# Lava Flow Simulation

GraphDeco

GRAPHics and Design with hEterogeneous COntent

ENS
ÉCOLE NORMALE SUPÉRIEURE

PSL RESEARCH UNIVERSITY PARIS

Inria

# Why lava flows ?

# Why lava flows ?

- Entertainment

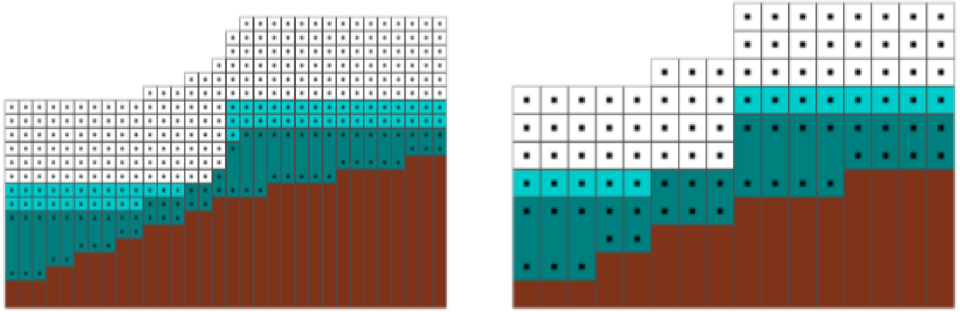
# Why lava flows ?

- Entertainment

- Prevention

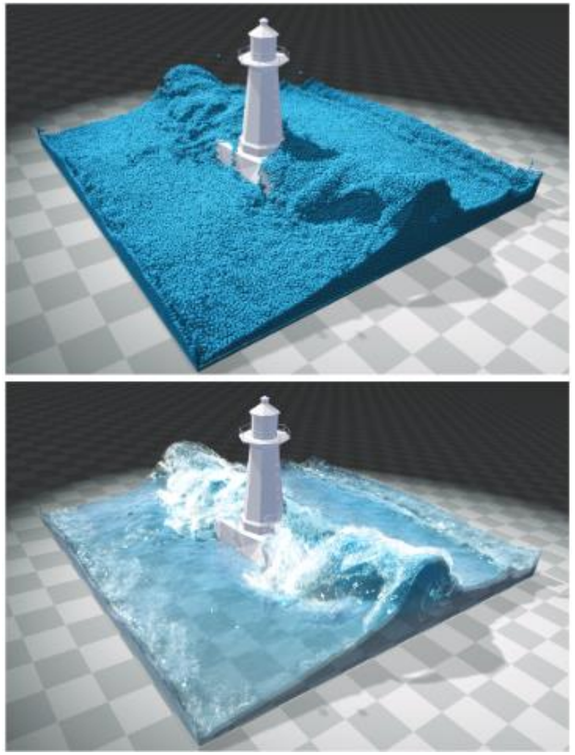
# Existing methods: Fluid simulation

## Eulerian models



from *Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid*, Müller et al.

## Lagrangian models



from *SPH Fluids in Computer Graphics*, Ihmsen et al.

# Existing methods: From 3D to 2D

## 3D Navier-Stokes Equation

$$\rho(\frac{d\vec{u}}{dt} + (\vec{u} \cdot \nabla)\vec{u}) = \rho\vec{g} - \nabla \cdot \mathrm{pI} + \rho\vec{F}_{ext}$$

| | |
|---|---|
| $\rho$ | Density |
| $\vec{u}$ | Velocity |
| $\vec{g}$ | Gravity |
| $p$ | Pressure |
| $I$ | Identity tensor |
| $\vec{F}_{ext}$ | External forces |

$\longrightarrow$

## 2D Shallow Water Equations

$$\frac{Du}{Dt} = -\vec{g}\nabla S + \vec{F}_{ext}$$

| | |
|---|---|
| $\vec{u}$ | Velocity |
| $\vec{g}$ | Gravity |
| $\vec{F}_{ext}$ | External forces |
| $S$ | Surface |

# Existing methods: Viscosity

## Explicit schemes

x(t+dt) = f(x(t))

+ Fast per time step

- Needs small steps to be stable

- Steps sizes depends on 1/viscosity

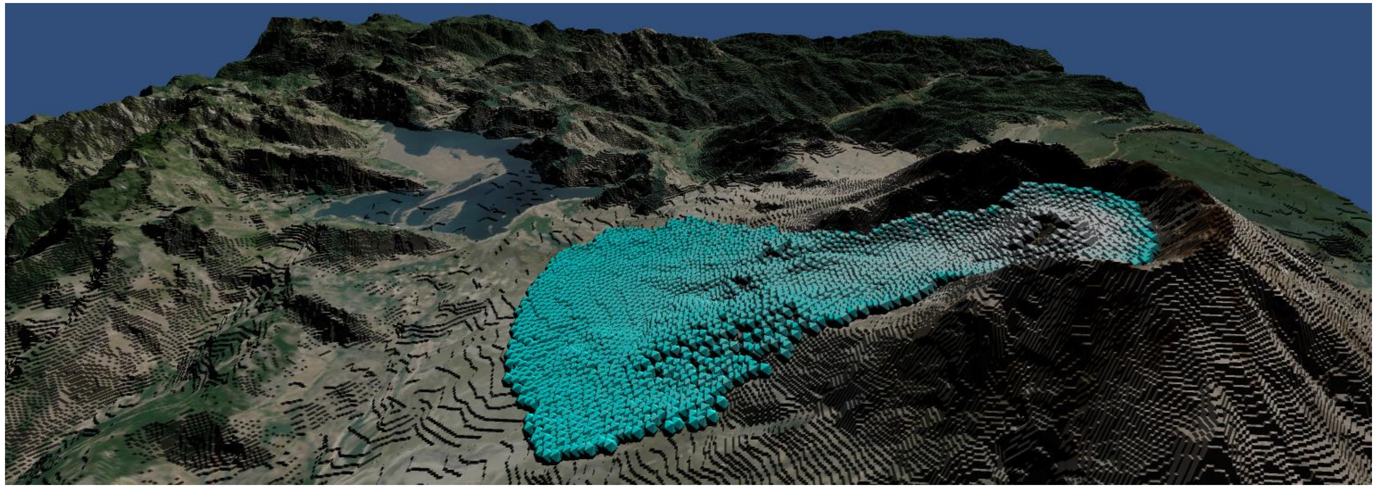
## Implicit schemes

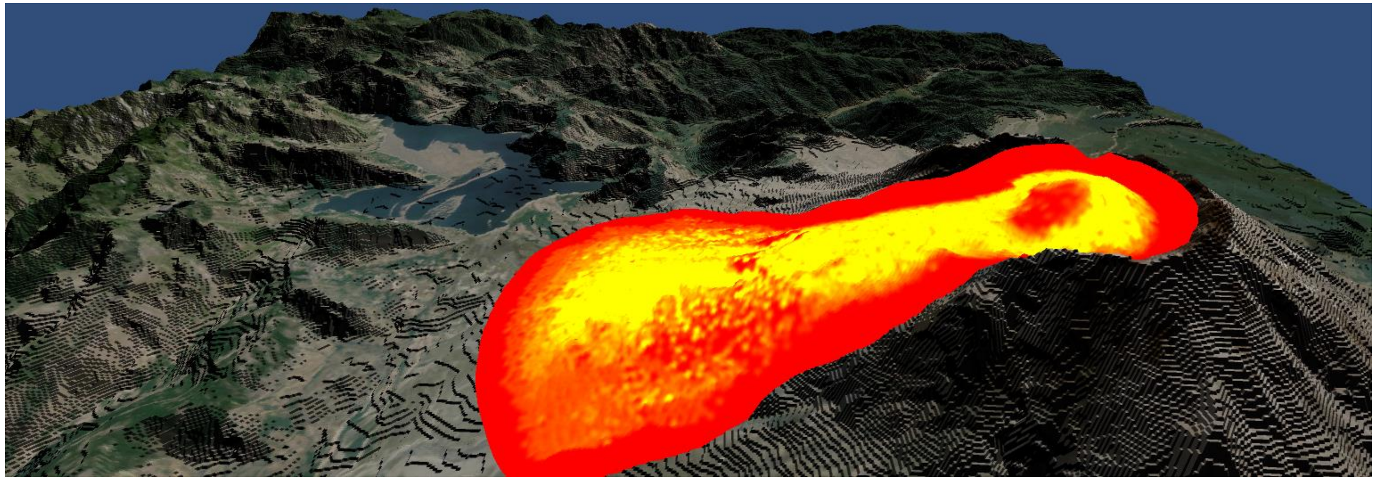
x(t+dt) = f(x(t+dt), x(t))

+ Time step size independent
- Linear system to solve

from *A Physically Consistent Implicit Viscosity Solver for SPH Fluids*, Weiler et al.

# Our method





# The algorithm

**Algorithm 1** The main loop of the simulation

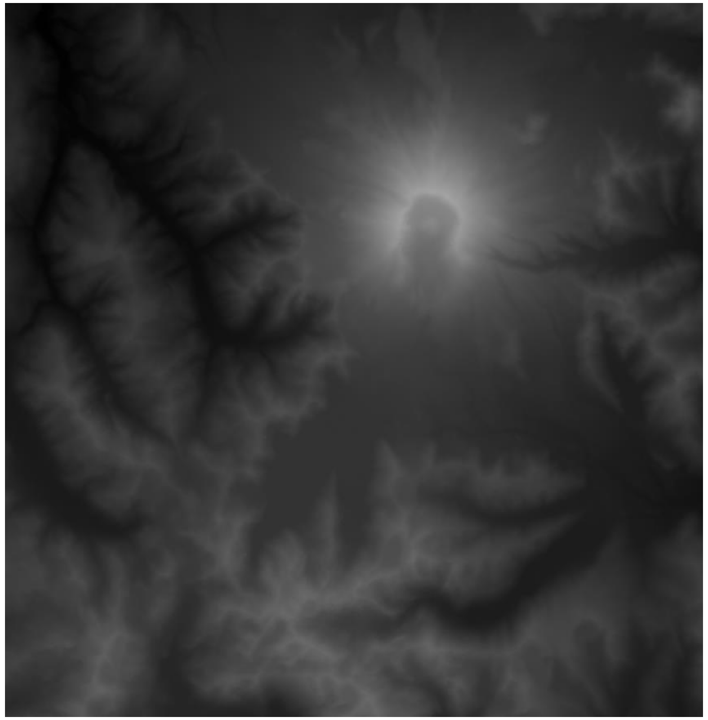
```
Require: Grids Initialization
  while True do
     if elapsedTime > dt then
        if Current number of particles < Maximum number of particles then
           Generate (elapsedTime/dt) particles
        end if
        Update neighbors
        Update heights
        Update temperatures
        Update velocities
        Update positions
        Update terrain and render
        Reset elapsedTime
     end if
  end while
```

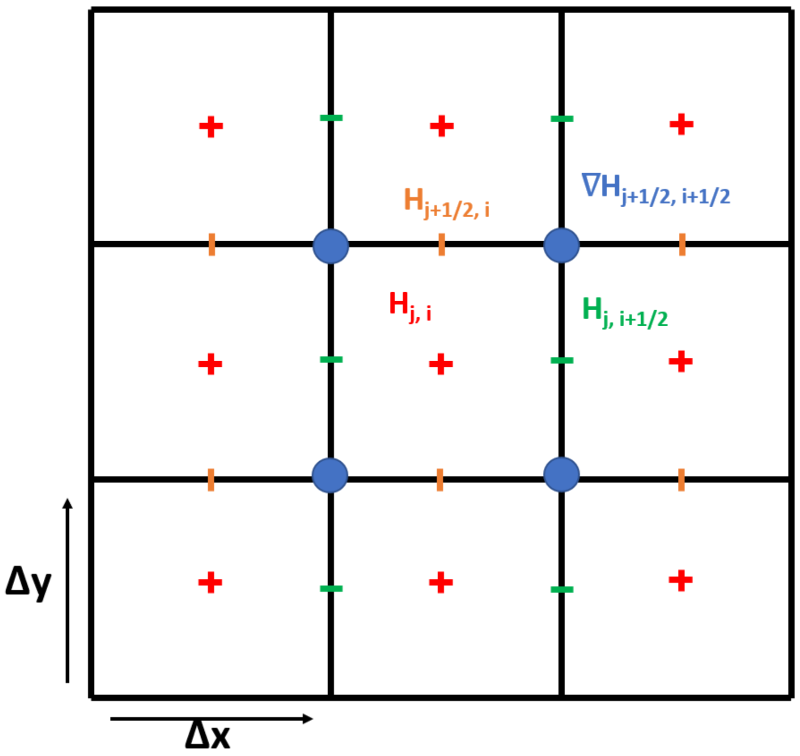
# The algorithm

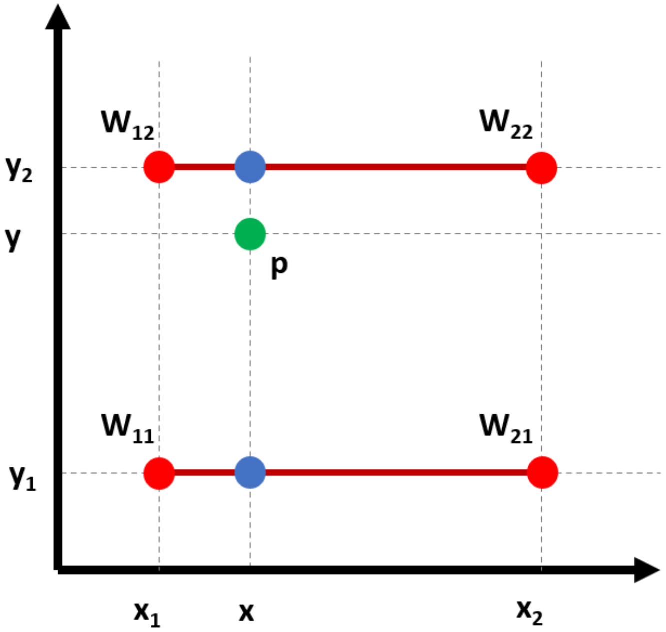
**Algorithm 1** The main loop of the simulation

```
Require: Grids Initialization
  while True do
    if elapsedTime > dt then
      if Current number of particles < Maximum number of particles then
        Generate (elapsedTime/dt) particles
      end if
      Update neighbors
      Update heights
      Update temperatures
      Update velocities
      Update positions
      Update terrain and render
      Reset elapsedTime
    end if
  end while
```

# The Staggered grid






Heightmap → Staggered grid → Interpolation

# The algorithm

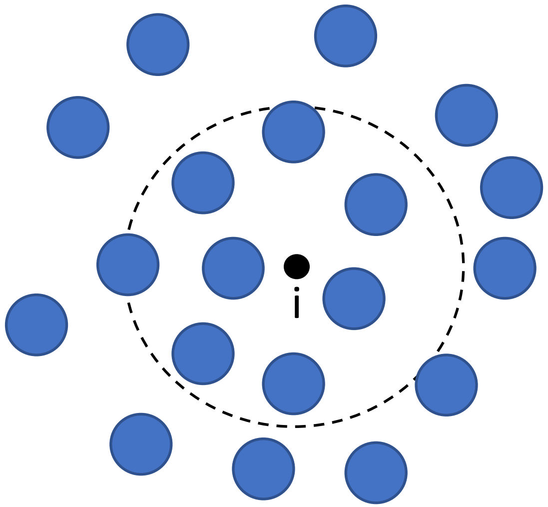
**Algorithm 1** The main loop of the simulation

**Require:** Grids Initialization
  **while** True **do**
    **if** $elapsedTime > dt$ **then**
      **if** Current number of particles < Maximum number of particles **then**
        Generate $(elapsedTime/dt)$ particles
      **end if**
      Update neighbors
      Update heights
      Update temperatures
      Update velocities
      Update positions
      Update terrain and render
      Reset $elapsedTime$
    **end if**
  **end while**

# Smoothed Particle Hydrodynamics

## 3D Version

$$W_{ij} = W(x_i - x_j, l)$$

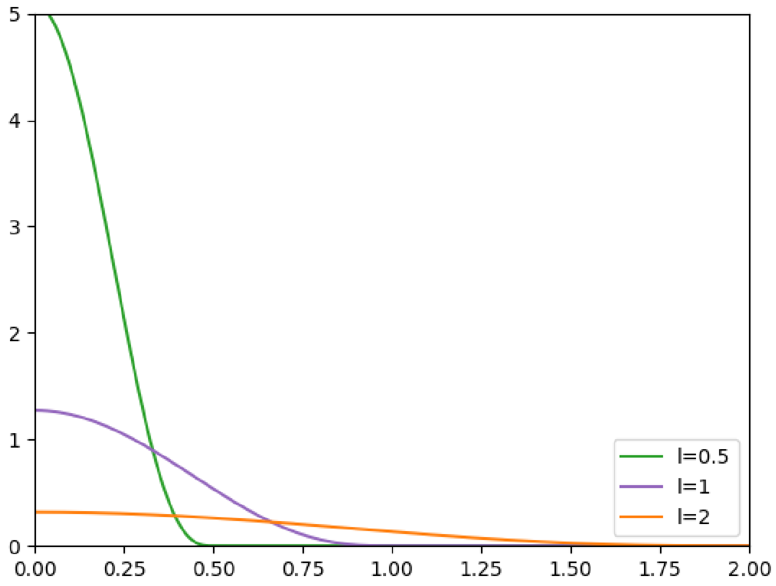



$$q_i = \sum_j \frac{m_j}{\rho_j} q_j W_{ij}$$

$$\nabla q_i = \sum_j \frac{m_j}{\rho_j} q_j \nabla W_{ij}$$

$$\nabla \cdot q_i = \sum_j \frac{m_j}{\rho_j} q_j \cdot \nabla W_{ij}$$

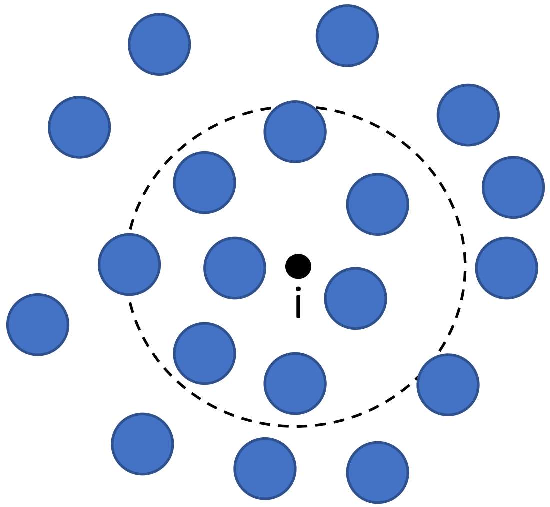
$$\nabla^2 q_i = \sum_j \frac{m_j}{\rho_j} q_j \nabla^2 W_{ij}$$




*Usual Smoothing kernel : Wpoly6*

# Smoothed Particle Hydrodynamics

## 2D Version



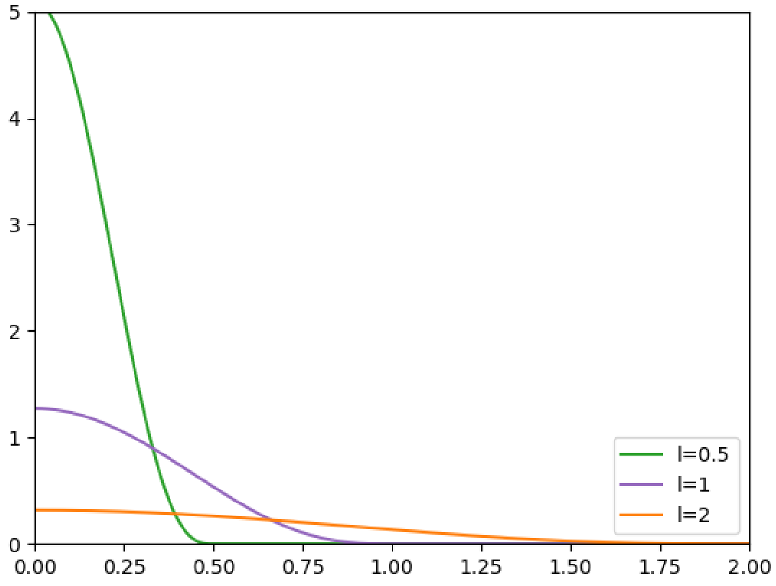

$$q_i = \sum_j \frac{V_j}{h_j} q_j W_{ij}$$

$$\nabla q_i = \sum_j \frac{V_j}{h_j} q_j \nabla W_{ij}$$

$$\nabla \cdot q_i = \sum_j \frac{V_j}{h_j} q_j \cdot \nabla W_{ij}$$

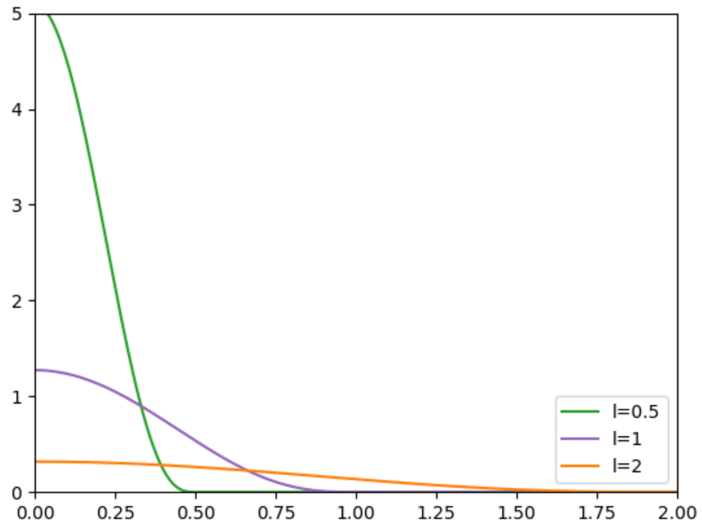
$$\nabla^2 q_i = \sum_j \frac{V_j}{h_j} q_j \nabla^2 W_{ij}$$

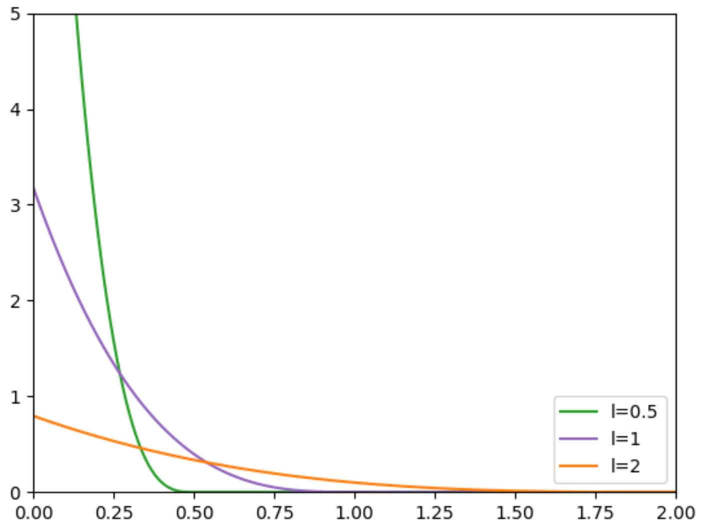
$$W_{ij} = W(x_i - x_j, l)$$

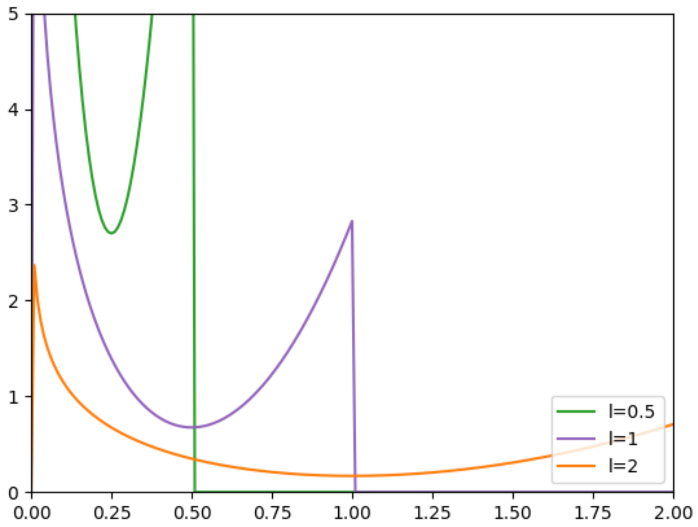



*Usual Smoothing kernel : Wpoly6*

# Smoothed Particle Hydrodynamics




*Wpoly6*

*Wspiky*

*Wvisc*

# The algorithm

**Algorithm 1** The main loop of the simulation

```
Require: Grids Initialization
  while True do
    if elapsedTime > dt then
      if Current number of particles < Maximum number of particles then
        Generate (elapsedTime/dt) particles
      end if
      Update neighbors
      Update heights
      Update temperatures
      Update velocities
      Update positions
      Update terrain and render
      Reset elapsedTime
    end if
  end while
```

# The viscosity

Glaciology

Stokes

$$\frac{d\vec{u}}{dt} + (\vec{u}\cdot\nabla)\vec{u} = -\frac{\nabla\cdot \mathrm{pI}}{\rho} + \frac{\nabla\cdot\tau}{\rho} + \vec{g}$$

$$\vec{0} = \nabla\cdot\tau + \rho\vec{g}\nabla S$$

$\vec{u} \rightarrow$ Velocity
$p \rightarrow$ Pressure
$I \rightarrow$ Identity tensor
$\tau \rightarrow$ Stress tensor
$\rho \rightarrow$ density
$\vec{g} \rightarrow$ Stress tensor

$\tau \rightarrow$ Stress tensor
$\rho \rightarrow$ density
$\vec{g} \rightarrow$ Stress tensor
$S \rightarrow$ Surface

# The viscosity

$$\vec{0} = \nabla \cdot \tau + \rho \vec{g} \nabla S$$

$$\tau = f(\theta)\mu\epsilon$$

$$\xrightarrow{\hspace{3cm}}$$

Volcanology

$$f(\theta)\mu \begin{pmatrix} \frac{\partial^2 u_x}{\partial x^2} + \frac{1}{2}\frac{\partial^2 u_x}{\partial y^2} + \frac{1}{2}\frac{\partial^2 u_x}{\partial z^2} \\ \frac{\partial^2 u_y}{\partial y^2} + \frac{1}{2}\frac{\partial^2 u_y}{\partial x^2} + \frac{1}{2}\frac{\partial^2 u_y}{\partial z^2} \end{pmatrix} = -\rho \vec{g} \nabla S$$

$\tau \rightarrow$ Stress tensor

$\rho \rightarrow$ density

$\vec{g} \rightarrow$ Stress tensor

$S \rightarrow$ Surface

$\tau \rightarrow$ Stress tensor

$\rho \rightarrow$ density

$\vec{g} \rightarrow$ Stress tensor

$S \rightarrow$ Surface

$\theta \rightarrow$ Temperature

$\mu \rightarrow$ Viscosity

$\epsilon \rightarrow$ Strain rate tensor

$u_x, u_y \rightarrow$ 2D velocity components

# The viscosity

$$f(\theta)\mu\begin{pmatrix}\frac{\partial^2 u_x}{\partial x^2}+\frac{1}{2}\frac{\partial^2 u_x}{\partial y^2}+\frac{1}{2}\frac{\partial^2 u_x}{\partial z^2}\\ \frac{\partial^2 u_y}{\partial y^2}+\frac{1}{2}\frac{\partial^2 u_x}{\partial x\partial y}+\frac{1}{2}\frac{\partial^2 u_y}{\partial z^2}\end{pmatrix}=-\rho\vec{g}\nabla S \xrightarrow{\text{Integrate}} \frac{\partial}{\partial x^2}u'+\frac{1}{2}\frac{\partial}{\partial y^2}u'-\frac{3}{2}u'h^4=-(h^3\frac{\rho\vec{g}\nabla S}{f(\theta)\mu})$$

$\tau \rightarrow$ Stress tensor

$\rho \rightarrow$ density

$\vec{g} \rightarrow$ Stress tensor

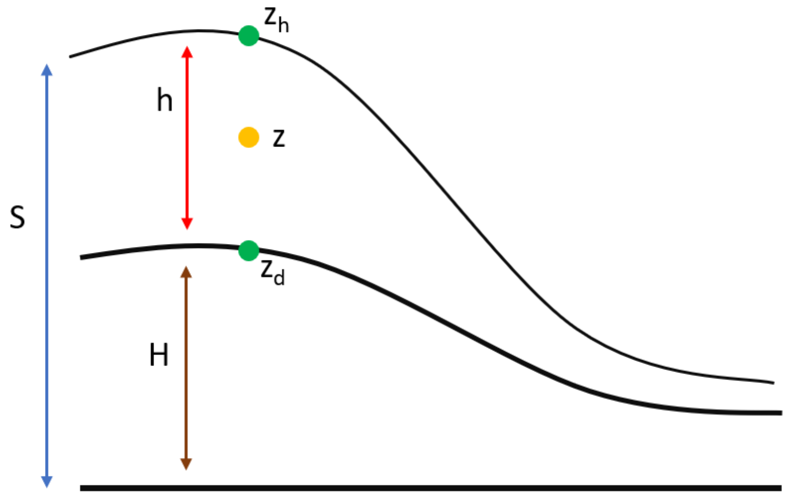
$S \rightarrow$ Surface

$\theta \rightarrow$ Temperature

$\mu \rightarrow$ Viscosity

$\epsilon \rightarrow$ Strain rate tensor

$u_x, u_y \rightarrow$ 2D velocity components

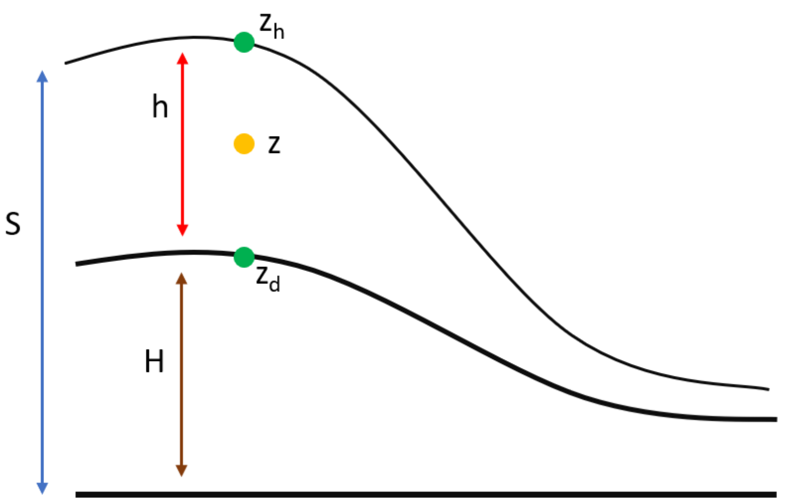



$$u' = h^3\bar{u}$$

# The viscosity

$$\frac{\partial}{\partial x^2}u' + \frac{1}{2}\frac{\partial}{\partial y^2}u' - \frac{3}{2}u'h^4 = -(h^3\frac{\rho\vec{g}\nabla S}{f(\theta)\mu})$$

$$u' = h^3\bar{u}$$




Known Equation

$$\nabla^2 u' - k^2 u' = b_j$$

$$G(u) = -\frac{1}{2\pi}K_0(kr)$$

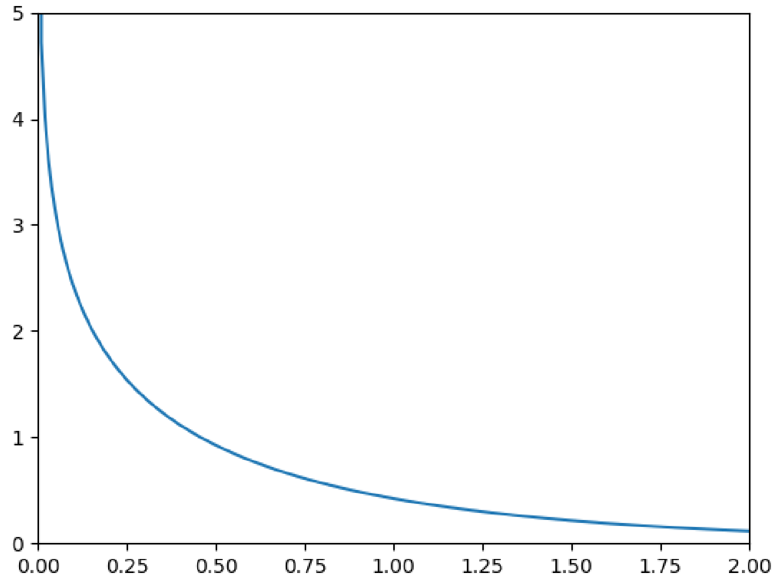
$$\bar{u_i} = h_i^3 \frac{\sum_j G(u_j) * b_j}{\sum_j G(u_j)}$$

$$\bar{u_i} = h_i^3 \frac{\sum_j K_0(kr) * b_j}{\sum_j K_0(kr)}$$

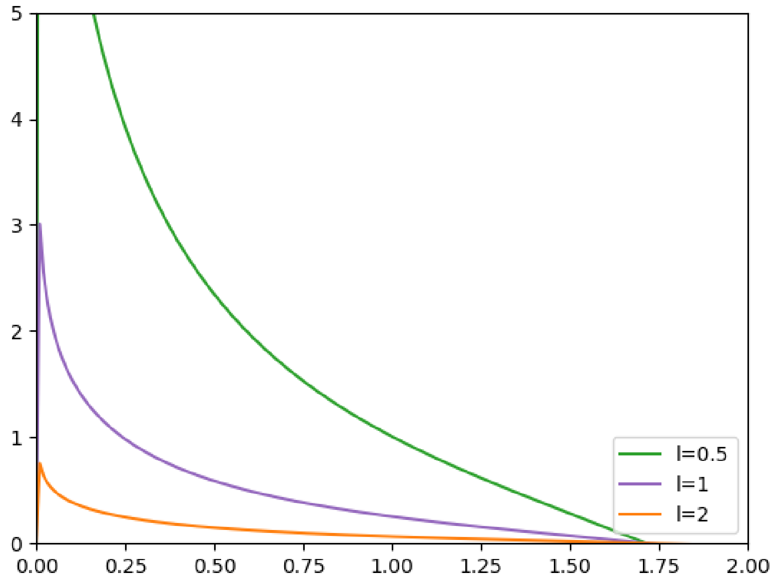
# The viscosity

$$W_{\mathrm{new}}(r,l) = \frac{2}{\pi l^2}\begin{cases}((-\log(r) - \gamma + \log(2)) + \frac{1}{4}r^2(-\log(r) - \gamma + 1 + \log(2))) & \text{if } 0 < r \le l \\ 0 & \text{otherwise}\end{cases}$$

$$K_0 = \int_0^{+\infty} \frac{\cos(xt)}{\sqrt{t^2+1}}\,dt$$




Modified Bessel function $K_0$



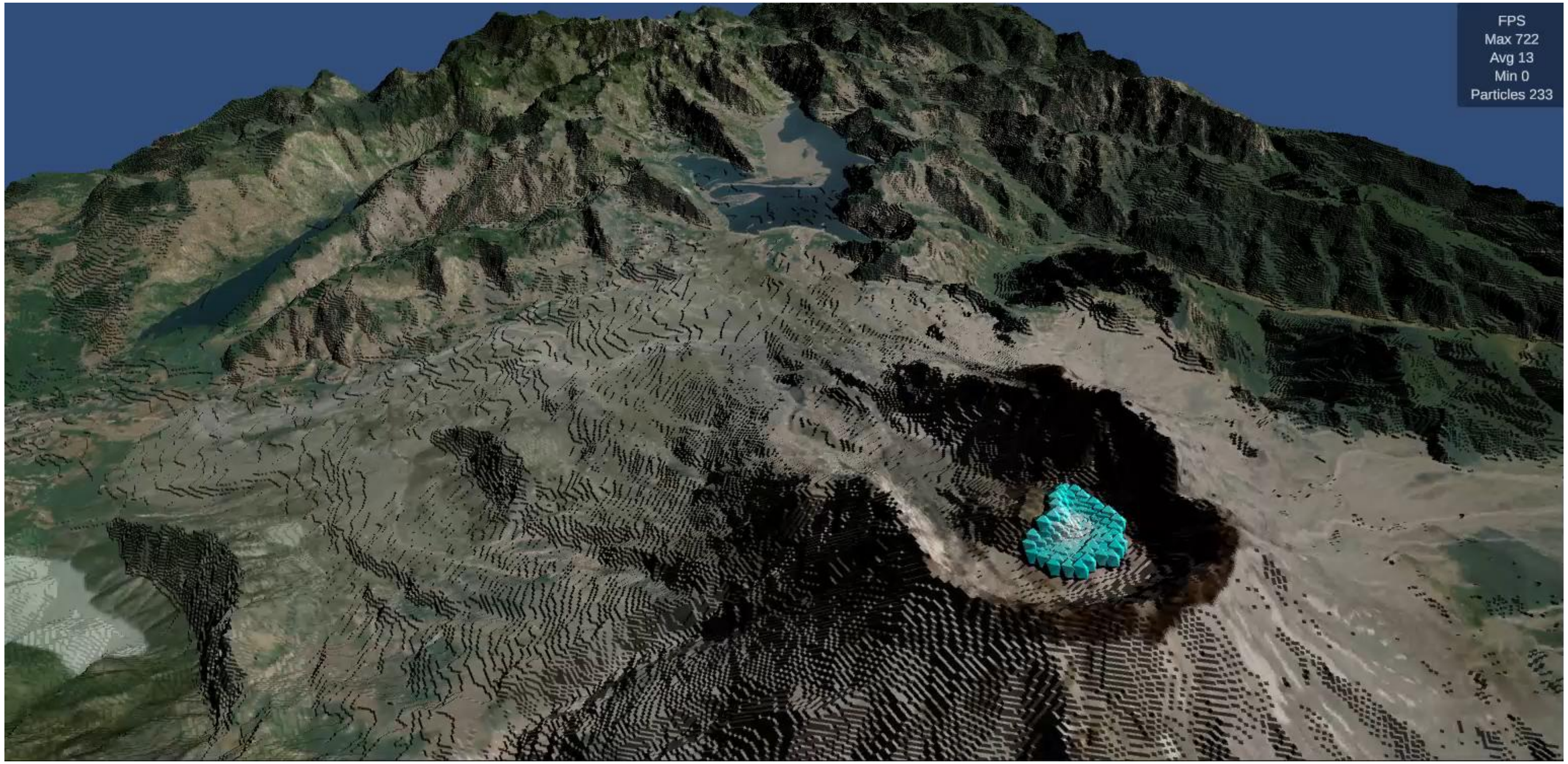

New smoothing kernel $W_{new}$

# The algorithm

**Algorithm 1** The main loop of the simulation

**Require:** Grids Initialization
  **while** True **do**
    **if** $elapsedTime > dt$ **then**
      **if** Current number of particles < Maximum number of particles **then**
        Generate $(elapsedTime/dt)$ particles
      **end if**
      Update neighbors
      Update heights
      Update temperatures
      Update velocities
      Update positions
      Update terrain and render
      Reset $elapsedTime$
    **end if**
  **end while**

# Final results




*Simulation running in a NVIDIA RTX A6000, using Mount St.Helens topography*

# What's next

- Implement a validation model
- Change the model for $\mu$
- Change the model for $\theta$

Thanks for your attention