JOURNAL OF COMPUTATIONAL PHYSICS **116,** 69–78 (1995)

A Fast Algorithm for Vortex Blob Interactions

CRISTINA I. DRAGHICESCU

Department of Mathematics, University of Houston, Houston, Texas 77204

AND

MIRCEA DRAGHICESCU

TCAMC, University of Houston, Houston, Texas 77204

Received January 26, 1994

We introduce a new fast summation algorithm which, unlike previous methods, is not restricted to potential functions. Its asymptotic complexity is $\mathcal{O}(nH^{d+1})$, where n is the number of blobs, H is the level of refinement, and d is the domain dimension. For a nonsingular kernel the complexity reduces to $\mathcal{O}(n)$. We present a complete error and complexity analysis of the algorithm and describe its implementation. As an example we use the vortex blob method to simulate vortex sheet motion in incompressible flow.

1. INTRODUCTION

Vortex methods have been shown to be a successful approach for the numerical simulation of incompressible, inviscid fluid flow. In the vortex blob method introduced by Chorin [8] the trajectory of a finite number of blobs is approximated by solving a system of ordinary differential equations

$$\begin{aligned} \frac{d\mathbf{x}_i(t)}{dt} &= \sum_{\substack{j=1 \\ j\neq i}}^{n} w_j K_\delta(\mathbf{x}_i(t) - \mathbf{x}_j(t)) \\ \mathbf{x}_i(0) &= \mathbf{x}_i^0. \end{aligned} \tag{1.1}$$

Here $\mathbf{x}_i(t) = (x_i^1(t), x_i^2(t)) \in \mathbf{R}^2$, $i = 1, ..., n$ represent the positions of the centers of the vortices for which the velocities are determined using the numerical integration of the Biot–Savart law, K_δ is a smooth approximation of the singular kernel K ($K_\delta \to K$ as $\delta \to 0$). For smooth flow, theoretical and practical results state that, under certain assumptions made on δ, the method is stable and convergent with arbitrary high order accuracy.

The straightforward evaluation of the sums in (1.1) for n values $\mathbf{x}_i$ require $\mathcal{O}(n^2)$ computational work per time step. The purpose of this paper is to introduce a numerical algorithm which reduces this time complexity to $\mathcal{O}(n)$.

Other methods for reducing the complexity of evaluating similar sums include the hierarchical solver by Appel [2A], Barnes and Hut [5], the algorithms based on multipole expansions in Anderson [2], van Dommelen and Rundensteiner [10], Carrier, Greengard, and Rokhlin [7], the multigrid solver of Brandt and Lubrecht [6], the particle-in-cell method introduced by Harlow [12], and the method of local corrections by Anderson [1].

With the exception of [6], these methods are restricted to potential-type kernels. What we propose here is a more general approach, in which high accuracy is achieved by high order Taylor approximation. A preliminary version of the method for a singular kernel and a different application was introduced in [11]. The complexity of the algorithm is $\mathcal{O}(nH^{d+1})$, where H is the level of refinement and d is the domain dimension. For a non-singular kernel H is constant while in the case of a singular kernel $H = \mathcal{O}(\log n)$ under a very weak assumption regarding the distribution of the points $\mathbf{x}_i$.

To illustrate our technique and test the theoretical work and error estimates we used the vortex sheet model; in incompressible flow this is just a surface across which the tangential fluid velocity has a jump discontinuity. A vortex sheet can be used, for example, to model the vorticity shed from an aircraft's wake (the time coordinate t represents the aircraft's line of motion). The study of the vortex sheet motion then reduces to solving a set of differential equations (1.1).

We start from the numerical study of the vortex sheet evolution using desingularized equations as described by Krasny [17]. The method presented in this paper allows computations with many more points and over much longer time intervals than were possible using direct summation. Computations were performed for times beyond previously achieved; they bring numerical evidence to confirm the convergence of the solutions of the desingularized equation to the actual vortex sheet as the smoothing parameter δ decreases.

The intuitive idea of the fast algorithm is as follows. Suppose that we need to compute the sums

$$S(\mathbf{x}_i) = \sum_{\substack{j=1 \\ j\neq i}}^{n} w_j f(\mathbf{x}_i, \mathbf{x}_j), \quad i = 1, ..., n, \tag{1.2}$$

0021-9991/95 $6.00

where $\mathbf{x}_i$ are n given points in a domain $\mathcal{S} \subset \mathbf{R}^d$, $d = 1, 2, 3$, $f: \mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}$ is a given function, and $w_j \in \mathbf{R}$ are some coefficients. Usually $f(\mathbf{x}, \mathbf{y})$ is either singular or very large at $\mathbf{x} = \mathbf{y}$ and we will assume it is fast decaying away from the diagonal (e.g., $(\partial^\lambda/\partial \mathbf{y}^\lambda) f(\mathbf{x}, \mathbf{y}) = \mathcal{O}(|\mathbf{x} - \mathbf{y}|^{-\lambda})$). The computational domain $\mathcal{S}$ is covered by a hierarchy of meshes of height H. On each mesh cell τ such that $\mathbf{x} \notin \tau$, $f(\mathbf{x}, \mathbf{y})$ is approximated by a weighted sum of some functions $\phi_{k,\tau}(\mathbf{y})$, achieving a "separation" of variables:

$$f(\mathbf{x}, \mathbf{y}) \approx \sum_{k=0}^{\lambda(\mathbf{x},\tau)} F_{k,\tau}(\mathbf{x}) \phi_{k,\tau}(\mathbf{y}), \quad \mathbf{y} \in \tau.$$

A natural choice for the approximation is a Taylor expansion around the center $\mathbf{y}_\tau$ of τ:

$$f(\mathbf{x}, \mathbf{y}) \approx \sum_{k=0}^{\lambda(\mathbf{x},\tau)} F_{k,\tau}(\mathbf{x})(\mathbf{y} - \mathbf{y}_\tau)^k, \quad \mathbf{y} \in \tau.$$

The above separation of variables is used as follows: for each $\mathbf{x}_i$ partition $\mathcal{S}$ into a collection $\mathcal{F}(\mathbf{x}_i)$ of mesh cells (at different levels in the hierarchy) and write (1.2) as

$$S(\mathbf{x}_i) = \sum_{\tau \in \mathcal{F}(\mathbf{x}_i)} \sum_{\mathbf{x}_j \in \tau} w_j f(\mathbf{x}_i, \mathbf{x}_j). \tag{1.3}$$

The second sum is approximated by

$$\begin{aligned} \sum_{\mathbf{x}_j \in \tau} w_j f(\mathbf{x}_i, \mathbf{x}_j) &\approx \sum_{\mathbf{x}_j \in \tau} w_j \sum_{k=0}^{\lambda(\mathbf{x},\tau)} F_{k,\tau}(\mathbf{x}_i) \phi_{k,\tau}(\mathbf{x}_j) \\ &= \sum_{k=0}^{\lambda(\mathbf{x},\tau)} F_{k,\tau}(\mathbf{x}_i) \sum_{\mathbf{x}_j \in \tau} w_j \phi_{k,\tau}(\mathbf{x}_j) \\ &= \sum_{k=0}^{\lambda(\mathbf{x},\tau)} F_{k,\tau}(\mathbf{x}_i) c_{k,\tau}, \end{aligned}$$

for some functions $F_{k,\tau}$ and coefficients $c_{k,\tau}$ which do not depend on $\mathbf{x}_i$. The reduction in complexity for the evaluation of sums (1.3) for $i = 1, \ldots, n$ is achieved by *precomputing* $c_{k,\tau}$ for all k and all possible cells τ.

More precisely, for a problem in d-space, the number of indices k is $\mathcal{O}(\Lambda^d)$, where $\Lambda = \max_{\mathbf{x},\tau \in \mathcal{F}(\mathbf{x})} \lambda(\mathbf{x}, \tau)$. The number of cells containing a given point $\mathbf{x}$ is H; therefore the precomputation can be done in $\mathcal{O}(nH\Lambda^d)$ time by adding the contribution of each point to the sum of all cells containing it. The evaluation of all the sums can then be done in $\mathcal{O}(nF\Lambda^d)$, where $F = \max_{\mathbf{x}} \text{card}(\mathcal{F}(\mathbf{x}))$. We will show that the cells in $\mathcal{F}(\mathbf{x})$ can be chosen such that both Λ and F are $\mathcal{O}(H)$, thus the total complexity is $\mathcal{O}(nH^{d+1})$. If f is not singular then H is constant and the complexity is $\mathcal{O}(n)$. If f is singular on the diagonal then $H = \mathcal{O}(\log n)$ assuming a reasonable distribution of points in $\mathcal{S}$ (in a sense to be made precise later); the complexity is thus $\mathcal{O}(n \log^{d+1} n)$. In the rest of the paper we will assume that $d = 2$.

The paper is organized as follows: We start with a brief overview of the vortex-sheet equations and their desingularized version in Section 2. The "direct" $\mathcal{O}(n^2)$ discretization method using the vortex blob technique is then briefly described. Based on this numerical approach, the implementation and analysis of the fast summation technique is described in Section 3. Numerical experiments and comparison with previous results for the simulated fuselage-flaps configuration [17] are presented in Section 4. Main results and conclusions are reiterated in Section 5.

2. THE VORTEX BLOB METHOD FOR VORTEX SHEET EVOLUTION

2.1. *The Vortex Sheet Equations*

We will define a vortex sheet by a curve $\mathbf{x}(\Gamma, t) = (x_1(\Gamma, t), x_2(\Gamma, t))$ in $\mathbf{R}^2$, where Γ is the circulation parameter and t is time. The evolution equation is given by

$$\frac{\partial \mathbf{x}(\Gamma, t)}{\partial t} = \int K(\mathbf{x}(\Gamma, t) - \mathbf{x}(\tilde{\Gamma}, t)) d\tilde{\Gamma}$$

$$K(\mathbf{x}) = \frac{(x_2, -x_1)}{2\pi |\mathbf{x}|^2}.$$

Here the Cauchy principal value of the integral is taken. The initial shape $\mathbf{x}(\Gamma, 0)$ and the limits of integration depend upon the problem considered. This equation is a special case of the Biot-Savart law in which the velocity is expressed as an integral over vorticity in incompressible flows.

One approach in the computational study of the vortex sheet evolution is to replace the singular kernel K by the smooth approximation K_δ, such that $K_\delta \to K$ as $\delta \to 0$. For example, one can use (cf. [17]),

$$K_\delta(\mathbf{x}) = K(\mathbf{x}) \frac{|\mathbf{x}|^2}{|\mathbf{x}|^2 + \delta^2}$$

to obtain a desingularized integral and solve the equation

$$\frac{\partial \mathbf{x}(\Gamma, t)}{\partial t} = \mathbf{u}_\delta(\mathbf{x}, t) = \int K_\delta(\mathbf{x}(\Gamma, t) - \mathbf{x}(\tilde{\Gamma}, t)) d\tilde{\Gamma}.$$

Numerical evidence for the convergence of the solution of this equation to the vortex sheet as $\delta \to 0$ can be found for example in [17, 18].

2.2. *Approximation Using the Vortex Blob Method*

Following [17] we will use the change of variable $\Gamma = \Gamma(\alpha)$, with $0 \leq \alpha \leq \pi$. The condition $\Gamma(0) = \Gamma(\pi) = 0$ is imposed such that the values $\alpha = 0, \pi$ correspond to the wing tips. With

this change of variable the evolution equation for the sheet then becomes

$$\frac{\partial \mathbf{x}(\Gamma(\alpha), t)}{\partial t} = \int_0^\pi K_\delta(\mathbf{x}(\Gamma(\alpha), t) - \mathbf{x}(\Gamma(\tilde{\alpha}), t)\Gamma'(\tilde{\alpha})d\tilde{\alpha}. \quad (2.1)$$

The initial condition is given by

$$\mathbf{x}(\Gamma(\alpha), 0) = (-\cos(\alpha), 0)$$

which coincides with the straight line segment $-1 \le x \le 1$, $x = -\cos\alpha$. For $0.7 \le |x| \le 1$ the circulation distribution is described by

$$\Gamma(\alpha) = \sin(\alpha),$$

while in each interval $0 \le |x| \le 0.3$ and $0.3 \le |x| \le 0.7$, Γ is defined by a cubic polynomial in $|x|$. The coefficients are chosen such that Γ and its derivative with respect to arc length $\partial\Gamma/\partial s$ (the strength of the vortex sheet) are continuous at $x = 0, \pm 0.3, \pm 0.7$. The value $\Gamma = 2$ at $|x| = 0.3$ is a maximum and $\Gamma = 1.4$ at $x = 0$ a local minimum.

The approach we are interested in is the vortex blob approximation technique (cf. [8, 17]). There the curve $\mathbf{x}(\Gamma, t)$ is discretized into a finite number of blobs $\mathbf{x}_i(t)$, $i = 1, \ldots, 2N + 1 = n$ and a quadrature rule is used for the approximation of the integral (2.1). The values $\mathbf{x}_i(t)$ are approximations to the vortex sheet's exact position $\mathbf{x}(\Gamma(\alpha_i), t)$ at values $\alpha_i = \pi(i - 1)/2N$. The quadrature we use is the trapezoidal rule with weights $w_i = \Gamma'(\alpha_i)\pi/2N$, $i = 2, \ldots, 2N$, and $w_i = \Gamma'(\alpha_i)\pi/4N$, $i = 1, 2N + 1$. This yields a system of ordinary differential equations for the motion of the blobs

$$\begin{aligned} \frac{\partial \mathbf{x}_i(t)}{\partial t} &= \mathbf{u}_\delta(\mathbf{x}_i(t), t) \\ \mathbf{x}_i(0) &= (-\cos(\alpha_i), 0), \end{aligned} \quad (2.2)$$

where

$$\mathbf{u}_\delta(\mathbf{x}_i(t), t) = \sum_{j \ne i} w_j K_\delta(\mathbf{x}_i(t) - \mathbf{x}_j(t)) \quad (2.3)$$

denotes the approximation of the induced velocity at $\mathbf{x}_i(t)$. The above system can be integrated in time using the desired time-stepping method.

3. THE FAST ALGORITHM

Let $\tilde{\mathbf{u}}_\delta$ be the approximation (to be defined next) to the velocity $\mathbf{u}_\delta$ in (2.3) and let $\tilde{\mathbf{x}}_i(t)$ be the respective approximations to the paths $\mathbf{x}_i(t)$ defined by

$$\begin{aligned} \frac{\partial \tilde{\mathbf{x}}_i(t)}{\partial t} &= \tilde{\mathbf{u}}_\delta(\tilde{\mathbf{x}}_i(t), t), \\ \tilde{\mathbf{x}}_i(0) &= \mathbf{x}_i(0). \end{aligned}$$

From now on we will assume t fixed and denote $\tilde{\mathbf{x}}_i(t)$ by $\mathbf{y}_i$, $i = 1, \ldots, n$.

3.1. *Mesh Structure*

Assume without loss of generality that $\mathcal{S}$ is an $a \times b$ rectangle of area 1 with $a \le b < 2a$. We cover $\mathcal{S}$ with a hierarchy of meshes with rectangular cells defined as follows.

DEFINITION 3.1 (Cells). Let $M > 1$ be a constant and

$$C = \min\left(\frac{4}{a^2 + b^2}, \frac{8}{4a^2 + b^2}\right), \quad (3.1)$$

$$H = H(\delta) = \left\lceil 2\log_2 \frac{M}{\delta} - \log_2 C \right\rceil. \quad (3.2)$$

The choice of these constants are motivated by relation (3.3), Definition 3.2 and Lemma 3.1 below. The set σ of all cells is

$$\sigma = \sigma_0 \cup \sigma_1 \cup \ldots \cup \sigma_H,$$

where σ_k is the set of cells of level k defined recursively as follows:

$k = 0$: $\sigma_0 = \{\mathcal{S}\}$.

$k \ge 1$: Two cells in σ_k are obtained by splitting a cell in σ_{k-1} into two equal rectangles (the longer side is split in half) and assigning their common boundary to one of them.

In σ_k there are 2^k cells of area 2^{-k}; they are pairwise disjoint and completely cover $\mathcal{S}$. The total number of cells is $2^{H+1} - 1$.

It is useful to think of the cells as forming a complete binary tree of height H with $\mathcal{S}$ at the root. We will often use the standard terminology from tree theory and refer to *children* of a cell τ (the two cells into which τ is split), the *parent* of a cell, etc.

We will denote by τ an arbitrary cell with center $\mathbf{y}_\tau$, area $A(\tau)$ and radius

$$\rho(\tau) = \sup\{|\mathbf{y} - \mathbf{y}_\tau| : \mathbf{y} \in \tau\}.$$

With C defined in (3.1) we have

$$A(\tau) \ge C\rho(\tau)^2. \quad (3.3)$$

For arbitrary point $\mathbf{x}$ and cell τ defined

$$d(\mathbf{x}, \tau) = \sqrt{|\mathbf{x} - \mathbf{y}_\tau|^2 + \delta^2}.$$

For each point $\mathbf{x}$ we will partition $\mathcal{S}$ into a collection $\mathcal{F}(\mathbf{x})$ of cells (at different levels) as follows.

DEFINITION 3.2 (Domain partition). 1. Set $\mathcal{F}(\mathbf{x}) = \emptyset$ and $\tau = \mathcal{S}$.

2. If τ is empty or τ contains only $\mathbf{x}$ then exit,

3. else if $d(\mathbf{x}, \tau) \geq M\,\rho(\tau)$ then add τ to $\mathcal{F}(\mathbf{x})$,

4. else split τ into its two children and apply steps 2–4, recursively, to the children.

Note that a cell $\tau \in \mathcal{F}(\mathbf{x})$ is not strictly included in (is not the strict descendant of) any other cell in $\mathcal{F}(\mathbf{x})$.

LEMMA 3.1. *The partition algorithm is correct, i.e., it will never attempt to split (in Step 4) a cell in σ_H (a leaf of the cells' tree).*

Proof. Let $\tau \in \sigma_h$ be a cell which is split in Step 4. τ failed the test in Step 3; therefore $\rho(\tau) > d(\mathbf{x}, \tau)/M > \delta/M$. Using (3.3), $A(\tau) = 2^{-h} \geq C\rho(\tau)^2 > C\delta^2/M^2$, therefore $h < 2\log_2(M/\delta) - \log_2 C \leq H$, i.e., $\tau \notin \sigma_H$.

3.2. *Taylor Approximation*

Let ε be the desired precision of the fast method relative to the direct summation. We will fix point $\mathbf{x}$ and cell $\tau \in \mathcal{F}(\mathbf{x})$ and let $d = d(\mathbf{x}, \tau)$ and $\rho = \rho(\tau)$. Recall the definition of K_δ:

$$K_\delta(\mathbf{x} - \mathbf{y}) = \frac{1}{2\pi}\frac{(x_2 - y_2, -x_1 + y_1)}{|\mathbf{x} - \mathbf{y}|^2 + \delta^2}.$$

We will approximate $K_\delta(\mathbf{x} - \mathbf{y})$, $\mathbf{y} \in \tau$, by its Taylor polynomial of degree $\lambda - 1$, with λ defined later in this section. The 2D Taylor expansion of degree $\lambda - 1$ of $f(\mathbf{x}, \mathbf{y}) = 1/(|\mathbf{x} - \mathbf{y}|^2 + \delta^2)$ with respect to $\mathbf{y}$ about $\mathbf{y}_\tau$ is

$$f(\mathbf{x}, \mathbf{y}) = P_{\lambda-1}(\mathbf{x}, \mathbf{y}, \mathbf{y}_\tau) + R_{\lambda-1}(\mathbf{x}, \mathbf{y}, \mathbf{y}_\tau),$$

where

$$P_{\lambda-1}(\mathbf{x}, \mathbf{y}, \mathbf{y}_\tau) = \sum_{|k| \leq \lambda-1} \frac{1}{k!} D_{\mathbf{y}}^k f(\mathbf{x}, \mathbf{y})_{\mathbf{y}=\mathbf{y}_\tau} (\mathbf{y} - \mathbf{y}_\tau)^k$$
$$= \sum_{|k| \leq \lambda-1} a_k(\mathbf{x}, \mathbf{y}_\tau)(\mathbf{y} - \mathbf{y}_\tau)^k.$$

Here $k = (k_1, k_2)$, $|k| = k_1 + k_2$, $k! = k_1!k_2!$, $D_{\mathbf{y}}^k = \partial^{|k|}/\partial y_1^{k_1}\partial y_2^{k_2}$, $\mathbf{y}^k = y_1^{k_1}y_2^{k_2}$, and the coefficients

$$a_k(\mathbf{x}, \mathbf{y}_\tau) = \frac{1}{k!} D_{\mathbf{y}}^k f(\mathbf{x}, \mathbf{y})_{\mathbf{y}=\mathbf{y}_\tau}$$

can be computed using the recurrence

$$\begin{aligned} a_{0,0} &= f \\ a_{k_1,k_2} &= f(2(u_1 - v_1)a_{k_1-1,k_2} \\ &\quad + 2(u_2 - v_2)a_{k_1,k_2-1} - a_{k_1-2,k_2} - a_{k_1,k_2-2}), \end{aligned} \tag{3.4}$$

where $f = f(\mathbf{u}, \mathbf{v})$ and $a_{i,j} = a_{i,j}(\mathbf{u}, \mathbf{v})$ for arbitrary $\mathbf{u}$, $\mathbf{v}$. In the last formula $(k_1, k_2) \neq (0, 0)$ and, by convention, $a_{i,j} = 0$ if $i < 0$ or $j < 0$.

The remainder $R_{\lambda-1}$ can be bounded as follows: Denoting $(\mathbf{x} - \mathbf{y}_\tau) \cdot (\mathbf{y} - \mathbf{y}_\tau)/d^2 = \alpha$ and $|\mathbf{y} - \mathbf{y}_\tau|/d = \beta$ we have $|\alpha| < \beta \leq \rho/d \leq M^{-1} < 1$. Using (3.4) we can easily find the following recurrence formula (R_{-1} is defined for convenience)

$$\begin{aligned} R_{-1} &= f(\mathbf{x}, \mathbf{y}) \\ R_0 &= f(\mathbf{x}, \mathbf{y}) - f(\mathbf{x}, \mathbf{y}_\tau) \\ R_k &= 2\alpha R_{k-1} - \beta^2 R_{k-2}, \quad k \geq 1, \end{aligned}$$

with the exact solution

$$R_{\lambda-1}(\mathbf{x}, \mathbf{y}, \mathbf{y}_\tau) = f(\mathbf{x}, \mathbf{y})\left((\alpha - \beta^2)\frac{z^\lambda - \bar{z}^\lambda}{z - \bar{z}} + \frac{z^\lambda + \bar{z}^\lambda}{2}\right),$$

where

$$z = \alpha + i\sqrt{\beta^2 - \alpha^2}.$$

But $f(\mathbf{x}, \mathbf{y})|\alpha - \beta^2| \leq \rho/(d^2|\mathbf{x} - \mathbf{y}|)$, $f(\mathbf{x}, \mathbf{y}) \leq ((d - \rho)|\mathbf{x} - \mathbf{y}|)^{-1}$, and $|z| \leq \rho/d$; therefore

$$|R_{\lambda-1}(\mathbf{x}, \mathbf{y}, \mathbf{y}_\tau)| \leq \frac{1}{|\mathbf{x} - \mathbf{y}|}\left(\frac{\rho}{d}\right)^\lambda\left(\frac{\lambda}{d} + \frac{1}{d - \rho}\right).$$

The right-hand side tends to 0 as $\lambda \to \infty$, so we can choose $\lambda = \lambda(\mathbf{x}, \tau)$ to be the smallest positive integer such that the right-hand side is less than, or equal to, $2\varepsilon/|\mathbf{x} - \mathbf{y}|$; therefore

$$|R_{\lambda-1}(\mathbf{x}, \mathbf{y}, \mathbf{y}_\tau)| \leq \frac{2\varepsilon}{|\mathbf{x} - \mathbf{y}|}. \tag{3.5}$$

Let

$$\begin{aligned} &K_\delta^{\lambda,\tau}(\mathbf{x} - \mathbf{y}) \\ &\quad = \frac{(x_2 - y_2, -x_2 + y_1)}{2\pi} P_{\lambda-1}(\mathbf{x}, \mathbf{y}, \mathbf{y}_\tau) \\ &\quad = K_\delta(\mathbf{x} - \mathbf{y}) - \frac{(x_2 - y_2, -x_1 + y_1)}{2\pi} R_{\lambda-1}(\mathbf{x}, \mathbf{y}, \mathbf{y}_\tau). \end{aligned} \tag{3.6}$$

We will now define the approximation to $\mathbf{u}_\delta$ as

$$\tilde{\mathbf{u}}_\delta(\mathbf{x}) = \sum_{\tau \in \mathcal{F}(\mathbf{x})} \tilde{\mathbf{u}}_\delta^\tau(\mathbf{x}), \tag{3.7}$$

where

$$\begin{aligned}\tilde{\mathbf{u}}_\delta^\tau(\mathbf{x}) &= \sum_{\mathbf{y}_j \in \tau} w_j K_\delta^{\lambda,\tau}(\mathbf{x} - \mathbf{y}_j) \\ &= \frac{1}{2\pi} \sum_{\mathbf{y}_j \in \tau} w_j \sum_{|k| \le \lambda - 1} a_k(\mathbf{x}, \mathbf{y}_\tau)(x_2 - y_{j2}, -x_1 + y_{j1})(\mathbf{y}_j - \mathbf{y}_\tau)^k \\ &= \frac{1}{2\pi} \sum_{|k| \le \lambda - 1} a_k(\mathbf{x}, \mathbf{y}_\tau)(x_2 A_\tau^k - B_\tau^k, -x_1 A_\tau^k + C_\tau^k),\end{aligned} \tag{3.8}$$

and

$$\begin{aligned}A_\tau^k &= \sum_{\mathbf{y}_j \in \tau} w_j (\mathbf{y}_j - \mathbf{y}_\tau)^k, \\ B_\tau^k &= \sum_{\mathbf{y}_j \in \tau} w_j (\mathbf{y}_j - \mathbf{y}_\tau)^k y_{j2}, \\ C_\tau^k &= \sum_{\mathbf{y}_j \in \tau} w_j (\mathbf{y}_j - \mathbf{y}_\tau)^k y_{j1}.\end{aligned} \tag{3.9}$$

The evaluation of $\tilde{\mathbf{u}}_\delta(\mathbf{x})$ reduces thus to the following two steps:

1. Preprocessing: evaluate the sums (3.9) for all cells τ and indices k.
2. For all cells $\tau \in \mathcal{F}(\mathbf{x})$ compute $\tilde{\mathbf{u}}_\delta^\tau(\mathbf{x})$ according to (3.8) and then add them together.

When computing $\tilde{\mathbf{u}}_\delta$ for all particles $\mathbf{x}$ only Step 2 needs to be repeated n times.

3.3. *Formal Description of the Algorithm*

The central data structure used in the algorithm is a binary tree T which will store a subtree (containing the root $\mathcal{S}$) of the complete tree of cells. Initially T is the one node tree corresponding to $\mathcal{S}$. New leaves are added dynamically whenever new cells are considered during the domain partitioning for some point $\mathbf{x}$. Each node has pointers to its parent and children and space for storing the coefficients $A_\tau^k, B_\tau^k, C_\tau^k$. To simplify the presentation, we will assume that we know $\Lambda = \max_{\mathbf{x}, \tau \in \mathcal{F}(\mathbf{x})} \lambda(\mathbf{x}, \tau)$; therefore the amount of space needed for each node is statically known (we will show later how to relax this requirement and make the algorithm work when Λ is not known).

Input: n, w_j, $\mathbf{y}_j$, $j = 1, \ldots, n$, Λ, M.
Output: next step approximation $\tilde{\mathbf{u}}_\delta(\mathbf{x})$ for all $\mathbf{x} = \mathbf{y}_j$, $j = 1, \ldots, n$.
Initialization: Set T to the one node tree corresponding to $\mathcal{S}$.
Step 1 (preprocessing, done only once).
For $j = 1, \ldots, n$, for all cells τ containing $\mathbf{y}_j$ and for all $k = (k_1, k_2)$, $|k| \le \Lambda - 1$, add $\alpha_j(\mathbf{y}_j - \mathbf{y}_\tau)^k$ to A_τ^k, $\alpha_j(\mathbf{y}_j - \mathbf{y}_\tau)^k y_{j2}$ to B_τ^k, and $\alpha_j(\mathbf{y}_j - \mathbf{y}_\tau)^k y_{j1}$ to C_τ^k.
Step 2 (partition the domain; this and following step are repeated for $\mathbf{x} = \mathbf{y}_j$, $j = 1, \ldots, n$).
Partition $\mathcal{S}$ as described in Definition 3.2. Whenever a leaf of T is split add two new nodes (leaves) to T.
Step 3 (computation).
For all cells $\tau \in \mathcal{F}(\mathbf{x})$:
1. compute $a_k(\mathbf{x}, \mathbf{y}_\tau)$ according to (3.4) for all k, $|k| \le \lambda(\mathbf{x}, \tau) - 1$;
2. compute $\tilde{\mathbf{u}}_\delta^\tau(\mathbf{x}, t)$ according to (3.8).

Compute $\sum_{\tau \in \mathcal{F}(\mathbf{x})} \tilde{\mathbf{u}}_\delta^\tau(\mathbf{x}, t)$.
End of algorithm.

The correctness and complexity of the algorithm are given by the following theorem; the proof is presented in the next two sections.

THEOREM 3.1. *For all t:*
1. $|\mathbf{u}_\delta(\mathbf{x}, t) - \tilde{\mathbf{u}}_\delta(\mathbf{x}, t)| \le \varepsilon$,
2. *Evaluation of $\tilde{\mathbf{u}}_\delta(\mathbf{x}_j, t)$ for $j = 1, \ldots, n$ can be done in $\mathcal{O}(n)$ operations.*

3.4. *Error Estimates*

We will now prove the first part of Theorem 3.1; i.e., we will show that the fast method is consistent with the vortex blob method. Using (2.3), (3.7), (3.8), (3.6), and the definition of w_j we obtain

$$\begin{aligned}&|\mathbf{u}_\delta(\mathbf{x}, t) - \tilde{\mathbf{u}}_\delta(\mathbf{x}, t)| \\ &\quad \le \sum_{\tau \in \mathcal{F}(\mathbf{x})} \sum_{\mathbf{y}_j \in \tau} |w_j| \, |K_\delta(\mathbf{x} - \mathbf{y}_j) - K_\delta^{\lambda,\tau}(\mathbf{x} - \mathbf{y}_j)| \\ &\quad = \frac{1}{2\pi} \sum_{\tau \in \mathcal{F}(\mathbf{x})} \sum_{\mathbf{y}_j \in \tau} |w_j| \, |\mathbf{x} - \mathbf{y}_j| \, |R_{\lambda-1}(\mathbf{x}, \mathbf{y}_j, \mathbf{y}_\tau)| \\ &\quad = \frac{1}{2n} \sum_{\tau \in \mathcal{F}(\mathbf{x})} \sum_{\mathbf{y}_j \in \tau} |\Gamma'(\alpha_j)| \, |\mathbf{x} - \mathbf{y}_j| \, |R_{\lambda-1}(\mathbf{x}, \mathbf{y}_j, \mathbf{y}_\tau)|.\end{aligned}$$

By the definition of Γ, $|\Gamma'(\alpha_j)| \le 1$. Using this fact and (3.5), we can write

$$\begin{aligned}|\mathbf{u}_\delta(\mathbf{x}, t) - \tilde{\mathbf{u}}_\delta(\mathbf{x}, t)| &\le \frac{1}{2n} \sum_{\tau \in \mathcal{F}(\mathbf{x})} \sum_{\mathbf{y}_j \in \tau} |\mathbf{x} - \mathbf{y}_j| \, |R_{\lambda-1}(\mathbf{x}, \mathbf{y}_j, \mathbf{y}_\tau)| \\ &\le \frac{\varepsilon}{n} \sum_{\tau \in \mathcal{F}(\mathbf{x})} \sum_{\mathbf{y}_j \in \tau} 1 \le \varepsilon.\end{aligned}$$

This concludes the proof of the first part of Theorem 3.1.

3.5. *Time Complexity*

To prove the second part of Theorem 3.1, we need the following lemmas.

LEMMA 3.2. $\max_{\mathbf{x}} \text{card}(\mathcal{F}(\mathbf{x})) = \mathcal{O}(H)$.

Proof. Fix $\mathbf{x}$ and let $\tau \in \mathcal{F}(\mathbf{x})$ of center $\mathbf{y}_\tau$ and radius $\rho = \rho(\tau)$. Let τ' be its parent of center $\mathbf{y}_{\tau'}$. $\tau' \notin \mathcal{F}(\mathbf{x})$; therefore

$$2\rho > \rho(\tau') > M^{-1} d(\mathbf{x}, \tau') > M^{-1} |\mathbf{x} - \mathbf{y}_{\tau'}|$$
$$> M^{-1}(|\mathbf{x} - \mathbf{y}_\tau| - \rho) > M^{-1} |\mathbf{x} - \mathbf{y}_\tau| - \rho.$$

Thus

$$|\mathbf{x} - \mathbf{y}_\tau| < 3\rho M.$$

By (3.3), $A(\tau) \geq C\rho^2$ and the area of $\{\mathbf{y} : |\mathbf{x} - \mathbf{y}| < 3\rho M\}$ is smaller than $c\rho^2$ for some constant c, so there are at most c/C cells of given radius ρ in $\mathcal{F}(\mathbf{x})$. There are at most $H + 1$ different possible radii (one for each level in T), which proves the lemma. ∎

LEMMA 3.3. *For arbitrary, fixed* $\mathbf{x}$*, the complexity of the partition algorithm (Definition* 3.2*) is* $\mathcal{O}(H^2)$.

Proof. For each cell included in $\mathcal{F}(\mathbf{x})$ the partition algorithm will visit all its ancestors. The only other cells visited are either empty or contain only $\mathbf{x}$. Their parents must be ancestors of some cell in $\mathcal{F}(\mathbf{x})$; otherwise they would have not been visited; we can thus bound their number by the number of ancestors of cells in $\mathcal{F}(\mathbf{x})$. The lemma then follows from the fact that an arbitrary cell has at most H ancestors and, according to Lemma 3.2, there are $\mathcal{O}(H)$ cells in $\mathcal{F}(\mathbf{x})$. ∎

LEMMA 3.4. $\Lambda = \max_{\mathbf{x}, \tau \in \mathcal{F}(\mathbf{x})} \lambda(\mathbf{x}, \tau) = \mathcal{O}(H - \log \varepsilon)$.

Proof. Fix $\mathbf{x}$, τ and let $\rho = \rho(\tau)$ and $\xi = \rho(\tau)/d(\mathbf{x}, \tau)$. From Definition 3.2 we know that

$$0 < \xi \leq M^{-1} < 1.$$

We can restate the definition of $\lambda = \lambda(\mathbf{x}, \tau)$ from Subsection 3.2 as follows: λ is the smallest positive integer such that

$$\xi^{\lambda+1} - 2\varepsilon\rho \frac{1 - \xi}{\lambda - \lambda\xi + 1} \leq 0. \tag{3.10}$$

The left-hand side has its maximum at $\xi = M^{-1}$; therefore $\lambda \leq \lambda'$, where λ' is the smallest positive integer such that $M^{-\lambda'-1} \leq 2\varepsilon\rho(M - 1)/(\lambda' M - \lambda' + M)$. It is easy to see that $\lambda' = \mathcal{O}(-\log \varepsilon\rho)$.

If $\tau \in \sigma_k$ then $A(\tau) = 2^{-k}$, therefore $-\log \rho = \mathcal{O}(k)$. But $k \leq H$, which proves the lemma. ∎

We can now compute the complexity of each step of the algorithm.

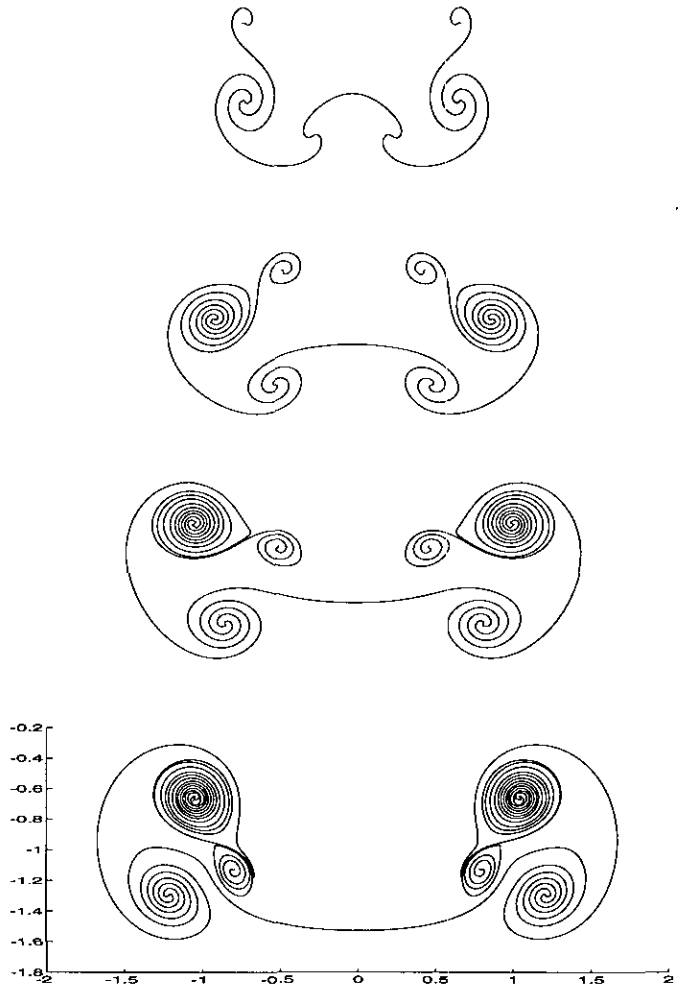

FIG. 1. The solution obtained using the fast summation is plotted at $T = 1, 2, 3$ and 4, using $\delta = 0.1$ and $\Delta t = 0.02$. The number of vortex blobs was $N = 40$ at $T = 0$ and reached $N = 931$ at $T = 4$. The precision was set to $\varepsilon = 0.001$. The running time at $T = 4$ was 173 s.

Step 1. We find all the cells in T containing $\mathbf{y}_j$ by moving down from the root to a leaf. The complexity of this step is thus $\mathcal{O}(nH\Lambda^2)$.

Step 2. Lemma 3.3 states that the complexity of this step is $\mathcal{O}(H^2)$.

Step 3. Using Lemma 3.2 the complexity of this step is $\mathcal{O}(H\Lambda^2)$.

The total complexity of the algorithm is the complexity of Step 1 plus n times the complexity of Steps 2 and 3, i.e., $\mathcal{O}(nH(H + \Lambda^2))$. Taking into consideration (3.2) and using Lemma 3.4, this becomes

$$\mathcal{O}(n(-\log \delta)(\log^2 \delta + \log^2 \varepsilon)), \tag{3.11}$$

which is $\mathcal{O}(n)$ when δ and ε are fixed.

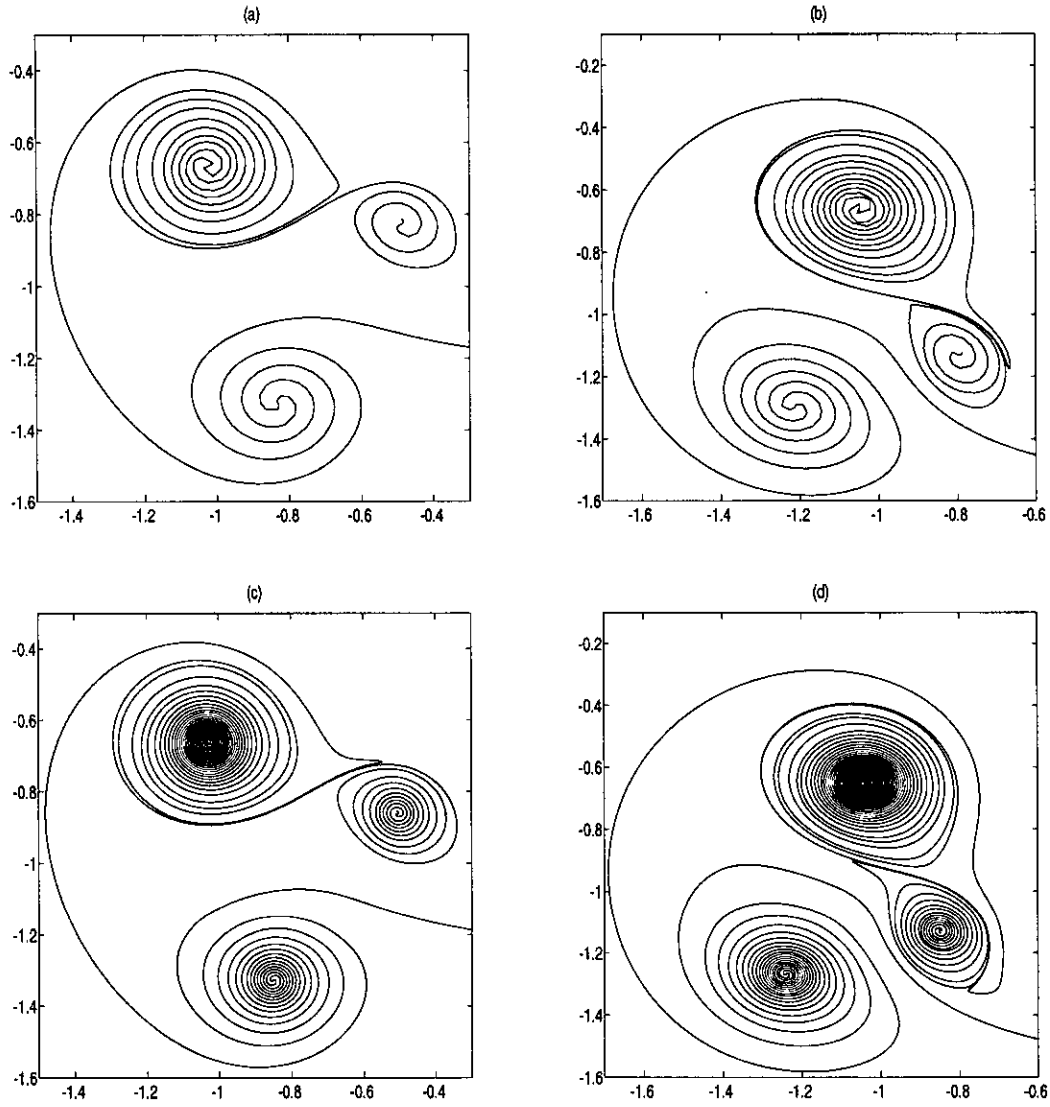

FIG. 2. A closeup view of the solution at $T = 3$ and $T = 4$ with $\delta = 0.1$ in (a), (b) and $\delta = 0.05$ in (c), (d).

The fast algorithm can also be used, with no changes, for singular kernels. Its complexity remains $\mathcal{O}(nH(H + \Lambda^2))$, but H will be defined differently. Setting $\delta = 0$ in the problem discussed here will make the computations unstable, but for reasons which have nothing to do with the fast algorithm. In [11] we discussed a well-posed problem with a singular kernel.

For a singular kernel the algorithm is faster than the direct summation assuming a reasonable distribution of the points $\mathbf{y}_i$, more precisely:

Assumption 1 (Point distribution). There exists a polynomial $P(x)$ such that any region $R \subseteq \mathcal{S}$ contains at most $P(n) \cdot \text{area}(R)$ points $\mathbf{y}_i$.

This a very weak condition on the distribution of points—we do not know any problem which does not satisfy this condition. (3.2) will be replaced by

$$H = H(n) = \lceil \log_2 P(n) \rceil + 1. \tag{3.12}$$

Now $H = \mathcal{O}(\log n)$ so the complexity of the algorithm becomes

$$\mathcal{O}(n \log n(\log^2 n + \log^2 \varepsilon)).$$

Taking $R = \mathcal{S}$ in the above assumption we see that $P(n) \geq n$. From a practical point of view we would like $P(n)$ to be close to n as this will lower H and, implicitly, the constant in the $\mathcal{O}$ complexity. Definition 3.2 remains correct.

LEMMA 3.5. *Under Assumption* 1 *and with H defined by* (3.12), *the partition algorithm will never attempt to split* (*in Step* 4) *a cell in* σ_H (*a leaf of the cells' tree*).

Proof. Let $\tau \in \sigma_h$ be a cell which is split in Step 4. Ac-

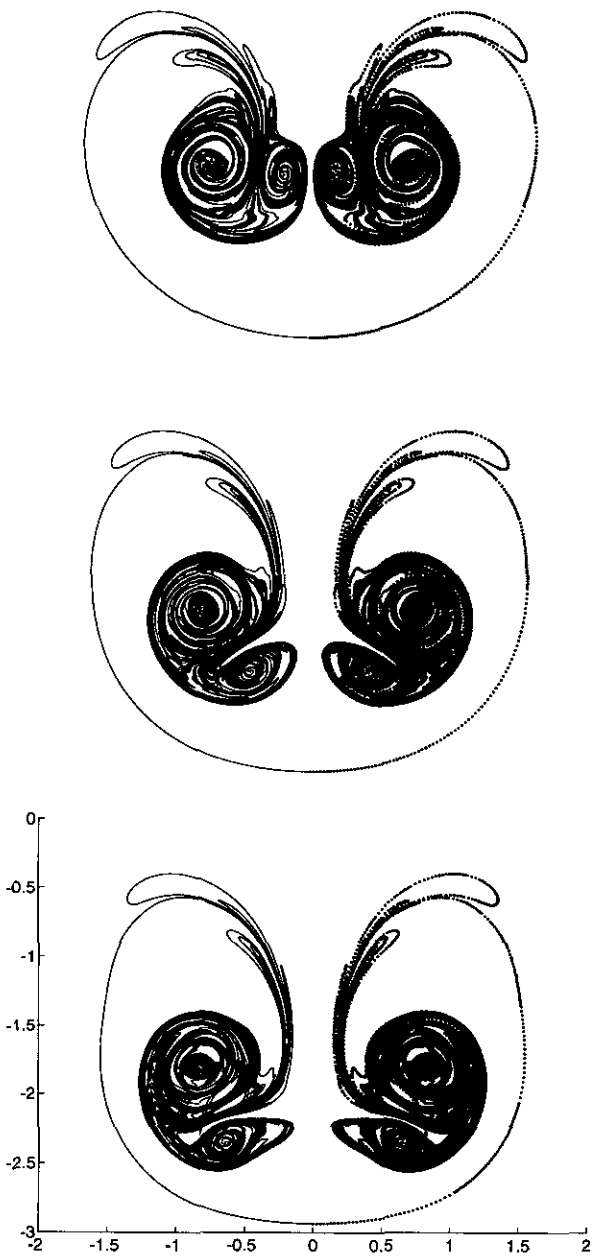

FIG. 3. The solution obtained using the fast summation is plotted at $T = 10$, 11, and 11.8, using $\delta = 0.1$ and $\Delta t = 0.02$. The number of vortex blobs was $N = 40$ at $T = 0$ and reached $N = 71346$ at $T = 11.8$. The precision was set to $\varepsilon = 0.001$. The total running time at $T = 11.8$ was 18 h 39 min.

cording to the point distribution assumption τ has at most $P(n)/2^h$ points. From Step 2, τ is also not empty, thus $P(n)/2^h \geq 1$; i.e., $h \leq \log_2 P(n) \leq H - 1$. ▌

The space complexity of the algorithm is $\mathbb{O}(n\Lambda^2)$ which represents the space needed for storing the nodes of T. For a nonsingular kernel this reduces to $\mathbb{O}(n)$ and for a singular kernel to $\mathbb{O}(n \log^2 n)$.

3.6. *Algorithm Details*

The value of H given by (3.2) or (3.12) is used only for proving the time complexity. The implementation does not need to know the (static) value of H; memory for the nodes of T is allocated dynamically as the tree grows.

The definition of Λ is not useful for computing its value. Fortunately, the algorithm can be easily modified to perform correctly for any value of Λ by splitting a cell which would require a $\lambda > \Lambda$. Its performance will, however, degrade if Λ is too small. Our current implementation uses $\Lambda = 15$ for efficient simulations with up to 100,000 points. Finally, note that Λ can be completely eliminated by implementing the second preprocessing optimization discussed below.

For given ε and Λ, the parameter M controls the balance between the number of cells in the partition and the degrees of the Taylor polynomials on these cells. A value close to 1 will decrease the number of cells (by allowing larger cells) but will increase the degrees λ, while a large value will have the opposite effect. In fact, M is not essential and was introduced mainly for the sake of clarity and conciseness of the presentation. Indeed, the correctness of the algorithm (Part 1 of Theorem 3.1) is assured by relation (3.10) for all points $\mathbf{x}$ and cells $\tau \in \mathcal{F}(\mathbf{x})$; to prove the time complexity we can either:

1. Bound ξ by a constant (M^{-1}); this is the approach taken in this paper.
2. Fix λ and include τ in $\mathcal{F}(\mathbf{x})$ iff ξ satisfies (3.10); this method was used in [11].
3. Let both ξ and λ vary, subject to (3.10) and try to minimize the computation time. This can be faster than both 1 and 2, but the complexity becomes harder to compute directly.

In our implementation we took the third approach. Specifically, for a given τ, the partition algorithm computes λ from (3.10) and determines the fastest way to compute the interactions of the points in τ by taking the minimum of:

1. the amount of work necessary for computing the Taylor approximation as a function of λ (a quadratic polynomial in λ whose coefficients can be determined experimentally for each machine and implementation),
2. The amount of work necessary for summing the interactions of points in τ directly as a function of the number of these points (a linear function), and
3. the amount of work necessary assuming we split τ and consider its children (computing this exactly would require a recursive lookup through the entire tree of cells; our implementation uses some heuristics to estimate this quantity without having to descend too deeply into the tree).

Note that direct summation in a cell τ requires knowledge of all the points in τ. This is achieved by linking all points in τ

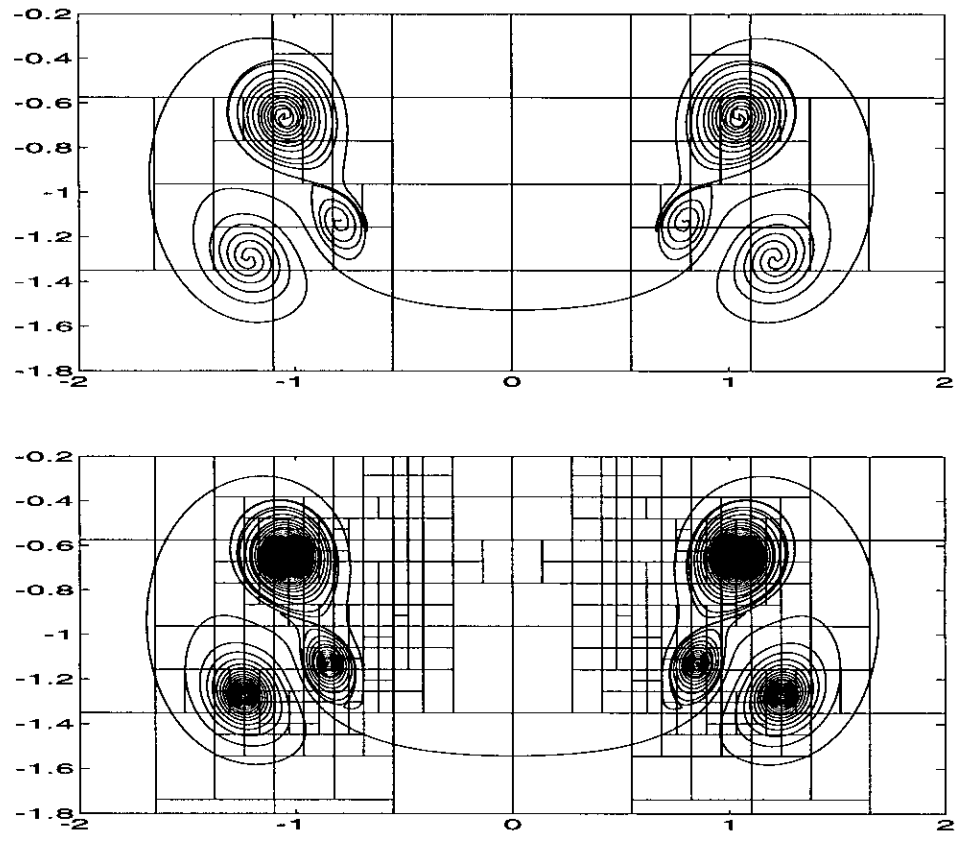

FIG. 4. The structure of the cells used for the fast summation in the computation of the solutions at $T = 4$ with $\delta = 0.1$ and $\delta = 0.05$.

into a list; these lists are constructed during the preprocessing phase for all τ.

There are two optimizations that can make the preprocessing phase more efficient:

1. The complexity of the preprocessing phase (Step 1) can be reduced from $\mathcal{O}(nH\Lambda^2)$ to $\mathcal{O}((n + 2^H)\Lambda^2)$ at the expense of using more memory. We can write, for example,

$$A_\tau^k = \sum_{\mathbf{y}_j \in \tau} w_j(\mathbf{y}_j - \mathbf{y}_\tau)^k = \sum_{\mathbf{y}_j \in \tau} w_j \sum_{p \le k} \binom{k}{p} \mathbf{y}_j^{k-p}(-\mathbf{y}_\tau)^p$$

$$= k! \sum_{p \le k} \frac{(-\mathbf{y}_\tau)^p}{p!} \sum_{\mathbf{y}_j \in \tau} \frac{w_j \mathbf{y}_j^{k-p}}{(k-p)!}.$$

The values $(-\mathbf{y}_\tau)^r/r!$, $w_j\mathbf{y}_j^r/r!$ for all r, $|r| < \Lambda$, cells τ, and points $\mathbf{y}_j$ can be computed in $\mathcal{O}((n + 2^H)\Lambda^2)$. We then evaluate the inside sums first for all leaves and then for all internal nodes by adding together the sums of the children in a bottom–up sweep through T (there are $\mathcal{O}(2^H)$ nodes and each one needs $\mathcal{O}(\Lambda^2)$ additions).

2. Alternately, instead of computing (in Step 1) the coefficients A_k^τ, B_k^τ, C_k^τ for all cells in T, we can wait until a cell τ is actually used in a computation for some $\mathbf{x}$ and then compute the coefficients using the list constructed in Step 1. The coefficients are then stored and need not be recomputed if τ is used again in a computation for a different point. This optimization also allows us to compute only as many coefficients as necessary ($\lambda(\mathbf{x}, \tau)$ instead of Λ) and, if storage for the coefficients is allocated dynamically, eliminates the need for choosing a static value for Λ.

4. NUMERICAL EXAMPLES

We present here some numerical tests for the problem of loading of a simulated fuselage-flaps configuration as described in Section 2.2. This problem has been numerically studied in detail by Krasny [17] using the vortex blob method. His calculations showed convergence of this approach with respect to both mesh refinement and smoothing parameter.

Using the initial data, smoothing parameters and time steps from [17], the fast method allowed us to duplicate previous results efficiently, using less CPU time. We were also able to extend the computations to longer simulation times and considerable larger number of points. Our program was written in C using double precision arithmetic and run on a Sun Sparc 2 workstation.

We used the four-step Adams–Bashforth method with time step Δt to integrate the two components of the system (2.2). To overcome the loss of resolution caused by the stretching of the vortex sheet curve, new points (vortex blobs) are inserted during computation whenever the distance between adjacent points becomes too large. We used the cubic point insertion method described in [17].

Figure 1 shows the vortex sheet evolution at times $T = 1$, 2, 3, 4. The smoothing parameter is $\delta = 0.1$ and the time step is $\Delta t = 0.0125$. We started with 40 blobs at $T = 0$ and, using point insertion, reached 931 blobs at $T = 4$. The running time at $T = 4$ (200 time steps) was 173 s.

A closer view of the curve computed with $\delta = 0.1$ at $T = 3$ and $T = 4$ is shown in Figs. 2a and b. The closeup of the solution with $\delta = 0.05$, half the previous value, is shown in Figs. 2c and d. The maximum number of points in the case reached $N = 4522$ at $T = 4$. Consistent with previous computations, it can be observed that at a given time each vortex has more turns as δ becomes smaller.

Figure 3 presents the long time vortex sheet evolution. Due to the computational expense of direct summation no previous data exists for these times. The maximum number of points reached 34908 at $T = 10$, 53981 at $T = 11$ and 71346 at $T = 11.8$, respectively. The total CPU time required for the computations at time $T = 11.8$ was 18.6 h. Estimated direct summation, if feasible, would take approximately one month on the same machine. The points are plotted on the right half of each figure and a line interpolating the points is plotted on the left. The computation becomes unstable at $T > 11.8$ for these particular values of the parameters δ, Δt, and ε.

A sample of the structure of the domain partition defined by the fast algorithm is given in Fig. 4. To save time, the program does not erase the cells created at previous time steps once they become empty (the cell tree grows monotonically as the program runs). The cell refinement pictured here thus reflects the evolution of the vortex sheet in time.

5. CONCLUSIONS

We presented a new version of the fast algorithm for computing pairwise interactions in a system of n particles first introduced in [11]. The main advantage of the new technique consists in its versatility: it can be used for any decaying kernel, in both two- and three-dimensional applications. Moreover, since it is defined by a Taylor expansion, accuracy and complexity can be easily studied. We illustrate the use of the new method by studying vortex sheet rollup using vortex blob approximations. The tests reproduced previous results obtained using direct summation and extended them to longer simulation times and larger number of points. We are currently working on a parallel implementation of the algorithm on a KSR1 machine.

ACKNOWLEDGMENTS

We thank Professor Robert Krasny for pointing out the vortex sheet simulation problem as a good application for this algorithm and for his continuous interest and support during the course of this work. Computations were performed at the University of Michigan and the University of Houston. This research was supported in part through the Postdoctoral Research Associateship from the CISE Directorate of the NSF ASC-9310236 and by NSF Grant DMS-9204271.

REFERENCES

1. C. R. Anderson, *J. Comput. Phys.* **62,** 111 (1986).
2. C. R. Anderson, *SIAM J. Sci. Stat. Comput.* **13,** 923 (1992).
2A. A. W. Appel, *SIAM J. Sci. Stat. Comput.*, **6** (1985), pp. 85–103.
3. J. T. Beale and A. Majda, *Math. Comput.* **39,** 1 (1982).
4. J. T. Beale and A. Majda, *Math. Comput.* **39,** 29 (1982).
5. J. Barnes and P. Hut, *Nature* **324,** 446 (1986).
6. A. Brandt and A. A. Lubrecht, *J. Comput. Phys.* **90,** 348 (1990).
7. J. Carrier, L. Greengard, and V. Rokhlin, *SIAM J. Sci. Stat. Comput.* **9,** 669 (1988).
8. A. J. Chorin, *J. Fluid Mech.* **57,** 785 (1973).
9. A. J. Chorin and P. S. Bernard, *J. Comput. Phys.* **13,** 423 (1973).
9A. H-Q. Ding, N. Karasawa, W. A. Goddard III, *J. Comput. Phys.*, **97** (1992), pp. 4309–4315.
10. L. van Dommelen and E. A. Rundensteiner, *J. Comput. Phys.* **83,** 126 (1989).
11. C. I. Draghicescu, *SIAM J. Numer. Anal.* **31,** 1090 (1994).
12. F. H. Harlow, *The Particle-in-Cell Method in Fluid Dynamics,* Vol. 3, *Methods of Computational Physics* (Academic Press, New York, 1964), p. 319.
13. L. Greengard and V. Rokhlin, *J. Comput. Phys.* **73,** 325 (1987).
14. L. Greengard and V. Rokhlin, in *Lecture Notes in Mathematics,* Vol. 1360 (Springer-Verlag, New York/Berlin, 1988), p. 121.
15. W. Hackbusch, *Boundary Elements IX,* 1 (Springer-Verlag, Berlin, 1987), p. 463.
16. R. W. Hockney and J. W. Eastwood, *Computer Simulations Using Particles* (McGraw–Hill, New York, 1981).
17. R. Krasny, *J. Fluid Mech.* **184,** 123 (1987).
18. R. Krasny, in *Lectures in Applied Math.* (Am. Math. Soc., Providence, RI, 1991), p. 385.
19. R. Krasny, *J. Comput. Phys.* **65,** 292 (1986).
20. L. Rosenhead, *Proc. R. Soc. London Ser. A* **134,** 170 (1932).
21. G. Tryggvason, *J. Comput. Phys.* **80,** 1 (1989).