HOW TO BUILD, RUN and VISUALIZE BVXM

Background

See this site: http://vxl.sourceforge.net/releases/install-release.html

Windows Build

To build the VXL/bvxm software:

1. Download and install the following software:
    1. Visual Studio C++ - http://www.microsoft.com/express/vc/
    2. CMake - http://www.cmake.org/HTML/index.html
    3. Python - http://www.python.org - download the source code, *not* the Windows installer
    4. TortoiseSVN - http://tortoisesvn.net/downloads
    5. (Optional) Expat XML Parser - http://sourceforge.net/projects/expat/ - Installed in folder C:/Program Files/Expat 2.0.1.
2. Directions for installing Python (from the source you downloaded):
    1. Note: The Debug version of Python is required, which does not come with the Windows.
    2. Navigate to folder C:/Python-2.5.2/PCbuild8.
    3. Double-click pcbuild.sln to launch Visual Studio.
    4. Select "Debug" from the pulldown on the icon bar.
    5. Select View->Solution Explorer.
    6. Select the top item (all projects) and right-click->Build.
    7. Note: Not all projects will build (15 succeeded and 5 failed). Ignore fails.
    8. The end result is the creation of python_d.exe (and associated dll's) in folder Win32Debug.
    9. (Optional) Repeat mutatis mutandi to create the "Release" (instead of "Debug") build.
        1. Creates python.exe in folder Win32Release.
3. To download the VXL/Brown software:
    1. Create an empty folder for the code (e.g., C:\VXL\sandbox).
    2. Checkout the code from sourceforge as follows:
        1. Right-click this folder (sandbox) and select "SVN Checkout" from the menu.

2. Enter the repository :ext:username@vxl.cvs.sourceforge.net:/cvsroot/vxl
3. Enter the checkout directory: C:\VXL\sandbox
4. Press OK.
5. Note: Public VXL repository is http://vxl.sourceforge.net/#download
6. Navigate into the C:\VXL\sandbox\vxl folder and create a "bin" folder.

3. If moving this software to another machine, zip it up.

4. Define Environment variables
    1. Right-click "My Computer" icon on your desktop and select Properties.
    2. Select "Advanced" tab at top and then "environment Variables" at bottom.
    3. Add a user variable called PYTHONPATH and edit it to something like the following paths:
        1. .\;C:\Python-2.5.2\;C:\Python-2.5.2\PCbuild8;C:\Python-2.5.2\PCbuild8\Win32Release;C:\Python-2.5.2\PCbuild8\Win32Debug;C:\VXL\sandbox\build\lib\Debug;C:\VXL\sandbox\build\lib\Release;C:\VXL\sandbox\vxl\contrib\brl\bseg\bvxm_batch
        2. .\;C:\Python-2.5.2\;C:\Python-2.5.2\PCbuild8;C:\Python-2.5.2\PCbuild8\Win32Release;C:\Python-2.5.2\PCbuild8\Win32Debug;D:\ben\ChangeDetection\code\midterm\build\lib\Debug;D:\ben\ChangeDetection\code\midterm\build\lib\Release;C:\Program Files\Expat 2.0.1\Bin
    4. Edit the system variable PATH and add the following paths to the end:
        1. C:\Python-2.5.2\PCbuild8\Win32Debug;C:\Python-2.5.2\PCbuild8\Win32Release;C:\VXL\sandbox\build\lib\Debug;C:\VXL\sandbox\build\lib\Release
        2. C:\Python-2.5.2\PCbuild8\Win32Debug;C:\Python-2.5.2\PCbuild8\Win32Release;D:\ben\ChangeDetection\code\midterm\build\lib\Debug;D:\ben\ChangeDetection\code\midterm\build\lib\Release

5. To build the VXL/Brown software:
    1. Run CMakeSetup.exe:
        1. Enter the source code path: C:\VXL\sandbox\vxl
        2. Enter the binary path: C:\VXL\sandbox\build (it is ideal to not build in the source tree; may avoid some strange linker errors we were seeing)
        3. Check the "Show Advanced values" checkbox on the right.
        4. Press Configure and let run.

1. Choose Visual Studio 9 2008 (or your version of VS)

5. You will get an error message about certain variables not being set. Hit OK and you will see a list of all variables. Enter the paths for the following variables in the list:
    1. PYTHON_INCLUDE_PATH C:/Python-2.5.2/include
    2. PYTHON_DEBUG_LIBRARY C:/Python-2.5.2/PCbuild8/Win32Debug/python25_d.lib
    3. PYTHON_LIBRARY C:/Python-2.5.2/PCbuild8/Win32Release/python25.lib
    4. PYTHON_PC_INCLUDE_PATH C:/Python-2.5.2/PC
    5. EXECUTABLE_OUTPUT_PATH to something like C:/VXL/sandbox/build/bin (usually, this is the same as LIBRARY_OUTPUT_PATH, only with a 'bin' on the end instead of a 'lib.')
    6. BUILD_TESTING to ON (for the self/unit tests)
    7. VXL_USE_LFS to ON (if you need very large file support; Under LINUX, large file support is automatic. However Windows, special file pointer functions have to be used that are Windows-specific. The VXL programmers have created a kludge around this by invoking a wrapper file pointer class that works under all operating systems. VXL_USE_LFS invokes this wrapper.)
    8. (optional) BUILD_VGUI to ON for some GUI tools (may require other options turned on, so might not build completely)
    9. (optional) EXPAT_LIBRARY C:/Program Files/Expat 2.0.1/Bin/libexpat.lib
    10. (optional) EXPAT_INCLUDE_DIR C:/Program Files/Expat 2.0.1/Source/lib
    11. Note that VXL does not support the `BUILD_SHARED_LIBRARY=ON feature` on Windows (but does on other platforms). See http://vxl.sourceforge.net/vxl-users-faq.html
6. Press Configure again and let it run.
7. Press OK and it will run and then disappear when done.

2. Run Visual Studio on "vxl.sln" in the bin folder (double-click it) to compile and build.
    1. Select "Debug" from the pulldown on the icon bar.
    2. Right-click ALL_BUILD on the left and select "Build".
    3. Watch the status on the lower left. When it's finished, it will say Build complete.
        1. Check the log messages (at the bottom) for errors and status. You should get "0 failed". If not search for the term "fatal error" in the build output to see

what the problem was. If there are problems, you may need to do a "clean build", esp. if you need to re-run CMAKE, because otherwise you may get errors about linkage errors with .lib files in build/lib. Ex:

========== Build: 493 succeeded, 0 failed, 1 up-to-date, 0 skipped ==========

2. When successful, check folder C:/VXL/sandbox/build/lib/Debug/ for bvxm_batch_d.lib.

4. (Optional) Repeat mutatis mutandi to create the "Release" (instead of "Debug") build.
    1. Check folder C:/VXL/sandbox/build/lib/Release/ for bvxm_batch.lib.

6. To test the Brown scripts:
    1. Run the self-tests / unit tests:
        1. *Your build should put a whole bunch of test programs in your EXECUTABLE_OUTPUT_PATH. One of them will be called "bvxm_test_all." That should run standalone from the command line, and report that all tests passed at the end of a run. Another useful test is "bvxm_pro_test_all" (to run this you must copy non-CXX files from vxl\contrib\brl\bseg\bvxm\pro\tests to the directory location of the bvxm_pro_test_all.exe file).*
    2. Start Python interpreter as follows:
        1. Bring up DOS command line window and navigate to C:\VXL\sandbox\vxl\contrib\brl\bseg\bvxm_batch.
        2. Type "python_d" to start the Debug version of Python.
            1. If it can't find python_d.exe, check folder C:/Python-2.5.2/PCbuild8/Win32Debug.
            2. if the file is there, check your environment variables (see Step 3 above).
        3. Type "import bvxm_batch" at the Python interpreter prompt and verify that Python can find this library module.
            1. If it can't, check folder C:/VXL/sandbox/vxl/bin/lib/Debug/ for bvxm_batch_d.lib.
            2. if the file is there, check your environment variables (see Step 3 above).
        4. Type exec file("file.py") to run Python script file "file.py":
            1. Replace "file.py" with the name of one of the Python scripts in this folder.

    2. I think you could also type "python file.py".
5. If you run the very long change_detection.py script, you may want to redirect stdout/stderr such as:
    1. `python change_detection.py > winlog.txt 2>&1`
6. If you run into problems with some of the scripts, make sure there is no trailing slash in the pathname of input_directory in the file world_model_params.xml (also you should make sure this directory actually exists); e.g.:

```
<input_directory type="string"
value="C:/VXL/Project/1/Temp" />
```

To debug in Visual Studio

1. In Visual Studio, make sure "Debug" is selected in the chocie list. Set any desired break points.
2. At a command prompt, go to the directory of your python script and execute python_d with no arguments to enter interactive mode
3. In Visual Studio, select Debug -> Attach to Process... and pick the python_d process.
4. Now go back to the command prompt, and execute your desired python script e.g. "execfile('change_detection.py')"
5. The debugger should stop at your break point. If not, it could be that certain parts of the source code may need to be re-bult/re-linked. Try re-building just the relevant projects first, then try re-building bvxm_batch only, and if still no luck, do a build on BUILD_ALL, and as a last resort, re-build on the entire BUILD_ALL.

Linux Build

The following was tried on Fedora Core 6.
Install the following packages:

- make (GNU Make 3.81 works)
- cmake-2.4.6 (this contains ccmake)
- gcc version is 4.1.2-13
- Python 2.4.4-1.fc6 from RPM / YUM (2.5-12.fc7 RPM should also work; note also source code is not necessary and may result in errors)
- Eclipse with Subclipse (in order to retrieve the code from subversion; there are other options for obtaining the source code as well)
- Note expat should not be necessary

The options are pretty much the same as for a windows build, except for the following:

1. Don’t build in your source tree. Set up a separate directory for each build (I sort by OS and chip architecture: ../linux/x86_64, .../solaris/i86pc, etc). Supposed you call this directory $VXLBUILD and the source directory $VXLSRC. The first time you run ccmake for this particular build, cd to $VXLBUILD and run “ccmake $VXLSRC”. After the first configuration, you can subsequently run ccmake just in $VXLBUILD by cd’ing to it and running “ccmake .”
2. Configuring with ccmake is iterative. You change a few options and press the ‘c’ key. This might lead ccmake to ask you about new options that were not relevant before the changes. You will know that you can generate Makefiles and exit ccmake when ccmake shows ‘g’ (for ‘generate’) as an option at the bottom of the screen.
3. For the VXL build, ccmake tries to set an include directory for option OPENGL_xmesa_INCLUDE_DIR. This causes the build to fail on Linux. When your are in ccmake, press ‘t’ to toggle into expert mode, and use the page and arrow keys to navigate down to that option ... they are in alphabetical order. When you get the cursor over the option, press ‘Enter’ to edit it, use the backspace/delete keys to blank the field out, and press ‘Enter’ again to accept the changes.
4. Try turning BUILD_VGUI on. After your next ‘c’ keypress you should see options that start with VGUI_USE, such as VGUI_USE_GTK2. If you don’t get one of those, you won’t be able to build any of Brown’s GUI tools until you install a library that VGUI supports. Supported libraries include: GTK2, GLUT, QT, and WXWINDOWS, although I’ve never been able to get VGUI to work with anything but GTK2 on Linux or the Microsoft Foundation Classes (MSFC) on Windows. **If VGUI won’t build**, it’s not a showstopper. You can still build and use voxel models without it. VGUI is just in the ‘nice to have’ category. Thus, if this option doesn't seem to work, you can simply turn BUILD_VGUI off.
5. Unless you have root permissions, you may need to change the CMAKE_INSTALL_PREFIX. This is where stuff gets installed if you type “make install” after the build. Under that directory, the Makefiles will create bin, lib, include and share if they don’t exist (and if you have write permissions). Again, with write permissions, “make install” will install what it thinks it needs to. You can build everything else locally as user, then switch to root to run “make install” if you want to install in a system-wide directory (/usr/local is the default).
6. I’ve been enabling BUILD_SHARED_LIBS. If you disable this, you’ll get lib*.a files rather than lib*.so files ... Mostly. There are some libraries that are hard-coded to be built as shared, including bvxm_batch.so. If you don’t enabled shared library building on an x86_64 box, you might end up with compile errors related to relocation of code. If this is the case, you will have to add ‘-fPIC’ to your CMAKE_C_FLAGS and CMAKE_CXX_FLAGS in ccmake. **Executive summary:** enable

BUILD_SHARED_LIBS. You might have to add wherever you install the libs to your LD_LIBRARY_PATH (if it's not somewhere 'visible' like /usr/local), but it's worth avoiding the –fPIC aggravation, in my opinion.

7. Enable BUILD_TESTING, so you can get their self-test executable, called bvxm_test_all or some such.
8. I noticed that earlier versions of VXL would not build *any* executables if you don't explicitly specify EXECUTABLE_OUTPUT_PATH. I just set it to $VXLBUILD/bin. You will notice that LIBRARY_OUTPUT_PATH is already prefilled with $VXLBUILD/lib, which makes it easy to copy, paste, and edit.
9. set PYTHON_INCLUDE_PATH to /usr/include/python2.4
10. set PYTHON_LIBRARY to /usr/lib64/python2.4/config/libpython2.4.a
11. (Optional) Set CMAKE_BUILD_TYPE to 'Release' or 'Debug'. Can also be left blank (not sure what the default is...).
12. When you are done hit 'c' and then 'g' to generate the files and the program will exit.
13. Type "make" in $VXLBUILD. It should run to completion with no errors.
14. The build will produce a file called libbvxm_batch.pyd in $VXLBUILD/lib. In order for the python interpreter to recognize that library for importing, rename it to bvxm_batch.so.

    When building python libraries in C, the name of the library needs to be named the same as the C init function. In this case, in bvxm_batch.cxx there is a function called initbvxm_batch, which sets up the python library, so the library must be called bvxm_batch. Why it builds with 'lib' in front of the name, I do not know. As for the change in extension, I wasn't able to load the library as a 'pyd' file. My understanding is that python should be able to load a 'pyd' file the same as 'so'. I searched a little to find out why I was having that trouble, but didn't find any results on the topic.

15. Modify PYTHONPATH such that it includes the directory where bvxm_batch.so is located, such as: PYTHONPATH=.:$VXLBUILD/lib
16. When running the pythons scripts, ensure the input_directory value in the file world_model_params.xml does not have a trailing slash (and also that the directory exists).
17. Modify the full_hiafa_images.txt and full_hiafa_cams.txt file for the proper image paths. In particular, make sure you change the new line style from CRLF to LF in these files.
18. If you run the very long change_detection.py script (it took ~7 hrs on a dual 3.4 GHz x86_64 fedora core 6 box), you may want to

pipe stdout/stderr to a file, such as "python change_detection.py >& testout.txt"

-fPIC errors

```
If you get errors like this:
 [ 95%] Building CXX object
contrib/brl/bseg/bvxm_batch/CMakeFiles/bvxm_batch.dir/batch_bvxm.o
 [ 95%] Building CXX object
contrib/brl/bseg/bvxm_batch/CMakeFiles/bvxm_batch.dir/reg_bvxm.o
 Linking CXX shared library ../../../../lib/libbvxm_batch.pyd
 /usr/bin/ld: /opt/Python-2.5.2/libpython2.5.a(object.o): relocation
R_X86_64_32S against `_Py_TrueStruct' can not
 be used when making a shared object; recompile with -fPIC
 /opt/Python-2.5.2/libpython2.5.a: could not read symbols: Bad value
 collect2: ld returned 1 exit status
 make[2]: *** [lib/libbvxm_batch.pyd] Error 1
 make[1]: *** [contrib/brl/bseg/bvxm_batch/CMakeFiles/bvxm_batch.dir/all]
Error 2
 make: *** [all] Error 2
```

The solution to this is to:

- Enable BUILD_SHARED_LIBS are described above.
- Make sure you are using the proper RPM version of Python.

This link says that "On certain architectures (AMD64 amongst them), shared libraries must be "PIC-enabled"."

- http://www.gentoo.org/proj/en/base/amd64/howtos/index.xml?part=1&chap=3

pyconfig.h errors

If you get errors like the following,

```
 [ 95%] Built target bvxm_pro
 Scanning dependencies of target bvxm_batch
 [ 95%] Building CXX object
contrib/brl/bseg/bvxm_batch/CMakeFiles/bvxm_batch.dir/batch_bvxm.o
 In file included from
/opt/igx/midterm/vxl/contrib/brl/bseg/bvxm_batch/batch_bvxm.h:14,
                 from
/opt/igx/midterm/vxl/contrib/brl/bseg/bvxm_batch/batch_bvxm.cxx:1:
 /opt/Python-2.5.2/Include/Python.h:8:22: error: pyconfig.h: No such file or
directory
 In file included from /opt/Python-2.5.2/Include/Python.h:57,
                 from
/opt/igx/midterm/vxl/contrib/brl/bseg/bvxm_batch/batch_bvxm.h:14,
```

```
                 from
/opt/igx/midterm/vxl/contrib/brl/bseg/bvxm_batch/batch_bvxm.cxx:1:
 /opt/Python-2.5.2/Include/pyport.h:761:2: error: #error "LONG_BIT definition
appears wrong for platform (bad   gcc/glibc config?)."
 make[2]: ***
[contrib/brl/bseg/bvxm_batch/CMakeFiles/bvxm_batch.dir/batch_bvxm.o] Error 1
 make[1]: *** [contrib/brl/bseg/bvxm_batch/CMakeFiles/bvxm_batch.dir/all]
Error 2
 make: *** [all] Error 2
```

It could be because you are using a custom python install instead of the RPM package.

First try the RPM python, and then if that doesn't work, try copying the pyconfig.h file from PYTHON_HOME to PYTHONE_HOME/Include.

Drishti Volume Rendering

The following was successfully performed on Windows XP SP2, AMD Opteron dual-core 2 GHz, 3.25 GB RAM:

1. **Install Drishti from sourceforge**
2. **Launch Drishti**
3. **The first time you launch drishti, it will ask you for the desired texture memory size. I chose 2048 MB. You should read the "Tips" section of the Drishti help as it explains how to determine what the proper values are as well as many other tips that will help drishti run better.**
4. **File -> Load Volume -> Load Raw Volume**
5. **Click the "Raw" button and choose the desired raw file output from vxl (i.e. world29.raw).**
6. **Choose "unsigned byte". The Grid Size should have been automatically chosen for you when you selected the raw file (to something like "500 500 100"). Leave everything else as the default values (voxel unit: millimeter, voxel size: 1 1 1, smoothing off, skip header bytes: 0, leave the metadata box as is with its default text).**
7. **Type in a path to a file named something like yourFile.pvl.nc in the "Pvl File" path (or click the PVL button to launch a browser).**
8. **Hit Process.**
9. **You will get a display that states "Remap original RAW data to 8 bit data". Hit 's' to confirm.**
10. **Wait a couple minutes while it processes everything.**

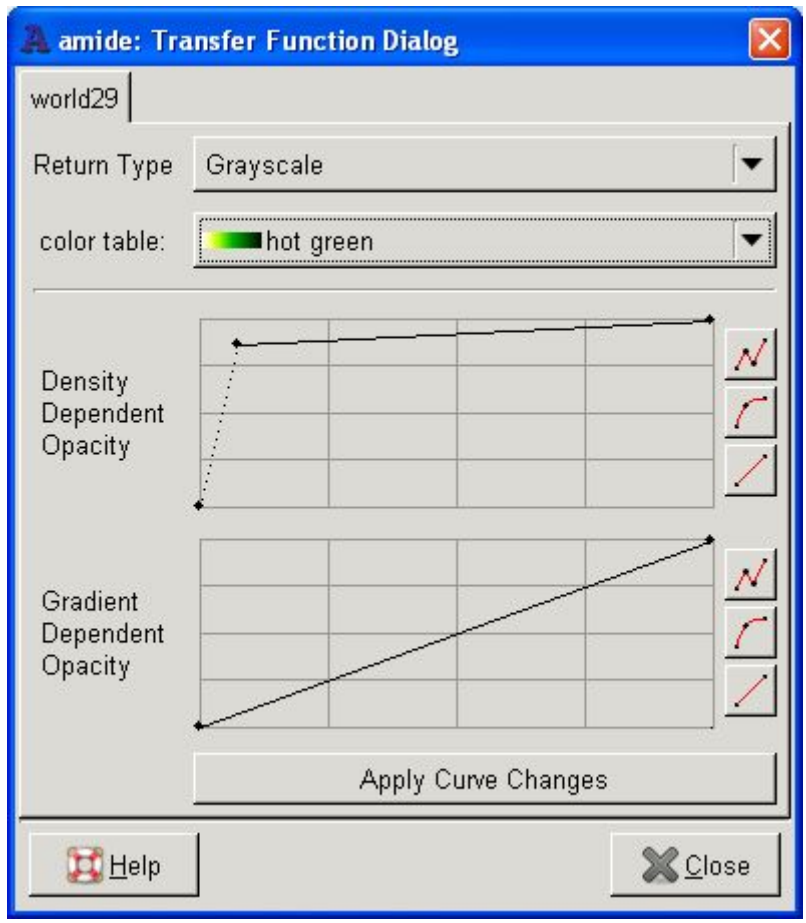
11. **You should get a volume displayed as well as a "transfer function editor" dialog box. You can immediately start to drag the volume around but it may not look like much until you define a good transfer function. You can either specify a New Transfer Function and then play around with the settings until the display looks good. Or you can load a previously saved transfer function that somebody else created.**
12. **Once you have a volume that looks good, the processed file has been saved to the .pvl.nc file that you specified. So, the next time you want to look at it, you don't have to re-process the RAW file. Instead, use File -> Load Volume -> Load Processed Volume and pick the .pvl.nc file. Then load the desired transfer function that you or somebody else saved off and voila, you've got your volume back.**

Amide Volume Rendering

The following was successfully performed on Windows XP SP2, AMD Opteron dual-core 2 GHz, 3.25 GB RAM:

1. **First launch the raw file in Drishti to determine what the correct grid dimensions (# voxels) are.**
2. **Download, install and launch Amide.**
3. **Import File (specify) -> Raw File**
4. **Pick the raw file (e.g. world29.raw)**
5. **Modality: other**
6. **data format: Unsigned Byte (8 bit)**
7. **read offset (bytes): 13**
8. **dimensions (# voxels): enter the values that Drishti gave you (500 500 100), but switch the values for x and z, so instead enter x, y, z, gates, frames as 100,500,500,1,1 (after entering this you should see the "total bytes to read through" increase to e.g. 25000013)**
9. **voxel size (mm): 1,1,1**
10. **scale factor 1.000**
11. **Hit OK**
12. **Now check the box next to the second layer that says e.g. "world29"**
13. **View -> Volume Rendering**
14. **Make sure the second layer that says e.g. "world29" is selected. Make sure Accelerate Rendering is selected if you have enough RAM to support it. Hit OK.**

15. **Wait a couple minutes.**
16. **Hit the green chart button in the top left ("Opacity and density transfer functions"). Drag the Density Dependent Opacity so that it looks something like:**



17. **The view that you get is likely not rotated in the ideal fashion. Take the axis buttons and drag them so the view is oriented as desired.**
18. **File -> Create Movie and choose the desired rotation settings.**

- Created (May 01, 2008) Leichtman, Andrea (Lockheed Martin)
- Updated (Jun 04, 2008) Garrett, Benjamin D (Lockheed Martin)
- Updated (Jun 6, 2008) - Tunali, Gamze (Brown University)