Solution of the 2D Incompressible Navier-Stokes Equations on a Moving Voronoi Mesh

Ronald Chan, Mike Howland, Suhas Jain Suresh, and Aaron Wienkers

February 17, 2018

1 Introduction

An incompressible 2D Navier-Stokes solver was implemented on a moving Voronoi mesh grid. The implementation of general Voronoi polygons for computational fluid dynamics (CFD) algorithms has recently gained interest due to the desire to simulate flows with complex physical phenomena. The solver has been implemented on a periodic domain and has been verified using a viscous Taylor Green vortex, diffusion of a shear layer, and simulations of the development of Kelvin-Helmholtz instabilities. The control volumes (CVs) are advected with the local velocity field to utilize the advantages of a Lagrangian fluid dynamics formulation. Python and the associated scientific computing packages [3] were used for coding. The algorithm is shown to be first order convergent in space and time.

1.1 Motivation

A fundamental choice in the design of any CFD algorithm is the selection of the mesh type. Typically and historically, CFD simulations have been performed on Eulerian grids which have high numerical errors associated with regions of local complexity (i.e. near physical barriers) or in flows with large mean bulk velocities [2]. To address some of these challenges, there has been a recent push toward Lagrangian based flow solvers in which the CVs are free to advect and deform according to the flow field. As such, significant topological complexities are introduced in mixed element CV-based advection schemes. In order to reduce the topological complexity, general Voronoi polygon mesh schemes have been proposed [1, 2]. As recently described in the CTR Summer Program 2016 by Sanjeeb Bose, 3D Voronoi schemes have the potential to significantly increase the scale resolving capabilities in complex terrain simulations at high Reynolds numbers [5].

Voronoi polygons are computed on a domain of n points such that each polygon has one generating point and that each point in the associated polygon is closer to that generating point than any other [4]. The Voronoi tessellation is continuously adaptive in nature and has sharp resolution at discontinuities. There is also reduced numerical diffusion and mixing associated with the Voronoi tessellation. The points for the Voronoi tessellation may be selectively seeded near regions of physical or flow complexity in order to decrease numerical uncertainty.

2 Voronoi Mesh

Definition: Let P be the set of n sites in the domain. Then, point q lies in the cell corresponding to a site P_i if and only if

$$dist(q, P_i) < dist(q, P_j) \qquad \forall P_j \in P, j \neq i \tag{1}$$

where $dist(a, b)$ is the Euclidean distance between a and b. Voronoi diagrams can be generated in Python using the open-source library *scipy.spatial.voronoi* [6]. It takes in the locations of the sites as input parameters and generates an unbounded Voronoi diagram, consisting of infinite half-edges as shown in Figure 1. It returns the locations of the two vertices of the ridges (for finite half-edges) or one vertex of the ridge (for infinite half-edges) indexed by the ridges between two cells.

This library uses *Fortune's line sweep algorithm* to construct the Voronoi diagram, wherein a horizontal line sweeps the set of sites from the top to the bottom of the domain generating a

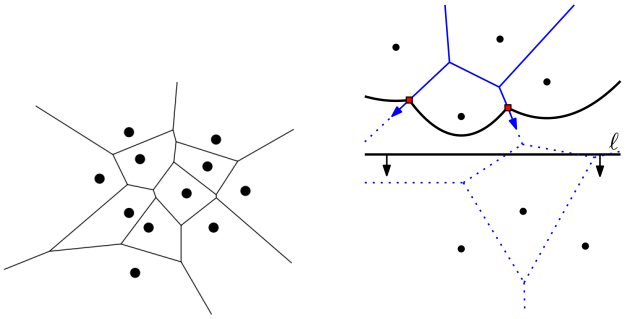

Figure 1: Schematic of a Voronoi diagram and its construction using the *line sweep algorithm*. The solid black curves on the right figure represent the *beach line*, the horizontal line represents the *sweep line* and the red squares represent the *break points*.

beach line - a minimum locus of the points equidistant from the sites above the sweep-line and the sweep-line itself as shown in Figure 1 - and the trace of *breakup points* form the ridges of the Voronoi diagram.

2.1 Properties of Voronoi Tessellations

Voronoi tessellations have some advantageous properties compared to a traditional structured mesh:

- Ridges are perpendicular to the line connecting the sites by construction. This increases the accuracy of gradient calculations and in turn the accuracy of fluxes.
- Voronoi tessellation meshing is a fully determined problem with a unique solution. Thus inverted cells are prevented and also the problem becomes easily parallelizable.
- The generation sites are *not* coincident with the barycenters of the cells. This decreases the accuracy of volume integrals when using the midpoint rule, and is necessary to take advantage of these other properties.

2.2 Mesh Types

Since the generated Voronoi diagram is by default an infinite diagram, the sites in the domain are repeated (for a periodic box) or mirrored (for a bounded domain) across all the boundaries of the domain prior to generating the mesh as shown in Figure 2.

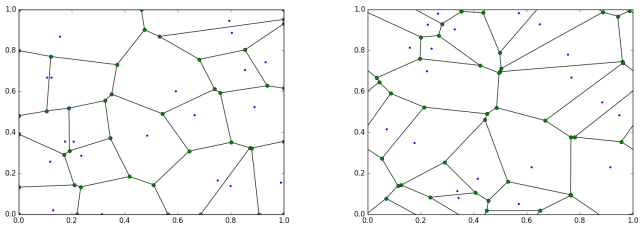

Figure 2: Illustration of bounded domain and periodic domain created by mirroring and repeating the sites outside the domain of interest.

2.3 Meshing

The `scipy.spatial.voronoi` library returns vertex locations which is sufficient to determine all the properties of the Voronoi diagram. With the vertex location outputs generated, each of the properties explicated in §A can be calculated and stored for future use in constructing the numerical

algorithms. The update of these properties is done by looping over the faces instead of the sites so that the complexity of the meshing process is O(N) as shown in Figure 3.

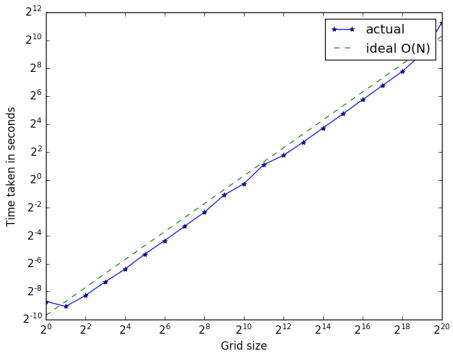

Figure 3: Complexity of meshing process estimated to be O(N) up to 1 million grid cells.

The mesh properties calculated were validated for basic mesh types such as Cartesian grids and stretched grids (shown in Figure 4) by ordering the input sites before the implementation of the actual solver for a generic Voronoi tessellation.

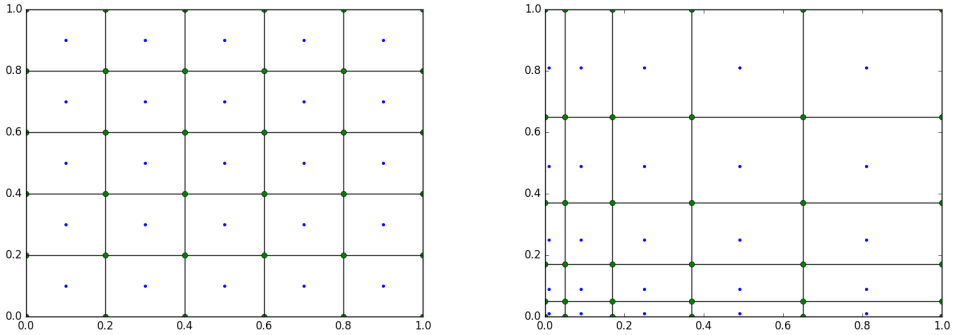

Figure 4: Cartesian grid and stretched grid generated using the `scipy.spatial.voronoi` library.

3 Numerical Methods

3.1 Control Volume Advection

The CVs are advected directly with the local cell center velocities, and marched forward in time using the first order explicit Forward Euler. The locations of the CV centers are approximated by

$$\mathbf{x_i^{n+1}} = \mathbf{x_i^n} + \mathbf{\Delta t u_i^n} \tag{2}$$

where $\mathbf{u_i}$ is the velocity and $\mathbf{x_i}$ is the corresponding location of the cell center for the i-th CV at time n. Thus the overall solution will be at most first order [1]. As a result of the direct cell-center advection, the nonlinear advection terms in the Navier-Stokes equations do not appear in the formulation, and so the Lagrangian derivatives may be evaluated directly.

3.2 Differential Operators on a Voronoi Mesh

Following the analysis in Ref. [1], discrete operators for the Laplacian, divergence, and gradient were formulated for the Voronoi problem. These operators (detailed in §A) are then used in constructing the numerical solution to the Navier-Stokes equations for the 2D system. Importantly, since the CVs are advected directly with the flow, the nonlinear advection terms may be neglected in these operators.

3.3 Navier-Stokes Solver

Python libraries are used for the sparse linear algebra solver to reduce the programming complexity of the code. The Scientific Tools package in Python (SciPy) [3] contains cheap direct and iterative sparse matrix solvers. For the matrices in this problem, the fastest routine (over CG variants or GMRES) was determined to be the direct solver `scipy.sparse.linalg.solve`. This algorithm rearranges the matrix into a nearly-banded form in order to back-solve quickly. Sparse matrix-matrix products were also computed with SciPy, although data was stored and computed using *numpy*, the scientific computing package in Python [3]. The solver algorithms can be seen in §A.

4 Code Validation

4.1 Parameters

The physical solution involves knowledge of the kinematic viscosity ν. In addition, the length of the domain is given by L in meters. The Reynolds number is then given by $\text{Re} = \frac{U}{\nu}$ where we set the reference U to be 1 m/s and the reference length to be 1 m. Other numerical parameters include the time t in seconds, the number of points along a direction N (generating $N_{CV} = N^2$ CVs and a characteristic grid size Δx_0), and the time step Δt in seconds.

4.2 Timing

One main detraction from any moving mesh numerical method is the relatively large expense of remeshing after every timestep compared to that time actually solving the PDE of interest. The scaling of this additional expense is investigated by solving the Taylor-Green Vortex test problem using the backward Euler solution scheme described in §3. Figure 5 shows the relative meshing and solving times scale proportionally to each other even for many degrees of freedom. It should be noted that other more extreme or less well-behaved tests will spend significantly more time solving than meshing, but still achieve similar scaling with the problem size, and so the two times remain within an order or magnitude of each other.

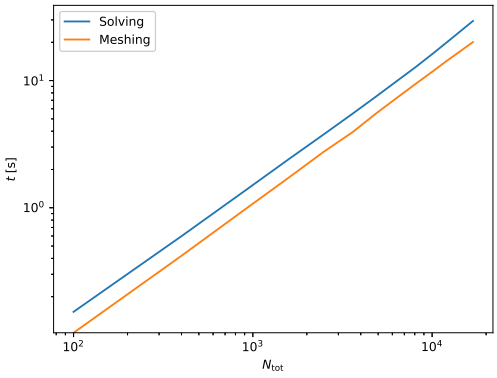

Figure 5: The relative time spent solving and meshing in simulating the Taylor-Green vortex problem for increasing problem size.

4.3 Convergence

4.3.1 Viscous Taylor-Green Vortex

The canonical test case of the Taylor-Green vortex with a known, analytical solution was used for the validation and convergence characterization of the code. The Taylor-Green (TG) vortex was simulated on a periodic domain $x, y \in [0, 2\pi]$ with $\text{Re} = 6$ & 600. Four vortices will be present in this domain, centered at $[x, y] = ([\pi/2, \pi/2], [3\pi/2, \pi/2], [\pi/2, 3\pi/2], [3\pi/2, 3\pi/2])$. It is important to keep the Reynolds number for the TG vortex low since the turbulent energy cascade can manifest

due to the inherent instabilities present in vortices. The initial conditions for the simulation are

$$\begin{aligned} u(x,y) &= \sin(x)\cos(y) \\ v(x,y) &= \cos(x)\sin(y) \end{aligned} \tag{3}$$

with the analytical solution

$$\begin{aligned} u(x,y) &= \sin(x)\cos(y)\ e^{-2\nu t} \\ v(x,y) &= \cos(x)\sin(y)\ e^{-2\nu t} \\ P(x,y) &= \frac{\rho}{4}(\cos(2x)+\cos(2y))e^{-4\nu t} \end{aligned} \tag{4}$$

The TG vortex was used to characterize the spatial (discretization) and temporal (N-S solver) errors in the code. For the purposes of the error convergence, a weighted L_2 norm was used, computed as

$$E_2 = \frac{\sqrt{\sum_i^{N_{CV}} A_i (u_i^* - u_i)^2}}{\sqrt{\sum_i^{N_{CV}} A_i u_i^2}} \tag{5}$$

where A_i is the area of the CV, u_i^* is the analytical solution at the associated CV, and u_i is the numerical solution at the same CV. The analytical velocities were computed at the final CV center locations upon the completion of the simulation. The CVs are initially randomly seeded in the domain.

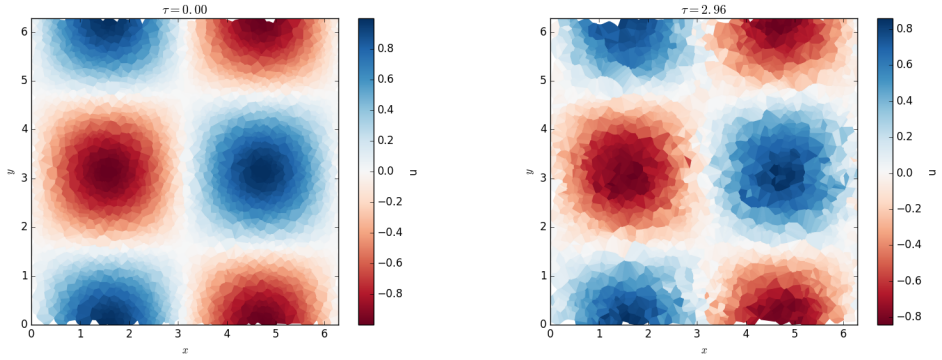

Figure 6: Example snapshots at $t = 0$ and $t = 2.96$ showing the deformation of the Voronoi CVs with $N_{CV} = 2500$. Contour colored with u velocity.

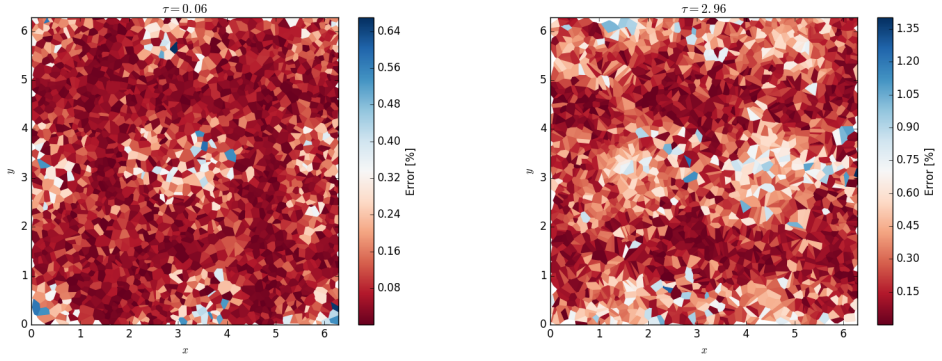

Figure 7: Example snapshots at $t = 0.06$ and $t = 2.96$ showing the error contributions per CV in the domain with $N_{CV} = 2500$. The error is represented as a percentage error in the given CV with respect to the overall area in the domain, such that each CV is colored with their contributions to the total error in order to display regions where the local error is pronounced.

The error shown in Figure 7 is shown to be increased near the locations of high magnitude velocities, occurring directly in the middle of two vortex centers. The bulk velocity in this simulation

is not significantly large to have Voronoi advantage. Errors are introduced at the locations where the CVs are significantly deforming and large velocity gradients may be present across the large deformed CV arising from the fact that the barycenters is not coincident with the cell sites.

4.3.2 Spatial Convergence

In order to characterize the discretization convergence, the TG vortex was simulated at Re = 6 for a range of N_{CV} with a fixed time step. The time step was chosen to be sufficiently small such that the discretization error is not contaminated with the solver error ($\Delta t << 1/\max(N_{CV})$). The discretization convergence can be seen in Figure 8. The spatial order of accuracy is indeed first order, as specified in Ref. [1].

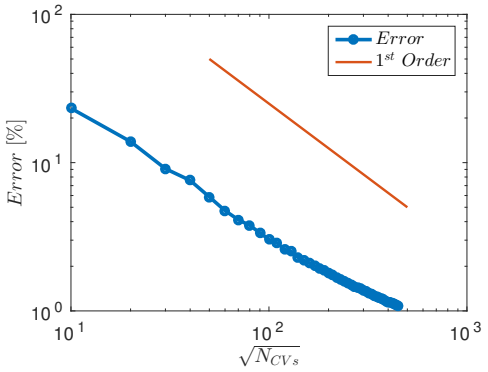

Figure 8: Spatial discretization error computed with Eq. 5 showing first order convergence.

4.3.3 Temporal Convergence

In order to characterize the temporal convergence, the TG vortex was simulated at Re = 6 for a range of time-step sizes Δt with fixed N_{CV} and for a fixed number of steps. In order to capture only the temporal convergence of the solver, N_{CV} must be selected such that $\min(\Delta t) << 1/N_{CV}$. The solver convergence was computed for backward Euler and Crank-Nicolson solver types. As mentioned in Ref. [1] the temporal convergence is first order. This was found to be independent of the solver type, because the Lagrangian cell advection scheme is still first order. Therefore, the use of the cheaper backward Euler scheme over the more expensive semi-implicit Crank-Nicolson is justified. The convergence results are summarized in Figure 9.

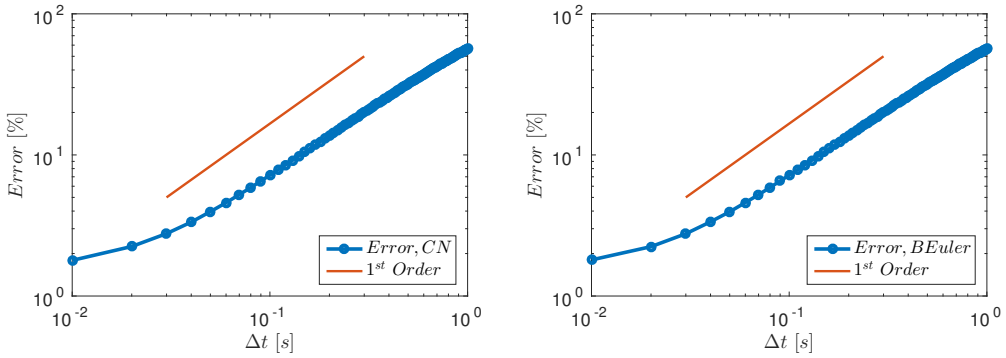

Figure 9: Solver temporal error computed with Eq. 5 showing first order convergence above a threshold Δt.

Notably, as $\Delta t \to 0$, the solver convergence is no longer first order. This result is likely due to the introduction of spatial errors contaminating the first order convergence of the solver. The temporal error can be approximated as

$$\epsilon_\tau = \frac{-\Delta t}{2}\left(\frac{\partial^2 \phi}{\partial t^2}\right)_j^n - v\frac{\Delta x^2}{6}\left(\frac{\partial^3 \phi}{\partial x^3}\right)_j^n + \mathcal{O}\left[(\Delta x)^4, (\Delta t)^4\right]. \tag{6}$$

Equation 6 carries dependence on Δx as well as Δt. As $\Delta t \to 0$, the Δx error will dominate.

4.3.4 Number of Faces

In addition to the mesh advection scheme, another factor which makes the Voronoi tessellation a more expensive mesh to use in the numerical algorithms is the larger number of faces per cell on average. Compared to a Cartesian mesh with 4 faces, or a tetrahedral mesh with 3 faces per cell, the average number of faces of a randomly distributed Voronoi mesh is nearly constant at 6. Alone, this is already at least a 50% increase in operations per time step. The evolution of the maximum number of faces per cell (and so the maximum number of elements in each discrete operator) is shown in Figure 10. Although there is a trend of an increasing maximum number of faces with the degrees of freedom of the system, the distribution of N_{face} (as is the mean) is nearly invariant of the problem size.

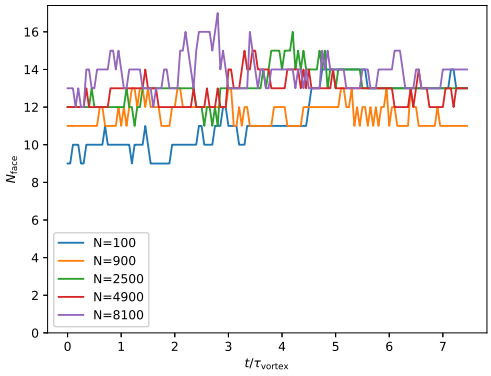

Figure 10: The evolution of the *maximum* number of faces a single cell contains as the simulation progresses, and for different resolutions.

4.4 Diffusion of Shear Layer

The diffusion of a shear layer was also investigated to verify the numerical properties of the code and to validate the code by comparison with expected physical scalings.

In this validation, we devised a numerical setup with periodic boundaries where a forward-moving (towards positive x) layer is sandwiched between two backward-moving (towards negative x) regions. This was accomplished through the imposition of the following initial velocity profile

$$u(y) = \frac{1}{2}\left\{\tanh\left[200\left(y+\frac{1}{6}\right)\right] - \tanh\left[200\left(y-\frac{1}{6}\right)\right] - 1\right\}, \tag{7}$$

where the origin of the coordinate system is located at the bottom-left corner of the domain and $y \in [0, L]$. Figure 11 illustrates this setup for $\text{Re} = 10$, $L = 2$, $\frac{\Delta x_0}{L} \approx 6.7\times10^{-3}$ and $\Delta t \approx 3.3\times10^{-3}$ with snapshots of the flow at the beginning of the simulation and after some time has elapsed. We see that the shear layer indeed diffuses outwards after some time as expected.

Figure 12 describes the evolution of the total momentum and energy in the system over time. The first-order accurate advection of the Voronoi cell centers incurs some errors in the time-advancement of the velocities, which manifest themselves in the conservation of the total momentum. However, we see that the energy of the system decreases over time, reflecting the numerical dissipation inherent in the developed scheme, as well as indicating the stability of the scheme.

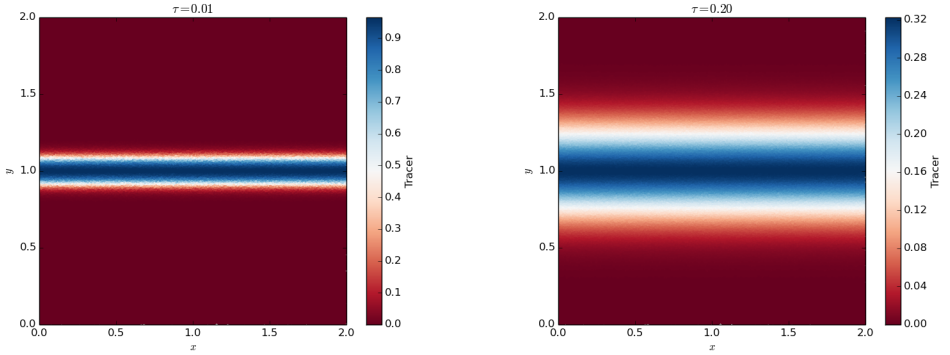

Figure 11: Concentration of tracer particles initially confined in the forward-moving (central) section of a shear layer at (left) $t = 0.01$ and (right) $t = 0.20$. The shear layer in the middle diffuses over time towards the top and bottom regions of the domain, which are initially traveling backwards.

The periodic boundary conditions allow us to approximate the shear layer as infinite in length. Although the Reynolds number of the system is not exceeding low, we can, to some approximation, analyze the system in a manner similar to Stokes' first problem for the flow in a semi-infinite region above an impulsively driven flat plate, which yields a growth rate of $t^{0.5}$. (This approximation is justified for small times $t \ll 1$ when the elapsed time is much smaller than the characteristic inertial time scale.) Figure 13 plots the growth of the shear layer in our simulation, which yields a reasonably close growth rate of $t^{0.57}$. This disagreement is attributed to the finite vertical bounds of the domain which, for this periodic domain increases the momentum transfer.

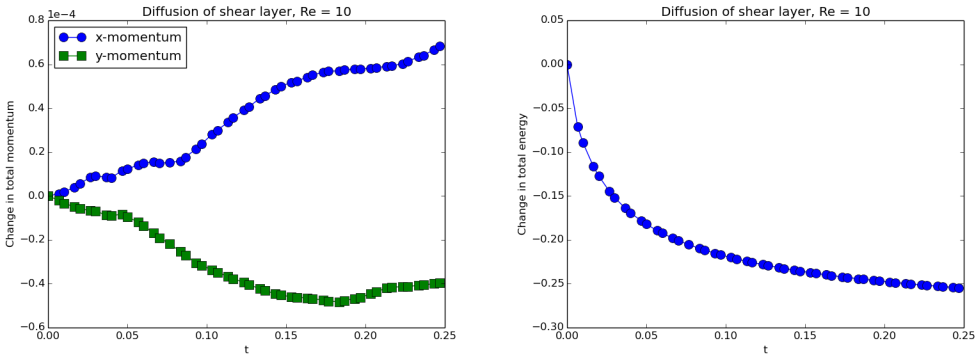

Figure 12: Evolution of momentum (left) and energy (right) in the computational domain over time.

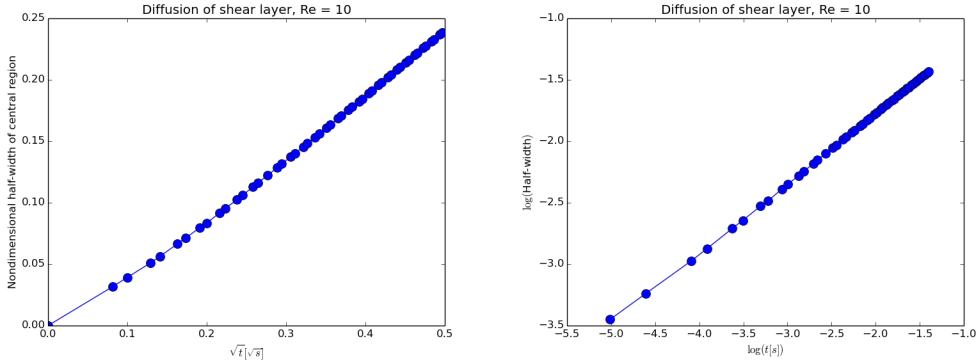

Figure 13: Evolution of the width of the shear layer over time in linear (left) and logarithmic (right) axes.

4.5 Kelvin-Helmholtz Instability

The final test case considered in validating this implementation is the Kelvin-Helmholtz shear layer instability. The problem set-up is physically similar to that of Section 4.4 but with a much higher Reynolds number. With little diffusion, the vortex sheet initialised at the shear layer interfaces rolls up into the prototypical vortex rolls reminiscent of a breaking wave. Linear instability analysis for a shear layer with constant density gives an ideal growth rate of

$$\sigma = \frac{1}{2} k \Delta U \tag{8}$$

for small amplitude perturbations. Thus as expected, small scale perturbations on the interface grow the fastest (in the absence of surface tension).

An initial $n = 2$ mode perturbation, $v' = \sin(kx)$, was given to the background shear with $v = \pm 1/2$ to seed the instability so that the ideal growth rate is $\sigma = 2\pi$. The ensuing evolution after 0.85 e-folding times is displayed in Figure 14, showing the tracer concentration field which (solely for visualization purposes) was allowed to diffuse at the same rate as momentum (ν).

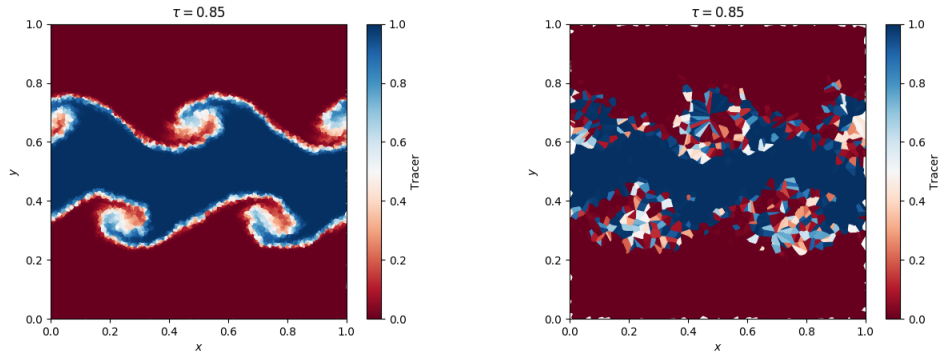

Figure 14: Tracer concentration showing the evolution of the Kelvin-Helmholtz instability for $\mathrm{Re} = 10^3$ of an initial discontinuous shear band, shown at $t = 0.85$ e-folding times for $N_{CV} = 150^2$ cells (left) and $N_{CV} = 50^2$ cells (right).

The Lagrangian nature of this technique helps to greatly reduce artificial diffusion of the solution, allowing better representation of sharp discontinuities and interfaces. This means the stream-wise velocity discontinuity is more faithfully captured, and consequently so is the shear instability and growth rate.

To achieve the ideal growth rate, the damping rate (due to numerical diffusivity) must be much smaller than the instability growth rate, σ. This is shown to be the case for the moving Voronoi mesh even with as few as $N_{CV} = 50^2$ cells, as seen in Figure 15. The initial disagreement before $t \sim 0.1$ is attributed to the phase adjustment of the two shear layers. The initial perturbation independent of y is not actually the eigenfunction of this modified shear-band Kelvin-Helmholtz instability. This is apparent from the staggered rolls between the top and bottom layers in Figure 14. Finally, saturation occurs near an amplitude of ~ 1.5 due to nonlinear effects which are not modeled by the linear stability analysis. After saturation, and with such low resolution, stable compact vortices remain which diffuse over much longer times that the simulation.

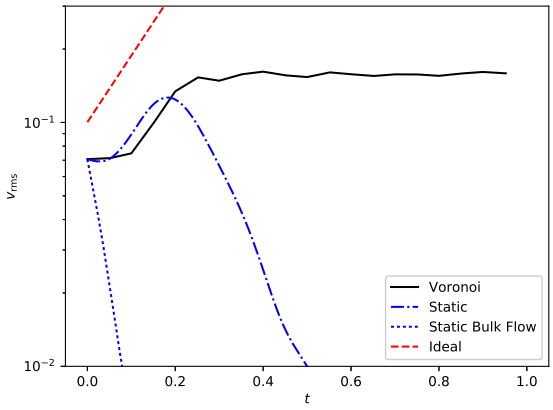

Figure 15: Vertical velocity amplitude growth using $N_{CV} = 50^2$ cells, showing agreement with the ideal Kelvin-Helmholtz growth rate. This is juxtaposed with the evolution using a static Cartesian grid, which in extreme cases produces an artificially stable solution.

This test was also conducted with an arbitrary Galilean boost and achieves identical results. Static-mesh codes, however, are *not* invariant to inertial boosts, and results in an increased truncation error scaling with the boost velocity. This is clearly the case from Figure 15. Although the instability is similarly captured on a static Cartesian mesh, when a mean bulk flow, $v_b = 100$ is added to the initial condition, the numerical diffusion overwhelms the ability of the shear layer to become unstable, and immediately decays. Thus a static-mesh scheme has difficulty resolving small fluctuations, resulting in an artificially stable system (producing results similar to Section 4.4). Example snapshots of the static mesh codes for $N_{CV} = 32^2, 64^2$ are reproduced in §A.

5 Conclusions

In this work we have designed, implemented, and validated a 2D incompressible Navier-Stokes solver on a moving Voronoi mesh in Python. The scheme has been shown formally to be first order, validated with an analytical viscous Taylor-Green Vortex test case. The scheme has also been validated with a viscous shear layer and Kelvin-Helmholtz instabilities.

The implementation of this Voronoi scheme have revealed advantages over Cartesian grids in: (1) better effective resolution due to Lagrangian nature and (2) Galilean invariance. These advantages are seen specifically with the K-H instability validation test case, where the introduction of a bulk velocity in the base flow on a coarse mesh eliminates the capabilities for a Cartesian grid based code to resolve the instabilities through the growth rate metric. Conversely, meshing increases the cost by about 2x, and the additional faces by another 2x. Additionally, extending this moving Voronoi mesh method to an explicit or local Riemann-based schemes would allow for easy iteration on the cells and parallelization. However, careful consideration of the stable CFL number for the variable Δx moving mesh is needed if utilizing an explicit scheme.

Future work would include the formulation of Dirichlet boundary conditions in order to allow for the formulation of complex geometries and immersed boundaries.

A Appendix

A.1 Voronoi Mesh Properties

- N_neighbors - a list that stores the number of neighbors of each cell.
- neighbors - a list of lists that stores the site indices of the neighbors of each cell.
- length - a list of lists that stores the vectors to the neighbor site locations relative to its own.
- face - a list of lists that stores the lengths of the ridges between the cell and its neighbors.
- face_center - a list of lists that stores the vectors to the centers of the ridge locations relative to the location of the site.
- grad_area - a list of lists that stores the values of the quantity $\partial A_i/\partial X_j$.
- grad_area_t - a list of lists that stores the values of the quantity $\partial A_j/\partial X_i$.
- is_boundary - a list that stores True if the cell is a boundary cell, and otherwise stores False.
- boundary - a list of lists that stores "Internal","East","West","North", or "South" depending on the location of the ridge.

A.2 Differential Operators on a Voronoi Mesh

Laplacian L^n:

$$\nabla^2\varphi \approx \frac{1}{A}\iint \nabla\cdot\nabla\varphi \,\mathrm{d}A \tag{9}$$

$$\approx L^n\varphi \equiv \frac{1}{A_i}\sum_{j=0}^{N_{\mathrm{b}i}-1}\left(f_{i,j}\frac{\varphi_j-\varphi_i}{\|\mathbf{l}_{i,j}\|}\right), \tag{10}$$

Divergence D^n:

$$\nabla\cdot\mathbf{u} = \nabla\cdot\mathbf{u}_{\mathrm{surf}} \approx \frac{1}{A}\iint\frac{DA}{Dt}\,\mathrm{d}A = \frac{1}{A}\iint\mathbf{u}\cdot\nabla A\,\mathrm{d}A \tag{11}$$

$$\approx D^n\mathbf{u} \equiv \frac{1}{A_i}\sum_{j=0}^{N_{\mathrm{b}i}-1}\left(\mathbf{u}_j\cdot\frac{\partial A_i}{\partial\mathbf{X}_j}\right) \tag{12}$$

$$= \frac{1}{A_i}\sum_{j=0}^{N_{\mathrm{b}i}-1}\left(\mathbf{u}_j\cdot\frac{f_{i,j}}{2}\left[\frac{\mathbf{l}_{i,j}}{\|\mathbf{l}_{i,j}\|}+\frac{\mathbf{l}_{i,j}-2\mathbf{c}_{i,j}}{\|\mathbf{l}_{i,j}\|}\right]\right), \tag{13}$$

$$\tag{14}$$

Gradient G^n:

$$\nabla\varphi \approx \frac{1}{A}\iint\frac{\partial\varphi}{\partial\mathbf{x}}\,\mathrm{d}A \tag{15}$$

$$\approx G^n\varphi \equiv \frac{1}{A_i}\sum_{j=0}^{N_{\mathrm{b}i}-1}\left(-\varphi_j\frac{\partial A_j}{\partial\mathbf{X}_i}\right) \tag{16}$$

$$= \frac{1}{A_i}\sum_{j=0}^{N_{\mathrm{b}i}-1}\left(\varphi_j\frac{f_{i,j}}{2}\left[\frac{\mathbf{l}_{i,j}}{\|\mathbf{l}_{i,j}\|}-\frac{\mathbf{l}_{i,j}-2\mathbf{c}_{i,j}}{\|\mathbf{l}_{i,j}\|}\right]\right). \tag{17}$$

A.3 Solvers

A.3.1 Backward Euler Solver

$$\begin{aligned}(I-\nu\Delta t L^n)\,u_i^* &= u_i^n\\ D^nG^n\Delta P &= D^nu_i^*\\ u_i^{n+1} &= u_i^* - G^n\Delta P\end{aligned} \tag{18}$$

A.3.2 Crank-Nicolson Solver

$$
\begin{aligned}
\left(I - \nu \Delta t L^n\right) u_i^* &= \left(I + \nu \Delta t L^n\right) - \frac{1}{2} \Delta t G^n P^n \\
u_i^{**} &= u_i^* + \frac{1}{2} \Delta t G^n P^n \\
\Delta t D^n G^n P^{n+1} &= 2 D^n u_i^{**} \\
u_i^{n+1} &= u_i^{**} - \frac{1}{2} \Delta t G^n P^{n+1}
\end{aligned}
\tag{19}
$$

A.4 Kelvin-Helmholtz Instability Static Code Validation

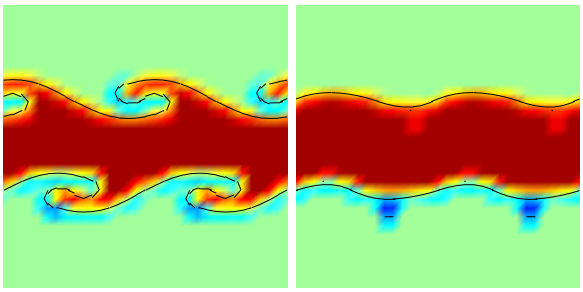

Figure 16: Tracer concentration showing the evolution of the Kelvin-Helmholtz instability for $\mathrm{Re} = 10^3$ of an initial discontinuous shear band, shown at $\tau = 0.85$ e-folding times for $N_{CV} = 32^2$ on a uniform Cartesian grid without a bulk translational velocity $u_b = 0$ (left) and with bulk translational velocity $u_b = 100$ (right).

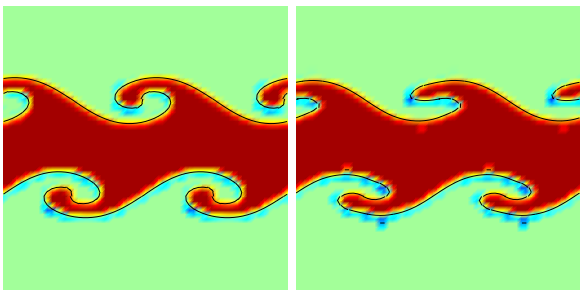

Figure 17: Tracer concentration showing the evolution of the Kelvin-Helmholtz instability for $\mathrm{Re} = 10^3$ of an initial discontinuous shear band, shown at $\tau = 0.85$ e-folding times for $N = 64^2$ on a uniform cartesian grid without a bulk translational velocity $u_b = 0$ (left) and with translational bulk velocity $u_b = 100$ (right).

References

[1] Börgers, C. and Peskin, C. S. "A Lagrangian method based on the Voronoi diagram for the incompressible Navier Stokes equations on a periodic domain." The Free-Lagrange method; Proceedings of the First International Conference, Hilton Head Island, SC, March 4-6, 1985.

[2] Springel, Volker. "Hydrodynamic simulations on a moving Voronoi mesh." arXiv preprint arXiv:1109.2218 (2011).

[3] Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/ [Online; accessed 2017-03-23].

[4] Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites.. Journal für die reine und angewandte Mathematik (Crelle's Journal), 1908(133), doi:10.1515/crll.1908.133.97

[5] Bose, Sanjeeb: Large Eddy Simulation for Design. CTR Summer Program 2016. Stanford, CA.

[6] https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Voronoi.html.