Proceedings of the ASME 2020 Fluids Engineering Division Summer Meeting
FEDSM2020
July 12-16, 2020, Orlando, USA

FEDSM2020-20486

OPEN-SOURCE ACCELERATED VORTEX PARTICLE METHODS FOR UNSTEADY FLOW SIMULATION

Mark J. Stock
Applied Scientific Research, Inc.
Irvine, California
Email: markjstock@gmail.com

Adrin Gharakhani
Applied Scientific Research, Inc.
Irvine, California
adrin@applied-scientific.com

ABSTRACT

A new open-source CFD solver is being developed based on the vorticity transport equations for simulation of unsteady incompressible flow in complex geometries. This is a hybrid approach, which uses a compact high-order finite-difference method to predict the flow in the near-boundary region and a Lagrangian Vortex Particle Method (LVPM) for the off-boundary vorticity-containing region. This paper focuses on the latter, presenting the particular LVPM implemented in the package and demonstrating selected benchmarks from simulations of flow in 2D lid-driven cavity, flow around a rotating cylinder (both at $\mathrm{Re} = 1{,}000$*), and impulsively started flow over a sphere at* $\mathrm{Re} = 25, 40, 60, 80, 100$*. In addition, novel ideas on the development of an efficient and lightweight Graphical User Interface (GUI), as well as new approaches to cross-platform hardware acceleration for Teraflop/s computing on a desktop - achieving over two orders of magnitude speedup vs. an optimized serial code - are discussed.*

NOMENCLATURE

A	Area of a body in 2D.
d	Number of spatial dimensions (2 or 3).
D	Diameter.
N_v	Number of vortex particles.
N_p	Number of panels.
r	Core radius of particle smoothing function.
Re	Reynolds number.
t	Time.
$\boldsymbol{n}$	Surface normal vector.
$\boldsymbol{u}$	Velocity vector.
$\boldsymbol{U}_\infty$	Freestream velocity vector.
$\boldsymbol{x}$	Position vector.
$\mathcal{D}$	Computational domain.
$\partial\mathcal{D}$	Boundary of computational domain.
Δt	Computational time step.
Δl	Length of a panel in 2D.
σ	Surface source sheet strength.
Ω	Body rotation rate.
$\boldsymbol{\Gamma}$	Circulation vector ($\boldsymbol{\omega}\, dV$).
$\boldsymbol{\gamma}$	Surface vortex sheet strength.
$\boldsymbol{\omega}$	Vorticity vector.

INTRODUCTION

Lagrangian Vortex Particle Methods are well-suited for simulation of unsteady vortex-dominated incompressible fluid flow in the moderate Reynolds number regime, including transitional flow. Key advantages of LVPM are (1) near absence of numerical diffusion, thanks to the Lagrangian accounting of convection; (2) dynamic solution adaptivity resulting from the inherent ability of vortex elements to automatically convect toward regions with high vorticity strength; and (3) excellent capacity for high-efficiency parallelization and vectorization due to the high arithmetic intensity of velocity (and velocity gradient) evaluation via the Biot-Savart formulation. However, accurate and efficient treatment of the flow near the boundary, which invariably requires the use of anisotropic vortex elements, has remained mostly elusive to date. On the other hand, compact high-order Eulerian CFD methods have recently demonstrated excellent accuracy and computational efficiency on canonical problems; however, they are rather chal-

lenging to apply to problems that involve multiple bodies in relative motion, such as those observed in many cardiovascular devices. To this end, leveraging the best attributes of these two very diverse approaches, we are developing a new open-source hybrid CFD solver, which combines a vorticity-based high-order Eulerian method for accurate prediction of the near-body flow with a LVPM for off-body CFD to accelerate the design cycle for cardiovascular and similarly complex devices. This paper focuses on the development of the particular LVPM used in the code, software details, and hardware acceleration for both the two-dimensional (*Omega2D* [1]) and three-dimensional (*Omega3D*) versions.

A cross-platform, immediate-mode GUI allows the user to either instantly start up a benchmark calculation or carefully design a specific simulation with much less effort than is normally possible with traditional grid-based CFD tools. The GUI allows quick problem set-up, offers run and pause buttons, and displays the simulation results as they complete, frequently in near real-time. The simulation engine can be run with or without the GUI, reads portable human-readable setup files, and writes results directly to ParaView-native VTU files. Simulations of several canonical flows compare well with published results.

This vortex code will be paired with a new high-order Eulerian method for near-body regions. This code uses the vorticity transport formulation and consists of the following key modules: convection, stretch (in 3D), diffusion, and Poisson equation solver. The code has been benchmarked successfully using various industry-standard test problems; e.g., the case of 2D lid-driven cavity flow at Re = 5,000 has shown excellent agreement with Ghia *et al.* [2] using *up to 65 times fewer degrees of freedom*. Details of this method are beyond the scope of this paper and will be presented elsewhere.

The key modules in the code take advantage of several approaches to increase performance: multithreading via OpenMP and C++11 constructs, explicit SSE/AVX vectorization using Vc [3], and OpenGL compute shaders for full platform-independent GPU acceleration. Compute-intensive portions of the code have been *accelerated by 30 to 200 times* using multithreading and vectorization vs. compiler-optimized serial versions. Furthermore, using standard OpenGL constructs available on every GPU, an *additional 7-fold performance gain* is observed, while the GUI remains responsive to the user. This accommodates Teraflop/s computing on typical modern desktops, while providing a measure of user interactivity with the code during the solution stage, which, to the best of our knowledge, is not available in any other existing CFD package.

VORTEX METHODS

The software uses a desingularized LVPM to discretize the vorticity and an augmented Boundary Element Method (BEM) to enforce boundary conditions. These methods are summarized below, but extensive background [4,5] and implementation-specific details [6,7] can be found elsewhere.

Lagrangian Vortex Particle Method

The governing equations of incompressible fluid flow in terms of the transport of vorticity are

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{\omega} = \boldsymbol{\omega}\cdot\nabla\boldsymbol{u} + \frac{1}{\mathrm{Re}}\nabla^2\boldsymbol{\omega} \tag{1}$$

$$\nabla\cdot\boldsymbol{u} = 0 \tag{2}$$

$$\nabla\times\boldsymbol{u} = \boldsymbol{\omega} \tag{3}$$

supplemented with the appropriate velocity and vorticity boundary conditions. In two dimensions, vorticity is a scalar and the stretching term is identically zero.

In the final hybrid code, the computational domain will consist of near- and off-body subdomains, which communicate their respective velocity and vorticity fields at the interface of the two subdomains using the general algorithm prescribed in previous work [8]. Vorticity transport in the near-body subdomain is discretized via a newly developed compact, high-order, discontinuous spectral difference method based on the flux reconstruction method of Huynh [9, 10]; flow in the off-body subdomain is accounted for using a LVPM. The algorithm for the high-order vorticity transport method is beyond the scope of this paper and will be presented elsewhere; the focus herein is on the LVPM.

In the LVPM, the velocity field satisfying Eqns. (2-3) and imposing the prescribed velocity boundary conditions can be formed from the superposition of several fields as

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{u}_\omega(\boldsymbol{x}) + \boldsymbol{u}_{B.C.}(\boldsymbol{x}) + \boldsymbol{U}_\infty \tag{4}$$

in which the velocity described by the continuous vorticity field

$$\boldsymbol{u}_\omega(\boldsymbol{x}) = \int_{\mathcal{D}} K(\boldsymbol{x}-\boldsymbol{x}')\times\boldsymbol{\omega}(\boldsymbol{x}')\,dV(\boldsymbol{x}') \tag{5}$$

$$K(\boldsymbol{x}) = -\frac{\boldsymbol{x}}{2^{d-1}\pi\,|\boldsymbol{x}|^d} \tag{6}$$

is discretized using N_v smooth vortex particles, assigned circulations $\boldsymbol{\Gamma}_i$, and with Green's function solution desingularized by inclusion of a smooth core radius r as

$$\boldsymbol{u}_\omega(\boldsymbol{x}_i) = \sum_{j=1}^{N_v} K^*(\boldsymbol{x}_i-\boldsymbol{x}_j, r_j)\times\boldsymbol{\Gamma}_j \tag{7}$$

$$K^*(\boldsymbol{x}, r) = -\frac{\boldsymbol{x}}{2^{d-1}\pi\,(\boldsymbol{x}^2+r^2)^{d/2}} \tag{8}$$

$$\boldsymbol{\Gamma}_i = \int_{\mathcal{D}} \boldsymbol{\omega}_i(\boldsymbol{x}')\,dV(\boldsymbol{x}') \qquad i = 1..N_v \tag{9}$$

This radius r is a multiple of the diffusion length scale; in the present work, the mean particle separation is $\Delta x = \sqrt{8\,\Delta t/\mathrm{Re}}$ and the core radii are $r = 1.5\,\Delta x$. Equation (8) is the Rosenhead-Moore [11] kernel for Biot-Savart integration, though the code supports the Winckelmans-Leonard [12] and exponential [6] (Gaussian for $d = 2$) core functions. The particle velocity gradient used in the $d = 3$ case is obtained by differentiating Eqn. (7) directly.

Boundary Element Method

The boundary velocities in Eqn. (4) can be expressed as

$$\begin{aligned}\boldsymbol{u}_{B.C.}(\boldsymbol{x}) = &\int_{\partial\mathcal{D}} K(\boldsymbol{x}-\boldsymbol{x}')\times\boldsymbol{n}(\boldsymbol{x}')\times\boldsymbol{u}(\boldsymbol{x}')\,dS(\boldsymbol{x}') \\ &-\int_{\partial\mathcal{D}} K(\boldsymbol{x}-\boldsymbol{x}')\boldsymbol{n}(\boldsymbol{x}')\cdot\boldsymbol{u}(\boldsymbol{x}')\,dS(\boldsymbol{x}')\end{aligned} \tag{10}$$

where $\boldsymbol{n}\times\boldsymbol{u}$ and $\boldsymbol{n}\cdot\boldsymbol{u}$ are the tangential and normal velocity boundary conditions, respectively.

The velocity boundary condition is imposed via the following boundary element formulation [13, 14] using a contiguous set of flat panels with piecewise-constant vortex and dilatation sheet strengths. The unknown panel strengths are found as the solution to the linear system

$$\frac{1}{2}\boldsymbol{\gamma}(\boldsymbol{x}_i) + \sum_{j=1}^{N_p} \boldsymbol{n}(\boldsymbol{x}_i)\times\boldsymbol{\gamma}(\boldsymbol{x}_j)\times\boldsymbol{I}_j(\boldsymbol{x}_i) = \boldsymbol{n}(\boldsymbol{x}_i)\times\boldsymbol{U}(\boldsymbol{x}_i) \tag{11}$$

$$\frac{1}{2}\sigma(\boldsymbol{x}_i) + \sum_{j=1}^{N_p} \sigma_j\,\boldsymbol{n}(\boldsymbol{x}_i)\cdot\boldsymbol{I}_j(\boldsymbol{x}_i) = \boldsymbol{n}(\boldsymbol{x}_i)\cdot\boldsymbol{U}(\boldsymbol{x}_i) \tag{12}$$

$$i = 1..N_p$$

where the imposed velocities are

$$\begin{aligned}\boldsymbol{U}(\boldsymbol{x}_i) = &-\boldsymbol{U}_\infty(\boldsymbol{x}_i) - \boldsymbol{u}_\omega(\boldsymbol{x}_i) \\ &+\sum_{j=1}^{N_p}\Big[\big(\boldsymbol{n}(\boldsymbol{x}_j)\times\boldsymbol{u}(\boldsymbol{x}_j)\big)\times\boldsymbol{I}_j(\boldsymbol{x}_i) + \boldsymbol{n}(\boldsymbol{x}_j)\cdot\boldsymbol{u}(\boldsymbol{x}_j)\,\boldsymbol{I}_j(\boldsymbol{x}_i)\Big]\end{aligned} \tag{13}$$

Note that when $d = 2$ there are two equations for each panel and four terms in the matrix for each panel-panel combination, while for $d = 3$, there are three equations for each panel (the vortex sheet strength is always tangential to the surface).

Because the system of Eqns. (11-12) does not emit a unique solution when $d = 2$ (where particle circulation Γ and body rotation Ω are scalar), an additional equation and unknown augment the system. The additional equation

$$\sum_{j=1}^{N_p}\Delta l_j\,\gamma_j + 2A\Omega^* = -\sum_{j=1}^{N_v}\Gamma_j \tag{14}$$

enforces zero circulation over the entire flowfield, with the body rotation rate Ω^* being the additional unknown.

The Green's function kernel for the influence of one panel on another

$$\boldsymbol{I}_j(\boldsymbol{x}_i) = \int_{S_j} K(\boldsymbol{x}_i - \boldsymbol{x}')\,dS(\boldsymbol{x}') \tag{15}$$

is evaluated analytically for $d = 2$ on a collocation point at the center of the target panel. For $d = 3$, though, this uses a dynamically recursive subpaneling integration: if the distance between the centers of the current panels or subpanels $(\boldsymbol{x}_j - \boldsymbol{x}_i)$ is under a threshold multiple of their size, each is subdivided into four subpanels and the integration repeated for each combination of subelements. This Galerkin approach greatly increases the accuracy of the scheme at a modest computational cost.

Convection and Diffusion

Given the discrete velocity and vorticity fields, Eqn. (1) is solved via a viscous splitting strategy that evaluates convection (and stretch when $d = 3$) in the Lagrangian reference frame

$$\frac{\mathrm{D}\boldsymbol{x}_i}{\mathrm{D}t} = \boldsymbol{u}(\boldsymbol{x}_i) \tag{16}$$

$$\frac{\mathrm{D}\boldsymbol{\Gamma}_i}{\mathrm{D}t} = \boldsymbol{\Gamma}_i\cdot\nabla\boldsymbol{u}(\boldsymbol{x}_i) \tag{17}$$

$$i = 1..N_v$$

where $\frac{\mathrm{D}}{\mathrm{D}t} = \frac{\partial}{\partial t} + \boldsymbol{u}\cdot\nabla$ is the material derivative, and diffusion in the Eulerian reference frame

$$\frac{\partial\boldsymbol{\Gamma}_i}{\partial t} = \frac{1}{\mathrm{Re}}\nabla^2\boldsymbol{\Gamma}_i \tag{18}$$

Convection (16) and stretch (17) are integrated in this work using a second-order Runge-Kutta method.

Diffusion (18) is evaluated using the Vorticity Redistribution Method (VRM) [15, 16]. VRM works by redistributing the vectorial circulation $\boldsymbol{\Gamma}_i$ of each diffusing particle to neighboring particles such that the moments of the diffusion equation are conserved up to arbitrary order; with new particles created only if required to satisfy the moment equations. In this work we conserve up to and including the 2nd moment. Additionally, in

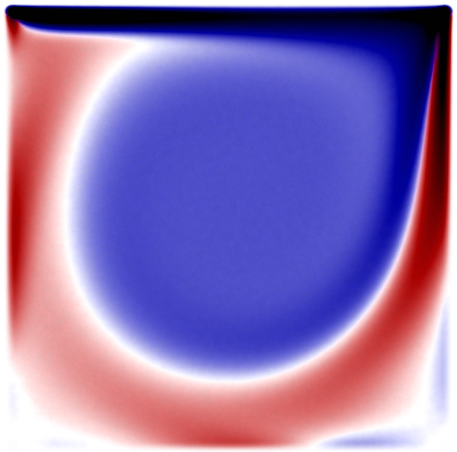

FIGURE 1. DRIVEN CAVITY, RE=1,000, VORTICITY AT $T = 50$.

order to prevent unbounded growth of N_v, VRM is applied only to particles with circulation greater than a dynamic threshold, in this case $|\mathbf{\Gamma}_i| > 10^{-4}\,|\mathbf{\Gamma}|_{\max}$.

Implementation

A solver implementing the above methods was written in C++11/14/17. It makes heavy use of the STL (C++ standard template library) and can compile on all three major desktop operating systems (Linux, Windows, macOS). All Biot-Savart summations currently use direct $O(N^2)$ summations and thus represent the bulk of the computational work required during execution. Algorithms with lower order of operations exist [17, 18] and may be implemented in the future. The matrix equation in the BEM uses a GMRES solver written with the Eigen library [19].

VALIDATION

Two canonical flows were created and run within the *Omega2D* program: a lid-driven cavity and flow over a rotating circular cylinder; and one within *Omega3D*: an impulsively started sphere. In each case, the resulting flow compares favorably with published results.

Lid Driven Cavity in 2D

A canonical test of internal flow is the lid driven cavity, which has well-known solutions at many Reynolds numbers. In this flow, a closed, unit square box with fluid at rest is driven by motion of the top lid at unit speed to the right. Here, we choose to compare with Ghia *et al.* [2] for the Re = 1,000 case, where the solution approaches a steady state.

Simulations using *Omega2D* were run to $t = 50$ using three different resolutions: $\Delta t = 0.04, 0.01, 0.0025$, and velocities along

TABLE 1. PARAMETERS AND RESULTS FOR LID DRIVEN CAVITY SIMULATIONS AT THREE RESOLUTIONS.

Parameter	Ghia *et al.*	Present method		
Δt	∞	0.04	0.01	0.0025
Δx	0.00781	0.01789	0.00894	0.00447
N_v at $t = 50$	1.6×10^4	4×10^3	1.5×10^4	6×10^4
$u_{y,min}$	-0.5155	-0.3837	-0.4747	-0.5113
x at $u_{y,min}$	0.9063	0.8817	0.8986	0.9057
$u_{x,min}$	-0.3828	-0.3033	-0.3598	-0.3810
y at $u_{x,min}$	0.1719	0.2231	0.1866	0.1758

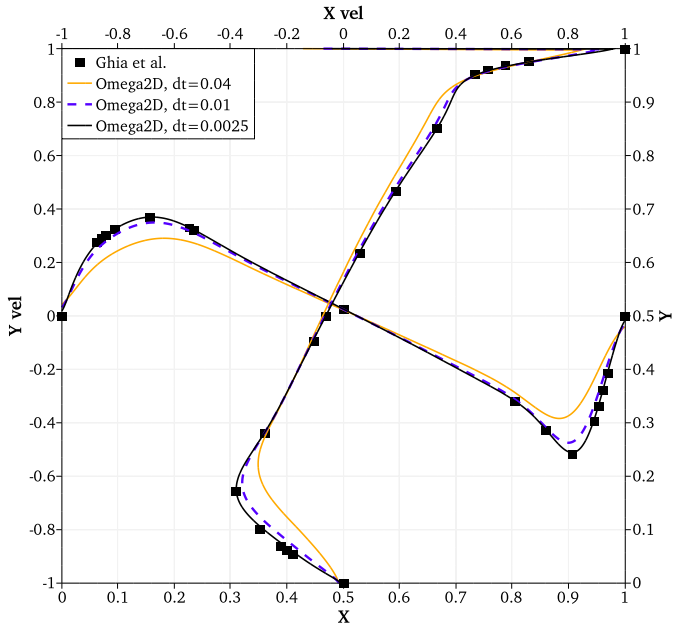

FIGURE 2. 2D DRIVEN CAVITY, RE=1,000, VERTICAL VELOCITY ALONG HORIZONTAL CENTERLINE, HORIZONTAL VELOCITY ALONG VERTICAL CENTERLINE.

the geometric centerlines in x and y exported for plotting. All simulations used the Rosenhead-Moore core function (8) and VRM diffusion used a relative particle strength of 10^{-5}. The final state of the $\Delta t = 0.0025$ run appears in Fig. 1.

Results for these three dynamic simulations appear in Table 1 and Fig. 2. The Ghia *et al.* [2] simulations were steady-state, used a uniform 129×129 point finite difference grid, and were solved with multigrid relaxation. Because of the relationship in

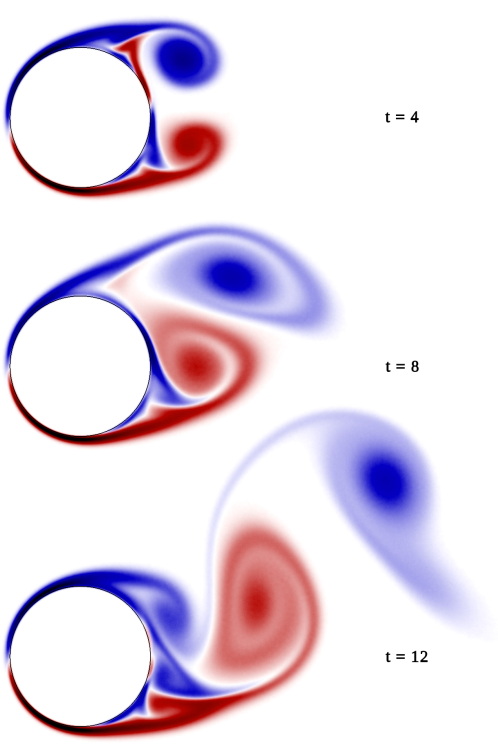

FIGURE 3. 2D FLOW OVER IMPULSIVELY STARTED ROTATING AND TRANSLATING CYLINDER, $\mathrm{RE}_D = 1{,}000$, VORTICITY AT $T = 4, 8, 12$.

the present method between Δt and Δx, the number of particles increases by four-fold as the time step is reduced by a similar factor. These results indicate that the Linf norm of the error of the present method is second order.

Rotating Circular Cylinder

To demonstrate the present method's ability to predict flow around moving and rotating geometries, we studied the flow and wake behind an impulsively started rotating and translating circular cylinder. Experimental [20] and numerical [20–22] results at a range of Reynolds numbers and nondimensional rotational velocities show a variety of wake vorticity structures, but we will focus on the case where $\mathrm{Re}_D = 1{,}000$ and the non-dimensional rotational velocity $\alpha = 0.5\,\Omega\,D\,U_\infty^{-1} = 0.5$.

The *Omega2D* run used $D = 2$, $U_\infty = 1$, $\Omega = 0.5$, $\Delta t = 0.01$, and particle nominal separation of $\Delta x = 0.01265$. The vorticity fields at various times are illustrated in Fig. 3. Figure 4 compares the streamwise velocity along a rake extending from the circle at $\theta = 90°$ (up when the freestream is to the right). Chang & Chern [21] and the present method slightly underpredict the

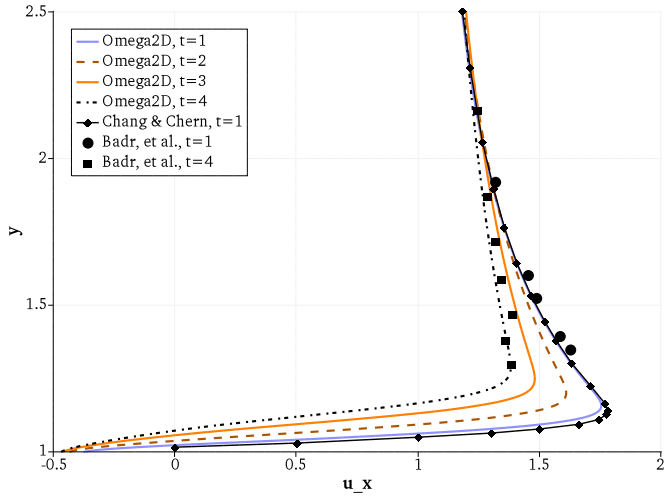

FIGURE 4. 2D FLOW OVER IMPULSIVELY STARTED ROTATING AND TRANSLATING CYLINDER, $\mathrm{RE}_D = 1{,}000$, FLOW-DIRECTION VELOCITY ALONG RADIAL LINE AT $\theta = 90°$.

streamwise velocity vs. the experiments of Badr *et al.* [20]. Note that the Chang & Chern [21] computations used a Lagrangian-Eulerian vortex method with regular remeshing, coarser circumferential resolution of $\Delta x_\theta = 0.02454D$ at the boundary, and finer innermost radial resolution of $\Delta r = 0.00447D$.

Impulsively Started Sphere

Impulsively started flow over a sphere is a well-studied problem in three-dimensional fluid dynamics, and herein we will demonstrate cases with $\mathrm{Re}_D = \{25, 40, 60, 80, 100\}$ compared to experiments from Taneda [23] and computations from Johnson & Patel [24].

The problem was set up in *Omega3D* with $D = 1$, $U_\infty = 1$, and a 1280-triangle sphere. The time steps varied by Reynolds number, with $\mathrm{Re} = 25 \rightarrow \Delta t = \frac{1}{100}$, $\mathrm{Re} = 40 \rightarrow \Delta t = \frac{1}{75}$, $\mathrm{Re} = 60 \rightarrow \Delta t = \frac{1}{60}$, and $\mathrm{Re} = 80, 100 \rightarrow \Delta t = \frac{1}{50}$. Nominal particle separation was set to $\Delta x = \sqrt{8\Delta t/\mathrm{Re}}$. The cutoff for VRM diffusion was a relative particle strength of 10^{-4}. Each case was run to $t \approx 14$, which is when the length of the recirculation bubble ceased growing appreciably; this often took the greater part of a day, and resulted in 1.3×10^6 to 2.5×10^6 particles.

Figure 5 shows the recirculation bubble length x_s/D, the angle of separation θ measured from the leading point of the sphere, and the location of the center of the recirculation vortex (x_c, y_c), where velocity vanishes with respect to the body. Notable are the present work's slight overestimation of θ and underestimation of y_c for the lowest Reynolds number tested. The difference in θ is partially a result of the resolution of the simulations: higher resolution tests push the measurement $\sim 2°$ closer. Figure 6 illustrates

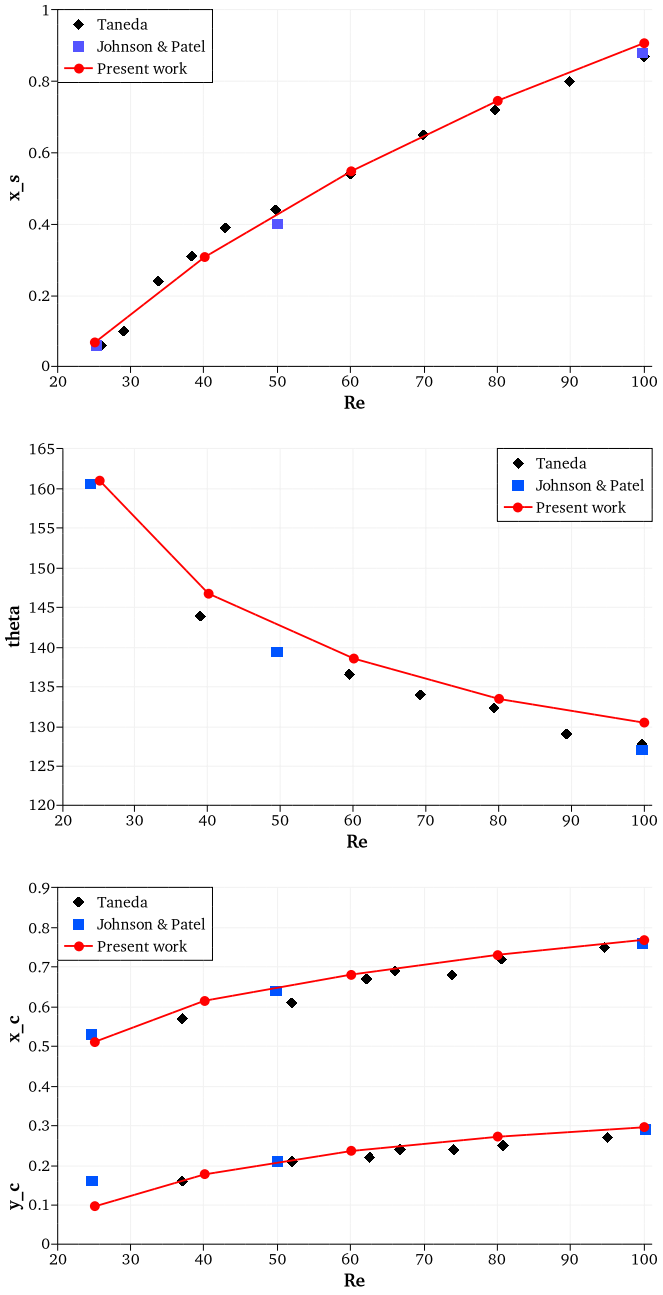

FIGURE 5. 3D FLOW OVER IMPULSIVELY STARTED SPHERE, RE_D = 25,40,60,80,100, LENGTH OF RECIRCULATION BUBBLE, SEPARATION ANGLE θ, LOCATION OF CENTER OF VORTEX.

that even the low-resolution simulations appear to match the separation point visually. Though the original image in Taneda [23] is cut off just before the end of the recirculation zone, we chose it because it was the closest match to our test cases.

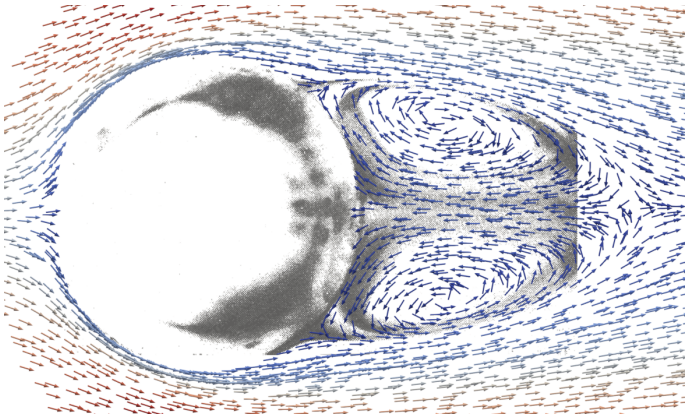

FIGURE 6. 3D FLOW OVER IMPULSIVELY STARTED SPHERE, RE_D = 100, FLOW DIRECTIONS AT T = 14, BACKGROUND IS TANEDA, RE_D = 104.

SOFTWARE DETAILS

While any piece of computational fluid dynamics software must have a solver capable of accurate results, its overall usefulness is aided by open, versatile, and easy simulation input, execution, and output. Being a new open-source project with a very small team, the *Omega2D* and *Omega3D* programs do not offer sophisticated visualization features, but, as a result of design choices and the aforementioned novel solver methods, these programs are functional and easy to use.

Lagrangian Vortex Particle Methods are one of a class of *grid-free* methods for CFD. This does not mean that they are entirely devoid of meshes, just that only solid boundaries require them: connected segments in two dimensions and triangle meshes in three. Because neither the user nor the program creates a volumetric mesh, many simulations can be set up in seconds. Additionally, because vortex methods are natively solution-adaptive, requiring only computational elements in vorticity-containing regions, many simulations (especially in 2D) will initially run in real- or near real-time.

Input is accomplished via a human-readable and editable JSON file. Most features available in the GUI are supported in the input file. Default values for every parameter mean that a simple input file need contain no more than a few lines to be useful. Geometry is defined by either a single entry with multiple parameters (such as a circular cylinder in 2D) or a single entry which includes a triangle mesh file name in one of six common formats (OBJ, STL, OFF, PLY, WRL, or MESH). Parameters governing objects' position and angular orientation can accept scalar values or formulae with a time variable, allowing prescribed kinematic motion. A JSON input file that works in *Omega2D* can be adapted to work in *Omega3D* with little to no changes. Batch (no GUI) versions of each of the two programs require a JSON file as input.

A graphical user interface (GUI) serves to ease the burden

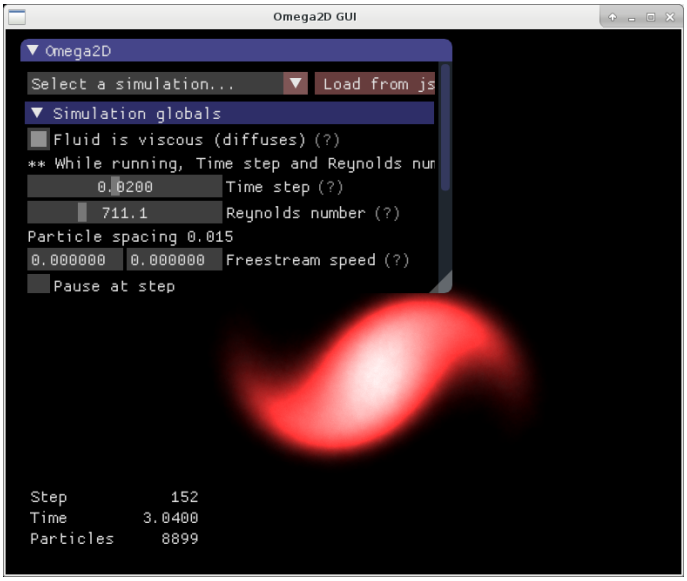

FIGURE 7. THE OMEGA2D GUI DURING A SIMULATION.

of learning and using a computer program, though it can limit experienced users' control. In the past, GUI libraries were often bloated, not portable, or not easy to integrate, and lightweight libraries were not powerful enough. Fortunately there now exist immediate-mode GUI libraries which are small and portable, yet powerful enough for reasonably-complicated applications. An *immediate-mode GUI* is one for which the entire GUI is recreated for every draw call or video output frame, simplifying user-process data management and greatly easing programmers' burden. For this suite, we use Dear ImGui [25], a simplified but extensible GUI library with a characteristic look and feel (see Fig. 7). It works with GLFW for windowing and OpenGL for rendering, both of which are cross-platform, open-source tools with wide support.

The GUI offers rudimentary visualization of the vorticity field, and the user can scroll and zoom across the simulation domain while the computation proceeds. A button writes a PNG image of the graphics window, and another will export an image for every simulation time step. Numerical data can be output at any time during the simulation, upon which the program will write a series of VTU (unstructured VTK) files, the native data format for the popular ParaView data visualization package [26].

The lifespan of any computer program is limited by the programming methodologies, paradigms, libraries, and technology that exist at the time of its writing and will inevitably succumb to changes in this software environment. This problem is exacerbated by poor initial choices and limited future support effort. In an effort to avoid this, we have decided to focus on the more-recent C++ specifications (C++11/14/17) and improved standard library, and endeavored to use well-supported, small, and few external libraries where their inclusion will ease future support needs. But more important to survival is whether the program is useful—we are optimistic that cost, ease-of-use, versatility, and performance will allow that.

TABLE 2. RESULTS FROM MULTITHREADING ON THREE CPUS. TIME SPENT EVALUATING VELOCITY AND VELOCITY GRADIENTS ON 30,000 VORTEX PARTICLES IN 3D, SPEEDUP FROM SERIAL TO OPENMP, GFLOP/S FOR OPENMP.

CPU	Cores	Serial	OpenMP	Speedup	GFlop/s
i7-7500U	2	22.92 s	8.73 s	2.6x	6.50
R7 2700X	8	17.64 s	1.43 s	12.4x	39.7
i9-7960X	16	15.81 s	0.919 s	17.2x	68.6

PERFORMANCE

Consumer-grade computation hardware (CPUs and GPUs) are capable of performing a tremendous amount of arithmetic, and compilers demonstrate increasing abilities to harness this power; but careful language, library, and algorithm selection still have a significant effect on actual numerical performance. In this section, we will review three methods to increase performance of compute-bound (as opposed to memory-bandwidth-limited) algorithms, and their effects on the present methods.

Multithreading

Modern CPUs are nearly all multicore: laptops frequently contain 4 cores, while server CPUs can have 64. OpenMP is a powerful and popular multithreading library for spreading workloads across available CPU cores, and is used as the first level of parallelization in the present work. While C++11 offers cross-platform thread creation in the standard library (and we use *std::async* to separate visualization from computation), the authors' experience with OpenMP and the simplicity with which it is able to be integrated made it the preferred multi-threading method for our computational kernels.

To test the effectiveness of OpenMP, we computed velocity and velocity gradient summations in *Omega3D* over $N_v = 30{,}000$ several times with and without OpenMP. All binaries were compiled with GCC 8.3.0 with full optimization and auto-vectorization (`-O3 -march=native`) and run on Linux. The Biot-Savart kernel requires 63 floating point operations (flops), and Table 2 reports performance in Gigaflops per second for several CPUs. Because most of the compute-intensive portions of the vortex methods calculation are trivially parallelizable, and each CPU allows more than one thread to run concurrently, we see super-linear speedup. Of particular note is the difference in

TABLE 3. RESULTS FROM VECTORIZATION ON THREE CPUS. TIME SPENT EVALUATING VELOCITY AND VELOCITY GRADIENTS ON 30,000 VORTEX PARTICLES IN 3D, SPEEDUP FROM OPENMP TO VC WITH OPENMP, GFLOP/S FOR VC WITH OPENMP.

CPU	OpenMP	OpenMP+Vc	Speedup	GFlop/s
i7-7500U	8.73 s	0.703 s	12.4x	80.6
R7 2700X	1.43 s	0.197 s	7.25x	288
i9-7960X	0.919 s	0.078 s	11.8x	811

TABLE 4. RESULTS FROM OPENGL COMPUTE SHADERS ON THREE GPUS. TIME SPENT EVALUATING VELOCITY AND VELOCITY GRADIENTS ON 300,000 VORTEX PARTICLES IN 3D, GFLOP/S ACHIEVED.

GPU	PEs	Time	GFlop/s
Intel HD 620	192	35.5 s	160.
NVIDIA GTX 980	2048	2.89 s	1964
NVIDIA GTX 1080Ti	3584	1.60 s	3931
AMD Radeon VII	3840	1.03 s	5499

multithreading ability between Intel and AMD CPUs in the test: all CPUs support running multiple threads per CPU core, but AMD's Ryzen 7 cores were more effective.

Vectorization

Each core in a contemporary CPU is able to simultaneously operate on registers containing more than one floating-point number. Extended instruction sets (SSE, AVX, etc.) trigger parallel operations on these registers, and performance-conscious programmers in C/C++ may have used assembly or intrinsic functions to access this capability. Fortunately, most current compilers are able to automatically *vectorize*: to recognize code with clear parallel patterns, and pack data and operations together into fewer instructions, though with varying degrees of effectiveness (in GCC, the `-march=native` option enables all instruction sets available on the compiling machine, and `-O3` turns on the vectorizer).

Of the available methods for explicit vectorization, the easiest and most effective was Vc [3], a C++, header-only, templated library. Vc allowed us to continue to use *std::vector* as our primary data type (using the structure-of-arrays concept). We did not have to rewrite our kernel functions at all—only make their input and output data types template parameters. And we did not have to write multiple versions of code for different vector register lengths—Vc silently chooses the widest vector type available and compiles for that.

It should be noted that vortex methods, unlike most Eulerian CFD codes, do not require double-precision calculations for accuracy. Thus, performance results presented here are typically fully single precision. Though, because the Biot-Savart integration may require the sum of a very large list of numbers, the option exists to use double-precision for only the summations.

We compared the arithmetic performance of the most compute-intensive portion of the code with and without the explicit vectorization provided by Vc. An ideal speedup for 32-bit floats is 8: each CPU supports AVX instructions and 256-bit registers, thus 8 arithmetic operations per instruction. Results from accelerating our Biot-Savart summations with both OpenMP and Vc appear in Table 3, using the same testing methods as above. Again, we see a difference between the AMD and Intel CPUs: though all CPUs use 256-bit registers and multiple floating point execution units per core, those from Intel were more effective at the present task. The **7 to 12-fold** speedup achieved using explicit vectorization implies that the compiler was unable to recognize or vectorize the inner loops by itself, which was confirmed by examining the machine code and from the compiler's vectorization reports.

Accelerator Hardware

It is well-established that graphics co-processors (GPUs) are able to perform roughly an order of magnitude more arithmetic than similarly-priced CPUs. In order to access this capability, programmers can choose from a variety of languages and libraries. Because of its stability, portability, and hardware support, we chose to accelerate our calculations using compute shaders in OpenGL. Thus, they are supported on any GPU from Intel, AMD, or NVIDIA, and computers running Linux or Windows operating systems.

Offloading of the calculations to the GPU via compute shaders is done in slivers—each GUI update (frame) is drawn after the GPU performs a small portion of the available computational work. This slivering of work is done to prevent the user from experiencing latency or lag during lengthy calculations.

To test the performance advantage gained by off-loading the Biot-Savart summations to the GPU, we ran a dynamic simulation of $N_v = 300{,}000$ particles in 3D with no diffusion (thus, the particle count did not change) and timed the duration between initially uploading the particle data to the GPU and completing the download of results back to the CPU. The corresponding wall-clock times and arithmetic performance (in Gigaflops per second) are reported in Table 4. Note that the problem size for these tests is larger than for the CPU-only tests. With this larger problem size, the computation took several frames, accrued more

overhead, and thus more accurately reflects expected performance of the application.

At the low end of performance, the laptop-grade Intel HD 620 GPU was able to only double the performance of the integrated i7-7500U CPU. The discrete, consumer-level GPUs (from AMD and NVIDIA), though, far outpaced their host CPU performance, even the highly-optimized OpenMP and Vc-accelerated builds. The NVIDIA 980 GTX was **6.8 times** faster than the AMD Ryzen 7 CPU, despite the GPU being three years older, and the AMD Radeon VII similarly performed **6.8 times** as much arithmetic as the high-end Intel i9-7960X CPU — achieving up to 5.5 TFlop/s while maintaining an interactive GUI.

CONCLUSIONS

We have presented both two- and three-dimensional, open-source, and highly-optimized computational fluid dynamics tools. An easy GUI allows novice users to set up and complete many simulations, while in the background OpenGL compute shaders accelerate the computations into the TFlop/s range. An augmented Boundary-Element Method solver allows multiple rotating and translating objects, and the software allows native output to powerful visualization tools. Many simulations can be set up in seconds, as the user is not required to design or optimize a volumetric grid, and simulation results are presented as fast as they are created, often in real time. Future versions of the software will include a novel High-Order Eulerian scheme for near-body vorticity generation while retaining the LVPM for mid- and far-field convection.

ACKNOWLEDGMENT

Research reported in this publication was supported by the National Institute Of Biomedical Imaging And Bioengineering of the National Institutes of Health under Award Number R01EB022180. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

REFERENCES

[1] Stock, M. J., and Gharakhani, A., 2020. "Omega2D: Two-Dimensional Flow Solver with GUI Using Vortex Particle and Boundary Element Methods". https://github.com/Applied-Scientific-Research/Omega2D.

[2] Ghia, U., Ghia, K., and Shin, C., 1982. "High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method". *J. Comput. Phys.,* ***48***, pp. 387–411.

[3] Kretz, M., and Lindenstruth, V., 2012. "Vc: A C++ Library for Explicit Vectorization". *Software: Practice and Experience,* ***42***(11), pp. 1409–1430.

[4] Spalart, P. R., 1988. "Vortex Methods for Separated Flows". Tech. Rep. NASA TM-100068.

[5] Cottet, G.-H., and Koumoutsakos, P., 1999. *Vortex Methods: Theory and Practice*. Cambridge Univ. Press, Cambridge, UK.

[6] Gharakhani, A., 2007. "3-D Vortex Simulation Of Accelerating Flow Over A Simplified Opening Bileaflet Valve". In Proceedings of FEDSM2007 5th Joint ASME/JSME Fluids Engineering Conference, no. FEDSM2007-37134.

[7] Stock, M. J., and Gharakhani, A., 2011. "Graphics Processing Unit-Accelerated Boundary Element Method and Vortex Particle Method". *Journal of Aerospace Computing, Information, and Communication,* ***8***(7), July, pp. 224–236.

[8] Stock, M. J., Stone, C. P., and Gharakhani, A., 2010. "Modeling Rotor Wakes with a Hybrid OVERFLOW-Vortex Method on a GPU Cluster". In 28th AIAA Applied Aerodynamics Conference, no. AIAA-2010-4553.

[9] Huynh, H. T., 2007. "A Flux Reconstruction Approach to High-Order Schemes Including Discontinuous Galerkin Methods". In 18th AIAA Computational Fluid Dynamics Conference, no. AIAA-2007-4079.

[10] Huynh, H. T., 2009. "A Reconstruction Approach to High-Order Schemes Including Discontinuous Galerkin for Diffusion". In 47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition, no. AIAA-2009-403.

[11] Moore, D., 1972. "Finite Amplitude Waves on Aircraft Trailing Vortices". *Aero. Quart.,* ***23***, pp. 307–314.

[12] Winckelmans, G., and Leonard, A., 1993. "Contributions to Vortex Particle Methods for the Computation of Three-Dimensional Incompressible Unsteady Flows". *J. Comput. Phys.,* ***109***, pp. 247–273.

[13] Wu, J., and Shankar, N., 1980. "Aerodynamic Force and Moment in Steady and Time-Dependent Viscous Flows". *AIAA Journal,* ***19***(4), p. 432.

[14] Lewis, R., 1981. "Surface Vorticity Modelling of Separated Flows from Two-Dimensional Bluff Bodies of Arbitrary Shape". *J. Mech. Eng. Sci.,* ***23***.

[15] Shankar, S., and van Dommelen, L., 1996. "A New Diffusion Procedure for Vortex Methods". *J. Comput. Phys.,* ***127***, pp. 88–109.

[16] Gharakhani, A., 2001. "Grid-Free Simulation of 3-D Vorticity Diffusion by a High-Order Vorticity Redistribution Method". In 15th AIAA Computational Fluid Dynamics Conference, no. AIAA-2001-2640.

[17] Barnes, J. E., and Hut, P., 1986. "A Hierarchical O(N log N) Force Calculation Algorithm". *Nature,* ***324***, pp. 446–449.

[18] Greengard, L., and Rokhlin, V., 1987. "A Fast Algorithm for Particle Simulations". *J. Comput. Phys.,* ***73***, pp. 325–348.

[19] Guennebaud, G., Jacob, B., et al., 2010. "Eigen v3". http://eigen.tuxfamily.org.

[20] Badr, H., Coutanceau, M., Dennis, S., and Ménard, C.,

1990. "Unsteady Flow Past a Rotating Circular Cylinder at Reynolds Numbers 10^3 and 10^4". *J. Fluid Mech.,* ***220***, pp. 459–484.

[21] Chang, C. C., and Chern, R. L., 1991. "Vortex Shedding From an Impulsively Started Rotating and Translating Circular Cylinder". *J. Fluid Mech.,* ***233***, pp. 265–298.

[22] Chew, Y., Cheng, M., and Luo, S. C., 1995. "A Numerical Study of Flow Past a Rotating Circular Cylinder using a Hybrid Vortex Scheme". *J. Fluid Mech.,* ***299***, pp. 35–71.

[23] Taneda, S., 1956. "Experimental Investigation of Wake Behind a Sphere at Low Reynolds Numbers". *J. Physical Society of Japan,* ***11***(10), pp. 1104–1108.

[24] Johnson, T., and Patel, V., 1999. "Flow Past a Sphere Up To a Reynolds Number of 300". *J. Fluid Mech.,* ***378***, pp. 19–70.

[25] Cornut, O., 2020. "Dear ImGui: Bloat-Free Immediate Mode Graphical User Interface for C++ with Minimal Dependencies". https://github.com/ocornut/imgui.

[26] Ayachit, U., 2015. *The ParaView Guide: A Parallel Visualization Application*. Kitware. ISBN: 978-1930934306.