J Sci Comput (2016) 66:116–140
DOI 10.1007/s10915-015-0014-7

Adaptive Gradient-Augmented Level Set Method with Multiresolution Error Estimation

Dmitry Kolomenskiy[1,2] · **Jean-Christophe Nave**[2] · **Kai Schneider**[3]

Received: 22 July 2014 / Revised: 12 March 2015 / Accepted: 13 March 2015
Published online: 22 March 2015

Abstract A space–time adaptive scheme is presented for solving advection equations in two space dimensions. The gradient-augmented level set method using a semi-Lagrangian formulation with backward time integration is coupled with a point value multiresolution analysis using Hermite interpolation. Thus locally refined dyadic spatial grids are introduced which are efficiently implemented with dynamic quadtree data structures. For adaptive time integration, an embedded Runge–Kutta method is employed. The precision of the new fully adaptive method is analysed and speed up of CPU time and memory compression with respect to the uniform grid discretization are reported.

Keywords Space–time adaptivity · Gradient augmented level-set method · Hermite multiresolution · Advection equation

Mathematics Subject Classification 35L65 · 35Q35 · 65M25 · 65M50

1 Introduction

In some advection dominated problems, the solution develops small-scale features but remains smooth during time evolution. These problems can be solved efficiently using numerical methods based on high-order interpolation on fixed Eulerian grids [21,35].

This work was supported by the CRM–ISM Fellowship, NSERC Discovery and Discovery Accelerator Programs.

✉ Dmitry Kolomenskiy
dkolom@gmail.com

1 CRM, 805 Sherbrooke W. Street, Montreal, QC H3A 0B9, Canada

2 The Department of Mathematics and Statistics, McGill University, 805 Sherbrooke W. Street, Montreal, QC H3A 0B9, Canada

3 M2P2–CNRS, Aix-Marseille Université, 39, rue Frédéric Joliot-Curie, 13453 Marseille Cedex 13, France

If the small-scale features are localized in some part of the computational domain, its non-uniform partition with grid points clustered at the same part of the domain allows reducing the cost of computation without loosing accuracy. However, if the location of these features changes in time, the efficiency of the numerical method can be significantly improved by adapting the partition dynamically to the solution (see, e.g., [14] and references therein).

When applied to pure advection problems, Eulerian schemes require some stabilization which introduces numerical diffusion and thus pollutes the solution. Another drawback are small time steps imposed by the stability limit of explicitly discretized Eulerian schemes. Semi-Lagrangian schemes combine advantages of Eulerian schemes, such as connectivity of the grid, with those of Lagrangian schemes, especially that they have less demanding restrictions on the time step. A review on semi-Lagrangian schemes introduced in the context of numerical weather prediction can be found, e.g., in [37]. These schemes have also been used in the context of plasma physics for solving the Vlasov equation, e.g., [36].

In this paper, we present an adaptive method for the two-dimensional advection equation based on a semi-Lagrangian approach. Advection problems are encountered for example in moving fronts for a given velocity field, or in transport of passive scalars modeling pollution or mixing in chemical engineering [28]. It can also be viewed as a simple model that partly describes other, more complex problems, such as advection-reaction-diffusion, fluid flow, elasticity, etc. Therefore the proposed numerical method may be relevant to those problems as well.

We present a generalization of the gradient-augmented level set method [9,26,34] to adaptive discretization in space and in time. There exists a large variety of approaches to introduce adaptivity which differ in many aspects such as mesh topology, refinement criteria and data structure management. A complete review is beyond the scope of the paper and we mention exemplarily just a few. Optimal grid adaptation based on a posteriori error estimators have been proposed in the context of finite element methods and are reviewed in [3,38]. Cartesian grid methods have a long history, see, e.g., [1], and adaptive mesh refinement techniques [4–6,23,41] which are mostly based on Cartesian meshes are now well established to perform efficient simulations of engineering and science problems governed by conservation laws on massively parallel computers. For a review on block-structured adaptive mesh refinement including implementation and application aspects we refer, e.g., to [12]. An overview on different adaptive mesh refinement schemes including also multiresolution techniques can be found in the proceedings volume [29].

In our approach, the refinement criterion uses an error estimate obtained from multiresolution analysis. Such multiresolution based methods were first developed for conservation laws by Harten [17]. Nowadays multiresolution techniques are known to yield an appropriate framework to construct fully adaptive schemes for hyperbolic conservation laws. Extensions and further developments of Harten's original approach can be found, e.g., in [8,19,33]. Recently fully adaptive multiresolution simulations of two-dimensional incompressible viscous flows have been proposed even on multicore architectures [32].

The main idea of these methods is the use of a multiresolution data representation. The decay of the detail coefficients, which describe the difference between two subsequent resolutions, yields information on local regularity of the solution. Thus the truncation error can be estimated and grids can be coarsened in regions where this error is small and the solution is smooth. Thresholding the multiresolution representation allows to introduce easily such adaptive grids where only significant coefficients are retained. Hence uniform grid computations can be accelerated considerably as the number of points can be significantly reduced, while controlling the accuracy of the discretization. Memory requirements could also be reduced if dynamic data structures are used. Possible approaches to data structure management include the use of space filling curves [11] or hash tables [7]. Our method is related to earlier work by Roussel and Schneider [33] and it uses tree data structures.

Adaptive tree codes have been used for particle methods, e.g., for computing vortex sheet roll-up [20]. Quad/octree based adaptive solvers for the time-dependent incompressible Euler equations, even coupled with volume-of-fluid techniques, have been proposed in [30,31]. In the context of level set methods, quadtree and octree data structures have been used for adaptive (non-graded) Cartesian grids [24]. Even water and smoke simulations have been performed exploiting octree data structures and mesh refinement [22]. Reviews on different multiresolution methods can be found, e.g., in the books of Cohen [10] and Müller [25] or in the overview article by Domingues et al. [15].

The paper is organized as follows. Section 2.1 describes the gradient-augmented level set method. Section 2.2 briefly presents the multiresolution analysis. Section 2.3 introduces the tree data structure. Section 2.4 discusses adaptive time stepping techniques. Section 2.5 summarizes the algorithm. Numerical validation and performance tests are presented in Sects. 3.1–3.5. Section 4 draws some conclusions and presents possible perspectives for future work.

2 Problem Definition and Description of the Method

2.1 Gradient-Augmented Level-Set Method for Advection Problems

The gradient-augmented level set method [9,26,34] is an efficient tool for numerical solution of advection problems. In this work, we consider the linear advection equation

$$u_t + \boldsymbol{a} \cdot \nabla u = 0, \quad \text{for } t > 0,\ x \in \Omega \subset \mathbb{R}^2, \tag{1}$$

with suitable initial and boundary conditions. In (1), $u(\boldsymbol{x}, t)$ is a scalar valued function, $\boldsymbol{a}(\boldsymbol{x}, t) = (a_1, a_2)$ is a velocity field, $\boldsymbol{x} = (x_1, x_2)$ is the position vector and t is time. In the present paper we only discuss two-dimensional problems, but it is straightforward to generalize the numerical method to three dimensions.

In practice, the method can be used to solve problems that have nonsmooth solutions. In the context of closely related Hermite methods, this class of problems was treated in [2]. However, here we assume that $u(\boldsymbol{x}, t)$ is smooth, to avoid additional complications.

The numerical method consists in solving an augmented system of equations. In addition to the level-set function u, its partial derivatives u_{x_1}, u_{x_2} and $u_{x_1 x_2}$ are also evolved. The corresponding evolution equations are obtained by differentiating (1). All quantities are stored at discrete adaptive grid points $\{\boldsymbol{x}_j\}_{j=1,J}$. An example of discretization grid is shown in Fig. 1. In this paper, we assume that the computational domain Ω in space is a unit square. The results may be easily rescaled to smaller or larger domains. Adaptivity and remeshing techniques are discussed in Sects. 2.2 and 2.3. However, in the current section, it is assumed that the discretization grid $\{\boldsymbol{x}_j\}_{j=1,J}$ is known.

We use a semi-Lagrangian approach [34]. Evolution of u and its derivatives along the characteristic lines is described by a system of ODEs,

$$\frac{\mathrm{d}\boldsymbol{X}}{\mathrm{d}t} = \boldsymbol{a}(\boldsymbol{X}, t) \tag{2}$$

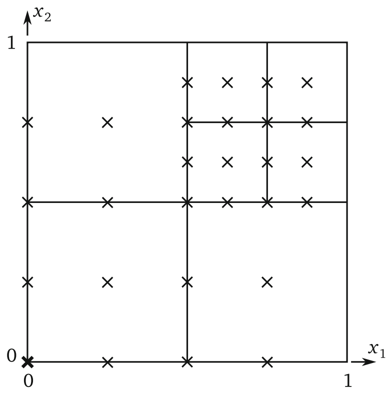

Fig. 1 Example of a discretization grid. Markers × show grid points $\boldsymbol{x}_j$

with appropriate initial conditions. At each time step, we consider points $\boldsymbol{x}_j^{\mathrm{sw}} = \boldsymbol{x}_j + \eta \boldsymbol{d}^{\mathrm{sw}}$, $\boldsymbol{x}_j^{\mathrm{se}} = \boldsymbol{x}_j + \eta \boldsymbol{d}^{\mathrm{se}}$, $\boldsymbol{x}_j^{\mathrm{nw}} = \boldsymbol{x}_j + \eta \boldsymbol{d}^{\mathrm{nw}}$ and $\boldsymbol{x}_j^{\mathrm{ne}} = \boldsymbol{x}_j + \eta \boldsymbol{d}^{\mathrm{ne}}$, where $\boldsymbol{d}^{\mathrm{sw}} = (-1,-1)$, $\boldsymbol{d}^{\mathrm{se}} = (1,-1)$, $\boldsymbol{d}^{\mathrm{nw}} = (-1,1)$, $\boldsymbol{d}^{\mathrm{ne}} = (1,1)$, and η is a small number compared to the minimum grid step size. We set $\eta = 10^{-7}$ in the examples shown in this paper, unless stated otherwise. The effect of varying η is discussed in Sect. 3.4. Characteristics that go through grid points $\boldsymbol{x}_j^{\mathrm{sw}}$, $\boldsymbol{x}_j^{\mathrm{se}}$, $\boldsymbol{x}_j^{\mathrm{nw}}$ and $\boldsymbol{x}_j^{\mathrm{ne}}$ are traced backwards in time from t_{n+1} to $t_n = t_{n+1} - \Delta t$. For every j, the solution $\boldsymbol{X}_j^{\mathrm{ft\ sw}} = \boldsymbol{X}(t_n)$ of the final value problem

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}\tau}\boldsymbol{X}(\tau) = \boldsymbol{a}(\boldsymbol{X}(\tau), \tau) \\ \boldsymbol{X}(t_{n+1}) = \boldsymbol{x}_j^{\mathrm{sw}} \end{cases} \tag{3}$$

is obtained approximately using one step of a Runge–Kutta scheme, as will be discussed in greater detail in Sect. 2.4. Points $\boldsymbol{X}_j^{\mathrm{ft\ se}}$, $\boldsymbol{X}_j^{\mathrm{ft\ nw}}$ and $\boldsymbol{X}_j^{\mathrm{ft\ ne}}$ are found similarly. Their average is then calculated,

$$\boldsymbol{X}_j^{\mathrm{ft}} = \frac{1}{4}\left(\boldsymbol{X}_j^{\mathrm{ft\ sw}} + \boldsymbol{X}_j^{\mathrm{ft\ se}} + \boldsymbol{X}_j^{\mathrm{ft\ nw}} + \boldsymbol{X}_j^{\mathrm{ft\ ne}}\right). \tag{4}$$

Values of $u(\boldsymbol{X}_j^{\mathrm{ft\ sw}}, t_n)$, $u(\boldsymbol{X}_j^{\mathrm{ft\ se}}, t_n)$, $u(\boldsymbol{X}_j^{\mathrm{ft\ nw}}, t_n)$ and $u(\boldsymbol{X}_j^{\mathrm{ft\ ne}}, t_n)$ are calculated by Hermite interpolation using the known grid-point values (see "Appendix"). It is important that the same interpolant is used for calculating all of these four points, as was pointed out in [34]. We use the interpolant defined by the cell that contains $\boldsymbol{X}_j^{\mathrm{ft}}$.

Then the grid-point values of u at time t_{n+1} are obtained by averaging, and the corresponding derivatives are calculated using second-order finite-difference approximations [34],

$$\begin{aligned}
u(\boldsymbol{x}_j, t_{n+1}) &= \frac{1}{4}\left(u(\boldsymbol{X}_j^{\mathrm{ft\ sw}}, t_n) + u(\boldsymbol{X}_j^{\mathrm{ft\ se}}, t_n) + u(\boldsymbol{X}_j^{\mathrm{ft\ nw}}, t_n) + u(\boldsymbol{X}_j^{\mathrm{ft\ ne}}, t_n)\right), \\
u_{x_1}(\boldsymbol{x}_j, t_{n+1}) &= \frac{1}{4\eta}\left(-u(\boldsymbol{X}_j^{\mathrm{ft\ sw}}, t_n) + u(\boldsymbol{X}_j^{\mathrm{ft\ se}}, t_n) - u(\boldsymbol{X}_j^{\mathrm{ft\ nw}}, t_n) + u(\boldsymbol{X}_j^{\mathrm{ft\ ne}}, t_n)\right), \\
u_{x_2}(\boldsymbol{x}_j, t_{n+1}) &= \frac{1}{4\eta}\left(-u(\boldsymbol{X}_j^{\mathrm{ft\ sw}}, t_n) - u(\boldsymbol{X}_j^{\mathrm{ft\ se}}, t_n) + u(\boldsymbol{X}_j^{\mathrm{ft\ nw}}, t_n) + u(\boldsymbol{X}_j^{\mathrm{ft\ ne}}, t_n)\right), \\
u_{x_1 x_2}(\boldsymbol{x}_j, t_{n+1}) &= \frac{1}{4\eta^2}\left(u(\boldsymbol{X}_j^{\mathrm{ft\ sw}}, t_n) - u(\boldsymbol{X}_j^{\mathrm{ft\ se}}, t_n) - u(\boldsymbol{X}_j^{\mathrm{ft\ nw}}, t_n) + u(\boldsymbol{X}_j^{\mathrm{ft\ ne}}, t_n)\right).
\end{aligned} \tag{5}$$

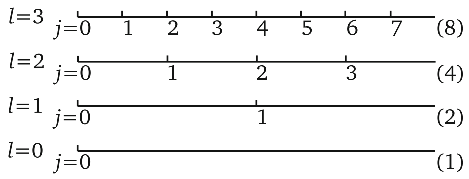

Fig. 2 One-dimensional nested grids. In this example, $m = 0$ and $M = 3$. Note that, by periodicity, the right end point of the interval is identical to the left end point

We assume the initial condition being prescribed analytically and being sufficiently regular. Therefore, we have access to the exact values of $u(\boldsymbol{x}_j, t_0)$, $u_{x_1}(\boldsymbol{x}_j, t_0)$, $u_{x_2}(\boldsymbol{x}_j, t_0)$ and $u_{x_1x_2}(\boldsymbol{x}_j, t_0)$ required for startup.

In this work we only consider u and $\boldsymbol{a}$ periodic in space. Implementation of Dirichlet or Neumann boundary conditions is less straightforward, but possible, as discussed in [26].

2.2 Multiresolution Analysis for Error Estimate

To obtain an error estimate required for mesh adaptation, we use discrete multiresolution analysis. Interpolatory multiresolution analysis based on Hermite interpolation was studied by Warming and Beam [40]. It is consistent with our numerical method as the gradient information is available. Hence, advecting the function values and its derivatives requires error control for both quantities. Fortunately, the computational overhead due to the error estimate is small, since the mid-point interpolation formulae are much simpler than interpolation at an arbitrary point, see "Appendix".

For introduction, let us first consider a one-dimensional multiresolution transform of data sampled on a uniform grid consisting of 2^M points. Let $\{x^l_j\}_{j=0,2^l}$ be a nested sequence of uniform dyadic grids on the unit interval [0, 1], such that

$$x^l_j = jh_l, \quad h_l = 1/2^l \tag{6}$$

and $l = m, m+1, \ldots, M$, where m is the coarsest and M is the finest level index. An example is shown in Fig. 2. It follows that the grid at level $l-1$ is formed from the grid at level l by removing grid points with odd indices:

$$x^{l-1}_j = x^l_{2j}, \quad j = 0, 1, 2, \ldots, 2^{l-1}. \tag{7}$$

Let

$$u^M_j = u(x^M_j), \quad u'^M_j = u'(x^M_j), \quad j = 0, \ldots, 2^M, \tag{8}$$

be the finest-grid point values of a scalar function and its derivative. It is convenient to scale the derivative by defining

$$v^M_j = h_M u'^M_j. \tag{9}$$

Note that, by periodicity, $u^l_{2^l} = u^l_0$ and $v^l_{2^l} = v^l_0$ for all l. Scaling is also required for stability of the multiresolution transform, when $M - m$ is large, but in our present work this is not a constraint.

From the point values, the multiresolution transform calculates coarsest-level values of u and v and their details at all levels,

$$\begin{aligned}&\left\{u^m_0, \ldots, u^m_{2^m-1}; r^m_0, \ldots, r^m_{2^m-1}; r^{m+1}_0, \ldots, r^{m+1}_{2^{m+1}-1}; r^{M-1}_0, \ldots, r^{M-1}_{2^{M-1}-1}\right\},\\ &\left\{v^m_0, \ldots, v^m_{2^m-1}; s^m_0, \ldots, s^m_{2^m-1}; s^{m+1}_0, \ldots, s^{m+1}_{2^{m+1}-1}; s^{M-1}_0, \ldots, s^{M-1}_{2^{M-1}-1}\right\}.\end{aligned} \tag{10}$$

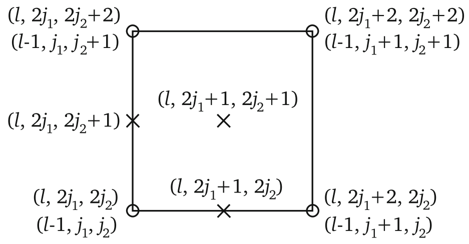

Fig. 3 Discretization grid cell at $(l-1, j_1, j_2)$. Markers $\times$ denote the 3 points where values of the function u and its derivatives are stored and residuals are computed. Markers $\circ$ denote the corner points that are used for interpolation

It is straightforward to project point values and derivative values at even-numbered grid points from level l to $l-1$,

$$u_j^{l-1} = u_{2j}^l, \quad v_j^{l-1} = 2v_{2j}^l, \quad j = 0, 2, \ldots, 2^{l-1} - 1. \tag{11}$$

The details at level $l-1$ are calculated as the difference between the exact and the interpolated values at odd points at level l,

$$r_j^{l-1} = u_{2j+1}^l - \tilde{u}_{2j+1}^l, \quad s_j^{l-1} = v_{2j+1}^l - \tilde{v}_{2j+1}^l, \tag{12}$$

where $\tilde{u}_{2j+1}^l$ and $\tilde{v}_{2j+1}^l$ are calculated by Hermite interpolation,

$$\begin{aligned} \tilde{u}_{2j+1}^l &= \frac{1}{2}\left(u_{2j}^l + u_{2j+2}^l\right) + \frac{1}{4}\left(v_{2j}^l - v_{2j+2}^l\right), \\ \tilde{v}_{2j+1}^l &= -\frac{3}{4}\left(u_{2j}^l - u_{2j+2}^l\right) - \frac{1}{4}\left(v_{2j}^l + v_{2j+2}^l\right). \end{aligned} \tag{13}$$

The advantage of the multiresolution representation is that many details are small or even zero in regions in which the function u is smooth. Thus high data compression ratios can be obtained for functions with inhomogeneous regularity, i.e., their Besov regularity is larger than their Sobolev regularity [13].

In [40], only a one-dimensional multiresolution transform was considered. We now discuss the two-dimensional case. We use Cartesian coordinates $\boldsymbol{x} = (x_1, x_2)$. Let $\{\boldsymbol{x}_{j_1,j_2}^l\}_{j_1=0,2^l; j_2=0,2^l}$, be a uniform dyadic grid on $[0,1] \times [0,1]$. This grid consists of points

$$\boldsymbol{x}_{j_1,j_2}^l = (j_1 h_l, j_2 h_l), \quad h_l = 1/2^l. \tag{14}$$

We consider a nested sequence of such grids that correspond to levels $l = m, m+1, \ldots, M$. It is assumed that values of the function and its scaled partial derivatives are given on the finest grid,

$$\begin{aligned} (u_0)_{j_1,j_2}^M &= u(\boldsymbol{x}_{j_1,j_2}^M), \\ (u_1)_{j_1,j_2}^M &= h_M \frac{\partial u}{\partial x_1}(\boldsymbol{x}_{j_1,j_2}^M), \quad (u_2)_{j_1,j_2}^M = h_M \frac{\partial u}{\partial x_2}(\boldsymbol{x}_{j_1,j_2}^M), \\ (u_3)_{j_1,j_2}^M &= h_M^2 \frac{\partial^2 u}{\partial x_1 \partial x_2}(\boldsymbol{x}_{j_1,j_2}^M), \end{aligned} \tag{15}$$

where $j_1 = 0, \ldots, 2^M$ and $j_2 = 0, \ldots, 2^M$. Their values are projected to coarser levels and, at every level, horizontal, vertical and diagonal details are computed. This requires interpolation at points $(l, 2j_1+1, 2j_2)$, $(l, 2j_1, 2j_2+1)$ and $(l, 2j_1+1, 2j_2+1)$, respectively, using the coarser-grid values at points $(l-1, j_1, j_2)$, $(l-1, j_1+1, j_2)$, $(l-1, j_1, j_2+1)$ and $(l-1, j_1+1, j_2+1)$, as explained in Fig. 3.

Thus we obtain an algorithm for a two-dimensional multiresolution decomposition,

```
**for** $l = M, M-1, \dots, m+1$
  **for** $j_1 = 0, 1, \dots, 2^{l-1}-1$
    **for** $j_2 = 0, 1, \dots, 2^{l-1}-1$
      **for** $\iota = 0, 1, 2, 3$
        $(u_\iota)^{l-1}_{j_1,j_2} = \alpha_\iota (u_\iota)^l_{2j_1,2j_2}$
        $(r^1_\iota)^{l-1}_{j_1,j_2} = (u_\iota)^l_{2j_1+1,2j_2} - (\tilde{u}_\iota)^l_{2j_1+1,2j_2}$
        $(r^2_\iota)^{l-1}_{j_1,j_2} = (u_\iota)^l_{2j_1,2j_2+1} - (\tilde{u}_\iota)^l_{2j_1,2j_2+1}$          (16)
        $(r^3_\iota)^{l-1}_{j_1,j_2} = (u_\iota)^l_{2j_1+1,2j_2+1} - (\tilde{u}_\iota)^l_{2j_1+1,2j_2+1}$
      **end**
    **end**
  **end**
**end**
```

and for reconstruction,

```
**for** $l = m+1, m+2, \dots, M$
  **for** $j_1 = 0, 1, \dots, 2^{l-1}-1$
    **for** $j_2 = 0, 1, \dots, 2^{l-1}-1$
      **for** $\iota = 0, 1, 2, 3$
        $(u_\iota)^l_{2j_1,2j_2} = (u_\iota)^{l-1}_{j_1,j_2}/\alpha_\iota$
        $(u_\iota)^l_{2j_1+1,2j_2} = (\tilde{u}_\iota)^l_{2j_1+1,2j_2} + (r^1_\iota)^{l-1}_{j_1,j_2}$
        $(u_\iota)^l_{2j_1,2j_2+1} = (\tilde{u}_\iota)^l_{2j_1,2j_2+1} + (r^2_\iota)^{l-1}_{j_1,j_2}$          (17)
        $(u_\iota)^l_{2j_1+1,2j_2+1} = (\tilde{u}_\iota)^l_{2j_1+1,2j_2+1} + (r^3_\iota)^{l-1}_{j_1,j_2}$
      **end**
    **end**
  **end**
**end**
```

In the above, $\alpha_0 = 1, \alpha_1 = \alpha_2 = 2$ and $\alpha_3 = 4$. The two-dimensional interpolation formulae for $\tilde{u}_\iota$ are given in “Appendix”. The computational complexity of the above multiresolution transform and its inverse is linear, since each of the details is only accessed once, and the number of details scales as the number of finest-level grid points.

In the grid adaptation process, which is part of the numerical method for the advection equation proposed in this work, only the multiresolution decomposition is used. The reconstruction procedure is required to obtain the solution on a regular grid to perform the error analysis, and thus we use the algorithm (17) in our validation tests in Sect. 3.1.

The outer loop in the multiresolution transform (16) is defined from the finest possible level M down to one level above the coarsest, $m+1$. This algorithm assumes that grid-point data are available at the finest level. However, for the nonuniform adaptive grid described in the following sections, the finest level varies depending on the position (j_1, j_2). Therefore, in that case, the decomposition starts from the local finest level at the given position. Also, when multiresolution is used for grid adaptation, it is unnecessary to compute the decomposition for all levels down to $m+1$. It is only computed for two levels downwards. These technicalities are described in Sect. 2.5.

The magnitude of details $(r^i_\iota)^l_{j_1,j_2}$ decreases with level l, with a rate that depends on the regularity of the function u and on the accuracy of interpolation. This property is used for data compression. We define a truncation operator,

$$(\hat{r}_\iota^i)_{j_1,j_2}^l = \begin{cases} (r_\iota^i)_{j_1,j_2}^l & \text{if } |(r_\iota^i)_{j_1,j_2}^l| > \varepsilon_l \text{ for any } i, \iota \\ 0 & \text{if } |(r_\iota^i)_{j_1,j_2}^l| \le \varepsilon_l \text{ for every } i, \iota \end{cases} \tag{18}$$

where $0 \le \iota \le 3$ and $1 \le i \le 3$ and ε_l is a threshold defined below. As one can see from (15) and (16), the index ι denotes the quantity for which the detail is computed (0: function value; 1: first derivative with respect to x_1; 2: first derivative with respect to x_2; 3: cross derivative with respect to x_1 and x_2). The index i denotes the direction of the detail (1: horizontal; 2: vertical; 3: diagonal), depending on which of the three points indicated by the "$\times$" marker in Fig. 3 is taken to compute the detail. In other words, in the context of the tree data structure discussed in the following section, there are 12 details per cell, and a cell with its three inner points can only be discarded if all of the details are less than ε_l in magnitude.

In practice, only non-zero values of $(\hat{r}_\iota^i)_{j_1,j_2}^l$ are stored and used in the reconstruction algorithm (17) instead of $(r_\iota^i)_{j_1,j_2}^l$. Thus, the obtained approximate values of $(\hat{u}_\iota^i)_{j_1,j_2}^M$ differ from the exact values $(u_\iota^i)_{j_1,j_2}^M$ and the error depends on the choice of ε_l. In general, ε_l may depend on l. However, in our method the threshold is scale-independent,

$$\varepsilon_l = \varepsilon, \tag{19}$$

which is required to control the error in the L^∞ norm. Later on, we also relate the time discretization error control to ε, and show some numerical evidence that the local error of the method at each time step is indeed proportional to ε.

2.3 Tree Data Structure

The performance of an adaptive numerical method strongly depends on the data structures, which are necessary for memory compression. For two-dimensional multiresolution, it is natural to use quadtrees. Even though the quadtree is a classical data structure, the way it is associated with an adaptive grid depends on the underlying numerical method. In this section we briefly describe its implementation for adaptive methods based on point-value multiresolution.

We define a quadtree node to be a discretization grid cell, as shown in Fig. 4. A node in the tree is indexed by level l and position j_1, j_2. The node at level 0 is called the root of the tree. A node at level $l-1$ may have four child nodes at level l. This corresponds to a partition of a parent cell into 4 child cells. In our implementation, a node either has four or no children. A node that does not have children is referred to as a leaf. The discretization grid is formed by the leaves. We construct graded trees such that the level difference between two neighbouring leaves is not greater than one. We use an algorithm described in [39].

Vertices at cell corner points store values of u, u_{x_1}, u_{x_2} and $u_{x_1x_2}$. Thus a cell contains all information required for Hermite interpolation (see "Appendix"). Note that many cells corresponding to different nodes at different levels may point at the same vertex that they have in common. An efficient algorithm is required to store and to access the grid point values. For interpolating multiresolution with graded trees, there is a one-to-one correspondence between quadtree nodes (or cells) and grid points [18]. A cell at $(l-1, j_1, j_2)$ has three points assigned to it, as shown in Fig. 3. Their Cartesian coordinates are $((2j_1+1)h_l, 2j_2h_l)$, $(2j_1h_l, (2j_2+1)h_l)$ and $((2j_1+1)h_l, (2j_2+1)h_l)$, where $h_l = 1/2^l$, $j_1, j_2 = 0, \ldots, 2^{l-1}-1$. These points are also corners of cells at level l or higher. To access a grid point, an algorithm similar to a tree search is employed, that requires $\mathcal{O}(\log N)$ operations, where N is the number of grid points. We note a possibility of reducing

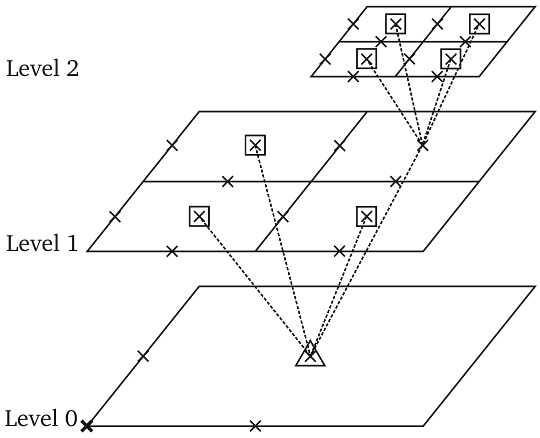

Fig. 4 Tree data structure. Every node of the tree corresponds to a Cartesian grid cell. *Squares* indicate the leaves and a *triangle* indicates the root of the tree. Markers $\times$ show grid points associated with the nodes of the tree (or cells). The bold marker $\boldsymbol{\times}$ denotes the point at the origin that is not associated with the tree

the cost of access to point data down to $\mathcal{O}(1)$ by using other kinds of data structures, such as space filling curves [11] or hash tables [7].

The list of all grid points plus the point at the origin is denoted as $\mathcal{L}_{points}$. If the tree is graded, and since the computational domain is a torus, the corner points of any cell coincide with four points from $\mathcal{L}_{points}$. Gradedness is required for several and different reasons. First, since we use nested grids obtained by cells with three interior points assigned to each cell, a non-graded tree can lead to a situation when there is no grid point at the corner of a cell. Second, after coarsening, we only add one level of refinement. Non-graded trees would require adding as many levels of refinement as the difference in level between neighbouring cells. Third, large differences in the size of neighbouring cells may have a negative effect on the stability of discretization schemes. However, for our scheme we do not have any clear evidence of this effect, and in the case of a one-dimensional advection equation the method is stable even when non-graded binary trees are employed.

2.4 Adaptive Time Stepping

The numerical method discussed in Sect. 2.1 does not have any stability restriction on the time step size Δt. However, for a given discretization in space, there exists an optimal Δt that minimizes the error. If Δt is too large, the time discretization error becomes large. If Δt is too small, the error becomes large because of accumulation of space discretization errors at every time step, such that the global error at the final time step grows in proportion to the number of time steps. This optimality condition is satisfied if the time step varies in accordance with the space grid size locally. Local time stepping is currently used in adaptive methods for PDEs (see, e.g., [14]), but its implementation is not staightforward for higher order time discretization schemes and we consider it as a possible future work.

In the adaptive time-stepping algorithms presented in this paper, the solution is evolved in time with the same Δt for all grid points. At each time iteration, after taking a step of size Δt, a new adapted value of time step size is calculated, which we denote Δt^*. We use the Dormand–Prince Runge–Kutta 4(3) method [16], that gives a local error estimate required for time step size selection. The third order error estimate of this scheme is more reliable at moderate step sizes Δt, compared to the fifth order estimate of the Runge–Kutta–Fehlberg method (see [16]). When written backwards in time, its coefficients read

$$\boldsymbol{k}_1 = -\Delta t \boldsymbol{a}(\boldsymbol{x}_j^{\mathrm{dir}}, t_{n+1})$$

$$\begin{aligned} \boldsymbol{k}_2 &= -\Delta t \boldsymbol{a}(\boldsymbol{x}_j^{\text{dir}} + \frac{1}{2}\boldsymbol{k}_1, t_{n+1} - \frac{1}{2}\Delta t) \\ \boldsymbol{k}_3 &= -\Delta t \boldsymbol{a}(\boldsymbol{x}_j^{\text{dir}} + \frac{1}{2}\boldsymbol{k}_2, t_{n+1} - \frac{1}{2}\Delta t) \\ \boldsymbol{k}_4 &= -\Delta t \boldsymbol{a}(\boldsymbol{x}_j^{\text{dir}} + \boldsymbol{k}_3, t_{n+1} - \Delta t) \end{aligned} \tag{20}$$

where $\boldsymbol{x}_j^{\text{dir}}$ are points of the finite-difference stencil introduced in Sect. 2.1, subscript 'dir' refers to a direction of shift relative to the j-th grid point ('sw', 'se', 'nw' or 'ne'). The coordinates of the corresponding Lagrangian point are calculated as

$$\boldsymbol{X}_j^{\text{ft dir}} = \boldsymbol{x}_j^{\text{dir}} + \frac{1}{6}\boldsymbol{k}_1 + \frac{1}{3}\boldsymbol{k}_2 + \frac{1}{3}\boldsymbol{k}_3 + \frac{1}{6}\boldsymbol{k}_4 \tag{21}$$

with the local truncation error $\mathcal{O}(\Delta t^5)$. This classical scheme is embedded in a five-step third-order formula that provides a reliable error estimate,

$$\boldsymbol{T}_j^{\text{dir}} = \lambda \left(\boldsymbol{k}_4 + \Delta t \boldsymbol{a}(\boldsymbol{X}_j^{\text{ft dir}}, t_{n+1} - \Delta t)\right), \tag{22}$$

where $\lambda = 1/10$, as suggested by [16]. The error estimate is averaged in the four directions,

$$\boldsymbol{T}_j = \frac{1}{4}\left(\boldsymbol{T}_j^{\text{sw}} + \boldsymbol{T}_j^{\text{se}} + \boldsymbol{T}_j^{\text{nw}} + \boldsymbol{T}_j^{\text{ne}}\right), \tag{23}$$

and used in a Taylor expansion of u that provides a local truncation error estimate to the semi-Lagrangian part of the method,

$$E_j = u_{x_1}(\boldsymbol{x}_j, t_{n+1})T_{1j} + u_{x_2}(\boldsymbol{x}_j, t_{n+1})T_{2j} + u_{x_1x_2}(\boldsymbol{x}_j, t_{n+1})T_{1j}T_{2j}, \tag{24}$$

where the derivatives of u at time t are calculated according to (5). Then the error norm is calculated, $||E||_\infty = \max_{\boldsymbol{x}_j \in \mathcal{L}_{points}} |E_j|$, where $\mathcal{L}_{points}$ is a list of all grid points.

The new adapted time step size Δt^* is

$$\Delta t^* = \min\left(1\ ,\ \Delta t \max\left(0.5\ , \min\left(2\ ,\ 0.75\left(\frac{\varepsilon}{||E||_\infty}\right)^{\frac{1}{4}}\right)\right)\right). \tag{25}$$

It is limited above by 1, its growth rate is limited by 2 and its decrease rate is limited by 0.5. The safety coefficient 0.75 helps to reduce the number of rejected time steps, as defined in the next paragraph.

If $||E||_\infty \le \varepsilon$, where ε is the threshold, the values $u(\boldsymbol{x}_j, t)$, $u_{x_1}(\boldsymbol{x}_j, t)$, $u_{x_2}(\boldsymbol{x}_j, t)$ and $u_{x_1x_2}(\boldsymbol{x}_j, t)$ are assumed being sufficiently accurate and the computation proceeds to the next time iteration with new step size $\Delta t = \Delta t^*$. Otherwise, the time step is rejected, and these quantities are recalculated using (20), (21) and (5) with step size Δt^* instead of Δt. As a result, the error estimate (24) and the new time step size (25) are computed again. The process is repeated until convergence. In general, we observe convergence within a few iterations.

The initial choice of Δt at $t = 0$ is important. It should be sufficiently small to ensure a good accuracy of error estimates based on Taylor series expansions. Thus, after building the initial grid and initializing the grid point values of the solution at $t = 0$, for each grid point we determine the nearest cell size h_j (of the four nearest cells, take the first found by tree search) and the velocity $\boldsymbol{a}_j = \boldsymbol{a}(\boldsymbol{x}_j, t)$. Then we compute an approximation to the locally optimal time step size,

$$\Delta t_j = \frac{2h_j}{\sqrt{|\boldsymbol{a}_j|^2 + 1}}. \tag{26}$$

When $|\boldsymbol{a}_j|$ is large, this approximation coincides with the local CFL condition with the Courant number equal to 2. When $|\boldsymbol{a}_j|$ is small, Δt_j is limited by twice the space step h_j. The initial time step size $\Delta t|_{t=0}$ is the minimum of these local estimates,

$$\Delta t|_{t=0} = \min_{\boldsymbol{x}_j \in \mathcal{L}_{points}} \Delta t_j. \tag{27}$$

2.5 Algorithm and Implementation Aspects

The implementation of the method is, in some aspects, similar to earlier work by Roussel and Schneider [33]. In the present work, we also use tree data structures. However, the present numerical method operates on point values rather than cell averages. Since the numerical approximation of the derivatives is based on Hermite interpolation, the same interpolant is used for the error estimate. This naturally led us to the point-value vector multiresolution [40].

The algorithm consists of a startup phase followed by time stepping.

1. *Startup.*
   - (a) Set the threshold ε, domain size L and time span T.
   - (b) Set the coarsest and the finest possible discretization levels, m and M, respectively.
   - (c) Pick a value $l_{init} \in [m, M]$. Construct a tree structure with l_{init} levels that corresponds to a uniform grid. l_{init} is the starting level for multiresolution analysis of the initial condition. Note that it may have to be greater than m. The initial uniform grid must be fine enough to be sensitive to the small-scale features of the initial condition. If both l_{init} and $l_{init} + 1$ are too coarse, the analysis will stop before capturing those small scales.
   - (d) Create a list of grid points $\mathcal{L}_{points}$.
   - (e) Evaluate the initial condition at each point in $\mathcal{L}_{points}$. It is assumed that the initial condition for u and its derivatives can be evaluated at any level with machine precision. For example, it is given analytically.
   - (f) Remesh as described below.
   - (g) Repeat steps (1d–1f) $M - l_{init}$ times.
   - (h) Set time $t = 0$.
   - (i) Initialize time step size Δt (27).

2. *Time steps.*
   - (a) Create a list of grid points $\mathcal{L}_{points}$.
   - (b) If $t + \Delta t > T$, adjust the time step size to $\Delta t = T - t$. Similar adjustment is made if it is required to evaluate the solution at a given time t_{output}.
   - (c) For all points $\boldsymbol{x}_j \in \mathcal{L}_{points}$,
     - compute $\boldsymbol{x}_j^{\mathrm{sw}}$, $\boldsymbol{x}_j^{\mathrm{se}}$, $\boldsymbol{x}_j^{\mathrm{nw}}$ and $\boldsymbol{x}_j^{\mathrm{ne}}$;
     - using Runge–Kutta integration (21), compute $\boldsymbol{X}_j^{\mathrm{ft\ sw}}$, $\boldsymbol{X}_j^{\mathrm{ft\ se}}$, $\boldsymbol{X}_j^{\mathrm{ft\ nw}}$ and $\boldsymbol{X}_j^{\mathrm{ft\ ne}}$;
     - estimate the local truncation error of time integration using (22): $\boldsymbol{T}_j^{\mathrm{sw}}$, $\boldsymbol{T}_j^{\mathrm{se}}$, $\boldsymbol{T}_j^{\mathrm{nw}}$ and $\boldsymbol{T}_j^{\mathrm{ne}}$;
     - determine $u(\boldsymbol{X}_j^{\mathrm{ft\ sw}}, t_n)$, $u(\boldsymbol{X}_j^{\mathrm{ft\ se}}, t_n)$, $u(\boldsymbol{X}_j^{\mathrm{ft\ nw}}, t_n)$ and $u(\boldsymbol{X}_j^{\mathrm{ft\ ne}}, t_n)$ by Hermite interpolation;
     - compute $u(\boldsymbol{x}_j, t + \Delta t)$, $u_{x_1}(\boldsymbol{x}_j, t + \Delta t)$, $u_{x_2}(\boldsymbol{x}_j, t + \Delta t)$ and $u_{x_1x_2}(\boldsymbol{x}_j, t + \Delta t)$ from (5) and store them in buffer variables;
     - calculate error estimate E_j (24).
   - (d) Compute $||E||_\infty$ using the local values E_j.

(e) Calculate the new time step size Δt^* given by (25).
(f) If $||E||_\infty > \varepsilon$, assign $\Delta t = \Delta t^*$ and go to step (2.5).
(g) Increment time t by Δt.
(h) Assign $\Delta t = \Delta t^*$.
(i) For all points $\boldsymbol{x}_j \in \mathcal{L}_{points}$, update the values of $u(\boldsymbol{x}_j, t)$, $u_{x_1}(\boldsymbol{x}_j, t)$, $u_{x_2}(\boldsymbol{x}_j, t)$ and $u_{x_1x_2}(\boldsymbol{x}_j, t)$ using the values stored in the buffer variables.
(j) Remesh.
(k) Repeat steps (2a–2j) until time t reaches the final value T.

The *remeshing* procedure is the same during the initial grid generation and during time iterations. It consists of the following steps.

1. Create a list of leaves of the tree structure, $\mathcal{L}_{leaves}$. This is also a list of grid cells. Note that the list $\mathcal{L}_{points}$ contains corner points of the cells in $\mathcal{L}_{leaves}$ as well as their interior points that are used for computation of the residuals (see Fig. 1).
2. For each cell in $\mathcal{L}_{leaves}$, use Hermite interpolation to estimate the values of u and its derivatives at the 3 interior points. Compute the residuals, similarly to (16).
3. Use the truncation formula (18) to determine which details can be discarded. Mark the corresponding cells for coarsening. Note that, in our implementation, coarsening implies removal of all four child cells of a parent cell. Therefore coarsening is only allowed if all four are marked.
4. Ensure that the tree nodes are marked such that, after coarsening, the tree is graded.
5. Ensure that all cells at levels $l \leq m$ are not marked and the cells at level $l = M$ are all marked for removal.
6. Coarsening: remove marked cells (i.e., marked nodes of the tree data structure).
7. Repeat steps (1–6) once again.
8. Update list of leaves $\mathcal{L}_{leaves}$.
9. Refinement: split every leaf cell into four. The uniform refinement method is, actually, the reason why two coarsening iterations are required.
10. Update list of leaves $\mathcal{L}_{leaves}$.
11. Assign new values to the interior points of all cells in $\mathcal{L}_{leaves}$ using Hermite interpolation of the corner-point values. Note that interpolation inside the leaf cells does not result in any undesirable loss of accuracy because the corresponding details are less than ε, as ensured by the thresholding rule. When handling the elements of $\mathcal{L}_{leaves}$, begin from the coarsest-level entries and end at the finest level, because interpolation at finer levels uses point values at coarser levels.

3 Numerical Results

3.1 Validation Tests of the Multiresolution Transform

Let us consider a periodic one-dimensional chirp,

$$u = \sin\left(\alpha\pi(x - 1/2)^3\right), \quad 0 \leq x \leq 1, \tag{28}$$

where α is a parameter. It is plotted in Fig. 5a for $\alpha = 256$.

The function u and its derivatives can be approximated with the desired accuracy, using only a finite number of details (i.e., grid points of an adaptive grid). We remind that the multiresolution transform is based on the interpolation formulae that have truncation error of order $\mathcal{O}(h^4)$, where $h = 2^{-l}$. Therefore, since the function (28) is smooth, the global rate

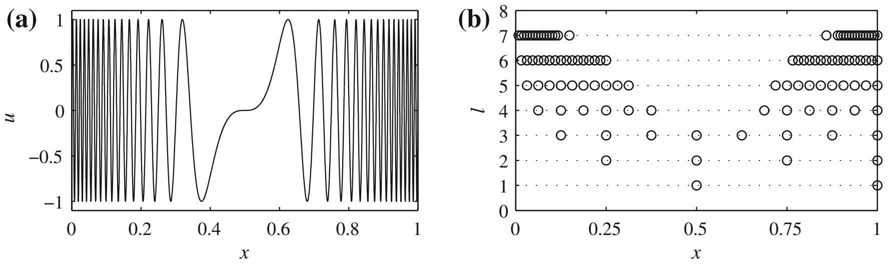

Fig. 5 **a** Plot of a periodic chirp (28) with $\alpha = 256$ and **b** a diagram of its non-zero details after thresholding (19) with $\varepsilon = 0.0664$

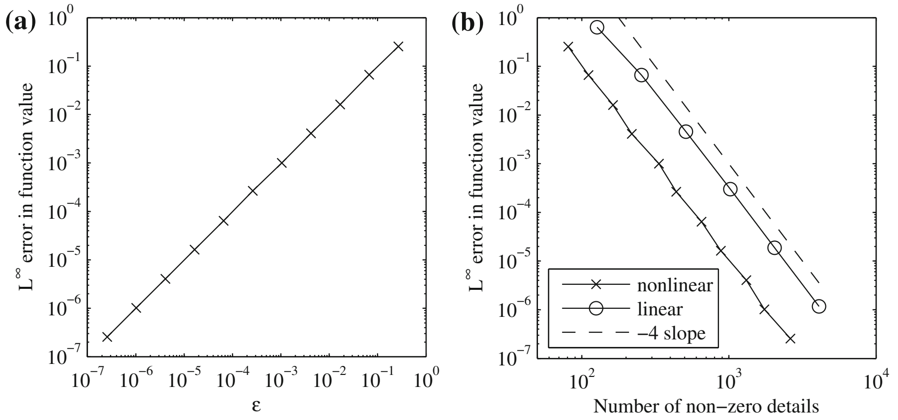

Fig. 6 **a** Decay of L^∞ error between the exact and reconstructed function values of a periodic chirp (28) with $\alpha = 256$; **b** decay of the L^∞ error versus the number of details retained by nonlinear (19) and linear filtering

of decay of its details is $\mathcal{O}(2^{-4l})$. Again, we remark that if the local regularity (measured in Besov spaces) of the function is larger than the global one (Sobolev) the detail coefficients even enjoy faster decay (see [13]). Here we compare the effect of nonlinear filtering, that is, discarding all details of magnitude less than ε, and of linear filtering, that is, discarding all details at levels greater than l_{max}. In the present case, more details are discarded with the nonlinear threshold, for a given accuracy in the L^∞ norm. This is illustrated in Fig 5b. Open circles in the diagram display level l and coordinate $x = (j+1)/2^l$ of the retained details. The threshold ε has been chosen such that the reconstructed function has the same error as if all details at 7 levels were retained. Details at $x \approx 0.5$ are smaller than those near $x = 0$ and $x = 1$ at the same level, hence they can be discarded without loss of accuracy in the L^∞ norm.

Figure 6a confirms that the nonlinear filtering error scales like ε. The error was estimated on a grid with 8192 points. The error versus the number of retained details $N_{details}$ is shown in Fig. 6b for the linear and for the nonlinear filtering. Note that, since this is a one-dimensional test, the number of details retained after the linear filtering is in inverse proportion to the finest-level grid step size. The figure shows that the error scales like $\mathcal{O}(N_{details}^{-4})$ in both cases. However, for the same value of $N_{details}$, the error of nonlinear filtering is one order of magnitude smaller than its linear counterpart.

Let us now consider a two-dimensional example. Let the function u be a Gaussian, periodized (approximately) in order to conform the boundary conditions,

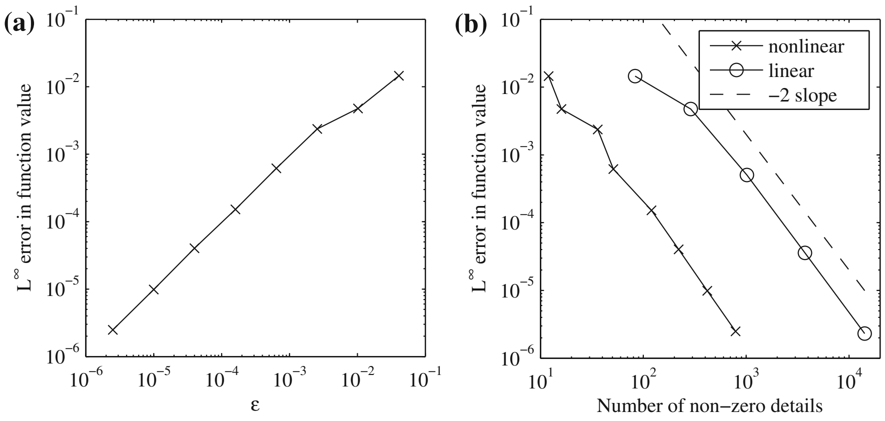

Fig. 7 **a** Decay of L^∞ error between the exact and reconstructed function values of a periodized two-dimensional Gaussian hump (29) with $\alpha = 0.05$; **b** decay of the L^∞ error versus the number of details retained by nonlinear (19) and linear filtering

$$
\begin{aligned}
u(x_1, x_2) &= \sum_{i_1=-p}^{p} \sum_{i_2=-p}^{p} g(x_1 - i_1, x_2 - i_2), \quad \text{where} \\
g(x_1, x_2) &= \exp\left(-((x_1 - 0.5)^2 + (x_2 - 0.75)^2)/r_0^2\right), \quad p = 30.
\end{aligned} \tag{29}
$$

In this example, we assign $r_0 = 0.05$. Figure 7a displays the same scaling of the nonlinear filtering error with ε as in the previous example. Figure 7b compares the linear and the nonlinear filtering. Now the number of details retained by the linear filtering is inversely proportional to the square of finest-level grid step size, therefore the error scales like $\mathcal{O}(N_{details}^{-2})$. This figure also suggests that a Gaussian with $r_0 = 0.05$ is sufficiently well localised for the nonlinear filtering to be efficient. For the same precision, it results in 10 to 20 times less non-zero details than the linear filtering.

3.2 Mixing of a Periodized Gaussian Hump

A classical benchmark for level-set methods is the deformation of a contour with an unsteady velocity field. Here we revisit the swirl test described in [34]. In this section, we only discuss one example of an adaptive computation. More detailed convergence and performance tests are presented in the following sections.

The initial condition $u_0(x_1, x_2)$ is a periodized Gaussian (29) with $r_0^2 = 0.1$. The velocity field is given by

$$
\boldsymbol{a}(x_1, x_2, t) = \cos\left(\frac{\pi t}{t_a}\right) \begin{pmatrix} \sin^2(\pi x_1) \sin(2\pi x_2) \\ -\sin(2\pi x_1) \sin^2(\pi x_2) \end{pmatrix}, \tag{30}
$$

where $t_a = 10$. Note that this field is divergence-free, i.e., $\nabla \cdot \boldsymbol{a} = 0$. The computational domain in space is a unit square and the time span is $T = 10$. A computation has been carried out with threshold $\varepsilon = 5 \cdot 10^{-3}$.

Figure 8 displays deformation of isocontours $u_c(r) = \exp(-10r^2)$ where $r = 0.1$, 0.15 and 0.2, at time instants $t = 0$, 5 and 10. The initial almost circular contours wrap around the centre of the domain. Maximum deformation occurs at $t = 5$. Then the flow direction is reversed and the contours unwrap and restore their initial shape. Thus the exact solution of

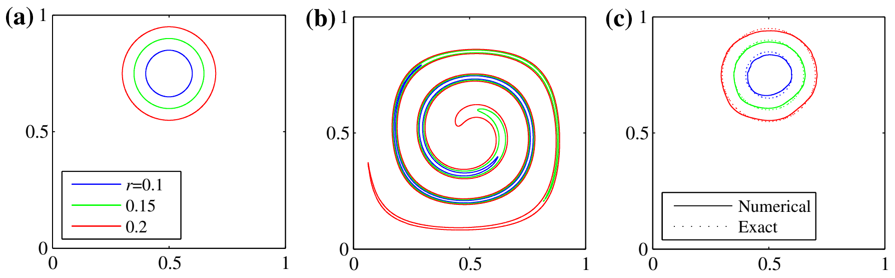

Fig. 8 Swirl test. Snapshots of isocontours $u = u_c(r)$ at time instants **a** $t = 0$, **b** $t = 5$ and **c** $t = 10$. *Continuous lines* correspond to the numerical solution, *dotted lines* show the exact solution at $t = 10$. *Red*, *green* and *blue* are isolines $r = 0.1, 0.15$ and 0.2, respectively (Color figure online)

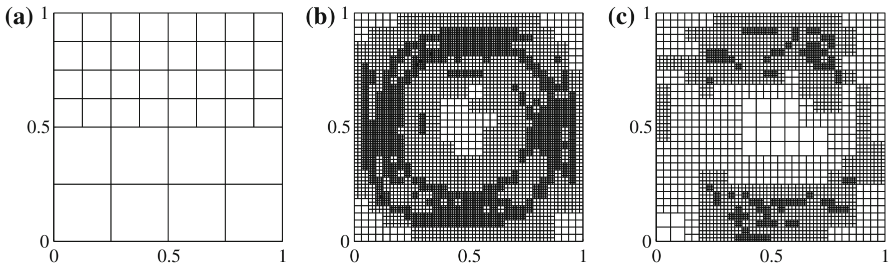

Fig. 9 Swirl test. Adaptive mesh at time instants **a** $t = 0$, **b** $t = 5$ and **c** $t = 10$

the equation at $t = 10$ coincides with the initial condition. The numerical solution slightly differs from it. The L^∞ error in the function value is equal to 0.044. The shape of the contours changes noticeably due to numerical dispersion. The area enclosed by contours $r = 0.2$ and 0.15 changes very little. However, the method does not perfectly conserve the area (it should be conserved by the exact solution, since $\nabla \cdot \boldsymbol{a} = 0$); this is seen in the decreased area of contour $r = 0.1$, which is nearer to the maximum point.

The adaptive mesh at the corresponding time instants is shown in Fig. 9. Multiresolution analysis of the initial condition produces a grid refined at the hump. By the time $t = 5$, the grid is refined everywhere in the domain. The most refinement occurs in an annulus around the centre of the domain, where deformation is the strongest. Note that Fig. 8 only displays three isocontours, but the solution is defined in the whole domain and small scale features appear at multiple locations (see Fig. 10). These locations are tracked by the mesh refinement algorithm. At $t = 10$ the grid coarsens. However it is finer than the original one at $t = 0$, and non-uniform. Because of low diffusion and some dispersion of the scheme, the numerical solution accumulates spurious small scales. Consequently, the grid has to be refined to capture these small features. The finest cells in this computation correspond to level $l = 8$, whereas the maximum allowed level is $M = 11$, at which the solution plotted in Fig. 8 is sampled. Note that the details coefficients at levels $M + 1$ and higher cannot be stored and they are assumed to be zero. Therefore, if M is fixed, the filtering becomes linear as $\varepsilon \to 0$.

Figure 11a displays the time evolution of the number of grid points over time t. The number of grid points N_{points} is related to the number of quadtree nodes N_{nodes}:

$$N_{points} = 3N_{nodes} + 1. \tag{31}$$

It varies between 160 and 31,024 in this simulation. At the final step, the number of grid points equals 17,104. The total number of time steps is equal to 52.

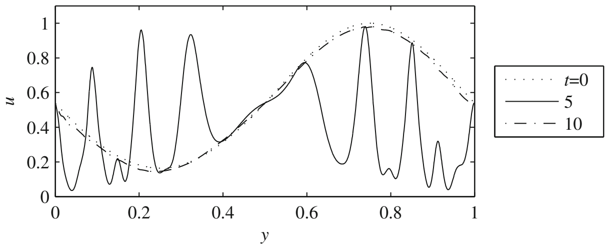

Fig. 10 Swirl test. The numerical solution u sampled at the vertical line $x = 0.5$ at time instants $t = 0$ (initial condition), $t = 5$ and $t = 10$

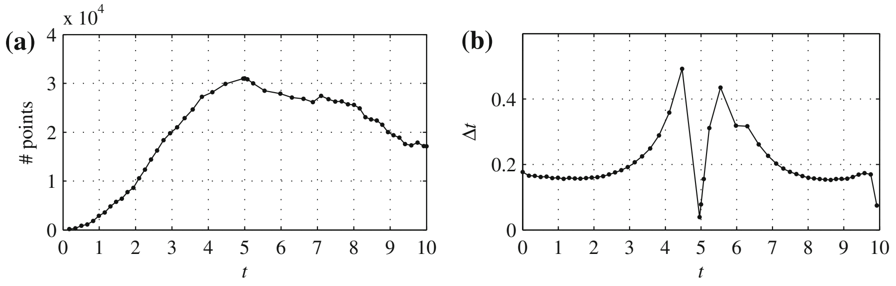

Fig. 11 Swirl test. **a** Number of grid points and **b** time step size Δt versus time t. Markers indicate discrete time t_n

The evolution of time step size Δt is shown in Fig. 11b. During the first quarter-time, it is almost constant. Then it increases as the velocity goes to zero. However, at the time instant preceding $t = 5$ (which is automatically detected by the algorithm), it is necessary to correct Δt in order to produce the isolines shown in Fig. 8b. Then Δt grows geometrically, because of a limiter in (25), until it restores to about the same size as before $t = 5$. Later, it decreases as the velocity increases, and returns to about the same size as in the beginning of the computation. Finally, one time step before the end of computation, Δt is again corrected to produce output at exactly $t = 10$.

3.3 Convergence Test on Constant Non-uniform Grids

Let us consider the non-uniform grid shown in Fig. 12, which has three levels only. It is obtained from multiresolution analysis of a Gaussian function centered in the domain. A finer grid can be constructed by splitting every cell into four finer cells. By repeating this process K times, a series of K grids is obtained. Each of them has the same difference between the finest and the coarsest levels: it is equal to 2.

A series of computations on six different grids has been carried out, with initial condition

$$u_0(x_1, x_2) = \cos(2\pi x_1)\cos(4\pi x_2) \tag{32}$$

and the velocity field given by (30) with $t_a = 1$. We recall that the exact solution of the problem coincides with the initial condition at time instants equal to multiples of t_a. Again, the adaptive time stepping method described in Sect. 2.4 has been employed.

The L^∞ error between the numerical solution at $t = 1$ and the exact solution has been calculated and plotted versus the maximum grid step size in each computation, h_{max}. The result is displayed in Fig. 13. For intermediate values of h_{max}, the convergence is slightly

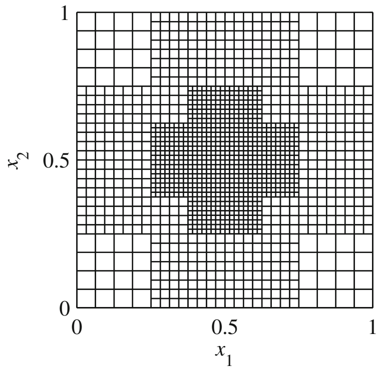

Fig. 12 Non-uniform grid used for convergence tests. By refining it uniformly, a series of grids was obtained

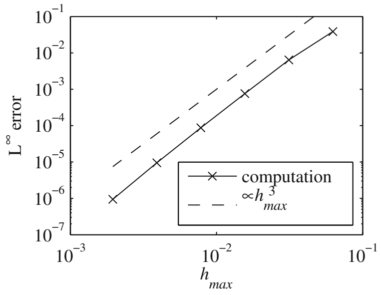

Fig. 13 Convergence test on constant non-uniform grids. Decay of the L^∞ error versus h_{max}

faster than $\mathcal{O}(h^3_{max})$ because the time discretization error decays faster and because of the adaptive time stepping. Overall, these tests indicate the third-order rate of global convergence. This is consistent with the $\mathcal{O}(h^4)$ local truncation error of the spatial discretization scheme which yields a $\mathcal{O}(h^3)$ rate of decay of the global error on fixed uniform grids, as reported earlier [34].

3.4 Convergence Tests on Adaptive Grids

Convergence of the gradient-augmented level-set method on uniform non-adaptive grids was studied in [9]. In this section, we consider a series of adaptive computations with different threshold values ε, to analyze its influence. The initial condition is again given by (32) and the velocity field is given by (30) with $t_a = 1$. In the simulations presented in this section, the maximum allowed level is set to $M = 15$. Note that, in practice, all grids generated during these computations were coarser: the smallest grid step size h_{min} in the most precise computation corresponded to $l = 13$. I.e., no grid saturation occurred.

Smaller values of ε result in finer grids. Let us define h_{max} - the maximum cell size in the computation and h_{min} - the minimum cell size. The maximum and minimum are taken over all time steps. Figure 14 confirms that both h_{max} and h_{min} become smaller with decreasing ε. The rate is, however, faster for h_{min} than for h_{max}. In fact, h_{max} in these tests typically corresponds to the largest cell of the mesh at $t = 0$, which is obtained by the multiresolution analysis of the initial condition. Therefore it asymptotically scales like $\varepsilon^{1/4}$. The minimum cell size h_{min} scales like $\varepsilon^{1/3}$, maybe because of accumulation of local small-scale errors.

The accuracy has been measured in the L^∞ error norm between the solution at $t = 1$ and the initial condition, calculated over the grid-point values at the last step of each simulation.

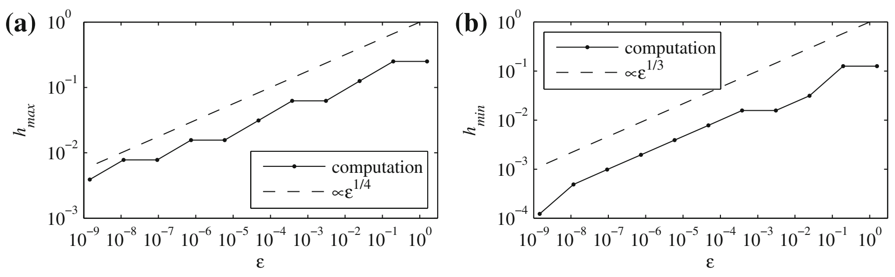

Fig. 14 Convergence tests. Decay of **a** h_{max} and **b** h_{min} versus ε

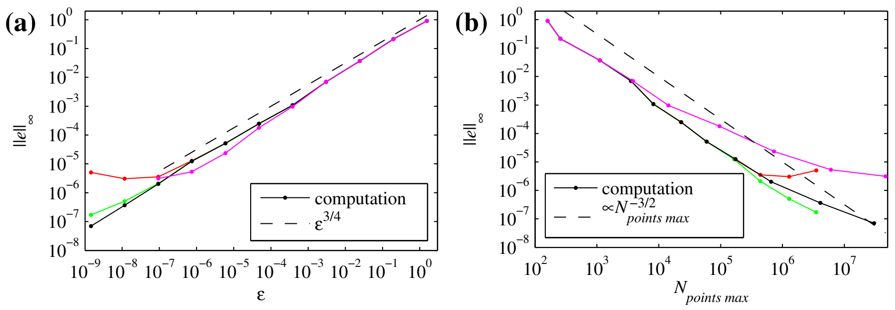

Fig. 15 Convergence tests. Decay of the L^∞ error **a** versus ε and **b** versus the maximum number of grid points $N_{points\ max}$. Different *colours* correspond to different values of η: $\eta = 10^{-5}$ (*red*), 10^{-6} (*green*), 10^{-7} (*black*), 10^{-8} (*magenta*) (Color figure online)

Figure 15a shows the decay of that error, $||e||_\infty$, versus ε. Unlike in the multiresolution transform tests in Sect. 3.1, the decay is slower than linear. This can be briefly explained as follows.

Let us consider the local error introduced for a single time step of the method, defined as $e_{loc} = u(\boldsymbol{x}, t_{n+1}) - u_{n+1}$, where $u(\boldsymbol{x}, t_{n+1})$ is the exact solution at point $\boldsymbol{x}$ and time $t_{n+1} = t_n + \Delta t$, and u_{n+1} is the numerical solution obtained after a single time step starting from time t_n and using $u_n = u(\boldsymbol{x}, t_n)$ as the initial condition. By construction, the time integration of the characteristic equation introduces an error of order ε at a single time step, as explained in Sect 2.4. The interpolation in space also introduces an error of order ε. This is ensured by the nonlinear filtering of the multiresolution decomposition discussed in Sects. 2.2 and 3.1. Hence, the local error scales like

$$e_{loc} \sim \varepsilon. \tag{33}$$

The global error can be approximately estimated as $||e||_\infty \sim N_{time\ steps}||e_{loc}||_\infty$, where the number of time steps $N_{time\ steps}$ depends on the adaptive time step size Δt. The time step size control assumes that the local truncation error of the scheme is $\mathcal{O}(\Delta t^4)$, therefore $\Delta t = \mathcal{O}(\varepsilon^{1/4})$. Then $N_{time\ steps} = \mathcal{O}(\varepsilon^{-1/4})$ and we obtain

$$||e||_\infty = \mathcal{O}(\varepsilon^{3/4}). \tag{34}$$

This estimate agrees well with the numerical results, as shown in Fig. 15a. Note that the error constant is of order unity.

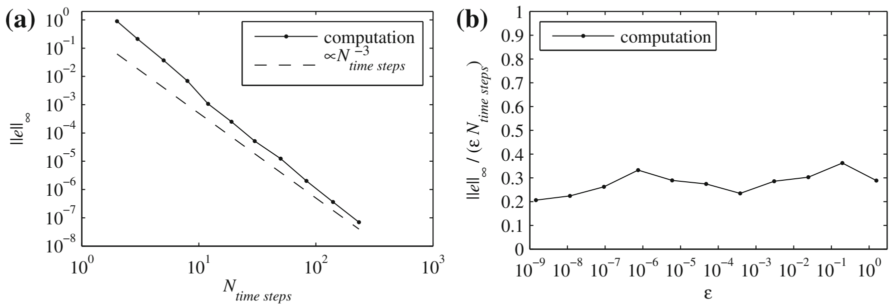

Fig. 16 Convergence tests. **a** Decay of the L^∞ error versus number of time steps $N_{time\ steps}$. **b** Ratio of the L^∞ error over its estimate $\varepsilon N_{time\ steps}$, versus ε. Time $t = 1$

Another important scaling is displayed in Fig. 15b. It depicts the decay of $||e||_\infty$ versus $N_{points\ max}$, the maximum number of grid points achieved in an adaptive simulation. For comparison, in a fixed uniform grid computation, the number of grid points does not change over time and it is proportional to $1/h^2$ (for a two-dimensional domain). Since the numerical method is globally third-order accurate, we obtain an estimate

$$||e||_\infty = \mathcal{O}(N_{points\ max}^{-3/2}) \tag{35}$$

that appears to hold not only for fixed-grid, but also for adaptive computations, if local refinement is moderate (i.e., when $h_{max}/h_{min} < 16$, in this example). Thus, the order of the discretization scheme is maintained by the adaptive method.

Further, Fig. 15 presents a test of sensitivity to the choice of finite-difference parameter η. In addition to the 'default' value $\eta = 10^{-7}$, similar computations have been carried out with $\eta = 10^{-5}$, 10^{-6} and 10^{-8}. The L^∞ error and the maximum number of grid points are both sensitive to η, i.e., when ε is small, the error due to the finite-difference approximation of $u_{x_1}, u_{x_2}, u_{x_1x_2}$ becomes dominant. In the case of $\eta = 10^{-5}$, this is essentially the truncation error. As ε decreases, $||e||_\infty$ first decreases, then saturates at $||e||_\infty \approx 3 \cdot 10^{-6}$. In the case of $\eta = 10^{-8}$, however, the round-off errors become dominant. The L^∞ error again saturates at the level of $||e||_\infty \approx 3 \cdot 10^{-6}$, and, in addition, the maximum number of grid points for a given ε increases. As discussed in [9], the round-off errors incurred by the finite-difference scheme are of magnitude $\mathcal{O}(\delta^{1/2})$, where δ is the accuracy of the floating point operations. The values $\eta = 10^{-6} \ldots 10^{-7}$ allow reaching the error as small as 10^{-7}. Therefore, all computations further in this paper have been carried out with $\eta = 10^{-7}$.

Since the finite-difference approximation is second-order accurate, and the gradient-augmented scheme is third-order accurate, it is possible to estimate the minimum allowed cell size as $h_{min\ inf} = \varkappa \eta^{2/3}$, where $\varkappa = \mathcal{O}(1)$, supposing that the error constants of both methods are of the same order of magnitude. For $\eta = 10^{-7}$ we obtain $h_{min\ inf} = 2 \cdot 10^{-5}$, which can be reached in a computation with $M = 15$ levels.

The decay rate of the L^∞ error versus number of time steps is consistent with the third-order error estimate formula used for time step size control. This is shown in Fig. 16a. Assuming that the local error at every time step is of order ε, the L^∞ error can be estimated as $\varepsilon N_{time\ steps}$. The ratio between the actual error and this estimate is plotted in Fig. 16b. It oscillates between 0.2 and 0.4 without any significant trend. This suggests that the adaptive method controls the local error as desired.

Using the scalings obtained in these convergence tests, it is possible to devise a method for computing with a given tolerance $tol \approx ||e||_\infty$ at $t = T$. First, a preliminary computation is

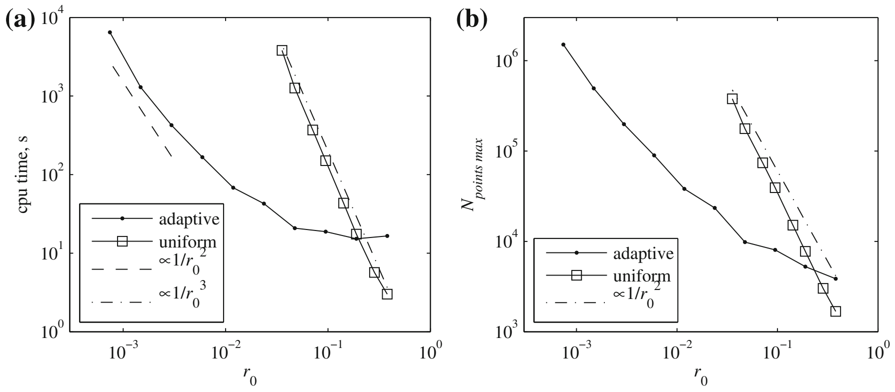

Fig. 17 Performance tests. **a** CPU time, in seconds, and **b** number of grid points $N_{points\ max}$, versus r_0

carried out with $\varepsilon = \varepsilon_1$, where ε_1 is relatively large such that the computation is reasonably fast. The number of time steps in this computation is $N_{time\ steps\ 1}$. Then the final computation is carried out with $\varepsilon = \varepsilon_2$, where

$$\varepsilon_2 = \left(\frac{tol\ \ \varepsilon_1^4}{C\ \ N_{time\ steps\ 1}} \right)^{1/5} \tag{36}$$

with $C \approx 0.3$ (see Fig. 16b).

3.5 Performance Tests

In this section, we compare computational cost of adaptive and uniform fixed-grid simulations. The initial condition is again a periodized Gaussian (29) of radius r_0. A series of adaptive simulations has been carried out with different values of r_0. The velocity field is given by (30) with $t_a = 3$, and the simulations are stopped at $t = 3$. Note that we chose $r_0 = \sqrt{0.1} \approx 0.316$ for the initial condition of the swirl test described in Sect. 3.2.

In each adaptive simulation, the threshold ε was set to an appropriate value such that the L^∞ error at $t = 3$ was equal to 0.01 ± 0.0004. The minimum possible level was set to $m = 4$. CPU time and maximum number of grid points (representative of memory usage) were measured. They are displayed in Fig. 17a, b, respectively.

For comparison, similar computations have been carried out using a different code, which implements the numerical method described in Sect. 2.1 on uniform Cartesian grids (see the original reference [34]). These results are also shown in Fig. 17.

If r_0 is large, the adaptive code is slower than the non-adaptive one. Note that, if the adaptive code is constrained to uniform space and time discretization, it is about 7 times slower than the specialized uniform-grid code. This factor partly consists of the cost of error estimation, which requires one extra level of grid refinement. If a uniform grid consists of N_{nodes} cells, the adaptive algorithm operates $3N_{nodes} + 1$ points, while the non-adaptive algorithm operates only N_{nodes} grid points. In addition, some overhead is due to the tree data structure, because it requires $\mathcal{O}(\log N_{nodes})$ operations every time a grid point value is accessed. In the present tests, the adaptive grid is, in general, not uniform. Therefore, the adaptive code is, at worst, only 5 times slower than the non-adaptive.

At $r_0 \approx 0.2$, both codes have equal efficiency. The CPU time of the non-adaptive computations scales like $1/r_0^3$ whereas adaptivity reduces it drastically. The parameter r_0 controls the size of the Gaussian hump. As it decreases, the grid has to be refined to ensure the desired accuracy. However, refinement is only necessary in a small circle of radius of order r_0. Therefore, the number of grid points in the adaptive computations depends more weakly on r_0 than in the fixed-grid computations. The number of time steps is, at worst, in inverse proportion to $h_{min} \propto r_0$. The CPU time in the adaptive computations shows two regimes. It is nearly constant if r_0 is large, and it scales like $1/r_0^2$ if r_0 is small.

In contrast, if the grid is uniform, the number of grid points required for a given accuracy behaves like $1/r_0^2$, since the grid step h should be proportional to r_0. If the time step is equal to the grid step size h, the computational complexity increases like $1/r_0^3$ as $r_0 \to 0$. Therefore, eventually, the adaptive code outperforms the non-adaptive one when r_0 is sufficiently small. We anticipate the same behaviour of this method applied to other problems that focus on time evolution of well localized features of point-singularity type.

4 Conclusions and Perspectives

A semi-Lagrangian adaptive numerical method has been developed for the two-dimensional advection equation. It is based on the gradient-augmented level set method [26]. This numerical scheme uses Hermite interpolation. It is compact, third order accurate, and unconditionally stable. Multiresolution decomposition is employed to obtain an error estimate required for adaptivity in space, while an embedded Runge–Kutta scheme is applied for the time discretization error estimate and for adapting automatically the time step. For consistency with the gradient-augmented method, the multiresolution scheme is based on Hermite interpolation [40]. Its implementation uses quadtree data structures with dynamic memory allocation.

The L^∞ error norm of the numerical solution is controlled by ε - threshold of the multiresolution scheme. Numerical experiments suggest that the error scales like $\varepsilon^{3/4}$, which we justified by heuristic arguments.

A series of numerical experiments has been carried out with advection of a Gaussian hump. The size of its support r_0 has been varied. These experiments show that, the more localized the hump is, the more beneficial the adaptive method becomes when compared to the uniform discretization approach, in terms of CPU time and memory compression.

The originality of the current work is the coupling of the gradient-augmented level set method with an adaptive multiresolution method and adaptive time stepping. This allows for speed up of CPU time and memory compression, i.e., the new adaptive solver is more efficient than the one on regular grids and in addition the errors in space and time are controlled. The order of the underlying discretization scheme on a regular grid is maintained.

It is straightforward to generalize the method to three-dimensional problems. A possible generalization to the incompressible Euler equation is of interest. There are possible applications in plasma physics, e.g., a further improvement of the particle-in-wavelet method for the Vlasov–Poisson equations [27]. Finally, we anticipate using similar techniques for solving elasticity equations.

Acknowledgments DK acknowledges financial support from the CRM–ISM Fellowship and thanks Alexey Eremin for useful discussions about Runge–Kutta schemes. JCN acknowledges support from the NSERC Discovery and Discovery Accelerator Programs. KS thankfully acknowledges financial support from the ANR project SiCoMHD (ANR-Blanc 2011-045).

Appendix: Two-Dimensional Hermite Interpolation

Suppose that values of a function $u(x_1, x_2)$ and its derivatives u_{x_1}, u_{x_2} and $u_{x_1x_2}$ are given at four vertices of a square of side h, as shown in Fig. 18. We will use superscripts sw, se, nw and ne to refer to these points and the corresponding values.

Let us rescale the coordinates x_1, x_2:

$$\tilde{x}_1 = \frac{x_1 - x_1^{\mathrm{sw}}}{h}, \qquad \tilde{x}_2 = \frac{x_2 - x_2^{\mathrm{sw}}}{h}, \tag{37}$$

and define basis functions

$$f(\tilde{x}) = 2\tilde{x}^3 - 3\tilde{x}^2 + 1, \qquad g(\tilde{x}) = \tilde{x}^3 - 2\tilde{x}^2 + \tilde{x} \tag{38}$$

The value of u at $(x_1, x_2) \in [x_1^{\mathrm{sw}}, x_1^{\mathrm{sw}} + h] \times [x_2^{\mathrm{sw}}, x_2^{\mathrm{sw}} + h]$ can be estimated using the following $\mathcal{O}(h^4)$ accurate formula:

$$\begin{aligned}\tilde{u}(x_1, x_2) = {} & (u^{\mathrm{sw}} f(\tilde{x}_1) f(\tilde{x}_2) + u^{\mathrm{se}} f(1-\tilde{x}_1) f(\tilde{x}_2) \\ & + u^{\mathrm{nw}} f(\tilde{x}_1) f(1-\tilde{x}_2) + u^{\mathrm{ne}} f(1-\tilde{x}_1) f(1-\tilde{x}_2)) \\ & + h\ (u_{x_1}^{\mathrm{sw}} g(\tilde{x}_1) f(\tilde{x}_2) - u_{x_1}^{se} g(1-\tilde{x}_1) f(\tilde{x}_2) \\ & + u_{x_1}^{\mathrm{nw}} g(\tilde{x}_1) f(1-\tilde{x}_2) - u_{x_1}^{\mathrm{ne}} g(1-\tilde{x}_1) f(1-\tilde{x}_2)) \\ & + h\ (u_{x_2}^{\mathrm{sw}} f(\tilde{x}_1) g(\tilde{x}_2) + u_{x_2}^{\mathrm{se}} f(1-\tilde{x}_1) g(\tilde{x}_2) \\ & - u_{x_2}^{\mathrm{nw}} f(\tilde{x}_1) g(1-\tilde{x}_2) - u_{x_2}^{\mathrm{ne}} f(1-\tilde{x}_1) g(1-\tilde{x}_2)) \\ & + h^2\ (u_{x_1x_2}^{\mathrm{sw}} g(\tilde{x}_1) g(\tilde{x}_2) - u_{x_1x_2}^{\mathrm{se}} g(1-\tilde{x}_1) g(\tilde{x}_2) \\ & - u_{x_1x_2}^{\mathrm{nw}} g(\tilde{x}_1) g(1-\tilde{x}_2) + u_{x_1x_2}^{\mathrm{ne}} g(1-\tilde{x}_1) g(1-\tilde{x}_2)).\end{aligned} \tag{39}$$

It is straightforward to obtain interpolation formulae for the first and second partial derivatives of u by derivating (39).

The values $\tilde{u}(x_1^{\mathrm{sw}} + \frac{h}{2}, x_2^{\mathrm{sw}})$, $\tilde{u}(x_1^{\mathrm{sw}}, x_2^{\mathrm{sw}} + \frac{h}{2})$ and $\tilde{u}(x_1^{\mathrm{sw}} + \frac{h}{2}, x_2^{\mathrm{sw}} + \frac{h}{2})$, as well as unscaled derivatives required for the error estimate, are also obtained from (39). For multiresolution decomposition (16) and reconstruction (17), we define scaled quantities:

$$\begin{array}{llll} u_0^{00} = u^{sw}, & u_0^{20} = u^{se}, & u_0^{02} = u^{nw}, & u_0^{22} = u^{ne}, \\ u_1^{00} = \frac{h}{2} u_{x_1}^{sw}, & u_1^{20} = \frac{h}{2} u_{x_1}^{se}, & u_1^{02} = \frac{h}{2} u_{x_1}^{nw}, & u_1^{22} = \frac{h}{2} u_{x_1}^{ne}, \\ u_2^{00} = \frac{h}{2} u_{x_2}^{sw}, & u_2^{20} = \frac{h}{2} u_{x_2}^{se}, & u_2^{02} = \frac{h}{2} u_{x_2}^{nw}, & u_2^{22} = \frac{h}{2} u_{x_2}^{ne}, \\ u_3^{00} = \frac{h^2}{4} u_{x_1x_2}^{sw}, & u_3^{20} = \frac{h^2}{4} u_{x_1x_2}^{se}, & u_3^{02} = \frac{h^2}{4} u_{x_1x_2}^{nw}, & u_3^{22} = \frac{h^2}{4} u_{x_1x_2}^{ne}, \end{array} \tag{40}$$

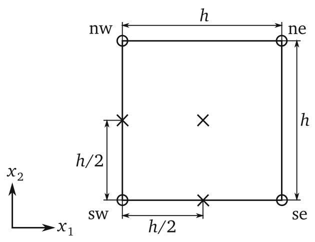

Fig. 18 Interpolation cell. Markers ∘ show the corner points, where the values of the function u and its derivatives u_{x_1}, u_{x_2} and $u_{x_1x_2}$ are known. Markers × show the 3 points that appear in mid-point interpolation formulae (42–45).

as well as

$$\begin{aligned}
\tilde{u}_0^{10} &= \tilde{u}\left(x_1^{\mathrm{sw}}+\tfrac{h}{2}, x_2^{\mathrm{sw}}\right), & \tilde{u}_0^{01} &= \tilde{u}\left(x_1^{\mathrm{sw}}, x_2^{\mathrm{sw}}+\tfrac{h}{2}\right),\\
& & \tilde{u}_0^{11} &= \tilde{u}\left(x_1^{\mathrm{sw}}+\tfrac{h}{2}, x_2^{\mathrm{sw}}+\tfrac{h}{2}\right),\\
\tilde{u}_1^{10} &= \tfrac{h}{2}\tilde{u}_{x_1}\left(x_1^{\mathrm{sw}}+\tfrac{h}{2}, x_2^{\mathrm{sw}}\right), & \tilde{u}_1^{01} &= \tfrac{h}{2}\tilde{u}_{x_1}\left(x_1^{\mathrm{sw}}, x_2^{\mathrm{sw}}+\tfrac{h}{2}\right),\\
& & \tilde{u}_1^{11} &= \tfrac{h}{2}\tilde{u}_{x_1}\left(x_1^{\mathrm{sw}}+\tfrac{h}{2}, x_2^{\mathrm{sw}}+\tfrac{h}{2}\right),\\
\tilde{u}_2^{10} &= \tfrac{h}{2}\tilde{u}_{x_2}\left(x_1^{\mathrm{sw}}+\tfrac{h}{2}, x_2^{\mathrm{sw}}\right), & \tilde{u}_2^{01} &= \tfrac{h}{2}\tilde{u}_{x_2}\left(x_1^{\mathrm{sw}}, x_2^{\mathrm{sw}}+\tfrac{h}{2}\right),\\
& & \tilde{u}_2^{11} &= \tfrac{h}{2}\tilde{u}_{x_2}\left(x_1^{\mathrm{sw}}+\tfrac{h}{2}, x_2^{\mathrm{sw}}+\tfrac{h}{2}\right),\\
\tilde{u}_3^{10} &= \tfrac{h^2}{4}\tilde{u}_{x_1x_2}\left(x_1^{\mathrm{sw}}+\tfrac{h}{2}, x_2^{\mathrm{sw}}\right), & \tilde{u}_3^{01} &= \tfrac{h^2}{4}\tilde{u}_{x_1x_2}\left(x_1^{\mathrm{sw}}, x_2^{\mathrm{sw}}+\tfrac{h}{2}\right),\\
& & \tilde{u}_3^{11} &= \tfrac{h^2}{4}\tilde{u}_{x_1x_2}\left(x_1^{\mathrm{sw}}+\tfrac{h}{2}, x_2^{\mathrm{sw}}+\tfrac{h}{2}\right).
\end{aligned} \tag{41}$$

Thus we obtain the following formulae:

$$\begin{aligned}
\tilde{u}_0^{10} &= \frac{1}{2}\left(u^{00}+u^{10}\right)+\frac{1}{4}\left(u_{x1}^{00}-u_{x1}^{10}\right),\\
\tilde{u}_0^{01} &= \frac{1}{2}\left(u^{00}+u^{01}\right)+\frac{1}{4}\left(u_{x2}^{00}-u_{x2}^{01}\right),\\
\tilde{u}_0^{11} &= \frac{1}{4}\left(u^{00}+u^{10}+u^{01}+u^{11}\right)\\
&\quad+\frac{1}{8}\left(\left(u_{x1}^{00}-u_{x1}^{10}+u_{x1}^{01}-u_{x1}^{11}\right)+\left(u_{x2}^{00}+u_{x2}^{10}-u_{x2}^{01}-u_{x2}^{11}\right)\right)\\
&\quad+\frac{1}{16}\left(u_{x_1x_2}^{00}-u_{x_1x_2}^{10}-u_{x_1x_2}^{01}+u_{x_1x_2}^{11}\right),
\end{aligned} \tag{42}$$

$$\begin{aligned}
\tilde{u}_1^{10} &= -\frac{3}{4}\left(u^{00}-u^{10}\right)-\frac{1}{4}\left(u_{x_1}^{00}+u_{x_1}^{10}\right),\\
\tilde{u}_1^{01} &= \frac{1}{2}\left(u_{x_1}^{00}+u_{x_1}^{01}\right)+\frac{1}{4}\left(u_{x_1x_2}^{00}-u_{x_1x_2}^{01}\right),\\
\tilde{u}_1^{11} &= \frac{3}{8}\left(-u^{00}+u^{10}-u^{01}+u^{11}\right)\\
&\quad-\frac{1}{8}\left(u_{x_1}^{00}+u_{x_1}^{10}+u_{x_1}^{01}+u_{x_1}^{11}\right)-\frac{3}{16}\left(u_{x_2}^{00}-u_{x_2}^{10}-u_{x_2}^{01}+u_{x_2}^{11}\right)\\
&\quad-\frac{1}{16}\left(u_{x_1x_2}^{00}+u_{x_1x_2}^{10}-u_{x_1x_2}^{01}-u_{x_1x_2}^{11}\right)
\end{aligned} \tag{43}$$

$$\begin{aligned}
\tilde{u}_2^{10} &= \frac{1}{2}\left(u_{x_2}^{00}+u_{x_2}^{10}\right)+\frac{1}{4}\left(u_{x_1x_2}^{00}-u_{x_1x_2}^{10}\right),\\
\tilde{u}_2^{01} &= -\frac{3}{4}\left(u^{00}-u^{01}\right)-\frac{1}{4}\left(u_{x_2}^{00}+u_{x_2}^{01}\right),\\
\tilde{u}_2^{11} &= \frac{3}{8}\left(-u^{00}-u^{10}+u^{01}+u^{11}\right)\\
&\quad-\frac{3}{16}\left(u_{x_1}^{00}-u_{x_1}^{10}-u_{x_1}^{01}+u_{x_1}^{11}\right)-\frac{1}{8}\left(u_{x_2}^{00}+u_{x_2}^{10}+u_{x_2}^{01}+u_{x_2}^{11}\right)\\
&\quad-\frac{1}{16}\left(u_{x_1x_2}^{00}-u_{x_1x_2}^{10}+u_{x_1x_2}^{01}-u_{x_1x_2}^{11}\right)
\end{aligned} \tag{44}$$

$$\begin{aligned}
\tilde{u}_3^{10} &= -\frac{3}{4}\left(u_{x_2}^{00}-u_{x_2}^{10}\right)-\frac{1}{4}\left(u_{x_1x_2}^{00}+u_{x_1x_2}^{10}\right),\\
\tilde{u}_3^{01} &= -\frac{3}{4}\left(u_{x_1}^{00}-u_{x_1}^{01}\right)-\frac{1}{4}\left(u_{x_1x_2}^{00}+u_{x_1x_2}^{01}\right),\\
\tilde{u}_3^{11} &= \frac{9}{16}\left(u^{00}-u^{10}-u^{01}+u^{11}\right)
\end{aligned}$$

$$+\frac{3}{16}\left(\left(u_{x_1}^{00}+u_{x_1}^{10}-u_{x_1}^{01}-u_{x_1}^{11}\right)+\left(u_{x_2}^{00}-u_{x_2}^{10}+u_{x_2}^{01}-u_{x_2}^{11}\right)\right)$$
$$+\frac{1}{16}\left(u_{x_1x_2}^{00}+u_{x_1x_2}^{10}+u_{x_1x_2}^{01}+u_{x_1x_2}^{11}\right) \tag{45}$$

In the notations of (16–17), we obtain

$$(\tilde{u}_\iota)^l_{2j_1+1,2j_2}=\tilde{u}_\iota^{10},\quad (\tilde{u}_\iota)^l_{2j_1,2j_2+1}=\tilde{u}_\iota^{01},\quad (\tilde{u}_\iota)^l_{2j_1+1,2j_2+1}=\tilde{u}_\iota^{11}, \tag{46}$$

where $\iota = 0, \ldots, 3$ and indices j_1, j_2 and l are determined by $\boldsymbol{x}^{sw}$ and h, as described in Sects. 2.2 and 2.3. Note that the computational cost of this procedure is much less than interpolation at an arbitrary point because the coefficients that include the basis functions (38) are pre-computed analytically.

References

1. Aftosmis, M.J.: Solution adaptive Cartesian grid methods for aerodynamic flows with complex geometries. von Karman Institute for Fluid Dynamics Lecture Series 1997–02, Rhode-Saint-Genèse (1997)
2. Appelö, D., Hagstrom, T.: On advection by Hermite methods. Pac. J. Appl. Math. **4**(2), 125–139 (2011)
3. Becker, R., Rannacher, R.: An optimal control approach to a posteriori error estimation in finite element methods. Acta Numer. **10**, 1–102 (2001)
4. Berger, M.J., Colella, P.: Local adaptive mesh refinement for shock hydrodynamics. J. Comput. Phys. **82**(1), 64–84 (1989)
5. Berger, M., LeVeque, R.: Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems. SIAM J. Numer. Anal. **35**(6), 2298–2316 (1998)
6. Berger, M.J., Oliger, J.: Adaptive mesh refinement for hyperbolic partial differential equations. J. Comput. Phys. **53**(3), 484–512 (1984)
7. Brun, E., Guittet, A., Gibou, F.: A local level-set method using a hash table data structure. J. Comput. Phys. **231**(6), 2528–2536 (2012)
8. Chiavassa, G., Donat, R.: Point value multiscale algorithms for 2D compressible flows. SIAM J. Sci. Comput. **23**(3), 805–823 (2001)
9. Chidyagwal, P., Nave, J.C., Rosales, R., Seibold, B.: A comparative study of the efficiency of jet schemes. Int. J. Numer. Anal. Model. Ser. B **3**(3), 297–306 (2012)
10. Cohen, A.: Wavelet methods in numerical analysis. In: Ciarlet, P., Lions, J. (eds.) Solution of Equation in $\mathbb{R}^n$ (Part 3), Techniques of Scientific Computing (Part 3), Handbook of Numerical Analysis, vol. 7, pp. 417–711. Elsevier, Amsterdam (2000)
11. Dahmen, W., Gotzen, T., Melian, S., Müller, S.: Numerical simulation of cooling gas injection using adaptive multiresolution techniques. Comput. Fluids **71**, 65–82 (2013)
12. Deiterding, R.: Block-structured adaptive mesh refinement: theory, implementation and application. ESAIM Proc. **34**, 97–150 (2011)
13. DeVore, R.: Nonlinear approximation. Acta Numer. **7**, 51–150 (1998)
14. Domingues, M., Gomes, S., Roussel, O., Schneider, K.: An adaptive multiresolution scheme with local time stepping for evolutionary PDEs. J. Comput. Phys. **227**(8), 3758–3780 (2008)
15. Domingues, M., Gomes, S., Roussel, O., Schneider, K.: Adaptive multiresolution methods. ESAIM Proc. **34**, 1–96 (2011)
16. Dormand, J.R., Prince, P.J.: New Runge–Kutta algorithms for numerical simulation in dynamical astronomy. Celest. Mech. **18**, 223–232 (1978)
17. Harten, A.: Multiresolution algorithms for the numerical solution of hyperbolic conservation laws. Commun. Pure Appl. Math. **48**(12), 1305–1342 (1995)
18. Harten, A.: Multiresolution representation of data: a general framework. SIAM J. Numer. Anal. **33**(3), 1205–1256 (1996)
19. Kaibara, M., Gomes, S.: A fully adaptive multiresolution scheme for shock computations. In: Toro, E. (ed.) Godunov Methods: Theory and Applications, pp. 503–597. Kluwer/Plenum, Dordrecht/New York (2001)
20. Lindsay, K., Krasny, R.: A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow. J. Comput. Phys. **172**(2), 879–907 (2001)
21. Liu, X.D., Osher, S., Chan, T.: Weighted essentially non-oscillatory schemes. J. Comput. Phys. **115**(1), 200–212 (1994)

22. Losasso, F., Gibou, F., Fedkiw, R.: Simulating water and smoke with an octree data structure. ACM Trans. Graph. TOG **23**(3), 457–462 (2004)
23. Martin, D.F., Colella, P.: A cell-centered adaptive projection method for the incompressible Euler equations. J. Comput. Phys. **163**(2), 271–312 (2000)
24. Min, C., Gibou, F.: A second order accurate level set method on non-graded adaptive Cartesian grids. J. Comput. Phys. **225**(1), 300–321 (2007)
25. Müller, S.: Adaptive multiscale schemes for conservation laws. In: Lecture Notes in Computational Science and Engineering, vol. 27. Springer, Berlin (2003)
26. Nave, J.C., Rosales, R., Seibold, B.: A gradient-augmented level set method with an optimally local, coherent advection scheme. J. Comput. Phys. **229**(10), 3802–3827 (2010)
27. Nguyen van yen, R., Sonnendrücker, E., Schneider, K., Farge, M.: Particle-in-wavelets scheme for the 1D Vlasov–Poisson equations. ESAIM Proc. **32**, 134–148 (2011)
28. Ottino, J.: The Kinematics of Mixing: Stretching, Chaos, and Transport. Cambridge University Press, Cambridge (1989)
29. Plewa, T., Linde, T., Weirs, V.G.: Adaptive mesh refinement: theory and applications. In: Proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, Sept. 3–5, 2003, Lecture Notes in Computational Science and Engineering, vol. 41. Springer, Berlin (2005)
30. Popinet, S.: Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. J. Comput. Phys. **190**(2), 572–600 (2003)
31. Popinet, S.: An accurate adaptive solver for surface-tension-driven interfacial flows. J. Comput. Phys. **228**(16), 5838–5866 (2009)
32. Rossinelli, D., Hejazialhosseini, B., van Rees, W., Gazzola, M., Bergdorf, M., Koumoutsakos, P.: MRAG-I2D: multi-resolution adapted grids for remeshed vortex methods on multicore architectures. J. Comput. Phys. **288**, 1–18 (2015)
33. Roussel, O., Schneider, K.: Coherent vortex simulation of weakly compressible turbulent mixing layers using adaptive multiresolution methods. J. Comput. Phys. **229**(6), 2267–2286 (2010)
34. Seibold, B., Rosales, R., Nave, J.C.: Jet schemes for advection problems. Discrete Contin. Dyn. Syst. Ser. B **17**(4), 1229–1259 (2012)
35. Shu, C.W., Osher, S.: Efficient implementation of essentially non-oscillatory shock-capturing schemes. J. Comput. Phys. **77**(2), 439–471 (1988)
36. Sonnendrücker, E., Roche, J., Bertrand, P., Ghizzo, A.: The semi-Lagrangian method for the numerical resolution of the Vlasov equation. J. Comput. Phys. **149**(2), 201–220 (1999)
37. Staniforth, A., Côté, J.: Semi-Lagrangian integration schemes for atmospheric models: a review. Mon. Weather Rev. **119**(9), 2206–2223 (1991)
38. Verfürth, R.: A Posteriori Error Estimation Techniques for Finite Element Methods. Oxford University Press, Oxford (2013)
39. Wang, S.: Elliptic interface problem solved using the mixed finite element method. Ph.D. thesis, Stony Brook University, (2007)
40. Warming, R., Beam, R.: Discrete multiresolution analysis using Hermite interpolation: biorthogonal multiwavelets. SIAM J. Sci. Comput. **22**(4), 1269–1317 (2000)
41. Ziegler, J.L., Deiterding, R., Shepherd, J.E., Pullin, D.I.: An adaptive high-order hybrid scheme for compressive, viscous flows with detailed chemistry. J. Comput. Phys. **230**(20), 7598–7630 (2011)