A Streaming IO Base Class and Support for Streaming the MRC and VTK File Formats

Release 1.00

Bradley C. Lowekamp[1] and David T. Chen[1]

June 9, 2010

[1]National Library Of Medicine

Abstract

This paper describes our contribution of three new classes to the Insight Toolkit community. We present a new `ImageIO` base class for streaming image file, along with two derived `ImageIO` classes for the VTK and the MRC file formats.

Contents

1 Introduction

We have created a new file reader for MRC image files, a base class of extracted reusable streaming code, and improvements to the VTK file reader that reuses our new streaming base class. We wrote an ImageIO

class to natively read yet another format to have access to information in the extended header such as acquisition parameters of the microscope and to be able to stream the data for out-of-core processing. The result is a robust reader and writer for ITK that can take advantage of many of the toolkits powerful features. The MRCImageIO class is derived from a base class which overrides common I/O methods to implement streaming and provide addition low level I/O methods. To demonstrate the utility and flexibility of the StreamingImageIOBase class, we derived a new VTKImageIO class from it. Thus our VTKImageIO class utilizes the base class's streaming methods.

Streaming is the process of sequentially processing sub-regions part of the largest possible region) of an image through the pipeline. At any moment only a sub-region of an image is processed by each filter. Then the entire image is reassembled from the sub-regions into memory or on the file system. To perform out-of-core processing, the ImageIO classes involved in the pipeline must support streamed reading and streamed writing or the entire image will be buffered in memory.

2 Implementation

At the core of our work is a new class from which new streaming ImageIO classes should be derived. Currently, the itk::ImageIOBase class contains a large number of methods, many of which need to be repetitively overridden for common features implemented in derived ImageIO classes. The purpose of the StreamingImageIOBase is to overload the methods needed for streaming when used by itk::ImageFileReader and itk::ImageFileWriter. These include:

```
bool CanStreamWrite( void );
bool CanStreamRead( void );
ImageIORegion GenerateStreamableReadRegionFromRequestedRegion( const ImageIORegion & requested ) const;
unsigned int  GetActualNumberOfSplitsForWriting( unsigned int numberOfRequestedSplits,
                                                 const ImageIORegion &pasteRegion,
                                                 const ImageIORegion &largestPossibleRegion );
```

The CanStreamWrite and CanStreamRead methods simply return true to indicate the derived ImageIO class supports this feature.

The StreamingImageIOBase class is designed to directly support the streaming of arbitrary regions for both reading and writing. To facilitate the use ofregions, the GenerateStreamableReadRegionFromRequestedRegion method returns the requested region passed as an argument.

The implementation of GetActualNumberOfSplitsForWriting similarly supports derived ImageIO classes to allow writing and pasting of arbitrary regions to a file. However, pasting to existing files adds the problem of compatibility between the image file and the format requested by an ImageIO object. Therefore this method must verify that an existing file image contains the same pixel type and image information. If this check fails, an exception will be generated.

These methods should work for a large number of ImageIO classes. However, if additional restriction are needed, those tests could be made in an overridden method, followed by a call to the super-class's (StreamingImageIOBase) implementation of the method.

There is one protected utility method provided in this class:

```
virtual bool RequestedToStream( void ) const;
```

This method performs the surprisingly complicated and error prone comparison of the IORegion to the image file's size. The complication arises from the fact that the IORegion may not have the same number of dimensions as the file's image. When the dimensions are different yet the IORegion references all the pixels in the image, the request is interpreted as non-streaming. The situation where the IORegion object has not been set, only an issue if itk::ImageFileReader or itk::ImageFileWriter are not used, is not considered. If RequestedToStream returns true, the entire image (the largest possible region) is not requested and therefore streaming functionality is required.

A large number of image file formats are simply a header followed by the raw binary data. To facilitate these ImageIO types the following methods are provided:

```
virtual SizeType GetDataPosition( void );
virtual bool StreamReadBufferAsBinary( std::istream& os, void *buffer );
virtual bool StreamWriteBufferAsBinary( std::ostream& os, const void *buffer );
void OpenFileForReading( std::ifstream& os, const char* filename );
void OpenFileForWriting( std::ofstream& os, const char* filename, bool truncate );
```

The GetDataPosition method should return the data position in a file where the raw image data begins. The type SizeType was carefully chosen because it is currently defined as std::streamoff. However the name may be confusing since it is overused across ITK.

Utilizing the IORegion and the other image information (set in the ImageIO), the streaming read and write methods can perform the random access and std::iostream operations needed. These methods are the heart of streaming input and output operations, and if they can be reused in derived ImageIO classes, it is a big win!

2.1 MRCImageIO

The MRC file type is a popular file format which is arguably becoming the standard for electron microscopy. It is supported by such applications as Boulder Laboratory's IMOD, UCSF's Chimera, 3D Visual's Amira and is among the formats included in LOCI's Bio-Formats. While ITK does not need to support every image format, the lack of streaming supported by many other ImageIO formats, the additional information available in the extended header, along with the prevalence of the format and common support in existing electron microscopy software are sufficient motivations to us for natively supporting this extension in ITK.

The format appears to have originated with a software package for crystallography from the MRC Laboratory of Molecular Biology [1]. While the original FORTRAN software is either no longer available or not open source, documentation about the original format still exists: http://www2.mrc-lmb.cam.ac.uk/image2000.html. Boulder Laboratory's IMOD is a currently leading software package for reconstruction and registration of 3D electron microscopy data. Their documentation provides some additional header fields which shall be regarded as options for our implementation: http://bio3d.colorado.edu/imod/doc/mrc_format.txt. These documents provide the working standard for the MRC file format.

The structure of the file format is not novel as it simply a header and an optional extended header, followed by the raw data. This file structure allowed us to create the previously described StreamingImageIOBase class with modular reading and writing methods. The implementation of MRCImageIO::Write and MRCImageIO::Read are good examples of how to use the base class to implement the streaming process. These methods illustrate the details of when to stream, read the entire image, erase the file and write the header. These details must be implemented correctly for paste streaming to operate correcly.

The MRC file format supports 3D images with the following pixel types:

- unsigned 8-bit integers
- signed 16-bit integers
- 32-bit floats
- 2 16-bit for complex data (not supported by ITK)
- 2 float for complex data
- unsigned 16-bit (semi-standard format)
- 3 unsigned 8-bit integer for RGB (semi-standard format)

One of the difficulties in handling an image format is ensuring that the geometric information including orientation is imported and exported correctly. As illustrated in section 4.1.4 of the ITK Software Guide[2], ITK uses a right handed 2-D image view, along with a right handed 3-D view of the world space coordinate system. The MRC world coordinates are similarly right-handed, so no corrections in the orientation matrix are needed. Unfortunately, the details of creating a image-to-world transformation matrix from the MRC header information is neither well specified nor consistent across applications. Utilizing the header member names from IMOD's implementation we use the following calculations for computing pixel spacing and origin:

$$\begin{aligned} P_{origin} &= \begin{pmatrix} x_{org} \\ y_{org} \\ z_{org} \end{pmatrix} \\ V_{spacing} &= \begin{pmatrix} x_{len}/mx \\ y_{len}/my \\ z_{len}/mz \end{pmatrix} \end{aligned} \tag{1}$$

The variable P_{origin} and $V_{spacing}$ are ITK's definition. Specifically P_{origin} is the world coordinates for the center of the pixel at the zero index. The origin is important to note because there currently exists discrepancies in how these attributes are interpreted among different applications. The units of $V_{spacing}$ spacing are usually angstroms (10^{-10} meters) for MRC files, as opposed to 10^{-4} meters, which is the convention for DICOM. Other fields not needed in the header maybe verified for sanity, or ignored. Additional fields which an `MRCImageIO` object does not use but have expected values may create a warning if important information is not handled correctly, such as the n_{start} fields.

During image acquisition additional per slice information can be recorded in the optional extended header. This information may include geometric information about the stage, magnification, exposure, etc. The `MRCHeaderObject` class contains public members to access data structures that contain this information. Correctly handling all extended headers is an area that will require future development for more types and formats. The header object is placed into the `MetaDataDictionary` of the `ImageIO` object and then copied to the output image of `itk::ImageFileReader`.

2.2 VTKImageIO

The VTK extension is a popular visualization file format, and part of the file specification is image data (or structured points)[3]. Our work converted the existing `VTKImageIO` class to utilize the

StreamingImageIOBase base class, enabling streaming, and fixed a few known bugs along the way. In this revision we paid special attention to matching the VTK file format specification. The VTK format differs from the canonical view of a StreamingImageIOBase derived class because the VTK format can have binary and ASCII data. Because ASCII numbers can vary in character length, random access is not efficient and prevents streaming. Therefore the GenerateStreamableReadRegionFromRequestedRegion and GetActualNumberOfSplitsForWriting methods were overloaded to exclude streaming in this case. To show the utility of StreamingImageIOBase here is an excerpt from VTKImageIO::Read:

```
std::ifstream file;

if( this->RequestedToStream() )
  {

  itkAssertOrThrowMacro( m_FileType != ASCII, "Can not stream with ASCII type files" );

  // open and stream read
  this->OpenFileForReading(file, this->m_FileName.c_str());

  itkAssertOrThrowMacro( this->GetHeaderSize() != 0, "Header size is unknown...");
  this->StreamReadBufferAsBinary(file, buffer);

  }
else
...
```

3 Future Work

Our work provides the needed structure for future streaming ImageIO classes by separating the implementation of methods in itk::ImageIOBase that do not support streaming ImageIOs from those that do. In our opinion this separation should be take further. While the itk::ImageIOBase class must provide the streaming I/O interface, its implementation should only directly support the non-streaming ImageIO methods. The streaming implementation should be moved to the StreamingImageIOBase class.

In the future, more ImageIO classes should enable streaming through the StreamingImageIOBase class. Such revisions will help to ensure that the StreamingImageIOBase streaming methods have a robust interface suitable for a variety of purposes. The easiest ImageIO class to upgrade would be RawImageIO through the new stream enabled binary read and write methods. However, upgrading the itk::TiffImageIO class would be the most useful because of its popularity as an intermediate format and because the implementation utilizes a library different from the classes presented here. Additionally, the itk::MetaImageIO class should change its parent class.

4 Software Requirements and Usage

The latest bug fixes and features of ITK are needed for these classes to run correctly.

- Insight Toolkit 3.18
- CMake 2.2

For the itk::ImageFileReader and itk::ImageFileWriter classes to utilize the new ImageIO classes they must be registered with the itk::ImageIOFactory. Included with the distributed source code is a LocalFactory which can be used to override and register the factory functions of VTKImageIO and MRCImageIO. Here is how to manually registering the factory:

```
itk::Local::LocalFactory::RegisterOneFactory();
```

5 Acknowledgement

The authors would like to thank the on going collaboration with Dr. Sriram Subramaniam's High Resolution Electron Microscopy Laboratory of Cell Biology at the National Cancer Institute. Our work was performed at the National Library of Medicine's Office of High Performance Computing and Communications under the supervision of Dr. Terry S. Yoo and Dr. Michael J. Ackerman.

References

[1] R. A. Crowther, R. Henderson, and J. M. Smith. Mrc image processing programs. *Journal of Structureal Biology*, 116:9–16, 1996. 2.1

[2] L. Ibanez, W. Schroeder, L. Ng, and J. Cates. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-15-7, http://www.itk.org/ItkSoftwareGuide.pdf, second edition, 2005. 2.1

[3] Kitware Inc. *The VTK User's Guide Updated for version 4.4"*, 2004. 2.2