# Efficient Algorithm for Local-Bound-Preserving Remapping in ALE Methods

Rao Garimella[1], Milan Kuchařík[2], and Mikhail Shashkov[3]

[1] Los Alamos National Laboratory, T-7, MS-284, Los Alamos, NM 87545, USA *rao@lanl.gov*
[2] Czech Technical University in Prague, Břehová 7, 115 19 Prague 1, Czech Republic *kucharik@karkulka.fjfi.cvut.cz*
[3] Los Alamos National Laboratory, T-7, MS-284, Los Alamos, NM 87545, USA *shashkov@lanl.gov*

## 1 Introduction

Conservative remapping is an essential part of the ALE (Arbitrary Lagrangian-Eulerian) method for fluid dynamics computations. This method tries to use advantages of both Lagrangian and Eulerian approach.

At first, several time steps of pure Lagrangian computation are used. The grid moves together with fluid, it may happen that the grid becomes distorted or tangled due to shear flow. Now comes the Eulerian part of the algorithm. We prepare a new rezoned grid and recompute (remap) the quantities from the distorted grid to the rezoned one.

We have several conditions, this remapping step must satisfy. It must be efficient to be usable in real computations. Total sum of the conservative quantities must be preserved – the algorithm must be conservative. We do not want to create new local extremes, we want it to be local-bound preserving. It must be stable and applicable to general unstructured meshes in 2D and 3D. I this article we introduce 3D algorithm, which satisfies these conditions. Similar procedure in 2D is described in [1].

## 2 Algorithm Description

### 2.1 Problem statement

Suppose, we have two grids: Lagrangian $C = \{c\}$ and rezoned $\tilde{C} = \{\tilde{c}\}$. The grids have the same topology. The rezoned grid is created from the original one just by small movement of the grid nodes. There exists some underlying function $g(\mathbf{r})$, $\mathbf{r} = (x, y, z)$ in the Lagrangian cells (for example $g = \rho$, $g = \rho\, u$, $g = \rho\, v$, $g = \rho\, w$, $g = \rho\, \left(\varepsilon + |\mathbf{U}|^2/2\right)$, where $\rho$ is mass density, $\mathbf{U} = (u, v, w)$ is a vector of velocities and $\varepsilon$ is the internal energy). We do not know the function itself, we know just the mean values

$$\bar{g}_c = \frac{\int_c g(\mathbf{r})\, dV}{V(c)} = \frac{m(c)}{V(c)}, \quad m(c) \equiv \int_c g(\mathbf{r})\, dV \tag{1}$$

in the grid cells. Total mass (momentum, energy) can be computed like

$$M \equiv \int_\Omega g(\mathbf{r})\, dV = \sum_{\forall c} \int_c g(\mathbf{r})\, dV = \sum_{\forall c} m(c)\,. \tag{2}$$

We want to compute new masses $m^*(\tilde{c})$ and corresponding mean values in the rezoned cells

$$\bar{\tilde{g}}^*_{\tilde{c}} = \frac{m^*(\tilde{c})}{V(\tilde{c})} \tag{3}$$

and we want them to be as close to the exact values as possible ($m^*(\tilde{c}) \approx m(\tilde{c}) = \int_{\tilde{c}} g(\mathbf{r})\, dV$). We also want not to create new local extremes

$$g_c^{\max} \geq \bar{\tilde{g}}^*_{\tilde{c}} \geq g_c^{min}\,, \quad g_c^{\max} = \max_{c_n \in \mathcal{C}(c)} g_{c_n}\,, \quad g_c^{\min} = \min_{c_n \in \mathcal{C}(c)} g_{c_n}\,, \tag{4}$$

where $\mathcal{C}(c) \subset C$ is neighborhood of cell $c$, and to be conservative (total mass must be the same)

$$\sum_{\forall c} m^*(\tilde{c}) = M\,.$$

If the underlying function is a linear function, we want our method to be exact

$$m^*(\tilde{c}) = m(\tilde{c}) = \int_{\tilde{c}} g(\mathbf{r})\, dV \quad \text{for} \quad g(\mathbf{r}) = a + b\,x + c\,y + d\,z\,.$$

### 2.2 Remapping Algorithm

We design our algorithm in three stages. In the first stage, we make a piecewise linear reconstruction of the underlying function on the original mesh. This can be done using different methods, with or without limiters. In the second stage, we integrate this reconstructed function to obtain means on the new grid. The most natural approach would be exact integration, but it needs computation of the intersections of the Lagrangian grid with the rezoned one. This intersection is very time consuming in 2D and almost unfeasible in 3D, so we use numerical quadrature - swept integration. It does not require finding these intersections so it is much faster. The problem is that it is approximate method and it may happen that the local extremes are violated, so we need also the third stage - repair - which ensures us this local-bound preservation.

### 2.3 Stage 1 – Piecewise Linear Reconstruction

We want to reconstruct the underlying function in the form

$$\rho_c(\mathbf{r}) = \rho_c(x,y,z) = \bar{\rho}_c + S_c^x\,(x - x_c) + S_c^y\,(y - y_c) + S_c^z\,(z - z_c)\,, \tag{5}$$

where

$$x_c = \frac{\int_c x\, dV}{V(c)}, \quad y_c = \frac{\int_c y\, dV}{V(c)}, \quad z_c = \frac{\int_c z\, dV}{V(c)} \tag{6}$$

are coordinates of the cell center and

$$V(c) = \int_c 1\, dV \tag{7}$$

is the volume of the cell.

For computation of slopes we use the limited form

$$S_c^x = \Phi_c\, S_c^{x\ \text{unlim}} \quad , S_c^y = \Phi_c\, S_c^{y\ \text{unlim}} \quad , S_c^z = \Phi_c\, S_c^{z\ \text{unlim}}\,, \tag{8}$$

where $S_c^{\{x,y,z\}\ \text{unlim}}$ are unlimited slopes and $\Phi_c$ is Barth-Jasperson limiter, which must be computed firstly.

**Unlimited Slopes** In 1D we can use just central difference as unlimited slope. To compute unlimited slopes in 2D we construct contour surrounding cell and use Green Theorem. In 3D this would require computing intersections of this neighborhood with the original grid, which would be too slow. So we must use another method.

Let's construct the functional

$$F(S_c^x, S_c^y, S_c^z) = \sum_{c_n \in \mathcal{C}(c)} \left( \bar{\rho}_{c_n} - \frac{\int_{c_n} \rho_c(x,y,z)\, dx\, dy\, dz}{V(c_n)} \right)^2 \tag{9}$$

for each cell, which measures the sum of differences between the mean values in the neighboring cells and average values of the reconstructed function from the original cell in the same neighboring cell. We want to minimize this functional, so we want the reconstructed function to be as close to the mean values in the neighboring cells as possible.

We easily compute derivative of this functional according to all three variables and let them be equal to zero. This gives us a linear system

$$\frac{\partial F(S_c^x, S_c^y, S_c^z)}{\partial S_c^{\{x,y,z\}}} = 0\,, \tag{10}$$

which can be easily solved and gives us our unlimited slopes.

**Limited Slopes** For computation of the slopes $S_c^{\{x,y,z\}}$ we use Barth-Jasperson limiter in each cell vertex $n$ and than the minimal of them as a cell limiter

$$\Phi_n = \begin{cases} \min\left(1, \frac{\rho_n^{\max} - \bar{\rho}_c}{\rho_n^{\mathrm{unlim}} - \bar{\rho}_c}\right) & \text{for } \rho_n^{\mathrm{unlim}} - \bar{\rho}_c > 0 \\ \min\left(1, \frac{\rho_n^{\min} - \bar{\rho}_c}{\rho_n^{\mathrm{unlim}} - \bar{\rho}_c}\right) & \text{for } \rho_n^{\mathrm{unlim}} - \bar{\rho}_c < 0 \\ 1 & \text{for } \rho_n^{\mathrm{unlim}} - \bar{\rho}_c = 0\,, \end{cases} \qquad \Phi_c = \min_{n \in \mathcal{N}(\rfloor)} \Phi_n\,, \tag{11}$$

which ensures us preservation of local extremes and also preservation of linear function. It is described in details in [2].

**Integration over Arbitrary Polyhedron** The only part in the functional, we don't know, is the integral

$$\int_{c_n} \rho_c(x,y,z)\, dx\, dy\, dz\,. \tag{12}$$

We also need to compute integrals in definition of cell centers $x_c$, $y_c$, $z_c$ (6) and cell volumes $V_{c_n}$ (7). So we need a method for integration of the linear function over arbitrary polyhedron. We note, that the boundary of the polyhedron is not obvious, we know just the vertices of each face. If the face vertices do not lie in one plane, the face is curved and the boundary is not uniquely defined.

We show our integration procedure on the example of the cell volume, the integration of arbitrary linear function is analogical. The cell volume can be written in the form

$$V = \int_V \mathbf{1}\, dV = \frac{1}{3} \int_V \mathrm{div}(x,y,z)\, dV \tag{13}$$

and using the Divergence Theorem we can rewrite it

$$V = \frac{1}{3} \int_{\partial V} (x,y,z)^{\intercal} \cdot \mathbf{S}\, dA\,. \tag{14}$$

The boundary integral can be split into face integrals

$$V = \frac{1}{3} \sum_{\Pi \in \partial V} \int_{\Pi} (x, y, z)^{\mathsf{T}} \cdot \mathbf{S} \, dA \tag{15}$$

Now just by averaging the coordinates of vertices of each face we compute its center, connect it with all face vertices and split these face integrals to the integrals over such defined triangles. On each this triangle the face normal $\mathbf{S}$ is constant so it can go in front of the integral

$$V = \frac{1}{3} \sum_{\Pi \in \partial V} \sum_{\Delta \in \Pi} \left( S^x \int_{\Delta} x \, dA + S^y \int_{\Delta} y \, dA + S^z \int_{\Delta} z \, dA \right) \tag{16}$$

Now we project all triangles to the coordinate planes. For each triangle we select such the coordinate plane, in which the triangle has the biggest area. This ensures us that we do not get into troubles due to numerical problems. Using Green Theorem we reduce these triangle integrals to 1D edge integrals, which can be computed directly from vertex coordinates. This algorithm gives us method for computing integral of the arbitrary linear function over arbitrary polyhedron. More details can be seen in [3].

### 2.4 Stage 2 – Swept Integration

Swept region quadrature concept has been explained in details in [1].

Swept region is created by movement of the face from the original grid to the new position. It is bordered by the old face, the new face, and by not flat quadrilaterals connecting each edge from the original face to the edge of the new face. We can compute volume and mass of such region - we talk about swept volume and swept mass. We use these terms in their signed sense. Suppose we have a cell on the original mesh and we move just one face as illustrated on the Fig. 1. In this case, the right face moves outward

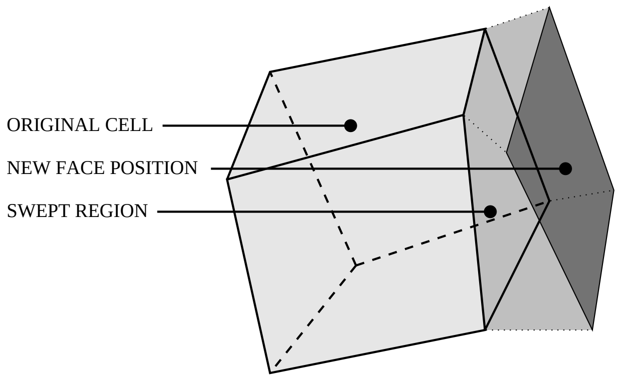



**Fig. 1.** One swept region

from the original cell and the middle part is the swept region. In fact, all faces can move in different ways and swept regions can be tangled. If the most of swept region goes outward from the original cell, the swept volume and swept mass are positive, otherwise they are negative. Mass of the swept region is computed by integration of the reconstructed function from cell, in which the most swept region lies. The new cell mass can be composed from the mass of the original cell and the masses of all swept regions

$$m^*(\tilde{c}) = m(c) + \sum_{f \in \mathcal{F}(c)} \delta m_f \,. \tag{17}$$

New mean value can be than computed as

$$\bar{\bar{\rho}}_{\tilde{c}} = \frac{m^*(\tilde{c})}{V(\tilde{c})} \tag{18}$$

and as noticed before, it can violate the local bounds due to the approximation of the integration. So the third stage is necessary to enforce local-bound preservation.

### 2.5 Stage 3 – Repair

The repair stage works as the conservative redistribution of conservative quantity. It corrects the overshoots back to their local bounds. At first, we must compute these local extremes. For each cell $c$ we define bound-determining neighborhood $\mathcal{C}(c)$, which is a piece of the original grid fully covering the new cell. Usually we use the original cell plus its nearest neighbors. We compute the local extremes in this neighborhood

$$\rho_c^{\min} = \min_{c_n \in \mathcal{C}(c)} \bar{\rho}_{c_n} \,, \quad \rho_c^{\max} = \max_{c_n \in \mathcal{C}(c)} \bar{\rho}_{c_n} \,. \tag{19}$$

We show the repair on the example of violation of the lower bound

$$\bar{\bar{\rho}}_{\tilde{c}} < \rho_c^{\min} \,, \tag{20}$$

upper bound is done similarly. At first we compute mass, which is needed in the cell to bring the mean value back to the local minimum

$$\delta m_{\tilde{c}}^{\text{needed}} = (\rho_c^{\min} - \bar{\bar{\rho}}_{\tilde{c}}) \, V(\tilde{c}) \,. \tag{21}$$

We want our algorithm to be conservative, so we do not just add this mass to the wrong cell, but we look for available mass in the bound-determining neighborhood. For each neighboring cell we compute the mass

$$\delta m_{\tilde{c}_n}^{\text{avail}} = \max\Big( (\bar{\bar{\rho}}_{\tilde{c}_n} - \rho_{c_n}^{\min}) \, V(\tilde{c}), 0 \Big) \tag{22}$$

which can safely be taken from the cell without violating the local bound also. The total available mass in the neighborhood is

$$\delta m_{\mathcal{C}(c)}^{\text{avail}} = \sum_{c_n \in \mathcal{C}(c)} \delta m_{\tilde{c}_n}^{\text{avail}} . \tag{23}$$

If the available mass is too small ($\delta m_{\mathcal{C}(c)}^{\text{avail}} < \delta m_{\tilde{c}}^{\text{needed}}$), we extend the stencil and look for the available mass in a larger area. If there is enough mass available, we perform the repair. We bring the wrong value back to the local minimum

$$m'(\tilde{c}) = \rho_c^{\min} \, V(\tilde{c}) \tag{24}$$

and we take the mass from the neighborhood proportionally to the mass available

$$m'(\tilde{c}_n) = m(\tilde{c}_n) - \frac{\delta m_{\tilde{c}_n}^{\text{avail}}}{\delta m_{\mathcal{C}(\tilde{c})}^{\text{avail}}} \, \delta m_{\tilde{c}}^{\text{needed}} \,. \tag{25}$$

In [1] we proved that this algorithm succeeds in a finite number of steps and the repair stage corrects all local-bound violations.

# 3 Numerical Tests

## 3.1 Orthogonal Uniform Grid

In the first example the underlying function is equal to zero everywhere, only in a spherical region around the center of the computation domain $\langle 0,1\rangle^3$ it is equal to 1

$$g(x,y,z) = \begin{cases} 1 & \text{for } r \le 0.25 \\ 0 & \text{else} \end{cases}, \qquad r = \sqrt{\left(x-\frac{1}{2}\right)^2 + \left(y-\frac{1}{2}\right)^2 + \left(z-\frac{1}{2}\right)^2}\,. \tag{26}$$

We define the uniform orthogonal grid in the computational domain, the initial function

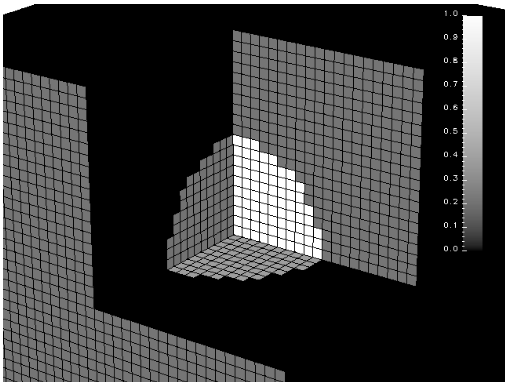



**Fig. 2.** Initial spherical function on $40^3$ orthogonal uniform grid

is shown on the Fig. 2. We move the grid as the tensor product movement

$$x_n^{\rm new} = (1-\alpha)\,x_n + \alpha\,x_n^{1.5}\,, y_n^{\rm new} = (1-\alpha)\,y_n + \alpha\,y_n^{2.0}\,, z_n^{\rm new} = (1-\alpha)\,z_n + \alpha\,z_n^{2.5}\,, \tag{27}$$

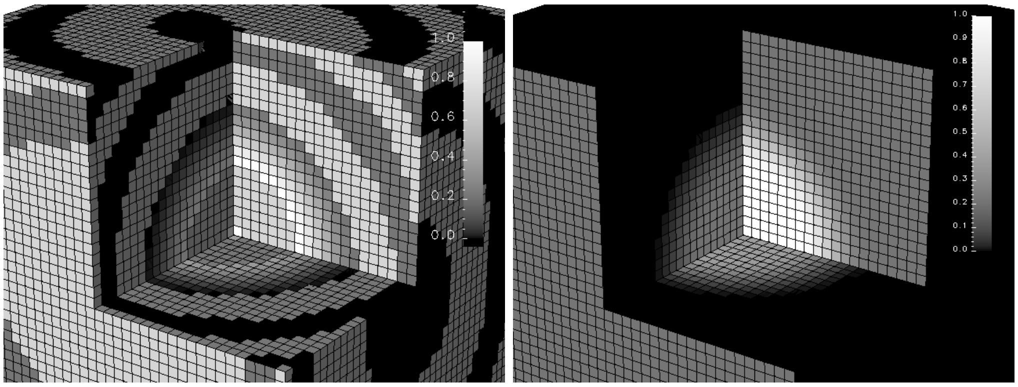
where

$$\alpha = 0.5\,\sin(4\,\pi\,t)\,, \qquad t = N/N_{\max}\,, \quad t \in \langle 0,1\rangle \text{ -- time of } N\text{th timestep.} \tag{28}$$

We make $N_{\max} = 200$ remappings to obtain accumulated errors and to have the problematic regions visible. On the Fig. 3 we can see this spherical function remapped using only unlimited slopes. This causes more errors, so the effect of the repair stage is more obvious. On the a) part of the figure we can see the function without repair stage. The light gray cells show areas, where the extremes are violated. On the b) part we see the same remapping with repair, no values violate the bounds.

## 3.2 Tetrahedral Grid with Random Movement

The second numerical example shows the same cubical computational domain with tetrahedral mesh inside. It includes about 9000 tetrahedrons. We use the same spherical function as before, we can see it on the Fig. 4. Now, we shake the grid randomly 10 times and remap between these grids. In the last time step we remap back to the original grid. On the Fig. 5 we can see the situation with the usage of the Barth-Jasperson limiter with and without the repair stage. Again, we can see several white cells in the a) part, where the bounds are violated. In the b) part, the repair stage corrects everything and no problem with bound preservation is observed.




a) b)

**Fig. 3.** 200times remapped spherical function using only unlimited reconstruction a) without repair b) with repair

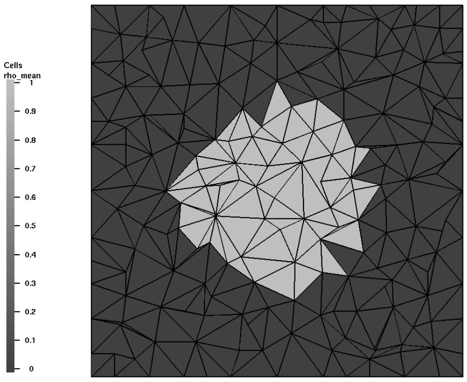
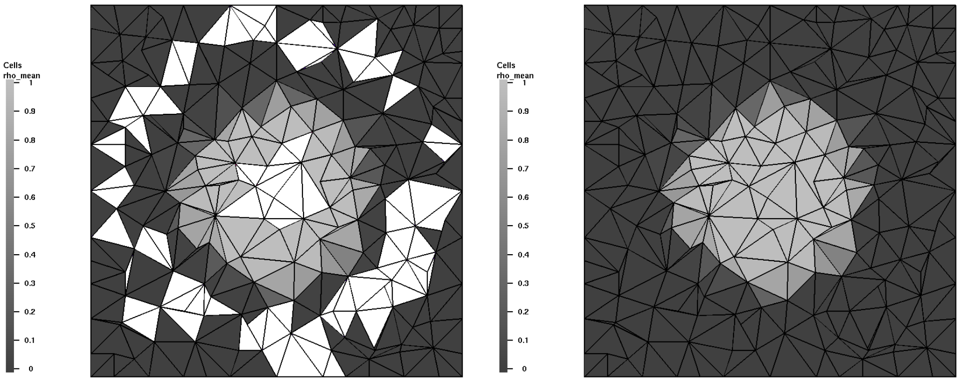



**Fig. 4.** Initial spherical function on about 9000 tetrahedrons




a) b)

**Fig. 5.** 10times remapped spherical function using Barth-Jasperson limiter a) without repair b) with repair

## 4 Conclusion

In this article we constructed an efficient algorithm for function remapping between two similar grids. It is face-based and usable in 3D unlike the most natural exact integration algorithm, which is not feasible in 3D. The algorithm is conservative (total mass remains constant), local-bound preserving (does not create new extremes), stable and linearity preserving. We presented several numerical examples to show, that we can use it for different types of grids and grid movements.

## 5 Acknowledgments

This work was performed under the auspices of the US Department of Energy at Los Alamos National Laboratory, under contract W-7405-ENG-36. The authors acknowledge the partial support of the DOE/ASCR Program in the Applied Mathematical Sciences and the Laboratory Directed Research and Development program (LDRD). R. Garimella and M. Shashkov acknowledge the partial support of DOE's Accelerated Strategic Computing Initiative (ASCI). M. Kuchařík also acknowledges the partial support of the Czech Technical University grant CTU0310614. The authors thank L. Margolin, B. Wendroff, B. Swartz, R. Liska, M. Berndt and V. Dyadechko for fruitful discussions and constructive comments.

## References

1. M. Kuchařík, M. Shashkov, and B. Wendroff. An efficient linearity-and-bound-preserving remapping method. *Journal of Computational Physics*, 188(2):462–471, 2003.
2. T. J. Barth. Numerical methods for gasdynamic systems on unstructured meshes. In C. Rohde D. Kroner, M. Ohlberger, editor, *An introduction to Recent Developments in Theory and Numerics for Conservation Laws, Proceedings of the International School on Theory and Numerics for Conservation Laws*, Berlin, 1997. Lecture Notes in Computational Science and Engineering, Springer.
3. B. Mirtich. Fast and accurate computation of polyhedral mass properties. *Journal of Graphics Tools*, 1(2):31–50, 1996.