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 3 (Issue Date: 22.05.2015, Due Date: 09.06.2015)

Rigid Registration

The goal of this exercise is to implement a rigid registration algorithm for two volumetric data sets. We will use the Euclidean Transformation (6 degrees of freedom), a nearest neighbor interpolation, the sum of squared differences (SSD) as similarity measure, and the Best Neighbor Optimizer. The individual steps during optimization will be saved as orthogonal maximum intensity projections (red: fixed image, green: moving image). The data set consists of two confocal recording of a growing Arabidopsis leaf (using the primary fluorescence of the chlorophyll) at two successive time points. Consider that the voxel extents corresponds to 2.0 µm in level-direction and 1.46484 µm in col and row direction, respectively. Hence all transformations between array and image domain should consider the anisotropic scaling parameter element_size_um=(2.0,1.46484,1.46484);

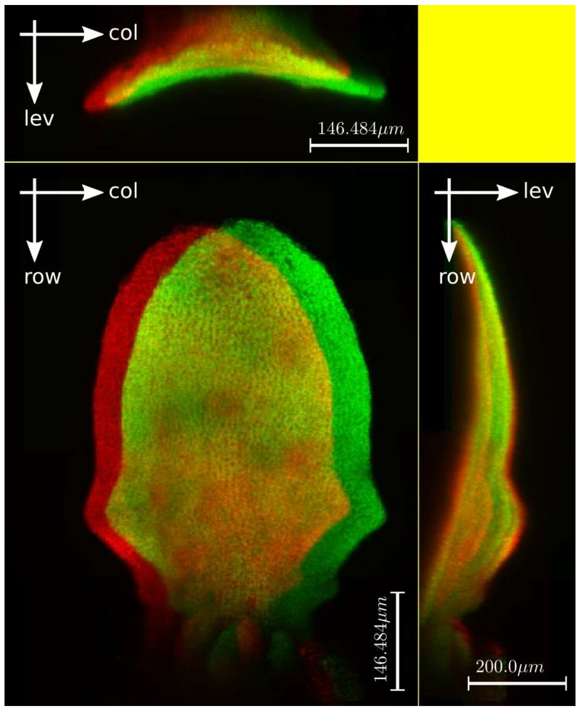

Fig. 1: Red - green overlay of orthogonal MIP projections of the fixed image and the moving image

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

2) write a function

```
blitz::Array<unsigned char,2> orthoMips( const blitz::Array<unsigned char,3>& arr)
```

that creates orthogonal maximum intensity projections of the given 3D array, and combines them into one image as in figure 1 (without annotations and in grayscale). **Hint:** compute all mips in parallel while iterating once through the 3D Array. For the combined image you do not have to consider rescaling the mips in row- and col-direction, i.e. they can stay skewed, though the aspect ratio is actually different compared to the mip in level-direction.

3) write a function

```
void savePPMImage( const blitz::Array<blitz::TinyVector<unsigned char,3>,2>& image,
                   const std::string& fileName)
```

that saves a color image in raw ppm (portable pixmap) format. The ppm format is very similar to the pgm format (see "man ppm" on a linux box). A blitz-Array of TinyVectors contains the raw data in the correct order, so you can directly save the memory starting at image.dataFirst().

4) Start with the main function: Load the raw data sets "leaf_t5_150x521x396_8bit.raw" and "leaf_t6_150x521x396_8bit.raw". Create an orthoMip of each dataset, combine them to a color image and save them as "iter_000.ppm". The output should be identical to figure 1. **Hint:** blitz-Arrays overload the operator[] for comfortable access of the TinyVector components (see http://blitz.sourceforge.net/resources/blitz-0.9.pdf, page 70). E.g., to access the zero component of all TinyVectors in an vector-array as scalar array (which correspond to the red channel in an RGB image), e.g. just use:

```
blitz::Array<blitz::TinyVector<unsigned char,3>,2> rgbImage( 480, 640);
blitz::Array<unsigned char,2> redChannel( 480, 640);
...
rgbImage[0] = redChannel;
```

5) write a function

```
blitz::TinyMatrix<float,4,4> createInverseRigidTransMatrix(
                              const blitz::TinyVector<size_t, 3>& srcArrShape,
                              const blitz::TinyVector<size_t, 3>& trgArrShape,
                              const blitz::TinyVector<float,3>& src_element_size_um,
                              const blitz::TinyVector<float,3>& trg_element_size_um,
                              blitz::TinyVector<float,6> params );
```

that creates the inverse rigid transformation 4x4 matrix. Use the function createInverseRotationMatrix() from the sample solution of exercise 2 (render_pollen_movie.cc) as basis and extend it appropriately. The params vector contains the 6 transformation parameters in the order: (shiftLev, shiftRow, shiftCol, rotateAroundLev, rotateAroundRow, rotateAroundCol). copy myproduct() functions from the sample solution.

6) Copy interpolNN() and transformArray() functions from the sample solution render_pollen_movie.cc. Use transformArray as Basis for the function:

```
float ssdOfFixedImAndTransformedMovingIm( const blitz::Array<unsigned char, 3>& movingIm,
                                          const blitz::TinyMatrix<float,4,4>& invMat,
                                          const blitz::Array<unsigned char, 3>& fixedIm,
                                          int step)
```

that computes the sum of squared differences between the fixed image and the transformed moving image (using a nearest neighbour interpolation). To speed up this function (it will be executed 12 times for each iteration of the optimizer), we only take a small part of the voxels into account using a subsampling in each direction with the factor 'step'. **Hints:** The pow() function of the standard library is very slow, because it allows arbitrary exponents, and therefore computes the results via multiple logarithms. Use blitz::pow2() instead, which just needs one multiplication.

7) Now continue writing the main function. Implement the best neighbor optimizer to find the best transformation parameters that registers the moving image (t6) to the fixed image (t5). Start with a step size of 16 (as well for the shifts as for the angles) and stop the iteration when the step size has become smaller than 0.125. Every time, when in an iteration better parameters where found, create a red-green orthoMip of the fixed image and the transformed moving image (like in figure 1) and save it as "iter_XXX.ppm", where XXX is the iteration number. Hints:

- For comfortable handling of the neighbors, create an array, that contains the 12 neighborDirections, i.e. (1,0,0,0,0,0), (-1,0,0,0,0,0), (0,1,0,0,0,0), ...
- Use blitz::TinyVector<float,6> to store a parameter set. You will need during the iteration lastParams, currentParams and bestParams
- Use step=4 for the subsampling in ssdOfFixedImAndTransformedMovingIm. This still provides a good estimate for the similarity while giving a speedup of a factor 64

8) (Extra Points) An initially good condition of the problem is crucial for successfully registering two images. Use the center of mass of both images to pre-align them before determining the Euclidean parameters using the Best Neighbor Optimizer.