IO support for CUB and CUB.GZ image formats

Release 1.00

Pablo D. Burstein[1] and Paul Yushkevich[1] and James C. Gee[1]

September 26, 2006

[1]University of Pennsylvania, Philadelphia, PA 19104, USA

Abstract

This document describes the implementation of IO support for .cub (along with its compressed version, .cub.gz) image format. For this aim, and complying with ITK architecture, the necessary subclasses of ImageIOBase and ImageIOBaseFactory, itkVoxBoCUBImageIO and itkVoxBoCUBImageIOFactory, respectively, are written. This enables seamless integration with the ImageReader and ImageWriter classes. This paper is properly accompanied by the source code, input data, parameters and output data that the authors used for testing the classes described in this paper. Source code files are submitted separately and are not included in the body of this paper. Since this paper does not present a new algorithm but a simple IO extension, no extensive testing is presented and no examples are shown. To learn more about .cub format, please refer to (`www.voxbo.org`).

Contents

1 Background

VoxBo (`www.voxbo.org`) [2] is a distributed processing framework, developed at the Center of Functional Neuroimaging at the University of Penssylvania, with the purpose of processing and analyzing brain functional MRI data. VoxBo is written in C++, and it is escentially a fair translation of some SPM functionality, originally written in Matlab. Along with the software, VoxBo defines two new image formats:

- .cub, is a 3D image format, used for anatomical data and single EPI frames. It is important to notice that this format defines a proper origin field, lacking, for instance, in Analyze.

- .tes is a 4D image format, designed to contain the temporal EPI information. It is worth noticing that this format organizes the voxels in such a way to optimize longitudinal (temporal) voxel access. Furthermore, the .tes format is compressed through encoding of the functional voxels alone, i.e., background voxels are not stored for each frame, but only for the first one.

In this work we present the necessary IO classes that allow ImageFileReader and ImageFileWriter to properly identify, read and write .cub (and cub.gz) files only. We plan to provide .tes support in a subsequent work. This effort is carried out with the purpose of letting VoxBo implementors to take advantage of ITK's power and overcome the limitations currently afflicting VoxBo.

2 Methods

We subclass `itk::(`ImageIOBase) and `itk::(`ImageIOBaseFactory) to implement `itk::(`VoxBoCUBImageIO) and `itk::(`VoxBoCUBImageIOFactory) [1], respectively, to support IO for .cub (and .cub.gz) image format. As opposed to Analyze and others, .cub format consists of a single file containing both the meta-data, header section, followed by the image data. The header occupies the first part of the file, and it is encoded in ASCII; this makes it (humanly) readable using any editor/viewer. All the meta-data is contained in this header. In addition to fields such as spacing and dimesion, which can be translated into Image fields, .cub meta-data defines other "special" fields, customized to keeping track of the processing path the image undergoes. Since for many of these fields there is no translation into Image fields, we use the `itk::(`MetaDataDictionary), refernced by `itk::(`DataObject), to keep consistent with the .cub format. We also take care to update those .cub specific fields requiring modification through the manipulation process, e.g., modification date. We do translate, though, .cub fields into (and back from) Image fields whenever possible; no redundancy is kept in the data stored in the MetaDataDictionary. Exceptions are managed as usual for `itk::(`ImageIOBase).

3 Software

Along with this submision we provide the corresponding source code files implementing the contributed classes:

- itkVoxBoCUBImageIO.cxx
- itkVoxBoCUBImageIO.h
- itkVoxBoCUBImageIOFactory.cxx
- itkVoxBoCUBImageIOFactory.h

4 Testing

Along with this submision we provide the corresponding test application and images:

- ConvertImage.cxx

- inputTestImage.cub
- inputTestImage.cub.gz
- outputTestImage1.cub
- outputTestImage1.cub.gz
- outputTestImage2.cub
- outputTestImage3.cub
- outputTestImage3.cub.gz

To test our classes we run the following combinations:

- `convertImage inputTestImage.cub outputTestImage1.cub`
- `convertImage inputTestImage.cub outputTestImage1.cub.gz`
- `convertImage inputTestImage.cub.gz outputTestImage2.cub`
- `convertImage outputTestImage1.cub outputTestImage3.cub.gz`
- `convertImage outputTestImage1.cub.gz outputTestImage3.cub`

Simply enough, convertImage reads the input image and writes a copy to the output image in the specified output format (given as the output file extension). The first test just copies inputTestImage.cub to a another .cub image. The second test converts inputTestImage.cub to its corresponding compressed version. The third test does just the opposite of the second, i.e., it converts inputTestImage.cub.gz to its uncompressed version. The fourth and fifth tests verify that the new generated images are readable. We check, using third party visualization software,that the output images are exact replicas of the used input images, the only differences being possible changes in the relevant fields updated as required by VoxBo IO libraries.

As an additional test, a custom version of ITK was compiled with our new classes and InsightSNAP was extended and compiled with this custom version. We loaded and saved .cub and .cub.gz images in InsightSNAP without any reported problems.

References

[1] Luis Ibanez, Will Schroeder, Lydia NG, and Josh Cates. *The ITK Software Guide: The Insight Segmentation and Registration Toolkit.* USA, 2000. 2

[2] CfN University of Pennsylvania. *www.voxbo.org*. 1