Digital Imaging Group
Proposal for Unifying and Standardizing Source Code Writing

23rd February 2004

1 Coding Language

The language used will be C++. Features of C++ (such as classes, default arguments, overloading, templates and exceptions) should be utilized to produce efficient and robust code. The extension of the source code and header files are .cpp and .h, respectively. C++ manuals are accessible at **`http://www.dig.cs.gc.cuny.edu/manuals/manuals.html`**

2 Image Object Model (Internal Representation of Images)

2.1 Five Dimensions

The image object model is a general structure that can be used to represent a series of images in different areas, thus allowing easy data sharing in our lab. Each file of this format has a header written in ASCII that provides information about the series

of images present in it. The values of the data dimensions represent the number of Voronoi neighborhoods per each dimension, while the values of other variables are based on units.

Even though the basis of the image model is 3D, we would also like to be able to pack multiple 2D images into a single file. However, packing 2D images as slices of a 3D image confuses the distinction between 2D and 3D and should be avoided. We can distinguish such differences by interpreting the file header. Since we also want to be able to store multiple 3D images we will need an additional dimension.

Most of the data sets we work with have single values at each pixel or voxel (gray scale values), but situations may arise where each voxel may have multiple values, for example, to represent colors, such as RGB (Red-Green-Blue) and CMY (Cyan-Magenta-Yellow). This requires yet another dimension in the image model.

The five dimensions are therefore:

1. Channels - one or more values associated with each voxel.
2. *x*-dimension.
3. *y*-dimension.
4. *z*-dimension.
5. Images - a series of images, typically with some commonality (such as 2D projections from the same particle, a tilt series, or a time series).

We assume a coordinate system with axes in the order *xyz* (i.e., *x* is the fastest varying index). The full data set includes also channels (*c*) and possibly several images (*n*) with the fixed order *cxyzn*. (That is, the data set is 5D with *c* being the fastest varying index and *n* the slowest.)

2.2 File Format

A file consists of a header written in XML followed by the data, that can be written in ASCII or binary. The header contains information that is general to all files, such as size and sampling rate, as well as application-specific information, such as the angle of the projection for SNARK03 prjfil files.

The specific headers to this date are:

1. SNARK03 recfil.
2. SNARK03 prjfil.

The library provides functions that validate the XML header of a file against schema files (.xsd), which are listed in Appendix A.

3 Conventions

It is important to note that this section is about the internal representation of data followed by the programs and not about file formats. These are two totally independent issues.

3.1 Data

1. Angles used internally are expressed in radians but they are provided by the user in degrees.

2. Figures of Merit (FOMs) are defined so that the FOM for the phantom is 1 and a smaller FOM indicates a "worse" reconstruction. The values should generally range from 0 to 1.

3. We will deal with 3D arrays of size N_x, N_y, N_z. The vector (N_x, N_y, N_z) is defined by the tag <dimensions> in the XML header of the data file (see Apendix A). (To represent a 2D array, we select one of N_x, N_y or N_z to be 1.) The origin will be put at the "center" (c_x, c_y, c_z) of the image/volume according to the following:

$$c_x = \left\lfloor \frac{N_x}{2} \right\rfloor, \; c_y = \left\lfloor \frac{N_y}{2} \right\rfloor, \text{ and } c_z = \left\lfloor \frac{N_z}{2} \right\rfloor.$$

   The indices (i, j, k) of such arrays will be in the ranges $-\left\lfloor \frac{N_x}{2} \right\rfloor \leq i \leq \left\lfloor \frac{N_x-1}{2} \right\rfloor$, $-\left\lfloor \frac{N_y}{2} \right\rfloor \leq j \leq \left\lfloor \frac{N_y-1}{2} \right\rfloor$ and $-\left\lfloor \frac{N_z}{2} \right\rfloor \leq k \leq \left\lfloor \frac{N_z-1}{2} \right\rfloor$.

4. The values in the data should be such that increase in value indicates increase in brightness. In particular, black should always be represented by a lower value than white.

3.2 Geometric Conventions for the 3D Coordinate System

1. Right-handed; i.e., if the z axis points towards the observer and the y axis points up, then the x axis points to the right.

2. Right-handed rotation (if an axis points towards the observer, then a positive rotation angle indicates an anticlockwise rotation around that axis).

3. Default view is down the z axis (vector (0,0,1), with z values decreasing with distance from the viewer), the x axis is horizontal left to right and the y axis is vertical bottom to top, and the center of the array is projected to the center of the screen.

4. All grids are stored as a cubic grid, without any compression. The grid types allowed are simple cubic grid (SC), fcc (FCC1 and FCC2), bcc (BCC1, BCC2, BCC3 and BCC4) and hexagonal (HEX1 and HEX2).

5. The geometric location of the point p corresponding to (i, j, k), with $(\overrightarrow{\Delta_x}, \overrightarrow{\Delta_y}, \overrightarrow{\Delta_z})$ denoting the sampling rate vectors defined by the tags <sampling_rate_x>, <sampling_rate_y> and <sampling_rate_z> (see Appendix A), respectively, according to the grid type, is:

   - For SC, $p = (i\overrightarrow{\Delta_x}, j\overrightarrow{\Delta_y}, k\overrightarrow{\Delta_z})$.

- For FCC1, $p = \begin{cases} (i\overrightarrow{\Delta_x}, j\overrightarrow{\Delta_y}, k\overrightarrow{\Delta_z}), & \text{if } (i+j+k) \bmod 2 = 0, \\ \text{undefined}, & \text{otherwise.} \end{cases}$
- For FCC2, $p = \begin{cases} (i\overrightarrow{\Delta_x}, j\overrightarrow{\Delta_y}, k\overrightarrow{\Delta_z}), & \text{if } (i+j+k) \bmod 2 = 1, \\ \text{undefined}, & \text{otherwise.} \end{cases}$
- For BCC1, $p = \begin{cases} (i\overrightarrow{\Delta_x}, j\overrightarrow{\Delta_y}, k\overrightarrow{\Delta_z}), & \text{if } (i \equiv j \equiv k) \bmod 2, \\ \text{undefined}, & \text{otherwise.} \end{cases}$
- For BCC2, $p = \begin{cases} (i\overrightarrow{\Delta_x}, j\overrightarrow{\Delta_y}, k\overrightarrow{\Delta_z}), & \text{if } (i \equiv j \not\equiv k) \bmod 2, \\ \text{undefined}, & \text{otherwise.} \end{cases}$
- For BCC3, $p = \begin{cases} (i\overrightarrow{\Delta_x}, j\overrightarrow{\Delta_y}, k\overrightarrow{\Delta_z}), & \text{if } (i \equiv k \not\equiv j) \bmod 2, \\ \text{undefined}, & \text{otherwise.} \end{cases}$
- For BCC4, $p = \begin{cases} (i\overrightarrow{\Delta_x}, j\overrightarrow{\Delta_y}, k\overrightarrow{\Delta_z}), & \text{if } (j \equiv k \not\equiv i) \bmod 2, \\ \text{undefined}, & \text{otherwise.} \end{cases}$
- For HEX1, $p = \begin{cases} (i\overrightarrow{\Delta_x}, j\sqrt{3}\overrightarrow{\Delta_y}, k\overrightarrow{\Delta_z}), & \text{if } (i+j) \bmod 2 = 0, \\ \text{undefined}, & \text{otherwise.} \end{cases}$
- For HEX2, $p = \begin{cases} (i\overrightarrow{\Delta_x}, j\sqrt{3}\overrightarrow{\Delta_y}, k\overrightarrow{\Delta_z}), & \text{if } (i+j) \bmod 2 = 1, \\ \text{undefined}, & \text{otherwise.} \end{cases}$

3.3 Discrete 3D Fourier Transform

Given a 3D array, denoted by $f(i, j, k)$, its *discrete Fourier transform* is a 3D array of the same size defined by

$$[\mathfrak{F} f](u, v, w) = \frac{1}{N_x N_y N_z} \sum_{i=-\lfloor \frac{N_x}{2} \rfloor}^{\lfloor \frac{N_x-1}{2} \rfloor} \sum_{j=-\left\lfloor \frac{N_y}{2} \right\rfloor}^{\left\lfloor \frac{N_y-1}{2} \right\rfloor} \sum_{k=-\left\lfloor \frac{N_z}{2} \right\rfloor}^{\left\lfloor \frac{N_z-1}{2} \right\rfloor} f(i, j, k)\, e^{-2\pi\sqrt{-1}\left(\frac{ui}{N_x}+\frac{vj}{N_y}+\frac{wk}{N_z}\right)},$$

for all u, v, w such that $-\lfloor \frac{N_x}{2} \rfloor \le u \le \left\lfloor \frac{N_x-1}{2} \right\rfloor$, $-\left\lfloor \frac{N_y}{2} \right\rfloor \le v \le \left\lfloor \frac{N_y-1}{2} \right\rfloor$ and $-\left\lfloor \frac{N_z}{2} \right\rfloor \le w \le \left\lfloor \frac{N_z-1}{2} \right\rfloor$. Similarly, the *discrete inverse Fourier transform* of $f(i, j, k)$ is defined by

$$[\mathfrak{F}^{-1} f](u, v, w) = \sum_{i=-\lfloor \frac{N_x}{2} \rfloor}^{\lfloor \frac{N_x-1}{2} \rfloor} \sum_{j=-\left\lfloor \frac{N_y}{2} \right\rfloor}^{\left\lfloor \frac{N_y-1}{2} \right\rfloor} \sum_{k=-\left\lfloor \frac{N_z}{2} \right\rfloor}^{\left\lfloor \frac{N_z-1}{2} \right\rfloor} f(i, j, k)\, e^{2\pi\sqrt{-1}\left(\frac{ui}{N_x}+\frac{vj}{N_y}+\frac{wk}{N_z}\right)},$$

for all u, v, w such that $-\lfloor \frac{N_x}{2} \rfloor \le u \le \left\lfloor \frac{N_x-1}{2} \right\rfloor$, $-\left\lfloor \frac{N_y}{2} \right\rfloor \le v \le \left\lfloor \frac{N_y-1}{2} \right\rfloor$ and $-\left\lfloor \frac{N_z}{2} \right\rfloor \le w \le \left\lfloor \frac{N_z-1}{2} \right\rfloor$.

4 Documentation

Programs should be profusely commented. In order to do so the programmer should add the documentation right into the source code he/she develops.

We use DOC++ for generating documentation from source code. There are two types of comments that the programmer should distinguish. One type is for remembering some implementation issues (this will be made with /* */ or //), while the other type is used for functions, classes, etc., so that he/she or someone else will be able to understand and use the code later on. Such comments are referred to as DOC++ comments (and are made with /** */ or ///). Each DOC++ comment generates a manual entry for the declaration following it in the source code.

Examples of DOC++ comments:

```
/** A global function. This is an example of how to document
global scope functions.
Extra comments about a function, such as what is its
purpose and how it is achieved, are welcome

@param c reference to input data object
@return whatever
@exception
@author Snoopy
@version 3.00 beta
@see DerivedClass */

int function( const DerivedClass& c ) ;
```

The programmer supplied fields @param, @return, @exception, @author and @version are part of our minimum function description and must be present (even when empty, like the @exception field above). In the previous and the following examples the information supplied by the user is written in bold. The other user supplied fields are optional.

```
/** A class. This is an example of how to document classes.
Extra comments about a class, such as what is its
purpose and how it is achieved, are welcome

@author Snoopy
@version 3.00 beta
@see ParentClass */

class ChildClass {
...
}

/** This is an example of how to document files.
Extra comments about a file, such as what it contains,
are welcome
```

```
@author Snoopy
@version 3.00 beta */
...
```

The minimum class and file descriptions include general comments plus the user supplied fields @author and @version.

The documentation must be generated in html and pdf and be made available at **`http://www.dig.cs.gc.cuny.edu/documentation/documentation.html`**. Since DOC++ does not generate pdf files directly, the programmer must use DOC++ to generate tex files that can be used to produce pdf files by executing programs like latex followed by dvipdf, or pdflatex.

For more information on how to use DOC++ consult the manual at **`http://www.dig.cs.gc.cuny.edu/manuals/manuals.html`**

5 Shared libraries

The libraries will be compiled as shared libraries (this is done to save disk space and to keep the programs updated). In practical terms this requires the following:

1. Compile with:
   `gcc -c -fPIC *.cpp -o *.o`
2. Link with:
   `gcc -shared -Wl -o $(LIBPROJ)`

In the command above, $(LIBPROJ) is a variable containing the names of the libraries to be linked with the program. The directories where the libraries are located will be added to the global profile that is loaded when the user starts a new shell.

6 CVS

CVS is a *version control system* that is used to keep track of the history of source files. In order to do that, CVS makes use of a centralized source file database called the *repository.*

For example, bugs sometimes are introduced when software is modified, and a bug may remain undetected for a long time after a modification. With CVS, an old version can be easily retrieved and the change that caused the bug can be tracked. CVS is also helpful when used by a group of developers working on the same project, because it controls changes in the source code and asks for confirmation when any attempt of writing outdated code to a project happens.

Our CVS server is located at `dig.cs.gc.cuny.edu` and the project name for this document is `Core`.To retrieve a project (in the examples that follows we will be working with a project called `Blue` and we assume that the repository is already set up), you should:

1. ask your favorite system manager to add you user name to the group cvs;

2. create a batch file like the following;
```
#!/bin/bash
export EDITOR=nedit
export CVS_RSH=ssh
export CVS="/usr/bin/cvs"
export HOST=dig.cs.gc.cuny.edu
export USER=roberto#NOTE: change the USER
export REMOTEDIR=/home/cvs/cvsroot
# shouldn't need to modify anything below this point
export CVSROOT=":ext:$USER@$HOST:$REMOTEDIR"
$CVS -d $CVSROOT $*
```

3. name it cvs.bat and make it executable, `chmod +x cvs.bat`

4. run the command:
```
cvs.bat checkout Blue
```

The sample session below shows how to use some other basic CVS commands.

6.1 Sample Session

This section describes a typical work-session using CVS.

The first thing you must do is to get your own working copy of the source for `'Blue'`. For this, you use the checkout command (this is made only once per session):

```
$ cvs.bat checkout Blue
```

This will create a new directory called `'Blue'` and fill it with the source files of the project `Blue`.

```
$ cd Blue
$ ls Blue
```

The result of the command `ls` depends on the project but `ls` should list the `'CVS'` directory that is used internally by CVS. Do not modify or remove any of the files in it.

After dowloading the project, you can start working on some file(s), for example, `'whatever.cpp'`.

6.1.1 Committing your changes

After checking that the files are still compilable you may submit a new version of `'whatever.cpp'` to the common repository. To do this the following command should be run:

```
$ cvs.bat commit whatever.cpp
```

As a result of this command, CVS starts an editor that will allow you to enter a log message. A comment, related to the bug(s) removed from the code, can be entered. After saving the temporary file, by exiting the editor, the message will be sent to the CVS repository with the updated code.

The environment variable $EDITOR determines which editor is started. If $EDITOR is not set, the editor defaults to `vi`. If you want to avoid the overhead of starting

an editor you can specify the log message on the command line using the '-m' flag instead, as below.

```
$ cvs.bat commit -m "fixed an awful bug" whatever.cpp
```

If two or more programers are working on the same file, when the second one tries to update the file, the CVS server will tell that the source file which was modified by that programer is outdated and any change should be made on the current source file.

6.1.2 Viewing differences

If a programmer does not know if 'whatever.cpp' was modified after the last checkout or submission, he/she should run the following commands.

```
$ cd CORE
$ cvs.bat diff whatever.cpp
```

This command runs diff to compare the version of 'whatever.cpp' in the repository with your working copy. Suppose that the only differences were comments that are present in the programmer´s working copy. Then, the programmer can commit the changes by running the command below.

```
$ cvs.bat commit -m "added some comments" whatever.cpp
Checking in whatever.cpp;
/usr/local/cvsroot/tc/whatever.cpp,v <-- whatever.cpp
new revision:  1.2; previous revision:  1.1
done
```

6.1.3 Adding new files

If you create a totally new file (for example 'whatnot.cpp') and want to submit it to the repository, type:

```
$ cvs.bat add whatnot.cpp
cvs server:  scheduling file 'whatnot.cpp' for addition
cvs server:  use 'cvs commit' to add this file permanently
$ cvs.bat commit whatnot.cpp
```

6.1.4 Other commands

The CVS manual (that can be found at **`http://www.dig.cs.gc.cuny.edu/manuals/manuals.html`**) contains a list of all commands with examples and explanations. You should read it, in particular the commands **status** and **update** that tell you if a file have been modified and how to incorporate changes made by other people in your working session.

7 Copyright and license

We need to incorporate a copyright notice. Replace '**Foobar**' by the name of the software package you are developing.

```
This file is part of Foobar.

Foobar is a free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published
by the Free Software Foundation; either version 2 of the License,
or (at your option) any later version.

Foobar is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with Foobar; if not, write to the Free Software Foundation,
Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

Appendix A

This appendix contains the Schema files that define the structure of our data files, as well as instances of them. The schema files give a full explanation of what a particular application file should contain, but schema files are not reader friendly. Therefore, please look at the instance files in order to get a better idea of how such application files should look like. For more information on XML and Schemas, see **http://www.w3.org/XML**.

- DIG.xsd `http://www.dig.cs.gc.cuny.edu/manuals/schemas/DIG.xsd` - a schema file describing the skeleton of a DIG data file.
- GENAPP.xsd `www.dig.cs.gc.cuny.edu/manuals/schemas/GENAPP.xsd` - a schema file describing the specifics of a general application data file. For a particular application, modify this file by removing or adding elements or attributes to the application header.
- RECFIL.xsd `www.dig.cs.gc.cuny.edu/manuals/schemas/RECFIL.xsd` - an application specific GENAPP.xsd file for SNARK03 recfils.
- RECFIL.exp `www.dig.cs.gc.cuny.edu/manuals/schemas/RECFIL.exp` - a commented SNARK03 recfil instance file. After removing comments, it can be validated against RECFIL.xsd (and DIG.xsd) and, since it is also a DIG data file, it can also be validated against GENAPP.xsd (and DIG.xsd).
- PRJFIL.xsd `www.dig.cs.gc.cuny.edu/manuals/schemas/PRJFIL.xsd` - an application specific GENAPP.xsd file for SNARK03 prjfils.
- PRJFIL.exp `www.dig.cs.gc.cuny.edu/manuals/schemas/PRJFIL.exp` - a commented SNARK03 prjfil instance file. After removing comments, it can be validated against PRJFIL.xsd (and DIG.xsd) and GENAPP.xsd (and DIG.xsd).