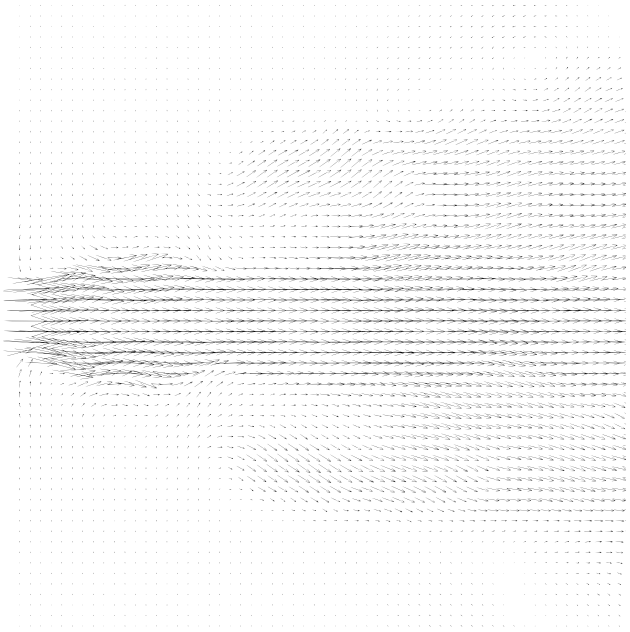
Design of a Particle-Image Velocimetry code and an Analysis of the Uncertainty of its Output

Kyle Horne



A Particle-Image Velocimetry code has been written and evaluated in various ways. The code supports direct and FFT correlation, multi-pass with window shifting, subpixel displacement, vector post-processing and parallel execution through the MPI interface. It can read and write most image file formats including the proprietary IM7 format used by LaVision's DaVis. Vectors are written in the VTK or Tecplot formats, as well as a custom binary format for rapid processing and small files. The performance of direct-correlation verses FFT-correlation is considered, as well as the parallel efficiency of the code on a small computer cluster. The direct-correlation method is significantly slower than the FFT-correlation. This is a direct result of the extra floating point operations required by the direct computation. The code demonstrates a nearly linear speedup as processors are added, resulting in parallel efficiencies in excess of 90%. Additionally, the uncertainty of the resulting vector fields has been computed using the Monte-Carlo method by way of a synthetic image generator. The bias and precision to 95% confidence are calculated to be on the order of a few hundredths of a percent.

Contents

List of Figures

List of Algorithms

List of Tables

Algorithm 1 Basic PIV

1. Read Images
2. Select locations of vectors
3. For each location:
   (a) Select interrogation region in each image
   (b) Create correlation map from regions
   (c) Locate peak in correlation map
   (d) Convert peak location to velocity vector
4. For each vector
   (a) Compare with neighbors and remove bad values
5. Write out vectors

I. Introduction

Particle-Image Velocimetry (PIV) is a relatively new technique capable of measuring the velocity of an entire two-dimensional sheet of fluid simultaneously. This is done by seeding the fluid with small particles and then illuminating the sheet with a pulsed laser. The laser light bounces off the seed particles in the flow and is captured by a high-speed camera. The images are taken in pairs due to bandwidth limitations between the camera and data acquisition computer. The PIV algorithm defines a method of using cross-correlations between the two images to compute the velocity vectors of the fluid flow.

This work documents the design, implementation and evaluation of a PIV code named *HPIV* for lack of greater creativity. The code is predominantly Fortran 95/2003, although the image I/O routines are implemented in C++ and several C libraries are used. The purposes of this code are to provide a better understanding of the PIV method to the author, to create a robust framework upon-which future PIV codes can be written without great effort, and to evaluate some of the uncertainties which result from the PIV algorithm.

This document will first cover the PIV algorithm itself, including methods of correlation, subpixel displacement calculation, multi-pass vector computation, and bad vector removal. The Monte-Carlo method and associated algorithms will be considered, including the creation of synthetic image pairs. The design of the computer code will then be discussed followed by an evaluation of the code's performance in terms of accuracy and speed.

II. PIV Algorithms

The basic PIV algorithm is shown in Alg. (1), and provides an overview of what must be done to compute the velocity field of the fluid flow. Once read, the images are stored as a two-dimensional array of pixel intensities, from which the correlations will be computed. *HPIV* computes a single set of vector locations which is used for every pass, as opposed to defining a region overlap. This results in the overlap being non-constant for different interrogation region sizes, but alleviates the need to interpolate velocity values. An interrogation region about each location is selected from both images and the correlation of the two regions is used to computed the velocity vector. Bad vectors are filtered out, an the results are written to a file for later processing and viewing. The majority of the computation time is spent in the correlation step, leading to many efforts at expediting the process.

A. Correlation

Two methods are commonly used to compute the cross-correlation of the interrogation regions from each image. The first is the direct method, in which the correlation map is simply computed from its definition.

Algorithm 2 Direct-Correlation

Implementation of the direct-correlation formula:

$$CM(i,j) = \sum_{l}\sum_{m}\left[IR_1(l,m)\cdot IR_2(l+i,m+j)\right]$$

1. Using the two interrogation regions:
   (a) For each offset in the correlation map:
      i. Position the regions according to the offset, padding with zeros
      ii. Multiply one region by the other, element-wise
      iii. Sum the products
      iv. Assign this value to that location in the correlation map

Algorithm 3 FFT-Correlation

Implementation of the FFT-correlation formula:

$$CM = \mathcal{F}^{-1}\left[\mathcal{F}(IR_1)^* \cdot \mathcal{F}(IR_2)\right]$$

1. Compute FFT of interrogation regions
2. Take the complex conjugate of the first region region
3. Element-wise multiply the two results
4. The the inverse FFT of the product
5. Shift the results

This method has the advantage of being very easy to implement, understand and adjust, but suffers from very poor performance. Because of this performance problem, other methods of computing the correlation map were investigated, and the FFT method resulted in being the fastest yet found. The FFT method relies on an identity which relates the cross-correlation to the Fourier transform of the two regions, when combined as $f \star g = \mathcal{F}^{-1}\left[\mathcal{F}(f)^* \cdot \mathcal{F}(g)\right]$.[1] While this identity only applies for functions of the complete set of $\mathcal{R}^2$, it can be applied in this case with only minimal side effects due to aliasing near the boundaries for larger interrogation regions.

The algorithms for the direct and FFT correlations are shown in (2)[2] and (3) respectively, the contents of each being largely self-explanatory. One component of the FFT-correlation must be explained, however. The FFT will compute the discrete Fourier transform of an array very rapidly, but not all algorithms organize the output exactly the same as the analytic Fourier transform would. This results in the need to shift the results of the FFTs to reconstruct the correct correlation map. For the FFT algorithm used by *HPIV* this shifting merely requires the exchange of array quadrants, as shown in Fig. (1).

Examples of various interrogation regions' FFT are given in Fig. (2a) and (2b), including the original data. The DC component of the FFTs was removed to allow the more interesting small-scale features to be seen in the results. The two images are not the same interrogation region from either image in a pair, but rather two regions from the same image.

After computing the correlation map of two interrogation regions, it is used to find the displacement most-likely to transform one region into the other. This is done by determining the location of the correlation map's maximum value. Subpixel displacement can be used to compute a non-integer location, but the general idea is still the same. This method of computing the displacement makes the assumption that the correlation map's maximum is well defined, and that there is only one of them. Neither of these assumptions is always true, but much of the time they both are. Figure (3) shows two examples of good correlation maps, for which there is only a single maximum and its location is well defined. On the contrary, Fig. (4) shows two examples of bad correlation maps. The left shows a map without a clearly defined maximum, perhaps due

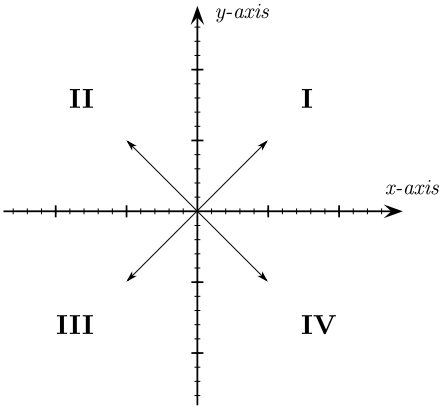
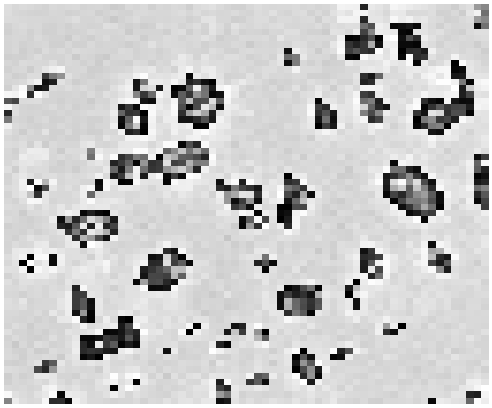
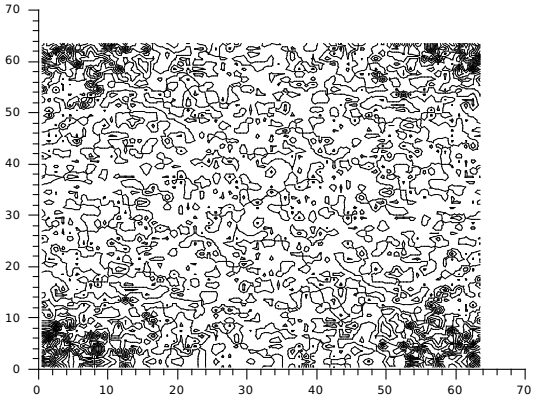
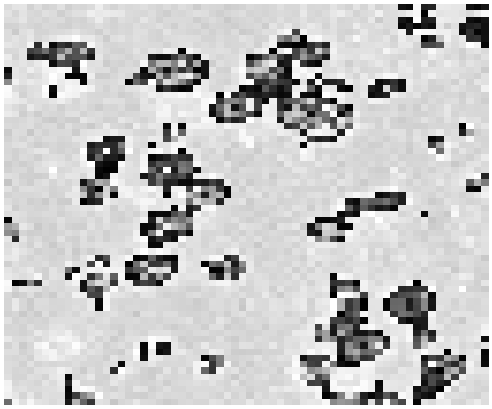
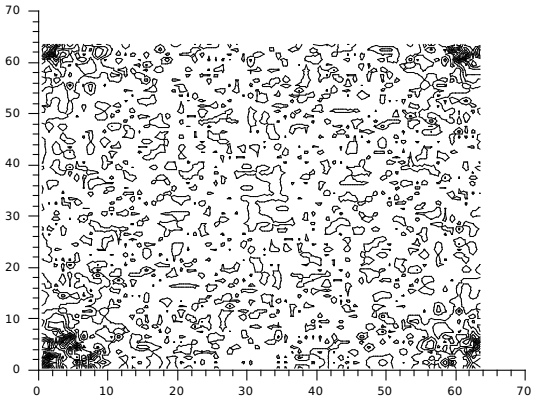


Figure 1: FFT-Shifting

Image:  FFT: 

(a) Region 1

Image:  FFT: 

(b) Region 2

Figure 2: FFT of Interrogation Regions

to a shadow in the images, while the right shows a map with two local maximums of which either could correspond to the correct displacement vector.

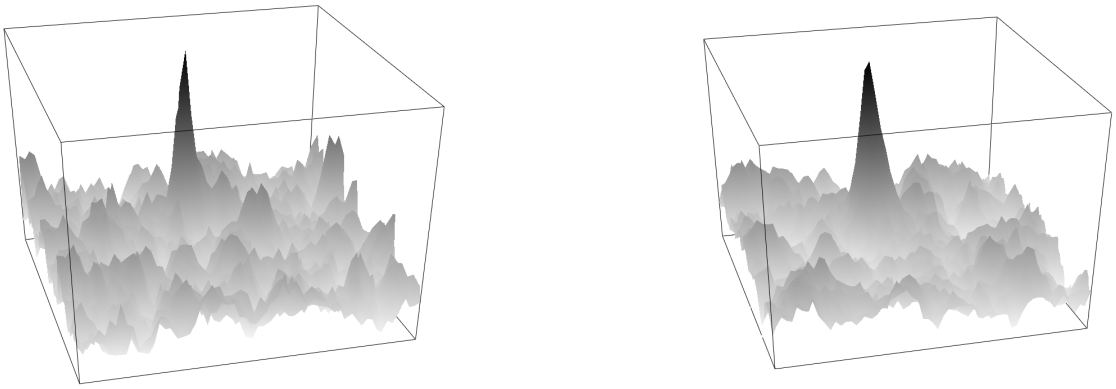


Figure 3: Good Correlation Maps

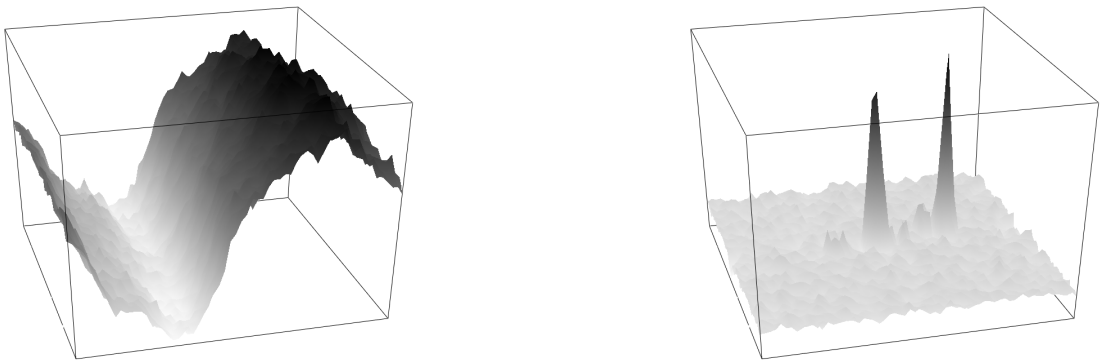


Figure 4: Bad Correlation Maps

B. Subpixel Displacement

As previously noted, subpixel displacements can be computed from the correlation map peak. This is done by fitting a gaussian curve to the correlation map in each direction. The analytic location for the maximum value of the gaussian can then be used to adjust the displacement vector to account for the relative intensities of the peak locations. Figure (5) shows a one dimensional example, along with a selection of data for computation.

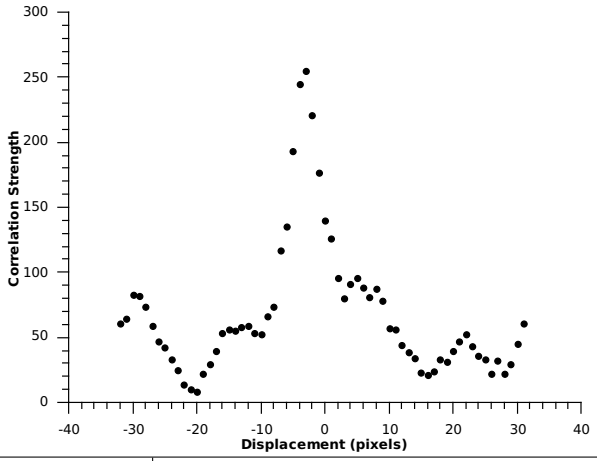
The expression used to compute the location of the maximum is given in Eq. (1).[2]

$$\begin{aligned} i &= \text{maxloc}\,(\phi) \\ d_{sp} &= \frac{1}{2}\frac{\ln\phi_{i-1} - \ln\phi_{i+1}}{\ln\phi_{i-1} - 2\ln\phi_i + \ln\phi_{i+1}} \end{aligned} \tag{1}$$

Using this equation and the values in Fig. (5), the location of the peak in the figure can be computed as the following:

$$i = \text{maxloc}\,(\phi) = -3$$

$$d_{sp} = \frac{1}{2}\frac{\ln\phi_{i-1} - \ln\phi_{i+1}}{\ln\phi_{i-1} - 2\ln\phi_i + \ln\phi_{i+1}} = \frac{1}{2}\frac{\ln 245 - \ln 221}{\ln 245 - 2\ln 255 + \ln 221} = -0.281518$$



Displacement	-5	-4	-3	-2	-1	0	1
Intensity	193	245	255	221	177	140	126

Figure 5: Subpixel Displacement Curve

Thus, the actual displacement for the data in Fig. (5) is $i + d_{sp} = (-3) + (-0.281518) = -3.281518$

C. Multi-Pass

Multi-pass is a means of getting better results without a a great amount of extra computation by reducing the needed size of the interrogation regions. The basis algorithm is covered in (4), and simplifies down to shifting the interrogation region in the second image by the displacement vector computed previously, and then adding the newly-computed displacement to the old one to achieve the final answer. As stated previously, *HPIV* does not interpolate velocities from a previous pass, but rather locates the vectors for each pass a the same locations. This causes the overlap to be a function of the interrogation region size, but precludes the need to interpolate the vectors. Several methods of region shifting are implemented in *HPIV*, such as shifting the second region the full displacement, or shifting both regions by only half the displacement, but only results from the first method are reported. Care must be taken when shifting the regions to prevent the region from leaving the image, especially in the case of a bad vector, which may result in very large (but erroneous) displacements that would require information far outside of the image. *HPIV* resolved this trouble by implementing two versions of each regions shifting method, the first allowing for full displacement, and the second limiting the possible displacement. If full displacements are requested, but found to leave the image, the limited displacement routine is called instead. It will only allow the region to shift by a specified amount. If that amount would be exceeded by the shift, the region is not shifted at all and a flag is set to prevent the resulting vector from being added to the previous pass.

D. Post-Processing

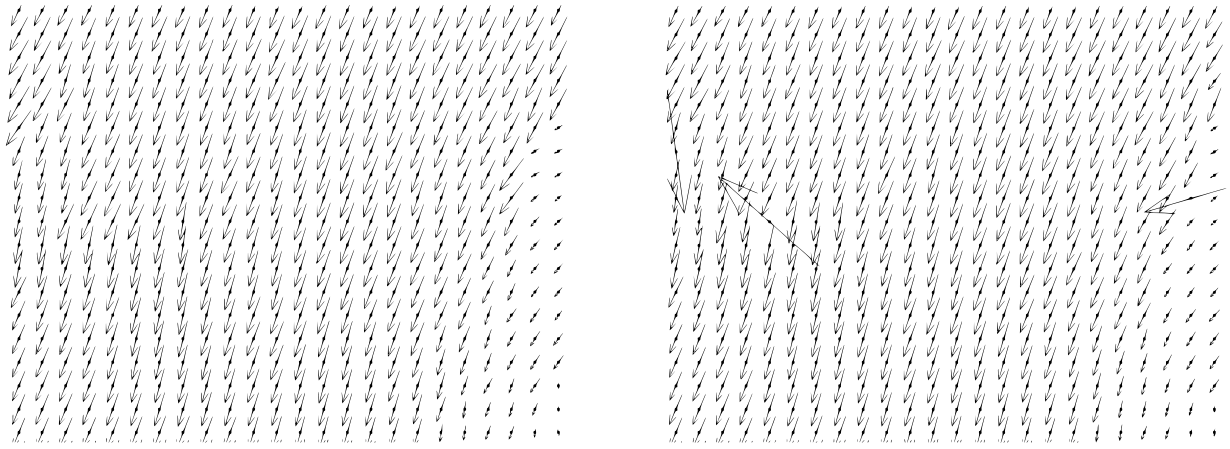
Because of the partly non-deterministic nature of the PIV algorithm, bad vectors will inevitably result during the course of computation. These should be identified and removed, especially in the case of multi-pass computation. *HPIV* classifies a vector as being bad if either of its components deviates from the median value its neighbors by 1.96 standard deviations.[1] The value 1.96 was selected so that in a uniform flow there would be a 95% confidence that the bad vector is guaranteed to be bad. Once a vector is identified as being bad, it is replaced by the median value, so that later passes will have a better change of catching the real value. The algorithm is shown in (5), while Fig. (6) shows PIV results with (6a) and without (6b) post-processing activated. The three bad vectors can be seen, as well as their replacements. The large deviation from their neighbors triggered the replacement.

Algorithm 4 Multi-Pass PIV

1. Compute velocity vectors as in Algorithm (1)
2. For each vector location:
   (a) Select interrogation region in each image, shifting the second region by the vector
   (b) Create correlation map from regions
   (c) Locate peak in correlation map
   (d) Convert peak location to velocity vector
   (e) Add new velocity vector to the old one
3. For each vector
   (a) Compare with neighbors and remove bad values

Algorithm 5 Post-Processing Vectors

1. For each vector:
   (a) Select self and nearest neighbors
   (b) Sort set
   (c) Select median vector from set
   (d) Compute mean and standard deviation for set, excluding the top and bottom vectors
   (e) If self differs from median by more than a standard deviation in u or v, replace self with median



(a) With (b) Without

Figure 6: Vector Post-Processing

Algorithm 6 Monte-Carlo Uncertainty Method

1. Generate random inputs based on uncertainty estimates
2. Use data reduction equation to compute result
3. Compute statistics on results

III. Monte-Carlo Algorithms

The appellation 'Monte-Carlo' was first used by a scientist working on the Manhattan project in connection with calculating nuclear physics results. An analytic probability could not be computed, but it was reasoned that simulating the random event many times on ENIAC (one of the earliest computers) would allow an accurate estimate to be computed. The 'Monte-Carlo' title from from the random games of chance played in casino which closely represent certain properties of quantum mechanics.[3] Any method which relies on a large quantity of random data to compute some result can be called a Monte-Carlo method.

A. Monte-Carlo Uncertainty Analysis

The Monte-Carlo Method (MCM) for uncertainty analysis simply consists of generating random inputs to the Data Reduction Equation (DRE) based on the probability distributions of each variable. These inputs are then used in the DRE, following which the bias and precision of the resulting distribution can be computing using statistical methods.[4] This process is summarized in algorithm (6). One major advantage of the MCM over the Taylor's Series Method (TSM) lies in its ability to handle non-gaussian distributions for the input variables. The MCM can also be used in the case of DRE which are too complex to practically use the TSM. Analyzing the uncertainties in PIV require the MCM because of the complexity of PIV and the inherently random inputs it uses.

PIV is more of an algorithm than a specific DRE, but for any fixed image size and PIV configuration, the PIV algorithm will result in a specific and definite DRE. This DRE could be expressed by tracing the effects of every pixel's intensity to the resulting vector field computed by the algorithm. Thus the DRE could be expressed as Eq. (2),

$$V_{m\times n} = f\left(I1_{M\times N}, I2_{M\times N}\right) \tag{2}$$

where m and n are the dimensions of the resulting vector field while M and N are the dimensions of the input images. The function f would be extremely complex, likely containing numerous conditional statements, but it could be used to describe the process used to compute the velocity vectors from the input images and code configuration. Another way of considering f is not as a single function of two images, but as $m \cdot n$ functions of $2 \cdot M \cdot N$ input variables. One function for each output vector and one input variable for each pixel intensity.

This function be different from one code to the next, but also with each set of images and configuration using the same code. Thus the DRE would be fantastically difficult to use with the TSM. Not only is the DRE created by the PIV algorithm for a particular configuration extremely complex, encouraging the use of MCMs, the inputs to that DRE are random in practice. For a fluid flow seeded with particles and illuminated by laser, certain global qualities of the images are very stable. The average diameter of seeded particles, and the average particle density are examples of these. The distribution of the particles will also be globally consistent, but there is no guarantee that a particular pixel's intensity will be affected by a particle. Thus each pixel's intensity is actually a random, based on a uniform distribution of randomly-located particles in the flow. Over large distances in the image there will be correlations due to average particle density, and over small distances the pixels will be correlated by particle diameter, but the fact remains that each pixel's intensity is largely random, especially when including noise from the camera's sensor.

The fundamentally random nature of the inputs to PIV mean MCMs must be used if the characteristic behaviors of the algorithm are to be measured. This can be accomplished by generating large amounts of synthetic image pairs, in which the spacial correlations of the images are carefully controlled, but the pixel values are still random. As more samples are taken, the law of large numbers requires that the computed values will converge the expected value, which cannot be directly calculated in this case.

Algorithm 7 Synthetic Image Generation

1. Specify particle diameter, seed density, noise level, and image size
2. Compute number of particles (N) needed to satisfy density requirement
3. Generate N particles with random locations within a region larger than the image
4. Fill images with noise
5. For each particle add the particle's intensity field to the first image
6. Displace each particle by a uniform amount
7. For each particle add the particle's intensity field to the second image

B. Synthetic Image Generation

The generation of synthetic image pairs is described by Alg. (7). Particles are assigned random locations and then the pixel intensities are modified to account for the presence of the particles. The equation used to compute the added intensity due to a particle as a function of two dimensional distance from the particle is given by Eq. (3), which is similar to the gaussian distribution and used for higher performance. Figure (7) shows both a scaled gaussian and $I(r)$ to demonstrate their similarity.

$$I(r) = \frac{1}{(16r^4 + 1) \cdot (4r^2 + 1)} \tag{3}$$

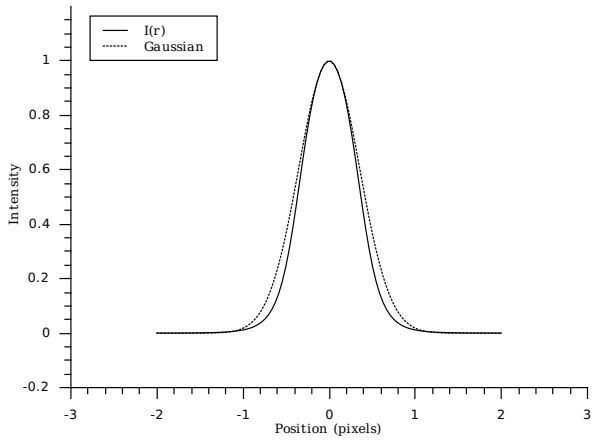


Figure 7: Comparison of $I(r)$ with a gaussian

The depth component of the particle's position is handled by assuming that the laser sheet is in fact also distributed by Eq. (3), resulting in the actual intensity added being $I(r) \cdot I(z)$, where z is the depth coordinate of the particle and ranges from -1 to 1. The distance r is scaled by the radius of the particle to make larger particles effective over longer ranges. Additionally, uniform white noise is added to the images to better represent a real image pair.[1]

An example of a synthetic image pair is shown in Fig. (8).

IV. Code Design

HPIV has been designed to be easy to maintain and yet still demonstrate high performance. To this end, each subroutine or function to complete a particular process for the PIV algorithm has been placed in a module for that process. The features implemented in *HPIV* are summarized in Table. (1), while Fig. (9) shows the module dependencies for the codebase. Many of the modules used in *HPIV* were written

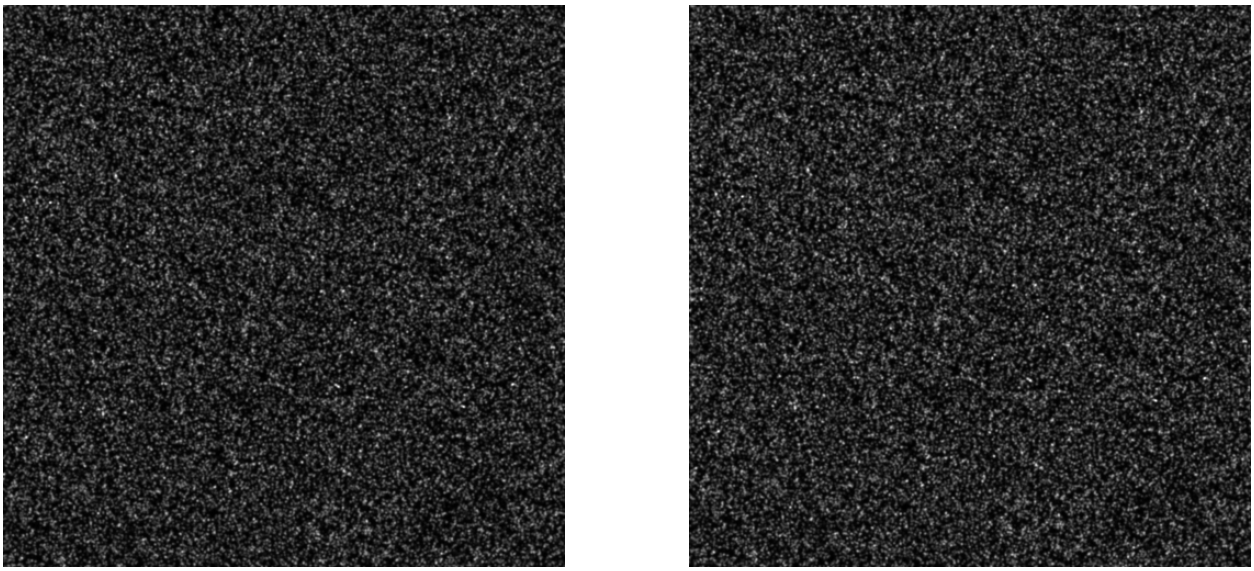


Figure 8: Synthetic Image Pair

previously for other projects and were reused here without great effort. By placing the various processes into different modules and making these modules as independent as possible, the code is made much more maintainable and understandable. This should allow any future users to upgrade or modify the software as needed.

The Fourier transform procedures are just wrappers to the FFTW3 library, the fastest implementation of the FFT algorithm available.[5] The image I/O routines are wrappers around the CImg library, which itself is just a C++ header file which provides a template for image processing.[6] CImg is compiled to use libpng and zlib in *HPIV*. The parallel execution capabilities of the code are provided by the MPI library, which provides a standardized means of passing messages from one process to another on computing clusters and is the standard message passing library in the HPC industry.[7] The entire code is built using the CMake build tool, which provides a cross-platform means of compilation.

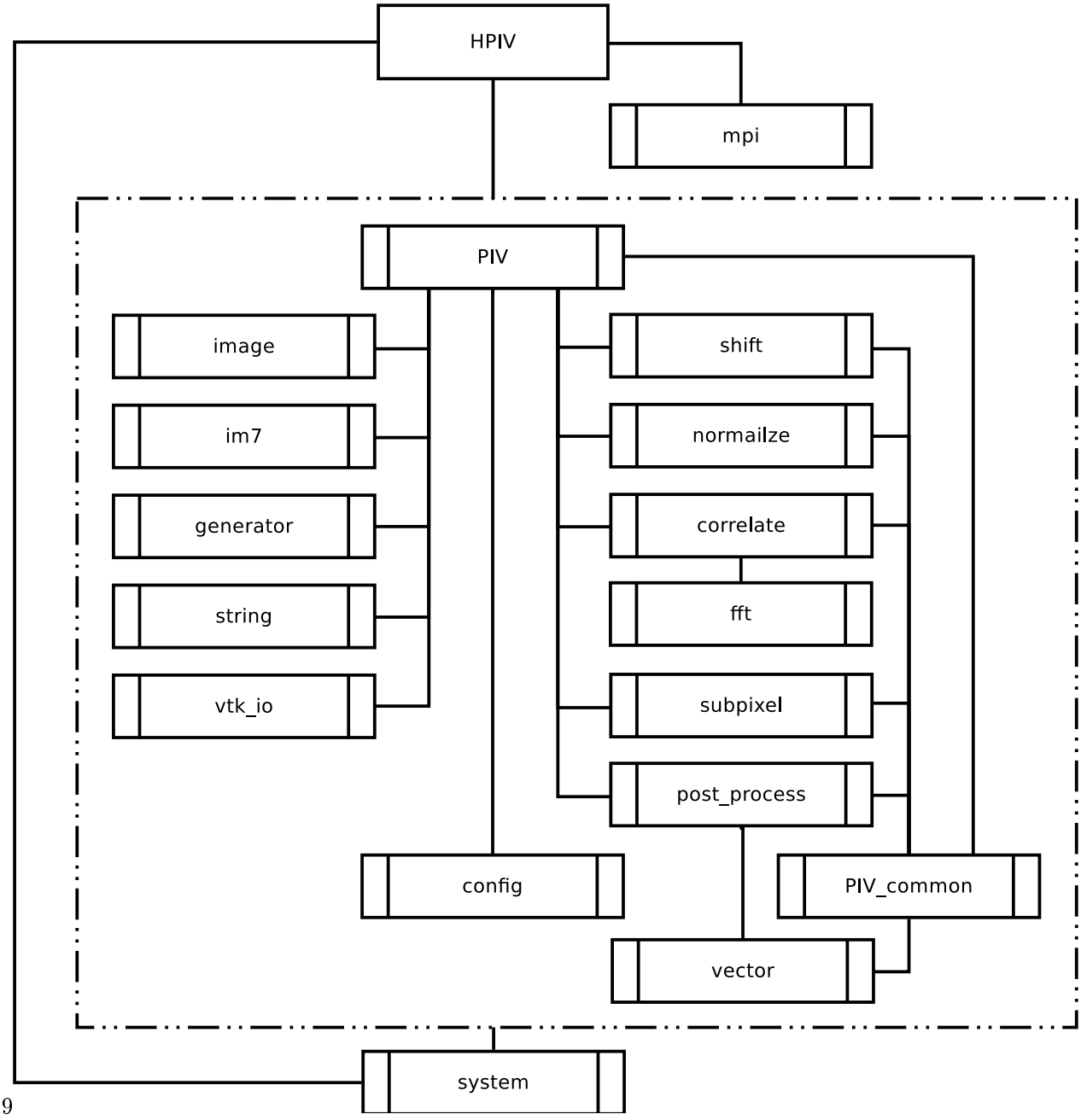
Table 1: Code features

(a) Computing best practices

Feature	Purpose
FFTW library for Fourier transforms	High-performance FFT library
MPI library for communication	Industry-standard communication interface
Read IM7, PNG, TIF, GIF, JPG, BMP	Read a variety of common image formats
Write VTK, PLT	Write common vector formats

(b) PIV features

Feature	Purpose
Both FFT and direct correlation	Compute displacement
Interrogation region windowing	Reduce FFT noise
Sub-pixel displacement	Improve vector resolution
Multi-pass with shifting	Improve vector dynamic range
Vector post-processing	Remove invalid vectors



9

Figure 9: *HPIV* Code Structure

V. Results

The results of this study are broken into three categories. The first is a set samples of the vector output generated by *HPIV*. The second is an overview of the code's performance. The third is an evaluation of the uncertainty as calculated using Monte-Carlo analysis.

A. Vectors

Figure (10) shows two examples of vectors computed by *HPIV*. Both are jet flows provided by the EFDL at USU. The two-jet flow was processed without post-processing, so some bad vectors are present, while the single-jet flow shows some very nice results.

B. Performance

The performance of *HPIV* was measured in two major ways. First, the relative performance for direct and FFT correlations was measured, and secondly the parallel processing performance was also bench-marked.

1. FFT vs Direct Correlation

The performance of the FFT-correlation method is much greater than that of the direct calculation, as seen in Fig. (11). This is entirely due to the greater number of floating point operations required for the direct method as opposed to the FFT method. The walltime increases greatly with increased interrogation window sizes, but more especially for the direct calculation. This results in the FFT method being dominant in most PIV codes.

2. Parallel Efficiency

The parallel performance of both image pair generation and processing was bench-marked and found to be satisfactory. *HPIV*'s performance in these cases is documented in Fig. (12) and (13) respectively. These results were produced on the Bombadil cluster, being composed of five nodes with two AMD Opteron processors per node operating at 2.3 GHz and four GB of RAM. Images were stored on the head node's RAID storage and accessed through NFS. The vectors were writing through the same mechanism. The cluster runs cAos linux, and the nodes are connected through gigabit Ethernet.

Because the PIV algorithm can be considered a naively parallel problem in the case of many image pairs, the nearly linear speedup is expected. It is important to document a codes performance in this way to provide other researchers who may be interested in using the code or writing a similar program an idea of how long results will take to generate. Additionally, the parallel performance metrics provide important insight to the scalability of the software. More simply, it answers the question: "will using twice as many computers make this go twice as fast?" In the case of *HPIV*, the answer is "yes."

C. Uncertainty

The uncertainty of the vector field resulting from *HPIV* was computed using 50k vectors computed from synthetic image pairs with a uniform displacement of 5.5 pixels up and 5.5 pixels to the right. The diameter of densities of the seed particles was varied, resulting in the uncertainty being a function of those two parameters. Three passes were run, with interrogation region sizes of 64x64, 32x32, and 16x16. Vector post-processing was used, as well as region normalization and correlation map windowing. The shifting method used allowed only the second region to move.

The results of these calculations are shown to a 95% confidence interval in Fig. (14), (15) and (16). While it is odd that the bias should get worse as the particle diameter increases, it must be remembered that these biases are still smaller than the expected accuracy of subpixel estimation. The bias seems largely independent of the particle density. The precision deviates very little as the density and diameter change, and can be considered negligible for later passes in all cases, but especially for particle diameters greater than one. It is logical that a particle diameter of one would lead to greater uncertainty, since there is no extra data for the subpixel displacement calculation.

(a) Dual-jet Flow

(b) Single-jet Flow

Figure 10: Example Vector Output

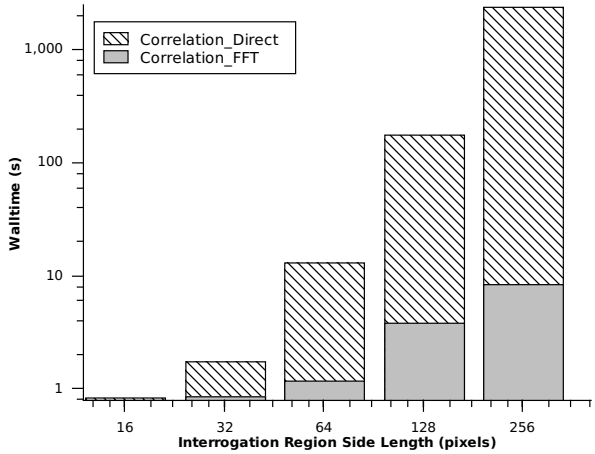
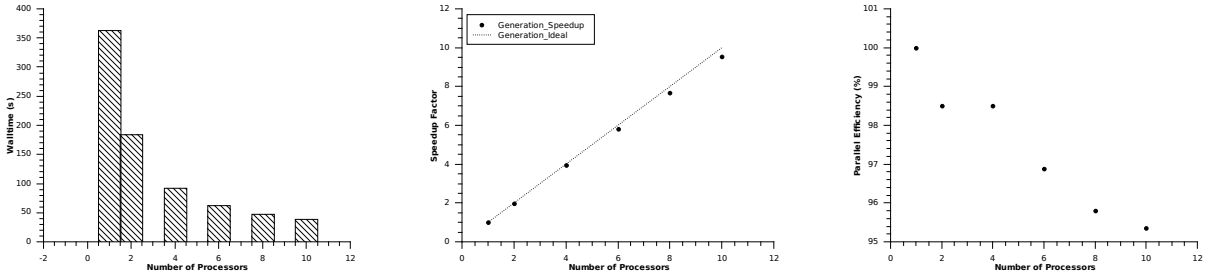
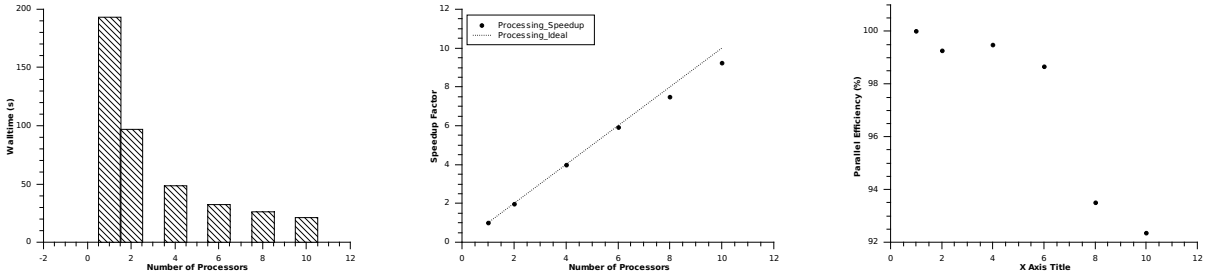


Figure 11: Comparison of FFT and Direct Correlation Performance



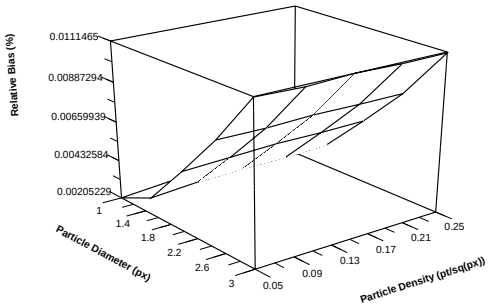
(a) Walltime (b) Speedup (c) Efficiency

Figure 12: Parallel Performance of Image Generation

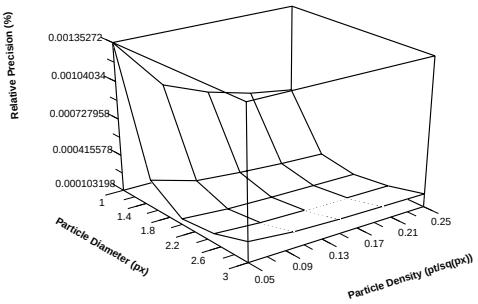


(a) Walltime (b) Speedup (c) Efficiency

Figure 13: Parallel Performance of Image Processing

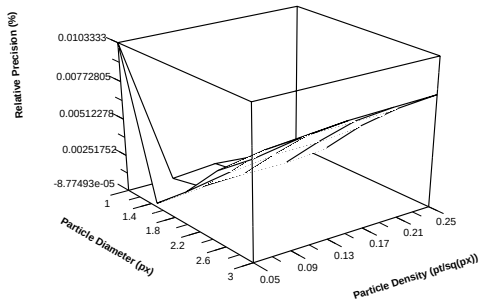


(a) Bias

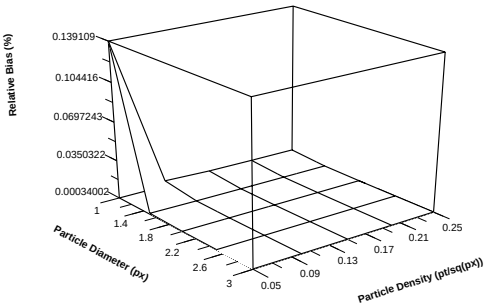


(b) Precision

Figure 14: First Pass Uncertainties

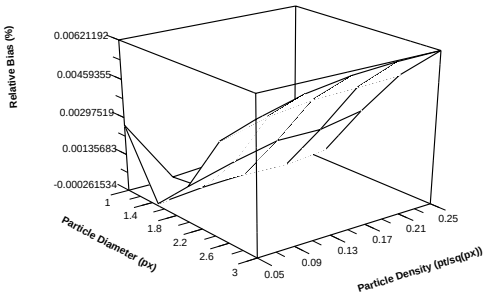


(a) Bias

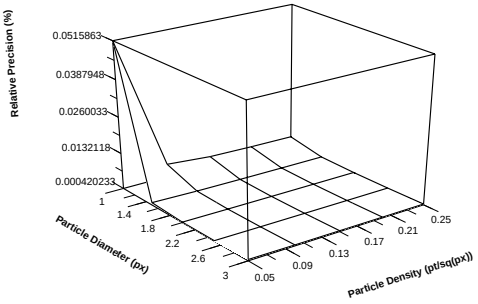


(b) Precision

Figure 15: Second Pass Uncertainties



(a) Bias



(b) Precision

Figure 16: Third Pass Uncertainties

VI. Conclusion

This work has explained the design and implementation of a PIV code called *HPIV*, as well as explored some of the possible uncertainties in the vectors resulting from using that code. The use of industry standard libraries and structured programing provide the code with a sound foundation and the potential for completely cross-platform compilation and operation. Direct computation of the correlation map is currently not an effective use of computing hardware, although that may change. The parallel performance of the code is shown to be acceptable up to ten processors on five nodes. The resulting uncertainties are shown to be largely independent of particle diameter and density to within 0.02% at a 95% confidence.

Source Code

CMakeLists.txt

```
1  cmake_minimum_required(VERSION 2.6)
2  enable_language(Fortran)
3  project(main)
4
5  set(BOMBADIL_FIX "OFF" CACHE BOOL "Activate fixes for Bombadil")
6
7  set(CMAKE_CXX_FLAGS "-D_LINUX -Dcimg_verbosity=0 -Dcimg_display=0 -
       Dcimg_use_png -Dcimg_use_zlib")
8  set(CMAKE_Fortran_FLAGS "")
9  set(CMAKE_Fortran_FLAGS_DEBUG "-g -Wall -Wtabs -fcheck-array-temporaries -
       fbacktrace -fbounds-check")
10
11 include_directories("/usr/include")
12 include_directories("/usr/include/SDL")
13 include_directories("${CMAKE_HOME_DIRECTORY}/LaVision")
14
15 set(sources_common
16    "config.f95"
17    "correlate.f95"
18    "fft.f95"
19    "generator.f95"
20    "normalize.f95"
21    "PIV_common.f95"
22    "PIV.f95"
23    "postprocess.f95"
24    "shift.f95"
25    "string.f95"
26    "subpixel.f95"
27    "system.f95"
28    "vector.f95"
29    "vtk_io.f95"
30    "window.f95"
31    )
32
33 set(sources_CImg
34    "image.f95"
35    "image_lib.cpp"
36    )
37
38 set(sources_lavision
39    "im7.f95"
40    "im7_lib.cpp"
41    "LaVision/ReadIM7.cpp"
42    "LaVision/ReadIMX.cpp"
43    )
44
45 add_executable(HPIV "main.f95" ${sources_common} ${sources_CImg} ${
       sources_lavision})
46 if(BOMBADIL_FIX)
47    target_link_libraries(HPIV "gfortranbegin")
48    target_link_libraries(HPIV "gfortran")
49    target_link_libraries(HPIV "libfftw3.a")
```

```
50     target_link_libraries(HPIV "libpng.a")
51     target_link_libraries(HPIV "libz.a")
52     target_link_libraries(HPIV "mpi_f90")
53     target_link_libraries(HPIV "mpi_f77")
54  else()
55     target_link_libraries(HPIV "fftw3")
56     target_link_libraries(HPIV "png")
57     target_link_libraries(HPIV "z")
58  endif()
59
60
61  set(sources_statistics
62     "config.f95"
63     "system.f95"
64     "string.f95"
65     "vector.f95"
66     )
67  add_executable(statistics "statistics.f95" ${sources_statistics})
68  if(BOMBADIL_FIX)
69     target_link_libraries(statistics "gfortranbegin")
70     target_link_libraries(statistics "gfortran")
71  endif()
```

config.f95

```
!Author: Kyle Horne
!License: BSD
!Filename: config.f95
!
!Purpose:
! Module which provides for reading a simply-formatted
! key=value config file and accessing the read values
! by their key.
!
!Changelog:
! 29Sep2009 - Initial versioning

module config_mod
   !Import datatypes and constants
   use system_mod
   !Prevent errors
   implicit none

   !Only export things explicitly
   private

   !IO stream
   integer,parameter::iou = 1777

   !Array of keys
   character(100),dimension(:),allocatable::keys
   !Array of values
   character(100),dimension(:),allocatable::vals

   !Export routines to be used by programs/modules
   public::read_config
   public::getInteger
   public::getReal
   public::getString

contains

   !Read a config file and parse it into the keys and values
   subroutine read_config(filename)
      !Name of file to read
      character(*),intent(in)::filename
      !IO status
      integer::sts
      !Line buffer
      character(512)::buf
      !Loop index
      integer::k
      !Line position
      integer::l

      !Open the file
      open(iou,file=filename,status='old')
      !Set the status
      sts = 0
```

```
55          !Zero line count
56          k = 0
57          !Read a line
58          read(iou,'(A)',iostat=sts) buf
59          !While there are lines to read
60          do while(sts==0)
61             !If the line starts with a '#' its a comment and should be
                   ignored
62             if(buf(1:1)=='#') then
63                !Read a new line
64                read(iou,'(A)',iostat=sts) buf
65             else
66                !Read a new line
67                read(iou,'(A)',iostat=sts) buf
68                !Incement the number of lines
69                k = k+1
70             end if
71          end do
72          !Start read the file again
73          rewind(iou)
74          !Allocate enough keys to hold all the non-comment lines
75          if(allocated(keys)) deallocate(keys)
76          allocate(keys(k))
77          !Allocate enough vals to hold all the non-comment lines
78          if(allocated(vals)) deallocate(vals)
79          allocate(vals(k))
80          !Loop through all the keys
81          do k=1,size(keys)
82             !Read a line
83             read(iou,'(A)') buf
84             !If its a comment, keep reading until its not
85             do while(buf(1:1)=='#')
86                read(iou,'(A)') buf
87             end do
88             !Find the '=' sign
89             l = index(buf,'=')
90             !The key is everything left of the equals
91             keys(k) = trim(adjustl(buf(1:l-1)))
92             !The value is everything right of the equals
93             vals(k) = trim(adjustl(buf(l+1:)))
94          end do
95          !Close the file
96          close(iou)
97       end subroutine read_config
98
99       !Loop through the keys and find the index of one
100      pure function findKey(key) result(o)
101         !Key to find
102         character(*),intent(in)::key
103         !Index to return
104         integer::o
105         !Loop through the keys
106         do o=1,size(keys)
107            !If this is it, exit
108            if(keys(o)==key) exit
```

```
109         end do
110         !If the key wasn't found, return -1
111         if(o>size(keys)) o = -1
112      end function findKey
113
114      !Return an integer value from its key
115      function getInteger(key) result(o)
116         !Key to find
117         character(*),intent(in)::key
118         !Return value
119         integer::o
120         !Index of value
121         integer::idx
122
123         !Get the index of the key
124         idx = findKey(key)
125         !If the index is positive
126         if(idx>0) then
127            !Read the value
128            read(vals(idx),*) o
129         else
130            !If the index is negative an error has occured
131            write(*,*) 'Config:Error␣trying␣to␣read␣integer␣for␣Key[',trim(
                  adjustl(key)),']'
132            !Return -1
133            o = -1
134         end if
135      end function getInteger
136
137      !Return an real value from its key
138      function getReal(key) result(o)
139         !Key to find
140         character(*),intent(in)::key
141         !Return value
142         real(wp)::o
143         !Index of value
144         integer::idx
145
146         !Get the index of the key
147         idx = findKey(key)
148         !If the index is positive
149         if(idx>0) then
150            !Read the value
151            read(vals(idx),*) o
152         else
153            !If the index is negative an error has occured
154            write(*,*) 'Config:Error␣trying␣to␣read␣real␣for␣Key[',trim(
                  adjustl(key)),']'
155            !Return -1
156            o = -1.0_wp
157         end if
158      end function getReal
159
160      !Return an string value from its key
161      function getString(key) result(o)
```

```
162         !Key to find
163         character(*),intent(in)::key
164         !Return value
165         character(100)::o
166         !Index of value
167         integer::idx
168 
169         !Get the index of the key
170         idx = findKey(key)
171         !If the index is positive
172         if(idx>0) then
173            !Read the value
174            o = vals(findKey(key))
175         else
176            !If the index is negative an error has occured
177            !write(*,*) 'Config:Error trying to read integer for Key[',trim(
                  adjustl(key)),']'
178            !Return -1
179            o = '-1'
180         end if
181      end function getString
182 
183   end module config_mod
```

correlate.f95

```
1  module correlate_mod
2     use system_mod
3     implicit none
4
5  contains
6
7     function correlate_direct(R1,R2) result(o)
8        real(wp),dimension(:,:),intent(in)::R1
9        real(wp),dimension(size(R1,1),size(R1,2)),intent(in)::R2
10       real(wp),dimension(-size(R1,1)/2:size(R1,1)/2-1+mod(size(R1,1),2),-
            size(R1,2)/2:size(R1,2)/2-1+mod(size(R1,2),2))::o
11
12       integer::i,j
13       integer::Nx,Ny
14
15       Nx = size(R1,1)
16       Ny = size(R1,2)
17
18       do i=-size(R1,1)/2,size(R1,1)/2-1
19          do j=-size(R1,2)/2,size(R1,2)/2-1
20             o(i,j) = sum( R1( max(1-i,1):min(Nx,Nx-i) , max(1-j,1):min(Ny,
                  Ny-j) ) * &
21                      & R2( max(1+i,1):min(Nx,Nx+i) , max(1+j,1):min(Ny,Ny+j)
                           ) )
22          end do
23       end do
24    end function correlate_direct
25
26    function normalize_fft(i) result(o)
27       real(wp),dimension(:,:),intent(in)::i
28       real(wp),dimension(-size(i,1)/2:size(i,1)/2-1+mod(size(i,1),2),-size
            (i,2)/2:size(i,2)/2-1+mod(size(i,2),2))::o
29
30       integer::r,c
31       real(wp)::Nr,Nc
32
33       Nr = size(i,1)
34       Nc = size(i,2)
35
36       forall(r=-size(i,1)/2:size(i,1)/2-1,c=-size(i,2)/2:size(i,2)/2-1)
37          o(r,c) = (1.0_wp-real(r,wp)/Nr)*(1.0_wp-real(c,wp)/Nc)
38       end forall
39    end function normalize_fft
40
41    function correlate_fft(R1,R2) result(o)
42       real(wp),dimension(:,:),intent(in)::R1
43       real(wp),dimension(size(R1,1),size(R1,2)),intent(in)::R2
44       real(wp),dimension(size(R1,1),size(R1,2))::o
45
46       o = fftshift(ifft(conjg(fft(R1))*fft(R2)))
47    end function correlate_fft
48
49    function fft(i) result(o)
50       use fft_mod, fft_i => fft
```

```
51
52          real(wp),dimension(:,:),intent(in)::i
53          complex(wp),dimension(size(i,1),size(i,2))::o
54
55          call fft_i(i,o)
56       end function fft
57
58       function ifft(i) result(o)
59          use fft_mod, ifft_i => ifft
60
61          complex(wp),dimension(:,:),intent(in)::i
62          real(wp),dimension(size(i,1),size(i,2))::o
63
64          call ifft_i(i,o)
65       end function ifft
66
67       function fftshift(i) result(o)
68          real(wp),dimension(:,:),intent(in)::i
69          real(wp),dimension(size(i,1),size(i,2))::o
70
71          integer::Nr,Nc
72          integer::r,c
73
74          Nr = size(i,1)
75          Nc = size(i,2)
76
77          r = Nr/2
78          c = Nc/2
79
80          o = 0.0_wp
81
82          ! Verified with scipy
83          o(:r,:c) = i(Nr-r+1:,Nc-c+1:)
84          o(Nr-r+1-mod(Nr,2):,Nc-c+1-mod(Nc,2):) = i(:r+mod(Nr,2),:c+mod(Nc,2)
               )
85
86          o(Nr-r+1-mod(Nr,2):,:c) = i(:r+mod(Nr,2),Nc-c+1:)
87          o(:r,Nc-c+1-mod(Nc,2):) = i(Nr-r+1:,:c+mod(Nc,2))
88       end function fftshift
89
90    end module correlate_mod
```

fft.f95

```
1  module fft_mod
2     use system_mod
3     implicit none
4     include 'fftw3.f'
5
6     private
7
8     interface fft
9        module procedure fft1_rr
10       module procedure fft1_rc
11       module procedure fft1_cr
12       module procedure fft1_cc
13
14       module procedure fft2_rr
15       module procedure fft2_rc
16       module procedure fft2_cr
17       module procedure fft2_cc
18    end interface
19
20    interface ifft
21       module procedure ifft1_rr
22       module procedure ifft1_rc
23       module procedure ifft1_cr
24       module procedure ifft1_cc
25
26       module procedure ifft2_rr
27       module procedure ifft2_rc
28       module procedure ifft2_cr
29       module procedure ifft2_cc
30    end interface
31
32    public::fft
33    public::ifft
34
35 contains
36
37    !============!
38    !== 1D FFT ==!
39    !============!
40
41    subroutine fft1_rr(input,output)
42       real(wp),dimension(:),intent(in)::input
43       real(wp),dimension(:),intent(out)::output
44
45       complex(dp),dimension(size(output,1))::buf
46       integer(8)::plan
47
48       buf = cmplx(input,0.0_wp,dp)
49
50       call dfftw_plan_dft_1d(plan,size(buf,1),buf,buf,FFTW_FORWARD,
            FFTW_ESTIMATE)
51       call dfftw_execute_dft(plan,buf,buf)
52       call dfftw_destroy_plan(plan)
53
```

```
54          output = real(buf,wp)
55      end subroutine fft1_rr
56
57      subroutine fft1_rc(input,output)
58          real(wp),dimension(:),intent(in)::input
59          complex(wp),dimension(:),intent(out)::output
60
61          complex(dp),dimension(size(output,1))::buf
62          integer(8)::plan
63
64          buf = cmplx(input,0.0_wp,dp)
65
66          call dfftw_plan_dft_1d(plan,size(buf,1),buf,buf,FFTW_FORWARD,
                FFTW_ESTIMATE)
67          call dfftw_execute_dft(plan,buf,buf)
68          call dfftw_destroy_plan(plan)
69
70          output = buf
71      end subroutine fft1_rc
72
73      subroutine fft1_cr(input,output)
74          complex(wp),dimension(:),intent(in)::input
75          real(wp),dimension(:),intent(out)::output
76
77          complex(dp),dimension(size(output,1))::buf
78          integer(8)::plan
79
80          buf = input
81
82          call dfftw_plan_dft_1d(plan,size(buf,1),buf,buf,FFTW_FORWARD,
                FFTW_ESTIMATE)
83          call dfftw_execute_dft(plan,buf,buf)
84          call dfftw_destroy_plan(plan)
85
86          output = real(buf,wp)
87      end subroutine fft1_cr
88
89      subroutine fft1_cc(input,output)
90          complex(wp),dimension(:),intent(in)::input
91          complex(wp),dimension(:),intent(out)::output
92
93          complex(dp),dimension(size(output,1))::buf
94          integer(8)::plan
95
96          buf = input
97
98          call dfftw_plan_dft_1d(plan,size(buf,1),buf,buf,FFTW_FORWARD,
                FFTW_ESTIMATE)
99          call dfftw_execute_dft(plan,buf,buf)
100         call dfftw_destroy_plan(plan)
101
102         output = buf
103     end subroutine fft1_cc
104
105     !============!
```

```
106     !== 2D FFT ==!
107     !============!
108
109     subroutine fft2_rr(input,output)
110        real(wp),dimension(:,:),intent(in)::input
111        real(wp),dimension(:,:),intent(out)::output
112
113        complex(dp),dimension(size(output,1),size(output,2))::buf
114        integer(8)::plan
115
116        buf = cmplx(input,0.0_wp,dp)
117
118        call dfftw_plan_dft_2d(plan,size(buf,1),size(buf,2),buf,buf,
              FFTW_FORWARD, FFTW_ESTIMATE)
119        call dfftw_execute_dft(plan,buf,buf)
120        call dfftw_destroy_plan(plan)
121
122        output = real(buf,wp)
123     end subroutine fft2_rr
124
125     subroutine fft2_rc(input,output)
126        real(wp),dimension(:,:),intent(in)::input
127        complex(wp),dimension(:,:),intent(out)::output
128
129        complex(dp),dimension(size(output,1),size(output,2))::buf
130        integer(8)::plan
131
132        buf = cmplx(input,0.0_wp,dp)
133
134        call dfftw_plan_dft_2d(plan,size(buf,1),size(buf,2),buf,buf,
              FFTW_FORWARD, FFTW_ESTIMATE)
135        call dfftw_execute_dft(plan,buf,buf)
136        call dfftw_destroy_plan(plan)
137
138        output = buf
139     end subroutine fft2_rc
140
141     subroutine fft2_cr(input,output)
142        complex(wp),dimension(:,:),intent(in)::input
143        real(wp),dimension(:,:),intent(out)::output
144
145        complex(dp),dimension(size(output,1),size(output,2))::buf
146        integer(8)::plan
147
148        buf = input
149
150        call dfftw_plan_dft_2d(plan,size(buf,1),size(buf,2),buf,buf,
              FFTW_FORWARD, FFTW_ESTIMATE)
151        call dfftw_execute_dft(plan,buf,buf)
152        call dfftw_destroy_plan(plan)
153
154        output = real(buf,wp)
155     end subroutine fft2_cr
156
157     subroutine fft2_cc(input,output)
```

```
158         complex(wp),dimension(:,:),intent(in)::input
159         complex(wp),dimension(:,:),intent(out)::output
160
161         complex(dp),dimension(size(output,1),size(output,2))::buf
162         integer(8)::plan
163
164         buf = input
165
166         call dfftw_plan_dft_2d(plan,size(buf,1),size(buf,2),buf,buf,
               FFTW_FORWARD, FFTW_ESTIMATE)
167         call dfftw_execute_dft(plan,buf,buf)
168         call dfftw_destroy_plan(plan)
169
170         output = buf
171     end subroutine fft2_cc
172
173     !==============!
174     !== 1D IFFT ==!
175     !==============!
176
177     subroutine ifft1_rr(input,output)
178         real(wp),dimension(:),intent(in)::input
179         real(wp),dimension(:),intent(out)::output
180
181         complex(dp),dimension(size(output,1))::buf
182         integer(8)::plan
183
184         buf = cmplx(input,0.0_wp,dp)
185
186         call dfftw_plan_dft_1d(plan,size(buf,1),buf,buf,FFTW_BACKWARD,
               FFTW_ESTIMATE)
187         call dfftw_execute_dft(plan,buf,buf)
188         call dfftw_destroy_plan(plan)
189
190         output = real(buf,wp)/real(size(buf),wp)
191     end subroutine ifft1_rr
192
193     subroutine ifft1_rc(input,output)
194         real(wp),dimension(:),intent(in)::input
195         complex(wp),dimension(:),intent(out)::output
196
197         complex(dp),dimension(size(output,1))::buf
198         integer(8)::plan
199
200         buf = cmplx(input,0.0_wp,dp)
201
202         call dfftw_plan_dft_1d(plan,size(buf,1),buf,buf,FFTW_BACKWARD,
               FFTW_ESTIMATE)
203         call dfftw_execute_dft(plan,buf,buf)
204         call dfftw_destroy_plan(plan)
205
206         output = buf/cmplx(size(buf),0.0_wp,wp)
207     end subroutine ifft1_rc
208
209     subroutine ifft1_cr(input,output)
```

```
210         complex(wp),dimension(:),intent(in)::input
211         real(wp),dimension(:),intent(out)::output
212
213         complex(dp),dimension(size(output,1))::buf
214         integer(8)::plan
215
216         buf = input
217
218         call dfftw_plan_dft_1d(plan,size(buf,1),buf,buf,FFTW_BACKWARD,
               FFTW_ESTIMATE)
219         call dfftw_execute_dft(plan,buf,buf)
220         call dfftw_destroy_plan(plan)
221
222         output = real(buf,wp)/real(size(buf),wp)
223     end subroutine ifft1_cr
224
225     subroutine ifft1_cc(input,output)
226         complex(wp),dimension(:),intent(in)::input
227         complex(wp),dimension(:),intent(out)::output
228
229         complex(dp),dimension(size(output,1))::buf
230         integer(8)::plan
231
232         buf = input
233
234         call dfftw_plan_dft_1d(plan,size(buf,1),buf,buf,FFTW_BACKWARD,
               FFTW_ESTIMATE)
235         call dfftw_execute_dft(plan,buf,buf)
236         call dfftw_destroy_plan(plan)
237
238         output = buf/cmplx(size(buf),0.0_wp,wp)
239     end subroutine ifft1_cc
240
241     !==============!
242     !== 2D IFFT ==!
243     !==============!
244
245     subroutine ifft2_rr(input,output)
246         real(wp),dimension(:,:),intent(in)::input
247         real(wp),dimension(:,:),intent(out)::output
248
249         complex(dp),dimension(size(output,1),size(output,2))::buf
250         integer(8)::plan
251
252         buf = cmplx(input,0.0_wp,dp)
253
254         call dfftw_plan_dft_2d(plan,size(buf,1),size(buf,2),buf,buf,
               FFTW_BACKWARD, FFTW_ESTIMATE)
255         call dfftw_execute_dft(plan,buf,buf)
256         call dfftw_destroy_plan(plan)
257
258         output = real(buf,wp)/real(size(buf),wp)
259     end subroutine ifft2_rr
260
261     subroutine ifft2_rc(input,output)
```

```
262         real(wp),dimension(:,:),intent(in)::input
263         complex(wp),dimension(:,:),intent(out)::output
264
265         complex(dp),dimension(size(output,1),size(output,2))::buf
266         integer(8)::plan
267
268         buf = cmplx(input,0.0_wp,dp)
269
270         call dfftw_plan_dft_2d(plan,size(buf,1),size(buf,2),buf,buf,
               FFTW_BACKWARD, FFTW_ESTIMATE)
271         call dfftw_execute_dft(plan,buf,buf)
272         call dfftw_destroy_plan(plan)
273
274         output = buf/cmplx(size(buf),0.0_wp,wp)
275     end subroutine ifft2_rc
276
277     subroutine ifft2_cr(input,output)
278         complex(wp),dimension(:,:),intent(in)::input
279         real(wp),dimension(:,:),intent(out)::output
280
281         complex(dp),dimension(size(output,1),size(output,2))::buf
282         integer(8)::plan
283
284         buf = input
285
286         call dfftw_plan_dft_2d(plan,size(buf,1),size(buf,2),buf,buf,
               FFTW_BACKWARD, FFTW_ESTIMATE)
287         call dfftw_execute_dft(plan,buf,buf)
288         call dfftw_destroy_plan(plan)
289
290         output = real(buf,wp)/real(size(buf),wp)
291     end subroutine ifft2_cr
292
293     subroutine ifft2_cc(input,output)
294         complex(wp),dimension(:,:),intent(in)::input
295         complex(wp),dimension(:,:),intent(out)::output
296
297         complex(dp),dimension(size(output,1),size(output,2))::buf
298         integer(8)::plan
299
300         buf = input
301
302         call dfftw_plan_dft_2d(plan,size(buf,1),size(buf,2),buf,buf,
               FFTW_BACKWARD, FFTW_ESTIMATE)
303         call dfftw_execute_dft(plan,buf,buf)
304         call dfftw_destroy_plan(plan)
305
306         output = buf/cmplx(size(buf),0.0_wp,wp)
307     end subroutine ifft2_cc
308 end module fft_mod
```

generator.f95

```
1  module generator_mod
2     use system_mod
3     use config_mod
4     implicit none
5
6     private
7
8     integer::Np
9     integer,dimension(2)::sz
10    real(wp)::d
11    real(wp)::l
12
13    type::particle_t
14       real(wp),dimension(3)::x
15       real(wp)::d
16    end type
17
18    real(wp),dimension(3)::uniform
19
20    public::initialize_generator
21    public::gen_pair
22
23 contains
24
25    subroutine initialize_generator
26       use string_mod
27       integer,dimension(:),allocatable::seed
28       integer::seed_size
29
30       real(wp)::rho
31
32       write(*,*) character('['+mpi_rank+']␣Initializing␣generator␣module')
33
34       rho = getReal('Generator.Density')
35       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Density␣=␣'+rho)
36       sz = [getInteger('Generator.Width'),getInteger('Generator.Height')]
37       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Width␣=␣'+sz(1))
38       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Height␣=␣'+sz(2))
39       d = getReal('Generator.Diameter')
40       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Diameter␣=␣'+d)
41       l = getReal('Generator.NoiseLevel')
42       write(*,*) character('['+mpi_rank+']␣␣␣Generator.NoiseLevel␣=␣'+l)
43
44       uniform(1) = getReal('Generator.Ux')
45       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Ux␣=␣'+uniform(1))
46       uniform(2) = getReal('Generator.Uy')
47       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Uy␣=␣'+uniform(2))
48       uniform(3) = getReal('Generator.Uz')
49       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Uz␣=␣'+uniform(3))
50
51       uniform(2) = -uniform(2)
52
53       Np = nint((1.2_wp*sz(1))*(1.2_wp*sz(2))*rho)
54       call random_seed(size=seed_size)
```

```
55        allocate(seed(seed_size))
56        seed = mpi_rank
57        call random_seed(put=seed)
58        deallocate(seed)
59
60        write(*,*) character('['+mpi_rank+']␣Done␣initializing␣generator␣
              module')
61     end subroutine initialize_generator
62
63     subroutine gen_pair(IM1,IM2)
64        real(wp),dimension(:,:),allocatable,intent(inout)::IM1,IM2
65
66        type(particle_t),dimension(Np)::particles
67        integer::k
68
69        if(allocated(IM1)) deallocate(IM1)
70        allocate(IM1(sz(1),sz(2)))
71        if(allocated(IM2)) deallocate(IM2)
72        allocate(IM2(sz(1),sz(2)))
73
74        call gen_particles(sz(1),sz(2),d,particles)
75
76        call random_number(IM1)
77        IM1 = l*IM1
78        call random_number(IM2)
79        IM2 = l*IM2
80
81        do k=1,size(particles)
82           call add_particle(particles(k),IM1)
83        end do
84
85
86        do k=1,size(particles)
87           call disp_uniform(particles(k))
88           call add_particle(particles(k),IM2)
89        end do
90     end subroutine gen_pair
91
92     subroutine gen_particles(lx,ly,d,particles)
93        integer,intent(in)::lx,ly
94        real(wp),intent(in)::d
95        type(particle_t),dimension(:),intent(inout)::particles
96
97        call random_number(particles(:)%x(1))
98        call random_number(particles(:)%x(2))
99        call random_number(particles(:)%x(3))
100
101       particles(:)%x(1) = real(1.2_wp*lx,wp)*particles(:)%x(1)-0.1_wp*lx
102       particles(:)%x(2) = real(1.2_wp*ly,wp)*particles(:)%x(2)-0.1_wp*ly
103       particles(:)%x(3) = 2.0_wp*particles(:)%x(3)-1.0_wp
104
105       particles(:)%d = d
106    end subroutine gen_particles
107
108    subroutine add_particle(p,IM)
```

```
109         type(particle_t),intent(in)::p
110         real(wp),dimension(:,:),intent(inout)::IM
111
112         integer::pi,pj
113         integer::r
114
115         integer::i,j
116         integer::i1,i2
117         integer::j1,j2
118
119         pi = nint(p%x(1))
120         pj = nint(p%x(2))
121         r = nint(1.96_wp*p%d)
122
123         i1 = max(pi-r,1)
124         i2 = min(pi+r,size(IM,1))
125
126         j1 = max(pj-r,1)
127         j2 = min(pj+r,size(IM,2))
128
129         forall(i=i1:i2,j=j1:j2)
130            IM(i,j) = IM(i,j)+intensity(p,i,j)
131         end forall
132      end subroutine add_particle
133
134      pure function intensity(p,i,j) result(o)
135         type(particle_t),intent(in)::p
136         integer,intent(in)::i,j
137         real(wp)::o
138
139         real(wp)::dx,dy,r
140
141         dx = (real(i,wp)-p%x(1))/p%d
142         dy = (real(j,wp)-p%x(2))/p%d
143         r = sqrt(dx**2+dy**2)
144
145         ! Distance
146         o = 1.0_wp/((16.0_wp*r**4+1.0_wp)*(4.0_wp*r**2+1.0_wp))
147         ! Depth
148         o = o*1.0_wp/((16.0_wp*p%x(3)**4+1.0_wp)*(4.0_wp*p%x(3)**2+1.0_wp))
149      end function intensity
150
151      subroutine disp_none(p)
152         type(particle_t),intent(inout)::p
153         p%x = p%x+0.0_wp
154      end subroutine disp_none
155
156      subroutine disp_uniform(p)
157         type(particle_t),intent(inout)::p
158
159         p%x = p%x+uniform
160      end subroutine disp_uniform
161
162   end module generator_mod
```

im7.f95

```
module im7_mod
   use system_mod
   use iso_c_binding
   implicit none

   private

   interface
      subroutine read_image_data(fn,nx,ny,nf) bind(C,name='
         IM7_read_image_data')
         use iso_c_binding
         character(c_char),dimension(*),intent(in)::fn
         integer(c_int),intent(out)::nx,ny,nf
      end subroutine read_image_data

      subroutine free_image_data() bind(C,name='IM7_free_image_data')
      end subroutine free_image_data

      subroutine getpixel(x,y,i) bind(C,name='IM7_getpixel')
         use iso_c_binding

         integer(c_int),value,intent(in)::x,y
         real(c_float),intent(out)::i
      end subroutine getpixel
   end interface

   public::read_image

contains

   function c2s(i) result(o)
      use iso_c_binding
      character(*),intent(in)::i
      character(c_char),dimension(len_trim(i)+1)::o

      integer::k

      forall(k=1:len_trim(i))
         o(k) = i(k:k)
      end forall
      o(len_trim(i)+1) = c_null_char
   end function c2s

   subroutine read_image(fn,I1,I2)
      character(*),intent(in)::fn
      real(wp),dimension(:,:),allocatable,intent(inout)::I1,I2

      integer(c_int)::w,h,f
      integer(c_int)::x,y
      real(c_float)::io

      call read_image_data(c2s(fn),w,h,f)

      if(allocated(I1)) deallocate(I1)
```

```
54        allocate(I1(w,h))
55        do x=0,w-1
56           do y=0,h-1
57              call getpixel(x,y,io)
58              I1(x+1,y+1) = io
59           end do
60        end do
61
62        if(f==1) then
63           call free_image_data
64           return
65        end if
66
67        if(allocated(I2)) deallocate(I2)
68        allocate(I2(w,h))
69        do x=0,w-1
70           do y=0,h-1
71              call getpixel(x,y+h,io)
72              I2(x+1,y+1) = io
73           end do
74        end do
75        call free_image_data
76     end subroutine read_image
77
78  end module im7_mod
```

im7_lib.cpp

```
1  #include "ReadIMX.h"
2  #include "ReadIM7.h"
3
4  #include "stdio.h"
5
6  BufferType buffer;
7  int allocated = 0;
8
9  extern "C" void IM7_getpixel(int x, int y, float* i) {
10    if(buffer.isFloat) *i = buffer.floatArray[y*buffer.nx+x];
11    else *i = (float) buffer.wordArray[y*buffer.nx+x];
12    }
13
14 extern "C" void IM7_read_image_data(char* fn, int* w, int* h, int* f) {
15    if(allocated == 1) DestroyBuffer(&buffer);
16    ReadIM7(fn,&buffer,NULL);
17    allocated = 1;
18    *w = buffer.nx;
19    *h = buffer.ny;
20    *f = buffer.nf;
21    }
22
23 extern "C" void IM7_free_image_data(void) {
24    DestroyBuffer(&buffer);
25    allocated = 0;
26    }
```

image.f95

```
1  module image_mod
2     use system_mod
3     use iso_c_binding
4     implicit none
5
6     private
7
8     interface read_image
9        module procedure read_image_g
10       module procedure read_image_rgb
11       module procedure read_image_rgba
12    end interface
13
14    interface write_image
15       module procedure write_image_g
16       module procedure write_image_rgb
17       module procedure write_image_rgba
18    end interface
19
20    interface
21       subroutine read_image_data(fn,c,w,h) bind(C,name='
            CImg_read_image_data')
22          use iso_c_binding
23          character(c_char),dimension(*),intent(in)::fn
24          integer(c_int),intent(out)::c,w,h
25       end subroutine read_image_data
26
27       subroutine copy_image_data(P) bind(C,name='CImg_copy_image_data')
28          use iso_c_binding
29          type(c_ptr),value,intent(in)::P
30       end subroutine copy_image_data
31
32       subroutine free_image_data() bind(C,name='CImg_free_image_data')
33       end subroutine free_image_data
34
35       subroutine write_image_data(fn,c,w,h,P) bind(C,name='
            CImg_write_image_data')
36          use iso_c_binding
37          character(c_char),dimension(*),intent(in)::fn
38          integer(c_int),intent(in)::c,w,h
39          type(c_ptr),value,intent(in)::P
40       end subroutine write_image_data
41    end interface
42
43    public::read_image
44    public::write_image
45
46 contains
47
48    function c2s(i) result(o)
49       use iso_c_binding
50       character(*),intent(in)::i
51       character(c_char),dimension(len_trim(i)+1)::o
52
```

```
53        integer::k
54
55        forall(k=1:len_trim(i))
56           o(k) = i(k:k)
57        end forall
58        o(len_trim(i)+1) = c_null_char
59     end function c2s
60
61     subroutine read_image_raw(fn,P)
62        character(*),intent(in)::fn
63        real(c_float),dimension(:,:,:),allocatable,target,intent(inout)::P
64
65        integer(c_int)::c,w,h
66
67        call read_image_data(c2s(fn),c,w,h)
68
69        if(allocated(P)) deallocate(P)
70        allocate(P(w,h,c))
71
72        call copy_image_data(c_loc(P))
73        call free_image_data()
74     end subroutine read_image_raw
75
76     subroutine read_image_g(fn,I)
77        character(*),intent(in)::fn
78        real(wp),dimension(:,:),allocatable,intent(inout)::I
79
80        real(c_float),dimension(:,:,:),allocatable::P
81
82        call read_image_raw(fn,P)
83
84        if(allocated(I)) deallocate(I)
85        allocate(I(size(P,1),size(P,2)))
86
87        select case(size(P,3))
88           case(4)
89              I = (1.0_c_float-P(:,:,4))+sqrt(sum(P(:,:,1:3)**2,3))*P(:,:,4)
                    /sqrt(3.0_wp)
90           case(3)
91              I = sqrt(sum(P**2,3))/sqrt(3.0_wp)
92           case(2)
93              I = (1.0_c_float-P(:,:,2))+P(:,:,1)*P(:,:,2)
94           case(1)
95              I = P(:,:,1)
96           case default
97              I = 0.0_wp
98        end select
99     end subroutine read_image_g
100
101    subroutine read_image_rgb(fn,R,G,B)
102       character(*),intent(in)::fn
103       real(wp),dimension(:,:),allocatable,intent(inout)::R,G,B
104
105       real(c_float),dimension(:,:,:),allocatable::P
106
```

```
107         call read_image_raw(fn,P)
108
109         if(allocated(R)) deallocate(R)
110         allocate(R(size(P,1),size(P,2)))
111
112         if(allocated(G)) deallocate(G)
113         allocate(G(size(P,1),size(P,2)))
114
115         if(allocated(B)) deallocate(B)
116         allocate(B(size(P,1),size(P,2)))
117
118         select case(size(P,3))
119            case(4)
120               R = (1.0_c_float-P(:,:,4))+P(:,:,1)*P(:,:,4)
121               G = (1.0_c_float-P(:,:,4))+P(:,:,2)*P(:,:,4)
122               B = (1.0_c_float-P(:,:,4))+P(:,:,3)*P(:,:,4)
123            case(3)
124               R = P(:,:,1)
125               G = P(:,:,2)
126               B = P(:,:,3)
127            case(2)
128               R = P(:,:,1)
129               G = 0.0_wp
130               B = P(:,:,2)
131            case(1)
132               R = P(:,:,1)
133               G = P(:,:,1)
134               B = P(:,:,1)
135            case default
136               R = 0.0_wp
137               G = 0.0_wp
138               B = 0.0_wp
139         end select
140
141      end subroutine read_image_rgb
142
143      subroutine read_image_rgba(fn,R,G,B,A)
144         character(*),intent(in)::fn
145         real(wp),dimension(:,:),allocatable,intent(inout)::R,G,B,A
146
147         real(c_float),dimension(:,:,:),allocatable::P
148
149         call read_image_raw(fn,P)
150
151         if(allocated(R)) deallocate(R)
152         allocate(R(size(P,1),size(P,2)))
153
154         if(allocated(G)) deallocate(G)
155         allocate(G(size(P,1),size(P,2)))
156
157         if(allocated(B)) deallocate(B)
158         allocate(B(size(P,1),size(P,2)))
159
160         if(allocated(A)) deallocate(A)
161         allocate(A(size(P,1),size(P,2)))
```

```
162
163         select case(size(P,3))
164            case(4)
165               R = P(:,:,1)
166               G = P(:,:,2)
167               B = P(:,:,3)
168               A = P(:,:,4)
169            case(3)
170               R = P(:,:,1)
171               G = P(:,:,2)
172               B = P(:,:,3)
173               A = 1.0_wp
174            case(2)
175               R = P(:,:,1)
176               G = P(:,:,1)
177               B = P(:,:,1)
178               A = P(:,:,2)
179            case(1)
180               R = P(:,:,1)
181               G = P(:,:,1)
182               B = P(:,:,1)
183               A = 1.0_wp
184            case default
185               R = 0.0_wp
186               G = 0.0_wp
187               B = 0.0_wp
188               A = 0.0_wp
189         end select
190
191      end subroutine read_image_rgba
192
193      subroutine write_image_g(fn,I)
194         character(*),intent(in)::fn
195         real(wp),dimension(:,:),intent(in)::I
196
197         real(c_float),dimension(:,:,:),allocatable,target::P
198
199         allocate(P(size(I,1),size(I,2),1))
200         P(:,:,1) = I(:,:)
201         call write_image_data(c2s(fn),size(P,3),size(P,1),size(P,2),c_loc(P)
               )
202      end subroutine write_image_g
203
204      subroutine write_image_rgb(fn,R,G,B)
205         character(*),intent(in)::fn
206         real(wp),dimension(:,:),intent(in)::R
207         real(wp),dimension(size(R,1),size(R,2)),intent(in)::G,B
208
209         real(c_float),dimension(:,:,:),allocatable,target::P
210
211         allocate(P(size(R,1),size(R,2),3))
212         P(:,:,1) = R(:,:)
213         P(:,:,2) = G(:,:)
214         P(:,:,3) = B(:,:)
```

```
215         call write_image_data(c2s(fn),size(P,3),size(P,1),size(P,2),c_loc(P)
               )
216     end subroutine write_image_rgb
217
218     subroutine write_image_rgba(fn,R,G,B,A)
219         character(*),intent(in)::fn
220         real(wp),dimension(:,:),intent(in)::R
221         real(wp),dimension(size(R,1),size(R,2)),intent(in)::G,B,A
222
223         real(c_float),dimension(:,:,:),allocatable,target::P
224
225         allocate(P(size(R,1),size(R,2),4))
226         P(:,:,1) = R(:,:)
227         P(:,:,2) = G(:,:)
228         P(:,:,3) = B(:,:)
229         P(:,:,4) = A(:,:)
230         call write_image_data(c2s(fn),size(P,3),size(P,1),size(P,2),c_loc(P)
               )
231     end subroutine write_image_rgba
232 end module image_mod
```

image_lib.cpp

```
#include <iostream>

#include "CImg.h"

using namespace cimg_library;

CImg<float>* image;

extern "C" void CImg_read_image_data(char* fn, int* c, int* w, int* h) {
   image = new CImg<float>(fn);
   *w = image->width();
   *h = image->height();
   *c = image->spectrum();
   }

extern "C" void CImg_copy_image_data(float* P) {
   int N = image->spectrum()*image->width()*image->height();
   float* data = image->data();
   for(int k=0;k<N;k++) P[k] = data[k];
   }

extern "C" void CImg_free_image_data() {
   delete image;
   }

extern "C" void CImg_write_image_data(char* fn, int* c, int* w, int* h,
   float* P) {
   CImg<float> img(*w,*h,1,*c);
   float* data = img.data();
   int N = (*c)*(*w)*(*h);
   for(int k=0;k<N;k++) data[k] = P[k];
   img.normalize(0.0,255.0);
   img.save(fn);
   }
```

main.f95

```
program HPIV
   use system_mod
   use config_mod
   use PIV_mod
   use generator_mod
   use mpi
   implicit none

   character(128)::cfn
   integer::pairs
   integer::k

   integer::ierr

   call MPI_Init(ierr)
   call MPI_Comm_rank(MPI_COMM_WORLD,mpi_rank,ierr)
   call MPI_Comm_size(MPI_COMM_WORLD,mpi_size,ierr)


   call get_command_argument(1,cfn)
   if(cfn=='') call doFatalError('No config file specified')

   call read_config(cfn)
   call inititlize_PIV
   call initialize_generator
   pairs = getInteger('Pairs')

   do k=1,pairs
      if(mod(k,mpi_size)==mpi_rank) call process_pair(k)
   end do

   call finalize_PIV
   call MPI_Finalize(ierr)

end program HPIV
```

normalize.f95

```
module normalize_mod
   use system_mod
   implicit none

contains

   function normalize_none(i) result(o)
      real(wp),dimension(:,:),intent(in)::i
      real(wp),dimension(size(i,1),size(i,2))::o

      o = i
   end function normalize_none

   function normalize_std(i) result(o)
      real(wp),dimension(:,:),intent(in)::i
      real(wp),dimension(size(i,1),size(i,2))::o

      real(wp)::N
      real(wp)::mean
      real(wp)::std

      N = real(size(i),wp)
      mean = sum(i)/N
      std = sqrt(1.0_wp/(N-1.0_wp)*sum((i-mean)**2))

      o = (i-mean)/std
   end function normalize_std

end module normalize_mod
```

PIV_common.f95

```
module PIV_common_mod
   use system_mod
   use vector_mod
   implicit none

   integer,parameter::str_len = 64

   type::pass_t
      integer::region_size
      character(str_len)::shift
      character(str_len)::normalize
      character(str_len)::window
      character(str_len)::correlate
      character(str_len)::subpixel
      character(str_len)::postprocess
   end type

   type::pair_t
      ! (x,y)
      real(wp),dimension(:,:),allocatable::I1,I2
      ! (i,j)
      type(vector_t),dimension(:,:),allocatable::X
      ! (i,j,n)
      type(vector_t),dimension(:,:,:),allocatable::U
      ! (i,j)
      logical,dimension(:,:),allocatable::Z
      !
      character(str_len)::correlation_map
   end type

end module PIV_common_mod
```

PIV.f95

```
1  module PIV_mod
2     use PIV_common_mod
3     use system_mod
4     use vector_mod
5     implicit none
6
7     private
8
9     type(pass_t),dimension(:),allocatable::passes
10    character(2*str_len)::image_directory
11    character(2*str_len)::vector_directory
12    integer::region_spacing
13
14    public::inititlize_PIV
15    public::finalize_PIV
16    public::process_pair
17
18 contains
19
20    subroutine inititlize_PIV
21       use config_mod
22       use string_mod
23
24       integer::N
25       integer::k
26       type(string_t)::key
27
28       write(*,*) character('['+mpi_rank+']␣'+'Initializing␣PIV␣module')
29       n = getInteger('Passes')
30       if(allocated(passes)) deallocate(passes)
31       allocate(passes(N))
32
33       do k=1,N
34          key = 'Pass('+k+').Correlation.RegionSize'
35          passes(k)%region_size = getInteger(character(key))
36          write(*,*) character('['+mpi_rank+']␣'+'␣␣'+key+'␣=␣'+passes(k)%
               region_size)
37
38          key = 'Pass('+k+').Correlation.Shift'
39          passes(k)%shift = trim(adjustl(getString(character(key))))
40          write(*,*) character('['+mpi_rank+']␣'+'␣␣'+key+'␣=␣'+trim(passes
               (k)%shift))
41
42          key = 'Pass('+k+').Correlation.Normalize'
43          passes(k)%normalize = trim(adjustl(getString(character(key))))
44          write(*,*) character('['+mpi_rank+']␣'+'␣␣'+key+'␣=␣'+trim(passes
               (k)%normalize))
45
46          key = 'Pass('+k+').Correlation.Window'
47          passes(k)%window = trim(adjustl(getString(character(key))))
48          write(*,*) character('['+mpi_rank+']␣'+'␣␣'+key+'␣=␣'+trim(passes
               (k)%window))
49
50          key = 'Pass('+k+').Correlation.Method'
```

```
51          passes(k)%correlate = trim(adjustl(getString(character(key))))
52          write(*,*) character('['+mpi_rank+']␣'+'␣␣'+key+'␣=␣'+trim(passes
               (k)%correlate))
53
54          key = 'Pass('+k+').Correlation.Subpixel'
55          passes(k)%subpixel = trim(adjustl(getString(character(key))))
56          write(*,*) character('['+mpi_rank+']␣'+'␣␣'+key+'␣=␣'+trim(passes
               (k)%subpixel))
57
58          key = 'Pass('+k+').Postprocess.Vectors'
59          passes(k)%postprocess = trim(adjustl(getString(character(key))))
60          write(*,*) character('['+mpi_rank+']␣'+'␣␣'+key+'␣=␣'+trim(passes
               (k)%postprocess))
61
62          if(k<N) write(*,*) character('['+mpi_rank+']')
63       end do
64
65       image_directory = getString('ImageDirectory')
66       vector_directory = getString('VectorDirectory')
67       region_spacing = getInteger('Correlation.RegionSpacing')
68
69       call free_string(key)
70       write(*,*) character('['+mpi_rank+']␣'+'Done␣initializing␣PIV␣module
            ')
71    end subroutine inititlize_PIV
72
73    subroutine finalize_PIV
74       use config_mod
75       use string_mod
76
77       write(*,*) character('['+mpi_rank+']␣'+'Finalizing␣PIV␣module')
78       if(allocated(passes)) deallocate(passes)
79       write(*,*) character('['+mpi_rank+']␣'+'Done␣finalizing␣PIV␣module')
80    end subroutine finalize_PIV
81
82    subroutine read_pair(n,pair)
83       use config_mod
84       use string_mod
85       use im7_mod, read_im7 => read_image
86       use image_mod, read_other => read_image
87       use generator_mod
88       integer,intent(in)::n
89       type(pair_t),intent(inout)::pair
90
91       character(128)::files,file1,file2
92       type(string_t)::key
93       real(wp),dimension(:,:),allocatable::buf
94       integer::ni,nj
95       integer::i,j
96       integer::max_rs
97       real(wp)::mn,std
98
99       ! Read images
100      key = 'Pair('+n+').Images'
101      files = adjustl(trim(getString(character(key))))
```

```
102         key = 'Pair('+n+').Image(1)'
103         file1 = adjustl(trim(getString(character(key))))
104         key = 'Pair('+n+').Image(2)'
105         file2 = adjustl(trim(getString(character(key))))
106
107         if(files/='-1') then
108            if(ends_with(files,'im7') .or. ends_with(files,'IM7')) then
109               call read_im7(trim(image_directory)//'/'//files,pair%I1,pair%
                     I2)
110
111               if(.not.allocated(pair%I1) .or. .not.allocated(pair%I2)) &
112               & call doFatalError(character('Too␣few␣images␣in␣multi-image␣
                     file␣in␣pair␣number␣'+n))
113            else if(ends_with(files,'gen')) then
114               call gen_pair(pair%I1,pair%I2)
115            else
116               call doFatalError(character('Wrong␣filetype␣for␣multi-image␣
                     file␣in␣pair␣number␣'+n))
117            end if
118         else if(file1/='-1' .and. file2/='-1') then
119            if(ends_with(file1,'im7') .or. ends_with(file1,'IM7')) then
120               call read_im7(trim(image_directory)//'/'//file1,pair%I1,buf)
121               if(allocated(buf)) deallocate(buf)
122            else
123               call read_other(trim(image_directory)//'/'//file1,pair%I1)
124            end if
125            if(.not.allocated(pair%I1)) call doFatalError(character('Error␣
                  reading␣Image(1)␣in␣pair␣number␣'+n))
126
127            if(ends_with(file2,'im7') .or. ends_with(file2,'IM7')) then
128               call read_im7(trim(image_directory)//'/'//file2,pair%I2,buf)
129               if(allocated(buf)) deallocate(buf)
130            else
131               call read_other(trim(image_directory)//'/'//file2,pair%I2)
132            end if
133            if(.not.allocated(pair%I1)) call doFatalError(character('Error␣
                  reading␣Image(2)␣in␣pair␣number␣'+n))
134         else
135            call doFatalError(character('Images␣not␣specified␣for␣image␣pair␣
                  number␣'+n))
136         end if
137
138         ! Normalize Images
139         mn = sum(pair%I1)/real(size(pair%I1),wp)
140         std = sqrt(sum((pair%I1-mn)**2)/real(size(pair%I1)-1,wp))
141         pair%I1 = (pair%I1-mn)/std
142         mn = sum(pair%I2)/real(size(pair%I2),wp)
143         std = sqrt(sum((pair%I2-mn)**2)/real(size(pair%I2)-1,wp))
144         pair%I2 = (pair%I2-mn)/std
145
146         ! Allocate X
147         max_rs = maxval(passes(:)%region_size)
148         ni = (size(pair%I1,1)-max_rs)/region_spacing
149         nj = (size(pair%I1,2)-max_rs)/region_spacing
150         if(allocated(pair%X)) deallocate(pair%X)
```

```
151         allocate(pair%X(ni,nj))
152
153         ! Compute X
154         do i=1,ni
155            do j=1,nj
156               pair%X(i,j) = real([max_rs/2+mod(max_rs,2)+region_spacing*(i
                     -1),max_rs/2+mod(max_rs,2)+region_spacing*(j-1)],wp)
157            end do
158         end do
159
160         ! Allocate U
161         if(allocated(pair%U)) deallocate(pair%U)
162         allocate(pair%U(ni,nj,size(passes)))
163         pair%U(:,:,:)%x(1) = 0.0_wp
164         pair%U(:,:,:)%x(2) = 0.0_wp
165
166         ! Allocate Z
167         if(allocated(pair%Z)) deallocate(pair%Z)
168         allocate(pair%Z(ni,nj))
169
170         ! Read correlation_map
171         key = 'Pair('+n+').Map'
172         pair%correlation_map = trim(adjustl(getString(character(key))))
173
174         call free_string(key)
175      end subroutine read_pair
176
177      subroutine write_pair(n,pair)
178         use config_mod
179         use string_mod
180         use vtk_io_mod
181         use image_mod
182         integer,intent(in)::n
183         type(pair_t),intent(inout)::pair
184
185         type(string_t)::key
186         character(2*str_len)::fn,sfn
187         integer::i,j,k
188
189         key = 'Pair('+n+').Vectors'
190         fn = trim(vector_directory)//'/'//getString(character(key))
191         if(ends_with(fn,'vtk')) then
192            open(100,file=fn,status='unknown')
193            call vtk_write_header(100,character(key))
194            ! Write X
195            call vtk_write_grid(100,pair%X(:,:)%x(1),size(pair%I1,2)-pair%X
                  (:,:)%x(2))
196            ! Write U
197            do k=1,size(passes)
198               where( pair%U(:,:,k)%x(1)/=pair%U(:,:,k)%x(1)) pair%U(:,:,k)%x
                     (1) = 0.0_wp
199               where( pair%U(:,:,k)%x(2)/=pair%U(:,:,k)%x(2)) pair%U(:,:,k)%x
                     (2) = 0.0_wp
200               call vtk_write_vector(100,character('U_Pass('+k+')'),pair%U
                     (:,:,k)%x(1),-pair%U(:,:,k)%x(2))
```

```
201            end do
202            close(100)
203         elseif(ends_with(fn,'plt')) then
204            open(100,file=fn,status='unknown')
205            pair%U(:,:,:)%x(2) = -pair%U(:,:,:)%x(2)
206            do i=lbound(pair%X,1),ubound(pair%X,1)
207               do j=lbound(pair%X,2),ubound(pair%X,2)
208                  write(100,*) pair%X(i,j), pair%U(i,j,:)
209               end do
210            end do
211            pair%U(:,:,:)%x(2) = -pair%U(:,:,:)%x(2)
212            close(100)
213         elseif(ends_with(fn,'dat')) then
214            ! ASCII
215         elseif(ends_with(fn,'bin')) then
216            ! Binary
217            open(100,file=fn,status='unknown',form='unformatted')
218            write(100) size(pair%U,1),size(pair%U,2)
219            pair%U(:,:,:)%x(2) = -pair%U(:,:,:)%x(2)
220            do k=1,size(passes)
221               write(100) pair%U(:,:,k)
222            end do
223            pair%U(:,:,:)%x(2) = -pair%U(:,:,:)%x(2)
224            close(100)
225         end if
226
227         key = 'Pair('+n+').Image(1).export'
228         sfn = getString(character(key))
229         if(sfn/='-1') then
230            fn = trim(image_directory)//'/'//sfn
231            call write_image(fn,pair%I1)
232         end if
233         key = 'Pair('+n+').Image(2).export'
234         sfn = getString(character(key))
235         if(sfn/='-1') then
236            fn = trim(image_directory)//'/'//sfn
237            call write_image(fn,pair%I2)
238         end if
239
240         call free_string(key)
241      end subroutine write_pair
242
243      subroutine free_pair(pair)
244         type(pair_t),intent(inout)::pair
245
246         if(allocated(pair%I1)) deallocate(pair%I1)
247         if(allocated(pair%I2)) deallocate(pair%I2)
248         if(allocated(pair%U)) deallocate(pair%U)
249         if(allocated(pair%X)) deallocate(pair%X)
250      end subroutine free_pair
251
252      subroutine execute_pass(n,pair)
253         use string_mod
254         use shift_mod
255         use normalize_mod
```

```
256         use window_mod
257         use correlate_mod
258         use subpixel_mod
259         use postprocess_mod
260         use image_mod
261
262         integer,intent(in)::n
263         type(pair_t),intent(inout)::pair
264
265         integer::i,j
266         real(wp),dimension(passes(n)%region_size,passes(n)%region_size)::R1,
               R2,CM,W,FI
267         type(string_t)::fn
268
269         W = 1.0_wp
270         ! Window
271         if(passes(n)%window=='NONE') then
272            W = window_none(W)
273         else if(passes(n)%window=='GAUSSIAN') then
274            W = window_gaussian(W)
275         else if(passes(n)%window=='QUADRATIC') then
276            W = window_quadratic(W)
277         end if
278
279         ! FFT Normalization
280         if(passes(n)%correlate=='FFT') then
281            FI = normalize_fft(FI)
282         end if
283
284         ! Zero mask
285         pair%Z = .false.
286
287         do i=1,size(pair%X,1)
288            do j=1,size(pair%X,2)
289
290               ! Shift
291               if(passes(n)%shift=='NONE') then
292                  call shift_none(pair,[i,j,n],R1,R2)
293               else if(passes(n)%shift=='LIMITED1') then
294                  call shift_limited1(pair,[i,j,n],maxval(passes(:)%
                        region_size)/4,R1,R2)
295               else if(passes(n)%shift=='FULL1') then
296                  call shift_full1(pair,[i,j,n],R1,R2)
297               else if(passes(n)%shift=='LIMITED2') then
298                  call shift_limited2(pair,[i,j,n],maxval(passes(:)%
                        region_size)/4,R1,R2)
299               else if(passes(n)%shift=='FULL2') then
300                  call shift_full2(pair,[i,j,n],R1,R2)
301               end if
302
303               ! Normalize
304               if(passes(n)%normalize=='NONE') then
305                  R1 = normalize_none(R1)
306                  R2 = normalize_none(R2)
307               else if(passes(n)%normalize=='STD') then
```

```
308                R1 = normalize_std(R1)
309                R2 = normalize_std(R2)
310             end if
311
312             ! Window
313             R1 = R1*W
314
315             !Correlate
316             if(passes(n)%correlate=='FFT') then
317                CM = correlate_fft(R1,R2)
318             else if(passes(n)%correlate=='DIRECT') then
319                CM = correlate_direct(R1,R2)
320             end if
321
322             if(pair%correlation_map/='-1') then
323                fn = trim(adjustl(vector_directory))+'/'+trim(adjustl(pair%
                      correlation_map))+'_'+n+'_'+i+','+j+'.png'
324                call write_image(character(fn),CM)
325             end if
326
327             ! Subpixel
328             if(passes(n)%subpixel=='NONE') then
329                pair%U(i,j,n) = subpixel_none(CM)
330             else if(passes(n)%subpixel=='GAUSSIAN') then
331                pair%U(i,j,n) = subpixel_gaussian(CM)
332             end if
333             if(passes(n)%shift/='NONE' .and. .not.pair%Z(i,j)) then
334                pair%U(i,j,n) = pair%U(i,j,n)+real(nint(pair%U(i,j,n-1)%x),
                      wp)
335             end if
336
337          end do
338       end do
339
340       ! Postprocess
341       if(passes(n)%postprocess=='NONE') then
342          do i=2,size(pair%X,1)-1
343             do j=2,size(pair%X,2)-1
344                pair%U(i,j,n) = postprocess_none(pair%U(i-1:i+1,j-1:j+1,n))
345             end do
346          end do
347       else if(passes(n)%postprocess=='STD') then
348          do i=2,size(pair%X,1)-1
349             do j=2,size(pair%X,2)-1
350                pair%U(i,j,n) = postprocess_std(pair%U(i-1:i+1,j-1:j+1,n)%x
                      (1),pair%U(i-1:i+1,j-1:j+1,n)%x(2))
351             end do
352          end do
353       else if(passes(n)%postprocess=='MEDIAN') then
354          ! Inner
355          do i=2,size(pair%X,1)-1
356             do j=2,size(pair%X,2)-1
357                pair%U(i,j,n) = postprocess_median(pair%U(i-1:i+1,j-1:j+1,n
                      ))
358             end do
```

```
359           end do
360           !Edges
361           do i=2,size(pair%X,1)-1
362              j = 1
363              pair%U(i,j,n) = postprocess_median(pair%U(i-1:i+1,j:j+1,n))
364              j = size(pair%X,2)
365              pair%U(i,j,n) = postprocess_median(pair%U(i-1:i+1,j-1:j,n))
366           end do
367           do j=2,size(pair%X,2)-1
368              i = 1
369              pair%U(i,j,n) = postprocess_median(pair%U(i:i+1,j-1:j+1,n))
370              i = size(pair%X,1)
371              pair%U(i,j,n) = postprocess_median(pair%U(i-1:i,j-1:j+1,n))
372           end do
373           ! Corners
374           i = 1
375           j = 1
376           pair%U(i,j,n) = postprocess_median(pair%U(i:i+1,j:j+1,n))
377           i = 1
378           j = size(pair%X,2)
379           pair%U(i,j,n) = postprocess_median(pair%U(i:i+1,j-1:j,n))
380           i = size(pair%X,1)
381           j = size(pair%X,2)
382           pair%U(i,j,n) = postprocess_median(pair%U(i-1:i,j-1:j,n))
383           i = size(pair%X,1)
384           j = 1
385           pair%U(i,j,n) = postprocess_median(pair%U(i-1:i,j:j+1,n))
386        end if
387
388        call free_string(fn)
389     end subroutine execute_pass
390
391     subroutine process_pair(n)
392        use string_mod
393        integer,intent(in)::n
394
395        type(pair_t)::pair
396        integer::k
397
398        write(*,*) character('['+mpi_rank+']␣'+'Processing␣Pair('+n+')')
399        call read_pair(n,pair)
400        do k=1,size(passes)
401           if(mpi_size==1)  write(*,*) character('['+mpi_rank+']␣'+'␣␣␣
              Executing␣Pass('+k+')')
402           call execute_pass(k,pair)
403        end do
404        call write_pair(n,pair)
405        call free_pair(pair)
406        if(mpi_size==1)  write(*,*) character('['+mpi_rank+']␣'+'Completed␣
           Pair('+n+')')
407     end subroutine process_pair
408
409  end module PIV_mod
```

postprocess.f95

```
module postprocess_mod
   use system_mod
   use vector_mod
   implicit none

contains

   function postprocess_none(U) result(o)
      type(vector_t),dimension(:,:),intent(in)::U
      type(vector_t)::o

      o = U((size(U,1)+1)/2,(size(U,2)+1)/2)
   end function postprocess_none

   function postprocess_median(U) result(o)
      type(vector_t),dimension(:,:),intent(in)::U
      type(vector_t)::o

      real(wp),dimension(size(U),3)::Us
      type(vector_t)::mu,std
      type(vector_t)::v,m,d

      v = U((size(U,1)+1)/2,(size(U,2)+1)/2)

      Us(:,1) = reshape(abs(U),[size(U)])
      Us(:,2) = reshape(U(:,:)%x(1),[size(U)])
      Us(:,3) = reshape(U(:,:)%x(2),[size(U)])

      Us = sort(Us)

      mu = sum(Us(2:size(Us,1)-1,2:3),1)/real(size(Us,1)-2,wp)
      std%x(1) = sqrt(sum((Us(2:size(Us,1)-1,1)-mu%x(1)))**2)/real(size(Us
         ,1)-2,wp)
      std%x(2) = sqrt(sum((Us(2:size(Us,1)-1,1)-mu%x(2)))**2)/real(size(Us
         ,1)-2,wp)

      m = Us(size(U)/2,2:3)

      d = abs(v%x-m%x)
      if(any(d%x>1.96_wp*std%x)) then
         o = m
      else
         o = v
      end if
   end function postprocess_median

   function postprocess_std(us,vs) result(o)
      real(wp),dimension(3,3),intent(in)::us,vs
      real(wp),dimension(2)::o

      real(wp)::au,du,av,dv
      real(wp),dimension(2)::d

      au = sum(us)/real(size(us),wp)
```

```
53        av = sum(vs)/real(size(vs),wp)
54
55        du = sqrt(sum((us-au)**2)/real(size(us),wp))
56        dv = sqrt(sum((vs-av)**2)/real(size(vs),wp))
57
58        d = ([us(2,2),vs(2,2)]-[au,av])/[du,dv]
59        if(sqrt(sum(d**2))>2.0_wp) then
60           o = [au,av]
61        else
62           o = [us(2,2),vs(2,2)]
63        end if
64     end function postprocess_std
65
66  end module postprocess_mod
```

shift.f95

```
module shift_mod
   use system_mod
   use PIV_common_mod
   implicit none

contains

   subroutine shift_full2(pair,loc,R1,R2)
      use string_mod
      type(pair_t),intent(inout)::pair
      integer,dimension(3)::loc
      real(wp),dimension(:,:),intent(out)::R1
      real(wp),dimension(size(R1,1),size(R1,2)),intent(out)::R2

      integer::i,j,n
      integer::xmin1,xmin2,xmax1,xmax2
      integer::ymin1,ymin2,ymax1,ymax2
      integer::dx1,dx2,dy1,dy2
      integer::Nx,Ny
      logical::left

      left = .false.

      i = loc(1)
      j = loc(2)
      n = loc(3)

      Nx = size(R1,1)
      Ny = size(R1,2)

      dx1 = floor(real(nint(pair%U(i,j,n-1)%x(1)),wp)/2.0_wp)
      dx2 = ceiling(real(nint(pair%U(i,j,n-1)%x(1)),wp)/2.0_wp)

      xmin1 = nint(pair%X(i,j)%x(1))-dx1-Nx/2+1
      xmax1 = xmin1+Nx-1
      xmin2 = nint(pair%X(i,j)%x(1))+dx2-Nx/2+1
      xmax2 = xmin2+Nx-1

      dy1 = floor(real(nint(pair%U(i,j,n-1)%x(2)),wp)/2.0_wp)
      dy2 = ceiling(real(nint(pair%U(i,j,n-1)%x(2)),wp)/2.0_wp)

      ymin1 = nint(pair%X(i,j)%x(2))-dy1-Ny/2+1
      ymax1 = ymin1+Ny-1
      ymin2 = nint(pair%X(i,j)%x(2))+dy2-Ny/2+1
      ymax2 = ymin2+Ny-1

      if(min(xmin1,xmin2,ymin1,ymin2)<1) left = .true.
      if(max(xmax1,xmax2)>size(pair%I1,1)) left = .true.
      if(max(ymax1,ymax2)>size(pair%I1,2)) left = .true.

      if(.not. left) then
         R1 = pair%I1(xmin1:xmax1,ymin1:ymax1)
         R2 = pair%I2(xmin2:xmax2,ymin2:ymax2)
      else
```

```
55          if(mpi_size/=1) write(*,*) character('['+mpi_rank+']␣'+'␣␣␣␣
               Warning,␣vector␣('+i+','+j+')␣attempted␣to␣leave␣image')
56          call shift_limited2(pair,loc,size(R1,1)/4,R1,R2)
57       end if
58    end subroutine shift_full2
59
60    subroutine shift_limited2(pair,loc,limit,R1,R2)
61       type(pair_t),intent(inout)::pair
62       integer,dimension(3)::loc
63       integer,intent(in)::limit
64       real(wp),dimension(:,:),intent(out)::R1
65       real(wp),dimension(size(R1,1),size(R1,2)),intent(out)::R2
66
67       integer::i,j,n
68       integer::xmin,xmax
69       integer::ymin,ymax
70       integer::dx,dy
71
72       i = loc(1)
73       j = loc(2)
74       n = loc(3)
75
76       dx = nint(pair%U(i,j,n-1)%x(1)/2.0_wp)
77       if(abs(dx)>limit) then
78          pair%U(i,j,n-1)%x(1) = 0.0_wp
79          dx = 0
80       end if
81
82       dy = nint(pair%U(i,j,n-1)%x(2)/2.0_wp)
83       if(abs(dy)>limit) then
84          pair%U(i,j,n-1)%x(2) = 0.0_wp
85          dy = 0
86       end if
87
88       xmin = pair%X(i,j)%x(1)-size(R1,1)/2+mod(size(R1,1)/2+1,2)
89       xmax = xmin+size(R1,1)-1
90
91       ymin = pair%X(i,j)%x(2)-size(R1,2)/2+mod(size(R1,2)/2+1,2)
92       ymax = ymin+size(R1,2)-1
93
94       R1 = pair%I1(xmin-dx:xmax-dx,ymin-dy:ymax-dy)
95       R2 = pair%I2(xmin+dx:xmax+dx,ymin+dy:ymax+dy)
96    end subroutine shift_limited2
97
98    subroutine shift_full1(pair,loc,R1,R2)
99       use string_mod
100      type(pair_t),intent(inout)::pair
101      integer,dimension(3)::loc
102      real(wp),dimension(:,:),intent(out)::R1
103      real(wp),dimension(size(R1,1),size(R1,2)),intent(out)::R2
104
105      integer::i,j,n
106      integer::xmin1,xmin2,xmax1,xmax2
107      integer::ymin1,ymin2,ymax1,ymax2
108      integer::dx1,dx2,dy1,dy2
```

```
109         integer::Nx,Ny
110         logical::left
111
112         left = .false.
113
114         i = loc(1)
115         j = loc(2)
116         n = loc(3)
117
118         Nx = size(R1,1)
119         Ny = size(R1,2)
120
121         dx1 = 0
122         dx2 = nint(pair%U(i,j,n-1)%x(1))
123
124         xmin1 = nint(pair%X(i,j)%x(1))-dx1-Nx/2+1
125         xmax1 = xmin1+Nx-1
126         xmin2 = nint(pair%X(i,j)%x(1))+dx2-Nx/2+1
127         xmax2 = xmin2+Nx-1
128
129         dy1 = 0
130         dy2 = nint(pair%U(i,j,n-1)%x(2))
131
132         ymin1 = nint(pair%X(i,j)%x(2))-dy1-Ny/2+1
133         ymax1 = ymin1+Ny-1
134         ymin2 = nint(pair%X(i,j)%x(2))+dy2-Ny/2+1
135         ymax2 = ymin2+Ny-1
136
137
138         if(min(xmin1,xmin2,ymin1,ymin2)<1) left = .true.
139         if(max(xmax1,xmax2)>size(pair%I1,1)) left = .true.
140         if(max(ymax1,ymax2)>size(pair%I1,2)) left = .true.
141
142         if(.not. left) then
143            R1 = pair%I1(xmin1:xmax1,ymin1:ymax1)
144            R2 = pair%I2(xmin2:xmax2,ymin2:ymax2)
145         else
146            if(mpi_size==1) write(*,*) character('['+mpi_rank+']␣'+'␣␣␣␣
                  Warning,␣vector␣('+i+','+j+')␣attempted␣to␣leave␣image')
147            call shift_limited1(pair,loc,size(R1,1)/4,R1,R2)
148         end if
149      end subroutine shift_full1
150
151      subroutine shift_limited1(pair,loc,limit,R1,R2)
152         type(pair_t),intent(inout)::pair
153         integer,dimension(3)::loc
154         integer,intent(in)::limit
155         real(wp),dimension(:,:),intent(out)::R1
156         real(wp),dimension(size(R1,1),size(R1,2)),intent(out)::R2
157
158         integer::i,j,n
159         integer::xmin,xmax
160         integer::ymin,ymax
161         integer::dx,dy
162
```

```
163         i = loc(1)
164         j = loc(2)
165         n = loc(3)
166
167         dx = nint(pair%U(i,j,n-1)%x(1))
168         if(abs(dx)>limit) then
169            pair%U(i,j,n-1)%x(1) = 0.0_wp
170            dx = 0
171         end if
172
173         dy = nint(pair%U(i,j,n-1)%x(2))
174         if(abs(dy)>limit) then
175            pair%Z(i,j) = .true.
176            dy = 0
177         end if
178
179         xmin = pair%X(i,j)%x(1)-size(R1,1)/2+mod(size(R1,1)/2+1,2)
180         xmax = xmin+size(R1,1)-1
181
182         ymin = pair%X(i,j)%x(2)-size(R1,2)/2+mod(size(R1,2)/2+1,2)
183         ymax = ymin+size(R1,2)-1
184
185         R1 = pair%I1(xmin-dx:xmax,ymin-dy:ymax)
186         R2 = pair%I2(xmin+dx:xmax+dx,ymin+dy:ymax+dy)
187      end subroutine shift_limited1
188
189      subroutine shift_none(pair,loc,R1,R2)
190         type(pair_t),intent(in)::pair
191         integer,dimension(3)::loc
192         real(wp),dimension(:,:),intent(out)::R1
193         real(wp),dimension(size(R1,1),size(R1,2)),intent(out)::R2
194
195         integer::i,j,n
196         integer::xmin,xmax
197         integer::ymin,ymax
198
199         i = loc(1)
200         j = loc(2)
201         n = loc(3)
202
203         xmin = pair%X(i,j)%x(1)-size(R1,1)/2+mod(size(R1,1)/2+1,2)
204         xmax = xmin+size(R1,1)-1
205
206         ymin = pair%X(i,j)%x(2)-size(R1,2)/2+mod(size(R1,2)/2+1,2)
207         ymax = ymin+size(R1,2)-1
208
209         R1 = pair%I1(xmin:xmax,ymin:ymax)
210         R2 = pair%I2(xmin:xmax,ymin:ymax)
211      end subroutine shift_none
212
213   end module shift_mod
```

statistics.f95

```
program statistics
   use system_mod
   use config_mod
   use vector_mod
   use string_mod
   implicit none

   integer::Ni,Np,Nx,Ny
   real(wp),dimension(2)::uniform

   type(vector_t),dimension(:,:,:,:),allocatable::vecs
   type(vector_t),dimension(:),allocatable::mean,std

   call initialize
   call read_data
   call compute_stats
   call write_stats

contains

   subroutine initialize
      type(string_t)::fn
      character(128)::cfn

      call get_command_argument(1,cfn)
      if(cfn=='') call doFatalError('No config file specified')
      call read_config(cfn)

      Ni = getInteger('Pairs')
      Np = getInteger('Passes')
      fn = trim(adjustl(getString('VectorDirectory')))
      fn = fn+'/'+trim(adjustl(getString('Pair(1).Vectors')))

      open(100,file=character(fn),status='old',form='unformatted')
      read(100) Nx,Ny
      close(100)

      allocate(vecs(Ni,Np,Nx,Ny))

      uniform(1) = getReal('Generator.Ux')
      uniform(2) = getReal('Generator.Uy')

      call free_string(fn)
   end subroutine initialize

   subroutine read_data
      type(string_t)::vd,key,fn
      integer::i,p
      integer::w,h

      vd = trim(adjustl(getString('VectorDirectory')))

      do i=1,Ni
         key = 'Pair('+i+').Vectors'
```

```
55          fn = getString(character(key))
56          open(100,file=character(vd+'/'+fn),status='old',form='unformatted
               ')
57          read(100) w,h
58          do p=1,Np
59             read(100) vecs(i,p,:,:)
60          end do
61          close(100)
62       end do
63
64       call free_string(vd)
65       call free_string(fn)
66    end subroutine read_data
67
68    subroutine compute_stats
69       integer::p
70       real(wp)::N
71
72       allocate(mean(Np))
73       allocate(std(Np))
74
75       mean(:)%x(1) = 0.0_wp
76       mean(:)%x(2) = 0.0_wp
77       std(:)%x(1) = 0.0_wp
78       std(:)%x(2) = 0.0_wp
79
80       N = real(Ni*Nx*Ny,wp)
81
82       do p=1,Np
83          mean(p)%x(1) = sum(vecs(:,p,:,:)%x(1))/N
84          mean(p)%x(2) = sum(vecs(:,p,:,:)%x(2))/N
85
86          std(p)%x(1) = sum((vecs(:,p,:,:)%x(1)-mean(p)%x(1))**2)/N
87          std(p)%x(2) = sum((vecs(:,p,:,:)%x(2)-mean(p)%x(2))**2)/N
88       end do
89    end subroutine compute_stats
90
91    subroutine write_stats
92       character(128)::cfn
93       integer::p
94
95       call get_command_argument(2,cfn)
96
97       open(100,file=trim(adjustl(cfn)),status='unknown')
98       do p=1,Np
99          write(100,*) uniform,mean(p),std(p),Ni*Nx*Ny
100      end do
101      close(100)
102   end subroutine write_stats
103end program statistics
```

string.f95

```
1  module generator_mod
2     use system_mod
3     use config_mod
4     implicit none
5
6     private
7
8     integer::Np
9     integer,dimension(2)::sz
10    real(wp)::d
11    real(wp)::l
12
13    type::particle_t
14       real(wp),dimension(3)::x
15       real(wp)::d
16    end type
17
18    real(wp),dimension(3)::uniform
19
20    public::initialize_generator
21    public::gen_pair
22
23 contains
24
25    subroutine initialize_generator
26       use string_mod
27       integer,dimension(:),allocatable::seed
28       integer::seed_size
29
30       real(wp)::rho
31
32       write(*,*) character('['+mpi_rank+']␣Initializing␣generator␣module')
33
34       rho = getReal('Generator.Density')
35       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Density␣=␣'+rho)
36       sz = [getInteger('Generator.Width'),getInteger('Generator.Height')]
37       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Width␣=␣'+sz(1))
38       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Height␣=␣'+sz(2))
39       d = getReal('Generator.Diameter')
40       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Diameter␣=␣'+d)
41       l = getReal('Generator.NoiseLevel')
42       write(*,*) character('['+mpi_rank+']␣␣␣Generator.NoiseLevel␣=␣'+l)
43
44       uniform(1) = getReal('Generator.Ux')
45       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Ux␣=␣'+uniform(1))
46       uniform(2) = getReal('Generator.Uy')
47       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Uy␣=␣'+uniform(2))
48       uniform(3) = getReal('Generator.Uz')
49       write(*,*) character('['+mpi_rank+']␣␣␣Generator.Uz␣=␣'+uniform(3))
50
51       uniform(2) = -uniform(2)
52
53       Np = nint((1.2_wp*sz(1))*(1.2_wp*sz(2))*rho)
54       call random_seed(size=seed_size)
```

```
55        allocate(seed(seed_size))
56        seed = mpi_rank
57        call random_seed(put=seed)
58        deallocate(seed)
59
60        write(*,*) character('['+mpi_rank+']␣Done␣initializing␣generator␣
              module')
61     end subroutine initialize_generator
62
63     subroutine gen_pair(IM1,IM2)
64        real(wp),dimension(:,:),allocatable,intent(inout)::IM1,IM2
65
66        type(particle_t),dimension(Np)::particles
67        integer::k
68
69        if(allocated(IM1)) deallocate(IM1)
70        allocate(IM1(sz(1),sz(2)))
71        if(allocated(IM2)) deallocate(IM2)
72        allocate(IM2(sz(1),sz(2)))
73
74        call gen_particles(sz(1),sz(2),d,particles)
75
76        call random_number(IM1)
77        IM1 = l*IM1
78        call random_number(IM2)
79        IM2 = l*IM2
80
81        do k=1,size(particles)
82           call add_particle(particles(k),IM1)
83        end do
84
85
86        do k=1,size(particles)
87           call disp_uniform(particles(k))
88           call add_particle(particles(k),IM2)
89        end do
90     end subroutine gen_pair
91
92     subroutine gen_particles(lx,ly,d,particles)
93        integer,intent(in)::lx,ly
94        real(wp),intent(in)::d
95        type(particle_t),dimension(:),intent(inout)::particles
96
97        call random_number(particles(:)%x(1))
98        call random_number(particles(:)%x(2))
99        call random_number(particles(:)%x(3))
100
101       particles(:)%x(1) = real(1.2_wp*lx,wp)*particles(:)%x(1)-0.1_wp*lx
102       particles(:)%x(2) = real(1.2_wp*ly,wp)*particles(:)%x(2)-0.1_wp*ly
103       particles(:)%x(3) = 2.0_wp*particles(:)%x(3)-1.0_wp
104
105       particles(:)%d = d
106    end subroutine gen_particles
107
108    subroutine add_particle(p,IM)
```

```
109         type(particle_t),intent(in)::p
110         real(wp),dimension(:,:),intent(inout)::IM
111 
112         integer::pi,pj
113         integer::r
114 
115         integer::i,j
116         integer::i1,i2
117         integer::j1,j2
118 
119         pi = nint(p%x(1))
120         pj = nint(p%x(2))
121         r = nint(1.96_wp*p%d)
122 
123         i1 = max(pi-r,1)
124         i2 = min(pi+r,size(IM,1))
125 
126         j1 = max(pj-r,1)
127         j2 = min(pj+r,size(IM,2))
128 
129         forall(i=i1:i2,j=j1:j2)
130            IM(i,j) = IM(i,j)+intensity(p,i,j)
131         end forall
132      end subroutine add_particle
133 
134      pure function intensity(p,i,j) result(o)
135         type(particle_t),intent(in)::p
136         integer,intent(in)::i,j
137         real(wp)::o
138 
139         real(wp)::dx,dy,r
140 
141         dx = (real(i,wp)-p%x(1))/p%d
142         dy = (real(j,wp)-p%x(2))/p%d
143         r = sqrt(dx**2+dy**2)
144 
145         ! Distance
146         o = 1.0_wp/((16.0_wp*r**4+1.0_wp)*(4.0_wp*r**2+1.0_wp))
147         ! Depth
148         o = o*1.0_wp/((16.0_wp*p%x(3)**4+1.0_wp)*(4.0_wp*p%x(3)**2+1.0_wp))
149      end function intensity
150 
151      subroutine disp_none(p)
152         type(particle_t),intent(inout)::p
153         p%x = p%x+0.0_wp
154      end subroutine disp_none
155 
156      subroutine disp_uniform(p)
157         type(particle_t),intent(inout)::p
158 
159         p%x = p%x+uniform
160      end subroutine disp_uniform
161 
162  end module generator_mod
```

subpixel.f95

```
module subpixel_mod
   use system_mod
   implicit none

contains

   function subpixel_none(c) result(o)
      real(wp),dimension(:,:),intent(in)::c
      real(wp),dimension(2)::o

      integer,dimension(2)::l

      l = maxloc(c(2:size(c,1)-1,2:size(c,2)-1))+[1,1]

      o = real(l-[size(c,1),size(c,2)]/2-1,wp)
   end function subpixel_none

   function subpixel_gaussian(c) result(o)
      real(wp),dimension(:,:),intent(in)::c
      real(wp),dimension(2)::o

      integer,dimension(2)::l
      real(wp),dimension(-1:1)::v
      real(wp),dimension(2)::e

      l = maxloc(c(2:size(c,1)-1,2:size(c,2)-1))+[1,1]
      v = c(l(1)-1:l(1)+1,l(2))
      v = v-min(2.0_wp*minval(v),-1.0_wp)+1.0_wp
      v = log(v)
      e(1) = 0.5_wp*(v(-1)-v(1))/(v(-1)-2.0_wp*v(0)+v(1))

      v = c(l(1),l(2)-1:l(2)+1)
      v = v-min(2.0_wp*minval(v),-1.0_wp)+1.0_wp
      v = log(v)
      e(2) = 0.5_wp*(v(-1)-v(1))/(v(-1)-2.0_wp*v(0)+v(1))

      o = real(l-[size(c,1),size(c,2)]/2-1,wp)+e
   end function subpixel_gaussian

end module subpixel_mod
```

system.f95

```
!System module to provide real kinds and other
!constants

module system_mod
   implicit none
   !Single precision
   integer,parameter::sp=kind(1.0)
   !Double precision
   integer,parameter::dp=kind(0d0)
   !Working precision
   integer,parameter::wp=dp

   !PI -- Achimedes' constant
   real(wp),parameter::PI=3.14159265359_wp
   !E -- Euler's constant
   real(wp),parameter::e=2.71828182846_wp

   !MPI global variables
   integer::mpi_rank
   integer::mpi_size

   interface operator(.re.)
      module procedure equals_rr
   end interface

   interface operator(.rne.)
      module procedure not_equals_rr
   end interface

contains

   subroutine doFatalError(msg)
      character(*),intent(in)::msg
      write(*,'(A)') msg
      stop 1
   end subroutine doFatalError

   subroutine doProgress(pc,msg,done)
      integer,intent(in),optional::pc
      character(*),intent(in),optional::msg
      logical,intent(in),optional::done
      integer,save::pcd

      if(present(msg)) then
         pcd = 0
         write(*,'(A)') msg
         write(*,'(A)') '|␣0␣-␣-␣-␣-␣-25-␣-␣-␣-␣-␣-50-␣-␣-␣-␣-␣-75-␣-␣-␣-␣
            100␣|'
         write(*,'(A)',advance='no') '|␣'
      end if

      if(present(pc).and.((pc-pcd)>=2)) then
         write(*,'(A)',advance='no') repeat('=',(pc-pcd)/2)
         pcd = pc
```

```
54          end if
55
56          if(present(done).and.(done)) then
57             write(*,'(A)') '␣|'
58          end if
59       end subroutine doProgress
60
61       function random_normal() result(o)
62          real(wp)::o
63          real(wp),dimension(12)::x
64
65          call random_number(x)
66          o = sum(x)-6.0_wp
67       end function random_normal
68
69       function random_uniform() result(o)
70          real(wp)::o
71
72          call random_number(o)
73          o = o*2.0_wp-1.0_wp
74       end function random_uniform
75
76       elemental function arg(z) result(o)
77          complex(wp),intent(in)::z
78          real(wp)::o
79          real(wp)::x,y
80          x = real(z,wp)
81          y = aimag(z)
82          o = atan2(y,x)
83       end function arg
84
85       elemental function equals_rr(a,b) result(o)
86          real(wp),intent(in)::a,b
87          logical::o
88
89          o = abs(a-b)<100.0_wp*epsilon(1.0_wp)
90       end function equals_rr
91
92       elemental function not_equals_rr(a,b) result(o)
93          real(wp),intent(in)::a,b
94          logical::o
95
96          o = abs(a-b)>100.0_wp*epsilon(1.0_wp)
97       end function not_equals_rr
98
99       function ends_with(str,cmp) result(o)
100         character(*),intent(in)::str,cmp
101         logical::o
102
103         o = str(len_trim(str)-len(cmp)+1:len_trim(str))==cmp
104      end function ends_with
105
106      function starts_with(str,cmp) result(o)
107         character(*),intent(in)::str,cmp
108         logical::o
```

```
109
110         o = str(1:len(cmp))==cmp
111      end function starts_with
112
113      function sort(a) result(o)
114         real(wp),dimension(:,:),intent(in)::a
115         real(wp),dimension(size(a,1),size(a,2))::o
116
117         real(wp),dimension(size(a,2))::tmp
118         integer::k
119         logical::swapped
120
121         o = a
122
123         swapped = .true.
124         do while(swapped)
125            swapped = .false.
126            do k=1,size(o,1)-1
127               if(o(k,1)>o(k+1,1)) then
128                  tmp = o(k+1,:)
129                  o(k+1,:) = o(k,:)
130                  o(k,:) = tmp
131                  swapped = .true.
132               end if
133            end do
134         end do
135      end function sort
136   end module system_mod
```

vector.f95

```
1  module vector_mod
2     use system_mod
3     implicit none
4
5     private
6
7     type::vector_t
8        real(wp),dimension(2)::x
9     end type
10
11    interface abs
12       module procedure abs_v
13    end interface
14
15    interface assignment(=)
16       module procedure assign_vr
17       module procedure assign_rv
18    end interface
19
20    interface operator(+)
21       module procedure add_vv
22       module procedure add_rv
23       module procedure add_vr
24    end interface
25
26    interface operator(-)
27       module procedure sub_vv
28       module procedure sub_rv
29       module procedure sub_vr
30    end interface
31
32    interface operator(*)
33       module procedure mul_sv
34       module procedure mul_vs
35    end interface
36
37    interface operator(/)
38       module procedure div_vs
39    end interface
40
41    interface operator(.o.)
42       module procedure dot_vv
43       module procedure dot_rv
44       module procedure dot_vr
45    end interface
46
47    interface operator(.x.)
48       module procedure cross_vv
49       module procedure cross_rv
50       module procedure cross_vr
51    end interface
52
53    public::vector_t
54    public::abs
```

```
55     public::assignment(=)
56     public::operator(+)
57     public::operator(-)
58     public::operator(*)
59     public::operator(/)
60     public::operator(.o.)
61     public::operator(.x.)
62
63  contains
64
65     pure elemental function abs_v(v) result(o)
66        type(vector_t),intent(in)::v
67        real(wp)::o
68
69        o = sqrt(sum(v%x**2))
70     end function abs_v
71
72     subroutine assign_vr(v,r)
73        type(vector_t),intent(out)::v
74        real(wp),dimension(2),intent(in)::r
75
76        v%x = r
77     end subroutine assign_vr
78
79     subroutine assign_rv(r,v)
80        real(wp),dimension(2),intent(out)::r
81        type(vector_t),intent(in)::v
82
83        r = v%x
84     end subroutine assign_rv
85
86     pure elemental function add_vv(u,v) result(o)
87        type(vector_t),intent(in)::u,v
88        type(vector_t)::o
89
90        o%x = u%x+v%x
91     end function add_vv
92
93     pure function add_rv(u,v) result(o)
94        real(wp),dimension(2),intent(in)::u
95        type(vector_t),intent(in)::v
96        type(vector_t)::o
97
98        o%x = u+v%x
99     end function add_rv
100
101    pure function add_vr(u,v) result(o)
102       type(vector_t),intent(in)::u
103       real(wp),dimension(2),intent(in)::v
104       type(vector_t)::o
105
106       o%x = u%x+v
107    end function add_vr
108
109    pure elemental function sub_vv(u,v) result(o)
```

```
110         type(vector_t),intent(in)::u,v
111         type(vector_t)::o
112
113         o%x = u%x-v%x
114     end function sub_vv
115
116     pure function sub_rv(u,v) result(o)
117         real(wp),dimension(2),intent(in)::u
118         type(vector_t),intent(in)::v
119         type(vector_t)::o
120
121         o%x = u-v%x
122     end function sub_rv
123
124     pure function sub_vr(u,v) result(o)
125         type(vector_t),intent(in)::u
126         real(wp),dimension(2),intent(in)::v
127         type(vector_t)::o
128
129         o%x = u%x-v
130     end function sub_vr
131
132     pure elemental function mul_sv(s,v) result(o)
133         real(wp),intent(in)::s
134         type(vector_t),intent(in)::v
135         type(vector_t)::o
136
137         o%x = s*v%x
138     end function mul_sv
139
140     pure elemental function mul_vs(v,s) result(o)
141         type(vector_t),intent(in)::v
142         real(wp),intent(in)::s
143         type(vector_t)::o
144
145         o%x = v%x*s
146     end function mul_vs
147
148     pure elemental function div_vs(v,s) result(o)
149         type(vector_t),intent(in)::v
150         real(wp),intent(in)::s
151         type(vector_t)::o
152
153         o%x = v%x/s
154     end function div_vs
155
156     pure elemental function dot_vv(u,v) result(o)
157         type(vector_t),intent(in)::u,v
158         real(wp)::o
159
160         o = sum(u%x*v%x)
161     end function dot_vv
162
163     pure function dot_rv(u,v) result(o)
164         real(wp),dimension(2),intent(in)::u
```

```
165         type(vector_t),intent(in)::v
166         real(wp)::o
167
168         o = sum(u*v%x)
169      end function dot_rv
170
171      pure function dot_vr(u,v) result(o)
172         type(vector_t),intent(in)::u
173         real(wp),dimension(2),intent(in)::v
174         real(wp)::o
175
176         o = sum(u%x*v)
177      end function dot_vr
178
179      pure elemental function cross_vv(u,v) result(o)
180         type(vector_t),intent(in)::u,v
181         real(wp)::o
182
183         o = u%x(1)*v%x(2)-u%x(2)*v%x(1)
184      end function cross_vv
185
186      pure function cross_rv(u,v) result(o)
187         real(wp),dimension(2),intent(in)::u
188         type(vector_t),intent(in)::v
189         real(wp)::o
190
191         o = u(1)*v%x(2)-u(2)*v%x(1)
192      end function cross_rv
193
194      pure function cross_vr(u,v) result(o)
195         type(vector_t),intent(in)::u
196         real(wp),dimension(2),intent(in)::v
197         real(wp)::o
198
199         o = u%x(1)*v(2)-u%x(2)*v(1)
200      end function cross_vr
201   end module vector_mod
```

vtk_io.f95

```
module vtk_io_mod
   use system_mod
   implicit none

   private

   interface vtk_write_grid
      module procedure vtk_write_grid_2d
      module procedure vtk_write_grid_3d
   end interface

   interface vtk_write_vector
      module procedure vtk_write_vector_2d
      module procedure vtk_write_vector_3d
   end interface

   interface vtk_write_scalar
      module procedure vtk_write_scalar_2d
      module procedure vtk_write_scalar_3d
   end interface


   public::vtk_write_header
   public::vtk_write_grid
   public::vtk_write_vector
   public::vtk_write_scalar
contains
   subroutine vtk_write_header(u,title)
      integer,intent(in)::u
      character(*),intent(in)::title

      write(u,"(A)") "# vtk DataFile Version 2.0"
      write(u,"(A)") title
      write(u,"(A)") "ASCII"
      write(u,"(A)") ""
   end subroutine vtk_write_header

   subroutine vtk_write_grid_2d(u,x,y)
      integer,intent(in)::u
      real(wp),dimension(:,:),intent(in)::x,y
      integer::k1,k2

      write(u,"(A)") "DATASET STRUCTURED_GRID"
      write(u,"(A,3(I5,X))") "DIMENSIONS",size(x,1),size(x,2),1
      write(u,"(A,I10,A)") "POINTS",size(x)," double"
      do k2=lbound(x,2),ubound(x,2)
         do k1=lbound(x,1),ubound(x,1)
            write(u,"(3(ES28.15E4,X))") x(k1,k2),y(k1,k2),0.0_wp
         end do
      end do
      write(u,"(A)") ""
      write(u,"(A,I10)") "POINT_DATA",size(x)
   end subroutine vtk_write_grid_2d

```

```
55      subroutine vtk_write_vector_2d(u1,vname,u,v)
56         integer,intent(in)::u1
57         character(*),intent(in)::vname
58         real(wp),dimension(:,:),intent(in)::u,v
59         integer::k1,k2
60
61         write(u1,"(A,A,A)") "VECTORS␣",vname,"␣double"
62         do k2=lbound(u,2),ubound(u,2)
63            do k1=lbound(u,1),ubound(u,1)
64               write(u1,"(3(ES28.15E4,X))") u(k1,k2),v(k1,k2),0.0_wp
65            end do
66         end do
67      end subroutine vtk_write_vector_2d
68
69      subroutine vtk_write_scalar_2d(u,sname,phi)
70         integer,intent(in)::u
71         character(*),intent(in)::sname
72         real(wp),dimension(:,:),intent(in)::phi
73         integer::k1,k2
74
75         write(u,"(A,A,A)") "SCALARS␣",sname,"␣double"
76         write(u,"(A)") "LOOKUP_TABLE␣default"
77         do k2=lbound(phi,2),ubound(phi,2)
78            do k1=lbound(phi,1),ubound(phi,1)
79               write(u,"(ES28.15E4)") phi(k1,k2)
80            end do
81         end do
82      end subroutine vtk_write_scalar_2d
83
84      subroutine vtk_write_grid_3d(u,x,y,z)
85         integer,intent(in)::u
86         real(wp),dimension(:,:,:),intent(in)::x,y,z
87         integer::k1,k2,k3
88
89         write(u,"(A)") "DATASET␣STRUCTURED_GRID"
90         write(u,"(A,3(I5,X))") "DIMENSIONS",size(x,1),size(x,2),size(x,3)
91         write(u,"(A,I10,A)") "POINTS",size(x),"␣double"
92         do k3=lbound(x,3),ubound(x,3)
93            do k2=lbound(x,2),ubound(x,2)
94               do k1=lbound(x,1),ubound(x,1)
95                  write(u,"(3(ES28.15E4,X))") x(k1,k2,k3),y(k1,k2,k3),z(k1,k2
                       ,k3)
96               end do
97            end do
98         end do
99         write(u,"(A)") ""
100        write(u,"(A,I10)") "POINT_DATA",size(x)
101     end subroutine vtk_write_grid_3d
102
103     subroutine vtk_write_vector_3d(u1,vname,u,v,w)
104        integer,intent(in)::u1
105        character(*),intent(in)::vname
106        real(wp),dimension(:,:,:),intent(in)::u,v,w
107        integer::k1,k2,k3
108
```

```
109         write(u1,"(A,A,A)") "VECTORS␣",vname,"␣double"
110         do k3=lbound(u,3),ubound(u,3)
111            do k2=lbound(u,2),ubound(u,2)
112               do k1=lbound(u,1),ubound(u,1)
113                  write(u1,"(3(ES28.15E4,X))") u(k1,k2,k3),v(k1,k2,k3),w(k1,
                        k2,k3)
114               end do
115            end do
116         end do
117      end subroutine vtk_write_vector_3d
118
119      subroutine vtk_write_scalar_3d(u,sname,phi)
120         integer,intent(in)::u
121         character(*),intent(in)::sname
122         real(wp),dimension(:,:,:),intent(in)::phi
123         integer::k1,k2,k3
124
125         write(u,"(A,A,A)") "SCALARS␣",sname,"␣double"
126         write(u,"(A)") "LOOKUP_TABLE␣default"
127         do k3=lbound(phi,3),ubound(phi,3)
128            do k2=lbound(phi,2),ubound(phi,2)
129               do k1=lbound(phi,1),ubound(phi,1)
130                  write(u,"(ES28.15E4)") phi(k1,k2,k3)
131               end do
132            end do
133         end do
134      end subroutine vtk_write_scalar_3d
135   end module vtk_io_mod
```

window.f95

```
module window_mod
   use system_mod
   implicit none

contains

   function window_none(i) result(o)
      real(wp),dimension(:,:),intent(in)::i
      real(wp),dimension(size(i,1),size(i,2))::o

      o = i
   end function window_none

   function window_gaussian(i) result(o)
      real(wp),dimension(:,:),intent(in)::i
      real(wp),dimension(size(i,1),size(i,2))::o

      integer::r,c
      real(wp)::mr,mc

      mr = real(size(i,1),wp)/2.0_wp
      mc = real(size(i,2),wp)/2.0_wp
      forall(r=1:size(i,1),c=1:size(i,2))
         o(r,c) = i(r,c)*exp(-4.0_wp*((r-mr)**2/mr**2+(c-mc)**2/mc**2))
      end forall
   end function window_gaussian

   function window_quadratic(i) result(o)
      real(wp),dimension(:,:),intent(in)::i
      real(wp),dimension(size(i,1),size(i,2))::o

      integer::r,c
      real(wp)::mr,mc
      real(wp)::x,y

      mr = size(i,1)/2
      mc = size(i,2)/2

      do r=1,size(i,1)
         do c=1,size(i,2)
            x = (real(r,wp)-real(size(i,1),wp)/2.0_wp)/real(size(i,1),wp)
            y = (real(c,wp)-real(size(i,2),wp)/2.0_wp)/real(size(i,2),wp)
            o(r,c) = 1.0_wp/((16.0_wp*x**4+1.0_wp)*(4.0_wp*x**2+1.0_wp)
               *(16.0_wp*y**4+1.0_wp)*(4.0_wp*y**2+1.0_wp))
         end do
      end do
   end function window_quadratic

end module window_mod
```

References

[1]Bolinder, J., "On the accuracy of a digital particle image velocimetry system," Tech. rep., LUND INSTITUTE OF TECHNOLOGY, 1999. A, D, B

[2]Huang, H., Dabiri, D., and Gharib, M., "On errors of digital particle image velocimetry," *Measurement Science and Technology*, Vol. 8, 1997, pp. 1427–1440. A, B

[3]Metropolis, N., "THE BEGINNING of the MONTE CARLO METHOD," *Los Alamos Science*, Vol. Specal Issue, 1987. III

[4]Coleman, H. W. and Steele, W. G., *Experimentation, Validation, and Uncertainty Analysis for Engineers*, John Wiley & Sons, Inc., 2009. A

[5]FFTW, *FFTW 3.2.2*. IV

[6]Tschumperlé, D., "Introduction to The CImg Library," Internet, Slides. IV

[7]"MPI-2: Extensions to the Message-Passing Interface," . IV