Journal of Computational Physics **162,** 1–32 (2000)
doi:10.1006/jcph.2000.6510, available online at http://www.idealibrary.com on IDEAL®

A New Volume of Fluid Advection Algorithm: The Stream Scheme

Dalton J. E. Harvie and David F. Fletcher

Department of Chemical Engineering, University of Sydney, New South Wales 2006, Australia
E-mail: davidf@chem.eng.usyd.edu.au

Received April 13, 1999; revised March 15, 2000

A new VOF advection algorithm is presented, termed the Stream scheme. The algorithm uses a linear piecewise method for free surface reconstruction, coupled to a unique fully multidimensional method of cell boundary flux integration. The performance of the new algorithm is compared against other VOF advection algorithms using a variety of standard advection tests. In general, the performance of the Stream scheme appears superior to existing multidimensional and dimensionally split advection algorithms.

1. INTRODUCTION

The Volume of Fluid (VOF) method is a convenient and powerful tool for modelling fluid flows which contain a free surface [10]. Under the VOF method, fluid location is recorded using a volume of fluid function. In a single fluid calculation, this function is defined as unity within fluid regions and zero elsewhere. In numerical fluid simulations, where the VOF function is averaged over each computational cell, the function becomes one in cells containing only fluid, zero in cells containing no fluid, and takes a value between these limits in cells which contain a free surface.

The VOF method is capable of modeling flows with complex free surface geometries, including flows where fluid volumes separate and combine, yet it is remarkably economical in computational terms, requiring only one mesh-sized array for storing the VOF function and an algorithm to advect the function during each computational time step. The method used to advect the VOF function is the subject of this work.

Excellent reviews of past and present VOF advection methods have been given by Rudman [17], Rider and Kothe [16], and Scardovelli and Zaleski [18], so only a brief overview of some of the methods available will be given here.

As a Lagrangian invariant of the fluid, the VOF function, F, satisfies [9]

$$\frac{\partial F}{\partial t} + (\boldsymbol{V} \cdot \boldsymbol{\nabla}) F = 0. \tag{1}$$

0021-9991/00 $35.00

Equation (1) describes the transport of a scalar quantity with the fluid, where the quantity varies continuously from one spatial point to the next. In reality however, the VOF function does not vary continuously from one point to the next, but rather experiences a discrete change over the infinitesimal dimension of each free surface interface. Consequently, special methods must be used to difference equation (1) so that diffusion of free surface interfaces does not result [17].

One such method uses the Flux-Corrected Transport (FCT) Algorithm developed by Zalesak [21]. The FCT algorithm was developed as a general method for advecting any scalar quantity, but it was applied to the process of VOF advection by Rudman [17]. Under the FCT method, Eq. (1) is differenced using a combination of first-order upwind and downwind difference schemes. The dependence on each scheme is chosen so that the advected solution contains no extrema that were not present in the previous time step solution. Rudman [17] demonstrated that the FCT–VOF advection method is not as accurate as modern piecewise linear advection methods.

The majority of VOF advection methods are not derived from a direct difference formulation of Eq. (1) but are instead developed using a two-stage process. First, free surface interfaces are "reconstructed" from the VOF data, so that a geometrical profile is found which approximates the actual free surface location. Changes in VOF values are then calculated by integrating fluid fluxes over cell boundaries, using the geometrical profile to indicate the location of fluid regions. The different advection algorithms based on this two-stage process can be loosely classified according to the technique used to reconstruct the free surfaces in each cell and by the method used to perform the boundary flux integrations [16].

VOF advection methods that represent free surface interfaces as lines directed parallel to one of the grid coordinates are known as piecewise constant schemes. The SLIC (Simple Line Interface Calculation) of Noh and Woodward [11] and the SURFER code of Lafaurie *et al.* [8] are examples of a piecewise constant scheme.

A variation on the piecewise constant technique is the method used in the SOLA-VOF code of Nichols *et al.* [10]. Under the Hirt–Nichols scheme, free surface interfaces are orientated in directions parallel to grid coordinates but are also allowed the greater freedom of a stair-shaped profile if local VOF distribution conditions permit. Similar algorithms include those developed by Chorin [3] and Barr and Ashurst [2].

The alternative to representing free surface interfaces as lines parallel to one of the grid coordinates is to orientate free surface interfaces in a direction perpendicular to the locally evaluated VOF gradient. Free surface interfaces within each cell can then acquire any orientation, and the geometrical profile of the fluid can more closely represent the actual fluid geometry. Such schemes are known as piecewise linear schemes and include those developed by Debar [4], Youngs [20], Ashgriz and Poo [1], Puckett *et al.* [15], Rider and Kothe [16], and Harvie and Fletcher [6]. These schemes tend to be more complex than their piecewise constant cousins, but they have been shown to be significantly more accurate [7, 12, 17].

The method of integration used to determine cell boundary fluxes is also used to classify VOF advection techniques. Under operator or dimensionally split schemes, boundary fluxes are calculated independently in each coordinate direction, often with some type of limiter employed to reduce possible undershoots or overshoots occurring in cell VOF values. Free surfaces are usually reconstructed between integrations in each of the coordinate directions under this technique. The Youngs algorithms are examples of operator split schemes [19, 20].

Multidimensional schemes can be more accurate and efficient in calculating cell boundary fluxes than operator split schemes [16]. Under a multidimensional scheme, cell boundary fluxes are calculated with a dependence between fluxes calculated in each of the coordinate directions. Example multidimensional schemes include those developed by Puckett *et al.* [15], Rider and Kothe [16], and Harvie and Fletcher [6].

The Stream scheme, as developed in this study, is a piecewise linear scheme with cell boundary fluxes integrated using a fully multidimensional technique. While the interface reconstruction technique used by the Stream scheme is similar to methods used by other VOF advection schemes, the multidimensional integration technique used by the Stream scheme is quite new and unique.

In this paper, we detail and analyze the new advection scheme. This is accomplished in three sections: The first section defines the conceptual and theoretical basis for the scheme and outlines the solution procedure used to implement the algorithm. A comparison of the performance of the Stream scheme against existing advection algorithms is then given, using for comparison several standard VOF advection tests. Finally, an analysis of the errors generated by the Stream scheme when advecting a fluid form, and associated convergence rates, is given.

2. THE STREAM VOF ADVECTION ALGORITHM

2.1. The Basic Method

The idea behind the Stream algorithm is simple and can be summarized as follows:

1. Fluid interfaces are reconstructed in each cell using a piecewise linear interface method.
2. A semicontinuous velocity field is defined throughout the computational region, based on the staggered cell boundary velocities.
3. Donating regions for each cell boundary are defined by integrating back in time for the duration of the computational time step along fluid streamlines passing through the examined boundary.
4. Boundary fluxes are calculated as the intersection between each donating region and all fluid locations.

The primary tasks involved in implementing the Stream algorithm are defining a suitable velocity field, reconstructing the fluid free surfaces, and integrating along streamlines to determine fluid volume fluxes occurring over each boundary. It is these three topics which we now examine.

2.2. Defining the Velocity Field

The velocity at any point in a given cell i, j is defined as

$$\mathbf{V}(x, y) = \{\chi_b x + \chi_y\}\mathbf{i} + \{-\chi_b y - \chi_x\}\mathbf{j}, \tag{2}$$

where

$$\chi_b = \frac{u_{i+\frac{1}{2}} - u_{i-\frac{1}{2}}}{x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}} = -\frac{v_{j+\frac{1}{2}} - v_{j-\frac{1}{2}}}{y_{j+\frac{1}{2}} - y_{j-\frac{1}{2}}}, \tag{3}$$

$$\chi_x = -\chi_b y_{j-\frac{1}{2}} - v_{j-\frac{1}{2}} \tag{4}$$

and

$$\chi_y = -\chi_b x_{i-\frac{1}{2}} + u_{i-\frac{1}{2}}. \tag{5}$$

Integer subscripts in these equations refer to cell-centered quantities, while integer values plus or minus a half refer to quantities located at cell upper or lower boundaries, respectively. The velocity in the x coordinate direction is u, and that in the y direction is v. Note that the equality in Eq. (3) follows from the continuity equation applied to the examined cell, namely

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \frac{u_{i+\frac{1}{2}} - u_{i-\frac{1}{2}}}{x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}} + \frac{v_{j+\frac{1}{2}} - v_{j-\frac{1}{2}}}{y_{j+\frac{1}{2}} - y_{j-\frac{1}{2}}} = 0. \tag{6}$$

The velocity field given in Eq. (2) can also be expressed in terms of the stream function,

$$\Psi = \chi_b xy + \chi_x x + \chi_y y, \tag{7}$$

where the standard stream function has been defined via

$$u = \frac{\partial \Psi}{\partial y} \quad \text{and} \quad v = -\frac{\partial \Psi}{\partial x}, \tag{8}$$

and we have assumed the boundary condition for integration, $\Psi(0, 0) = 0$.

Equation (2) specifies a simple velocity field which varies linearly between cell boundaries and satisfies the continuity equation everywhere. The normal components of the velocity field at cell boundaries are continuous between cells, but the tangential components are not. The velocity field is not continuous at cell vertex points. The velocity field discontinuities would be problematic if we were to integrate exactly along streamlines from cell vertices, but the integration method we present later avoids this inconvenience.

The stream function defined by Eq. (7) is continuous throughout the computational domain. Note that fluid fluxes over cell boundaries are constant along the length of each boundary, and consequently streamlines intersecting each boundary, which are separated by equal volume fluxes, are equally spaced.

Examples of various in-cell velocity fields, represented as fluid streamlines, are shown in Fig. 1. The cells shown in the figure have the nondimensional dimensions of 1×1, and the corresponding velocity field constants are shown alongside each streamline trace. Cases (D) and (E) in Fig. 1 are the only cases with $\chi_b \neq 0$—the streamlines in these two examples are curved.

2.3. Free Surface Interface Reconstruction

A piecewise linear interface method is used by the Stream scheme for locating fluid regions. The method involves two steps for each interface reconstruction—determining the gradient of the interface within each cell, and positioning the interface within each cell to equate the calculated cell VOF volume to the volume contained between the free surface and cell boundaries.

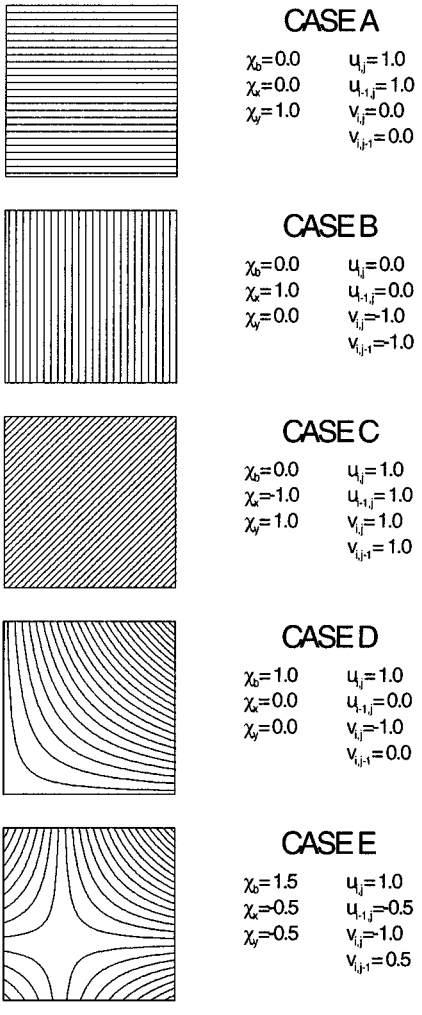

FIG. 1. Example in cell velocity fields, represented by fluid streamlines. In each case, streamlines having stream functions equally spaced in the range $-1 \leq \Psi \leq 1$ are shown.

2.3.1. *Free Surface Gradient Calculation*

In this study we have used two alternative methods for calculating the free surface gradient. The first method, of Youngs [20], is taken from the RIPPLE code of Kothe *et al.* [7]. It is referred to as the Youngs method. The second method used in this study is an error minimization method. The error minimization concept and the specific method used here are both from Puckett [14]. This second method is referred to as the Puckett method.

The implementation of the two interface gradient calculation methods is now presented. A comparative analysis of the accuracy of the methods is included in Section 3.

The Youngs method. Under the Youngs method, interface normals are first calculated at each of the four vertices of each cell as the gradient of the VOF function,

$$\mathbf{n} = \nabla F. \tag{9}$$

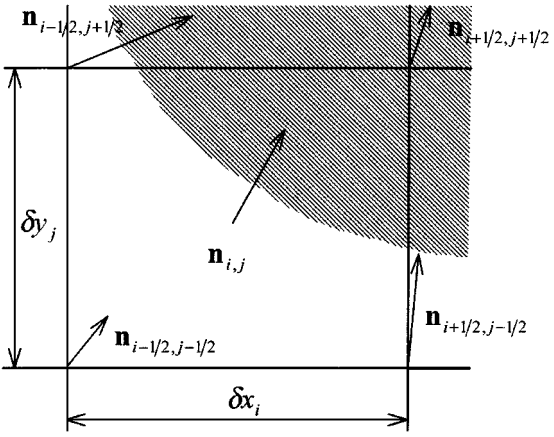

FIG. 2. VOF normal locations used by the Youngs method of interface gradient calculation [7]. The shaded area is the actual fluid location.

Following [7], and referring to Fig. 2 for notation, Eq. (9) is differenced using

$$\mathbf{n}_{i+1/2,\,j+1/2} = \left\{ \frac{(F_{i+1,j+1} - F_{i,j+1})\delta y_j + (F_{i+1,j} - F_{i,j})\delta y_{j+1}}{(\delta y_j + \delta y_{j+1})\delta x_{i+1/2}} \right\} \mathbf{i} + \left\{ \frac{(F_{i+1,j+1} - F_{i+1,j})\delta x_i + (F_{i,j+1} - F_{i,j})\delta x_{i+1}}{(\delta x_i + \delta x_{i+1})\delta y_{j+1/2}} \right\} \mathbf{j}. \tag{10}$$

Cell-centered normals are calculated by averaging these vertex normals, so that the centered normals are effectively calculated using VOF data from a 3×3 mesh,

$$\mathbf{n}_{i,j} = \frac{1}{4}\left(\mathbf{n}_{i+1/2,\,j+1/2} + \mathbf{n}_{i+1/2,\,j-1/2} + \mathbf{n}_{i-1/2,\,j-1/2} + \mathbf{n}_{i-1/2,\,j+1/2}\right). \tag{11}$$

The orientation of the fluid interface is set perpendicular to this cell-centered normal.

The Puckett method. Under the Puckett method, each interface orientation within each cell has associated with it an error function. When this error function is minimized, we find the optimal free surface orientation. The error function for the Puckett method is defined as follows;

1. Given a gradient, the interface is reconstructed such that the volume of fluid contained between the interface and cell boundaries is equal to the volume of fluid within the examined cell. The method used here follows that described in Section 2.3.2.

2. The fluid interface is continued beyond the boundaries of the examined cell so that it traverses a total of $3 \times 3 = 9$ cells, with the examined cell at the center.

3. Volume of fluid functions, F^*, are calculated for each of the surrounding 8 cells based on the extended reconstructed interface.

4. Finally, the error associated with the reconstructed interface is calculated. The error function is defined as

$$E_{i,j} = \sum_{\substack{n=i-1,i+1 \\ m=j-1,j+1}} (F_{n,m} - F^*_{n,m})^2, \tag{12}$$

where $F_{n,m}$ is the actual VOF function for cell n, m, and $F^*_{n,m}$ is the VOF function for cell n, m based on the extended reconstructed interface, as described above.

Under the Stream scheme, the Puckett method is implemented for each examined cell using a three-step process. First, an estimation of the interface gradient is calculated using the Youngs method. The interface is then rotated from this first approximation, using discrete, variably sized steps, until the local minimum of the gradient error function, $E_{i,j}$, is bounded. A standard Golden Section search routine, taken from Press *et al.* [13], is then used to determine the final, minimum error, interface gradient. The tolerance used in this work for determining the minimum error orientation was 1×10^{-6} rad.

Experimentation with the error minimization technique showed that finding a minimum to the error function in the locality of the Youngs orientation estimate, rather than attempting to find an absolute minimum to the error function over all possible interface orientations, tended to produce greater free surface reconstruction accuracy.

2.3.2. *Free Surface Interface Positioning*

Once the interface gradient has been determined, positioning of the interface within each cell is accomplished by equating the volume of fluid contained under the interface to the volume of fluid contained within the cell. Under the Stream scheme each interface location is described by two points—a left point and a right point. Both points lie on a cell boundary and both points lie on the interface line. The relationship between the points determines the position of the fluid within the cell—when looking along the interface line from the left point toward the right point, fluid is located to the left of the interface.

Interface positioning is accomplished in the code using first a series of logic steps to determine which boundaries of each given cell the interface intersects with, followed by a number of analytical expressions which equate the area under the interface to the area contained within the cell. For example, in the cell shown in Fig. 3, fluid is located to the left of the interface line, and the left and right interface points would be calculated using

$$x_{\text{left}} = x_{i-1} + F_{i,j}\,\delta x + \frac{1}{2}\,\delta y \frac{n_y}{n_x}, \tag{13}$$

$$y_{\text{left}} = y_{j-1}, \tag{14}$$

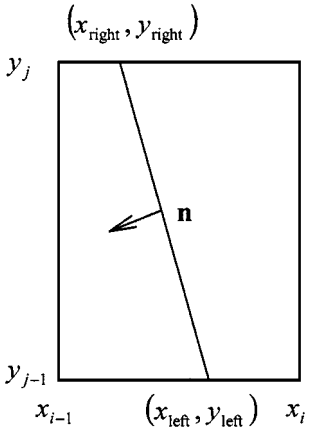

FIG. 3. Example cell showing location of left and right points.

$$x_{\text{right}} = x_{i-1} + F_{i,j}\,\delta x - \frac{1}{2}\,\delta y\,\frac{n_y}{n_x}, \tag{15}$$

and

$$y_{\text{right}} = y_j. \tag{16}$$

The method used to calculate cell boundary fluxes is now examined.

2.4. The Fluid Boundary Flux Calculation

Ideally, we would like to integrate Eq. (2) exactly with respect to time to determine donating regions associated with each boundary and then determine the intersection between these regions and the fluid regions to calculate fluid boundary fluxes. In practice however, such a procedure would be extremely computationally complex for two main reasons;

1. A fluid particle that flows through a particular boundary during a discrete time step and exists on a particular streamline may pass through many cells en route to that boundary. It is the starting positions of such particles that define a donating region for a given cell boundary. The difficulty is that integrating along the path of a fluid particle, or streamline, becomes more and more complex the more cells the particle passes through, as each cell has a different velocity field. As a result, defining donating regions for boundaries which are fluxing fluid from many cells becomes impractically complex. Determining the intersection between these irregular donating regions and the reconstructed fluid regions would add further complications.
2. As previously discussed, the velocity field defined by Eq. (2) is continuous throughout the computational domain, except at cell vertices. Thus, trying to integrate away from vertex points to define the fluid donating regions may not be computationally possible.

It is for these reasons that an approximate method of integration has been developed to determine fluid boundary fluxes.

2.4.1. *Approximate Integration Method*

The idea behind this approximate integration method is simple. Each boundary flux is split into a discrete number of streamtubes, each "tube" representing an equal flux of total fluid and void volume. Fluid streamlines, determined using Eq. (7), define the upper and lower boundaries of each tube. The length of each tube is determined by the length of the computational time step. The number of tubes that each boundary is split into, n_{stream}, is determined by the user—the more tubes the greater the accuracy of the integration technique.

To illustrate this point, Fig. 4 shows an example where the velocity at the right boundary of cell i, j is positive, and fluid is fluxing through this boundary from cells i, j and $i, j+1$ during the solution time step. This right boundary of cell i, j is referred to as the "final" boundary, as it is the last boundary a fluid particle would pass through before entering cell $i+1, j$.

The volume of fluid in each streamtube which is fluxed through a particular final boundary during a time step δt is approximately

$$V_{\text{fluid in tube}} \approx \int_0^L w(l)\, f(l)\, dl, \tag{17}$$

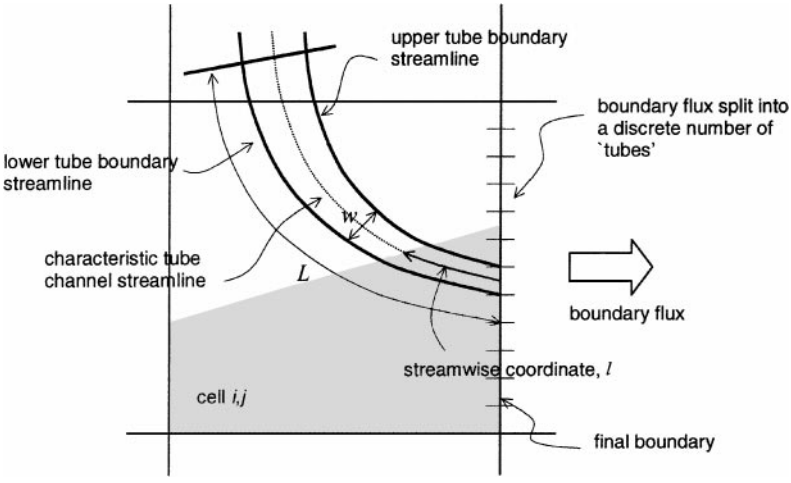

FIG. 4. Example tube used to calculate the fluid flux over the right boundary of cell i, j.

where a cell of unit depth has been assumed. In Eq. (17) $f(l)$ is the VOF function evaluated along the central characteristic streamline of the tube, w is the nonconstant width of the tube measured normal to the central tube streamline, l is a streamwise coordinate along the length of the tube directed upstream from the final boundary, and L is the streamwise length of the tube.

To determine the streamwise width of the tube, we note that the total volume of a small section of the tube, located at l and of length δl, is approximately

$$\delta V = w(l)\,\delta l. \tag{18}$$

Also, we note that as no fluid may cross the upper and lower boundaries of the tube, the volume contained within this small section is equal to the volume flowrate through the final boundary of the tube multiplied by the time taken for a fluid particle to pass over the small section of the tube. Thus,

$$\delta V = \frac{u_{i,j} \times \delta y_j}{n_{\text{stream}}}[t(l) - t(l + \delta l)], \tag{19}$$

where t represents the time taken for a fluid particle to flow from the beginning of the tube to position l. Combining Eqs. (18) and (19), rearranging, and taking the limit as $\delta l \to 0$ yields for the width of the tube,

$$w(l) = -\frac{u_{i,j} \times \delta y_j}{n_{\text{stream}}} \lim_{\delta l \to 0} \frac{1}{\delta l}[t(l + \delta l) - t(l)] = -\frac{u_{i,j} \times \delta y_j}{n_{\text{stream}}} \frac{dt}{dl}. \tag{20}$$

Substituting Eq. (20) into Eq. (17) gives

$$V_{\text{fluid in tube}} = -\frac{u_{i,j}\delta y_j}{n_{\text{stream}}} \int_0^L f(l) \frac{dt}{dl}\, dl = \frac{u_{i,j}\delta y_j}{n_{\text{stream}}} \int_0^{\delta t} f(t)\, dt, \tag{21}$$

or

$$V_{\text{fluid in tube}} = \frac{u_{i,j}\delta y_j}{n_{\text{stream}}} t_{\text{total fluid}}, \tag{22}$$

where $t_{\text{total fluid}}$ is defined as the total time a fluid particle would spend in fluid regions, when moving for time δt along the central tube streamline toward the final cell boundary over a stationary fluid geometry.

Once individual tube fluid fluxes have been determined, VOF values are incremented using

$$F_A = F_A + \frac{V_{\text{fluid in tube}}}{(\delta x \times \delta y)_A} \tag{23}$$

and

$$F_D = F_D - \frac{V_{\text{fluid in tube}}}{(\delta x \times \delta y)_D} \tag{24}$$

for all streamtubes along each boundary. In Eqs. (23) and (24) the subscript A identifies the accepting cell, which is defined as the cell immediately downstream of the final boundary, while the subscript D identifies the donating cell, which is the cell immediately upstream of the final boundary. Note that although fluid volumes may be calculated using fluid geometries and velocity fields from other cells, the calculated flux refers to the amount of fluid passing through each examined cell boundary during each time step. Thus, during a boundary flux calculation only the VOF values for the accepting and donating cells are changed.

2.4.2. *Fluid Particle Trajectory Calculation*

In calculating $t_{\text{total fluid}}$ for each tube, we recognize that a streamtube may pass through more than one cell en route to the final cell boundary. To simplify computational calculations, we divide the total tube length into sections contained within individual computational cells. Thus, we can specify

$$t_{\text{total fluid}} = \sum_{\substack{\text{all cells along central} \\ \text{tube streamline}}} t_{\text{fluid}}, \tag{25}$$

where t_{fluid} is the time a fluid particle would spend in the fluid region of an individual cell, when moving along the central tube streamline over a stationary fluid geometry. We also define t_{remain} as the length of time elapsed when a fluid particle travels between the start of a central tube streamline and a particular cell. For example, at the final boundary of a tube, $t_{\text{remain}} = \delta t$.

Using these definitions, the integration problem now becomes one of travelling backward along each central tube streamline, from each examined final boundary, calculating t_{fluid} for each cell passed through en route. When we reach the start of the streamtube, $t_{\text{remain}} \to 0$, indicating that the tube integration is complete.

Details of the numerical procedure used to implement these calculations can be found in [5]. The procedure itself is straightforward; however, it relies on a few mathematical utilities concerning the velocity field and interface positioning which we will now examine. In the following, Ψ_c is defined as the characteristic stream value for the central tube streamline being calculated. It is evaluated using the stream equation (7) at the center of the streamtube under examination.

Length of tube remaining. It is necessary to calculate the starting point for a tube, in order to ascertain whether the tube finishes within a particular cell or not. For the example shown in Fig. 4, where fluid is flowing in the positive direction over the right vertical boundary of cell i, j, the starting x coordinate of the tube, x_{end}, is given by

$$x_{\text{end}} = \begin{cases} x_{i+\frac{1}{2}} - \chi_y t_{\text{remain}} & \text{if } \chi_b = 0 \\ \frac{1}{\chi_b}\left[\left(\chi_b x_{i+\frac{1}{2}} + \chi_y\right)\exp(-\chi_b t_{\text{remain}}) - \chi_y\right] & \text{if } \chi_b \neq 0 \end{cases}. \tag{26}$$

Intersection between cell boundaries and central tube streamline. We also need to know whether a streamtube continues beyond the boundaries of an individual cell. Again using the example of Fig. 4, the x coordinate marking the intersection between the central streamline and the upper boundary of cell i, j is given by

$$x_{\text{top}} = \frac{\Psi_c - \chi_x y_{j+\frac{1}{2}}}{\chi_b y_{j+\frac{1}{2}} + \chi_x} \tag{27}$$

if $\chi_b y_{j+\frac{1}{2}} + \chi_x \neq 0$, otherwise the intersection point is not defined.

Intersection between a fluid interface and central tube streamline. We need to determine the intersection points between the free surface interfaces and central tube streamlines in each cell that the streamtube traverses. Free surface interface lines are defined using the left and right point method, as discussed in Section 2.3.2. In calculating the streamline and free surface intersection points, we cast the interface curves into a different form, namely

$$\lambda_x x + \lambda_y y = \lambda_c. \tag{28}$$

The constants appearing in Eq. (28) are evaluated in each cell using

$$\lambda_x = \frac{y_{\text{left}} - y_{\text{right}}}{\sqrt{(x_{\text{left}} - x_{\text{right}})^2 + (y_{\text{left}} - y_{\text{right}})^2}}, \tag{29}$$

$$\lambda_y = \frac{x_{\text{right}} - x_{\text{left}}}{\sqrt{(x_{\text{left}} - x_{\text{right}})^2 + (y_{\text{left}} - y_{\text{right}})^2}} \tag{30}$$

and

$$\lambda_c = \lambda_x x_{\text{right}} + \lambda_y y_{\text{right}}. \tag{31}$$

Finding the fluid interface and central streamline intersection points now becomes a question of solving the stream equation (7) and interface equation (28) simultaneously. The form of the solution to this problem depends on the form of the two curves. The different alternatives are listed below for the case of flux through a vertical cell boundary:

1. Horizontal interface, $\lambda_x = 0$. If $\chi_b \lambda_c + \chi_x \lambda_y \neq 0$ then

$$x_{\text{intersection}} = \frac{\Psi_c \lambda_y - \chi_x \lambda_c}{\chi_b \lambda_c + \chi_x \lambda_y}, \tag{32}$$

otherwise no intersection takes place.

2. Vertical interface, $\lambda_y = 0$.

$$x_{\text{intersection}} = \frac{\lambda_c}{\lambda_x}. \tag{33}$$

3. Streamline is a straight line, $\chi_b = 0$. If $\lambda_x \chi_y - \lambda_y \chi_x \neq 0$ then

$$x_{\text{intersection}} = -\frac{\Psi_c \lambda_y - \chi_y \lambda_c}{\lambda_x \chi_y - \lambda_y \chi_x}, \tag{34}$$

otherwise no intersection takes place.

4. Streamline has curvature and interface finite nonzero gradient. Define

$$b = \frac{\chi_y \lambda_x - \chi_x \lambda_y - \chi_b \lambda_c}{\chi_b \lambda_x} \tag{35}$$

and

$$c = \frac{\Psi_c \lambda_y - \chi_y \lambda_c}{\chi_b \lambda_x}. \tag{36}$$

If the determinate of the solution to these equations, $\det = \sqrt{b^2 - 4c}$, is real, then at least one intersection point occurs. If $\det = 0$, then only one intersection point occurs, namely

$$x_{\text{intersection}} = \frac{-b}{2}. \tag{37}$$

If the determinate is real and nonzero, then two intersection points occur, namely

$$x_{\text{intersection}} = \frac{-b + \sqrt{\det}}{2} \quad \text{and} \quad x_{\text{intersection}} = \frac{-b - \sqrt{\det}}{2}. \tag{38}$$

Similar solutions are used for intersections occurring in cells with horizontal flux boundaries.

Determining whether a region is void or fluid. The integration algorithm requires knowledge of which regions along the central tube streamline are fluid or void. The method used here is taken from Rider *et al.* [16]. Defining a function

$$H = (y_{\text{left}} - y_{\text{right}})(x - x_{\text{left}}) + (x_{\text{right}} - x_{\text{left}})(y - y_{\text{left}}), \tag{39}$$

the point (x, y) is located within a fluid region if $H > 0$.

Time taken to traverse fluid region. Once the fluid region has been defined in terms of intersection points on the characteristic tube streamline, t_{fluid} can be calculated. For example, say points x_1 and x_2 define the extremes of the fluid region within a particular streamline in a particular cell. The length of time a fluid particle takes to cross this region is given by

$$t_{\text{fluid}} = \left| \int_{x_2}^{x_1} \frac{1}{u(x, y)}\, dx \right| = \begin{cases} \left| \frac{1}{\chi_y}(x_1 - x_2) \right| & \text{if } \chi_b = 0 \\ \left| \frac{1}{\chi_b} \ln\left[\frac{\chi_b x_2 + \chi_y}{\chi_b x_1 + \chi_y} \right] \right| & \text{if } \chi_b \neq 0 \end{cases}, \tag{40}$$

where we have assumed the particle exits the cell through a vertical boundary.

2.5. Algorithm Accuracy

2.5.1. *Fluid Volume Conservation*

The accuracy of the Stream VOF advection algorithm in conserving volume is dependent on the accuracy of the integration algorithm used to calculate boundary fluid fluxes. As previously mentioned, the accuracy of the integration algorithm is dependent on the user specified variable, n_{stream}, the number of streamtubes used in calculating each boundary flux. The greater the number of tubes, the greater the accuracy.

Figure 5 shows an example of a "worst-case scenario" for integration accuracy. Here we see that fluid velocity streamlines are aligned with the free surface orientation, and the magnitude of error that we could expect from a single boundary flux calculation could be as high as

$$\text{Error}(F) = O\left(\frac{u\delta t}{2n_{\text{stream}}\delta x}\right) = O\left(\frac{C}{2n_{\text{stream}}}\right), \tag{41}$$

where the notation $O(z)$ specifies "of the order z," and $C = u\delta t/\delta x$ is the Courant number.

As detailed by Eqs. (23) and (24), after each boundary fluid flux is calculated, fluid is added to the accepting cell, and the same amount of fluid is subtracted from the donating cell. As a result, fluid volume is rigorously conserved after the advection step. The difficulty is that integration inaccuracies can cause undershoot and overshoot F values, which when brought back into the zero to one range, can cause net changes in fluid volume.

The solution is to correct F value overshoots and undershoots using a local redistribution algorithm, after all boundary fluxes have been incremented, but before the $0 \leq F \leq 1$ check. The specific redistribution algorithm can be summarized by the schematic of Fig. 6.

2.5.2. *Fluid and Void Wisp Generation*

Integration inaccuracies can also cause "wisp" generation of either fluid in void regions, or alternatively void in fluid regions. Wisps are generated when a free surface interface moves across the computational domain. If, for example, a fluid interface is sweeping across a void region, integration inaccuracies at the interface cause small amounts of void to remain in

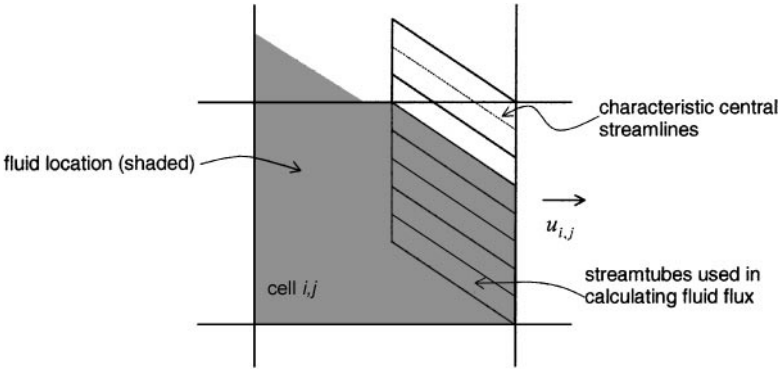

FIG. 5. A worst-case scenario for border fluid flux calculation. The fluid free surface is orientated parallel to the integration tubes, and located close to a characteristic streamline.

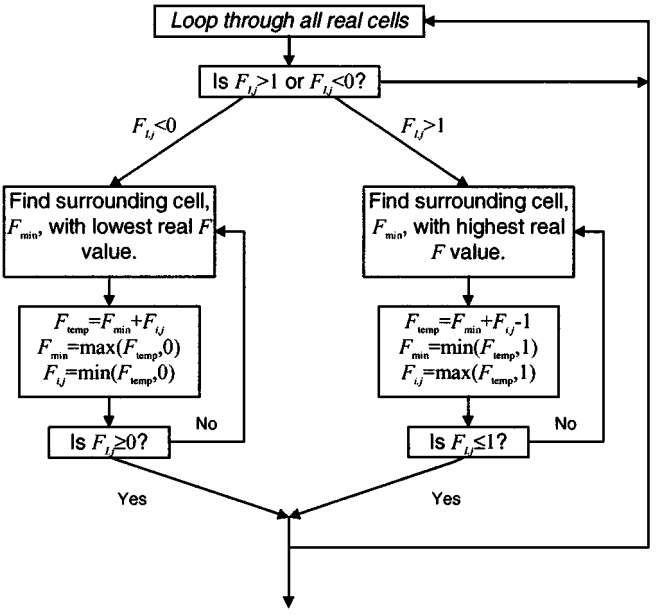

FIG. 6. Schematic showing the algorithm used to locally redistribute nonreal F values.

the cells after the interface has passed, thus creating wisps of void within the fluid region. Similarly, wisps of fluid in void regions can be created when a void interface sweeps over an area of fluid.

Other terms previously used for VOF debris include "flotsam" and "jetsam." "Wisps" was chosen in this study, as it was felt that the term most accurately describes the low total fluid volume nature of the fluid or void trails produced under the Stream algorithm.

Obviously the level of wisp generation depends on the accuracy of the integration scheme, and thus the magnitude of the user set variable n_{stream}. In general, the amount of fluid or void involved in wisps is small, even for small values of n_{stream}, but their presence is detrimental to computational efficiency, as they require the discretized Navier–Stokes equations to be solved in cells that should not contain fluid.

Under the Stream scheme, wisps are eradicated using an algorithm similar to the undershoot and overshoot algorithm detailed above. For the case of a fluid wisp in a void region, a cell is considered to contain a wisp if the cell, and all of its eight neighbors, contains less than a certain proportion of fluid. This critical proportion is specified by the user set variable, F_{wisp}.

Once a fluid wisp is established, fluid is moved from the wisp cell to a neighboring cell which is estimated to be closer to the free surface interface. Exactly which cell is closer to the free surface interface is specified by the direction of the free surface interface normal in the wisp cell. In this case, the free surface interface normal is defined as the maximum gradient of the VOF function, calculated using the methods outlined in Section 2.3.1, in a 3×3 array of cells surrounding the examined wisp cell. The VOF gradient within a wisp cell is determined in this fashion so that even if a wisp cell is separated from the interface region by two cell dimensions, fluid is still moved in the direction of the actual free surface boundary.

For the case of a void wisp in a fluid region, a cell is considered to contain a wisp if the cell and all eight neighbors have VOF values greater than $1 - F_{\text{wisp}}$. As for the fluid wisp

case, once a void wisp is established, fluid is moved from cells closer to the free-surface interface to the examined cell, thus removing the void in fluid wisp.

By destroying a wisp once it is generated, rather than using various criteria to limit fluid boundary fluxes, the Stream method wisp eradication technique has negligible effect on fluid geometry and is capable of removing both wisps of fluid in void as well as wisps of void in fluid.

While the wisp eradication algorithm is successful in removing wisps from computations, its use unfortunately imposes a maximum time step on fluid dynamics computations. In its present form the wisp eradication algorithm can move the contents of a wisp cell one cell dimension closer to the actual free surface interface per time step. It follows that if the actual fluid interface is traveling a greater distance than one-cell dimension per time step, the wisp eradication algorithm is not capable of reuniting wisps with the main fluid bodies and will consequently fail. For this reason the maximum time step which the Stream scheme can operate under without producing fluid wisps corresponds to a Courant number of unity.

Note that if the cell boundary flux integration technique were exact, or alternatively if the wisp eradication algorithm could move wisps with a velocity comparable with actual fluid velocities, this time step limitation would not apply. In practice however most computational codes are limited by other stability criteria to a Courant number of less than or equal to unity, so development of a more complex wisp eradication algorithm is not justified.

Equation (41) gives the maximum expected error per boundary flux calculation. In calculating a suitable level for F_{wisp}, we note that each cell could be involved in up to four boundary flux calculations per time step, corresponding to four cell boundaries. Thus, a suitable level for the critical wisp VOF value may be

$$F_{\text{wisp}} = \frac{2C}{n_{\text{stream}}}, \tag{42}$$

where C is the maximum Courant number within the domain. In practice the level specified by Eq. (42) is overly conservative. In the advection test cases calculated in the following sections, F_{wisp} was set to half the value given above and still was successful in removing all fluid wisps.

3. PERFORMANCE OF THE STREAM SCHEME

In this section the performance of the Stream algorithm is compared against other VOF advection algorithms using a variety of advection tests.

3.1. Rudman Translation Tests

Translation tests provide the most basic measure of VOF advection algorithm performance. To facilitate comparison of the Stream algorithm against other algorithms, the form of the test used here is taken from Rudman [17].

For this test a computational domain of dimensions 4×4 m^2 is composed of 200×200 uniformly sized square cells. Two separate uniform and constant velocity fields, having the components (1, 0) and (2, 1), are used to advect the different fluid forms. Three fluid forms are advected—a hollow square orientated with sides parallel to the coordinate axis, a hollow square orientated at 26.57° to the axis, and a hollow circle. The major dimension of each form is 0.8 m. A Courant number of 0.25 is used in all computations, and each test is

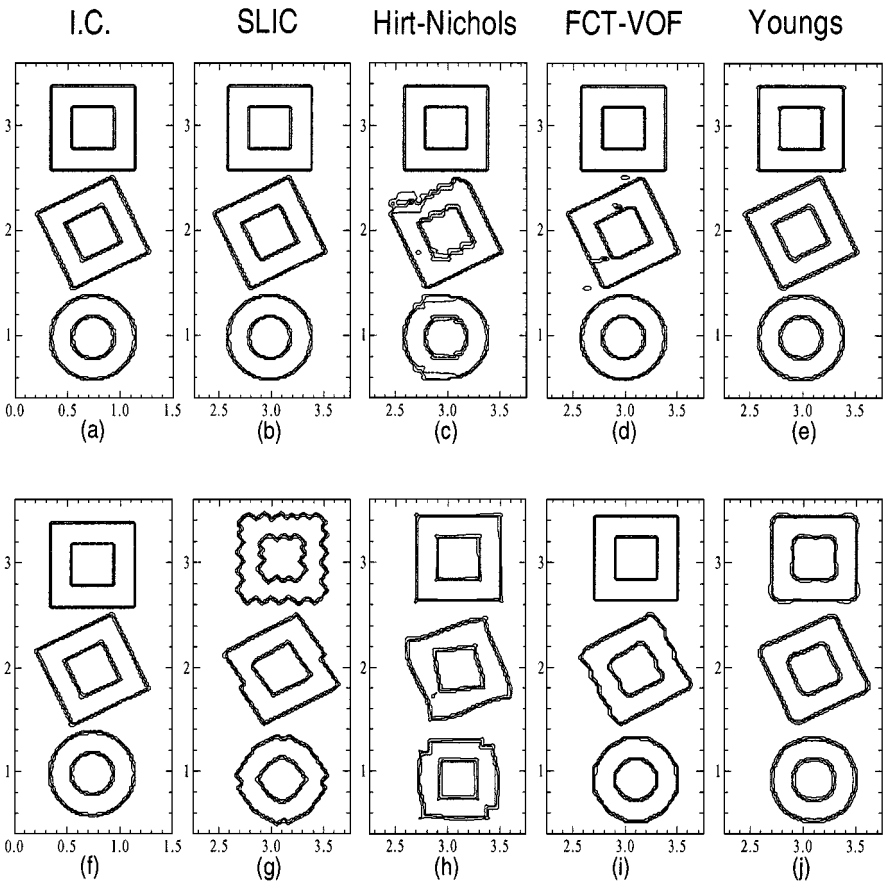

FIG. 7. Advection with unidirectional velocity fields (1, 0) (top) and (2, 1) (bottom). At the left are the initial conditions (I.C.) followed by the results for SLIC, Hirt–Nichols' VOF, FCT–VOF, and Youngs' method. (Figure and caption are reproduced with permission from Rudman [17].)

performed over approximately 500 time steps. Further details of the form of the tests may be found in Rudman [17].

Figure 7, taken from Rudman [17], shows the initial position and final position fluid forms for the above translation tests calculated using a number of VOF advection algorithms. Figure 8 shows the final position fluid forms for the same tests as calculated by the Stream algorithm. In both figures each plot was generated using three VOF contour intervals, 0.025, 0.5, and 0.975, and for compactness the three different fluid forms tested are displayed on the same computational domain. Quantitative errors for each test are shown in Table I. As in [17], these errors were calculated using

$$E = \frac{\sum_{i,j} \left| F^n_{i,j} - F^e_{i,j} \right|}{\sum_{i,j} F^0_{i,j}}, \tag{43}$$

where $F^n_{i,j}$ is the calculated VOF function at the end of the test, $F^e_{i,j}$ is the exact VOF function at the end of the test, and $F^0_{i,j}$ is the initial VOF function. The Stream algorithm calculations were performed using $n_{\text{stream}} = 1000$ and $F_{\text{wisp}} = 2.5 \times 10^{-4}$.

A detailed comparison of the performance of the algorithms shown in Fig. 7 is given in Rudman [17]. In general the SLIC algorithm, which uses piecewise constant interface

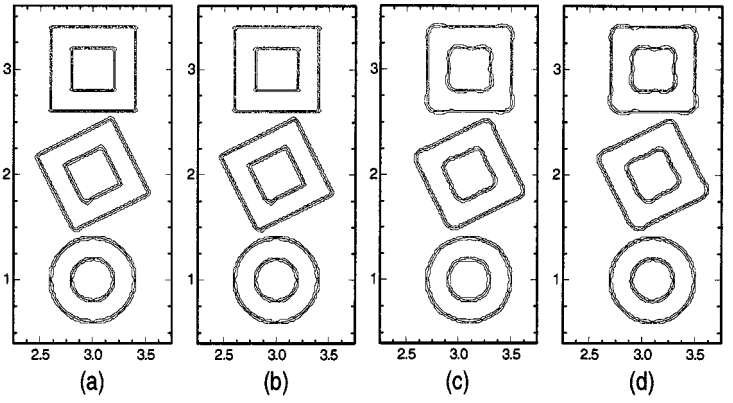

FIG. 8. The straight translation tests of Rudman [17] repeated using the Stream algorithm. Cases (a) and (b) show results for the (1, 0) velocity field, cases (c) and (d) show results for the (2, 1) field. Cases (a) and (c) were calculated using the Youngs method of gradient calculation, cases (b) and (d) using the Puckett method of gradient calculation.

reconstructions, produces good translation results when the velocity field is aligned with the coordinate axis, but poor results otherwise. The Hirt–Nichols algorithm, implemented using a dimensionally split flux calculation, produces relatively poor results in all tests. As noted in Rudman [17], it is surprising that the Hirt–Nichols results are no better than the SLIC results, given that the Hirt–Nichols algorithm is the more complex of the two linear constant reconstruction algorithms. The FCT–VOF algorithm produces results that are generally superior to the results generated by the linear constant algorithms, but results that are generally inferior to the piecewise linear Youngs algorithm.

The Stream algorithm implemented with the Youngs interface gradient calculation produces a similar level of accuracy to the original Youngs advection algorithm. Note that while

TABLE I
Translation Test Error Results

Advection algorithm	Square (0°)	Square (26°)	Circle
Velocity field (1, 0)			
SLIC	8.42×10^{-8}	1.46×10^{-2}	1.30×10^{-2}
Hirt–Nichols	1.03×10^{-8}	6.91×10^{-2}	4.55×10^{-2}
FCT–VOF	3.89×10^{-8}	2.32×10^{-2}	1.28×10^{-2}
Youngs	1.08×10^{-3}	5.35×10^{-3}	3.08×10^{-3}
Stream/Youngs	1.09×10^{-3}	5.86×10^{-3}	3.03×10^{-3}
Stream/Puckett	1.61×10^{-3}	4.57×10^{-3}	1.42×10^{-3}
Velocity field (2, 1)			
SLIC	1.32×10^{-1}	1.08×10^{-1}	9.18×10^{-2}
Hirt–Nichols	6.86×10^{-3}	1.60×10^{-1}	1.90×10^{-1}
FCT–VOF	1.63×10^{-8}	8.15×10^{-2}	3.99×10^{-2}
Youngs	2.58×10^{-2}	3.16×10^{-2}	2.98×10^{-2}
Stream/Youngs	2.70×10^{-2}	3.08×10^{-2}	2.66×10^{-2}
Stream/Puckett	3.33×10^{-2}	3.15×10^{-2}	6.96×10^{-3}

Note. All results expect those calculated using the Stream algorithm are taken from Rudman [17]. Stream algorithm results were calculated using both the Youngs and Puckett methods of free surface gradient calculation.

both methods use the same interface reconstruction technique, the Youngs algorithm uses a dimensionally split flux calculation technique, while the Stream algorithm uses a multidimensional flux technique. It is not surprising that both results calculated using the Youngs method of interface reconstruction are similar, as the increased accuracy afforded by the Stream flux calculation method is only realized when velocity field streamlines are curved.

For the square fluid form translation tests, the Stream algorithm coupled to the Puckett gradient calculation method produces errors that are similar in magnitude to the errors produced using the Youngs gradient calculation method, while for the circular fluid form tests, the Stream algorithm coupled to the Puckett gradient calculation method produces errors which are superior those produced using the Youngs gradient calculation method.

As the curvature at the corners of a square is infinite, the errors generated during a square fluid form translation test will always be limited by the resolution of the computational mesh when a linear piecewise advection algorithm is used. However, this is not the necessarily the case when using a linear constant advection algorithm. Indeed, as shown in Table I, under some specific circumstances the reduced resolution of the linear constant methods is able to predict the translation of the square forms to the precision of the floating point arithmetic.

Fluid forms such as the square are not physically realistic fluid forms, however, as surfaces with infinite curvatures do not occur in real fluid flow situations. Therefore, any difficulties experienced by piecewise linear methods in representing such surfaces do not occur in practice, and it is for this reason that the Stream algorithm, coupled to the Puckett method of interface gradient calculation, is judged to be the most accurate algorithm in these translation tests.

3.2. Rudman–Zalesak Slotted Disk Rotation Test

The Zalesak slotted disk test has become a benchmark test for comparison of scalar advection algorithms. Originally devised by Zalesak [21], the form of the test used here is taken from Rudman [17]. The test involves rotating a slotted disk through one complete revolution within the computational domain under the action of a uniform vorticity velocity field. Advection algorithm accuracy can be gauged by comparing the initial and final positions of the disk.

The Zalesak test performed here uses the same computational domain as was used for the translation tests above. The disk has a diameter of 1 m, and one revolution of the disk is completed in exactly 2524 time steps. This time step corresponds to a Courant number, based on the maximum coordinate velocity existing within the domain, of approximately 0.25. Further details of the form of the test can be found in Rudman [17]. The Stream algorithm tests were performed with $n_{\text{stream}} = 1000$ and a corresponding $F_{\text{wisp}} = 2.5 \times 10^{-4}$.

Figure 9, reproduced from Rudman [17], shows disks calculated using several different advection algorithms, while Fig. 10 shows the test repeated using the Stream algorithm. Quantitative errors for the different advection schemes, again calculated using equation (43), are shown in Table II.

As with the translation tests, a detailed comparison of the performance of the four schemes evaluated in Fig. 9 is given in Rudman [17]. In general the Youngs method, which uses a piecewise linear interface reconstruction technique, produces results which are significantly more accurate than the results produced by the linear constant methods. Again the FCT–VOF method appears more accurate than either linear constant method, but less accurate than the Youngs method.

TABLE II
Slotted Disk Rotation Test Results

Advection algorithm	Error
SLIC	8.38×10^{-2}
Hirt–Nichols	9.62×10^{-2}
FCT–VOF	3.29×10^{-2}
Youngs	1.09×10^{-2}
Stream/Youngs	1.07×10^{-2}
Stream/Puckett	1.00×10^{-2}

Note. All results except those calculated using the Stream algorithm are taken from Rudman [17]. Stream algorithm results were calculated using both the Youngs and Puckett methods of free surface gradient calculation.

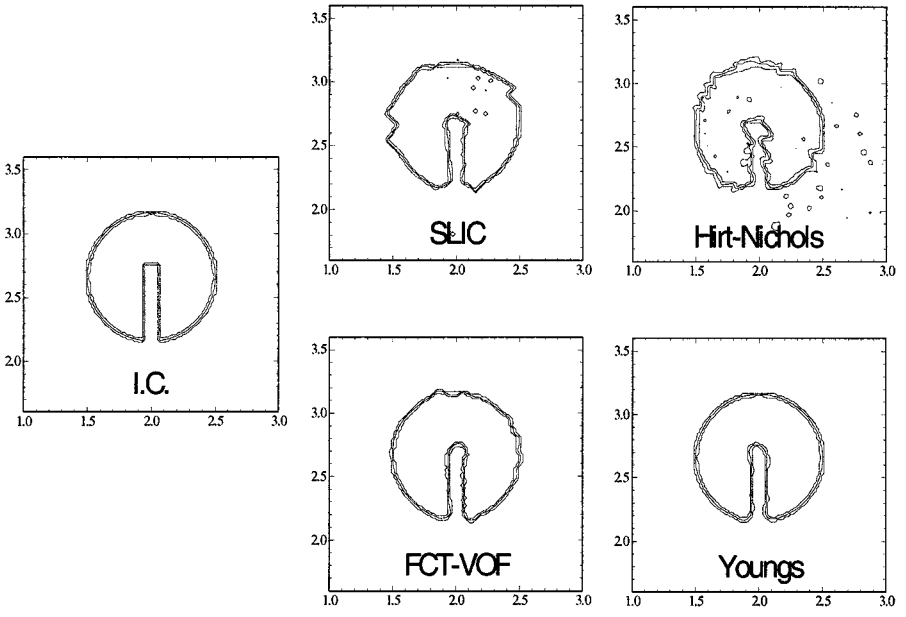

FIG. 9. Zalesak's test problem for solid body rotation: F contours for initial conditions (I.C.) and results after one revolution for each of the four schemes. (Figure and caption are reproduced with permission from Rudman [17].)

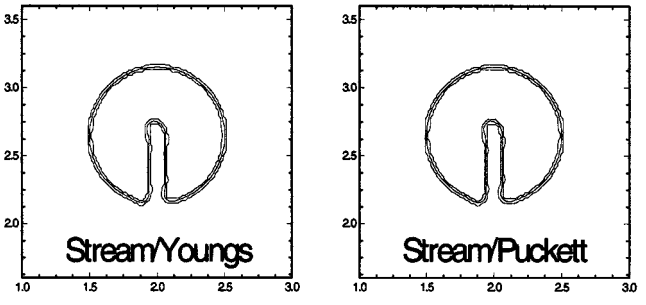

FIG. 10. The Zalesak slotted disk test from Rudman [17] repeated using the Stream algorithm. Results calculated using both the Youngs and Puckett methods of interface gradient calculation are shown.

Comparing the results of Fig. 9 with those of Fig. 10, the accuracy of the Stream and Youngs algorithms appears similar. Examining the errors shown in Table II, however, it appears that the Stream advection method coupled to the Youngs interface reconstruction method is slightly more accurate than the Youngs method, indicating that the Stream boundary flux calculation technique is marginally more accurate than the Youngs dimensionally split flux calculation technique in this instance. The Stream scheme operating with the Puckett method of interface reconstruction produces the most accurate results in this test.

The Zalesak rotation test results presented here are informative, but should be interpreted with care. An analysis of the errors calculated in Table II has shown that for the advection algorithms tested, the primary region of error generation in each test was at the sharp corners which define the slot in the disk. Thus, the Zalesak test results presented are largely a measure of the employed interface reconstruction technique to represent fluid interfaces having high curvatures, rather than a measure of the accuracy of the different VOF flux calculation techniques. It is for this reason that there are only minor differences in the errors generated by the different piecewise linear algorithms.

3.3. Rider–Kothe Reversed Single Vortex Test

A more thorough test of VOF advection is made when the velocity field contains nonuniform vorticity, causing the fluid to deform and shear as it translated throughout the computational domain. Such a test, taken from the work of Rider and Kothe [16], is performed here.

In the reversed single vortex test, a cylinder of fluid, of radius 0.15 m and centered at (0.5, 0.75), is deformed in a velocity field specified by the Stream function

$$\Psi = \frac{1}{\pi}\sin^2(\pi x)\sin^2(\pi y)\cos\left(\frac{\pi t}{T}\right). \tag{44}$$

The computational domain used in the test had the dimensions 1×1 m^2, and the duration of each test was T s. A Courant number of 1, based on the maximum coordinate velocity within the computational domain, was used and the tests were performed with $n_{\text{stream}} = 100$ and $F_{\text{wisp}} = 0.01$.

Equation (44) specifies a vortex which shears the cylinder of fluid into a spiral type form. The temporal component of Eq. (44) is responsible for reversing the vortex. At time $t = T/2$ s, the deformation of the cylinder should be at a maximum, while at the end of the test, the fluid should return to the initial position. Thus, like the Zalesak test, an indication of the accuracy of the advection algorithm can be gauged by comparing the initial and final positions of the fluid form. Further details of the form of the test can be found in Rider and Kothe [16].

Figure 11 shows the reversed vortex test computed using the Stream scheme coupled to the Youngs gradient calculation method, calculated using a variety of mesh sizes and three different test durations. Figure 12 shows the same tests computed using the Stream scheme coupled to the Puckett gradient calculation method. In both figures the VOF function is not represented by contour lines, but rather as individual blocks of fluid with each cell interface represented using the techniques outlined in Section 2.3. This method of representation is consistent with the results presented in Rider and Kothe [16] and allows comparison of the accuracy of the alternative interface reconstruction techniques.

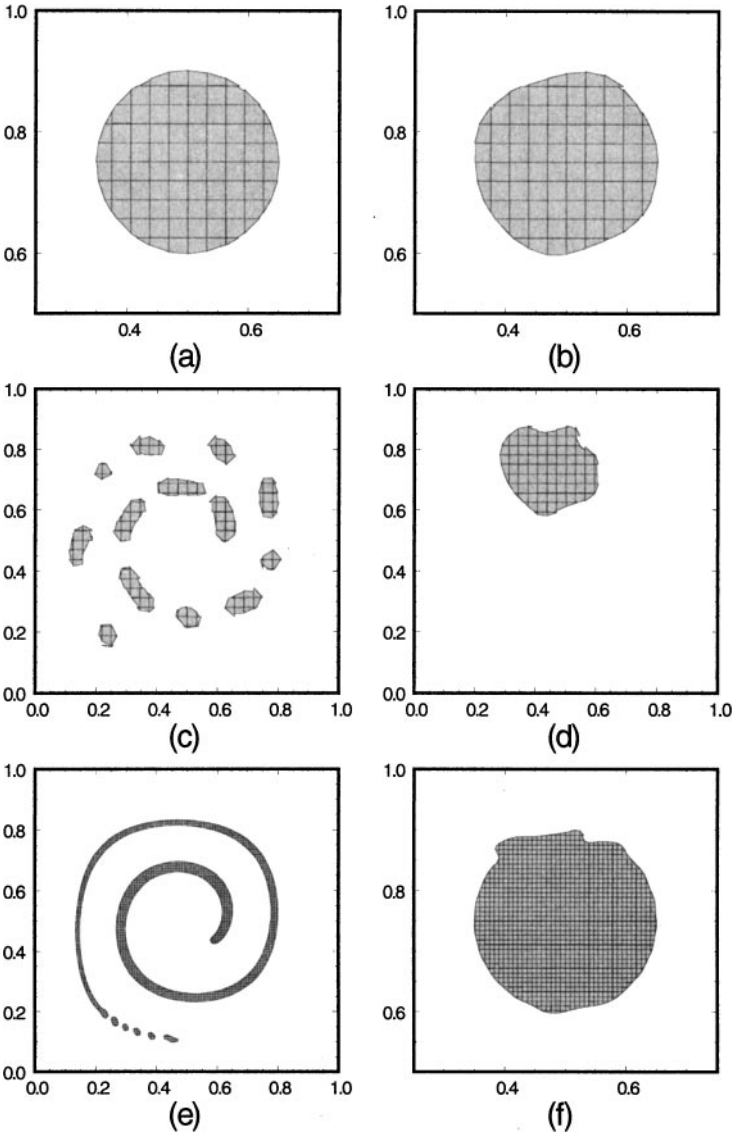

FIG. 11. The reversed single vortex test performed using the Stream scheme coupled to the Youngs method of interface gradient calculation. Cases (a–d) are performed on a 32^2 mesh, and cases (e) and (f) are performed on a 128^2 mesh. Case (a) uses $T = 0.5$, case (b) $T = 2.0$, and cases (c–f) $T = 8.0$. Cases (c) and (e) show the cylinder at $t = T/2$, while the remainder show the cylinder at $t = T$. All times are measured in seconds.

Figures 11 and 12 show qualitatively similar results. Case (a) in both figures shows the final fluid form after a $T = 0.5$ s test performed on a coarse 32^2 mesh. In both figures the fluid has returned to the starting position with reasonable accuracy. Case (b) in both figures shows the test repeated, but over a longer duration of $T = 2.0$ s. The final fluid position is now less accurate, a result of the greater amount of deformation the fluid is subjected to during the test.

Cases (c) and (d) in Figs. 11 and 12 show the coarse mesh computations repeated using $T = 8.0$ s. The final fluid form in both figures, shown by case (d), is significantly in error.

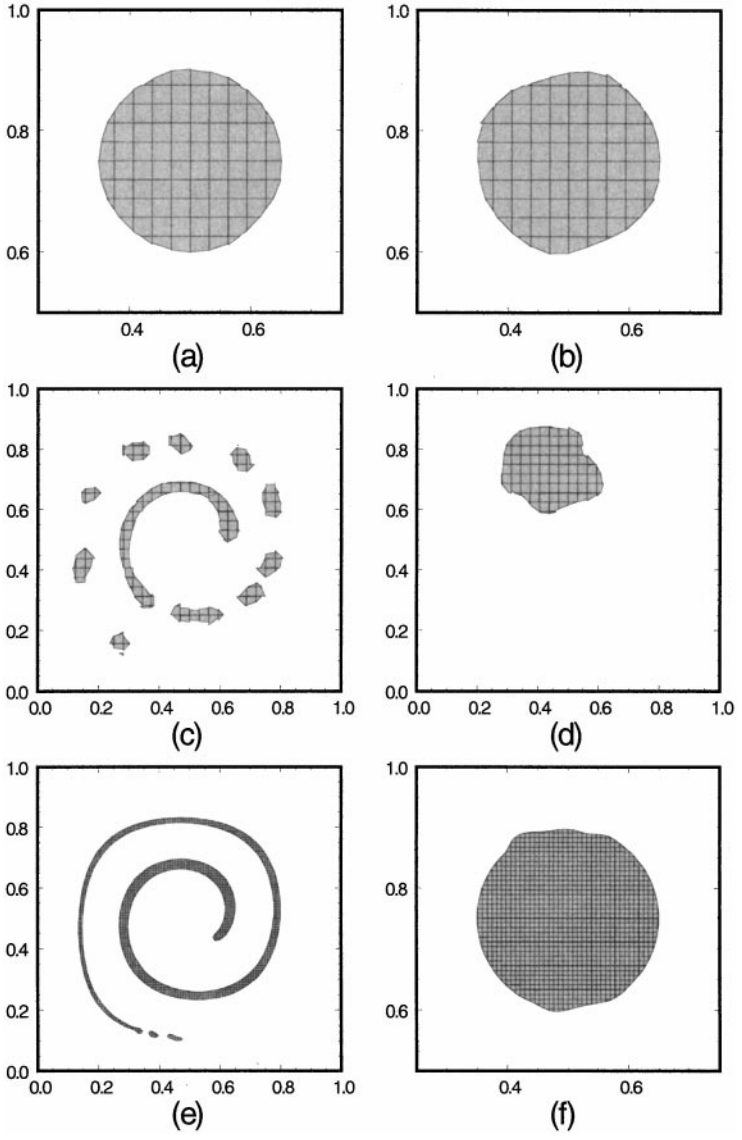

FIG. 12. The reversed single vortex test performed using the Stream scheme coupled to the Puckett method of interface gradient calculation. Cases (a–d) are performed on a 32^2 mesh, and cases (e) and (f) are performed on a 128^2 mesh. Case (a) uses $T = 0.5$, case (b) $T = 2.0$, and cases (c–f) $T = 8.0$. Cases (c) and (e) show the cylinder at $t = T/2$, while the remainder show the cylinder at $t = T$. All times are measured in seconds.

As shown by case (c), this is a result of the fluid breaking into a number of discrete "globs" at the time of maximum deformation, $t = T/2$ s.

Breakup of the spiral occurs because the width of the fluid form becomes less than the computational cell dimension. In these cases, the interface reconstruction technique tends to arrange the small amounts of fluid in each cell as close together as possible, causing the fluid to glob. This process can be thought of as numerical surface tension. As shown, its effect is only evident when the dimensions of the fluid region are similar to or smaller than the dimensions of the computational cells. In real fluid simulations, where the fluid form should have dimensions larger than the dimensions of the cells, its effect is negligible.

Cases (e) and (f) in Figs. 11 and 12 show the long duration test repeated using a finer 128^2 mesh size. As indicated in case (e), the spiral is now more accurately represented at $t = T/2$ s, which results in a more accurate fluid reconstruction at the end of the test. Inaccuracies present at the top of the circle in case (f) are caused by the minor breakup of the spiral at its thinnest end in case (e), while inaccuracies present at the bottom of the circle in case (f) are caused by inaccuracies in the reconstruction of the thicker end of the spiral throughout the duration of the test.

Comparing Figs. 11 and 12, it appears that the Puckett method of free surface reconstruction results in a higher accuracy advection calculation than the Youngs method. This is particularly evident in the long duration tests, where the degree of fluid breakup at the time of maximum deformation is significantly less when using the Puckett method rather than the Youngs method.

In some cases small amounts of fluid appears in cells which are separated from the free surface interface by approximately one cell dimension or less. These amounts are due to the approximate boundary flux integration technique employed by the Stream scheme and are not redistributed by the wisp eradication algorithm as they are in the close vicinity of an interface. There were no wisps generated in any of the tests which did not remain in the close vicinity of an interface.

Table III shows quantitative errors calculated during the reversed vortex tests. To maintain consistency with Rider and Kothe [16], errors are calculated here using

$$E = \sum_{i,j} \delta x_i \delta y_j \left| F^n_{i,j} - F^e_{i,j} \right|, \tag{45}$$

TABLE III
Geometrical Advection Test Errors and Convergence Rates

	$T = 0.5$		$T = 2.0$		$T = 8.0$	
Grid	Error	Order	Error	Order	Error	Order
			Rider–Kothe/Puckett			
32^2	7.29×10^{-4}		2.36×10^{-3}		4.78×10^{-2}	
		2.36		2.01		2.78
64^2	1.42×10^{-4}		5.85×10^{-4}		6.96×10^{-3}	
		1.86		2.16		2.27
128^2	3.90×10^{-5}		1.31×10^{-4}		1.44×10^{-3}	
			Stream/Puckett			
32^2	5.51×10^{-4}		2.37×10^{-3}		3.72×10^{-2}	
		2.32		2.07		2.45
64^2	1.10×10^{-4}		5.65×10^{-4}		6.79×10^{-3}	
		1.71		2.10		2.53
128^2	3.38×10^{-5}		1.32×10^{-4}		1.18×10^{-3}	
			Stream/Youngs			
32^2	3.42×10^{-4}		2.49×10^{-3}		3.61×10^{-2}	
		0.99		1.82		1.85
64^2	1.72×10^{-4}		7.06×10^{-4}		1.00×10^{-2}	
		0.84		1.66		2.22
128^2	9.60×10^{-5}		2.23×10^{-4}		2.16×10^{-3}	

Note. Results shown for the Rider–Kothe scheme are taken from Rider and Kothe [16].

where as previously $F^n_{i,j}$ is the calculated VOF function at the end of the test, and $F^e_{i,j}$ is the exact VOF function at the end of the test. This error has the units of m^2.

Comparing the Stream algorithm errors computed using the two alternative interface reconstruction methods, the Youngs method appears generally less accurate than the Puckett method, except during the coarsest grid tests. This result supports the observations made in Rider and Kothe [16], where it was found that the Youngs method provides a higher level of reconstruction accuracy than error minimization methods when the curvature of the fluid feature has dimensions which are of magnitude similar to the cell dimensions. For the majority of the tests, however, the Puckett method results in errors which are significantly smaller than errors calculated using the Youngs method.

Comparing the errors calculated using the Stream scheme coupled to the Puckett interface gradient method, and the Rider–Kothe scheme coupled to the same interface gradient method, the Stream scheme appears to be more accurate. In all but two of the nine tests, the errors calculated by the Stream algorithm are significantly smaller than those calculated by the other multidimensional advection algorithm.

The order values shown in Table III are calculated as the convergence rates of the advection test errors as the grid size is refined. As a constant Courant number is used in all tests, as the mesh is refined, the time step used in the tests is also decreased. Consequently, the convergence rates shown in the table are an indication of the combined spatial and temporal accuracy of the algorithms, as will be discussed further in Section 4.3. In general the schemes employing the Puckett reconstruction technique are of second order accuracy, while the scheme employing the Youngs method is of a slightly lower accuracy.

In the next section, an analysis of the origin of errors generated during the different advection tests performed using the Stream algorithm is given.

4. ALGORITHM ERRORS AND ERROR CONVERGENCE RATES

Fluid volume errors calculated by the Stream algorithm during a fixed duration advection test are composed of errors resulting from free surface interface reconstructions and errors resulting from cell boundary flux calculations. Thus, we may represent the total volume error found during an advection test by

$$E = E_{\text{free surface reconstruction}} + E_{\text{cell boundary flux}}. \tag{46}$$

The free surface reconstruction error is composed of a spatial and temporal error,

$$E_{\text{free surface reconstruction}} = A_1 \delta x^{n_1} f(C), \tag{47}$$

where A_1 is a constant, n_1 is the spatial order of the free surface reconstruction algorithm, and $f(C)$ is a function of the Courant number,

$$C = \frac{u\delta t}{\delta x}. \tag{48}$$

The cell boundary flux error is composed of three terms,

$$E_{\text{cell boundary flux}} = A_2 \delta x^{n_2} + A_3 \delta t^{n_3} + A_4 \left(\frac{u}{n_{\text{stream}}} \right)^{n_4}. \tag{49}$$

The first two terms on the right side of Eq. (49) represent errors resulting from the difference

between the velocity field assumed by the Stream algorithm, and the actual velocity field imposed in the advection test. The third term on the right side of Eq. (49) represents errors resulting from the approximate integration method employed by the Stream algorithm.

During an advection test where the velocity field is constant in time, the temporal term in Eq. (49) becomes zero. Also, if the actual velocity field assumed in the advection test can be represented exactly by the velocity field assumed by the Stream algorithm, the first spatial term on the right side of Eq. (49) is also zero. A velocity field which satisfies these two criteria is that used in straight translation tests.

In the remainder of this section, we analyze the convergence rates and behavior of the errors given by Eqs. (47) and (49).

4.1. Approximate Cell Boundary Flux Integration Error

A detailed discussion of the convergence rate of the approximate cell boundary flux integration error is included in this study, as among VOF advection algorithms, this error is unique to the Stream algorithm.

An estimate of the order of convergence of the flux integration error, n_4, can be found by consideration of Eq. (41). As each cell has four individual boundaries, the total volume error produced during a single computational time step is

$$\text{Error}(V) = O\left(\frac{2u\delta t\delta x}{n_{\text{stream}}}\right), \tag{50}$$

where V represents fluid volume.

Under the Stream algorithm, integration errors only occur in cells which are in the vicinity of a free surface. Thus, given a length of free surface interface S, the number of cells in the computational domain which will sustain integration errors during a single time step is proportional to $S/\delta x$. Also, over an advection test of duration T, the number of time steps completed is proportional to $T/\delta t$. Consequently, applying Eq. (50) over the volume and duration of an advection test, we may expect the cell boundary flux integration error to be proportional to

$$E_{\text{flux integration error}} \propto ST\frac{u}{n_{\text{stream}}}. \tag{51}$$

This implies that the order of convergence of the integration error, n_4, is one.

To validate this theory, we perform a simple translation advection test. In this test a circle of fluid, of radius 0.2 and initially located at coordinates (0.25, 0.25), is translated with equal velocity components (1, 1) for 0.5 s throughout a computational domain having dimensions 1×1 m^2. Under an exact advection method, the fluid should remain in the form of a circle and be centered at (0.75, 0.75) on completion of the test. A Courant number of unity is used, and Eq. (45) is used to quantify the advection test error.

As the streamlines assumed in this simple translation test are straight and constant in time, the first two error terms in Eq. (49) can be neglected. Also, as the Courant numbers in both dimensions are constant and equal to one throughout the domain, the fluid contained within each cell should translate entirely to one adjacent cell during each computational time step. As a result, free surface interface reconstructions should be identical at all times. Thus, the error term given in Eq. (47) can be neglected, and the total advection test error calculated is composed of only the boundary flux integration error.

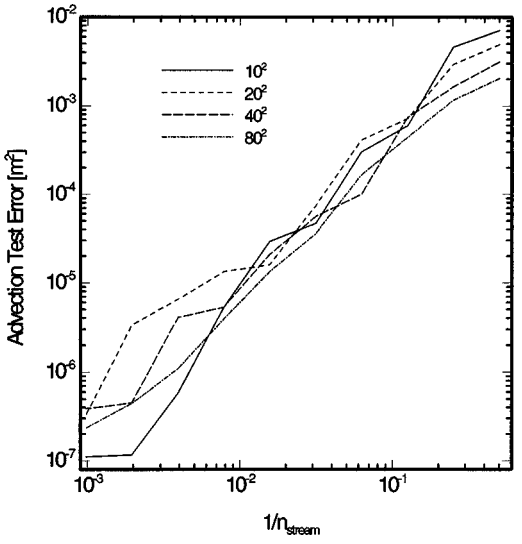

FIG. 13. Boundary flux integration errors calculated during a simple translation test. The variable n_{stream} ranges between 2 and 1024 and results found using four different mesh sizes are displayed.

Figure 13 shows the advection test errors calculated using four different grid sizes, and values of n_{stream} ranging from 2 to 1024. For these tests the dewisping algorithm detailed in Section 2.5.2 was deactivated, and the Puckett interface gradient calculation method was employed. Figure 13 shows that the order of convergence of the boundary flux integration error is approximately constant, and as suggested by the above analysis, is independent of grid size and time step. The average order of convergence calculated over the four tests was $n_4 = 1.6$. This is slightly higher than the order of unity suggested by equation (51), but is of a similar magnitude.

As discussed by Rudman [17], during fluid flow computations the CPU time expended on advecting the VOF function is generally small compared with the time spent solving the discretized Navier–Stokes equations, so the computational efficiency of the overall code is not critically dependent on the efficiency of the VOF advection algorithm. Indeed, the present authors have found that a higher level of accuracy in the VOF advection process can often allow a larger time step to be used in the calculation, thus reducing the total computational cost of a fluid simulation. However, an analysis of the computational expense of the Stream algorithm is included, primarily to show the effect the variable n_{stream} has on computational efficiency.

Figure 14 shows the CPU time spent on calculating the reversed single vortex test described in Section 3.3. The test was calculated using $T = 2.0$ s, and performed using the Stream algorithm with either the Youngs or Puckett methods of interface gradient calculation. The variable n_{stream} was varied between 2 and 1024.

The results of Fig. 14 show that the computational expense of a fluid simulation increases as n_{stream} increases and as the number of computational cells increases. The Puckett method of gradient calculation is more expensive than the Youngs method, as the Puckett method is iterative, requiring a total of nine cell interface reconstructions in each cell for each interface orientation iteration.

It was found that the computational expense of the Stream scheme is comparable with other multidimensional schemes when the variable n_{stream} is less than 10. Increasing n_{stream}

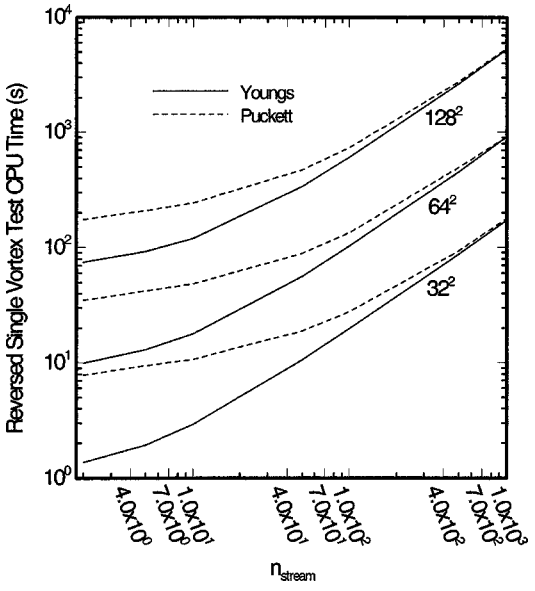

FIG. 14. CPU time used when performing the reversed single vortex test with $T = 2.0$ s.

to 100 results in an approximately order of magnitude increase in the expense of the scheme, but at these levels the use of the scheme is still viable. As shown in Fig. 14, a value of 1000 for n_{stream} increases the computational expense by approximately a further order of magnitude. As demonstrated by the results of Fig. 13, such a high level of n_{stream} is unjustified in real fluid simulations—the additional computational time would be better spent on a finer mesh computation.

4.2. Free Surface Reconstruction Errors

As indicated by Eq. (47), the free surface reconstruction error is composed of a spatial and temporal component. We will consider first the spatial component, followed by the temporal component.

To determine the order of convergence of the spatial component, the simple circle translation test detailed in the previous section is repeated, but with a number of changes. As previously, as the test involves only straight translation, the first two terms on the right side of Eq. (49) are zero. In order to examine just the reconstruction error, a nonzero and constant Courant number is used throughout the domain, and the variable n_{stream} is set to 1024. Such a high value for n_{stream} ensures that approximate integration errors generated during the test are several orders of magnitude smaller than free surface reconstruction errors, and as a result, advection test errors found during this test approximate free surface reconstruction errors.

Figure 15 shows the free surface reconstruction errors calculated during the translation test using both the Youngs and Puckett methods of interface gradient calculation. The advection test errors calculated using the Puckett method converge consistently with grid refinement, and the average convergence rate found over the three series of tests was $n_1 = 1.7$. This result is in agreement with the spatial surface reconstruction results presented in Rider and Kothe [16], which conclude that the Puckett method generally produces second-order surface reconstructions.

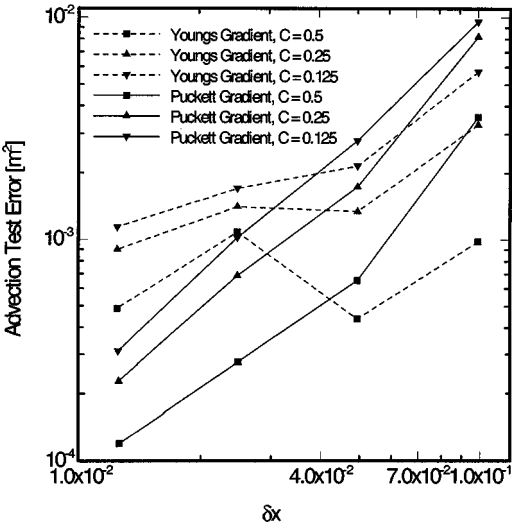

FIG. 15. Free surface reconstruction errors calculated during the circle translation test using both the Youngs and Puckett methods for interface gradient calculation. The results shown were calculated using four computational cell sizes and three Courant numbers.

The convergence of errors calculated using the Youngs method is less consistent, with notable increases in reconstruction error occurring during two of the tests when the computational cell size is reduced from 0.05 to 0.025 m. Experience using the Youngs method has shown that generally the method does converge consistently with grid refinement, although the rate of convergence is significantly lower than that found using the Puckett method of gradient calculation. The study of Rider and Kothe [16] concluded that the Youngs method produces generally first-order surface reconstructions.

The errors calculated using Youngs method of gradient calculation on the coarsest grids are lower than the errors calculated using the Puckett method on the same grids. This observation supports both of the observations made in Section 3.3 and the grid reconstruction results presented in Rider and Kothe [16]. The Youngs method is more accurate than the Puckett method in approximating a continuous free surface interface when the curvature of the interface is of a similar magnitude to the computational cell size.

The dependence of the free surface reconstruction error on the Courant number is described by $f(C)$, as defined in Eq. (47). To determine the form of this function, we repeat the circle translation test using the same conditions that were used to determine the spatial reconstruction error convergence rate, however in this test the Courant number is reduced from 1 down to 3.91×10^{-3} while the mesh size remains constant.

Figure 16 shows the free surface reconstruction errors calculated during the advection test as a function of the inverse Courant number. The test has been repeated using both the Youngs and Puckett gradient calculation methods and using three different mesh sizes.

The results of Fig. 16 indicate the form $f(C)$. The reconstruction error is at a minimum when the Courant number is unity, but approaches a maximum value asymptotically as the Courant number approaches zero. This suggests that to minimize the free surface reconstruction error when using a piecewise linear reconstruction algorithm, the time step should be set as high as possible. Both the Youngs and Puckett gradient calculation methods result in similar forms for $f(C)$.

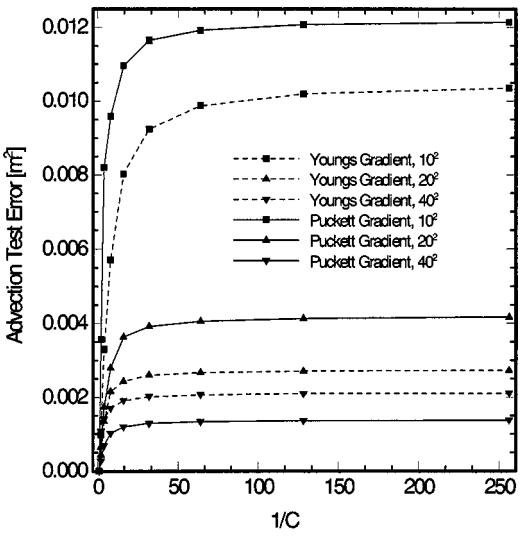

FIG. 16. Free surface reconstruction errors calculated as a function of the Courant number. The test has been repeated using both the Youngs and Puckett methods of free surface gradient calculation and using three different grid sizes.

That the free surface error approaches a maximum value asymptotically as the Courant number is reduced is important to the viability of the method. If the error were to increase unbounded as the time step approached zero, the method could not be relied upon when small time steps were required, and would not be viable.

In the simplest terms the form of $f(C)$ is caused by the nature of the reconstruction technique. Each time the free surface is reconstructed, a discrete amount of error is added to the simulation, because the linear interfaces used in the computation can never exactly reproduce the form of the actual free surface. The smaller the Courant number, the more time steps are required to calculate fluid behavior over a set duration, and consequently the greater the amount of free surface reconstruction error is introduced. The reconstruction error remains bounded as the Courant number approaches zero because under such conditions, the change in surface reconstruction between time steps is only slight, and so the reconstruction error introduced at each time step is only small.

Note that in this advection test there is no free surface reconstruction error when the Courant number is one. As was described above, this behavior is a result of the simple translation form of the advection test. In realistic free surface flow computations, the Courant number would not be constant and uniform throughout the computational domain, and the reconstruction error would not vanish at a Courant number of one. However, during actual free surface flow computations the free surface reconstruction error would still be minimized at the maximum Courant number.

4.3. Cell Boundary Flux Errors

The final two error terms to consider are the first two terms on the right side of Eq. (49)—the cell boundary flux errors. These two terms are not dependent on the accuracy of flux integration specified by n_{stream}, but as discussed above, they account for errors due to spatial and temporal differences between the velocity field assumed by the Stream algorithm and

the actual velocity field used in the advection test. As discussed above, during a simple translation test, these terms vanish.

Generally, the free surface reconstruction term is of a magnitude larger than any of the flux calculation error terms shown in Eq. (49) during an advection test. Even in the reversed single vortex test described in Section 3.3, where the velocity field could not be represented exactly by the Stream velocity field and the velocity field was not constant in time, the advection test errors were dominated by free surface reconstruction errors. This is evident in the high degree of dependence the test results had on the method of interface gradient calculation.

Nevertheless, the results of Section 3.3 do indicate that the cell boundary flux errors introduced by the Stream algorithm are lower than comparable errors introduced by the multidimensional Rider–Kothe scheme. This is shown by the lower total advection test errors generated by the Stream scheme compared to the Rider–Kothe scheme when using the same free surface reconstruction technique. The reason for the lower boundary flux errors generated by the Stream algorithm can be explained using Fig. 17.

Figure 17 shows example donating regions for a right cell boundary calculated using four alternative multidimensional advection schemes. In all cases the horizontal component of the velocity at the boundary is assumed to be directed toward the right, and the local velocity field may contain nonuniform vorticity.

As shown by Fig. 17, it is evident that the Stream algorithm will produce the donating region which most faithfully represents the actual donating region for a given cell boundary,

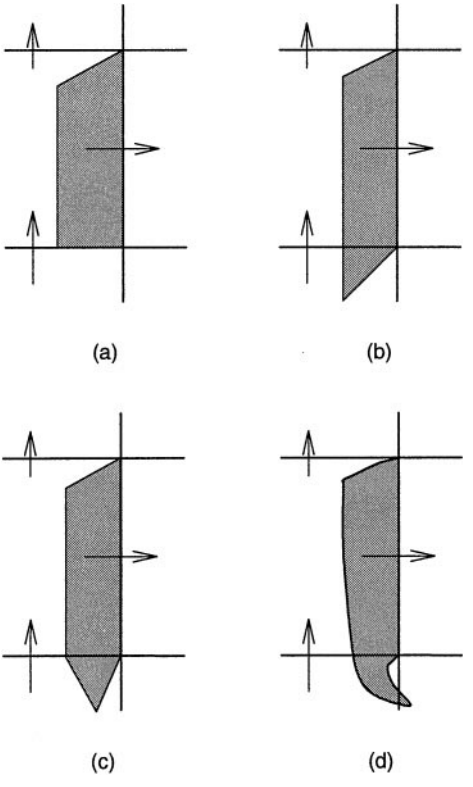

FIG. 17. Example donating regions defined using the (a) DDR, (b) Rider–Kothe, (c) Puckett *et al.*, and (d) Stream schemes.

as the donating region may traverse as many computational cells as the time step allows, and it is not limited in form to simple straight-sided geometries. This donating region flexibility is what produces the lower values for the cell boundary flux terms under the Stream scheme. Obviously the relative advantages of the Stream flux calculation technique can only be realized when the velocity field is not uniform, a fact supported by the translation advection test results of Section 3.1.

Determining the rate of convergence of the cell boundary flux terms is more difficult. The error convergence rates calculated during the reversed single vortex tests are a combination of all the error convergence rate terms which were given in Eqs. (47) and (49). Unfortunately, as the errors generated during the reversed spiral tests are dominated by the free surface reconstruction errors, it is difficult to determine the relative magnitudes of each of the constituent convergence rates. Consequently, no conclusions as to the magnitude of n_2 or n_3 can be drawn.

Some general observations can be made from Fig. 17 about the different multidimensional advection techniques. The Defined Donating Region (DDR) algorithm, developed by Harvie and Fletcher [6], appears to be the least accurate of the alternative schemes. However, fluid crossing a boundary under this method can only originate from the one donating cell adjacent to the boundary. This property ensures that under the DDR scheme fluid volume is conserved rigorously, and no fluid flotsam or wisps are produced. Thus, the DDR scheme is applicable to fluid dynamics problems where stability of the free surface interface is paramount.

Donating regions defined by either the Rider–Kothe or Puckett *et al.* [15] schemes may traverse one, two, or three cells. This produces a level of accuracy greater than the DDR scheme, but it is at the expense of fluid volume conservation. As the donating regions are defined independently for each cell boundary, when the velocity field is spatially varying, donating regions can overlap or not include some fluid regions Rider and Kothe [16]. Consequently, both the Rider–Kothe and Puckett *et al.* schemes require a local fluid redistribution algorithm in order to achieve fluid conversation.

The Stream scheme also requires a local fluid redistribution algorithm, but the reason for its inclusion is significantly different to the reason it is included in other multidimensional schemes. The purpose for the redistribution algorithm under the Stream scheme is to correct errors resulting from the approximate integration technique. As the user can specify the accuracy of the integration technique through the variable n_{stream}, the user has control over the dependence on the fluid redistribution algorithm. This is not the case with other VOF advection algorithms.

5. CONCLUSIONS

A new VOF advection algorithm, termed the Stream scheme, has been presented. The algorithm uses a linear piecewise free surface reconstruction method, combined with a unique fully multidimensional boundary flux integration technique. The performance of the new algorithm has been compared against the performance of other VOF advection schemes which use a variety of interface reconstruction techniques and a variety of boundary flux calculation techniques. In almost all tests performed, the Stream algorithm displayed a level of accuracy higher than the alternative algorithms, although possibly at a greater computational expense.

Additionally, an analysis of errors generated by the Stream scheme when performing an advection calculation has been included. It was found that, in general, free surface

reconstruction errors dominate total advection test errors. The error minimization method of free surface gradient calculation due to Puckett [14] generally produces spatially second order surface reconstructions, while the gradient calculation method due to Youngs [20] produces lower order reconstructions. Interestingly, when using a piecewise linear reconstruction technique, surface reconstruction errors are minimized when the time step is maximized; however, the reconstruction error remains bounded as the time step is reduced to zero, ensuring the viability of the technique.

REFERENCES

1. N. Ashgriz and J. Y. Poo, FLAIR: Flux line-segment model for advection and interface reconstruction, *J. Comput. Phys.* **93**, 449 (1991).
2. P. K. Barr and W. T. Ashurst, *An Interface Scheme for Turbulent Flame Propagation* (Technical Report SAND82-8773, Sandia National Laboratories, 1984).
3. A. J. Chorin, Flame advection and propagation algorithms, *J. Comput. Phys.* **35**(1), 1 (1980).
4. R. Debar, *Fundamentals of the KRAKEN Code* (Technical Report UCIR-760, LLNL, 1974).
5. D. J. E. Harvie, *A Hydrodynamic and Thermodynamic Simulation of Droplet Impacts on Hot Surfaces* (Ph.D. thesis, Department of Mechanical and Mechatronic Engineering, University of Sydney, NSW, Australia, 1999).
6. D. J. E. Harvie and D. F. Fletcher, A new volume of fluid advection algorithm: The defined donating region scheme, *Int. J. Numer. Methods Fluids*, in press.
7. D. B. Kothe, R. C. Mjolsness, and M. D. Torrey, *RIPPLE: A Computer Program for Incompressible Flows with Free Surfaces* (Technical Report LA-12007-MS, Los Alamos National Laboratory, 1994).
8. B. Lafaurie, C. Nardone, R. Scardovelli, S. Zaleski, and G. Zanetti, Modelling merging and fragmentation in multiphase flows with SURFER, *J. Comput. Phys.* **113**, 134 (1994).
9. L. Margolin, J. M. Reisner, and P. K. Smolarkiewicz, Application of the volume-of-fluid method to the advection-condensation problem, *Mon. Weather Rev.* **125**, 2265 (1997).
10. B. D. Nichols, C. W. Hirt, and R. S. Hotchkiss, *SOLA-VOF: A Solution Algorithm for Transient Fluid Flow with Multiple Free Boundaries* (Technical Report LA-8355, Los Alamos Scientific Laboratory, 1980).
11. W. Noh and P. Woodward, SLIC (simple line interface method), *Lecture Notes Phys.* **59**, 330 (1976).
12. J. E. Pilliod, Jr., *An Analysis of Piecewise Linear Interface Reconstruction Algorithms for Volume-of-Fluid Methods* (Master's thesis, University of California at Davis, 1992).
13. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in FORTRAN: The Art of Scientific Computing* (second ed., Cambridge University Press, Cambridge, UK, 1992).
14. E. G. Puckett, A volume of fluid interface tracking algorithm with applications to computing shock wave rarefraction, in *Proceedings of the 4th International Symposium on Computational Fluid Dynamics, 1991*, pp. 933–938.
15. E. G. Puckett, A. S. Almgren, J. B. Bell, D. L. Marcus, and W. J. Rider, A high-order projection method for tracking fluid interfaces in variable density incompressible flows, *J. Comput. Phys.* **130**, 269 (1997).
16. W. J. Rider and D. B. Kothe, Reconstructing volume tracking, *J. Comput. Phys.* **141**, 112 (1998).
17. M. Rudman, Volume-tracking methods for interfacial flow calculations, *Int. J. Numer. Methods Fluids* **24**, 671 (1997).
18. R. Scardovelli and S. Zaleski, Direct numerical simulation of free-surface and interfacial flow, *Annu. Rev. Fluid Mech.* **31**, 567 (1999).
19. D. L. Youngs, Time-dependent multimaterial flow with large fluid distortion, in *Numerical Methods for Fluid Dynamics*, edited by K. Morton and M. Baines (Academic Press, New York, 1982), pp. 273–285.
20. D. L. Youngs, *An Interface Tracking Method for a 3D Eulerian Hydrodynamics Code* (Technical Report 44/92/35, AWRE, 1984).
21. S. T. Zalesak, Fully multidimensional flux-corrected transport algorithms for fluids, *J. Comput. Phys.* **31**, 335 (1979).