State Redistribution For Embedded Boundary Methods

Ann Almgren
Lawrence Berkeley National Laboratory

September 7, 2024

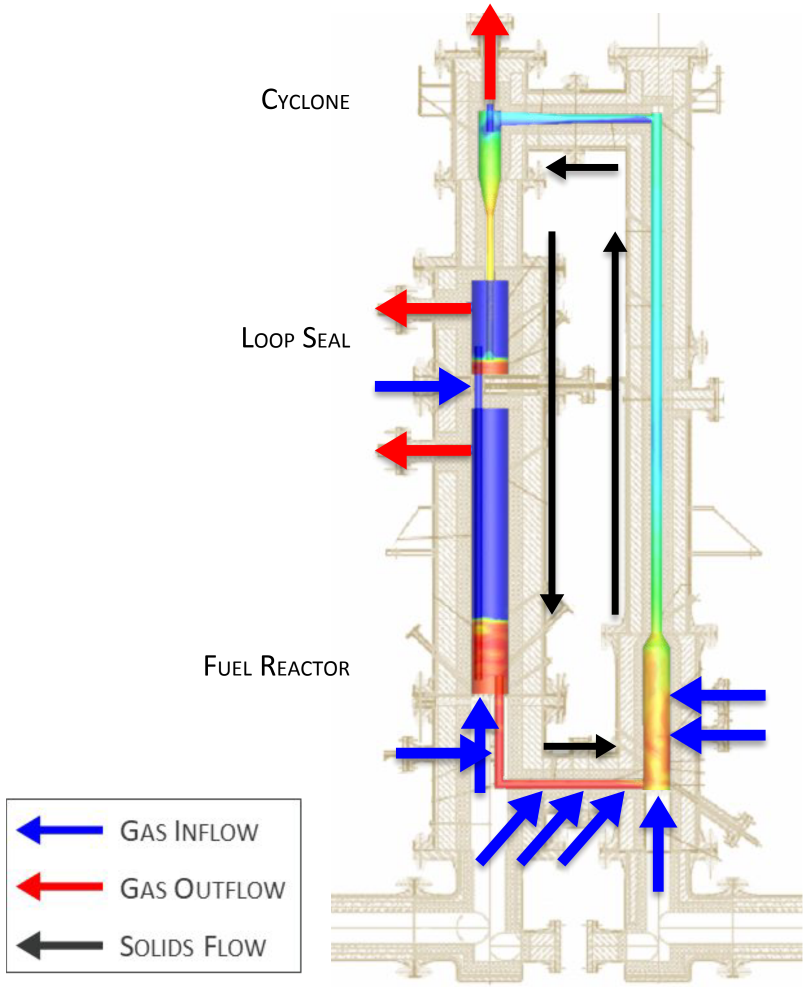
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

References

- **Berger, Giuliani**, A state redistribution algorithm for finite volume schemes on cut cell meshes, JCP, 2021.
- **Giuliani, Almgren, Bell, Berger, Henry de Frahan, Rangarajan,** A weighted state redistribution algorithm for embedded boundary grids, JCP, 2022.
- **Berger, Giuliani**, A new provably stable weighted state redistribution algorithm, 2024.
- **Barrio Sanchez, Almgren, Bell, Henry de Frahan, Zhang**, A new re-redistribution scheme for weighted state redistribution with adaptive mesh refinement, JCP, 2024.

Motivation



Chemical looping reactor (courtesy of NETL)

We often want to simulate fluid flow in a non-rectangular geometry.

In some cases our focus is on the interaction of the fluid and solid with models for how both evolve.

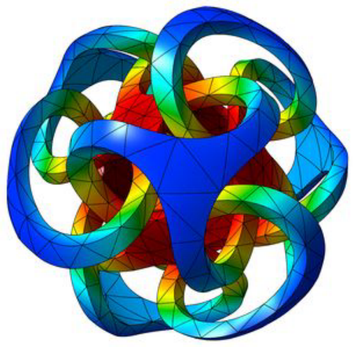
In some cases our focus is on the behavior of the fluid near the boundary.

But in some cases our focus is on what is happening in the regions of the fluid away from the walls – the purpose of the geometry here is to force the fluid in certain directions, and often specifically to generate mixing.

There are trade-offs in any design choice

In the chemical looping reactor, we solve fluid equations on a mesh, and also advect (and react) particles within the flow.

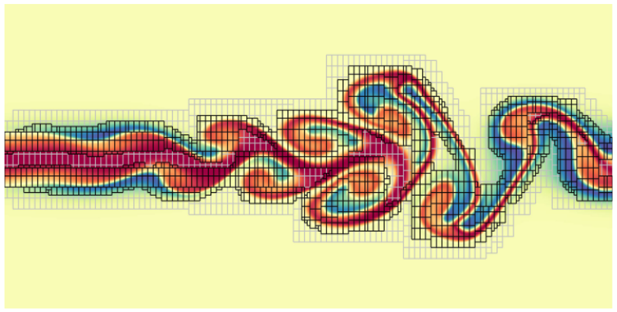
First design choice: structured vs unstructured mesh.



https://ceed.exascaleproject.org/vis/

Unstructured:

- Can fit the mesh to any geometry – much more generality
- No loss of accuracy at domain boundaries
- More "book-keeping" for connectivity information, etc
- Geometry generation becomes time-consuming

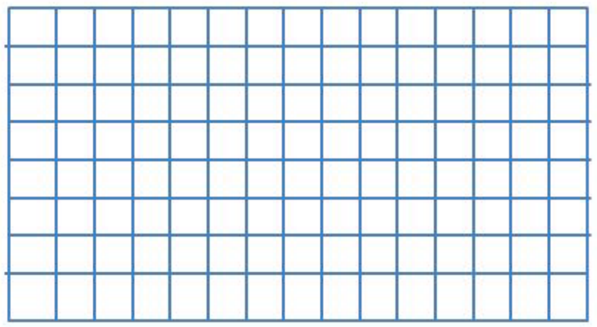


AMReX: Emmanuel Motheau

Structured:

- Easier to write discretizations
- Simple data access patterns
- Extra order of accuracy due to cancellation of error
- What about the complex geometry?

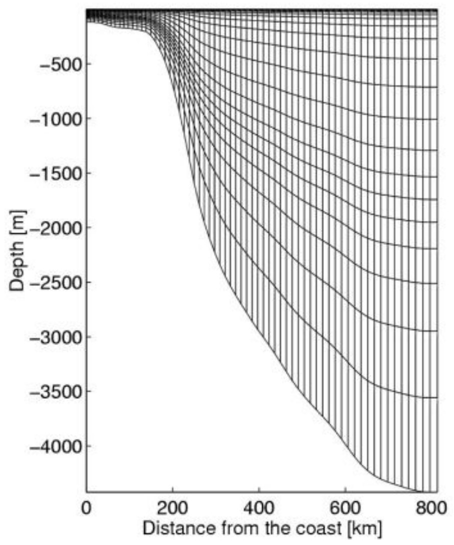
Logically rectangular != physically rectangular

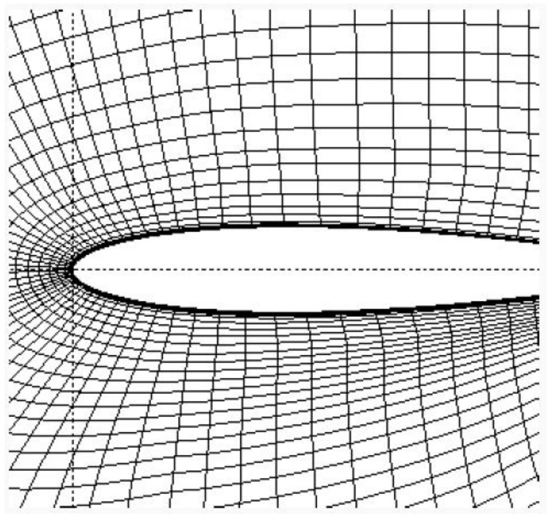


https://commons.wikimedia.org/wiki

Structured but body-fitted splits pros and cons of structured vs unstructured:

- Can fit (simple) non-rectangular boundaries while still having known connectivity
- Finer in certain regions (mesh refinement)
- Harder to maintain accuracy

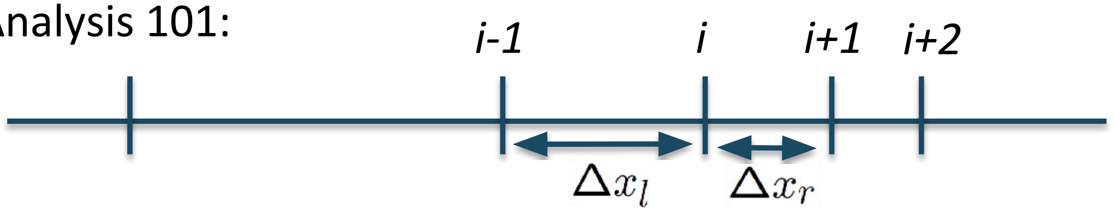




http://www.cfoo.co.za/simocean/modelsroms.php

Why Is Uniform Cell Size Good?

Numerical Analysis 101:



$$\phi_{i+1} = \phi(x_i) + \Delta x_r \phi_x + {}^1\!/\!_2(\Delta x_r)^2 \phi_{xx} + O(\Delta x_r{}^3)$$

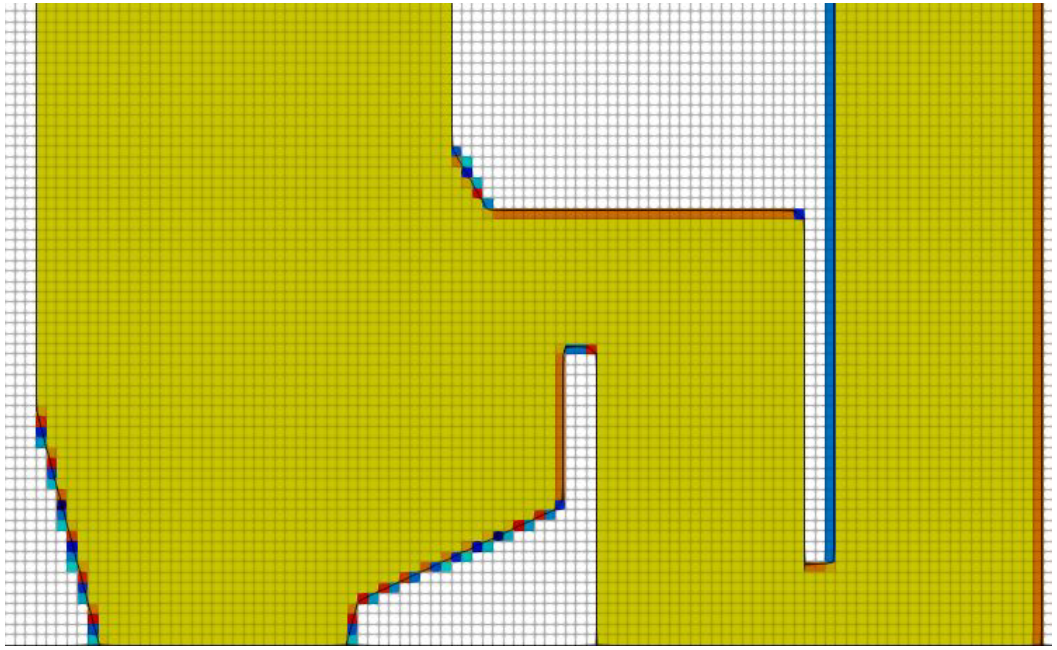
$$\phi_{i-1} = \phi(x_i) - \Delta x_l \phi_x + {}^1\!/\!_2(\Delta x_l)^2 \phi_{xx} + O(\Delta x_l{}^3)$$

We might use a **centered difference** as an approximation for a gradient at *i*,

$$\frac{\phi_{i+1} - \phi_{i-1}}{\Delta x_l + \Delta x_r} = \phi_x + {}^1\!/\!_2 (\Delta x_r - \Delta x_l) \phi_{xx} + O(\Delta x^2)$$

Note we only get second-order accuracy if we use constant cell spacing or sufficiently smooth mesh.

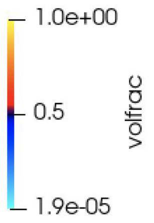
Representing geometry in a structured mesh



In most of the domain, the structured mesh looks completely uniform, which

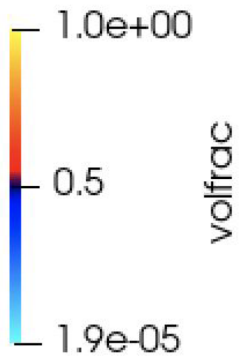
- makes it easier to write discretizations
- has simple data access patterns
- has extra order of accuracy due to cancellation of error

Here the colors represent the "volume fraction", i.e. the fraction of each cell that is "in the fluid"



We can represent complex geometries by intersecting the solid body with the uniform mesh to generate "cut cells"

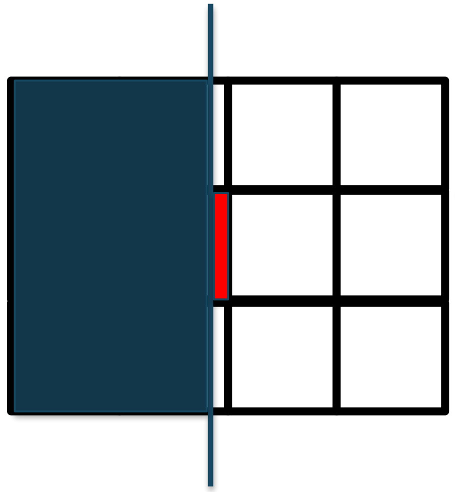
There is a problem though …

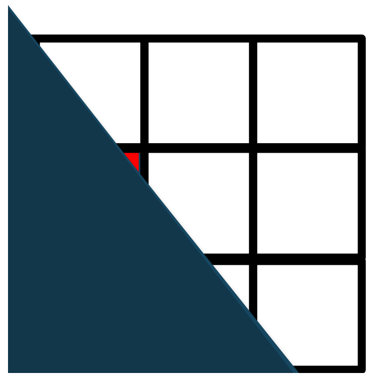


Note – this color bar from the previous picture tells us that some of the grid cells have volume fraction less than 0.00002

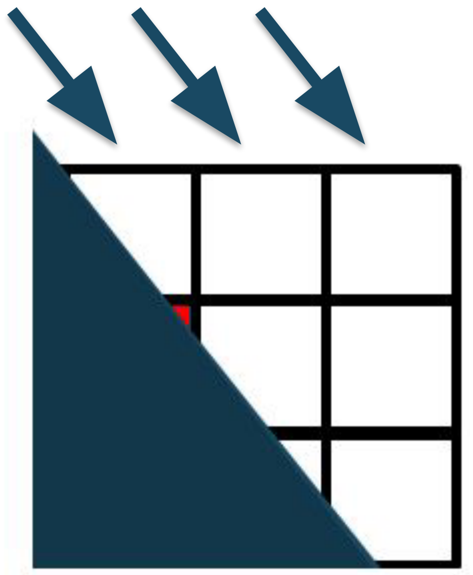
A cell with volume fraction of .00005 could have area fractions of 1 (in x) and 0.00005 (in y)

Or the fluid region could be a triangle with area fractions of 0.01 (in x) and 0.01 (in y)





We have to address the small cell problem



Consider

$$U_t + \nabla \cdot F = 0$$

discretized with an explicit finite volume scheme

If we write the divergence as an integral around the boundary of the cut cell and assume no flux through the wall, we see that

$$\frac{U^{n+1} - U^n}{\Delta t} = -\frac{F_{i,j+1}A_{i,j+1} + F_{i+1,j}A_{i+1,j}}{Vol_{i,j}}$$

Imagine we are transporting a scalar along the interface with uniform speed 1, and let Δx = 1 and Δt = 0.5 using the advective CFL condition based on the full cell size

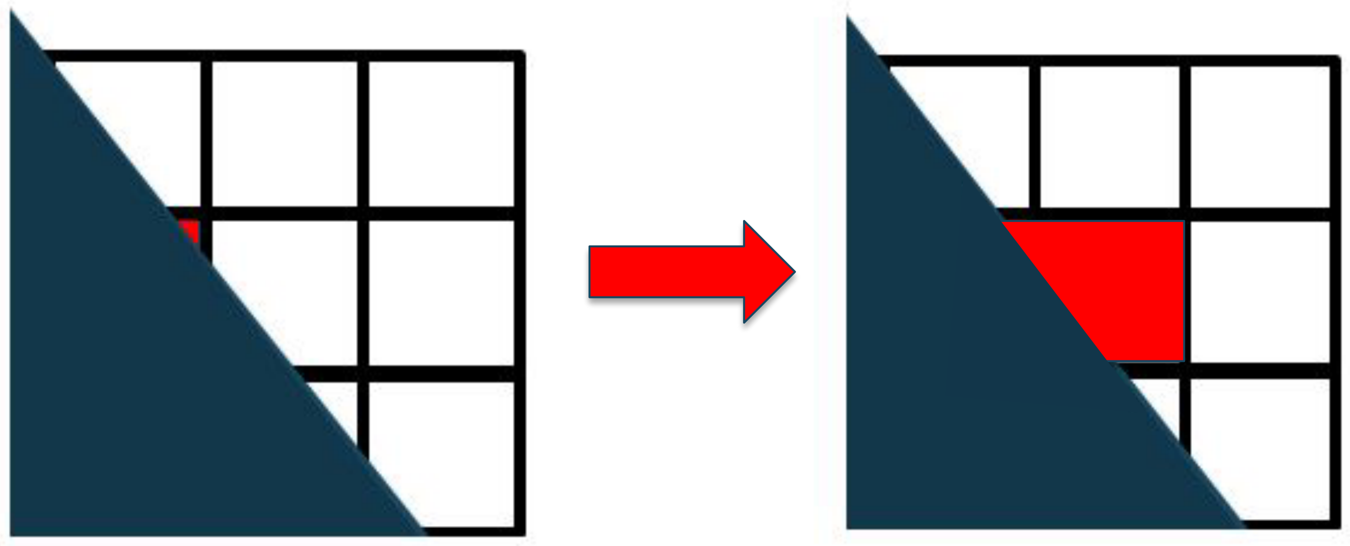
$$U^{n+1} = U^n - 100(F_{i,j+1} + F_{i+1,j})$$

Several solutions have been proposed

There is a long history of solutions to the small cell problem.

One popular approach is cell merging – in this case we simply “eliminate” small cells

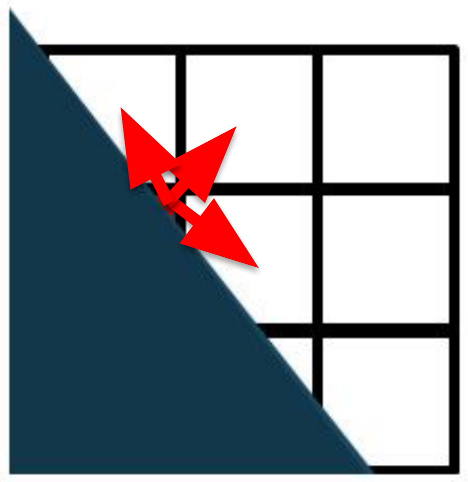
Cell merging algorithms tend to be based on heuristics that can be non-trivial to implement, and may not always work for complicated geometries



Another often-used approach is flux redistribution

With flux redistribution, we redefine how much of the update is allowed to go into the small cell. Flux redistribution satisfies:

- stability
- conservation



Flux redistribution also has several drawbacks:

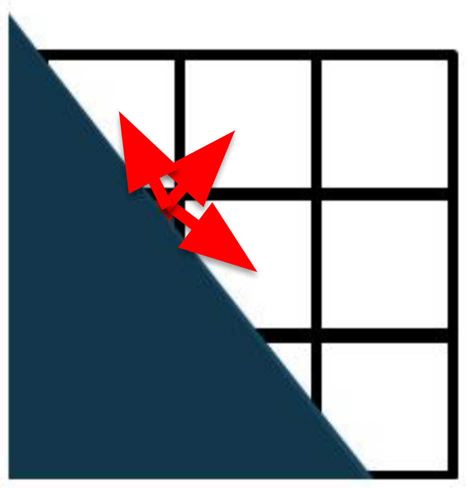
- Even for linear scalar advection, it can introduce new maxima and minima
- For more complicated flows, flux redistribution can cause the algorithm to fail

A new alternative is state redistribution

This was first introduced by Marsha Berger and Andrew Giuiliani in 2021

State redistribution also gives us

- stability
- conservation



But state redistribution often works where flux redistribution fails.

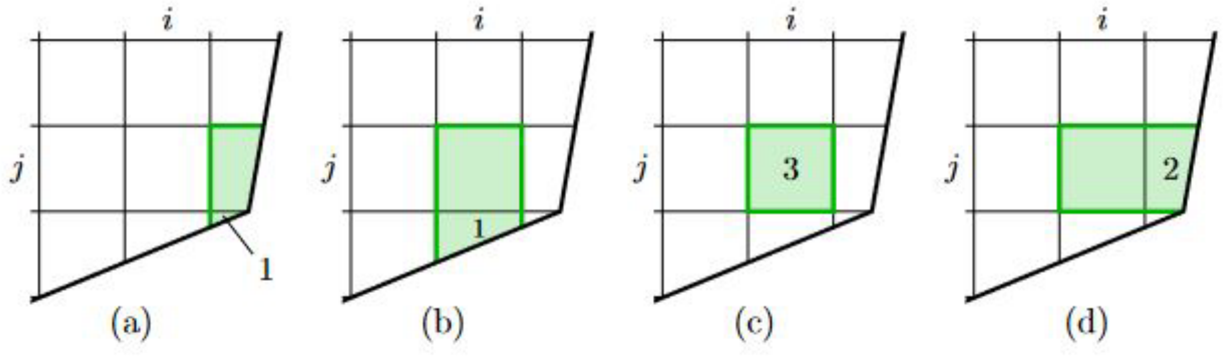
So what exactly is state redistribution?

- Like flux redistribution, state redistribution is a "post-processing" step, i.e. it occurs after we have created a conservative but unstable update to the solution.

- While flux redistribution operates on the update to the solution, state redistribution works on the solution itself

Imagine that we have updated the solution to define $\widehat{U}_{i,j}$ as an approximation to U^{n+1}

We first introduce the concept of "neighborhoods". Below we consider four cells and the neighborhoods of each.



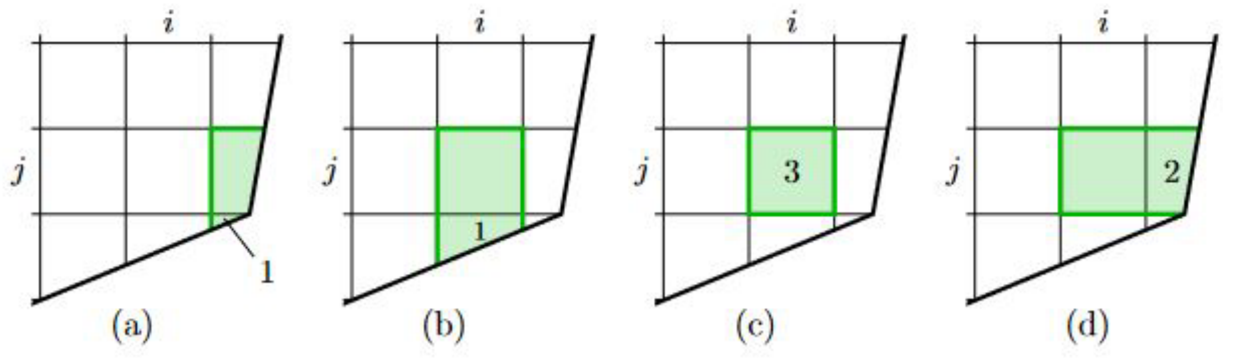
First we need to understand "neighborhoods"

In (a), we define the neighborhood of (i+1,j-1) to include itself and (i+1,j).

In (b), we define the neighborhood of (i,j-1) to include itself and (i,j).

In (c), we define the neighborhood of (i,j) to include only itself

In (d), we define the neighborhood of (i+1,j-1) to include (i,j).



We now ask how many neighborhoods (total!) each cell is in and call this $N_{i,j}$

(i+1,j-1) is in only one nbhd – itself

(i,j-1) is in only one nbhd – itself

(i+1,j) is in two nbhds – that of itself plus that of (i+1,j-1)

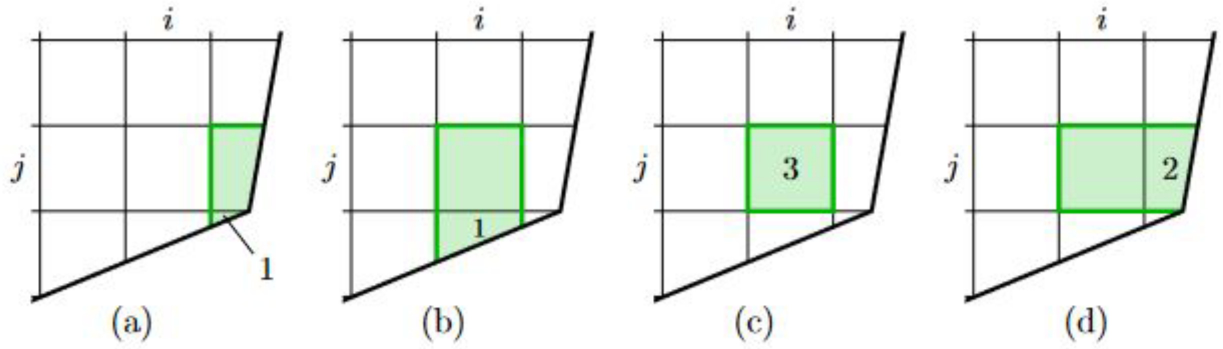
(i,j) is in three nbhds – that of itself plus that of (i,j-1) plus that of (i+1,j)

State Redistribution vs Cell Merging

This resembles cell merging in that we are creating neighborhoods in order to “merge” small cells with larger ones

This differs from cell merging in that each cell can be in more than one neighborhood – one can think of this as overlapping neighborhoods

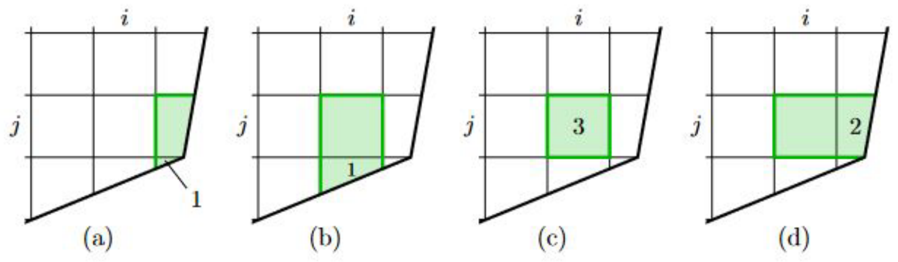
For stability we create neighborhoods such that the total (unweighted) volume of the nbhd is at least half of the volume of a regular cell. But note that the volume of a cell such as (i,j) can “count” towards the volume of more than one neighborhood.



We create an intermediate solution

To apply state redistribution, we first compute

$$\widehat{Q}_{i,j} = \frac{1}{\widehat{V}_{i,j}} \sum_{(r,s)\in M_{i,j}} \frac{V_{r,s}}{N_{r,s}} \widehat{U}_{r,s}.$$



Here $\widehat{U}_{i,j}$ is the (possibly unstable) solution that we got after applying our standard update formula using divergence of fluxes.

Here $(r, s) \in M_{i,j}$ is the set of (r,s) that are in the neighborhood of (i,j)

$V_{r,s}$ is the volume fraction of cell (r,s) ;

$\widehat{V}_{i,j} = \sum_{(r,s)\in M_{i,j}} \frac{V_{r,s}}{N_{r,s}}$ is the weighted volume fraction of nbhd (i,j)

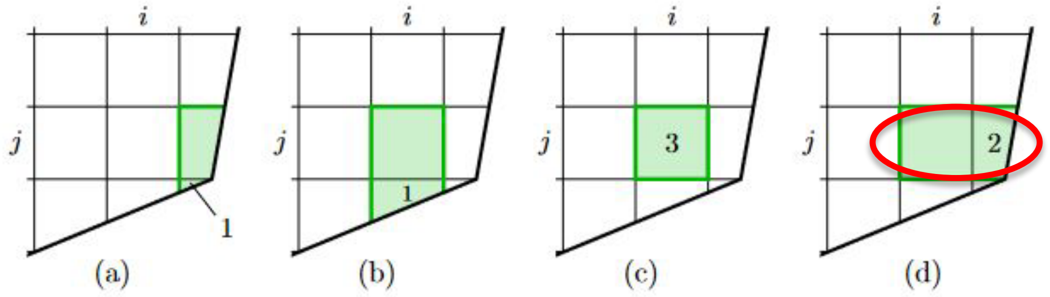
$N_{r,s}$ is the number of neighborhoods that cell (r,s) is in

U.S. DEPARTMENT OF ENERGY | Office of Science

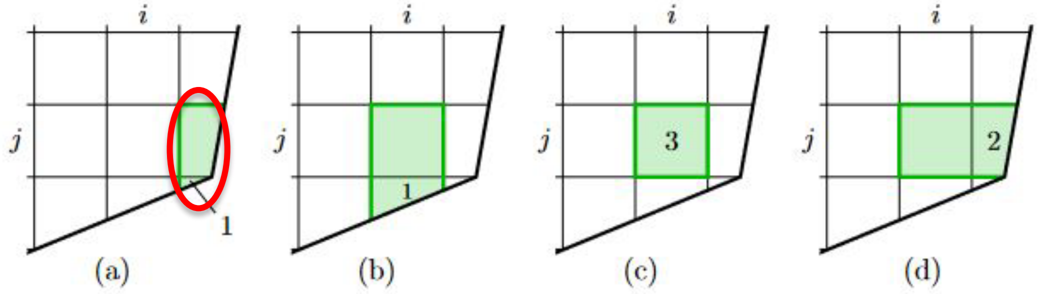
BERKELEY LAB

For example,

So, for example $$\widehat{Q}_{i+1,j} = \frac{1}{\widehat{V}_{i+1,j}}(\frac{1}{3}V_{i,j}\widehat{U}_{i,j} + \frac{1}{2}\widehat{U}_{i+1,j})$$

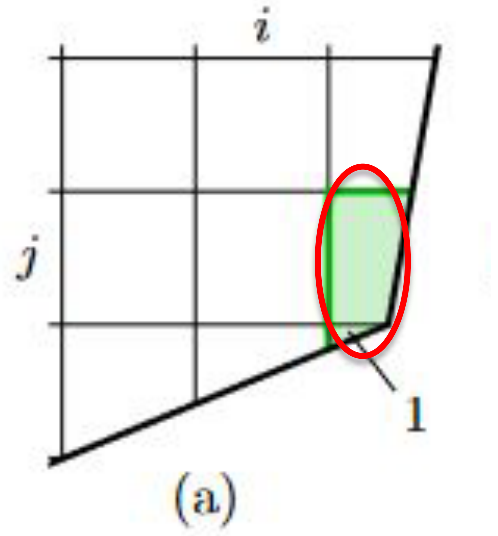


$$\widehat{Q}_{i+1,j-1} = \frac{1}{\widehat{V}_{i+1,j-1}}(V_{i+1,j-1}\widehat{U}_{i+1,j-1} + \frac{1}{2}V_{i+1,j}\widehat{U}_{i+1,j})$$



Understanding $\widehat{Q}$

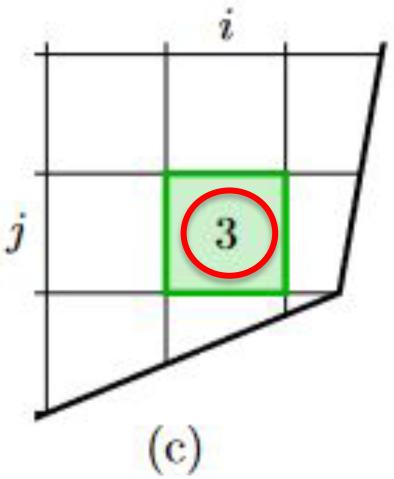
$$\widehat{Q}_{i+1,j-1} = \frac{1}{\widehat{V}_{i+1,j-1}}\left(V_{i+1,j-1}\widehat{U}_{i+1,j-1} + \frac{1}{2}V_{i+1,j}\widehat{U}_{i+1,j}\right)$$



In the limit where the volume of (i+1,j-1) is very small relative to that of (i+1,j), we see that this formula essentially sets the value of $\widehat{Q}$ in the small cell (i+1,j-1) equal to the value in the larger cell (i+1,j)

Understanding $\widehat{Q}$

$$\widehat{Q}_{i,j} = \frac{1}{\widehat{V}_{i,j}} \sum_{(r,s)\in M_{i,j}} \frac{V_{r,s}}{N_{r,s}} \widehat{U}_{r,s}.$$



For full cells such as (i,j), there is only one cell in the neighborhood of (i,j) and that is cell (i,j) itself, so for full cells,

$$\widehat{Q}_{i,j} = \widehat{U}_{i,j}$$

Once we have the solution in neighborhoods…

Once we have $\widehat{Q}$, a single value for each neighborhood associate with cell (i,j), we can construct slopes within each neighborhood, and use those to define a bilinear function that gives us the value of the intermediate solution at each centroid within the neighborhood.

We construct the slopes using a least squares approach applied to the $\widehat{Q}$ field of all nearby neighborhoods.

We call this $\widehat{q}_{i,j}(x,y)$ where (x,y) are assumed to lie in the neighborhood of (i,j)
Now to reconstruct the final solution, we define

$$U_{i,j}^{n+1} = \frac{1}{N_{i,j}} \sum_{(r,s)\in W_{i,j}} \widehat{q}_{r,s}(x_{i,j}, y_{i,j})$$

where $W_{i,j}$ is the set of neighborhoods that include cell (i,j) *

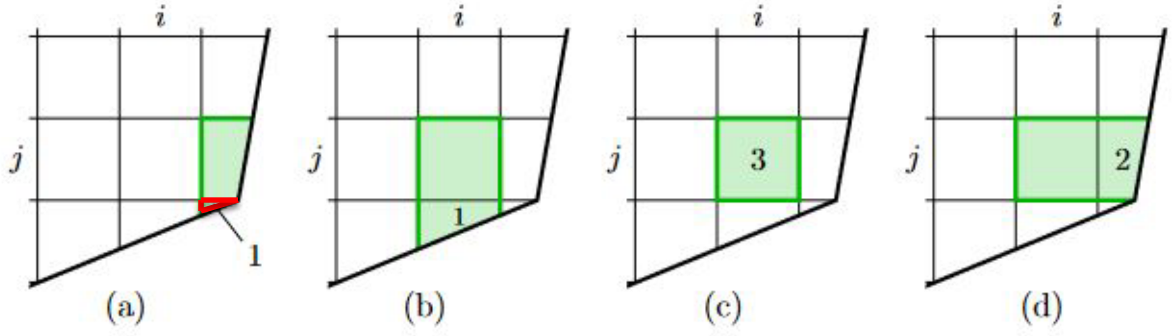
$(x_{i,j}, y_{i,j})$ is the location of the centroid of cell (i,j)

*not the cells in the nbhd of (i,j)

We can define the final solution…

To update cell (i+1,j-1) below, we note that it was only in its own neighborhood, so this becomes simply

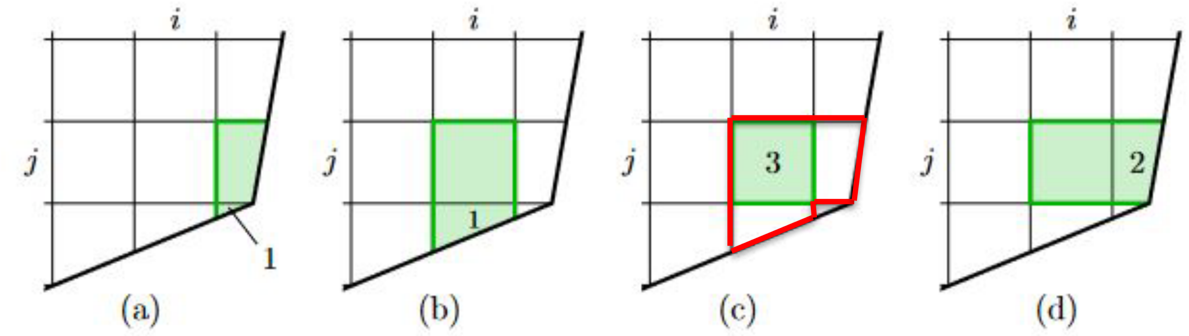
$$U^{n+1}_{i+1,j-1} = \widehat{Q}_{i+1,j-1} = \frac{1}{\widehat{V}_{i+1,j-1}}\left(V_{i+1,j-1}\widehat{U}_{i+1,j-1} + \frac{1}{2}V_{i+1,j}\widehat{U}_{i+1,j}\right)$$



Another example …

To update cell (i,j) below, we must sum over all the neighborhoods cell (i,j) is in, i.e.

$$U_{i,j}^{n+1} = \frac{1}{3}(\hat{q}_{i,j-1}(x_{i,j}, y_{i,j}) + \hat{q}_{i+1,j}(x_{i,j}, y_{i,j}) + \hat{q}_{i,j}(x_{i,j}, y_{i,j}))$$



Another way to look at it

Another way is to express the algorithm is in matrix form.

Once we know what neighborhoods we want to use, we can define matrix A by

$$\widehat{V} = AV$$

Then

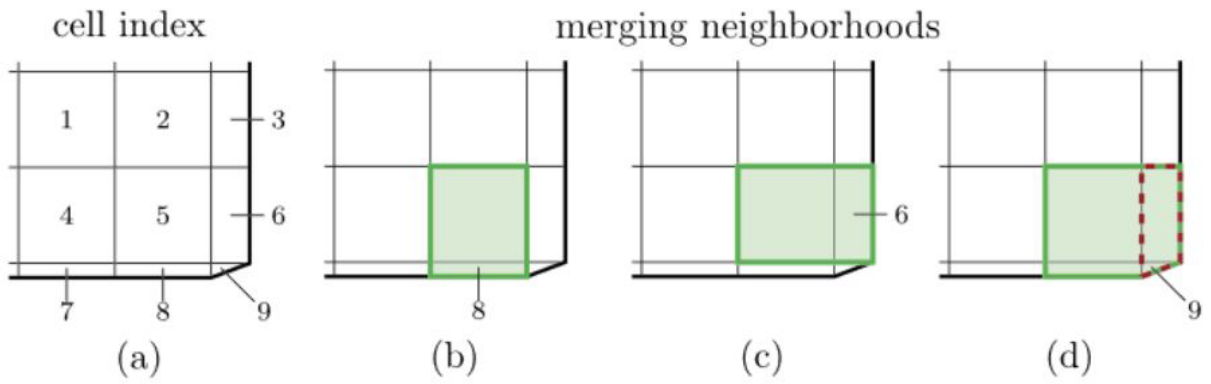
$$\widehat{Q} = \mathrm{Diag}(\widehat{V})^{-1} A\ \mathrm{Diag}(V)\widehat{U}$$

And if we set slopes = 0,

$$U^{n+1} = A^T \widehat{Q}$$

Matrix form

We consider a new problem with “central merging” in addition to “normal merging”



Then the matrix can be written

$$A^{\text{orig}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & 0 & \frac{1}{2} & 1 \end{bmatrix}$$

Matrix form helps us generalize SRD

As long as the sum along each column of A is 1, state redistribution is guaranteed to be conservative.

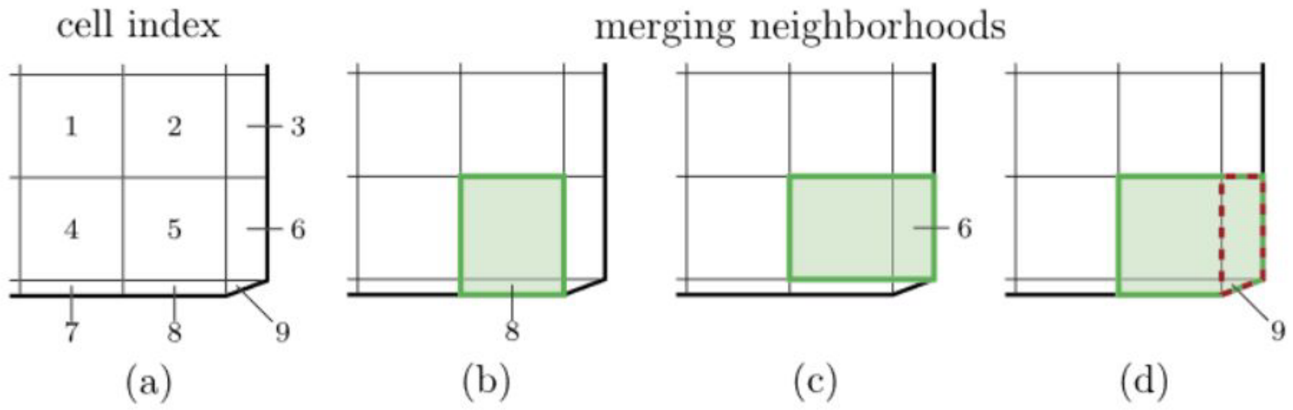
As long as we maintain conservation, we can think about using different weightings to improve the algorithm.

We define this generalized approach as “weighted state redistribution,” which is introduced in paper by Guiliani, Almgren, Bell, Berger, de Frahan and Rangarajan in 2022.

Weighted state redistribution

In the original implementation of SRD, a cell is defined as "needing merging" if it's volume fraction is < 0.5. This results in a sharp change in behavior between a cell with volume fraction slightly greater than 0.5 to slightly less than 0.5

An alternative is to only merge just enough to create neighborhood volumes slightly greater than 0.5 -- we call this "weighted state redistribution"



Matrix form gives us an easy way to express this

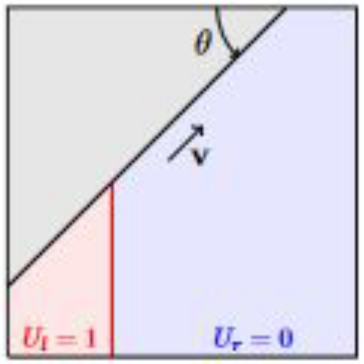
$$A^{\text{orig}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & 0 & \frac{1}{2} & 1 \end{bmatrix}
\qquad
A^{\text{wght}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \alpha_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\beta_3}{2} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha_4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \alpha_5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\beta_6}{4} & \alpha_6 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\beta_7}{2} & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\beta_8}{4} & 0 & 0 & \alpha_8 & 0 \\ 0 & 0 & 0 & 0 & \frac{\beta_9}{4} & \frac{\beta_9}{2} & 0 & \frac{\beta_9}{2} & 1 \end{bmatrix}.$$

$$\beta_{i,j,k} = (V_{target} - V_{i,j,k}) \ / \sum_{(r,s,t)\in M^-_{i,j,k}} V_{r,s,t}$$

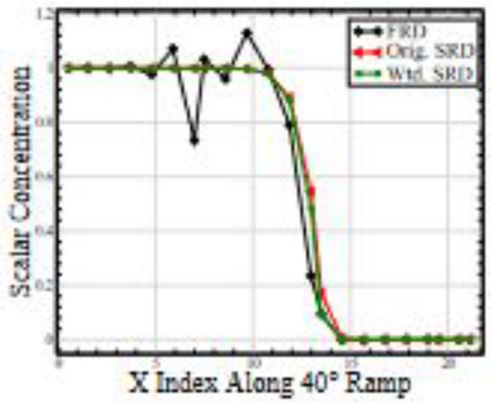
$$\alpha_{i,j,k} = 1 - \frac{1}{N_{i,j,k}} \sum_{(r,s,t)\in W^-_{i,j,k}} \beta_{r,s,t}$$

The columns of the weighted matrix still sum to 1 conservation

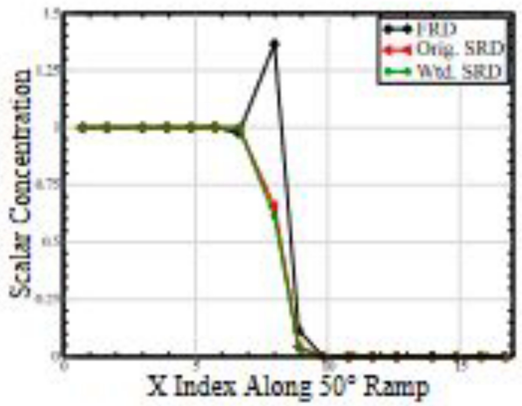
SRD eliminates over/under-shoot



(a) Set up for flow along a ramp.



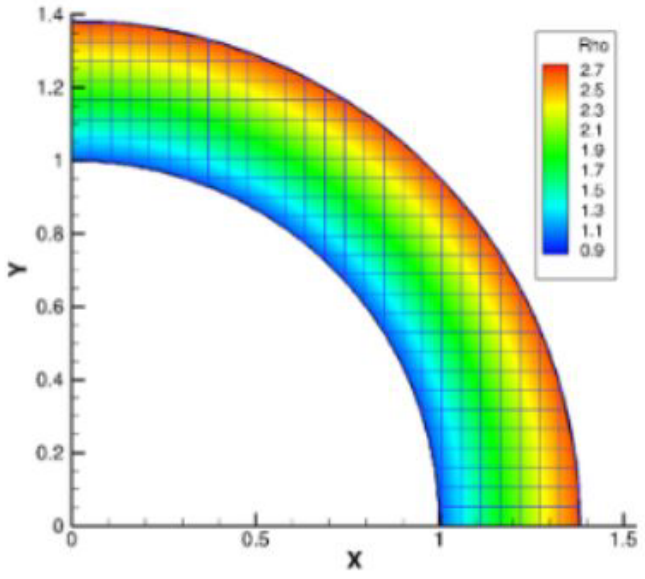
(b) Solution along $\theta = 40°$ ramp.

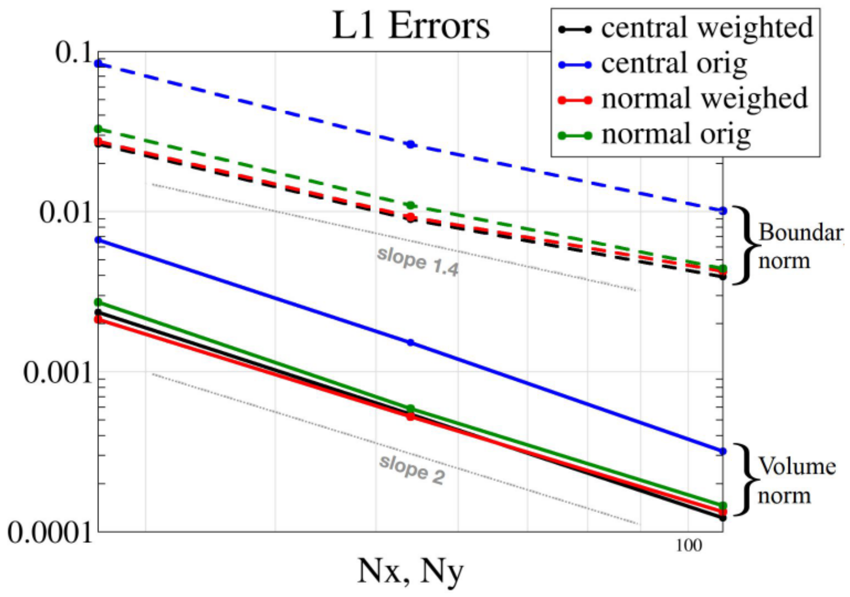


(c) Solution along $\theta = 50°$ ramp.

In this example of linear advection of a passive scalar along a slanted wall, flux redistribution (FRD) shows undershoots and overshoots, while state redistribution (SRD) does not.

Weighted SRD reduces error and is less sensitive to choice of neighborhood





Supersonic vortex test problem in 27x27 domain.

Generalizations include

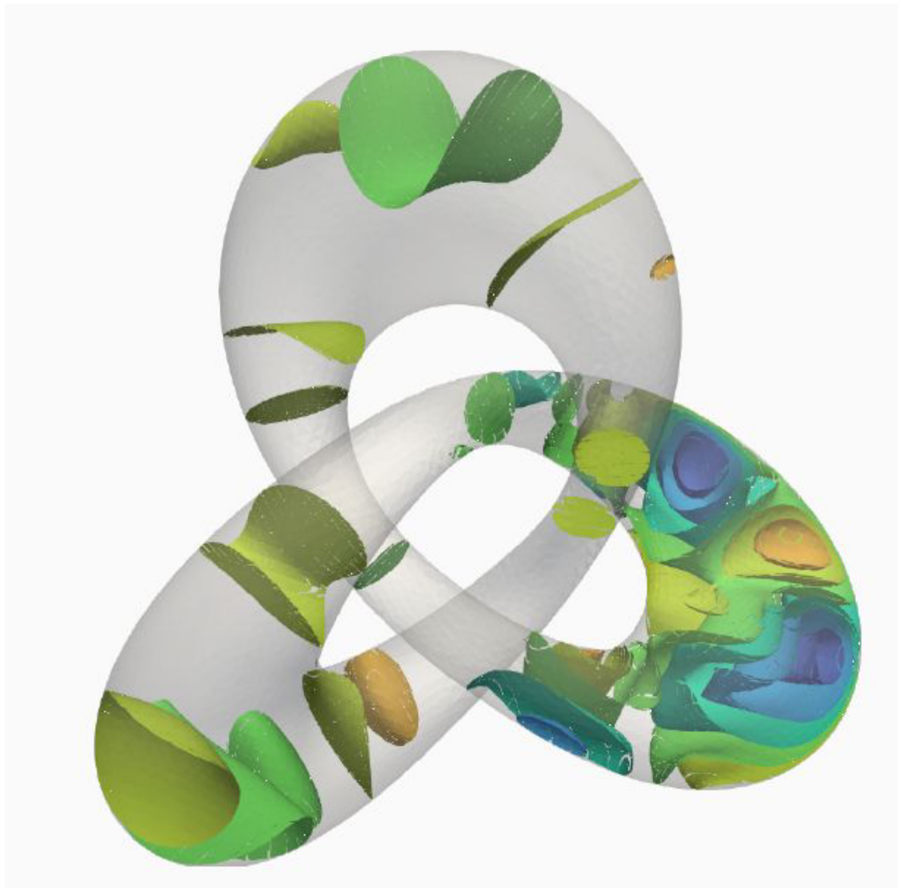
- Solutions updated with more than just advective terms – this works with diffusive updates and pretty much any time the solution needs “regularization”

- SRD was introduced for 2D compressible flow but has been extended to
  - 3D
  - compressible and incompressible,
  - Euler and Navier-Stokes,
  - reacting and non-reacting, and
  - multiphase flow (solid particles in gas)

- There are still more options to explore with respect to how we choose the weights (talk to Marsha!)

Acoustic scattering in a trefoil geometry

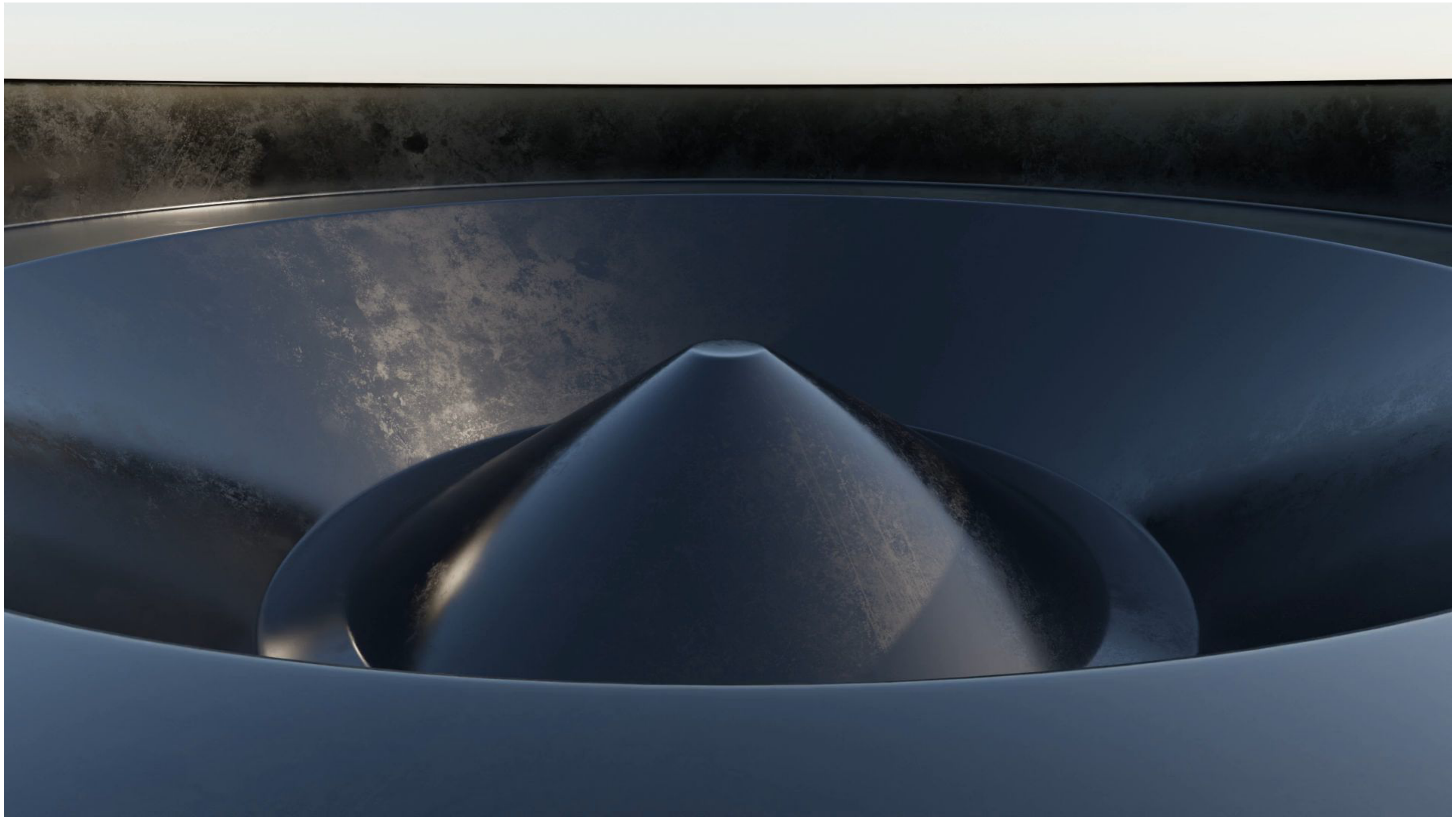
Isobars after acoustic scattering propagates through a trefoil cavity



- Third order finite volume
- RK3-SSP in time
- Monotone weighted SRD

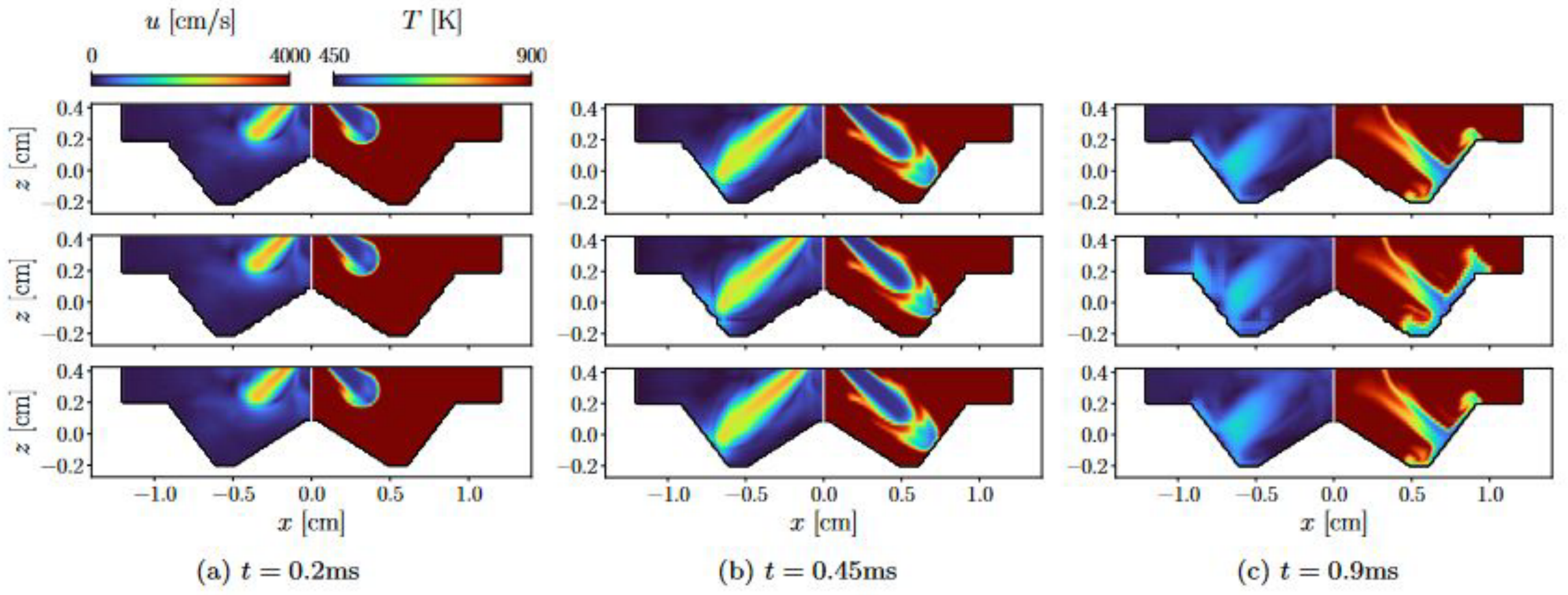
Simulation and visualization by Andrew Guiliani

3-D Low Mach Number Combustion Example



Animation courtesy of the Pele Combustion Team, which is supported under DOE ASCR's Exascale Computing Program. Simulation carried out using PeleLMeX on the Crusher machine at ORNL by L. Esclapez, N. Wimer, J. Rood, and M. Henry de Frahan. Visualization by N. Brunhart-Lupo.

We have generalized SRD to allow changes in refinement level at the EB interface

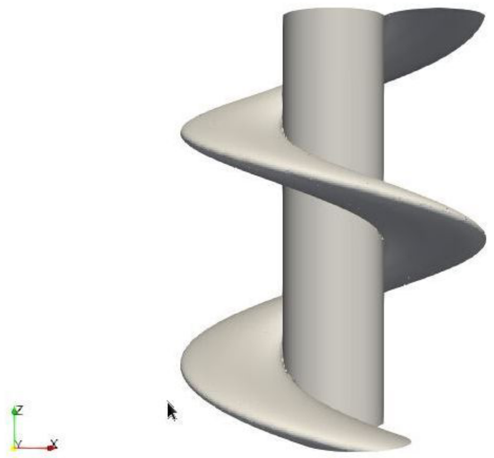


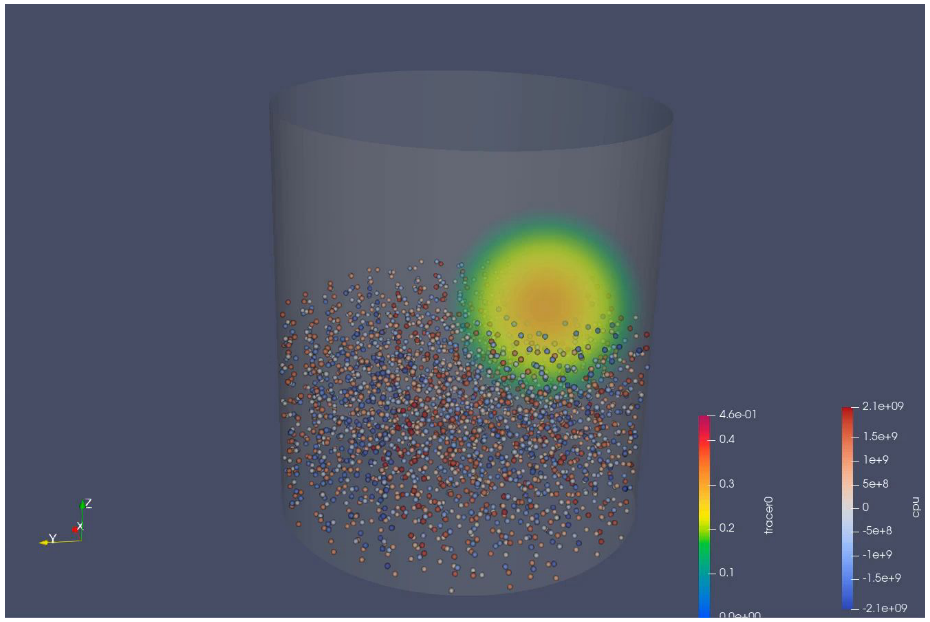
(a) $t = 0.2$ms (b) $t = 0.45$ms (c) $t = 0.9$ms

Top row: adaptively refined (to resolution of finest EB in bottom row, but 4x faster)
Middle row: everywhere coarser EB
Bottom row: everywhere finest EB

Work in progress includes moving EB

The machinery of SRD can be used to support a moving EB algorithm, in which cells become covered and uncovered – we can use the SRD merging concept to fill in cells which change volume and become uncovered due to the motion of the body



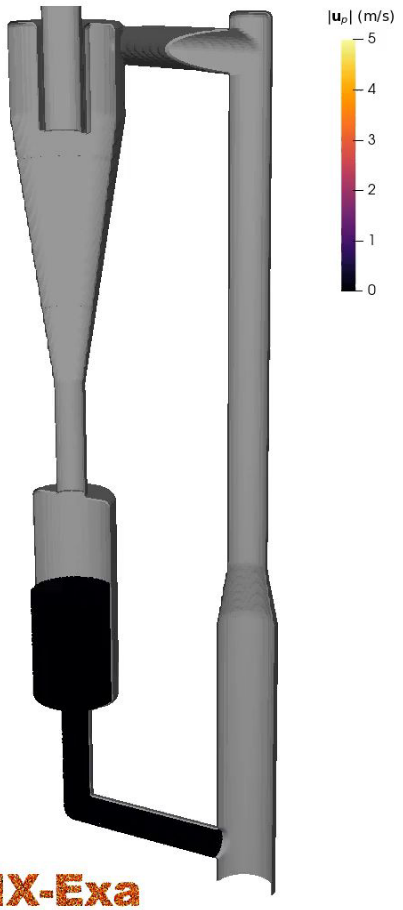


Thank you!

2D and 3D implementations of SRD are available at :

https://github.com/AMReX-Codes/AMReX-Hydro

- **Berger, Giuliani** , A state redistribution algorithm for finite volume schemes on cut cell meshes, JCP, 2021.
- **Giuliani, Almgren, Bell, Berger, Henry de Frahan, Rangarajan,** A weighted state redistribution algorithm for embedded boundary grids, JCP, 2022.
- **Berger, Giuliani** , A new provably stable weighted state redistribution algorithm, 2024.
- **Barrio Sanchez, Almgren, Bell, Henry de Frahan, Zhang**, A new re-redistribution scheme for weighted state redistribution with adaptive mesh refinement, JCP, 2024.



MFIX-Exa