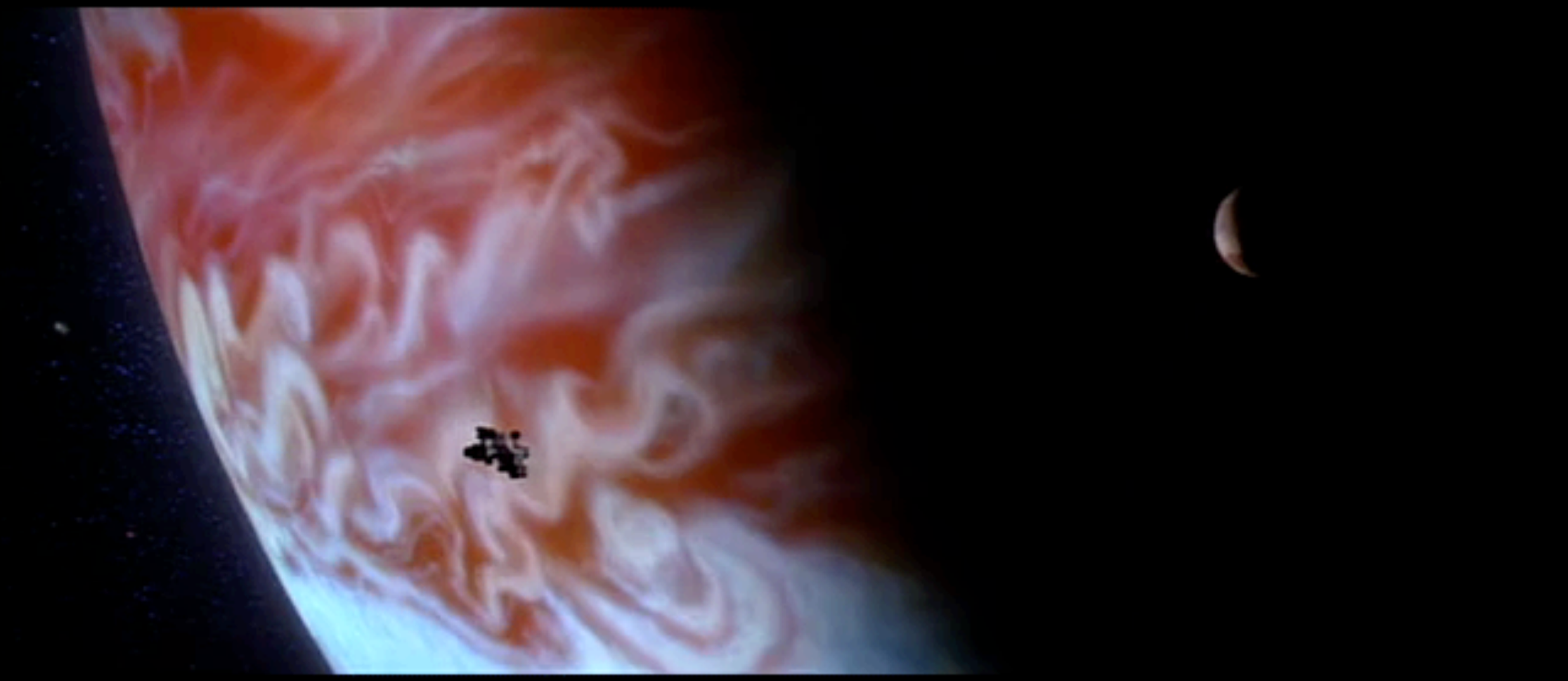
Stable Fluids by Jos Stam

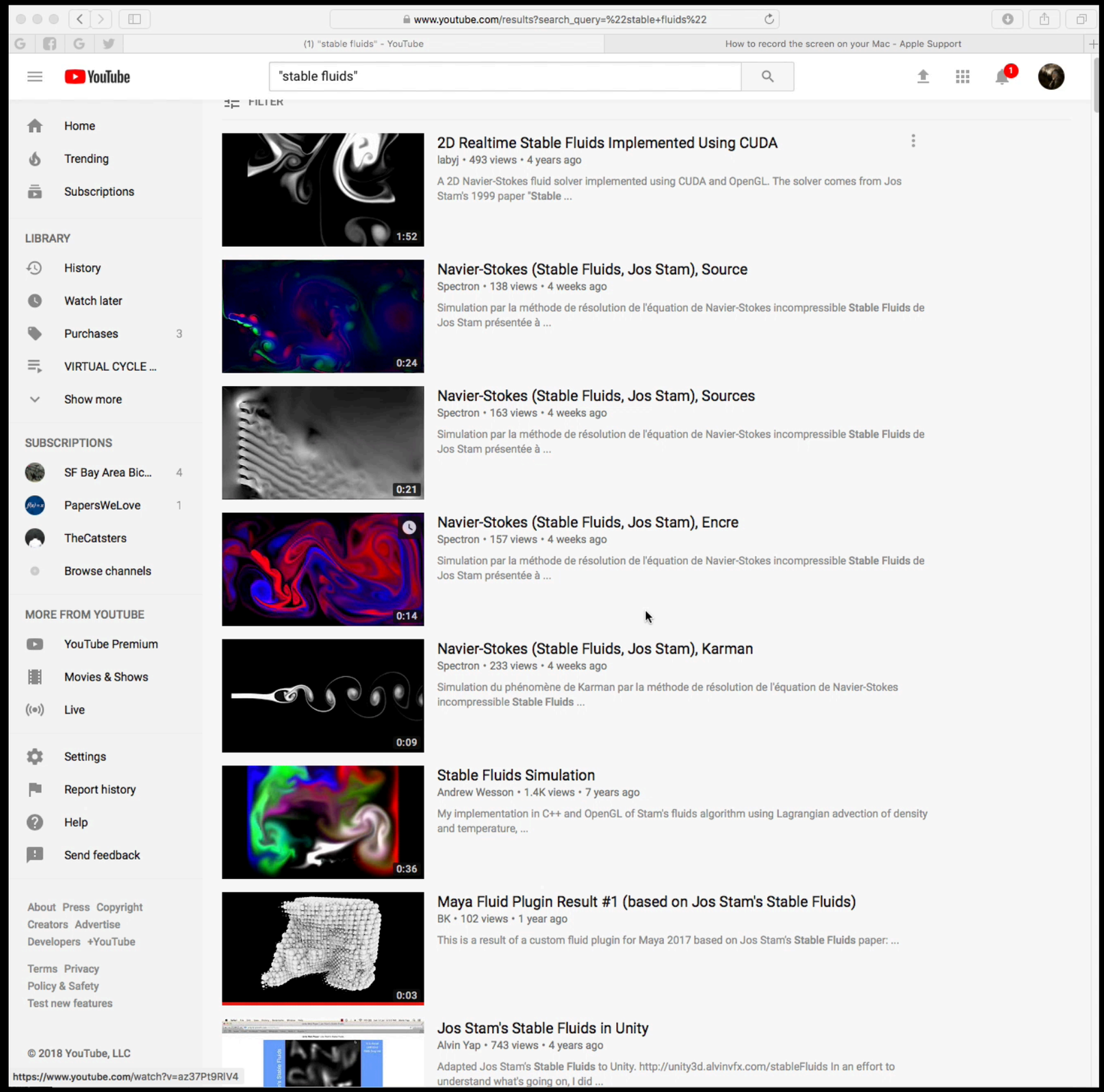
Dan Piponi

2010: The Year We Make Contact (1984)



© MGM

Stable Fluids



Stable

Fluids

From Cray X-MP to SGI Octane

1984

1999

Version 2:
From Cray X-MP to Compaq iPaq

1984

2001

Stable Fluids

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

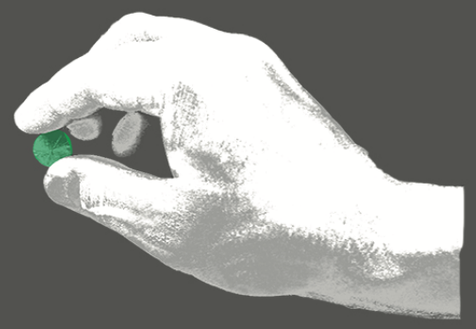
articles have been published in various areas on how to compute these equations numerically. Which solver to use in practice depends largely on the problem at hand and on the computing power available. Most engineering tasks require that the simulation provide accurate bounds on the physical quantities 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 tur-

Euler vs Lagrange

BENGE

TONE FLOW



01 - Kinematic Gate (6:09)
02 - Trajectory & Continuum (6:02)
03 - Geometry Of Motion (7:03)
04 - A Eulerian Description (9:44)
05 - A Lagrangian Description (10:59)

Total duration: 40:08

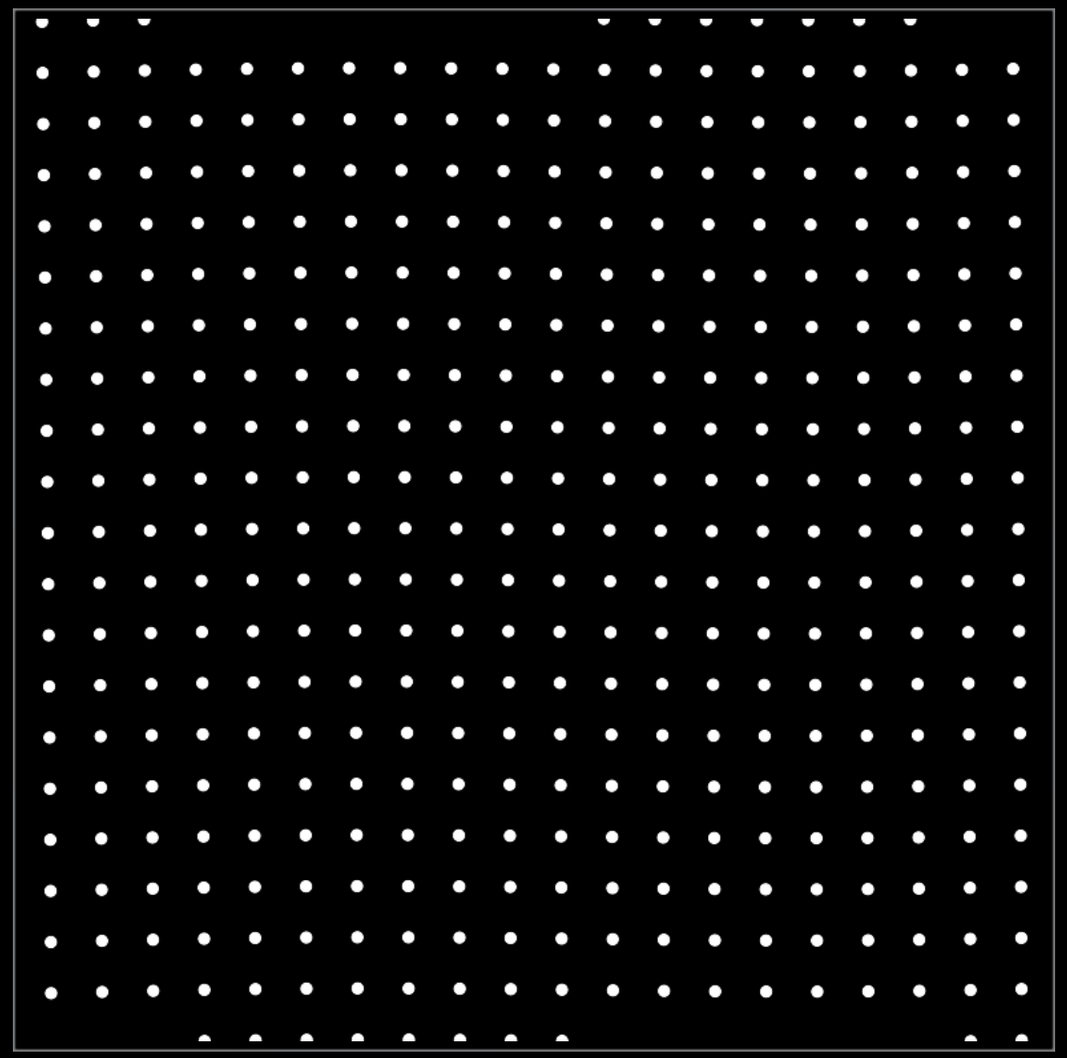
Tone Flow: a continuum phenomenon

Due to the temporal nature of music, it can be said that there are two ways of perceiving it. On the one hand (the Lagrangian specification) the listener may allow themselves to be carried along with the music, as a leaf might be carried along when landing upon a stream of water. On the other hand (the Eulerian specification), it is also possible to perceive the flowing music as a rock might experience the stream, allowing the water to move around it in a continuously changing and immersive flow

The following instruments were used on this album:
Moog 3C, Emu and Polyfusion Modular synthesisers, Moog 12 Stage Phaser
Korg DSS sampler (used to sample sustained tones from the Moog 3C)
Roland Space Echo 201, Bel BD80S delay unit and EMT 140 reverb, Dual Moog 900 and EML 400 analog sequencers, Emu Digital Memory Sequencer

Written, recorded and designed by Benge at Memetune Studios, England
All music and imagery (c) Memesongs 2018

Euler vs Lagrange



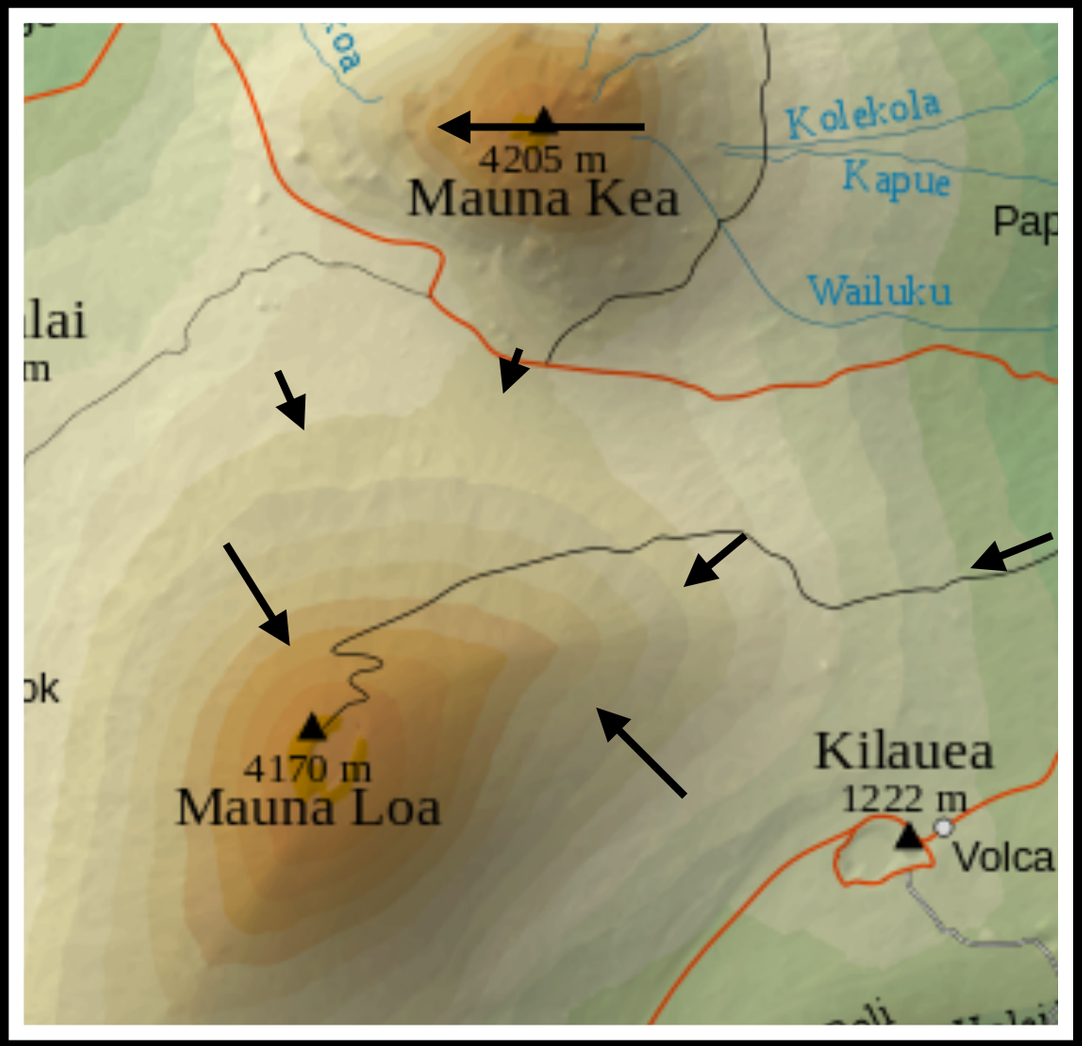
Some vector calculus…

div

$\nabla_{\mathbf{v}}$

grad

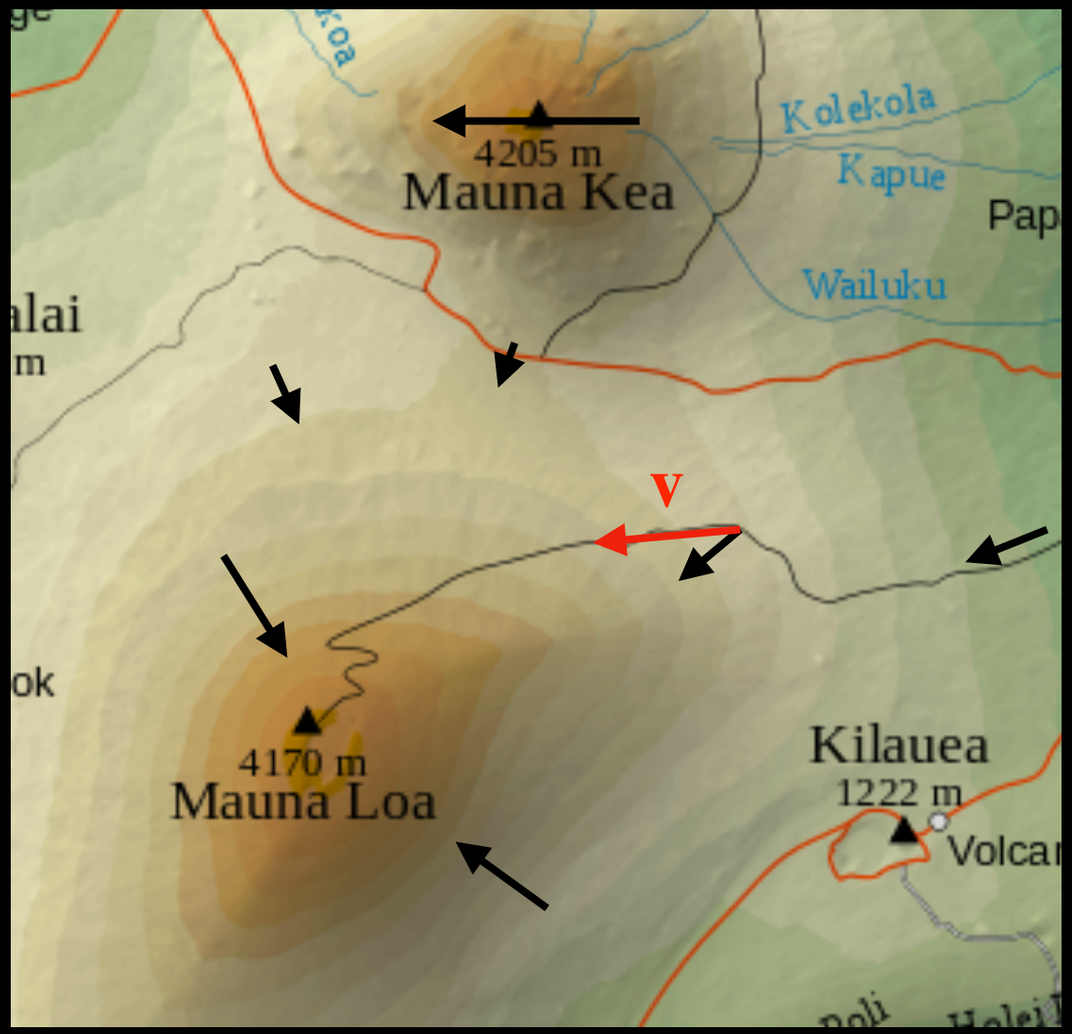
Gradient



© Sémhur / Wikimedia Commons / CC-BY-SA-3.0, or Free Art License

$$\textbf{grad}\ \phi = \left(\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}\right)$$

Directional Derivative

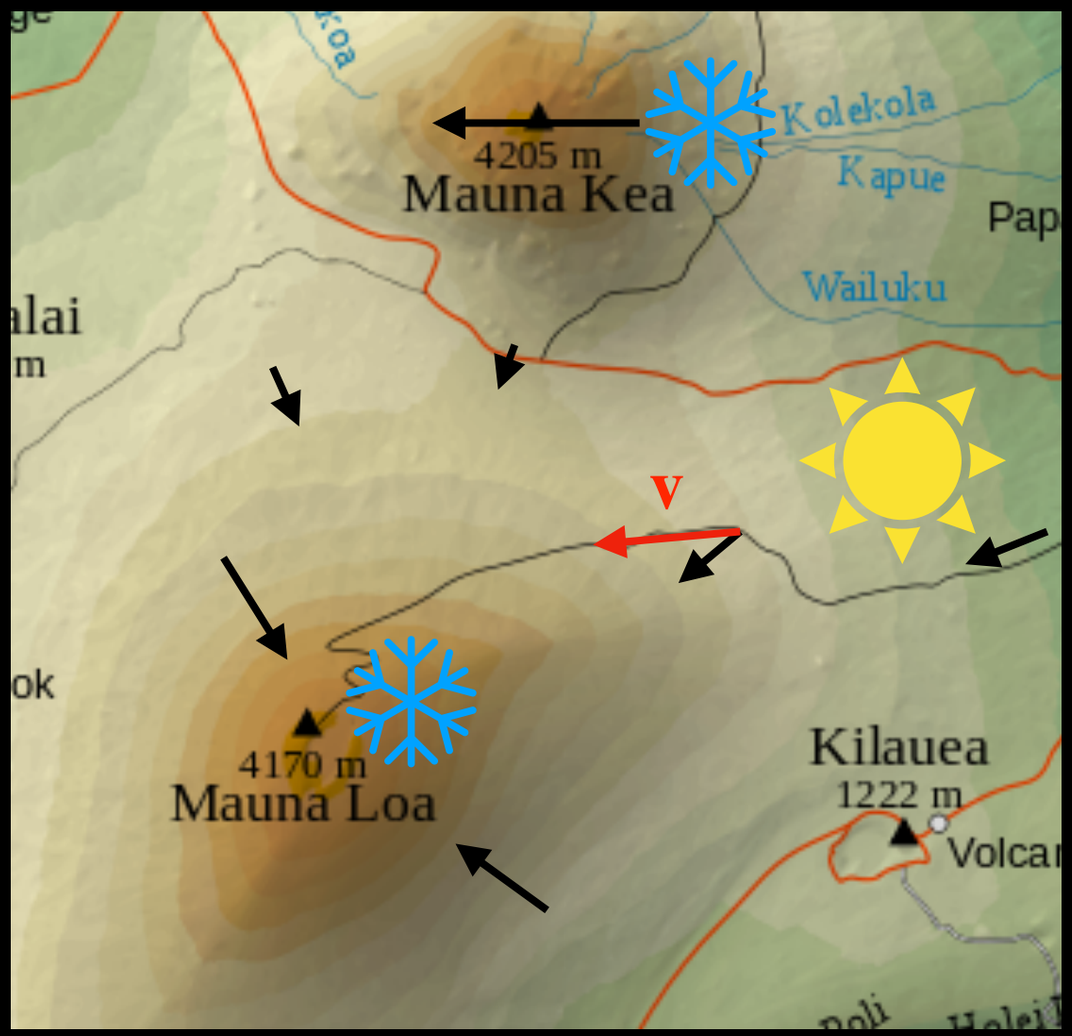


© Sémhur / Wikimedia Commons / CC-BY-SA-3.0, or Free Art License

$$\nabla_{\mathbf{v}}\phi = \mathbf{v} \cdot \mathbf{grad}\ \phi$$

$$= (v_x \frac{\partial \phi}{\partial x}, v_y \frac{\partial \phi}{\partial y})$$

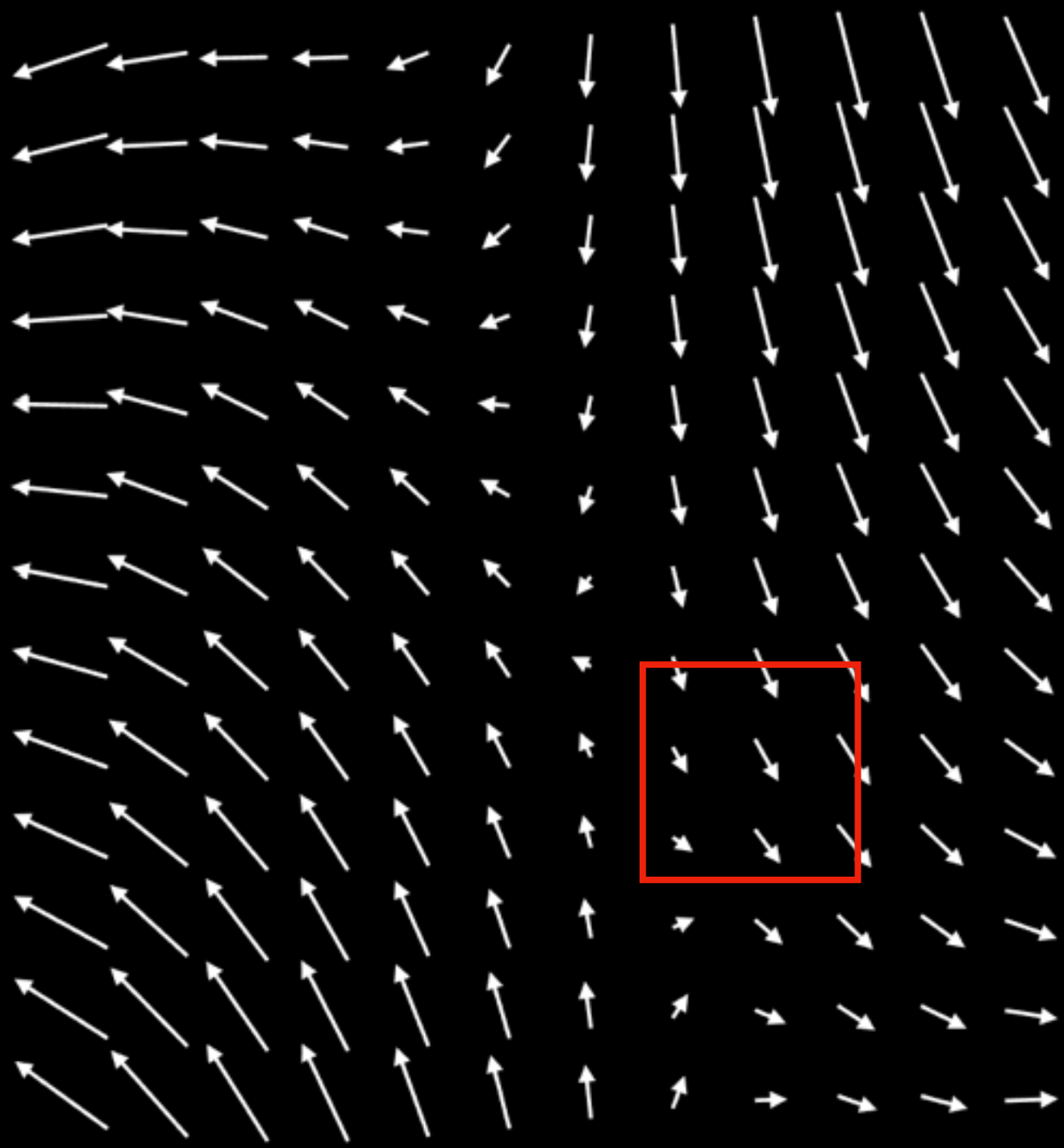
Convective Derivative



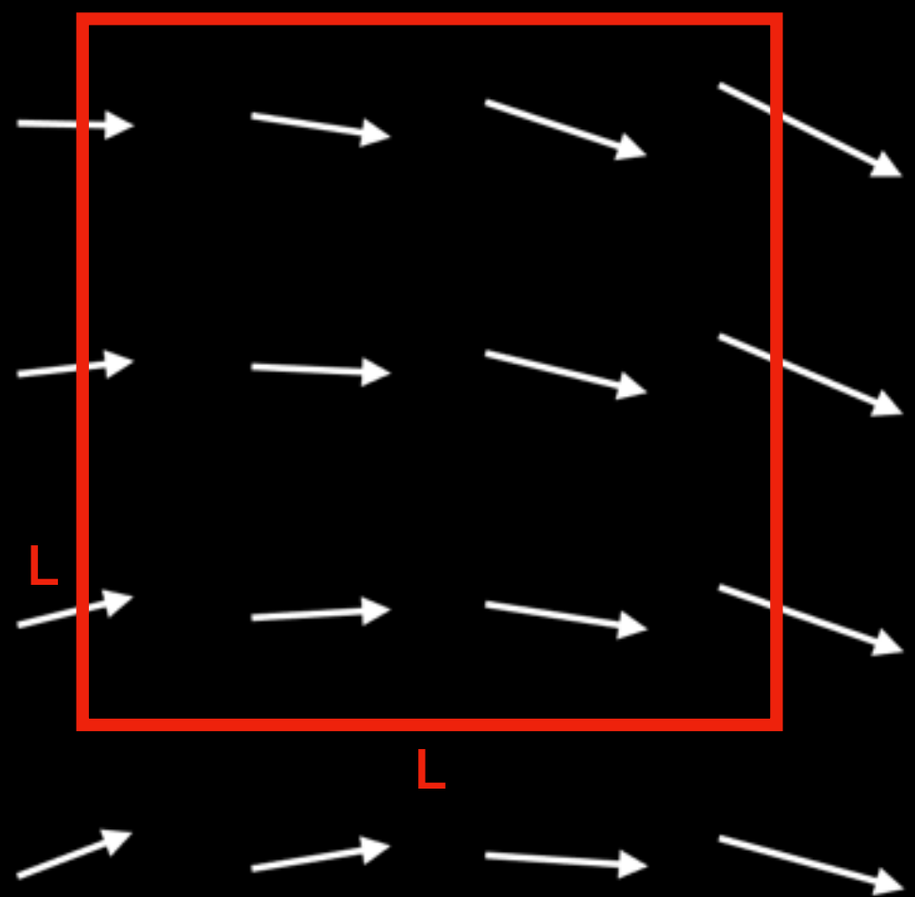
© Sémhur / Wikimedia Commons / CC-BY-SA-3.0, or Free Art License

$$\frac{D\phi}{Dt} = \frac{\partial\phi}{\partial t} + \nabla_{\mathbf{v}}\phi$$

Divergence



Divergence, formally



Net outflow through vertical walls is

$$L\frac{\partial v_0}{\partial x_0}$$

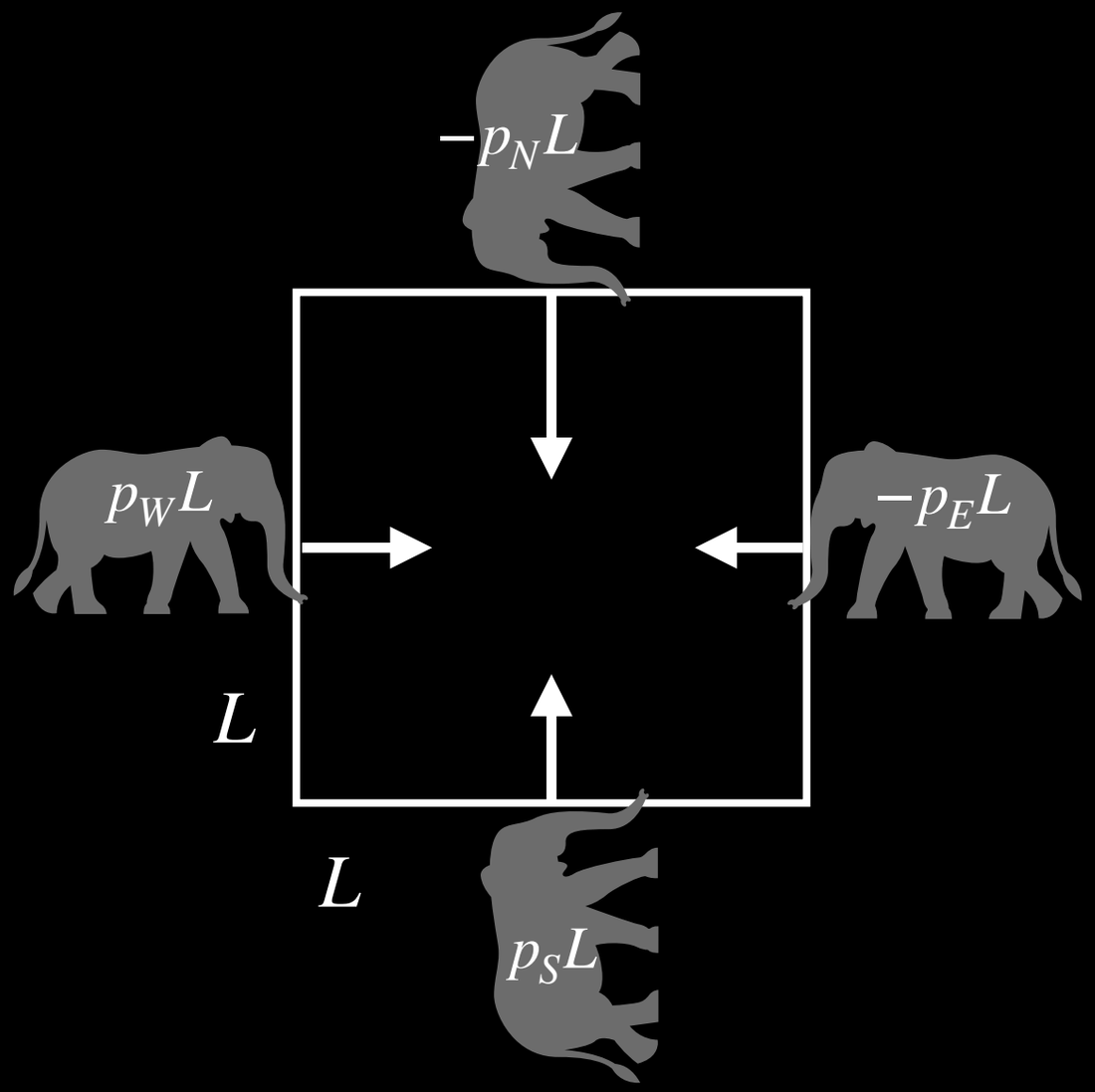
Net outflow through horizontal walls is

$$L\frac{\partial v_1}{\partial x_1}$$

Overall outflow per unit area

$$\textbf{div}\ v = \sum_i \frac{\partial v_i}{\partial x_i}$$

Pressure

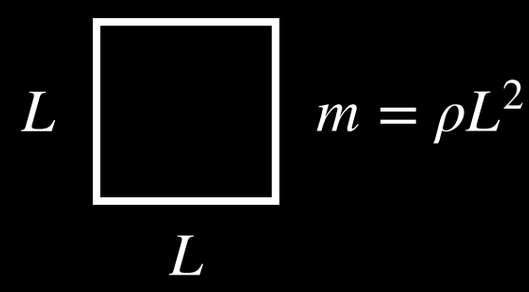


Net force is $L(p_W - p_E, p_S - p_N)$

Force per unit area is $-\textbf{grad}\ p = (\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y})$

The Story of a Parcel

$$ma = F$$



$$\rho L^2 \frac{D\mathbf{v}}{dt} = -L^2 \mathbf{grad}\ p$$

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = -\frac{1}{\rho} \mathbf{grad}\ p$$

First step towards an algorithm

$$\frac{\partial \mathbf{v}}{\partial t} = -\nabla_{\mathbf{v}}\mathbf{v} - \frac{1}{\rho}\mathbf{grad}\ p$$

$$\mathbf{v}(t+\delta t) = \mathbf{v}(t) + \delta t(-\nabla_{\mathbf{v}}\mathbf{v} - \frac{1}{\rho}\mathbf{grad}\ p)$$

But we don’t know pressure and we’re not constraining to incompressible flow yet

$$\mathbf{div}\ \mathbf{v} = 0$$

Navier-Stokes Equations

time $t = 0$, then the evolution of these quantities over time is given by the Navier-Stokes equations [3]:

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

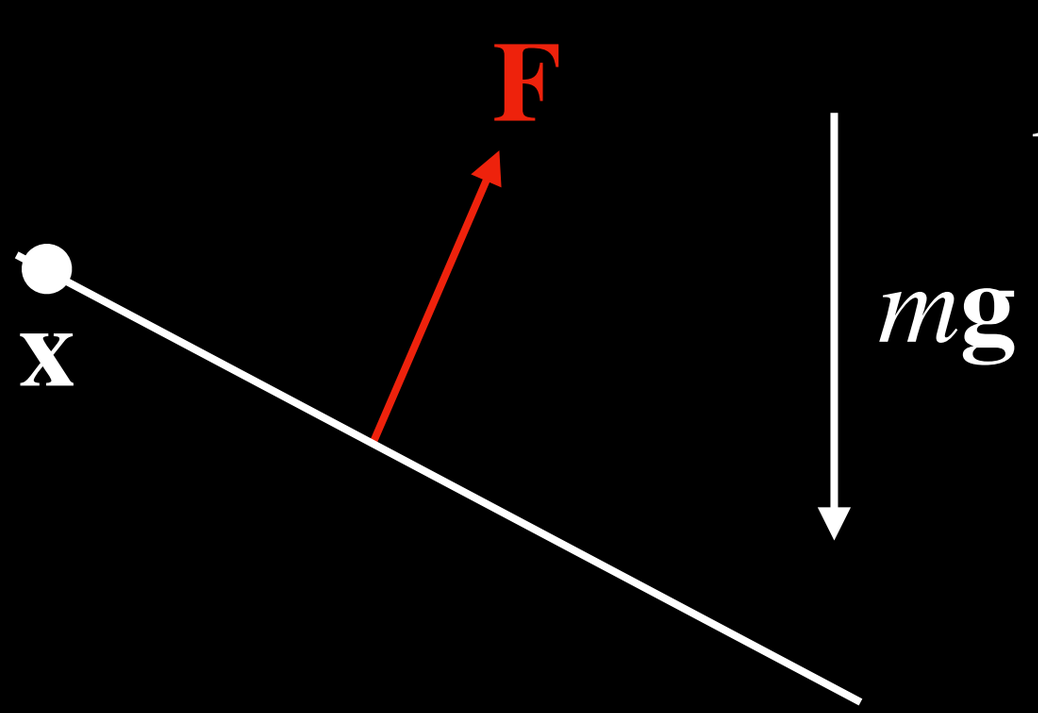
$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f}, \tag{2}$$

where ν is the kinematic viscosity of the fluid, ρ is its density and $\mathbf{f}$ is an external force. Some readers might be unfamiliar with this

Constrained Dynamics

$$m\frac{d^2\mathbf{x}}{dt^2} = m\mathbf{g}+\mathbf{F}$$

P is projection in direction of motion



$$P(m\frac{d^2\mathbf{x}}{dt^2}) = P(m\mathbf{g}+F)$$

$$m\frac{d^2\mathbf{x}}{dt^2} = mP\mathbf{g}$$

Simplified equations

$$\frac{\partial \mathbf{v}}{\partial t} = -\mathbf{v} \cdot \nabla \mathbf{v} - \frac{1}{\rho}\textbf{grad}\ p$$

$$P(\frac{\partial \mathbf{v}}{\partial t}) = P(-\mathbf{v} \cdot \nabla \mathbf{v} - \frac{1}{\rho}\textbf{grad}\ p)$$

$$\frac{\partial \mathbf{v}}{\partial t} = -P(\mathbf{v} \cdot \nabla \mathbf{v})$$

Second step towards algorithm

$$\mathbf{v}(t + \delta t) = P(\mathbf{v}(t) - \mathbf{v} \cdot \nabla \mathbf{v})$$

The general procedure is illustrated in Figure 1. The steps are:

$$\mathbf{w}_0(\mathbf{x}) \overset{\text{add force}}{\longrightarrow} \mathbf{w}_1(\mathbf{x}) \overset{\text{advect}}{\longrightarrow} \mathbf{w}_2(\mathbf{x}) \overset{\text{diffuse}}{\longrightarrow} \mathbf{w}_3(\mathbf{x}) \overset{\text{project}}{\longrightarrow} \mathbf{w}_4(\mathbf{x}).$$

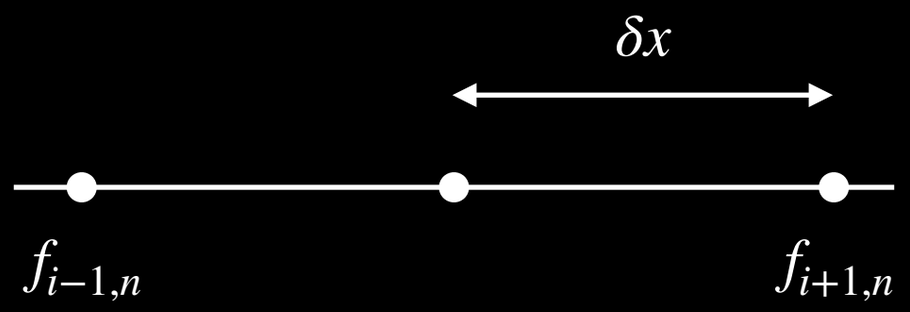
The solution at time $t + \Delta t$ is then given by the last velocity field:

Advection the hard way

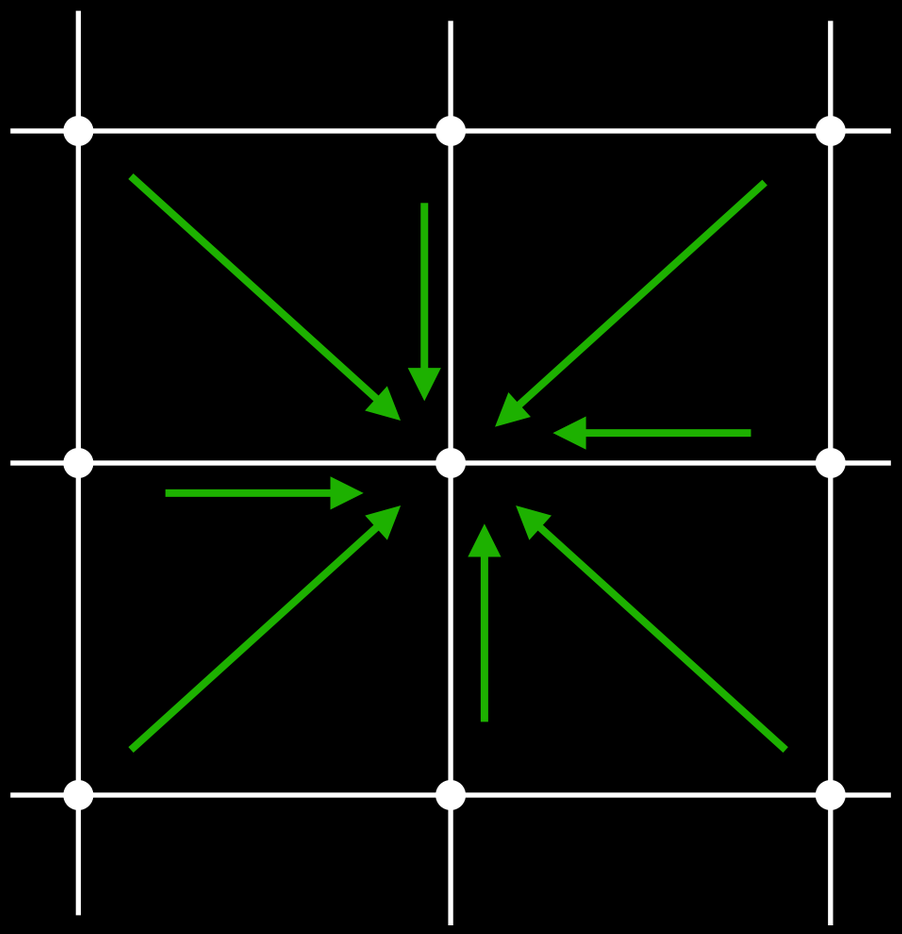
$$\frac{\partial \mathbf{v}}{\partial t} = -P(\mathbf{v} \cdot \nabla \mathbf{v})$$

Suppose we store our velocities on a grid so $f(i\delta x, n\delta t) = f_{i,n}$

$$\frac{\partial f}{\partial x} = \frac{1}{2\delta x}(f_{i+1,n} - f_{i-1,n})$$

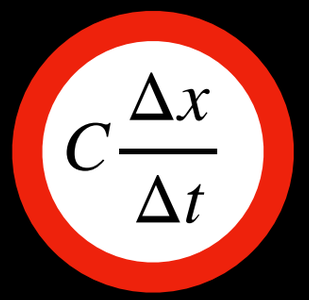


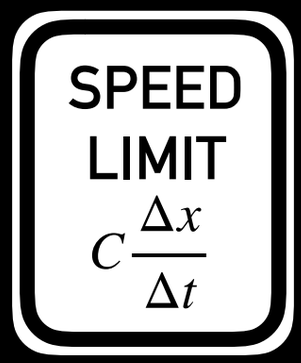
Information flow



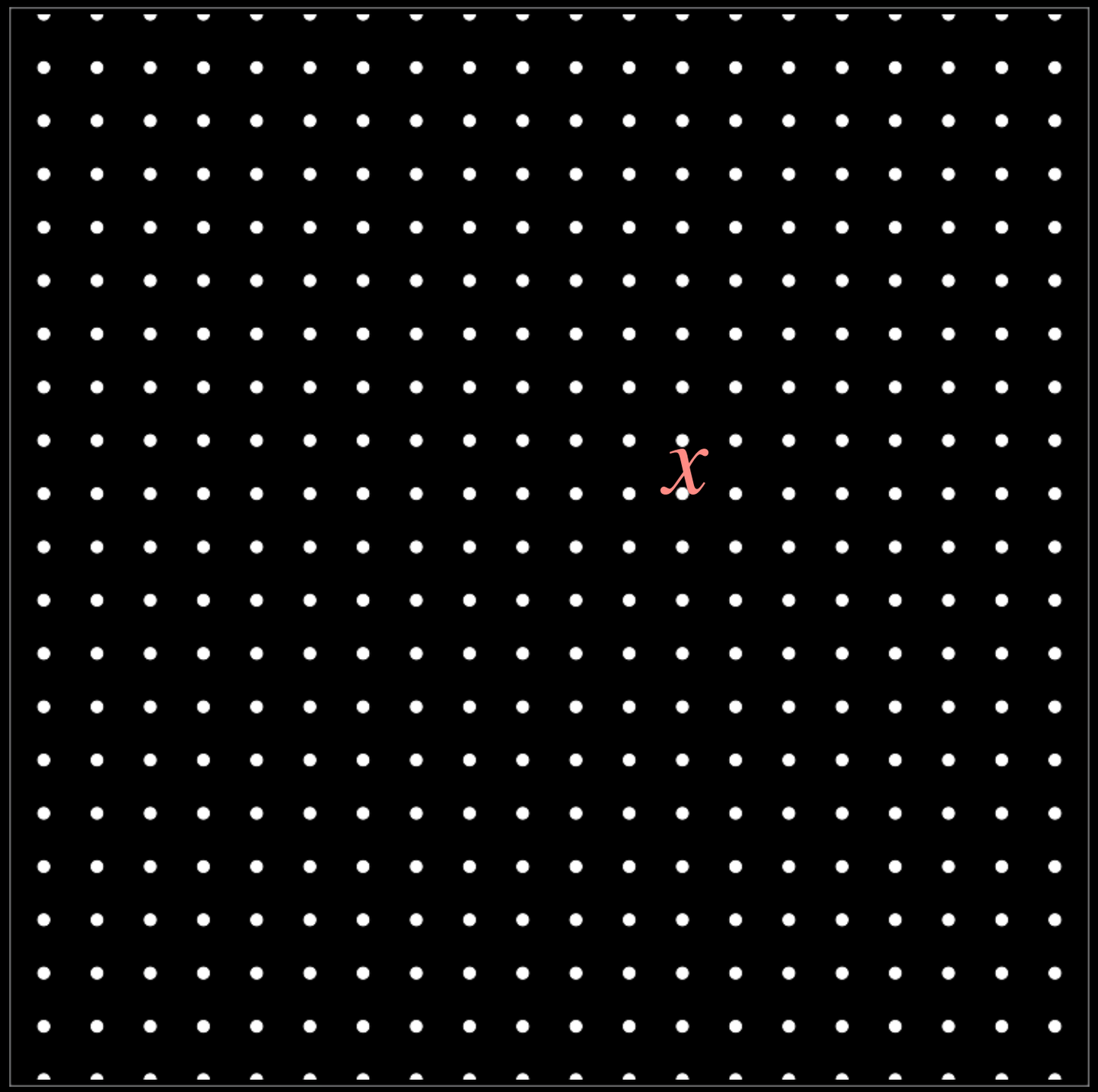
Courant condition

$$u < C\frac{\Delta x}{\Delta t}$$





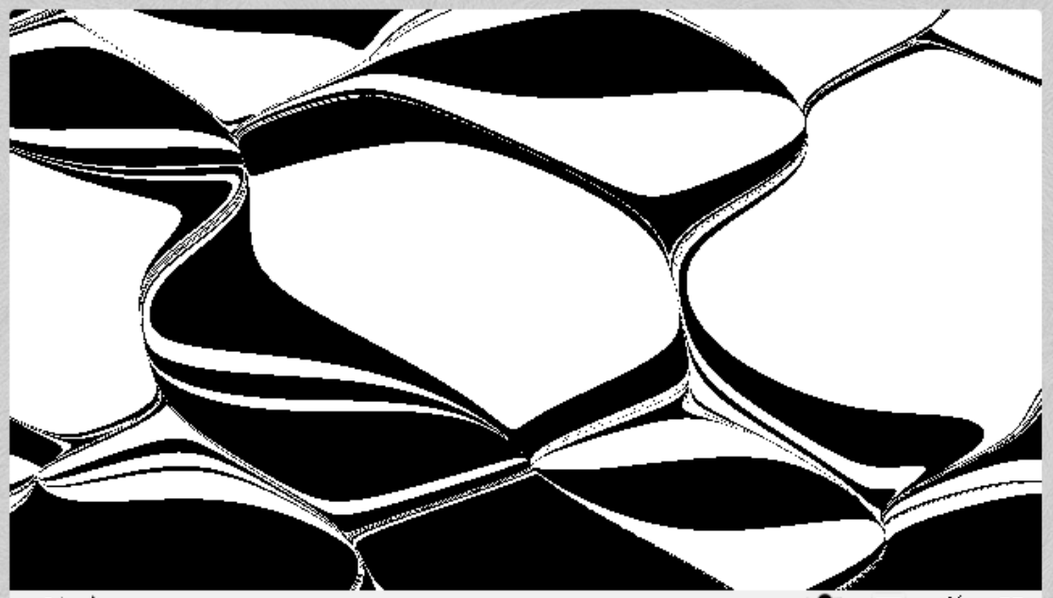
Semi-Lagrangian



Shadertoy BETA | dan vector field

shadertoy.com/view/ltVyR3

Shadertoy | Search... | Welcome dpiponi | Browse | New | Logout



11.91 59.9 fps REC

dan vector field

+ share 0

0 views, Tags: vectorfield

Created by dpiponi in -

Experiment with vector fields

private Save

Comments (0)

Your comment...

?

Post

Image

Shader Inputs

```
vec2 flow2(float a, vec2 k, float phase, vec2 x) {
    return a*cos(dot(k, x)+phase)*vec2(k.y, -k.x);
}

vec2 flow(float a, vec2 k, float phase, vec2 x) {
    return a*cos(dot(k, x)+phase)*k;
}

float phi(float a, vec2 k, float phase, vec2 x) {
    return a*sin(dot(k, x)+phase);
}

vec4 flowImage(vec2 uv) {
    float dt = 0.00025*iTime;
    // float t = 0.01*iTime;
    for (int i = 0; i < 100; ++i) {
        vec2 v = vec2(0.0);
        v += flow(0.3, vec2(3.0, 2.0), 0.0, uv);
        v += flow(0.1, vec2(10.0, 0.0), 0.0, uv);
        v += flow(0.1, vec2(0.0, 9.0), 0.0, uv);
        v += flow(0.1, vec2(-13.0, 7.0), 0.0, uv);
        v += flow(0.05, vec2(22.0, 12.0), 0.0, uv);

        uv = uv-dt*v;
    }

    // Time varying pixel color
    float col = cos(50.*uv.x)*cos(50.*uv.y) > 0.0 ? 1.0 : 0.0;

    // Output to screen
    return vec4(col, col, col, 1.0);
}

vec4 phiImage(vec2 uv) {
```

951 chars M ?

iChannel0 iChannel1 iChannel2 iChannel3

How To API Terms & Privacy Feedback Events Changelog Roadmap About Follow us Support us by Beautypi

Let’s make up some incompressible flows

Choose k and a so that they are perpendicular

$$\mathbf{a} \cdot \mathbf{k} = 0$$

$$\mathbf{v} = \mathbf{a} \cos(\mathbf{k} \,.\, \mathbf{x})$$

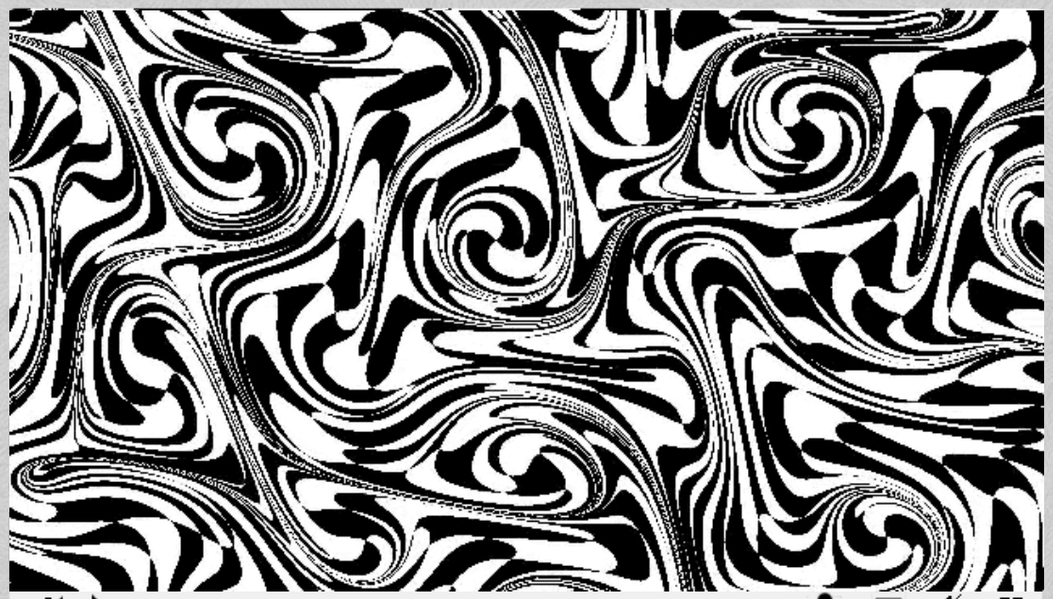
$$\mathbf{div}\ \mathbf{v} = -\mathbf{k} \cdot \mathbf{a} \sin(\mathbf{k} \,.\, \mathbf{x}) = 0$$

So v is an incompressible flow.
A sum of a number of terms of this form is also incompressible.
Let’s try it in ShaderToy.

Shadertoy BETA | dan vector field

shadertoy.com/view/ltVyR3

Shadertoy | Search... | Welcome dpiponi | Browse | New | Logout



11.91 59.9 fps REC VR

dan vector field

0 ♥

0 views, Tags: vectorfield

Created by **dpiponi** in -

Experiment with vector fields

private | Save

Comments (0)

Your comment...

?

Post

Image

Shader Inputs

```
vec2 flow2(float a, vec2 k, float phase, vec2 x) {
    return a*cos(dot(k, x)+phase)*vec2(k.y, -k.x);
}

vec2 flow(float a, vec2 k, float phase, vec2 x) {
    return a*cos(dot(k, x)+phase)*k;
}

float phi(float a, vec2 k, float phase, vec2 x) {
    return a*sin(dot(k, x)+phase);
}

vec4 flowImage(vec2 uv) {
    float dt = 0.00025*iTime;
    // float t = 0.01*iTime;
    for (int i = 0; i < 100; ++i) {
        vec2 v = vec2(0.0);
        v += flow(0.3, vec2(3.0, 2.0), 0.0, uv);
        v += flow(0.1, vec2(10.0, 0.0), 0.0, uv);
        v += flow(0.1, vec2(0.0, 9.0), 0.0, uv);
        v += flow(0.1, vec2(-13.0, 7.0), 0.0, uv);
        v += flow(0.05, vec2(22.0, 12.0), 0.0, uv);

        uv = uv-dt*v;
    }

    // Time varying pixel color
    float col = cos(50.*uv.x)*cos(50.*uv.y) > 0.0 ? 1.0 : 0.0;

    // Output to screen
    return vec4(col, col, col, 1.0);
}
```

951 chars M ?

iChannel0 iChannel1 iChannel2 iChannel3

How To API Terms & Privacy Feedback Events Changelog Roadmap About Follow us Support us by **Beautypi**

Now try the opposite

Choose k and a so that they are parallel, i.e.

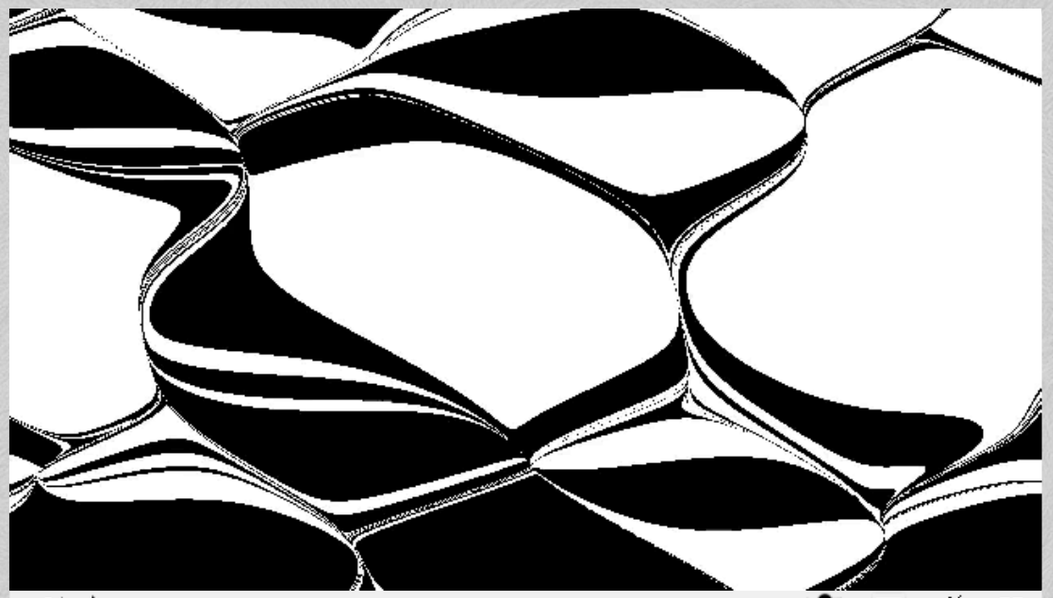
$$\mathbf{v} = \mathbf{k}\cos(\mathbf{k} \cdot \mathbf{x})$$

Let’s try it in ShaderToy.

Shadertoy BETA | dan vector field

shadertoy.com/view/ltVyR3

Shadertoy | Search... | Welcome dpiponi | Browse | New | Logout



11.91 59.9 fps REC VR

dan vector field

0 views, Tags: vectorfield

Created by dpiponi in -

Experiment with vector fields

private | Save

Comments (0)

Your comment...

Post

Image

Shader Inputs

```glsl
vec2 flow2(float a, vec2 k, float phase, vec2 x) {
    return a*cos(dot(k, x)+phase)*vec2(k.y, -k.x);
}

vec2 flow(float a, vec2 k, float phase, vec2 x) {
    return a*cos(dot(k, x)+phase)*k;
}

float phi(float a, vec2 k, float phase, vec2 x) {
    return a*sin(dot(k, x)+phase);
}

vec4 flowImage(vec2 uv) {
    float dt = 0.00025*iTime;
    // float t = 0.01*iTime;
    for (int i = 0; i < 100; ++i) {
        vec2 v = vec2(0.0);
        v += flow(0.3, vec2(3.0, 2.0), 0.0, uv);
        v += flow(0.1, vec2(10.0, 0.0), 0.0, uv);
        v += flow(0.1, vec2(0.0, 9.0), 0.0, uv);
        v += flow(0.1, vec2(-13.0, 7.0), 0.0, uv);
        v += flow(0.05, vec2(22.0, 12.0), 0.0, uv);

        uv = uv-dt*v;
    }

    // Time varying pixel color
    float col = cos(50.*uv.x)*cos(50.*uv.y) > 0.0 ? 1.0 : 0.0;

    // Output to screen
    return vec4(col, col, col, 1.0);
}
```

951 chars | M

iChannel0 | iChannel1 | iChannel2 | iChannel3

How To | API | Terms & Privacy | Feedback | Events | Changelog | Roadmap | About | Follow us | Support us | by Beautypi

These flows are gradients

Choose k and a so that they are parallel, i.e.

$$\mathbf{v} = \mathbf{k}\cos(\mathbf{k} \,.\, \mathbf{x})$$

$$\mathbf{v} = \textbf{grad}\ \phi$$

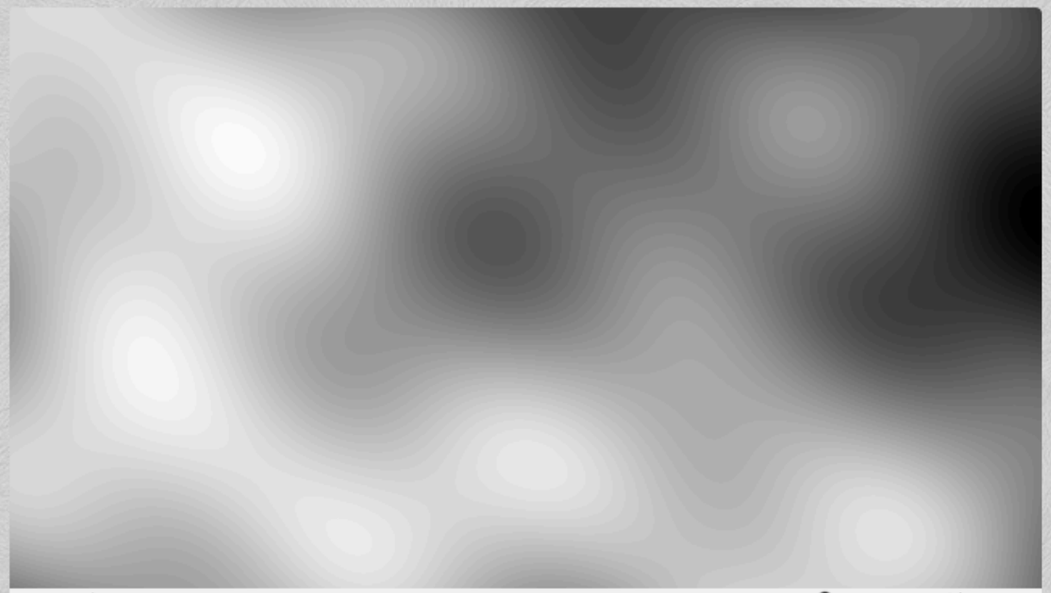
$$\phi = \sin(\mathbf{k} \cdot \mathbf{x})$$

Let’s view ϕ in ShaderToy.

Shadertoy BETA | dan vector field

shadertoy.com/view/ltVyR3

Shadertoy | Search... | Welcome dpiponi | Browse | New | Logout



0.00 60.0 fps REC

dan vector field

0 views, Tags: vectorfield

Created by dpiponi in -

0

Experiment with vector fields

private | Save

Comments (0)

Your comment...

Post

Image

Shader Inputs

```
21          v += flow2(0.1, vec2(-13.0, 7.0), 0.0, uv);
22          v += flow2(0.05, vec2(22.0, 12.0), 0.0, uv);
23
24          uv = uv-dt*v;
25      }
26
27      // Time varying pixel color
28      float col = cos(50.*uv.x)*cos(50.*uv.y) > 0.0 ? 1.0 : 0.0;
29
30      // Output to screen
31      return vec4(col, col, col, 1.0);
32  }
33
34  vec4 phiImage(vec2 uv) {
35      float v = 0.0;
36      v += phi(0.3, vec2(3.0, 2.0), 0.0, uv);
37      v += phi(0.1, vec2(10.0, 0.0), 0.0, uv);
38      v += phi(0.1, vec2(0.0, 9.0), 0.0, uv);
39      v += phi(0.1, vec2(-13.0, 7.0), 0.0, uv);
40      v += phi(0.05, vec2(22.0, 12.0), 0.0, uv);
41
42      // Output to screen
43      return vec4(0.5+v, 0.5+v, 0.5+v, 1.0);
44  }
45
46
47  void mainImage( out vec4 fragColor, in vec2 fragCoord )
48  {
49      // Normalized pixel coordinates (from 0 to 1)
50      vec2 uv = fragCoord/iResolution.xy;
51      fragColor = phiImage(uv);
52  //    fragColor = flowImage(uv);
53
54  }
```

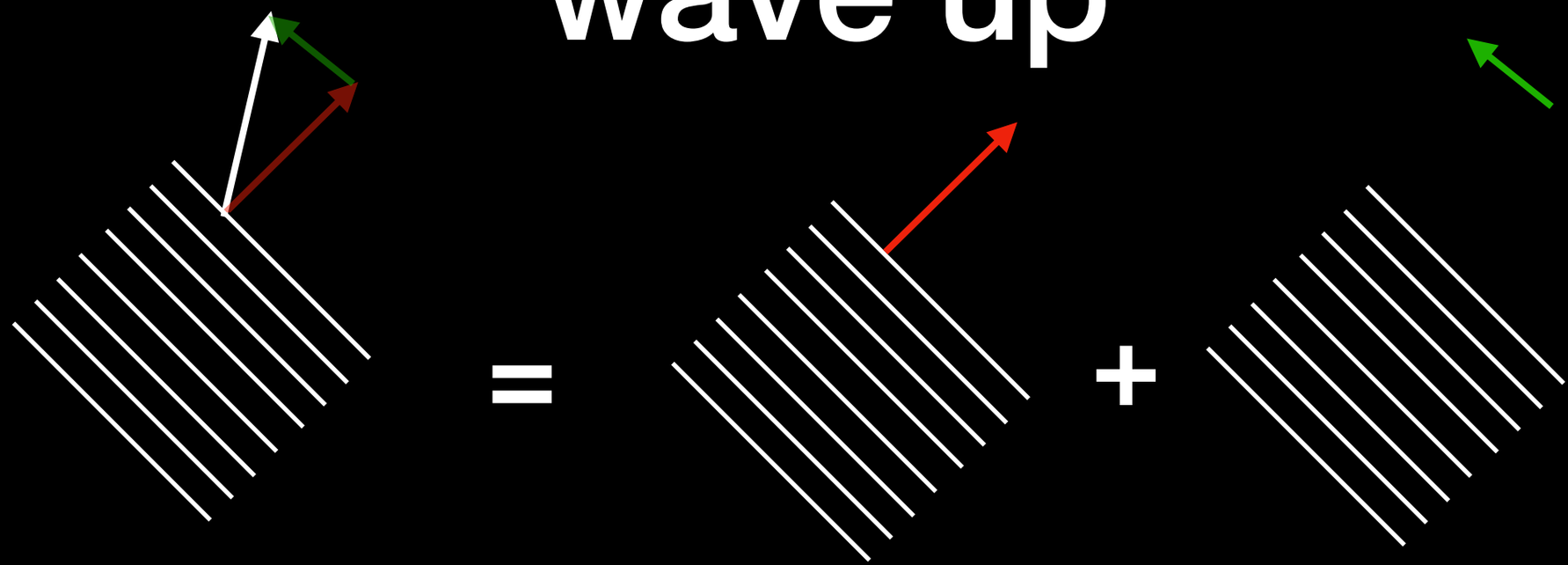
955 chars | M | ?

iChannel0 | iChannel1 | iChannel2 | iChannel3

How To | API | Terms & Privacy | Feedback | Events | Changelog | Roadmap | About | Follow us | Support us | by Beautypi

capture (7).webm | Show All

We can split any velocity wave up



$$\mathbf{v} = \mathbf{a} \cos \mathbf{k} \cdot \mathbf{x}$$

Split v into parallel and perpendicular parts

$$\mathbf{a} = \mathbf{a}^{\parallel} + \mathbf{a}^{\perp}$$

$$\mathbf{v} = \mathbf{a} \cos \mathbf{k} \cdot \mathbf{x}$$

$$\mathbf{v} = \mathbf{a}^{\parallel} \cos \mathbf{k} \cdot \mathbf{x} + \mathbf{a}^{\perp} \cos \mathbf{k} \cdot \mathbf{x}$$

Helmholtz Decomposition

$$\mathbf{v} = \mathbf{a}^{\parallel} \cos \mathbf{k} \cdot \mathbf{x} + \mathbf{a}^{\perp} \cos \mathbf{k} \cdot \mathbf{x}$$

$$\mathbf{v} = \mathbf{v}^{\parallel} + \mathbf{v}^{\perp}$$

$$\mathbf{v}^{\parallel} = \nabla \phi$$ **Gradient part**

$$\nabla \cdot \mathbf{v}^{\perp} = 0$$ **Incompressible part**

But from Fourier we know that any vector field can be written as a sum of plane waves so we can

1. **split any vector field into gradient and incompresible parts and**
2. **we have given a practical algorithm for doing so: use a Fourier transform**

The Projection

Indeed the diffusion operator and the projection operators in the Fourier domain are

$$\mathbf{I} - \nu\Delta t\nabla^2 \longrightarrow 1 + \nu\Delta t k^2 \quad \text{and}$$

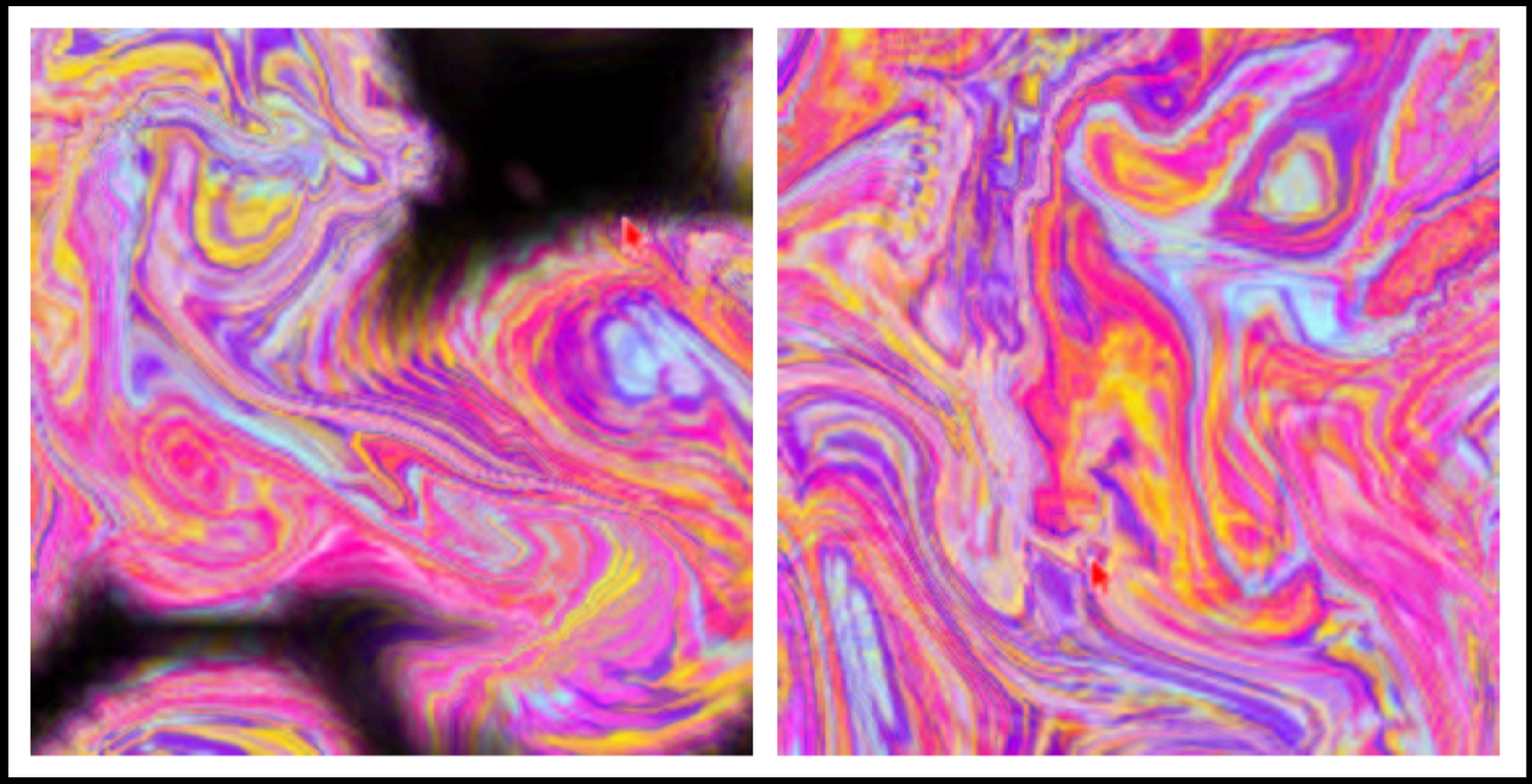
$$\mathbf{P}\mathbf{w} \longrightarrow \hat{\mathbf{P}}\hat{\mathbf{w}}(\mathbf{k}) = \hat{\mathbf{w}}(\mathbf{k}) - \frac{1}{k^2}(\mathbf{k}\cdot\hat{\mathbf{w}}(\mathbf{k}))\mathbf{k},$$

where $k = |\mathbf{k}|$. The operator $\hat{\mathbf{P}}$ projects the vector $\hat{\mathbf{w}}(\mathbf{k})$ onto the plane which is normal to the wave number $\mathbf{k}$. The Fourier transform

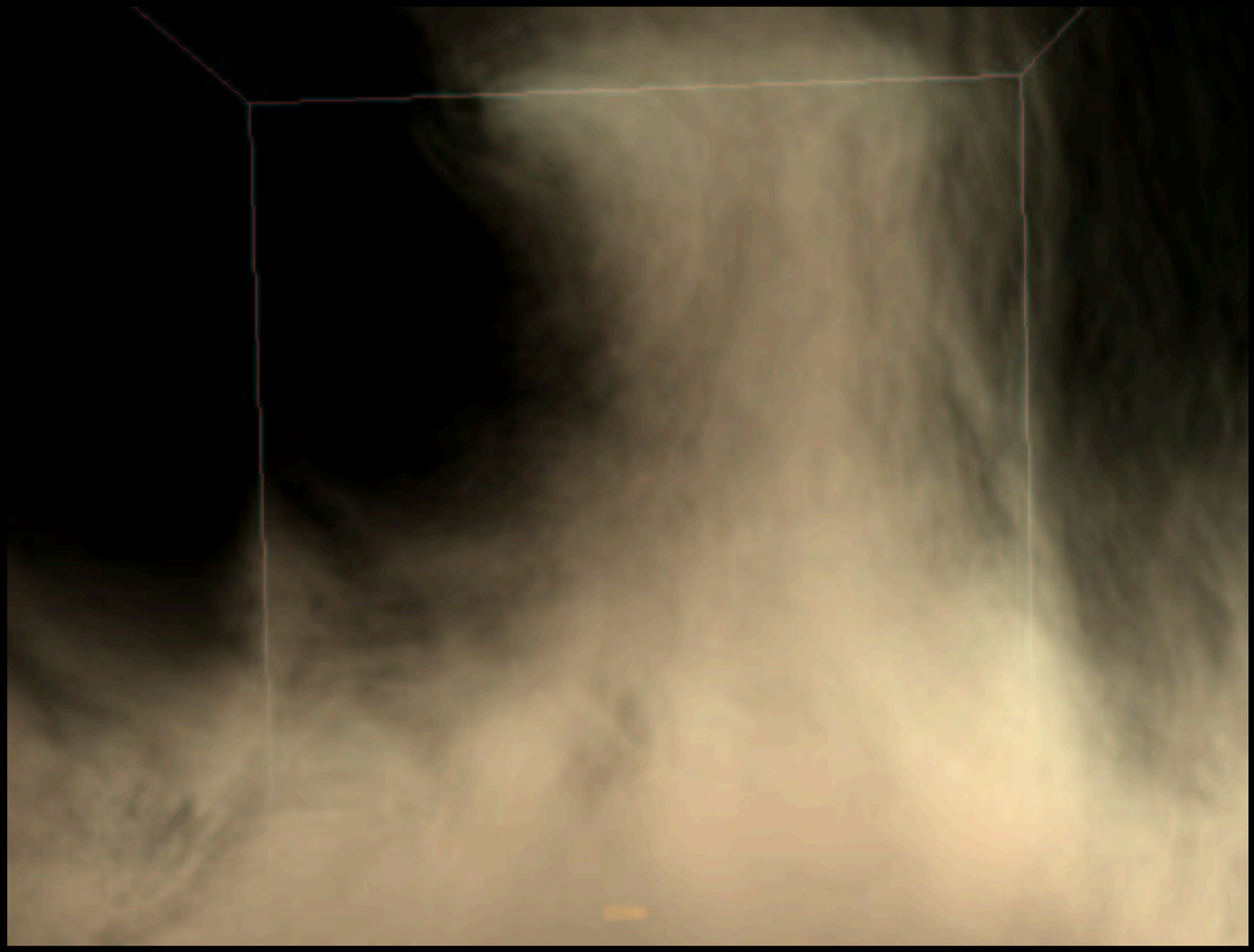
Complete Method

* Advect
* Project

Examples from paper

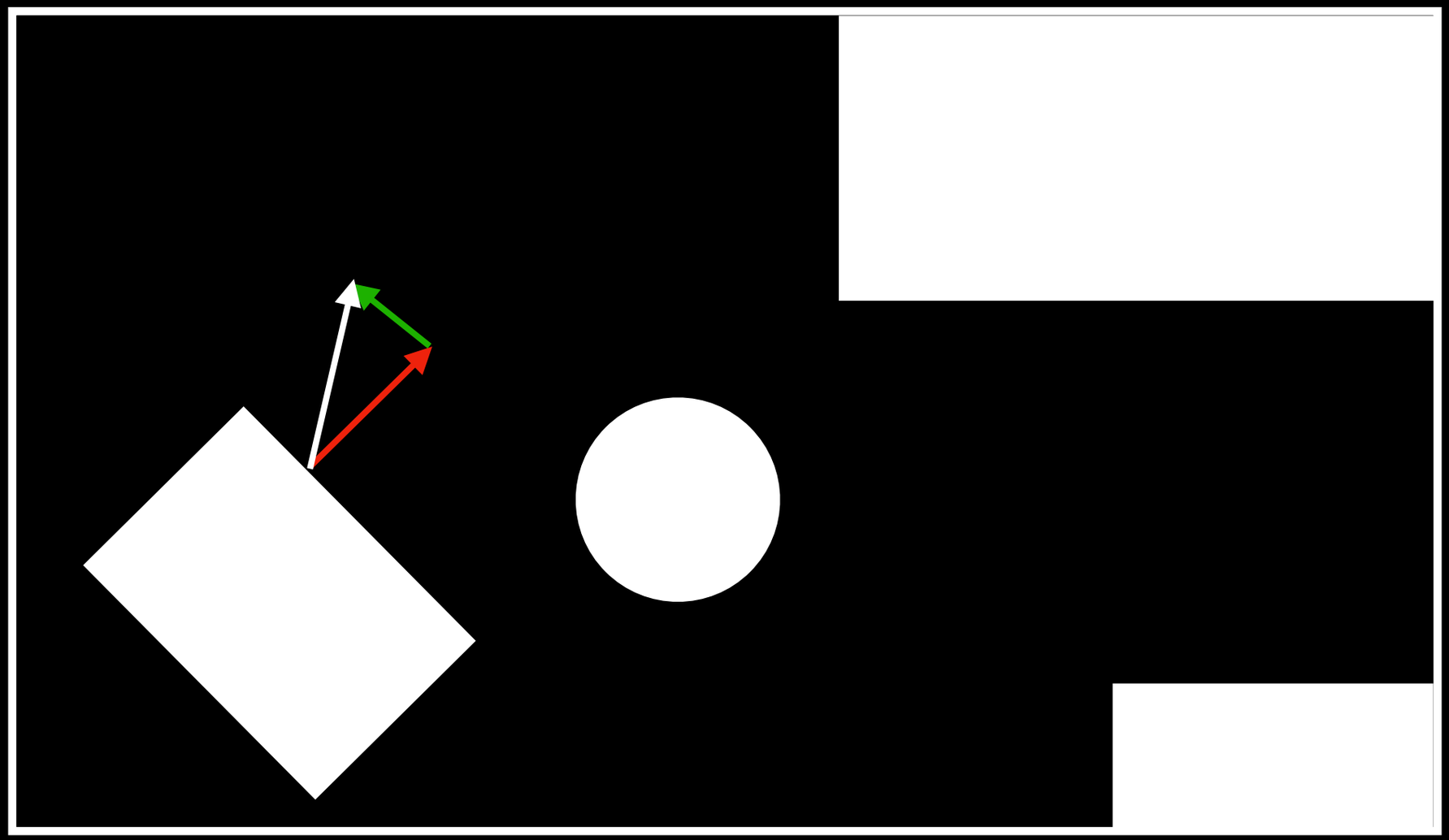


Examples from paper

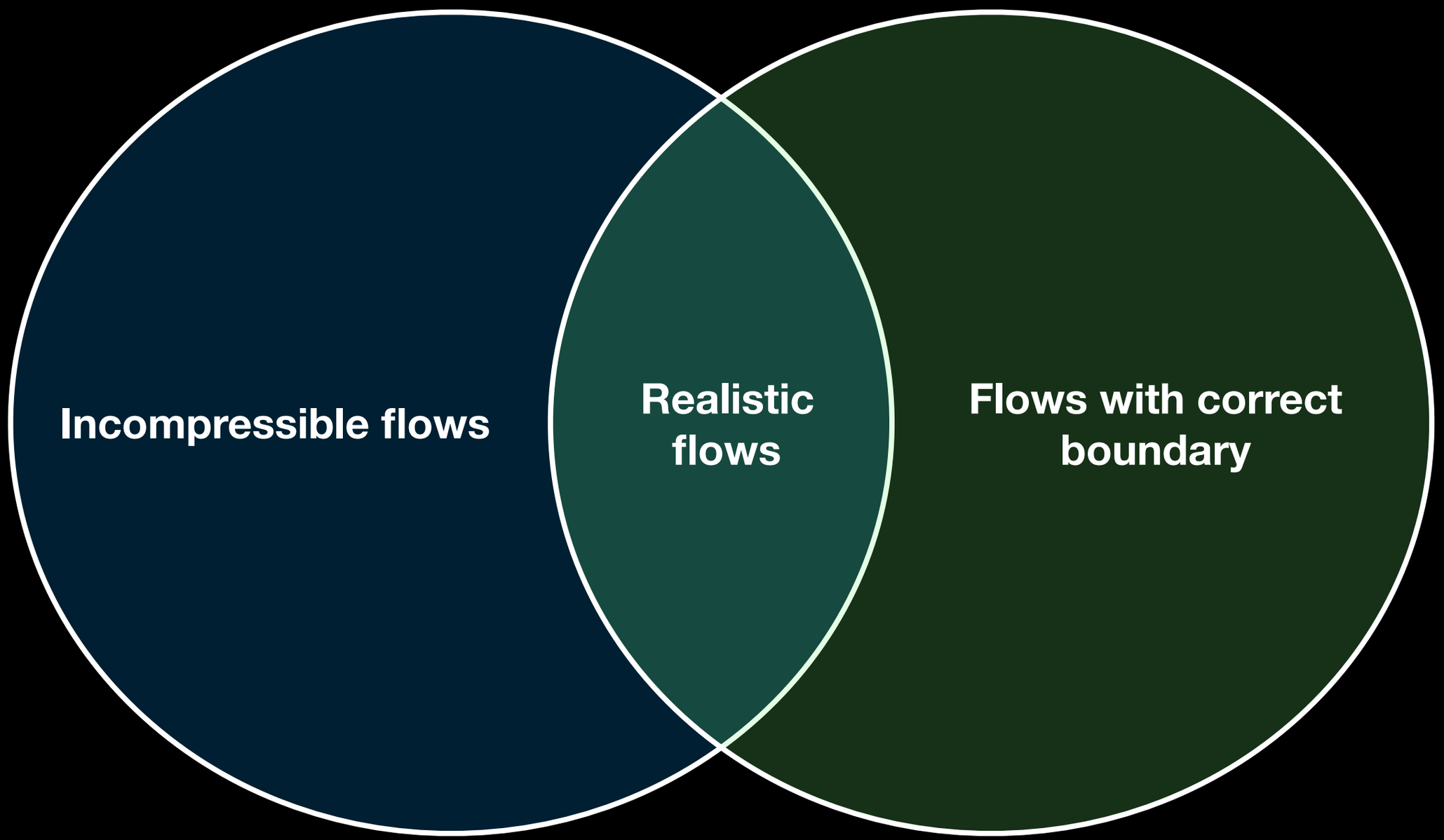


Interactive demo 1999 style

What about boundaries?



A Venn diagram



Alternating Projection

Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2008)
M. Gross and D. James (Editors)

Low Viscosity Flow Simulations for Animation

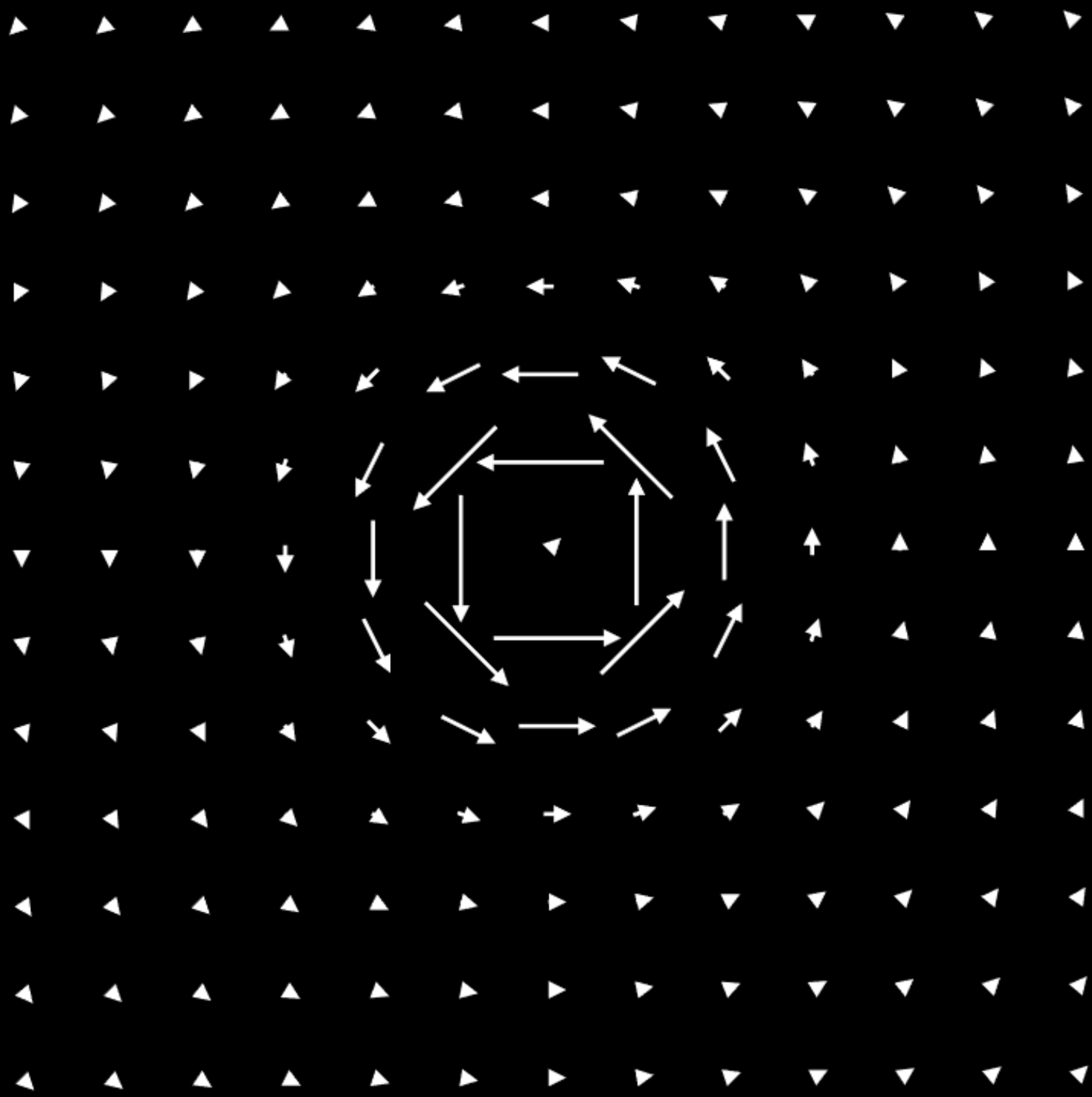
Jeroen Molemaker[1,2], Jonathan M. Cohen[1,3], Sanjit Patel[1], Jonyong Noh[1,4]

[1]Rhythm and Hues, USA
[2]Institute of Geophysics and Planetary Physics, UCLA, USA
[3]NVIDIA, USA
[4]Graduate School of Culture Technology, KAIST, Korea

Abstract

We present a combination of techniques to simulate turbulent fluid flows in 3D. Flow in a complex domain is modeled using a regular rectilinear grid with a finite-difference solution to the incompressible Navier-Stokes equations. We propose the use of the QUICK advection algorithm over a globally high resolution grid. To calculate pressure over the grid, we introduce the Iterated Orthogonal Projection (IOP) framework. In IOP a series of orthogonal projections ensures that multiple conditions such as non-divergence and boundary conditions arising through complex domains shapes or moving objects will be satisfied simultaneously to specified accuracy. This framework allows us to use a simple and highly efficient multigrid method to enforce non-divergence in combination with complex domain boundary conditions. IOP is amenable to GPU implementation, resulting in over an order of magnitude improvement over a CPU-based solver. We analyze the impact of these algorithms on the turbulent energy cascade in simulated fluid flows and the resulting visual quality.

Viscosity



Back and Forth Error Compensation and Correction

An Unconditionally Stable
MacCormack Method*

Andrew Selle† Ronald Fedkiw† ByungMoon Kim‡
Yingjie Liu§ Jarek Rossignac‡

June 29, 2007

Abstract

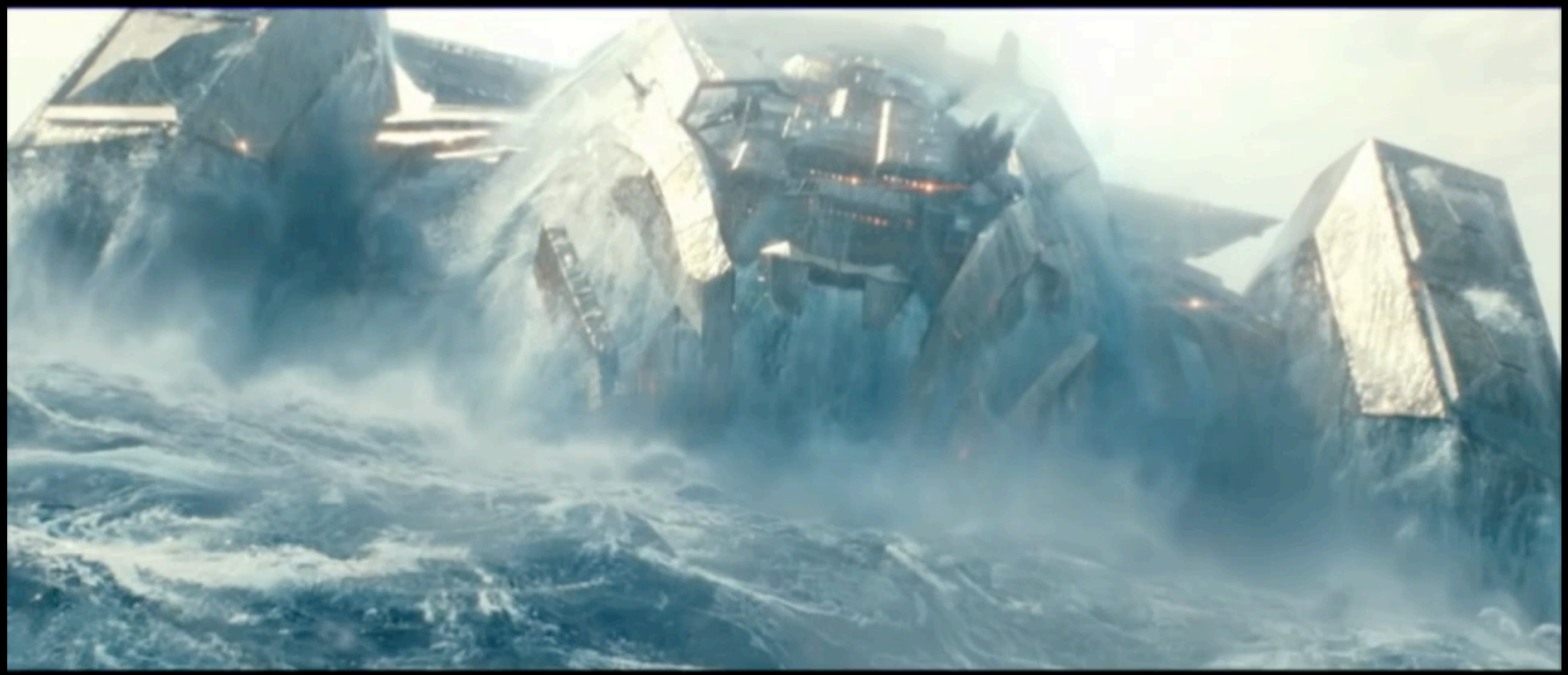
The back and forth error compensation and correction (BFECC) method advects the solution forward and then backward in time. The result is compared to the original data to estimate the error. Although inappropriate for parabolic and other non-reversible partial differential equations, it is useful for often troublesome advection terms. The error estimate is used to correct the data before advection raising the method to second order accuracy, even though each individual step is only first order accurate. In this paper, we rewrite the MacCormack method to illustrate that it estimates the error in the same exact fashion as BFECC. The difference is that the MacCormack method uses this error estimate to correct the already computed forward advected data. Thus, it does not require the third advection step in BFECC reducing the cost of the method while still

Plume Fire/Smoke/Dust

2011-05-30 bs113 76527

© Paramount Pictures

Plume spray



© Universal Pictures

Loon

Optimal trajectory planning for balloons in wind currents can be carried out by solving the Hamilton-Jacobi-Bellman equation.
This is amenable to a semi-Lagrangian approach.

What about today?

Many hybrid algorithms now

An important one is PIC/FLIP

Hybrid method

* **Maintain a set of particles**
* **Transfer velocities to grid**
* **Project**
* **~~Transfer velocities back to particles~~**
* **Add velocity changes back to particles**

Credits

* Jos Stam of course for giving us so a wealth of graphics techniques that go beyond fluid dynamics
* My ex-colleagues Olivier Maury and Ian Sachs who I worked with on Plume

The End