Junior-Prof. Dr. Olaf Ronneberger,
Image Analysis Lab, Institute of Computer Science, Albert-Ludwigs-University Freiburg

Exercises for 3D Image Analysis

Summer term 2015

Exercise 4 (Issue Date: 09.06.2015, Due Date: 23.06.2015, 1pm)

Landmark based Registration with the Umeyama Method

The goal of this exercise is to implement the Umeyama Method [1]. It will be used to estimate the rigid transformation matrix for two set of landmarks with known correspondences. In this exercise you will estimate the rigid transformation of zebrafish embryos, stored in hdf5 file format. To display the data and the results, we use Fiji or the viewer "vivi". In the first part of this exercise you set up the required software and the second part focuses on implementing the methods.

1) Preliminaries – Set up libraries

1) The central part in the Umeyama method is a singular value decomposition (SVD). We will use the gsl_linalg_SV_decomp() algorithm from the **Gnu Scientific Library (GSL)**. This library is available as package for all major linux distributions, e.g. on Ubuntu use

```
sudo aptitude install libgsl0-dev
```

to install it. Like all system libraries, the headers will be installed in the standard paths /usr/include and the library in /usr/lib -- the compiler automatically searches in these paths, so you don't need to provide them in the compiler call. A minimal program using the gsl is

```
#include <iostream>
#include <gsl/gsl_matrix.h>

int main( int argc, char** argv)
{
  // allocate a gsl matrix and fill it with zeros
  gsl_matrix* A = gsl_matrix_calloc( 3, 3);


  // set the values in the matrix, e.g.  with gsl_matrix_set (A, i, j, 1);
  // do something...

  // free the memory of the matrix
  gsl_matrix_free( A);
  return 0;
}
```

You can compile this program using

```
g++ -Wall -O2 -g test_gsl.cc  -lgsl -lgslcblas -lm -o test_gsl
```

Go to the gsl homepage http://www.gnu.org/software/gsl/ download the manual and read at least chapter "Vectors and Matrices" and the section "Singular Value Decomposition" in chapter "Linear Algebra". For further reading a detailed description of Singular Value Decomposition is given in [2].
2) We will use the **HDF5 format (hierarchical data format)**, see http://www.hdfgroup.org/HDF5/,

which allows comfortable handling of scientific multidimensional data. At first, the HDF5 library must be installed. For reading and writing blitz++-arrays we use a simple interface library that is provided in the additional material on the course webpage (BlitzHDF5Helper.hh). **The appendix gives further information about the installation of the HDF5 library and the usage of the interface library.** In each HDF5 file you can store any number of arrays and access them by a hierarchical name like "/path/to/resource". To show the contents of a HDF5 file you can use the command line tools **h5dump** and **h5ls**, e.g.

```
h5ls -r zebrafish0.h5

/                        Group
/channel0                Dataset {81/Inf, 207/Inf, 387/Inf}
/landmarks_um             Dataset {11/Inf, 3/Inf}
```

3) To view the contents of HDF-datasets you can use **Fiji**. You need the HDF5 plugin from our homepage: http://lmb.informatik.uni-freiburg.de/resources/opensource/imagej_plugins/hdf5.html. After installing the plugin you can import HDF datasets using “File/Import/HDF5”. You can use the standard slice view of Fiji to display 3d stacks, but you can also try the orthogonal view “Image/Stacks/Orhotgonal Views” (see Fig.1 right) and the nice 3d viewers in “Plugins/3D Viewer and Volume Viewer”. Alternatively, you can use the **“vivi”** viewer developed in our group to display HDF5 datasets. vivi displays many HDF5 files (one per line, see Fig. 1 left). To display data from multiple files add them by clicking on the “Add” button below the list of files. To create a column showing the central slice of the channel0 data set of all files click on the “Assistant” button below the list of pipelines. In the “Pipeline Assistant” dialog select “Level central slice” as preset and check the channel0 data set. Click on “Create column” and close the dialog.

You can download **Fiji** from http://fiji.sc . **vivi** is available on the course webpage.

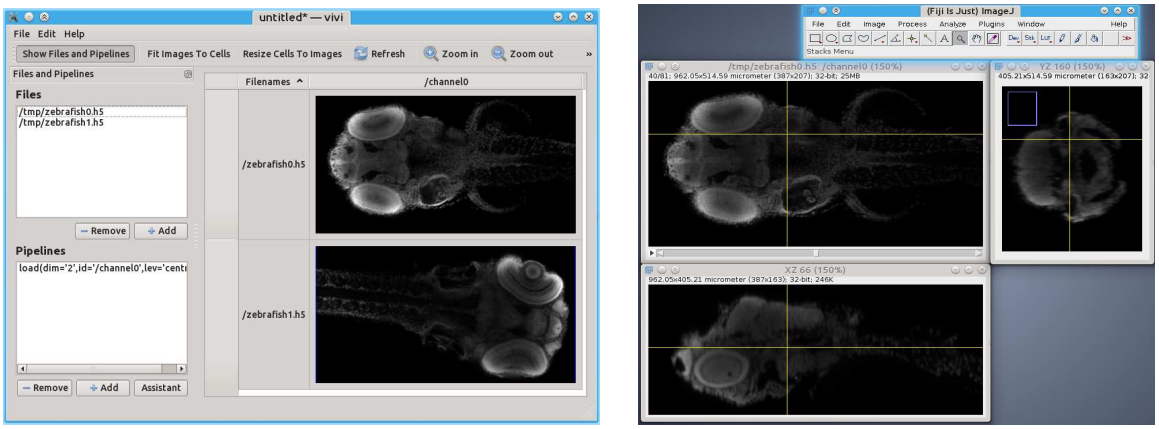

Fig. 1: left: vivi showing channel0 of the two zebrafishes, right: Ortho view of channel0 of the first zebrafish in Fiji

4) Try if the libraries work correctly with the following small demo program:

```
//
// Test program for blitzHDF5 interface library. compile with
//
// g++ -Wall -O2 -g demo.cc -lblitz -o demo -lhdf5
//

#include "BlitzHDF5Helper.hh"

int main( int argc, char** argv)
{
   std::string inFileName="zebrafish0.h5";
   std::string outFileName="demo.h5";

  // read HDF5 file into blitz++-array
  blitz::Array<float,3> raw;
  readHDF5toBlitz(inFileName, "channel0", raw);
  blitz::TinyVector<float,3> element_size_um;
  readBlitzTinyVectorFromHDF5Attribute(element_size_um, "element_size_um", "/channel0",
       inFileName);

  // perform computations
  raw = blitz::max(raw) - raw;

  // write blitz++-array to HDF5 file
  writeBlitzToHDF5(raw, "channel0_inverted", outFileName);
  writeBlitzTinyVectorToHDF5Attribute(element_size_um, "element_size_um", "/channel0_inverted",
       outFileName);

  return 0;
}
```

Look at the result with Fiji or vivi.

2) Programming

1) The program should be named "umeyama.cc"

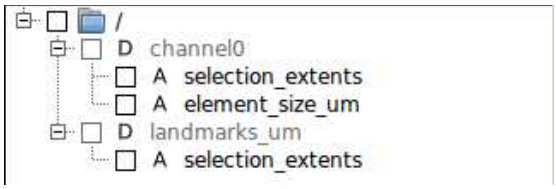

Fig 2.: Structure of hdf5 files zebrafish0.h5 and zebrafish1.h5

2) First you load the zebrafish data. They are stored as 3d float arrays in “/channel0”. As in the previous exercises, the voxels are not cubic, so read the element sizes from the hdf5 files with:

```
blitz::TinyVector<float,3> element_size_um;
readBlitzTinyVectorFromHDF5Attribute(element_size_um, "element_size_um", "/channel0",
       inFileName);
```

In the end you will write your resulting image(s) to a hdf5 file, so do not forget to write the element sizes as an attribute to the data. Important: Keep the name “element_size_um” for the attribute, other

programs rely on it.

```
writeBlitzTinyVectorToHDF5Attribute(element_size_um, "element_size_um", "/channel0",
       outFileName);
```

3) Load the landmarks from the hdf5 files

```
blitz::Array<blitz::TinyVector<double,3>,1> Landmarks_um;
readHDF5toBlitz(inFileName,"landmarks_um", Landmarks_um);
```

More documentation is found in the Header-File **BlitzHDF5Helper.hh**

4) Your images consist of float values, so overload your existing interpolNN and transformArray function (or take it from the sample solution), so that they work with float values.

```
void transformArray( const blitz::Array<float, 3>& srcArr,
                     const blitz::TinyMatrix<float,4,4>& invMat,
                     blitz::Array<float, 3>& trgArr)

float interpolNN( const blitz::Array<float, 3>& arr,
                  blitz::TinyVector<float,4> pos)
```

5) Write a Function

```
void markLandmark(    blitz::Array<float,3>& Image,
                      const blitz::Array<blitz::TinyVector<double,3>,1>& landmarks,
                      blitz::TinyVector<float,3> element_size_um)
```

that highlights the landmarks with a cross, a cube or a ball in a given image.
Hint: Use the blitz::Range operator to do this efficiently.

6) Write a Function

```
blitz::Array<blitz::TinyVector<float,3>,3 > overlay(const blitz::Array<float,3>& fixedImage,
                                                    const blitz::Array<float,3>& movingImage)
```

that takes two images as input and returns an rgb image, where fixedImage is in one channel and movingImage is in another channel, see Fig. 3. Note, that the two input images do not have the same size! **Bonus point:** Avoid using loops, use the blitz::Range operator.

7) Write a wrapper for the gsl_linalg_SV_decomp() function that takes blitz::TinyMatrix as arguments.

```
void mySVD( blitz::TinyMatrix<float,3,3>& A,
       blitz::TinyVector<float,3>& S,
       blitz::TinyMatrix<float,3,3>& V)
```

use the gsl_matrix_get/set functions to access the values within the gsl_matrix.

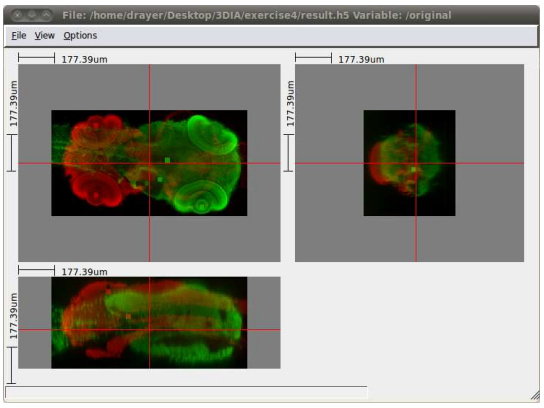

Fig 3.: Visualization of the overlay of the two zebrafishes before transforming with umeyama. Landmarks are highlighted as cubes

8) Start writing the main() function. Do some simple tests to check, whether the SVD works as expected. You may want to use the general versions (for arbitrary sized matrices) of myproduct() provided in "my_blitz.hh". Read and understand "my_blitz.hh" -- there are some other functions, that you might need for this exercise.

9) Implement the Umeyama method:

```
blitz::TinyMatrix<float,4,4> umeyama(
              const blitz::Array<blitz::TinyVector<float,4>,1>& points,
              const blitz::Array<blitz::TinyVector<float,4>,1>& transformedPoints)
```

The corresponding point sets have to be provided in homogeneous coordinates. The returned matrix should fulfill transformedPoints(i) = H * points(i) for all i. Write some tests that transform a set of random points (use, e.g. rand()%100 to get an integer between 0 and 99) with a given rigid transformation Matrix and check, whether your umeyama method can reconstruct the correct matrix.

Bonus point: If your rotation matrix has det=-1, it does not only rotate but it mirrors. Check whether your matrix is a rotation matrix and if not modify A or V so that it becomes one.

10) Now, you load the 3d zebrafish images, the corresponding landmarks and element sizes. Mark the landmarks in the images (landmarks are given in um), create an overlay as in Figure 2 and write it to an output hdf5 file. Compute the transformation of the landmarks with the umeyama method. Measure the absolute distance of the landmarks before and after transformation (before 4477.44, after 246.621). The last step is to transform the movingImage. Since the images are in the voxel domain, you have to slightly modify your transformation matrix , which is in um. After the transformation, mark the landmarks again, create an overlay and store it in hdf5 file (do not forget to store the element size).
Hint: (um_to_voxel*Umeyama*voxel_to_um)*vector

Hints:

- Comparing float values for equality

  Due to rounding errors float values are usually not exactly equal although they should be theoretically. That's why this expression does not work as expected: `(a == b)`
  In this case better use an expression like the following to correctly check for equality:
  `abs(a - b) < epsilon`, where epsilon is chosen according to the expected accuracy, e.g. `1e-10`

- Component-wise comparison for equality of blitz arrays of type float

  Using `blitz::all( a == b)` the comparison is performed component-wise, or for comparing float values use `blitz::all( abs(a – b) < epsilon)` respectively.

References:

[1] S. Umeyama - Least-Squares Estimation of Transformation Parameters Between Two Point Patterns, Pattern Analysis and Machine Intelligence, 1991

[2] Singular Value Decomposition, Numerical Recipes in C (section 2.6 sample pages)

The references can be downloaded from the course webpage, see “Additional Material”.

Appendix 1: Installation of HDF5 Library

The HDF5 library is available as package for linux distributions, e.g. on Ubuntu use

```
sudo apt-get install libhdf5-dev
```

to install it. Like all system libraries, the headers will be installed in the standard paths /usr/include and the library in /usr/lib -- the compiler automatically searches in these paths, so you don't need to provide them in the compiler call.

Alternatively, you can download the latest version from the link below, and install it according to the documentation:

http://www.hdfgroup.org/HDF5/

Note: It should be already installed on the computers in the pool.

Appendix 2: Usage of BlitzHDF5 Interface Library

Copy the header file **`BlitzHDF5Helper.hh`** into your source code directory and include it in your program code:

```
#include "BlitzHDF5Helper.hh"
```

The header file provides a minimalistic interface library for reading and writing HDF5 from and to Blitz. The following functions are provided:

```
readHDF5toBlitz
writeBlitzToHDF5
readBlitzTinyVectorFromHDF5Attribute
writeBlitzTinyVectorToHDF5Attribute
```

Note: The provided header file and the contained functions are in an **experimental state**.

Section 1.4 provides a small demo program for using the blitzHDF5 interface library. For testing the interface library and further hints on the usage of the provided functions, you can have a look at the file **`BlitzHDF5Helper_test.cc`**, which is available in the additional material on the course webpage.