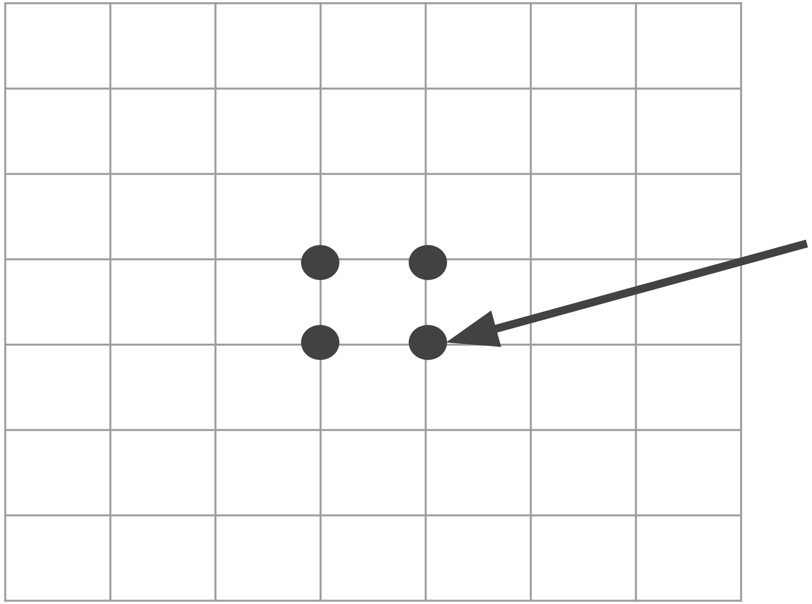
18847 Presentation

Michael Lee, Anna Paek, Siqi (Edward) Guo

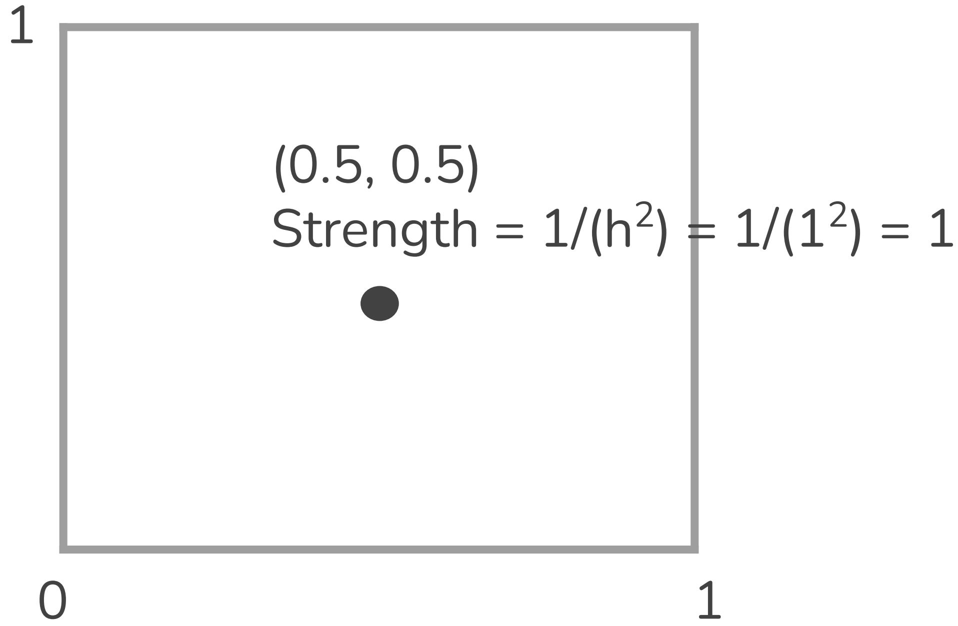
Analogy

Start with a Grid



Each grid point (corners) stores information on the fluid motion at that specific location

What are Particles?



How do we swirl the particles?

1. **Particles create swirls**
   a. Each particle has a **swirl strength (vorticity)** which causes the fluid around it to spin
2. **Swirl becomes the fluid**
   a. We **spread** each particle's swirl to nearby grid points, creating a smooth vorticity field onto a grid
3. **Swirl becomes motion**
   a. We **solve a math equation** (using FFT) → **stream function** that shows how the fluid flows
   b. Compute the velocity at each grid point
4. **Particles ride the flow**
   a. Each particle checks the velocity at its exact position (using interpolation)
   b. We move the particle in that direction

Deposition– Creating the Vorticity Field

- Particle's **vorticity ($\boldsymbol{\omega}_k$), index ($\boldsymbol{k}$), position ($\boldsymbol{x}^k$),**
- Spread the particle's strength to the 4 nearby grid points
  - Closer grid points get more of the swirl (via bilinear weights)
- **Grid vorticity field $\boldsymbol{\omega}^g$, grid spacing $\boldsymbol{h}$**

$$i^k = \left\lfloor \frac{x^k}{h} \right\rfloor$$

Bottom-left grid index

$$s^k = \frac{x^k - i^k h}{h}$$

Normalized offset

$$\omega^g_{i^k} += \omega^k (1 - s^k_0)(1 - s^k_1)$$

Bottom-left corner

$$\omega^g_{i^k+(1,0)} += \omega^k s^k_0 (1 - s^k_1)$$

Bottom-right corner

$$\omega^g_{i^k+(0,1)} += \omega^k (1 - s^k_0) s^k_1$$

Top-left corner

$$\omega^g_{i^k+(1,1)} += \omega^k s^k_0 s^k_1$$

Top-right corner

Convolution– Turning Swirl into Flow

- Computing the fluid flow caused by the swirling
  - Stream function via convolution: each point on the grid is influenced by swirl from all other points
- Green's function ***G*** to describe how one point's swirl affects other
- Convolution is computed efficiently using Hockney's FFT method

Vorticity at point *j*

$$\psi_i = \sum_{j \in \mathbb{Z}^2} G(i - j)\, \omega_j^g$$

Stream function at grid point ***i***

Influence kernel (Green's function)

Computing Grid Velocities

- Compute the velocity components by taking the derivative of the stream function
    - Use central differences to approximate the derivatives numerically
    - Grid spacing ***h***, index of current grid point ***i***
- (0, 1): one grid step up (in y-direction)
- (1, 0): one grid step right (in x-direction)

$$\vec{U}^g_{\boldsymbol{i}} = \left(\frac{\psi_{\boldsymbol{i}+(0,1)} - \psi_{\boldsymbol{i}-(0,1)}}{2h}, -\frac{\psi_{\boldsymbol{i}+(1,0)} - \psi_{\boldsymbol{i}-(1,0)}}{2h} \right)$$

Interpolation - Grid to Particles

$$\vec{U}_k = \sum_{i \in \mathbb{Z}^2} \vec{U}_i \Psi(x_k - ih)$$

This equation shows how we calculate each particle's velocity by sampling from the entire velocity grid. We use Ψ as our interpolation function that determines how much each grid point contributes based on distance

$$i_k = \left\lfloor \frac{x_k}{h} \right\rfloor, \quad s_k = \frac{x_k - i_k h}{h}$$

We first find the nearest bottom-left grid point (ik) and calculate how far the particle is from this point as a normalized offset (sk)

We then apply bilinear interpolation using the four nearest grid points. Each corner's contribution is weighted by proximity

The interpolation of the velocity field is done as follows:

$$\vec{U}_k = \vec{U}_g^i(1 - s_k^0)(1 - s_k^1) + \vec{U}_g^{i+(1,0)} s_k^0 (1 - s_k^1) + \vec{U}_g^{i+(0,1)}(1 - s_k^0) s_k^1 + \vec{U}_g^{i+(1,1)} s_k^0 s_k^1$$

RK4 Time Integration

$$k_1 = \Delta t F(t^n, X^n)$$

$$k_2 = \Delta t F(t^n + \frac{1}{2}\Delta t, X^n + \frac{1}{2}k_1)$$

$$k_3 = \Delta t F(t^n + \frac{1}{2}\Delta t, X^n + \frac{1}{2}k_2)$$

$$k_4 = \Delta t F(t^n + \Delta t, X^n + k_3)$$

$$X^{n+1} = X^n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

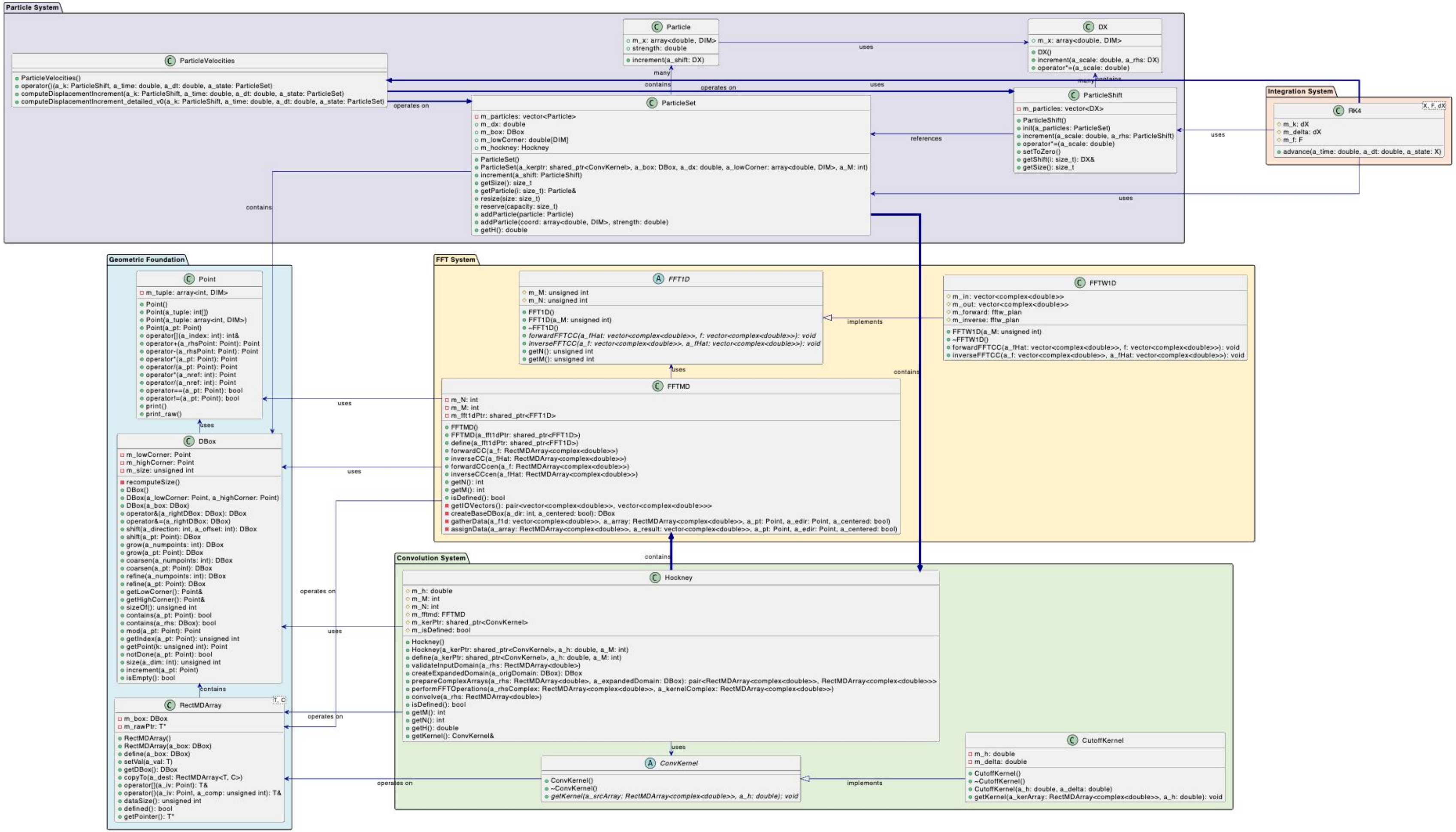
4th-order Runge-Kutta method to accurately update particle positions

RK4 computes four intermediate estimates (k_1, k_2, k_3, k_4) of particle velocities:

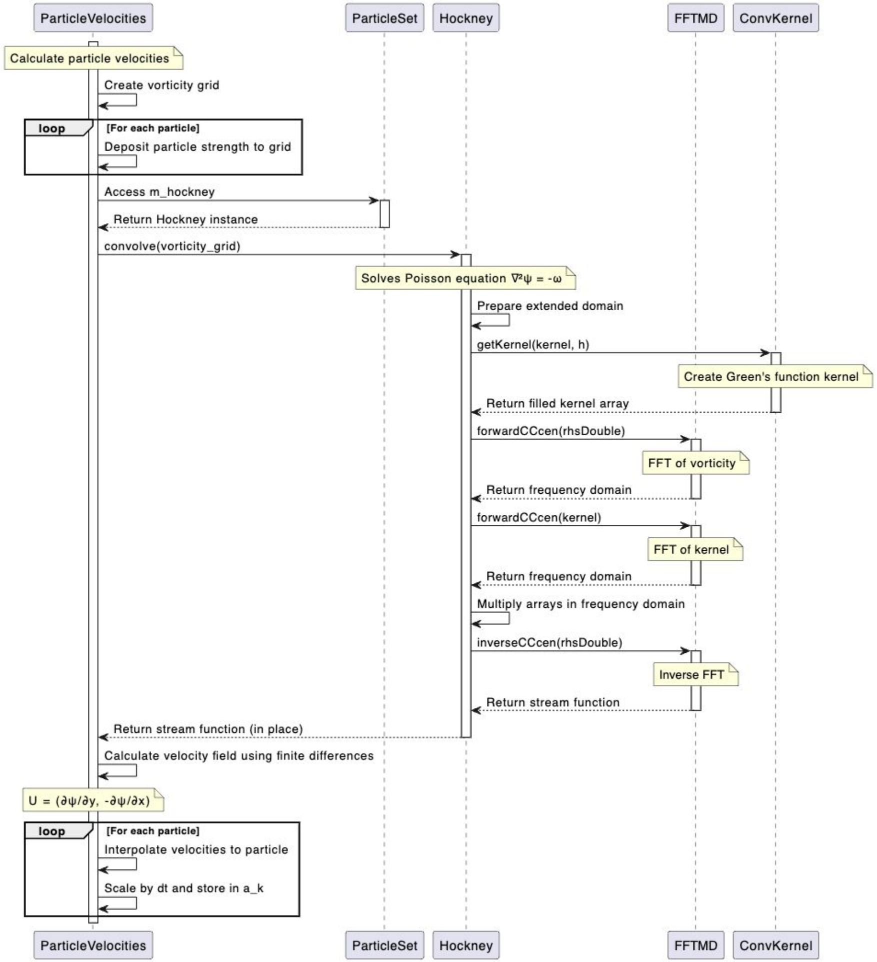
- k_1: Velocity at current position
- k_2: Velocity at position halfway moved by k_1
- k_3: Velocity at position halfway moved by k_2
- k_4: Velocity at position fully moved by k_3

Final update combines all four estimates with appropriate weights

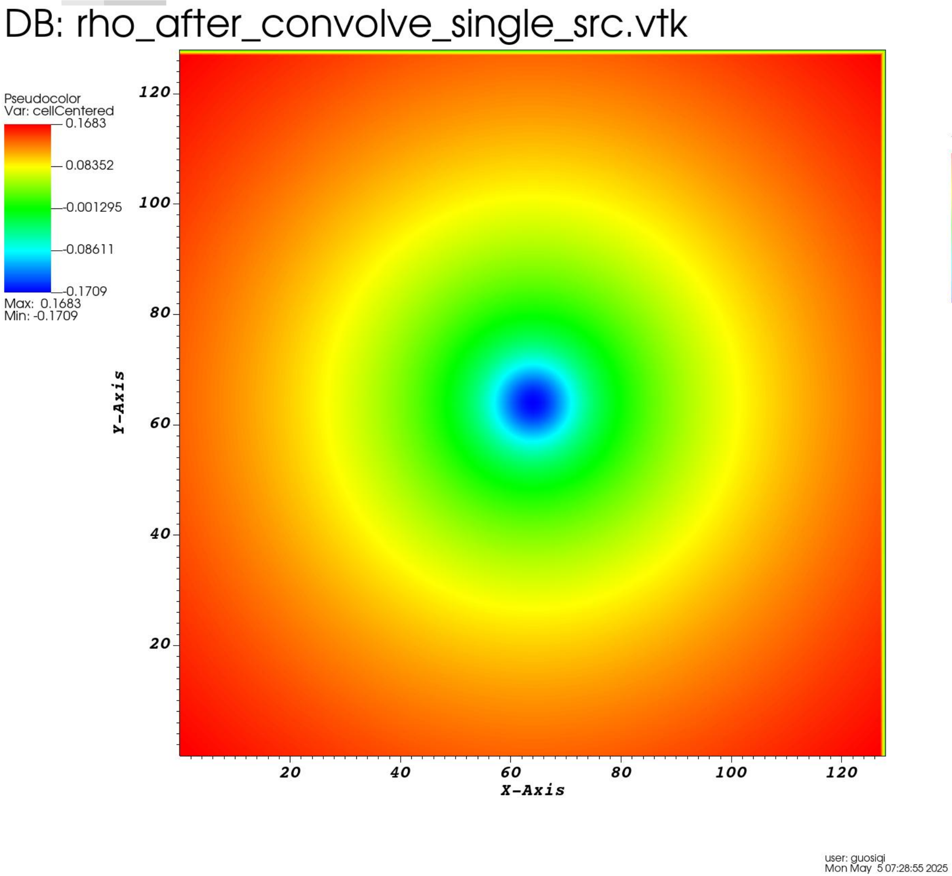
System UML

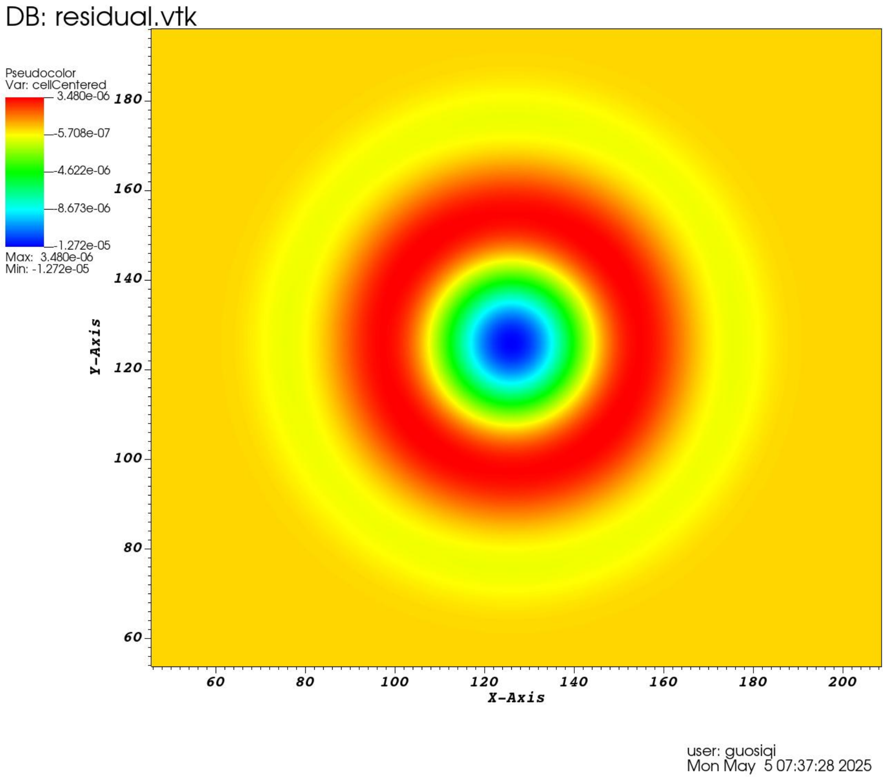

Particle System
Particle
m_x: array<double, DIM>
strength: double
increment(a_shift: DX)
DX
m_x: array<double, DIM>
DX()
increment(a_scale: double, a_rhs: DX)
operator*=(a_scale: double)
uses
ParticleVelocities
ParticleVelocities()
operator()(a_k: ParticleShift, a_time: double, a_dt: double, a_state: ParticleSet)
computeDisplacementIncrement(a_k: ParticleShift, a_time: double, a_dt: double, a_state: ParticleSet)
computeDisplacementIncrement_detailed_v0(a_k: ParticleShift, a_time: double, a_dt: double, a_state: ParticleSet)
operates on
many
contains
operates on
uses
many
contains
ParticleSet
m_particles: vector<Particle>
m_dx: double
m_box: DBox
m_lowCorner: double[DIM]
m_hockney: Hockney
ParticleSet()
ParticleSet(a_kerptr: shared_ptr<ConvKernel>, a_box: DBox, a_dx: double, a_lowCorner: array<double, DIM>, a_M: int)
increment(a_shift: ParticleShift)
getSize(): size_t
getParticle(i: size_t): Particle&
resize(size: size_t)
reserve(capacity: size_t)
addParticle(particle: Particle)
addParticle(coord: array<double, DIM>, strength: double)
getH(): double
ParticleShift
m_particles: vector<DX>
ParticleShift()
init(a_particles: ParticleSet)
increment(a_scale: double, a_rhs: ParticleShift)
operator*=(a_scale: double)
setToZero()
getShift(i: size_t): DX&
getSize(): size_t
references
uses
uses
contains
Integration System
RK4
X, F, dX
m_k: dX
m_delta: dX
m_f: F
advance(a_time: double, a_dt: double, a_state: X)
Geometric Foundation
Point
m_tuple: array<int, DIM>
Point()
Point(a_tuple: int[])
Point(a_tuple: array<int, DIM>)
Point(a_pt: Point)
operator[](a_index: int): int&
operator+(a_rhsPoint: Point): Point
operator-(a_rhsPoint: Point): Point
operator*(a_pt: Point): Point
operator/(a_pt: Point): Point
operator*(a_nref: int): Point
operator/(a_nref: int): Point
operator==(a_pt: Point): bool
operator!=(a_pt: Point): bool
print()
print_raw()
uses
DBox
m_lowCorner: Point
m_highCorner: Point
m_size: unsigned int
recomputeSize()
DBox()
DBox(a_lowCorner: Point, a_highCorner: Point)
DBox(a_box: DBox)
operator&(a_rightDBox: DBox): DBox
operator&=(a_rightDBox: DBox)
shift(a_direction: int, a_offset: int): DBox
shift(a_pt: Point): DBox
grow(a_numpoints: int): DBox
grow(a_pt: Point): DBox
coarsen(a_numpoints: int): DBox
coarsen(a_pt: Point): DBox
refine(a_numpoints: int): DBox
refine(a_pt: Point): DBox
getLowCorner(): Point&
getHighCorner(): Point&
sizeOf(): unsigned int
contains(a_pt: Point): bool
contains(a_rhs: DBox): bool
mod(a_pt: Point): Point
getIndex(a_pt: Point): unsigned int
getPoint(k: unsigned int): Point
notDone(a_pt: Point): bool
size(a_dim: int): unsigned int
increment(a_pt: Point)
isEmpty(): bool
contains
RectMDArray
T, C
m_box: DBox
m_rawPtr: T*
RectMDArray()
RectMDArray(a_box: DBox)
define(a_box: DBox)
setVal(a_val: T)
getDBox(): DBox
copyTo(a_dest: RectMDArray<T, C>)
operator[](a_iv: Point): T&
operator()(a_iv: Point, a_comp: unsigned int): T&
dataSize(): unsigned int
defined(): bool
getPointer(): T*
FFT System
FFT1D
m_M: unsigned int
m_N: unsigned int
FFT1D()
FFT1D(a_M: unsigned int)
~FFT1D()
forwardFFTCC(a_fHat: vector<complex<double>>, f: vector<complex<double>>): void
inverseFFTCC(a_f: vector<complex<double>>, a_fHat: vector<complex<double>>): void
getN(): unsigned int
getM(): unsigned int
implements
FFTW1D
m_in: vector<complex<double>>
m_out: vector<complex<double>>
m_forward: fftw_plan
m_inverse: fftw_plan
FFTW1D(a_M: unsigned int)
~FFTW1D()
forwardFFTCC(a_fHat: vector<complex<double>>, f: vector<complex<double>>): void
inverseFFTCC(a_f: vector<complex<double>>, a_fHat: vector<complex<double>>): void
uses
contains
FFTMD
m_N: int
m_M: int
m_fft1dPtr: shared_ptr<FFT1D>
FFTMD()
FFTMD(a_fft1dPtr: shared_ptr<FFT1D>)
define(a_fft1dPtr: shared_ptr<FFT1D>)
forwardCC(a_f: RectMDArray<complex<double>>)
inverseCC(a_fHat: RectMDArray<complex<double>>)
forwardCCcen(a_f: RectMDArray<complex<double>>)
inverseCCcen(a_fHat: RectMDArray<complex<double>>)
getN(): int
getM(): int
isDefined(): bool
getIOVectors(): pair<vector<complex<double>>, vector<complex<double>>>
createBaseDBox(a_dir: int, a_centered: bool): DBox
gatherData(a_f1d: vector<complex<double>>, a_array: RectMDArray<complex<double>>, a_pt: Point, a_edir: Point, a_centered: bool)
assignData(a_array: RectMDArray<complex<double>>, a_result: vector<complex<double>>, a_pt: Point, a_edir: Point, a_centered: bool)
uses
uses
operates on
uses
operates on
contains
Convolution System
Hockney
m_h: double
m_M: int
m_N: int
m_fftmd: FFTMD
m_kerPtr: shared_ptr<ConvKernel>
m_isDefined: bool
Hockney()
Hockney(a_kerPtr: shared_ptr<ConvKernel>, a_h: double, a_M: int)
define(a_kerPtr: shared_ptr<ConvKernel>, a_h: double, a_M: int)
validateInputDomain(a_rhs: RectMDArray<double>)
createExpandedDomain(a_origDomain: DBox): DBox
prepareComplexArrays(a_rhs: RectMDArray<double>, a_expandedDomain: DBox): pair<RectMDArray<complex<double>>, RectMDArray<complex<double>>>
performFFTOperations(a_rhsComplex: RectMDArray<complex<double>>, a_kernelComplex: RectMDArray<complex<double>>)
convolve(a_rhs: RectMDArray<double>)
isDefined(): bool
getM(): int
getN(): int
getH(): double
getKernel(): ConvKernel&
uses
ConvKernel
ConvKernel()
~ConvKernel()
getKernel(a_srcArray: RectMDArray<complex<double>>, a_h: double): void
implements
operates on
CutoffKernel
m_h: double
m_delta: double
CutoffKernel()
~CutoffKernel()
CutoffKernel(a_h: double, a_delta: double)
getKernel(a_kerArray: RectMDArray<complex<double>>, a_h: double): void

Particle Velocity Workflow

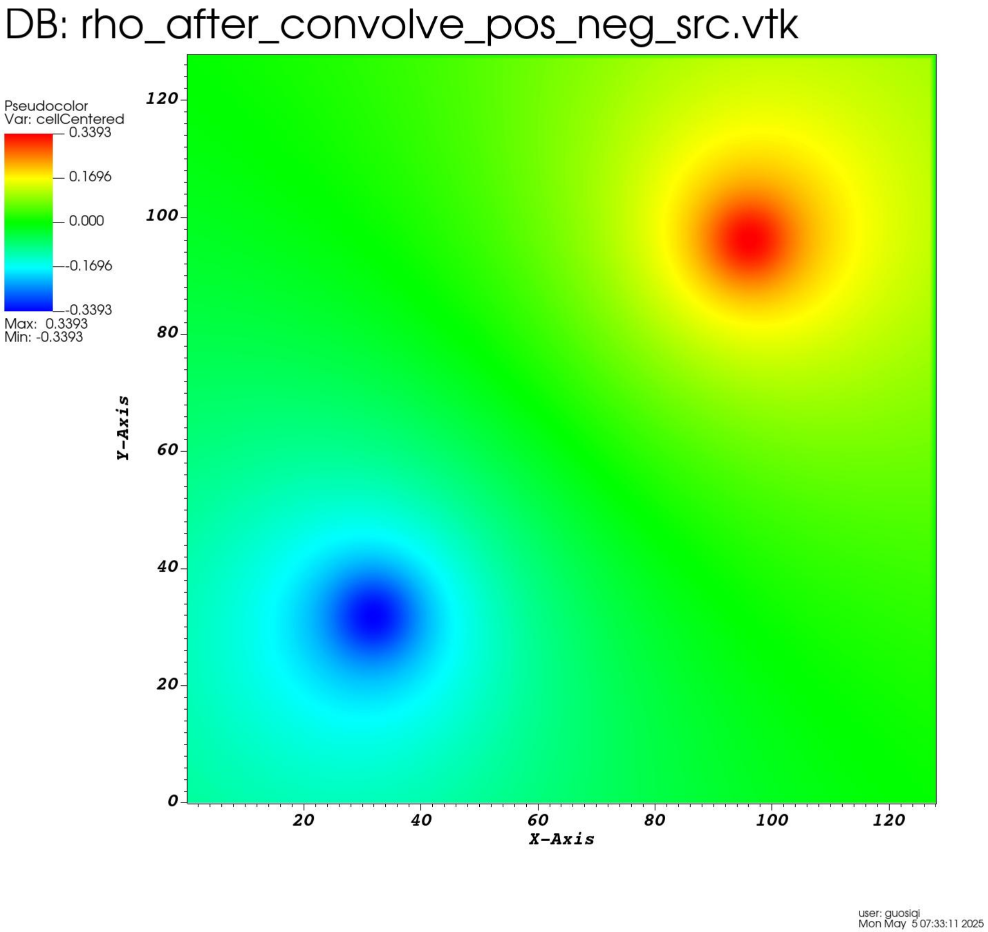


Hockney Test 1: Single Source Point

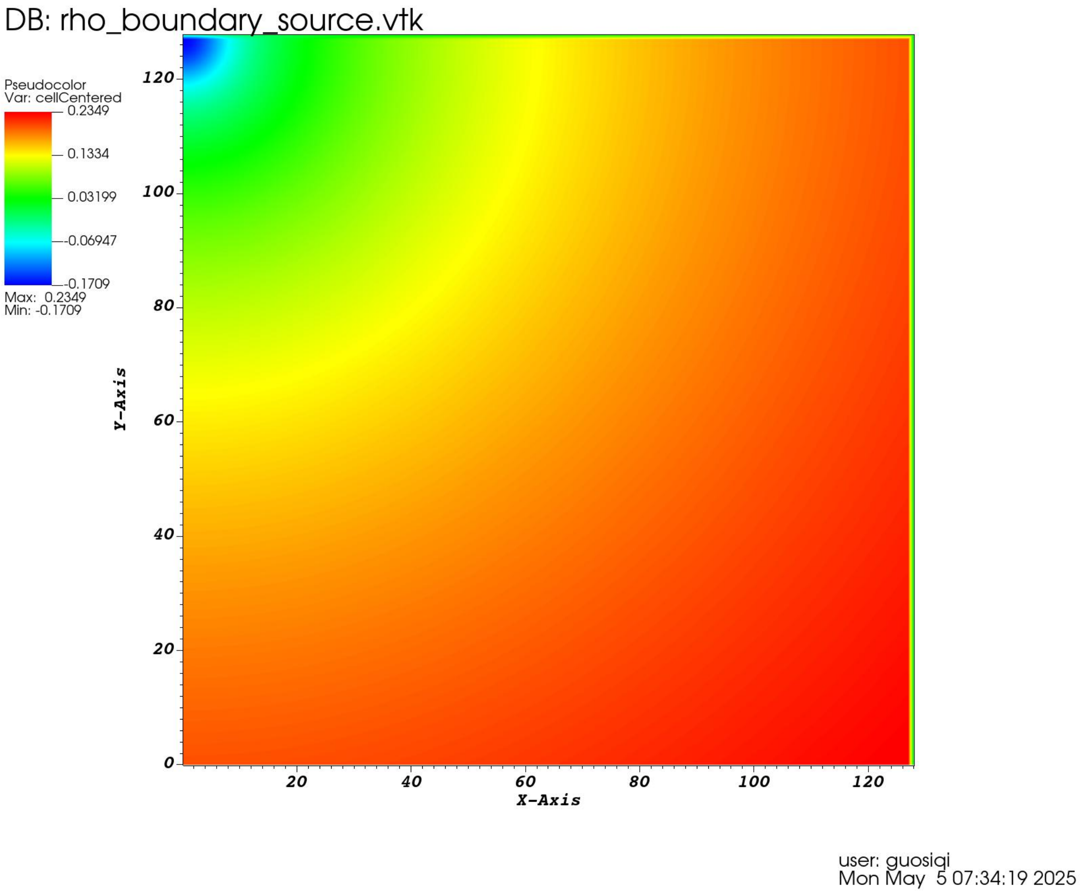




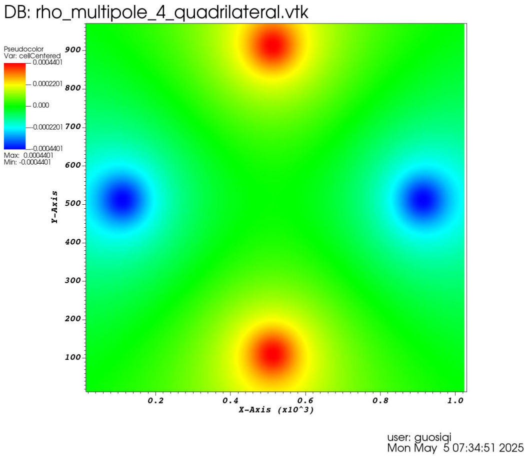
Hockney Test 2: Two Source Points

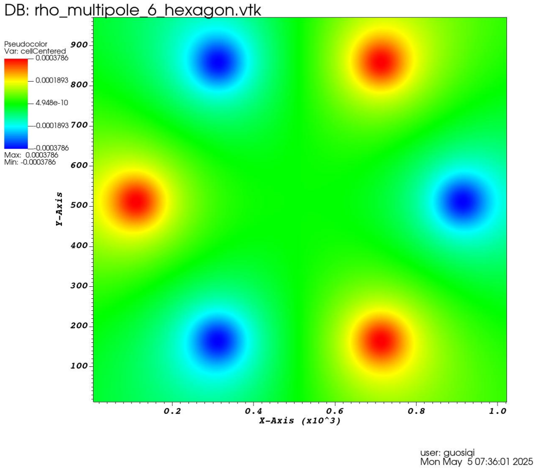


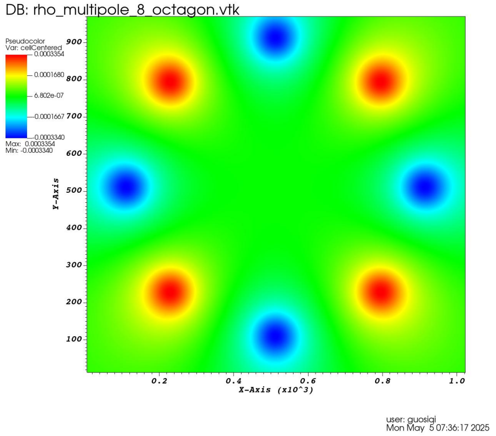
Hockney Test 3: Boundary Source Point

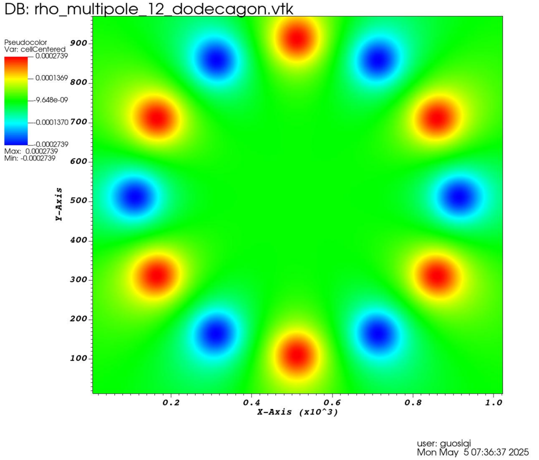


Hockney Test 4: Cancel out in Polygon

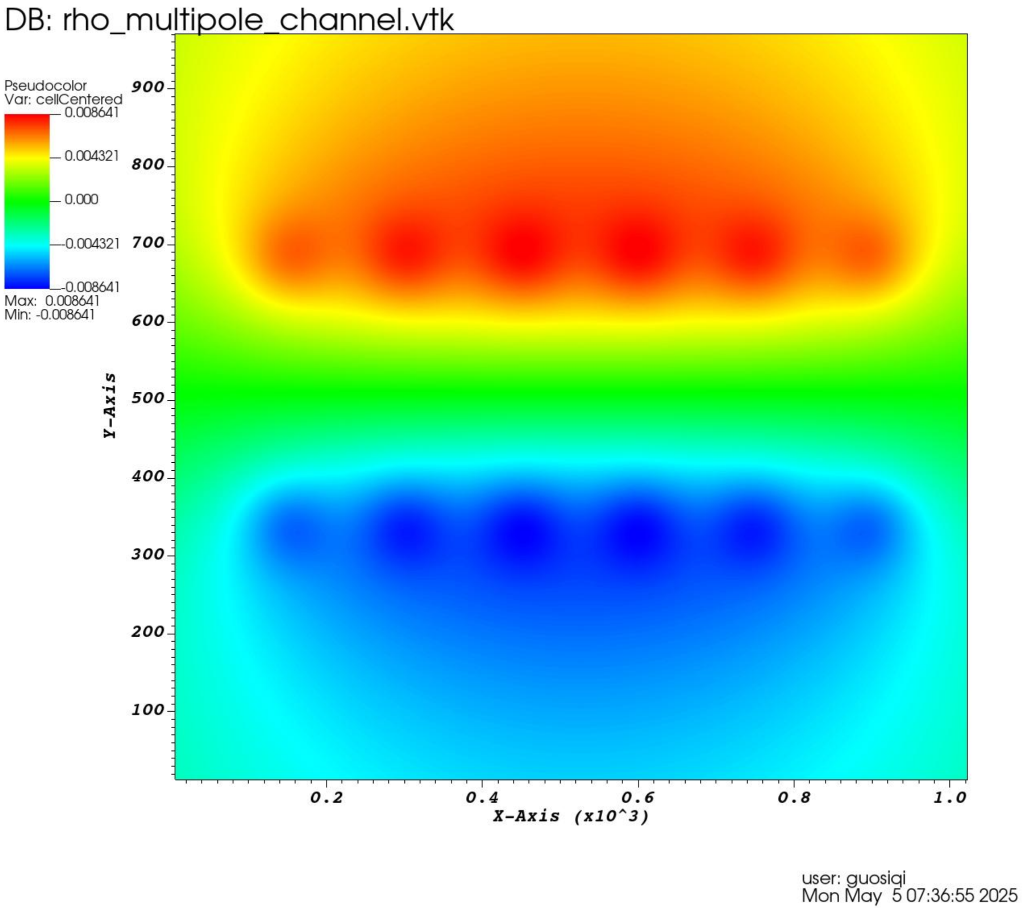








Hockney Test 5: Channel



Vortex Test 1: Single Particle

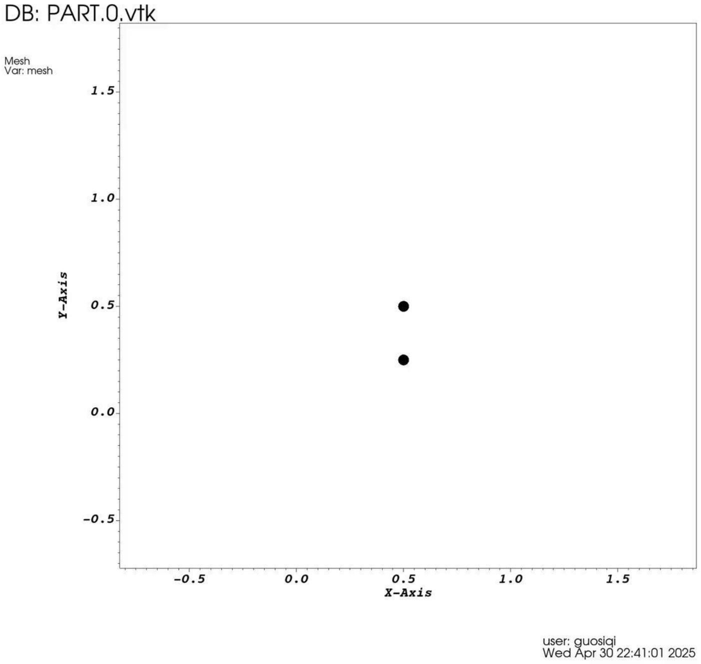
Requirements: A single particle with strength $1/h^2$ positioned at (0.5, 0.25), Grid size N = 64 (M = 6) Time integration to t = 10

Expected Behavior: The particle should remain stationary (zero displacement). This is because the velocity field induced by a vortex particle on itself should be zero

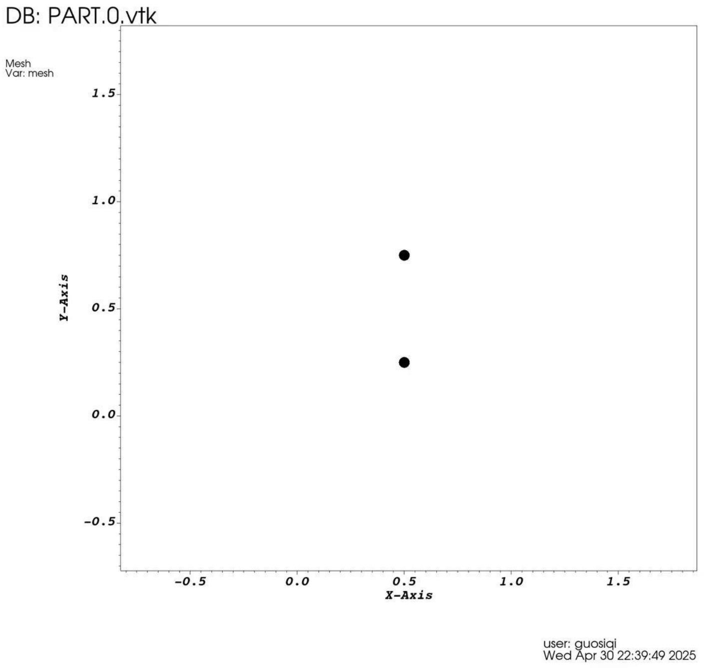
```
input log_2(number of grid points)
6
input test = 1,2, other
1
input particle refinement factor
1
enter stopping time
10
number of particles per cell = 1
number of particles = 1
ParticleShift initialized.
ParticleVelocities initialized.
First outField done
m_delta and m_k initialized
DEBUG: Particle positions at time 0:
  Particle 0: (0.5, 0.25) with shift (0, 0)
```

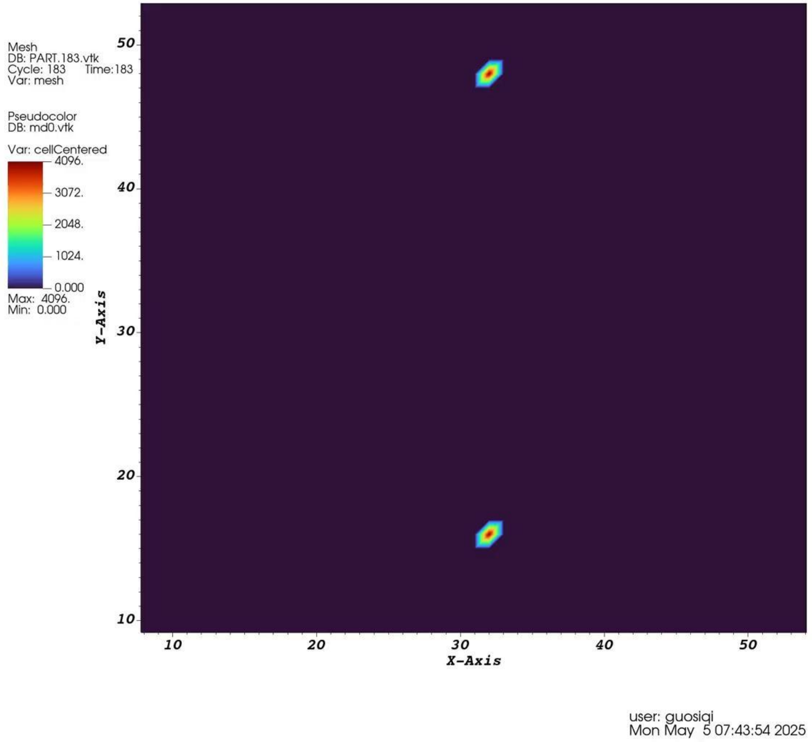
```
DEBUG: Particle positions at time 9.98047:
  Particle 0: (0.5, 0.25) with shift (4.68918e-17, -1.68045e-18)
Particle velocities computation started.
Deposited vorticity onto grid.
Deposited vorticity to grid.
Convolved vorticity grid.
Velocity field computed.
Computed velocity field.
Interpolated velocities to particles.
Interpolated velocities to particles.
Particle velocities computed.
k3 done
DEBUG: Particle positions at time 10.0078:
  Particle 0: (0.5, 0.25) with shift (-9.8825e-17, -9.71445e-17)
Particle velocities computation started.
Deposited vorticity onto grid.
Deposited vorticity to grid.
Convolved vorticity grid.
Velocity field computed.
Computed velocity field.
Interpolated velocities to particles.
Interpolated velocities to particles.
Particle velocities computed.
k4 done
time = 10.0078  dt 0.0546875
```

Test 2: One Strong, One Zero-Strength Particle

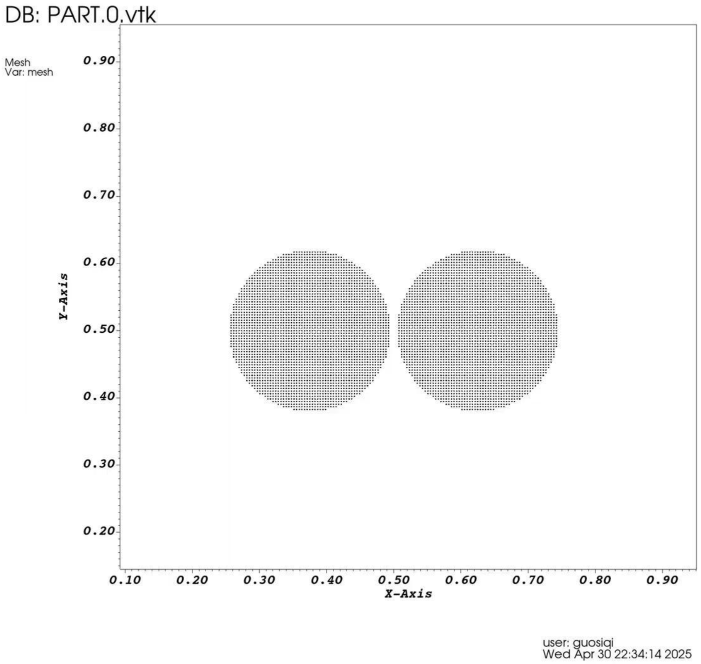


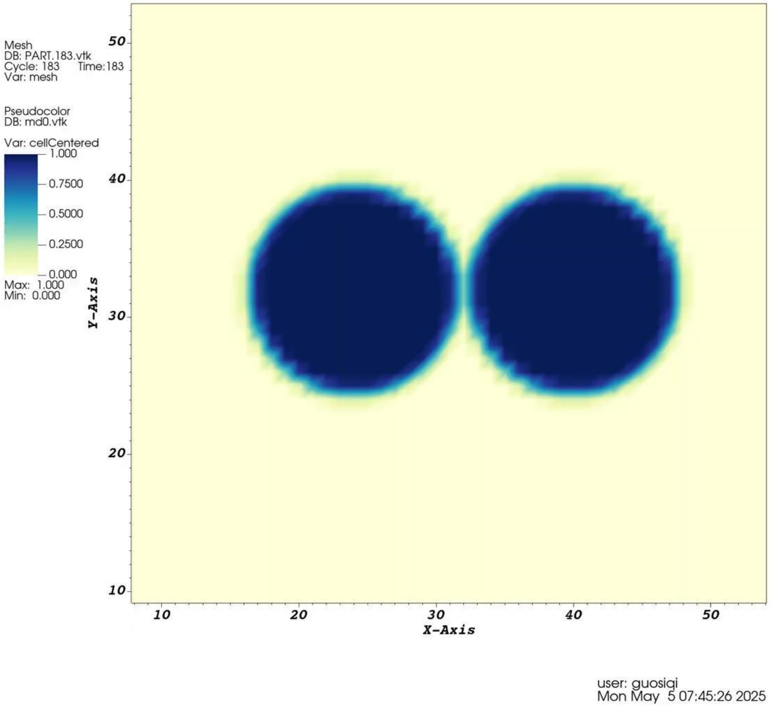
Test 3: Two Equal Strength Particles



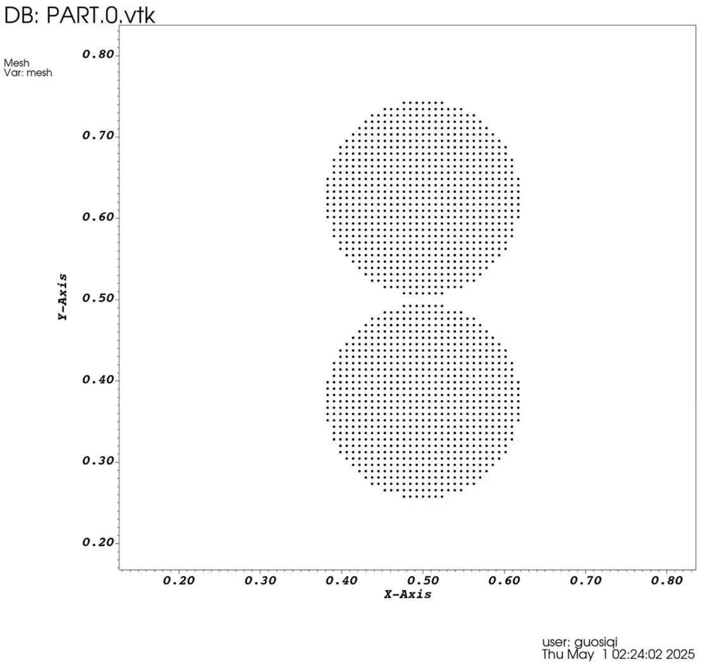


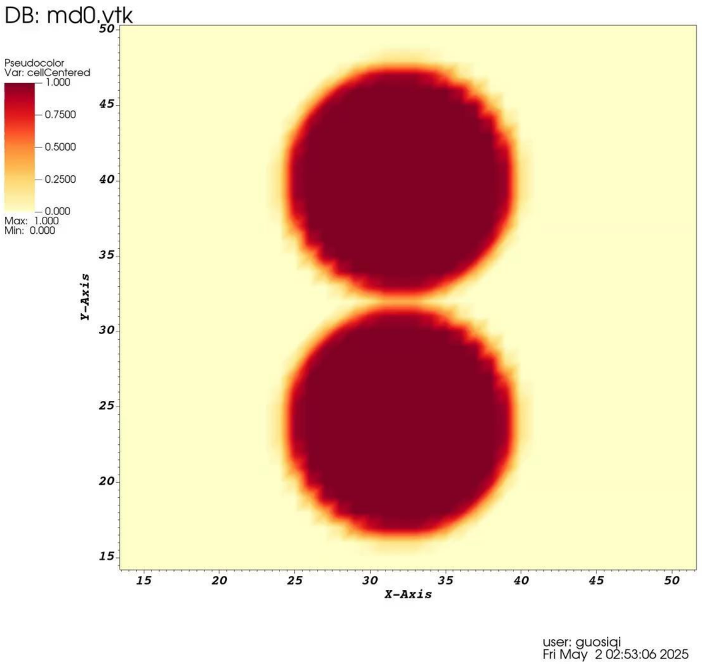
Test 4: Two-Patch Problem



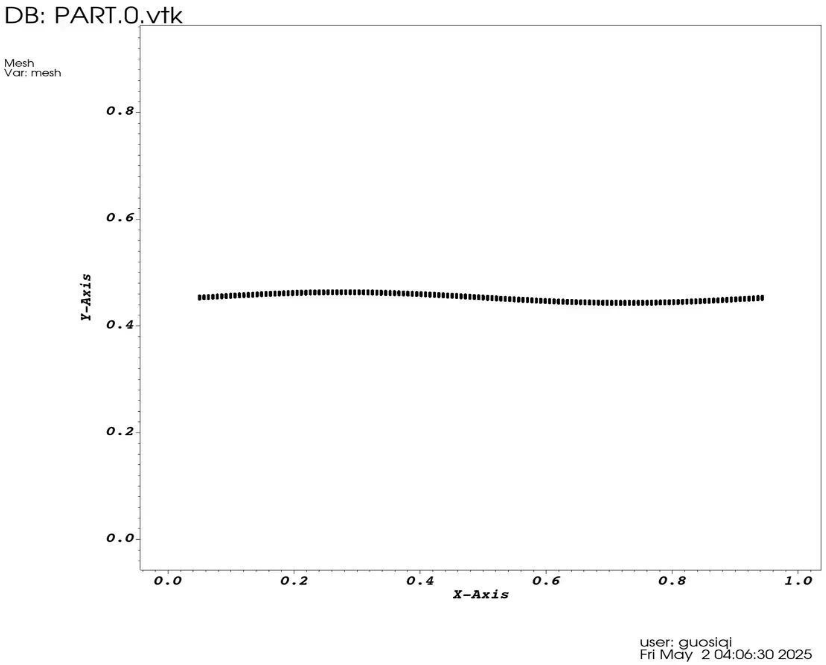


Test 5: Two-Patch Problem Long-Horizon

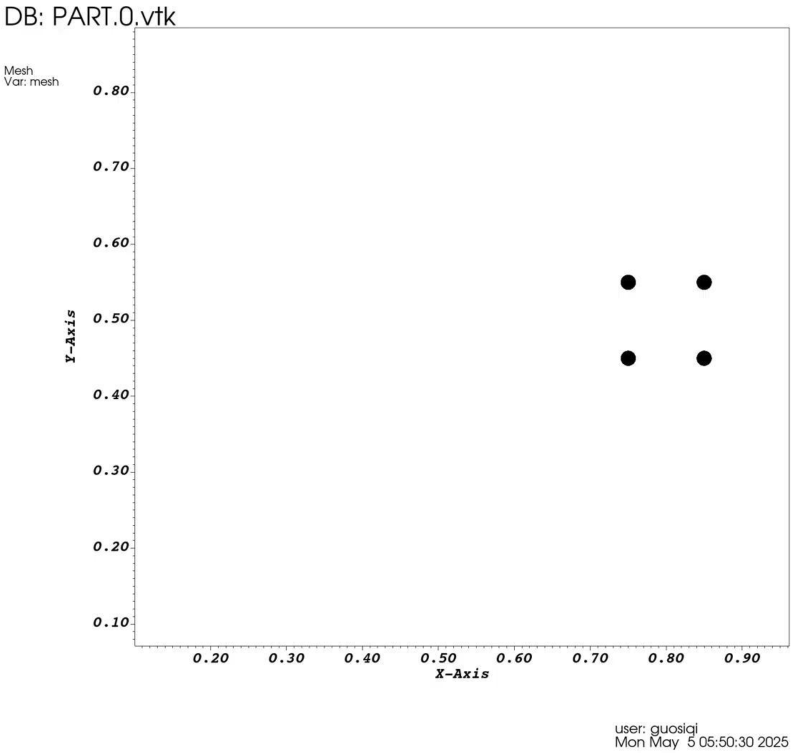


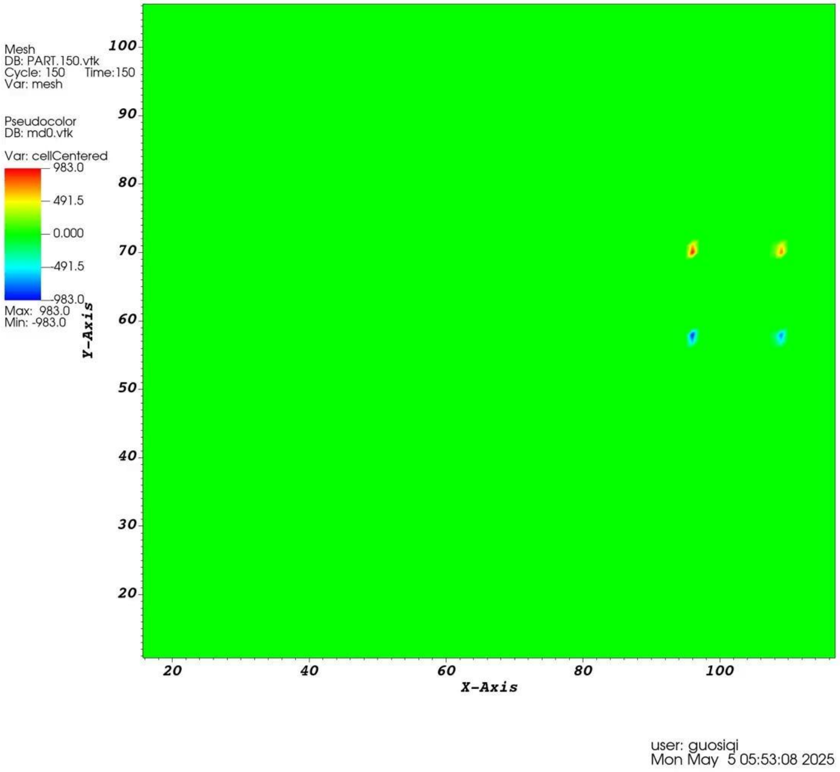


Test 6: (Kelvin-Helmholtz Instability)

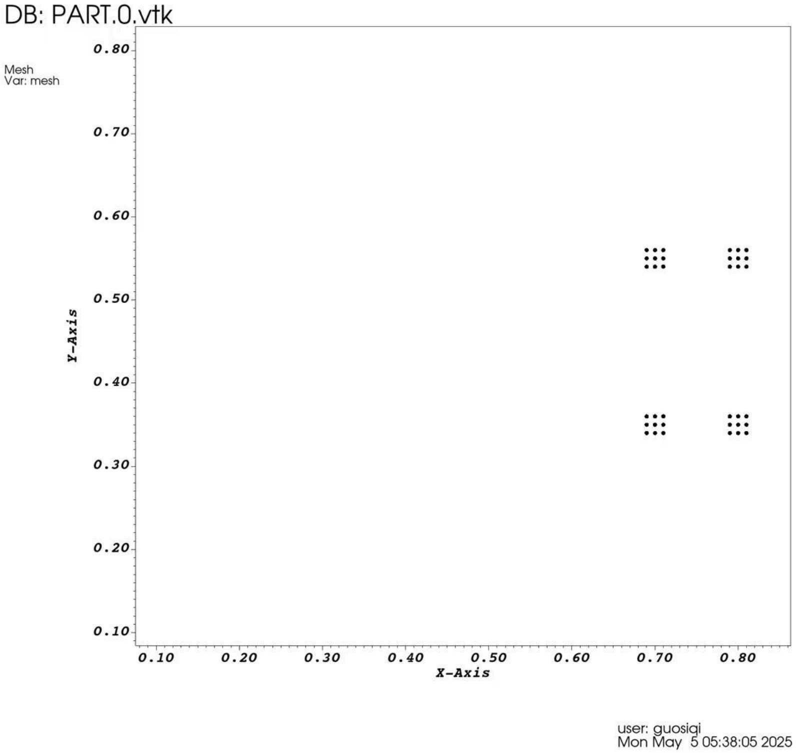


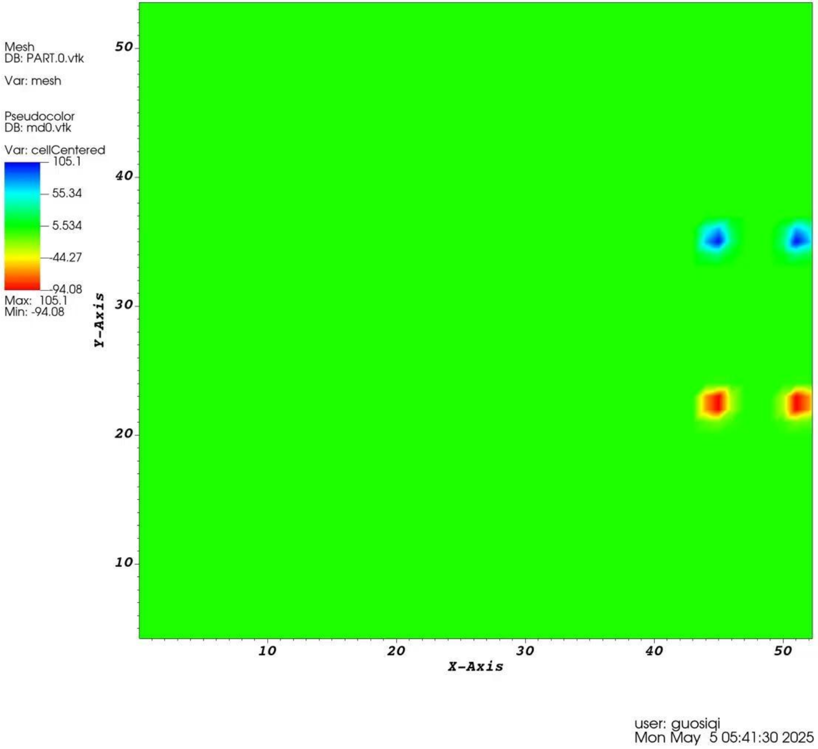
Test 7: Vortex Rings



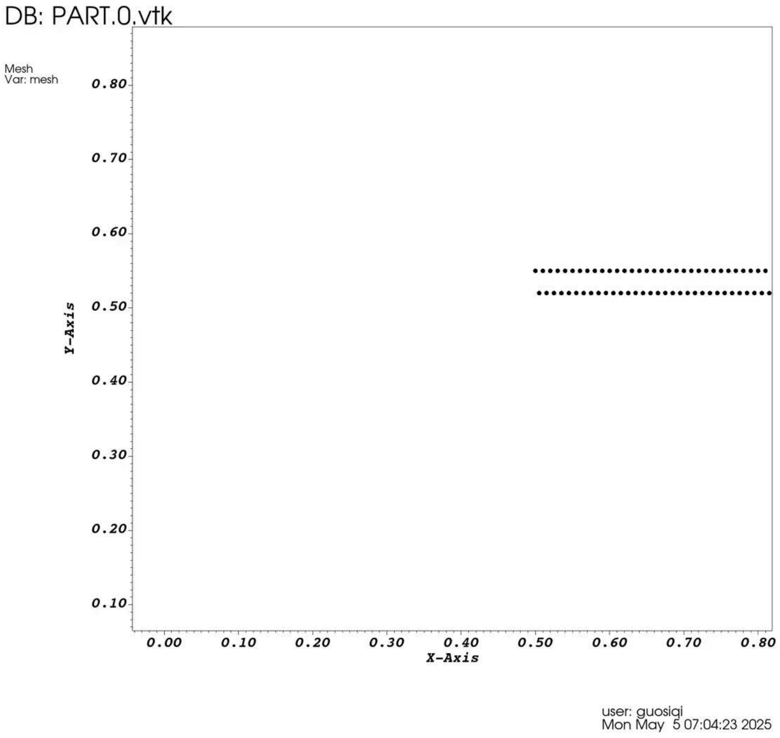


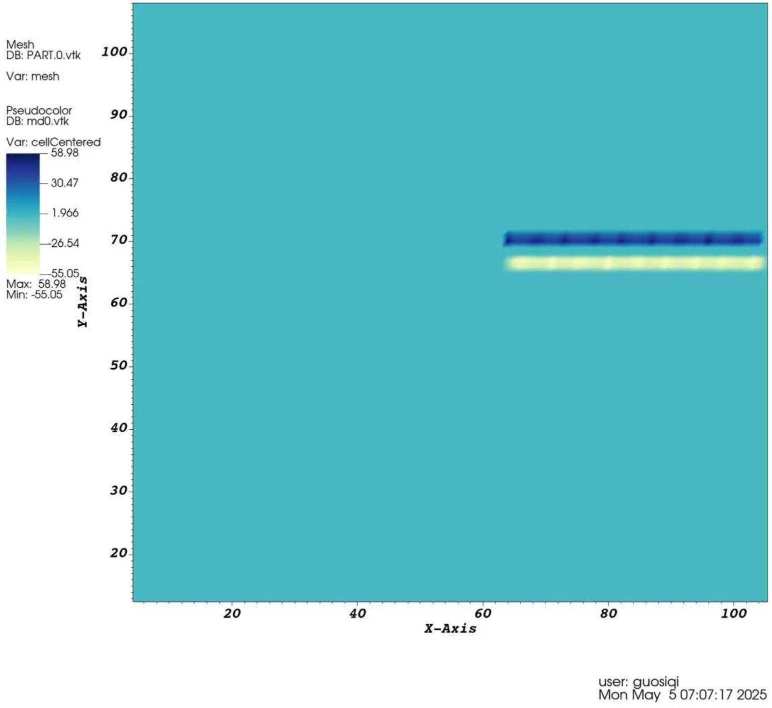
Test 8: Vortex Rings in Patches





Test 9: Karman Vortex Street





Challenges and Solutions

- Adding position to itself, not taking delta t itself.
  Bugs from Particle Set implementation.

  Lessons: Should design unit tests for even some small components.

In the Future

- Add error estimation and check the convergence.