Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 211 (2006) 385–404
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

The repair paradigm: New algorithms and applications to compressible flow

Raphaël Loubère *, Martin Staley, Burton Wendroff

Los Alamos National Laboratory, Theoretical Division T-7, MS B284, Los Alamos, NM 87545, USA

Received 14 October 2004; received in revised form 5 April 2005; accepted 20 May 2005

Abstract

The repair paradigm leads to several algorithms for redistributing mass, momentum and energy, while adhering to local maximum principles, as an adjunct to the remapping step in certain compressible flow codes that use remapping, such as Arbitrary–Lagrangian–Eulerian codes, or for just redistributing mass in advection codes. In the case of advection of a concentration, repair keeps the newly computed concentration in a cell between the maximum and minimum concentrations in neighboring old cells, thus guaranteeing at least that the new concentration is between zero and one. For compressible flow, density, velocity and internal energy are similarly constrained while maintaining conservation of mass, momentum and total energy. In this way, positive density and internal energy are achieved as a side effect. We propose a new algorithm, combining both local and global repair, that maintains causality and is efficient in a parallel computational setting. The local/global algorithm is independent of the order in which the distribution is performed, and it maintains 1D symmetry. This is applied to advection in two dimensions, and to, among others, the LeBlanc problem, the Sedov problem, and an interacting 2D blast wave problem. The latter is done with a Lagrangian code for which rezoning, remapping and repair are essential.

Keywords: Repair; Mass redistribution; Conservative reconstruction; Remapping; Advection; Hydrodynamics

1. Introduction

A critical part of Lagrangian-based methods for computational fluid dynamics (CFD) is the ability to remap or interpolate data from one computational mesh to another. This is the case for the popular

* Corresponding author. Tel.: +1 505 667 1407.
E-mail addresses: loubere@lanl.gov (R. Loubère), mstaley@lanl.gov (M. Staley), bbw@lanl.gov (B. Wendroff).

0021-9991/$ - see front matter
doi:10.1016/j.jcp.2005.05.010

ALE schemes that perform Lagrangian steps followed by remaps to fixed grids. Remapping is also essential for pure Lagrangian methods, because they can lead to tangled grids that must then be untangled with a concomitant remap step. Even if the basic scheme produces only physically meaningful quantities, a remapping method can create out-of-bounds quantities such as negative densities or pressures. In some CFD codes, the offending values are simply set to a small positive number when this occurs, at which point mass or total energy is no longer conserved. In most instances the error thereby created is negligible, but we shall show that in at least one example the error is significant.

It is possible, by taking great care with the remapping in the CFD context, to maintain positive mass density. This is done by first extending the given mean densities in each original cell to the whole domain so that the new distribution is everywhere positive, and then computing new mean values by exact integration over the cells of the new grid. Total energy can be remapped in this way, but then there is no guarantee that internal energy will be positive. Furthermore, in more than one dimension, exact integration is computationally intensive.

Another context in which non-physical data can occur is in divergence-free advection of a concentration that must retain values between zero and one. High quality advection schemes, some of which are based on remapping ideas [1,2], unavoidably have this fault [3].

The goal in this paper is to improve upon and apply the repair idea introduced in [4,5]. A repair method can be viewed as a way to correct values on a discrete mesh by redistributing the conserved quantity so that conservation and a maximum principle are preserved. The maximum principle is that new values should obey certain upper and lower bounds obtained from old values. In this way, not only are non-physical quantities eliminated, but oscillations are reduced (albeit not necessarily eliminated). We therefore seek repair algorithms that can be applied to CFD problems, advection problems, or other situations where values of a discrete variable must be placed in bounds without violating a conservation law and without introducing significant errors in the dynamics.

As stated in [4] (Section 8, p275), repair is a mass redistribution nonlinear filter. Other methods for the correction of nonphysical data, such as flux corrected transport, are discussed in [3].

The rest of this paper is arranged as follows. We first present notation, goals and expected properties of repair methods. We then review a local repair method [5] which repairs out-of-bounds values and distributes the resulting mass discrepancies locally. This method can produce different results depending on the order in which cells are visited, and it is therefore called order-dependent. Next we review a simple global repair process [4] which repairs out-of-bounds values and distributes the resulting mass discrepancy across the entire grid. The next two sections introduce order-independent local methods, and we conclude with a discussion of repair methods in advection and hydrodynamics contexts, where numerical tests are performed to show the effects of such methods.

2. Notation, goals and properties

Repair methods can be used for many kinds of variables, including density, velocity, energy, pressure, and concentration, but we will henceforth call our variable to be repaired a density ρ, or equivalently, a mass m. If we denote old cells by c and new cells by $\tilde{c}$, then the quantity to be conserved is the total mass $m = \sum_c m(c) = \sum_c \rho(c)V(c)$, where $m(c)$, $\rho(c)$, and $V(c)$ denote the mass, density, and volume, respectively, of cell c.

Consider an old mesh $\mathscr{M}$ with cell-averaged densities (called old densities), and a new mesh $\widetilde{\mathscr{M}}$ with remapped cell-averaged densities (called new densities). We assume for simplicity that the connectivity is the same for the old and new grids.

In the case of advection the meshes would coincide, but typically the new mesh is a small perturbation of the old one. Define the *bound* neighborhood $N(c)$ of a cell c as a patch of surrounding cells,

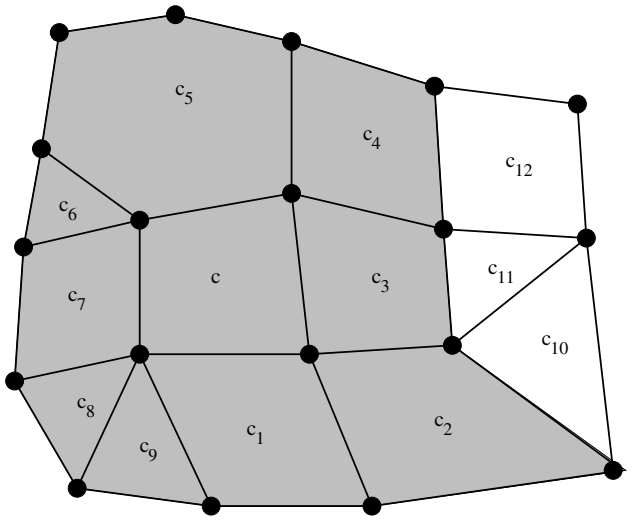

Fig. 1. An example of a neighborhood. Any cell in contact with c is an element of the set: $N(c) = \{c, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9\}$. In this example, c_{10}, c_{11}, and c_{12} are not part of the neighborhood of cell c.

as in Fig. 1. The notion of neighborhood is essential for any repair method. However it is a user choice how to define the notion of neighborhood. In our case the neighborhood of a given cell c is composed of all cells in contact with c, through either a node or an edge. (Alternatively, we could have defined the neighborhood as the set of all cells in contact with c through an edge, for example.) Formally,

$$N(c) = \{c', \quad c' \bigcap c \neq \emptyset\}. \tag{1}$$

$|N(c)|$ denotes the number of elements in the set $N(c)$. For example, $|N(c)| = 10$ in Fig. 1. The neighborhood can be expanded with an iterative process defined as follow ($N^1(c) = N(c)$):

$$\forall v > 1, \quad N^v(c) = \bigcup_{c' \in N^{v-1}(c)} N(c'). \tag{2}$$

Using this neighborhood definition, we can define maximum and minimum density bounds as $\rho_+(c) = \max_{s \in N(c)} \rho(s)$ and $\rho_-(c) = \min_{s \in N(c)} \rho(s)$. (There are, of course, other reasonable ways to define density bounds.) No matter how these bounds are defined, a feasibility condition is required in order for repair to work at all.

Feasibility. The total mass m must not exceed (respectively be below) the total upper bound mass (respectively the total lower bound mass), that is, the total mass if each new cell were at its upper (respectively lower) bound.

Let's illustrate the feasibility condition on a simple example: if the mass in each cell, of a 1D mesh having K cells, is m, and each upper bound mass is u and $m > u$, then the total mass is $M = K \cdot m$ and the upper bound mass is $U = K \cdot u$; hence $M > U$ and it is not feasible to repair every cell. M units of mass (which we must keep in order to maintain mass conservation) simply cannot fit into U units of mass.

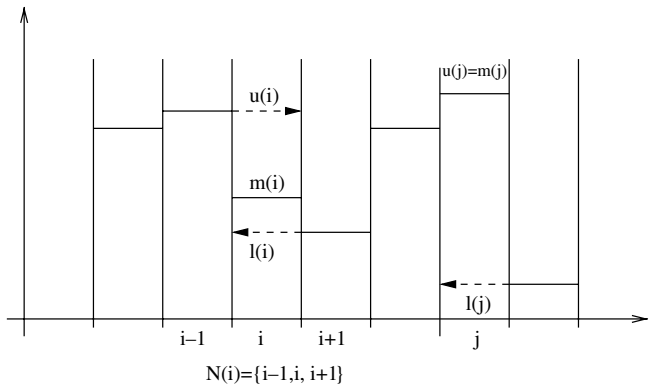

Fig. 2. Illustration of upper and lower bounds in 1D. We examine the cell masses before remapping takes place. The neighborhood of cell i consists of cells $i-1$, i, and $i+1$. Then, $u(i) = m(i-1)$ and $l(i) = m(i+1)$, while $u(j) = m(j)$ and $l(j) = m(j+1)$. After remapping, $m(i)$ can be out-of-bounds. For example, if $m(i) > u(i)$ then cell i needs repair.

Repair is not possible if this condition is violated. However, any repair procedure of the type that truncates out-of-bounds values and redistributes the discrepancy does work if it is satisfied.

If a remapping process produces negative densities $\rho(\tilde{c})$, or more generally produces out-of-bounds densities, then a repair step must be done to make these densities obey their bounds. The properties to be fulfilled by a repair method are:

Conservation:

$$\sum_c m(c) = \sum_c \rho(c)V(c) = \sum_{\tilde{c}} \rho(\tilde{c})V(\tilde{c}) = \sum_{\tilde{c}} m(\tilde{c}). \tag{3}$$

Maximum principle:

$$\forall c, \quad \rho_-(c) \leqslant \rho(\tilde{c}) \leqslant \rho_+(c). \tag{4}$$

The original grid plays no role in the repair process, other than to provide bounds, so for brevity we henceforth use i rather than $\tilde{c}$ to represent new cell indices. Let us write $u(i)$ and $l(i)$ for the upper and lower mass bounds, respectively, of cell i. Then,

$$u(i) = V(i)\rho_+(i) = V(i) \max_{s\in N(i)} \rho(s), \tag{5}$$

$$l(i) = V(i)\rho_-(i) = V(i) \min_{s\in N(i)} \rho(s). \tag{6}$$

These have dimensions of mass, while the mass of cell i is $m(i) = V(i)\rho(i)$.

In Fig. 2 we illustrate $u(i)$ and $l(i)$ in 1D. The neighborhood of a cell i is given by i and the two adjacent cells: $i-1$ and $i+1$. Therefore, we have $u(i) = \max(m(i-1), m(i), m(i+1))$ and $l(i) = \min(m(i-1), m(i), m(i+1))$.

3. Local order-dependent repair

This is perhaps the most obvious local repair algorithm. The underlying idea is to expand the neighborhood of a cell i which needs repair, until enough room is found in this neighborhood. Suppose cell i has a negative density, but the minimum bound is 0. This repair algorithm expands the neighborhood of cell i

until enough mass can be found and removed from the neighborhood to fill cell i and produce a repaired density equal to the minimum bound, that is, to 0. Then, the next cell is checked and repaired if necessary. A similar concept is applied to repair an over-bound value.

3.1. Algorithm

Suppose cell i, with upper bound $u(i)$, must be repaired because $m(i) > u(i)$, that is, because cell i's mass is above its upper bound. (A value that is below its lower bound would be treated similarly.) The first step is to compute the value needed to repair cell i:

$$m_{\mathtt{need}}(i) = m(i) - u(i). \tag{7}$$

This value is "needed" in the sense that if we repair $m(i)$ by reducing it to $u(i)$, then the loss $m_{\mathtt{need}}(i)$ must be added elsewhere in order to maintain conservation.

Next, we choose a set of neighboring cells $N(i)$, called the search neighborhood, and we check $N(i)$ to see if enough room is available to accommodate $m_{\mathtt{need}}(i)$:

$$m^{\mathtt{total}}_{\mathtt{avail}}(i) = \sum_{j \in N(i)} \max(u(j) - m(j), 0) \tag{8}$$

$$= \sum_{j \in N(i)} m_{\mathtt{avail}}(j), \tag{9}$$

where $m_{\mathtt{avail}}(j) \equiv \max(u(j) - m(j), 0)$ is the amount by which cell j's value, $m(j)$, can be increased without exceeding its upper bound, $u(j)$. If enough room is available in the neighborhood, that is, if $m^{\mathtt{total}}_{\mathtt{avail}}(i) \ \mathsf{P} \ m_{\mathtt{need}}(i)$, then the repair of cell i succeeds, and we make the following adjustments:

$$m(i) \leftarrow u(i), \tag{10}$$

$$\forall j \in N(i), \quad j \neq i, \quad m(j) \leftarrow m(j) + m_{\mathtt{need}}(i) \ \frac{m_{\mathtt{avail}}(j)}{m^{\mathtt{total}}_{\mathtt{avail}}(i)}. \tag{11}$$

That is, we clip $m(i)$ to its upper bound, then add its lost mass proportionally to all acceptor cells (cells whose masses are below their upper bounds, so that they can accept more mass) in its neighborhood.

If the search neighborhood $N(i)$ does not have enough room to accept cell i's excess mass, we extend the neighborhood until enough mass is available. It was shown in [4] that a sufficient condition for such an extension process to successfully terminate for repair of densities is that the (old) cells of the *bound* neighborhood cover the new cell i.

For each cell i, the local repair algorithm performs the procedure outlined above if the cell's value is above its upper bound, or performs a similar procedure if the cell's value is below its lower bound.

3.2. Properties and issues

This repair algorithm is order-dependent, meaning that the final result depends on the order in which cells are visited. This is because out-of-bounds cells are repaired as soon as they are found, and once it has been repaired, a cell's other properties, such as its ability to accept excess mass from elsewhere, change. Moreover, in repairing a cell the properties of some of its neighbors change as well, as mass is transferred between them and the cell being repaired.

This unphysical order-dependence is unacceptable in many practical situations, so we now focus our attention on developing order-independent repair algorithms.

4. Global order-independent repair

A simple, global order-independent repair algorithm clips out-of-bounds values to their bounds, counts the total discrepancy this produces in the quantity to be conserved, and spreads the discrepancy over the entire mesh. This method is order-independent because any cell that has to be repaired is immediately brought to its nearest bound and contributes to a total discrepancy which is not accounted for until all individual cells have been repaired.

4.1. Algorithm

The global repair algorithm begins by repairing every cell that needs repair, while keeping track of the discrepancy, that is, the change in total mass due to performing the repairs. Let each new cell have computed mass $m(i)$, upper bound $u(i)$ and lower bound $l(i)$ as defined in (5) and (6), and a neighborhood $N(i)$ as defined in (1). The upper and lower bounds are fixed numbers, while the values m and the discrepancy Δ evolve as repair progresses, as follows.

$$\Delta = \sum_i \max(0, m(i) - u(i)) - \sum_i \max(0, l(i) - m(i)), \tag{12}$$

$$m(i) \leftarrow \begin{cases} u(i) & \text{if } m(i) > u(i), \\ m(i) & \text{if } l(i) \leqslant m(i) \leqslant u(i), \\ l(i) & \text{if } m(i) < l(i). \end{cases} \tag{13}$$

Note that values above their upper bounds make positive contributions to Δ, while values below their lower bounds make negative contributions to Δ.

Each m is now within its bounds. If the total discrepancy, Δ, is zero, we stop. (This unlikely scenario would mean that the total above-bound mass precisely equaled the total below-bound mass.) Otherwise, we must add a total of Δ to the values of cells in the mesh that can accept it. Note that this works regardless of Δ's algebraic sign. A positive Δ means the bulk of out-of-bounds masses were above their upper bounds, in which case the procedure decreased those masses to their upper bounds, and we must add the (positive) Δ elsewhere to make up for the loss. A negative Δ means the bulk of out-of-bounds masses were below their lower bounds, in which case the procedure increased those masses to their lower bounds, and we must add the (negative) Δ elsewhere to get rid of the excess.

To adjust for the discrepancy Δ, we begin by computing the following:

$$k(i) = \begin{cases} u(i) - m(i), & \text{if } \Delta > 0, \\ m(i) - l(i), & \text{if } \Delta < 0. \end{cases} \tag{14}$$

For each cell i, this is the amount by which the cell's value is allowed to increase (if $\Delta > 0$) or decrease (if $\Delta < 0$) without going out of bounds. By construction, $k(i)$ is always positive or zero, because $m(i)$ has already been placed between its bounds. Whether $k(i)$ is interpreted as an allowable increase or an allowable decrease is determined by the algebraic sign of Δ.

Finally, this repair procedure cancels the discrepancy Δ as follows:

$$\forall i, \quad m(i) \leftarrow m(i) + \Delta \frac{k(i)}{K}, \tag{15}$$

where $K = \sum_i k(i)$ is the total allowable increase (if $\Delta > 0$) or decrease (if $\Delta < 0$) over all cells. Again by construction, $K > 0$, and we see that for applicable cells i, $m(i)$ increases if $\Delta > 0$ and decreases if $\Delta < 0$. (Of course, for some cells $k(i) = 0$, and $m(i)$ for those cells is unchanged in this step.) Using the fraction $k(i)/K$ means cells are adjusted in proportion to how much room they have.

The above procedure can be illustrated with the following example. Consider a mesh of four cells, for which the first cell's mass is 3 units above its allowable maximum, the second cell's mass is within bounds and 5 units below its maximum, the third cell's mass is within bounds and 6 units below its maximum, and the fourth cell's mass is at its maximum. The first cell is fixed by decreasing its mass by 3 units. To conserve total mass, we must add 3 units elsewhere. The second cell can take 5 units, the third cell 6 units, and the fourth cell 0 units. According to (15) cell 2's mass is increased by $3 \times 5/11$, and cell 3's mass is increased by $3 \times 6/11$.

Cell 3	Cell 4
$\mathtt{u} = 6$	$\mathtt{u} = 1$
$\mathtt{m} = 0$	$\mathtt{m} = 1$
$\mathtt{l} = 0$	$\mathtt{l} = 0$
Cell 1	**Cell 2**
$\mathtt{u} = 0$	$\mathtt{u} = 5$
$\mathtt{m} = 3$	$\mathtt{m} = 0$
$\mathtt{l} = 0$	$\mathtt{l} = 0$

$\Longrightarrow$

Cell 3	Cell 4
$\mathtt{u} = 6$	$\mathtt{u} = 1$
$\tilde{\mathtt{m}} = 0 + 3 \leftarrow 6/11$	$\tilde{\mathtt{m}} = 1$
$\mathtt{l} = 0$	$\mathtt{l} = 0$
Cell 1	**Cell 2**
$\mathtt{u} = 0$	$\mathtt{u} = 5$
$\tilde{\mathtt{m}} = 0$	$\tilde{\mathtt{m}} = 0 + 3 \leftarrow 5/11$
$\mathtt{l} = 0$	$\mathtt{l} = 0$

4.2. Properties and issues

Clearly, this algorithm is conservative and order-independent. It is also symmetry-preserving, in the sense that equivalent cells (cells which have the same mass and bounds) are treated in the same manner. Moreover, the algorithm is well suited for parallelization.

The drawback is that such a repair process can violate the physics in computational fluid dynamics. Consider its behavior on a simple example, the Sod Riemann problem in 2D, in which a 1D rarefaction wave, a contact discontinuity, and a shock wave begin to differentiate from each other. Such waves are separated by plateaus. Suppose an over-bound value of energy in the post-shocked region has to be repaired. With the present algorithm, this excess energy is spread uniformly over all acceptor cells – all cells that can accept some mass – even if those cells are in the rarefaction wave or are far from the shock wave. So, energy that should have been distributed close to the shocked region appears far away. By repeating such a repair process on every time step, energy is gradually removed from its physical position and scattered elsewhere, and the result is a poor approximation of shock speed, density/energy plateaus, etc. In other words, such a repair process on hydrodynamics problems can severely perturb the physics of the phenomena one is trying to study.

Such an argument can actually be applied to every repair method we present. Because a redistribution of an out-of-bounds cell's value is always involved, its excess energy appears elsewhere instantaneously. However, with local algorithms, in which out-of-bounds values are redistributed in a neighborhood of the cell in question, we have not experienced such modification of the physics. Expanding a cell's neighborhood only as far as necessary provides the smallest neighborhood into which mass must be redistributed, while with the global repair algorithm a cell's "neighborhood" is the entire domain.

For a pure advection problem, the global repair algorithm can be appreciated for its simplicity and its ability to parallelize.

5. Local order-independent repair

We have developed a new, iterative, order-independent repair algorithm that addresses the disadvantages of the previous algorithm. This algorithm is known to converge, as we will show in the next section.

5.1. Algorithm

This is a two-stage algorithm: one stage repairs all values that are above their upper bound, and the other stage repairs all values that are below their lower bound. Upper bounds can be fixed before lower bounds, or vice versa. The order affects the result, but given a choice of order, the algorithm produces the same result regardless of the order in which *cells* are examined. Repair of the upper bounds proceeds as follows.

Upper-bounds repair. If $m(i) \leqslant u(i)$ for every cell i, then no upper-bounds repair is needed. Otherwise:

(1) $\forall i$, let $\delta(i) = 0$. As the algorithm proceeds, $\delta(i)$ can accumulate portions of nearby cells that are above their upper bounds, if there is room for those portions in cell i.
(2) While $\exists i$ with $m(i) > u(i)$, iterate the following.
(3) $\forall i$ with $m(i) > u(i)$, do the following. First, let $e(i) = m(i) - u(i)$. This is cell i's excess mass, which we must distribute to nearby cells. Find the smallest neighborhood $N(i)$ whose acceptor cells j (cells whose values are below their maximum bounds) can accept, in total, at least $e(i)$ units of mass: $a \equiv \sum_j (u(j) - m(j))$ P $e(i)$. For each of these acceptor cells j, let $\delta(j) \leftarrow \delta(j) + e(i) \cdot (u(j) - m(j))/a$. That is, distribute cell i's excess mass into the δ's of neighboring acceptor cells j, in proportion to what each of these acceptor cells can receive. Note that we haven't yet modified m anywhere.
(4) $\forall i$, if $m(i) > u(i)$ then repair the cell: set $m(i) \leftarrow u(i)$. Otherwise, check $\delta(i)$ to see if cell i was an acceptor for some other cell's excess mass. If $\delta(i) > 0$ then it was, and we now "accept" that mass: set $m(i) \leftarrow m(i) + \delta(i)$. However, doing this might put $m(i)$ above its upper bound, in which case we set a flag indicating that another iteration is necessary.
(5) Set $\delta(i) = 0$, because $\delta(i)$ has now been accounted for.
(6) Iterate (go to step 3) if necessary.

We omit the procedure for repairing lower bounds, which is similar.

In step 4 we remarked that the modification $m(i) \leftarrow m(i) + \delta(i)$ might put $m(i)$ above its upper bound. Recall that the individual contributions to any cell's δ, in the form $e(i) \cdot (u(j) - m(j))/a$ as in step 3, are by construction too small to put the cell's value above its upper bound. However, an acceptor cell can accept such contributions from many nearby out-of-bounds cells, and the sum of those contributions, accumulated in δ and finally added to $m(i)$ in step 4, can possibly put $m(i)$ above its upper bound. If this happens, then this $m(i)$ will be fixed in the next iteration.

Each iteration fixes all values that were above their upper bounds at the beginning of the iteration, but as we have just seen, doing so can put other values above their upper bounds. Although this can happen, the algorithm is guaranteed to converge. Once a cell is repaired, it can never receive a nonzero δ, and can therefore never be broken. Because each iteration fixes at least one cell, and never breaks a cell that has already been fixed, the algorithm must converge (in at most N iterations, if N is the number of cells) if repair is feasible at all.

This algorithm is conservative, and it achieves order-independence by distributing excess mass into δ and only taking δ into account later. However, in the above form the algorithm does not preserve a 1D symmetry. Consider the following symmetric example, in which cells 1, 2, and 3 are each 1 unit above their maximum, and cells 4, 5, and 6 are within their bounds. Under the new algorithm, the upper-bounds repair proceeds as follows:

Cell 4	Cell 5	Cell 6
u = 0.2	u = 0.2	u = 0.2
m = −1	m = −1	m = −1
l = −1	l = −1	l = −1
Cell 1	**Cell 2**	**Cell 3**
u = 0	u = 0	u = 0
m = 1	m = 1	m = 1
l = 0	l = 0	l = 0

$\Longrightarrow$

Cell 4	Cell 5	Cell 6
$\delta = 1/2 + 1/3$	$\delta = 1/2 + 1/3 + 1/2$	$\delta = 1/3 + 1/2$
Cell 1	**Cell 2**	**Cell 3**
$e = 1$	$e = 1$	$e = 1$

Cell 4 is one of Cell 1's *two* acceptor neighbors, and one of Cell 2's *three* acceptor neighbors, so it receives δ portions of 1/2 and 1/3 from those cells, respectively. Similarly, cell 5 receives portions from cells 1, 2, and 3, while cell 6 receives portions from cells 2 and 3. After the first iteration, we get values as follows:

Cell 4	Cell 5	Cell 6
$m \leftarrow m + \delta = -1/6$	$m \leftarrow m + \delta = 1/3$	$m \leftarrow m + \delta = -1/6$
Cell 1	**Cell 2**	**Cell 3**
$m \leftarrow u = 0$	$m \leftarrow u = 0$	$m \leftarrow u = 0$

which are in-bounds but are not symmetric.

5.2. Symmetry preservation

The loss of symmetry in this algorithm occurs because cells that should be equivalent, because they have the same values and same bounds, are treated differently according to whether or not they are near the boundary. In the above example, cells 1, 2, and 3 have the same values and same bounds, but cells 1 and 3, due to their proximity to the boundary, have only two neighboring acceptor cells, while cell 2 has three neighboring acceptor cells.

To preserve symmetry, we can double-count boundary cells in an appropriate way. Equivalently, we can introduce ghost cells with the property that a modification made to a ghost cell is later transferred to its corresponding real cell. Fig. 3 illustrates this process.

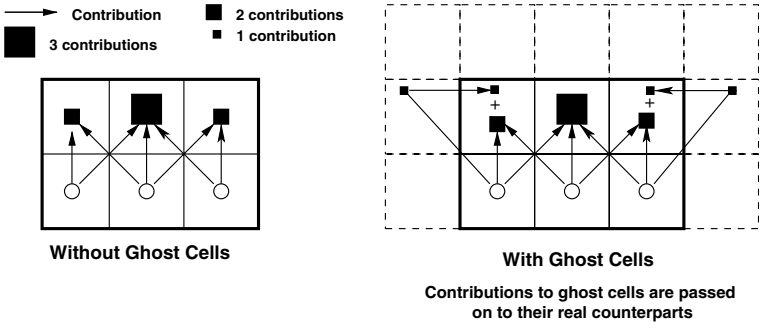

Fig. 3. Preservation of basic symmetry. In the absence of ghost cells, the repair algorithm transfers more mass from broken cells (open circles) to nearby internal cells (big black squares) than to nearby boundary cells (medium black squares). If we introduce ghost cells, then the acceptor cells on the boundary receive mass from broken cells both directly, and indirectly through their ghost copy (small black squares).

Consider the previous example, now with ghost cells (denoted by primes). Initially, we have:

Cell 4'	Cell 4'	Cell 5'	Cell 6'	Cell 6'
Cell 4'	**Cell 4** u = 0.2 m = −1 l = −1	**Cell 5** u = 0.2 m = −1 l = −1	**Cell 6** u = 0.2 m = −1 l = −1	Cell 6'
Cell 1'	**Cell 1** u = 0 m = 1 l = 0	**Cell 2** u = 0 m = 1 l = 0	**Cell 3** u = 0 m = 1 l = 0	Cell 3'
Cell 1'	Cell 1'	Cell 2'	Cell 3'	Cell 3'

Cells 1, 2, and 3 now account for the ghost cells when they distribute portions of their excess mass:

Cell 4'	Cell 4'	Cell 5'	Cell 6'	Cell 6'
Cell 4' $\delta' = 1/3 \longrightarrow$	**Cell 4** $\delta = \delta'$ $+1/3$ $+1/3$	**Cell 5** $\delta = 1/3$ $+1/3$ $+1/3$	**Cell 6** $\delta = \delta'$ $+1/3$ $+1/3$	Cell 6' $\longleftarrow \delta' = 1/3$
Cell 1'	**Cell 1** $e = 1$	**Cell 2** $e = 1$	**Cell 3** $e = 1$	Cell 3'
Cell 1'	Cell 1'	Cell 2'	Cell 3'	Cell 3'

Each broken cell now has three acceptor cells. Consider cell 1. Its nonzero δ is spread over cells 4′, 4, and 5. Because cell 4′ is a ghost cell, its δ, which we call δ', is ultimately transferred to the real cell 4. So, cell 4 receives one contribution from cell 1, one contribution from cell 2, and one contribution from its ghost version, cell 4′. The final values show preservation of symmetry:

Cell 4	Cell 5	Cell 6
$\mathtt{m} \leftarrow \mathtt{m} + \delta = 0$	$\mathtt{m} \leftarrow \mathtt{m} + \delta = 0$	$\mathtt{m} \leftarrow \mathtt{m} + \delta = 0$
Cell 1	**Cell 2**	**Cell 3**
$\mathtt{m} \leftarrow \mathtt{u} = 0$	$\mathtt{m} \leftarrow \mathtt{u} = 0$	$\mathtt{m} \leftarrow \mathtt{u} = 0$

In practice, because the neighborhood size might not be fixed, it may be desirable to use an appropriate bookkeeping scheme near the boundaries, in lieu of using ghost cells. For example, we can reflect indices at the boundaries, i.e., treat the grid as a torus.

5.3. Properties and issues

This algorithm converges, is conservative and order-independent, and can preserve a 1D symmetry with the modifications outlines above. However, on parallel machines this algorithm (as well as the local order-dependent method) is slow. This is largely due to the neighborhood expansion needed by both algorithms. To define how far we need to look for mass/room, the algorithms expand the neighborhood of a cell, but there is no way to know, *a priori*, how many expansion steps are needed. On a parallel machine, this leads to excessive communication between processors if the expansion process reaches a border between processors. If ghost cells are used, a possible solution is to create thicker layers of ghost cells (as thick as we anticipate will be needed) along processor boundaries. Another solution is described next.

6. A mixed local/global order-independent repair

The global repair algorithm that was described earlier needs very little communication and can be used very efficiently in a parallel framework. Once every cell has been repaired for every processor, a single communication is performed to give the total discrepancy Δ, and the final update of the masses is made on each processor without additional communication. However, the global algorithm can violate causality unless most of the cells are within or close to their bounds. Therefore, we consider here an amalgamation of the local algorithm, which gives more physically meaningful results, and the global algorithm, which is more parallelizable.

6.1. Algorithm

The mixed local/global order-independent repair algorithm is based on the assumption that most of the out-of-bounds cells can be fixed locally (using only the immediate neighborhood) because they are due to very small disturbances, and that, as a corollary, only a few cells need to find room/mass far away from their location.

The idea of this algorithm is to repair as many cells as possible with the local order-independent algorithm, and then if some of the cells are still out-of-bounds, to repair them with the global repair algorithm. That is, this method consists of the following two steps:

(1) *Local treatment*. For all out-of-bounds cells, try to repair with the Local order-independent symmetry-preserving algorithm, but without expanding any immediate neighborhoods. For example, if cell i has Δ units of excess mass, then check if some of its mass can be spread out in the $|N(i)|$ *closest* cells (recall that $|N(i)|$ is the number of cells in $N(i)$). If $0 < n \leqslant |N(i)|$ cells can accept A unit of mass in total, then give $\Delta a(j)/A$ to each of these cells, where $a(j)$ is the mass that can be accepted by cell j. If cell i still has excess mass, then leave it out of bounds. Iterate this process in order to converge to a situation where either every cell is repaired, or the remaining unrepaired cells cannot be repaired using their closest neighborhood. As in the local order-independent repair method, we try to correct the upper bounds first and then the lower bounds, or vice versa.
(2) *Global treatment*. For any remaining out-of-bounds cells, perform the global repair.

Our experimentation indicated that few iterations are needed to perform the local treatment. The global treatment finally fixes the remaining out-of-bounds cells, the number of which is presumably small, and which should be out of bounds only by small amounts.

6.2. Properties and issues

The mixed local/global repair algorithm is conservative, because each of its steps is conservative. Moreover, both the local and global treatments are order-independent and symmetry preserving (by using the method described in 5.2), and there is no particular difficulty with parallelization because there is no indefinite neighborhood expansion.

The local neighborhood used in the first step is user-dependent and can consist of 0, 1, or more layers of the neighborhood as defined in (1). In our simulations, we used the 1-layer neighborhood.

If any cells are still out-of-bounds after the first step, the global repair fixes them. The earlier argument stating that this method can violate causality still holds, but the effect is far less pronounced because very few cells will remain to be fixed after the initial local treatment, and the amounts by which they need to be fixed will be less. Therefore, the causality violation could be negligible. In the next section, we present an example of advection where the data show that most of the cells are indeed fixed by the local treatment.

7. Application to advection

As in [4], we compute a pure advection problem in 2D with the following equation:

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} + v\frac{\partial \rho}{\partial y} = 0, \tag{16}$$

where ρ is the density and u the velocity.

Three revolutions of a cone are computed on a 100×100 quadrilateral mesh on the domain $\Omega = [0,2] \times [0,2]$ with a CENO type numerical scheme. In the continuum case, the cone is not disturbed after the rotations. In the discrete case, we can measure the impact of a given numerical scheme, including diffusion and oscillations, because after three revolutions the cone should be as close as possible to the original one. The original cone has a peak at 5 and a minimum value of 0 (see Fig. 4).

The density field ρ as a function of the radius r is defined as follows:

$$\rho(r) = \begin{cases} 20(0.2 - r) + 1, & \text{if } r \leqslant 0.2, \\ 1, & \text{if } 0.2 \leqslant r \leqslant 0.4, \\ 0, & \text{else.} \end{cases} \tag{17}$$

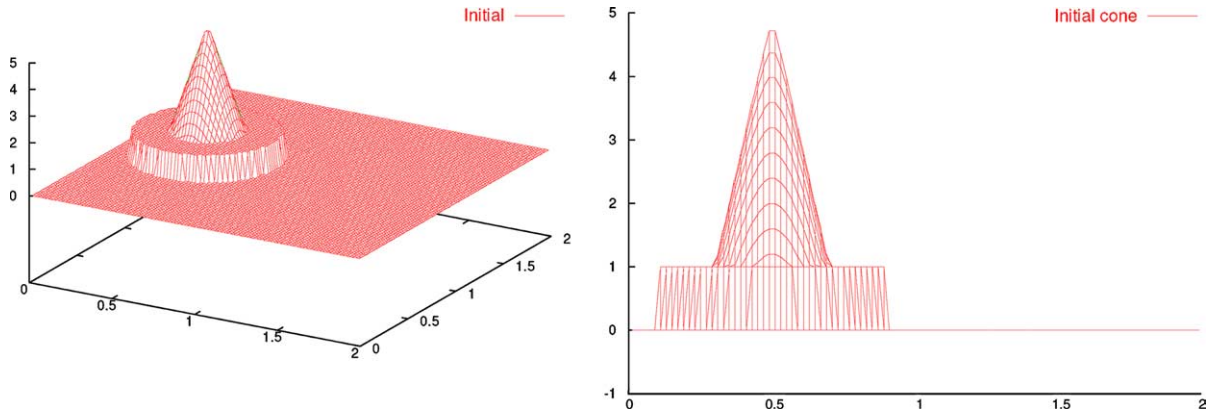

Fig. 4. Initial cone for the advection problem. This is the exact cone after three rotations in the continuum case. Left: 2D view. Right: 1D view plane x, ρ.

The center of the cone is shifted to $(0.5, 1)$, the boundary conditions are treated as walls (normal velocity equal to 0), and the velocity field is given by $u(x, y) = y - 1$, $v(x, y) = -x + 1$.

The intent at the moment is not to write a good advection scheme, but to show that a repair method can help *any* advection scheme to perform better. Indeed, in this case our advection scheme itself performs poorly on the input data because it creates unphysical oscillations that require repair.

In Fig. 5 we present the third revolution when no repair algorithm is used (top-left), and when the global (top-right), local order-independent (bottom-left), and mixed local–global (bottom right) repair methods are used. The figure is a 1D view along the x-axis. The local order-*dependent* repair method is not used because its order dependence is too serious an issue.

This problem is quite difficult for any repair method because so many cells must be repaired per iteration.

No repair. Some parasitic oscillations which generate negative values are present when no repair method is used. The maximum and minimum densities are 4.46 and −0.405, whereas the exact values are 5.0 and 0.0.

Global repair. No parasitic oscillations can be seen. The time spent to solve the problem is about 6 times larger than the time spent to solve the problem without any repair.

Order-independent and symmetry-preserving local repair. The time ratio is 7 and the maximum and minimum values are 4.12 and 0.0. No oscillations can be seen, and the shape of the cone is respected.

Mixed local/global repair. During this simulation, there are an average of 2000–2500 out-of-bounds cells per cycle. This is (∼20–25% of the total number of cells.) After the local step, the percent of cells still out-of-bounds is between 0.05% and 2%, meaning that most of the cells have been fixed locally. This was one of the assumptions to motivate the development of such a repair algorithm. The few remaining out-of-bounds cells are finally fixed by the global repair step. The time ratio is 7, and the minimum/maximum values are 0.0 and 4.16.

In Table 1 we gather the results of these methods: the time spent and the minimum/maximum density values.

This advection problem shows the ability of our repair methods to fix unphysical oscillations without destroying the shape of the cone, and to maintain reasonable accuracy at the maximum value of the cone even if the peak has been clipped. Recall that we specifically designed our advection scheme to produce very poor results; almost every cell needs repair every 4 time steps, and therefore, the repair stage constitutes a

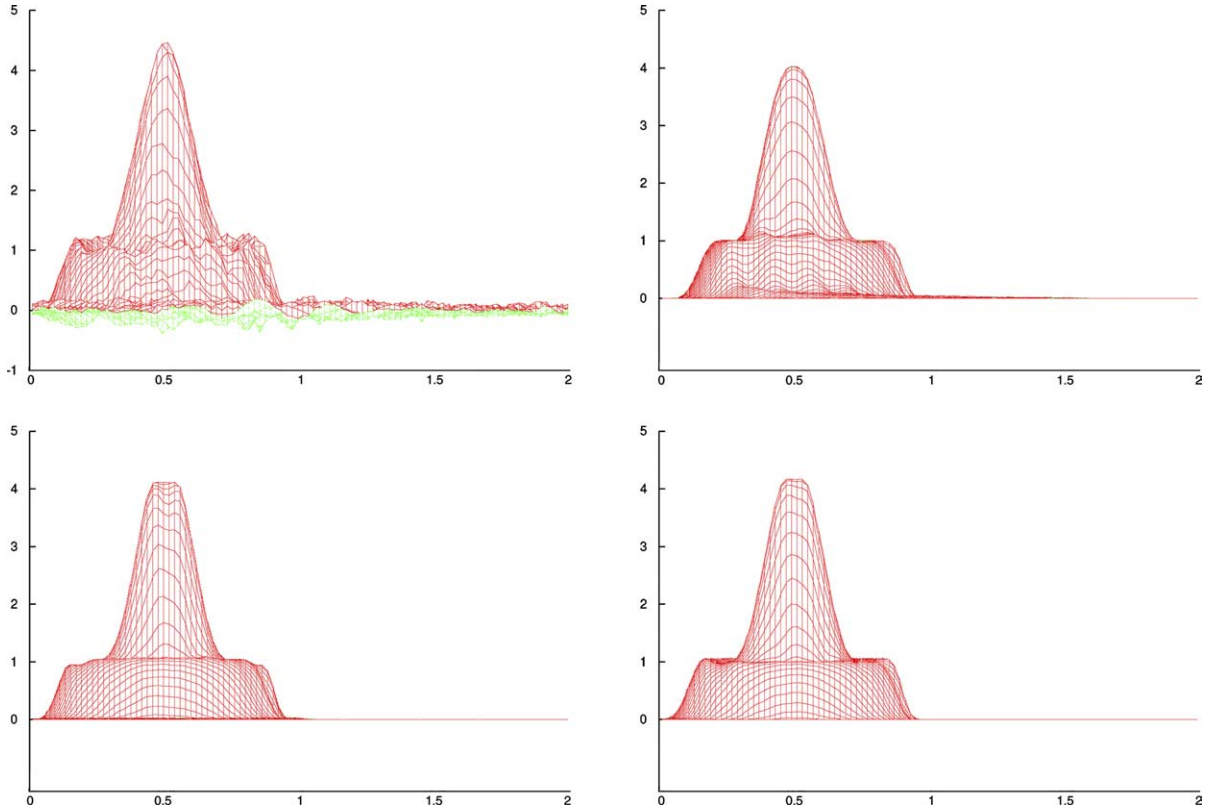

Fig. 5. Revolutions of a cone with different repair algorithms. Plane (x, ρ). Top left: without repair. Top right: with global repair. Bottom left: with order-independent local repair. Bottom right: with mixed local/global repair.

Table 1
Revolutions of a cone with four repair algorithms

Methods	Time ratio δ_T	Min/max values
No repair	1	−0.405/4.46
Global	6	0.0/4.03
Order-independent	7	0.0/4.12
Mixed local/global	7	0.0/4.16
Exact	–	0.0/5.0

Minimum and maximum values and δ_T, the ratio between the time spent to solve the problem with and without the repair method.

large percentage of the total run-time of the code. In a real application this will not be the case, and the time ratio of 1:7 will decrease dramatically. Our ratios reflect worst-case scenarios.

8. Application to hydrodynamics

We now apply the repair idea to some compressible flow problems in which remapping is required, either because the scheme is an Eulerian one of Lagrange-remap type, or because it is a Lagrangian scheme

needing rezoning. There are many ways to do the remapping, but, without going into details, we have chosen to use limited piecewise linear reconstruction on the original cells, and integration over the intersection of the old and new cells. In one dimension this integration is exact, while in two dimensions we use a quadrature as described in [6,5]. The variables remapped are mass, momentum, and total energy. Because a quadrature is used, the remapped density can be negative. Because the internal energy is obtained by subtracting the new kinetic energy from the total energy, this can negative, and frequently is.

For repair, we first repair density to satisfy its maximum principle. [5] provides a simple and easily satisfied condition, relating the bounds and the mesh, that implies the feasibility condition stated in the Introduction. We next repair the velocity components (defined as the ratio of momentum to repaired density) to satisfy their own maximum principle, maintaining conservation of momentum. A sufficient condition for the success of this step is stated in [4], and although that condition cannot be verified in this application, we have never observed a failure of velocity repair. The final step is to repair internal energy, defined as the remapped total energy minus the kinetic energy as obtained from the repaired density and velocity. It is shown in [4] that if the new total kinetic energy does not exceed the old kinetic energy, then internal energy can be repaired without violating local lower bounds, in particular, without being negative. Lower bound energy repair has not failed in our examples. In theory, local upper bounds on internal energy cannot *always* be satisfied. However, we attempt to satisfy them when possible.

8.1. Staggered grids

Staggered polygonal grids are sometimes used in ALE codes. In this scenario, fluid variables live in different places: density and specific internal energy at cell centers, and velocity at nodes. Density can also be given in subcells, where a subcell is a quadrilateral defined by joining a cell's center, one of its nodes, and the centers of the cell's edges that are linked to the node. With staggered grids, the repair algorithms require no special treatment because they make no assumption about the meaning (staggered or otherwise) of an underlying grid. Dealing with a mesh (for energy repair), a subcell mesh (for density repair), or a median mesh (for velocity repair), is irrelevant to the algorithms themselves.

8.2. Enforcement of boundary conditions

Different types of boundary conditions are used in Lagrangian numerical schemes. These include piston (nonzero velocity), wall (zero velocity), vacuum (zero pressure), and compression/expansion (nonzero pressure). Such boundary conditions (BCs) are implemented by enforcing the velocity of all the boundary nodes or by enforcing the pressure of all the boundary cells. For example, assume a nonzero boundary velocity is enforced during an ALE calculation. After the rezone and remap parts occur, velocities on the boundary nodes will have changed and will probably violate the boundary conditions. If this happens, a convenient way to re-enforce the BC is to repair the nodal velocities using upper and lower bounds equal to the required boundary velocity. Then, no momentum is lost (because the repair is conservative) and the required velocity at the boundary is satisfied.

8.3. Numerical results

We now present some numerical tests in a compressible hydrodynamics framework where the use of a repair method can improve the results.

To produce the following results we used an ALE code called ALE INC(ubator) that is designed for general polygonal grids [7]. This code is split into a Lagrangian scheme, a rezoning phase, and a remapping phase. In the remapping phase, we avoid the computation of polygon intersections because it can be expensive in 2D, and unaffordable or simply infeasible in 3D. Mostly because of this approximation, the

remapping can produce out-of-bounds values, and therefore a repair process must be used to ensure at least the positivity of density and energy. For these numerical experiments, a full repair is performed. That is, density, velocity and specific internal energy are all corrected.

The code can be used in the Eulerian regime (as Lagrange + Remap), in the Lagrangian regime, or in the ALE regime (with Rezone and Remap phases). Moreover, it can be run in 1D or 2D on general polygonal grids.

Sod Riemann problem. This is a very simple 1D Riemann problem for an ideal gas with $\gamma = 1.4$. The data are given by a left state $(\rho,u,v,p)_L = (1,0,0,1)$ and a right state $(\rho,u,v,p)_R = (0.125,0,0,0.1)$. The discontinuity is located at $X = 0.5$ on a domain $[0{:}1]\times[0{:}y_{\max}]$ where $y_{\max}$ is defined so that the initial cells are squares. Most numerical schemes produce decent results on this problem. The first run of the code is made without conservative repair; if a density or an energy is negative, we simply clip it to zero. The second run is performed with a conservative repair method for density, velocity, and specific internal energy.

The results given with and without conservative repair are plotted in Fig. 6 for a perfect quadrilateral 101×11 mesh at time $t = 0.25$ in the Eulerian regime. Clearly, the results without conservative repair fit the exact solution. Using a repair method does not break the behavior and the results are almost identical. Also, in both cases we checked that 1D symmetry is preserved.

Le Blanc Riemann problem. This is a very strong 1D shock tube; the jump in pressure being 10^9 and the jump in density 10^3. The data are given by a left state $(\rho,u,v,p)_{\mathrm{L}} = (1,0,0,\frac{2}{3}10^{-1})$ and a right state $(\rho,u,v,p)_{\mathrm{R}} = (10^{-3},0,0,\frac{2}{3}10^{-10})$ with $\gamma = 5/3$ and a discontinuity located at $X = 3$ on $[0{:}9]\times[0{:}y_{\max}]$ where $y_{\max}$ is defined so that the initial cells are squares.

Most numerical schemes produce an overshoot after the contact discontinuity in specific internal energy, and a bad approximation of the shock speed. The results given with and without conservative repair are plotted in Fig. 7.

A perfect quadrilateral mesh is used with our ALE code, which is run in the Eulerian regime (as Lagrange + Remap). Without a conservative repair method, and for a 601×3 mesh, the conservation in total energy is slightly violated $[((E_{\mathrm{final}} - E_{\mathrm{initial}})/E_{\mathrm{initial}}) \simeq 10^{-6}]$. This violation occurs because negative internal energies are created, and must be cut to 0 for the code to run properly. With any conservative repair method, on the other hand, conservation is preserved to machine error. So, in this problem, the use of a conservative repair method preserves the positivity, reduces the overshoot, and stabilizes the profile in the

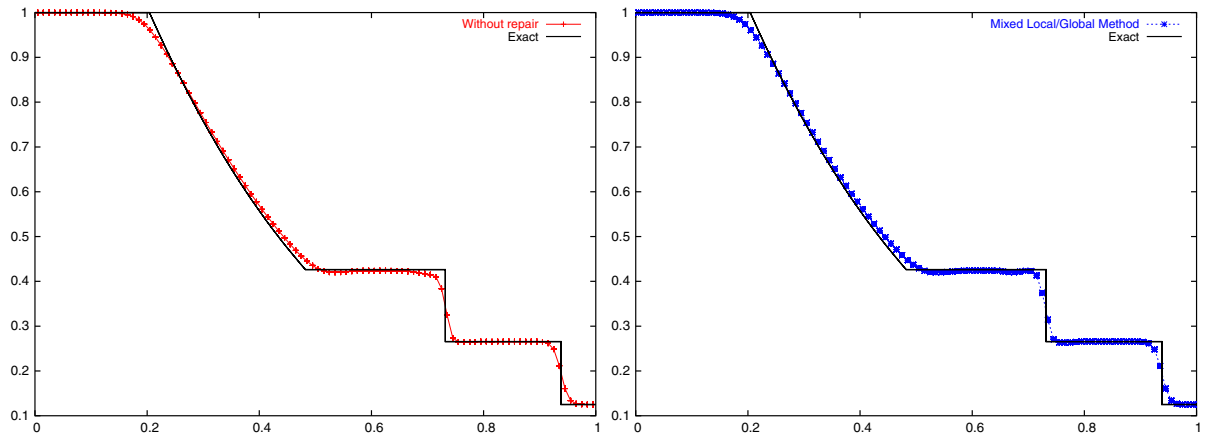

Fig. 6. Sod shock tube – density at $t = 0.25$ without (left) and with (right) a conservative repair method on a quadrilateral 101×11 mesh – Eulerian regime versus the exact solution.

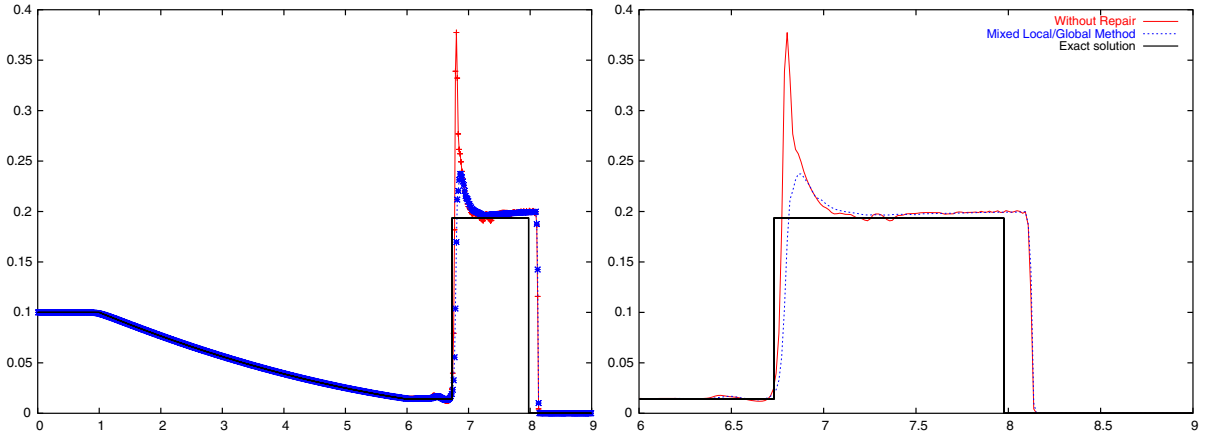

Fig. 7. Le Blanc shock tube. Specific internal energy at $t = 6.0$ with and without conservative repair, on a 601×3 mesh. Entire domain (left) and zoom (right).

plateau as shown in the zoom in Fig. 7. In this run the mixed local/global repair method is used, but such general behavior can be seen with all of our repair methods.

Interaction of blast waves. On $\Omega = [-1{:}1] \times [0{:}1]$ we initiate two half-disks of radius 0.1 centered at $X_1 = -(0.4{,}0)$ and $X_2 = (0.4{,}0)$. The perfect equation of state is used with $\gamma = 1.4$. The density is constant and equal to 1, and the velocity is zero everywhere. The internal energies are $\epsilon_1 = 30$, $\epsilon_2 = 60$ in the disks and 0 elsewhere in the domain. This high energy generates two non-symmetric cylindrical blast waves which interact. At the same time they reflect onto the walls. The final time is $t = 0.7$ and the code ran in its ALE–10 regime (rezone and remap every 10 Lagrangian cycles) on a quadrilateral mesh of 4950 nodes and 4802 cells, which has been adapted to fit the disks.

The first remappings create negative specific internal energies. Therefore, without special treatment, the code cannot go further. On the other hand, with a conservative repair method the negative energies are removed, and the conservation of mass, momentum and total energy are preserved. Our code, used with the mixed local/global repair method, produces the meshes and densities plotted in Fig. 8.

Sedov blast wave. The computational domain is one quarter of a circular disk with a radius of $r_{\max} = 1.2$. A polygonal mesh is constructed in the computational domain using a Voronoi diagram for the set of points defined as follows:

$$x_{i,j} = r_j \sin(\theta_{i,j})\ y_{i,j} = r_j \sin(\theta_{i,j}); \qquad j = 1, \ldots, J; \quad i = 1, \ldots, I(j),$$

where

$$r_j = r_{\max} \cdot \frac{j-1}{J}, \quad I(j) = \text{round}\left((j-1)\frac{\pi}{2}\right), \quad \theta_{i,j} = \frac{i-1}{I(j)} \cdot \frac{\pi}{2}, \quad J = 31$$

and the function round(x) returns the closest integer to x. According to these formulas, on each circle of radius r_j the points are distributed so that the distance between adjacent points along the circle is approximately equal to $\Delta r = r_{\max}/(J-1)$. The total number of points is 775, and there is exactly one Voronoi cell corresponding to each point. The mesh consists of a mixture of convex quadrilaterals, pentagons and hexagons, with a total of 1325 vertices; see Fig. 9. The disk is filled at $t = 0$ with an ideal gas ($\gamma = 1.4$) at rest whose density is uniformly equal to 1. The specific internal energy is zero except in the pentagonal cell c in contact with the origin, where $\epsilon(c) = E/m(c) = E/V(c)$. $V(c)$ is the volume of cell c and E is the total energy

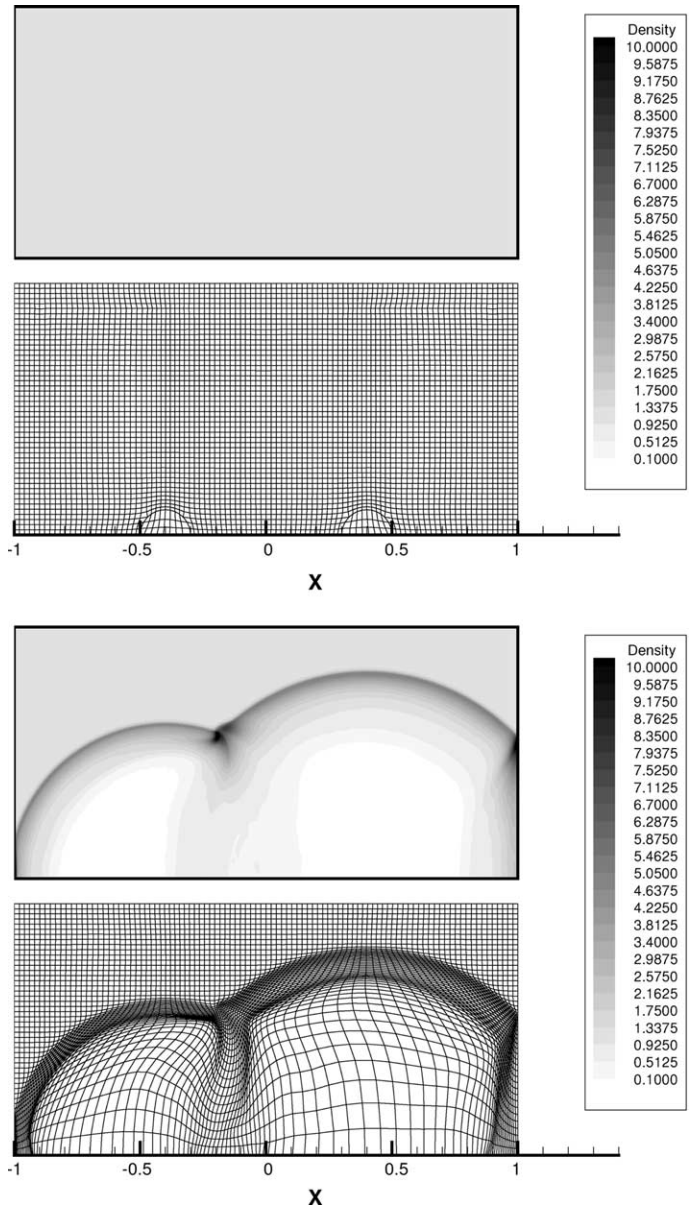

Fig. 8. Interaction of cylindrical blast waves on a quadrilateral mesh. ALE–10 regime. Mesh and density contours (exponential scale) at $t = 0.1$ and $t = 0.7$.

in the system, chosen as 0.244816. This choice of E leads to a diverging shock wave that, at $t = 1.0$, should be at radius 1.0. The peak in density should be equal to 6.

The code is used in its ALE–10 regime. When no repair is performed in this test case, the code stops due to the creation of negative internal energy after the fourth remapping. On the other hand, the use of a repair method fixes the parasitic negative values and allows us to observe good results. The maximum density with the mixed local/global repair method is 5.62.

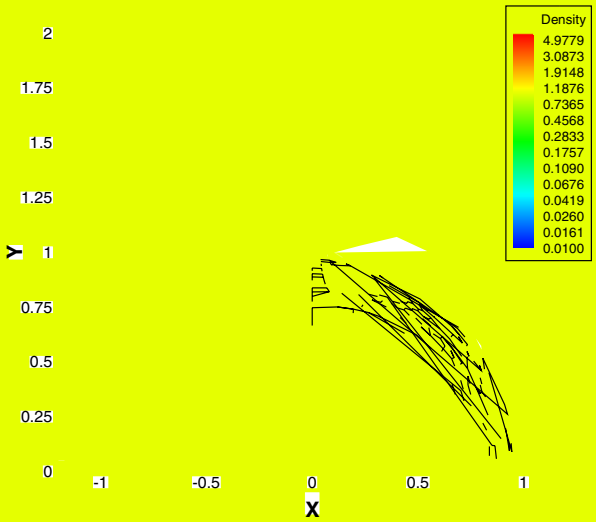

Fig. 9. Sedov blast wave on a polygonal mesh (1325 nodes and 775 cells). ALE–10 regime. Mesh and density contours (exponential scale) at $t = 0.1$ and $t = 1.0$.

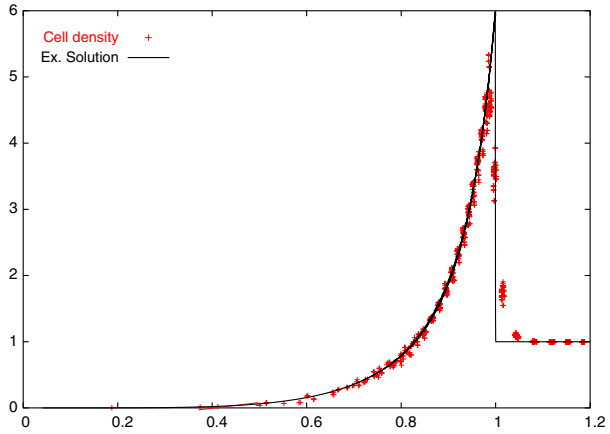

Fig. 10. Sedov blast wave on a polygonal mesh (1325 nodes and 775 cells). ALE–10 regime. Density function of the radius).

Of the methods presented here, we believe that the mixed local/global repair scheme best meets the requirements of locality and efficiency. This method was applied to an advection example and to test cases in an ALE hydrodynamics framework, where the use of a conservative repair algorithm allowed us to:

- preserve the accuracy of the underlying method, as in the Sod Riemann problem;
- stabilize and improve bad profiles, as in the Le Blanc Riemann problem;
- maintain physical and reasonable results, as in the blast wave interaction problem and the Sedov problem.

Acknowledgements

The authors thank M. Shashkov for fruitful discussions. 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). The authors also acknowledge the partial support of DOE's Advanced Simulation and Computing (ASC) program.

References

[1] P. Colella, Multidimensional upwind methods for hyperbolic conservation laws, Journal of Computational Physics 87 (1990) 171–200.
[2] J.K. Dukowicz, J.R. Baumgardner, Incremental remapping as a transport/advection algorithm, Journal of Computational Physics 160 (2000) 318–335.
[3] A. Oliveira, A.B. Fortunato, Toward an oscillation-free, mass conservative, Eulerian–Lagrangian transport model, Journal of Computational Physics 183 (2002) 142–164.
[4] M. Shashkov, B. Wendroff, The repair paradigm and application to conservation laws, Journal of Computational Physics 198 (2004) 265–277.
[5] M. Kucharich, M. Shashkov, B. Wendroff, An efficient linearity-and-bound-preserving remapping method, Journal of Computational Physics 188 (2003) 462–471.
[6] L.G. Margolin, M. Shashkov, Second-order sign-preserving conservative interpolation (remapping) on general grid, Journal of Computational Physics 184 (2003) 266–298.
[7] R. Loubere, M.J. Shashkov. A subcell remapping method on staggered polygonal grids for Arbitrary–Lagrangian–Eulerian methods. Journal of Computational Physics (2004) (accepted).