

# Fluid dynamics

Thomas Sangild

GPGPU lecture

University of Aarhus

25/11-2004

## A talk on the article

- Handout

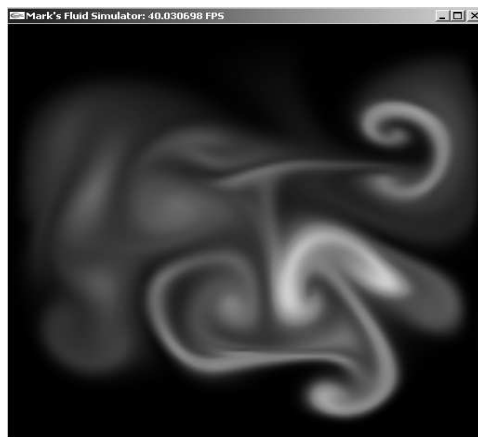
- Harris MJ. Fast Fluid Dynamics Simulation on the GPU. Chapter 38. GPU Gems. 2004.

- [http://download.nvidia.com/developer/SDK/Individual\\_Samples/DEMOS%5COpenGL%5Csrc%5Cgpgpu\\_fluid%5Cdocs%5CGPU\\_Gems\\_Fluids\\_Chapter.pdf](http://download.nvidia.com/developer/SDK/Individual_Samples/DEMOS%5COpenGL%5Csrc%5Cgpgpu_fluid%5Cdocs%5CGPU_Gems_Fluids_Chapter.pdf)

## Recommended additional reading

- Stam J. Stable Fluids. Siggraph 1999.
  - <http://www.dgp.toronto.edu/people/stam/reality/Research/pdf/ns.pdf>

## Fluid Simulator Demo (more)



Source: Mark Harris / GPGPU tutorial@Siggraph 04

## Project work

- Allan Rasmusson
- Søren Skov
- Niels Jeppe Anders Nielsen
- Nicolai Lundgaard

## Vector field operations

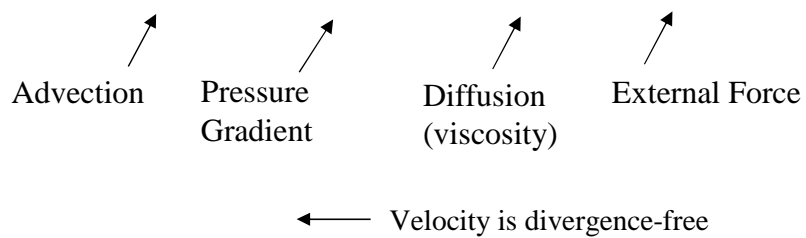
- Pressure (scalar field)  $p: \mathbb{R}^m \rightarrow \mathbb{R}; m \in \{2,3\}$
- Velocity (vector field)  $\mathbf{u}: \mathbb{R}^m \rightarrow \mathbb{R}^m; m \in \{2,3\}$

**Table 38-1.** Vector Calculus Operators Used in Fluid Simulation

| Operator   | Definition  | Finite Difference Form  |
|------------|---|---|
| Gradient   | $\nabla p = \left( \frac{\partial p}{\partial x}, \frac{\partial p}{\partial y} \right)$  | $\frac{p_{i+1,j} - p_{i-1,j}}{2\delta x}, \frac{p_{i,j+1} - p_{i,j-1}}{2\delta y}$                              |
| Divergence | $\nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$ | $\frac{u_{i+1,j} - u_{i-1,j}}{2\delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{2\delta y}$                             |
| Laplacian  | $\nabla^2 p = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}$      | $\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{(\delta x)^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{(\delta y)^2}$ |

# Navier-Stokes Equations

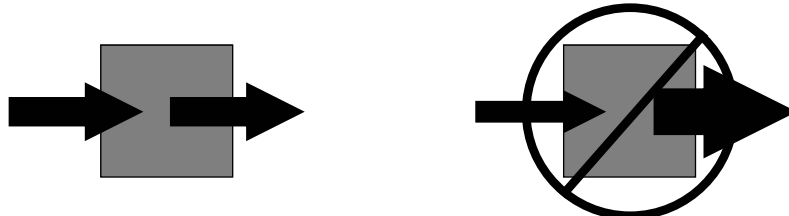
- Describe flow of an incompressible fluid



Source: Mark Harris / GPGPU tutorial@Siggraph 04

# Divergence-Free?

- In any element of fluid, inward velocity is balanced by outward velocity
  - No sources or sinks
- Ensures mass / momentum conservation



Source: Mark Harris / GPGPU tutorial@Siggraph 04

# The Helmholtz-Hodge decomposition

## Helmholtz-Hodge Decomposition Theorem

A vector field  $w$  on  $D$  can be uniquely decomposed in the form:

$$w = u + \nabla p, \quad (7)$$

where  $u$  has zero divergence and is parallel to  $\partial D$ ; that is,  $u \cdot n = 0$  on  $\partial D$ .

Proof in: Chorin and Marsden. A mathematical introduction to fluid mechanics. 1993.

From: Harris MJ. Fast Fluid Dynamics Simulation on the GPU. Chapter 38. GPU Gems. 2004.

# Realizations

## First Realization

Solving the Navier-Stokes equations involves three computations to update the velocity at each time step: advection, diffusion, and force application. The result is a new velocity field,  $w$ , with *nonzero* divergence. But the continuity equation requires that we end each time step with a divergence-free velocity. Fortunately, the Helmholtz-Hodge Decomposition Theorem tells us that the divergence of the velocity can be corrected by subtracting the gradient of the resulting pressure field:

$$u = w - \nabla p. \quad (8)$$

From: Harris MJ. Fast Fluid Dynamics Simulation on the GPU. Chapter 38. GPU Gems. 2004.

# Realizations

## Second Realization

The theorem also leads to a method for computing the pressure field. If we apply the divergence operator to both sides of Equation 7, we obtain:

$$\nabla \cdot \mathbf{w} = \nabla \cdot (\mathbf{u} + \nabla p) = \nabla \cdot \mathbf{u} + \nabla^2 p. \quad (9)$$

But since Equation 2 enforces that  $\nabla \cdot \mathbf{u} = 0$ , this simplifies to:

$$\nabla^2 p = \nabla \cdot \mathbf{w}, \quad (10)$$

which is a Poisson equation (see Section 38.2.3) for the pressure of the fluid, sometimes called the *Poisson-pressure equation*. This means that after we arrive at our divergent velocity,  $\mathbf{w}$ , we can solve Equation 10 for  $p$ , and then use  $\mathbf{w}$  and  $p$  to compute the new divergence-free field,  $\mathbf{u}$ , using Equation 8. We'll return to this later.

From: Harris MJ. Fast Fluid Dynamics Simulation on the GPU. Chapter 38. GPU Gems. 2004.

# Projection operator

- Define a projection operator  $\mathcal{P}$  that projects a vector field  $\mathbf{w}$  onto divergence-free component  $\mathbf{u}$ .
- $\mathcal{P}\mathbf{w} = \mathcal{P}\mathbf{u} + \mathcal{P}(\nabla p)$ 
  - Since by definition  $\mathcal{P}\mathbf{w} = \mathcal{P}\mathbf{u} = \mathbf{u}$  then  $\mathcal{P}(\nabla p) = 0$
- And then

$$\mathbb{P} \frac{\partial \mathbf{u}}{\partial t} = \mathbb{P} \left( -(\mathbf{u} \cdot \nabla) \mathbf{u} - \frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{F} \right)$$

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbb{P} \left( -(\mathbf{u} \cdot \nabla) \mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{F} \right)$$

## Algorithm

- Break it down [Stam 2000]:
  - Add forces:
  - Advect:
  - Diffuse:
  - Solve for pressure:
  - Subtract pressure gradient:

Source: Mark Harris / GPGPU tutorial@Siggraph 04

## Algorithm

- Break it down [Stam 2000]:
  - Add forces:

Source: Mark Harris / GPGPU tutorial@Siggraph 04

## Add Forces

- Scale force by time step, add to velocity
- In demo, we just “splat”
  - Color of splat encodes direction and strength of force (determined by mouse motion)
  - Simple fragment program adds gaussian splat to velocity texture

Source: Mark Harris / GPGPU tutorial@Siggraph 04

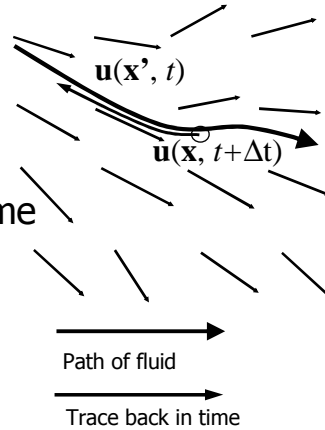
## Algorithm

- Break it down [Stam 2000]:
  - Add forces:
  - Advect:

Source: Mark Harris / GPGPU tutorial@Siggraph 04

## Advection

- Advection: quantities in a fluid are carried along by its velocity
- Want velocity at position  $x$  at new time  $t + \Delta t$
- Follow velocity field back in time from  $x$  :  $(x - w_1 \Delta t)$ 
  - Like tracing particles!
  - Simple in a fragment program



Source: Mark Harris / GPGPU tutorial@Siggraph 04

## Algorithm

- Break it down [Stam 2000]:
  - Add forces:
  - Advect:
  - Diffuse:

Source: Mark Harris / GPGPU tutorial@Siggraph 04

## Numerical integration (1)

- Explicit time step
  - Unstable for large timesteps/viscosity

$$\frac{dw_2}{dt} = \frac{w_2(x, t+h) - w_2(x, t)}{h} = \nu \nabla^2 w_2(x, t) \Rightarrow$$

$$\underbrace{w_2(x, t+h)}_{\mathbf{w}_3(\mathbf{x})} = w_2(x, t) + h\nu \nabla^2 w_2(x, t)$$

## Numerical integration (2)

- Implicit time step
  - Stable for large timesteps

$$\frac{dw_2}{dt} = \frac{w_2(x, t+h) - w_2(x, t)}{h} = \nu \nabla^2 w_2(x, t+h) \Rightarrow$$

$$w_2(x, t+h) = w_2(x, t) + h\nu \nabla^2 w_2(x, t+h) \Rightarrow$$

$$(I - h\nu \nabla^2) \underbrace{w_2(x, t+h)}_{\mathbf{w}_3(\mathbf{x})} = w_2(x, t)$$

## Viscous Diffusion

- Viscous fluid exerts drag on itself
  - Causes diffusion of velocity
- Implicit, discrete form of
  - Explicit form is unstable
- Solution is just like the next step
  - Simpler to explain the next step...

Source: Mark Harris / GPGPU tutorial@Siggraph 04

## Algorithm

- Break it down [Stam 2000]:
  - Add forces:
  - Advect:
  - Diffuse:
  - Solve for pressure:

Source: Mark Harris / GPGPU tutorial@Siggraph 04

# Poisson-Pressure Solution

- Poisson Equation
  - Discretize, solve using iterative solver (relaxation)
  - Jacobi, Gauss-Seidel, Multigrid, etc.
    - Jacobi easy on GPU, the rest are trickier
    - Demo uses Jacobi iteration (15-20 iters.)
  - Boils down to repeated evaluation of:

|          |                       |
|----------|-----------------------|
| $\delta$ | = grid spacing        |
| $u, v$   | = components of $w_3$ |
| $i, j$   | = grid coordinates    |
| $n$      | = solution iteration  |

Source: Mark Harris / GPGPU tutorial@Siggraph 04

# Algorithm

- Break it down [Stam 2000]:
  - Add forces:
  - Advect:
  - Diffuse:
  - Solve for pressure:
  - Subtract pressure gradient:

Source: Mark Harris / GPGPU tutorial@Siggraph 04

## Subtract Pressure Gradient

- Last computation of the time step
  - $u$  is now a divergence-free velocity field
- Very simple fragment program

|            |                                |
|------------|--------------------------------|
| $\delta$   | = grid spacing                 |
| $i, j$     | = grid coordinates             |
| $u, v$     | = components of $\mathbf{u}$   |
| $u_3, v_3$ | = components of $\mathbf{w}_3$ |
| $p$        | = pressure                     |

Source: Mark Harris / GPGPU tutorial@Siggraph 04

- To be continued on tuesday...