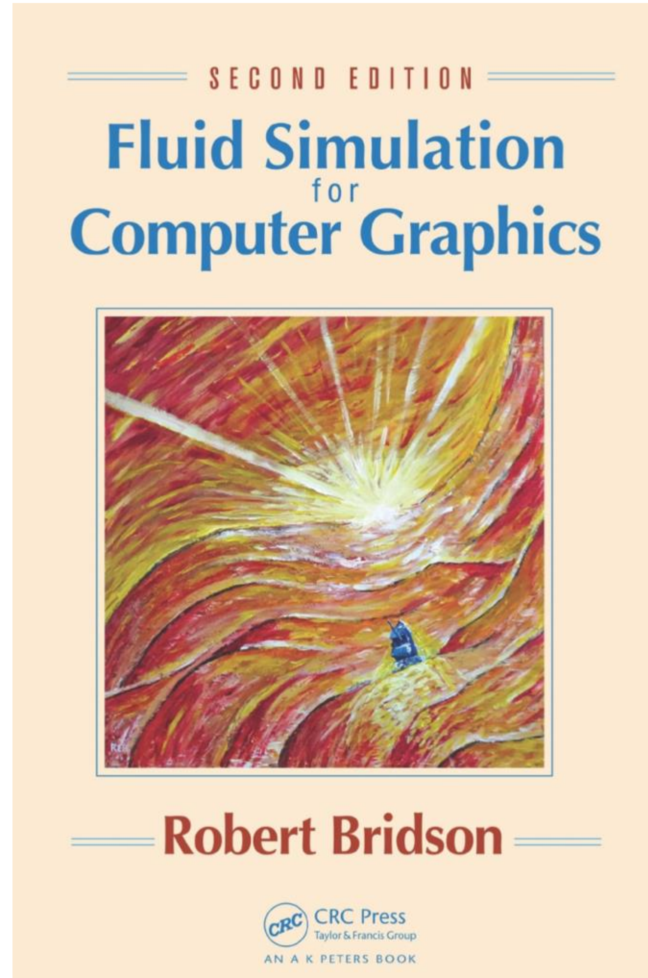


Oscars 2015 Tech: Robert Bridson

Several slides are modified from GAMES-103

Outline

- Navier-Stokes Equations
- Lagrangian and Eulerian Viewpoints
- Numerical methods
- Eulerian methods
  - Smoke
  - Water
- Lagrangian methods
  - Smoothed Particle Hydrodynamics (SPH) method



Oscars 2015 Tech: Robert Bridson

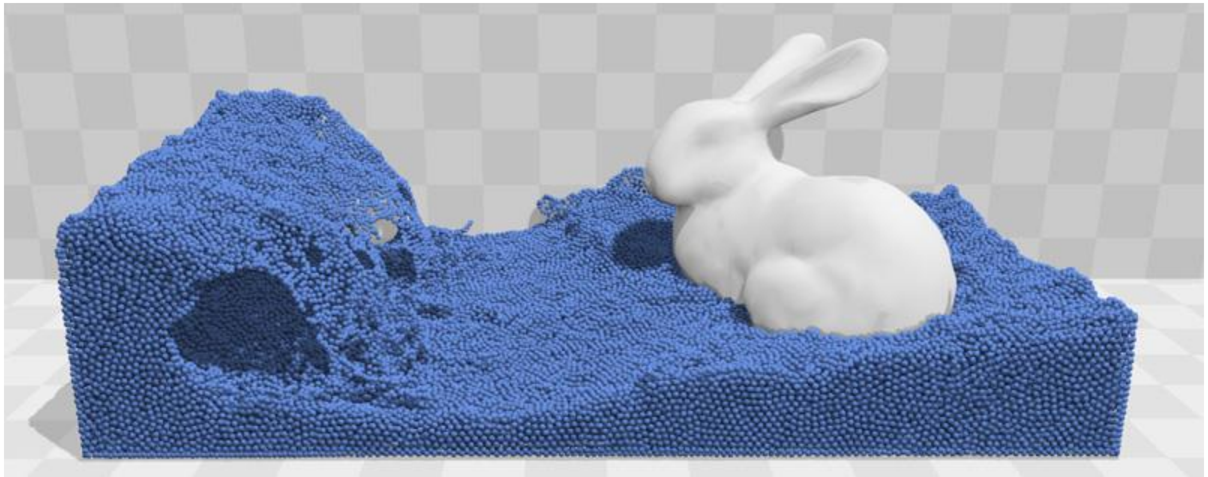
Several slides are modified from GAMES-103

Fluid Simulation

- Simple way: treat as a particle system just like clothes…
  - Particles interact like springs

$$ma = F = k(l - l_0)$$

  - Can we do better?

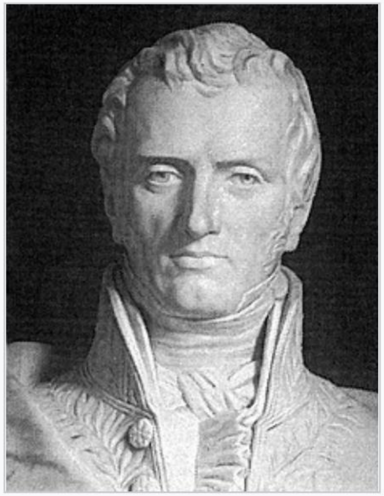


Navier-Stokes Equations

Incompressible, Viscous Navier-Stokes Equations

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho} \nabla p + \nu \Delta u$$

$$\nabla \cdot u = 0$$



Claude-Louis Navier

George Gabriel Stokes

Navier-Stokes Equations

Incompressible, Viscous Navier-Stokes Equations

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho} \nabla p + \nu \Delta u$$

$$\nabla \cdot u = 0$$

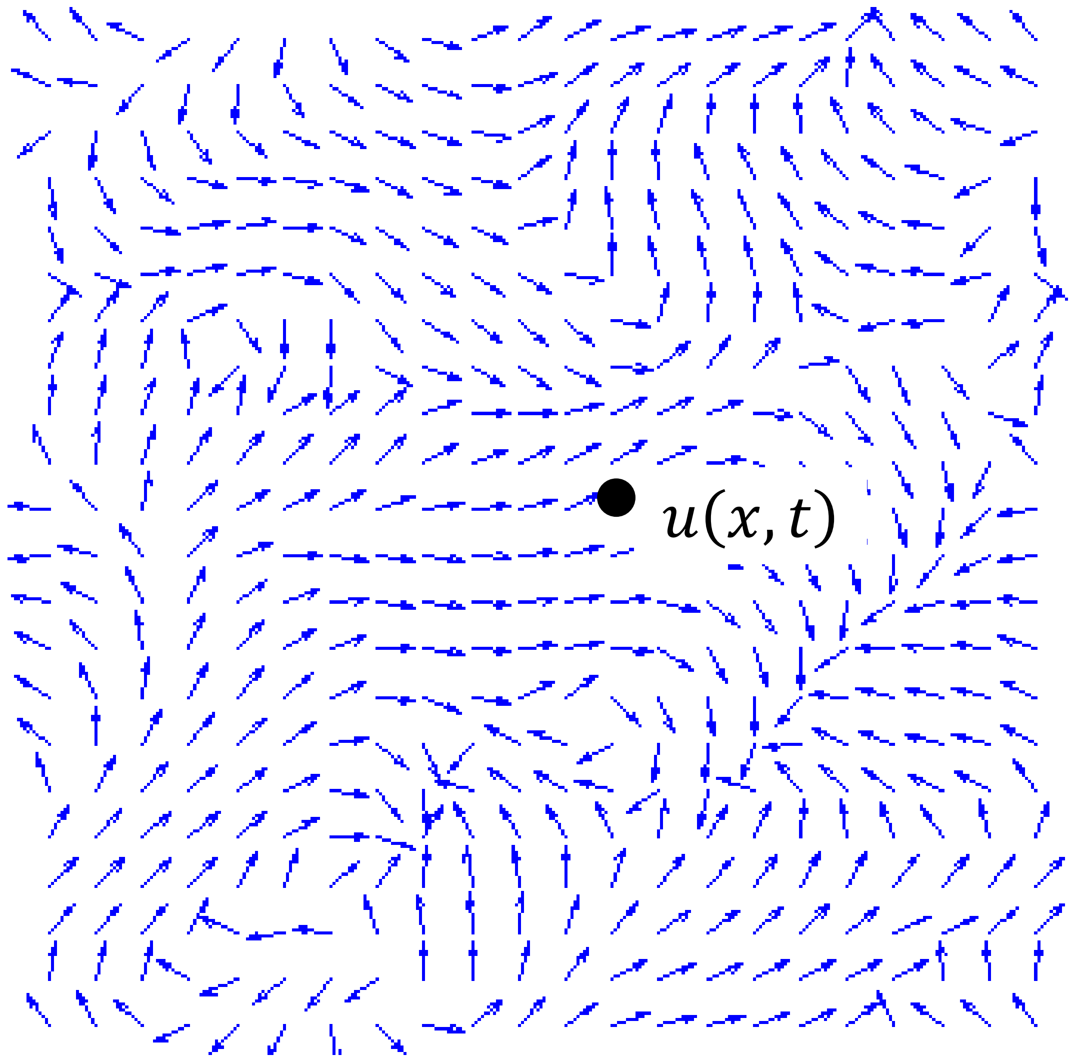
u: velocity. Why not v?

g: external force, e.g. gravity

p: pressure

ρ: density

Velocity Field



The Momentum Equation

Incompressible, Viscous Navier-Stokes Equations

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho}\nabla p + \nu \Delta u$$

$$\Updownarrow$$

$$\frac{\partial u(x,t)}{\partial t} + u(x,t) \cdot \nabla u(x,t) =$$

$$g(x,t) - \frac{1}{\rho(x,t)}\nabla p(x,t) + \nu \Delta u(x,t)$$

The Momentum Equation

Incompressible, Viscous Navier-Stokes Equations

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho}\nabla p + \nu \Delta u$$ 流体本身的粘滞性

$$Ma = Mg - \nabla \frac{kx^2}{2} - k_d v$$

mass-spring system

The Momentum Equation

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho}\nabla p + \nu \Delta u$$

$$u = u(x,t) \quad \Rightarrow \quad \frac{du}{dt} = \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x}\underbrace{\frac{dx}{dt}}_{u}$$

$$\Downarrow$$

$$a = \frac{du}{dt} = \frac{\partial u}{\partial t} + u \cdot \nabla u$$

若u为向量则不好懂

$u = \begin{bmatrix} u \\ v \end{bmatrix}$

$\nabla u = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix}$

The Momentum Equation

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho}\nabla p + \nu \Delta u$$

$$\Downarrow$$

Material Derivative:

$$\frac{D}{Dt}\varphi = \frac{\partial}{\partial t}\varphi + u \cdot \nabla \varphi$$

$$\frac{Du}{Dt} = g - \frac{1}{\rho}\nabla p + \nu \Delta u$$

$$\boxed{\frac{d\dot{x}}{dt} = a}$$

External Acceleration / Body Forces

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = \boldsymbol{g} - \frac{1}{\rho} \nabla p + \nu \Delta u$$

Viscosity

ν: kinematic viscosity

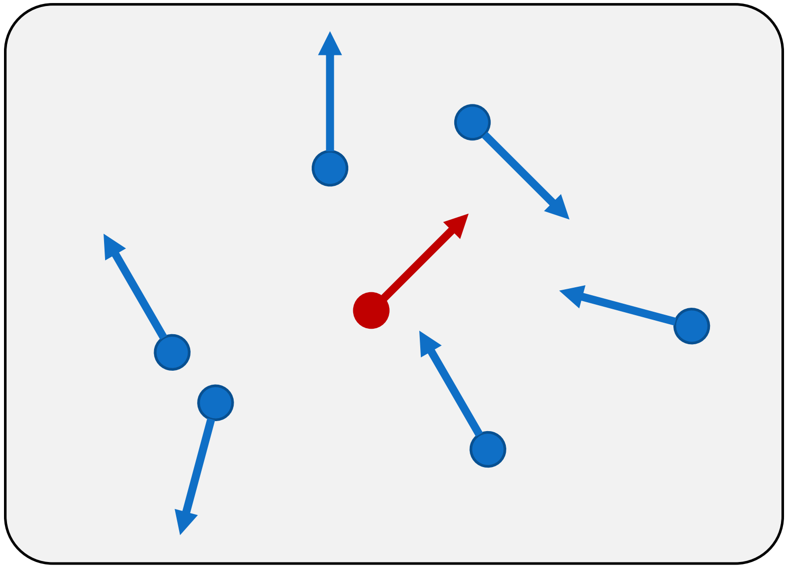
$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho} \nabla p + \nu \boldsymbol{\Delta u}$$

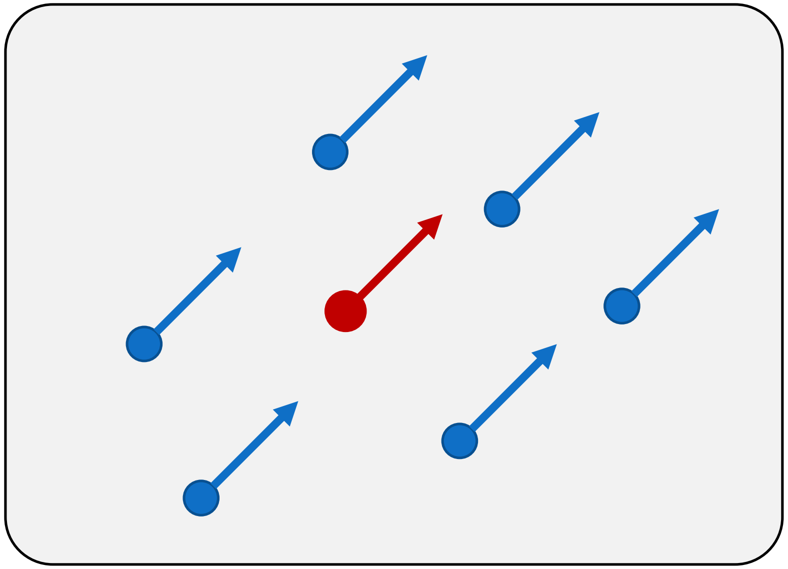
Viscosity

ν: kinematic viscosity

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho}\nabla p + \nu \Delta \boldsymbol{u}$$

Laplacian operator $\Delta u = \nabla \cdot \nabla u$

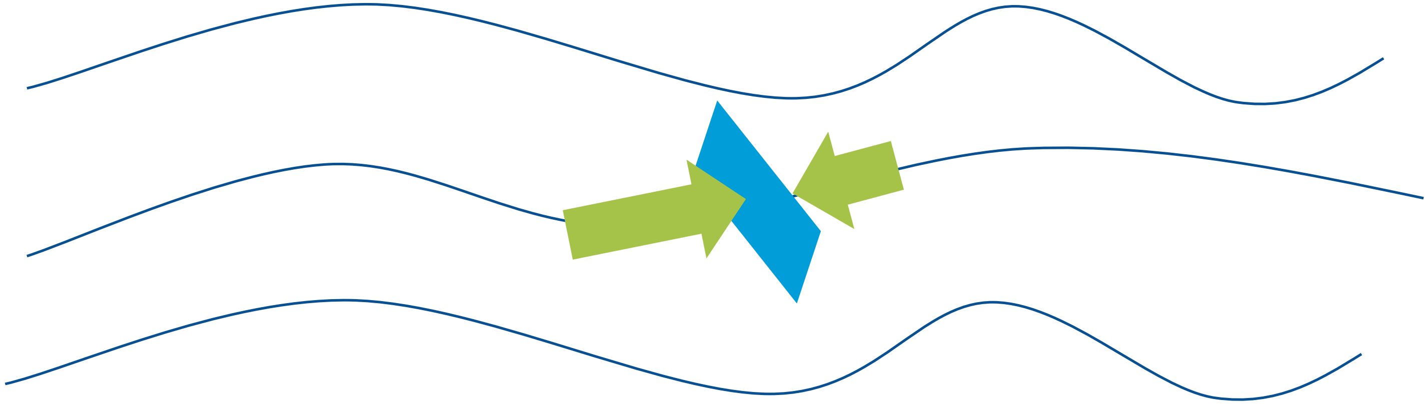




Internal Pressure

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho}\nabla \boldsymbol{p} + \nu \Delta u$$

$$\frac{P_2 - P_1}{\rho \Delta . L}$$



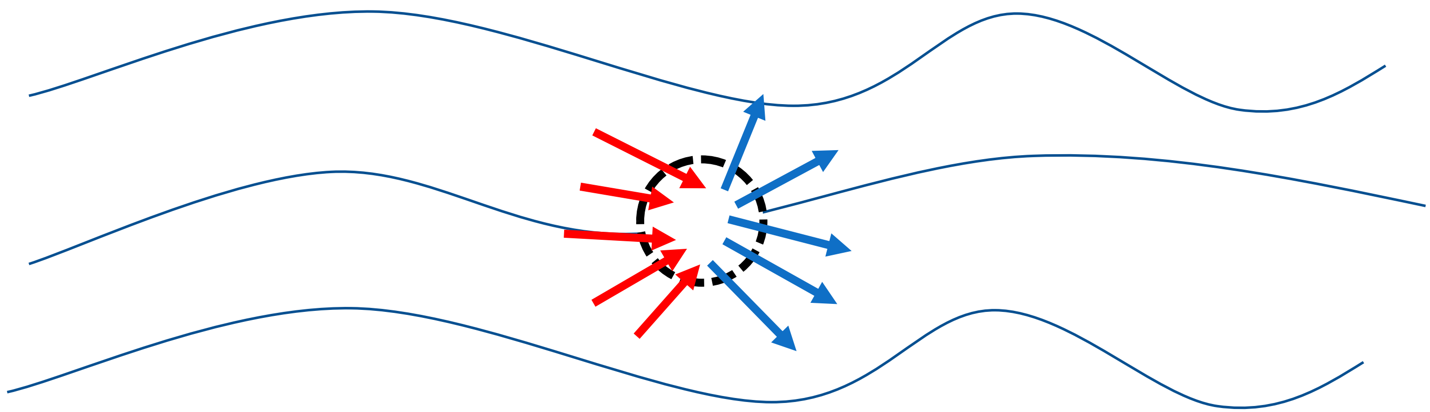
Incompressibility

$$\nabla \cdot u = 0$$

Also called divergence-free condition

流入与流出液体一样

$$\int_{\partial V} u \cdot n \, dS = \int_V \nabla \cdot u \; dV$$



Fluids are nearly incompressible

Compressible fluids are much more difficult to simulate

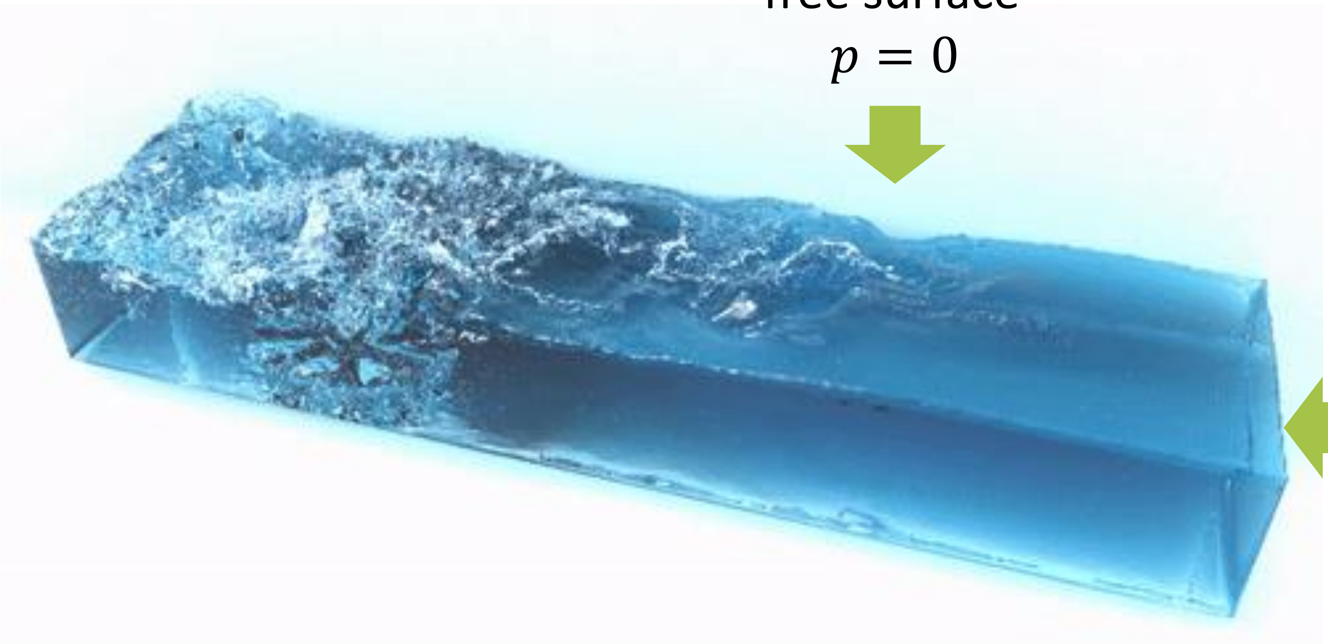
Internal Pressure & Incompressibility

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho} \nabla \boldsymbol{p} + \nu \Delta u$$

$$\nabla \cdot u = 0$$

Boundary Condition

free surface
$p = 0$



solid wall
$u \cdot n = u_{wall} \cdot n$

https://github.com/InteractiveComputerGraphics/SPlisHSPlasH

Navier-Stokes Equations

Incompressible, Viscous Navier-Stokes Equations

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho} \nabla p + \nu \Delta u$$

$$\nabla \cdot u = 0$$

u: velocity. Why not v?

g: external force, e.g. gravity

p: pressure

ρ: density

Navier-Stokes Equations

Incompressible, Viscous Navier-Stokes Equations

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho}\nabla p + \cancel{\nu \Delta u}$$

Inviscid fluid

$$\nabla \cdot u = 0$$

u: velocity. Why not v?

g: external force, e.g. gravity

p: pressure

ρ: density

Navier-Stokes Equations

Euler Equations: Inviscid fluid

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho} \nabla p$$

$$\nabla \cdot u = 0$$

u: velocity. Why not v?

g: external force, e.g. gravity

p: pressure

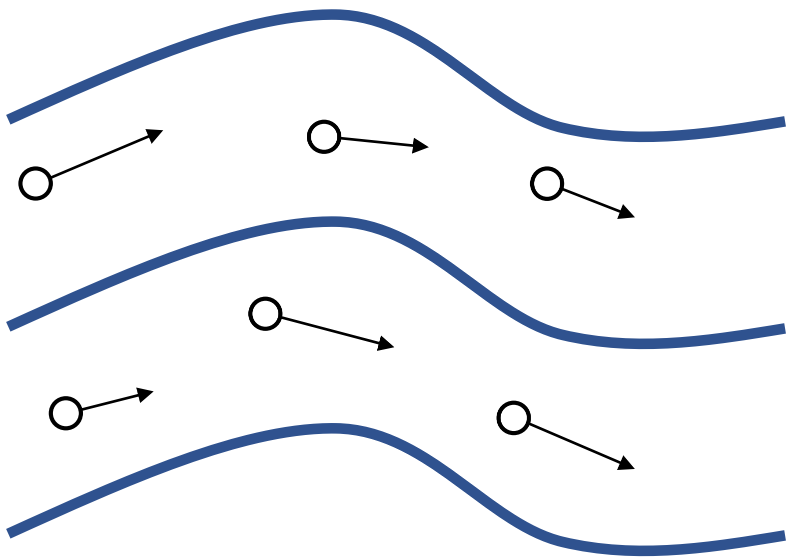
ρ: density

Lagrangian and Eulerian Viewpoints

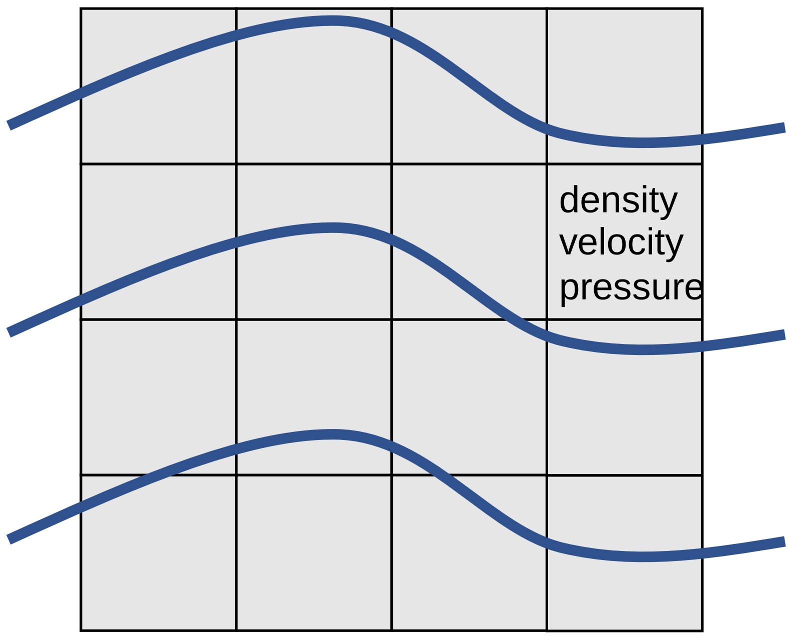
Lagrangian Viewpoint

Eulerian Viewpoint

Lagrangian and Eulerian Viewpoints



Lagrangian Approach
(dynamic particles or mesh)
Node movement carries physical quantities
(mass, velocity, …).



Eulerian Approach
(static grid or mesh)
Grid/Mesh doesn’t move.
Stored physical quantities change.

Advection

A generic quantity q of the fluid

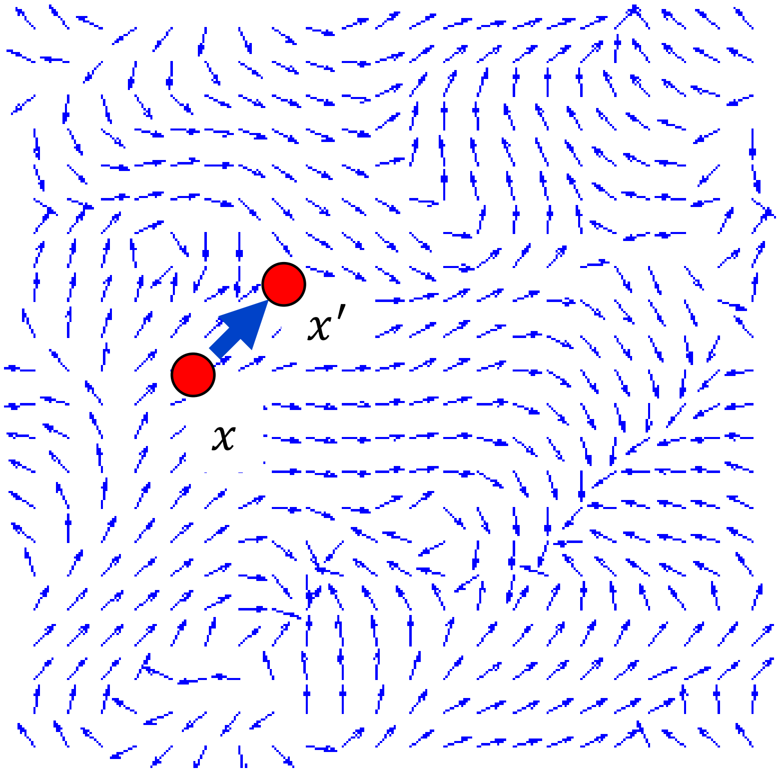
$$q = q(x, t)$$

is advected by the velocity field of the fluid

$$\frac{dq}{dt} = \frac{\partial q}{\partial t} + \nabla q \cdot \frac{dx}{dt}$$

$$= \frac{\partial q}{\partial t} + \nabla q \cdot u$$

$$\equiv \frac{Dq}{Dt}$$



Material Derivative:

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + u \cdot \nabla$$

Advection

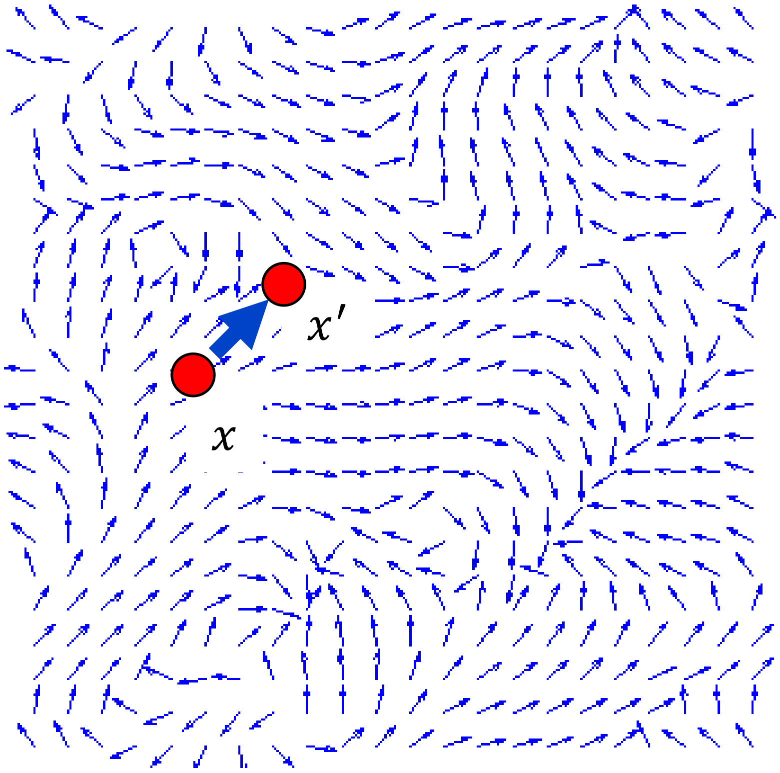
A generic quantity q of the fluid

$$q = q(x, t)$$

is advected by the velocity field of the fluid

$$\frac{Dq}{Dt} = 0$$

$$\frac{\partial q}{\partial t} + \nabla q \cdot u = 0$$



Note: we assume $\nabla \cdot u = 0$

Advection

A generic quantity q of the fluid

$$q = q(x, t)$$

is advected by the velocity field of the fluid

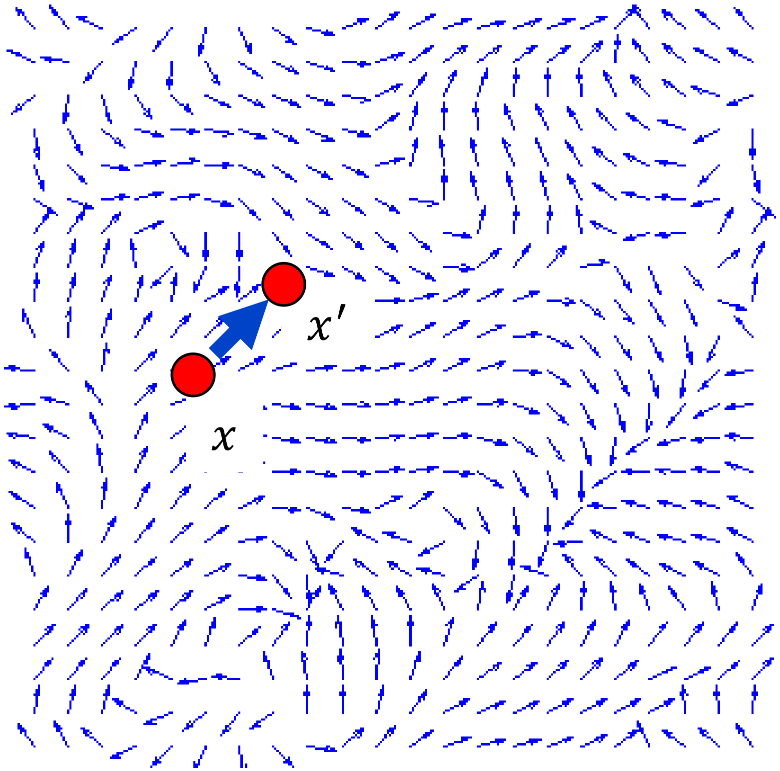
随着粒子移动的物理量变化

Lagrangian viewpoint $$\frac{Dq}{Dt} = 0$$

粒子原来位置的物理量变化情况

Eulerian viewpoint $$\frac{\partial q}{\partial t} = -\nabla q \cdot u$$

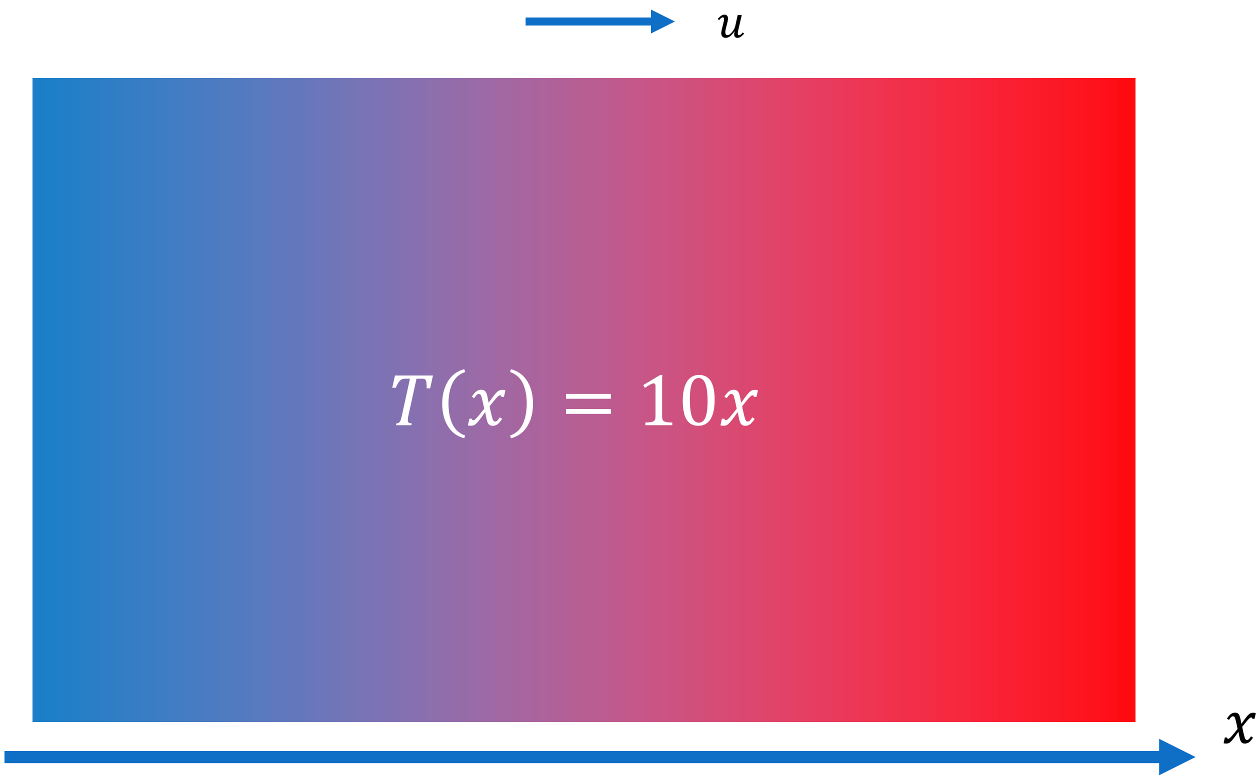
Note: we assume $\nabla \cdot u = 0$



Advection Example

Advection

$$\frac{DT}{Dt} = \frac{\partial T}{\partial t} + u \cdot \nabla T = 0$$



Numerical Simulation

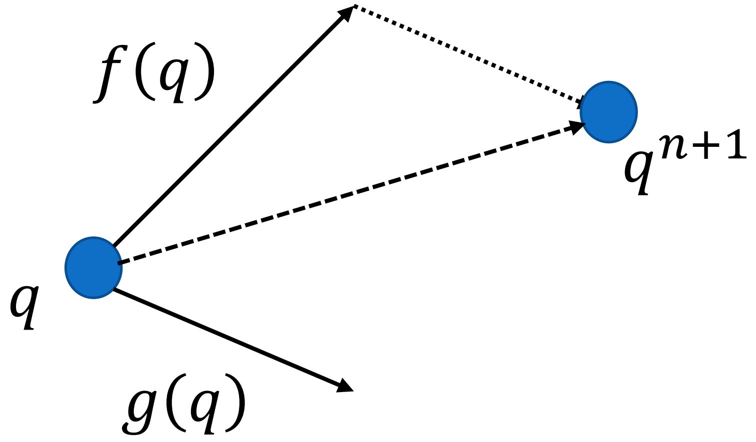
Numerical Integration

$$\frac{dq}{dt} = f(q) + g(q)$$

q 为任意物理量

Forward Euler Integration:

$$q^{n+1} = q^n + \delta t\big(f(q^n) + g(q^n)\big)$$



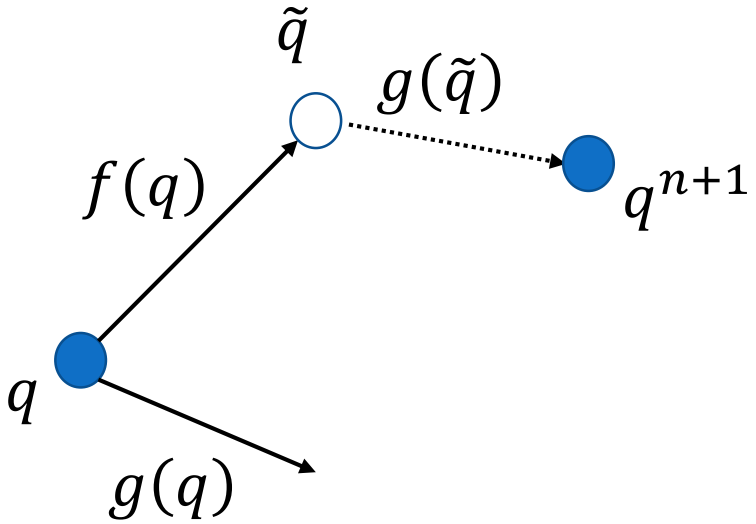
Splitting in Numerical Simulation

$$\frac{dq}{dt} = f(q) + g(q)$$

Splitting with forward Euler integration:

$$\tilde{q} = q^n + \delta t f(q^n)$$

$$q^{n+1} = \tilde{q} + \delta t g(\tilde{q})$$



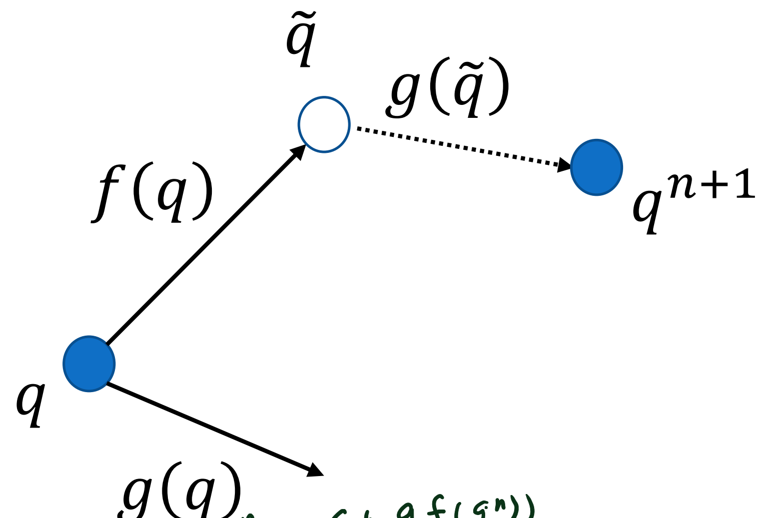
Splitting in Numerical Simulation

$$\frac{dq}{dt} = f(q) + g(q)$$

Splitting with forward Euler integration:

$$\tilde{q} = q^n + \delta t f(q^n)$$

$$q^{n+1} = \tilde{q} + \delta t g(\tilde{q})$$



Still first-order accurate

$\delta t g(q^n + \delta t f(q^n))$

假设f, g足够光滑.

$\delta t \, (g(q^n) + \delta t \, g f(q^n))$

$= \delta t \, g(q^n) + \delta^2 t^2 \underbrace{gf(q^n)}_{\text{有界}} \Rightarrow G(\delta^2 t^2)$

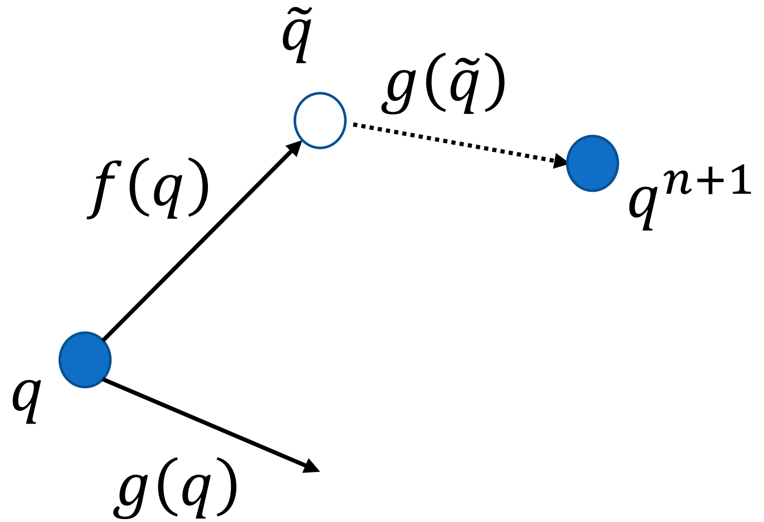
Splitting in Numerical Simulation

$$\frac{dq}{dt} = f(q) + g(q)$$

Splitting integration

$$\frac{dq}{dt} = f(q)$$

$$\frac{d\tilde{q}}{dt} = g(\tilde{q})$$



> Solve simpler equations
> Using the best integrator for each equation

Splitting the Fluid Equations

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u - \frac{1}{\rho}\nabla p + g + \nu \Delta u$$

$$\nabla \cdot u = 0$$

body force $\quad \dfrac{\partial u}{\partial t} = g$

advection $\quad \dfrac{\partial u}{\partial t} = -u \cdot \nabla u$

diffusion $\quad \dfrac{\partial u}{\partial t} = \nu \Delta u$

projection $\quad \dfrac{\partial u}{\partial t} = -\dfrac{1}{\rho}\nabla p$

s.t. $\nabla \cdot u = 0$

Stable Fluids

Jos Stam*

Alias | wavefront

Abstract

Building animation tools for fluid-like motions is an important and challenging problem with many applications in computer graphics. The use of physics-based models for fluid flow can greatly assist in creating such tools. Physical models, unlike key frame or procedural based techniques, permit an animator to almost effortlessly create interesting, swirling fluid-like behaviors. Also, the interaction of flows with objects and virtual forces is handled elegantly. Until recently, it was believed that physical fluid models were too expensive to allow real-time interaction. This was largely due to the fact that previous models used unstable schemes to solve the physical equations governing a fluid. In this paper, for the first time, we propose an unconditionally stable model which still produces complex fluid-like flows. As well, our method is very easy to implement. The stability of our model allows us to take larger time steps and therefore achieve faster simulations. We have used our model in conjuction with advecting solid textures to create many fluid-like animations interactively in two- and three-dimensions.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: animation of fluids, Navier-Stokes, stable solvers, implicit elliptic PDE solvers, interactive modeling, gaseous phenomena, advected textures

1 Introduction

One of the most intriguing problems in computer graphics is the simulation of fluid-like behavior. A good fluid solver is of great importance in many different areas. In the special effects industry there is a high demand to convincingly mimic the appearance and behavior of fluids such as smoke, water and fire. Paint programs can also benefit from fluid solvers to emulate traditional techniques such as watercolor and oil paint. Texture synthesis is another possible application. Indeed, many textures result from fluid-like processes, such as erosion. The modeling and simulation of fluids is, of course, also of prime importance in most scientific disciplines and in engineering. Fluid mechanics is used as the standard mathematical framework on which these simulations are based. There is a consensus among scientists that the *Navier-Stokes* equations are a very good model for fluid flow. Thousands of books and articles have been published in various areas on how to compute these equations numerically. Which solver to use in practice depends largely on the problem at hand and on the computing power available. Most engineering tasks require that the simulation provide accurate bounds on the physical quantities involved to answer questions related to safety, performance, etc. The visual appearance (shape) of the flow is of secondary importance in these applications. In computer graphics, on the other hand, the shape and the behavior of the fluid are of primary interest, while physical accuracy is secondary or in some cases irrelevant. Fluid solvers, for computer graphics, should ideally provide a user with a tool that enables her to achieve fluid-like effects in real-time. These factors are more important than strict physical accuracy, which would require too much computational power.

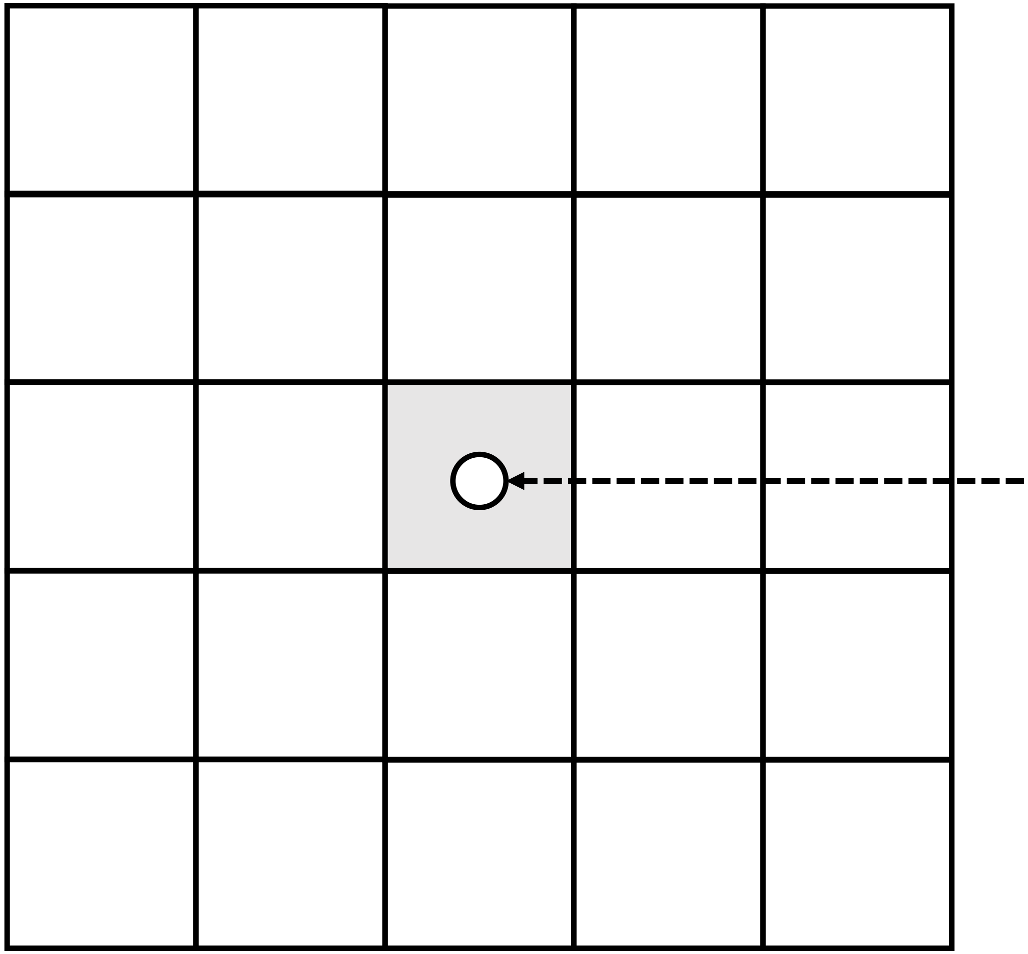
In fact, most previous models in computer graphics were driven by visual appearance and not by physical accuracy. Early flow models were built from simple primitives. Various combinations of these primitives allowed the animation of particles systems [15, 17] or simple geometries such as leaves [23]. The complexity of the flows was greatly improved with the introduction of random turbulences [16, 20]. These turbulences are mass conserving and, therefore, automatically exhibit rotational motion. Also the turbulence is periodic in space and time, which is ideal for motion "texture mapping" [19]. Flows built up from a superposition of flow primitives all have the disadvantage that they do not respond dynamically to user-applied external forces. Dynamical models of fluids based on the Navier-Stokes equations were first implemented in two-dimensions. Both Yaeger and Upson and Gamito et al. used a vortex method coupled with a Poisson solver to create two-dimensional animations of fluids [24, 8]. Later, Chen et al. animated water surfaces from the pressure term given by a two-dimensional simulation of the Navier-Stokes equations [2]. Their method unlike ours is both limited to two-dimensions and is unstable. Kass and Miller linearize the shallow water equations to simulate liquids [12]. The simplifications do not, however, capture the interesting rotational motions characteristic of fluids. More recently, Foster and Metaxas clearly show the advantages of using the full three-dimensional Navier-Stokes equations in creating fluid-like animations [7]. Many effects which are hard to key frame manually such as swirling motion and flows past objects are obtained automatically. Their algorithm is based mainly on the work of Harlow and Welch in computational fluid dynamics, which dates back to 1965 [11]. Since then many other techniques which Foster and Metaxas could have used have been developed. However, their model has the advantage of being simple to code, since it is based on a finite differencing of the Navier-Stokes equations and an explicit time solver. Similar solvers and their source code are also available from the book of Griebel et al. [9]. The main problem with explicit solvers is that the numerical scheme can become unstable for large time-steps. Instability leads to numerical simulations that "blow-up" and therefore have to be restarted with a smaller time-step. The instability of these explicit algorithms sets serious limits on speed and interactivity. Ideally, a user should be able to interact in real-time with a fluid solver without having to worry about possible "blow ups".

In this paper, for the first time, we propose a stable algorithm that solves the full Navier-Stokes equations. Our algorithm is very

*Alias | wavefront, 1218 Third Ave, 8th Floor, Seattle, WA 98101, U.S.A. jstam@aw.sgi.com

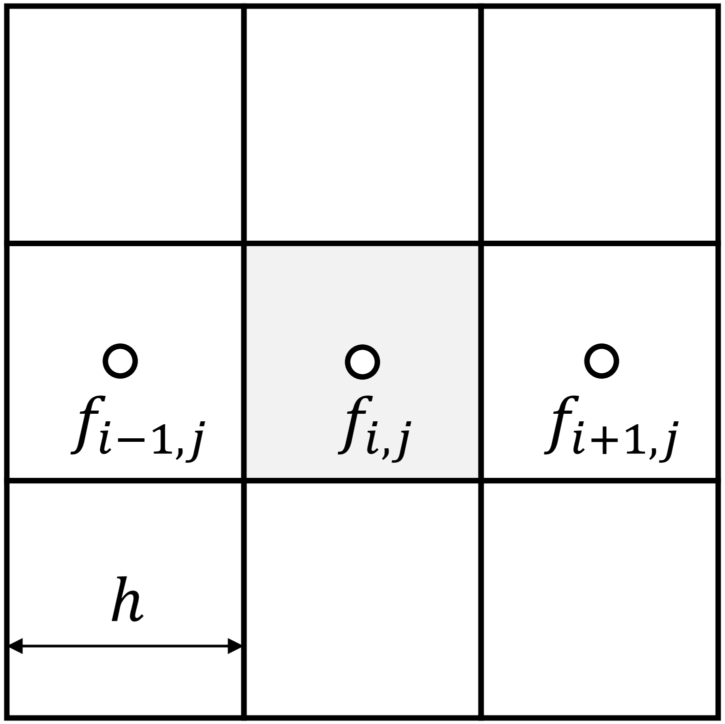
Jos Stam. 1999. Stable Fluids. TOG (SIGGRAPH).

Eulerian Grid Representation



- Scalars
  - Density/color
  - Pressure
  - Temperature
  - …
- Vectors
  - Velocities

Finite Differencing on Grid



Forward/backward differences:

Biased and accurate to $O(h)$

$$\frac{\partial f_{i,j}}{\partial x} = \frac{f_{i+1,j} - f_{i,j}}{h}$$

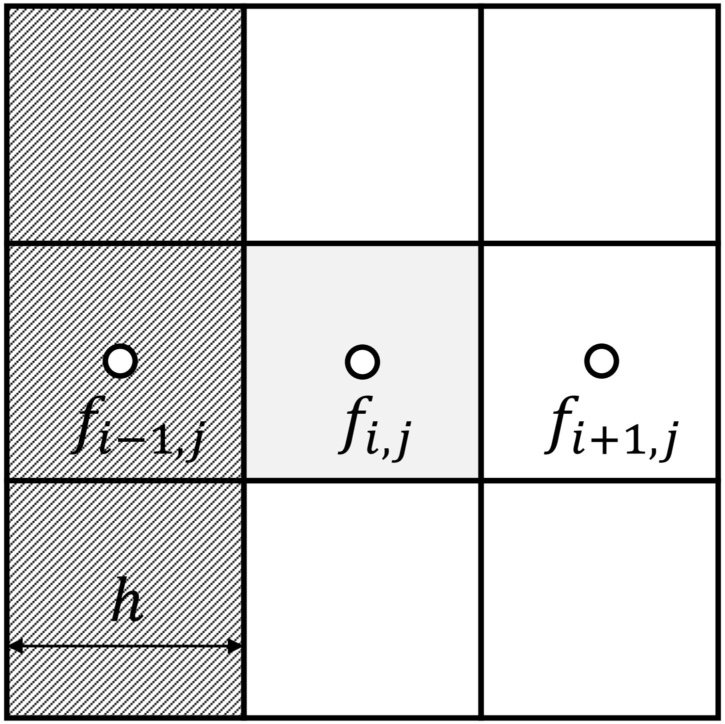
$$\frac{\partial f_{i,j}}{\partial x} = \frac{f_{i,j} - f_{i-1,j}}{h}$$

Center differences:

Unbiased and accurate to $O(h^2)$

$$\frac{\partial f_{i,j}}{\partial x} = \frac{f_{i+1,j} - f_{i-1,j}}{2h}$$

Boundary Condition



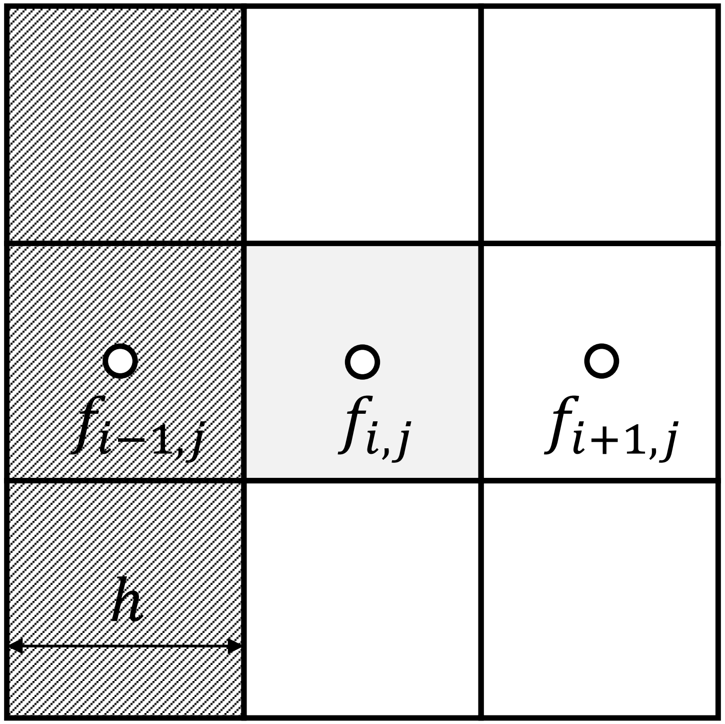
Center differences:

Unbiased and accurate to $O(h^2)$

$$\frac{\partial f_{i,j}}{\partial x} = \frac{f_{i+1,j} - \boldsymbol{f_{i-1,j}}}{2h}$$

How could we compute $f_{i-1,j}$ when it's outside?

Boundary Condition



Center differences:

Unbiased and accurate to $O(h^2)$

$$\frac{\partial f_{i,j}}{\partial x} = \frac{f_{i+1,j} - \boldsymbol{f_{i-1,j}}}{2h}$$

How could we compute $f_{i-1,j}$ when it's outside?

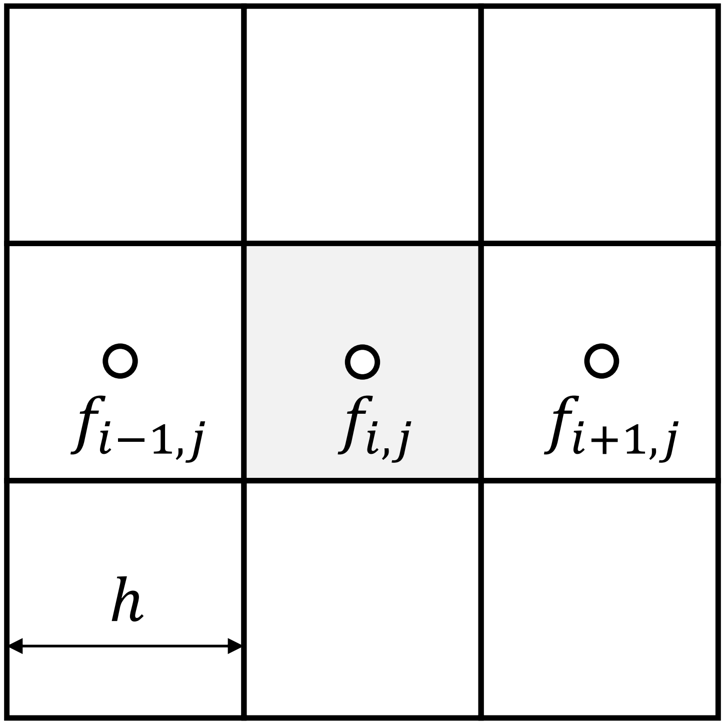
A Dirichlet boundary: $f_{i-1,j} = C$

$$\frac{\partial f_{i,j}}{\partial x} = \frac{f_{i+1,j} - \boldsymbol{C}}{2h}$$

A Neumann boundary: $f_{i-1,j} = f_{i,j}$

$$\frac{\partial f_{i,j}}{\partial x} = \frac{f_{i+1,j} - \boldsymbol{f_{i,j}}}{2h}$$

Problem with Central Differencing



Center differences:

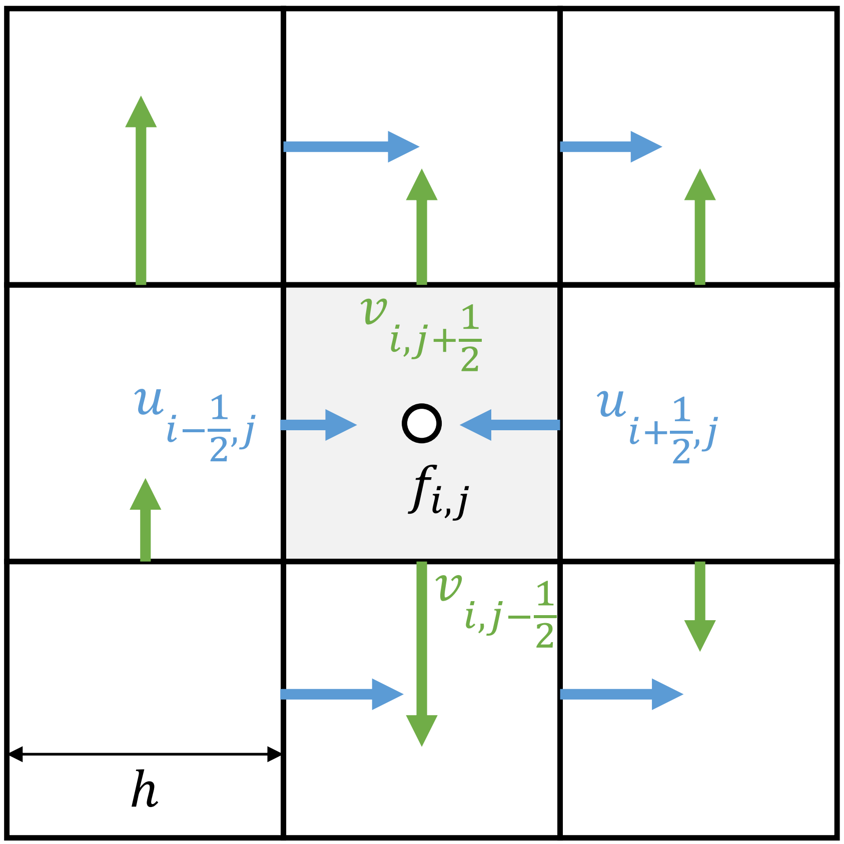
Unbiased and accurate to $O(h^2)$

$$\frac{\partial f_{i,j}}{\partial x} = \frac{f_{i+1,j} - f_{i-1,j}}{2h}$$

- $f_{i,j}$ is not considered when computing $\frac{\partial f_{i,j}}{\partial x}$
- Nontrivial null-space
  consider $f_{i,j} = (-1)^i$

Staggered Grid

Put some physical quantities on faces, especially velocities



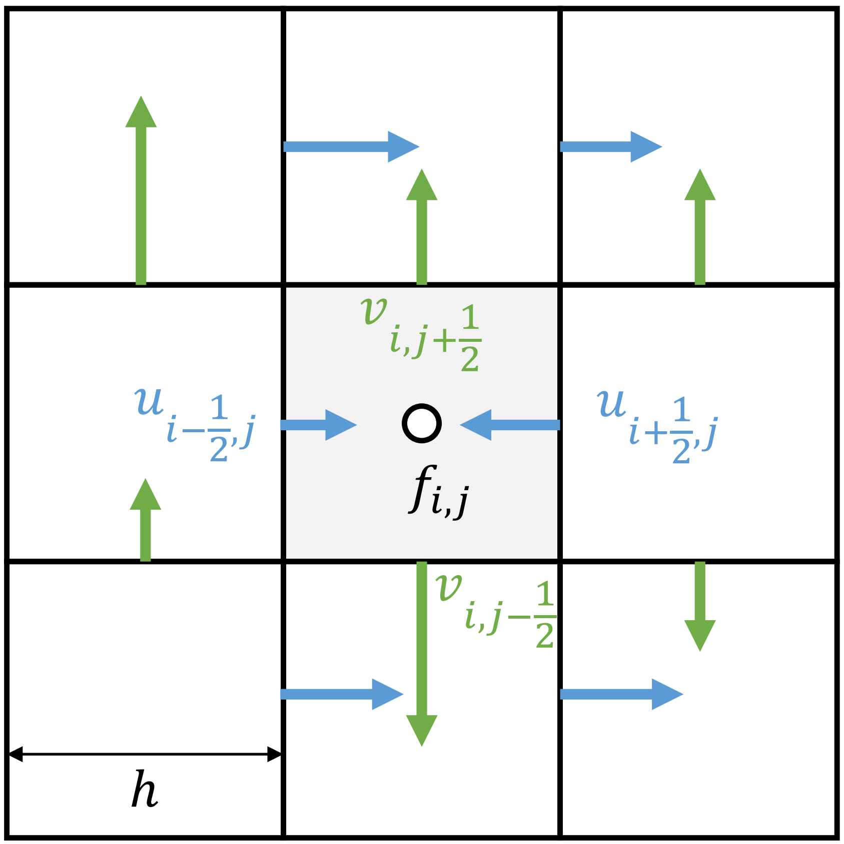
Marker-and-Cell (MAC) Grid

In 2D cases: $\boldsymbol{u} = (u, v)$

- The x-part of the velocity is defined on vertical faces.
- The y-part of the velocity is defined on horizonal faces.

Finite Differencing

Put some physical quantities on faces, especially velocities



Marker-and-Cell (MAC) Grid

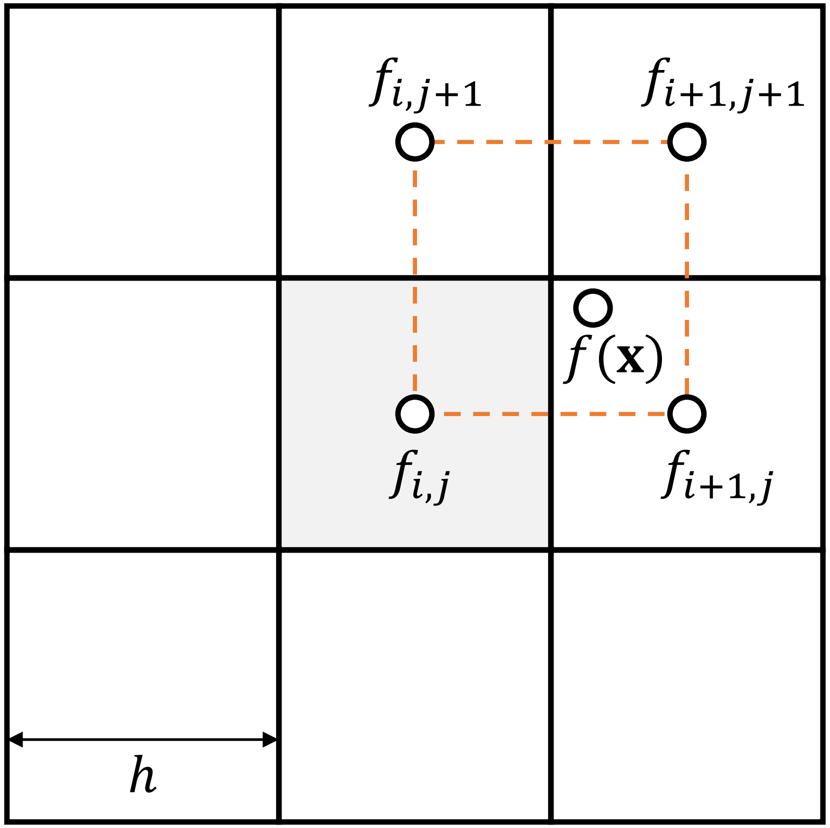
In 2D cases: $\boldsymbol{u} = (u, v)$

$$\nabla \cdot u_{i,j} = \frac{\partial u_{i,j}}{\partial x} + \frac{\partial u_{i,j}}{\partial y}$$

$$= \frac{u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j}}{h} + \frac{v_{i,j+\frac{1}{2}} - v_{i,j-\frac{1}{2}}}{h}$$

Bilinear Interpolation

Use bilinear/trilinear interpolation to interpolate physical quantities



Marker-and-Cell (MAC) Grid

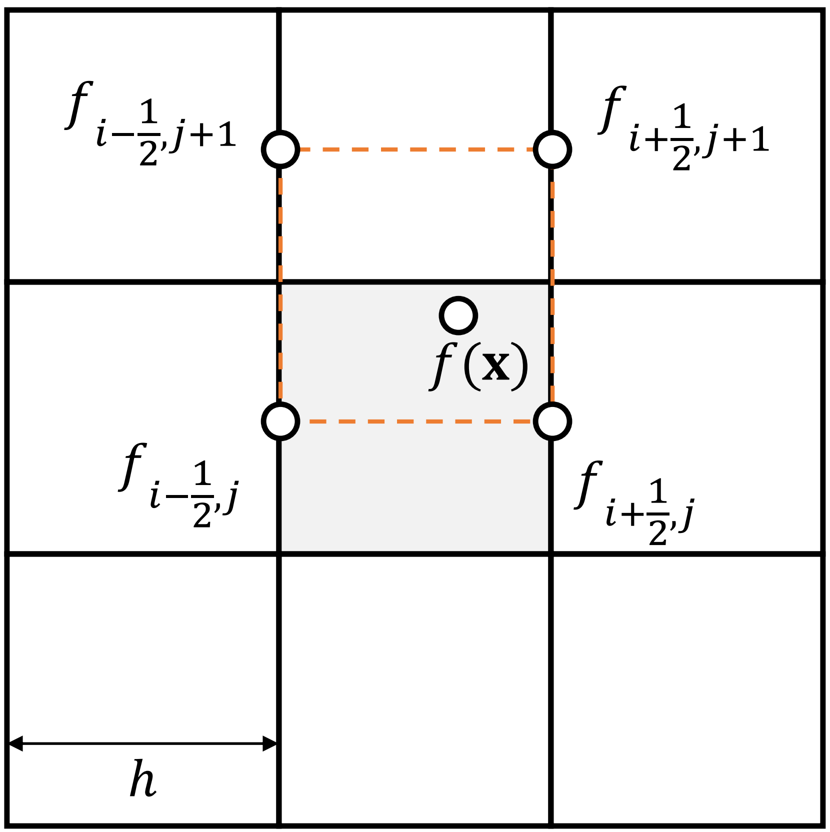
$$i \leftarrow \lfloor x \rfloor$$

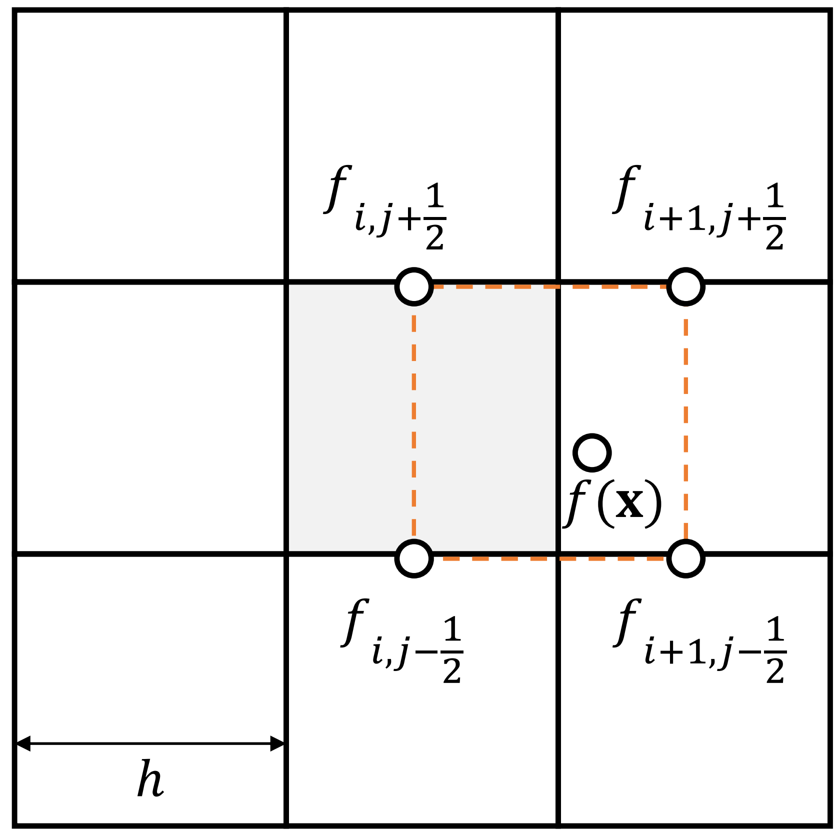
$$j \leftarrow \lfloor y \rfloor$$

$$\begin{aligned} f(\mathbf{x}) \leftarrow\ & (i+1-x)(j+1-y)f_{i,j} \\ & +(x-i)(j+1-y)f_{i+1,j} \\ & +(i+1-x)(y-j)f_{i,j+1} \\ & +(x-i)(y-j)f_{i+1,j+1} \end{aligned}$$

Bilinear Interpolation

Use bilinear/trilinear interpolation to interpolate physical quantities

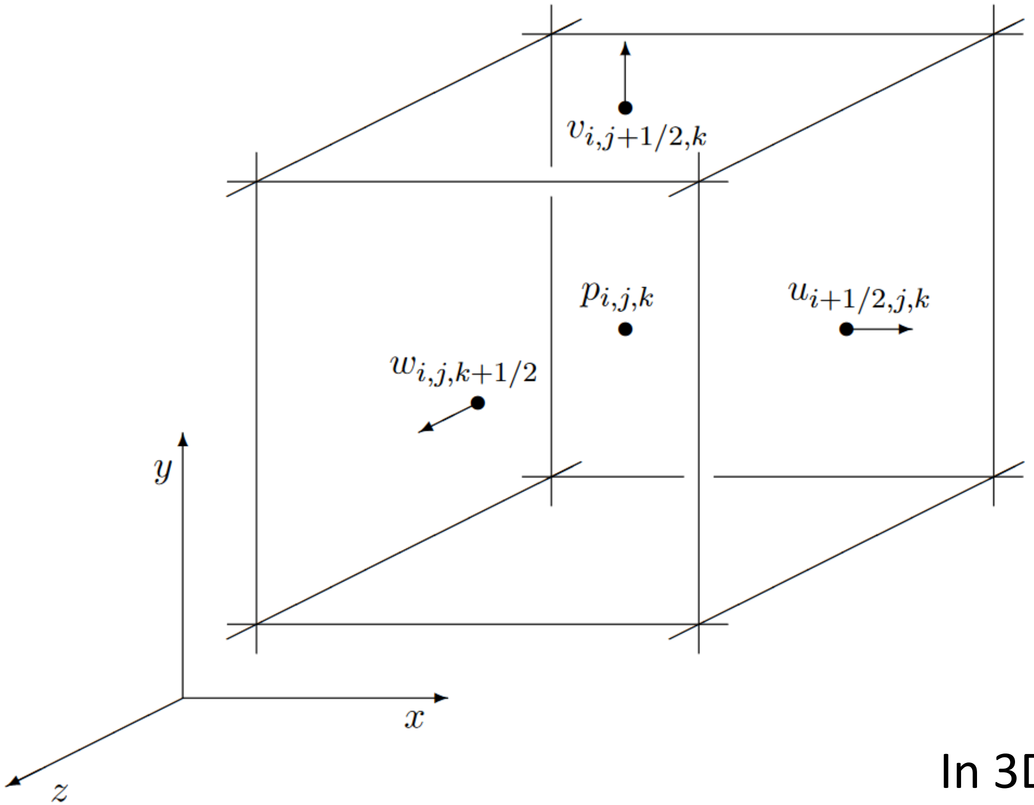




Marker-and-Cell (MAC) Grid

Staggered Grid

Put some physical quantities on faces, especially velocities



In 3D cases: $\boldsymbol{u} = (u, v, w)$

Marker-and-Cell (MAC) Grid

Splitting the Fluid Equations

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u - \frac{1}{\rho} \nabla p + g + \nu \Delta u$$

$$\nabla \cdot u = 0$$

body force $\quad \dfrac{\partial u}{\partial t} = g$

advection $\quad \dfrac{\partial u}{\partial t} = -u \cdot \nabla u$

diffusion $\quad \dfrac{\partial u}{\partial t} = \nu \Delta u$

projection $\quad \dfrac{\partial u}{\partial t} = -\dfrac{1}{\rho} \nabla p$

s.t. $\nabla \cdot u = 0$

Stable Fluids

Jos Stam*

Alias | wavefront

Abstract

Building animation tools for fluid-like motions is an important and challenging problem with many applications in computer graphics. The use of physics-based models for fluid flow can greatly assist in creating such tools. Physical models, unlike key frame or procedural based techniques, permit an animator to almost effortlessly create interesting, swirling fluid-like behaviors. Also, the interaction of flows with objects and virtual forces is handled elegantly. Until recently, it was believed that physical fluid models were too expensive to allow real-time interaction. This was largely due to the fact that previous models used unstable schemes to solve the physical equations governing a fluid. In this paper, for the first time, we propose an unconditionally stable model which still produces complex fluid-like flows. As well, our method is very easy to implement. The stability of our model allows us to take larger time steps and therefore achieve faster simulations. We have used our model in conjuction with advecting solid textures to create many fluid-like animations interactively in two- and three-dimensions.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: animation of fluids, Navier-Stokes, stable solvers, implicit elliptic PDE solvers, interactive modeling, gaseous phenomena, advected textures

1 Introduction

One of the most intriguing problems in computer graphics is the simulation of fluid-like behavior. A good fluid solver is of great importance in many different areas. In the special effects industry there is a high demand to convincingly mimic the appearance and behavior of fluids such as smoke, water and fire. Paint programs can also benefit from fluid solvers to emulate traditional techniques such as watercolor and oil paint. Texture synthesis is another possible application. Indeed, many textures result from fluid-like processes, such as erosion. The modeling and simulation of fluids is, of course, also of prime importance in most scientific disciplines and in engineering. Fluid mechanics is used as the standard mathematical framework on which these simulations are based. There is a consensus among scientists that the *Navier-Stokes* equations are a very good model for fluid flow. Thousands of books and articles have been published in various areas on how to compute these equations numerically. Which solver to use in practice depends largely on the problem at hand and on the computing power available. Most engineering tasks require that the simulation provide accurate bounds on the physical quantities involved to answer questions related to safety, performance, etc. The visual appearance (shape) of the flow is of secondary importance in these applications. In computer graphics, on the other hand, the shape and the behavior of the fluid are of primary interest, while physical accuracy is secondary or in some cases irrelevant. Fluid solvers, for computer graphics, should ideally provide a user with a tool that enables her to achieve fluid-like effects in real-time. These factors are more important than strict physical accuracy, which would require too much computational power.

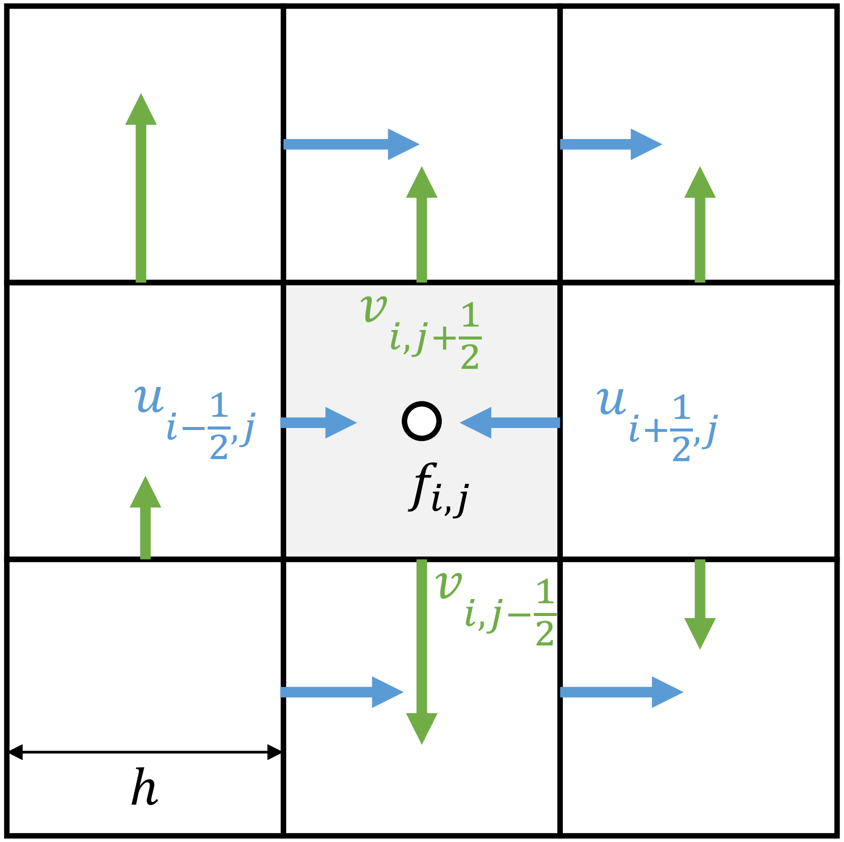
In fact, most previous models in computer graphics were driven by visual appearance and not by physical accuracy. Early flow models were built from simple primitives. Various combinations of these primitives allowed the animation of particles systems [15, 17] or simple geometries such as leaves [23]. The complexity of the flows was greatly improved with the introduction of random turbulences [16, 20]. These turbulences are mass conserving and, therefore, automatically exhibit rotational motion. Also the turbulence is periodic in space and time, which is ideal for motion "texture mapping" [19]. Flows built up from a superposition of flow primitives all have the disadvantage that they do not respond dynamically to user-applied external forces. Dynamical models of fluids based on the Navier-Stokes equations were first implemented in two-dimensions. Both Yaeger and Upson and Gamito et al. used a vortex method coupled with a Poisson solver to create two-dimensional animations of fluids [24, 8]. Later, Chen et al. animated water surfaces from the pressure term given by a two-dimensional simulation of the Navier-Stokes equations [2]. Their method unlike ours is both limited to two-dimensions and is unstable. Kass and Miller linearize the shallow water equations to simulate liquids [12]. The simplifications do not, however, capture the interesting rotational motions characteristic of fluids. More recently, Foster and Metaxas clearly show the advantages of using the full three-dimensional Navier-Stokes equations in creating fluid-like animations [7]. Many effects which are hard to key frame manually such as swirling motion and flows past objects are obtained automatically. Their algorithm is based mainly on the work of Harlow and Welch in computational fluid dynamics, which dates back to 1965 [11]. Since then many other techniques which Foster and Metaxas could have used have been developed. However, their model has the advantage of being simple to code, since it is based on a finite differencing of the Navier-Stokes equations and an explicit time solver. Similar solvers and their source code are also available from the book of Griebel et al. [9]. The main problem with explicit solvers is that the numerical scheme can become unstable for large time-steps. Instability leads to numerical simulations that "blow-up" and therefore have to be restarted with a smaller time-step. The instability of these explicit algorithms sets serious limits on speed and interactivity. Ideally, a user should be able to interact in real-time with a fluid solver without having to worry about possible "blow ups".

In this paper, for the first time, we propose a stable algorithm that solves the full Navier-Stokes equations. Our algorithm is very

*Alias | wavefront, 1218 Third Ave, 8th Floor, Seattle, WA 98101, U.S.A. jstam@aw.sgi.com

Jos Stam. 1999. Stable Fluids. TOG (SIGGRAPH).

Body Forces



$$\frac{\partial u}{\partial t} = g$$

Forward Euler

$$u^{n+1} = u^n + \delta t g$$

Splitting the Fluid Equations

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u - \frac{1}{\rho} \nabla p + g + \nu \Delta u$$

$$\nabla \cdot u = 0$$

body force $\quad \dfrac{\partial u}{\partial t} = g$

advection $\quad \dfrac{\partial u}{\partial t} = -u \cdot \nabla u$

diffusion $\quad \dfrac{\partial u}{\partial t} = \nu \Delta u$

projection $\quad \dfrac{\partial u}{\partial t} = -\dfrac{1}{\rho} \nabla p$

s.t. $\nabla \cdot u = 0$

Stable Fluids

Jos Stam*

Alias | wavefront

Abstract

Building animation tools for fluid-like motions is an important and challenging problem with many applications in computer graphics. The use of physics-based models for fluid flow can greatly assist in creating such tools. Physical models, unlike key frame or procedural based techniques, permit an animator to almost effortlessly create interesting, swirling fluid-like behaviors. Also, the interaction of flows with objects and virtual forces is handled elegantly. Until recently, it was believed that physical fluid models were too expensive to allow real-time interaction. This was largely due to the fact that previous models used unstable schemes to solve the physical equations governing a fluid. In this paper, for the first time, we propose an unconditionally stable model which still produces complex fluid-like flows. As well, our method is very easy to implement. The stability of our model allows us to take larger time steps and therefore achieve faster simulations. We have used our model in conjuction with advecting solid textures to create many fluid-like animations interactively in two- and three-dimensions.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: animation of fluids, Navier-Stokes, stable solvers, implicit elliptic PDE solvers, interactive modeling, gaseous phenomena, advected textures

1 Introduction

One of the most intriguing problems in computer graphics is the simulation of fluid-like behavior. A good fluid solver is of great importance in many different areas. In the special effects industry there is a high demand to convincingly mimic the appearance and behavior of fluids such as smoke, water and fire. Paint programs can also benefit from fluid solvers to emulate traditional techniques such as watercolor and oil paint. Texture synthesis is another possible application. Indeed, many textures result from fluid-like processes, such as erosion. The modeling and simulation of fluids is, of course, also of prime importance in most scientific disciplines and in engineering. Fluid mechanics is used as the standard mathematical framework on which these simulations are based. There is a consensus among scientists that the *Navier-Stokes* equations are a very good model for fluid flow. Thousands of books and articles have been published in various areas on how to compute these equations numerically. Which solver to use in practice depends largely on the problem at hand and on the computing power available. Most engineering tasks require that the simulation provide accurate bounds on the physical quantities involved to answer questions related to safety, performance, etc. The visual appearance (shape) of the flow is of secondary importance in these applications. In computer graphics, on the other hand, the shape and the behavior of the fluid are of primary interest, while physical accuracy is secondary or in some cases irrelevant. Fluid solvers, for computer graphics, should ideally provide a user with a tool that enables her to achieve fluid-like effects in real-time. These factors are more important than strict physical accuracy, which would require too much computational power.

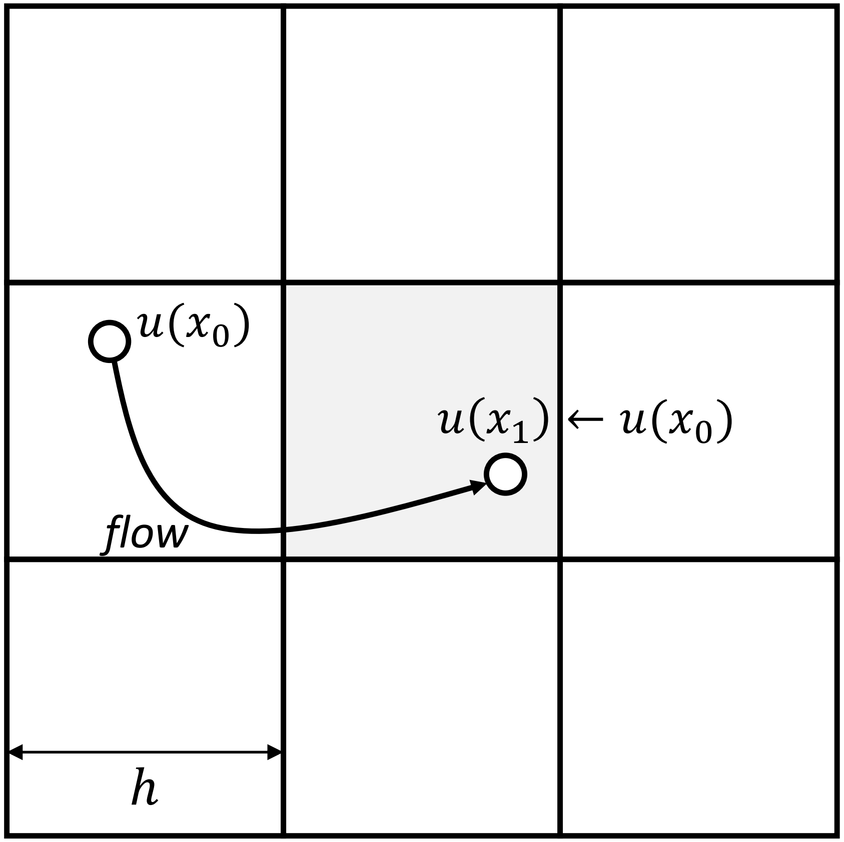
In fact, most previous models in computer graphics were driven by visual appearance and not by physical accuracy. Early flow models were built from simple primitives. Various combinations of these primitives allowed the animation of particles systems [15, 17] or simple geometries such as leaves [23]. The complexity of the flows was greatly improved with the introduction of random turbulences [16, 20]. These turbulences are mass conserving and, therefore, automatically exhibit rotational motion. Also the turbulence is periodic in space and time, which is ideal for motion "texture mapping" [19]. Flows built up from a superposition of flow primitives all have the disadvantage that they do not respond dynamically to user-applied external forces. Dynamical models of fluids based on the Navier-Stokes equations were first implemented in two-dimensions. Both Yaeger and Upson and Gamito et al. used a vortex method coupled with a Poisson solver to create two-dimensional animations of fluids [24, 8]. Later, Chen et al. animated water surfaces from the pressure term given by a two-dimensional simulation of the Navier-Stokes equations [2]. Their method unlike ours is both limited to two-dimensions and is unstable. Kass and Miller linearize the shallow water equations to simulate liquids [12]. The simplifications do not, however, capture the interesting rotational motions characteristic of fluids. More recently, Foster and Metaxas clearly show the advantages of using the full three-dimensional Navier-Stokes equations in creating fluid-like animations [7]. Many effects which are hard to key frame manually such as swirling motion and flows past objects are obtained automatically. Their algorithm is based mainly on the work of Harlow and Welch in computational fluid dynamics, which dates back to 1965 [11]. Since then many other techniques which Foster and Metaxas could have used have been developed. However, their model has the advantage of being simple to code, since it is based on a finite differencing of the Navier-Stokes equations and an explicit time solver. Similar solvers and their source code are also available from the book of Griebel et al. [9]. The main problem with explicit solvers is that the numerical scheme can become unstable for large time-steps. Instability leads to numerical simulations that "blow-up" and therefore have to be restarted with a smaller time-step. The instability of these explicit algorithms sets serious limits on speed and interactivity. Ideally, a user should be able to interact in real-time with a fluid solver without having to worry about possible "blow ups".

In this paper, for the first time, we propose a stable algorithm that solves the full Navier-Stokes equations. Our algorithm is very

*Alias | wavefront, 1218 Third Ave, 8th Floor, Seattle, WA 98101, U.S.A. jstam@aw.sgi.com

Jos Stam. 1999. Stable Fluids. TOG (SIGGRAPH).

Advection



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

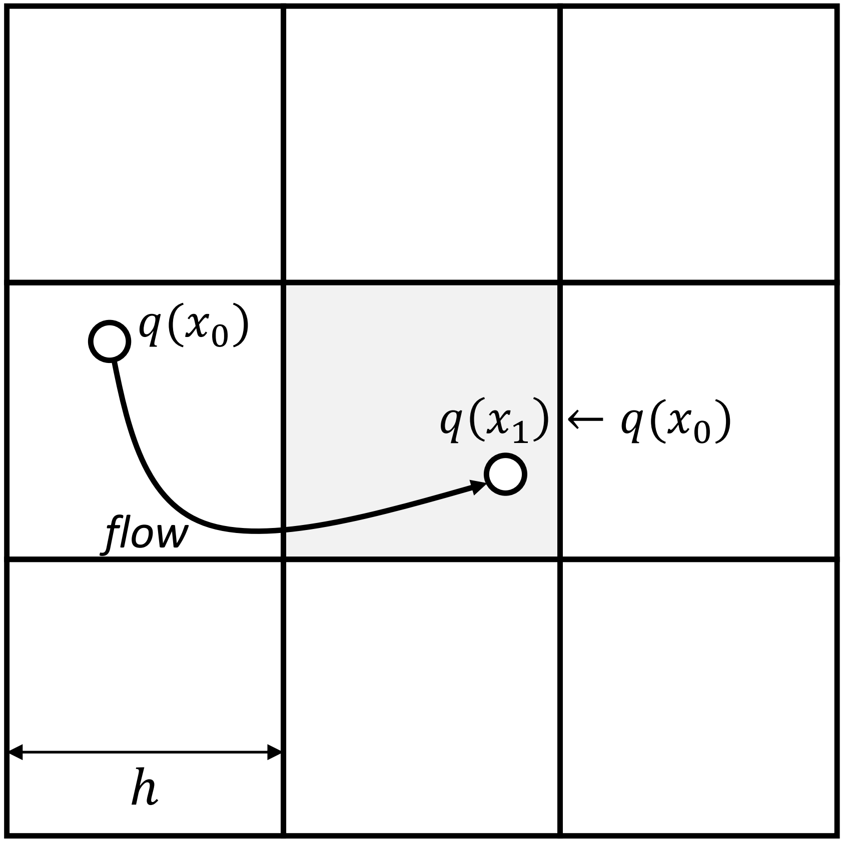
$$\boxed{u \cdot \nabla u = u \cdot \frac{\partial u}{\partial x} + v \cdot \frac{\partial v}{\partial v}}$$

$$u^{n+1} = u^n - \delta t(u \cdot \nabla u)$$

Advection

For a generic quantity q

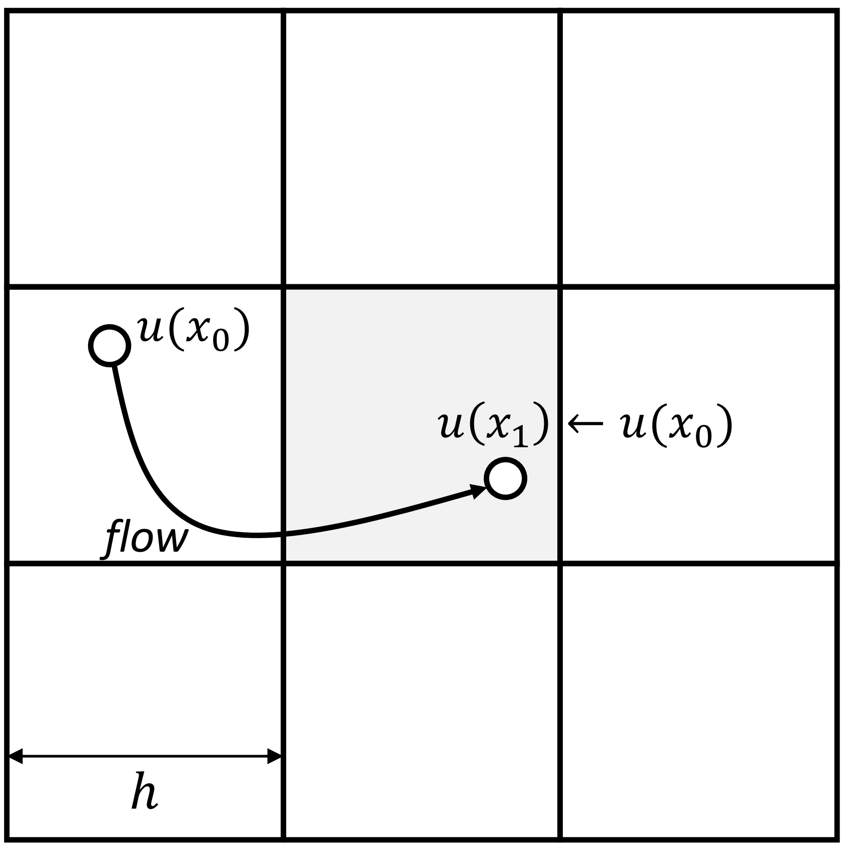
$$\frac{\partial q}{\partial t} = -u \cdot \nabla q$$



$$\boxed{u \cdot \nabla q = u \cdot \frac{\partial q}{\partial x} + v \cdot \frac{\partial q}{\partial v}}$$

$$q^{n+1} = q^n - \delta t(u \cdot \nabla q)$$

Advection



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

$$\boxed{u \cdot \nabla u = u \cdot \frac{\partial u}{\partial x} + v \cdot \frac{\partial v}{\partial v}}$$

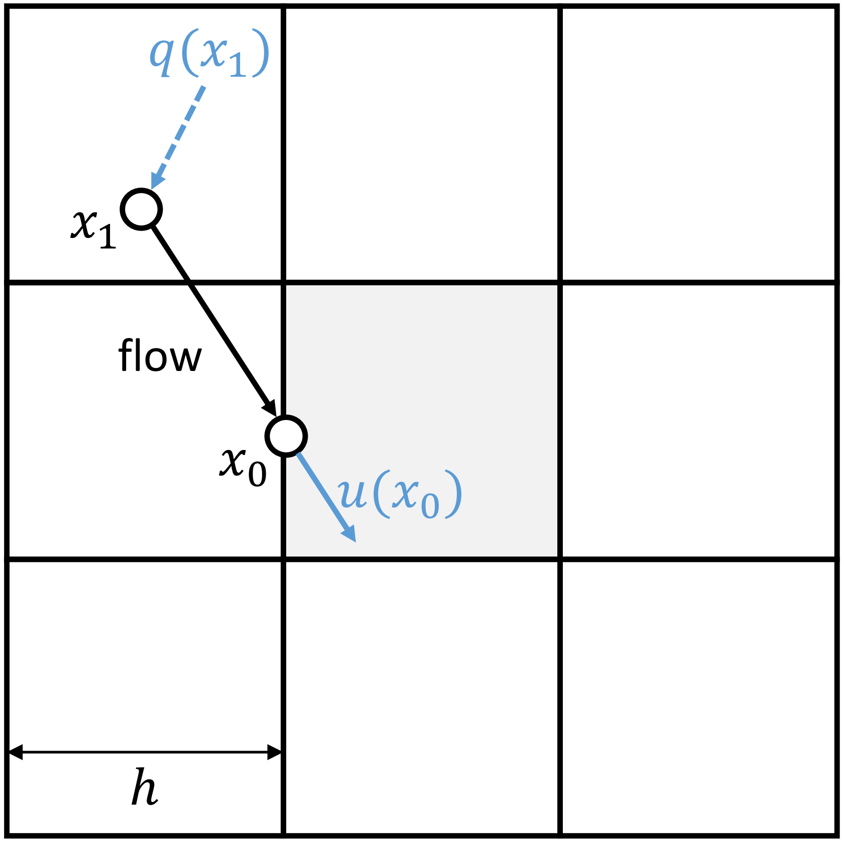
$$u^{n+1} = u^n - \delta t(u \cdot \nabla u)$$

This may be unstable!!
* Spatial forward Euler…

Advection: Semi-Lagrangian Method

Solution: Trace a virtual particle backward over time.

This is Lagrangian viewpoint



Stable Fluids

Jos Stam*

Alias | wavefront

Abstract

Building animation tools for fluid-like motions is an important and challenging problem with many applications in computer graphics. The use of physics-based models for fluid flow can greatly assist in creating such tools. Physical models, unlike key frame or procedural based techniques, permit an animator to almost effortlessly create interesting, swirling fluid-like behaviors. Also, the interaction of flows with objects and virtual forces is handled elegantly. Until recently, it was believed that physical fluid models were too expensive to allow real-time interaction. This was largely due to the fact that previous models used unstable schemes to solve the physical equations governing a fluid. In this paper, for the first time, we propose an unconditionally stable model which still produces complex fluid-like flows. As well, our method is very easy to implement. The stability of our model allows us to take larger time steps and therefore achieve faster simulations. We have used our model in conjuction with advecting solid textures to create many fluid-like animations interactively in two- and three-dimensions.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: animation of fluids, Navier-Stokes, stable solvers, implicit elliptic PDE solvers, interactive modeling, gaseous phenomena, advected textures

1 Introduction

One of the most intriguing problems in computer graphics is the simulation of fluid-like behavior. A good fluid solver is of great importance in many different areas. In the special effects industry there is a high demand to convincingly mimic the appearance and behavior of fluids such as smoke, water and fire. Paint programs can also benefit from fluid solvers to emulate traditional techniques such as watercolor and oil paint. Texture synthesis is another possible application. Indeed, many textures result from fluid-like processes, such as erosion. The modeling and simulation of fluids is, of course, also of prime importance in most scientific disciplines and in engineering. Fluid mechanics is used as the standard mathematical framework on which these simulations are based. There is a consensus among scientists that the *Navier-Stokes* equations are a very good model for fluid flow. Thousands of books and articles have been published in various areas on how to compute these equations numerically. Which solver to use in practice depends largely on the problem at hand and on the computing power available. Most engineering tasks require that the simulation provide accurate bounds on the physical quantities involved to answer questions related to safety, performance, etc. The visual appearance (shape) of the flow is of secondary importance in these applications. In computer graphics, on the other hand, the shape and the behavior of the fluid are of primary interest, while physical accuracy is secondary or in some cases irrelevant. Fluid solvers, for computer graphics, should ideally provide a user with a tool that enables her to achieve fluid-like effects in real-time. These factors are more important than strict physical accuracy, which would require too much computational power.

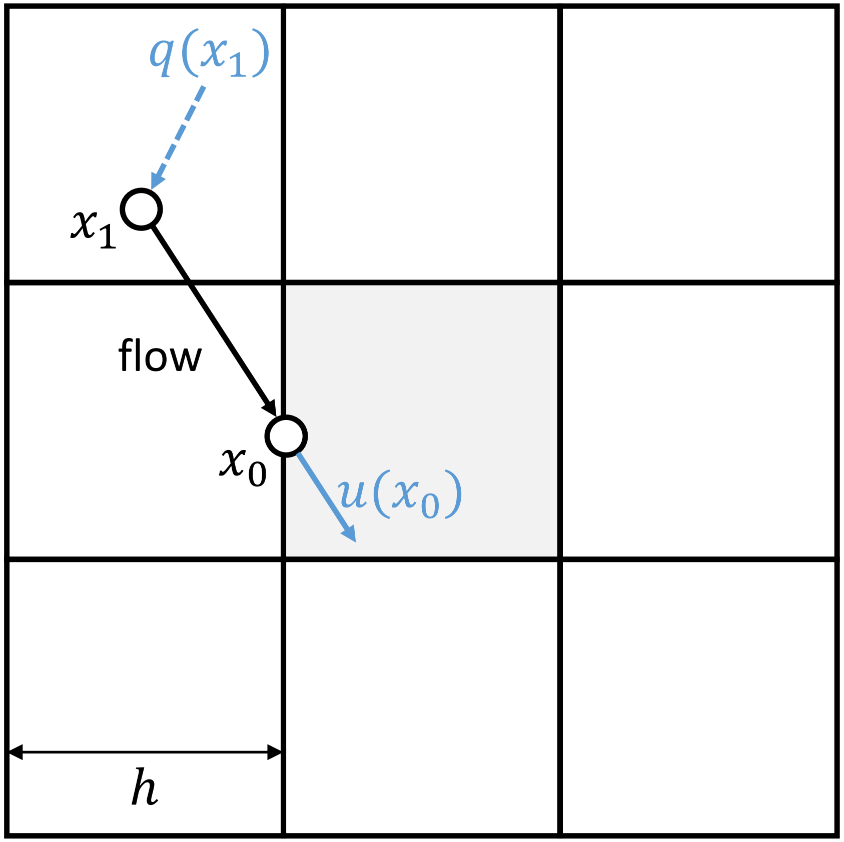
In fact, most previous models in computer graphics were driven by visual appearance and not by physical accuracy. Early flow models were built from simple primitives. Various combinations of these primitives allowed the animation of particles systems [15, 17] or simple geometries such as leaves [23]. The complexity of the flows was greatly improved with the introduction of random turbulences [16, 20]. These turbulences are mass conserving and, therefore, automatically exhibit rotational motion. Also the turbulence is periodic in space and time, which is ideal for motion "texture mapping" [19]. Flows built up from a superposition of flow primitives all have the disadvantage that they do not respond dynamically to user-applied external forces. Dynamical models of fluids based on the Navier-Stokes equations were first implemented in two-dimensions. Both Yaeger and Upson and Gamito et al. used a vortex method coupled with a Poisson solver to create two-dimensional animations of fluids [24, 8]. Later, Chen et al. animated water surfaces from the pressure term given by a two-dimensional simulation of the Navier-Stokes equations [2]. Their method unlike ours is both limited to two-dimensions and is unstable. Kass and Miller linearize the shallow water equations to simulate liquids [12]. The simplifications do not, however, capture the interesting rotational motions characteristic of fluids. More recently, Foster and Metaxas clearly show the advantages of using the full three-dimensional Navier-Stokes equations in creating fluid-like animations [7]. Many effects which are hard to key frame manually such as swirling motion and flows past objects are obtained automatically. Their algorithm is based mainly on the work of Harlow and Welch in computational fluid dynamics, which dates back to 1965 [11]. Since then many other techniques which Foster and Metaxas could have used have been developed. However, their model has the advantage of being simple to code, since it is based on a finite differencing of the Navier-Stokes equations and an explicit time solver. Similar solvers and their source code are also available from the book of Griebel et al. [9]. The main problem with explicit solvers is that the numerical scheme can become unstable for large time-steps. Instability leads to numerical simulations that "blow-up" and therefore have to be restarted with a smaller time-step. The instability of these explicit algorithms sets serious limits on speed and interactivity. Ideally, a user should be able to interact in real-time with a fluid solver without having to worry about possible "blow ups".

In this paper, for the first time, we propose a stable algorithm that solves the full Navier-Stokes equations. Our algorithm is very

*Alias | wavefront, 1218 Third Ave, 8th Floor, Seattle, WA 98101, U.S.A. jstam@aw.sgi.com

Advection: Semi-Lagrangian Method

Solution: Trace a virtual particle backward over time.



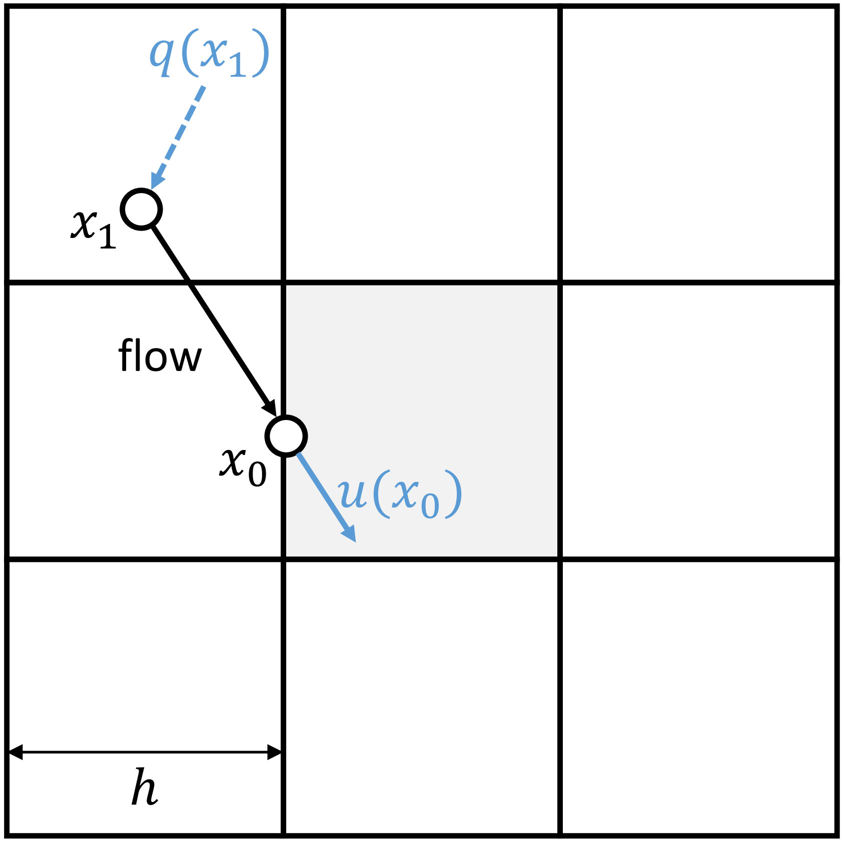
Advect q over u:

$$\frac{\partial q}{\partial t} = -u \cdot \nabla q$$

- Define $x_0 \leftarrow (i - \frac{1}{2}, j)$
- Compute $u(x_0)$
- $x_1 \leftarrow x_0 - \delta t\, u(x_0)$
- Compute $q^n(x_1)$
- $q^{n+1}(x_0) \leftarrow q^n(x_1)$

Advection: Semi-Lagrangian Method

Solution: Trace a virtual particle backward over time.



Advect q over u:

$$\frac{\partial q}{\partial t} = -u \cdot \nabla q$$

$$q \Leftarrow u_{i-\frac{1}{2},j}$$

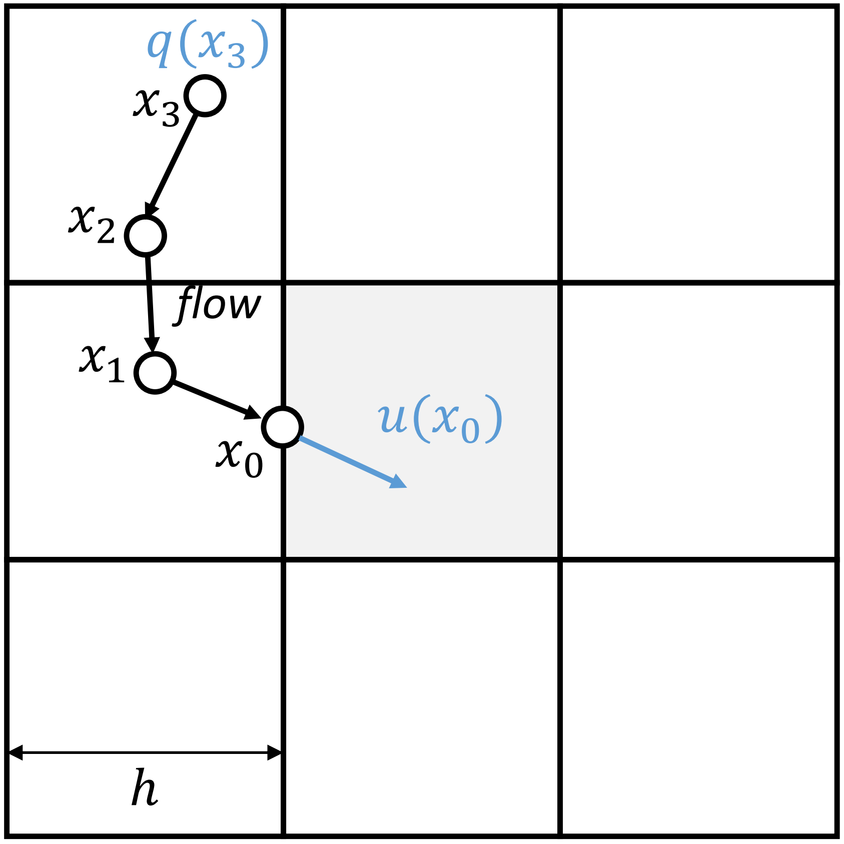
$$q \Leftarrow v_{i,j-\frac{1}{2}}$$

$$q \Leftarrow T$$

$$q \Leftarrow \sigma$$

Advection: Semi-Lagrangian Method

Subdivide the time step for better tracking...



- Define $x_0 \leftarrow (i - \frac{1}{2}, j)$
- Compute $u(x_0)$
- $x_1 \leftarrow x_0 - \frac{1}{3}\delta t\, u(x_0)$
- Compute $u(x_1)$
- $x_2 \leftarrow x_1 - \frac{1}{3}\delta t\, u(x_1)$
- Compute $u(x_2)$
- $x_3 \leftarrow x_2 - \frac{1}{3}\delta t\, u(x_2)$
- Compute $q^n(x_3)$
- $q^{n+1}(x_0) \leftarrow q^n(x_3)$

Splitting the Fluid Equations

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u - \frac{1}{\rho}\nabla p + g + \nu \Delta u$$

$$\nabla \cdot u = 0$$

body force $\quad \dfrac{\partial u}{\partial t} = g$

advection $\quad \dfrac{\partial u}{\partial t} = -u \cdot \nabla u$

diffusion $\quad \dfrac{\partial u}{\partial t} = \nu \Delta u$

projection $\quad \dfrac{\partial u}{\partial t} = -\dfrac{1}{\rho}\nabla p$

s.t. $\nabla \cdot u = 0$

Stable Fluids

Jos Stam*

Alias | wavefront

Abstract

Building animation tools for fluid-like motions is an important and challenging problem with many applications in computer graphics. The use of physics-based models for fluid flow can greatly assist in creating such tools. Physical models, unlike key frame or procedural based techniques, permit an animator to almost effortlessly create interesting, swirling fluid-like behaviors. Also, the interaction of flows with objects and virtual forces is handled elegantly. Until recently, it was believed that physical fluid models were too expensive to allow real-time interaction. This was largely due to the fact that previous models used unstable schemes to solve the physical equations governing a fluid. In this paper, for the first time, we propose an unconditionally stable model which still produces complex fluid-like flows. As well, our method is very easy to implement. The stability of our model allows us to take larger time steps and therefore achieve faster simulations. We have used our model in conjuction with advecting solid textures to create many fluid-like animations interactively in two- and three-dimensions.

CR Categories: 1.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: animation of fluids, Navier-Stokes, stable solvers, implicit elliptic PDE solvers, interactive modeling, gaseous phenomena, advected textures

1 Introduction

One of the most intriguing problems in computer graphics is the simulation of fluid-like behavior. A good fluid solver is of great importance in many different areas. In the special effects industry there is a high demand to convincingly mimic the appearance and behavior of fluids such as smoke, water and fire. Paint programs can also benefit from fluid solvers to emulate traditional techniques such as watercolor and oil paint. Texture synthesis is another possible application. Indeed, many textures result from fluid-like processes, such as erosion. The modeling and simulation of fluids is, of course, also of prime importance in most scientific disciplines and in engineering. Fluid mechanics is used as the standard mathematical framework on which these simulations are based. There is a consensus among scientists that the *Navier-Stokes* equations are a very good model for fluid flow. Thousands of books and articles have been published in various areas on how to compute these equations numerically. Which solver to use in practice depends largely on the problem at hand and on the computing power available. Most engineering tasks require that the simulation provide accurate bounds on the physical quantities involved to answer questions related to safety, performance, etc. The visual appearance (shape) of the flow is of secondary importance in these applications. In computer graphics, on the other hand, the shape and the behavior of the fluid are of primary interest, while physical accuracy is secondary or in some cases irrelevant. Fluid solvers, for computer graphics, should ideally provide a user with a tool that enables her to achieve fluid-like effects in real-time. These factors are more important than strict physical accuracy, which would require too much computational power.

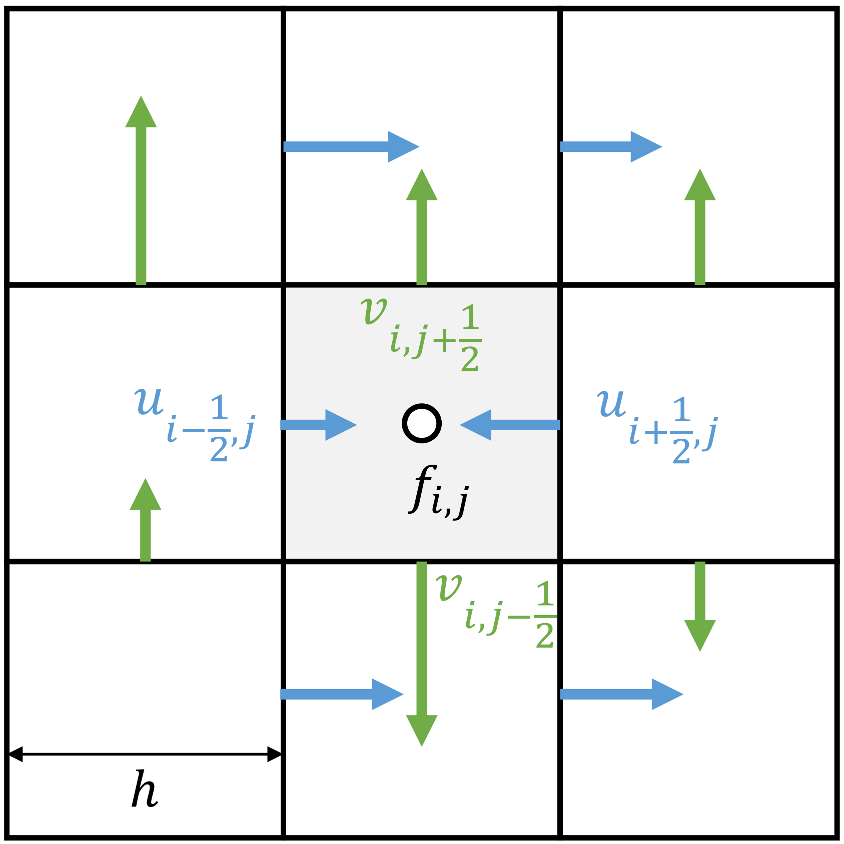
In fact, most previous models in computer graphics were driven by visual appearance and not by physical accuracy. Early flow models were built from simple primitives. Various combinations of these primitives allowed the animation of particles systems [15, 17] or simple geometries such as leaves [23]. The complexity of the flows was greatly improved with the introduction of random turbulences [16, 20]. These turbulences are mass conserving and, therefore, automatically exhibit rotational motion. Also the turbulence is periodic in space and time, which is ideal for motion "texture mapping" [19]. Flows built up from a superposition of flow primitives all have the disadvantage that they do not respond dynamically to user-applied external forces. Dynamical models of fluids based on the Navier-Stokes equations were first implemented in two-dimensions. Both Yaeger and Upson and Gamito et al. used a vortex method coupled with a Poisson solver to create two-dimensional animations of fluids [24, 8]. Later, Chen et al. animated water surfaces from the pressure term given by a two-dimensional simulation of the Navier-Stokes equations [2]. Their method unlike ours is both limited to two-dimensions and is unstable. Kass and Miller linearize the shallow water equations to simulate liquids [12]. The simplifications do not, however, capture the interesting rotational motions characteristic of fluids. More recently, Foster and Metaxas clearly show the advantages of using the full three-dimensional Navier-Stokes equations in creating fluid-like animations [7]. Many effects which are hard to key frame manually such as swirling motion and flows past objects are obtained automatically. Their algorithm is based mainly on the work of Harlow and Welch in computational fluid dynamics, which dates back to 1965 [11]. Since then many other techniques which Foster and Metaxas could have used have been developed. However, their model has the advantage of being simple to code, since it is based on a finite differencing of the Navier-Stokes equations and an explicit time solver. Similar solvers and their source code are also available from the book of Griebel et al. [9]. The main problem with explicit solvers is that the numerical scheme can become unstable for large time-steps. Instability leads to numerical simulations that "blow-up" and therefore have to be restarted with a smaller time-step. The instability of these explicit algorithms sets serious limits on speed and interactivity. Ideally, a user should be able to interact in real-time with a fluid solver without having to worry about possible "blow ups".

In this paper, for the first time, we propose a stable algorithm that solves the full Navier-Stokes equations. Our algorithm is very

*Alias | wavefront, 1218 Third Ave, 8th Floor, Seattle, WA 98101, U.S.A. jstam@aw.sgi.com

Jos Stam. 1999. Stable Fluids. TOG (SIGGRAPH).

Diffusion



$$\frac{\partial u}{\partial t} = v\Delta u$$

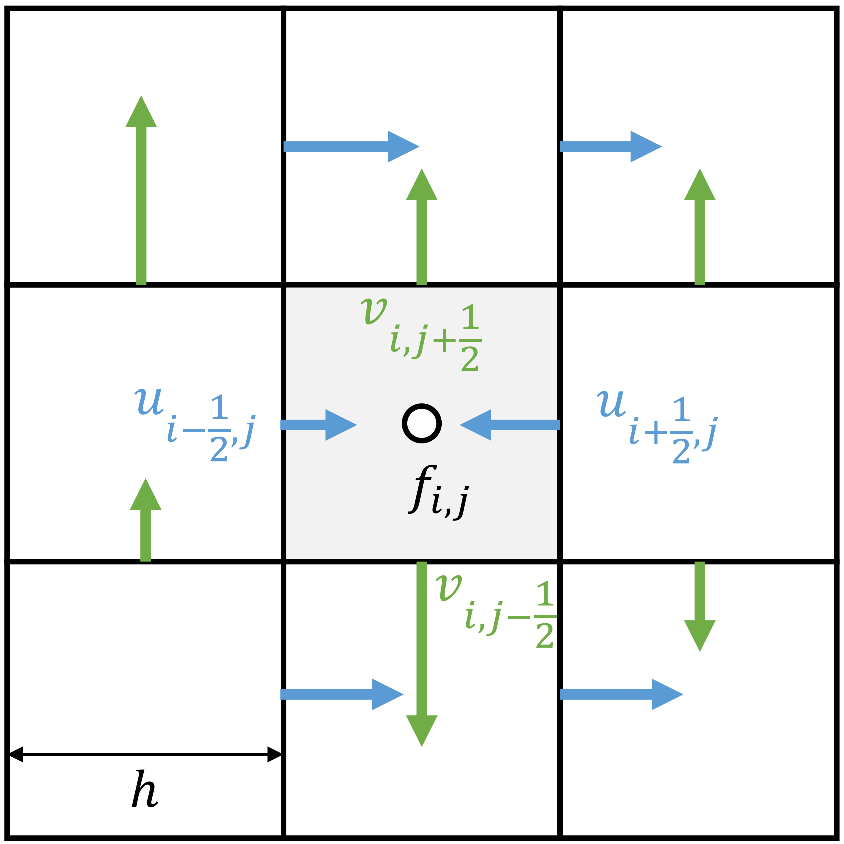
Forward Euler

$$u^{n+1}_{i-\frac{1}{2},j} = u^{n}_{i-\frac{1}{2},j} + v\delta t \Delta u_{i-\frac{1}{2},j}$$

$$v^{n+1}_{i,j-\frac{1}{2}} = u^{n}_{i,j-\frac{1}{2}} + v\delta t \Delta u_{i,j-\frac{1}{2}}$$

Unstable when $v\delta t$ is large...

Diffusion



$$\frac{\partial u}{\partial t} = v\Delta u$$

Forward Euler

$$u^{\text{temp}}_{i-\frac{1}{2},j} = u^{n}_{i-\frac{1}{2},j} + v\frac{\delta t}{2}\Delta u_{i-\frac{1}{2},j}$$

$$u^{n+1}_{i-\frac{1}{2},j} = u^{\text{temp}}_{i-\frac{1}{2},j} + v\frac{\delta t}{2}\Delta u^{\text{temp}}_{i-\frac{1}{2},j}$$

Sub-steps can help...

Splitting the Fluid Equations

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u - \frac{1}{\rho} \nabla p + g + \nu \Delta u$$

$$\nabla \cdot u = 0$$

body force $\qquad \dfrac{\partial u}{\partial t} = g$

advection $\qquad \dfrac{\partial u}{\partial t} = -u \cdot \nabla u$

diffusion $\qquad \dfrac{\partial u}{\partial t} = \nu \Delta u$

projection $\qquad \dfrac{\partial u}{\partial t} = -\dfrac{1}{\rho} \nabla p$

s.t. $\nabla \cdot u = 0$

Stable Fluids

Jos Stam*

Alias | wavefront

Abstract

Building animation tools for fluid-like motions is an important and challenging problem with many applications in computer graphics. The use of physics-based models for fluid flow can greatly assist in creating such tools. Physical models, unlike key frame or procedural based techniques, permit an animator to almost effortlessly create interesting, swirling fluid-like behaviors. Also, the interaction of flows with objects and virtual forces is handled elegantly. Until recently, it was believed that physical fluid models were too expensive to allow real-time interaction. This was largely due to the fact that previous models used unstable schemes to solve the physical equations governing a fluid. In this paper, for the first time, we propose an unconditionally stable model which still produces complex fluid-like flows. As well, our method is very easy to implement. The stability of our model allows us to take larger time steps and therefore achieve faster simulations. We have used our model in conjuction with advecting solid textures to create many fluid-like animations interactively in two- and three-dimensions.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: animation of fluids, Navier-Stokes, stable solvers, implicit elliptic PDE solvers, interactive modeling, gaseous phenomena, advected textures

1 Introduction

One of the most intriguing problems in computer graphics is the simulation of fluid-like behavior. A good fluid solver is of great importance in many different areas. In the special effects industry there is a high demand to convincingly mimic the appearance and behavior of fluids such as smoke, water and fire. Paint programs can also benefit from fluid solvers to emulate traditional techniques such as watercolor and oil paint. Texture synthesis is another possible application. Indeed, many textures result from fluid-like processes, such as erosion. The modeling and simulation of fluids is, of course, also of prime importance in most scientific disciplines and in engineering. Fluid mechanics is used as the standard mathematical framework on which these simulations are based. There is a consensus among scientists that the *Navier-Stokes* equations are a very good model for fluid flow. Thousands of books and articles have been published in various areas on how to compute these equations numerically. Which solver to use in practice depends largely on the problem at hand and on the computing power available. Most engineering tasks require that the simulation provide accurate bounds on the physical quantities involved to answer questions related to safety, performance, etc. The visual appearance (shape) of the flow is of secondary importance in these applications. In computer graphics, on the other hand, the shape and the behavior of the fluid are of primary interest, while physical accuracy is secondary or in some cases irrelevant. Fluid solvers, for computer graphics, should ideally provide a user with a tool that enables her to achieve fluid-like effects in real-time. These factors are more important than strict physical accuracy, which would require too much computational power.

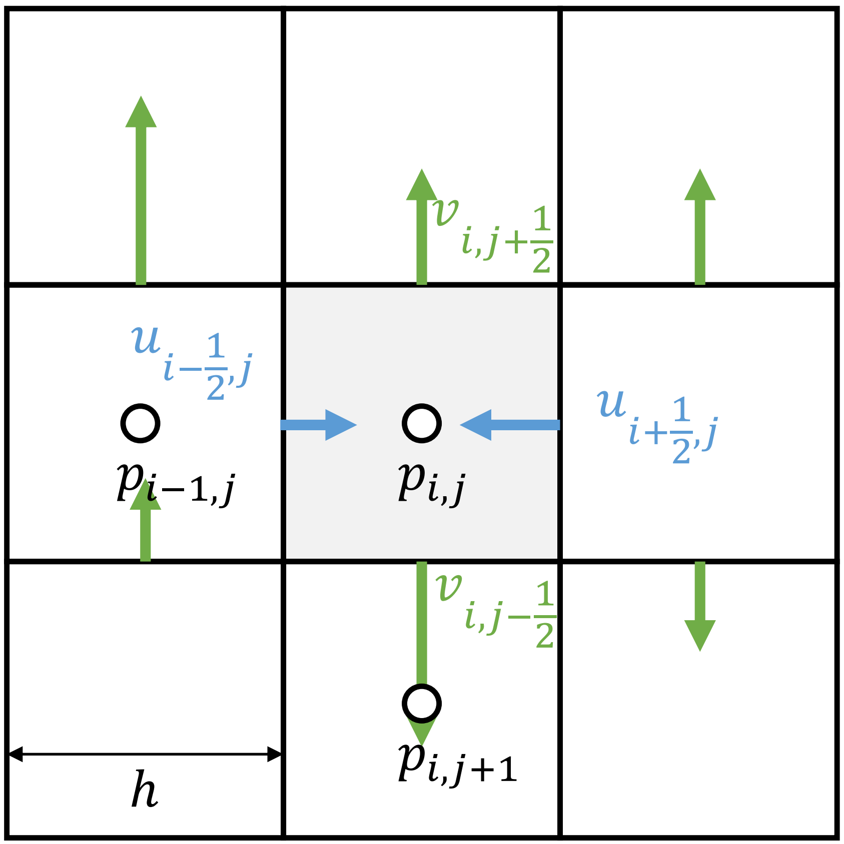
In fact, most previous models in computer graphics were driven by visual appearance and not by physical accuracy. Early flow models were built from simple primitives. Various combinations of these primitives allowed the animation of particles systems [15, 17] or simple geometries such as leaves [23]. The complexity of the flows was greatly improved with the introduction of random turbulences [16, 20]. These turbulences are mass conserving and, therefore, automatically exhibit rotational motion. Also the turbulence is periodic in space and time, which is ideal for motion "texture mapping" [19]. Flows built up from a superposition of flow primitives all have the disadvantage that they do not respond dynamically to user-applied external forces. Dynamical models of fluids based on the Navier-Stokes equations were first implemented in two-dimensions. Both Yaeger and Upson and Gamito et al. used a vortex method coupled with a Poisson solver to create two-dimensional animations of fluids [24, 8]. Later, Chen et al. animated water surfaces from the pressure term given by a two-dimensional simulation of the Navier-Stokes equations [2]. Their method unlike ours is both limited to two-dimensions and is unstable. Kass and Miller linearize the shallow water equations to simulate liquids [12]. The simplifications do not, however, capture the interesting rotational motions characteristic of fluids. More recently, Foster and Metaxas clearly show the advantages of using the full three-dimensional Navier-Stokes equations in creating fluid-like animations [7]. Many effects which are hard to key frame manually such as swirling motion and flows past objects are obtained automatically. Their algorithm is based mainly on the work of Harlow and Welch in computational fluid dynamics, which dates back to 1965 [11]. Since then many other techniques which Foster and Metaxas could have used have been developed. However, their model has the advantage of being simple to code, since it is based on a finite differencing of the Navier-Stokes equations and an explicit time solver. Similar solvers and their source code are also available from the book of Griebel et al. [9]. The main problem with explicit solvers is that the numerical scheme can become unstable for large time-steps. Instability leads to numerical simulations that "blow-up" and therefore have to be restarted with a smaller time-step. The instability of these explicit algorithms sets serious limits on speed and interactivity. Ideally, a user should be able to interact in real-time with a fluid solver without having to worry about possible "blow ups".

In this paper, for the first time, we propose a stable algorithm that solves the full Navier-Stokes equations. Our algorithm is very

*Alias | wavefront, 1218 Third Ave, 8th Floor, Seattle, WA 98101, U.S.A. jstam@aw.sgi.com

Jos Stam. 1999. Stable Fluids. TOG (SIGGRAPH).

Pressure Projection

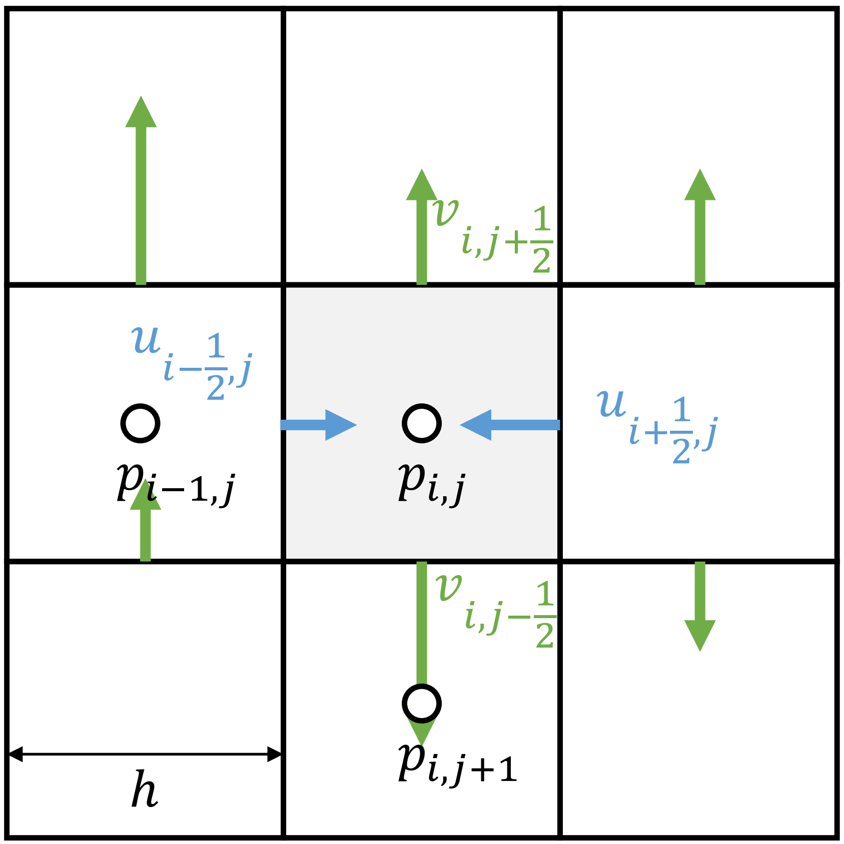


$$\frac{\partial u}{\partial t} = -\frac{1}{\rho}\nabla p$$

$$\text{s.t. } \nabla \cdot u = 0$$

Incompressibility

Pressure Projection



$$\frac{\partial u}{\partial t} = -\frac{1}{\rho}\nabla p$$

Forward Euler

$$u^{n+1}_{i-\frac{1}{2},j} = u^{n}_{i-\frac{1}{2},j} - \frac{1}{\rho}\left(p_{i,j} - p_{i-1,j}\right)$$

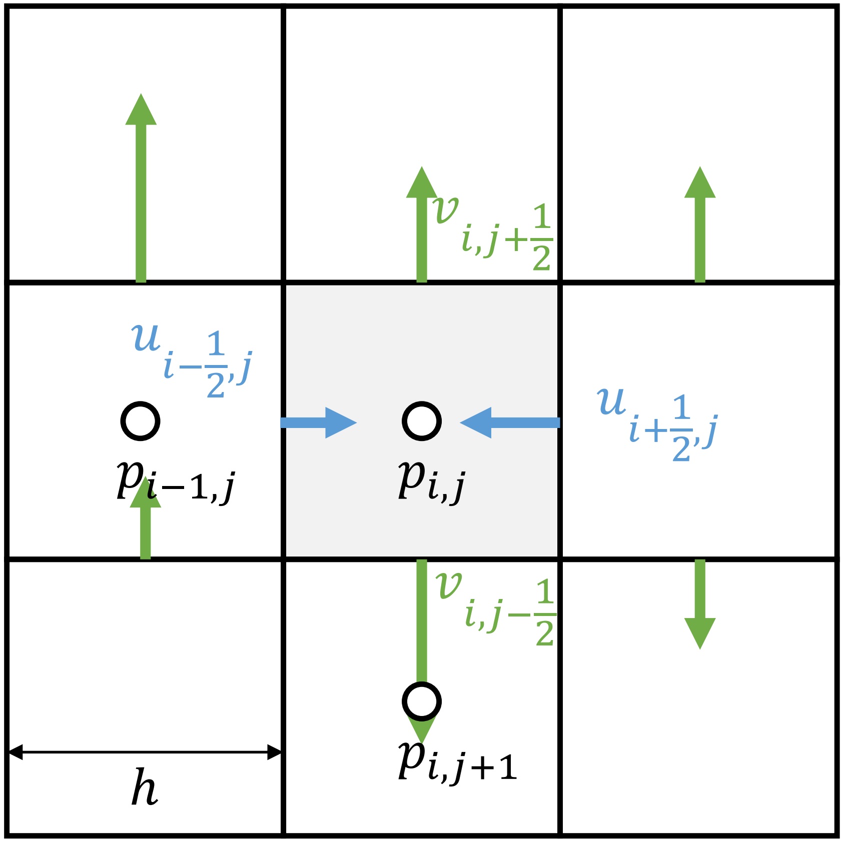
$$v^{n+1}_{i,j-\frac{1}{2}} = u^{n}_{i,j-\frac{1}{2}} - \frac{1}{\rho}\left(p_{i,j} - p_{i,j-1}\right)$$

How to determine p??

Pressure Projection

Incompressibility $\nabla \cdot u = 0$

* The pressure is caused by incompressibility
* The pressure needs to maintain $\nabla \cdot u = 0$

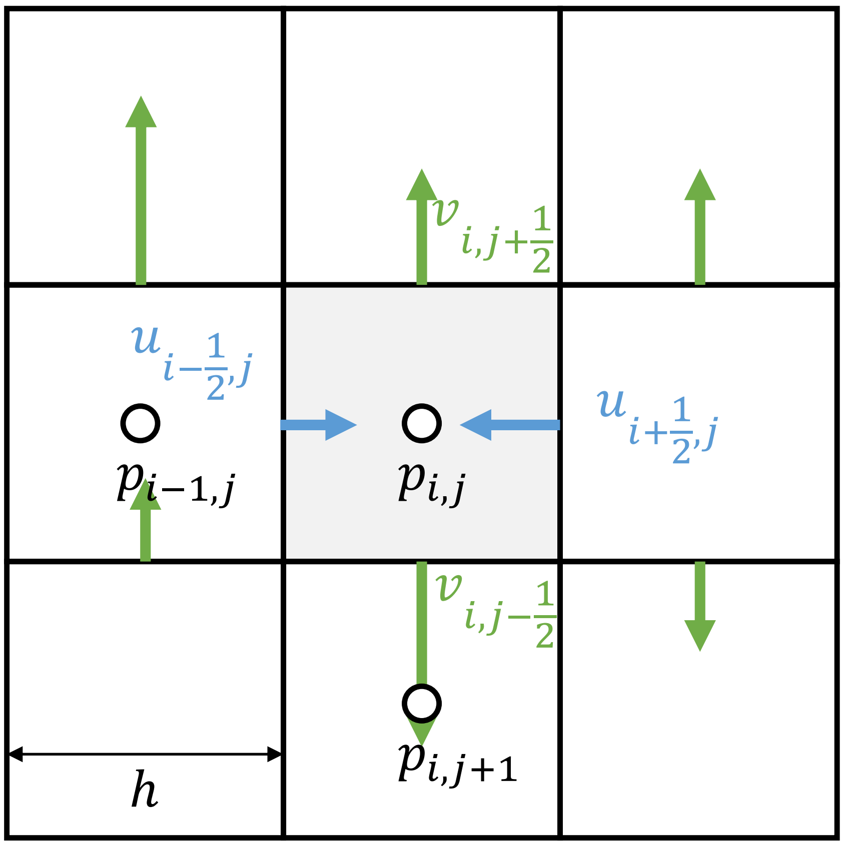


$$\nabla \cdot u^{n+1} = 0$$

$$\nabla \cdot u_{i,j}^{n+1} = \frac{\partial u_{i,j}^{n+1}}{\partial x} + \frac{\partial u_{i,j}^{n+1}}{\partial y}$$

$$= \frac{u_{i+\frac{1}{2},j}^{n+1} - u_{i-\frac{1}{2},j}^{n+1}}{h} + \frac{v_{i,j+\frac{1}{2}}^{n+1} - v_{i,j-\frac{1}{2}}^{n+1}}{h}$$

Pressure Projection



$$\frac{\partial u}{\partial t} = -\frac{1}{\rho}\nabla p$$

$$\text{s.t. } \nabla \cdot u = 0$$

$$Ap = b$$

A: large, sparse, constant, SPD

b: computed from u^n

Once we solve $\boldsymbol{p}$, we update $\mathbf{u}$ and done.

Splitting the Fluid Equations

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u - \frac{1}{\rho}\nabla p + g + \nu \Delta u$$

$$\nabla \cdot u = 0$$

body force $\quad \dfrac{\partial u}{\partial t} = g$

advection $\quad \dfrac{\partial u}{\partial t} = -u \cdot \nabla u$

diffusion $\quad \dfrac{\partial u}{\partial t} = \nu \Delta u$

projection $\quad \dfrac{\partial u}{\partial t} = -\dfrac{1}{\rho}\nabla p$

s.t. $\nabla \cdot u = 0$

Stable Fluids

Jos Stam*

Alias|wavefront

Abstract

Building animation tools for fluid-like motions is an important and challenging problem with many applications in computer graphics. The use of physics-based models for fluid flow can greatly assist in creating such tools. Physical models, unlike key frame or procedural based techniques, permit an animator to almost effortlessly create interesting, swirling fluid-like behaviors. Also, the interaction of flows with objects and virtual forces is handled elegantly. Until recently, it was believed that physical fluid models were too expensive to allow real-time interaction. This was largely due to the fact that previous models used unstable schemes to solve the physical equations governing a fluid. In this paper, for the first time, we propose an unconditionally stable model which still produces complex fluid-like flows. As well, our method is very easy to implement. The stability of our model allows us to take larger time steps and therefore achieve faster simulations. We have used our model in conjuction with advecting solid textures to create many fluid-like animations interactively in two- and three-dimensions.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: animation of fluids, Navier-Stokes, stable solvers, implicit elliptic PDE solvers, interactive modeling, gaseous phenomena, advected textures

1 Introduction

One of the most intriguing problems in computer graphics is the simulation of fluid-like behavior. A good fluid solver is of great importance in many different areas. In the special effects industry there is a high demand to convincingly mimic the appearance and behavior of fluids such as smoke, water and fire. Paint programs can also benefit from fluid solvers to emulate traditional techniques such as watercolor and oil paint. Texture synthesis is another possible application. Indeed, many textures result from fluid-like processes, such as erosion. The modeling and simulation of fluids is, of course, also of prime importance in most scientific disciplines and in engineering. Fluid mechanics is used as the standard mathematical framework on which these simulations are based. There is a consensus among scientists that the *Navier-Stokes* equations are a very good model for fluid flow. Thousands of books and articles have been published in various areas on how to compute these equations numerically. Which solver to use in practice depends largely on the problem at hand and on the computing power available. Most engineering tasks require that the simulation provide accurate bounds on the physical quantities involved to answer questions related to safety, performance, etc. The visual appearance (shape) of the flow is of secondary importance in these applications. In computer graphics, on the other hand, the shape and the behavior of the fluid are of primary interest, while physical accuracy is secondary or in some cases irrelevant. Fluid solvers, for computer graphics, should ideally provide a user with a tool that enables her to achieve fluid-like effects in real-time. These factors are more important than strict physical accuracy, which would require too much computational power.

In fact, most previous models in computer graphics were driven by visual appearance and not by physical accuracy. Early flow models were built from simple primitives. Various combinations of these primitives allowed the animation of particles systems [15, 17] or simple geometries such as leaves [23]. The complexity of the flows was greatly improved with the introduction of random turbulences [16, 20]. These turbulences are mass conserving and, therefore, automatically exhibit rotational motion. Also the turbulence is periodic in space and time, which is ideal for motion "texture mapping" [19]. Flows built up from a superposition of flow primitives all have the disadvantage that they do not respond dynamically to user-applied external forces. Dynamical models of fluids based on the Navier-Stokes equations were first implemented in two-dimensions. Both Yaeger and Upson and Gamito et al. used a vortex method coupled with a Poisson solver to create two-dimensional animations of fluids [24, 8]. Later, Chen et al. animated water surfaces from the pressure term given by a two-dimensional simulation of the Navier-Stokes equations [2]. Their method unlike ours is both limited to two-dimensions and is unstable. Kass and Miller linearize the shallow water equations to simulate liquids [12]. The simplifications do not, however, capture the interesting rotational motions characteristic of fluids. More recently, Foster and Metaxas clearly show the advantages of using the full three-dimensional Navier-Stokes equations in creating fluid-like animations [7]. Many effects which are hard to key frame manually such as swirling motion and flows past objects are obtained automatically. Their algorithm is based mainly on the work of Harlow and Welch in computational fluid dynamics, which dates back to 1965 [11]. Since then many other techniques which Foster and Metaxas could have used have been developed. However, their model has the advantage of being simple to code, since it is based on a finite differencing of the Navier-Stokes equations and an explicit time solver. Similar solvers and their source code are also available from the book of Griebel et al. [9]. The main problem with explicit solvers is that the numerical scheme can become unstable for large time-steps. Instability leads to numerical simulations that "blow-up" and therefore have to be restarted with a smaller time-step. The instability of these explicit algorithms sets serious limits on speed and interactivity. Ideally, a user should be able to interact in real-time with a fluid solver without having to worry about possible "blow ups".

In this paper, for the first time, we propose a stable algorithm that solves the full Navier-Stokes equations. Our algorithm is very

*Alias|wavefront, 1218 Third Ave, 8th Floor, Seattle, WA 98101, U.S.A. jstam@aw.sgi.com

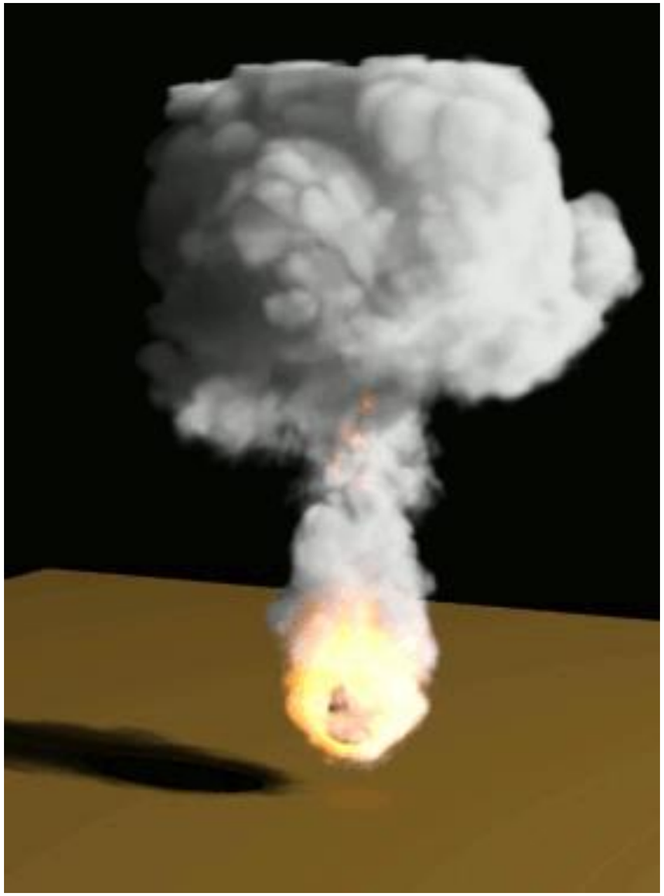
Jos Stam. 1999. Stable Fluids. TOG (SIGGRAPH).

Example: Smoke

- Step1:
  update the flow (the velocity field) u

- Step2:
  advect other physical quantities, e.g.
  - temperature T
  - density of smoke σ,

using the semi-Lagrangian method



Example: Smoke

- Air is not actually incompressible, its density is not constant
  - In the open air, the volume of the smoke is determined by its temperature
  - Hot air is lighter, smoke partials make it heavier…
- A simplified approximation: buoyant acceleration

$$b = \left(\alpha\sigma - \beta(T - T_{amb})\right)g$$

Example: Water

- Step1:
  update the flow (the velocity field) u

- Step2:
  advect other physical quantities

- How to keep track where the water is?
  - Marker particles
  - Level set
    - Often signed distance function (SDF)

Uncovered Topics

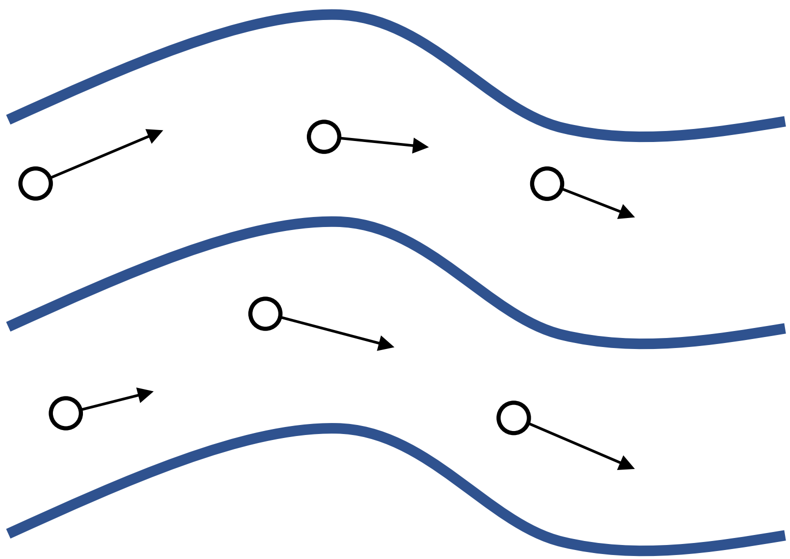
- Compressible fluid
- (Numerical) Dissipation
- Volume Loss
- Turbulence
- Bubble
- Surface tension
- Coupling with other fluids/solids
- ……

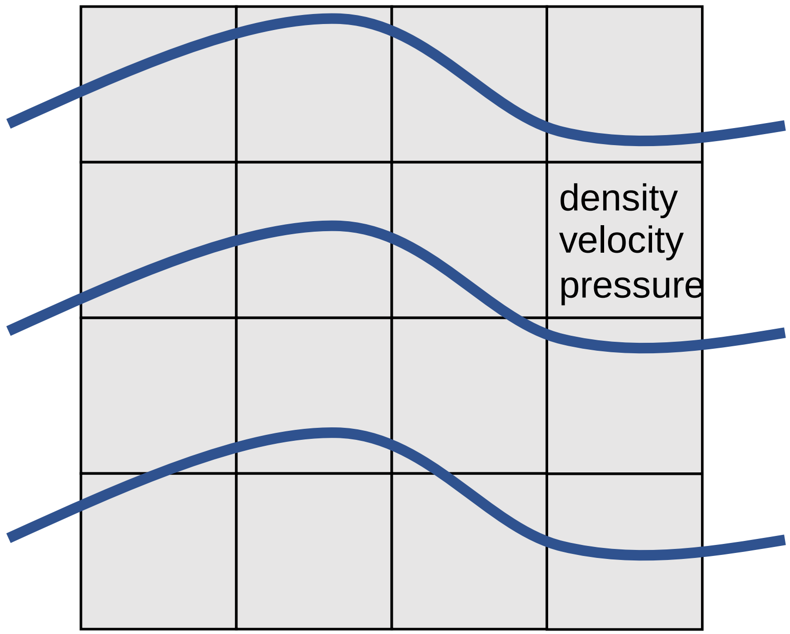
Smoothed Particle Hydrodynamics (SPH) Method

Faster Simulation?

- Procedure Water
  - Example: superimposing sine waves onto surface
- Heightfield Approximations
  - Shallow wave equation
- Particle-based Methods
  - Smoothed Particle Hydrodynamics (SPH)

Recall: Lagrangian and Eulerian Viewpoints

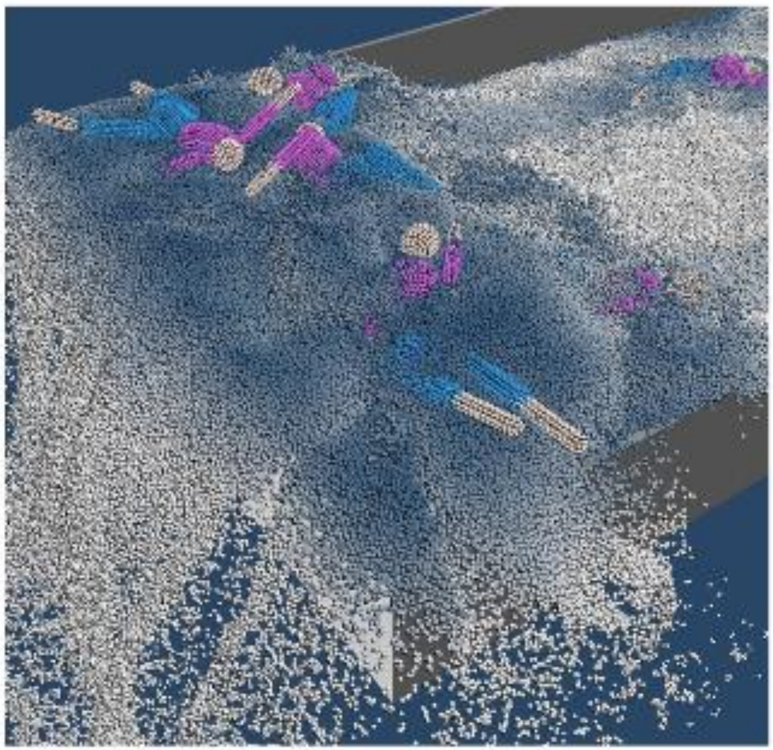




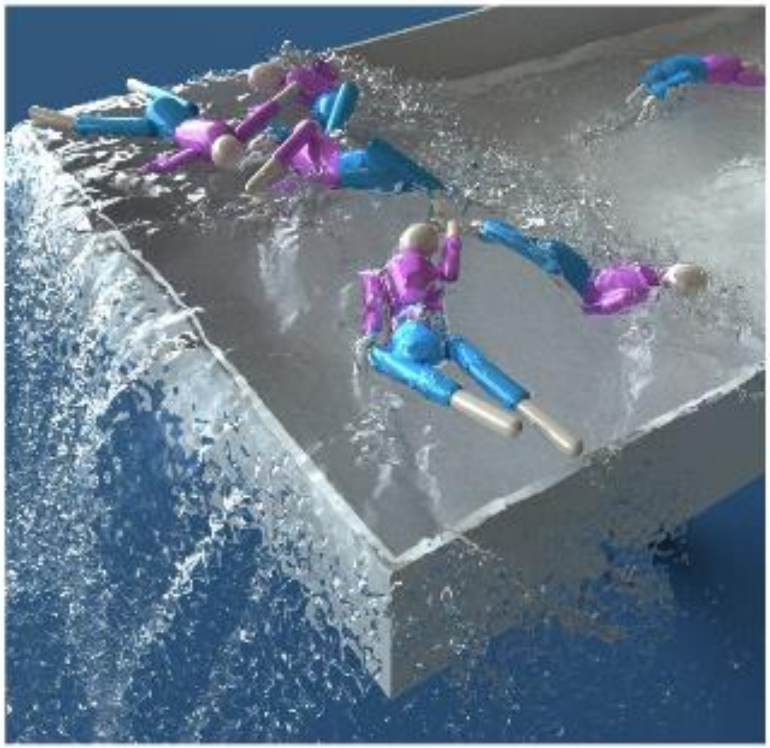
Lagrangian Approach
(dynamic particles or mesh)
Node movement carries physical quantities
(mass, velocity, …).

Eulerian Approach
(static grid or mesh)
Grid/Mesh doesn’t move.
Stored physical quantities change.

SPH Model: a Lagrangian Approach



representation



typical visualization

SPH Model: a Lagrangian Approach

Eurographics/SIGGRAPH Symposium on Computer Animation (2003)
D. Breen, M. Lin (Editors)

Particle-Based Fluid Simulation for Interactive Applications

Matthias Müller, David Charypar and Markus Gross

Department of Computer Science, Federal Institute of Technology Zürich (ETHZ), Switzerland

Abstract

Realistically animated fluids can add substantial realism to interactive applications such as virtual surgery simulators or computer games. In this paper we propose an interactive method based on Smoothed Particle Hydrodynamics (SPH) to simulate fluids with free surfaces. The method is an extension of the SPH-based technique by Desbrun to animate highly deformable bodies. We gear the method towards fluid simulation by deriving the force density fields directly from the Navier-Stokes equation and by adding a term to model surface tension effects. In contrast to Eulerian grid-based approaches, the particle-based approach makes mass conservation equations and convection terms dispensable which reduces the complexity of the simulation. In addition, the particles can directly be used to render the surface of the fluid. We propose methods to track and visualize the free surface using point splatting and marching cubes-based surface reconstruction. Our animation method is fast enough to be used in interactive systems and to allow for user interaction with models consisting of up to 5000 particles.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

1.1. Motivation

Fluids (i.e. liquids and gases) play an important role in every day life. Examples for fluid phenomena are wind, weather, ocean waves, waves induced by ships or simply pouring of a glass of water. As simple and ordinary these phenomena may seem, as complex and difficult it is to simulate them. Even though Computational Fluid Dynamics (CFD) is a well established research area with a long history, there are still many open research problems in the field. The reason for the complexity of fluid behavior is the complex interplay of various phenomena such as convection, diffusion, turbulence and surface tension. Fluid phenomena are typically simulated off-line and then visualized in a second step e.g. in aerodynamics or optimization of turbines or pipes with the goal of being as accurate as possible.

Less accurate methods that allow the simulation of fluid effects in real-time open up a variety of new applications. In the fields mentioned above real-time methods help to test whether a certain concept is promising during the design phase. Other applications for real-time simulation techniques for fluids are medical simulators, computer games or any type of virtual environment.

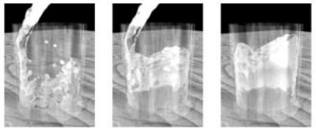


Figure 1: *Pouring water into a glass at 5 frames per second.*

1.2. Related Work

Computational Fluid Dynamics has a long history. In 1822 Claude Navier and in 1845 George Stokes formulated the famous Navier-Stokes Equations that describe the dynamics of fluids. Besides the Navier-Stokes equation which describes conservation of momentum, two additional equations namely a continuity equation describing mass conservation and a state equation describing energy conserva-

Annu. Rev. Astron. Astrophys. 1992. 30: 543–74

SMOOTHED PARTICLE HYDRODYNAMICS

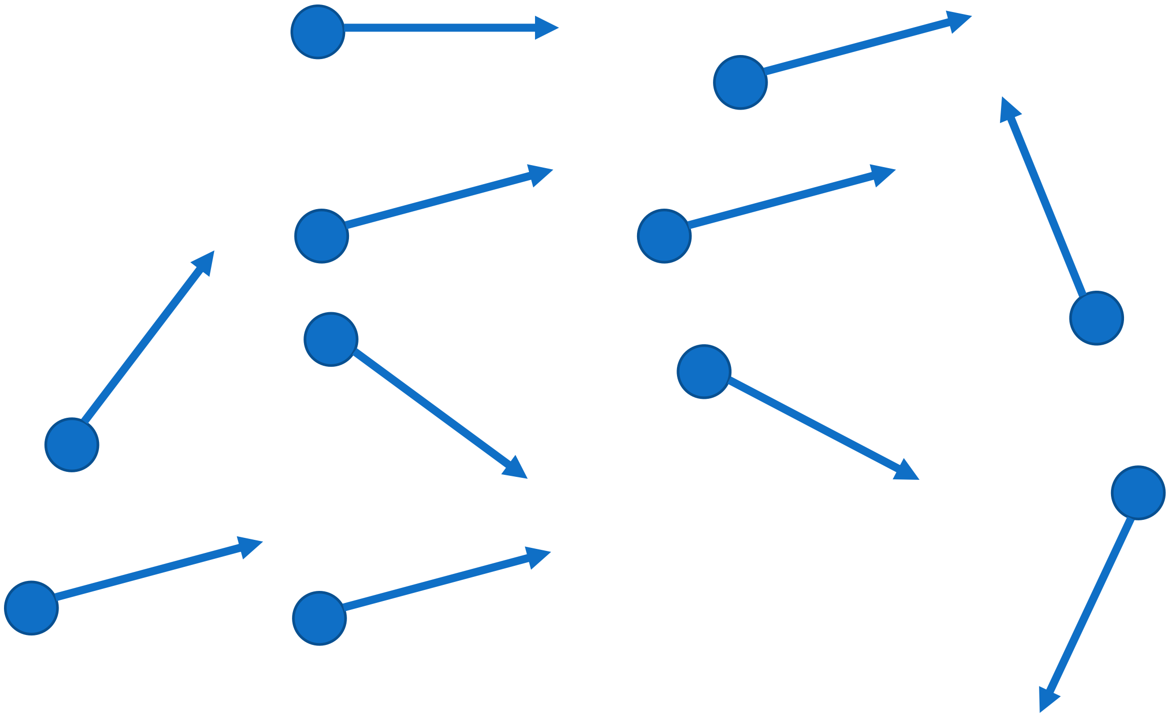
J. J. Monaghan

Department of Mathematics, Monash University, Clayton, Victoria 3168, Australia

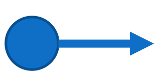
KEY WORDS: computational-fluid dynamics, numerical analysis

Matthias Müller, David Charypar, and Markus Gross. 2003. ***Particle-based fluid simulation for interactive applications***. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (SCA '03)

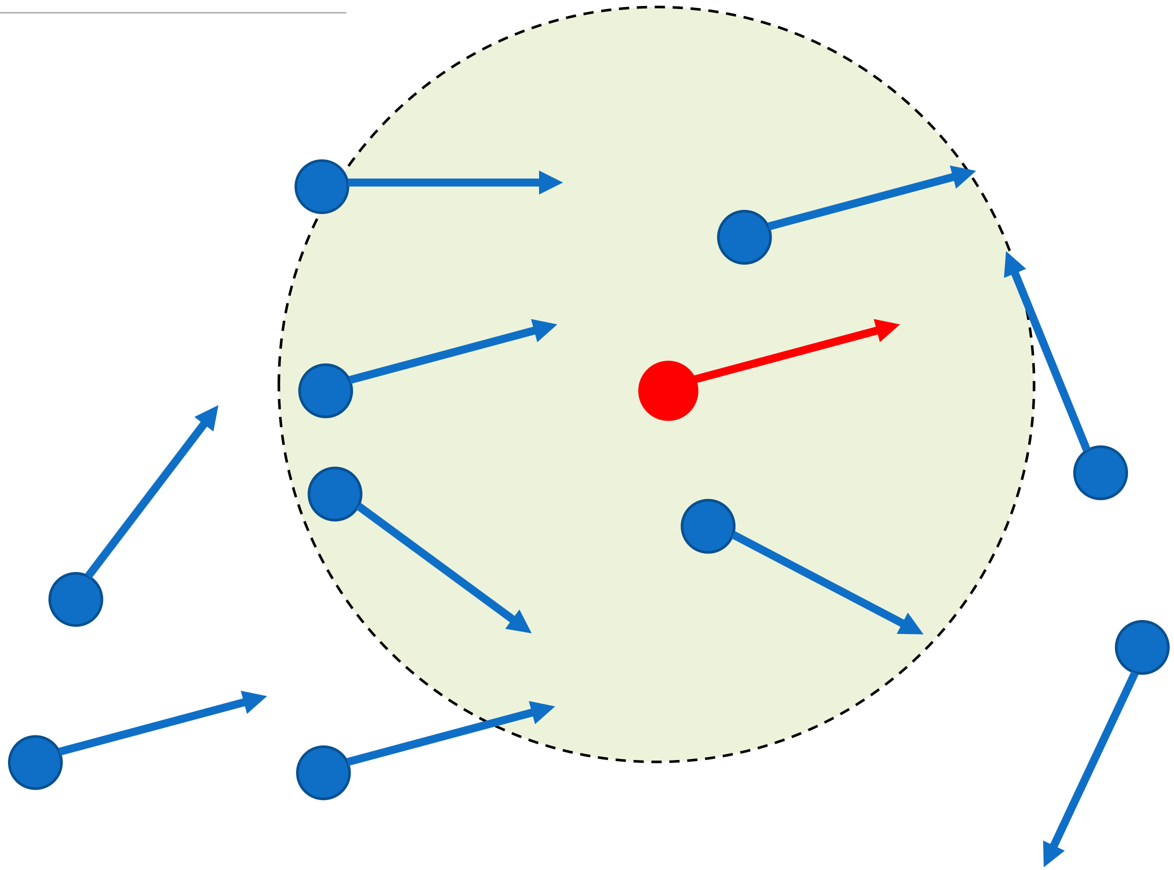
Particle System Again



$ma = f$

 mass, density, velocity, pressure, temperature …

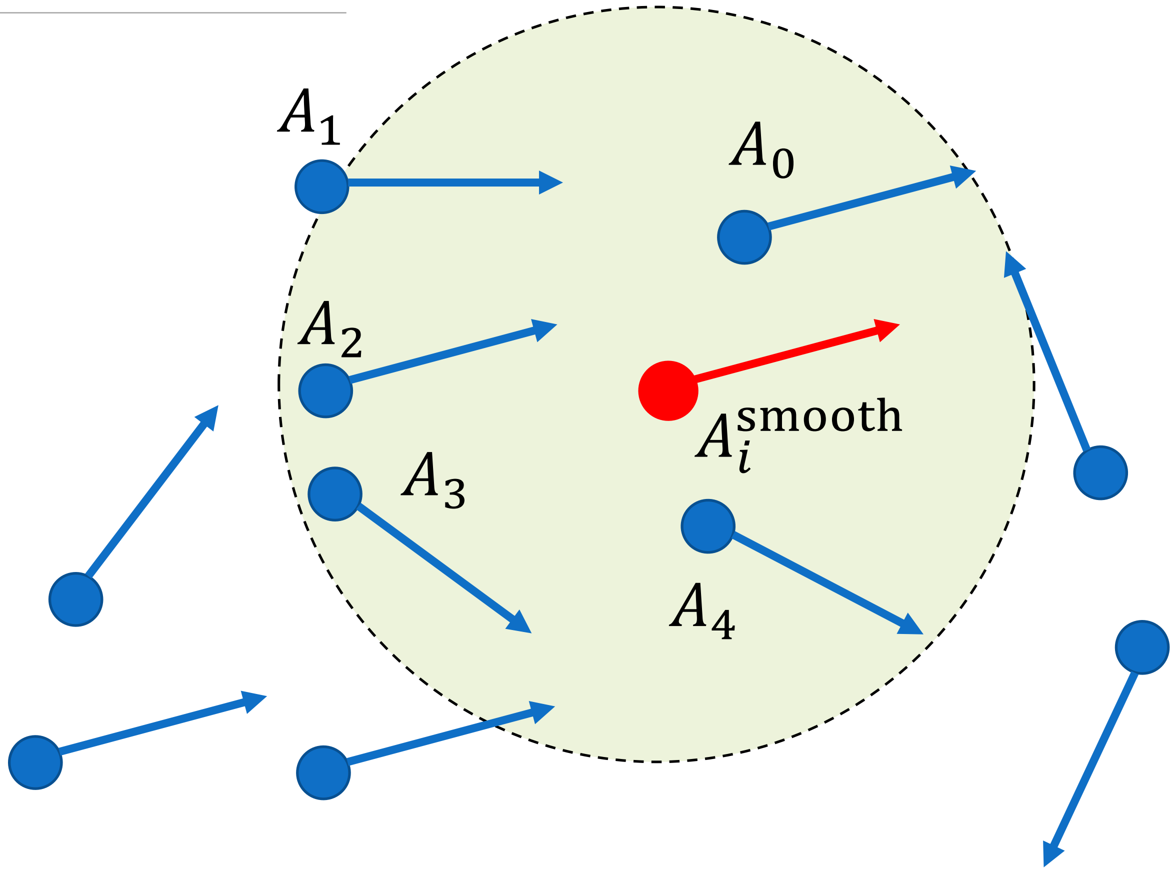
Smoothed Interpolation on Particles



$ma = f$

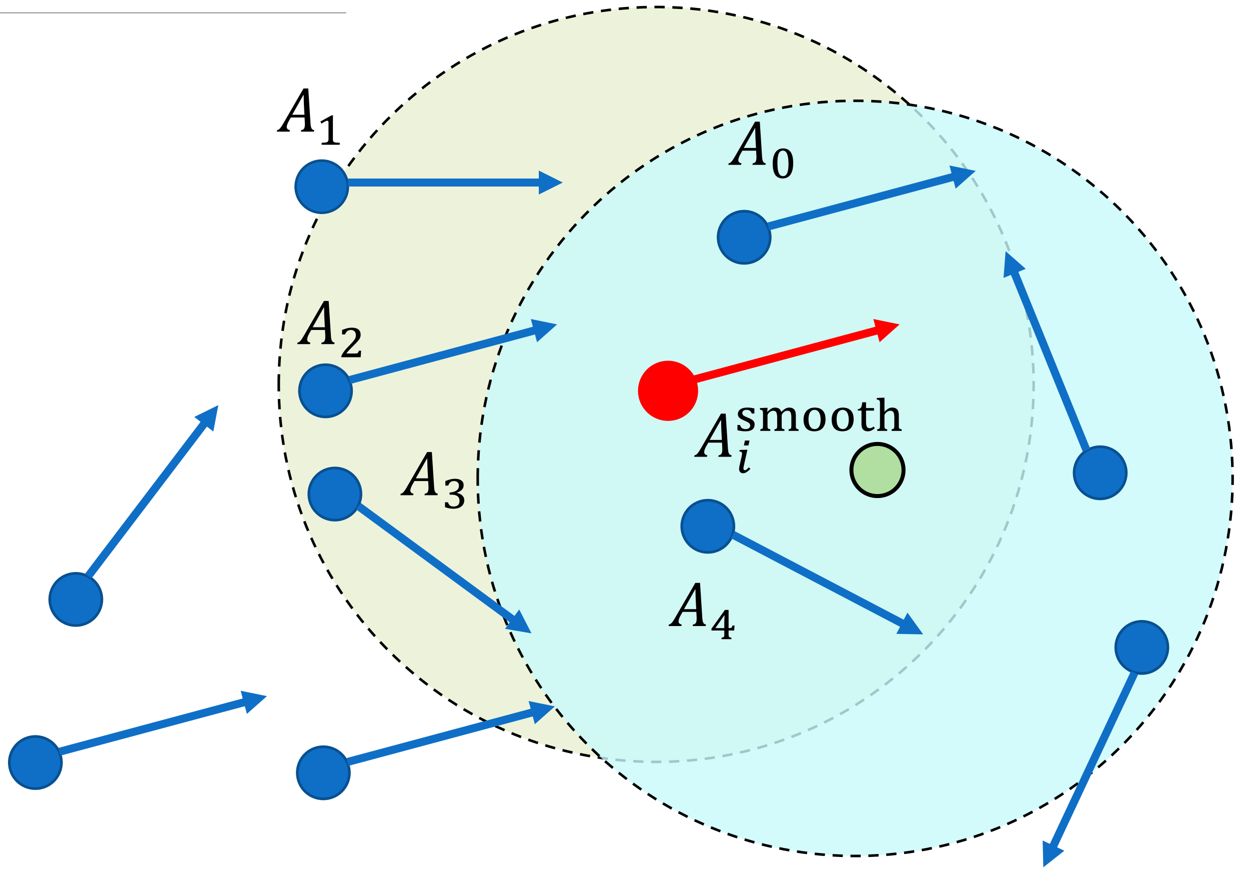
mass, density, velocity, pressure, temperature …

Smoothed Interpolation on Particles



$$A_i^{\text{smooth}} = \frac{1}{N}\sum_j A_j \quad \textbf{?}$$

Smoothed Interpolation on Particles



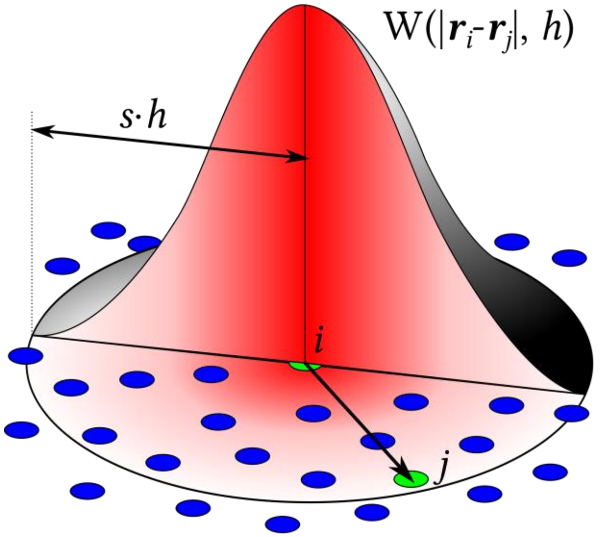
$$A_i^{\text{smooth}} = \frac{1}{N}\sum_j A_j \quad ?$$

Smoothed Interpolation on Particles

Interpolation with a Kernel Function

$$A^{\text{smooth}} = \int_V A(x')W(\|x' - x\|)dV$$

$$\int_V W(\|x' - x\|)dV = 1$$



Smoothed Interpolation on Particles

Interpolation with a Kernel Function

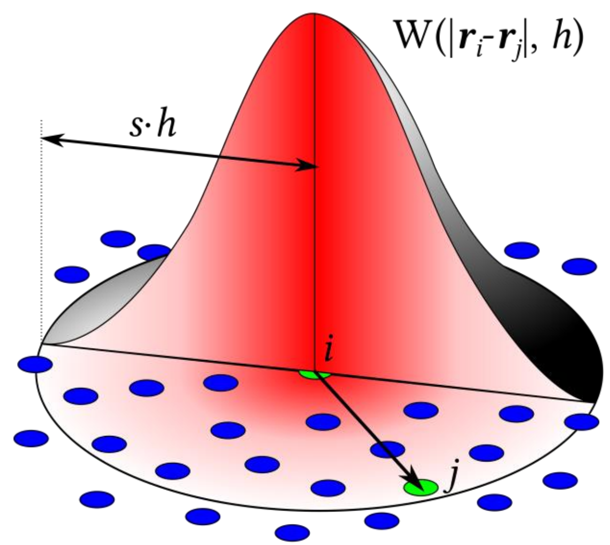
$$A^{\text{smooth}} = \int_V A(x')W(\|x' - x\|)dV$$

discretize

$$A_i^{\text{smooth}} = \sum_j V_j A(x_j) W(\|x_j - x_i\|)$$

What is V_j ??

$$\int_V W(\|x' - x\|)dV = 1$$



Smoothed Interpolation on Particles

$$A_i^{\text{smooth}} = \sum_j V_j A(x_j) W(\|x_j - x_i\|)$$

What is V_j ??

> Assume each particle has a fixed mass

> Then the density is determined by the number of nearby particles

$$\rho_i = \sum_j m_j W(\|x_j - x_i\|)$$

> Then the volume of the particle (note it is not constant!)

$$V_i = \frac{m_i}{\rho_i} = \frac{m_i}{\sum_j m_j W_{ij}}$$

Smoothed Interpolation on Particles

$$A_i^{\mathrm{smooth}} = \sum_j V_j A(x_j) W(\|x_j - x_i\|)$$

$$\Downarrow$$

$$A_i^{\mathrm{smooth}} = \sum_j m_j \frac{A(x_j)}{\rho_j} W(\|x_j - x_i\|)$$

$$\rho_i = \sum_j m_j W(\|x_j - x_i\|)$$

Differentiation

$$A_i^{\mathrm{smooth}} = \sum_j m_j \frac{A(x_j)}{\rho_j} W(\|x_j - x_i\|)$$

$$\Downarrow$$

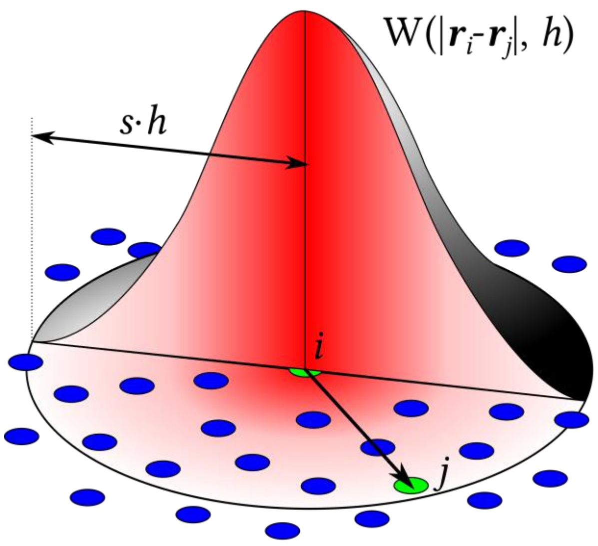
$$\nabla A_i^{\mathrm{smooth}} = \sum_j m_j \frac{A(x_j)}{\rho_j} \nabla W(\|x_j - x_i\|)$$

$$\Delta A_i^{\mathrm{smooth}} = \sum_j m_j \frac{A(x_j)}{\rho_j} \Delta W(\|x_j - x_i\|)$$

Kernel Functions

A kernel function needs to be:

- Smooth
- Symmetric
- Compactly supported



$$\int_V W(\|x' - x\|)dV = 1$$

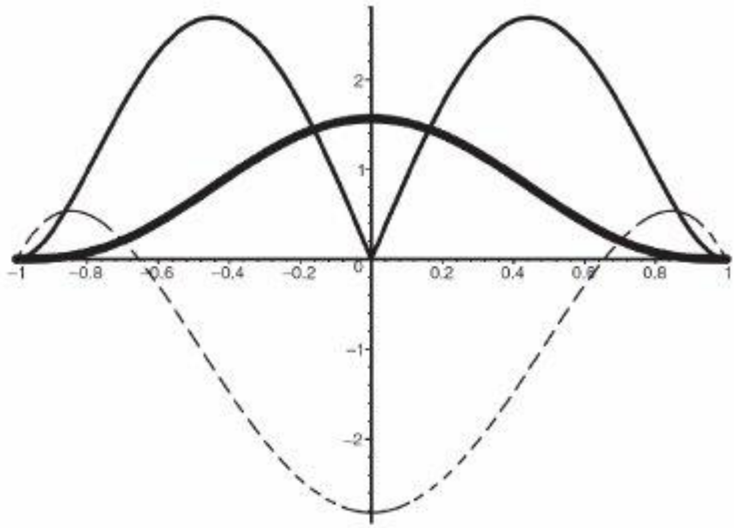
A popular choice: ***poly6*** kernel

$$W_{poly6}(r;h) = \begin{cases} \alpha_1 (h^2 - r^2)^3, & 0 \le r \le h \\ 0, & otherwise \end{cases}$$

$$\alpha_1 = \frac{4}{\pi h^8}, \frac{315}{64\pi h^9} \quad \text{for 2D, 3D}$$

Kernel Functions

A popular choice: ***poly6*** kernel



$$W_{poly6}(r;h) = \begin{cases} \alpha_1 (h^2 - r^2)^3, & 0 \le r \le h \\ 0, & otherwise \end{cases}$$

$$\alpha_1 = \frac{4}{\pi h^8}, \frac{315}{64\pi h^9}$$ for 2D, 3D

$$\nabla W_{poly6}(r;h) = \begin{cases} \alpha_2 (h^2 - r^2)^2 \boldsymbol{r}, & 0 \le r \le h \\ 0, & otherwise \end{cases}$$

$$\alpha_2 = -\frac{24}{\pi h^8}, -\frac{945}{32\pi h^9}$$ for 2D, 3D

Navier-Stokes Equations

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = g - \frac{1}{\rho} \nabla p + \nu \Delta u$$

$$\Downarrow$$

$$\rho \frac{Du}{Dt} = \rho g - \nabla p + \mu \Delta u$$

$\mu = \rho\nu$: dynamic viscosity coefficient

SPH Simulation:

$$\rho \frac{Du}{Dt} = \rho g - \nabla p + \mu \Delta u$$

$\mu = \rho \nu$: dynamic viscosity coefficient

$$Ma = F$$

Particle System!

Body Forces and External Forces

$$\rho \frac{Du}{Dt} = \boxed{\rho g} - \nabla p + \mu \Delta u$$

$$f_i^{\text{external}} = \rho g$$

Pressure Forces

$$\rho \frac{Du}{Dt} = \rho g - \boxed{\nabla p} + \mu \Delta u$$

$$f_i^{\text{pressure}} = -\nabla p(x_i) = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W\left(\|x_j - x_i\|\right)$$

* Note this is not symmetric…

* Consider two particles and $\nabla W(0) = 0$
The pressure forces are not balanced!

Pressure Forces

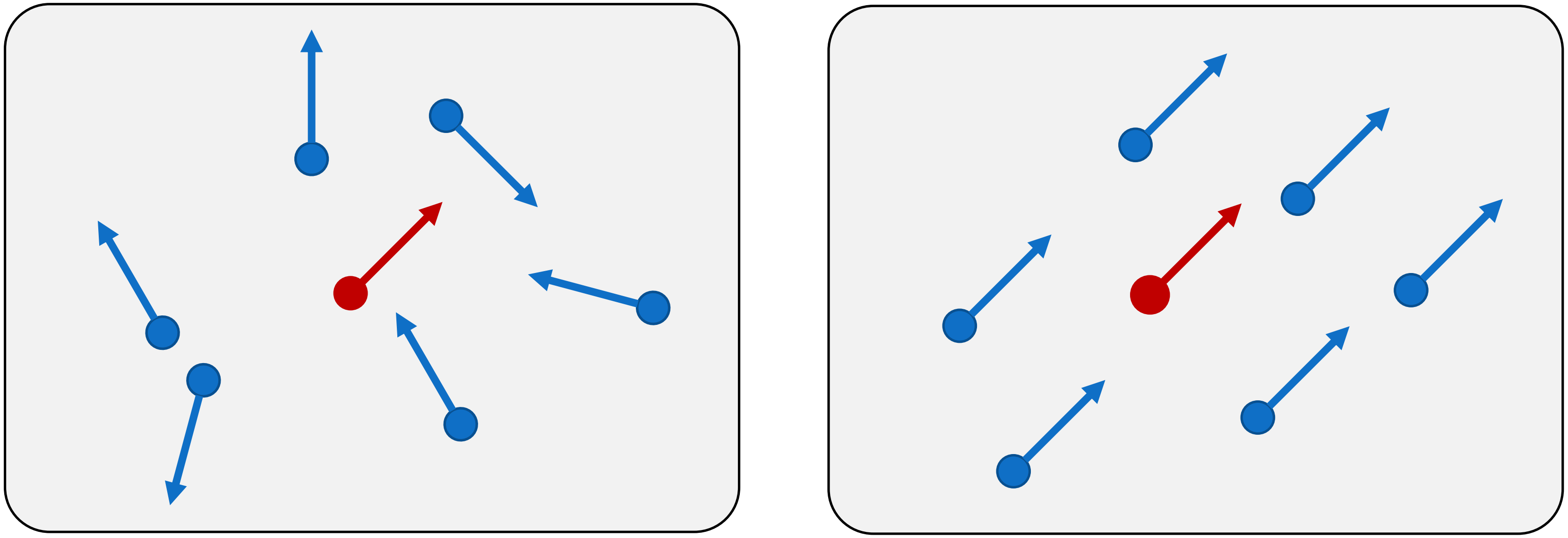
$$\rho \frac{Du}{Dt} = \rho g - \boxed{\nabla p} + \mu \Delta u$$

$$f_i^{\text{pressure}} = -\nabla p(x_i) = -\sum_j m_j \frac{p_j + p_i}{2\rho_j} \nabla W\left(\|x_j - x_i\|\right)$$

* A simple way of symmetrization

Viscosity

$$\rho \frac{Du}{Dt} = \rho g - \nabla p + \boxed{\mu \Delta u}$$



Viscosity

$$\rho \frac{Du}{Dt} = \rho g - \nabla p + \boxed{\mu \Delta u}$$

$$f_i^{\text{pressure}} = \mu \, \Delta u(x_i) = -\sum_j m_j \frac{u_i}{2\rho_j} \Delta W\left(\left\| x_j - x_i \right\|\right)$$

* Note this is not symmetric as well…

Viscosity

$$\rho \frac{Du}{Dt} = \rho g - \nabla p + \boxed{\mu \Delta u}$$

$$f_i^{\text{pressure}} = \mu \, \Delta u(x_i) = -\sum_j m_j \frac{u_j - u_i}{2\rho_j} \Delta W\left(\left\|x_j - x_i\right\|\right)$$

* Again, we need summarization

SPH: Put them together

$$\rho \frac{Du}{Dt} = \rho g - \nabla p + \mu \Delta u$$

$\mu = \rho\nu$: dynamic viscosity coefficient

For each time step:

For each particle i:

Neighbor Search

Estimate density ρ_i

Compute forces $\rho_i g, \nabla p_i, \Delta u_i$

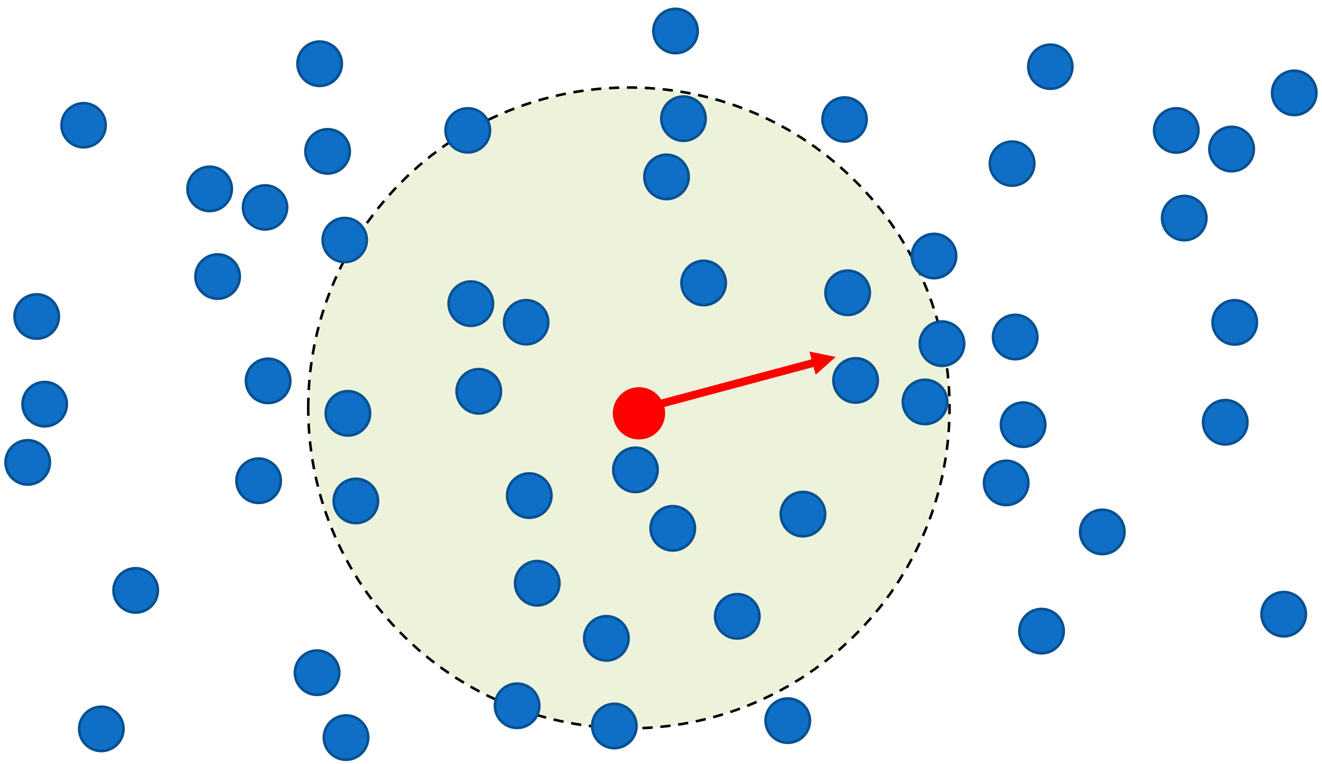
For each particle i:

Update $u_i = u_i + \delta t\ f/\rho$

Update $x_i = x_i + \delta t\ u_i$

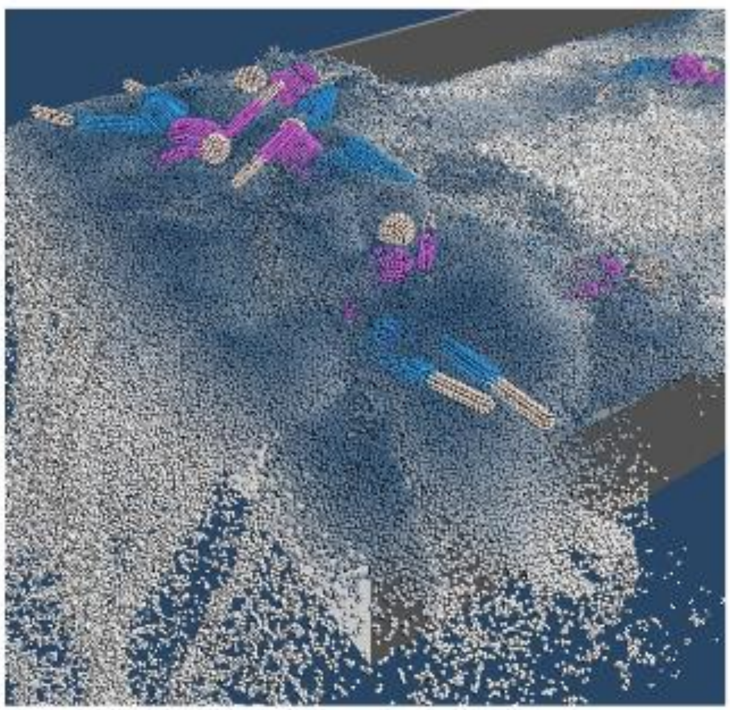
What's the bottleneck?

- Exhaustive Neighborhood Search
  - Search over every particle pair? $O(N^2)$
  - 10M particles means: 100 Trillion pairs…

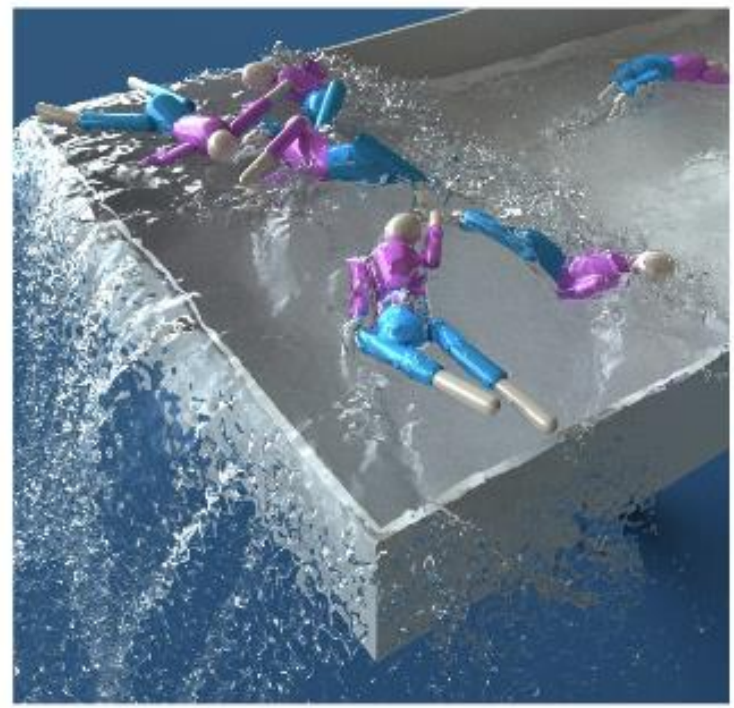


Fluid Display

- Need to reconstruct the water surface from particles!



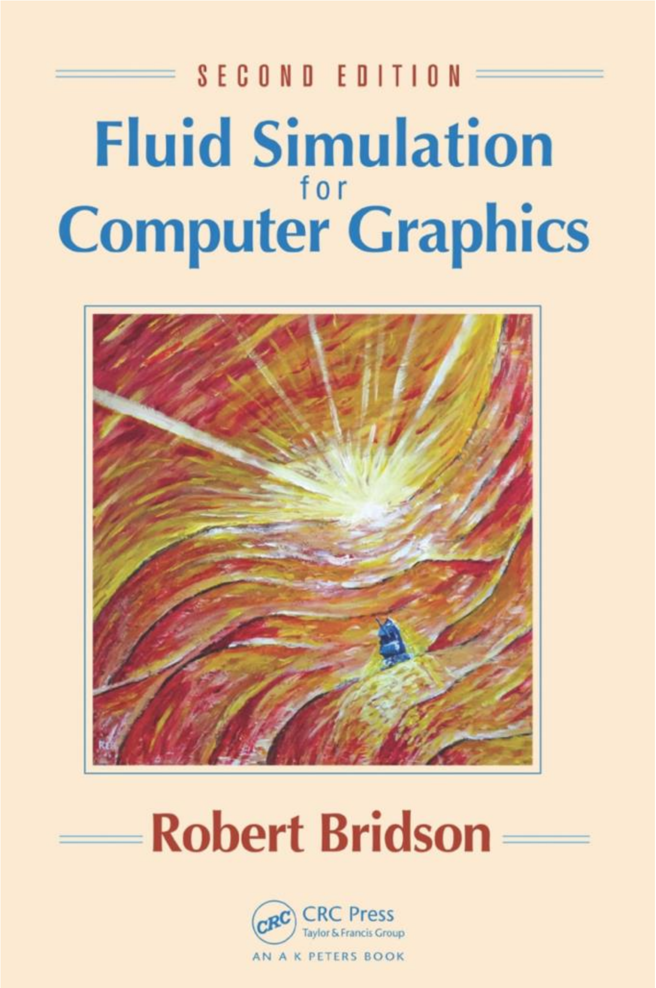
representation



typical visualization

Outline

- Navier-Stokes Equations
- Lagrangian and Eulerian Viewpoints
- Numerical methods
- Eulerian methods
  - Smoke
  - Water
- Lagrangian methods
  - Smoothed Particle Hydrodynamics (SPH) method



Oscars 2015 Tech: Robert Bridson

Several slides are modified from GAMES-103

Questions?