Accelerating Eulerian fluid simulation with convolutional networks.

J Tompson, K Schlachter, P Sprechmann, K Perlin

Presenter: Robin Walters
CS 7180

March 28, 2019

In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3424-3433. JMLR. org, 2017.

Goal

Simulate fluid flow in 3D space.

Goal

Simulate fluid flow in 3D space.

- Around solid objects.

Goal

Simulate fluid flow in 3D space.

- Around solid objects.
- Fluid: gas or liquid

Goal

Simulate fluid flow in 3D space.

- Around solid objects.
- Fluid: gas or liquid
- We assume fluid is *incompressible* (like water)

Goal

Simulate fluid flow in 3D space.

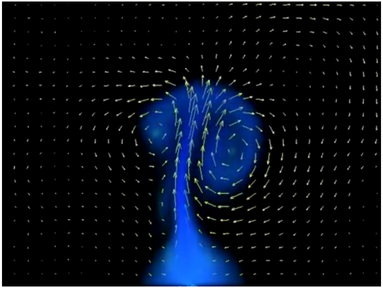
- Around solid objects.
- Fluid: gas or liquid
- We assume fluid is *incompressible* (like water)
- And *has no viscosity.*

wikipedia.org

Mathematical Representation for Fluid Flow

Represent the fluid's *velocity* at every time t and point in space $\vec{x}$ as vector field

$$\vec{u}(t, \vec{x})$$



Evolution of the Flow

Flow evolves with time $\frac{\partial \vec{u}}{\partial t}$, by Navier-Stokes equations.
First: Newton's second law

$$\vec{F} = m\vec{a}$$

Evolution of the Flow

Flow evolves with time $\frac{\partial \vec{u}}{\partial t}$, by Navier-Stokes equations.
First: Newton's second law

$$\vec{F} = m\vec{a}$$

Uniform density $\implies m = 1$. So...

Evolution of the Flow

Flow evolves with time $\frac{\partial \vec{u}}{\partial t}$, by Navier-Stokes equations.
First: Newton's second law

$$\vec{F} = \vec{a}$$

Evolution of the Flow

Flow evolves with time $\frac{\partial \vec{u}}{\partial t}$, by Navier-Stokes equations.
First: Newton's second law

$$\vec{a} = \vec{F}$$

Evolution of the Flow

Flow evolves with time $\frac{\partial \vec{u}}{\partial t}$, by Navier-Stokes equations.
First: Newton's second law

$$\vec{a} = \vec{F}$$

$\vec{u}$ is a vel., so $\frac{\partial \vec{u}}{\partial t}$ is accel. So

$$\frac{\partial \vec{u}}{\partial t} = \vec{F} = \text{ sum of all forces}$$

Navier-Stokes for no viscosity (AKA Euler)

- External Forces
  (gravity, bouyancy, ...)

$$\frac{\partial \vec{u}}{\partial t} = \boxed{\vec{F}_{ext}} - \vec{u} \cdot \nabla_{\vec{x}} \vec{u} - \boxed{\nabla_{\vec{x}} p}$$

Navier-Stokes for no viscosity (AKA Euler)

- External Forces
  (gravity, bouyancy, ...)

$$\frac{\partial \vec{u}}{\partial t} = \boxed{\vec{F}_{ext}} - \boxed{\vec{u} \cdot \nabla_{\vec{x}} \vec{u}} - \boxed{\nabla_{\vec{x}} p}$$

- Convection
  Flow moves because of flow

Navier-Stokes for no viscosity (AKA Euler)

- External Forces
  (gravity, bouyancy, ...)

$$\frac{\partial \vec{u}}{\partial t} = \boxed{\vec{F}_{ext}} - \vec{u} \cdot \nabla_{\vec{x}} \vec{u} - \boxed{\nabla_{\vec{x}} p}$$

- Convection
  Flow moves because of flow
- Pressure

Both equations

- Newton's 2nd

$$\frac{\partial \vec{u}}{\partial t} = \vec{F}_{ext} - \vec{u} \cdot \nabla_{\vec{x}} \vec{u} - \nabla_{\vec{x}} p \qquad \text{(N)}$$

Both equations

- Newton's 2nd

$$\frac{\partial \vec{u}}{\partial t} = \vec{F}_{ext} - \vec{u} \cdot \nabla_{\vec{x}} \vec{u} - \nabla_{\vec{x}} p \qquad \text{(N)}$$

- Incompressible

$$\nabla_{\vec{x}} \cdot u \stackrel{\rightarrow}{=} 0 \qquad \text{(I)}$$

Both equations

- Newton's 2nd

$$\frac{\partial \vec{u}}{\partial t} = \vec{F}_{ext} - \vec{u} \cdot \nabla_{\vec{x}} \vec{u} - \nabla_{\vec{x}} p \qquad \text{(N)}$$

- Incompressible

$$\nabla_{\vec{x}} \cdot u \vec{=} 0 \qquad \text{(I)}$$

- Two equations, two unknowns $\vec{u}$ and p

Numerical Solving

Discretize

$$\vec{u}_{n,i} = \vec{u}(t_n, \vec{x}_i)$$

Numerical Solving

Discretize

$$\vec{u}_{n,i} = \vec{u}(t_n, \vec{x}_i)$$

Solve back and forth

$$\ldots \implies \vec{u}_n \overset{(N),p=0}{\implies} \hat{u}_{n+1} \overset{(I)}{\implies} p_{n+1} \implies \vec{u}_{n+1} \overset{(N),p=0}{\implies} \hat{u}_{n+2} \overset{(I)}{\implies} p_{n+2} \ldots$$

Numerical Solving

Discretize

$$\vec{u}_{n,i} = \vec{u}(t_n, \vec{x}_i)$$

Solve back and forth

$$\ldots \implies \vec{u}_n \overset{(N),p=0}{\implies} \hat{u}_{n+1} \overset{(I)}{\implies} p_{n+1} \implies \vec{u}_{n+1} \overset{(N),p=0}{\implies} \hat{u}_{n+2} \overset{(I)}{\implies} p_{n+2} \ldots$$

$p_{n-1} \implies \vec{u}_{n-1} \implies \hat{u}_n$ **is easy and fast,**
But $\hat{u}_n \implies p_n$ **is hard and slow!**

$\hat{u}_{n+1} \implies p_{n+1}$

By (I),

$$\Delta p = \nabla \cdot \hat{u}_n$$

$(b = \nabla \cdot \hat{u}_n)$

$$\Delta p = b$$

$\hat{u}_{n+1} \implies p_{n+1}$

By (I),

$$\Delta p = \nabla \cdot \hat{u}_n$$

$(b = \nabla \cdot \hat{u}_n)$

$$\Delta p = b$$

$$\frac{\partial}{\partial x}^2 p + \frac{\partial}{\partial y}^2 p = b$$

$\hat{u}_{n+1} \implies p_{n+1}$

By (I),

$$\Delta p = \nabla \cdot \hat{u}_n$$

$(b = \nabla \cdot \hat{u}_n)$

$$\Delta p = b$$

$$\frac{\partial}{\partial x}^2 p + \frac{\partial}{\partial y}^2 p = b$$

Replace derivatives with finite differences

$$\frac{\frac{p_{i+1,j} - p_{i,j}}{\Delta x} - \frac{p_{i,j} - p_{i-1,j}}{\Delta x}}{\Delta x} + \frac{\frac{p_{i,j+1} - p_{i,j}}{\Delta y} - \frac{p_{i,j} - p_{i,j-1}}{\Delta y}}{\Delta y} = b_{i,j}$$

$\hat{u}_{n+1} \implies p_{n+1}$

By (I),

$$\Delta p = \nabla \cdot \hat{u}_n$$

$(b = \nabla \cdot \hat{u}_n)$

$$\Delta p = b$$

$$\frac{\partial}{\partial x}^2 p + \frac{\partial}{\partial y}^2 p = b$$

Replace derivatives with finite differences

$$\frac{\frac{p_{i+1,j}-p_{i,j}}{\Delta x} - \frac{p_{i,j}-p_{i-1,j}}{\Delta x}}{\Delta x} + \frac{\frac{p_{i,j+1}-p_{i,j}}{\Delta y} - \frac{p_{i,j}-p_{i,j-1}}{\Delta y}}{\Delta y} = b_{i,j}$$

$$\frac{1}{l^2}\left(p_{i+1,j} + p_{i-1,j} + p_{i,j-1} + p_{i,j+1} - 4p_{i,j}\right) = b_{i,j}$$

Note each $b_{i,j}$ depends is a linear combination of 5 values of $p_{i,j}$ (7 in 3D).

The (very sparse) matrix is then

$$\begin{pmatrix} \vdots & & & \vdots & & & & \vdots & & \vdots \\ \cdots & 0 & 1 & 1 & -4 & 1 & 1 & 0 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & 1 & -4 & 0 & 1 & 1 & \cdots \\ \vdots & & & \vdots & & & & \vdots & & \vdots \end{pmatrix} \begin{pmatrix} \vdots \\ p_{i,j-1} \\ p_{i-1,j} \\ p_{i,j} \\ p_{i+1,j} \\ p_{i,j+1} \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ \vdots \\ \vdots \\ b_{i,j} \\ b_{i+1,j} \\ \vdots \\ \vdots \end{pmatrix} l^2$$

$\hat{u}_{n+1} \implies p_{n+1}$

The (very sparse) matrix is then

$$\begin{pmatrix} \vdots & & & \vdots & & & & \vdots & & \vdots \\ \cdots & 0 & 1 & 1 & -4 & 1 & 1 & 0 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & 1 & -4 & 0 & 1 & 1 & \cdots \\ \vdots & & & \vdots & & & & \vdots & & \vdots \end{pmatrix} \begin{pmatrix} \vdots \\ p_{i,j-1} \\ p_{i-1,j} \\ p_{i,j} \\ p_{i+1,j} \\ p_{i,j+1} \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ \vdots \\ \vdots \\ b_{i,j} \\ b_{i+1,j} \\ \vdots \\ \vdots \end{pmatrix} l^2$$

(This is the graph Laplacian for a grid graph.)

$\hat{u}_{n+1} \implies p_{n+1}$

The (very sparse) matrix is then

$$\begin{pmatrix} \vdots & & & \vdots & & & & & \vdots & \vdots \\ \cdots & 0 & 1 & 1 & -4 & 1 & 1 & 0 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & 1 & -4 & 0 & 1 & 1 & \cdots \\ \vdots & & & \vdots & & & & & \vdots & \vdots \end{pmatrix} \begin{pmatrix} \vdots \\ p_{i,j-1} \\ p_{i-1,j} \\ p_{i,j} \\ p_{i+1,j} \\ p_{i,j+1} \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ \vdots \\ \vdots \\ b_{i,j} \\ b_{i+1,j} \\ \vdots \\ \vdots \end{pmatrix} l^2$$

(This is the graph Laplacian for a grid graph.)
We need to solve this equation $Lp = b$ for p in terms of b! AHH!

Solving Methods (Related Work)

- Iterative technique: Conjugate Descent
    - Takes a long time

Solving Methods (Related Work)

- Iterative technique: Conjugate Descent
  - Takes a long time
- Iterative technique: Jacobi Method
  - Faster, but low accuracy

Solving Methods (Related Work)

- Iterative technique: Conjugate Descent
  - Takes a long time
- Iterative technique: Jacobi Method
  - Faster, but low accuracy
- Yang et al: Why not use a CNN!?! $p = \mathtt{CNN}(b)$
  - Faster, but hard to train (need true p)
  - generalizes poorly
  - Long term stability issues

Solving Methods (Related Work)

- Iterative technique: Conjugate Descent
  - Takes a long time
- Iterative technique: Jacobi Method
  - Faster, but low accuracy
- Yang et al: Why not use a CNN!?! $p = \texttt{CNN}(b)$
  - Faster, but hard to train (need true p)
  - generalizes poorly
  - Long term stability issues
- This work
  - Use unsupervised training
  - Use multi-resolution CNN
  - Use long-term loss function

Unsupervised Training

- Start with $\vec{u}_n$.

Unsupervised Training

- Start with $\vec{u}_n$.
- Get $\hat{u}_{n+1}$ from (N) without pressure term ($p = 0$)

$$\frac{\partial \vec{u}}{\partial t} = \vec{F}_{ext} - \vec{u} \cdot \nabla_{\vec{x}} \vec{u} - \cancel{\nabla_{\vec{x}} p} \quad \text{(N)}, p = 0$$

Unsupervised Training

- Start with $\vec{u}_n$.
- Get $\hat{u}_{n+1}$ from (N) without pressure term ($p = 0$)

$$\frac{\partial \vec{u}}{\partial t} = \vec{F}_{ext} - \vec{u} \cdot \nabla_{\vec{x}} \vec{u} - \cancel{\nabla_{\vec{x}} p} \quad \text{(N)}, p = 0$$

- p_{n+1} is correction to $\hat{u}_{n+1}$ so (I) holds too.

$$\nabla_{\vec{x}} \cdot \vec{u} = 0 \quad (I)$$

Unsupervised Training

- Start with $\vec{u}_n$.
- Get $\hat{u}_{n+1}$ from (N) without pressure term ($p = 0$)

$$\frac{\partial \vec{u}}{\partial t} = \vec{F}_{ext} - \vec{u} \cdot \nabla_{\vec{x}} \vec{u} - \cancel{\nabla_{\vec{x}} p} \quad \text{(N)}, p = 0$$

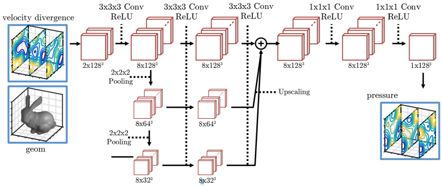
- p_{n+1} is correction to $\hat{u}_{n+1}$ so (I) holds too.

$$\nabla_{\vec{x}} \cdot \vec{u} = 0 \quad (I)$$

- Don't need true p_{n+1}, just use $|\nabla_{\vec{x}} \cdot \vec{u}| = |\mathrm{div}(\vec{u})|$ as loss.

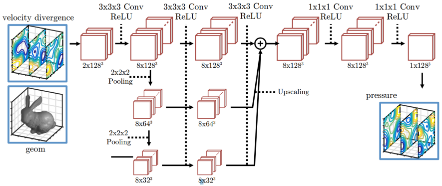
Multi-Resolution CNN and Long-term Loss

- CNN

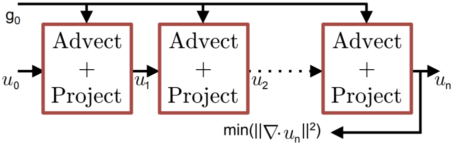


Multi-Resolution CNN and Long-term Loss

- CNN



- Long-term Loss



Results

- Fast runtimes (for capped loss)

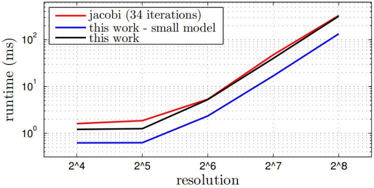


Figure 5. Pressure projection time (ms) versus resolution (PCG not shown for clarity).

Results

- Fast runtimes (for capped loss)

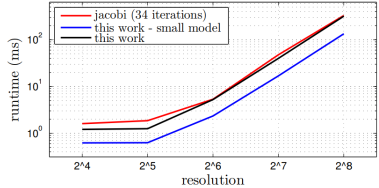
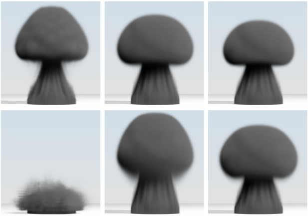


Figure 5. Pressure projection time (ms) versus resolution (PCG not shown for clarity).

- Visually comparable results



Discussion

Strengths

1. Hybrid approach use NN when best and PDE when best

Discussion

Strengths

1. Hybrid approach use NN when best and PDE when best
2. Unsupervised training technique is very clever

Discussion

Strengths

1. Hybrid approach use NN when best and PDE when best
2. Unsupervised training technique is very clever

Weakness/Improvement

1. Lack of quantitative metrics (could use simpler “wind tunnel” tests and macro effects)

Discussion

Strengths

1. Hybrid approach use NN when best and PDE when best
2. Unsupervised training technique is very clever

Weakness/Improvement

1. Lack of quantitative metrics (could use simpler “wind tunnel” tests and macro effects)
2. No 2D examples, No moving solids

Discussion

Strengths

1. Hybrid approach use NN when best and PDE when best
2. Unsupervised training technique is very clever

Weakness/Improvement

1. Lack of quantitative metrics (could use simpler “wind tunnel” tests and macro effects)
2. No 2D examples, No moving solids
3. Comparison to Yang et al. fails

Discussion Questions

1. Possible quantitative metrics?

Discussion Questions

1. Possible quantitative metrics?
2. Applications other than animation?

Discussion Questions

1. Possible quantitative metrics?
2. Applications other than animation?
3. Why is a CNN good at solving spare linear systems?

Discussion Questions

1. Possible quantitative metrics?
2. Applications other than animation?
3. Why is a CNN good at solving spare linear systems?
4. “vorticity confinement?”

More Piazza Discussion.