Lessons from Pluto AMR

Adaptive mesh refinement in numerical astrophysics

Maitraya Bhattacharyya

Max-Planck-Institut für Astronomie, Heidelberg, Germany

[with David Melon Fuksman, Mario Flock, Giancarlo Mattia, Andrea Mignone & gPLUTO devs]

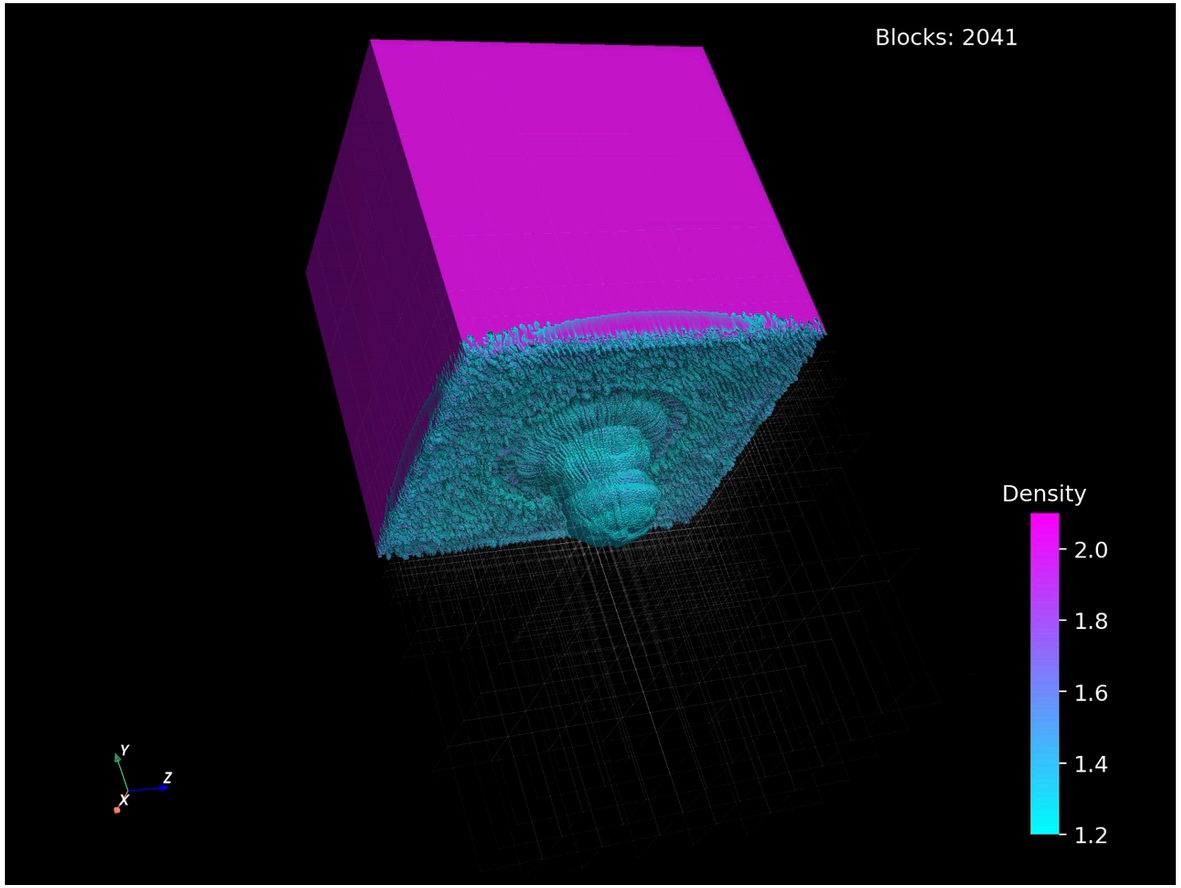
AthenaK Summer School 2026

July 10, 2026

Motivation

Why adapt the mesh at all?

- You already know why! **See Jim's talk from Monday**.
- a uniform grid at that resolution is *hopelessly expensive* so put cells where the physics is!



3D Rayleigh-Taylor instability

[Melon Fuksman w gPLUTO]

Outline

Mesh refinement strategies

- Patch based
- Oct tree
- Moving meshes
- No mesh!

PLUTO → gPLUTO

same physics for GPUs is gBetter™ → oct-trees

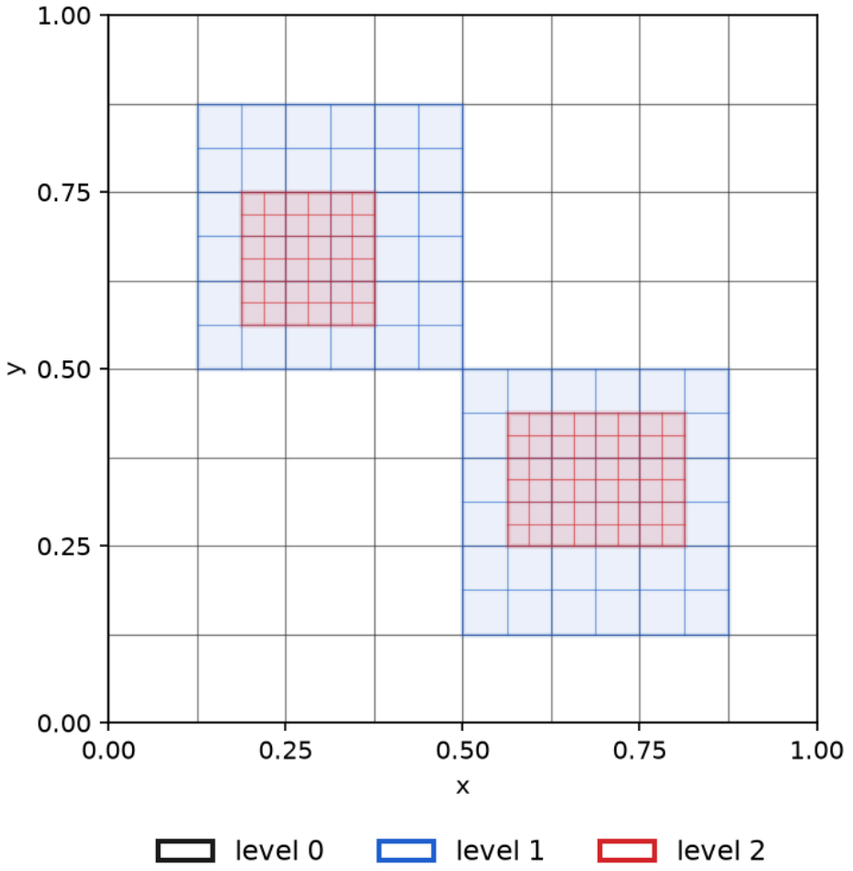
[Melon Fuksman]

Staggered-grid AMR

$\nabla \cdot \mathbf{B} = 0$ for staggered grid

Patch based methods

- **[Berger & Oliger 1984]**
- whole domain is covered by a global coarse grid (level 0)
- refinement overlays rectangular patches on the regions that need resolution
- levels increase resolution by 2
- only restriction: a fine patch (level $l+1$) lies entirely within its parent (level l)
- **Trade-off**: keeping the boxes rectangular covers some “calm” space around the feature



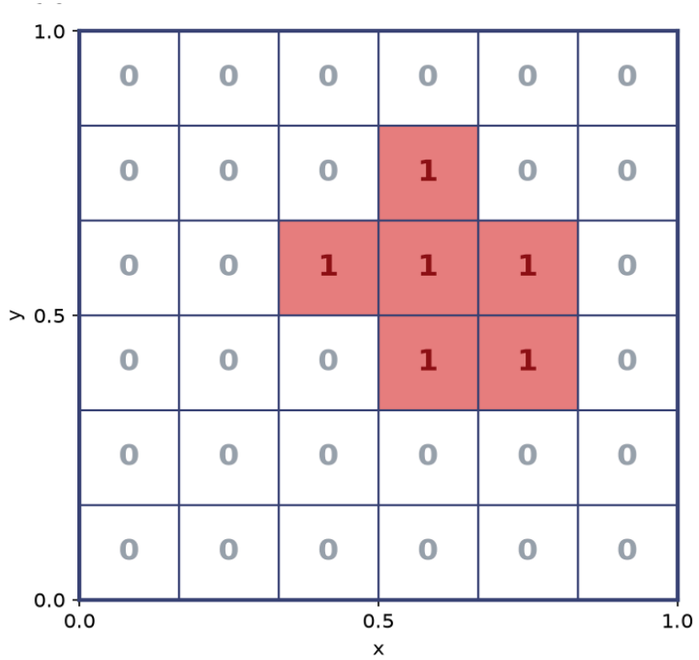
Refinement

- **Richardson extrapolation [Berger & Oliger 1984]**: criteria based on local estimation of the truncation error.

  One step on the $2h$ grid (u_c) vs two steps on the h grid (u_f) estimates this:

$$\tau_{loc} \approx \frac{u_c - u_f}{2^p - 1} \qquad (\text{order } p)$$

- **tag** cell **1** if $|\tau_{loc}| >$ threshold, else **0**
- Patches must cover the flagged cells → partitioning algorithms!



threshold $\varepsilon = 0.5$

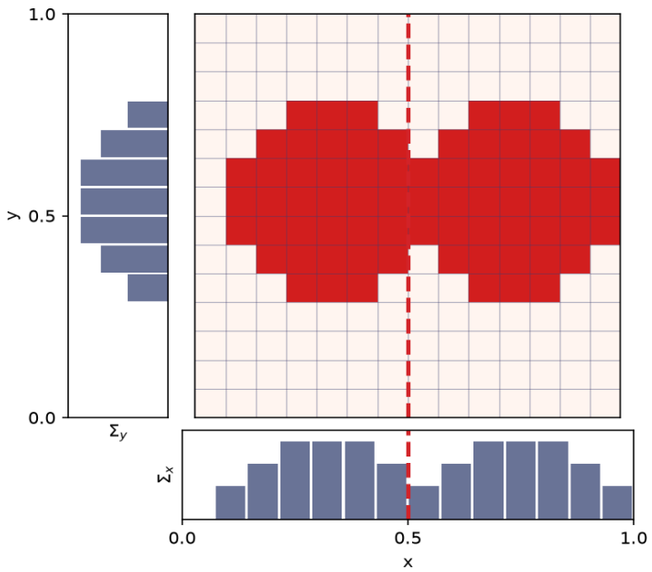
Clustering algorithm

- **[Berger & Rigoutsos 1991]** turn the flagged cells into a few rectangles
- **1D signatures**:

$$\Sigma_x(x) = \sum_y f(x,y), \qquad \Sigma_y(y) = \sum_x f(x,y)$$

- **split** – at a **hole** ($\Sigma = 0$) or an **inflection point** (a zero-crossing of $\Delta^2\Sigma$) to isolate clusters
- **stop** – recurse until every box meets the fill target:

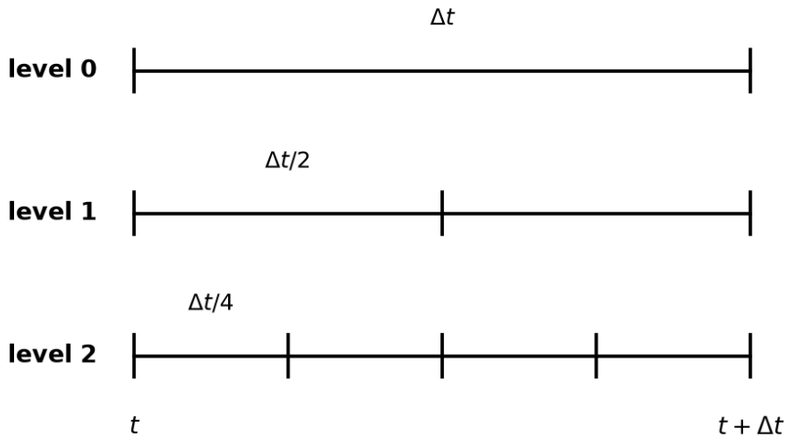
$$\eta = \frac{\sum f}{N_{\text{box}}} \geq \eta_{\text{target}}$$



split at a hole ($\Sigma_x = 0$) or at an inflection point ($\Delta^2\Sigma_x$ zero-crossing)

Temporal sub-cycling

- Because of CFL constraint decouple the clocks
- **time BCs**: fine ghost zones between coarse time points are filled by **linear interpolation in space *and* time**



Refluxing

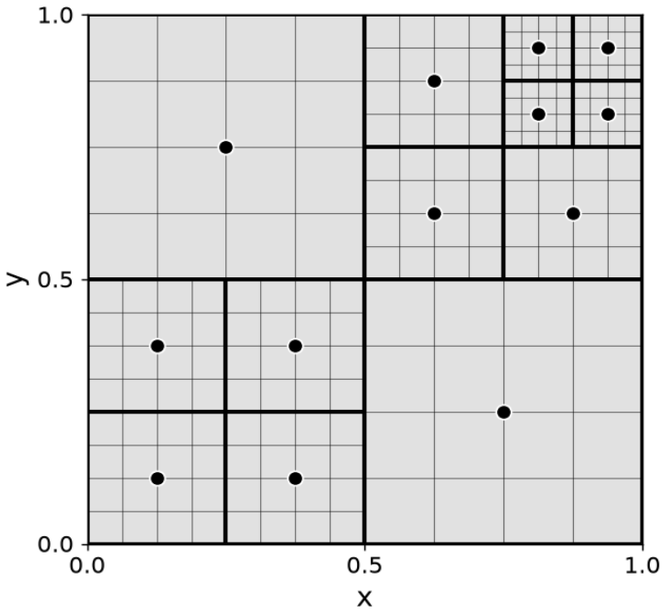
- **[Berger & Colella 1989]**
- coarse step and the r fine sub-steps compute *different* total fluxes across their shared face
- loss of conservation: uncorrected, mass & energy are created/destroyed at the patch perimeter
- Store the time-averaged fine flux minus the coarse flux:

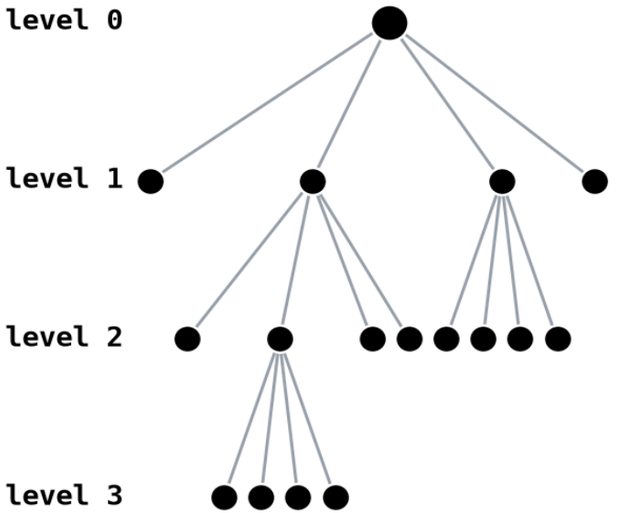
$$\delta F_{i+1/2} = \frac{1}{r} \sum_{k=1}^{r} F_{\text{fine}}^{(k)} - F_{i+1/2}^{C}$$

- trust the fine grid: apply δF to the coarse cells just outside the patch → strict conservation

Tree-based AMR

- The fundamental unit of the grid is a contiguous block, the **MeshBlock**
- A block flagged for refinement splits into 2^d children → every child keeps the same number of points
- Blocks never overlap, so coarse and fine levels only need to communicate at their shared boundaries
- A logical tree representation → neighbours are easy to find





refinement

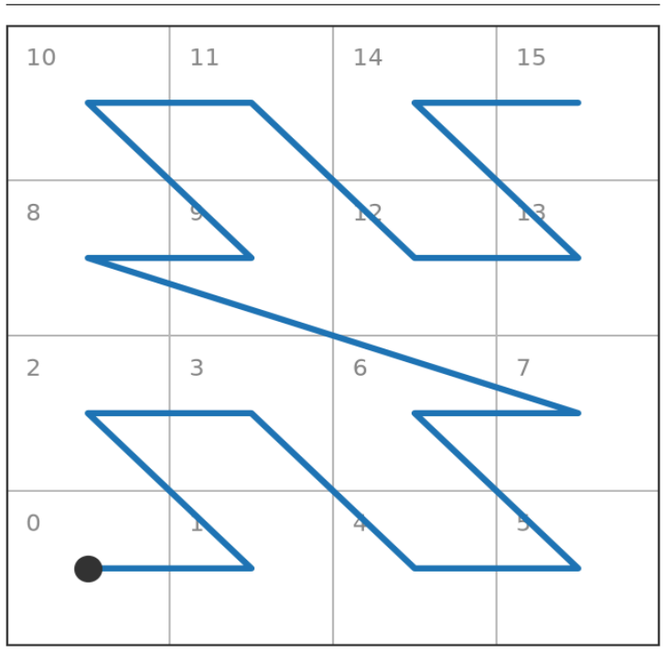
Space-filling curves and Morton order — Part I

In a large problem with thousands of meshblocks, these must be sensibly distributed across many GPUs. To keep inter-block communication efficient, neighbours should be assigned together whenever possible.

*A **space-filling curve** maps 3D physical coordinates onto a 1D line while preserving spatial locality.*

- **The Hilbert curve**: benchmark SFC: strictly continuous, folding on itself without ever crossing or jumping
- Cut the 1D curve into equal segments and every GPU gets the most compact volume possible → neighbors share face
- **Cons:** the Hilbert index needs stateful rotations and sequential bitwise XORs → regridding is expensive

- **Morton / Z-order**: just interleave the bits of (i, j, k) and level info
- cheap to compute
- occasional long “Z” jumps



Space-filling curves and Morton order — Part II

Shift each block onto the finest grid (by its level ℓ), then interleave bits of (i, j, k) into one **Morton key**:

$$\text{key} = \text{morton}\big(c \ll (L - \ell)\big)$$

leaf blocks (i, j) with level:

[(0,0)L0, (1,0)L0, (0,1)L0, (1,1)L0, (2,0)L0, (4,2)L1, (4,3)L1, (5,3)L1, (10,4)L2, (11,4)L2, (10,5)L2, (11,5)L2]

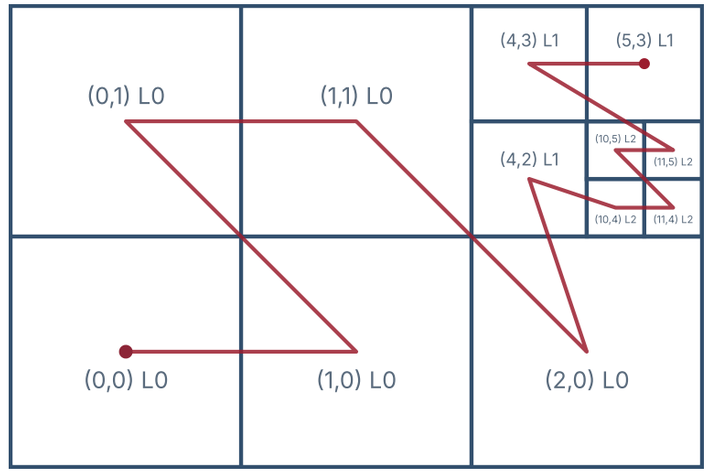
Morton keys

[0, 16, 32, 48, 64, 96, 104, 108, 100, 101, 102, 103]

sort!

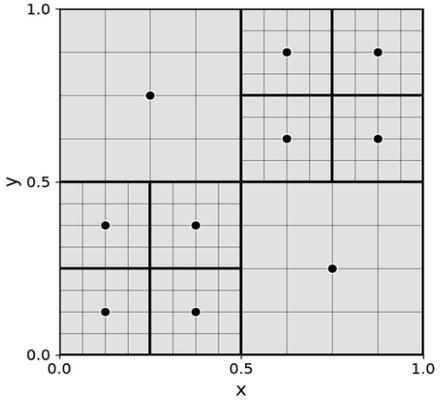
[0, 16, 32, 48, 64, 96, 100, 101, 102, 103, 104, 108]

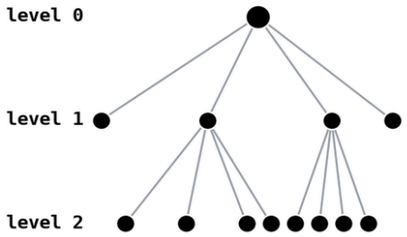
the sorted keys **are** the Z-order — the flat linear octree



Neighbor finding

- Ghost exchange, flux matching and the 2:1 ripple all ask **“who is next to me?”**
- **Tree walking** — the pointer-tree way:
  - **walk up** — follow *parent* pointers to the smallest ancestor that also contains the neighbour
  - **walk down** — descend its *child* pointers toward the neighbour until you reach a leaf
  - $O(\log N)$ pointer chases per query, scattered in memory → cache-unfriendly
- **Flat oct-tree** — no traversal! step the coords $(i, j, k) \to (i-1, j, k)$, recompute the **level-aware Morton key**, and probe the hash table → the neighbour’s address in $O(1)$
- **across levels** — if the key’s **parent** is a leaf → the neighbour is **coarser**; if finer keys fill its **span** → **finer** (2:1 balance allows only these)

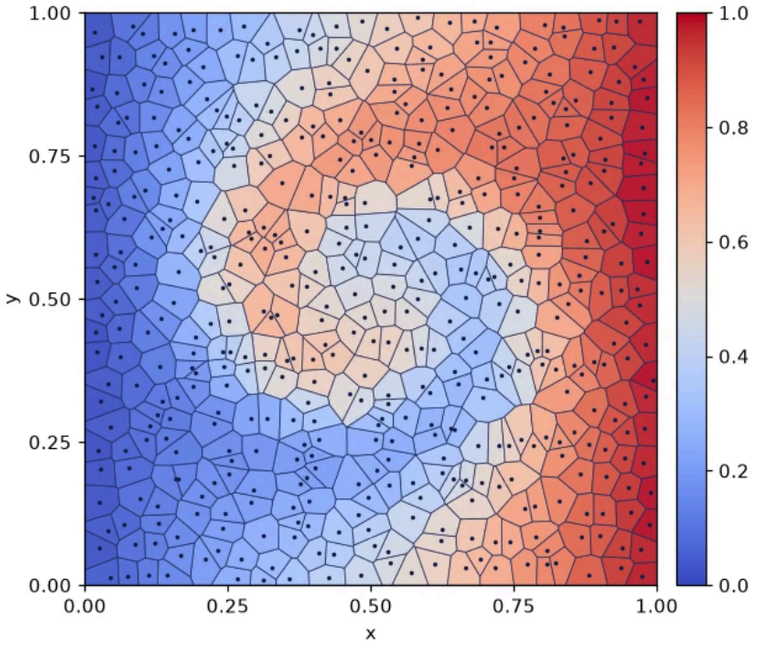




Moving meshes

A different philosophy: stop refining a *fixed* grid – let the mesh move with the fluid.

- middle ground between fixed grid and mesh-less methods → has Galilean invariance and can handle shocks better than SPH.
- cells = **Voronoi** regions around points that advect at the flow velocity **[Springel 2010]**
- the tessellation rebuilds every step: topology changes for free
- resolution **follows the mass** “natural refinement

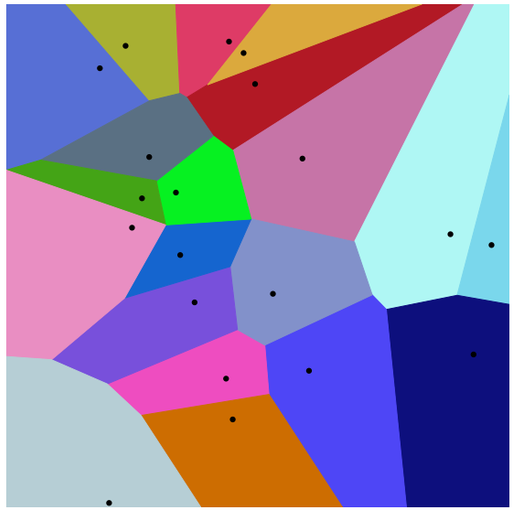


the mesh advecting with the flow

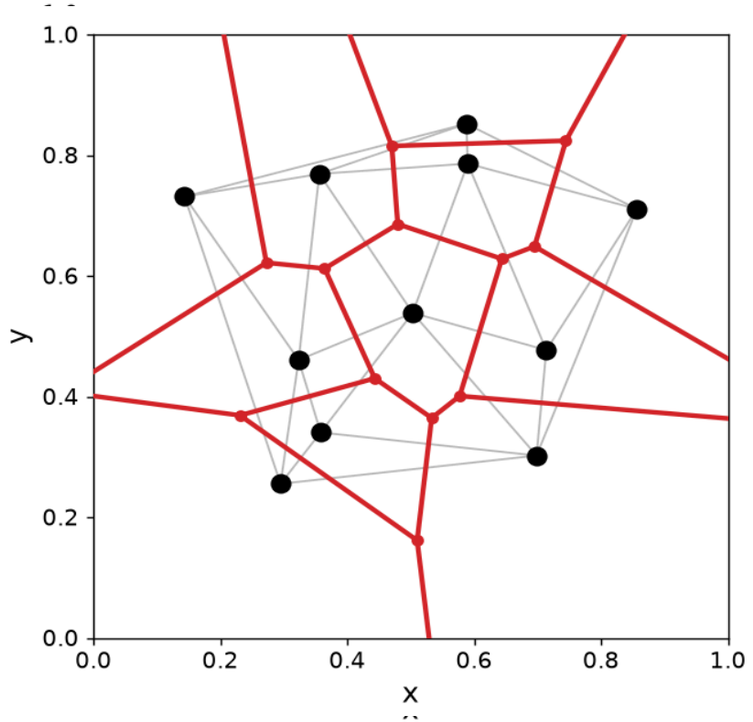
Building a Voronoi mesh

A Voronoi mesh tiles space into cells — each cell is everything closer to its generator r_i than to any other point.

In practice you build its dual, the **Delaunay triangulation**.



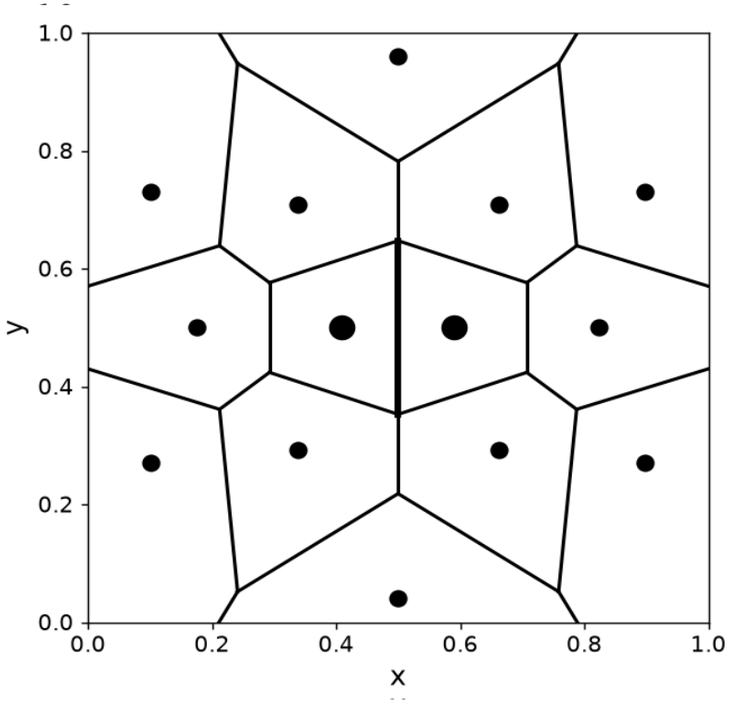
Voronoi diagram, Wikimedia Commons



Delaunay triangulation and its dual

Refinement and derefinement — I

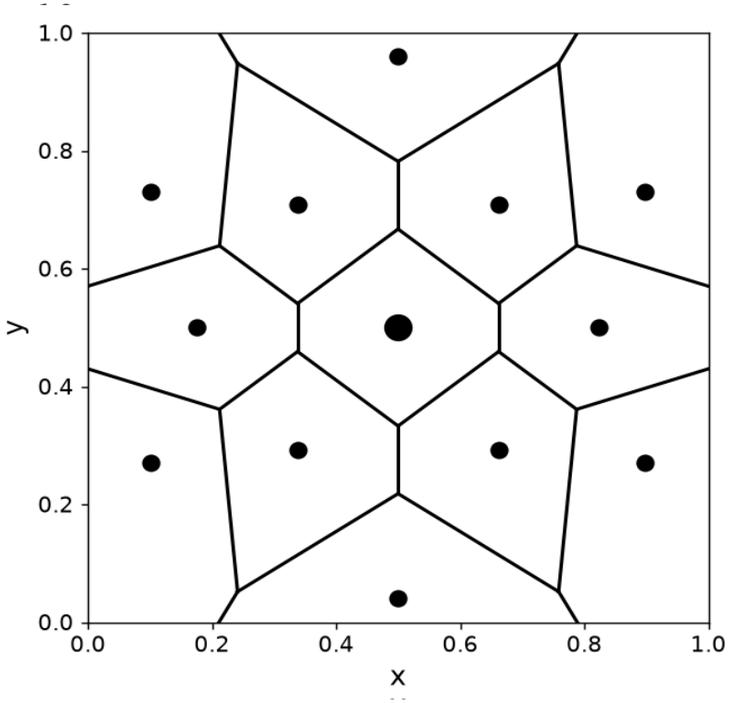
- Keep cell masses approximately equal
- Check against a threshold mass
- **Point splitting**: delete the flagged parent, inject **two** new points in its place, displaced slightly from the centre along the cell's longest dimension
- **Conserve the physics**: mass, momentum and internal energy split in half $m_{\text{child}} = \frac{1}{2}\, m_{\text{parent}}\,, \quad \mathbf{p}_{\text{child}} = \frac{1}{2}\, \mathbf{p}_{\text{parent}}$



one generator → two points along the long axis → the cell splits

Refinement and derefinement — II

- Just exact reverse
- **Point merging**: delete the two flagged neighbouring points and inject one point at their centre of mass
- Sum the children back into the parent



two points → merge to one at the centre of mass → the cells fuse

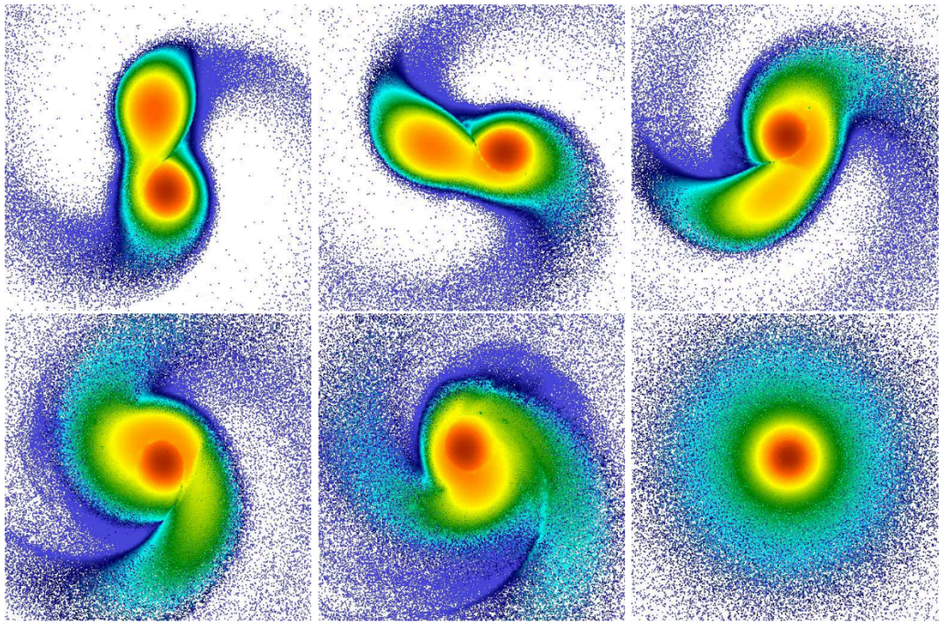
Disadvantages — expensive Voronoi rebuild each step · memory-intensive (connectivity + geometry) · mesh noise from irregular, moving cells

Meshless/SPH approaches

- No “mesh refinement” as there is no mesh! The particles are the fluid.
- Galilean invariance is perfect; mass, momentum conserved by construction.
- Empty space costs zero memory and compute.

Some cons:

- low-order & noisy; artificial viscosity for shocks
- $\nabla \cdot B$ by cleaning, not CT



[Diehl et al. 2008] – $q = 1.3$ DD (double-degenerate) merger with **NuGrid**

The SPH kernel

- Each particle carries fixed mass m_i and an interaction radius or the "smoothing length" h.
- Paricles move with the local fluid velocity.

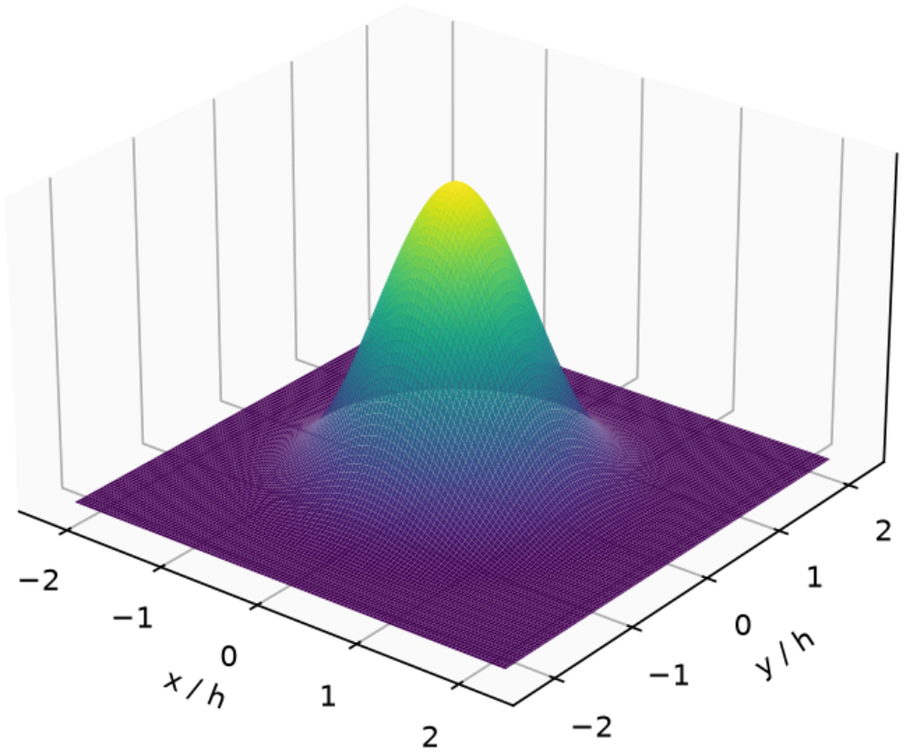
A quantity is a **kernel-weighted average [Gingold & Monaghan 1977, Lucy 1977]**:

$$\langle A(\mathbf{r})\rangle = \int A(\mathbf{r}')\, W(|\mathbf{r}-\mathbf{r}'|, h)\, d\mathbf{r}' \quad \longrightarrow \quad \langle A\rangle_i = \sum_j m_j \frac{A_j}{\rho_j} W_{ij}$$

The equations of evolution are functions of W and its derivatives ∇W **[Rosswog 2009, 2015]**:

$$\rho_i = \sum_j m_j\, W_{ij}, \qquad \frac{d\mathbf{v}_i}{dt} = -\sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}\right) \nabla_i W_{ij}$$

$$\frac{du_i}{dt} = \sum_j m_j \frac{P_i}{\rho_i^2} (\mathbf{v}_i - \mathbf{v}_j)\cdot \nabla_i W_{ij} \qquad W_{ij} \equiv W(|\mathbf{r}_i - \mathbf{r}_j|, h)$$

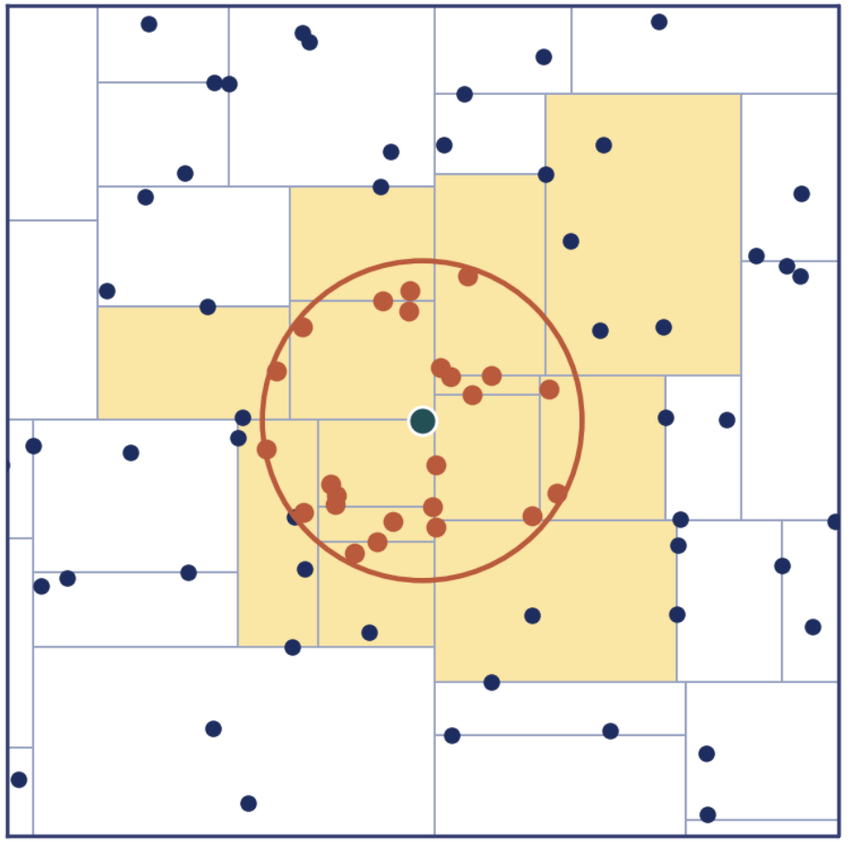


A cubic-spline kernel $W(r, h)$

Finding neighbours

Every particle needs the $\sim N_{\mathrm{SPH}}$ others inside its kernel support $\lesssim 2h$, but

- distance calculation $O(N^2)$ expensive!
- sort the particles into a spatial tree; walk only the cells overlapping the $2h$ sphere → $O(N \log N)$
- Rosswog et. al. uses a **recursive coordinate bisection (RCB) tree** [Gafton & Rosswog 2011]



Searches only within cells overlapping the $2h$ circle

A comparison

	Patch based (AMReX, Chombo)	Block oct-tree (AthenaK, gPLUTO)	Moving mesh (AREPO)	SPH (SPHINCS_BSSN)
refinement unit	arbitrary rectangle	fixed block “meshblock”/“amrblock”	Voronoi cell	particle
adaptivity	discrete, flagged	discrete, per block	continuous, co-moving	continuous, adaptive h
data structure	patches + clustering	oct-tree of blocks	Voronoi tessellation	neighbour lists
Galilean-invariant	✗	✗	✓	✓
$\nabla \cdot B = 0$	CT + reflux	**staggered CT + reflux**	cleaning	cleaning

The PLUTO code

Solves a system of **conservation laws** [Mignone+ 2007]:

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{T}(\mathbf{U}) = \mathbf{S}(\mathbf{U})$$

For **MHD**, the conserved state and flux:

$$\mathbf{U} = \left[\rho,\ \rho\mathbf{v},\ \mathbf{B},\ E\right]^{\mathsf{T}}$$

$$\mathbf{T} = \begin{bmatrix} \rho\mathbf{v} \\ \rho\mathbf{v}\mathbf{v} - \mathbf{B}\mathbf{B} + p_t\mathbf{I} \\ \mathbf{v}\mathbf{B} - \mathbf{B}\mathbf{v} \\ (E + p_t)\mathbf{v} - (\mathbf{v}\cdot\mathbf{B})\mathbf{B} \end{bmatrix}$$

- **PLUTO-AMR** [Mignone+ 2012] – AMR via **Chombo**

PLUTO/gPLUTO devs and users @ MPIA, 2026

Public repo – gitlab.com/PLUTO-code/gPLUTO
Webpage – plutocode.ph.unito.it

PLUTO → gPLUTO

- Built within the EuroHPC ***S**calable **P**arallel **A**strophysical **C**odes for **E**xascale* (SPACE)
- Ready PLUTO for exascale on heterogeneous **CPU + GPU**

Directive-based port is easy and incremental

- one C++ source; annotate the hot loops for target hardware:

```
// OpenACC
#pragma acc parallel loop
for (i = 0; i < n; i++) update(i);

// OpenMP
#pragma omp target teams distribute parallel for
for (i = 0; i < n; i++) update_something(i);

// hierarchical: gang over blocks, vector within
#pragma acc parallel loop gang
for (b = 0; b < nblk; b++)
  #pragma acc loop vector
  for (i = 0; i < bs; i++) update_something(b, i);
```

Jan 2023 — **Simple 2D HD mini-app (C, ~1000 lines)**
code restructure, array re-organization

Mar 2023 — **Added OpenACC**
~10× speedup (GPU vs CPU)

Oct 2023 — **New kernels (MHD, MPI, Riemann solvers)**
+ user-interface; more physics & portability

Jan 2024 — **Switched to C++, relativistic modules**
templates, classes, vectors; up to 50×

Feb 2025 — **4th-order methods, ResRMHD; α-release**
MPI on CPU+GPU; more physics; docs started

Mar 2025 — **Poisson solver, particles, diffusion op.**

2026 — **1st stable release**

[Andrea Mignone]

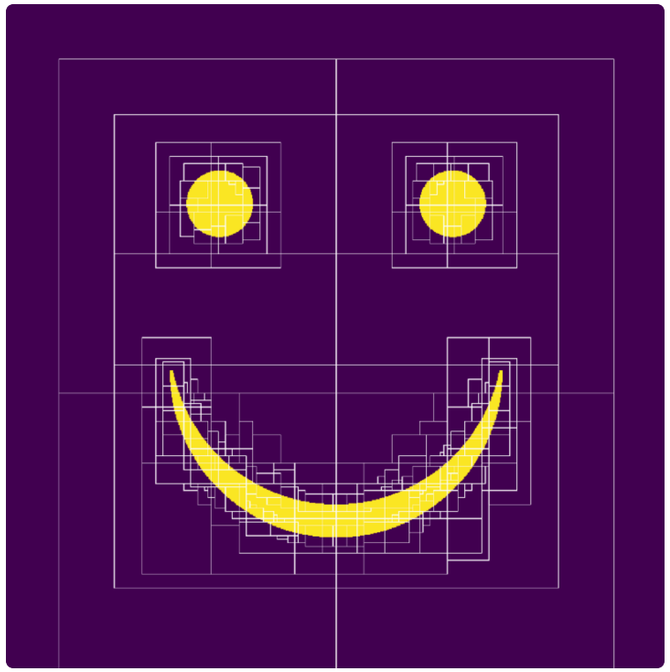
Status of Port

	PLUTO	gPLUTO	Pub
Basic physics			
HD (Euler)	✓	✓	✓
MHD	✓	✓	✓
RHD (rel. hydro)	✓	✓	✓
RMHD (ideal)	✓	✓	✓
ResRMHD (non-ideal)	✓	✓	✓
GRMHD	✗	~ *dev*	–
Dissipation			
Viscosity	✓	✗ *planned*	–
Thermal conduction	✓	✗ *planned*	–
Resistivity	✓	✓	✓
Ambipolar diffusion	✗	~ *wip*	–
Hall MHD	✓	✗	–
EoS			
Ideal / isothermal	✓	✓	✓
Synge gas	✓	✓	✓
Non-constant γ	✓	✗	–

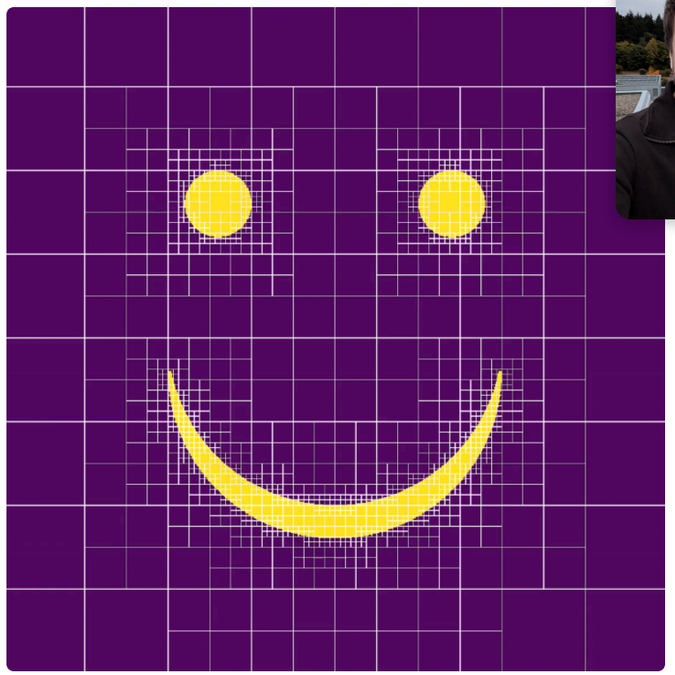
	PLUTO	gPLUTO	Pub
Geometry			
Cartesian	✓	✓	✓
Cylindrical	✓	✓	✓
Spherical	✓	✓	✓
Cooling			
Tabulated	✓	✓	✓
Power law	✓	✗ *planned*	–
SNEq	✓	✓	✓
MINEq	✓	✗	–
Radiation / Particles			
Radiation (classical)	✓	✓	✓
Radiation (relativistic)	✓	✓	✓
Cosmic rays	✓	~ *wip*	✗
Dust	✓	✗ *planned*	–
Lagrangian particles	✓	✓	✗

[A. Mignone]

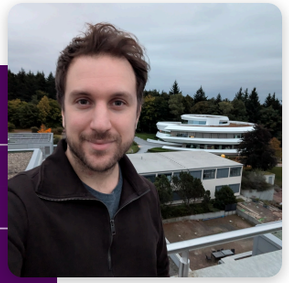
A new AMR for gPLUTO



Legacy PLUTO uses CHOMBO



gPLUTO oct-tree [David Melon Fuksman]



The AMR structure of gPLUTO

- gPLUTO implements the **oct-tree**! → this is easier than a patch-based approach!
- AMR support for **non-Cartesian coordinates** (cylindrical, spherical)
- **AMRBlock** – the basic unit of the grid, equivalent to AthenaK's **MeshBlock**
- **AMRRoot** – the class holding all AMRBlocks, arrays and methods
- Each MPI ranks gets a collection of **AMRBlocks** following the Morton curve
- Send and receive data through buffers happens through async MPI comunications
- the tree structure is stored on **rank 0** for now – *changing soon*

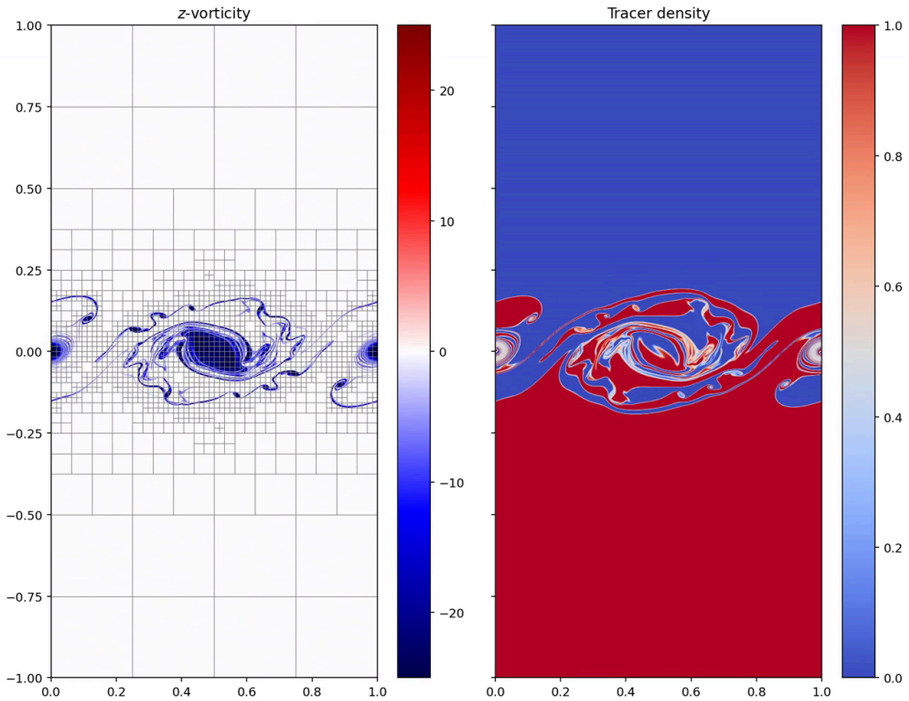
Three things happen each step

1. **Fill boundaries**
   prolong / restrict / copy
2. **Integrate equations**
   in each block
3. **Ensure conservation**
   refluxing

Refluxing at work — Kelvin–Helmholtz

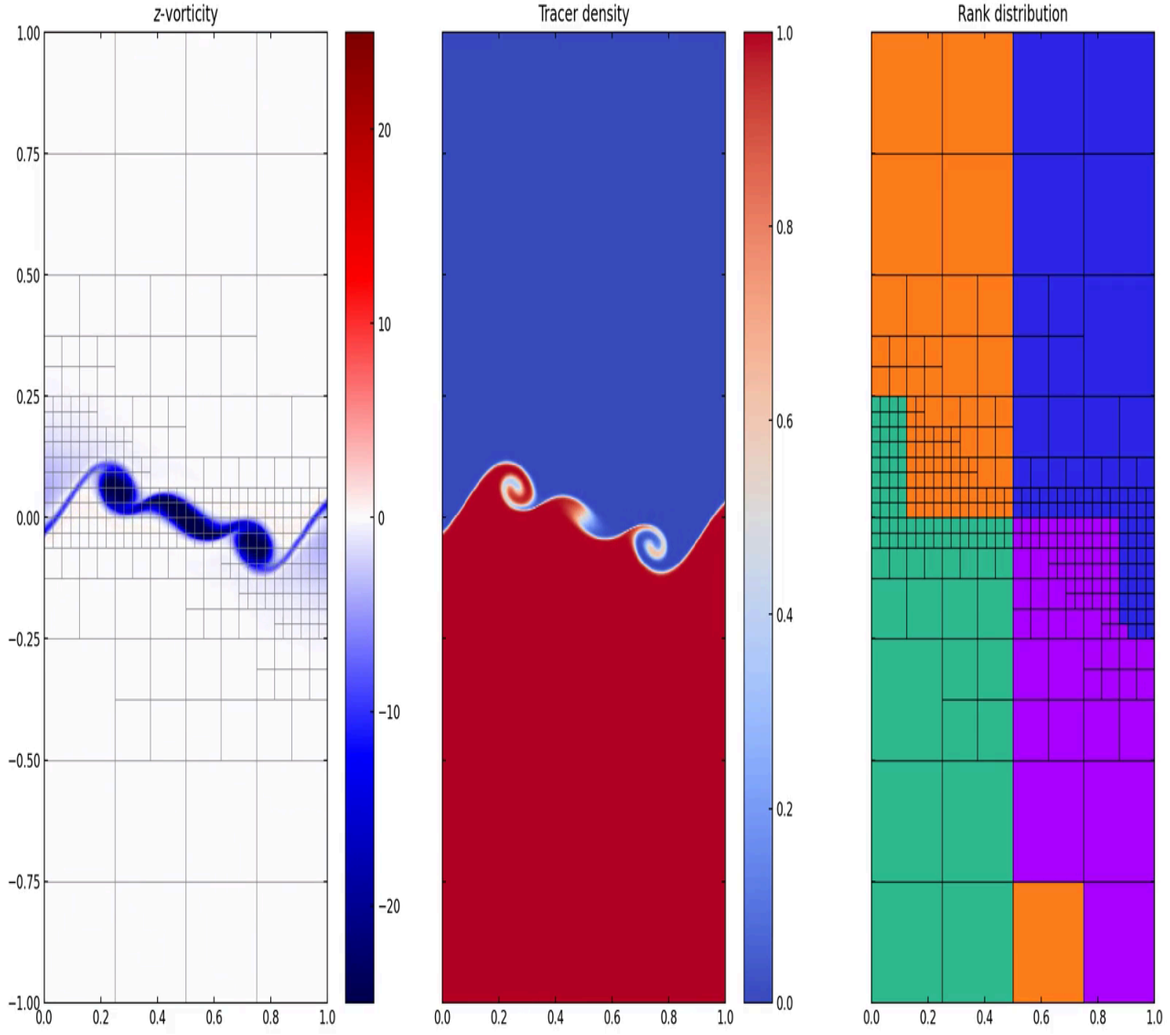
- the KH roll-up straddles coarse-fine boundaries, so it is a sharp test of **flux conservation**
- **fixed refluxing** keeps the vortices sharp and free of spurious features at the level jumps

[D. Melon Fuksman · gPLUTO]



Kelvin-Helmholtz, max level 5, Roe, fixed refluxing

Load balancing

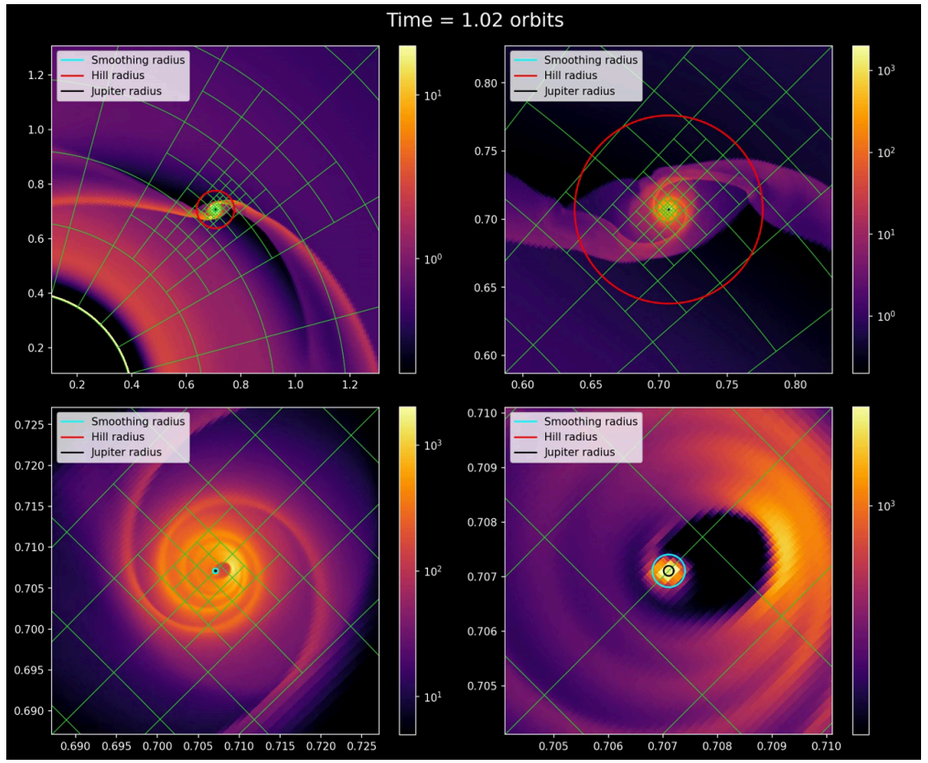


Load balancing, colors are ranks

[Melon Fuksman]

Non-Cartesian geometry AMR

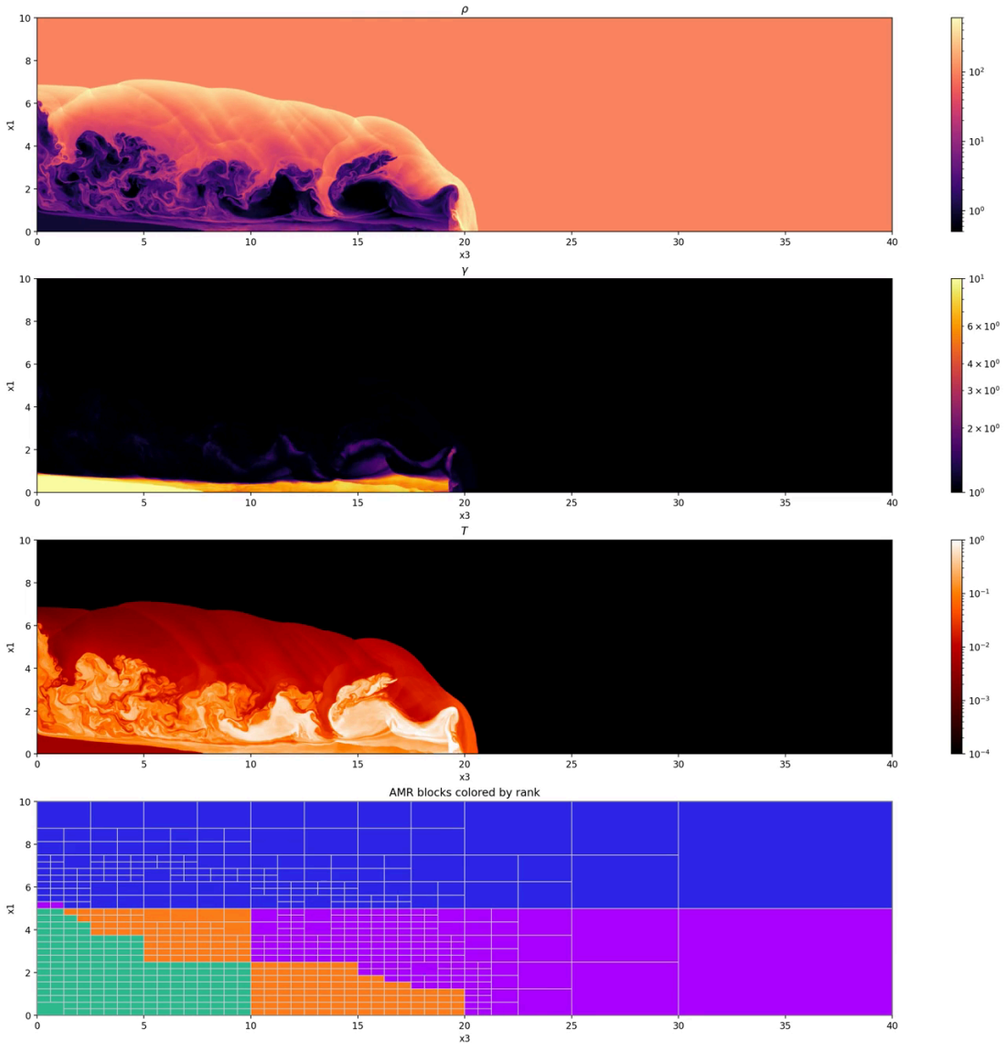
- 10 refinement levels
- similar strategy for oct-tree as cartesian
- Effctive resolution: 32768 × 49152



[Melon Fuksman]

Planet-disk 2d cylindrical with AMR

Relativistic AMR — the jet



relativistic jet with a strong shock

[Melon Fuksman]

Solenoidal constraint

- no magnetic monopoles!
- $\partial_t(\nabla \cdot \mathbf{B}) = -\nabla \cdot (\nabla \times \mathbf{E}) = 0$

But numerically

- discrete fluxes don't enforce $\nabla \cdot \mathbf{B} = 0$
- also truncation error at every step

Issues

- spurious force $\propto (\nabla \cdot \mathbf{B})\,\mathbf{B}$ – plasma pushed along $\mathbf{B}$
- wrong jump conditions, negative pressures, crashes

Solutions

- Source terms: Powell's eight-wave
- Divergence cleaning (GLM)
- Constrained transport – zero by construction

Powell's eight-wave Powell+ (1999)

Add a source $\propto \nabla \cdot \mathbf{B}$ to each affected equation

$$\partial_t(\rho\mathbf{v}) + \ldots = -(\nabla \cdot \mathbf{B})\,\mathbf{B}$$

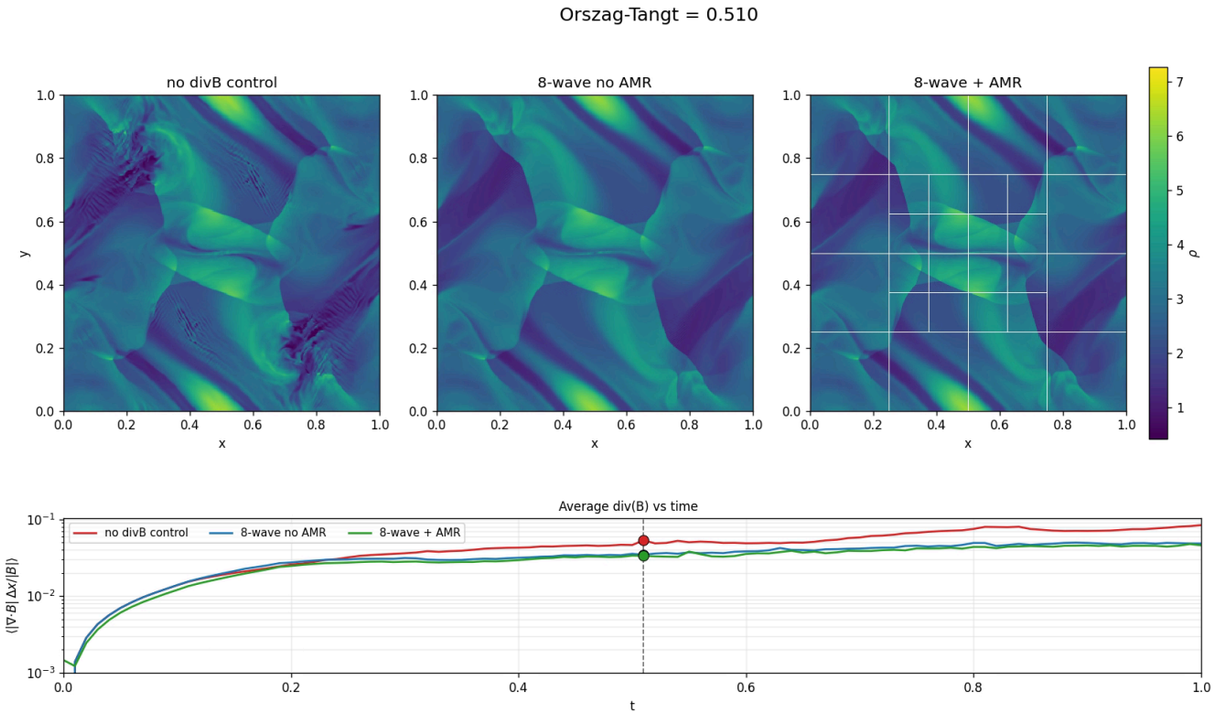
$$\partial_t\mathbf{B} + \ldots = -(\nabla \cdot \mathbf{B})\,\mathbf{v}$$

$$\partial_t E + \ldots = -(\nabla \cdot \mathbf{B})\,(\mathbf{v} \cdot \mathbf{B})$$

- Adds an **8th wave** – $\nabla \cdot \mathbf{B}$ is advected with the flow
- local monopole swept off the numerical domain
- simple, robust but **non-conservative**

The Orszag–Tang vortex

- **Domain:** $[0,1] \times [0,1]$, 256^2, all boundaries **periodic**
- **Uniform Density:** $\rho = 25/9$
- **Uniform Pressure:** $p = 5/3$ (uniform)
- **Velocity:** $\mathbf{v} = (-\sin 2\pi y,\ \sin 2\pi x,\ 0)$
- **Magnetic field:** $\mathbf{B} = (-\sin 2\pi y,\ \sin 4\pi x,\ 0)$
- **EOS:** ideal, $\gamma = 5/3$
- **Scheme:** ideal MHD + RK2 + linear reconstruction



Divergence cleaning (GLM)

Couple $\nabla \cdot \mathbf{B}$ to a new scalar field ψ [Dedner+ 2002]:

$$\partial_t \mathbf{B} + \ldots = -\nabla \psi$$

$$\partial_t \psi + c_h^2 \left(\nabla \cdot \mathbf{B} \right) = -\frac{c_h^2}{c_p^2} \psi$$

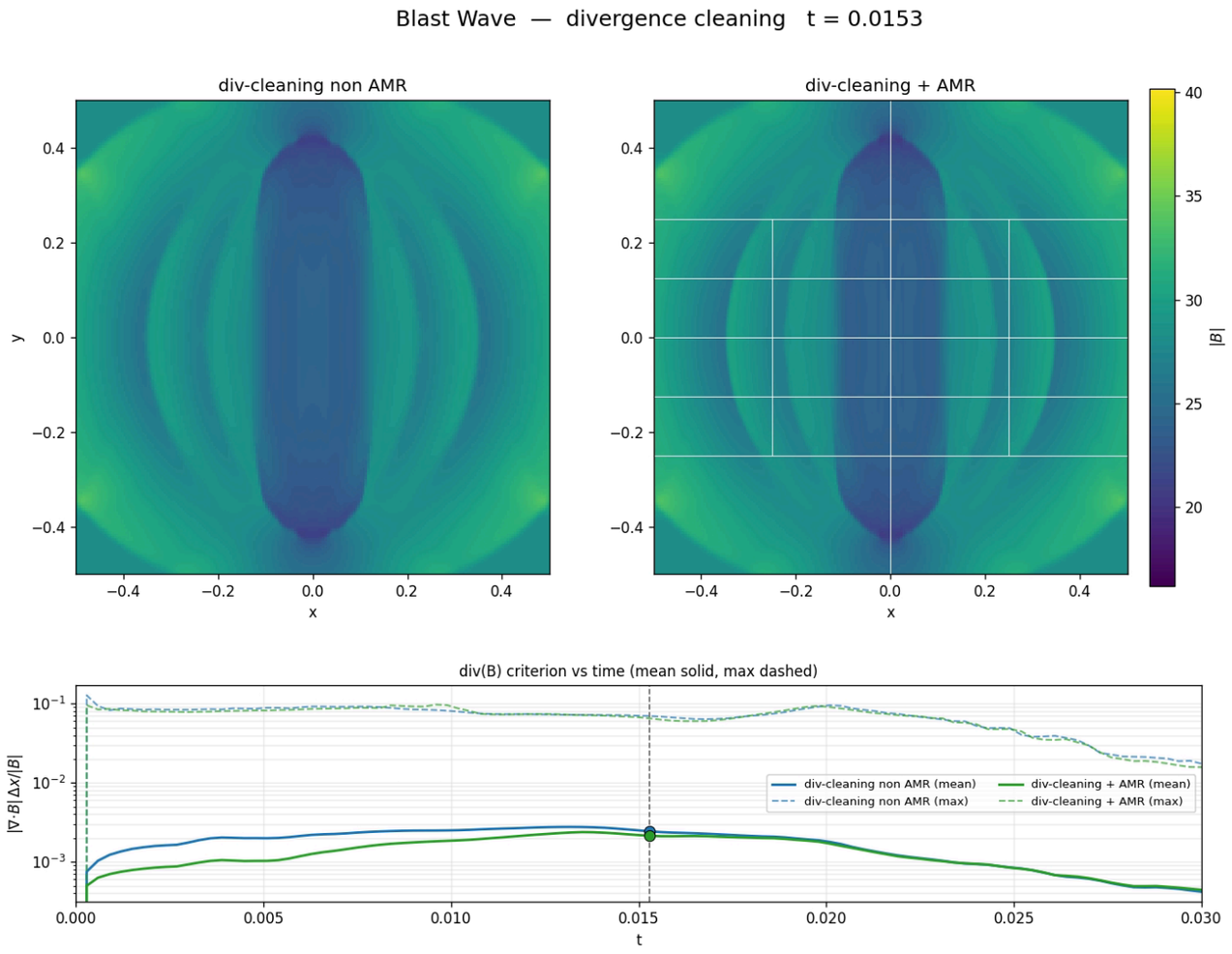
- ψ is a generalized Lagrange multiplier sourced by $\nabla \cdot \mathbf{B}$

$D \equiv \nabla \cdot \mathbf{B}$ obeys a damped wave equation:

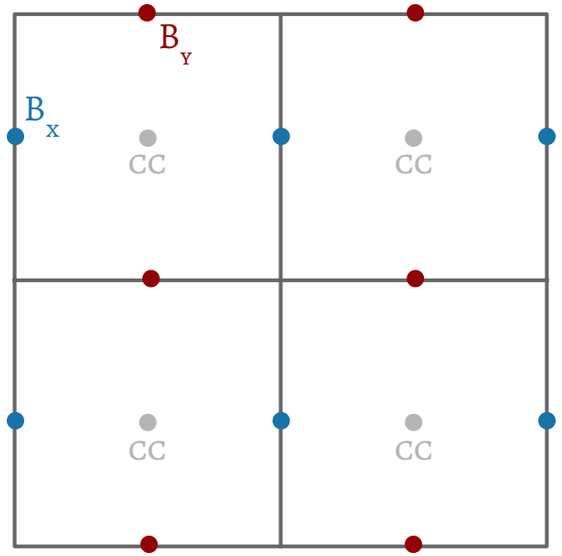
$$\partial_t^2 D = \underbrace{c_h^2 \nabla^2 D}_{\text{propagation}} - \underbrace{\frac{c_h^2}{c_p^2} \partial_t D}_{\text{damping}}$$

- error is **propagated away** at speed c_h and **damped** on a timescale set by c_p
- conservative and cheap — but only damps; AMR must support this extra equation

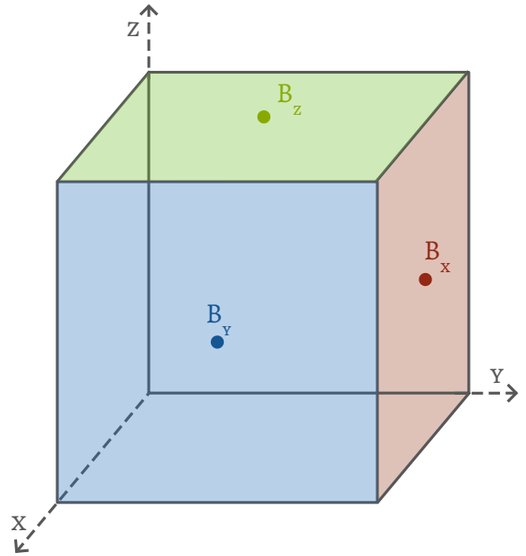
Blast wave


Blast Wave — divergence cleaning t = 0.0153
div-cleaning non AMR
div-cleaning + AMR
y
x
|B|
div(B) criterion vs time (mean solid, max dashed)
$|\nabla\cdot B|\,\Delta x/|B|$
t
div-cleaning non AMR (mean)
div-cleaning non AMR (max)
div-cleaning + AMR (mean)
div-cleaning + AMR (max)

The staggered grid: Magnetic field

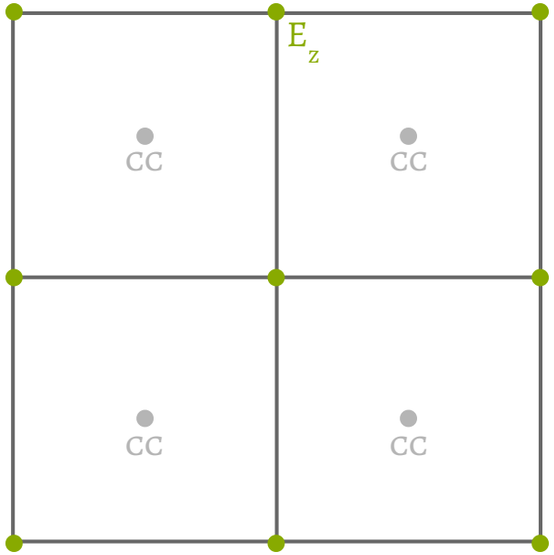


- B_x, B_y at midpoints of x and y faces respectively
- Each face midpoint shared between two neighbouring cells

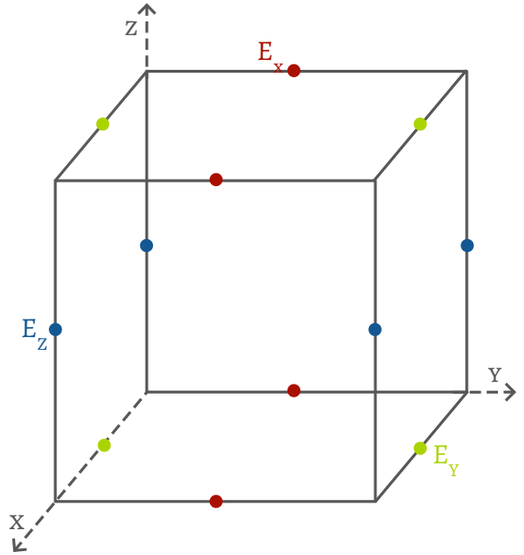


- B_x on yz-faces, B_y on xz-faces, B_z on xy-faces
- Each face value shared between two neighboring cells

The staggered grid: Electric field

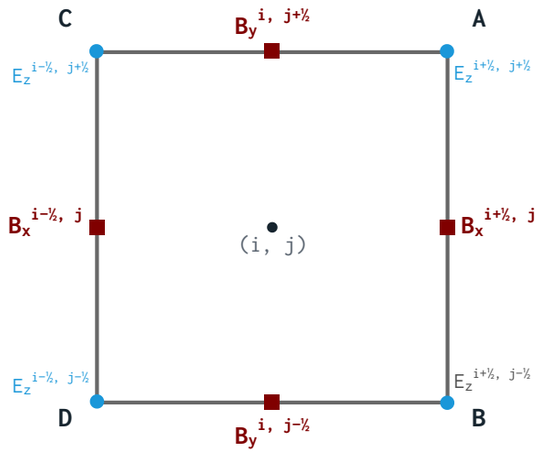


- E_z lives at **corners** of grid cells
- Each corner value shared by **4 adjacent cells**



- E_x, E_y, E_z at the midpoint of their own edge – x, y, z edges respectively
- Each edge value shared between four neighboring cells

$\nabla \cdot \mathbf{B} = 0$ preserved to machine precision



For 2D:

$$\frac{\partial B_x^{i+\frac{1}{2},j}}{\partial t} = -\frac{E_z^{i+\frac{1}{2},j+\frac{1}{2}} - E_z^{i+\frac{1}{2},j-\frac{1}{2}}}{\Delta y}, \qquad \frac{\partial B_y^{i,j+\frac{1}{2}}}{\partial t} = +\frac{E_z^{i+\frac{1}{2},j+\frac{1}{2}} - E_z^{i-\frac{1}{2},j+\frac{1}{2}}}{\Delta x}$$

Discrete divergence at cell:

$$(\nabla \cdot \mathbf{B})_{i,j} = \frac{B_x^{i+\frac{1}{2},j} - B_x^{i-\frac{1}{2},j}}{\Delta x} + \frac{B_y^{i,j+\frac{1}{2}} - B_y^{i,j-\frac{1}{2}}}{\Delta y}$$

Four corner EMFs: $A = E_z^{i+\frac{1}{2},j+\frac{1}{2}}$, $B = E_z^{i+\frac{1}{2},j-\frac{1}{2}}$, $C = E_z^{i-\frac{1}{2},j+\frac{1}{2}}$, $D = E_z^{i-\frac{1}{2},j-\frac{1}{2}}$

$$\frac{1}{\Delta x}\left(\dot{B}_x^{i+\frac{1}{2},j} - \dot{B}_x^{i-\frac{1}{2},j}\right) = -\frac{A - B - C + D}{\Delta x\, \Delta y}, \qquad \frac{1}{\Delta y}\left(\dot{B}_y^{i,j+\frac{1}{2}} - \dot{B}_y^{i,j-\frac{1}{2}}\right) = +\frac{A - B - C + D}{\Delta x\, \Delta y}$$

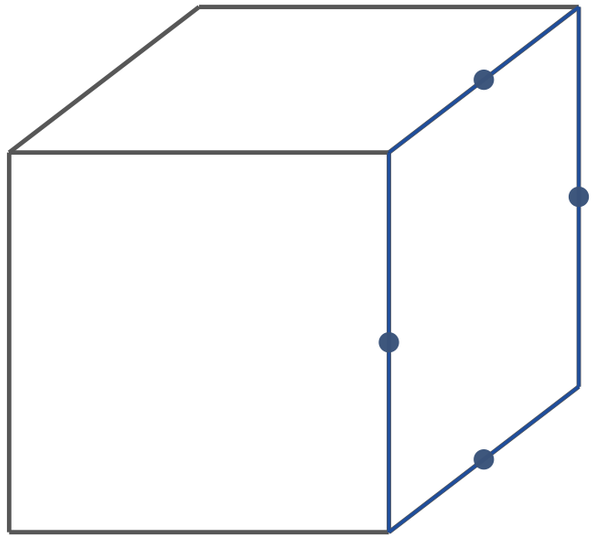
Hence:

$$\frac{d}{dt}(\nabla \cdot \mathbf{B})_{i,j} = 0$$

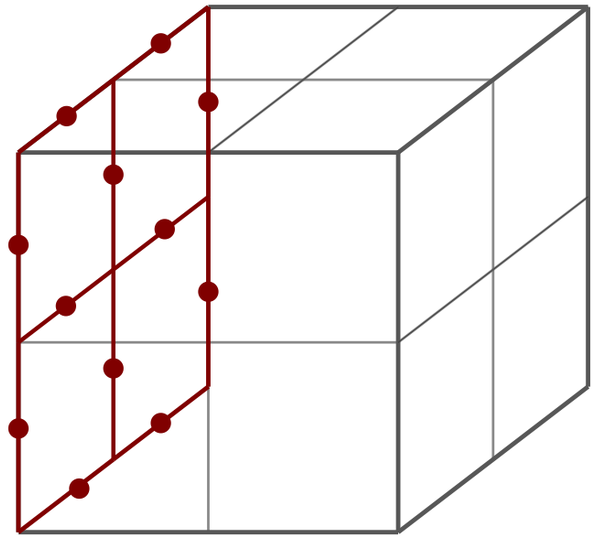
Now add AMR — the real difficulty

- Naïve interpolation of a **face-staggered** field across a coarse-fine boundary **destroys** $\nabla \cdot \mathbf{B} = 0$
- Need **divergence-free prolongation & restriction** for face-allocated fields
- And a **single, consistent EMF** on edges shared between levels **[Stone et. al. 2020]**

EMF correction for face sharers

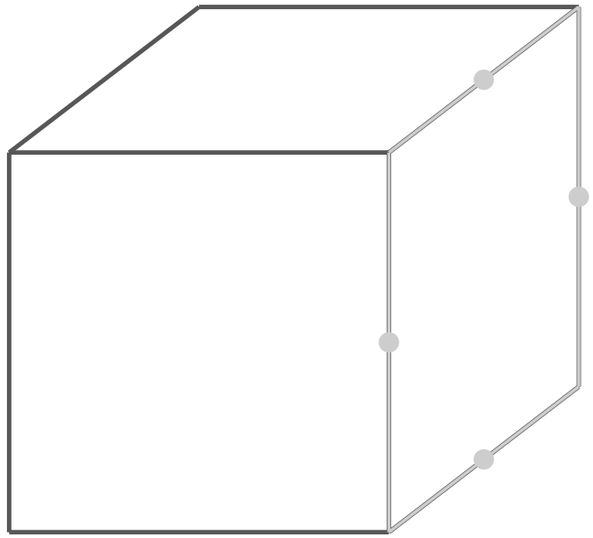


coarse

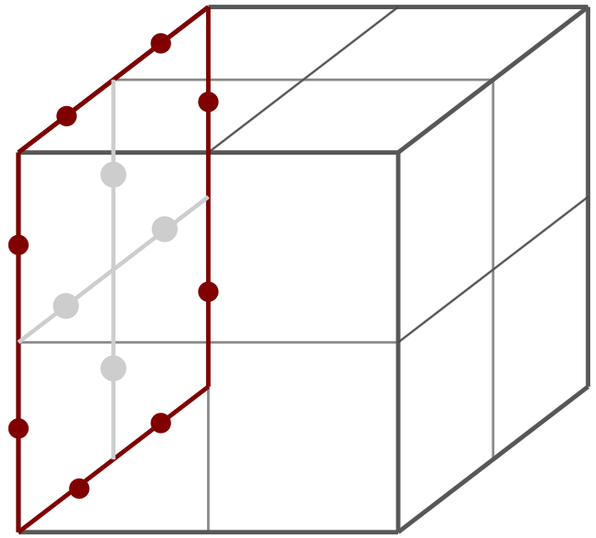


fine

EMF correction for face sharers



coarse

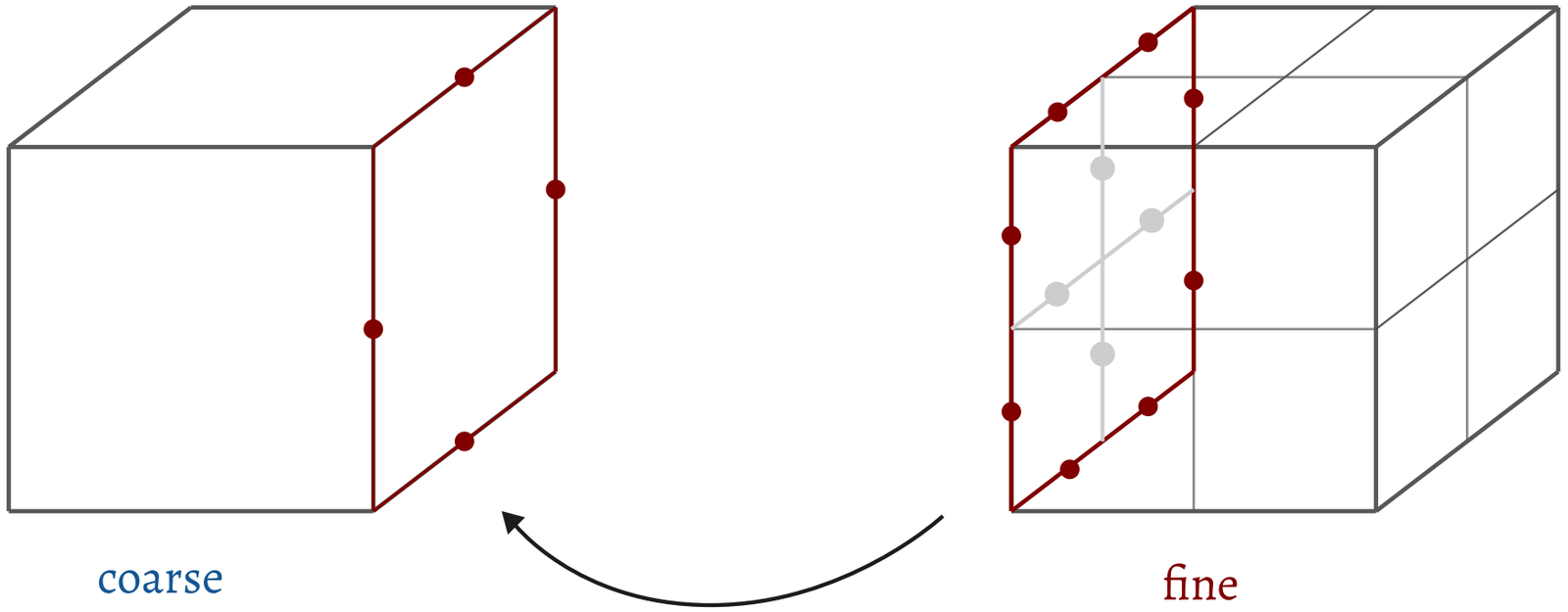


fine

- discard coarse values

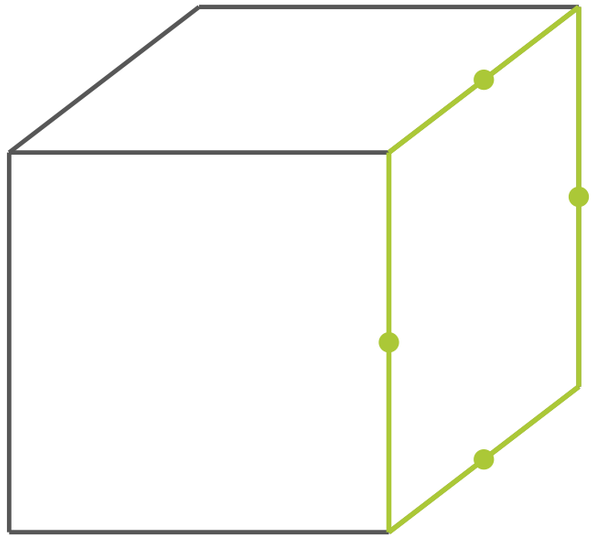
- discard non-overlapping edges

EMF correction for face sharers

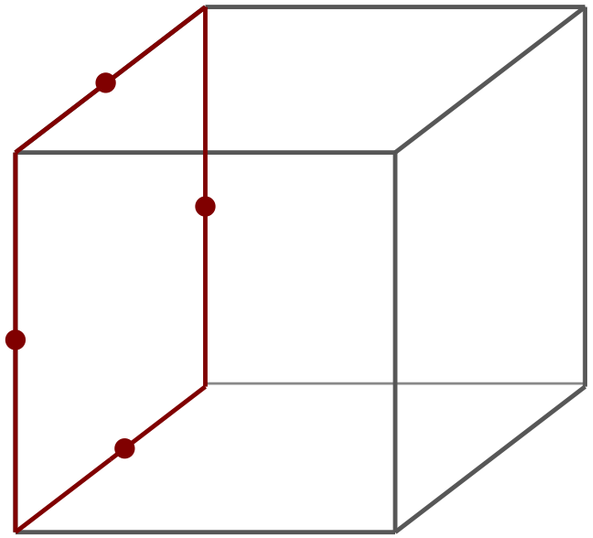


$$\varepsilon_{\text{coarse}}\,\Delta l_{\text{coarse}} = \sum \varepsilon_{\text{fine}}\,\Delta l_{\text{fine}}$$

EMF correction for face sharers

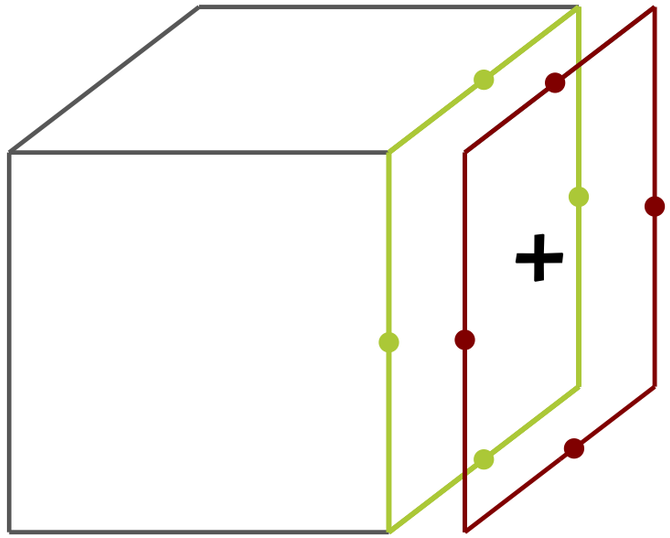


same level

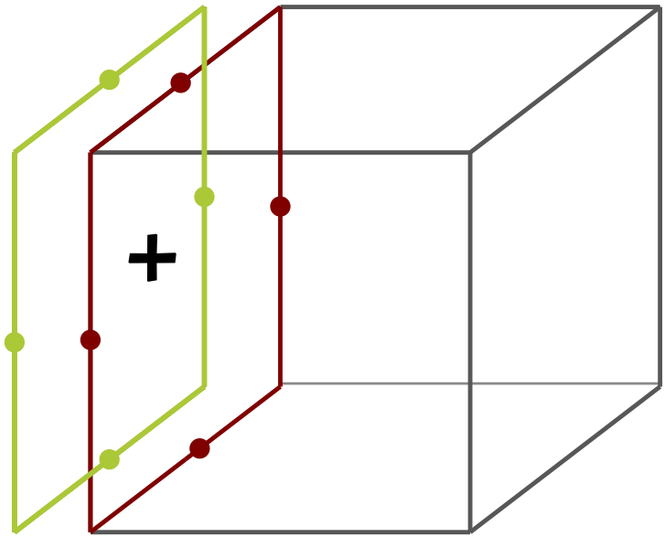


same level

EMF correction for face sharers



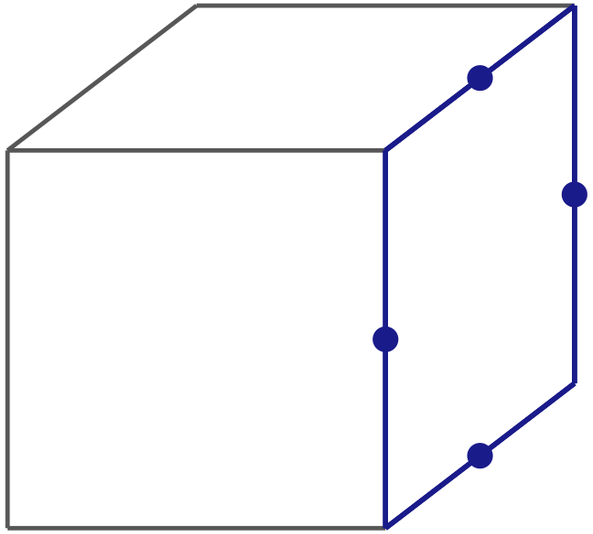
same level



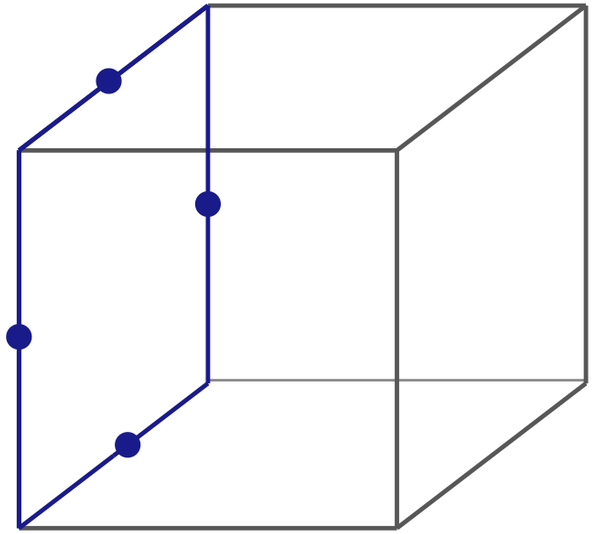
same level

- add the two shared-edge EMFs

EMF correction for face sharers



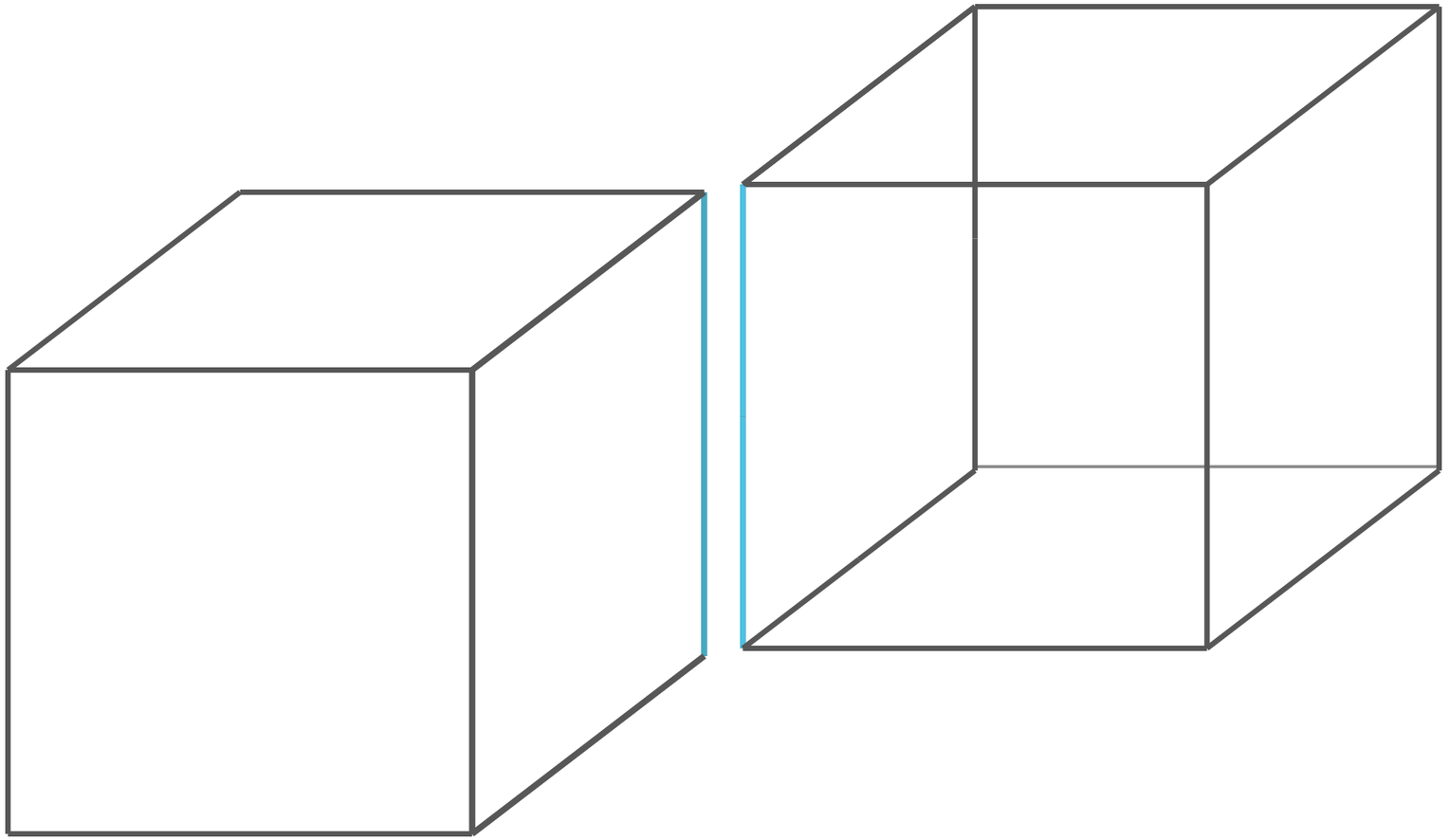
averaged value



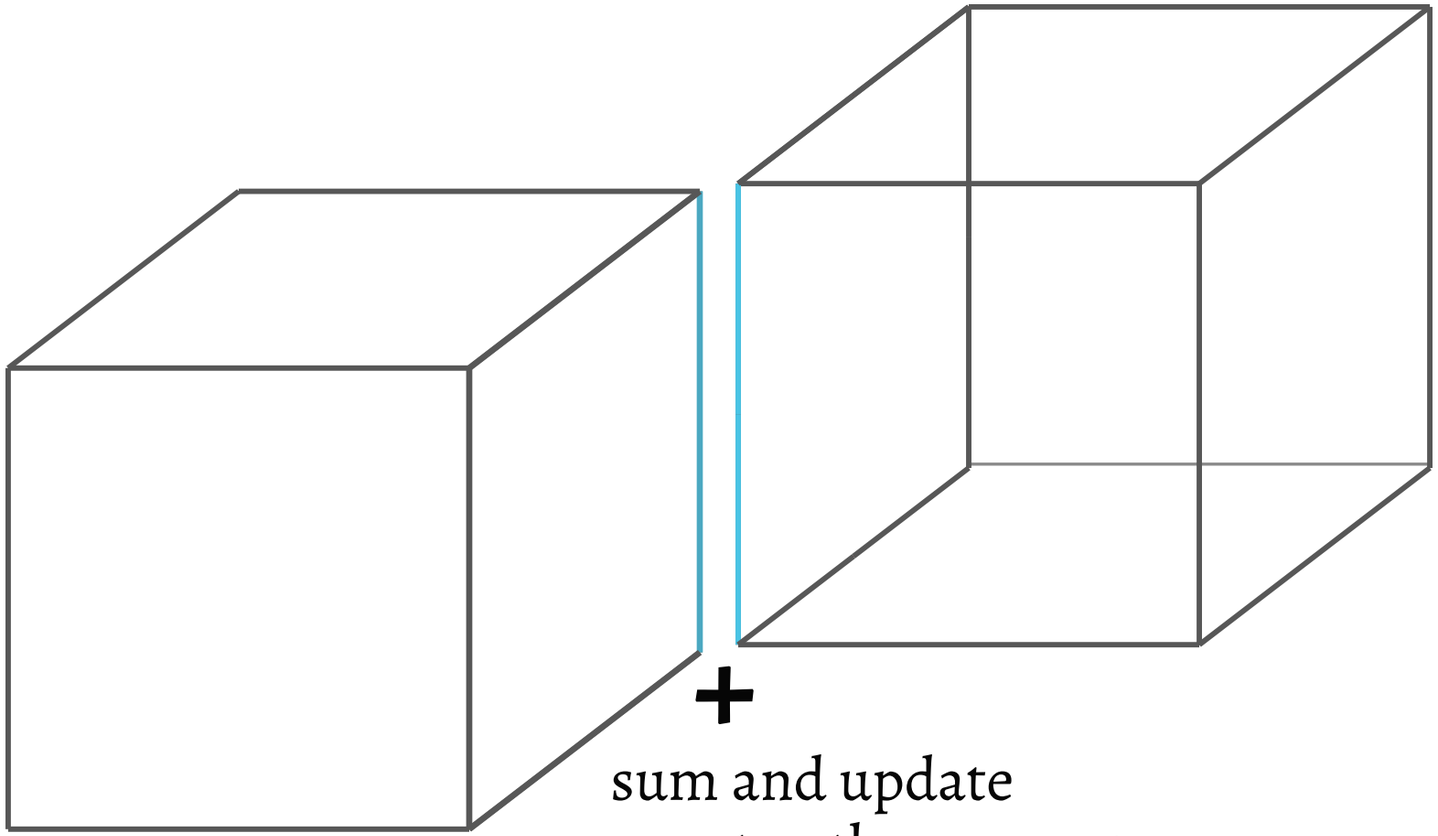
averaged value

$$\varepsilon_L = \varepsilon_R = \frac{1}{2}\left(\varepsilon_L + \varepsilon_R\right)$$

EMF corrections for edge sharers



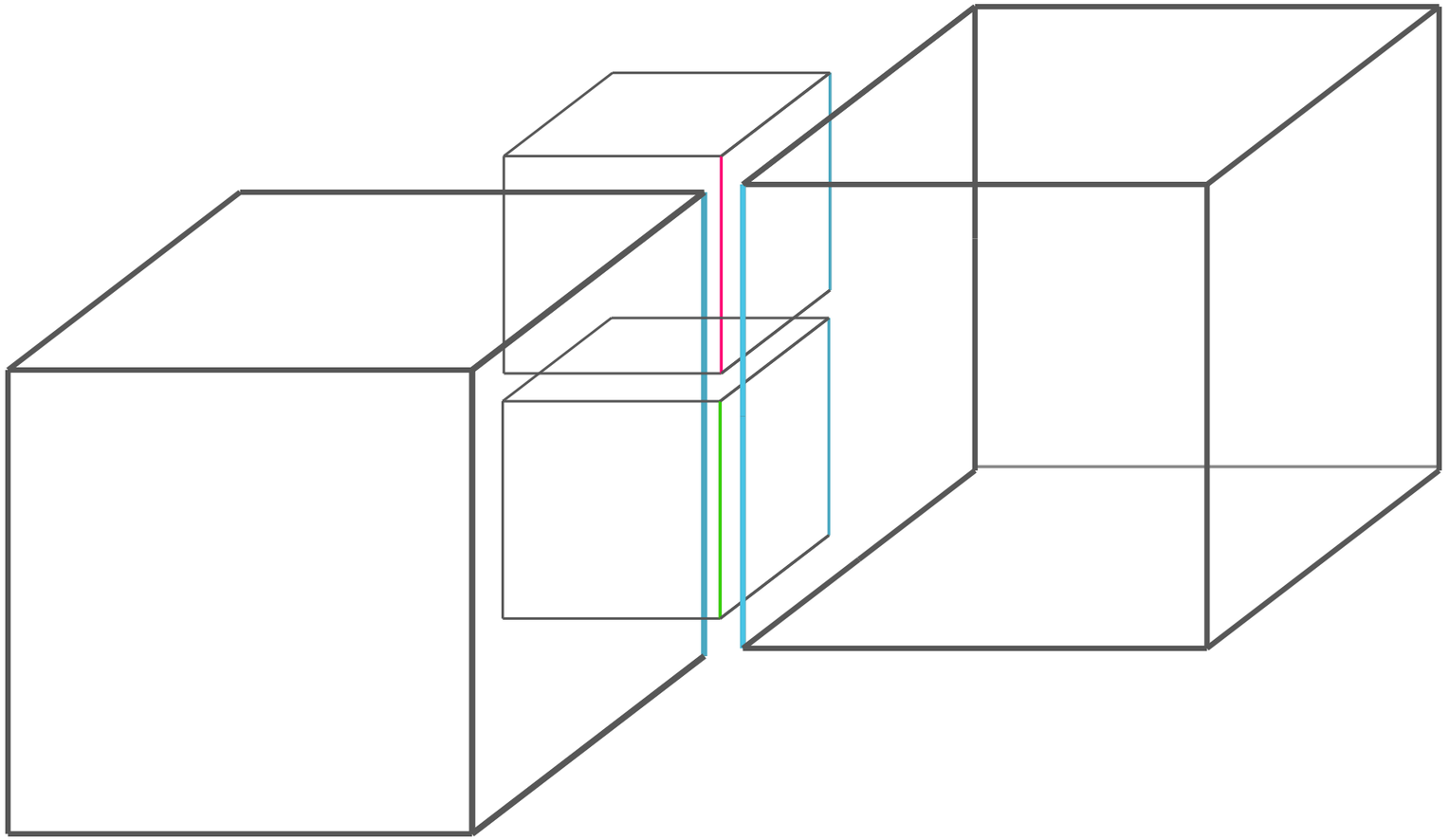
EMF corrections for edge sharers



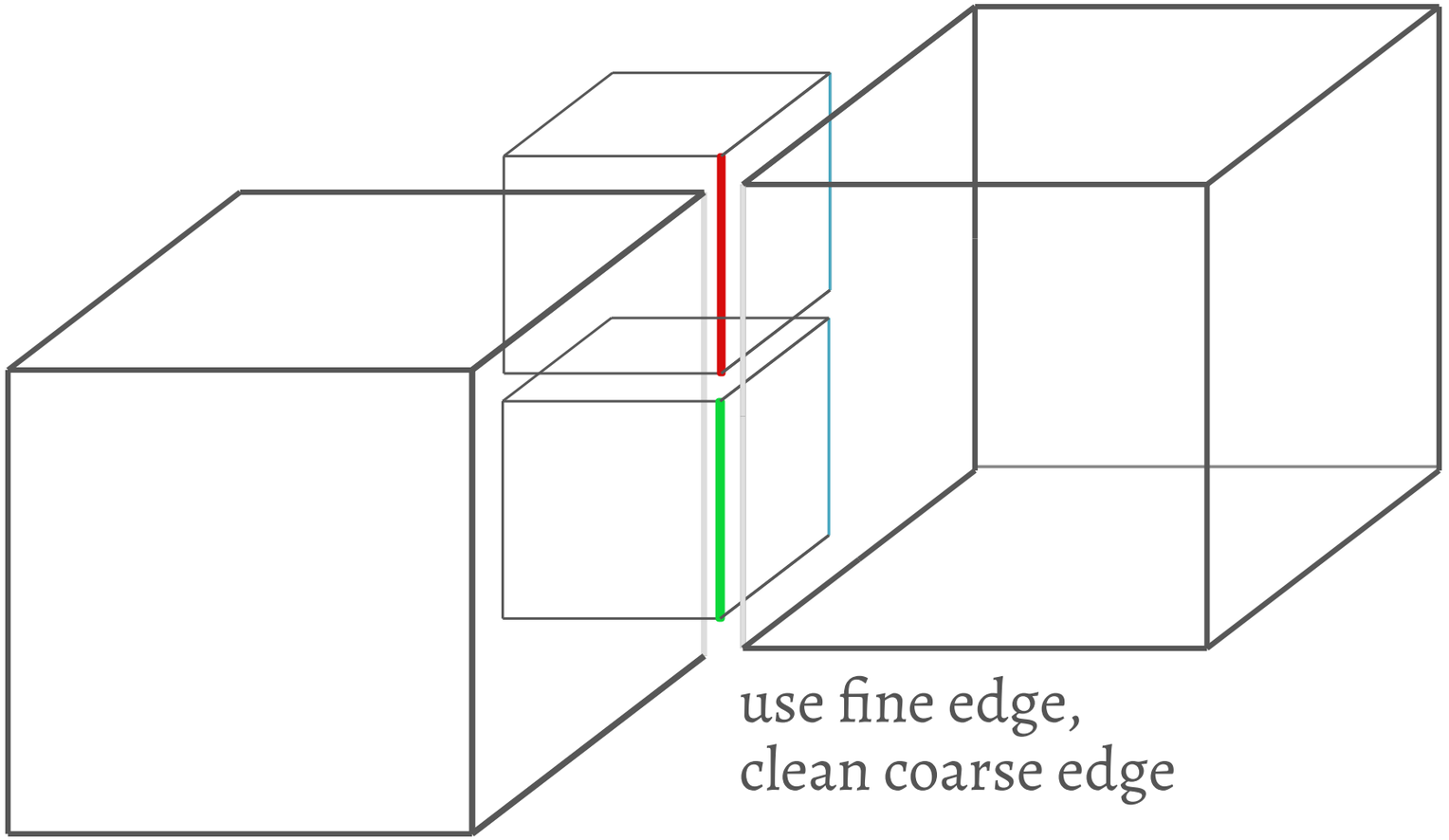
+

sum and update
counter, then average

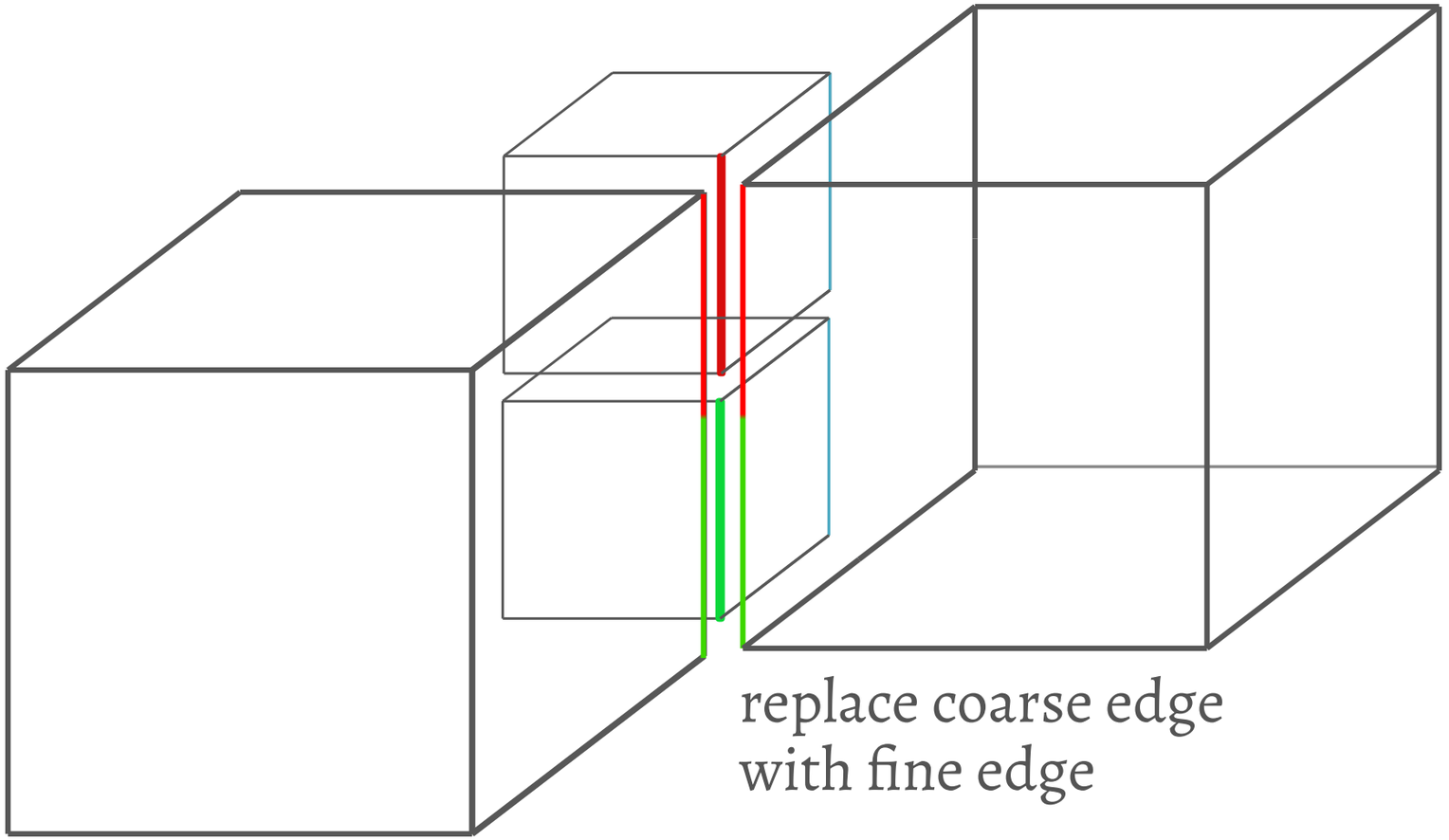
EMF corrections for edge sharers



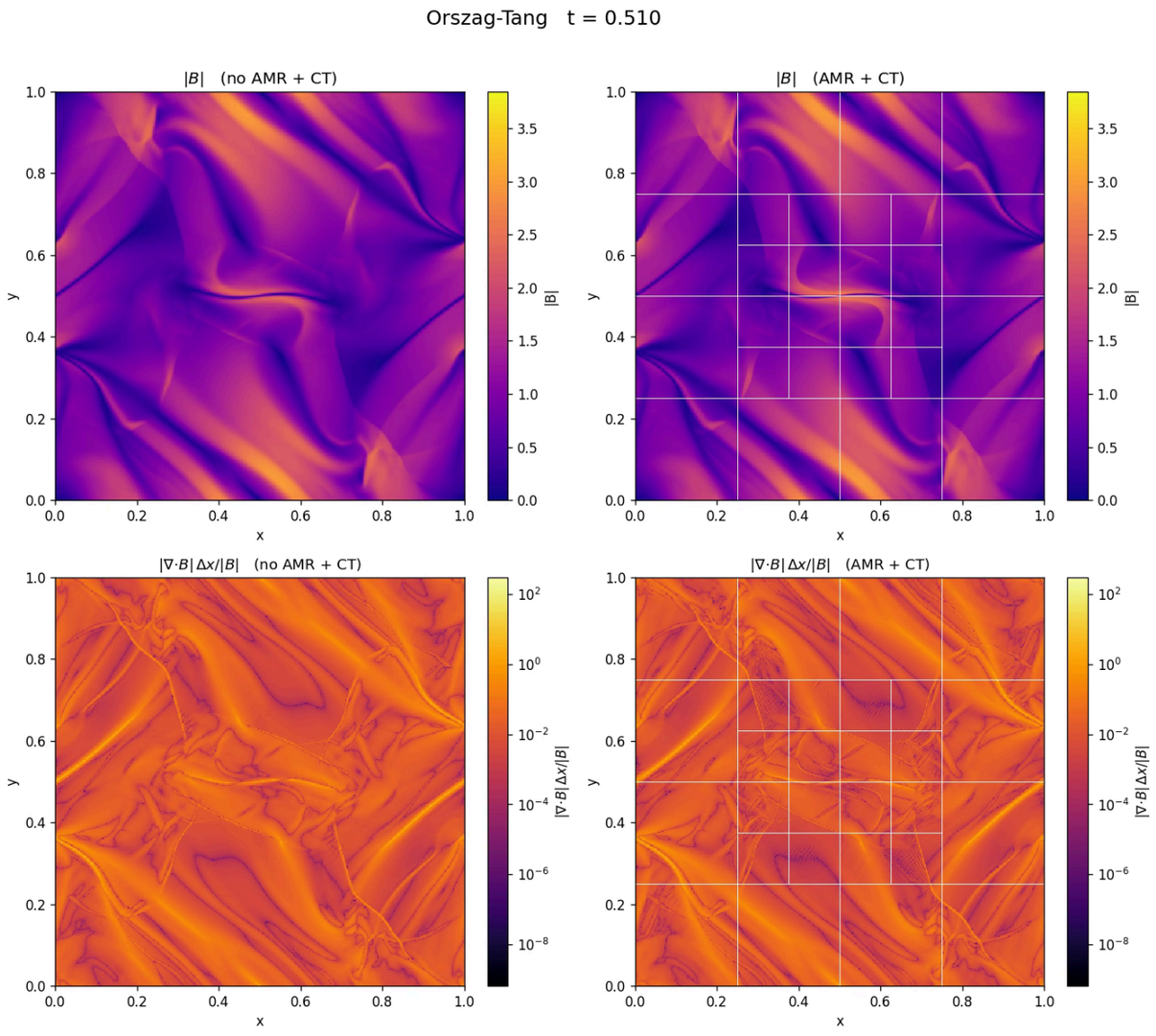
EMF corrections for edge sharers



EMF corrections for edge sharers



Tests for AMR + CT



Thanks

Maitraya Bhattacharyya · MPIA
mabhattacharyya@mpia.de

Questions?