Documentation

High performance vessel analysis tools: 3D vessel reconstruction and analysis tool using whole-slide images.

Google Summer of Code 2015

Sayan Maity

Introduction

Extensive research been performed to monitor the structural changes and spatial relationship of any diseased organ using whole slide histopathology images; resulted in a framework to study 3D whole-slide microscopy image datasets and reconstruct 3D vessel structure to analyze serial liver slices. The proposed standalone framework performs all the sequence of modules of, image registration, segmentation, vessel cross-section association, interpolation, and volumetric rendering. The simulation based research for the research study been developed in Matlab®. To make this cutting-edge novel idea applicable to large dataset acquired from the patients biopsy result in daily basis, in this project under Google Summer of Code 2015, an efficient and scalable version of HPC implementation is done using C++ programming language leveraging the powerful and rich open source 3rd party libraries such as openCV, Boost, Matlab api, Armadillo, Boost, Alglib, Cbc, VTK etc.

In this documentation, we will be explaining the pipeline framework components implemented for sequential processing: Registered Images=> Segmentation=> Vessel association => 3D Rendering. The HPC version of the two-step sequential image-based registration component is already been developed before Google Summer of Code 2015. The remaining steps of the pipeline frame work been implemented listed below:

<u>Image Segmentation:</u>

Two subsections comprised of the entire segmentation section:

1) Identify vessel tube probability map & produce initial level set function.
2) Segment vessel regions with level set method for the image sequence.

<u>Two stage Vessel association:</u>

Two subsections comprised of the entire vessel association section:

1) Local Bislide between the successive frames: 1-2;3-4;5-6.
2) Global association between the Local Bislide pairs; using the pair generated in the previous step: (1-2)<=> (3-4)<=> (5-6).

<u>3D Rendering:</u>

Two subsections comprised of the entire 3D rendering section:

1) Using the associated vessel and their boundaries create the interpolated boundaries using spline interpolation technique, and save the resulted boundaries as binary series of images (*.tif) .

2) Use the interpolated boundaries (stacks of series of binary images) to build tetrahedral mesh representation for 3D visualization.

The remainder of this project documentation is organized as follows: we will describe the implementation of all the three tasks: Image Segmentation, Two stage Vessel association & 3D Rendering. For each of them the implemented function definitions will be explained in details including: function definition in C++; technical functionality it will be performing, the data-type of the inputs & outputs of each function. In addition, the third party libraries used in the implementation of a task in which context will be listed in each task. At the end, a script map (flowchart) along with the data-type in the pipeline framework exchanged & output of the sections will be portrayed. An appendix section is added with the issues faced at the time of implementation.

Image Segmentation

II.I. Function Definition

To describe the function definition I have used tree like structure using "tab" to represent the dependencies of functions in color coded.

Name of function: File name=> "segmentation.cpp"
Inputs=>Registered images ("output_image.XX.bmp")
Functionality=> Set the values for color de-convolution, stain i.e. Hematoxylin, DAB, and red_marker. *Just to note the 3 channels in a RGB image follow BGR order in OpenCV where as in Matlab the order is RGB.*
Output=> The generated initial level set function is saved in yml. Can be read to Matlab and convert to Mat file.

> **Name of function=> Ocv_ColorDeconvolution(vector< vector<double> > stains, cv::Mat img, Size s)**

Inputs=> The stain vales defined by the calling functions; The Registered image ("output_image.XX.bmp") passed by the calling function; & the size of the image

Functionality=> Deconvolves multiple stains from RGB image 'img', given stain matrix 'stains'.

Output=> intensity - an intensity image in C++ double datatype. Each channel is a stain intensity. Channels are ordered same as columns of 'stain' following BGR order.

Name of function=>vec_transpose

Input=> vector<vector<double>>

Functionality => Transpose the 2 dimensional C++ STL vector.

Output=> vector<vector<double>>

Name of function=> ocv_im2vec

Input=> A openCV matrix (cv::Mat)

Functionality => Converts a RGB color image to 3 x MN matrix (Where M,N=> Size of the RGB color image)

Output=> 3 x MN dimensional C++ STL vector: vector<vector<int>>

Name of function=> vec_deconvolution_normalize

Input=> 3 x MN dimensional C++ STL vector: vector<vector<int>>

Functionality => Normalize raw color values according to Rufriok and Johnston's color deconvolution scheme.

Output=> 3 x MN dimensional C++ STL vector: vector<vector<double>>

Name of function=> vec_deconvolution_denormalize

Input=> 3 x MN dimensional C++ STL vector: vector<vector<int>>

Functionality => De-normalize raw color values according to Rufriok and Johnston's color deconvolution scheme.

Output=> 3 x MN dimensional C++ STL vector: vector<vector<double>>

Name of function=> reshpe_back(vector< vector<double> > , Size s)

Input=> 3 x MN dimensional C++ STL vector: vector<vector<int>> & value of M &N explicitly

Functionality => Reshape back 3 x MN dimensional C++ STL vector to MXN openCV matrix (cv::Mat)

Output=> MXN openCV matrix (cv::Mat)

Name of function=> ocv_multiScaleFilter2D (cv::Mat , Size s)

Inputs=> Single channel cv::Mat; the corresponding DAB stain channel & the size of the image

Functionality=> Perform filtering operation in multiple scale & angle. Store the filtered results generated of different scales & angles.

Output=> cv::Mat object containing the filtered result @ different scales & angles.

Name of function=> ocv_Hessian2D (cv::Mat, double sigmas)

Input=> Single channel cv::Mat; the corresponding DAB stain channel; & C++ double precision value ‘sigmas’ for the kernel generation.

Functionality => Generate kernel coordinates to Build the gaussian 2nd derivatives filters.

Output=> 3-channel cv::Mat containing the filtering result in xx, xy & yy directionality.

Name of function=> ocv_Hessian2D (cv::Mat, double sigmas)

Input=> Single channel cv::Mat; the corresponding DAB stain channel; & C++ double precision value ‘sigmas’ for the kernel generation.

Functionality => Generate kernel coordinates to Build the gaussian 2nd derivatives filters.

Output=> 3-channel cv::Mat containing the filtering result in xx, xy & yy directionality.

Name of function: File name=> “LSbatch.cpp”

Inputs=>Registered images (“output_image.XX.bmp”); the produced initial level set & color convoluted intensity images after filtering

Functionality=> Update the input level set function in iteration & Segment vessel regions with level set method for the image sequence.

Output=> The segmented vessel contour saved in yml. Can be read to Matlab and convert to Mat file.

Name of function=> matread(const char *file, std::vector<double>)

Inputs=>name of the matlab *.mat file variable to read & the vector to save it’s content

Functionality=> Read a matlab *.mat file in C++ & store the data in double precision in a C++ STL vector.

Output=> C++ STL vector.

Name of function=> updatef(cv::Mat u,cv::Mat smoothImg,cv::Mat Pp,cv::Mat g, cv::Mat K,double epsilon, double sigma)

Inputs=>Multiple inputs set by the user,

Functionality=> Update f1, f2 (need to update level set function)

Output=> std::vector<cv::Mat > ; Containing cv::Mat object inside C++ STL vector.

Name of function=> Heaviside(cv::Mat u, double epsilon)

Inputs=>cv::Mat object the & the constant value.

Functionality=> Compute heaviside function.

Output=> cv::Mat

Name of function=> NeumannBoundCond(cv::Mat u)

Inputs=> cv::Mat object

Functionality=> Make a function satisfy Neumann boundary condition

Output=> cv::Mat object

Name of function=> div_norm(cv::Mat u)

Inputs=> cv::Mat object

Functionality=> compute curvature for u with central difference scheme.

Output=> cv::Mat object

Name of function=> Dirac(cv::Mat u, double epsilon)

Inputs=> cv::Mat object

Functionality=> Compute Dirac delta (The delta function is a generalized function that can be defined as the limit of a class of delta sequences. The delta function is sometimes called "Dirac's delta function" or the "impulse symbol")

Output=> cv::Mat object.

Name of function=> distReg_p2(cv::Mat u)

Inputs=> cv::Mat object

Functionality=> First compute first order derivative of the double-well potential & then compute the distance regularization term with the double-well potential.

Output=> Index of the associated vessel object.

II.II. External Libraries Used

Lirbaries Used	Functions served
Matlab api ("mat.h" header file)	To read vessel segmentation information from *.mat files.
OpenCV	To preprocess the segmentation binary images: image dilation; find close contours in the preprocessed binary images, generating kernel for image filtering, image filtering operation.
Boost	For accessing constant values such as pi, epsilon (compiler dependent)

Two stage Vessel association

III.I. Function Definition

Name of function: File name=> "vessel_assosiation.cpp"

Inputs=>Registered images ("output_image.XX.tif"); & the segmented vessel contour from the Segmentation operation.

Functionality=> Generate bi_slide vessel componets by performing a two level vessel objects association computation between the successive slides.

Output=> The segmented vessel contour saved in yml. Can be read to Matlab and convert to Mat file.

> **Name of function=> read_and_dialate(const char *imagename, const char *seg_file1,cv::Mat)**
>
> Inputs=>Name of the image to read, name of the matlab *.mat file variable to read & the cv::Mat to save it's content
>
> Functionality=> Read a matlab *.mat file in C++ & store the data in double precision in a C++ STL vector and convert to cv::Mat object and apply image dialation.
>
> Output=> cv::Mat object.
>
> > **Name of function=> matread(const char *file, std::vector<double>)**
> >
> > Inputs=>name of the matlab *.mat file variable to read & the vector to save it's content
> >
> > Functionality=> Read a matlab *.mat file in C++ & store the data in double precision in a C++ STL vector.
> >
> > Output=> C++ STL vector.
>
> **Name of function=> get_boundary_centroid(cv::Mat dialated)**
>
> Inputs=> The cv:: Mat component generated after reading & performing dilation operation.
>
> Functionality=> Compute the boundary and centroid of the vessel object detected for the given image.
>
> Output=> vector<vector<Point>> blobs_sorted (the vessel objects sorted by area), vector<double> bolb_area_sorted (area),vector<Point> bolb_center1_sorted (center), vector<Vec4i> hierarchy
>
> **Name of function=> get_FSDs(vector<double> X,vector<double> Y)**
>
> Inputs=> X & Y coordinates of the boundary of the vessel objects in C++ STL vector.
>
> Functionality=> Calculate Fourier shape descriptors for given boundary.
>
> Output=> vector<double> Fsd (Generated Fourier feature descriptor in a C++ STL vector)

Name of function=> calculate_likelihood(int i,vector<vector <double>> Fsd_slide_1,vector<Point> bolbcenter_1, vector <double> bolbarea_1,vector<vector <double>> Fsd_slide_2,vector<Point> bolbcenter_2, vector <double> bolbarea_2,int problem_id,vector<vector<Point>> blobs_2,vector<double> pre_vec_j,vector<vector<double>> vec3,)

Inputs=>Vessel objects Fourier shape descriptors; area, center.

Functionality=> Calculate the likelihood for each possible association with one-to-one, one-to-two and one-to-none and none-to-one cases.

Output=> vector<double> norm_p_likelihood, vector<vector<int>> C_coeff (Generated Fourier feature descriptor in a C++ STL vector)

> **Name of function=> get_likelihood_value(vector<double> feature1,Point centroid1,double area1,vector<double> feature2,Point centroid2,double area2,vector<double> cur_vec,vector<double> pre_vec_j,int type, int problem_id)**
>
> Inputs=> Vessel objects Fourier shape descriptors; area, center & corresponding problem ID of one-to-one, one-to-two and one-to-none and none-to-one cases.
>
> Functionality=> To compute the value of likelihood by taking into account apperance similarity, spatial distance and trajectory smoothness (for MAP)..
>
> Output=> double “like_values” (Likelihood values in C++ double precision).

Name of function=> get_matching_result(vector<double> vsol, vector<vector<int>> C_coeff, vector<vector <double>> Fsd_slide_1, vector<vector <double>> Fsd_slide_2, vector<Point> bolbcenter_1, vector<Point> bolbcenter_2)

Inputs=> Vessel objects Fourier shape descriptors; area, center & the integer programming optimized matched result.

Functionality=> To get the ids of matched objects in adjacent frame pairs.

Output=> vector<int> parent, vector<vector<int>> t_id (Index of the matched vessel object in a C++ STL container)

III.II. External Libraries Used

Lirbaries Used	Functions served
Matlab api ("mat.h" header file)	To read vessel segmentation information from *.mat files.
OpenCV	To preprocess the segmentation binary images: image dilation; find close contours in the preprocessed binary images etc.
Armadillo	To incorporate numerical functions such as interpolation to find equally spaced points on each closed contour.
Boost	For polygon operation such as union & intersection two compute one-to-two vessel object association.
Cbc	For mixed binary integer programming

3D Rendering

IV.I. Function Definition

Name of function: File name=> "vessel_assosiation.cpp"

Inputs=> After generating bi_slide vessel componets by performing a two level vessel objects association computation between the successive slides the associated id's of the objects been used.

Functionality=> Using the associated vessel and their boundaries create the interpolated boundaries using spline interpolation technique, and save the resulted boundaries as binary series of images (*.tif).

Output=> Series of *.tif 2d binary images.

Name of function: File name=> "vtk_ImageRendering.cpp"

Inputs=> Series of *.tif 2d binary images

Functionality=> Use the interpolated boundaries (stacks of series of binary images) to build tetrahedral mesh representation for 3D visualization.

Output=> User interaction window tetrahedral mesh representation for 3D visualization.

IV.II. External Libraries Used

Lirbaries Used	Functions served
Alglib	Spline interpolation.
VTK	3D visualization of the vessel structure.

Implementation Flowchart

Pairwise Image Registration

- Already implemented in C++: Used as input for the Segmentation section.

<u>Visualization(Output of the section)</u>

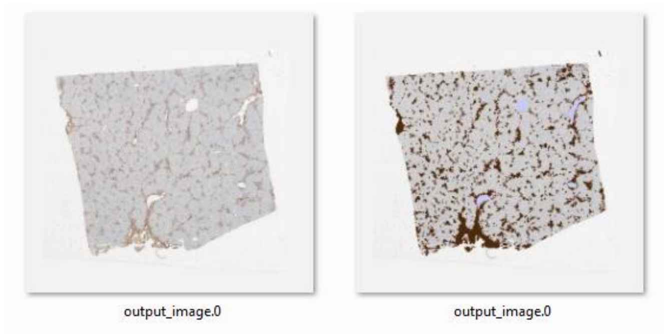
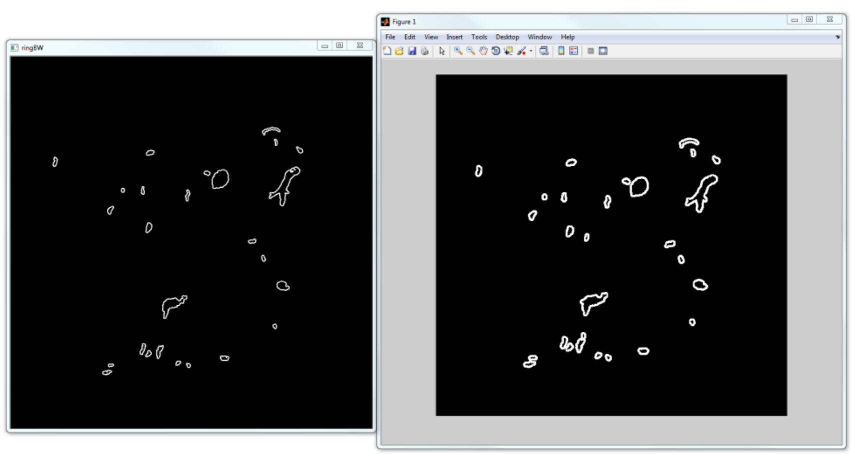


Image Segmentation

- Identify vessel tube probability map & produce initial level set function.
- Segment vessel regions with level set method for the image sequence.

<u>Visualization(Output of the section)</u>



Left Output of C++ compared with Matlab right

Vessel object association

- Local Bislide between the successive frames, for example: 1-2;3-4;5-6.
- Global association between the Local Bislide pairs; using the pair generated in the previous step: (1-2)<=> (3-4)<=> (5-6).

<u>Visualization(Output of the section)</u>

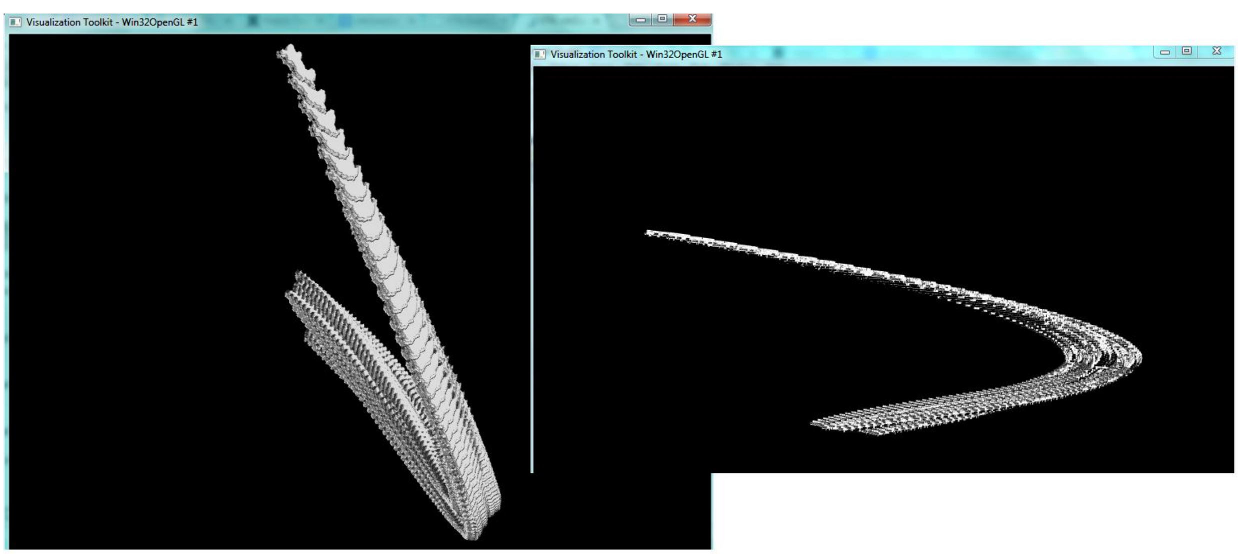
Parent Vessel Object ID	Associated Vessel Object ID in Successive frame
…	…
…	…

Abstract (toy) table structure to show the output of this section

3D Volume rendering

- Using the associated vessel and their boundaries create the interpolated boundaries using spline interpolation technique, and save the resulted boundaries as binary series of images (*.tif) .
- Use the interpolated boundaries (stacks of series of binary images) to build tetrahedral mesh representation for 3D visualization.

<u>Visualization(Output of the section)</u>



User Interactive 3D vessel structure (3D window)

Appendix

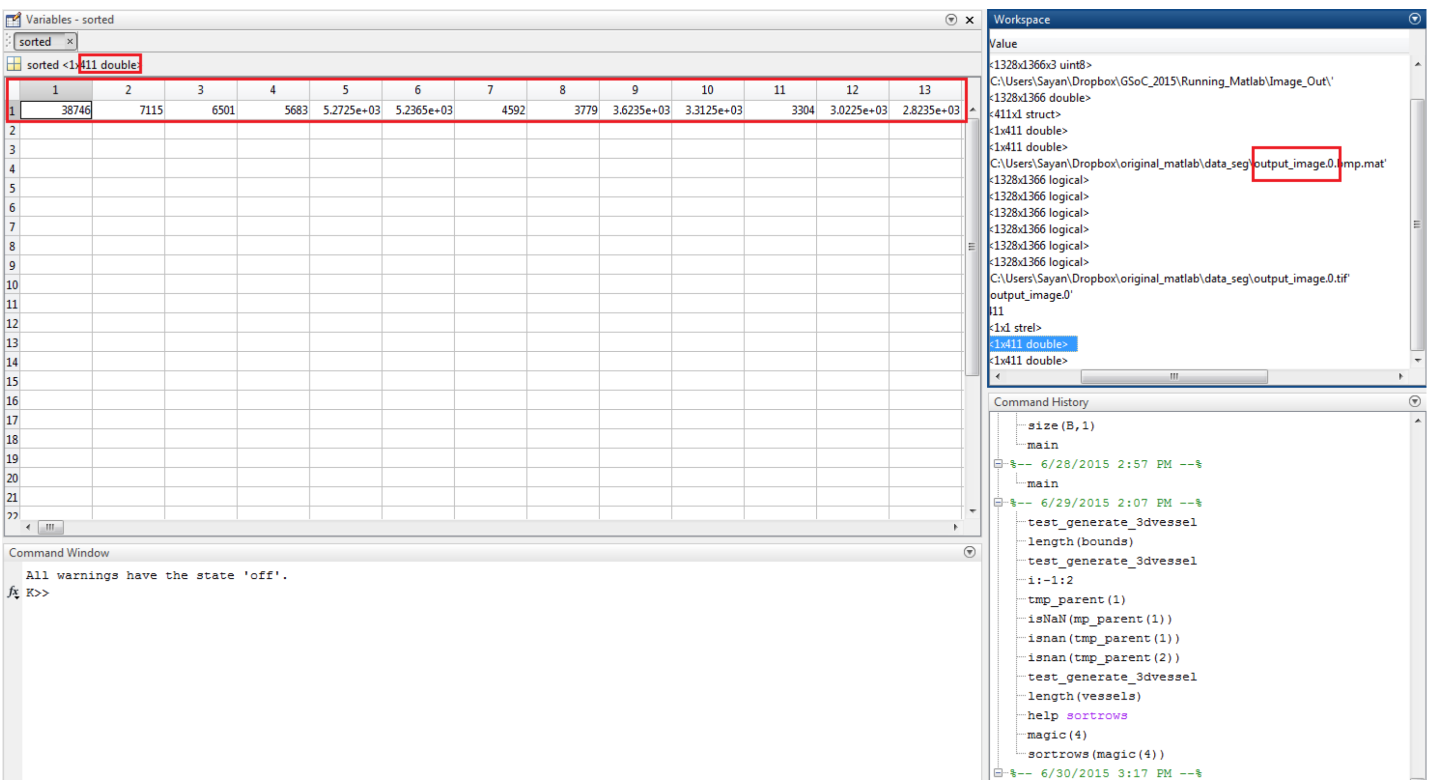
While Implementing the C++ framework there were few discrepancies faced those will be listed below.

Vessel object detected from binary-segmented image in C++ & Matlab comparison with screenshots

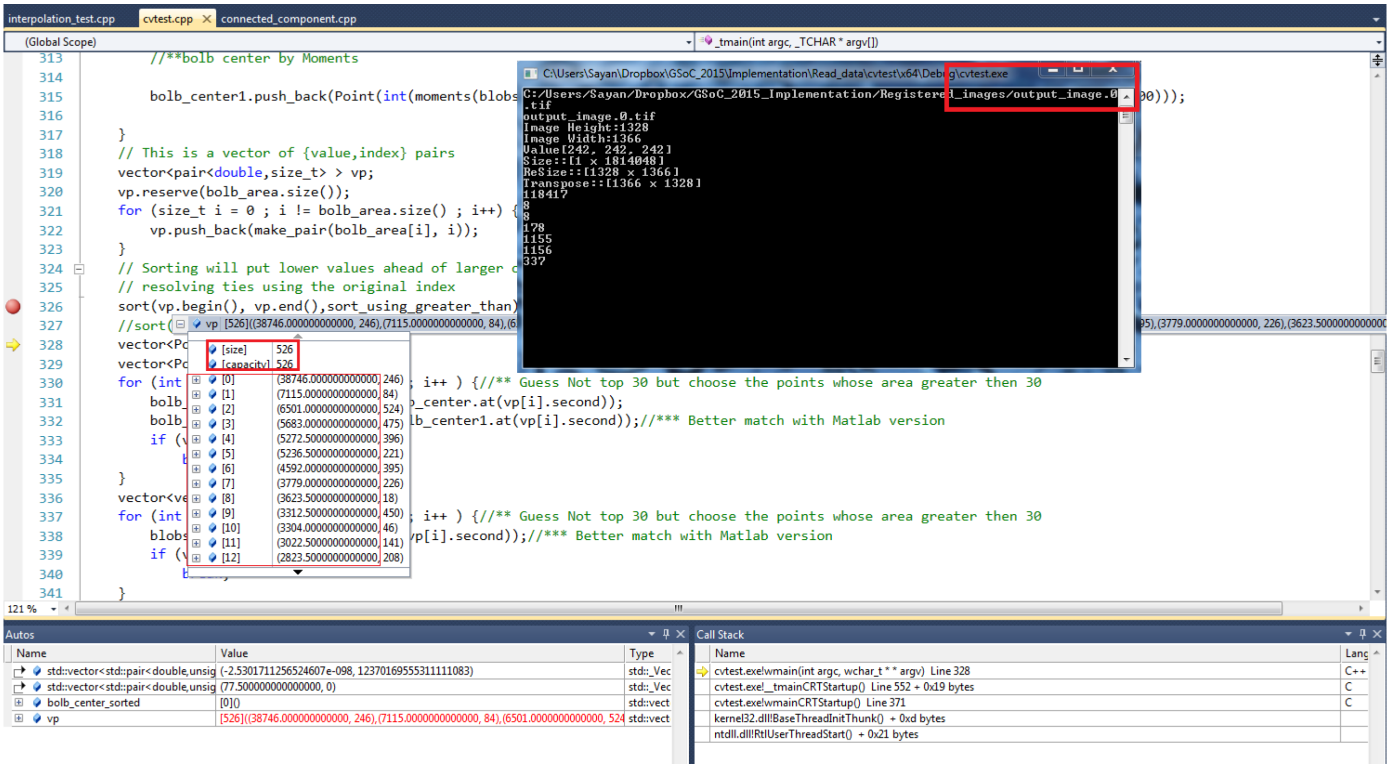
The result screen shot of Variable explorer in Matlab and STL container in C++ (Visual Studio) to show the behavior of "bwboundaries()" in Matlab & "findContours()" function in OpenCV both used to detect closed contours in binary images.

In the "original_matlab\data_seg" directory for the segmented file "output_image.0.bmp" I am attaching the marked screenshot below:

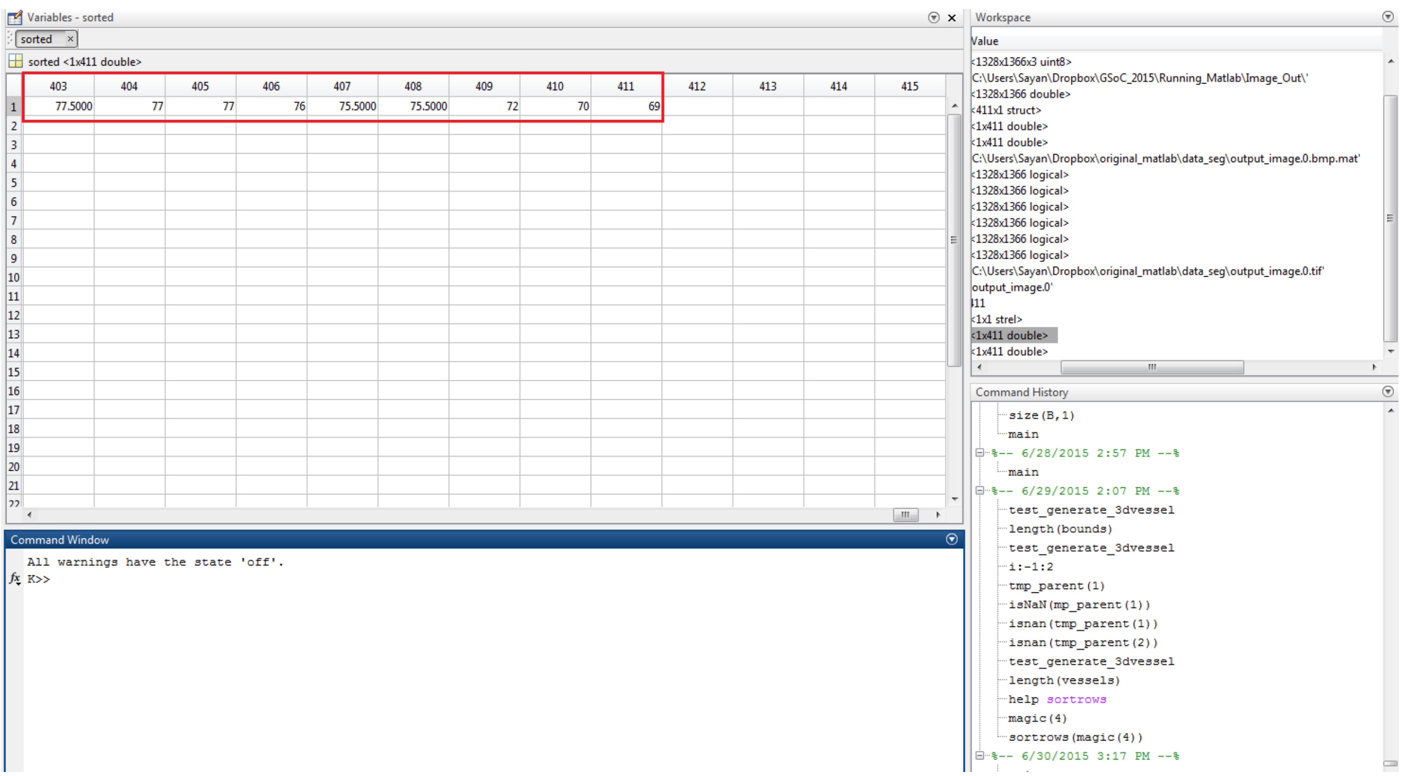
Matlab "bwboundaries()" total 411 closed contours vessel object and 13 largest vessel objects (in areas):



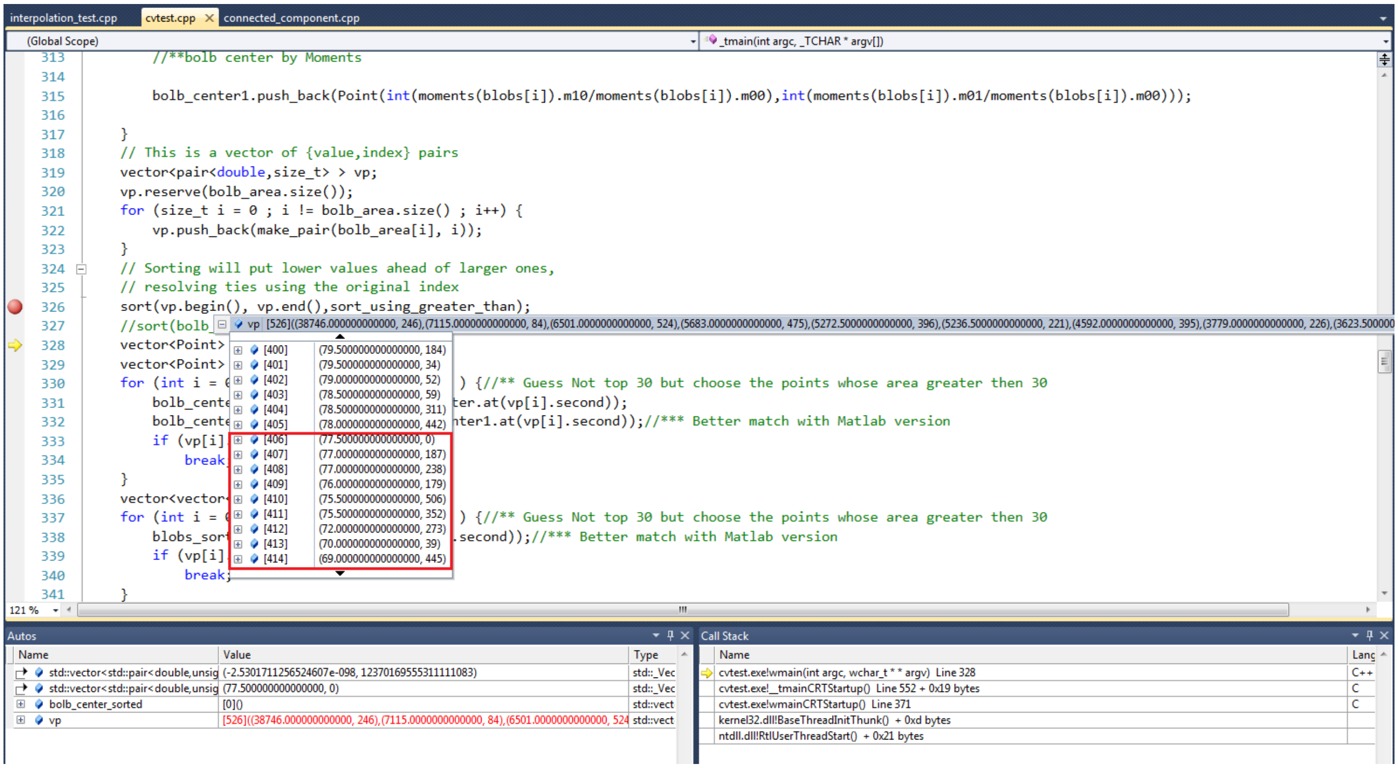
OpenCV "findContours()" total 526 closed contours vessel object and 13 largest vessel objects (in areas): [Compare the above and below screenshot You can see the areas are matched & did one-one plot also to verify in details]



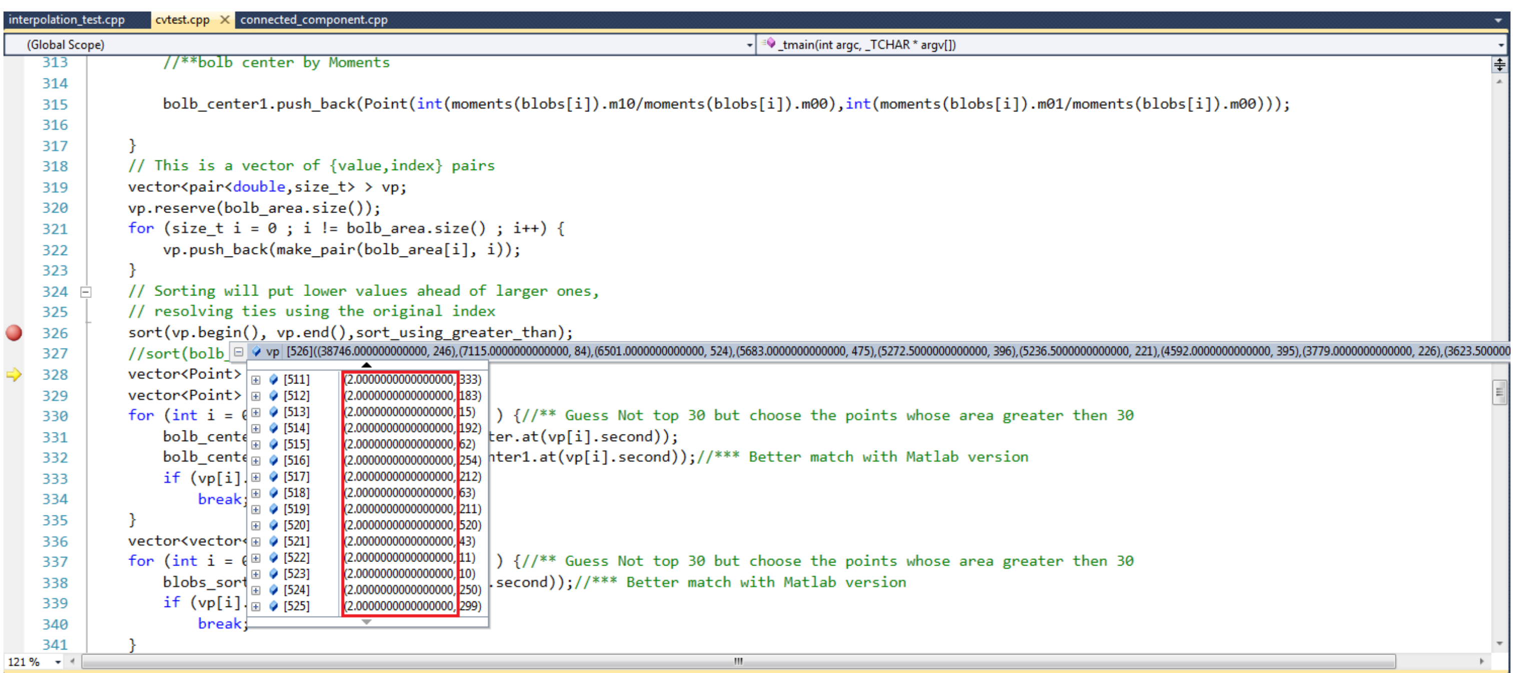
Matlab "bwboundaries()" 9 smallest vessel objects (in areas):



OpenCV "findContours()" 9 smallest vessel objects compared with Matlab "bwboundaries()" (in areas): [Compare the above and below screenshot You can see the areas are matched and the number of vessel objects are closely matched if we put threshold of area '70']



OpenCV "findContours()" detecting vessel object upto area "2.0" where we see before Matlab "bwboundaries()" detecting vessel object upto area "69.0"



Data type Precision Mismatch in C++ & Matlab comparison with screenshots

While working on the Segmentation section implementation in C++ I am mostly using Open CV library. I faced some issues regarding recreating the same result in C++ compare to Matlab due to precision mismatch of double values in two different compiler.

I did a through research and find out there are some existing issue that haven't been solved regarding Matlab & C++(OpenCV) Precision mismatch.

http://stackoverflow.com/questions/11151609/precision-differences-in-matlab-and-c

http://stackoverflow.com/questions/7622012/matlab-and-c-differ-with-cos-function

I found out:

In **Matlab** 'double' datatype set the CPU floating point precision to 80 bits.

Where-as in **C++**:

1) Type float, 32 bits long, has a precision of 7 digits.

2) Type double, 64 bits long, has a 15 digits precision.

3) Type long double is nominally 80 bits, though a given compiler/OS pairing may store it as 12-16 bytes for alignment purposes.

So to compare result in C++ with Matlab we are supposed to use Type long double in C++.

Here is the **bottle-neck**; **OpenCV** library has its own datatype and it can only handle up to precision of 64 bit :

1) CV_32F: "float"=> 32 bits long

2) CV_64F: "double"=> 64 bits long

3) OpenCV didn't support long double.

Based on the facts above when I'm trying to perform filtering operation (In 'multiScaleFilter2D.m' Matlab function) when ever the pixel values are hitting very small exponential (Please See attached Figure 1&2: **OpenCV can only go up-to e-16 compare to Matlab e-31**) any operation (such as squaring, square root & exponential) on those resulting in a huge variation in the result in C++ & OpenCV.

Thus I tried to match the resulted "Ifiltered" & "angles" variables in the 'multiScaleFilter2D.m' Matlab function in the corresponding C++ version:

Outcome 1) "angles" in both Matlab function in the corresponding C++ version is similar as we are only performing inverse tangent on them (Please See attached Figure 3)

Outcome 2) "Ifiltered" been going through squaring & exponential operation thus there is some substantial variation between the Matlab function & the corresponding C++ version :

Test-1: Number of '0' value pixels in image "Ifiltered":

Matlab: 1451231

OpenCV: 1673210 (Not too different)

Thus "imshow" command on "Ifiltered" variable in C++ version is just a black image (as pixels values are very low) . Just for debugging purpose, I multiply the "Ifiltered" variable in C++ with 30 to see how it looks compare with the Matlab version (Please See attached Figure 4) .

Figure 1:

Matlab (double) => 64 bit to 80 bit

Dxy <1328x1366 double>

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	30.7442	10.2645	1.1253	4.3555e-15	4.3555e-15	4.3555e-15	4.3555e-15	4.3555e-15	4.3555e-15	4.3555e-15	4.3555e-15	4.3555e-15	4.3555e-15	4.3555e-1
2	10.2645	3.4270	0.3757	1.4542e-15	1.4542e-15	1.4542e-15	1.4542e-15	1.4542e-15	1.4542e-15	1.4542e-15	1.4542e-15	1.4542e-15	1.4542e-15	1.4542e-1
3	1.1253	0.3757	0.0412	1.5942e-16	1.5942e-16	1.5942e-16	1.5942e-16	1.5942e-16	1.5942e-16	1.5942e-16	1.5942e-16	1.5942e-16	1.5942e-16	1.5942e-1
4	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
5	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
6	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
7	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
8	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
9	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
10	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
11	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
12	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
13	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
14	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
15	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
16	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
17	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
18	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
19	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
20	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
21	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
22	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3
23	0	-1.3323e-15	5.5511e-17	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-31	-1.9722e-3

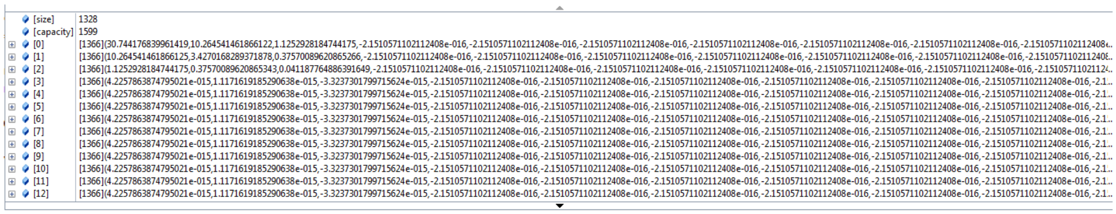
Matlab (single) => 32 bit

Dxy1 <1328x1366 single>

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	30.7442	10.2645	1.1253	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
2	10.2645	3.4270	0.3757	1.7881e-07	1.7881e-07	1.7881e-07	1.7881e-07	1.7881e-07	1.7881e-07	1.7881e-07	1.7881e-07	1.7881e-07	1.7881e-07	1.7881e-0
3	1.1253	0.3757	0.0412	-1.7881e-07	-1.7881e-07	-1.7881e-07	-1.7881e-07	-1.7881e-07	-1.7881e-07	-1.7881e-07	-1.7881e-07	-1.7881e-07	-1.7881e-07	-1.7881e-0
4	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
5	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
6	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
7	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
8	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
9	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
10	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
11	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
12	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
13	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
14	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
15	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
16	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
17	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
18	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
19	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
20	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
21	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
22	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0
23	7.7486e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-07	2.9802e-0

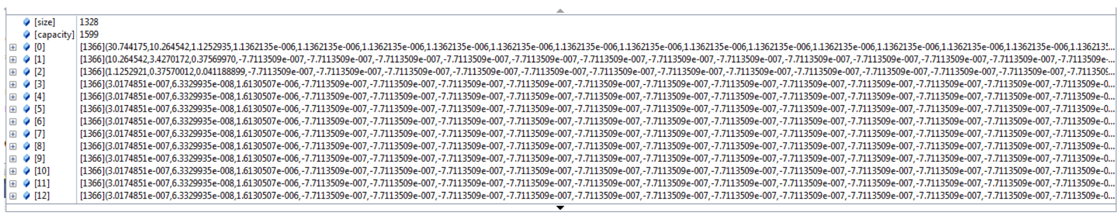
Figure 2:

OpenCV: CV_64F => double



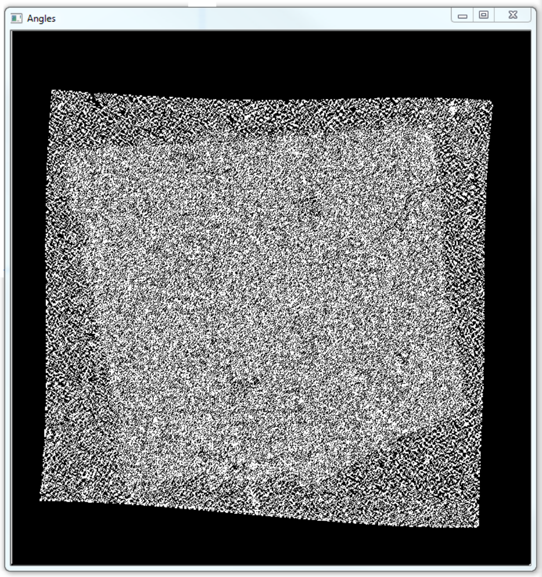
[size]	1328
[capacity]	1599
[0]	[1366](30.744176839961419,10.264541461866122,1.1252928184744175,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e..
[1]	[1366](10.264541461866125,3.4270168289371878,0.37570089620865266,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.151057110211240..
[2]	[1366](1.1252928184744175,0.37570089620865343,0.041187764886391649,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.15105711021124..
[3]	[1366](4.2257863874795021e-015,1.1171619185290638e-015,-3.3237301799715624e-015,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1...
[4]	[1366](4.2257863874795021e-015,1.1171619185290638e-015,-3.3237301799715624e-015,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1...
[5]	[1366](4.2257863874795021e-015,1.1171619185290638e-015,-3.3237301799715624e-015,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1...
[6]	[1366](4.2257863874795021e-015,1.1171619185290638e-015,-3.3237301799715624e-015,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1...
[7]	[1366](4.2257863874795021e-015,1.1171619185290638e-015,-3.3237301799715624e-015,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1...
[8]	[1366](4.2257863874795021e-015,1.1171619185290638e-015,-3.3237301799715624e-015,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1...
[9]	[1366](4.2257863874795021e-015,1.1171619185290638e-015,-3.3237301799715624e-015,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1...
[10]	[1366](4.2257863874795021e-015,1.1171619185290638e-015,-3.3237301799715624e-015,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1...
[11]	[1366](4.2257863874795021e-015,1.1171619185290638e-015,-3.3237301799715624e-015,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1...
[12]	[1366](4.2257863874795021e-015,1.1171619185290638e-015,-3.3237301799715624e-015,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1510571102112408e-016,-2.1...

OpenCV: CV_32F => float

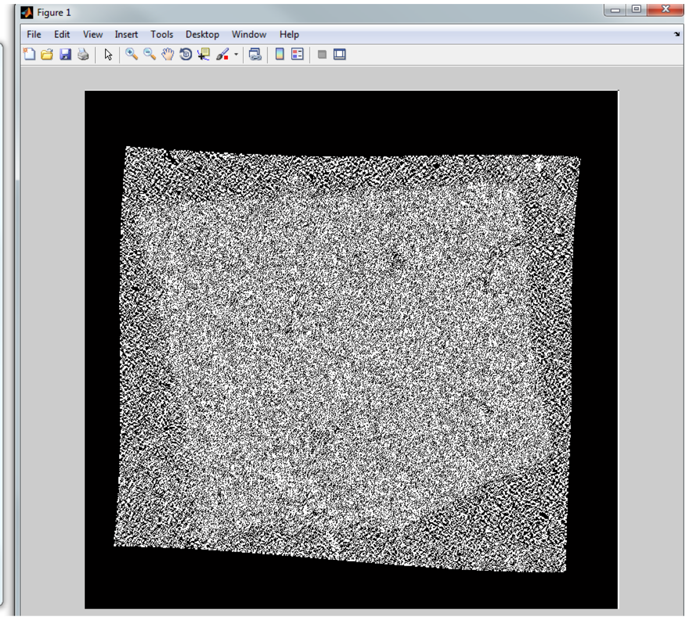


[size]	1328
[capacity]	1599
[0]	[1366](30.744175,10.264542,1.1252935,1.1362135e-006,1.1362135e-006,1.1362135e-006,1.1362135e-006,1.1362135e-006,1.1362135e-006,1.1362135e-006,1.1362135e-006,1.1362135e-006,1.1362135e-006,1.1362135e-006,1.1362135e-006,1.136213...
[1]	[1366](10.264542,3.4270172,0.37569970,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-...
[2]	[1366](1.1252921,0.37570012,0.041188899,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.711350...
[3]	[1366](3.0174851e-007,6.3329935e-008,1.6130507e-006,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-0...
[4]	[1366](3.0174851e-007,6.3329935e-008,1.6130507e-006,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-0...
[5]	[1366](3.0174851e-007,6.3329935e-008,1.6130507e-006,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-0...
[6]	[1366](3.0174851e-007,6.3329935e-008,1.6130507e-006,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-0...
[7]	[1366](3.0174851e-007,6.3329935e-008,1.6130507e-006,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-0...
[8]	[1366](3.0174851e-007,6.3329935e-008,1.6130507e-006,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-0...
[9]	[1366](3.0174851e-007,6.3329935e-008,1.6130507e-006,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-0...
[10]	[1366](3.0174851e-007,6.3329935e-008,1.6130507e-006,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-0...
[11]	[1366](3.0174851e-007,6.3329935e-008,1.6130507e-006,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-0...
[12]	[1366](3.0174851e-007,6.3329935e-008,1.6130507e-006,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-007,-7.7113509e-0...

Figure 3:

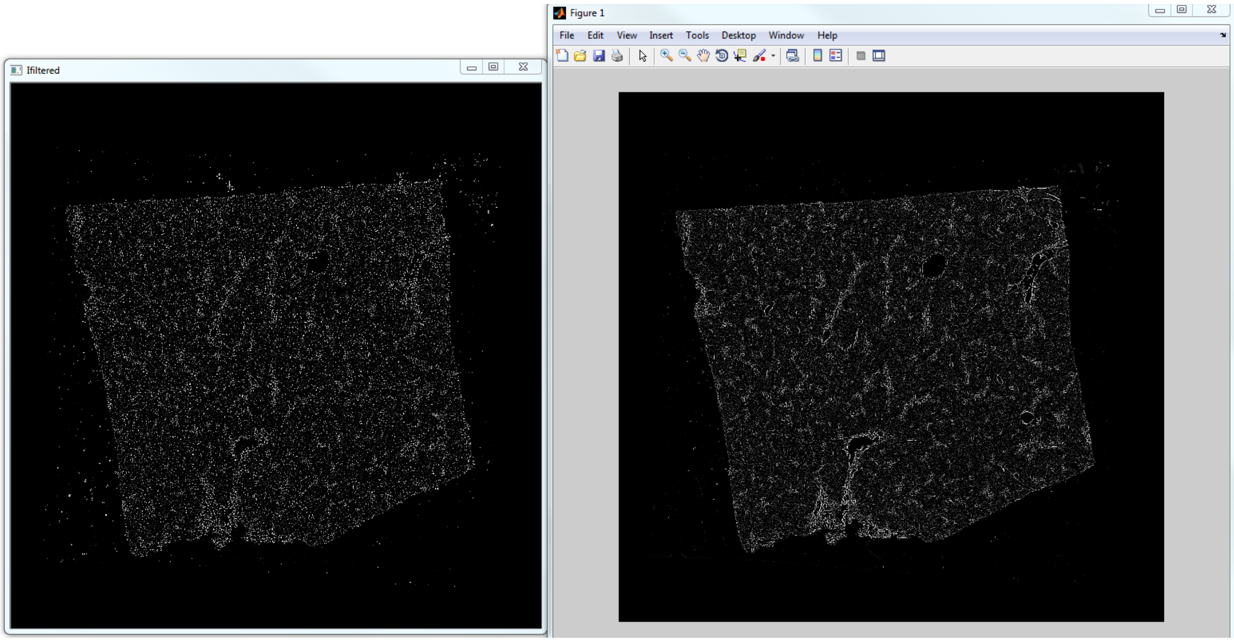


Open CV



Matlab

Figure 4:



Open CV *Matlab*