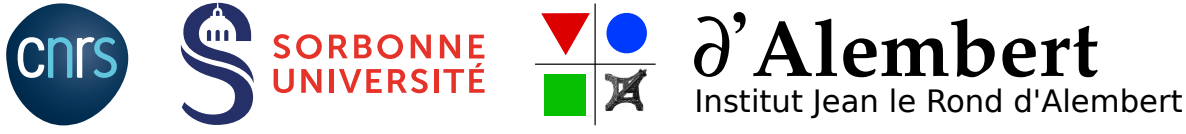
Basilisk: a programming framework and social network for modelling with partial differential equations

Stéphane Popinet et al.

Institut Jean le Rond ∂'Alembert, CNRS & Sorbonne Université



Why partial differential equations?

There are **only three** (families of) equations in **(multi)-physics**:

Advection / transport / parabolic equations

$$\partial_t f + \nabla \cdot (u\, f) = 0$$

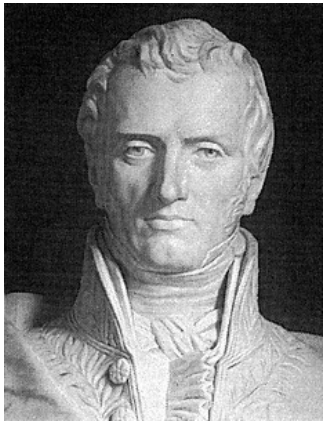
Diffusion / wave / elliptic equations

$$\lambda\, \phi + \nabla \cdot (\mu\, \nabla \phi) = 0$$

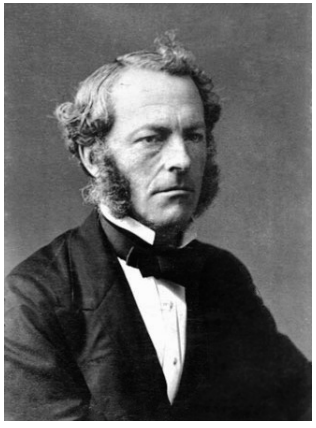
Path integrals / convolution

$$S = \oint \left(\frac{dx}{dt} \right)^2 dt$$

Example: the Navier–Stokes equations



Claude-Louis Navier
1822



George Gabriel Stokes
1845

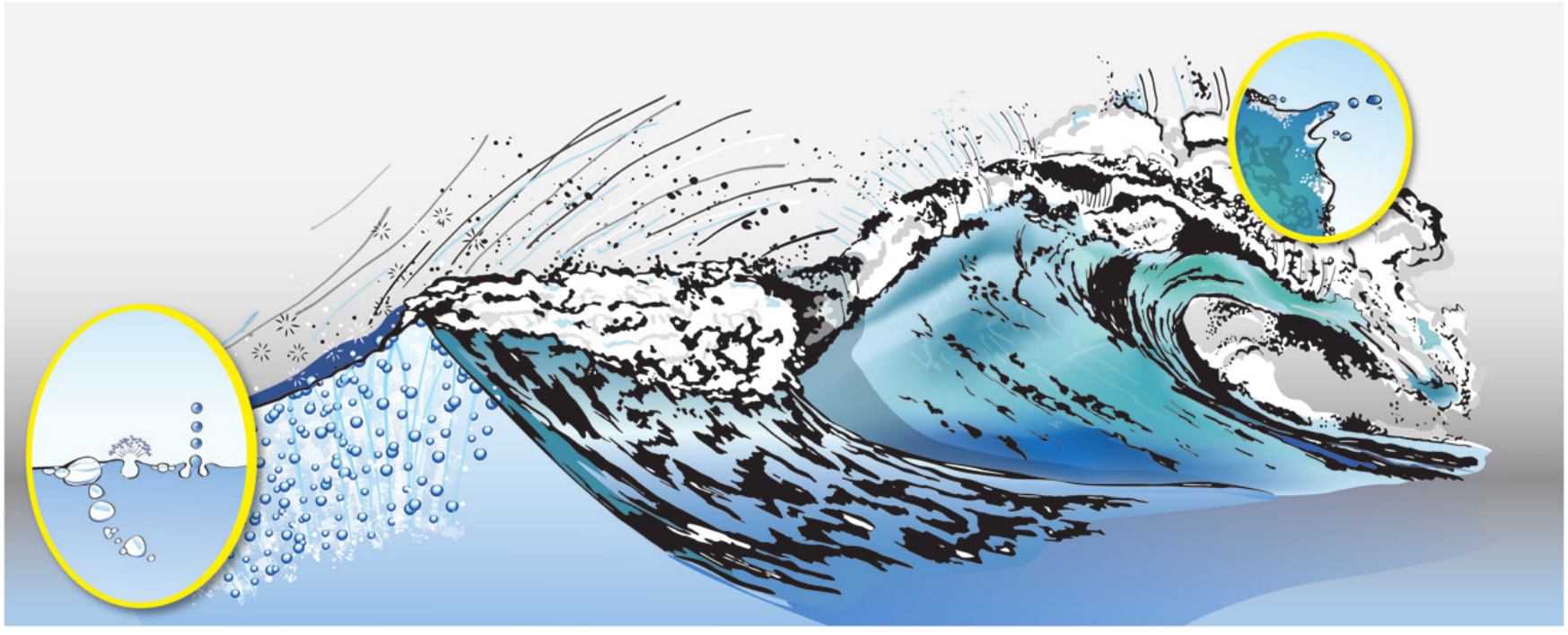


Adhémar de Saint-Venant
1819? 1843

Variable density and viscosity, incompressible Navier–Stokes equations

$$
\begin{aligned}
\partial_t \rho + \boldsymbol{\nabla} \cdot (\rho\, \mathbf{u}) &= 0 \\
\partial_t (\rho\, \mathbf{u}) + \boldsymbol{\nabla} \cdot (\rho\, \mathbf{u} \otimes \mathbf{u}) &= -\boldsymbol{\nabla}\, p + \boldsymbol{\nabla} \cdot [\mu\, (\boldsymbol{\nabla} \mathbf{u} + \boldsymbol{\nabla}^T \mathbf{u})] + \boldsymbol{S} \\
\boldsymbol{\nabla} \cdot \mathbf{u} &= 0
\end{aligned}
$$

Source terms $\boldsymbol{S}$: gravity, surface tension, Coriolis etc.
Important advances in their numerical approximation in the past 25 years

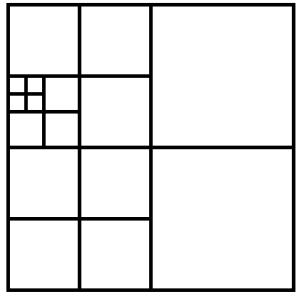


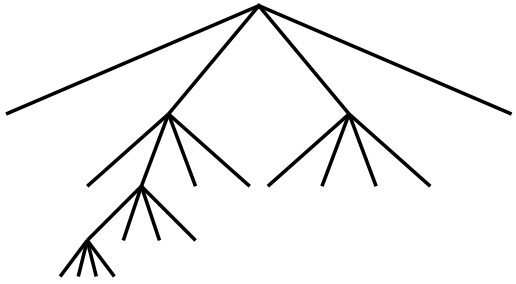
From Véron, ARFM 2015

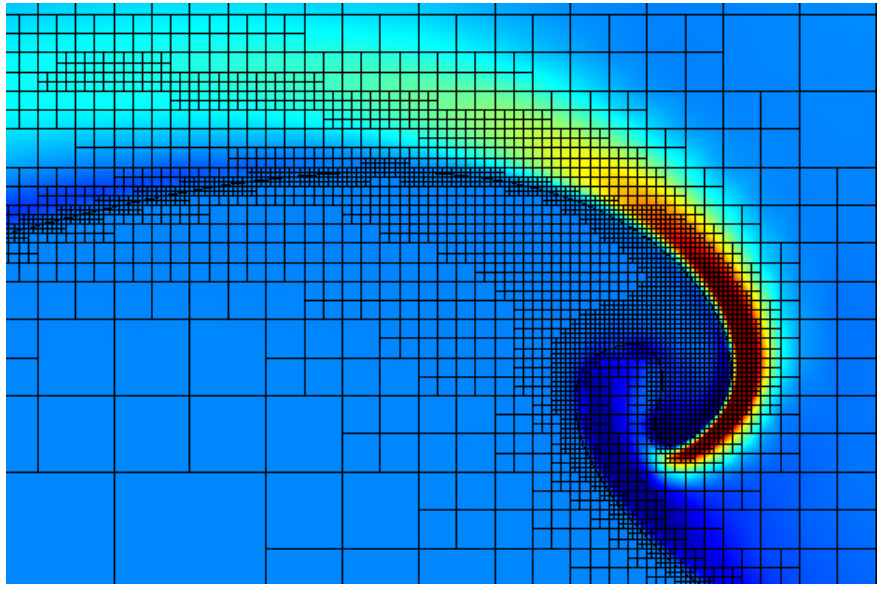
"Multiphysics": multiphase turbulence, heat and mass transfer, phase change, surfactants, reactive flows, chemistry and biology

Is Direct Numerical Simulation possible (and useful)?

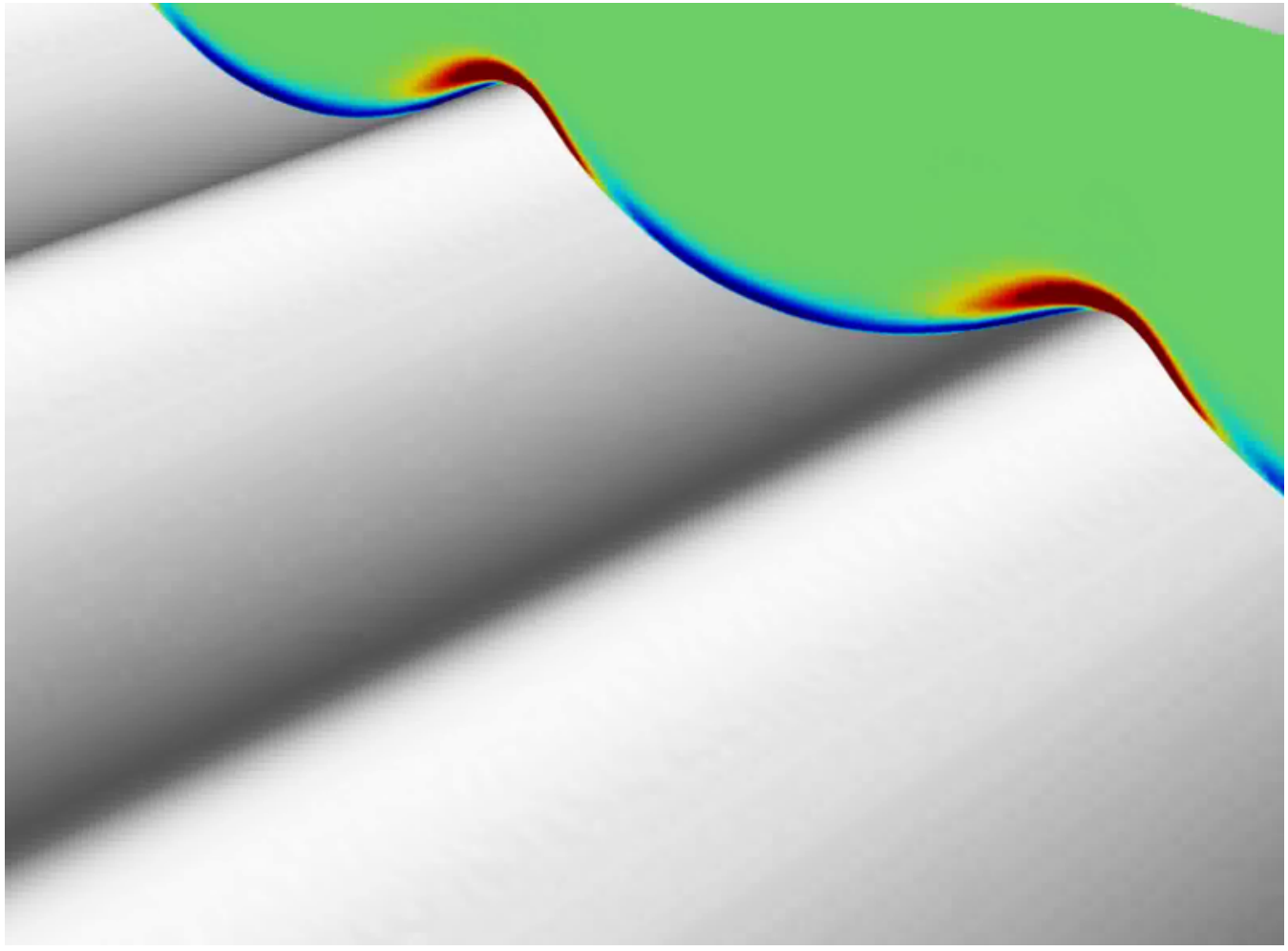
- Variable-density Navier–Stokes with surface tension
- Compressible or incompressible
- Non-diffusive, geometric VOF (PLIC)
- Two-phase momentum conservation
- Balance continuum surface tension formulation (see ARFM, 2018)
- Free Software: Gerris and Basilisk, see `basilisk.fr`







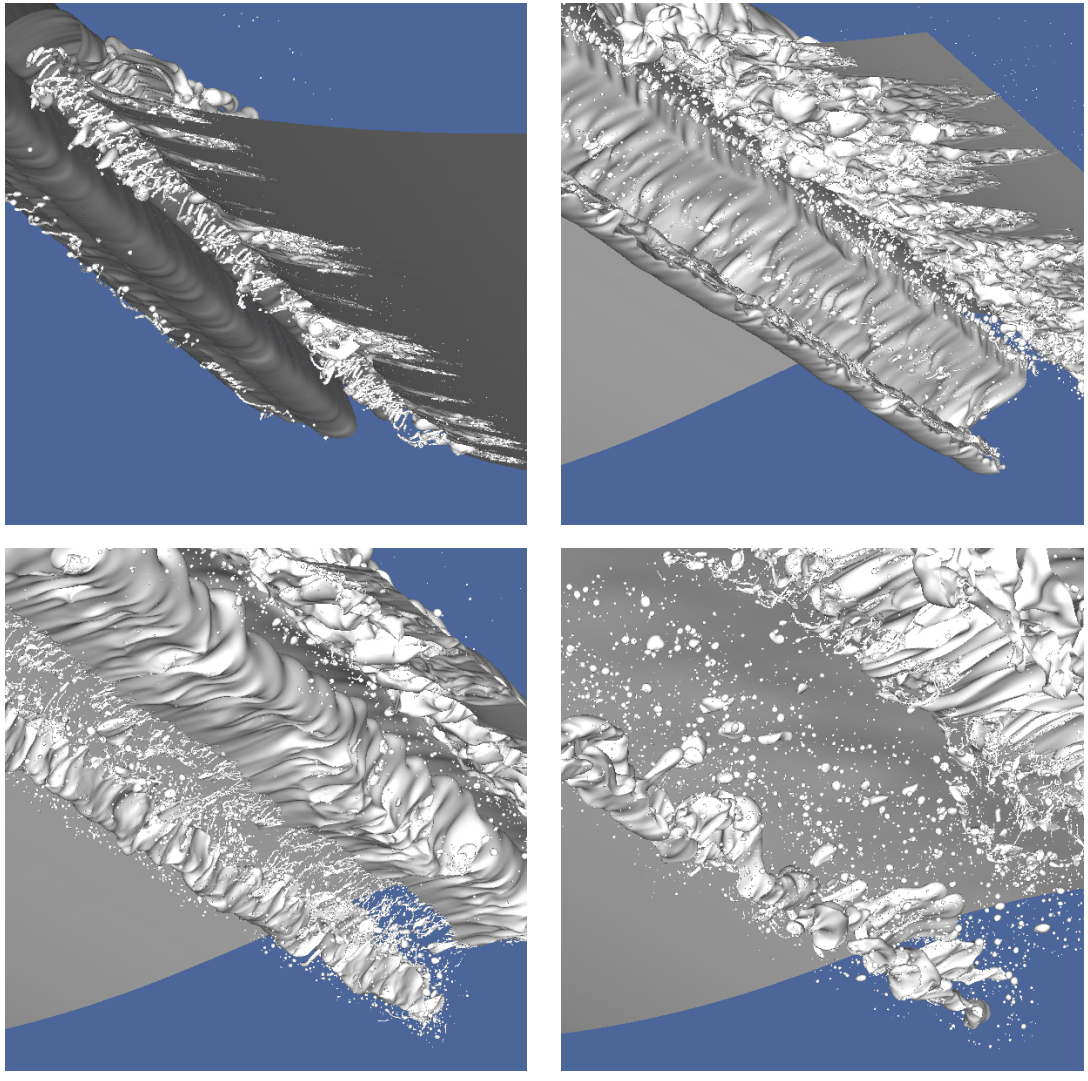
"DNS" of breaking waves: Mostert et al, *JFM*, 2022

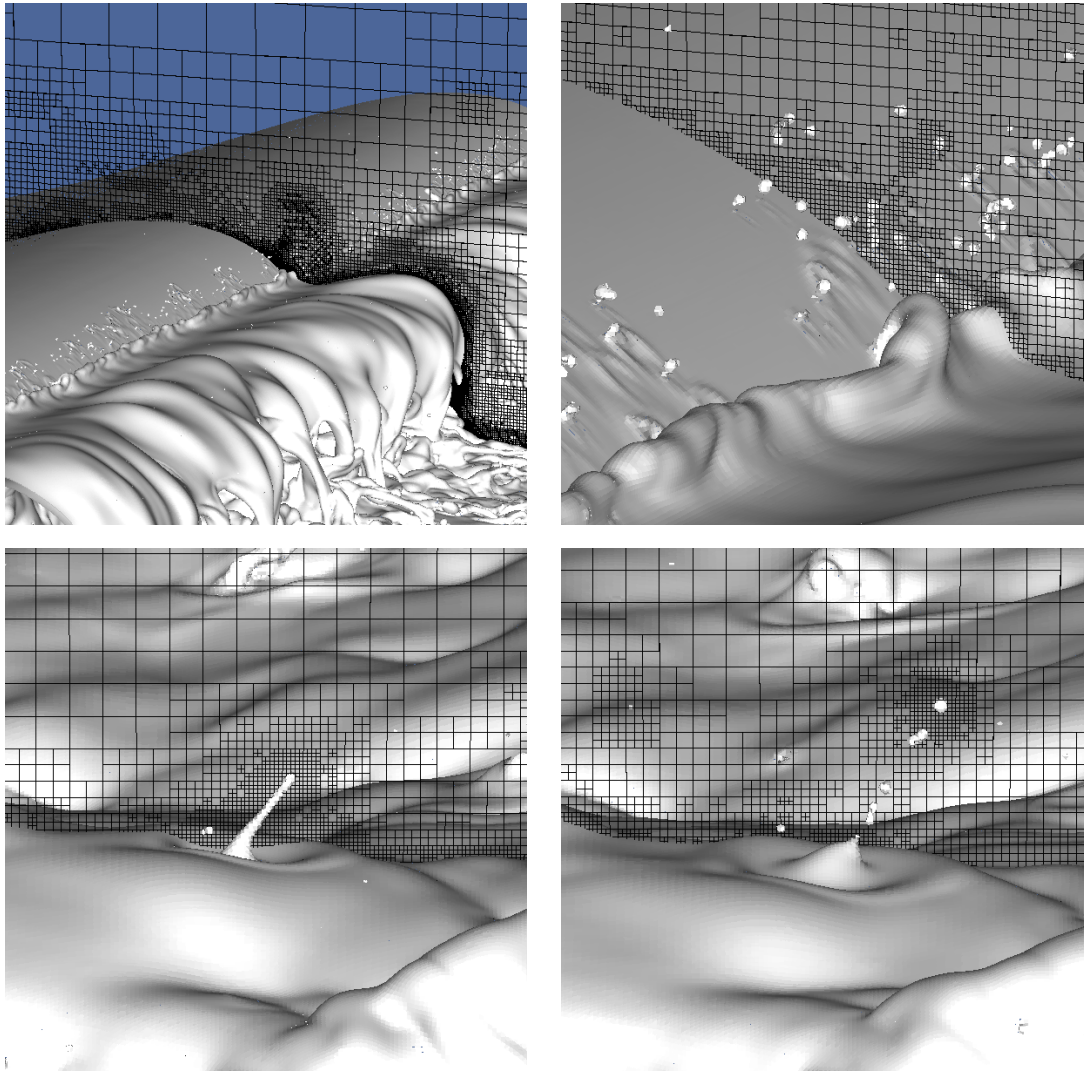


$\mathrm{Re} = 10^5$, $\mathrm{Bo} = 500$, $a\,k = 0.55$

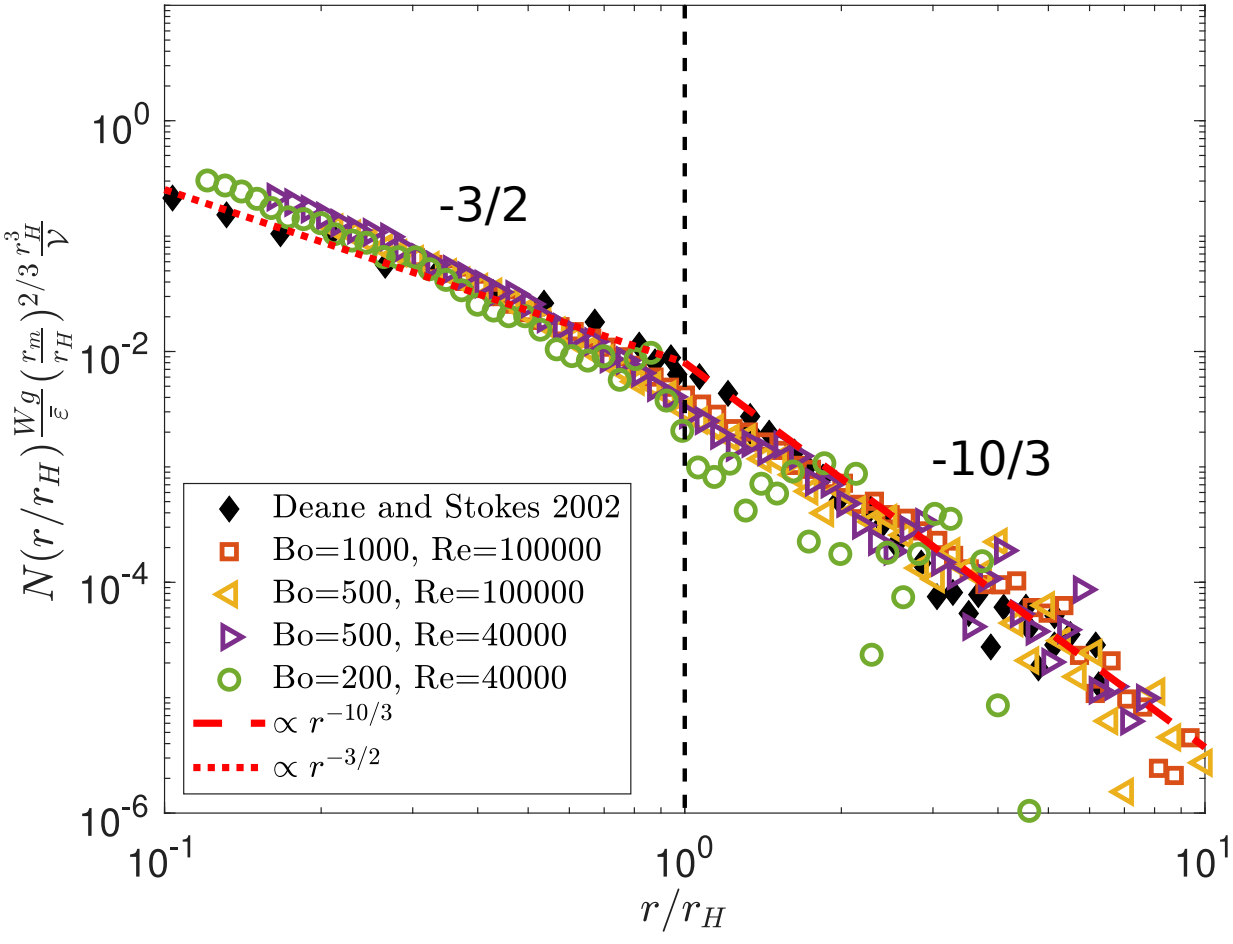
2048^3 with adaptive mesh refinement

Underwater bubble breakup





Time-averaged bubble size distributions



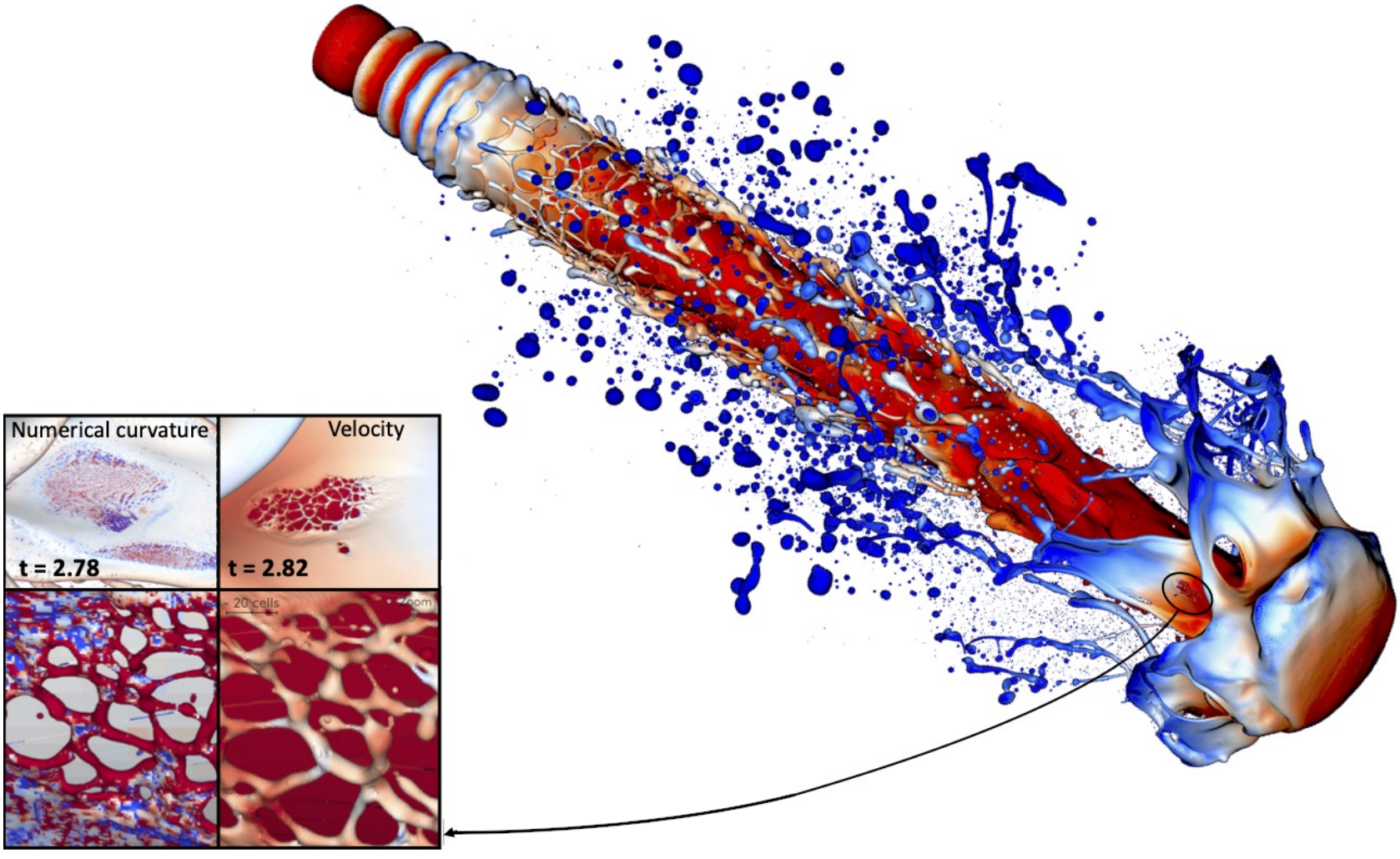
$N(r) \propto r^{-10/3}$ (Garett et al., *Nature*, 2000)

These relations are valid also in industrial systems



Methanation: $CO_2 + H_2 + \text{Energy} \rightarrow \text{Methane}$ (Rolls-Royce MethanQuest)

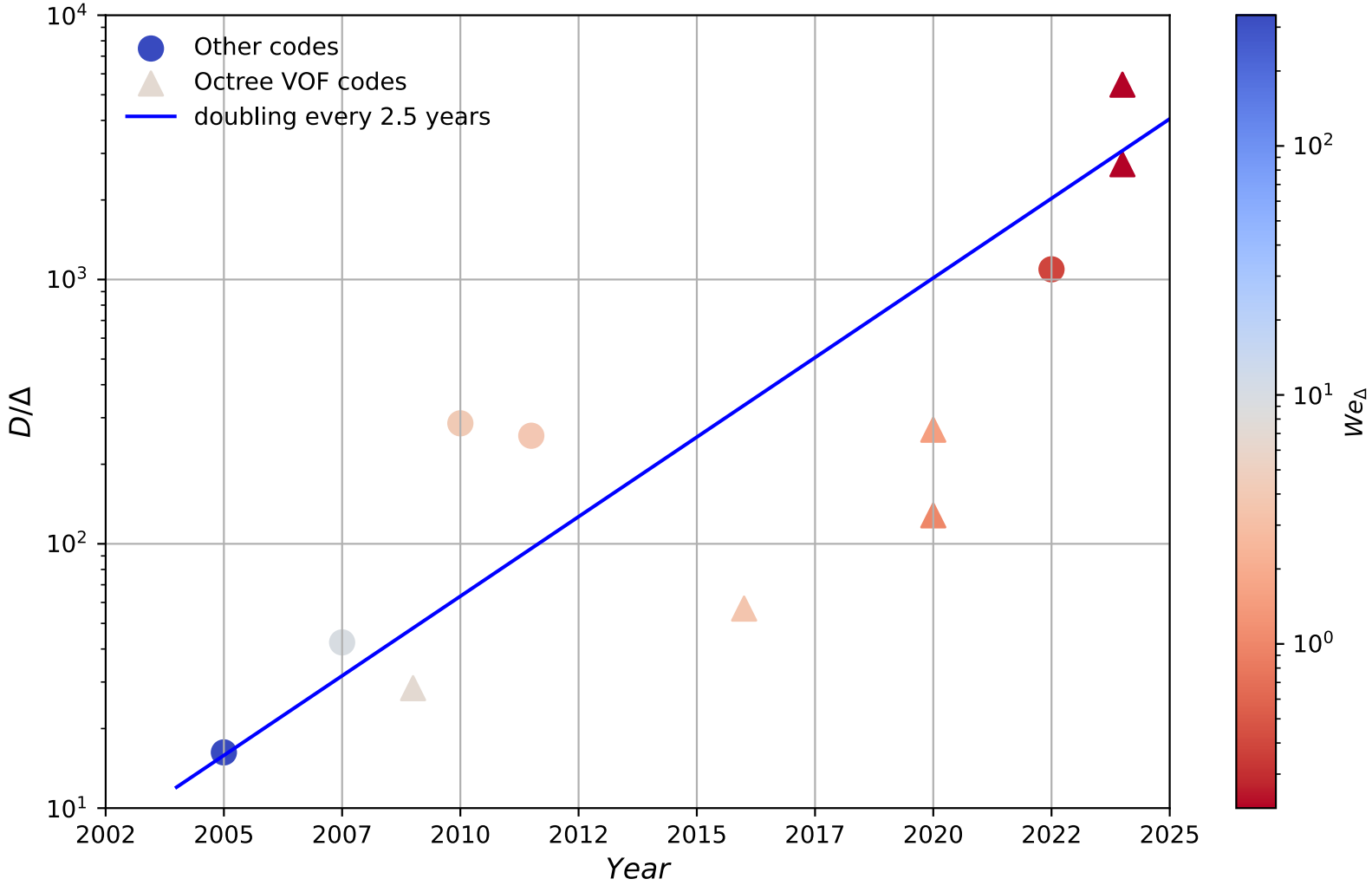
Multiphase turbulence at smaller scales: fuel injection



Adaptive mesh refinement: 16384^3 equivalent maximum resolution

Kulkarni et al. *The atomizing pulsed jet.* submitted to JFM, 2024

Computational efficiency (and accuracy) matters



Converged and reliable atomization statistics require high resolution (low We_Δ)

Challenges for computational frameworks

- *High Performance Computing hardware and software is extremely complex*: Parallelism, vectorization, distributed memory, cache hierarchies, heterogeneous architectures etc.
- *These machines are rarely designed primarily for scientific reasons*
  Geopolitics, economics, technology etc.
- *Performance depends strongly on the choice and design of solution algorithms*

  This can lead to bad *scientific* choices: "solving the wrong set of equations" and/or very poor performance

  Beware the "every problem looks like a nail" effect
- *Education and job market: few people master the combination*
  computer science + applied mathematics + physics / engineering

A solution: Free Software and Open Science

- Free Software: unrestricted worldwide access to the source code, documentation, expertise

  $\Rightarrow$ maximizes the size of the pool where specialists can be **"recruited"**

- The social (and political) aspects of Free Software (Open Source) are key: *"People matter more than software"*

- An example with Gerris and Basilisk

  – Gerris first released under the General Public License (GPL) in 2001

  – Basilisk: a complete re-design started in 2013

  $>$ 800 registered users worldwide

  $\approx$ 70 publications/year (JFM, PRL, PRF, IJMF, JCP, JGR etc.)

  20–30 regular contributors

Basilisk: design goals

- **Accessibility**

  "Our M2/PhD students must be able to program new numerical schemes within Basilisk"

- **Flexibility**

  Multi-equations (multi-physics), multi-environments (HPC, hardware etc.)
  Usable in combination with any other (UNIX-compatible) language/library

- **Performance**

  Adaptive Mesh Refinement, no-storage multigrid solvers, parallelism, GPUs
  Accuracy, high-resolution, higher-order schemes

- **Documentation**

  Literate programming: 1 line of code $\Leftrightarrow$ 1 line of doc

- **Community**

  Open and easy contributions/discussions of applications, code, documentation

Basilisk: design choices

- **Accessibility**

  Back to basics: written in standard **C99** (not C++!), self-contained
  No object oriented programming, **No** pointers, **No** operator overloading etc.
  Only: if, loops, functions (the first three chapters of K&R, 1989) $\Rightarrow$ **C--**

- **Flexibility**

  A simple abstraction: *Generalised Cartesian Grids*
  C99 is the "least common denominator" of all languages (Python, Julia etc.)

- **Performance**

  C99, OpenMP, MPI, OpenGL Shader Language (GLSL) on GPUs (***not*** *CUDA*)
  Example: 1 nanosecond point$\times$timestep for geometric VOF advection on GPUs

- **Documentation and community**

  Freely-editable wiki
  The "website is the code"
  Open and reproducible science, regular "in person" and virtual (zoom) meetings

A minimal extension of C99 (not really a "Domain-Specific Language")

a) New types for scalar, vector and tensor fields: `scalar`, `vector`, `tensor`

b) Operations on local regular Cartesian *stencils* (5×5 by default in 2D) :

```
scalar a[];
```

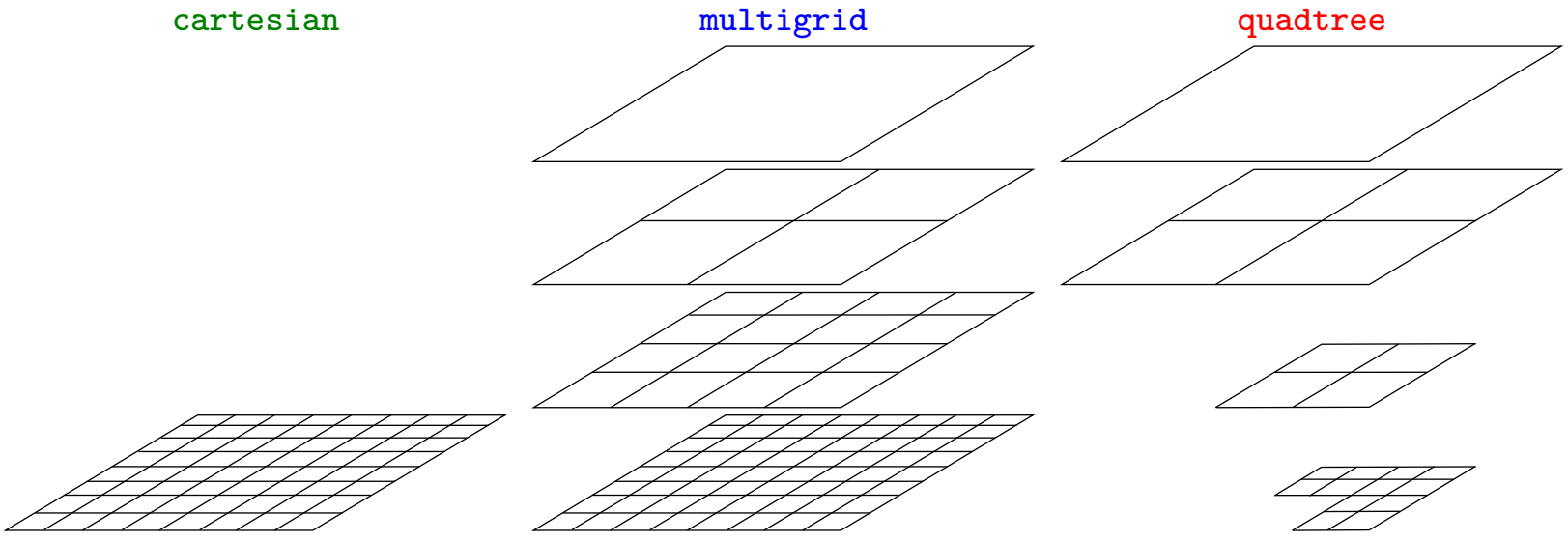
a[-1,1]	a[0,1]	a[1,1]
a[-1,0]	a[0,0]	a[1,0]
a[-1,-1]	a[0,-1]	a[1,-1]

`a[] = a[0,0]` (different from C : `a[0][0]`)

c) Iterators : `foreach()`, `foreach_dimension()`

The corresponding C code is generated automatically **at compilation time**

A transparent hierarchy of discretisations



Operators

`foreach()`	`coarse()`, `fine()`	`foreach_cell()`
`foreach_dimension()`	`level`, `depth()`, `child.x`	`cell`, `parent`
`scalar`, `vector`, `tensor`	`foreach_level()`	`leaf`, `active`, `halo`
`v[i,j]`	`foreach_fine_to_coarse()`	
`x`, `y`, `delta`		

Numerical schemes are independent from the choice of grid

Numerical schemes are independent from the details of (hardware) implementation (MPI / OpenMP parallelism, GPUs etc.)

Example: a multigrid solver for elliptic equations

Works on multigrid and quadtree discretisations, using MPI, OpenMP or GPUs etc.

```
void mg_cycle (scalar a, scalar res, scalar dp,
               void (* relax) (scalar dp, scalar res, int depth),
               int nrelax, int minlevel)
{
  restriction ({res});
  for (int l = minlevel; l <= depth(); l++) {
    if (l == minlevel)
      foreach_level (l)
        dp[] = 0.;
    else
      /* bilinear interpolation from coarser level */
      foreach_level (l)
        dp[] = (9.*coarse(dp,0,0) +
                3.*(coarse(dp,child.x,0) + coarse(dp,0,child.y)) +
                coarse(dp,child.x,child.y))/16.;
    for (int i = 0; i < nrelax; i++)
      relax (dp, res, l);
  }
  foreach()
    a[] += dp[];
}
```

```
void restriction (scalar v)
{
  foreach_fine_to_coarse()
    v[] = (fine(v,0,0) + fine(v,1,0) + fine(v,0,1) + fine(v,1,1))/4.;
}
```

Relaxation for Poisson equation (in 2D) i.e.

$$\nabla^2 \mathrm{dp} = \mathrm{res}$$

```
void relax (scalar dp, scalar res, int l)
{
  foreach_level(l)
    dp[] = (dp[1] + dp[-1] + dp[0,1] + dp[0,-1] - sq(Delta)*res[])/4.;
}
```

The most important aspect of a successful (Open Source / Free Software) project is the **community** (i.e. people) not the source code $\Rightarrow$ **communication!**

The basilisk.fr website

- the website **is the code**
- the code can be executed directly on the server (via the web browser)
- a freely editable **wiki**: anybody can contribute **documented code**

  Core Basilisk code : 20 000 lines (preprocessor) + 30 000 lines (solvers)

  Contributed code : $>$ 300 000 lines

  Recent contributions: phase change, multispecies combustion, capsule biomechanics, etc. (Master Students, PhDs, post-docs, permanent researchers etc.)
- the website/code is version-controlled and can be retrieved/mirrored/modified locally using standard tools (as a hierarchy of files editable with a text editor)
- *Continuous integration* is encouraged $\Rightarrow$ more than 300 test cases

- An active user forum / mailing list

  Google group: 854 members

  All levels: beginners, expert developers, scientific discussions

- Monthly interactive video presentations (2×30 minutes) and an archive

  Recorded on Zoom, public (in person at ∂'Alembert)

  http://basilisk.fr/BMM/README

- Basilisk Users' Meetings (2011, 2012, 2014, 2017, 2019, 2023)

  Paris, Wellington, Sevilla, Princeton, Paris (twice)

  $\approx$ 100 international participants, 2/3 days, free of charge (sponsored)

- Basilisk is part of the curriculum for several academic classes (Master level in Paris, Sevilla, Princeton etc.)

Literate Programming: Donald Knuth (1984) "programmers are illiterate"

A solver for the Saint-Venant equations

Note that the multilayer solver provides the same functionality and should be prefered for most applications.

The Saint-Venant equations can be written in integral form as the hyperbolic system of conservation laws

$$\partial_t \int_\Omega \mathbf{q} d\Omega = \int_{\partial\Omega} \mathbf{f}(\mathbf{q}) \cdot \mathbf{n} d\partial\Omega - \int_\Omega hg\nabla z_b$$

where Ω is a given subset of space, $\partial\Omega$ its boundary and $\mathbf{n}$ the unit normal vector on this boundary. For conservation of mass and momentum in the shallow-water context, Ω is a subset of bidimensional space and $\mathbf{q}$ and $\mathbf{f}$ are written

$$\mathbf{q} = \begin{pmatrix} h \\ hu_x \\ hu_y \end{pmatrix}, \quad \mathbf{f}(\mathbf{q}) = \begin{pmatrix} hu_x & hu_y \\ hu_x^2 + \frac{1}{2}gh^2 & hu_xu_y \\ hu_xu_y & hu_y^2 + \frac{1}{2}gh^2 \end{pmatrix}$$

where $\mathbf{u}$ is the velocity vector, h the water depth and z_b the height of the topography. See also Popinet, 2011 for a more detailed introduction.

User variables and parameters

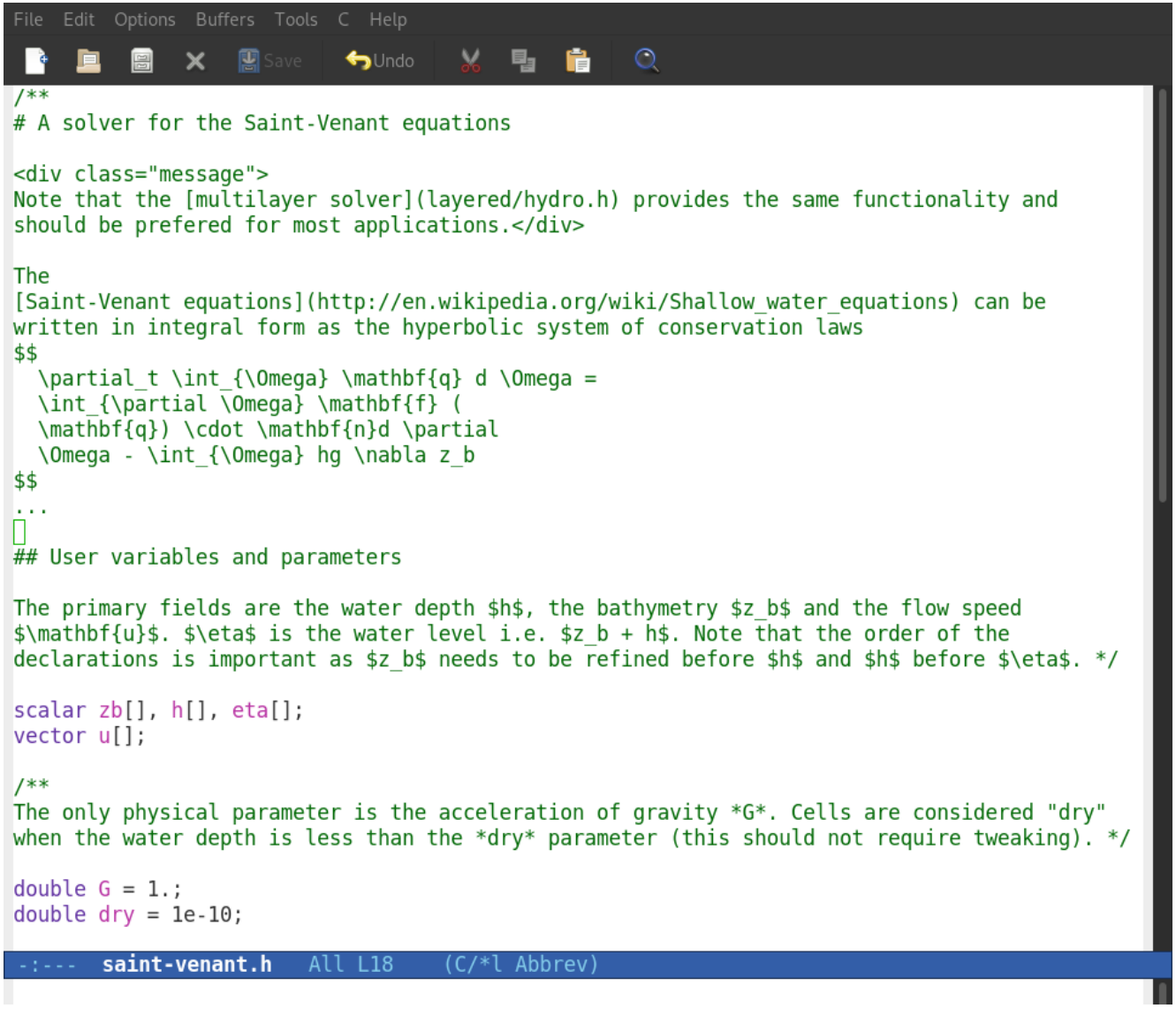
The primary fields are the water depth h, the bathymetry z_b and the flow speed $\mathbf{u}$. η is the water level i.e. $z_b + h$. Note that the order of the declarations is important as z_b needs to be refined before h and h before η.

```
scalar zb[], h[], eta[];
vector u[];
```

The only physical parameter is the acceleration of gravity *G*. Cells are considered "dry" when the water depth is less than the *dry* parameter (this should not require tweaking).

```
double G = 1.;
double dry = 1e-10;
```

The corresponding "unformatted" source



Stéphane Popinet. A vertically-Lagrangian, non-hydrostatic, multilayer model for multiscale free-surface flows. Journal of Computational Physics, *418:109609, 2020.*

All results were obtained using a timestep given by the stability condition (35), with the exception of the Green–Naghdi model which proved unstable, as expected, and had to use the more restrictive hydrostatic condition (17). Note that for the maximum value of $\hat{k}H \approx 60$ displayed on the figure and for 128 grid points per wavelength, the ratio of the non-hydrostatic timestep to hydrostatic timestep is approximately $\sqrt{H/\Delta} = \sqrt{128\,\hat{k}\,H/2\,\pi} \approx 35$.

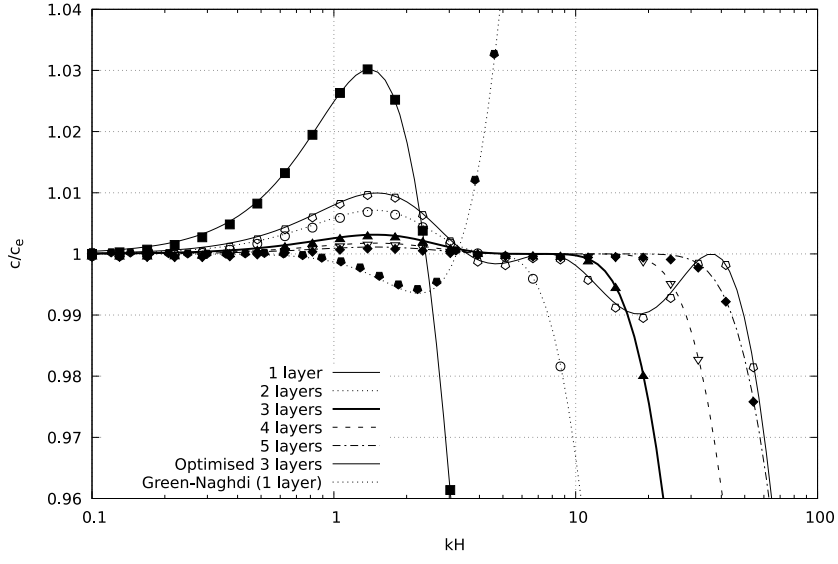


Figure 4. Dispersion relation. The lines are the analytical solutions and the symbols the numerical results (http://basilisk.fr/src/test/dispersion.c).

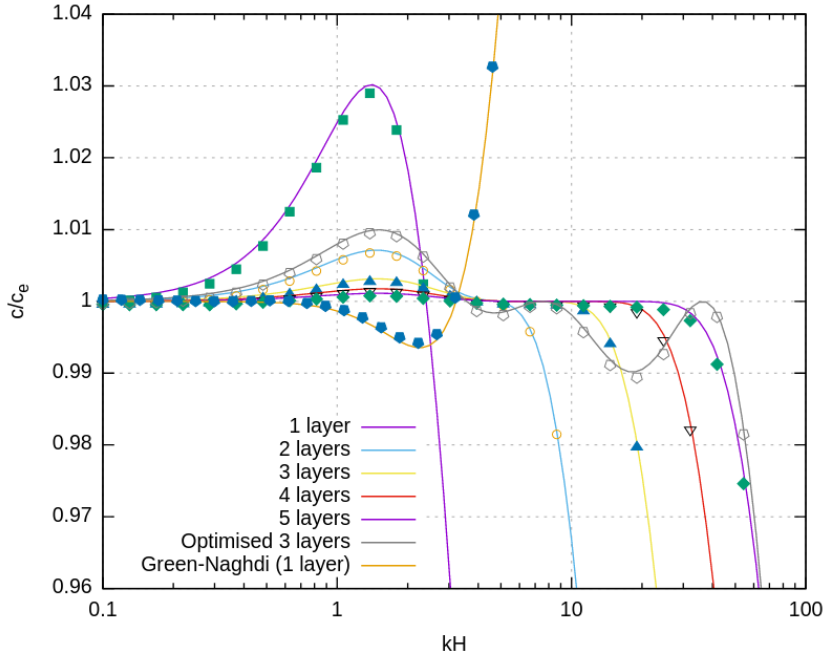
The same test case is used to estimate the numerical total energy variation per period,

Corresponding code (test case) on the website

The code runs and generates the (up to date) figures on the website

Note that the standard Green-Naghdi model has a [2,2] Padé approximant dispersion relation (see e.g. Clamond et al. (2017)). The same as Nwogu (1993). Here we used the optimized version of Chazel et al. (2011).

The discrete dispersion relations can be computed using this Maxima script.



Dispersion relation (script)

```
g = 1.
k = 1.

omega1_keller(h_0)=sqrt((4*h_0*g*k**2)/(h_0**2*k**2+4))

omega2_keller(h_0,h_1)=sqrt(((4*h_0*h_1**2+4*h_0**2*h_1)*g*k**4+(16*h_1+16*h_0)*g*k**2)/(h_0**2*h_1**2*k**4+(4*h_1**2+16*h_0*h_1+4*h_0*

omega3_keller(h_0,h_1,h_2)=sqrt((((4*h_0*h_1**2+4*h_0**2*h_1)*h_2**2+4*h_0**2*h_1**2*h_2)*g*k**6+((16*h_1+16*h_0)*h_2**2+(16*h_1**2+64*
```

Conclusions

- People are more important than Software $\Rightarrow$ **communication!**
- Free Software / Open Source / Open Science can leverage a much broader development community than possible using other *management* approaches
- This is particularly important for *multiphysics* applications
- This will only work if development environments are **accessible**

  Simplicity, Flexibility and Efficiency
- Many tools are available: wikis, forums, markdown, version control, continuous integration etc.
- **basilisk.fr** is one example of how this can be done

