GAMES103: Intro to Physics-Based Animation

Eulerian Fluids

Huamin Wang

Dec 2021

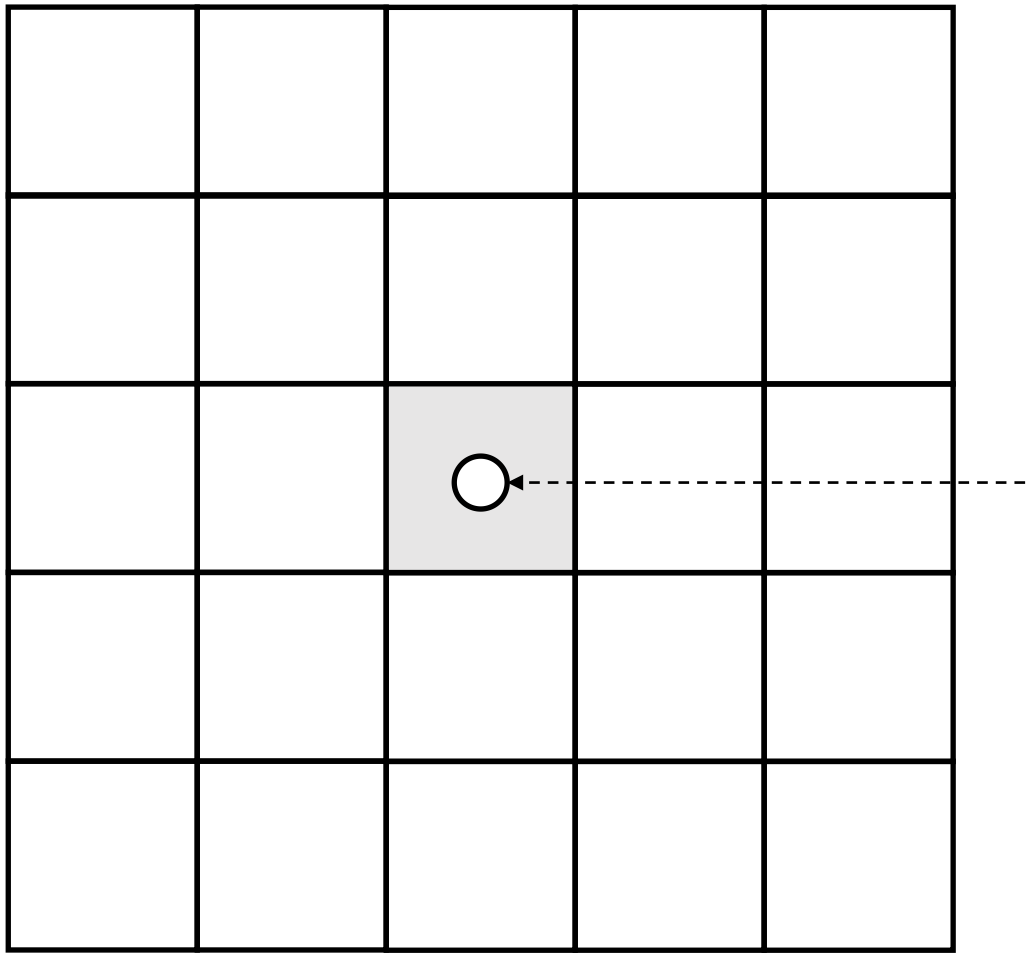
Topics for the Day

- A Grid Representation and Finite Differencing
- Incompressible, Viscous Navier Stokes' equations
- Air and liquid

A Grid Representation and Finite Differencing

A Regular Grid Representation

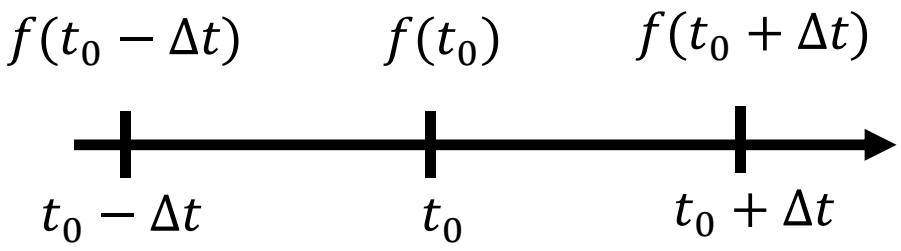
In an Eulerian grid representation, we store physical quantities over a grid. The most popular grid is a regular one.



- Scalars
  - Density/color
  - Pressure
  - Temperature
  - …
- Vectors
  - Velocities
- The grid can be viewed as a scalar/vector function, also known as a scalar/vector field.

Central Differencing

Recall that we can use central differencing to estimate the derivative.



$$f(t_0 + \Delta t) = f(t_0) + \Delta t \frac{df(t_0)}{dt} + \frac{\Delta t^2 d^2 f(t_0)}{2 \quad dt^2} + \ldots$$

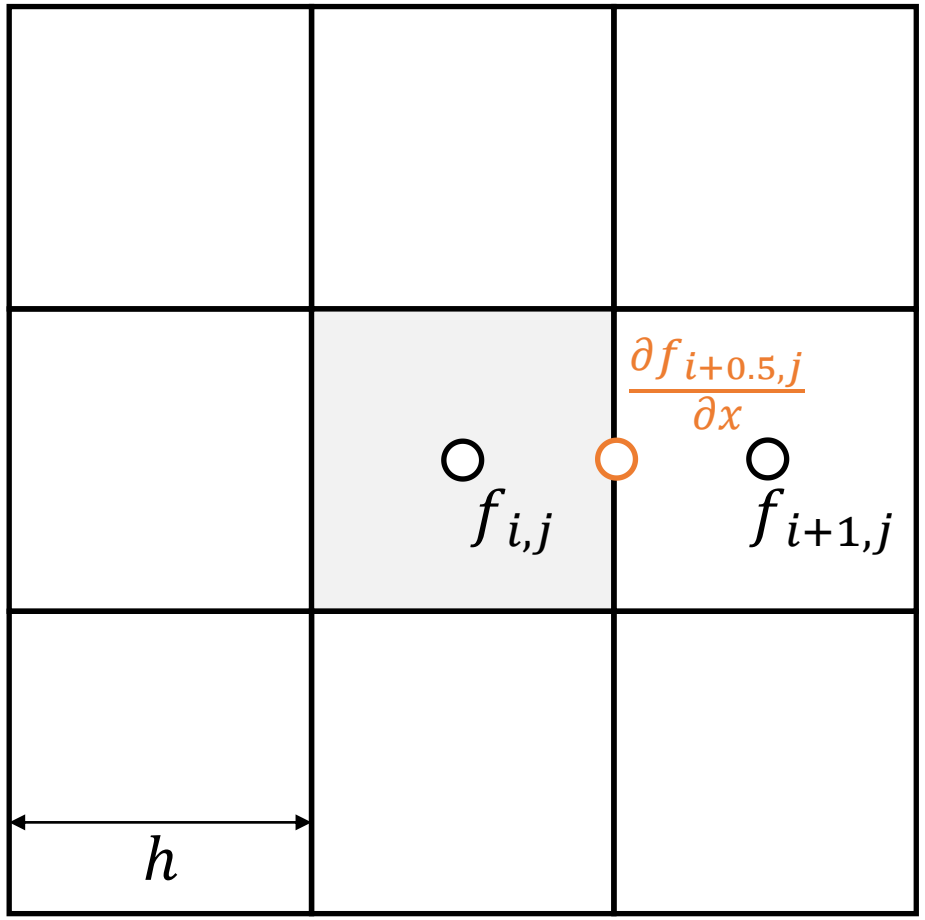
$$f(t_0 - \Delta t) = f(t_0) - \Delta t \frac{df(t_0)}{dt} + \frac{\Delta t^2 d^2 f(t_0)}{2 \quad dt^2} + \ldots$$

Central differencing (second-order)

$$\frac{df(t_0)}{dt} \approx \frac{f(t_0 + \Delta t) - f(t_0 - \Delta t)}{2\Delta t}$$

Finite Differencing on Grid

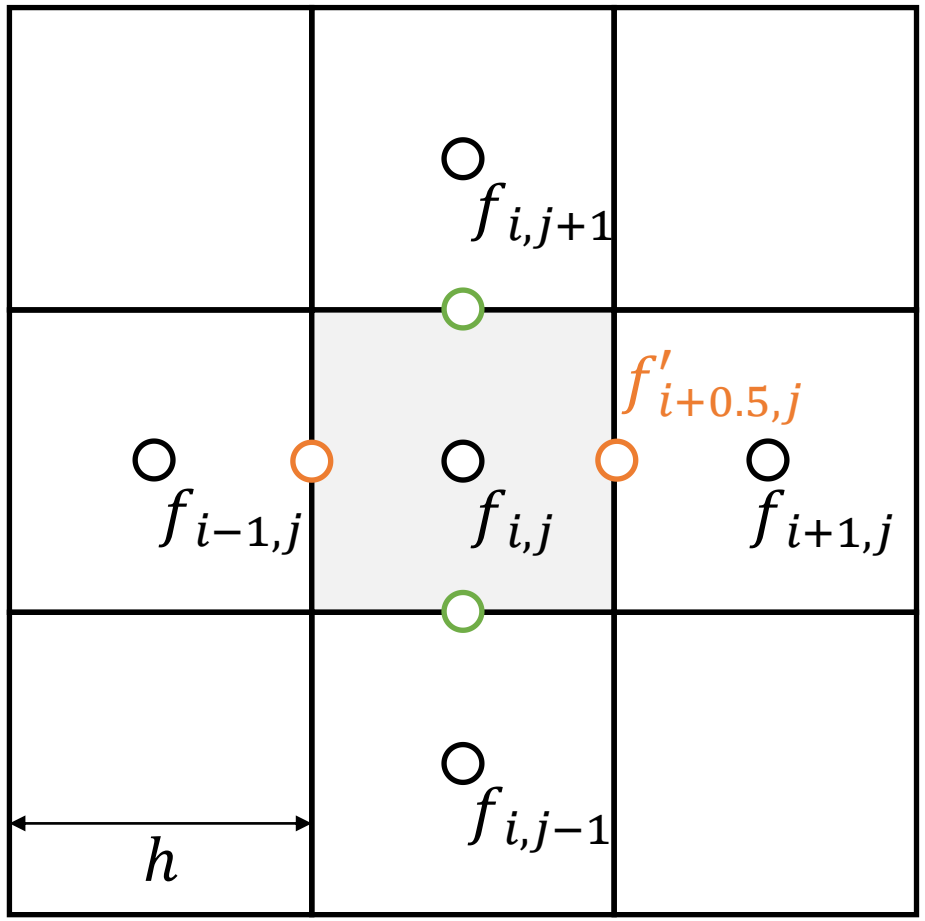
The grid is very friendly with central differencing.



$$\frac{\partial f_{i+0.5,j}}{\partial x} \approx \frac{f_{i+1,j} - f_{i,j}}{h}$$

Finite Differencing on Grid

The grid is very friendly with central differencing.



$$\frac{\partial f_{i-0.5,j}}{\partial x} \approx \frac{f_{i,j} - f_{i-1,j}}{h}$$

$$\frac{\partial f_{i+0.5,j}}{\partial x} \approx \frac{f_{i+1,j} - f_{i,j}}{h}$$

$$\frac{\partial^2 f_{i,j}}{\partial x^2} \approx \frac{\frac{\partial f_{i+0.5,j}}{\partial x} - \frac{\partial f_{i-0.5,j}}{\partial x}}{h} \approx \frac{f_{i-1,j} + f_{i+1,j} - 2f_{i,j}}{h^2}$$

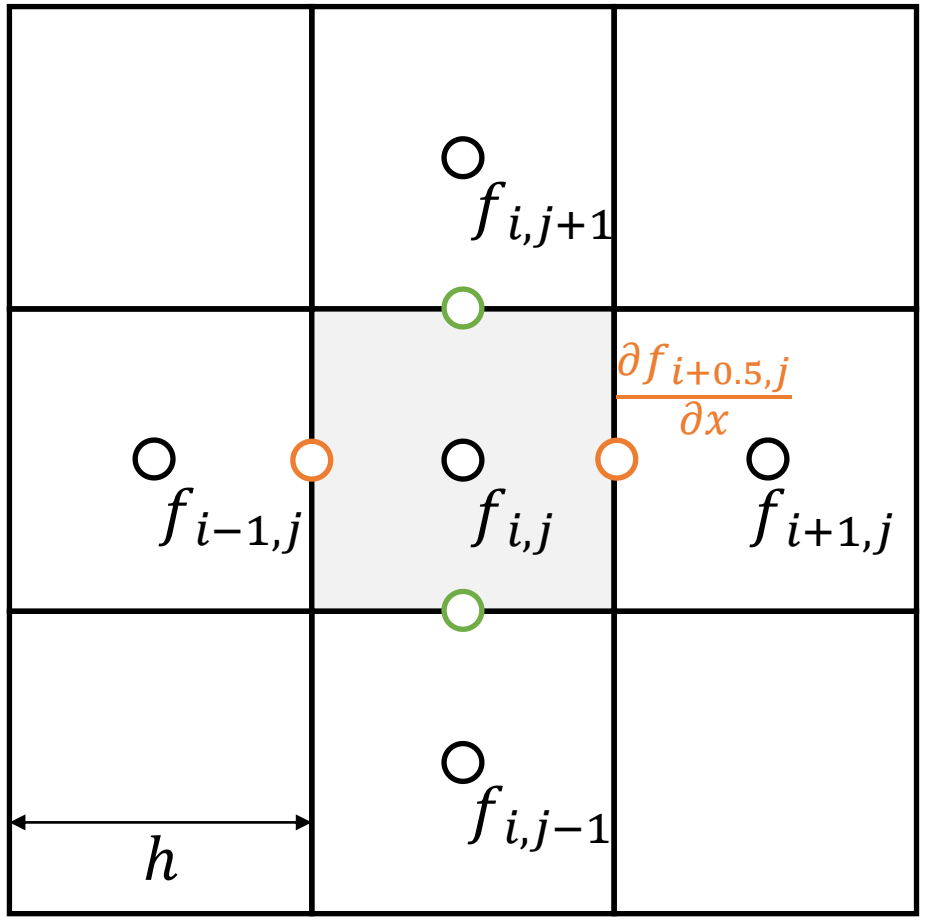
$$\frac{\partial f_{i,j-0.5}}{\partial y} \approx \frac{f_{i,j} - f_{i,j-1}}{h}$$

$$\frac{\partial f_{i,j+0.5}}{\partial y} \approx \frac{f_{i,j+1} - f_{i,j}}{h}$$

$$\frac{\partial^2 f_{i,j}}{\partial y^2} \approx \frac{\frac{\partial f_{i,j+0.5}}{\partial y} - \frac{\partial f_{i,j-0.5}}{\partial y}}{h} \approx \frac{f_{i,j-1} + f_{i,j+1} - 2f_{i,j}}{h^2}$$

Discretized Laplacian

We can then obtain the discretized Laplacian operator on grid.



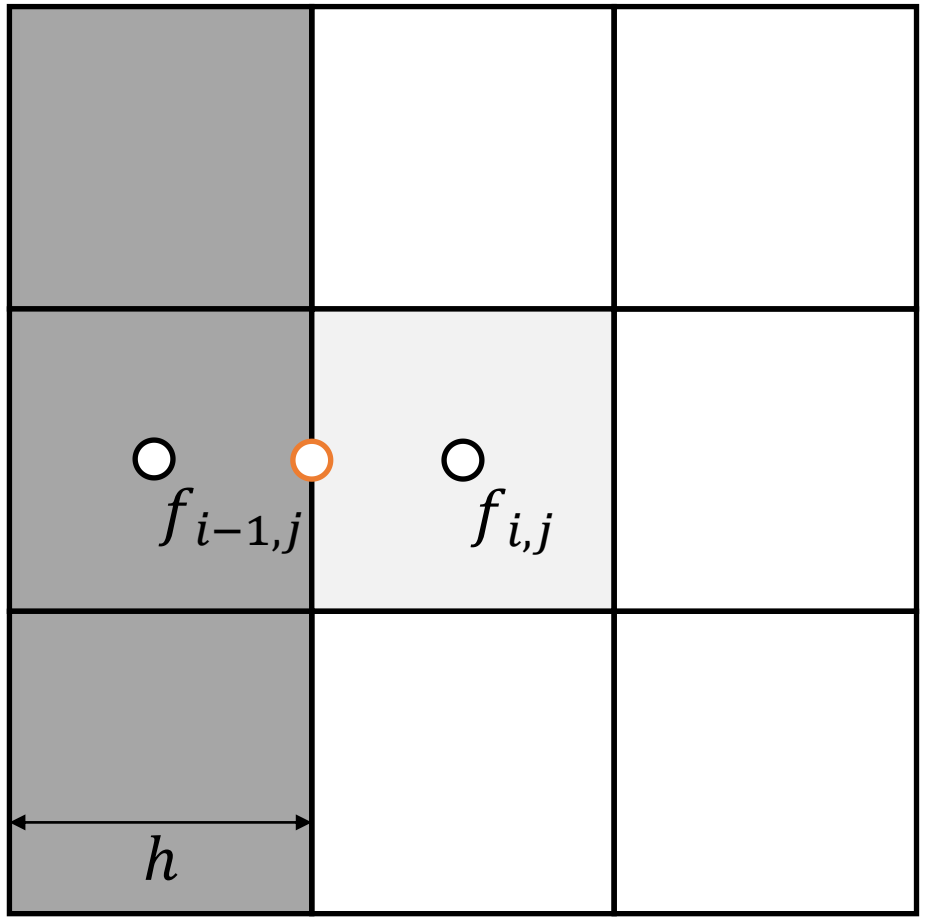
$$\frac{\partial^2 f_{i,j}}{\partial x^2} \approx \frac{\frac{\partial f_{i-0.5,j}}{\partial x} - \frac{\partial f_{i+0.5,j}}{\partial x}}{h} \approx \frac{f_{i-1,j} + f_{i+1,j} - 2f_{i,j}}{h^2}$$

$$\frac{\partial^2 f_{i,j}}{\partial y^2} \approx \frac{\frac{\partial f_{i,j+0.5}}{\partial y} - \frac{\partial f_{i,j-0.5}}{\partial y}}{h} \approx \frac{f_{i,j-1} + f_{i,j+1} - 2f_{i,j}}{h^2}$$

$$\Delta f_{i,j} = \frac{\partial^2 f_{i,j}}{\partial x^2} + \frac{\partial^2 f_{i,j}}{\partial y^2} \approx \frac{f_{i-1,j} + f_{i+1,j} + f_{i,j-1} + f_{i,j+1} - 4f_{i,j}}{h^2}$$

Boundary Conditions

The boundary condition specifies $f_{i-1,j}$ if it's outside.



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

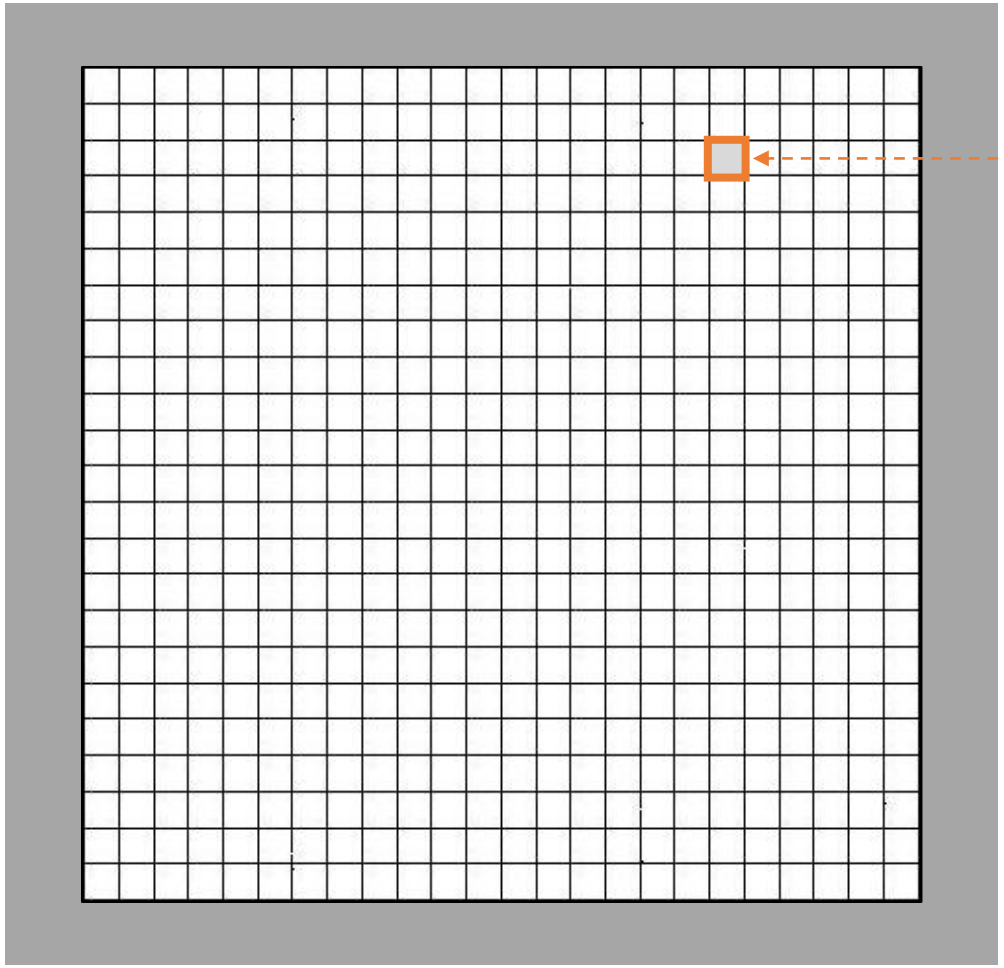
$$\Delta f_{i,j} \approx \frac{C + f_{i+1,j} + f_{i,j-1} + f_{i,j+1} - 4f_{i,j}}{h^2}$$

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

$$\Delta f_{i,j} \approx \frac{f_{i+1,j} + f_{i,j-1} + f_{i,j+1} - 3f_{i,j}}{h^2}$$

Example: Laplace's Equation

With the grid, we can discretize Laplace's equation: $\Delta f = 0$.



$$f_{i-1,j} + f_{i+1,j} + f_{i,j-1} + f_{i,j+1} - 4f_{i,j} = 0$$

$c + c + c + c - 4c$

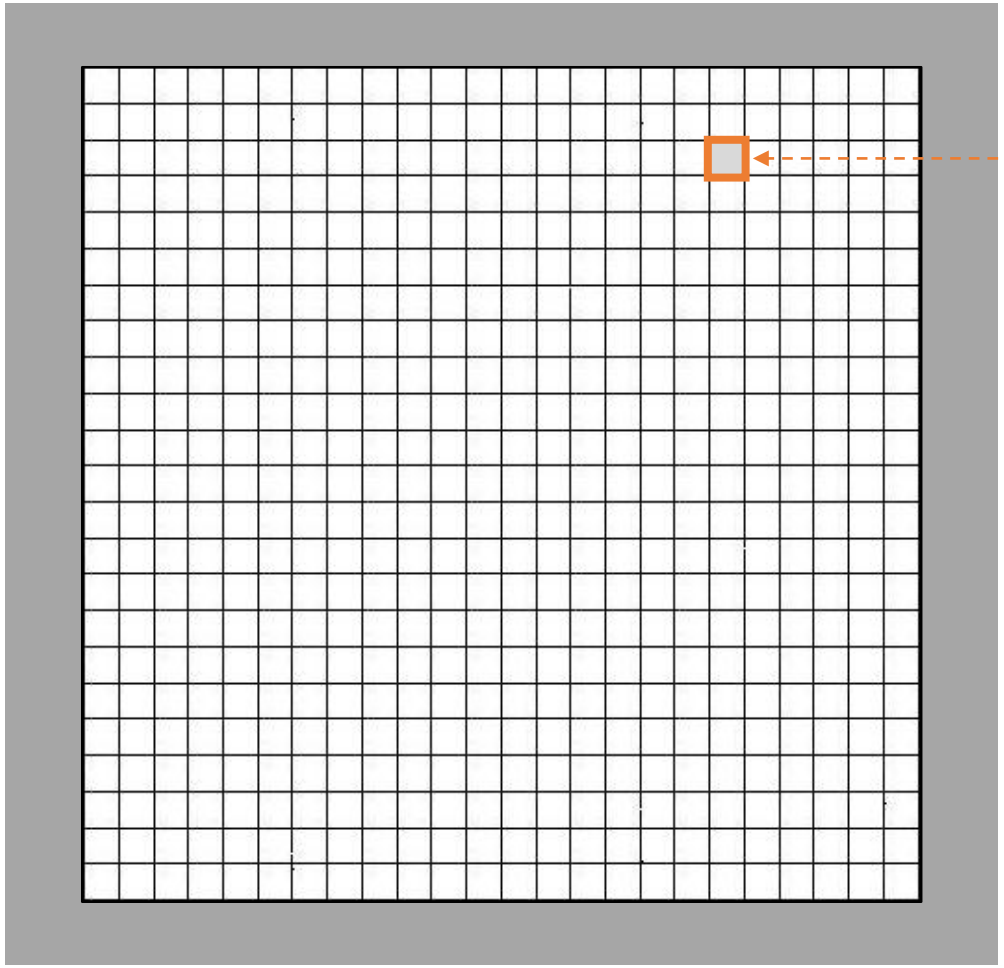
with boundary conditions

Note: At least one condition should be ~~Neumann~~ Dirichlet. (Why?)

singular 无穷解

Example: Laplace's Equation

To solve Laplace's equation, we can use the Jacobi method.



$$f_{i-1,j} + f_{i+1,j} + f_{i,j-1} + f_{i,j+1} - 4f_{i,j} = 0$$

The Jacobi Method

Initialize f

For $k = 0 \ldots K$

  For $i,\ j$

    $f_{ij}^{\text{new}} \leftarrow f_{ij} + \alpha\left(4f_{ij} - f_{i-1,j} - f_{i+1,j} - f_{i,j-1} - f_{i,j+1}\right)$

  For $i,\ j$

    $f_{ij} \leftarrow f_{ij}^{\text{new}}$

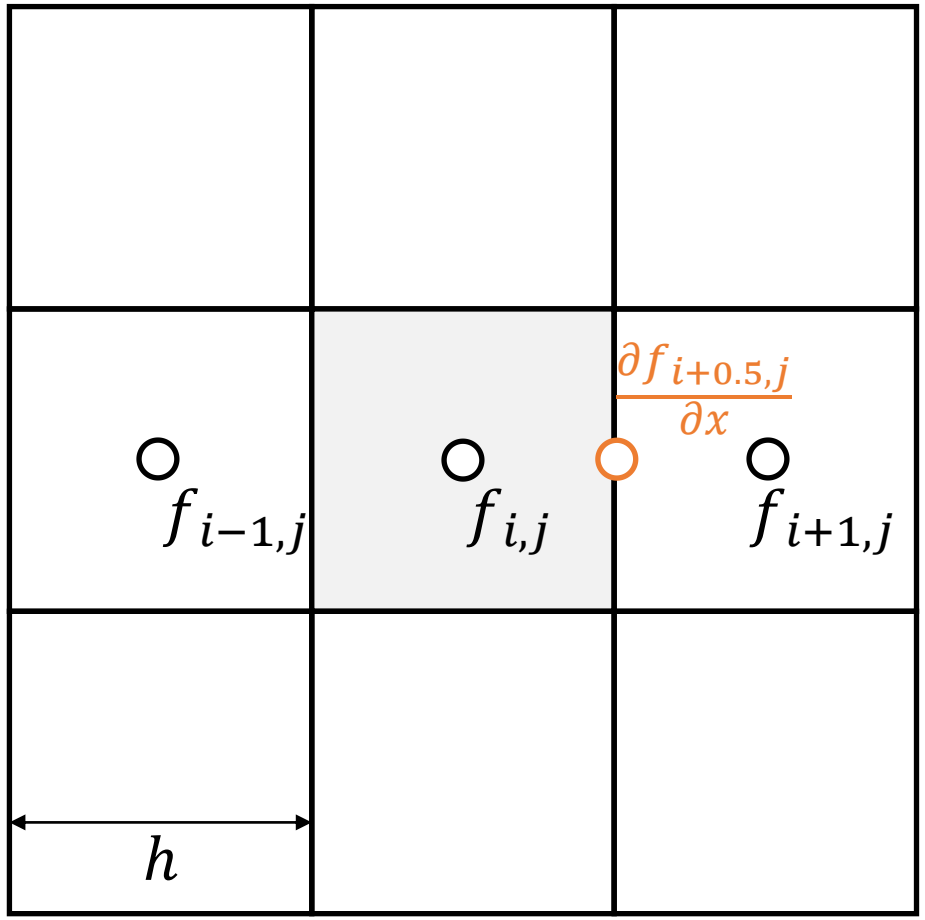
Laplacian Smoothing

Diffusion

The process of applying Laplacian smoothing is called diffusion. See in-class demo.

Problem with Central Differencing

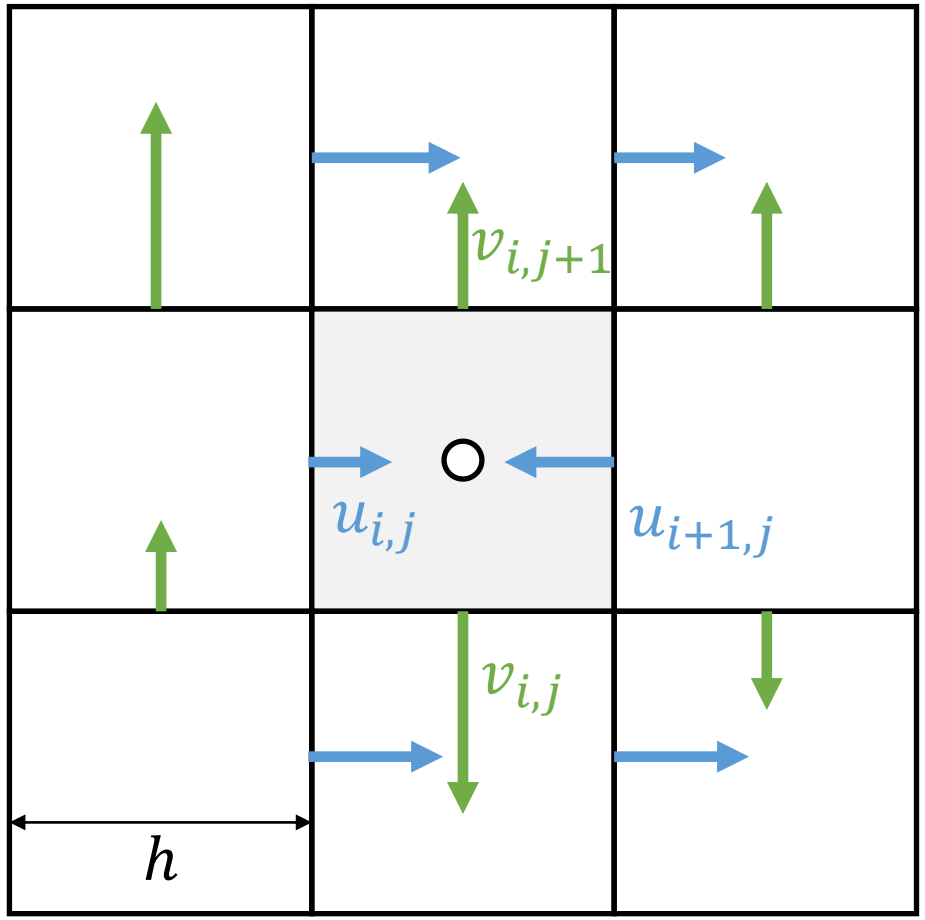
Central differencing gives the derivative in the middle.



- The cell doesn't exist at (i+0.5, j).
- To get $\frac{\partial f_{i,j}}{\partial x}$, we need $f_{i-1,j}$ and $f_{i+1,j}$. But this is weird, because $f_{i,j}$ is unused.

Solution: Staggered Grid

We define some physical quantities on faces, specifically velocities.

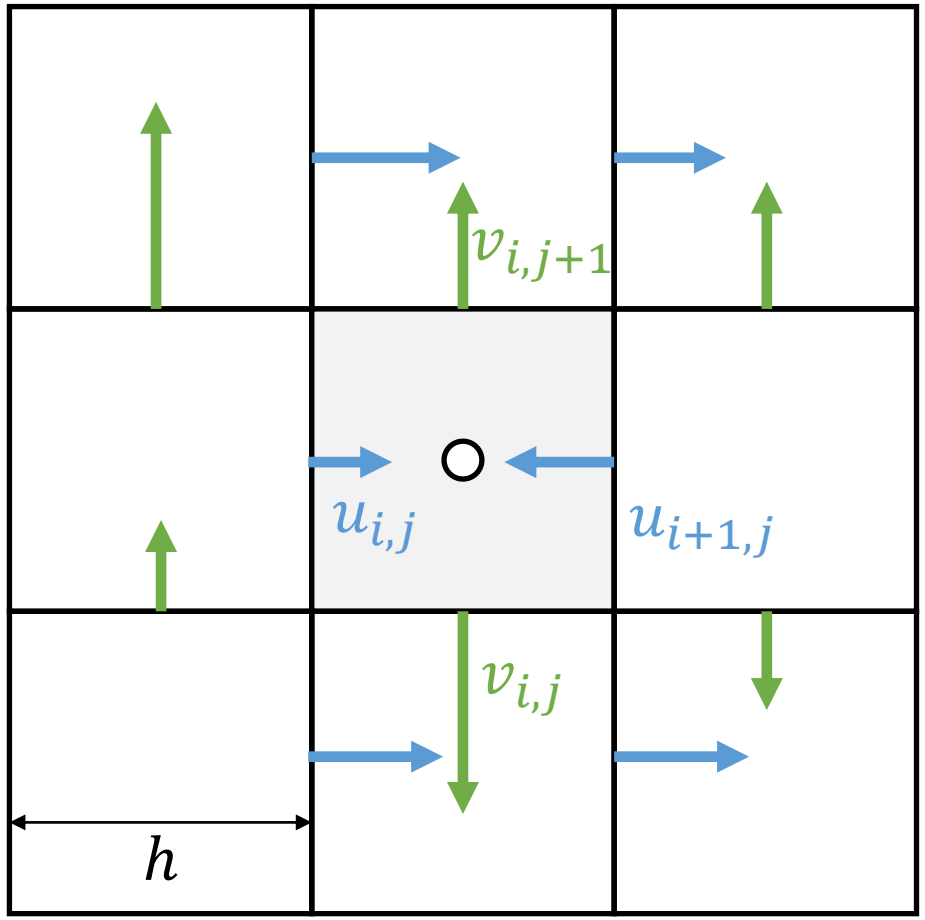


- The x-part of the velocity is defined on vertical faces.
- The y-part of the velocity is defined on horizonal faces.
- Intuitively, they represent the flow speed between two cells. For example, we write the volume changing speed at cell (i, j) as:

$$u_{i+1,j} + v_{i,j+1} - u_{i,j} - v_{i,j}$$

Divergence-Free Condition

No volume change is equal to say the fluid is incompressible. This can be formally written as a divergence-free velocity field.



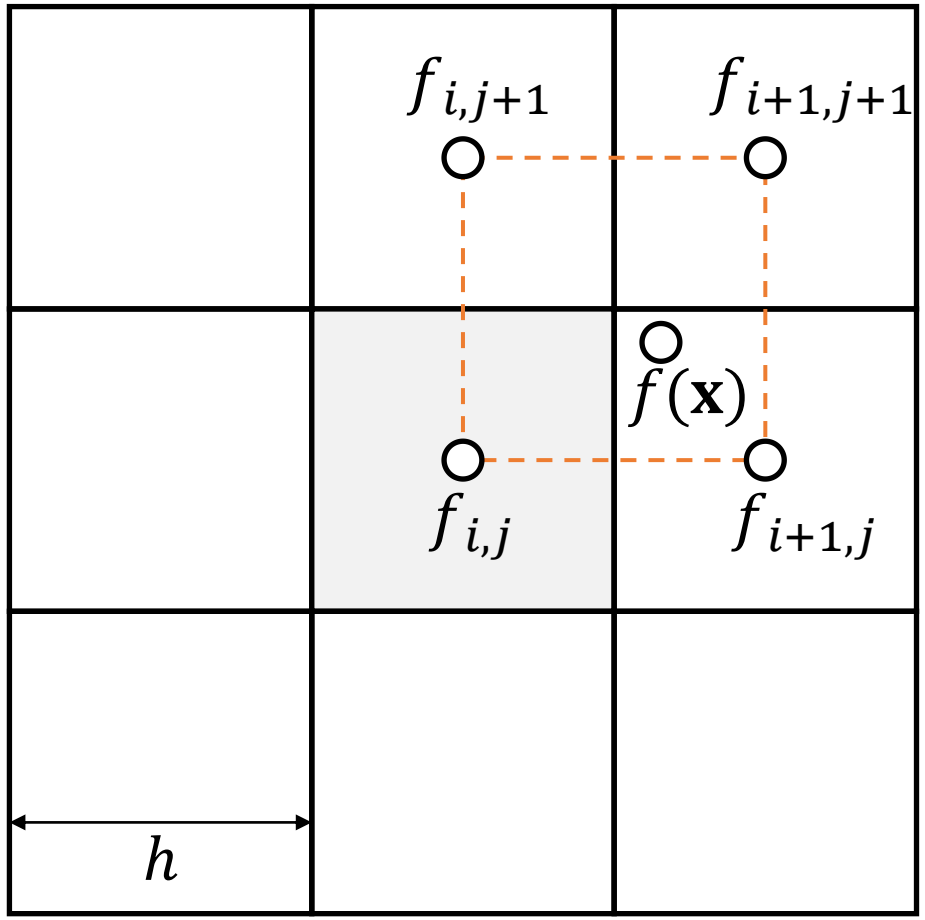
$$u_{i+1,j} + v_{i,j+1} - u_{i,j} - v_{i,j} = 0$$

$$\nabla \cdot \mathbf{u}_{i,j} = \frac{\partial u_{i,j}}{\partial x} + \frac{\partial v_{i,j}}{\partial y} \approx \frac{u_{i+1,j} - u_{i,j}}{h} + \frac{v_{i,j+1} - v_{i,j}}{h} = 0$$

velocity field

Bilinear Interpolation

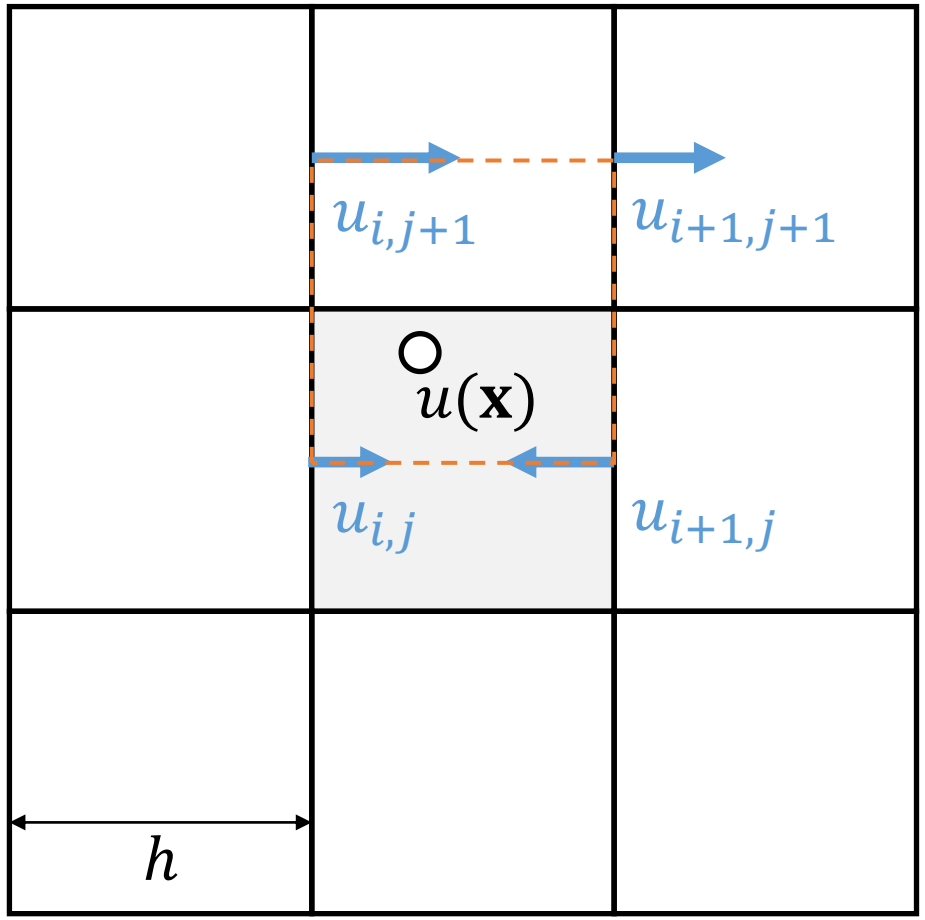
We use bilinear interpolation to interpolate physical quantities.



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

Bilinear Interpolation

We use bilinear interpolation to interpolate staggered velocities as well.



$$x \leftarrow x - 0.5$$
$$i \leftarrow \lfloor x \rfloor$$
$$j \leftarrow \lfloor y \rfloor$$
$$\begin{aligned} u(\mathbf{x}) \leftarrow\ & (i+1-x)(j+1-y)u_{i,j} \\ & + (x-i)(j+1-y)u_{i+1,j} \\ & + (i+1-x)(y-j)u_{i,j+1} \\ & + (x-i)(y-j)u_{i+1,j+1} \end{aligned}$$

Incompressible, Viscous Navier-Stokes Equations

Equation Fomulation

Incompressibility

$$\nabla \cdot \mathbf{u} = 0$$

Momentum

advection

Internal pressure

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{g} - (\mathbf{u} \cdot \nabla)\mathbf{u} + \mu \Delta \mathbf{u} - \nabla p$$

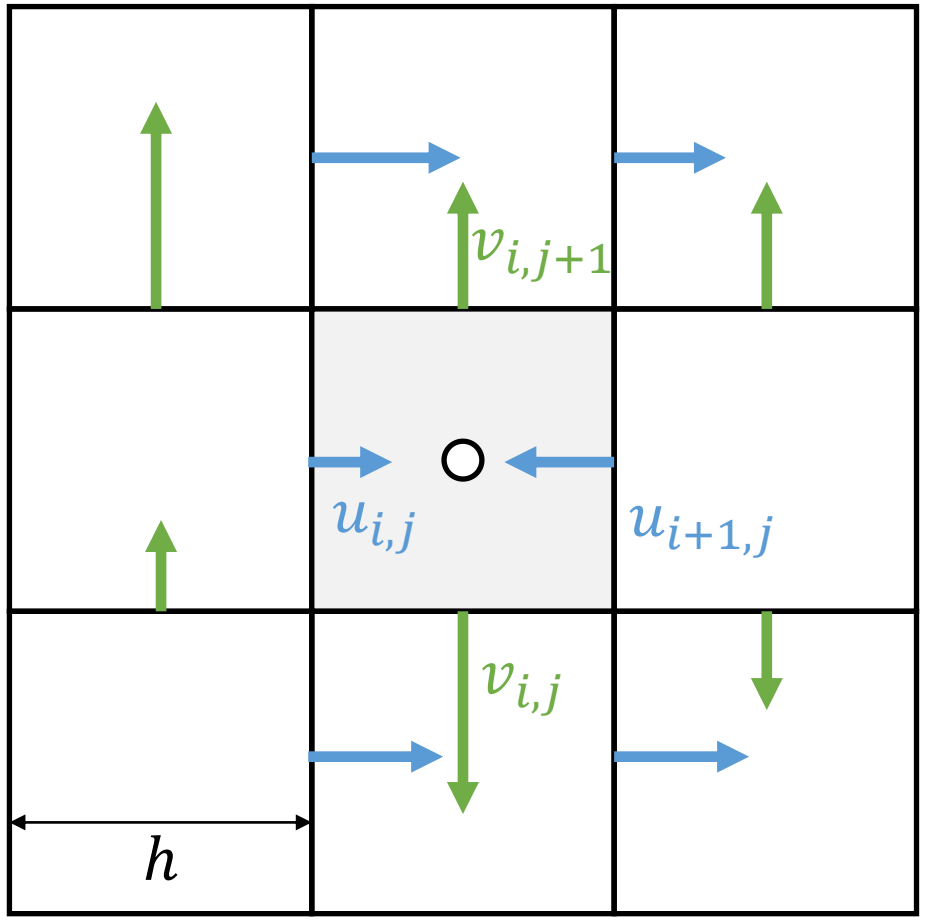
external acceleration

(Laplacian) diffusion

- Method of Characteristics: solving a long partial differential equation (PDE) in steps
- Step 1: Update $\mathbf{u}$ by solving $\partial \mathbf{u} / \partial t = \mathbf{g}$
- Step 2: Update $\mathbf{u}$ by solving $\partial \mathbf{u} / \partial t = -(\mathbf{u} \cdot \nabla)\mathbf{u}$
- Step 3: Update $\mathbf{u}$ by solving $\partial \mathbf{u} / \partial t = \upsilon \Delta \mathbf{u}$
- Step 4: Update $\mathbf{u}$ by solving $\partial \mathbf{u} / \partial t = -\nabla p$

Step 1: External Acceleration

The Update of $\mathbf{u}$ by $\partial \mathbf{u}/\partial t = \mathbf{g}$ is straightforward, just add acceleration to u and v.



If gravity is the only external acceleration, we get:

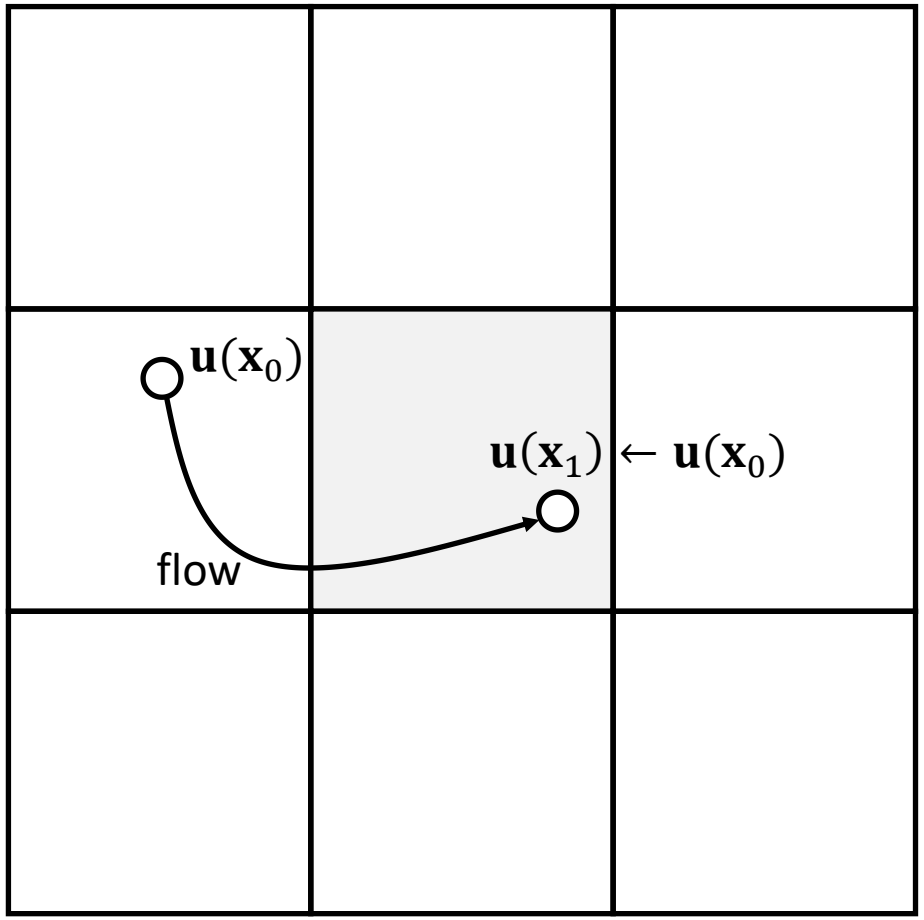
$$v_{i,j}^{new} \leftarrow v_{i,j} + \Delta t g$$

gravity

time step

Step 2: Advection

Next we need to update $\mathbf{u}$ by solving $\partial \mathbf{u} / \partial t = -(\mathbf{u} \cdot \nabla)\mathbf{u}$.



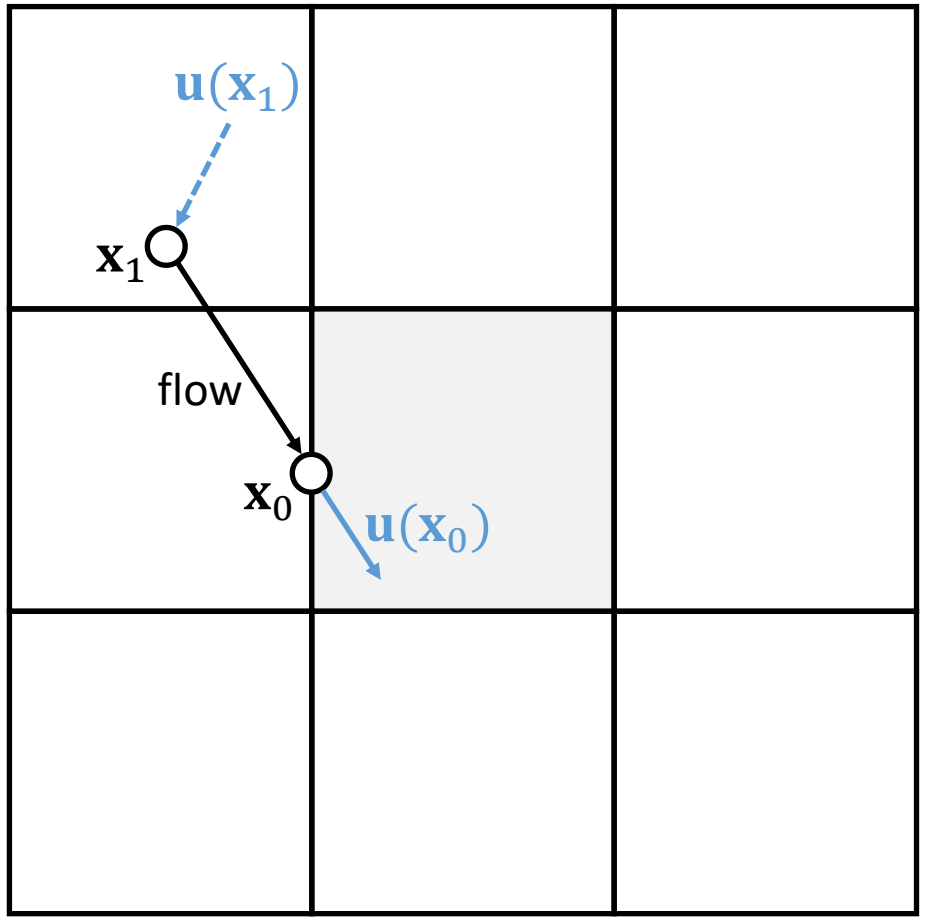
$$(\mathbf{u} \cdot \nabla)\mathbf{u} = \begin{bmatrix} u \cdot \dfrac{\partial u}{\partial x} + v \cdot \dfrac{\partial u}{\partial y} \\ u \cdot \dfrac{\partial v}{\partial x} + v \cdot \dfrac{\partial v}{\partial y} \end{bmatrix}$$

Solving this in an Eulerian way can be a source of instability.

To solve this problem, we come to realize that advection means to carry physical quantities by velocity.

Solution: Semi-Lagrangian Method

The solution is to trace a virtual particle backward over time.

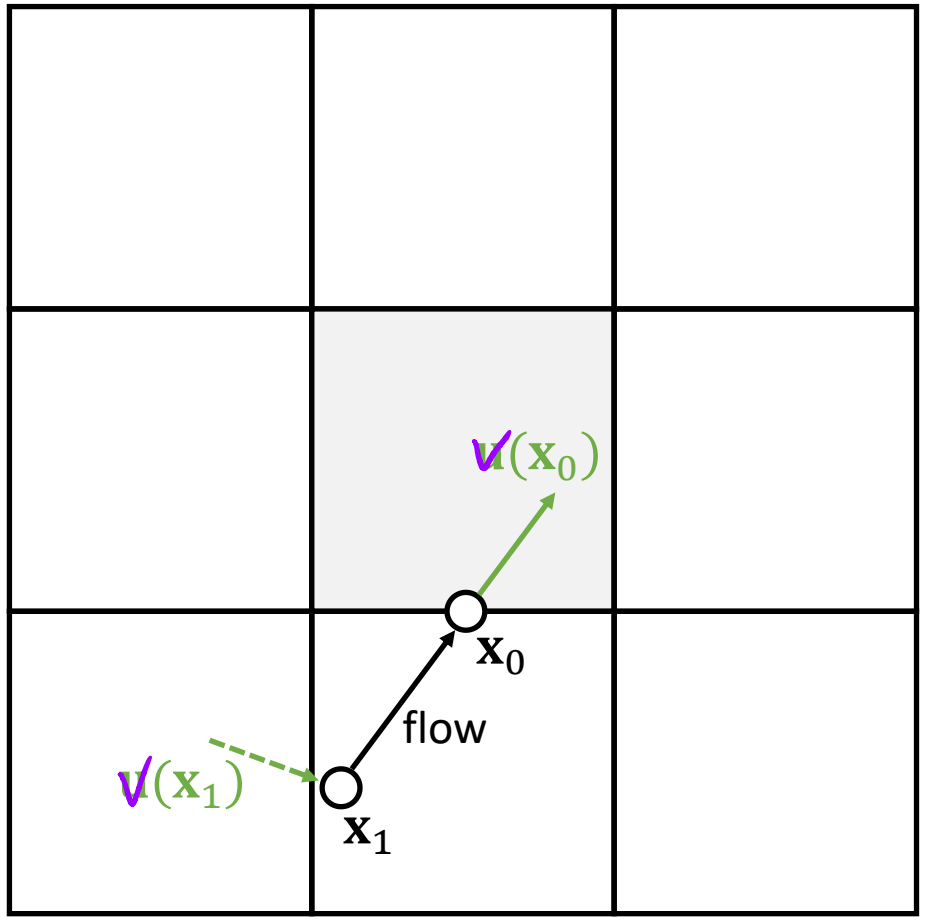


- Define $\mathbf{x_0} \leftarrow (i - 0.5,\ j)$
- Compute $\mathbf{u}(\mathbf{x}_0)$
- $\mathbf{x}_1 \leftarrow \mathbf{x}_0 - \Delta t\ \mathbf{u}(\mathbf{x}_0)$
- Compute $\mathbf{u}(\mathbf{x}_1)$
- $u_{i,j}^{new} \leftarrow u(\mathbf{x}_1)$

Note that if the velocities are staggered, we need to do staggered bilinear interpolation.

Solution: Semi-Lagrangian Method

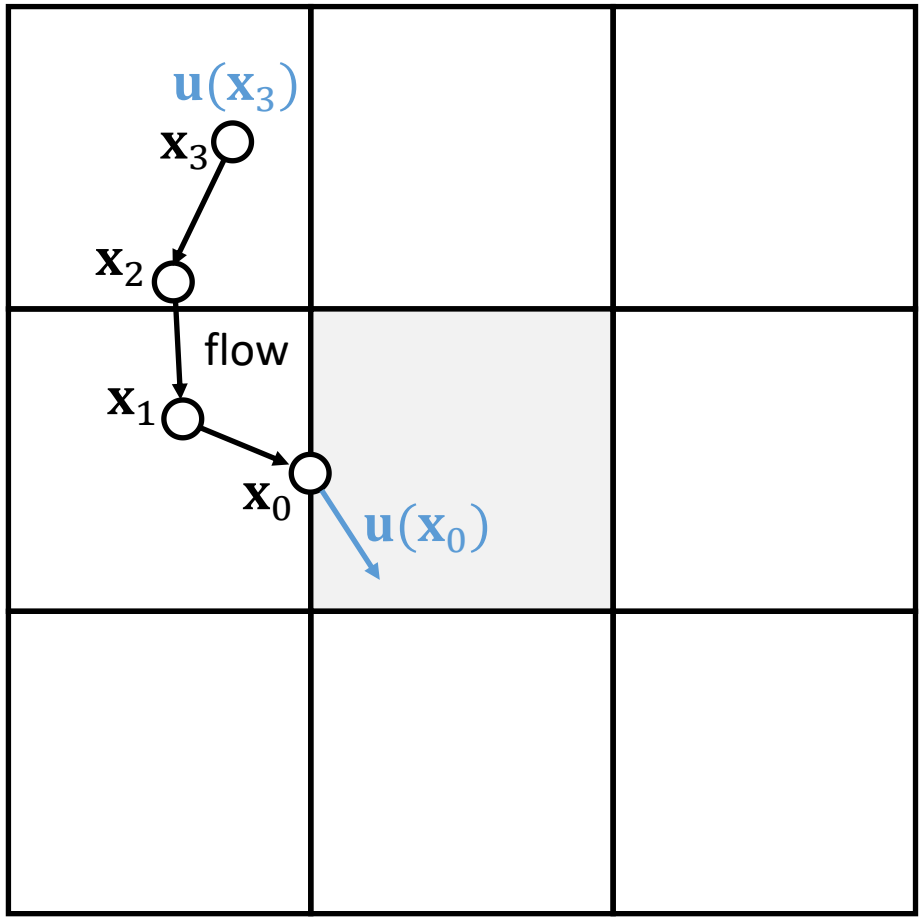
The solution is to trace a virtual particle backward over time.



- Define $\mathbf{x_0} \leftarrow (i,\ j - 0.5)$
- Compute $\mathbf{u}(\mathbf{x}_0)$
- $\mathbf{x}_1 \leftarrow \mathbf{x}_0 - \Delta t\ \mathbf{u}(\mathbf{x}_0)$
- Compute $\mathbf{u}(\mathbf{x}_1)$
- $v_{i,j}^{new} \leftarrow v(\mathbf{x}_1)$

Solution: Semi-Lagrangian Method

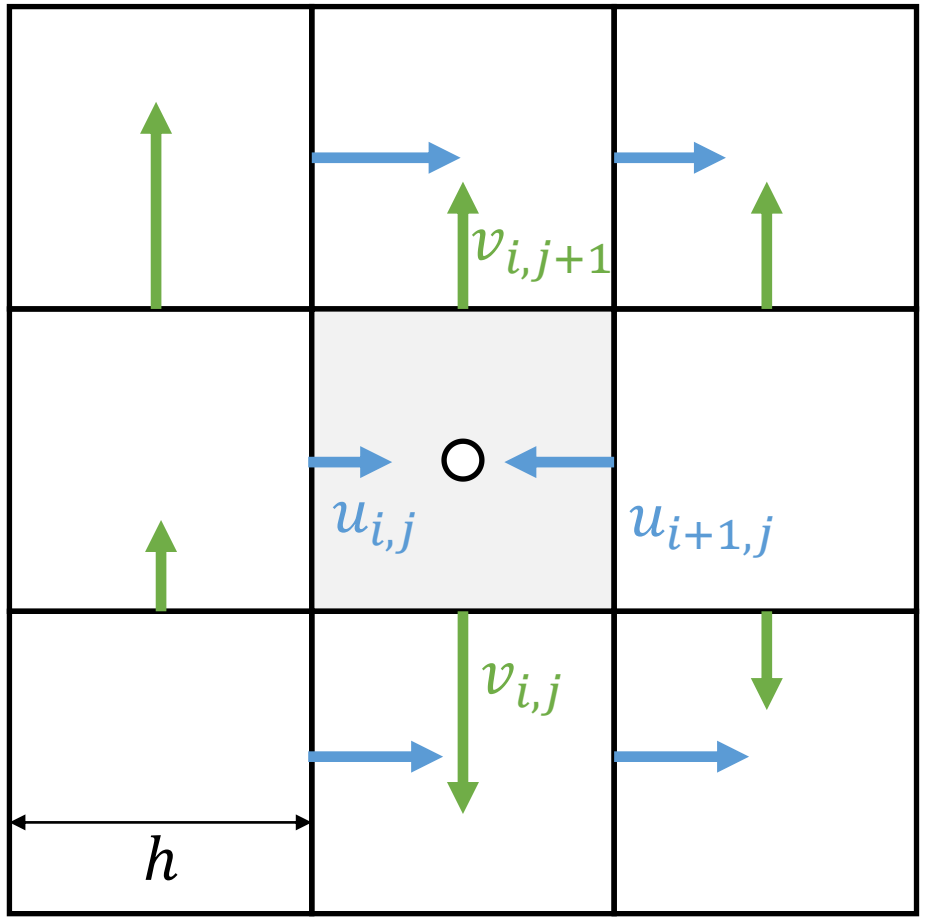
We could also subdivided the time step for better tracing.



- Define $\mathbf{x}_0 \leftarrow (i - 0.5,\ j)$
- Compute $\mathbf{u}(\mathbf{x}_0)$
- $\mathbf{x}_1 \leftarrow \mathbf{x}_0 - \frac{1}{3}\Delta t\ \mathbf{u}(\mathbf{x}_0)$
- Compute $\mathbf{u}(\mathbf{x}_1)$
- $\mathbf{x}_2 \leftarrow \mathbf{x}_1 - \frac{1}{3}\Delta t\ \mathbf{u}(\mathbf{x}_1)$
- Compute $\mathbf{u}(\mathbf{x}_2)$
- $\mathbf{x}_3 \leftarrow \mathbf{x}_2 - \frac{1}{3}\Delta t\ \mathbf{u}(\mathbf{x}_2)$
- Compute $\mathbf{u}(\mathbf{x}_3)$
- $u_{i,j}^{new} \leftarrow u(\mathbf{x}_3)$

Step 3: Diffusion

Next we need to update $\mathbf{u}$ by solving $\partial \mathbf{u}/\partial t = \upsilon \Delta \mathbf{u}$.



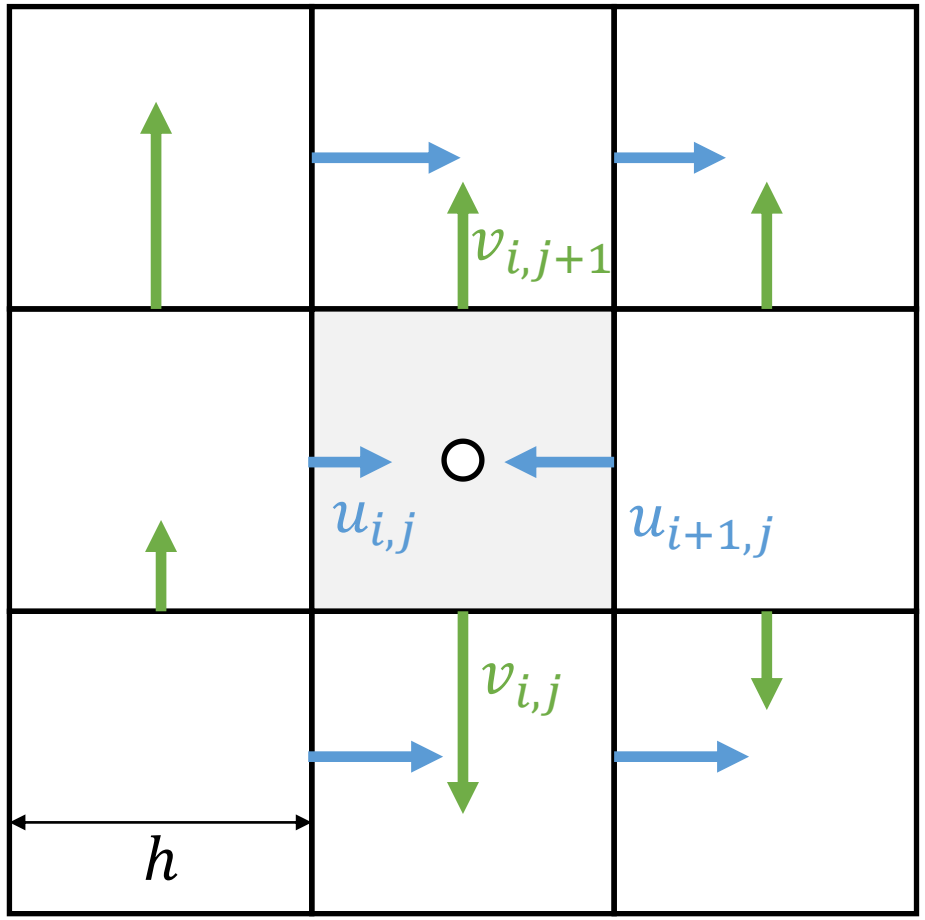
$$u_{i,j}^{new} \leftarrow u_{i,j} + \upsilon \Delta t \frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}}{h^2}$$

$$v_{i,j}^{new} \leftarrow v_{i,j} + \upsilon \Delta t \frac{v_{i-1,j} + v_{i+1,j} + v_{i,j-1} + v_{i,j+1} - 4v_{i,j}}{h^2}$$

If $\upsilon \Delta t$ is large, the above formulae can be unstable.

Step 3: Diffusion

Next we need to update $\mathbf{u}$ by solving $\partial \mathbf{u}/\partial t = \upsilon \Delta \mathbf{u}$.



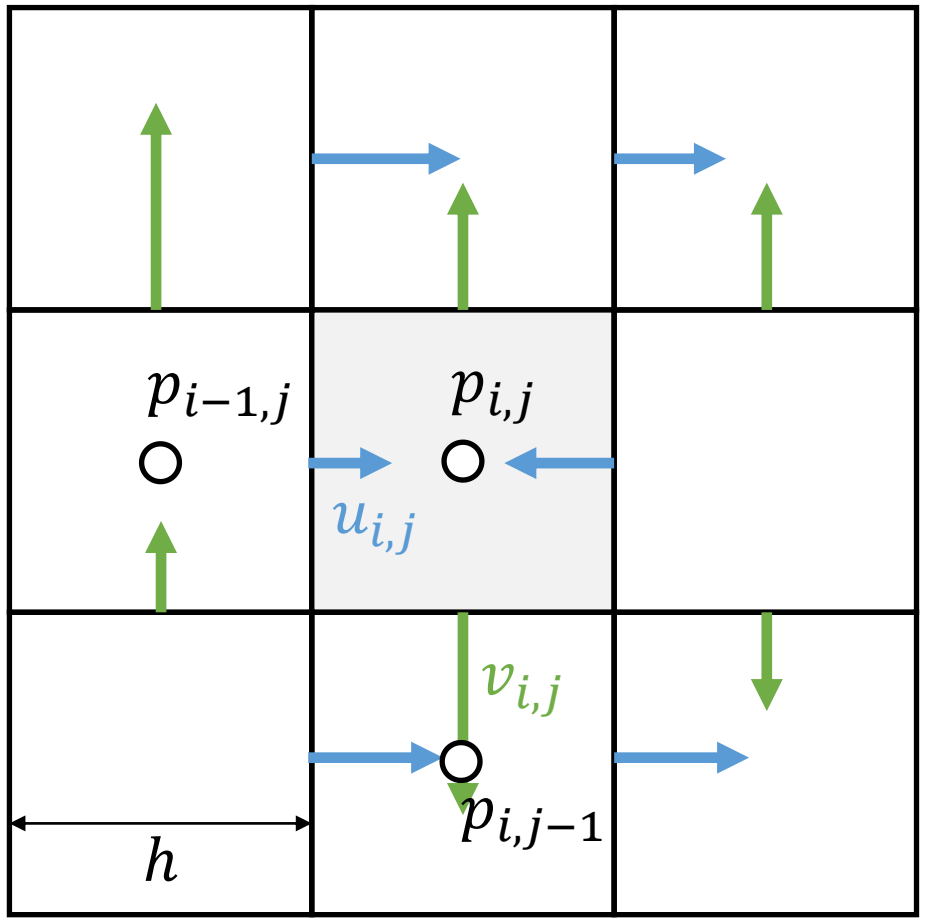
$$u_{i,j}^{temp} \leftarrow u_{i,j} + \upsilon \frac{\Delta t}{2} \frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}}{h^2}$$

$$u_{i,j}^{new} \leftarrow u_{i,j}^{temp} + \upsilon \frac{\Delta t}{2} \frac{u_{i-1,j}^{temp} + u_{i+1,j}^{temp} + u_{i,j-1}^{temp} + u_{i,j+1}^{temp} - 4u_{i,j}^{temp}}{h^2}$$

We could also use even smaller sub-steps…

Step 4: Pressure Projection

Finally, we need to update $\mathbf{u}$ by solving $\partial \mathbf{u}/\partial t = -\nabla \boldsymbol{p}$.



Staggering makes this very straightforward:

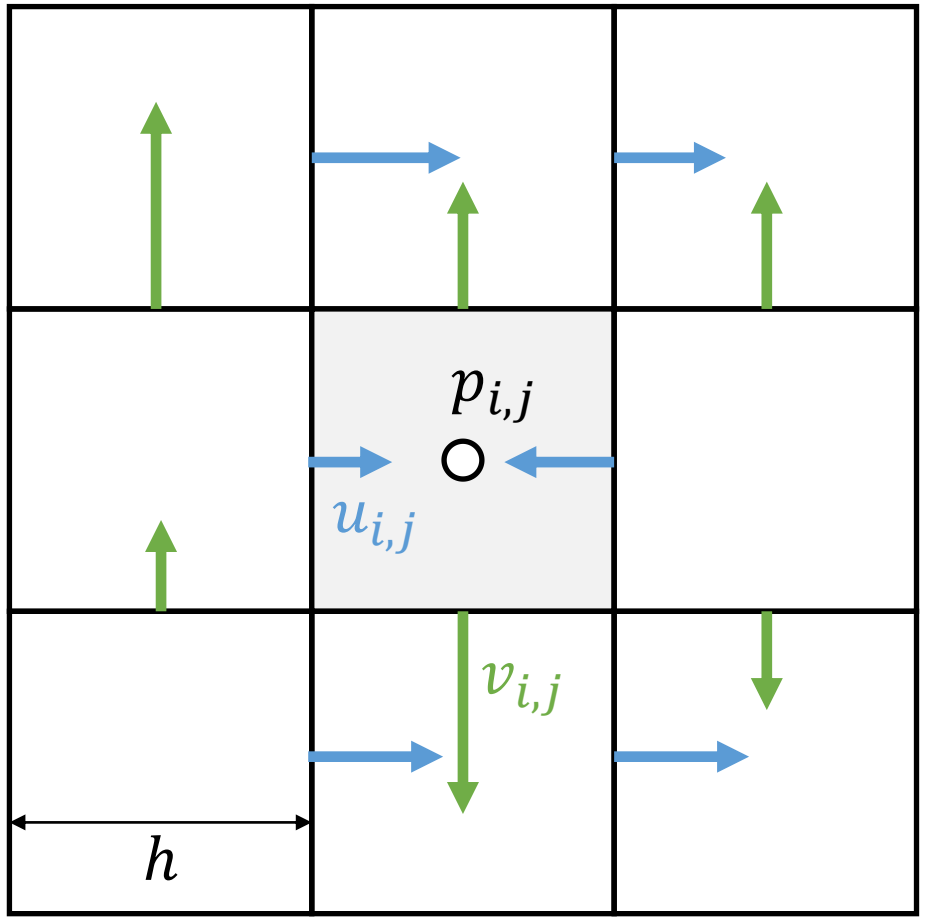
$$u_{i,j}^{new} \leftarrow u_{i,j} - \frac{\Delta t}{h}\left(p_{i,j} - p_{i-1,j}\right)$$

$$v_{i,j}^{new} \leftarrow v_{i,j} - \frac{\Delta t}{h}\left(p_{i,j} - p_{i,j-1}\right)$$

But what is $\boldsymbol{p}$?

Step 3: Pressure Projection

The pressure is caused by incompressibility.



In other words, after this update by pressure, we should achieve:

$$\nabla \cdot \mathbf{u}^{new} = 0$$

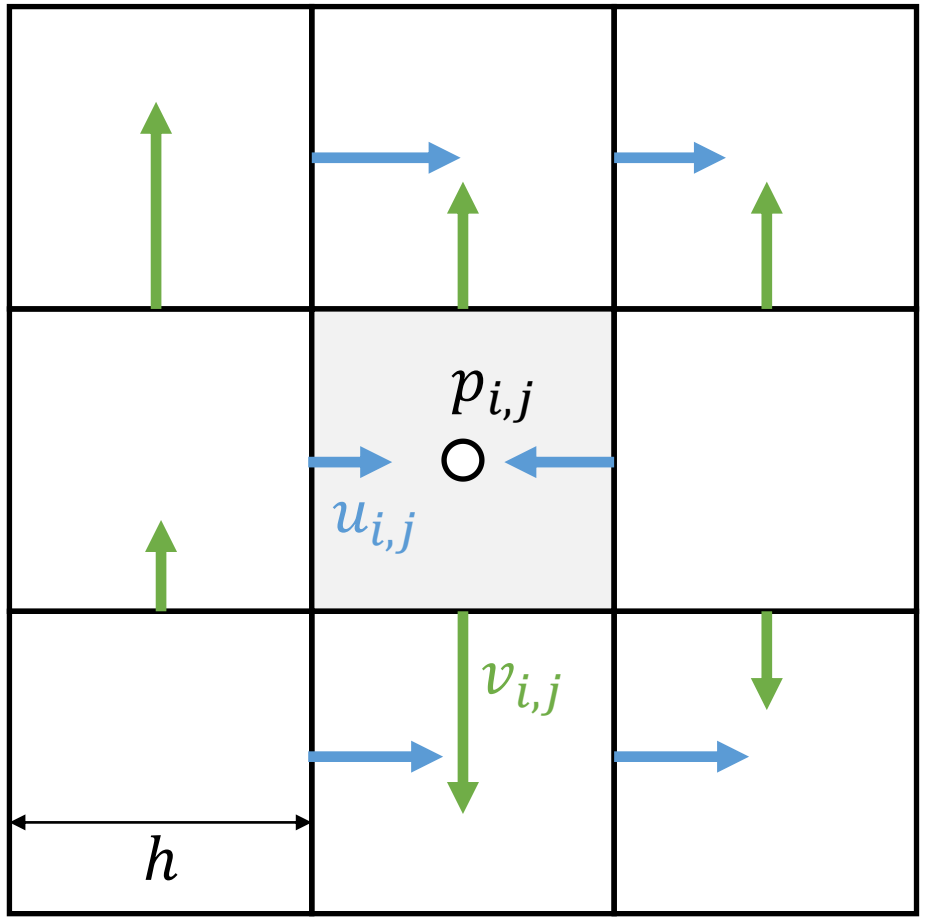
which means

$$u_{i+,j}^{new} + v_{i,j+1}^{new} - u_{i,j}^{new} - v_{i,j}^{new} = 0$$

$$u_{i+1,j} - \frac{\Delta t(p_{i+1,j} - p_{i,j})}{h} + v_{i,j+1} - \frac{\Delta t(p_{i,j+1} - p_{i,j})}{h} - u_{i,j} - \frac{\Delta t(p_{i,j} - p_{i-1,j})}{h} - v_{i,j} - \frac{\Delta t(p_{i,j} - p_{i,j-1})}{h} = 0$$

Step 3: Pressure Projection

The pressure is caused by incompressibility.



Eventually, we get a Poisson equation:

$$4p_{i,j} - p_{i-1,j} - p_{i+1,j} - p_{i,j-1} - p_{i,j+1} = h(-u_{i+1,j} - v_{i,j+1} + u_{i,j} + v_{i,j})/\Delta t$$

with boundary conditions:

Dirichlet boundary (open) $p_{i-1,j} = P$

Neumann boundary (close) $p_{i-1,j} = p_{i,j}$

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

After-Class Reading

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

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

Air and Smoke

Air Simulation

- Air simulation is done in two steps.
- In Step 1, we update the flow (the velocity field) $\mathbf{u}$.
- In Step 2, we use semi-Lagrangian (page 22) advect all of the other physical quantities, i.e., density, temperature…
- Typically we use Dirichlet boundaries for an open space (or Neumann boundaries for a container.)
- We can use it to simulate underwater as well.

Water Simulation

- Two representations
  - Volume-of-fluid (as the name suggests…)
  - A signed distance function defined over the grid.
- How to advect?
  - Semi-Lagrangian (volume loss)
  - Level set method (volume loss)
  - Needs corrections.

But what if there is an air-water boundary???

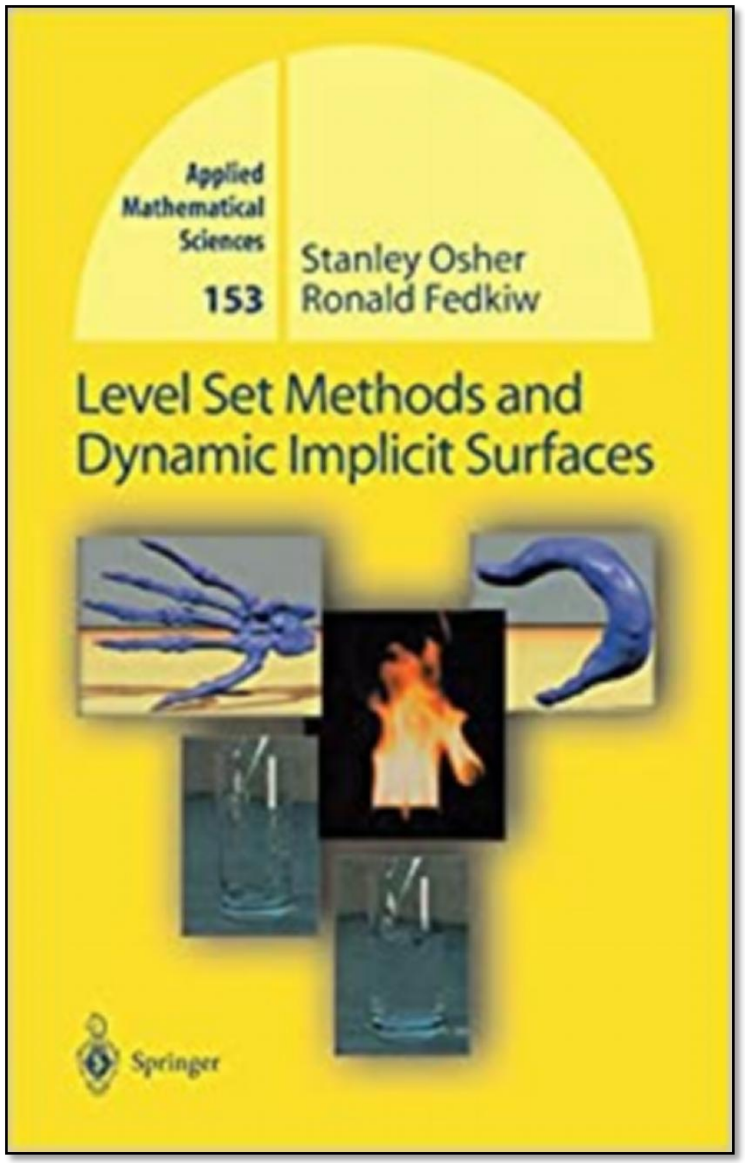
Water Simulation

- Two representations
  - Volume-of-fluid (as the name suggests…)
  - A signed distance function defined over the grid.
- How to advect?
  - Semi-Lagrangian (volume loss)
  - Level set method (volume loss)
  - Needs corrections.

But what if there is an air-water boundary???

After-Class Reading

Osher and Fedkiw.
Level Set Methods and Dynamic Implicit Surfaces.



A Summary For the Day

- The Eulerian grid presentation is very friendly with finite differencing. This makes calculus a lot easier.
- For velocity fields, we can use staggered grid.

- For low-speed, incompressible, viscous flow, we need to solve the Navier-Stokes equations.
- To solve the equations, we can do this in step-by-step (method of characteristics).

- To simulate air and water, we need to advect some physical quantities.
  - Smoke (density); water (volume-of-fluid, or signed distance function)
  - Volume loss issue in water (how to fix it?)
  - If you need to create a mesh from grid for rendering, you need something like marching cube.