ADAPTIVE FAST INTERFACE TRACKING METHODS

JELENA POPOVIC[1] AND OLOF RUNBORG[2]

Abstract. In this paper, we present a fast time adaptive numerical method for interface tracking. The method uses an explicit multiresolution description of the interface, which is represented by wavelet vectors that correspond to the details of the interface on different scale levels. The complexity of standard numerical methods for interface tracking, where the interface is described by N marker points, is $O(N/\Delta t)$, when a time step Δt is used. The methods that we propose in this paper have $O(\text{TOL}^{-1/p}\log N + N\log N)$ computational cost, at least for uniformly smooth problems, where TOL is some given tolerance and p is the order of the time stepping method that is used for time advection of the interface. The adaptive method is robust in the sense that it can handle problems with both smooth and piecewise smooth interfaces (e.g. interfaces with corners) while keeping a low computational cost. We show numerical examples that verify these properties.

Keywords: interface tracking, multiresolution analysis, fast algorithms, adaptive methods

AMS subject classification: 42C40, 65D99, 65T60

1. Introduction

Propagating interfaces occur in many applications. Some examples are wave propagation, multiphase flow, crystal growth, melting, epitaxial growth or flame propagation. Therefore, development of fast and accurate numerical methods for interface tracking is very important. The interface can be a curve or a surface in $\mathbb{R}^3$ or a curve in $\mathbb{R}^2$ that moves with a given propagation velocity F. It is represented either explicitly or implicitly, depending on the numerical method that is used for its evolution. For instance, standard front tracking methods are based on explicit representation of the interface. The interface is in this case described by a set of marker points and their evolution represents the dynamics of the interface; see for example [14, 15, 29, 30] for different applications. An implicit interface representation is used for example in the level set method, [21]. The interface is then represented by the zero level set of a continuous function and its time evolution is described by a partial differential equation. We should also mention the segment projection method [28], which is a compromise between the front tracking and level set methods, where the interface is divided into segments chosen such that they can be given as functions of one variable.

In this paper we develop a fast method for a subclass of interface tracking problems, in which the local velocity of the interface only depends on the location, not on the local shape of the interface. Examples of such applications are found for instance in geophysics where tracking of high frequency wave fronts is common, [30, 18]. The method uses an explicit multiresolution representation of the interface. It is based on previous work in [23, 24, 25]. See also [8]. The interface is represented by wavelet vectors that correspond to

Date: February 7, 2017.

[1] Department of Numerical Analysis, CSC, KTH, 100 44 Stockholm, Sweden, email: `jelenap@csc.kth.se`.

[2] Department of Mathematics and Swedish e-Science Research Center (SeRC), KTH, 100 44 Stockholm, Sweden, email: `olofr@kth.se`.

the details of the interface on different scale levels. It is well-known that, for a fixed smooth curve or surface, the size of the wavelet vectors decays fast as the scale becomes finer, [7]. Moreover, the size of their time derivatives decays in the same way [25], which means that the finer scales of the interface evolve more slowly than the coarse scales. Hence, it would be computationally advantageous to use shorter time steps for coarse scales and longer time steps for fine scales. This is the idea behind the fast interface tracking methods developed in [24, 25]. The computational cost of these methods is significantly reduced, compared to standard front tracking and level set methods.

We assume that the interface can be parameterized so that, for a fixed time t, it is described by the function $x(t,s) : \mathbb{R}^+ \times \mathbb{R}^q \to \mathbb{R}^d$, with the parameterization $s \in \Omega \subset \mathbb{R}^q$ and $q < d$. We consider the case when the interface is moving in a time-varying velocity field that does not depend on the front itself. Then $x(t,s)$ satisfies the parameterized ordinary differential equation (ODE)

$$\frac{\partial x(t,s)}{\partial t} = F(t, x(t,s)), \qquad x(0,s) = \gamma(s), \qquad s \in \Omega, \tag{1}$$

where $F(t,x) : \mathbb{R}^+ \times \mathbb{R}^d \to \mathbb{R}^d$ is a given function representing the velocity field and $\gamma(s) : \mathbb{R}^q \to \mathbb{R}^d$ is the initial interface. We will consider curves in two dimensions, $d = 2$, $q = 1$ and surfaces in three dimensions, $d = 3$, $q = 2$.

Using standard front tracking methods in 1D, one would approximate $x_j(t) = x(t, s_j)$ and use a numerical method for ODEs to solve

$$\frac{dx_j(t)}{dt} = F(t, x_j(t)), \qquad x_j(0) = \gamma(s_j), \tag{2}$$

where $s_0 < s_1 < \ldots < s_N$ is a discretization of Ω. Then, assuming N marker points $x_j(t)$, the cost of such a method would be $O(N/\Delta t)$ where Δt is the time step. As for the standard level set method, the cost is $O(N^2/\Delta t)$ if the full domain is discretized in 2D with N points in each coordinate direction. There are, however, several clever versions that localize the computations around the interface, e.g. local level set methods [22], and tree methods [26, 27]. These bring down the complexity to $O(N \log N/\Delta t)$, or almost the same as front tracking. As a comparison, the cost of the fast method described in [24, 25] is $O(\log(N)/\Delta t + N)$. The $O(\log N/\Delta t)$ is the cost to propagate the interface. It is thus essentially the same as the cost to propagate just one marker point. The $O(N)$ part of the complexity comes from the reconstruction of the interface from the wavelet vectors. The cost of the fast time adaptive method that we propose in this paper is $O(\log N/\text{TOL}^{1/p} + N \log N)$, where TOL is some given tolerance and p is the order of the numerical ODE method used for time propagation.

The idea behind the method presented in this paper is similar to the idea behind the methods presented in [24, 25], i.e. to use longer time steps for finer scales and shorter time steps for coarser scales. In [24, 25] a time step doubling technique was used for the interface evolution; the time step was doubled at every scale level. In that way all wavelet vectors within a level are moved using the same time step, but the time step varies for different scale levels. We will refer to this version of the fast method as the *basic method.* The technique assumes that the wavelet coefficients, within the same scale level, change with an approximately uniform rate in time, which is not always the case in practice. For instance, for problems in which the interface has corners, the wavelet vectors in a neighborhood of the corner, and their time derivatives, decay much slower with scale level than in the smooth regions. The basic method is not suitable for such problems and will fail to approximate the front accurately. We have to use other methods. In this article we propose to build a fast method on an adaptive ODE solver, instead of an ODE solver with a uniform time step that is precisely doubled at every scale level, as in the basic method. The adaptivity mechanism is a standard one where the local truncation error is estimated and used to choose the time step, for each wavelet vector individually. The main new difficulty is the need for an additional interpolation step, which must be done recursively to maintain accuracy.

We emphasize the difference from the methods in [24, 25]: Instead of using a time step that is simply predetermined by the asymptotic rate of change of the wavelet vectors, we use a more precise choice of time step, that is determined automatically from the estimated local truncation error. For smooth problems, the methods would give roughly the same time steps. However, the adaptive method is robust in the sense that it can handle problems with both smooth and non-smooth interfaces (i.e. interfaces with corners) while maintaining a low computational cost. Thus, for practical applications adaptivity is essential. Indeed, in the numerical examples presented in Section 6 the adaptive method clearly outperforms the basic method for a variety of problems.

Since we have one time step per wavelet vector the method we propose falls into the general category of multirate or multiadaptive ODE methods; see, for instance [19, 13, 4, 16]. Our method is tailored to the wavelet ODE system. It is a proof of concept that shows the potential of using this kind of adaptive methods. More elaborate multirate and multiadaptive methods for this problem can doubtlessly be developed.

One should finally note that although this method is fast, it is not as versatile as, for instance, the level set method, which is much more suitable for handling topological changes in the interface; like with other front tracking based algorithms, this would be difficult with the proposed method.

The article is organized as follows. In Section 2 a brief description of subdivision schemes is given. Section 3 presents the multiresolution representation of the interface and the governing ODEs are derived. In Section 4 we introduce the notations and give a short overview of the method described in [24]. Our adaptive method is subsequently described in detail in Section 5. Numerical experiments are presented in Section 6. Section 7 concludes the paper and discusses some open problems.

2. Subdivision Schemes

Subdivision is a procedure to iteratively create smooth curves and surfaces from an initial mesh, [3, 5, 6, 11]. Let $\boldsymbol{x} = \{x_k\}$ be a sequence. We consider here local, interpolatory, stationary subdivision schemes. They are characterized by a *mask*, a finite sequence $\boldsymbol{a} = \{a_\ell\}$, which defines a bounded linear operator $S : \ell_\infty \mapsto \ell_\infty$ as follows

$$(S\boldsymbol{x})_{2k+1} = \sum_{\ell} a_\ell x_{k+\ell}, \qquad (S\boldsymbol{x})_{2k} = x_k. \tag{3}$$

The width B of S is defined by $B = 2\max\{|\ell|; a_\ell \neq 0\}$. We assume henceforth that the number of non-zero elements in $\boldsymbol{a}$ is even, so that a_ℓ is non-zero only for $\ell \in [-B/2+1, B/2]$. To build a smooth function, we start from a sequence $\boldsymbol{x}_0 = \{x_{0,k}\}$ and associate to it a function $f_0(x)$ which is piecewise linear and interpolates $x_{0,k}$ on the integer grid, $f_0(k) = x_{0,k}$. We can then apply the subdivision scheme S iteratively and define $\boldsymbol{x}_j$ for all $j > 0$ by

$$\boldsymbol{x}_{j+1} = S\boldsymbol{x}_j.$$

Sequences at level $j > 0$ similarly represent samples of piecewise linear functions $f_j(x)$ on grids with the increasingly fine spacing 2^{-j}. More precisely, $\boldsymbol{x}_j = \{x_{j,k}\}$ is associated with the grid $\boldsymbol{s}_j$, where $s_{j,k} = k2^{-j}$, and $x_{j,k} = f_j(s_{j,k})$. Note that f_{j+1} interpolates f_j on the coarser grid: $f_{j+1}(s_{j,k}) = f_j(s_{j,k})$ since $s_{j+1,2k} = s_{j,k}$. For a large class of subdivision operators S, the process converges, $f_j \to f$ where f is a smooth function.

The *order* of the subdivision scheme S is the largest q such that $SP(\boldsymbol{k}) = P(\boldsymbol{k}/2)$ for all polynomials P of degree $p < q$, where $\boldsymbol{k}$ is the linear sequence $\{k\}$ and P is applied elementwise, $P(\boldsymbol{k}) = \{P(k)\}$.

A special example of a subdivision scheme is the midpoint interpolating scheme S_2 where $(S_2\boldsymbol{x})_{2k+1} = (x_k + x_{k+1})/2$. Hence the mask is $[1/2, 1/2]$. This scheme has order $q = 2$ and yields piecewise linear limit functions. Some examples of masks for higher order interpolating subdivision schemes are:

- "4-point" subdivision scheme, order $q = 4$
$$\left[\frac{1}{16}\{-1, 9, 9, -1\}\right],$$
- "6-point" subdivision scheme, order $q = 6$
$$\left[\frac{1}{256}\{3, -25, 150, 150, -25, 3\}\right],$$
- "8-point" subdivision scheme, order $q = 8$
$$\left[\frac{1}{2048}\{-5, 49, -245, 1225, 1225, -245, 49, -5\}\right].$$

These are called the Lagrange subdivision schemes and have $q = B$. We will use S_q to denote the Lagrange subdivision scheme of order q.

Remark 2.1. In the description above the subdivision operators act on infinite or periodic sequences. In numerical computations one must often restrict them to finite length sequences, corresponding to samples of functions on bounded intervals. The subdivision operators can then be modified near the boundaries, such that a different sequence $\boldsymbol{a}$ is used in (3) for those points. For interpolatory schemes, a modification is necessary when the width B exceeds two. (Hence, S_2 does not have to be modified.) By using an appropriate $\boldsymbol{a}$, high order schemes can then still be constructed. For instance, in the case of the 6-point scheme, the mask above is replaced by the following skewed masks at $k = 0$ and $k = 1$,

$$(S_6\boldsymbol{x})_1 = \frac{1}{256}\left(63x_0 + 315x_1 - 210x_2 + 126x_3 - 45x_4 + 7x_5\right), \tag{4}$$
$$(S_6\boldsymbol{x})_3 = \frac{1}{256}\left(-7x_0 + 105x_1 + 210x_2 - 70x_3 + 21x_4 - 3x_5\right),$$

and similar at the right boundary. An example with these masks is given in Figure 10.

3. Multiresolution Description of the Interface Propagation

In standard front tracking algorithms, marker points are used to represent the interface. We will, instead, consider a multiresolution representation, which involves a hierarchy of increasingly detailed meshes. Each new mesh level is computed from the previous one by first predicting a new point using a subdivision schemes, and then correcting the predicted point by a wavelet (or detail) vector. Only the wavelet vectors need to be stored and due to the curve or surface smoothness most wavelet vectors will be small.

To be precise, we describe the curve $x(t, s)$ where $0 \le s \le 1$, as follows. In particular, this can be a closed, 1-periodic, curve. We introduce the parameter indices $s_{j,k}$ and define

$$x_{j,k}(t) := x(t, s_{j,k}), \qquad 0 \le k \le 2^j, \qquad s_{j,k} = k2^{-j}.$$

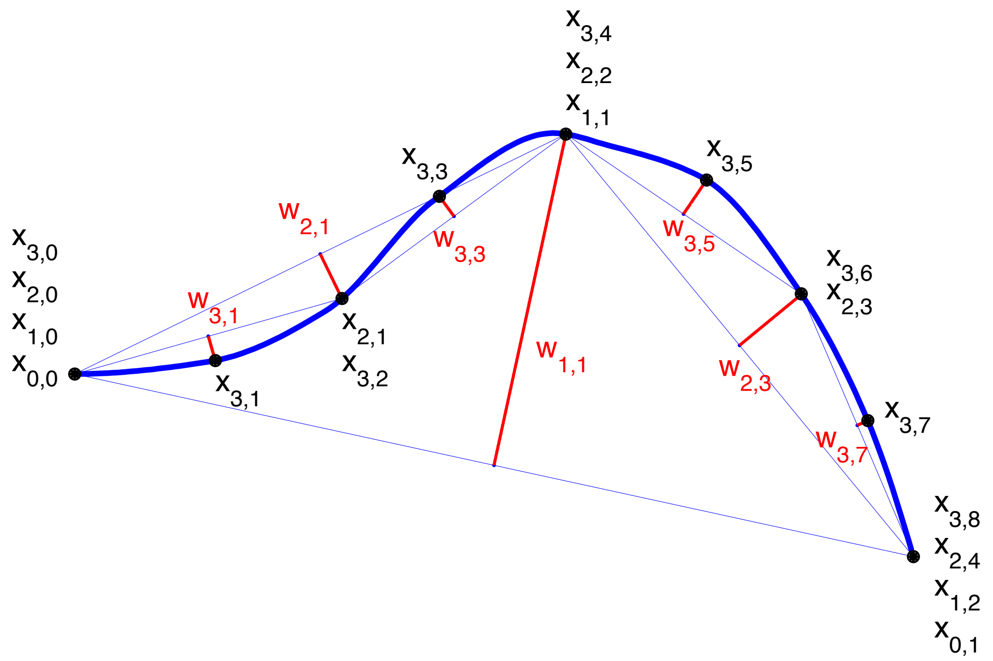

FIGURE 1. The notation for a simple interface with $J = 3$ levels and midpoint subdivision $S = S_2$. Bold line is the interface. Wavelet vectors $w_{j,k}$ are indicated by semibold red lines and point markers $x_{j,k}$ by filled circles.

Note that $x_{j+1,2k} = x_{j,k}$ and that, for a fixed j, the markers $\{x_{j,k}\}$ will be a discretization of the interface with a level of detail that increases with the fixed j-value. We assume that we start from a given fine discretization with 2^J points on the interface, which thus corresponds to level J. We let $\boldsymbol{x}_j(t) = \{x_{j,k}(t)\}_{k=0}^{2^j}$ and for $j \leq J$ define the wavelet vectors

$$\boldsymbol{w}_{j+1}(t) = \boldsymbol{x}_{j+1}(t) - S\boldsymbol{x}_j(t), \tag{5}$$

where S is an interpolatory subdivision operator, for example the midpoint scheme. This is done recursively and gives an alternative description of the interface in terms of $\boldsymbol{x}_0(t)$ together with $\boldsymbol{w}_j(t)$ for $j = 1, \ldots, J$. Note that wavelet vectors $w_{j,k}$ with even k are zero since S is interpolatory. Moreover, marker points $x_{j,k}$ have in general several aliases, but wavelet vectors with odd k have unique names. See Figure 1 for an example.

The wavelet sequences $\boldsymbol{w}_j(t) =: \{w_{j,k}(t)\}_{k=0}^{2^j-1}$ can be computed from the original discretization $\boldsymbol{x}_J(t)$ at an $O(N)$ cost, where $N = 2^J$ is the number of discretization points. Similarly, with an inverse wavelet transform based on reversing the recursion in (5), the points $\boldsymbol{x}_J(t)$ can be computed from $\{\boldsymbol{w}_j\}$ and $\boldsymbol{x}_0$ at a $O(N)$ cost. Moreover, for a smooth $x(t,s)$ the wavelet vectors decay in j as 2^{-jq} where q is the order of the subdivision scheme S, see e.g. [25]. The fast decay of the wavelet vectors gives the representation good compression properties.

For the dynamic case, we insert (5) in (1) to obtain

$$\frac{d\boldsymbol{w}_{j+1}(t)}{dt} = F(t, \boldsymbol{x}_{j+1}(t)) - SF(t, \boldsymbol{x}_j(t)) = F(t, S\boldsymbol{x}_j(t) + \boldsymbol{w}_{j+1}(t)) - SF(t, \boldsymbol{x}_j(t)).$$

Setting

$$G(t, \boldsymbol{x}, \boldsymbol{w}) = F(t, S\boldsymbol{x} + \boldsymbol{w}) - SF(t, \boldsymbol{x}),$$

we thus have the following alternative system of ODEs

$$\frac{d\boldsymbol{w}_{j+1}(t)}{dt} = G(t, \boldsymbol{x}_j(t), \boldsymbol{w}_{j+1}(t)), \qquad \frac{d\boldsymbol{x}_0(t)}{dt} = F(t, \boldsymbol{x}_0(t)), \tag{6}$$

which together with (5) describe the dynamics of the system.

In [25] it was shown that also the time derivatives of $\boldsymbol{w}_j(t)$ decay fast with j, and faster the higher the order of S. More precisely, if $x(t,s) \in C^{q+\ell}([0,T]\times[0,1];\mathbb{R}^d)$ and S is a linear, stationary and interpolatory subdivision operator of order q then

$$\left|\frac{d^\ell \boldsymbol{w}_j(t)}{dt^\ell}\right| \le C_\ell(T)2^{-jq}, \qquad 0 \le t \le T. \tag{7}$$

Hence, both the wavelet vectors and their derivatives decay as 2^{-qj}. That means that the fine spatial scales change more slowly than the coarse spatial scales. Thus, it is possible to use longer time steps for the fine scales. This is exactly the idea behind the method in [24, 25] that will be described in the next section: use shorter time steps at lower levels and longer time steps at higher levels. Note also that fine scales depend on coarser scales, but there is no dependence in the other direction. This means that we can compute the different scale levels sequentially from coarse to fine, one after the other. This is the same idea that is used in the inverse wavelet transform.

4. Basic Fast Numerical Method

In this section, we give a short overview of the fast interface tracking method described in [24, 25]. Henceforth, we shall refer to this method as the *basic method.* The interface is described using the multiresolution representation described above. The ODE (6) together with (5) is solved using a *time-step doubling* technique, i.e. the time step is doubled at every level j such that we use longer time steps for larger j.

We start by writing down the equations for the wavelet vectors in component form. Letting S being an interpolatory subdivision scheme with mask $\{a_\ell\}$ and width B we have

$$(S\boldsymbol{x}_{j-1}(t))_{2k+1} = \sum_{\ell=-B/2+1}^{B/2} a_\ell x_{j-1,k+\ell}(t), \qquad (S\boldsymbol{x}_{j-1}(t))_{2k} = x_{j-1,k}(t), \tag{8}$$

and

$$\begin{aligned} x_{j,2k+1}(t) &= (S\boldsymbol{x}_{j-1}(t))_{2k+1} + w_{j,2k+1}(t), \\ x_{j,2k}(t) &= x_{j-1,k}(t). \end{aligned} \tag{9}$$

Since $(S\boldsymbol{x}_{j-1}(t))_{2k+1}$ only depends $x_{j-1,\ell}(t)$ with $\ell \in [k-B/2+1, k+B/2]$ the componentwise form of the ODEs (6) for the wavelet vectors can be written

$$\frac{dw_{j,2k+1}(t)}{dt} = G\left(t, x_{j-1,k-B/2+1}(t), \ldots, x_{j-1,k+B/2}(t), w_{j,2k+1}(t)\right), \tag{10}$$

$$\frac{dx_{0,k}(t)}{dt} = F(t, x_{0,k}). \tag{11}$$

Recall that the even wavelet vectors $w_{j,2k}(t)$ are identically zero since S is interpolatory.

For the numerical solution of these equations we must restrict ourselves to a bounded interface, which is described by a finite number of unknowns; for each level j the k-index is restricted to the set $k \in I_j = 0, \ldots, 2^j$, and the level j is bounded by a max level denoted J. We assume that S is well-defined also for this case. Note that, if $B > 2$ we must then either modify S close to the end points of I_j, by using a different mask there, or to consider periodic (closed) interfaces where $x_{j,k+2^j} = x_{j,k}$.

To solve (10) and (11) we apply standard ODE methods, where we use time-steps, denoted $\Delta t_{j,k}$, which vary with j, the level, and in general also with k, the spatial position[1]. We introduce the time levels

$$t^n_{j,k} = n\Delta t_{j,k},$$

and the numerical approximations of marker points and wavelets

$$x^n_{j,k} \approx x_{j,k}(t^n_{j,k}), \qquad w^n_{j,k} \approx w_{j,k}(t^n_{j,k}),$$

where $j = 0,\dots,J$ and $k \in I_j$. Since we are interested in computing the solution up to time T, we also define the integers $M_{j,k}$ such that $T = M_{j,k}\Delta t_{j,k} = t^{M_{j,k}}_{j,k}$. In the basic time doubling strategy we let

$$\Delta t_{j,2k+1} = 2\Delta t_{j-1,k}, \qquad \Delta t_{j,2k} = \Delta t_{j-1,k}, \qquad j = 1,\dots,J, \quad k \in I_{j-1},$$

and

$$\Delta t_{0,0} = \Delta t_{1,1} = \Delta t,$$

where Δt is called the reference time step. Hence, the time step used for wavelets $w_{j,2k+1}$ is doubled in each level, but is constant for all such wavelets within the same level j. For each new level, the marker points corresponding to this level is reconstructed from the marker points on the previous level and the computed wavelets according to (9). Note that even marker points are simply copied from the previous level, $x^n_{j,2k} = x^n_{j-1,k}$, and even wavelet vectors are zero, $w^n_{j,2k} = 0$.

We will call $\boldsymbol{t}_{j,k} = \{t^n_{j,k}\}_{n=0}^{M_{j,k}}$ the *time vector* for point $x_{j,k}$ and wavelet $w_{j,k}$. It contains the time levels on which $x_{j,k}$ and $w_{j,k}$ are approximated. See Figure 2 for an example. Since $\Delta t_{j,2k} = \Delta t_{j-1,k}$ we have $t^n_{j,2k} = t^n_{j-1,k}$ and therefore $\boldsymbol{t}_{j,2k} = \boldsymbol{t}_{j-1,k}$. Note also that in the time doubling case we have the relationship

$$t^n_{j,2k+1} = t^{2n}_{j-1,k}. \tag{12}$$

The overall scheme is thus built on a pairing of a subdivision scheme and a numerical time-stepping scheme. In [25, Theorem 5.1] it was shown that the overall scheme is stable if the order of the subdivision q is strictly larger than the order of the time-stepping p, in the time doubling case. We now describe the scheme in more detail for two particular pairings:

Forward Euler with midpoint method (FE–S_2). Here the first order Forward Euler method ($p = 1$) is coupled with the second order midpoint method ($q = 2$). On the zeroth level, we do

$$x^{n+1}_{0,k} = x^n_{0,k} + \Delta t_{0,k} F(t^n_{0,k}, x^n_{0,k}), \qquad k \in \{0,1\},$$

for $n = 0,1,\dots,M_{0,k}-1$, and at level $j = 1,\dots,J$,

$$w^{n+1}_{j,2k+1} = w^n_{j,2k+1} + \Delta t_{j,2k+1} G(t^n_{j,2k+1}, x^{2n}_{j-1,k}, x^{2n}_{j-1,k+1}, w^n_{j,2k+1}), \qquad k \in I_{j-1},$$

[1]Eventhough the time step is constant in k for the basic method described here, we keep this notation for easier comparison with the adaptive method below.

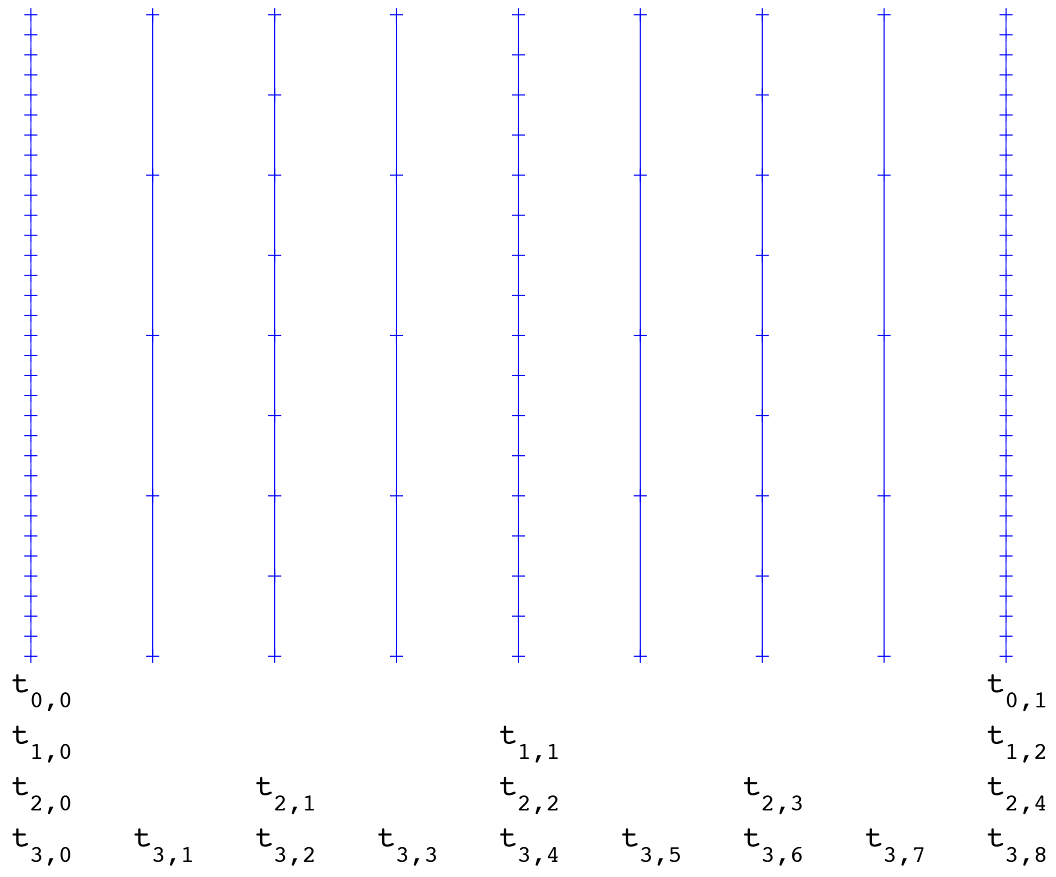

FIGURE 2. Time vectors $\boldsymbol{t}_{j,k}$ in time-doubling case. Time levels in $\boldsymbol{t}_{j,k}$ are indicated by horizontal lines in the figure, with $t = 0$ at the bottom, and $t = T$ at the top. For odd wavelet vectors on level j, approximated at time levels in $\boldsymbol{t}_{j,2k+1}$ the surrounding points on level $j-1$, are approximated at matching time levels in $\boldsymbol{t}_{j-1,k}$.

for $n = 0, 1, \ldots, M_{j,2k+1} - 1$. The point markers $x^n_{j,k}$ are reconstructed from the wavelet vectors and marker points on the previous level via

$$\begin{aligned} x^n_{j,2k+1} &= \frac{x^{2n}_{j-1,k} + x^{2n}_{j-1,k+1}}{2} + w^n_{j,2k+1}, \\ x^n_{j,2k} &= x^{2n}_{j-1,k}. \end{aligned} \tag{13}$$

Runge–Kutta 4 with 6-point scheme (RK4-S$_6$). Here the fourth order Runge–Kutta method ($p = 4$) is coupled with the sixth order Lagrange subdivision scheme ($q = 6$). On the zeroth level, we do

$$\begin{aligned}
\xi_1 &= F(t^n_{0,k}, x^n_{0,k}),\\
\xi_2 &= F(t^{n+1/2}_{0,k}, x^n_{0,k} + \xi_1 \Delta t_{0,k}/2),\\
\xi_3 &= F(t^{n+1/2}_{0,k}, x^n_{0,k} + \xi_2 \Delta t_{0,k}/2),\\
\xi_4 &= F(t^{n+1}_{0,k}, x^n_{0,k} + \xi_3 \Delta t_{0,k}),\\
x^{n+1}_{0,k} &= x^n_{0,k} + \frac{\Delta t_{0,k}}{6}(\xi_1 + 2\xi_2 + 2\xi_3 + \xi_4),
\end{aligned}$$

for $n = 0, 1, \ldots, M_{0,k} - 1$, $k \in \{0, 1\}$, and at level $j = 1, \ldots, J$,

$$\begin{aligned}
\xi_1 &= G\left(t^n_{j,2k+1}, \{x^{2n}_{j-1,\ell}\}^{k+3}_{\ell=k-2}, w^n_{j,2k+1}\right),\\
\xi_2 &= G\left(t^{n+1/2}_{j,2k+1}, \{x^{2n+1}_{j-1,\ell}\}^{k+3}_{\ell=k-2}, w^n_{j,2k+1} + \xi_1 \Delta t_{j,2k+1}/2\right),\\
\xi_3 &= G\left(t^{n+1/2}_{j,2k+1}, \{x^{2n+1}_{j-1,\ell}\}^{k+3}_{\ell=k-2}, w^n_{j,2k+1} + \xi_2 \Delta t_{j,2k+1}/2\right),\\
\xi_4 &= G\left(t^{n+1}_{j,2k+1}, \{x^{2(n+1)}_{j-1,\ell}\}^{k+3}_{\ell=k-2}, w^n_{j,2k+1} + \xi_3 \Delta t_{j,2k+1}\right),\\
w^{n+1}_{j,2k+1} &= w^n_{j,2k+1} + \frac{\Delta t_{j,2k+1}}{6}(\xi_1 + 2\xi_2 + 2\xi_3 + +\xi_4),
\end{aligned}$$

for $n = 0, 1, \ldots, M_{j,2k+1} - 1$, $k \in I_{j-1}$. The point markers $x^n_{j,k}$ are reconstructed from the wavelet vectors and marker points on the previous level via

$$\begin{aligned}
x^n_{j,2k+1} &= \sum_{\ell=-2}^{3} a_\ell x^{2n}_{j-1,k+\ell} + w^n_{j,2k+1},\\
x^n_{j,2k} &= x^{2n}_{j-1,k},
\end{aligned} \tag{14}$$

where $\{a_\ell\}$ is the mask for S_6 given in Section 2.

Note that in order to evaluate the reconstruction step (9) accurately, the point markers $x_{j-1,k-B/2+1}(t), \ldots, x_{j-1,k+B/2}(t)$ and $w_{j,2k+1}(t)$ that appear in the right hand side, should all be approximated on the *same* time level. When the time doubling strategy is used, this is never a problem, since by (12) we have $\boldsymbol{t}_{j,2k+1} \subset \boldsymbol{t}_{j-1,k}$. Approximations of $x_{j-1,k}(t)$ are therefore always available on the same time levels as approximations of $w_{j,2k+1}(t)$, cf. Figure 2. In fact, for Runge–Kutta 4, more is needed, since G is then evaluated also at half time steps, $t^{n+1/2}_{j,2k+1}$. But, still by (12), we have $t^{n+1/2}_{j,2k+1} = t^{2n+1}_{j-1,k} \in \boldsymbol{t}_{j-1,k}$, and this is again not a problem with the time doubling strategy. For the general adaptive case described below, however, $\boldsymbol{t}_{j,2k+1} \not\subset \boldsymbol{t}_{j-1,k}$ and interpolation must instead be used, cf. Figure 4.

4.1. **Cost and Accuracy.**

Let us now give the cost and the approximation error estimate for the methods above. Suppose the total number of points is $N = 2^J$ where J is the number of levels. The cost at each level is simply the number of wavelets $\{w_{j,2k+1}\}$ to compute multiplied by the number of time steps per ODE, $\lfloor T/\Delta t_{j,2k+1} \rfloor$. Noting that by construction $\Delta t_{j,2k+1} = 2^j \Delta t$, the total cost of propagating the interface is then the sum of the costs on every level, i.e.

$$\text{cost} \sim \sum_{j=0}^{J} \sum_{k=0}^{2^j} \left\lfloor \frac{T}{\Delta t_{j,2k+1}} \right\rfloor \sim T \sum_{j=0}^{J} \frac{2^j}{2^j \Delta t} \sim O\left(\frac{\log_2 N}{\Delta t}\right). \tag{15}$$

The cost of the method is the sum of the total propagation cost and the cost for reconstructing the interface at the final time which is $O(N)$. This gives the cost $O(\log_2 N/\Delta t + N)$. Hence, this is significantly improved as compared to the standard methods where the corresponding cost is $O(N/\Delta t)$.

Let us now consider the approximation error for a p-th order method. We use the estimate (7) that wavelet vectors and their derivatives decay as 2^{-qj}, where q is the order of the subdivision scheme used to compute predicted points and j is the level. Let $\tau_{j,k}^n$ be the local truncation error in time step n for wavelet coefficient k at level j, which we assume is proportional to the $p+1$-th derivative of the exact solution. Then, assuming stability, the global error is formally estimated as

$$\begin{aligned}(16)\qquad \varepsilon_J &\leq \sum_{j=0}^{J} \sum_{n=0}^{[T/\Delta t_j]} \max_{k\in I_j} |\tau_{j,2k+1}^n| \leq \sum_{j=0}^{J} \sum_{n=0}^{[T/\Delta t_{j,2k+1}]} \max_{k\in I_j} \left| \Delta t_j^{p+1} \frac{d^{p+1} w_{j,2k+1}}{dt^{p+1}} \right| \\ &\overset{(7)}{\leq} CT\Delta t^p \sum_{j=0}^{J} 2^{(p-q)j} \leq C'\Delta t^p.\end{aligned}$$

In the last step we assumed $p < q$, i.e. a stable pairing, so that the sum is convergent. The estimate is then independent of the number of points N and the accuracy is $O(\Delta t^p)$, where $p = 1$ for the FE–S_2 method ($q = 2$) and $p = 4$ for the RK4–S_6 method ($q = 6$). This result is proved rigorously in [25, Theorem 5.1].

4.2. **Surfaces in Three Dimensions.**

The method described above for curves in two dimensions can be generalized to the case where the interface is a surface in three dimensions. We only give a brief account of this here and refer to [24] for more details.

The multiresolution description of a curve given in Section 3 is replaced by a multiresolution of a triangulated surface, where we assume that finer triangulations are obtained precisely by splitting each triangle on level j into four sub triangles on level $j + 1$. Hence, we assume for simplicity that there are no interior extra-ordinary vertices on level $j \geq 1$. The relationship between triangles, marker points and wavelet vectors on level j and $j + 1$ are illustrated in Figure 3 and given by the steps:

(1) Start from a triangulation on level j, with two adjacent triangles; see Figure 3a.
(2) On level $j+1$ new nodes appear roughly in between old nodes. Let the wavelet vectors be the difference between new nodes on level $j + 1$ and the average of nodes on level j; see Figure 3b.
(3) Connect the new points to form a refined triangulation on level $j + 1$; see Figure 3c.

This multiresolution scheme gives a $\sim 2^{-2j}$ decay of the wavelet vectors and their time derivatives as the level j increases. Multiresolution schemes can also be built on higher order subdivision schemes for surfaces like Butterfly [12, 31], Loop [20] and Catmull–Clarke [2], which would give faster decay. Once the multiresolution description is given, the fast interface tracking method works in the same way as for curves. Formally, the error is still controlled as in (16). However, the number of wavelet vectors increases faster with j for surfaces than for curves: $\sim 2^{2j}$ (four sub triangles) instead of $\sim 2^j$ (one intermediate point). Therefore, the cost increases faster than in (15), although still slower than for standard interface tracking. See [24] for a longer discussion of this.

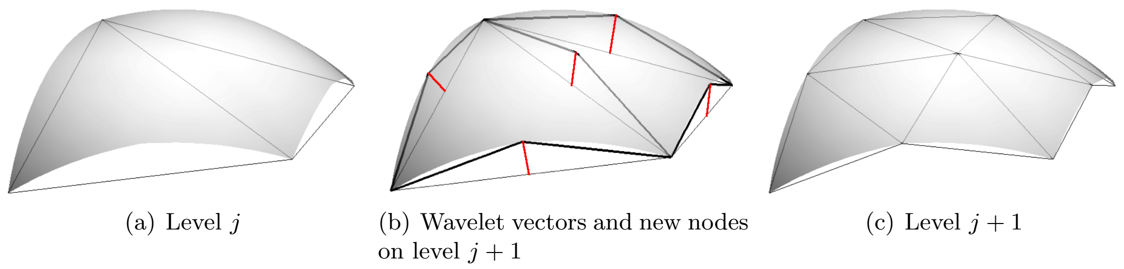
(a) Level j (b) Wavelet vectors and new nodes on level $j+1$ (c) Level $j+1$

FIGURE 3. Multiresolution description of a surface; definition of the wavelet vectors.

5. Time Adaptive Method

In the basic method described above the time step was doubled at every level j. This was motivated by the known asymptotic decay rate of the wavelet vectors and works well when wavelets at the same level all change in time at approximately the same rate. However, when this is not true, the method will be inefficient, since in order to maintain accuracy, the reference time step has to be based on the fastest rate of change. Moreover, if the interface is only piecewise smooth, then, in a neighborhood of the non-smooth points, the wavelets and their time derivatives will decay slower than elsewhere. The error will then not be bounded when N increases, as in (16), and the basic method will not work.

In this section we propose an adaptive time stepping scheme to tackle the problems of the basic method. In the new numerical method there is no predetermined doubling of the time step in each level. Instead, the method selects the best time step itself, based on the local rate of change of the computed wavelet vector solution. In general, it will therefore pick longer time steps for higher levels, but only if justified by the actual solution trajectory. In principle, such an adaptive numerical method will perform approximately the same as the basic method in smooth cases where the wavelets decay uniformly in space, but it would be more robust and optimal in cases where the decay is heterogeneous. It will also be able to handle piecewise smooth interfaces, including e.g. interfaces with corners.

5.1. **Adaptive ODE Solver.**

We will use a rudimentary adaptive ODE solver as basis for the time adaptive fast method. The method is taken from [17]. There are many more advanced adaptive methods, such as embedded Runge-Kutta methods [9, 10] and duality based adaptive methods [1]. For proof of concept, this simple method will, however, be sufficient.

Consider the general initial value problem

$$y' = f(t, y), \qquad t \in [0, T], \qquad y(0) = y^0. \tag{17}$$

The numerical approximation is denoted

$$y^n \approx y(t^n), \qquad t^n = \begin{cases} 0, & n = 0, \\ \sum_{i=0}^{n-1} \Delta t^i, & n \geq 1, \end{cases}$$

where Δt^i is the i:th time step. The adaptivitiy is governed by an estimate of the local truncation error τ^n; the goal is to have approximately the same contribution to the total error in each step, meaning that the local truncation error scaled by the local time step, should be approximately constant, equal to a prescribed tolerance TOL,

$$\frac{\tau^n}{\Delta t^n} \approx \text{TOL}.$$

The adaptive method is built on an explicit one-step method $\Phi(\Delta t, t, y)$,

$$y^{n+1} = y^n + \Delta t^n \Phi(\Delta t^n, t^n, y^n).$$

To estimate the local truncation error τ^n in step n we use a time step Δt_* which we initially set to equal the current time step Δt^n. (The initial value of Δt_* at $n = 0$ must be provided by the user.) We then start by taking one step with Δt_*,

$$y_1 = y^n + \Delta t_* \Phi(\Delta t_*, t^n, y^n). \tag{18}$$

Next, we make two steps with half the time step $\Delta t_*/2$,

$$y_m = y^n + \frac{1}{2}\Delta t_* \Phi\left(\frac{1}{2}\Delta t_*, t^n, y^n\right), \qquad y_2 = y_m + \frac{1}{2}\Delta t_* \Phi\left(\frac{1}{2}\Delta t_*, t^n + \frac{1}{2}\Delta t_*, y_m\right). \tag{19}$$

From these two values we estimate the local truncation error for y_2 as

$$\tau_* \approx \frac{y_2 - y_1}{2^p - 1},$$

where p is the order of the method Φ. If $|\tau_*|/\Delta t_* > \text{TOL}$ the step is rejected. Let $\tilde{\tau}_*$ be the corresponding local truncation error with time step $\Delta\tilde{t}_*$ instead of Δt_*. Based on the assumption that $\tau_* \approx C(\Delta t_*)^{p+1}$ for some C independent of Δt_*, we get that

$$\frac{\tilde{\tau}_*}{\Delta\tilde{t}_*} \approx C(\Delta\tilde{t}_*)^p \approx \frac{\tau_*(\Delta\tilde{t}_*)^p}{(\Delta t_*)^{p+1}} \approx \text{TOL} \quad \text{if} \quad (\Delta\tilde{t}_*)^p = \frac{(\Delta t_*)^{p+1}\text{TOL}}{|\tau_*|}.$$

We therefore reduce the time step by a factor β,

$$\Delta t_* \to \beta\Delta t_*, \qquad \beta = \left(\frac{0.9\Delta t_* \text{TOL}}{|\tau_*|}\right)^{1/p},$$

and repeat the calculations above of y_1, y_2 and τ_*. Eventually we get $|\tau_*|/\Delta t_* < \text{TOL}$ and the step is accepted. We then finally set

$$y^{n+1} = y_2 - \tau_*, \qquad \Delta t^{n+1} = \beta\Delta t_*, \qquad \beta = \left(\frac{\Delta t_* \text{TOL}}{|\tau_*|}\right)^{1/p}.$$

In practical calculations we cap the rate of time step change in each iteration to five, by modifying the β factors as $\beta \to \max(\min(\beta, 5), 1/5)$.

In this paper we will use two one-step methods: Forward Euler ($p = 1$) and Runge–Kutta 4 ($p = 4$). Note that in the adaptive method we only consider absolute errors, not relative errors. This is important for the efficiency of the method since we exploit the fact that small errors for small wavelet coefficients can be neglected.

5.2. **Adaptive Method Applied to the Wavelet ODEs.** In this section, we describe how the time adaptive ODE solvers in Section 5.1 can be applied to (6) together with (5), i.e. to the ODEs used in the basic fast interface tracking method. Both a Forward Euler and a Runge-Kutta 4 version will be shown. Note that for these schemes the time step is chosen by the methods themselves; it is not known in advance as in the basic method. We choose only Δt^0 and in general $t^n \neq n\Delta t^0$.

To describe the methods, we use the same notation as for the basic method, with the difference that the time steps are also indexed by n as in Section 5.1. Note that the time step will in general change in time (n), by level of detail (j) but also vary for different points and vectors at the same level (k). We define

$$t_{j,k}^n = \sum_{i=0}^{n-1} \Delta t_{j,k}^i, \qquad t_{j,k}^{n+1/2} = t_{j,k}^n + \frac{1}{2}\Delta t_{j,k}^n, \qquad T = \sum_{i=0}^{M_{j,k}-1} \Delta t_{j,k}^i = t_{j,nK}^{M_{j,k}},$$

where $\Delta t_{j,k}^i$ is the local time step and the integer $M_{j,k}$ is the length of the time vector $\boldsymbol{t}_{j,k} = \{t_{j,k}^n\}_{n=0}^{M_{j,k}}$.

In the same way as for the basic method, we solve (6) level by level. We first calculate the edge points $\boldsymbol{x}_0(t) = \{x_{0,0}(t), x_{0,1}(t)\}$ by applying the adaptive method to the right ODE in (6) in exactly the same way as in the previous section. Next, we compute wavelet vectors $\boldsymbol{w}_1(t)$, $\boldsymbol{w}_2(t)$, etc. using the adaptive method applied to the left ODE in (6).

A major difficulty when solving for $\boldsymbol{w}_j(t)$ compared to the basic method is the fact that wavelet vectors are in general given at different time levels, see Figure 4. This makes the reconstruction step more difficult. For example, in order to calculate $\{w_{j,2k+1}^n\}$ with the FE–S_2 scheme we need to evaluate the function G at the time levels $\{t_{j,2k+1}^n\} = \boldsymbol{t}_{j,2k+1}$ and at the neighboring points on the previous level, $x_{j-1,k}, \dots, x_{j-1,k+1}$. However, since wavelets on the previous level were computed at time levels $\{t_{j-1,k}^n\} = \boldsymbol{t}_{j-1,k}$ and as mentioned above $\boldsymbol{t}_{j,2k+1} \not\subset \boldsymbol{t}_{j-1,k}$ (see Figure 4) we cannot find them by the simple reconstruction formula (13). For RK4–S_6 we need to evaluate G at all the points $x_{j-1,k-2}, \dots, x_{j-1,k+3}$ and also at half time-steps $t_{j,2k+1}^{n+1/2}$, which means (14) cannot be used.

The solution to the reconstruction problem is to use interpolation. It is, however, important to only interpolate wavelets $\boldsymbol{w}_j(t)$, not points $\boldsymbol{x}_j(t)$, in order to avoid large errors. Indeed, the size of the interpolation error for a p-th order scheme depends on the p-th derivative of the interpolated function multiplied by the local time step to the power of p. Recall that $\boldsymbol{w}_j(t)$ and $\boldsymbol{x}_j(t)$ are typically given on time levels separated by big time steps, when j is large. Interpolation would therefore introduce large errors for $\boldsymbol{x}_j(t)$. However, interpolating wavelets $\boldsymbol{w}_j(t)$ is safe since, by the method's construction, the local time steps and the size of the derivatives of the wavelet vectors are balanced such that their product is small.

Let us now describe the interpolation. We denote the piecewise cubic interpolation operator by Π. Then we set

$$\bar{x}_{0,k}(t) = \left(\prod\{\{x_{0,k}^n\}_{n=0}^{M_{0,k}}\}\right)(t), \qquad \bar{w}_{j,k}(t) = \left(\prod\{\{w_{j,k}^n\}_{n=0}^{M_{j,k}}\}\right)(t), \tag{20}$$

and

$$\bar{\boldsymbol{x}}_0(t) = \{\bar{x}_{0,k}(t)\}_{k\in I_0}, \qquad \bar{\boldsymbol{w}}_j(t) = \{\bar{w}_{j,k}(t)\}_{k\in \bar{I}_j}.$$

Since for the exact solution we have the reconstruction formula

$$\begin{aligned}\boldsymbol{x}_j(t) &= \boldsymbol{w}_j(t) + S\boldsymbol{x}_{j-1}(t) = \boldsymbol{w}_j(t) + S(\boldsymbol{w}_{j-1}(t) + S\boldsymbol{x}_{j-2}(t)) \\ &= \boldsymbol{w}_j(t) + \sum_{i=1}^{j-1} S^{j-i}\boldsymbol{w}_i(t) + S^j\boldsymbol{x}_0(t).\end{aligned}$$

we define the interpolated approximation $\bar{\boldsymbol{x}}_j(t)$ for $j \geq 1$ as

$$\bar{\boldsymbol{x}}_j(t) = \bar{\boldsymbol{w}}_j(t) + \sum_{i=1}^{j-1} S^{j-i}\bar{\boldsymbol{w}}_i(t) + S^j\bar{\boldsymbol{x}}_0(t), \qquad \bar{\boldsymbol{x}}_j(t) = \{\bar{x}_{j,k}(t)\}_{k\in I_j}. \tag{21}$$

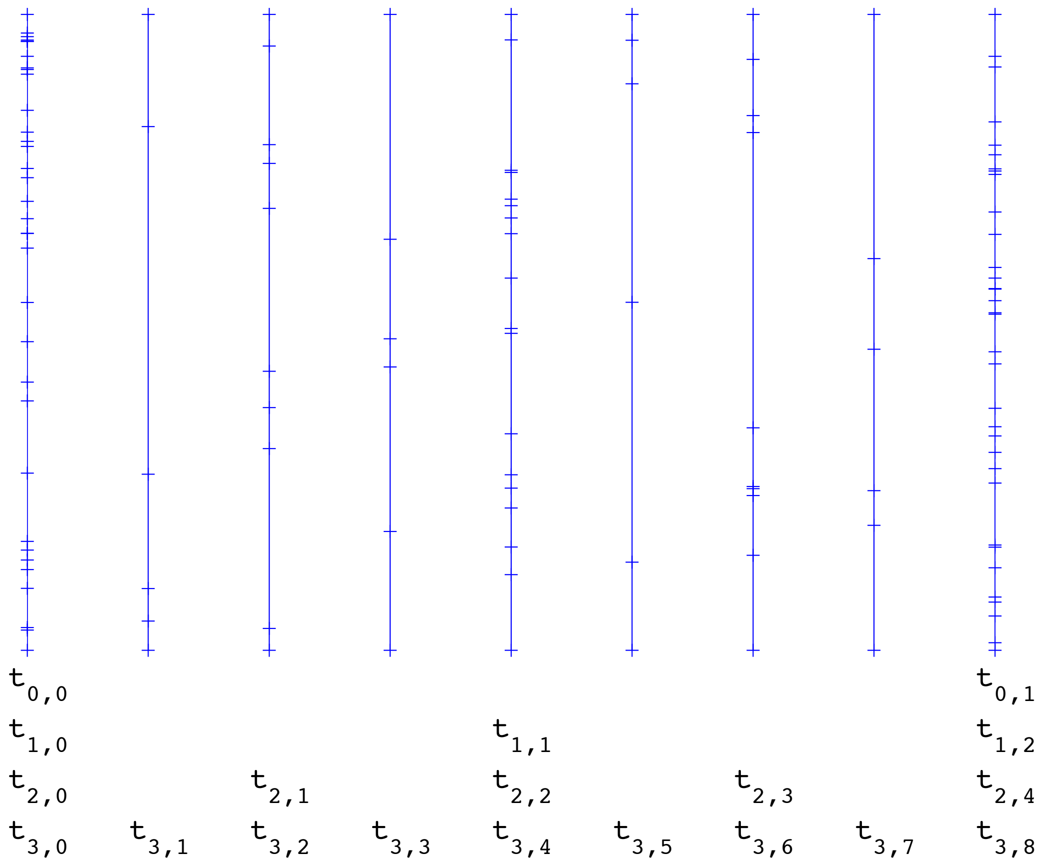

FIGURE 4. Time vectors $\boldsymbol{t}_{j,k}$ in adaptive case. Time levels in $\boldsymbol{t}_{j,k}$ are indicated by horizontal lines in the figure, with $t = 0$ at the bottom, and $t = T$ at the top. In order to calculate $w^n_{j,2k+1}$ values of $x_{j-1,k}$ and $x_{j-1,k+1}$ are needed at time level $t^n_{j,2k+1}$. Since in general $\boldsymbol{t}_{j,2k+1} \not\subset \boldsymbol{t}_{j-1,k}$ these are not available from earlier calculations, but need to be interpolated.

Hence, we only interpolate $\boldsymbol{x}_0$ and wavelets $\boldsymbol{w}_j$. To do this we use the full hierarchical description to reconstruct points. We stress again that

$$\bar{x}_{j,k}(t) \neq \left(\prod \{\{x^n_{j,k}\}_{n=0}^{M_{j,k}}\}\right)(t), \qquad j \geq 1.$$

Finally, we give the details of the implementation of the adaptive methods for our problem.

Adaptive Forward Euler with midpoint method (A–FE–S$_2$). On the zeroth level, we apply the described adaptive FE method to

$$\frac{dx_{0,k}}{dt} = F(t, x_{0,k}), \qquad k \in \{0, 1\}, \tag{22}$$

and at level $j \geq 1$ to

$$\frac{dw_{j,2k+1}}{dt} = G(t, x_{j-1,k}, x_{j-1,k+1}, w_{j,2k+1}), \qquad k \in I_{j-1}. \tag{23}$$

When the function G is evaluated, $\{x_{j-1,l}\}_{l=k}^{k+1}$ have to be interpolated in the way described above. Hence, the main Forward Euler step (18) in the adaptive method is implemented as

$$\Delta t_* = \Delta t^n_{j,2k+1}, \qquad y_1 = w^n_{j,2k+1} + \Delta t_* G(t^n_{j,2k+1}, \{\bar{x}^n_{j-1,l}\}_{l=k}^{k+1}, w^n_{j,2k+1})$$

where $n \le M_{j,2k+1} - 1$ and $\bar{x}^n_{j-1,k} = \bar{x}_{j-1,k}(t^n_{j,2k+1})$. The steps (19) are implemented analogously.

Adaptive Runge–Kutta 4 with 6-point scheme (A–RK4–S_6). On the zeroth level, we apply the described adaptive RK 4 method to (22) and at level $j \ge 1$ to (23). Again $\{x_{j-1,l}\}_{l=k-2}^{k+3}$ have to be interpolated and the main adaptive step (18) reads

$$\begin{aligned}
\Delta t_* &= \Delta t^n_{j,2k+1},\\
\xi_1 &= G(t^n_{j,2k+1}, \{\bar{x}^n_{j-1,l}\}_{l=k-2}^{k+3}, w^n_{j,2k+1})\\
\xi_2 &= G(t^{n+1/2}_{j,2k+1}, \{\bar{x}^{n+1/2}_{j-1,l}\}_{l=k-2}^{k+3}, w^n_{j,2k+1} + \xi_1 \Delta t/2)\\
\xi_3 &= G(t^{n+1/2}_{j,2k+1}, \{\bar{x}^{n+1/2}_{j-1,l}\}_{l=k-2}^{k+3}, w^n_{j,2k+1} + \xi_2 \Delta t/2)\\
\xi_4 &= G(t^{n+1}_{j,2k+1}, \{\bar{x}^{n+1}_{j-1,l}\}_{l=k-2}^{k+3}, w^n_{j,2k+1} + \xi_3 \Delta t)\\
y_1 &= w^n_{j,2k+1} + \frac{\Delta t_*}{6}(\xi_1 + 2\xi_2 + 2\xi_3 + \xi_4),
\end{aligned}$$

where $n \le M_{j,2k+1} - 1$ and $\bar{x}^n_{j-1,k} = \bar{x}_{j-1,k}(t^n_{j,2k+1})$. The steps (19) use interpolated values for $\{x_{j-1,l}\}_{l=k-2}^{k+3}$ in the same way.

The implementation of the adaptive Forward Euler method for surfaces, when the interface is represented by a triangulation as described in the last part of the previous section, do not differ from the implementation of the adaptive Forward Euler method for curves when the two point subdivision scheme is used to represent the interface. Higher order subdivision schemes and the adaptive Runge-Kutta method for surfaces are not considered in this paper.

5.3. **Cost and Accuracy.** We will here make a simplified analysis of the cost and accuracy for the adaptive method, when it is applied to an idealized uniformly smooth problem where

$$\frac{d^p w_{j,k}(t)}{dt^p} \approx c 2^{-jq} \tag{24}$$

is a good approximation for all t, j and k. For simplicity, we ignore the interpolation error in the analysis.

For the cost, we first consider the cost of taking one step in the numerical method. This cost is dominated by the new interpolation step (21). We note that for a subdivision scheme S, the repeated application S^j is a local operation, and in its matrix representation each row has at most a fixed number of non-zero entries, which is independent of j. For instance, if $S = S_2$ then, using (5) we obtain

$$\begin{aligned}
\bar{x}_{j,2k+1}(t) &= \bar{w}_{j,2k+1}(t) + \frac{1}{2}\left(\bar{x}_{j-1,k}(t) + \bar{x}_{j-1,k+1}(t)\right)\\
&= \bar{w}_{j,2k+1}(t) + \frac{1}{2}\left(\bar{w}_{j-1,k}(t) - S\bar{\boldsymbol{x}}_{j-2}(t) + \bar{w}_{j-1,k+1}(t) - S\bar{\boldsymbol{x}}_{j-2}(t)\right)\\
&= \sum_{\ell=0}^{j}\sum_{m=0}^{2^\ell} \alpha_{\ell,m,j,k}\bar{w}_{\ell,m}(t) + \alpha_{0,0,j,k}\bar{x}_{0,0}(t) + \alpha_{0,1,j,k}\bar{x}_{0,1}(t),
\end{aligned}$$

where the number of non zero $\{\alpha_{\ell,m,j,k}\}$ for fixed (ℓ, j, k) is at most two. Consequently, the cost to evaluate (21) for some $x_{j,k}(t)$ is directly proportional to the length j of the sum, which is bounded by $J = \log_2 N$. Hence, the interpolation cost and therefore the cost of taking one step is of the order $O(\log N)$.

Furthermore, the adaptive method will strive to select a time-step $\Delta t^n_{j,k}$ such that the local truncation error satisfies

$$\frac{\tau^n_{j,k}}{\Delta t^n_{j,k}} \approx (\Delta t^n_{j,k})^p \frac{d^{p+1} w_{j,k}(t^n_{j,k})}{dt^{p+1}} \approx \text{TOL}. \tag{25}$$

From this relation and the assumption (24) we get the time step size used by the adaptive method

$$\Delta t^n_{j,k} \sim \text{TOL}^{1/p} 2^{qj/p}, \tag{26}$$

which is now (approximately) independent of k and n. In the same way as in (15), upon including the $\log N$-factor, we then get the following expression for the propagation cost

$$\text{cost} \sim \sum_{j=0}^{J} 2^j \left\lfloor \frac{T}{\Delta t^n_{j,k}} \right\rfloor \log N = T \log N \sum_{j=0}^{J} 2^{j(1-q/p)} \text{TOL}^{-1/p} \sim \text{TOL}^{-1/p} \log N,$$

as long as $q > p$, which is the same stability condition as in the basic method. In addition to the propagation cost, there is also a reconstruction cost from interpolating all N points at the final time. Since each interpolation cost $O(\log N)$, we obtain the total cost

$$O(\text{TOL}^{-1/p} \log N + N \log N).$$

Note that if the solution is sought at any additional intermediate time, only the reconstruction part is needed and the complexity is $O(N \log N)$.

As for the accuracy, we use (25) and make the same formal error analysis as in (16). We get the error estimate,

$$\varepsilon_J \le \sum_{j=0}^{J} \max_{k \in I_j} \sum_{n=0}^{M_{j,2k+1}} |\tau^n_{j,2k+1}| \le c \sum_{j=0}^{J} \max_{k \in I_j} \sum_{n=0}^{M_{j,2k+1}} \text{TOL} \Delta t^n_{j,2k+1} = cTJ\,\text{TOL} \sim \text{TOL} \log N. \tag{27}$$

Hence, also for the adaptive method we should expect an error that is essentially (upto a log-factor) bounded independently of N.

Remark 5.1. We could also use different TOL at different levels of detail. For instance, if we take

$$\text{TOL}_j = \text{TOL}\, 2^{-\alpha j}, \qquad \alpha > 0,$$

we get, in the same way as above, an estimate of the accuracy that is independent of N

$$\varepsilon_J \le c \sum_{j=0}^{J} \max_{k \in I_j} \sum_{n=0}^{M_{j,2k+1}} \text{TOL}_j \Delta t^n_{j,2k+1} = cT\,\text{TOL} \sum_{j=0}^{J} 2^{-j\alpha} \le c' T\,\text{TOL}, \tag{28}$$

and, if $\alpha < q - p$, a cost estimate with the same bound,

$$\text{cost} \sim T \log N \sum_{j=0}^{J} 2^{j(1-q/p)} \text{TOL}_j^{-1/p} \sim T\,\text{TOL} \log N \sum_{j=0}^{J} 2^{j(1-q/p+\alpha/p)} \sim \text{TOL}^{-1/p} \log N.$$

In principal, we can hence remove the $\log N$ factor in the error estimate in this way if $q > p$.

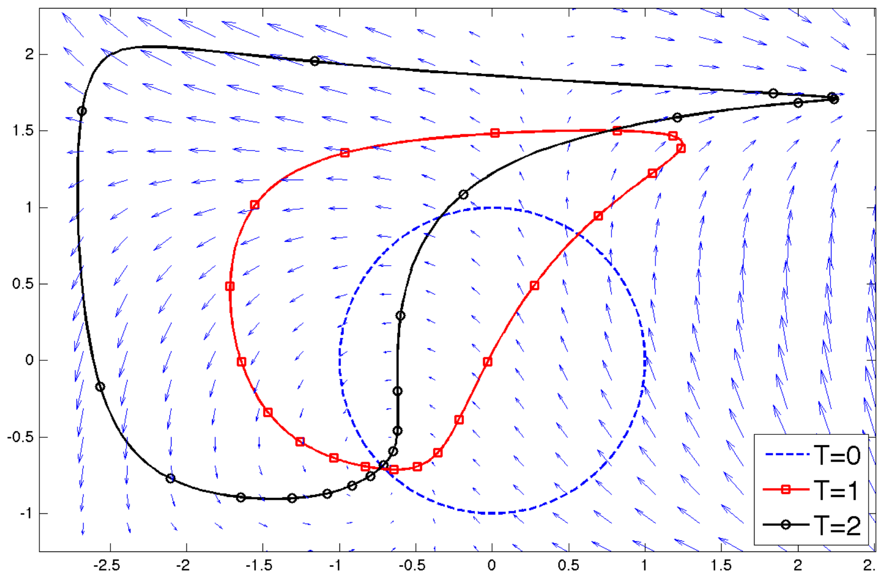

Figure 5. Example 1: The solution at times $T = 0, 1, 2$ for the velocity field (29) (overlaid).

6. Numerical Examples

In this section, we compare the adaptive methods A–FE–S_2 and A–RK4–S_6 against the corresponding basic methods FE–S_2 and RK4–S_6 for different velocity fields F. We plot the error and the cost for different tolerances to verify the theoretical predictions in previous sections. To make a fair comparison we select the tolerance in the adaptive method and the reference time step in the basic method such that the resulting errors are of (roughly) the same size. Our implementation of the adaptive method is to show a proof of concept, and it is not optimized. Therefore, for the computational cost we simply plot the total lengths of all time vectors, which we take as proxy for the actual computational time. We also plot time vector lengths where we include the steps that were rejected in the adaptive solver, calling this the cost with hidden steps.

6.1. **Example 1.**

We begin with a problem where the velocity field is given by

$$F(t, \mathbf{x}) = \begin{bmatrix} y \sin(x) - 0.5 \\ (x + 0.2) \cos(y) + 0.4 \end{bmatrix}. \tag{29}$$

We let the initial curve be the unit circle. The solution at $T = 0, 1, 2$ and the vector field F are shown in Figure 5. Since there are large differences in the velocity along the interface, adaptivity will be important. In order to compare the adaptive and basic method, we will have to choose a very small reference time step which makes the basic method more expensive. Hence, we expect the adaptive method to behave much better.

We first solve (6) till $T = 1$ with F as in (29) using the adaptive Forward Euler method (A–FE–S_2). The decay of wavelet vectors is shown in Figure 6 (left). The decay is $\sim 2^{-2j}$ as expected. In Figure 7 we plot the error and the cost as functions of N for the adaptive method and the corresponding basic method (FE–S_2). As predicted by the theory, both

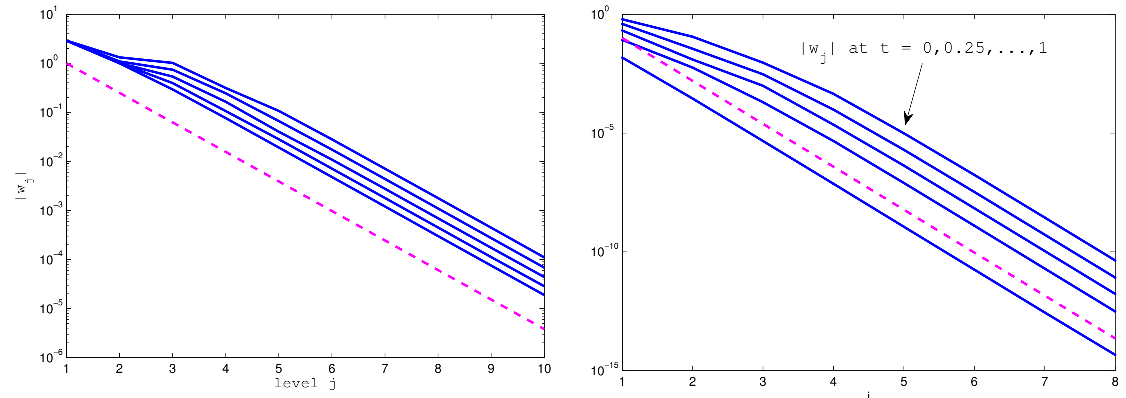

FIGURE 6. Example 1: Decay of wavelet coefficients; $\max_k |w_{j,k}(t)|$ plotted at times $t = 0, 0.25, \ldots, 1$ for the A–FE–S_2 scheme (left) and the A–RK4–S_6 scheme (right). The dashed line represents 2^{-2j} (left) and 2^{-6j} (right).

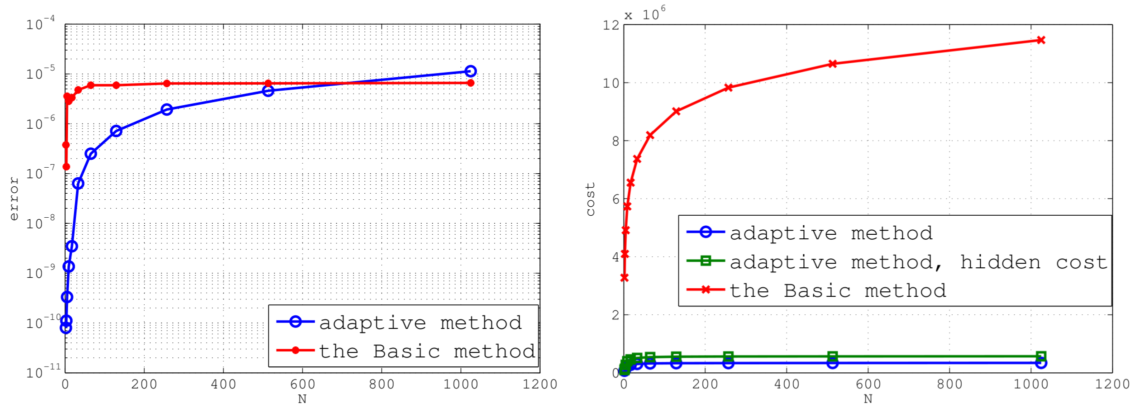

FIGURE 7. Example 1: Cost and accuracy of the adaptive Forward Euler method (A–FE–S_2) for $T = 1$. Plots of the error as a function of N (left) and the cost as a function of N (right) for A–FE–S_2 and the corresponding basic method (FE–S_2) with reference time step $\Delta t_{\mathrm{ref}} = 1.2 \cdot 10^{-6}$.

errors and costs are bounded essentially independently of the number of points. The cost of the adaptive method is clearly lower than the cost of the basic method.

We next solve the problem until $T = 2$ with the adaptive 4th order Runge-Kutta method with the 6-point subdivision scheme (A–RK4–S_6). The decay of the wavelets is $\sim 2^{-6j}$ as expected; see Figure 6 (right). In Figure 8, we plot the error and the cost of the adaptive method and the basic version (RK4–S_6) as a function of N. Again, the error and the cost of our method behave as expected: the error is be bounded almost independently of the number of points N, and the cost of the basic method is significantly higher.

Figure 9 shows how the time step size changes within and between levels. It is notable that the size of the time step varies over several magnitudes within a level. The adaptive solver detects the difference in the velocity of different points on the same level and adjusts the time step according to that. The basic method, on the other hand, will use the same time

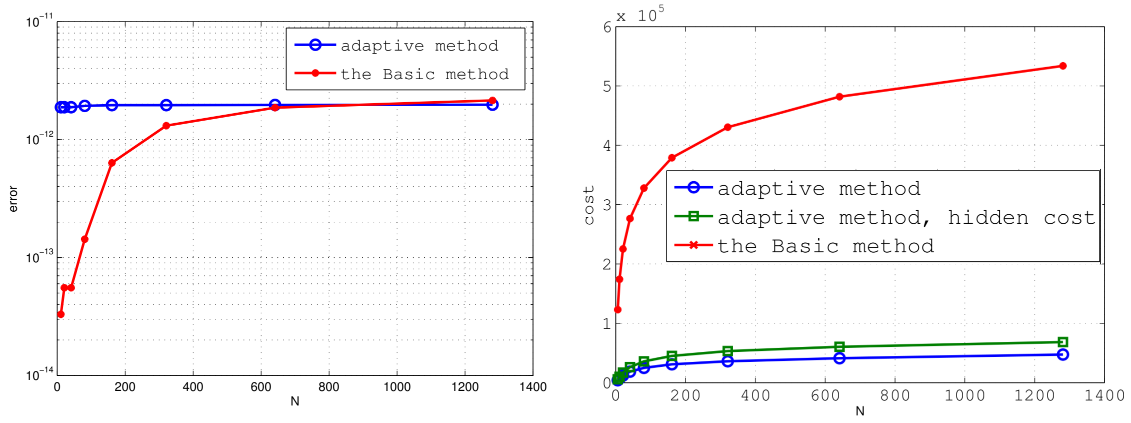

FIGURE 8. Example 1: Cost and accuracy of the adaptive Runge–Kutta 4 method (A–RK4–S_6) for $T = 2$. Plots of the error as a function of N (left) and the cost as a function of N (right) for A–RK4–S_6 and the corresponding basic method (RK4–S_6) with reference time step $\Delta t_{\text{ref}} = 10^{-4}$.

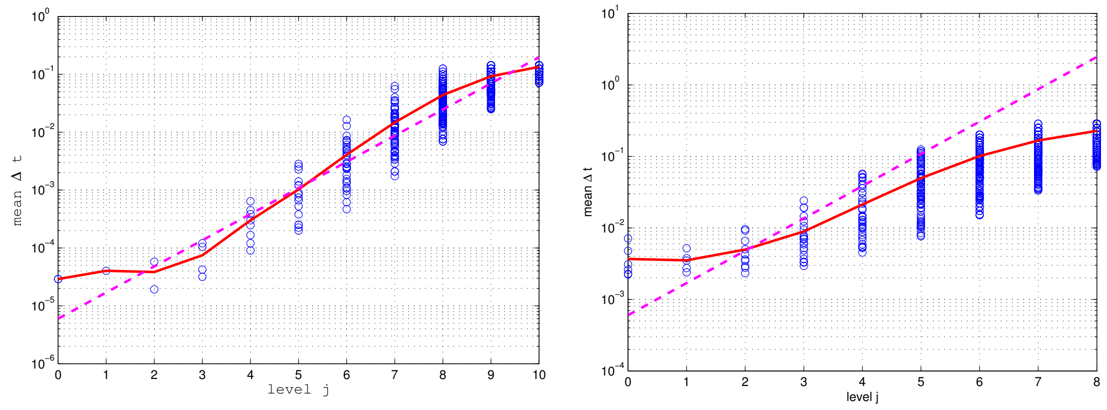

FIGURE 9. Example 1: Time steps as a function of level j when we use the adaptive Forward Euler method (left) the adaptive Runge–Kutta 4 method (right). Circles show the distribution of time steps, and the solid line indicates the average time step size. The dashed line is the theoretical time step for smooth problems given by (26). The reference time steps and final times are as in Figures 7 and 8 respectively.

step for all points within the same level. To ensure the same error level, this time step size would be in the lower part of the time step variations in the adaptive method. Furthermore, the average time step on each level roughly follows (26), the time step size that the adaptive method would choose for uniformly smooth problems. Note also that the cap on the rate of time step change means that there is a lower and upper bound for the time step employed in a fixed time interval. This explains why the average time step curve flattens out for small and high levels, respectively.

In Figure 10 we finally show an example of the propagation of an open interface, where the initial interface is $y = -x$ and $x \in [-1, 1]$. The high order adaptive Runge–Kutta 4 method

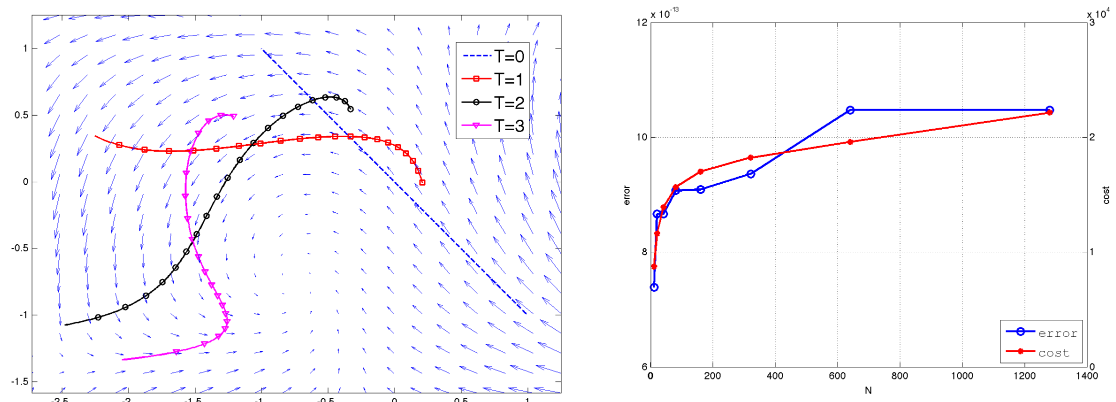

FIGURE 10. Example 1: Open interface. Cost and accuracy of the adaptive Runge–Kutta 4 method (A–RK4–S_6). Plots of the solution at times $T = 0, 1, 2, 3$ (left); the cost and accuracy at $T = 3$ as a function of N (right).

A–RK4–S_6 is used, with the modified masks in (4) employed at the interface edges. The cost and accuracy curves follow the same pattern as for the closed interfaces.

6.2. **Example 2.** In this example we keep the initial curve as the unit circle but change the velocity field to

$$F(t, \mathbf{x}) = \begin{bmatrix} 6y^3 \\ -x \end{bmatrix}. \tag{30}$$

The initial circle, the solution at $T = 1, 2$ and the vector field F are shown in Figure 11.

As in Example 1 we solve the problem using the adaptive Forward Euler method A–FE–S_2 until $T = 1$ and the adaptive Runge-Kutta method A–RK4–S_6 until $T = 2$. We then compare with the corresponding basic methods FE–S_2 and RK4–S_6. The error and cost curves are shown in Figures 12 and 13, respectively. As for Example 1, they show that the error in both cases is essentially bounded independently of the number of points on the interface, and that the adaptive versions are significantly faster than the basic versions.

In addition, we solve the same problem with standard front tracking method where we represent the interface by marker points and solve (2) using the time adaptive Forward Euler method. In Figure 14 (left) the cost is compared to the cost of the basic and adaptive fast interface tracking methods (FE–S_2 and A–FE–S_2) where the same tolerance and final time was used. The cost of the standard method grows linearly with N, and becomes more expensive than both the fast methods, for large N.

The time vectors $\boldsymbol{t}_{j,k}$ for the A–FE–S_2 computation is shown in Figure 14 (right). Note that the distribution of time instances is quite different from the time doubling strategy in Figure 2. More points are used where the interface moves rapidly. The solid horizontal lines are artifacts of the cap on the rate of time step change (max a factor five).

6.3. **Example 3.** We now consider the velocity field given by (29), but we choose an initial interface with four sharp corners given by $y = \pm(1 - |x|^{1/2})$ for $x \in [-1, 1]$. This interface is thus only piecewise smooth. The initial interface and the solution at $T = 1$ is shown in Figure 15.

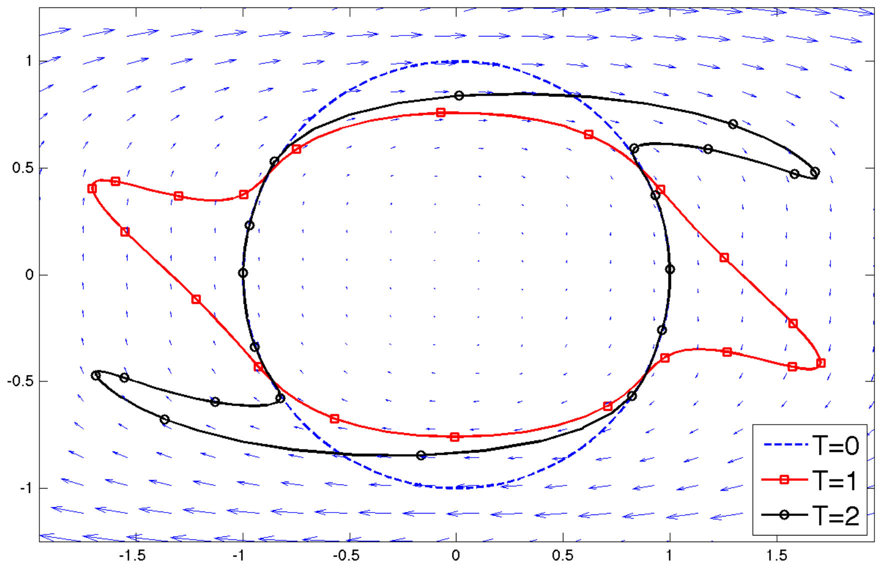

FIGURE 11. Example 2: The solution at times $T = 0, 1, 2$ for the velocity field (30) (overlaid).

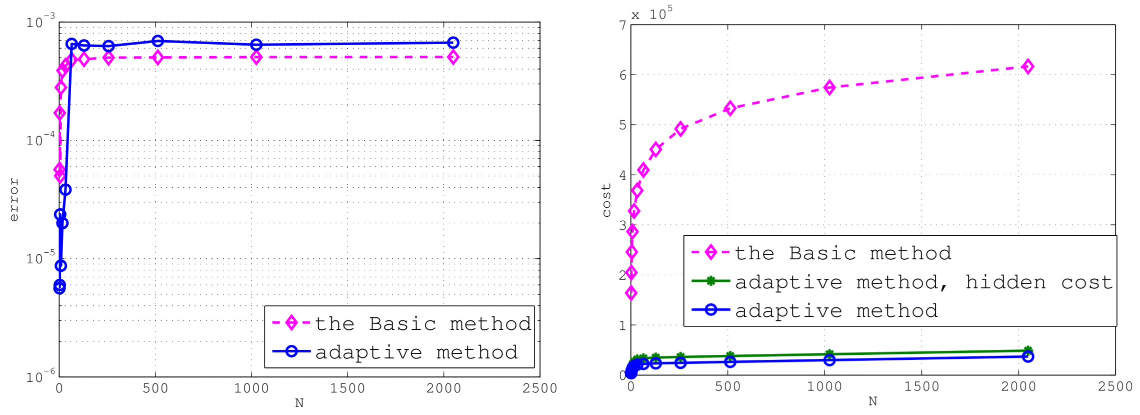

FIGURE 12. Example 2: Cost and accuracy of the adaptive Forward Euler method (A–FE–S_2) for $T = 1$. Plots of the error as a function of N (top left) and the cost as a function of N (top right) for A–FE–S_2 and the corresponding basic method (FE–S_2) with reference time step $\Delta t_{\mathrm{ref}} = 2.5 \cdot 10^{-5}$.

In this case the wavelet vectors near the corners decay with a slower rate than in the smooth parts This implies that the time derivatives of those wavelets also decay slower, which invalidates the theoretical foundation for the basic method. The adaptive method, however, will recognize the slower decay and adjust time step sizes accordingly; for wavelet vectors close to the corners shorter time steps will be used. Since the basic method has the same time step for all wavelets within one level it will not be suitable for this problem.

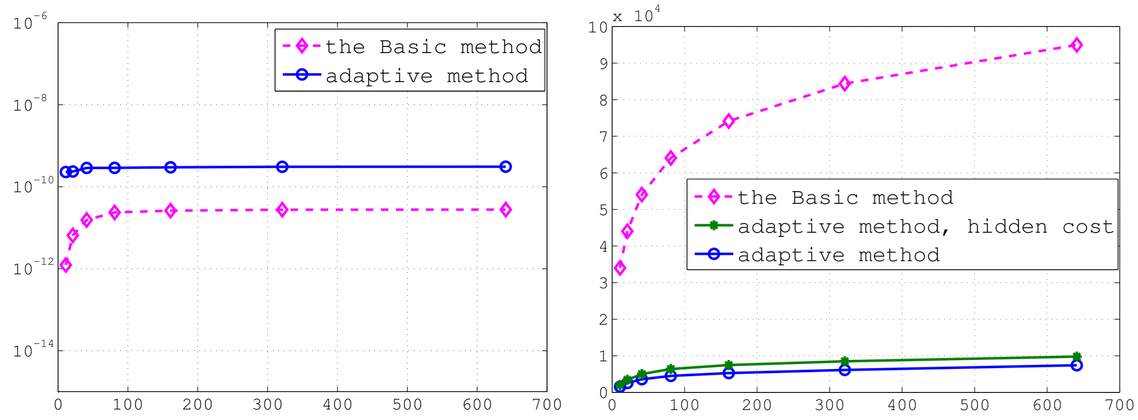

FIGURE 13. Example 2: Example 1: Cost and accuracy of the adaptive Runge–Kutta 4 method (A–RK4–S_6) for $T = 2$. Plots of the error as a function of N (left) and the cost as a function of N (right) for A–RK4–S_6 and the corresponding basic method (RK4–S_6) with reference time step $\Delta t_{\text{ref}} = 5 \cdot 10^{-4}$.

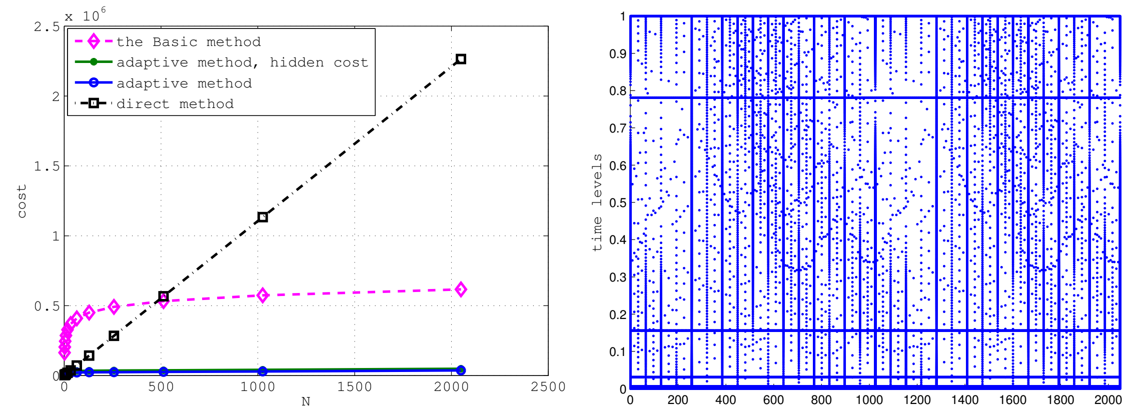

FIGURE 14. Example 2: Left: Cost of the direct, standard adaptive front tracking method based on Forward Euler compared to FE–S_2 and A–FE–S_2. Right: Time vectors $\boldsymbol{t}_{j,k}$ of the edge points and wavelet vectors for A–FE–S_2 with $N = 2049$. The time instances $\{t_{j,k}^n\}$ plotted as dots, cf. Figure 2 and Figure 4.

As in previous examples we solve the problem to $T = 1$ using the adaptive Forward Euler method A–FE–S_2 and the adaptive Runge-Kutta method A–RK4–S_6. In Figure 16 we plot the largest of the norms of the wavelet vectors at each level, as a function of j, for different times. Since the largest vectors are the ones at the corners, the decay rate is not the expected $\sim 2^{-2j}$ and 2^{-6j} respectively, but $2^{-j/2}$ for both methods.

The error and cost curves for the methods are shown in Figures 17 and 18, respectively. They are compared with the curves for the basic method. As in previous examples, the error is bounded independently of N for the adaptive methods. However, for the basic methods

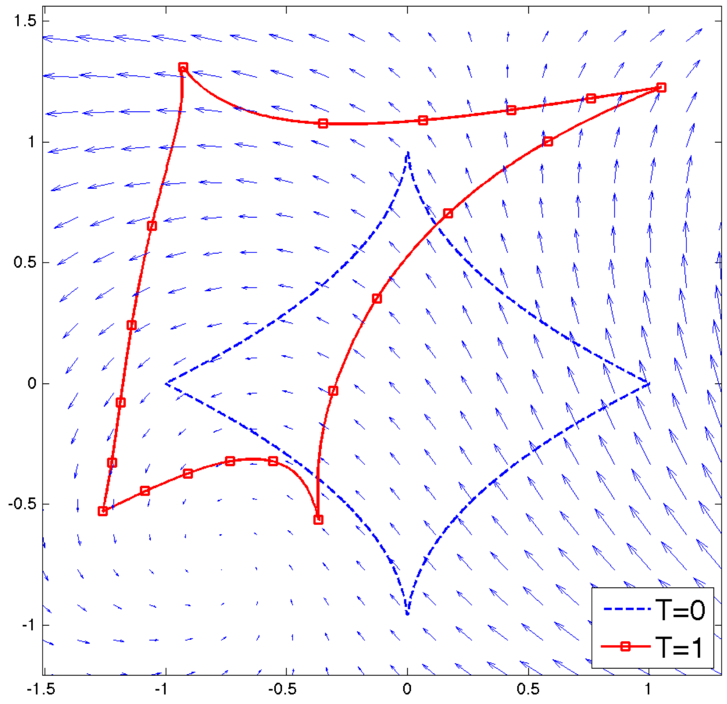

FIGURE 15. Example 3: The solution at times $T = 0, 1$ for the velocity field (29) (overlaid) with a non-smooth initial curve.

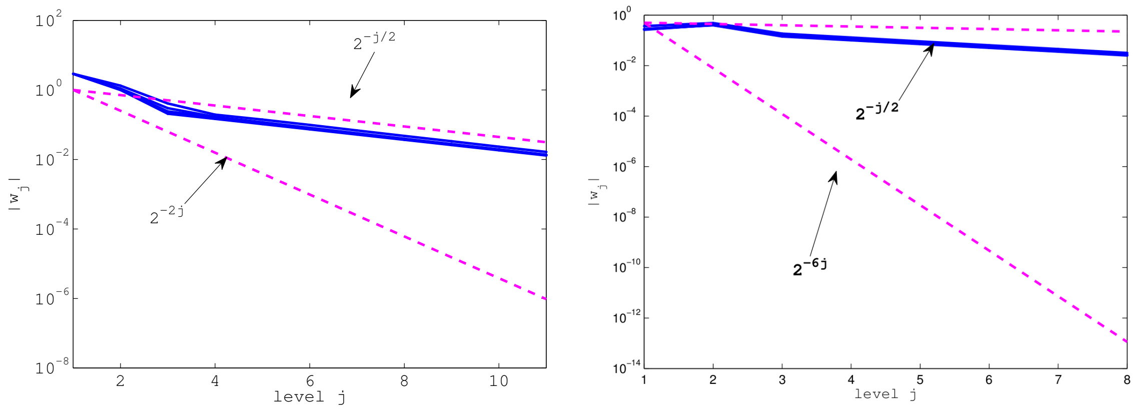

FIGURE 16. Example 3: Decay of wavelet coefficients The max size $\max_k |w_{j,k}(t)|$ is plotted at times $t = 0, 0.25, \ldots, 1$ for the adaptive Forward Euler scheme A–FE–S_2 (left) and the adaptive Runge–Kutta 4 scheme A–RK4–S_6 (right).

the error grows with N. Hence, adaptivity is *necessary* to maintain the fast character of the methods for this non-smooth problem.

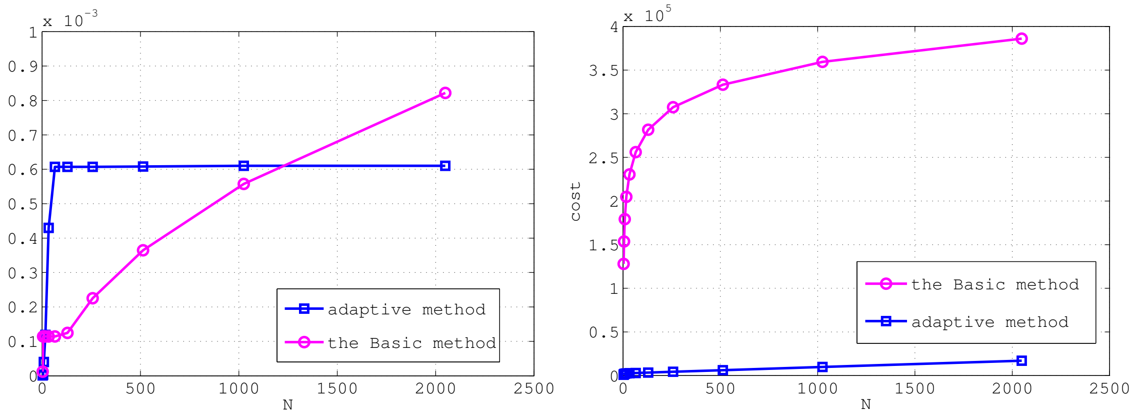

FIGURE 17. Example 3: Cost and accuracy of the adaptive Forward Euler method (A–FE–S_2) for $T = 1$. Plots of the error as a function of N (left) and the cost as a function of N (right) for A–FE–S_2 and the corresponding basic method (FE–S_2) with reference time step $\Delta t_{\text{ref}} = 2 \cdot 10^{-5}$.

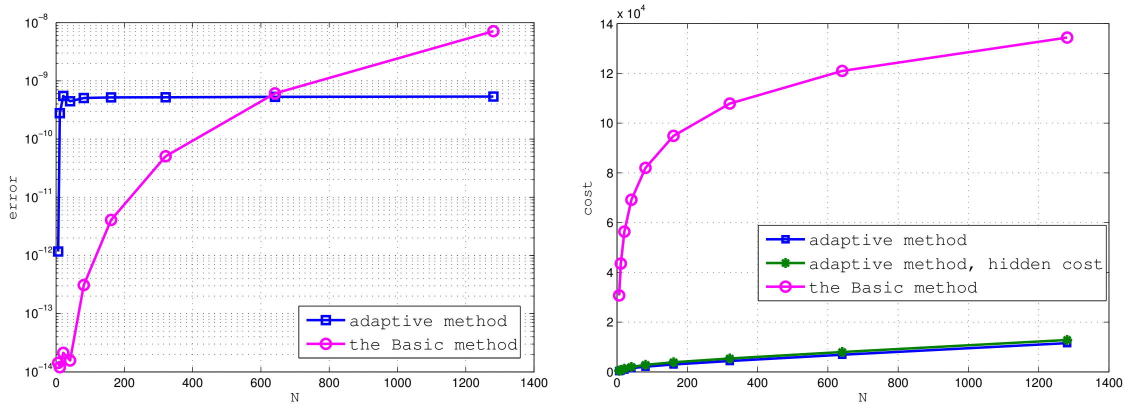

FIGURE 18. Example 3: Cost and accuracy of the adaptive Runge–Kutta 4 method (A–RK4–S_6) for $T = 1$. Plots of the error as a function of N (left) and the cost as a function of N (right) for A–RK4–S_6 and the corresponding basic method (RK4–S_6) with reference time step $\Delta t_{\text{ref}} = 2 \cdot 10^{-4}$.

6.4. **Example 4.** We now consider an example where the interface is a non-smooth surface moving in three dimensional space. The velocity field is given by

$$F(t, \mathbf{x}) = \begin{bmatrix} y \sin(x) - 0.5 \\ (x + 0.2)\cos(y) + 0.4 \\ \cos(z + xy) \end{bmatrix}. \tag{31}$$

We let the initial surface be the four-sided pyramid, $|x| + |y| + |z| = 1$, $z \geq 0$. The initial surface and the solution at $T = 1$ are shown in Figure 19.

We solve the problem to $T = 1$ with the surface version of the fast method based on Forward Euler, using both the adaptive and basic version. The errors and costs are compared in Figure 20. As in the case of curves, the adaptive version outperforms the basic version.

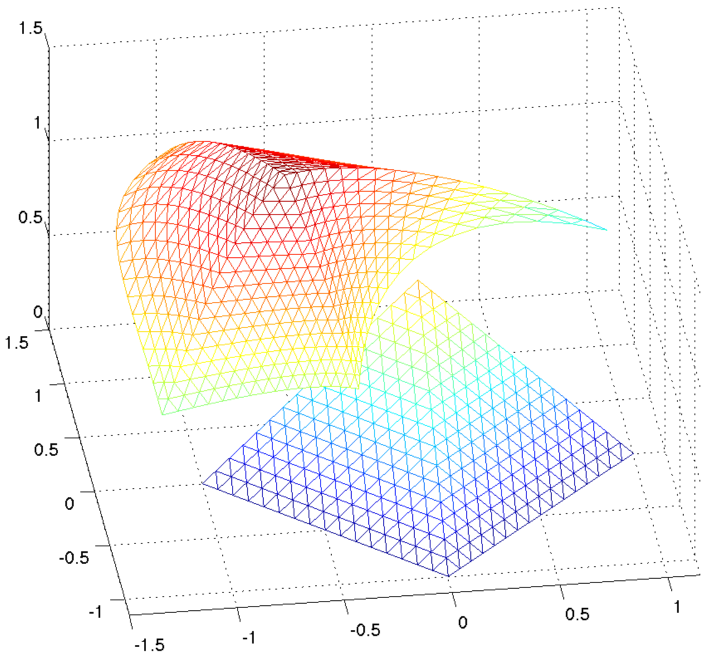

FIGURE 19. Example 4: The solution at times $T = 0$ (pyramid) and $T = 1$ for the velocity field (31).

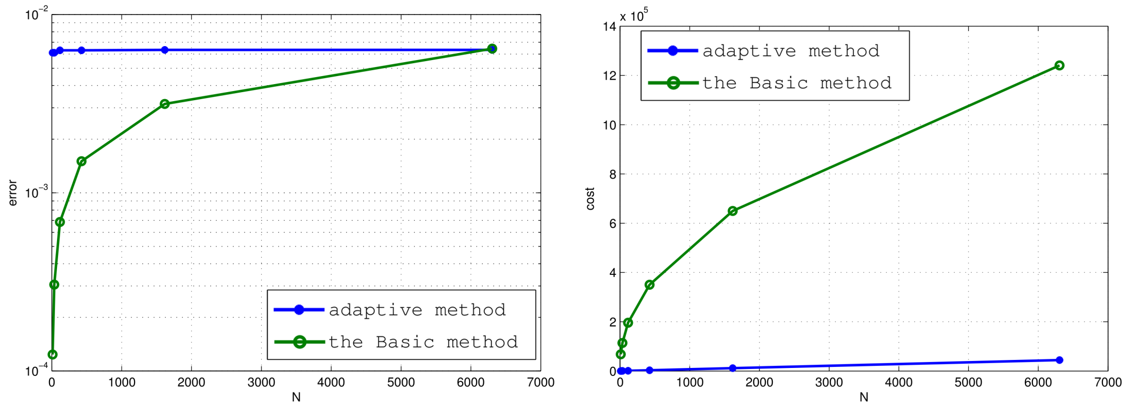

FIGURE 20. Example 4: Cost and accuracy of the adaptive Forward Euler method (A–FE–S_2) for $T = 1$. Plots of the error as a function of N (left) and the cost as a function of N (right) for A–FE–S_2 and the corresponding basic method (FE–S_2) with reference time step $\Delta t_{\text{ref}} = 2 \cdot 10^{-4}$.

The computational cost to achieve the same error level is much smaller, and the relative efficiency of adaptivity increases with N.

7. Conclusions

We have presented a time adaptive front tracking method based on multiresolution representation of the interface. The interface is described hierarchically by wavelet vectors which are moved in time using an adaptive Forward Euler or Runge–Kutta 4 method. For uniformly smooth problem, the cost is shown to be $O(\log N/\text{TOL}^{1/p} + N \log N)$ where p is the order of the time adaptive method used and TOL is the desired tolerance. We have applied our methods to a number of test cases, including non-smooth problems and surfaces in three dimensions, and compared against a non-adaptive method, that uses a simple time step doubling strategy, [24, 25]. The adaptive method shows significantly better results in terms of cost and accuracy for these cases, particularly for the non-smooth problems.

In the examples above, the length, or area, of the interface do not change significantly during the evolution, something which often happens in practice, e.g. for problems with expanding interfaces. Then the initial number of wavelet vectors would not be enough to resolve the interface after some time. An adaptive method which adds and removes wavelet vectors as the spatial resolution of the interface changes would then be desirable. Initial work on such spatial adaptivity for fast interface tracking methods can be found in [8, 23].

References

[1] R. Becker and R. Rannacher. An optimal control approach to a posteriori error estimation in finite element methods. *Acta Numerica*, pages 1–102, 2001.

[2] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 1978.

[3] A. S. Cavaretta, W. Dahmen, and C. A. Micchelli. Stationary subdivision. *Memoirs Amer. Math. Soc.*, 93(493), 1991.

[4] E. M. Constantinescu and A. Sandu. Multirate timestepping methods for hyperbolic conservation laws. *J. Sci. Comput.*, 22:239–278, 2007.

[5] I. Daubechies and J. C. Lagarias. Two-scale difference equations I. Existence and global regularity of solutions. *SIAM J. Math. Anal.*, 22(5):1388–1410, 1991.

[6] I. Daubechies and J. C. Lagarias. Two-scale difference equations II. Local regularity, infinite products of matrices and fractals. *SIAM J. Math. Anal.*, 23(4):1031–1079, 1992.

[7] I. Daubechies, O. Runborg, and W. Sweldens. Normal multiresolution approximation of curves. *Constr. Approx.*, 20:399–463, 2004.

[8] Y. Di, J. Popovic, and O. Runborg. An adaptive fast interface tracking method. *J. Comp. Math.*, accepted 2015.

[9] J. R. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulae. *J. Comp. Appl. Math.*, 6:19–26, 1980.

[10] J. R. Dormand and P. J. Prince. Higher order embedded Runge-Kutta formulae. *J. Comp. Appl. Math.*, 7:67–75, 1981.

[11] N. Dyn, D. Levin, and J. Gregory. A 4-point interpolatory subdivision scheme for curve design. *Comput. Aided. Geom. Des.*, 4:257–268, 1987.

[12] N. Dyn, D. Levin, and J. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. on Graphics*, 9(2):160–169, 1990.

[13] C. Gear and D. Wells. Multirate linear multistep methods. *BIT*, 24:484–52, 1984.

[14] J. Glimm, J. W. Grove, X. L. Li, K.-M. Shyue, Y. Zeng, and Q. Zhang. Three-dimensional front tracking. *SIAM J. Sci. Comput.*, 19(3):703–727, 1998.

[15] J. Glimm, E. Isaacson, D. Marchesin, and O. McBryan. Front tracking for hyperbolic systems. *Adv. Appl. Math.*, 2:91–119, 1981.

[16] M. Günther and P. Rentorp. Mutirate ROW methods and latency of electric circiuts. *Appl. Numer. Math.*, 13:83–102, 1993.

[17] E. Hairer, S. P. Nörsett, and G. Wanner. *Solving ordinary differential equations I.* Springer-Verlag Berlin Heidelberg, 1987.

[18] G. Lambaré, P. S. Lucio, and A. Hanyga. Two-dimensional multivalued traveltime and amplitude maps by uniform sampling of the ray field. *Geophys. J. Int.*, 125:584–598, 1996.

[19] A. Logg. Multi-adaptive time integration. *Appl. Num. Math.*, 48:339–354, 2004.

[20] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.

[21] S. J. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, 1988.

[22] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A PDE based fast local level set method. *J. Comput. Phys.*, 155(2):410–438, 1999.

[23] J. Popovic. *Fast Adaptive Numerical Methods for High Frequency Waves and Interface Tracking.* PhD thesis, KTH Royal Institute of Technology, 2012.

[24] O. Runborg. Fast interface tracking via a multiresolution representation of curves and surfaces. *Commun. Math. Sci.*, 7(2):365–398, 2009.

[25] O. Runborg. Analysis of high order fast interface tracking methods. *Numer. Math.*, 128(2):339–375, 2014.

[26] J. Strain. Tree methods for moving interfaces. *J. Comput. Phys.*, 151(2):616–648, 1999.

[27] J. Strain. A fast modular semi-Lagrangian method for moving interfaces. *J. Comput. Phys.*, 161(2):512–536, 2000.

[28] A.-K. Tornberg and B. Engquist. The segment projection method for interface tracking. *Comm. Pure Appl. Math.*, 56(1):47–79, 2003.

[29] S. O. Unverdi and G. Tryggvason. A front tracking method for viscous, incompressible, multi- fluid flows. *J. Comput. Phys.*, 100:25–37, 1992.

[30] V. Vinje, E. Iversen, and H. Gjøystdal. Traveltime and amplitude estimation using wavefront construction. *Geophysics*, 58(8):1157–1166, 1993.

[31] D. Zorin, P. Schröder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 189–192, 1996.