Heimdali Documentation

Release 0.0.1

Dominique Béréziat, David Froger, Isabelle Herlin

March 17, 2015

Contents

1 Command line tools **3**
- 1.1 Installation . . . 3
- 1.2 Format options . . . 4
- 1.3 h5toinr . . . 4
- 1.4 inrtoh5 . . . 4
- 1.5 par . . . 5
- 1.6 tpr . . . 5
- 1.7 Artithmetic operations . . . 5
- 1.8 Local arithmetic operations . . . 6
- 1.9 cco . . . 7
- 1.10 extg . . . 7
- 1.11 raz . . . 8
- 1.12 melg . . . 8
- 1.13 fzoom . . . 8

2 Using ITK **11**
- 2.1 Converting INRimage simulation code to ITK . . . 11

3 Heimdali API **13**

4 Community **15**

5 Developer guide **17**
- 5.1 Build Heimdali in development mode . . . 17
- 5.2 Writting documentation . . . 18

6 Indices and tables **21**

Heimdali is a set of commmand line tools to perform Image processing tools, based on ITK and HDF5.

It also features some helper functions to work with ITK.

CHAPTER 1

Command line tools

1.1 Installation

The recommanded way to install **Heimdali** is using the Conda package manager.

1.1.1 *Conda* installation

Download the Python *2.7* version of *miniconda* corresponding to your plateform from http://conda.pydata.org/miniconda.html

Execute the downloaded file: this will install the Conda package manager.

Add the *dfroger binstar* channel to your config:

```
conda config --add channels https://conda.binstar.org/dfroger
```

1.1.2 Heimdali installation

Install the *heimdali* package in a environment called *heim* or whather name you want:

```
conda create -n heim heimdali
```

Activate the environment:

```
source activate heim
```

Your are ready to use **Heimdali**.

1.1.3 Test Heimdali installation

Download some input data to test:

```
git clone https://github.com/dfroger/heimdali-data
```

Execute the *par* command:

```
par heimdali-data/imtest_z5_y4_x3_c2.h5
```

1.1.4 Heimdali update

You can update **Heimdali** when a new version is released:

```
conda update heimdali
```

You may also want to keep the currently installed **Heimdali** version, and install a new version in another *Conda* environment:

```
conda create -n heim0.1 heimdali==0.1.0
```

You can now switch between the two version of **Heimdali**:

```
source activate heim
```

or

```
source activate heim0.1
```

1.2 Format options

These are options describing the image format. There are used each time an image is created.

option	description
-z N	Number of planes
-y N	Number of lines
-x N	Number of pixel per line
-v N	Number of value per pixel
-r	Floating point values

1.3 h5toinr

h5toinr convert an *HDF5* image into *INRimage* image.

1.3.1 Synopsis

```
h5toinr  [--] [--version] [-h] <inputFilename> <outputFilename>
```

1.4 inrtoh5

inrtoh5 convert an *INRimage* image into *HDF5* image.

1.4.1 Synopsis

```
inrtoh5  [--] [--version] [-h] <inputFilename> <outputFilename>
```

1.5 par

par prints format parameters of images.

1.5.1 Synopsis

```
par  [--wr <output.txt>] [--x0] [--y0] [--z0] [-o] [-x] [-y] [-z] [--]
     [--version] [-h] <INPUT> ...
```

1.5.2 Description

par print on *stdout* in the file *outout.txt*, the format parameters of iamges given as arguments.

The *–wr* options can be given the special file names *stdout* and *stderr*. If the file name *output.txt* starts with >>, result are written at the end of the file.

If one or more options *–x0 –y0 –z0 -x -y -z* is given, **par** print the corresponding parameters, in the *Format options*. This allow to use **par** in command subsitution, as for example:

```
create image-copy.h5 `par -x -y image.h5` -r
```

If no options are given, all format parameters are printed for all images on argument.

1.6 tpr

tpr prints the pixel values of a image subregion

1.6.1 Synopsis

```
tpr  [-x <NX>] [-y <NY>] [-z <NZ>] [-i <IX>] [-j <IY>] [-k <IZ>] [--]
        [--version] [-h] <IMAGE> ...
```

1.6.2 Description

tpr prints on standard output pixel values of *IMAGE* given in argument.

option	description
-ix, -iy, -iz	Index (counted from 0) of the subregion
-x, -y, -z	Size (counted from 0) of the subregion

1.7 Artithmetic operations

Arithmetic operations between two images element by element.

1.7.1 Synopsis

```
ad image0-in image1-in [image-out]
so image0-in image1-in [image-out]
mu image0-in image1-in [image-out]
di image0-in image1-in [image-out]
min image0-in image1-in [image-out]
max image0-in image1-in [image-out]
```

1.7.2 Description

All of these commands perform an operation between *image0-in* and *image1-in* and write result to *image-out*. If argument *image0-in* or *image1-in* is equal to -, the command reads on standard input. If argument *image-out* is absent, the command writes to standard output.

command	description
ad	Add two images
so	Subtract two images
mu	Multiply two images
div	Divide two images
min	Compute minimum of two images
max	Compute maximum of two images

All operation are performed on pixels element by element.

1.7.3 See also

See also *Local arithmetic operations*.

1.8 Local arithmetic operations

1.8.1 Synopsis

```
bi -n value [image-in] [image-out]
sc -n coeff [image-in] [image-out]
sd -n coeff [image-in] [image-out]
lo value [image-in] [image-out]
exp [image-in] [image-out]
ra [image-in] [image-out]
sba -n threshold [image-in] [image-out]
sha -n threshold [image-in] [image-out]
mb -n threshold [image-in] [image-out]
mh -n threshold [image-in] [image-out]
mo [image-in] [image-out]
car [image-in] [image-out]
vb -n threshold value [image-in] [image-out]
vh -n threshold value [image-in] [image-out]
```

1.8.2 Description

All of these commands perform an operation on *image-in* (which can be of any type), and write result in *image-out*.

image-in and *image-out* must have the same dimension.

If argument *image-in* is absent or egal to -, the command reads on standard input.

If argument *image-out* is absent, the command writes on standard output.

Values passed with *-n* are *float*.

command	description
bi	Add *value* to each pixel.
sc	Multiply each pixel by *coeff*.
sd	Divied each pixel by *coeff*.
lo	Compute logarithm of each pixel.
exp	Compute exponential of each pixel.
ra	Compute square root of each pixel.
sba	Every pixel inferior or egual to *threshold* is replaced by *threshold*.
sha	Every pixel greater or egual to *threshold* is replaced by *threshold*.
mb	Every pixel inferior or egual to *threshold* is replaced by 1 and others by 0.
mh	Every pixel greater or egual to *threshold* is replaced by 1 and others by 0.
mo	Compute each pixel modulo
car	Compute each pixel square
vb	Every pixel inferior or egual to *threshold* is replaced by *value*.
vh	Every pixel greater or egual to *threshold* is replaced by *value*.

1.9 cco

cco change pixel type of image

1.9.1 Synopsis

```
cco  [--] [--version] [-h] <inputFilename> <outputFilename>
```

1.9.2 Description

cco convert *IMAGE-IN* into *IMAGE-OUT*.

1.10 extg

extg extract a image subregion

1.10.1 Synopsis

```
extg  [-x <NX>] [-y <NY>] [-z <NZ>] [-i <IX>] [-j <IY>] [-k <IZ>] [--]
         [--version] [-h] <FILE-IN> <FILE-OUT>
```

1.10.2 Description

extg extract a subregion in *FILE-IN* and write it in *FILE-OUT*.

option	description
-ix, -iy, -iz	Index (counted from 0) of the subregion
-x, -y, -z	Size (counted from 0) of the subregion

1.11 raz

raz Fill an image with zero values

1.11.1 Synopsis

```
raz  [-x <NX>] [-y <NY>] [-z <NZ>] [-i <IX>] [-j <IY>] [-k <IZ>] [--]
        [--version] [-h] <IMAGE> ...
```

1.11.2 Description

raz fill the image *IMAGE* given in arguments with zero values.

option	description
-x, -y, -z	Size (counted from 0) of the subregion

1.12 melg

melg mix two images, testing on pixel values.

1.12.1 Synopsis

```
melg  [--idv <IDV>] [--idx <IDX>] [--idy <IDY>] [--idz <IDZ>] [--ivo
      <IXO>] [--ixo <IXO>] [--iyo <IYO>] [--izo <IZO>] [--ivi <IXI>]
      [--ixi <IXI>] [--iyi <IYI>] [--izi <IZI>] [--] [--version] [-h]
      <IMAGE-IN> <IMAGE-OUT
```

1.12.2 Description

melg replace pixels of a subregion *IMAGE-OUT* with pixels of *IMAGE-IN*.

option	description
-ixi, -iyi, -izi	Index (counted from 0) of *IMAGE-IN* subregion
-ixo, -iyo, -izo	Index (counted from 0) of *IMAGE-OUT* subregion
-idx, -idy, -idz	Size of subregion

1.13 fzoom

fzoom enlarge or reduce an iamge.

1.13.1 Synopsis

```
fzoom  [--sc <X>] [--] [--version] [-h] <IMAGE-IN> <IMAGE-OUT>
```

1.13.2 Description

fzoom increse or reduce the size of an image by lineary interpolation.

CHAPTER 2

Using ITK

2.1 Converting INRimage simulation code to ITK

2.1.1 Image definition

The file heimdali/itkhelper.hxx defines the type of an image.

typedef float **PixelFloat**

const unsigned int **ImageDimension**
All images are 3-dimensional.

typedef itk::VectorImage< PixelFloat, ImageDimension > **ImageFloat**

Images are 3-dimensionial, of size *(nz, ny, nx)*.

Images contains *nx* * *ny* * *nx* pixels, and each pixel is a vector of *nv* value of type *float*.

nz	Number of planes
ny	Number of rows
nx	Number of columns
nv	Number of values per pixel

Internally, the image is stored in a continous block of memory, ie is a *float**.

2.1.2 Building

2.1.3 Proving command line interface

2.1.4 Creating image

ImageFloat::Pointer **CreateImage** (unsigned int *nx*, unsigned int *ny*, unsigned int *nz* = 1, unsigned int *nv* = 1)

itkhelper.hxx defines the `CreateImage` function to create an image in the temporary program memory. It does not do any persistant operation on the disk.

For example:

```
Heimdali::ImageFloat::Pointer image = Heimdali::CreateImage(5,5,5,2);
```

create a image of 3-dimensional image with 5 planes, 5 rows and 5 columns, where is value is a vector of 2 floats.

The following code:

```
Heimdali::ImageFloat::Pointer image = Heimdali::CreateImage(5,5);
```

create a image of 3-dimensional image with 1 planes, 5 rows and 5 columns, where is value is a vector of 1 float.

See the file createInputImage.cxx for a running example.

2.1.5 Read and writting image

Working with the whole image

Working line by line

Working plane by plane

2.1.6 Modifiyting image

Using C pointer

Using wrapped C pointer

Using GetPixel, SetPixel

Using ITK iterators

Using ITK convolution

Using ITK filter

CHAPTER 3

Heimdali API

CHAPTER 4

Community

CHAPTER 5

Developer guide

5.1 Build Heimdali in development mode

Development mode is usefull for developer. It consist in iterating in the cycle:

- modify source code
- build
- run the test

without having to run the *make install* step.

5.1.1 Install dependencies

Create a *conda* enviromnent named *heimdali-dev* containing all dependencies:

```
conda config --add channels http://conda.binstar.org/dfroger
conda create -n heimdali-dev h5unixpipe itk-heimdali-dbg libinrimage tclap cmake pip pexpect
```

For the rest of the section, we need to activate the conda environment, and set the CONDA_ENV_PATH environment variable:

```
source activate heimdali-dev
hash -r
CONDA_ENV_PATH=$(conda info -e | grep '*' | tr -s ' ' | cut -d" " -f3)
```

Install lettuce:

```
pip install lettuce
```

5.1.2 Build Heimdali

Build heidmali, asking CMake to search dependances in the Conda environment:

variable	meaning
CONDA_ENV_PATH	For example, *~/miniconda/envs/heimdali-dev*
CMAKE_PREFIX_PATH	Where *CMake* will search for dependent libraries
..	Path to Heimdali main CMakeLists.txt

On Mac OS X your will need to install */Developer/SDKs/MacOSX10.5*, and use it:

```
export MACOSX_DEPLOYMENT_TARGET=10.5

cd heimdali
mkdir build; cd build
cmake -DCMAKE_PREFIX_PATH=$CONDA_ENV_PATH ..
make
```

5.1.3 Configure examples

As before, the Conda environment is used. Moreover, because Heimdali has been built in *heimdali/build* and is not installed (development mode), we need to specified all paths to CMake.

```
cd heimdali
HEIMDALI_ROOT=$PWD
[ `uname` == 'Darwin' ] && EXT=dylib || EXT=so
cd example
mkdir build; cd build
cmake \
    -DCMAKE_PREFIX_PATH=$CONDA_ENV_PATH \
    -DHEIMDALI_INCLUDE=$HEIMDALI_ROOT/libheimdali \
    -DITKINRIMAGEIO_INCLUDE=$HEIMDALI_ROOT/itkINRimageIO/include \
    -DHEIMDALI_LIBRARY=$HEIMDALI_ROOT/build/libheimdali/libheimdali.$EXT \
    -DITKINRIMAGEIO_LIBRARY=$HEIMDALI_ROOT/build/itkINRimageIO/libitkINRImageIO.$EXT \
    ..
```

5.1.4 Run functional tests

Get Heimdali data files, and set *HEIMDALI_DATA_DIR*:

```
git clone https://github.com/dfroger/heimdali-data
export HEIMDALI_DATA_DIR=/path/to/heimdali-data
```

Add path to the built executables:

```
cd heimdali
export PATH=$PWD/build/cmd:$PATH
```

Run the functional tests:

```
cd tests
lettuce
```

5.2 Writting documentation

Install Sphinx and Doxygen:

```
sudo apt-get install doxygen
conda create -n heimdali-doc sphinx
source activate heimdali-doc
```

Build the documentation:

```
cd doc
make html
```

View the documentation:

```
cd doc
firefox _build/html/index.html
```

Note that breathe, a Sphinx extension, is already provided in *heimdali/doc/ext/breathe*.

CHAPTER 6

Indices and tables

- *genindex*
- *modindex*
- *search*