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*

Summary. The remapping algorithm is an essential part of the ALE (Arbitrary Lagrangian-Eulerian) method. In this talk we present such an algorithm based on linear function reconstruction, approximate integration and mass redistribution.

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 the Lagrangian and Eulerian approaches.

At first, several time steps of the pure Lagrangian computation are used. As the grid moves together with the 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 extrema, we want it to be local-bound preserving. It must be stable and applicable to general unstructured meshes in 2D and 3D. In this article we introduce a 3D algorithm, which satisfies these conditions. A 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 ρ is mass density, $\mathbf{U} = (u, v, w)$ is a vector of velocities and ε is the internal energy). We do not know the function itself, we know just the mean values in the grid cells and their masses and volumes

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

Total mass (momentum, energy) in the computational domain Ω can be computed as

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

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

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

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 extrema

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

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 an approximate method and it may happen that the local extrema 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

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

are coordinates of the cell center and $V(c)$ is the volume of the cell defined in (1).

For computation of slopes we use the limited form

$$S_c^x = \Phi_c\,S_c^{x\ \mathrm{unlim}}\quad, S_c^y = \Phi_c\,S_c^{y\ \mathrm{unlim}}\quad, S_c^z = \Phi_c\,S_c^{z\ \mathrm{unlim}}\,, \tag{7}$$

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

Unlimited Slopes In 1D we can use just the central difference as the unlimited slope. To compute unlimited slopes in 2D we construct a contour surrounding the cell and use Green's 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

$$F(S_c^x, S_c^y, S_c^z) = \sum_{c_n \in C(c)} \left(\bar{\rho}_{c_n} - \frac{\int\limits_{c_n} \rho_c(x,y,z)\,dx\,dy\,dz}{V(c_n)} \right)^2 \tag{8}$$

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 with respect to all three variables and let them be equal to zero. This gives us a linear system

$$\frac{\partial F(S_c^x, S_c^y, S_c^z)}{\partial S_c^{\{x,y,z\}}} = 0\,, \tag{9}$$

which can be easily solved and gives us our unlimited slopes $S_c^{\{x,y,z\}\ \mathrm{unlim}}$.

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

$$\Phi_n = \begin{cases} \min\left(1, \frac{\rho_n^{\max}-\bar{\rho}_c}{\rho_n^{\text{unlim}}-\bar{\rho}_c}\right) & \text{for } \rho_n^{\text{unlim}} - \bar{\rho}_c > 0 \\ \min\left(1, \frac{\rho_n^{\min}-\bar{\rho}_c}{\rho_n^{\text{unlim}}-\bar{\rho}_c}\right) & \text{for } \rho_n^{\text{unlim}} - \bar{\rho}_c < 0 \\ 1 & \text{for } \rho_n^{\text{unlim}} - \bar{\rho}_c = 0\,, \end{cases} \qquad \Phi_c = \min_{n\in\mathcal{N}(c)} \Phi_n\,, \tag{10}$$

which ensures us preservation of local extrema and also preservation of a linear function. Here ρ_n^{unlim} is the value of the reconstructed function (using the unlimited slopes) in the node n. It is described in details in [2].

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

$$\int_{c_n} \rho_c(x,y,z)\, dx\, dy\, dz\,. \tag{11}$$

We also need to compute the integrals in the definition of cell centers x_c, y_c, z_c (6) and cell volumes V_{c_n} (1). So we need a method for integration of the linear function over an arbitrary polyhedron. We note, that the boundary of the polyhedron is uniquely defined , 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 demonstrate our integration procedure for the example of the cell volume, the integration of an arbitrary linear function is similar. The cell volume can be written in the form

$$V(c) = \int_c 1\, dV = \frac{1}{3}\int_c \operatorname{div}(x,y,z)\, dV \tag{12}$$

and using the Divergence Theorem we can rewrite it as an integral over the boundary ∂c

$$V(c) = \frac{1}{3}\int_{\partial c} (x,y,z)^{\mathsf{T}} \cdot \mathbf{S}\, dA\,. \tag{13}$$

Here the superscript $^{\mathsf{T}}$ means the transposition of a vector and $\mathbf{S}$ is the vector normal to the boundary. The boundary integral can be split into the sum over all faces Π of the face integrals

$$V(c) = \frac{1}{3}\sum_{\Pi\in\partial V}\int_{\Pi} (x,y,z)^{\mathsf{T}} \cdot \mathbf{S}\, dA \tag{14}$$

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 Δ. On each this triangle the face normal **S** is constant so it can go in front of the integral

$$V(c) = \frac{1}{3} \sum_{\Pi \in \partial V} \sum_{\Delta \in \Pi} \left(S^x \int_{\Delta} x\, dA + S^y \int_{\Delta} y\, dA + S^z \int_{\Delta} z\, dA \right) \tag{15}$$

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

2.4 Stage 2 – Swept Integration

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

The swept region is created by the 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 necessarily flat quadrilaterals connecting each edge from the original face to the edge of the new face. We can compute the volume and mass of a 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

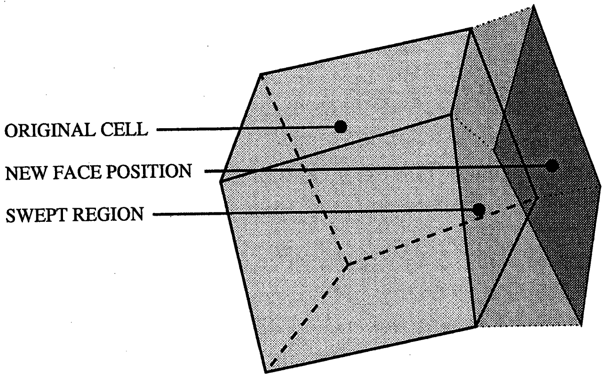


Fig. 1. One swept region

outward 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 most of the swept region goes outward from the original cell, the swept volume and swept mass are positive, otherwise they are negative. The mass of the swept region is computed by integration of the reconstructed function over the 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

$$m^*(\tilde{c}) = m(c) + \sum_{f \in \mathcal{F}(c)} \delta m_f \,. \tag{16}$$

Here f means a swept region from the set $\mathcal{F}(c)$ of all swept regions of the cell c. The new mean value can be than computed as

$$\bar{\bar{\rho}}_{\tilde{c}} = \frac{m^*(\tilde{c})}{V(\tilde{c})} \tag{17}$$

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 a conserved quantity. It corrects the overshoots back to their local bounds. At first, we must compute these local extrema. For each cell c we define a 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 extrema in this neighborhood

$$\rho_c^{\min} = \min_{c_n \in \mathcal{C}(c)} \bar{\rho}_{c_n} \,, \quad \rho_c^{\max} = \max_{c_n \in \mathcal{C}(c)} \bar{\rho}_{c_n} \,. \tag{18}$$

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

$$\bar{\bar{\rho}}_{\tilde{c}} < \rho_c^{\min} \,, \tag{19}$$

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

$$\delta m_{\tilde{c}}^{\text{needed}} = (\rho_c^{\min} - \bar{\bar{\rho}}_{\tilde{c}}) \, V(\tilde{c}) \,. \tag{20}$$

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

$$\delta m_{\tilde{c}_n}^{\text{avail}} = \max\Big((\bar{\bar{\rho}}_{\tilde{c}_n} - \rho_{c_n}^{\min}) \, V(\tilde{c}), 0 \Big) \tag{21}$$

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

$$\delta m_{\mathcal{C}(c)}^{\text{avail}} = \sum_{c_n \in \mathcal{C}(c)} \delta m_{\tilde{c}_n}^{\text{avail}} . \tag{22}$$

If the available mass is too small ($\delta m^{\text{avail}}_{C(c)} < \delta m^{\text{needed}}_{\tilde{c}}$), 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

$$m'(\tilde{c}) = \rho^{\min}_{c} V(\tilde{c}) \tag{23}$$

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

$$m'(\tilde{c}_n) = m(\tilde{c}_n) - \frac{\delta m^{\text{avail}}_{\tilde{c}_n}}{\delta m^{\text{avail}}_{C(\tilde{c})}} \, \delta m^{\text{needed}}_{\tilde{c}} \, . \tag{24}$$

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

$$g(x,y,z) = \begin{cases} 1 & \text{for } r \le 0.25 \\ 0 & \text{else} \end{cases} , \qquad r = \sqrt{\left(x-\frac{1}{2}\right)^2 + \left(y-\frac{1}{2}\right)^2 + \left(z-\frac{1}{2}\right)^2} \, . \tag{25}$$

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

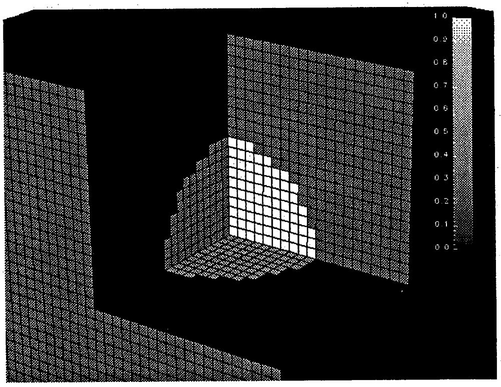


Fig. 2. Initial spherical function on 40^3 orthogonal uniform grid

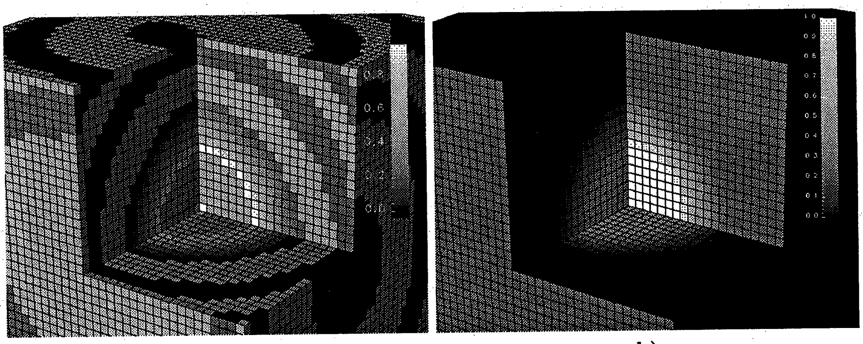
function is shown on the Fig. 2. We move the grid as the tensor product movement

$$x_n^{\text{new}} = (1-\alpha)\, x_n + \alpha\, x_n^{1.5}\,, y_n^{\text{new}} = (1-\alpha)\, y_n + \alpha\, y_n^{2.0}\,, z_n^{\text{new}} = (1-\alpha)\, z_n + \alpha\, z_n^{2.5}\,, \tag{26}$$

where

$$\alpha = 0.5\,\sin(4\pi t)\,, \qquad t = N/N_{\max}\,, \quad t \in \langle 0, 1\rangle \text{ -- time of } N\text{th timestep.} \tag{27}$$

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



a) b)

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

remapped using only unlimited slopes. This causes more errors, so the effect of the repair stage is more obvious. In the a) part of the figure we can see the function without the repair stage. The light gray cells show areas where the extrema are violated. In 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.

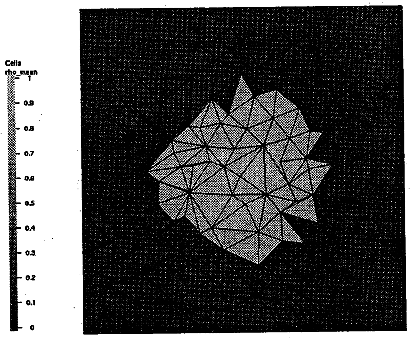


Fig. 4. Initial spherical function on about 9000 tetrahedrons

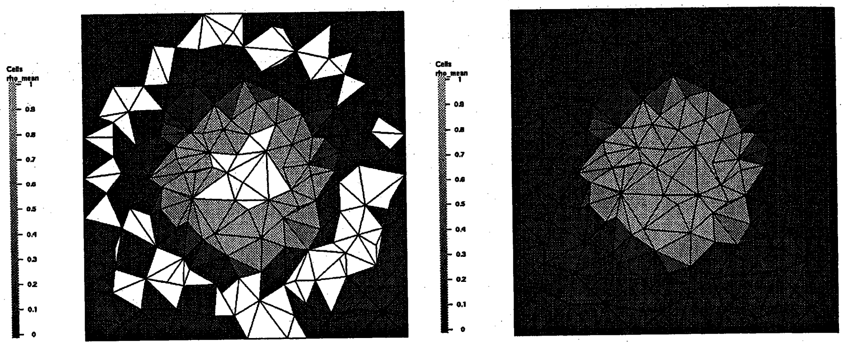


Fig. 5. 10times remapped spherical function using Barth-Jasperson limiter a) without repair b) with repair

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 extrema), 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.