2-D Vortex Methods and Singular Quadrature Rules

John Strain *
Department of Mathematics
and
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

January 13, 2000

Abstract

A new high-order vortex method for the 2-D Euler equations is presented. The method eliminates smoothing by constructing a singular quadrature rule for the Biot-Savart law at each time step, using quadtrees and orthogonal polynomials. Theory and numerical experiments show that the method is accurate and efficient, yielding excellent long-term accuracy in almost optimal CPU time.

*Research supported by a NSF Young Investigator Award, Air Force Office of Scientific Research Grant FDF49620-93-1-0053, and the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

Contents

List of Figures

1 Introduction

Vortex methods are powerful and sophisticated numerical methods for computing incompressible turbulent flows Because they are grid-free and naturally adaptive, they create little or no numerical diffusion and preserve features which other methods may distort. Vortex methods are particularly useful when computing free-surface, free-space and external flows, because only the support of the vorticity need be discretized. Some recent work on vortex methods can be found in [?].

A typical vortex method involves several steps; velocity evaluation, vortex motion, diffusion and boundary conditions. In this paper, we focus on the most expensive and difficult step, the velocity evaluation. We employ standard techniques for the vortex motion and consider inviscid free-space flow to eliminate diffusion and boundary conditions. General background material on vortex methods is presented in Section 2.

The standard velocity evaluation approximates the Biot-Savart law by a fixed quadrature rule, with weights conserved by incompressibility and independent of the singularity in the Biot-Savart kernel. Smoothing is required to make the quadrature rule accurate.

There are three major and interrelated difficulties with the standard approach. First, the use of fixed quadrature weights loses accuracy as the flow becomes disorganized. Perlman [15] and Beale and Majda [4] observed large oscillations in the error during long-time integrations. These oscillations are not present in triangulated vortex methods [17], regridded methods [14], or Beale's method [3], which generate new weights at each step. Second, smoothing is required because the quadrature weights ignore the singularity; this lowers the order of convergence. Third, if any product integration [9, 10] or similar techniques are used to treat the singularity, the variable weights preclude the use of the fast multipole method [5] on which the practicality of the method depends.

This paper presents a different velocity evaluation method which overcomes these difficulties. A new quadrature rule at each step preserves long-time accuracy. Smoothing is unnecessary since the method integrates the Biot-Savart law with order-q accuracy for any fixed q. Only the nearby weights vary with the singularity, so the fast multipole method can still be applied.

Our method is described in Section 3. It is based on locally-corrected quadrature rules for multidimensional singular kernels [19] and proceeds in

stages. First a data structure groups the N vortices into cells convenient for integration. Then a global order-q quadrature rule which ignores the singularity is built. The fast multipole method evaluates this rule efficiently (yielding a regridded vortex method if smoothing is used). Finally, we correct the weights of vortices near the evaluation point and the appropriate terms of the velocity, using the detailed calculations from Appendix A.

Numerical results presented in Section 4 show that this method has several nice features. It runs in $O(N \log N)$ CPU time with N vortices and achieves essentially qth order accuracy for any specified q. It deals effectively with arbitrary initial distributions of vortices. Long-time accuracy is preserved. The method is naturally parallel since each point is corrected independently.

The method extends naturally to 3-D calculations, Navier-Stokes equations, and flows in bounded domains. It is equipped with a natural interpolation which gives the vorticity at any point (see Appendix B).

2 Vortex methods for 2-D Euler

The Euler equations for the velocity field $(u(x,y,t), v(x,y,t))$ of two-dimensional incompressible inviscid flow are

$$\begin{aligned} \dot{u} + uu_x + vu_y + p_x/\rho &= 0 && (2.1)\\ \dot{v} + uv_x + vv_y + p_y/\rho &= 0 && (2.2)\\ u_x + v_y &= 0, && (2.3) \end{aligned}$$

where subscripts denote partial derivatives, overdots denote time derivatives, ρ is the (constant) density of the fluid and $p(x,y,t)$ the pressure.

The vorticity $\omega := v_x - u_y$ satisfies the vorticity equation

$$\dot{\omega} + u\omega_x + v\omega_y = 0,$$

which implies that the vorticity is passively transported along streamlines. By incompressibility, we can write u in terms of a stream function ψ:

$$u = \psi_y, \qquad v = -\psi_x.$$

The definition of vorticity then yields a Poisson equation for the stream function:

$$-\Delta\psi = \omega.$$

In flow without boundaries with zero velocity at infinity, this implies the Biot-Savart law

$$(u(x,y,t), v(x,y,t)) = \int_{I\!R^2} K(z-z')\omega(z',t)\,dx'dy'. \qquad (2.4)$$

Here it is convenient to introduce the complex variable $z = x + \iota y$ (where $\iota = \sqrt{-1}$) and the Biot-Savart kernel

$$K(z) = \frac{\iota}{2\pi\bar{z}} = \frac{(-y,x)}{2\pi(x^2+y^2)}.$$

The flow map $\varphi : I\!R^2 \times [0,T] \to I\!R^2$ is defined so that $\varphi(\xi,t)$ is the position at time t of the fluid particle initially at ξ. Since a fluid particle moves with velocity (u,v), the Biot-Savart law (2.4) implies that $\varphi(\xi,t)$ satisfies

$$\dot{\varphi}(\xi,t) = \int_{I\!R^2} K(\varphi(\xi,t) - z)\omega(z,t)\,dxdy.$$

Changing variables $z \leftarrow \varphi(z,t)$ in the integral gives

$$\begin{aligned} \dot{\varphi}(\xi,t) &= \int_{I\!R^2} K(\varphi(\xi,t)-\varphi(z,t))\omega(\varphi(z,t),t)\,dxdy \\ &= \int_{I\!R^2} K(\varphi(\xi,t)-\varphi(z,t))\omega(z,0)\,dxdy \end{aligned} \tag{2.5}$$

since the Jacobian of $\varphi(\xi,t)$ is unity and vorticity is constant along streamlines.

Vortex methods use various recipes for evaluating the Biot-Savart integral numerically. Lagrangian methods usually evaluate (2.5), tracing back the vorticity to the initial time and usually losing accuracy as the initial grid distorts. Free-Lagrangian methods approximate the velocity at each time t via (2.4).

The point vortex method [16], for example, approximates (2.5) by

$$\dot{z_i} = \sum_{j\neq i} K(z_i - z_j)\omega(\xi_j,0)h^2$$

where initially the vortices $z_i(t)$ are the N vertices ξ_j of an equidistant grid with side h. This is very physical since it moves N point vortices with circulations $\Gamma_j = \omega(\xi_j)h^2$. Although the method converges [11], it presents serious computational difficulties: Since the kernel is unbounded, the computed velocity can blow up if vortices come too close. Chorin and Bernard showed in [?] that the point vortex method can give problems when computing the motion of vortex sheets.

Most vortex methods, however, have been based on Chorin's version [7] in which the singularity is smoothed by convolution with a blob function $g_\delta(z)$:

$$K_\delta = K * g_\delta, \qquad g_\delta(z) = \frac{1}{\delta^2} g\left(\frac{z}{\delta}\right).$$

This gives the standard vortex blob method

$$\dot{z_i} = \sum_{j=1}^{N} K_\delta(z_i - z_j)\omega(\xi_j,0)h^2.$$

Convergence theory for this method is presented in [1, 4, 8, 12]. The numerical behavior of this method has been studied in [15, 3], for example. It has been very widely used in practice and generalized to model complex

three-dimensional viscous turbulent flows with boundaries and combustion [6, 7, 13].

Triangulated vortex methods [17, ?] approximate ω in (2.4) by a piecewise linear function on a triangulation. For each t let $\mathcal{T}_h(t) = \{\tau_i(t)\}_{i=1}^{N_T}$ be a triangulation covering the support of ω, with N vertices $\{z_j(t)\}_{j=1}^{N}$, and let

$$V_h = \{v(z) \in C^0(I\!R^2) : v|_{\tau_i} \text{ is linear for each } i\}$$

be the space of continuous piecewise linear functions over $\mathcal{T}_\langle(\sqcup)$. At each time t the vorticity $\omega(z,t)$ is approximated by the piecewise linear interpolant $\omega_h(z,t) \in V_h$. The velocity is approximated by

$$u_h(z,t) = \int_{I\!R^2} K(z-z')\omega_h(z',t)\,dz' = \sum_{i=1}^{N_T} \int_{\tau_i} K(z-z')\omega_h(z',t)\,dz'.$$

A variant of the fast multipole method and a fast Delaunay triangulation scheme are used to speed up the calculation, and an adaptive initial triangulation scheme to resolve complex initial data. This method appears straightforward to extend to flows in bounded domains, three-dimensional problems, and viscous flows. However, it appears quite difficult to make a triangulated vortex method with higher than second-order accuracy in space.

Triangulated vortex methods can be viewed as generating a new quadrature rule—one based on product integration—at each time step. Another interesting approach to constructing a new rule at each step was presented by Beale [3].

In this paper, we construct a qth-order quadrature rule for the evaluation of the Biot-Savart law (2.4) at each time step. Thus we obtain efficient and accurate free-Lagrangian vortex methods of any desired order.

A general vortex method is outlined in Figure 1. Here the vortices are moved by a second-order Runge-Kutta method for simplicity; any ODE solver can be used. The velocity evaluation is described in detail in the next section.

Algorithm

Read parameters from input file:

   Time step k, initial and final times t_i and t_f.
   Initial vortex locations ξ_j and strengths $\omega_j = \omega(\xi_j, t_i)$ for $1 \le j \le N$.
   Control parameters
   Exact solution parameters (if available)

Set $t = t_i$.

Time loop: while $t < t_f$ do
   $t = t + k$
   Compute weights w_{ij} and evaluate velocities u_i of vortices z_i:
   $u_i = \sum_{j=1}^{N} w_{ij} K(z_i - z_j)\omega_j \approx \int K(z_i - z)\omega(z,t)dxdy$
   First half-step of second-order Runge-Kutta:
   $Z_i = z_i + ku_i$
   Compute weights w_{ij} and evaluate velocities U_i of temporary positions Z_i:
   $U_i = \sum_{j=1}^{N} w_{ij} K(Z_i - Z_j)\omega_j \approx \int K(Z_i - z)\omega(z,t+k)dxdy$
   Second half-step of second-order Runge-Kutta:
   $z_i = z_i + k(u_i + U_i)/2$
   Measure error, plot results, write output, etc.
End of time loop: end do

Figure 1: General outline of a free-Lagrangian vortex method.

3 Velocity evaluation

We now describe the velocity evaluation used in our method, focusing on quadrature rules for the Biot-Savart law with arbitrary quadrature points.

Our construction has four stages. First, we partition the smallest rectangle $B = [a, b] \times [c, d]$ containing all the vortices into rectangular cells each containing a fixed number of vortices, determined by the order of accuracy q. Second, we construct a quadrature rule of order q on smooth functions, ignoring the singularity. Third, the weights of this smooth rule are used to do an uncorrected velocity evaluation with the fast multipole method. Finally, we correct locally by recomputing the weights of vortices near each evaluation point and re-evaluating the appropriate terms of the sum. We conclude with error bounds for the corrected rule.

3.1 Data structure

We first partition B into rectangular cells containing precisely p or $p + 1$ points z_j each, where p will be chosen in Section 3.2.

Let $B = B_1$ be the level-0 root of the tree. Divide B_1 in half along its longest edge, with the dividing plane located so that each half of B_1 contains either $\lfloor N/2 \rfloor$ or $\lfloor N/2 \rfloor + 1$ points. This gives the level-1 cells B_2 and B_3. Recursively, split B_2 and B_3 along their longest edges to get B_4 through B_7, each containing $\lfloor N/4 \rfloor$ or $\lfloor N/4 \rfloor + 1$ points z_j. Repeat this procedure L times to get $M = 2^L$ cells B_i on the finest level L, numbered from $i = M$ to $i = 2M - 1$, each containing $p = \lfloor N/M \rfloor$ or $p + 1$ points z_j. The union of all the cells on any given level is B. The tree structure is stored by listing the boundaries of each cell $B_i = [a_i, b_i] \times [c_i, d_i]$ from $i = 1$ to $i = 2M - 1$, a total of $4 \cdot 2M$ numbers, and indexing the points into a list so that the points $z_j \in B_i$ are given by $j = j(s)$ for $s = b(i), \ldots, e(i)$ and three integer functions j, b and e. This can be done in $O(N \log N)$, but the simplest method requires sorting each cell before each subdivision, giving a total cost $O(N \log^2 N)$ for the tree construction with an $O(N \log N)$ sorting method such as Heapsort. Figure 2 shows an example. We note that hierarchical data structures with similar properties – though not this particular structure – have been extensively discussed in [18].

Remark: The tree structure permits efficient $O(L)$ lookup of the level-L cell containing any point $z \in B$. Begin at the root B_1 and discard all children not containing z; repeat recursively with B_1 replaced by the remaining child

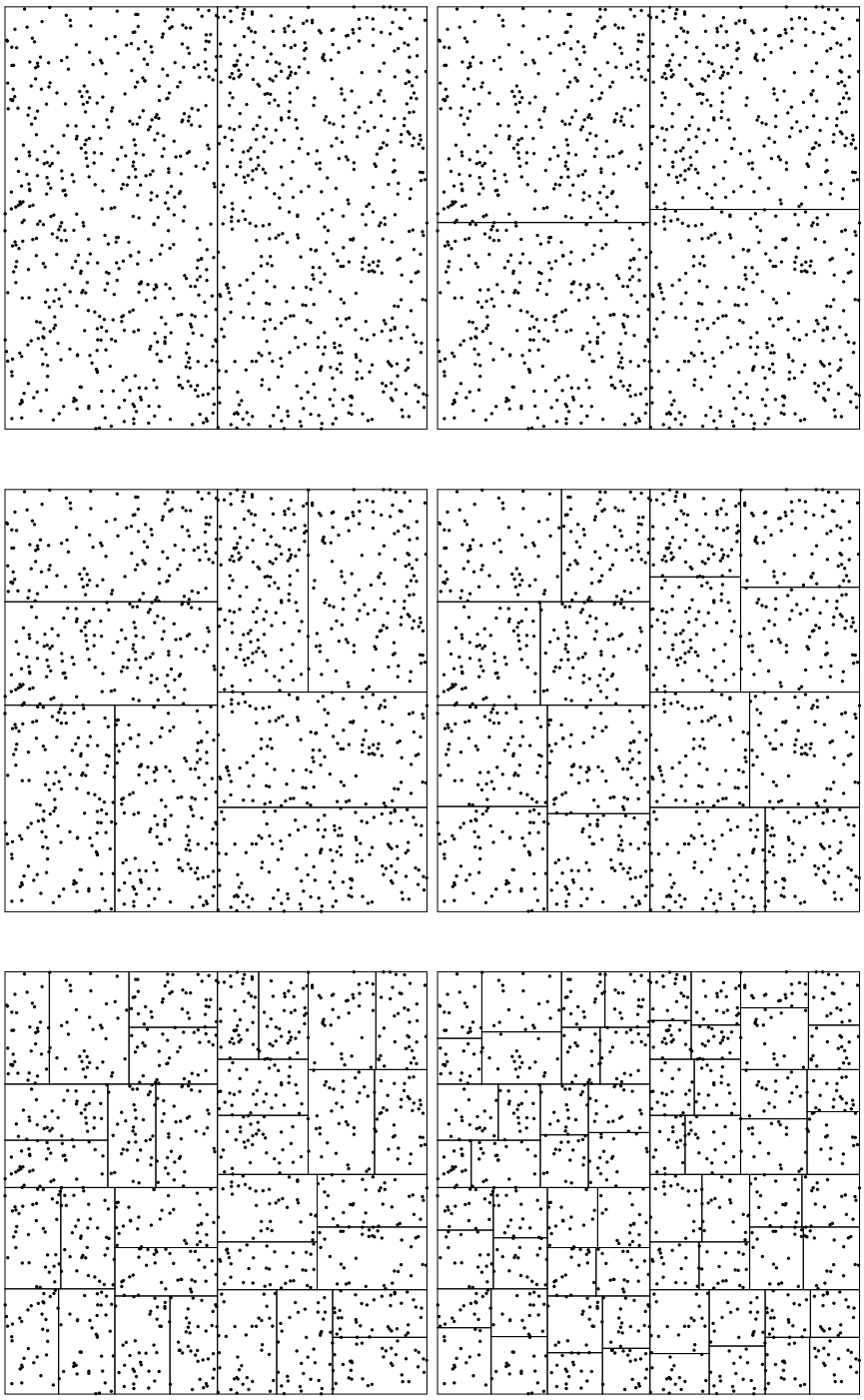

Figure 2: Levels 1 through 6 in the tree structure with $N = 1137$ pseudo-random uniformly distributed points on $[0, 1]^2$.

until level L is reached. Similarly, all cells intersecting a given rectangle R can be listed in time proportional to L times their number. We will use this to construct singular rules.

The cell structure can likewise be used to determine the smallest distance between two vortices

$$d_m = \min_{i \neq j} |z_i - z_j|.$$

First, the minimum distance d_m between any two distinct vortices in the same level-L cell is computed. This requires $O(Np^2)$ CPU time. Then for each vortex z_i, level-L cells intersecting the ball of radius d_m around z_i are listed and d_m is replaced by the minimum distance to any other vortex in those cells. Since the minimum d_m found so far is used at each step, it is rarely necessary to search additional cells and the total cost is $O(N)$.

3.2 Smooth rules

We now construct qth-order quadrature rules with the N quadrature points z_j for integrating smooth functions over the rectangle B. Let $q \geq 1$ be the desired order of accuracy of the rule, assume $N \geq m := q(q+1)/2$, and choose an integer $L \geq 0$ with $p := \lfloor N/2^L \rfloor \geq m$. The data structure just constructed divides B into $M = 2^L$ rectangular subcells B_i with disjoint interiors such that B is their union and each B_i contains either p or $p+1$ points z_j. On each B_i, we construct local weights W_i^j for $z_j \in B_i$ which integrate the m monomials $x^k y^l$ with $0 \leq k+l \leq q-1$ exactly over B_i. Because of the well-known ill-conditioning of the power basis, we construct these weights by solving the following equivalent system of m linear equations in at least p unknowns:

$$\sum_{z_j \in B_i} P_k(x_j) P_l(y_j) W_j^i = \int_{B_i} P_k(x) P_l(y) dx dy = \delta_{k0}\delta_{l0}|B_i| \qquad 0 \leq k+l \leq q-1. \tag{3.1}$$

Here $|B_i| = (b_i - a_i)(d_i - c_i)$ is the area of B_i and

$$P_k(x) = p_k(t), \qquad x = x_m + t x_h,$$

where $p_k(t)$ are the usual Legendre polynomials on $[-1,1]$ and $x_m = (b_i + a_i)/2$, $x_h = (b_i - a_i)/2$, with similar expressions for the y variable. Since $p \geq m$, this system of m equations in at least p unknowns generically has solutions. We compute the solution W_j^i of least Euclidean norm, using the

QR decomposition routine from LAPACK [2]. Section 3.3 discusses what to do when no solution exists. The weights of the rule W are then defined to be $W_j = W_j^i$ where $z_j \in B_i$.

Remark: The technique employed to construct this quadrature rule has many generalizations and applications. The general idea is that we approximate a linear functional $F(\varphi)$ of functions φ on each cell by a weighted combination of evaluations $E_i(\varphi) = \varphi(z_i)$ at the points z_i, with the weights determined to integrate polynomials of degree $\leq q-1$ exactly over each cell *and* have small norm. For example, suppose we want to interpolate the values of the vorticity to a point $z = (x, y)$ which is not one of the vortices z_i. We can construct weights for interpolation by requiring them to be exact on monomials of degree $\leq q-1$ on the cell B_i where z lies, yielding qth-order accuracy on smooth functions. This yields a least-squares problem similar to the equations (3.1) for the smooth rule integration weights, but with $P_k(x)P_l(y)$ replacing $\int_{B_i} P_k(x)P_l(y)dxdy$ on the right-hand side. This technique, which we use to contour the vorticity, is discussed further in Appendix B.

3.3 Error bounds

The weights W_j now integrate all monomials $x^k y^l$ with $0 \leq k+l \leq q-1$ exactly over all level-L cells B_i for $M \leq i \leq 2M-1$. In [19], we showed that this property alone results in order-q accuracy, with a condition number appearing in the error bound:

Theorem 1 *Let $B = \cup_{i=1}^M B_i$ where $B_i = [a_i, b_i] \times [c_i, d_i]$. Suppose that W integrates $x^k y^l$ exactly over each B_i for $0 \leq k+l \leq q-1$. Then for any C^q function g on B, the error*

$$E = \int_B g(z)dxdy - \sum_{j=1}^N W_j g(z_j)$$

satisfies

$$|E| \leq \Omega |B| (h/2)^q \sum_{k+l=q} \frac{1}{k!l!} ||\partial_x^k \partial_y^l g||_{C^0(B)}$$

where $h = \max_i \max(b_i - a_i, d_i - c_i)$ is the longest cell edge,

$$\Omega = 1 + \frac{1}{|B|} \sum_{j=1}^N |W_j|$$

is the condition number of the rule W, $|B| = (b-a)(d-c)$ *is the area of* B, *and the* C^0 *norm is defined by*

$$||\varphi||_{C^0(B)} := \max_{z \in B} |\varphi(z)|$$

for continuous functions φ *on* B.

Note that Ω plays the role of a condition number for W, mediating between the intrinsic difficulty of integrating g (as measured by the derivatives of g) and the accuracy of the final result. In general, Ω cannot be bounded for arbitrary points, but we can easily compute it a posteriori, yielding an excellent diagnostic for the quality of the rule. If all the weights are positive, $\Omega = 2$; otherwise, $\Omega > 2$.

Remark: There are several ways to reduce each cell condition number $\Omega^i = 1 + \frac{1}{|B_i|} \sum_{z_j \in B_i} |W_j|$ and thus obtain a better global condition number $\Omega = \sum_i \Omega_i$. Usually taking more points per cell reduces Ω, since the additional degrees of freedom are not needed to satisfy (3.1) and can be applied to reducing the 2-norm of W_j^i. However, this increases the cost of computing W considerably and increases the cell size h, so taking larger p is not cost-effective if applied globally. It can be applied adaptively, however, by going up to a different level of the tree structure when necessary. To implement this, we specify a tolerance Ω_m. When $\Omega^i \geq \Omega_m$, we merge B_i with its sibling in the tree structure, obtaining a cell B_I containing twice as many points z_j. We then recompute all weights W_j for which $z_j \in B_I$, usually obtaining $\Omega^I \ll \Omega_m$ at the cost of a larger QR decomposition and a larger cell size h. If Ω^I is still too large, the process may be repeated.

This adaptive technique also permits treatment of the degenerate cases when no solution exists to (3.1) on cell B_i, because the points z_j are not in sufficiently general position. Such a cell can be merged with its sibling, after which a solution is much more likely to exist. The process may be repeated if necessary.

3.4 Uncorrected velocity evaluation

Next we evaluate the uncorrected sums

$$u(z_i) = \sum_{j=1}^{N} W_j K(z_i - z_j) \omega_j \qquad 1 \leq i \leq N$$

which approximate the Biot-Savart law (2.4), using the weights W_j designed for smooth functions and ignoring the singularity of the Biot-Savart kernel $K(z)$ when $z = 0$. We exclude the infinite term $j = i$. Note that if a smoothed kernel is used in place of K, we have a regridded vortex method which may be more accurate than standard vortex methods.

Direct evaluation of these sums costs $O(N^2)$ CPU time, which rapidly becomes prohibitive for the large numbers of vortices needed to model interesting flows. Thus we use the adaptive fast multipole method (FMM) of [5] to evaluate this sum to any prescribed accuracy ϵ in $O(N \log N \log \epsilon)$ CPU time. In practice, the FMM is faster than direct evaluation on a Sparc-2 with $\epsilon = 10^{-7}$ whenever $N \geq 300$, and much faster when $N \geq 1000$ or so.

3.5 Singular rules

We now select and correct certain weights W_j of the smooth rule W, for each evaluation point z_i, to produce a singular rule w which will integrate singular functions $f(z) = \varphi(z)K(z - z_i)$ more accurately when φ is smooth. For convenience let $\sigma(z) = K(z - s)$ where $s = z_i$.

The weights to be corrected are selected by forming a list of cells B_i in the tree structure built for the smooth rule W and correcting all the weights W_j for vortices z_j lying in some cell on the list. For each cell B_i on the list, we construct the corrected weights w_j for $z_j \in B_i$ by requiring w_j to satisfy the linear system of $2m$ equations which expresses that $P_k(x)P_l(y)\sigma(z)$ is integrated exactly for $0 \leq k + l \leq q - 1$:

$$\sum_{z_j \in B_i} w_j P_k(x_j) P_l(y_j) \sigma(z_j) = \int_{B_i} P_k(x) P_l(y) \sigma(z) dx dy \tag{3.2}$$

for $0 \leq k + l \leq q - 1$ and both real and imaginary parts of σ. Exact formulas for the integrals on the right-hand side of (3.2) are derived in Appendix A.

In order for these equations generically to have solutions w, we cannot use the cells B_i on the lowest level L of the tree structure, because each of these contains only $p \geq m$ or $p + 1$ of the points z_j. Instead, we use half as many larger cells on level $L' := L - 1$, each containing at least $p' := N/2^{L'} \geq 2m$ points. Thus (3.2) will generically be solvable by a QR decomposition, obtaining w as least 2-norm solution if it exists.

In order to keep the number of corrected cells bounded while correcting enough to ensure accuracy, we select cells for correction by the following

approach. The user specifies a dimensionless correction radius r_c, typically of order unity. We find the cell $B_i = [a_i, b_i] \times [c_i, d_i]$ in which the evaluation point lies, and let $R = [-r_c(b_i - a_i)/2, r_c(b_i - a_i)/2] \times [-r_c(d_i - c_i)/2, r_c(d_i - c_i)/2]$. We then correct all cells intersecting the rectangle $s + R$. This scales the size of the corrected area to the local cell size and therefore to the local density of nodes, keeping the number of corrected points per evaluation point s bounded as $N \to \infty$ with r_c fixed. We found $r_c \approx 1$ to give excellent results in practice. The lookup of cells to be corrected costs only $O(L)$ per cell. The complete algorithm is presented in Figure 3.

3.6 Error bounds

The key requirement in the error bound proved in [19] is that we must correct all cells sufficiently close to the evaluation point s. For notational convenience, let's renumber the M cells used in the singular rule, so that the first n are corrected and the last $M - n$ are not: thus $B = \cup_{i=1}^{M} B_i$ where each cell B_i contains at least $2m$ points for $1 \le i \le n$ and at least m points for $n + 1 \le i \le M$. Let $h = \max_i \max(b_i - a_i, d_i - c_i)$ be the maximum cell edge. Then we have weights w_j such that

$$\int_{B_i} \sigma(z) x^k y^l dx dy = \sum_{z_j \in B_i} w_j \sigma(z_j) x_j^k y_j^l$$

for $0 \le k + l \le q - 1$ and $1 \le i \le n$, while

$$\int_{B_i} x^k y^l dx dy = \sum_{z_j \in B_i} w_j x_j^k y_j^l$$

for $0 \le k + l \le q - 1$ and $n + 1 \le i \le M$. Assume that σ is C^q and that its qth order derivatives satisfy a growth condition:

$$|\partial_x^k \partial_y^l \sigma(z)| \le C \delta^{-2-k-l} \qquad |z - s| \ge \delta > 0$$

for $k + l = 0$ and $k + l = q$. This assumption is very mild since it does not even guarantee that σ is in $L^1(B)$. It is satisfied by the Biot-Savart kernel $\sigma(z) = K(z - s)$.

With these assumptions, we proved the following error bound in [19]:

Theorem 2 *Fix $\epsilon > 0$ and correct the $O(1)$ cells intersecting $s + R$ where $r_c = \epsilon^{-1/q}$. Then the error in integrating $\varphi \cdot \sigma$ over B with the locally corrected*

Velocity evaluation

Set parameters:
  Degrees of freedom required per cell: $m = q(q+1)/2$.
  Top level in cell structure: $L = \lfloor \log_2(N/m) \rfloor$.
  Points per cell: $p = N/2^L$.
Construct cell data structure:
  $B_1 = B =$ smallest rectangle enclosing all the points z_i.
  do $l = 1, L-1$
    Divide level-l cells along longest edge with half the points
    in each subcell, yielding level-$l+1$ cells.
  end do
  Result: 2^L cells on level L with p or $p+1$ points each.
Compute smooth weights W_i one cell at a time.
  do $i = 1, 2^L$
    Compute least-2-norm solution W of
    $\sum_{z_j \in B_i} W_j P_k(x_j) P_l(y_j) = \delta_{k0}\delta_{l0}|B_i|$ for $0 \le k+l \le q-1$
  end do
Use smooth weights and FMM to evaluate uncorrected velocity field U:
  do $i = 1, N$
    $U_i = \sum_{j=1}^{N} W_j K(z_i - z_j)\omega_j$
  end do
Correct velocity field u_i one point at a time.
  do $i = 1, N$
    $u_i = U_i$
    Find cell B_I containing evaluation point z_i.
    List cells B_n intersecting rectangle centered at z_i, with size $r_c B_I$.
    foreach B_n do
      Form and solve least-squares problem for W_j on cell B_n:
      $\sum_{z_j \in B_n} P_k(x_j) P_l(y_j) K(z_i - z_j) W_j = \int_{B_n} P_k(x) P_l(y) K(z_i - z) dxdy$
      Correct contribution from B_n to U_i to get u_i:
      $u_i = u_i + \sum_{z_j \in B_n} (w_j - W_j) K(z_i - z_j)\omega_j$
    end for
  end do

Figure 3: Velocity evaluation algorithm.

rule w is bounded by

$$\begin{aligned} E &= \left| \int_B \varphi(z)\sigma(z)dxdy - \sum_{j=1}^{N} w_j \varphi(z_j)\sigma(z_j) \right| \\ &\leq C(\Omega + \Omega_\sigma) \left(|\log h| h^q ||\varphi||_{C^q(B)} + \epsilon ||\varphi||_{C^0(B)} \right) \end{aligned}$$

where C depends only on B and the derivatives of σ,

$$\Omega = 1 + \frac{1}{|B|} \sum_{j=1}^{M} |w_j| \qquad \text{and} \qquad \Omega_\sigma = 1 + \frac{1}{|B|} \sum_{j=1}^{M} |w_j \sigma(z_j)|.$$

This theorem implies that we need only correct a fixed number of points as $N \to \infty$ if we are satisfied with an "$O(\epsilon + h^k)$" error bound. This error bound is natural in this context, since the fast multipole method itself evaluates the uncorrected velocity field only to accuracy ϵ.

Remark: It is not necessary to correct the local weights w_j to the same order of accuracy as the global weights W. Indeed, an efficient approach is to choose distinct global and local orders q_g and q_l, with $q_g \geq q_l$, corresponding to roughly equal degrees of freedom $m_g = q_g(q_g + 1)/2$ and $m_l = q_l(q_l + 1)$. This better balances the work required by the global and local weight constructions, since cells of the same size can be used for both calculations. Table 1 shows good choices of global and local weights which roughly balance the degrees of freedom. In practice, almost all the CPU time is devoted to local corrections.

q_g	1	2	3	4	4	5	6	7	9	10
q_l	1	1	2	2	3	3	4	5	6	7
m_g	1	3	6	10	10	15	21	28	45	55
m_l	2	2	6	6	12	12	20	30	42	56

Table 1: Global and local orders (q_g, q_l) and required degrees of freedom (m_g, m_l) per cell.

Remark: In practice, one does not know the exact number of vortices in advance, and choice of L may therefore be difficult. Too many points per cell is wasteful, while too few can lead to large errors if the number of degrees of freedom is not enough to achieve the desired order of accuracy. Thus our

code determines L internally, using a user-specified parameter $S \geq 1$ which indicates the degree of safety desired. The code determines L such that each level-L cell contains at least $\lfloor Sm \rfloor$ points, where $m = q(q+1)/2$. If $q_g \neq q_l$, two safety parameters S_g and S_l are used.

4 Implementation and numerical results

We implemented a version of the algorithm described above in Fortran and studied several numerical examples. We measured the accuracy and efficiency of the velocity evaluation scheme in isolation, using smooth initial vorticity fields. Then we measured the error in a long-time calculation with Perlman's standard test example. Finally, we studied the long-time interaction of several smooth patches of vorticity.

4.1 Velocity evaluation

First, we studied the accuracy of the velocity evaluation for several choices of the orders q_g and q_l, using two smooth initial vorticity fields for which the velocity can be evaluated analytically. These are the Gaussian

$$\omega_G(x,y,t) = e^{-(x^2+y^2)/\rho^2}, \qquad \rho = \frac{1}{2},$$

and the Perlman test case

$$\omega_P(x,y,t) = \Big(\max(0, 1-x^2-y^2)\Big)^7.$$

The corresponding velocity fields are

$$(u,v) = \frac{\rho^2(e^{-(x^2+y^2)/\rho^2}-1)}{2(x^2+y^2)}(y,-x)$$

and

$$(u,v) = \frac{(\max(0,1-x^2-y^2))^8 - 1}{16(x^2+y^2)}(y,-x).$$

These are stationary radial solutions of the Euler equations with shear and popular test cases for vortex methods.

In order to test our method, we used several different techniques to generate the initial distribution of vortices. The most severe was simply to generate uniformly distributed random vortices on the support of the initial vorticity. The least severe is to use the vertices of an equispaced grid or a grid adapted to the initial vorticity, as in [17]. An intermediate choice is the following adaptive random grid. Given N and n with $n^2 < N$, first distribute n^2 vortices uniformly over the smallest rectangle R enclosing the support of

the vorticity. To do this, divide R into a $n \times n$ grid and choose a point z_i randomly in the ith grid cell. Of the remaining $M = N - n^2$ vortices, put

$$m_i = \left\lfloor \frac{M|\omega(z_i, 0)|}{\sum_i |\omega(z_i, 0)|} \right\rfloor$$

or $m_i + 1$ random vortices in the ith cell of the $n \times n$ grid. Thus the remaining vortices are distributed in regions where the vorticity is large, providing some degree of adaptivity despite their randomness.

We generated $N = 200, 400, \ldots, 51200$ vortices in an adaptive random grid on the square $[-2, 2]^2$ with $n^2 \approx N/4$ for each of the above two examples and evaluated the velocity at each of the vortices, using rules of orders $(q_g, q_l) = (2, 1), (3, 2), (5, 3)$ and $(6, 4)$. We took $r_c = 1$ and $S_g = S_l = 1.5$ in all cases. The resulting relative discrete L^1 errors E_G and E_P for ω_G and ω_P respectively, memory usages M (in thousands of integers), CPU times T in seconds on a Sparc-2 workstation and other statistics are reported in Table 2. Figure 4 shows some of the cell structures for these calculations.

The velocity evaluation clearly requires CPU times and memory proportional to the number of vortices N, with a constant of proportionality depending on the order q. The CPU time is dominated by the correction of local weights, which in turn is mostly due to solving least-squares problems. Exact evaluation of the Biot-Savart kernel and forming the matrix of Legendre polynomials require less than a few percent of the correction time all told. Thus we expect the method would be a natural candidate for parallel computing, since each point is corrected independently. At some cost in memory, the least-squares systems could be farmed out to many small processors to solve.

In all cases, the error appears to be decreasing roughly according to the theoretical estimate $O(\epsilon + h^{q_l})$. Note that when N doubles, the maximum cell size h decreases by a factor $\sqrt{2}$, so we expect the error to decrease by a factor $2^{q_l/2}$ until ϵ is reached. The value of ϵ appears to be of order 10^{-3} to 10^{-4}, sufficient for two to three digit accuracy; smaller ϵ would require larger r_c, implying more corrected cells per point and longer running time. As one would expect, higher-order methods require more vortices to yield higher-order convergence. However, we note that higher-order methods as well as lower-order ones have $\Omega \approx 2$ for large N.

$(q_g, q_l) = (2, 1)$						
N	E_G	E_P	M	T	L	Ω
200	.55-0	.25+1	4.1	2.5	5	2.32
400	.94-1	.49-0	8.3	4.7	6	2.23
800	.35-1	.63-1	16.5	7.6	7	2.07
1600	.17-1	.26-1	33.9	13.8	8	2.06
3200	.87-2	.10-1	65.7	26.1	9	2.02
6400	.49-2	.55-2	131	53.9	10	2.04
12800	.31-2	.34-2	263	107	11	2.02
25600	.21-2	.23-2	525	217	12	2.02
51200	.14-2	.16-2	1050	423	13	2.04

$(q_g, q_l) = (3, 2)$						
N	E_G	E_P	M	T	L	Ω
200	.14+1	.61+1	3.8	7.3	4	6.89
400	.20-0	.22+1	7.6	15.3	5	3.36
800	.70-1	.92-1	15.1	25.0	6	2.41
1600	.88-2	.46-1	30.1	44.0	7	2.14
3200	.47-2	.84-2	60.1	78.3	8	2.10
6400	.22-2	.35-2	120	147	9	2.08
12800	.14-2	.15-2	240	294	10	2.06
25600	.10-2	.11-2	480	579	11	2.06
51200	.62-3	.70-3	961	1180	12	2.10

$(q_g, q_l) = (5, 3)$						
N	E_G	E_P	M	T	L	Ω
200	.14+1	.17+2	3.6	16.8	3	22.7
400	.18+1	.49+2	7.2	41.2	4	66.0
800	.27-0	.37+1	14.4	73.1	5	12.2
1600	.24-1	.56-0	28.7	123	6	3.6
3200	.33-2	.30-1	57.3	223	7	3.6
6400	.90-3	.27-2	114	438	8	2.9
12800	.66-3	.96-3	229	860	9	2.8
25600	.30-3	.39-3	458	1770	10	2.8
51200	.27-3	.31-3	916	3530	11	2.8

$(q_g, q_l) = (6, 4)$						
N	E_G	E_P	M	T	L	Ω
200	.20-0	.16+1	3.5	39.3	2	47.2
400	.99-0	.41+2	7	115	3	814
800	.10+1	.13+2	14	298	4	219
1600	.10-1	.14+1	28	591	5	18.8
3200	.95-2	.11-0	56	1030	6	4.39
6400	.57-3	.50-2	112	1730	7	2.60
12800	.24-3	.66-3	223	2810	8	2.33
25600	.15-3	.25-3	447	5540	9	2.22
51200	.72-4	.91-4	893	10900	10	2.18

Table 2: Relative L_1 errors E_P and E_G, memory usage (in K) M, CPU times T (in seconds on a Sparc-2), number of levels L and condition number Ω for evaluating the velocity due to Perlman-type and Gaussian vorticity fields with N points and a quadrature rule of orders (q_g, q_l).

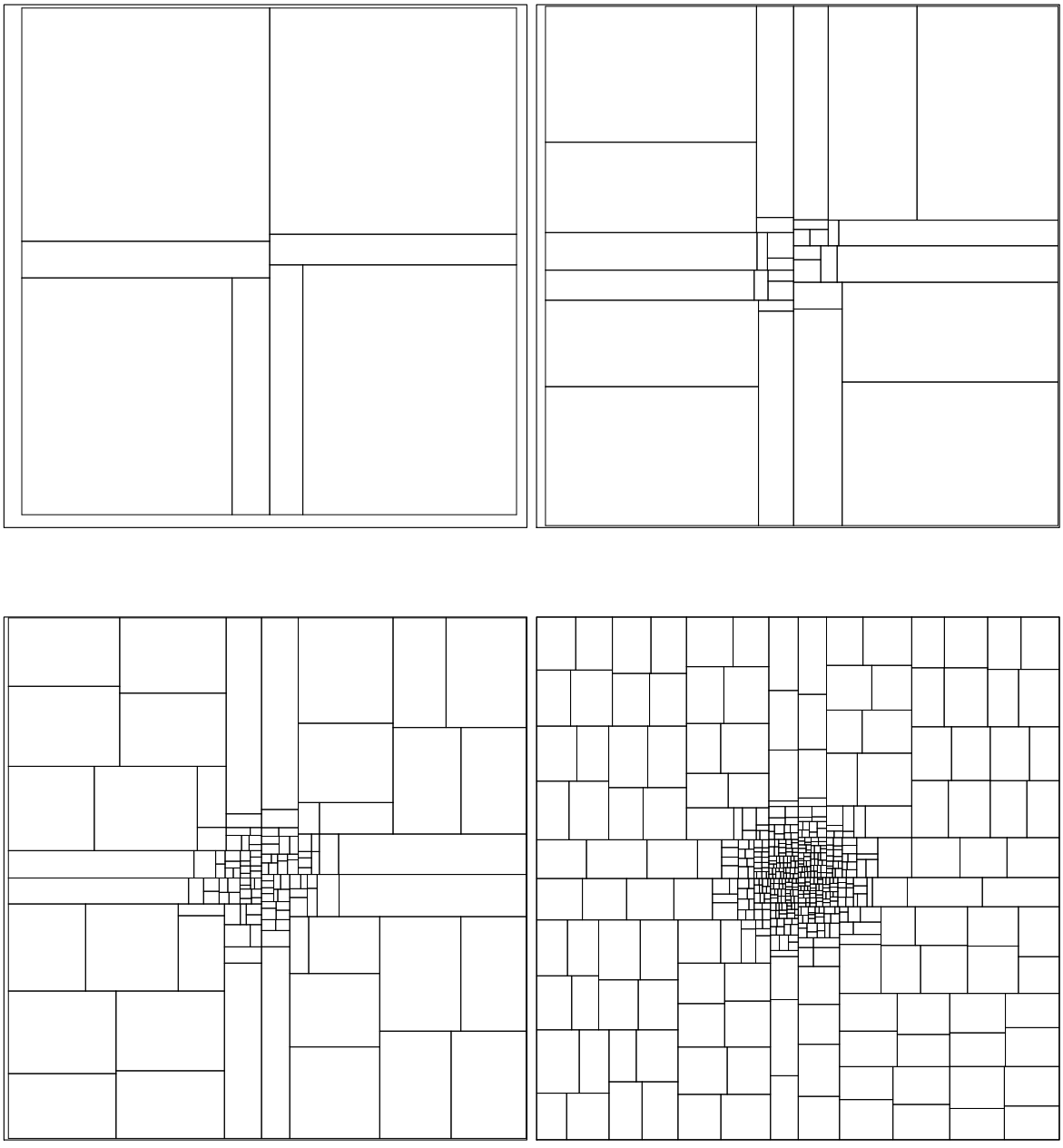

Figure 4: Cell structures for velocity evaluation with adaptive random grid for a Gaussian vorticity with $(q_g, q_l) = (3, 2)$ and $N = 100, 400, 1600, 6400$ vortices.

4.2 Long-time accuracy

We also tested the long-time accuracy of the method on the Perlman test case, running for $0 \le t \le 100 \approx 32\pi$, a final time at which the fastest-moving particles of fluid (near the origin) have completed 8 revolutions while the slowest have completed only one. This strong shear is usually considered a severe test for a vortex method. We started with an adaptive random grid and used fourth-order Runge-Kutta for the time integration, with quadrature of orders $(2, 2)$. We used $N = 400, 800, 1600, 3200, 6400, 12800$ vortices with $N_T = 100, 140, 200, 280, 400, 560$ time steps from $t = 0$ to $t = 100$. In all cases we took $r_c = 1$ and $S_g = S_l = 1.5$. The resulting relative discrete L^1 errors in the velocity are shown in Figure 5, in base-2 logarithmic scale to aid in the study of convergence. They clearly confirm the long-time second-order accuracy of the method; the oscillations observed by Perlman [15] are not seen. This pleasant behavior of the error is undoubtedly due to the ab initio calculation of the quadrature rule at each time step. Particularly encouraging is the slower increase of the error when more vortices are employed. To ensure that this behavior was not due to the reduction of the time step, the computation has been repeated with smaller time steps, yielding the same results.

4.3 Interacting vortex patches

We also computed the evolution of several interacting vortex patches, each given by a shifted and scaled Gaussian. Thus the initial vorticity is given by

$$\omega(x, y, 0) = \sum_{j=1}^{m} \Omega_j \exp(-((x - x_j)^2 + (y - y_j)^2)/\rho_j^2)$$

where $(x_j, y_j) \in [-2, 2]^2$, $\rho_j \in [0, 1]$ and $\Omega_j \in [-1, 1]$ are pseudorandom uniformly distributed numbers such that the circles of radius ρ_j with centers (x_j, y_j) do not overlap. We used $m = 4$ patches, with parameters shown in Table 3. We used $N = 6400, 12800$ and 25600 vortices with quadratures of orders $(q_g, q_l) = (2, 2)$, with $r_c = 1$ and $S_g = S_l = 1.5$. The evolution of this flow is shown in Figure 6. The figures show the final result at $t = 28$ with three different values of N; the large-scale features of the results are clearly converged.

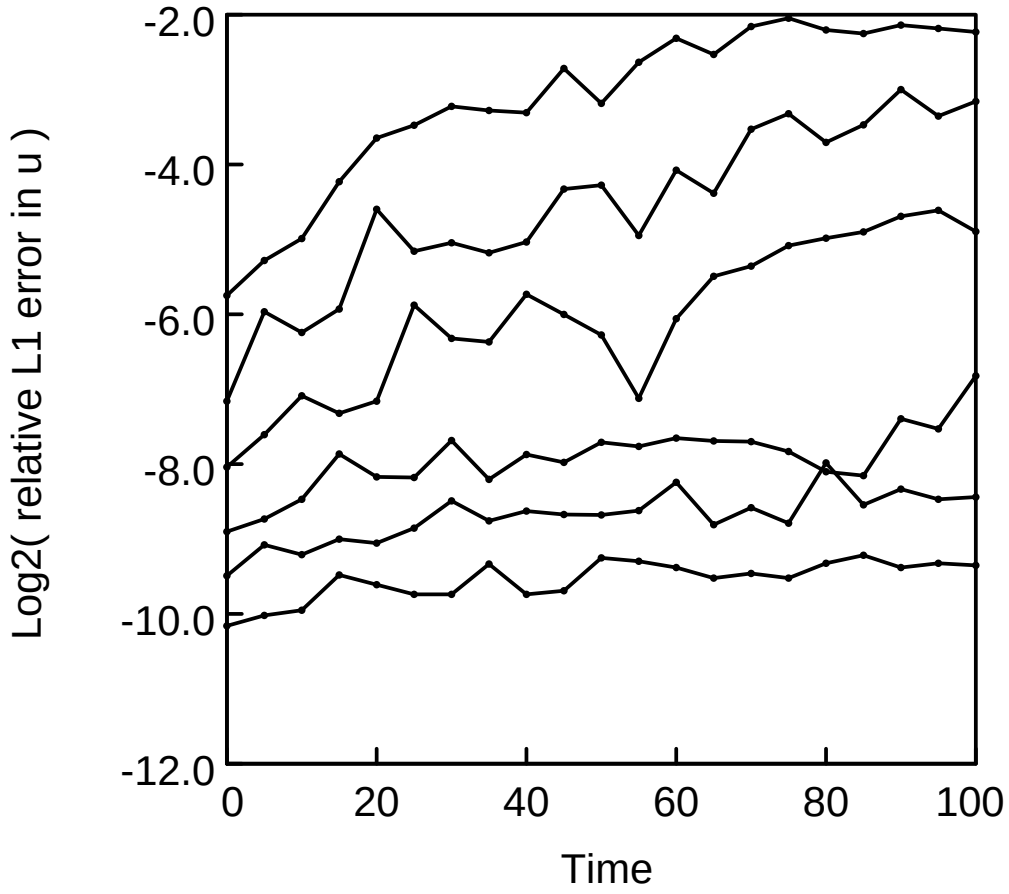

Figure 5: Relative L_1 errors in the evolution of a Perlman patch to time $t = 100$, computed with a quadrature rule of orders $(2, 2)$. From top to bottom, the lines plotted are the base-2 logarithms of the relative discrete L^1 errors in the velocity computed with $N = 400, 800, 1600, 3200, 6400, 12800$ points and $N_T = 100, 140, 200, 280, 400, 560$ time steps up to $t = 100$.

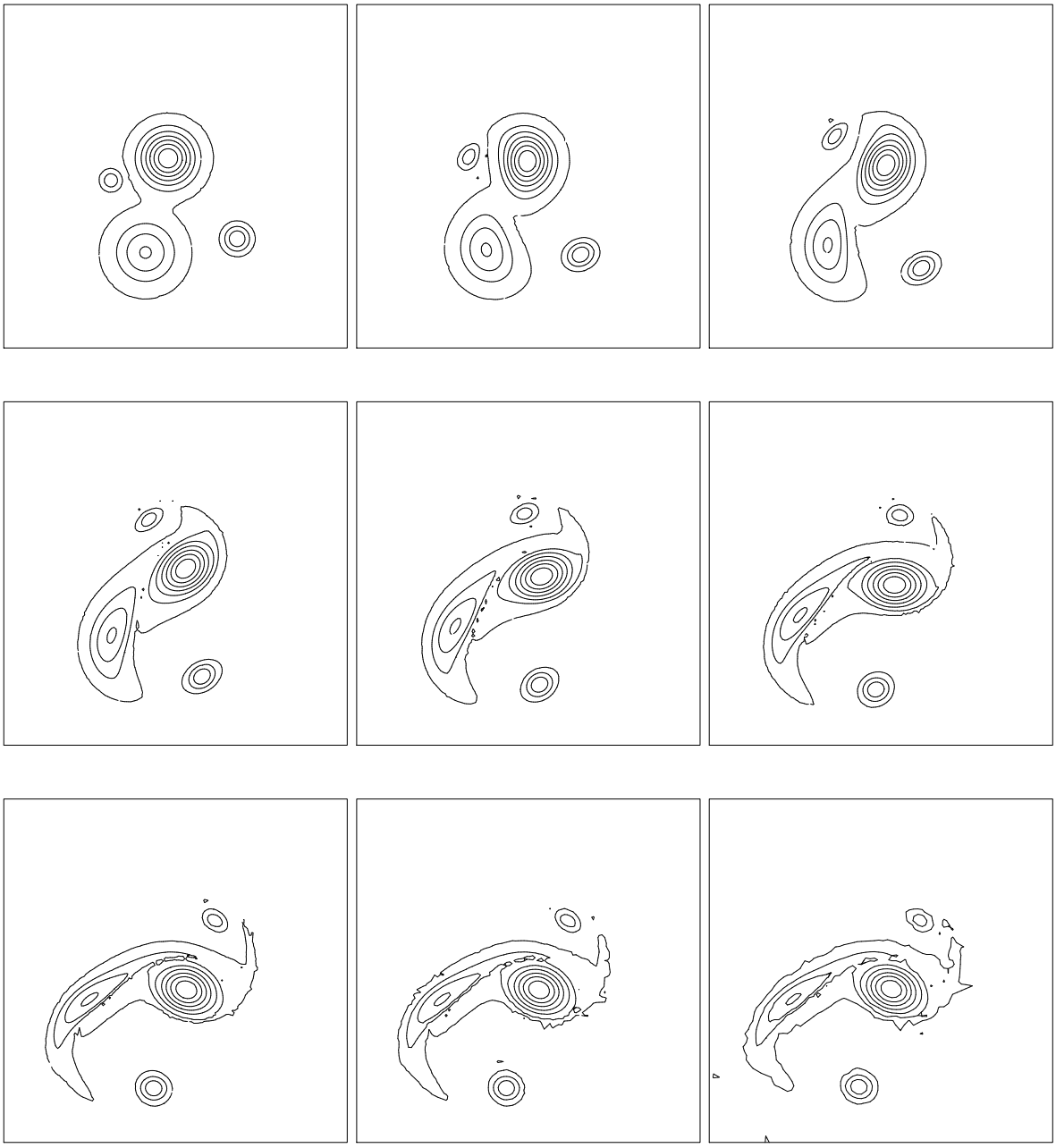

Figure 6: Evolution of four patches of Gaussian vorticity, computed with $(q_g, q_l) = (2, 2)$ and $N = 25600$ vortices. The bottom row shows the final result ($t = 28$) computed with (from left to right) $N = 25600$, 12800 and 6400 vortices.

j	x_j	y_j	ρ_j	Ω_j
1	-0.6988	-1.7756	0.6768	-0.4515
2	1.4363	-1.4566	0.3294	0.4968
3	-0.1722	0.4175	0.5807	-0.9643
4	-1.5009	-0.0937	0.2504	0.3418

Table 3: Strengths Ω_j, centers (x_j, y_j) and scales ρ_j for four Gaussian patches of vorticity.

A Exact integration formulas

Given z_0, a cell $C = [a, b] \times [c, d]$ and a degree q, we require the integrals

$$u_{ij}(z_0) = \int_C \frac{1}{z_0 - z} P_i(x) P_j(y) dxdy$$

for $0 \leq i + j \leq p = q - 1$. Here

$$P_i(x) = p_i(t) \qquad\qquad t = (x - x_m)/x_h$$

with $x_m = (b + a)/2$, $x_h = (b - a)/2$ and p_i the Legendre polynomial on $|t| \leq 1$ defined by the recurrence

$$p_0(t) = 1, \qquad p_1(t) = t, \qquad p_{i+1}(t) = \frac{2i + 1}{i + 1} p_1(t) p_i(t) - \frac{i}{i + 1} p_{i-1}(t)$$

for $i \geq 1$. Similar expressions hold for the y variable.

The calculation proceeds in three steps: First, we express the product of Legendre polynomials in the form

$$P_i(x) P_j(y) = \sum_{k=0}^{p} \sum_{l=0}^{p} Q_{kl}^{ij} (z_0 - z)^k (\bar{z}_0 - \bar{z})^l \tag{A.1}$$

where $z = x + \iota y$. We define $Q_{kl}^{ij} = 0$ for convenience, whenever any of i, j, k or l is negative or $k + l$ exceeds $i + j$. Then we have

$$\begin{aligned} u_{ij}(z_0) &= \sum_{k=0}^{p} \sum_{l=0}^{p} Q_{kl}^{ij} \int_C (z_0 - z)^{k-1} (\bar{z}_0 - \bar{z})^l dxdy \\ &= \sum_{k=0}^{p} \sum_{l=0}^{p} Q_{kl}^{ij} S_{kl}(z_0 - C) \end{aligned}$$

where

$$S_{kl}(C) = \int_C z^{k-1} \bar{z}^l dxdy$$

Step two is to evaluate S_{kl} when $k = 0$, and step three is to evaluate S_{kl} when $k > 0$. Note that we need only evaluate S_{kl} once and for all, for $0 \leq k+l \leq p$.

Step one is done by using the recurrence for Legendre polynomials, in the form

$$P_{j+1}(y) = \frac{2j + 1}{j + 1} P_1(y) P_j(y) - \frac{j}{j + 1} P_{j-1}(y)$$

which follows from (A.1). Multiplying by $P_i(x)$, using (A.1) twice and equating coefficients gives

$$Q_{kl}^{i,j+1} = \frac{2j+1}{j+1}\left[Q_{00}^{01}Q_{kl}^{ij} + Q_{10}^{01}Q_{k-1,l}^{ij} + Q_{01}^{01}Q_{k,l-1}^{ij}\right] - \frac{j}{j+1}Q_{kl}^{i,j-1}.$$

We use this recurrence to evaluate $Q_{kl}^{i,j+1}$ for $i = 0, 1, 2, \ldots, p$ and $j = 1, 2, \ldots, p-i$.

To evaluate the first two columns of the recurrence, for which $j = 0, 1$, we use the corresponding recurrence on i, which is derived by interchanging x and y and i and j:

$$Q_{kl}^{i+1,j} = \frac{2i+1}{i+1}\left[Q_{00}^{10}Q_{kl}^{ij} + Q_{10}^{10}Q_{k-1,l}^{ij} + Q_{01}^{10}Q_{k,l-1}^{ij}\right] - \frac{i}{i+1}Q_{kl}^{i-1,j}$$

for $j = 0, 1$ and $i = 1, 2, \ldots, p-j$. This leaves only the four sets of coefficients with $i, j = 0, 1$ to be evaluated, and three are easy to compute directly from the definition:

$$\begin{aligned}
Q_{00}^{00} &= 1 \\
Q_{00}^{10} &= \frac{1}{2x_h}(z_0 + \bar{z}_0 - 2x_m), \qquad Q_{10}^{10} = \frac{-1}{2x_h}, \qquad Q_{01}^{10} = \frac{-1}{2x_h} \\
Q_{00}^{01} &= \frac{1}{2\iota y_h}(z_0 - \bar{z}_0 - 2\iota y_m), \qquad Q_{10}^{01} = \frac{-1}{2\iota y_h}, \qquad Q_{01}^{01} = \frac{1}{2\iota y_h}
\end{aligned}$$

The fourth set can be calculated most easily by multiplying:

$$\begin{aligned}
P_1(x)P_1(y) &= \frac{1}{x_h y_h}(x - x_m)(y - y_m) \\
&= P_1(x)P_0(y)P_0(x)P_1(y) \\
&= (Q_{00}^{10} + Q_{10}^{10}(z_0 - z) + Q_{01}^{10}(\bar{z}_0 - \bar{z}))(Q_{00}^{01} + Q_{10}^{01}(z_0 - z) + Q_{01}^{01}(\bar{z}_0 - \bar{z}))
\end{aligned}$$

implies

$$\begin{aligned}
Q_{00}^{11} &= Q_{00}^{10}Q_{00}^{01} \\
Q_{10}^{11} &= Q_{00}^{10}Q_{10}^{01} + Q_{10}^{10}Q_{00}^{01} \\
Q_{01}^{11} &= Q_{00}^{10}Q_{01}^{01} + Q_{01}^{10}Q_{00}^{01} \\
Q_{20}^{11} &= Q_{10}^{10}Q_{10}^{01} \\
Q_{11}^{11} &= Q_{10}^{10}Q_{01}^{01} + Q_{01}^{10}Q_{10}^{01} \\
Q_{02}^{11} &= Q_{01}^{10}Q_{01}^{01}.
\end{aligned}$$

The recurrence pattern is shown in the following table:

$$\begin{array}{ccccccc} Q^{00} & Q^{01} & \rightarrow & Q^{02} & \rightarrow & \ldots & Q^{0p} \\ & & \searrow & & & & \\ Q^{10} & Q^{11} & \rightarrow & Q^{12} & \rightarrow & \ldots & \\ \downarrow & \downarrow & \searrow & & & & \\ Q^{20} & Q^{21} & \rightarrow & Q^{22} & & & \\ \vdots & \vdots & & \vdots & & & \\ \downarrow & \downarrow & & & & & \\ Q^{p-1,0} & Q^{p-1,1} & & & & & \\ \downarrow & & & & & & \\ Q^{p0} & & & & & & \end{array} \tag{A.2}$$

Several approaches are possible to step two, using either complex or real variable techniques. The complex approach is superficially simpler but encounters difficulty when programming a convenient branch of the complex logarithm. Hence we present a real-variable approach to the integrals

$$\begin{aligned} S_{0l}(C) &= \int_C \frac{1}{z}\bar{z}^l dxdy \\ &= \int_C \frac{\bar{z}}{|z|^2}\bar{z}^l dxdy \\ &= \int_a^b \int_c^d \frac{x-\iota y}{x^2+y^2}(x-\iota y)^l dydx \\ &= \int_c^d \int_a^b \left(\frac{1}{2}\log(x^2+y^2)\right)_x (x-\iota y)^l dxdy \\ &- \iota \int_a^b \int_c^d \left(\frac{1}{2}\log(x^2+y^2)\right)_y (x-\iota y)^l dydx \end{aligned}$$

where subscripts x and y denote partial derivatives. When we integrate by parts, the double integrals cancel, and two one-dimensional integrals remain:

$$\begin{aligned} S_{0l}(C) &= \frac{1}{2}\int_c^d \log(x^2+y^2)(x-\iota y)^l dy \mid_a^b \\ &- \frac{\iota}{2}\int_a^b \log(x^2+y^2)(x-\iota y)^l dx \mid_c^d . \end{aligned} \tag{A.3}$$

Integrating by parts again gives further cancellation, eliminates the logarithms, and yields

$$S_{0l}(C) = \frac{\iota}{l+1}\left[F_{l+1}(x,y) - (-\iota)^{l+1}F_{l+1}(y,-x)\right] \mid_{x=a}^{x=b} \mid_{y=c}^{y=d}$$

where

$$F_{l+1}(x,y) = \int_0^x \frac{x}{x^2+y^2}(x-\iota y)^{l+1}dx.$$

Pulling out one factor of $x-\iota y$ from the power and using that $x^2 = x^2+y^2-y^2$ gives

$$F_{l+1}(x,y) = \int_0^x (x-\iota y)^l dx - \iota y G_{l+1}(x,y)$$

where

$$\begin{aligned} G_{l+1}(x,y) &= \int_0^x \frac{1}{x^2+y^2}(x-\iota y)^{l+1}dx \\ &= F_l(x,y) - \iota y G_l(x,y). \end{aligned}$$

The second line comes from applying the same trick to G_{l+1}. Thus we have a pair of coupled recurrence relations for the F's and the G's, which can easily be solved to yield

$$\begin{aligned} (l+1)S_{0l} &= (2x)^{l+1}X_l - (-2\iota y)^{l+1}Y_l \,|_{x=a}^{x=b}\,|_{y=c}^{y=d} \\ X_l &= \frac{-\iota}{2}\sum_{k=1}^{l}\frac{1}{k}\left(\frac{x-\iota y}{2x}\right)^k - \frac{\iota}{4}\log(x^2+y^2) + \tan^{-1}\left(\frac{y}{x}\right) \\ Y_l &= \frac{-\iota}{2}\sum_{k=1}^{l}\frac{1}{k}\left(\frac{x-\iota y}{-2\iota y}\right)^k - \frac{\iota}{4}\log(x^2+y^2) - \tan^{-1}\left(\frac{y}{x}\right). \end{aligned}$$

Since X_l and Y_l satisfy trivial recurrence relations, this formula is easy to evaluate. Note that $(2x)^{l+1}X_l$ vanishes if $x=0$ and $(-2\iota y)^{l+1}Y_l$ vanishes if $y=0$.

Step three involves integrating polynomials over a rectangle since $k>0$, so can be done several ways. Perhaps the simplest is to employ product Gaussian quadrature of sufficiently high order to be exact on polynomials of the degree involved. We present a slightly more efficient approach, based on the Cauchy integral formula and recurrence relations.

The Cauchy integral formula for a possibly non-analytic function reads

$$\chi_C(z)f(z) = \frac{1}{2\pi\iota}\int_{\partial C}\frac{f(\xi)}{\xi-z} - \int_C \frac{\partial f}{\partial\bar{\xi}}\frac{1}{\xi-z}dA(\xi)$$

where χ_C is the characteristic function of the set C and z is not on the boundary ∂C of C. Put

$$f(z) = \frac{1}{l+1}z^k\bar{z}^{l+1}$$

so that

$$\frac{\partial f}{\partial \bar{z}} = z^k \bar{z}^l.$$

For $z = 0$, we have $f = 0$ and therefore

$$S_{kl}(C) = \frac{1}{2\imath(l+1)} \int_{\partial C} \xi^{k-1} \bar{\xi}^{l+1} d\xi$$

Parametrize each edge of C as a line segment

$$\xi(t) = t\xi_{j+1} + (1-t)\xi_j$$

where ξ_j are the vertices of C ($1 \le j \le 5$, with $\xi_5 = \xi_1$ for convenience). Then

$$S_{kl}(C) = \frac{1}{2\imath(l+1)} \sum_{j=1}^{4} (\xi_{j+1} - \xi_j) T_{kl}(\xi_j, \xi_{j+1})$$

where

$$T_{kl}(a,b) = \int_0^1 (tb + (1-t)a)^{k-1} (t\bar{b} + (1-t)\bar{a})^{l+1} dt$$

for $0 \le k + l \le q$ and complex numbers a and b.

T_{kl} can be evaluated exactly by the binomial theorem or by Gaussian integration since $k \ge 1$. We present a recurrence relation based on integration by parts. First, observe that when $k = 1$ the integral is trivial:

$$T_{1l} = \frac{1}{(l+2)(\bar{b} - \bar{a})} (\bar{b}^{l+2} - \bar{a}^{l+2}).$$

For $k \ge 2$, we integrate by parts to obtain

$$T_{kl} = \frac{1}{(l+2)(\bar{b} - \bar{a})} (b^{k-1}\bar{b}^{l+2} - a^{k-1}\bar{a}^{l+2}) - \frac{(k-1)(b-a)}{(l+2)(\bar{b} - \bar{a})} T_{k-1,l+1}.$$

Thus k can be reduced and l increased until the first exponent goes to 1, whereupon $T_{1,k+l-1}$ is trivial. The recurrence can be solved explicitly, but the resulting formula is best evaluated by recurrence.

B Natural interpolation and contouring

A common difficulty in vortex methods is that the vorticity is known only at scattered data points, so some form of interpolation must be used to evaluate

the vorticity at other points. One advantage of the approach of this paper is the natural interpolation technique provided by the tree structure. Suppose we have vortices z_j in a cell C and we want to know the vorticity at a point z in C. We approximate $\omega(z)$ by a weighted sum

$$\omega(z) \approx \sum_{z_j \in C} \Omega_j(z)\omega(z_j),$$

where the interpolation weights $\Omega_j(z)$ form the least 2-norm solution of the underdetermined linear system

$$\sum_{z_j \in C} \Omega_j(z)P_k(x_j)P_l(y_j) = P_k(x)P_l(y), \qquad 0 \le k+l \le q-1.$$

This gives an qth order interpolation formula on each cell, with reasonably small weights if there are enough interpolation points z_j in C.

We found this technique useful in contouring the vorticity produced by our method. To contour the vorticity, we first interpolated ω to a sufficiently fine equidistant grid on the computational domain, then found the level sets of the linear interpolant to the grid values. This produces continuous contour lines. Figure 7 shows the points, the cells, and five contour levels produced when this technique is applied to the function

$$\omega(x,y) = \cos(kx)\cos(ky) + \epsilon\sin(kx)\sin(ky), \quad k = 11, \quad \epsilon = 1/10.$$

(The points and cells are omitted from the last picture for clarity.) We generated $N = 129$, 515, 2051 and 8197 pseudorandom uniformly distributed points in $[0,1]^2$, interpolated them to an equidistant grid with $M = 10$, 20, 40 and 80 points per side with fourth-order accuracy, and contoured the resulting values. With 8197 points on a 80 by 80 grid, for example, we obtained three-digit accuracy at each grid point, and $|\omega(z)|$ was less than 0.5×10^{-2} at each endpoint of the 2018 segments obtained. The 10 by 10 grid, of course, cannot resolve this function, but the 20 by 20 grid does well.

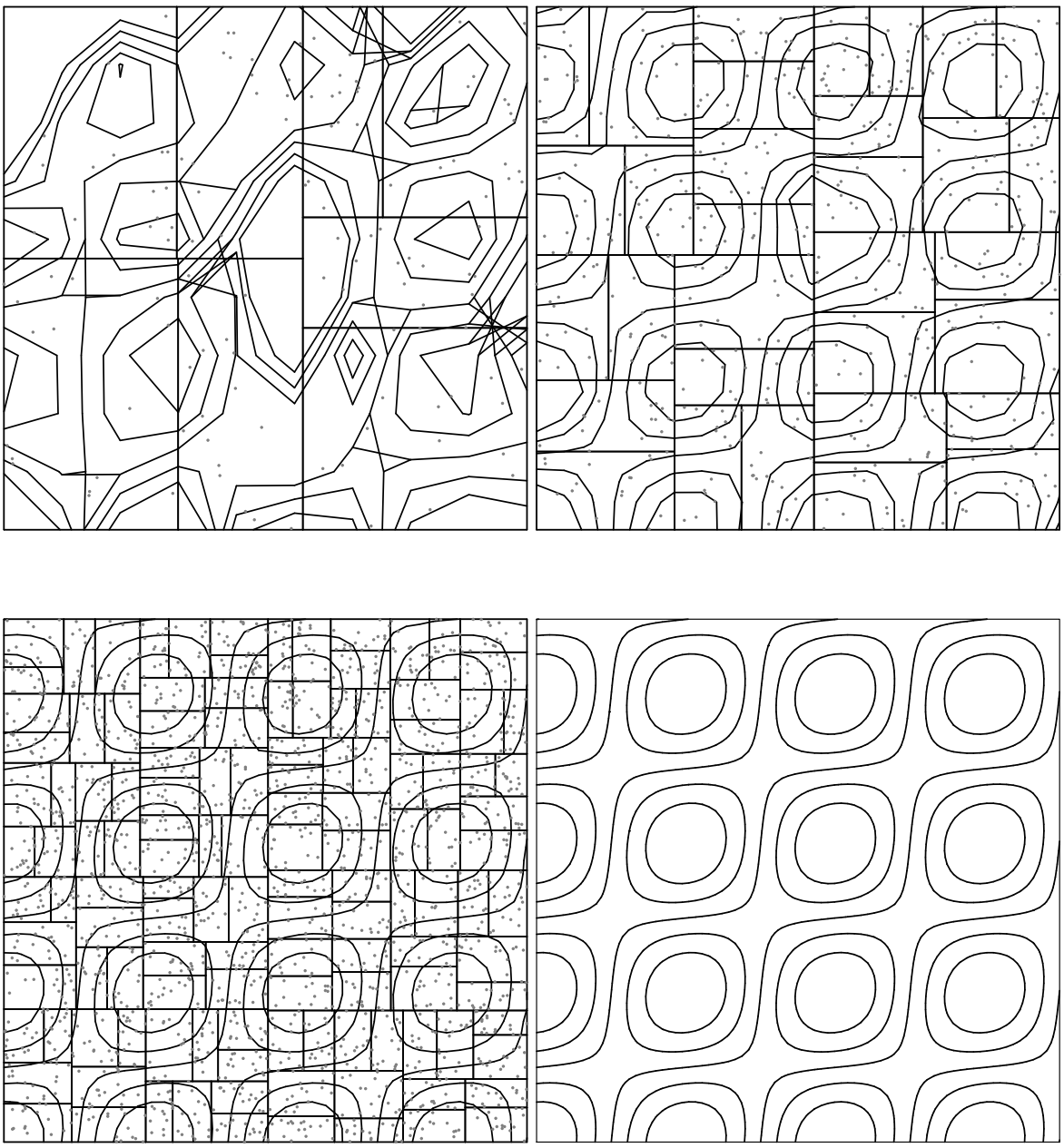

Figure 7: Contour lines produced by fourth-order scattered data interpolation, for random points on $[0,1]^2$.

References

[1] C. Anderson and C. Greengard. On vortex methods. *SIAM J. Math. Anal.*, 22:413–440, 1985.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[3] J. T. Beale. On the accuracy of vortex methods at large times. In B. Engquist, M. Luskin, and A. Majda, editors, *Computational fluid dynamics and reacting gas flow*, volume 12 of *IMA volumes in mathematics and applications*. Springer-Verlag, 1988.

[4] J. T. Beale and A. Majda. Vortex methods II: high order accuracy in two and three dimensions. *Math. Comp.*, 39:29–52, 1982.

[5] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole method for particle simulations. *SIAM J. Sci. Stat. Comput.*, 9:669–686, 1988.

[6] Y. Choi, J. A. C. Humphrey, and F. S. Sherman. Random vortex simulation of transient wall-driven flow in a rectangular enclosure. *J. Comput. Phys.*, 75:359–383, 1988.

[7] A. J. Chorin. *Computational Fluid Mechanics: Selected Papers*. Academic Press, 1989.

[8] G. H. Cottet. A new approach for the analysis of vortex methods. *Ann. Inst. H. Poincaré, Analyse non Lineaire*, 5:227–285, 1988.

[9] P. J. Davis and P. Rabinowitz. *Methods of Numerical Integration*. Computer science and applied mathematics. Academic Press, second edition, 1984.

[10] T. O. Espelid and A. Genz, editors. *Numerical integration : recent developments, software, and applications*. Kluwer Academic, Dordrecht; Boston, 1992.

[11] J. Goodman, T.Y. Hou, and J. Lowengrub. Convergence of the point vortex method for the 2-d Euler equations. *Comm. Pure Appl. Math.*, 43:415–430, 1990.

[12] O. H. Hald. Convergence of vortex methods for Euler's equations III. *SIAM J. Numer. Anal.*, 24:538–582, 1987.

[13] A. Leonard. Computing three dimensional flows with vortex elements. *Ann Rev. Fluid Mech.*, 17:523–559, 1985.

[14] H. O. Nordmark. High-order vortex methods with regridding. *J. Comput. Phys.*, 97:366, 1991.

[15] M. Perlman. On the accuracy of vortex methods. *J. Comput. Phys.*, 59:200–223, 1985.

[16] L. Rosenhead. The formation of vortices from a surface of discontinuity. *Proc. R. Soc. Lon. A.*, 134:170–192, 1931.

[17] G. Russo and J. Strain. Fast triangulated vortex methods for the 2-D Euler equations. *J. Comput. Phys.*, 111:291–323, 1994.

[18] H. Samet. *The design and analysis of spatial data structures.* Addison-Wesley, Reading, Massachusetts, 1990.

[19] J. Strain. Locally-corrected multidimensional quadrature rules for singular functions. *SIAM J. Sci. Comput.*, 16:1–26, 1995.

AMS Subject Classifications: 76M10, 76M25, 65M50, 65M60, 65Y25, 65D32, 65D05, 65D30, 65R20

Key words and phrases: quadrature, vortex methods, Euler equations, Legendre polynomials, least-squares problems, quadtrees, data structures, free-Lagrangian methods, adaptive methods, interpolation, product integration.

E-mail address: strain@math.berkeley.edu.