Filtering and Unwrapping Doppler-OCT for Extended Range of Microscopic Fluid Velocity Measurement

Yang Xu[1,2], Donald Darga[2], Jason Smid[2], Adam M. Zysk[2], Daniel Teh[1], Stephen A. Boppart[1,2], P. Scott Carney[1,2]

[1] Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, 306 North Wright Street, Urbana, Illinois 61801, USA
[2] Diagnostic Photonics, Inc., 222 West Merchandise Mart Plaza, Suite 1230, Chicago, Illinois 60654, USA

Background

Doppler optical coherence tomography (D-OCT) [1] is a technique for microscopic fluid velocity measurement using optical interferometry. Applications of D-OCT include blood flow monitoring [1, 2]. Ideally, an cross-sectional velocity map similar to Fig1. is desired. However, depending on the physical configuration of a D-OCT system, it is only able to measure velocity within a range [-Vmax, +Vmax]. Velocity outside of this range wraps around, causing the banding visible in Fig2. (orange = +Vmax, blue = -Vmax). The measurement tend to be noisy, making direct unwrapping unviable.

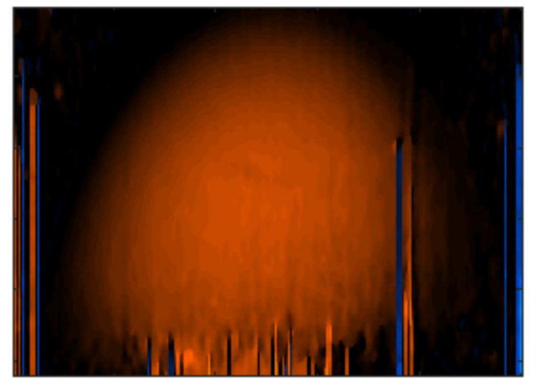


Fig.1 Ideal cross-section fluid velocity map in a tube

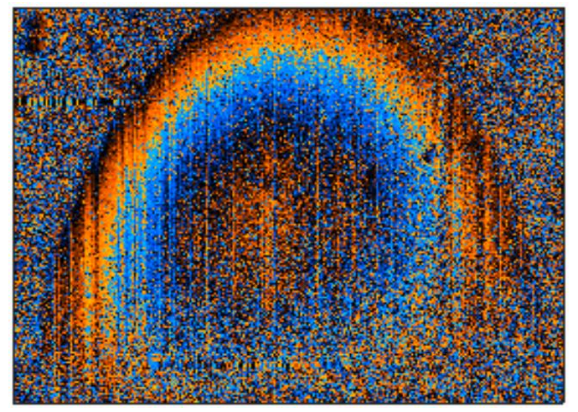


Fig.2 Noisy output of a D-OCT measurement

Robust Phase Tracker (Modified for GPU)

Inspired by the Robust Phase Tracker method [3] , we created a simpler version for our purpose. The method extracts the information and rejects the noise by taking advantage of the spatial correlation and redundancy of nearby pixels. The algorithm is suitable for massive parallel processing:

- Scan a n-by-n window around each pixel across the input image, with zero boundary condition.
- For window centered at (x0, y0),
  - Fit the phase using spatial frequencies

$$\hat{\phi}(x,y) = \omega_x (x - x_0) + \omega_y (y - y_0) + \phi_0$$

  - Use the cost function

$$C\left(\omega_x, \omega_y, \phi_0\right) = \sum_{\text{window}} \left|\cos\left(\phi(x,y)\right) - \cos\left(\hat{\phi}(x,y)\right)\right|^2 + \left|\sin\left(\phi(x,y)\right) - \sin\left(\hat{\phi}(x,y)\right)\right|^2$$

  - Solve optimization problem:

$$\left[\hat{\omega}_x, \hat{\omega}_y, \hat{\phi}_0\right] = \underset{(\omega_x, \omega_y, \phi_0) \in \text{search space}}{\operatorname{argmin}} C\left(\omega_x, \omega_y, \phi_0\right)$$

  - Output image:

$$\phi_{out}(x_0, y_0) = \hat{\phi}_0$$

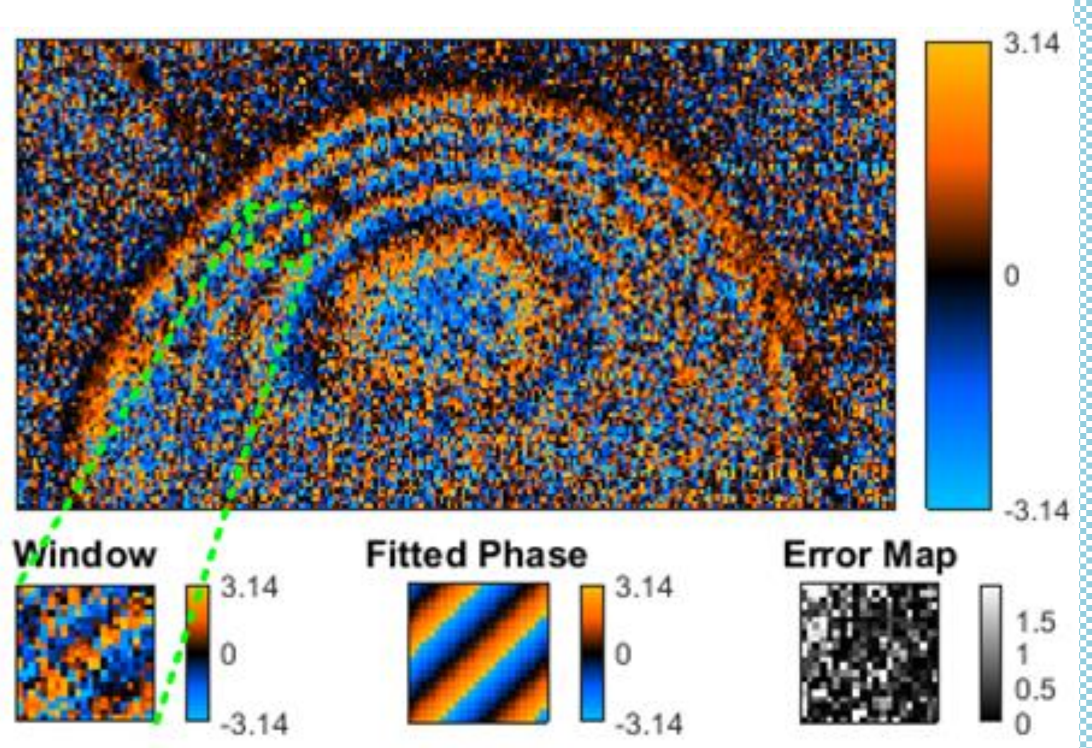


Fig.3 Visualization of the window, the fitted phase, and the error map for pixel (74, 56), generated at intermediate steps in CUDA kernel

Simulation Using Physical Model

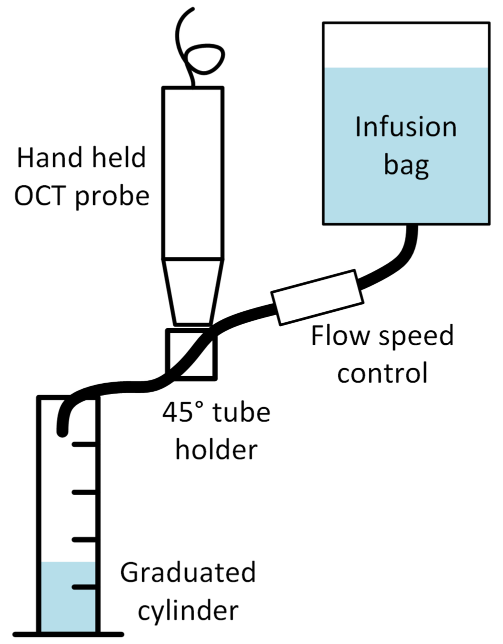
Simulations were conducted using the model of laminar flow in cylindrical tubes to estimate the true cross-sectional flow velocity map. When the average flow velocity is known, the cross-sectional velocity map can be modeled as

$$V(\rho) = 2V_{avg}\left(1 - \frac{\rho^2}{r^2}\right)$$

where r is the inner-radius of the tube, and ρ is the radius variable of the polar coordinate system. According to this model, the fluid velocity at the inner wall of the tube is 0, and the velocity at the center is 2Vavg.

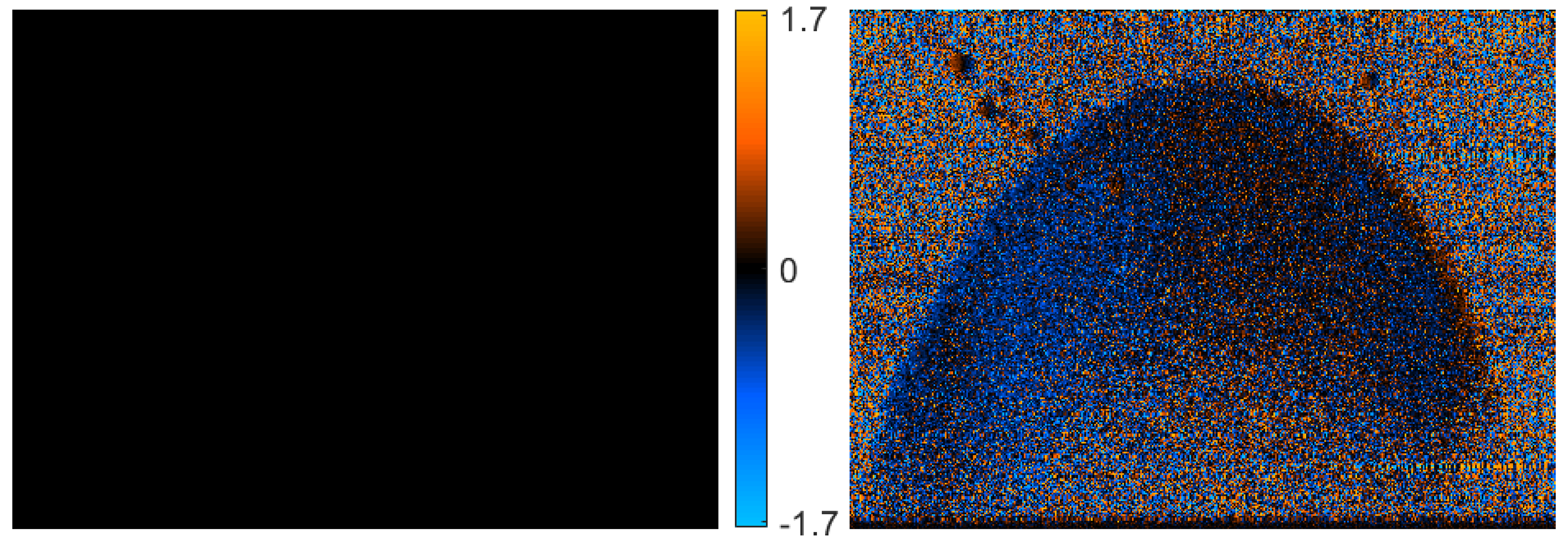
Experimental Data Collection and Processing

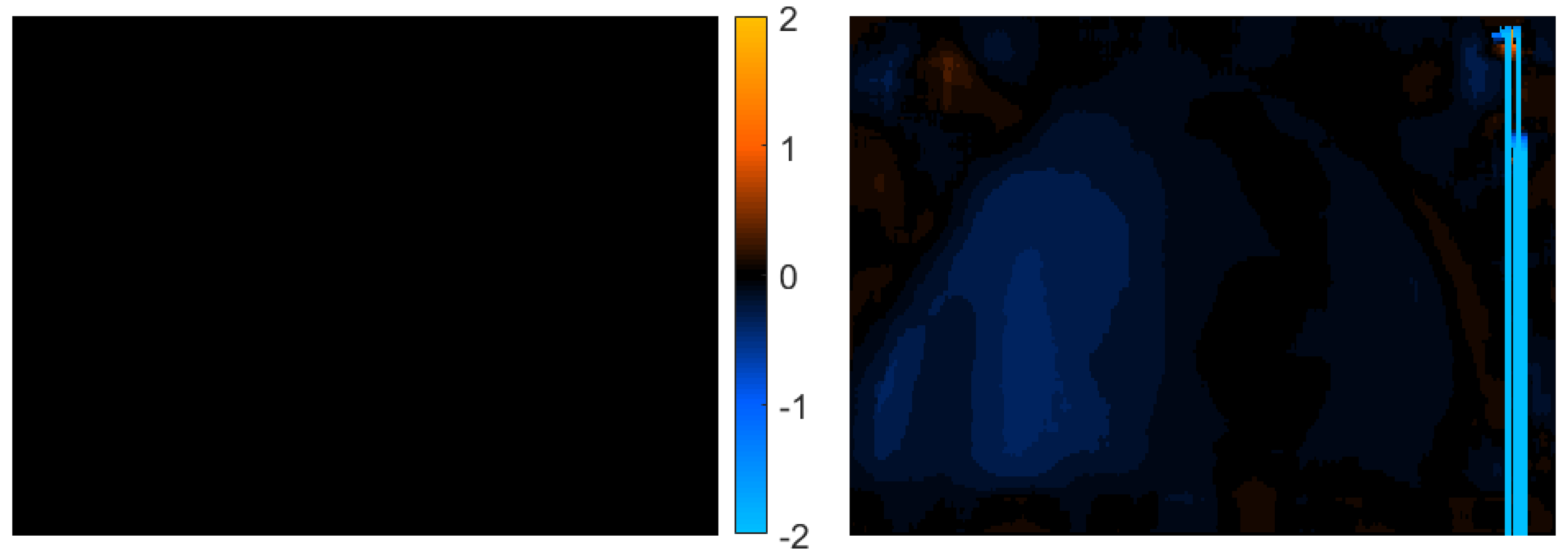
A hand-held OCT probe was used to measure the velocity map of diluted milk flowing through a tube. The flow rate was controlled by a roller clamp. The system used in the experiment is a commercial swept-spectrum OCT imaging system (Diagnostic Photonics, Inc.) operating at 1310 nm, with a spectral bandwidth of 100 nm, an A-scan rate of 50kHz and an imaging aperture of 0.05 NA. The maximum velocity this system can measure without wrapping is 1.74 cm/s. Beyond this velocity, the wrapping shows up.



Results

Experimental datasets were processed using the algorithm. Below shows a comparison of the simulation and the experimental results. The average flow velocity ranges from 0 cm/s to 8.6 cm/s, which corresponds to a peak velocity from 0 cm/s to 17.2 cm/s. Our algorithm generates a smooth velocity gradient from the noisy raw data, and the results agrees well with the physically simulated velocity profile. Because of our modification on the original robust phase tracker that eliminates the inner dependence between output pixels, the massively parallel floating point computation capability of the graphics card can be fully utilized by splitting the workload (pixels and optimization dimensions) onto the CUDA cores.





Conclusions and Discussions

Using the modified Robust Phase Tracker algorithm, we demonstrated D-OCT velocity map unwrapping of datasets with peak velocity as high as 17.2 cm/s (~10 times Vmax). The results is verified by the simulation to be accurate and physical. This algorithm significantly extends the measureable range of D-OCT velocity maps. The implementation of this algorithm on CUDA brings significant acceleration to the algorithm, reducing the processing time for each of the images to the order of seconds. This makes semi-realtime filtering and unwrapping of such datasets possible.

Reference

[1]: Joseph A. Izatt, Manish D. Kulkarni, Siavash Yazdanfar, Jennifer K. Barton, and Ashley J. Welch, "In vivo bidirectional color Doppler flow imaging of picoliter blood volumes using optical coherence tomography," Opt. Lett. **22**, 1439-1441 (1997)

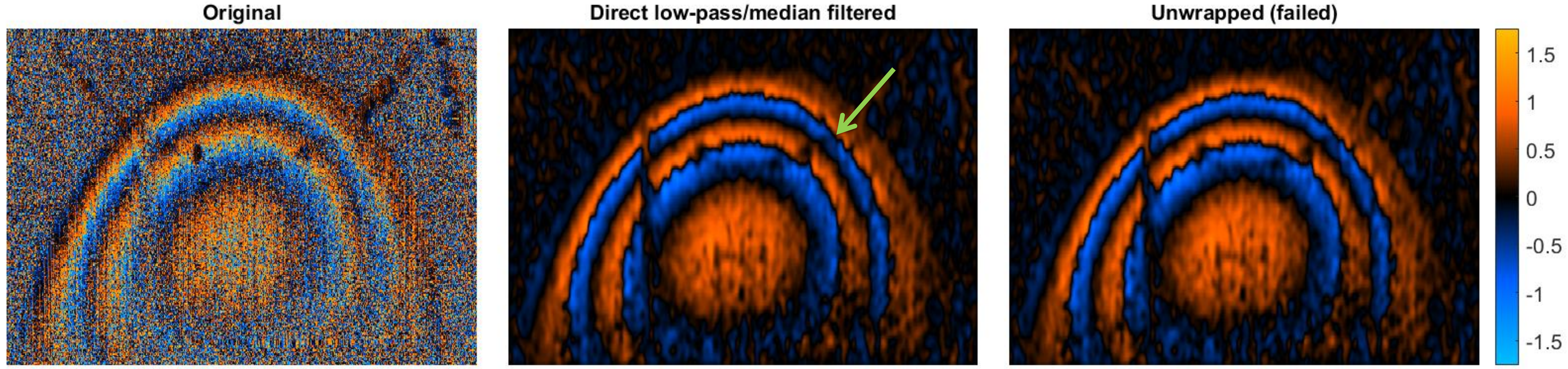
[2]: Yonghua Zhao, Zhongping Chen, Christopher Saxer, Shaohua Xiang, Johannes F. de Boer, and J. Stuart Nelson, "Phase-resolved optical coherence tomography and optical Doppler tomography for imaging blood flow in human skin with fast scanning speed and high velocity sensitivity," Opt. Lett. **25**, 114-116 (2000)

[3]: Li Kai and Qian Kemao, "A generalized regularized phase tracker for demodulation of a single fringe pattern," Opt. Express **20**, 12579-12592 (2012)

OSG Optical Science Group
ILLINOIS
BECKMAN INSTITUTE FOR ADVANCED SCIENCE AND TECHNOLOGY
DIAGNOSTIC PHOTONICS

Robust Phase Tracker (Modified for GPU)

Simple low-pass filtering don't work here, because it also softens the +Vmax to -Vmax discontinuous jumps in the image, making unwrapping impossible.



Inspired by the Robust Phase Tracker method [3] , we created a simpler version for our purpose. The method extracts the information and rejects the noise by taking advantage of the spatial correlation and redundancy of nearby pixels. The algorithm is suitable for massive parallel processing:

- Scan a n-by-n window around each pixel across the input image, with zero boundary condition.
- For window centered at (x0, y0),
  - Fit the phase using spatial frequencies

$$\hat{\phi}(x,y) = \omega_x(x-x_0)+\omega_y(y-y_0)+\phi_0$$

  - Use the cost function

$$C(\omega_x,\omega_y,\phi_0) = \sum_{(x,y)\text{ in window }(x_0,y_0)} \left|\cos(\phi(x,y)) - \cos(\hat{\phi}(x,y))\right|^2 + \left|\sin(\phi(x,y)) - \sin(\hat{\phi}(x,y))\right|^2$$

  - Solve optimization problem:

$$\left[\hat{\omega}_x,\hat{\omega}_y,\hat{\phi}_0\right] = \underset{(\omega_x,\omega_y,\phi_0)\in\text{search space}}{\operatorname{argmin}} C(\omega_x,\omega_y,\phi_0)$$

  - Output image:

$$\phi_{out}(x_0,y_0)=\hat{\phi}_0$$

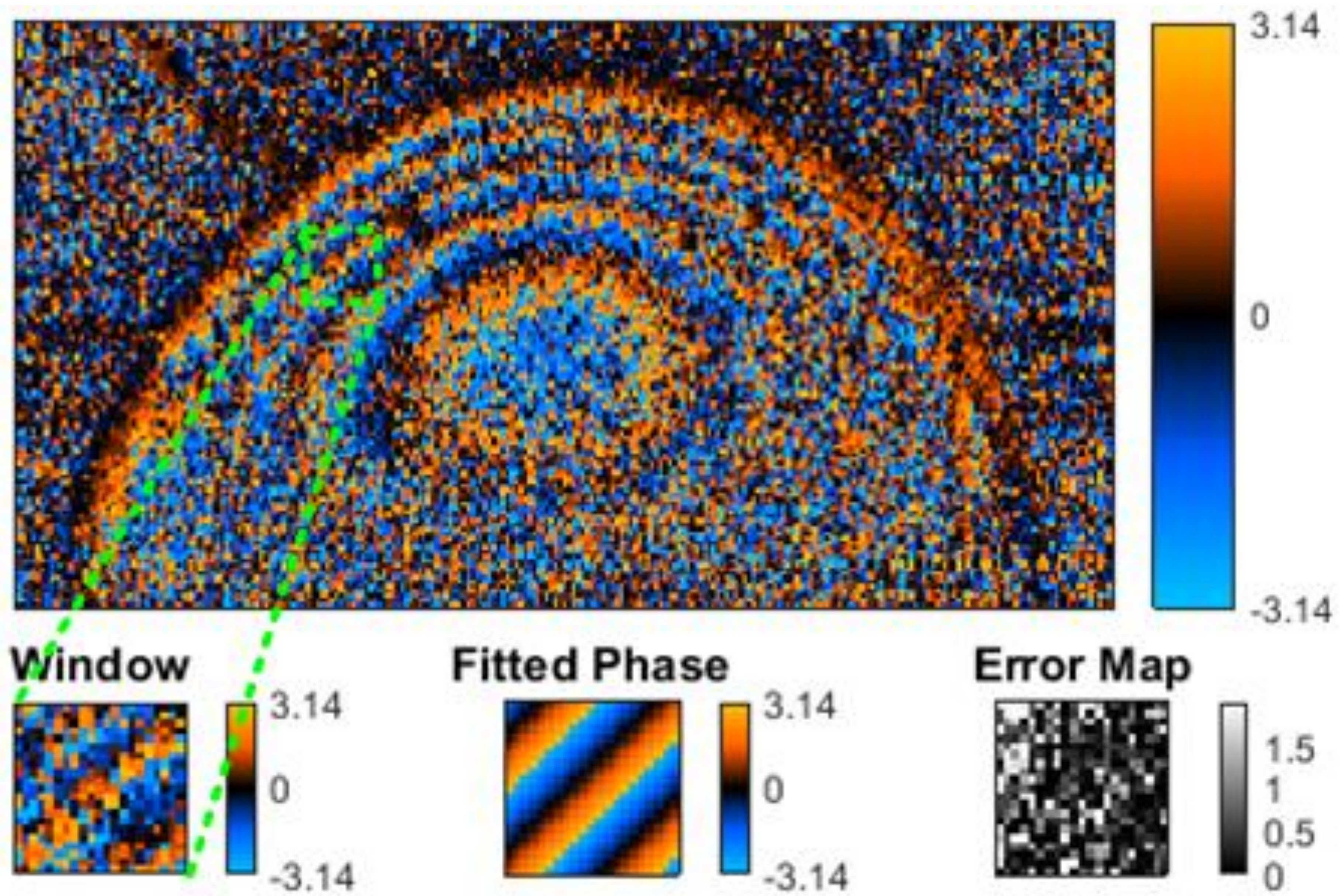
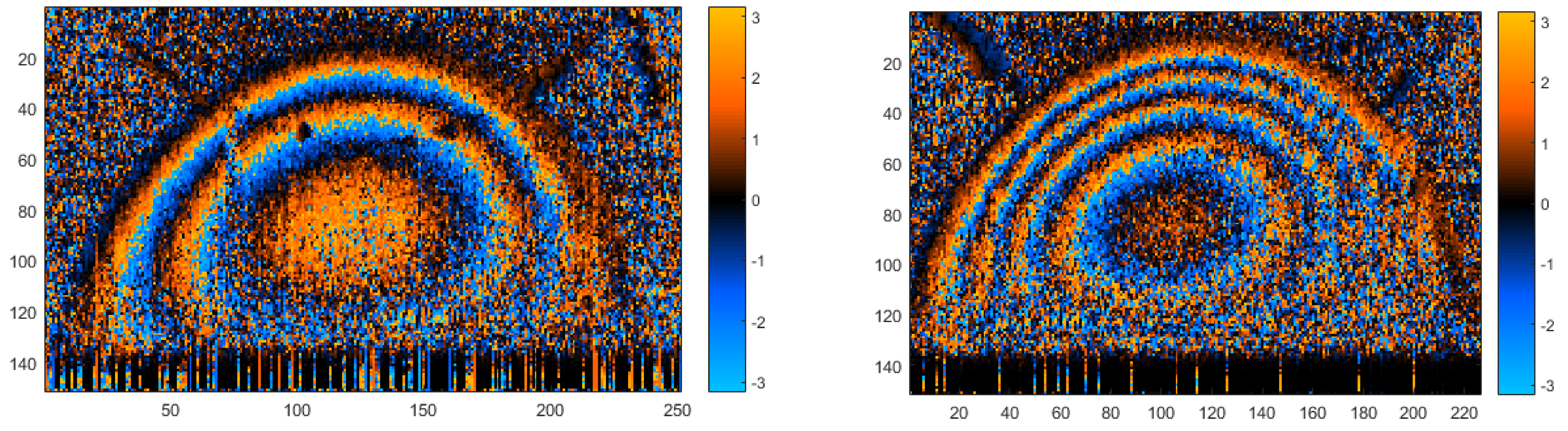
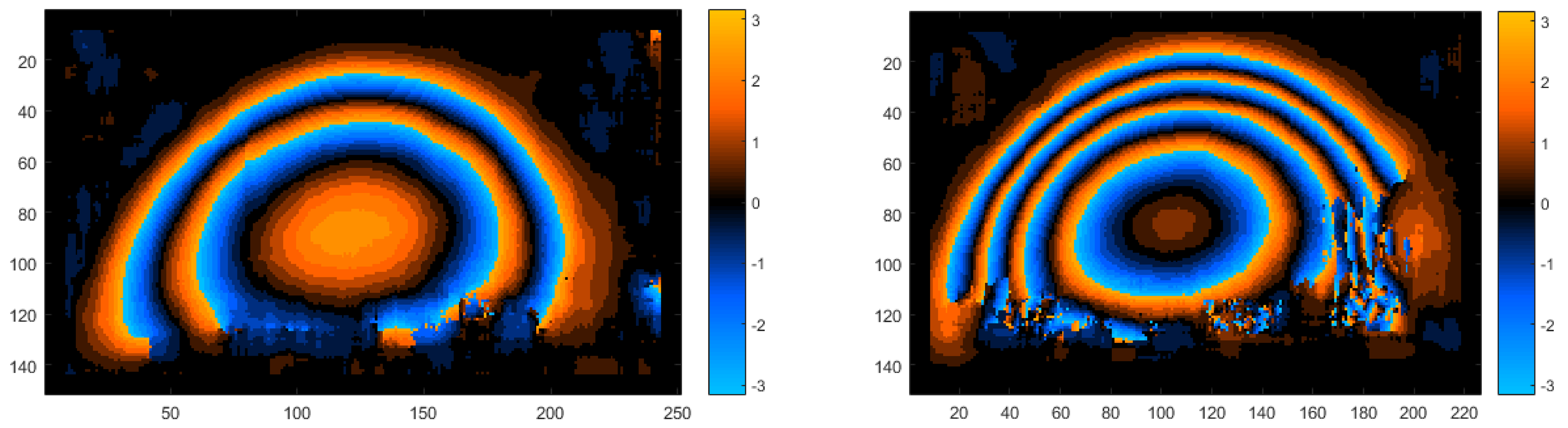


Fig.3 Visualization of the window, the fitted phase, and the error map for pixel (74, 56), generated at intermediate steps in CUDA kernel



Two input datasets with different fluid velocity

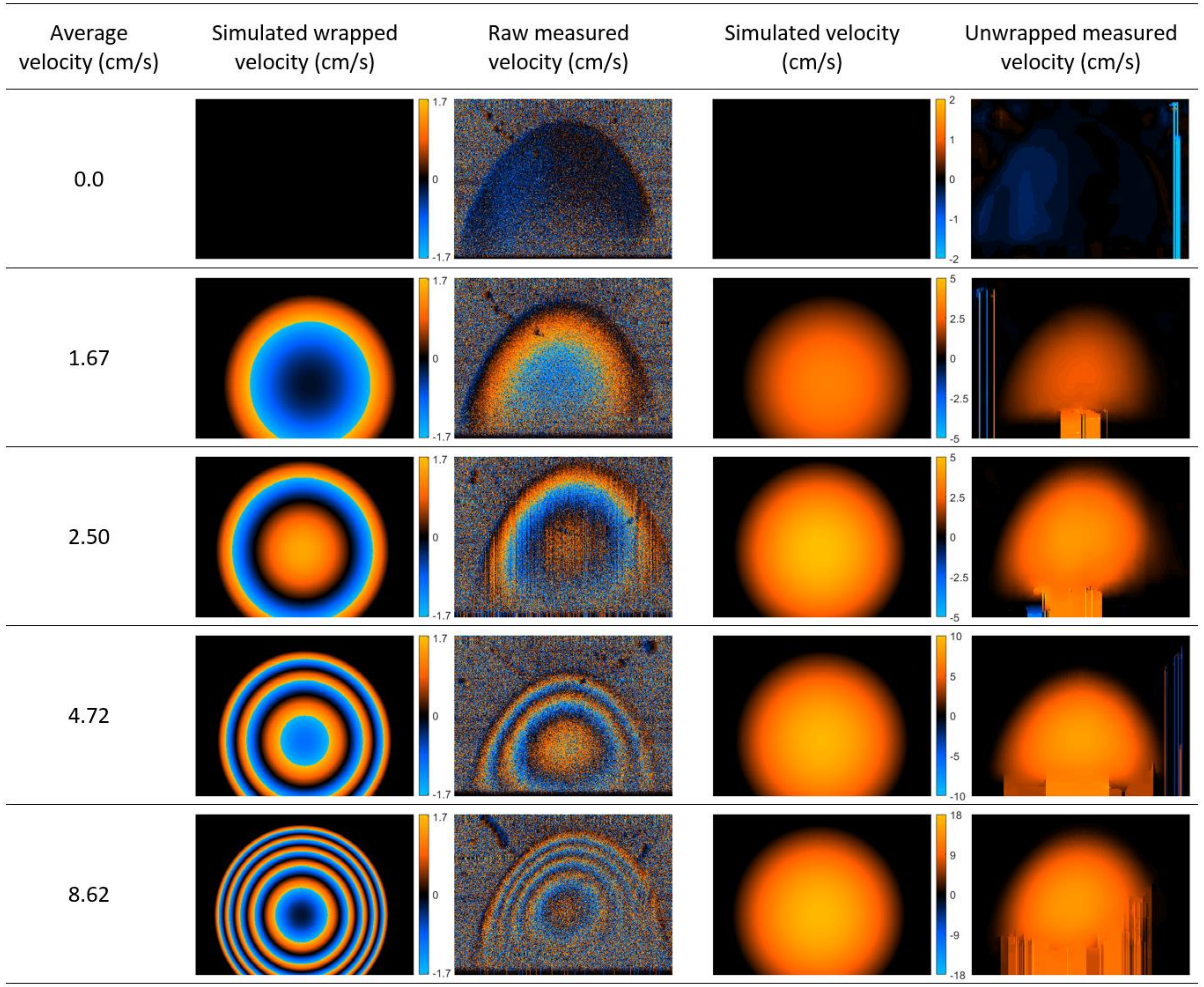


The corresponding filtered output

Implementation and Results

Experimental datasets were processed using the algorithm. The table shows a comparison of the simulation and the experimental results. The average flow velocity ranges from 0 cm/s to 8.6 cm/s, which corresponds to a peak velocity from 0 cm/s to 17.2 cm/s. Our algorithm generates a smooth velocity gradient from the noisy raw data, and the results agrees well with the physically simulated velocity profile.

Because of our modification on the original robust phase tracker that eliminates the inner dependence between output pixels, the massively parallel floating point computation capability of the graphics card can be fully utilized by splitting the workload (pixels and optimization dimensions) onto the CUDA cores. Below shows the pseudo code of the CUDA kernel and some performance statistics.




```
Kernel_2a(input_Image, temp)
{
    Focus on window around (blockDim.x,blockDim.y)
    Load 1 pixel to shared memory array(tx,ty)
    syncThreads()

    Phi = PhiSet(blockDim.z)
    Wx = tx;
    Wy = ty;

    For each (x,y) belong to win
        Calculate pixel_Err
        acumulate pixel_Err to window_Err (local variable)
    end

    move window_Err to shared_Err[Wy][Wx]
    Search minmum in window_E(Wx,Wy) with reduction tree
    syncThreads()
    Set temp(x,y,phi) to E_min
}

Kernel_2b(temp, output_Image)
{
    Each thread responsible for 1 output pixel
    Search temp again in Phi dimension
}
```

	Average Runtime Per Pixel	Average Memory Access	Average FLOPS
Achieved performance on Gtx-750Ti (21 × 21 window) (21 × 21 × 21 parameter space)	0.08 ms	463.37 MB/s	820 - 1230 GFLOPS