pyDM:
An interface between Python and Gatan Digital Micrograph

Robert A. McLeod

Scripting Interfaces with Digital Micrograph

- For TIA-class instruments, FEI provides TEMScripting.dll interface. This is build on Win32COM and allows interface (via TIA) with Digital Micrograph's detectors. Unfortunately it is very slow.
  - Overall, calls to create an Acquisition object in FEI's TEMscripting add 600 ms of overhead over-and-above what a Digital Micrograph *.s script requires.
  - Calling asSafeArray() to return the data to Python has about 600+2500 ms of overhead, which is really unacceptable.
  - A hack to bypass asSafeArray(), by saving asFile() to a TIFF on a RAMdisk load to Python with skimage has about 600+50 ms overhead.
- From my recollection, JEOL's JEM Toolbox for Matlab is also very slow.
- Hitachi's Maestro should be fast (Michael Bergen says 'milliseconds').
- Conclusion: we desire a fast interface that returns images acquired in DM to Python as a numpy array, quickly.

Project Overview

- **pyDM.pyd**: a Python .pyd package that acts as a client to a Digital Micrograph plugin, **pyDMPipeline.dll**.
  - Uses boost.python and boost.interprocess
  - boost.numpy added as a static library extension
- **TEM.py**: A pure python interface that encapsulates both pyDM and TEMScripting.
  - Provides classes for both a TEM simulator and FEI-TIA microscopes, and potentially can be extended to FEI and Hitachi COM interfaces.

Benchmarking USC Detector

- 2k x 2k pixel array
- Measure average acquisition time for 1 ms image x 30 times to average overhead costs above Digital Micrograph code.
- Most of Gatan Scripting time is CCD read-out, but there is evidently significant time required for image display (possibly from histogram?)
- Probably about 25 ms latency for inter-process communication.

Binning	x8	x4	x2	x1
Gatan Scripting	0.235 s	0.341 s	0.660 s	1.818 s
FEI COM with asSafeArray()	1.081 s	1.912 s	4.97 s	15.8 s
FEI COM with asFile() + RAMDISK	0.853 s	1.017 s	1.545 s	3.29 s
pyDM	0.270 s	0.380 s	0.686 s	1.665 s

Project Design Decisions

- Gatan requires use of Microsoft Visual C++ with MFC classes
- They also use some Boost libraries, so we need to import that too.
  - Boost is basically a giant library of C++ functions that may one day make it into the C++ standard.
- Because of these dependencies, I choose to use Boost for interprocess communication between the two DLLs and Boost.Python to generate the Python interface for the C++ library.
- Python interface DLL and Digital Micrograph plugin cannot be the same file
  - Each file that calls the DLL gets a separate instance of the DLL in memory.
  - We have to create a server/client type system to share data.

Compiler Notes

- GMS1.X (tested on 1.85) must be compiled with Visual C++ .Net 2003 Professional
  - Need GMS SDK3.8.2
  - Request it directly from Gatan
- GMS2.X (tested on 2.1.1) must be compiled with Visual C++ 2008 Professional.
  - Need GMS SDK2
  - Request it directly from Gatan

Project Requirements

- Need:
  - WinPython 2.7.6.4-32bit
  - Gatan SDK for GMS 2.X and/or for GMS 1.X
    - //Ruska/Tempo (or email Gatan)
  - Microsoft Visual Studio 2008 Professional
    - Difficult to buy
    - MSVC .NET 2003 for GMS 1.X
  - ATK headers for MFC (Microsoft Foundation Classes)
    - not included in MSVC2008 Express (the free one), get Professional version
    - Windows development kit 7.1 is for Win XP should include MFC but does not appear to.
  - Boost 1.5.2 for MSVC-9.0-32bit (and/or MSVC-7.1-32bit)
    - DMPluginUtility.h uses shared_static_cast which became deprecated with Boost 1.5.3
    - Must compile to use python27.dll, pre-built binaries found on Internet depend on python26.dll
  - Boost.NumPy
    - https://github.com/ndarray/Boost.NumPy
    - Build project with Cmake and then compile project in MSVC2008

MSVC 9.0 Project Setup

pyDM.pyd Project

- Add to C++/General/Additional Include Directories:
  - Local/boost_1_5_2
  - GitHub/Boost.NumPy
  - WinPython-32bit-2.7.6.4/python-2.7.6/include
  - WinPython-32bit-2.7.6.4/python-2.7.6/lib/site-packages/numpy/core/include
- Add to Linker/General/Additional Libraries
  - Local/boost_1_5_2/stage/lib
  - WinPython-32bit-2.7.6.4/python-2.7.6/libs
  - GitHub/Boost.NumPy/lib/Release (if you did not move the .lib by hand)
- Statically link
  - GitHub\Boost.NumPy\lib\Release\boost_numpy.lib

pyDMPipeline_GMS21.dll Project

- Add to C++/General/Additional Include Directories:
  - GMS2_SDK/include
  - Local/boost_1_5_2
- Add to Linker/General/Additional Libraries
  - GMS2_SDK/lib/Win32
  - Local/boost_1_5_2/stage/lib
- Statically link
  - DMPlugInBasic.lib
  - Foundation.lib

MSVC 7.1 Project Setup

- Can be painful because MSVC 7.1 is a bastardized development environment that does nothing well.
  - Only pyDMPipeline_GMS18.dll needs to be built, you can build pyDM.pyd in MSVC9.0
  - pyDM1 project file is included but I do not plan to support it.

pyDMPipeline_GMS18.dll

- Add to C++/General/Additional Include Directories:
  - GMS2_SDK/Win32/VisualStudio.Net/include
  - Local/boost_1_5_2
- Add to Linker/General/Additional Libraries
  - GMS1_SDK/Win32/VisualStudio.Net/lib
  - Local/boost_1_5_2/stage/lib
- Statically link
  - DMPlugInBasic_dll.lib

Boost Library

- Required for the plugin.
- Version 1.5.2 is the latest that will link to the Gatan SDK.
  - Because: SDK calls static_shared_cast
- Binaries of the library are available for MSVC9.0 but I didn't use them.

Building Boost

- Open Visual Studio 2008 Command prompt from Start Menu
  - Sets vcvarsall.bat and such so that you use MSVC9
- Chdir to \local\boost_1_52_0
  - Booststrap.bat
- Build linking libraries
  - .\b2 toolset=msvc-9
- Build .dlls
  - .\b2 toolset=msvc-9 link=shared threading=multi variant=release
- Puts files in local\boost_1_52_0\stage\lib
- Copy boost_python-vc90-mt-1_52.dll to the DM plugins folder (or put the Boost DLLs on the system path), because DM doesn't link it.
- http://www.codeproject.com/Articles/11597/Building-Boost-libraries-for-Visual-Studio

Boost.NumPy

- Unofficial extension for Boost.Python that includes Numpy arrays (for example, returning images).
- Best is to compile it into a static .lib and link that to pyDM, but it should act seamlessly like a part of boost.python
- See on GitHub
  - TO DO

Cmake on Boost.NumPy

- Using WinPython 2.7.6.4-32bit
- In Cmake-Gui:
  - Set PYTHON_EXECUTABLE=D:/WinPython-32bit-2.7.6.4/python-2.7.6/python.exe
  - Set BOOST_LIBRARYDIR=D:/local/boost_1_52_0
  - Make sure it points to local/boost_1_52_0 and not boost_1_55_0 or such

- Use the following Windows environment variables:
  - PATH = D:\WinPython-32bit-2.7.6.4\python-2.7.6;D:\WinPython-32bit-2.7.6.4\python-2.7.6\Scripts;
  - PYTHONPATH = D:\WinPython-34bit-2.7.6.4\python-2.7.6
  - PYTHONHOME = D:\WinPython-32bit-2.7.6.4\python-2.7.6\Lib;D:\GitHub\ram;D:\GitHub\pyDM\
- Register 32-bit install of WinPython as the registered version with the Python Control Panel.
- Remember that you have to reload Cmake-Gui if you change environment vars.

Is a PYD file the same as a DLL?

- Yes, .pyd files are dll's, but there are a few differences. If you have a DLL named spam.pyd, then it must have a function initspam(). You can then write Python "import spam", and Python will search for spam.pyd (as well as spam.py, spam.pyc) and if it finds it, will attempt to call initspam() to initialize it.
- Note that the search path for spam.pyd is PYTHONPATH, not the same as the path that Windows uses to search for spam.dll. Also, spam.pyd need not be present to run your program, whereas if you linked your program with a dll, the dll is required. Of course, spam.pyd is required if you want to say "import spam". In a DLL, linkage is declared in the source code with __declspec(dllexport). In a .pyd, linkage is defined in a list of available functions.
- CATEGORY: windows

Dependency Issues

- Download Dependency Walker to test if .dll files are missing something if you get weird library import errors.
- Typically must copy boost_python-vc90-mt-1_52.dll into PYTHONPATH (same directory as pyDM.pyd is easiest).
- There is some nonsense related to Vista with the DLL's I have compiled but no one sane uses that OS so probably not a problem.

Importing pyDM.pyd into Python

- Call import pyDM But the pyDM file should be found on the PYTHONPATH
- Connect to TEM
  - pyDM.connect()

and use like an other Python library

- AcquireImage can be modal

  AcquireImage(…, block=True)
- Or non-modal

  AcquireImage(…, block=False)

  … wait or process code …

  getImage(…)

Non-modal is easier to use, but for maximum performance you can do calculations while Digital Micrograph is acquiring the image. Useful for series measurements.

Loading Plugin

- Ensure Python is not running when you start Digital Micrograph.
  - pyDMPlugin is loaded at runtime.
  - If it cannot claim shared memory space it may push an error (but it should not crash).
- Otherwise I have not had troubles crashing DM.
- The pyDM library may have trouble finding the shared memory regions if Digital Micrograph has been running for a long time (many hours) before you pyDM.connect()

IMPLEMENTATION NOTES

Multi-threading Plugin

- pyDMPipeline needs its own thread to poll the message_queue. Otherwise it locks Digital Micrograph.
- For compilation reasons I chose to use the windows.h native CriticalSection. It is not object oriented, but works in both MSVC2008 and MSVC2003 for both versions of GMS.

Boost.Interprocess Notes

- Using mapped_region for transferring images (and spectra)
  - Memcpy() can convert the data from a DM::Image to a Boost::Python:NDArray without any ravel/interlacing issues.
- Using 2 message_queue() objects to send commands to and from each DLL.

GMS 2.1 IMPLEMENTATION

GMS 2.1 with SDK2

- A lot has changed with the new SDK2 and unfortunately it is not documented (although fortunately Gatan is using verbose naming conventions).
- Doing a dumpbin.exe –headers of the .lib files is helpful.

- I have not messed with the low-level parameters as without documentation or a scripting interface understanding them would be quite painful.

Print Functions

- Want to be able to get and print to GMS results windows various stuff.
- HighLevelReadoutParameters in GatanCameraTypes.h is interesting.
- GatanImaging.h is also important?

GMS2 Speedup Strategy

- Since the CCD accumulates dark count while it is idle, it has to be cleared between exposures. When calling AcquireImage, it seems like the CCD may be cleared both before and after. So a continuous mode for series would be overall better.
- The scripting may be in-between us and the Gatan C++ guts.

Camera Mode Notes

- US1000.1 is the upper camera
- US1000FTXP is the GIF quantum camera
  - Testing with unprocessed:
  - EF-CCD and EF-Cinema are essentially different cameras
  - Readmode=0, qualitylevel=0, t_out=0.615 s
  - Readmode = 0, qualitylevel = 1, t_out = 1.63 s
  - Readmode=0, qualitylevel=2, t_out = 1.63 s but output is really strange/flat
  - Readmode=1 is cinema mode, t_out = 0.598 s
  - Looks like background is 256 in the low-quality mode?
  - Readmode=2 is some narrow strip-readout on top half of CCD (obviously for EELS). But it's slow, at 0.9155 s per frame
  - Readmode=3 is two strips, top and middle, perhaps the DualView mode? t_out = 0.124 s
    - Potentially it is only shifting half a frame per frame.
  - Readmode=4 is like 3 but the top strip is wider, t_out = 1.23
  - Readmode=5 is fast narrow strip, t_out = 0.493 s
  - Repeats double-style 6-7 as above but faster
  - So the double style 6 or 7 with two spectra per frame would be fastest overall?
  - 8 is standard full-frame
  - 9 hangs Digital Micrograph (and the Gatan Instrumentation Bin), so let's not go there.

Readmode 5: Conventional EELS, Sub-shift…

- Frame is from 764:1283
- How can I tell if it is doing software or hardware binning?
- [Quantum shows binning numbers of 1,2,5,10,13,26,65,130]
- Binning on the USFTXP is really strange compared to the vanilla US1000, as it doesn't seem to hardware bin no matter what settings I use.
- Cinema mode seems slower than full-frame…
- Testing it by hand it is blindingly fast when binned full-frame.

Upper US1000.1

- Reads out in 0.195 s per 256 x 256 frame
- 0.3 s per 512 x 512 frame
- 1.53 s per 2048 x 2048 frame
- Only the one quality mode.

DMPluginCamera.h

SetStandardParameters(…)

Same as what I used before

For David, there is SetBinnedReadArea and such which is probably quite important for high-speed readout.

Inline void GetReadMode

Questions for Gatan

- Why does US1000FTXP always software bin when called from SDK? The US1000.1 seems to work as expected.
- Is there a newer SDK? (Ours compiled March 2011).
- Documentation and/or examples for Gatan::Camera::AcquisitionImageSource and or DataSlice?
- Is Gatan::Camera::AcquisitionImageSource intended to be a non-modal version of Gatan::Camera::AcquireImage? Is there another non-modal way to acquire images or spectra?
- What is Imaging::I_LockedImageDataSequence and how to create it? Is it supposed to be mutex locked?
- How to use the ReadModes? Note that ReadMode=9 hard crashed the Gatan Camera Controller hardware, which really the SDK should not be able to do.

FIN

Backup of Python 64-bit Environment Vars

- PATH = %SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;C:\WinPython-64bit-2.7.6.4\python-2.7.6.amd64;C:\WinPython-64bit-2.7.6.4\python-2.7.6.amd64\Scripts;%systemroot%\System32\WindowsPowerShell\v1.0\;D:\local\fftw-3.2.2-dll64;D:\Microsoft Visual Studio 12.0\VC\bin;D:\Multislice\Gromacs\bin;C:\Program Files (x86)\CMake 2.8\bin;C:\Program Files\HDF_Group\HDF-JAVA\2.10.0\bin;D:\Multislice\MATLAB Compiler Runtime\v716\runtime\win64;C:\util\ffmpeg\bin

- PYTHONPATH = C:\WinPython-64bit-2.7.6.4\python-2.7.6.amd64

- PYTHONHOME = C:\WinPython-64bit-2.7.6.4\python-2.7.6.amd64\Lib;D:\GitHub\pyVincent;D:\GitHub\plottools;D:\GitHub\ram;D:\GitHub\