PCL Development Build on Apple Mac OS X
5 September 2012
by Ken Spratlin

Introduction

This document describes the steps to build PCL and its dependencies on Mac OS X.

My primary goal was to configure a development environment on Mac OS X that would support evaluation of the KinFu and KinFu Large Scale applications in PCL. Therefore, to date, little attention has been paid to other portions of PCL that are not required by KinFu and KinFu Large Scale.

An additional goal was to understand the state of PCL with respect to building entirely from source code on Mac OS X.

MacPorts, Homebrew, and other alternatives WERE NOT USED.

Only the Apple compilers (c++ and g++) and the additional Development Tools installed as described below were used.

I have only listed those versions of PCL and its dependencies that I used - other versions may work as well. I generally selected the most recent versions of each dependency available at the time (August 2012) - newer versions may now be available.

You may also want to check out the SuperBuild for PCL before attempting to follow this process - see http://www.pcl-developers.org/A-SuperBuild-for-PCL-td5706899.html.

Computer Configuration

MacBook Pro Retina (2012)
2.6 GHz Intel Core i7
16 GB 1600 MHz DDR2 memory
500 GB SSD
Intel HD Graphics 4000 (integrated graphics)
NVIDIA GeForce GT 650M 1024 MB (discrete graphics)
Mac OS X 10.8.1 (Mountain Lion)

This installation was performed on top of a clean install of Mountain Lion, updated from 10.8 to 10.8.1.

Mac OS X / iOS Development Tools

This section provides information about Apple's Xcode and the compilers that accompany Xcode.

Multiple versions of Xcode can be installed at the same time.

> The tool xcode-select is used to specify which installation of Xcode is used, and may be overridden by the DEVELOPER_DIR environment variable.

The compilers and other development tools (e.g. cvs, git, svn, c++, g++, make, otool, install_name_tool) are contained within the Xcode application and may be invoked on the command line directly via the xcrun command.

> xcrun provides a means to locate or invoke coexistence- and platform-aware developer tools from the command-line, without requiring users to modify makefiles or otherwise take inconvenient measures to support multiple Xcode tool chains.

Type "man xcode-select" and "man xcrun" at the command line for more information.

The development tools may also be installed separately from Xcode by installing the Command Line Tools available in the downloads section of Apple's Developer site. Note that if both Xcode and Command Line Tools are installed, there are then two copies of each tool installed on the computer per version.

The compilers and other development tools are located inside Xcode in the directory /Applications/Xcode/Contents/Developer/usr/bin. After installation of the Command Line Tools, these same compilers and development tools are also located in /usr/bin.

You may install the Command Line Tools without installing Xcode.

Unless otherwise noted explicitly, the compiler (c++, g++, nvcc) detected and selected by the CMake script for each package was used in each build phase listed here.

Below is listed information about recent Xcode versions. “--->” indicates a symbolic link in the /usr/bin directory.

Xcode 4.0
Default compiler is gcc4.2

Xcode 4.1
Default compiler is llvm-gcc-4.2

Xcode 4.2
Default compiler is Apple LLVM compiler 3.0

Xcode 4.3.3 (release version for Mac OS X 10.7 Lion)
Default compiler is Apple LLVM compiler 3.1
The default Xcode settings for language standard are -std=gnu99 and -std=gnu++98

c++ ---> clang++
cc ---> clang
clang
clang++ ---> clang
g++ ---> llvm-g++-4.2
gcc ---> llvm-gcc-4.2

Xcode 4.4 (release version for Mac OS X 10.8 Mountain Lion)
Default compiler is Apple LLVM compiler 4.0
The default Xcode settings for language standard are -std=gnu99 and -std=gnu++11

c++ ---> clang++
cc ---> clang
clang
clang++ ---> clang
g++ ---> llvm-g++-4.2
gcc ---> llvm-gcc-4.2

Xcode 4.5 (developer preview version supporting iOS 6)
Default compiler is Apple LLVM compiler 4.0
The default Xcode settings for language standard are -std=gnu99 and -std=gnu++11

c++ ---> clang++
cc ---> clang
clang
clang++ ---> clang
g++ ---> llvm-g++-4.2
gcc ---> llvm-gcc-4.2

Mac OS X Developer Tools Installation

Version numbers and miscellaneous comments are usually indicated below as {#}.

Install Xcode (from the Apple App Store) and/or Command Line Tools (from the Apple Developer Site).

Set the default Xcode version, for example:

```
sudo xcode-select -switch /Applications/Xcode.app
```

or

```
sudo xcode-select -switch /Applications/Xcode45-DP1.app {developer preview 4.5}
```

mkdir ~/dev {my development directory}

All downloads and builds were performed in ~/dev except where explicitly noted.

The following additional development tools were needed for this installation:

CMake Prebuilt into /Applications {2.8.8}

From www.cmake.org, download and install CMake {2.8.8} for Platform Mac OS X 64/32-bit Universal (for Intel, Snow Leopard/10.6 or later) dmg.

Modify environment variable as follows:

```
PATH=/Applications/CMake\ 2.8-8.app/Contents/bin:$PATH
export PATH=$PATH
```

pkg-config from Source into /usr/local {0.25}

From pgkconfig/freedesktop.org/releases/, download pkg-config {0.25} Source. {0.26 and 0.27 would not configure, so I used 0.25}. Follow the build and install instructions in the INSTALL file, which are:

```
cd ~/dev/pkg-config-0.25
./configure
make
sudo make install
```

hg Prebuilt into /usr/local {2.3.3}

From mercurial.selenic.com, download and install mercurial hg command {2.2.3} from Mercurial mpkg.

javac Prebuilt into /usr/bin {version not noted}

During an attempt to build the OpenNI/Sensor dependency from source, Mac OS X prompted to install the Java for OS X 2012-004...Runtime in order to install the Java compiler. I accepted this request. Attempts to build OpenNI/Sensor from source have been deferred for now.

PCL Dependencies Installation

libusb from Source into /usr/local {1.0.9}

From http://sourceforge.net/projects/libusb/files/libusb-1.0/, download libusb {1.0.9} Source. Follow the build and install instructions in the INSTALL file, which are:

```
cd ~/dev/libusb-1.0.9
./configure
make
sudo make install
```

Boost from Source into /usr/local {1.50.0}

From www.boost.org, download boost_1_xx_0.tar.bz2 {1_50_0 here} Source. Follow the build and install instructions at http://www.boost.org/doc/libs/1_51_0/more/getting_started/unix-variants.html, which are summarized here:

```
cd /usr/local
sudo tar --bzip2 -xf /path/to/boost_1_xx_0.tar.bz2
Can now delete the .bz2 file.
cd boost_1_xx_0
sudo ./bootstrap.sh --help {check defaults and modify as needed;
    no changes were made in this installation}
sudo ./bootstrap.sh
To enable MPI support, add "using mpi ;" to user-config-jam in tools/build/v2
    {note the required space between the "i" and the ";"}
sudo ./b2 install
```

Boost outputs the following information:

```
The following directory should be added to the compiler include paths:
    /usr/local/boost
The following directory should be added to the linker library paths:
    /usr/local/lib

>>> MPI auto-detection failed: unknown wrapper compiler mpic++
You will need to manually configure MPI support. <<<
```

I have not resolved this MPI issue yet.

eigen3 from Source into /usr/local {3.0.5}

From eigen.tuxfamily.org, download eigen {3.0.5} Source. Follow the build and install instructions in the INSTALL file, which are:

```
cd ~/dev/eigen-eigen-43d9075b23ef {directory name will vary by version}
mkdir build
cmake ..
make
sudo make install
```

flann from Source into /usr/local {1.7.1}

From http://www.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN, download FLANN {1.7.1} Source. Follow the build and install instructions in the FLANN manual, bottom of page 3, which are:

```
cd flann-x.y.z-src {replace x.y.z with the corresponding version number}
mkdir build
cd build
cmake ..
make
sudo make install {this step not listed in FLANN manual}
```

vtk from Source into /usr/local {5.10.0}

From www.vtk.org/VTK/resources/software.html, download VTK {5.10.0} Source. Follow the build and install instructions in the README file, which are:

```
cd ~/dev/VTK
ccmake .
make
sudo make install
```

Note that I made NO changes to the VTK build scripts. The defaults for Carbon and Cocoa in CMakeCache.txt are:

```
VTK_USE_CARBON:BOOL=OFF
VTK_USE_COCOA:BOOL=ON
```

X11 is not installed by default in Mountain Lion. In CMakeCache.txt, the various X11 variables therefore typically indicate something like "..._PATH-NOTFOUND".

Modify environment variable as follows:

```
DYLD_LIBRARY_PATH="/usr/local/lib/vtk-5.10:$DYLD_LIBRARY_PATH"
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH
```

I have not determined why I needed to add VTK to the DYLD_LIBRARY_PATH.

qhull from Source into /usr/local {2012.1_2}

From www.qhull.org/download/, download Qhull {2012.1_2} Source. Follow the build and install instructions in the README file for "Installing Qhull with CMake 2.6 or later", which are:

```
Extract Qhull from qhull...tgz or qhull...zip
cd build
cmake ..
make
sudo make install
```

OpenNI & Sensor from PCL Pkg's into /usr, /etc, /var {1.3.2.1, 5.0.3.3}

To date, I have not completed building OpenNI & Sensor from Source. Therefore, I am currently using the PCL prebuilt packages at www.pointclouds.org/downloads/macosx.html:

```
OpenNI 1.3.2.1:  OpenNI-MacOSX-v1.3.2.1.pkg
Sensor 5.0.3.3:  Sensor-MacOSX-v5.0.3.3.pkg
```

Install these packages by double-clicking on them and following the instructions.

These two packages install files in several directories, including (not sure if this is a complete list):

```
/usr/bin
/usr/lib
/usr/include/ni
/usr/etc/ni
/usr/share/java
/etc/openni
/etc/primesense
/var/log/primesense/XnSensorServer
```

For the Xbox Kinect Sensor, make the following changes to this file:

```
/etc/openni/SamplesConfig.xml

diff SamplesConfig-orig.xml SamplesConfig.xml
19c19
<              <Mirror on="true"/>
---
             <Mirror on="false"/>
24c24
<              <Mirror on="true"/>
---
             <Mirror on="false"/>
```

Otherwise, the Kinect depth and RBG images will be mirrored (flipped left-right).

CUDA from NVIDIA Pkg {5.0.24 Release Candidate}

From developer.nvidia.com/cuda/cuda-pre-production, download "Mac OS X: CUDA 5.0 Release Candidate" {5.0.24 Release Candidate} pkg. Install the package by double-clicking on it and following the instructions.

Modify environment variable as follows:

```
PATH="/Developer/NVIDIA/CUDA-5.0/bin:$PATH"
DYLD_LIBRARY_PATH="/Developer/NVIDIA/CUDA-5.0/lib:
    $DYLD_LIBRARY_PATH"

export PATH=$PATH
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH
```

THE COMPILERS, TOOLS, and DEPENDENCIES ARE NOW READY TO GO!

PCL Trunk Build with Apple g++ Compiler

PCL trunk compiles with Apple's g++ (llvm-g++-4.2) compiler. Version 7081 is known to work. I haven't tried more recent versions.

Download and build PCL trunk as follows:

```
cd ~/dev
svn co -r 7081 http://svn.pointclouds.org/pcl/trunk pcl-trunk

{there changes to 5 files needed to get things working - see below}

cd pcl-trunk && mkdir build && cd build
cmake -DCMAKE_C_COMPILER=gcc -DCMAKE_CXX_COMPILER=g++
    -DCMAKE_BUILD_TYPE=RelWithDebInfo ..
make
```

These are changes to 5 files currently needed to get things working.

1) Change kinfu_app and pcl_kinfu_largeScale to an Application Bundle so that keyboard inputs in VTK work.

```
Index: gpu/kinfu/tools/CMakeLists.txt
===================================================================
--- gpu/kinfu/tools/CMakeLists.txt    (revision 7081)
+++ gpu/kinfu/tools/CMakeLists.txt    (working copy)
@@ -19,7 +19,7 @@

  source_group("Source Files" FILES ${srcs} )

-   PCL_ADD_EXECUTABLE(${the_target} ${SUBSYS_NAME} ${srcs} ${hdrs})
+   PCL_ADD_EXECUTABLE_OPT_BUNDLE(${the_target} ${SUBSYS_NAME} $
{srcs} ${hdrs})
  target_link_libraries(${the_target} pcl_common pcl_io ${OPENNI_LIBRARIES}
pcl_visualization pcl_gpu_kinfu)

  if(OpenCV_FOUND)

Index: gpu/kinfu_large_scale/tools/CMakeLists.txt
===================================================================
--- gpu/kinfu_large_scale/tools/CMakeLists.txt    (revision 6954)
+++ gpu/kinfu_large_scale/tools/CMakeLists.txt    (working copy)
@@ -29,7 +29,8 @@
```

```
   source_group("Source Files" FILES ${srcs} )

-    PCL_ADD_EXECUTABLE(${the_target} ${SUBSYS_NAME} ${srcs} ${hdrs})
+#    PCL_ADD_EXECUTABLE(${the_target} ${SUBSYS_NAME} ${srcs} ${hdrs})
+    PCL_ADD_EXECUTABLE_OPT_BUNDLE(${the_target} ${SUBSYS_NAME} $
{srcs} ${hdrs})
   target_link_libraries(${the_target} pcl_common pcl_io ${OPENNI_LIBRARIES}
pcl_visualization pcl_gpu_kinfu_large_scale pcl_octree)

  ## STANDALONE MARCHING CUBES
```

2) Change the number of voxels in the KinFu TSDF Volume from 512 to 256 or 128 to accommodate the 1024 MB memory in the NVIDIA GPU in this MacBook Pro configuration.

```
Index: gpu/kinfu_large_scale/tools/tsdf_volume.h
===================================================================
=
--- gpu/kinfu_large_scale/tools/tsdf_volume.h    (revision 6954)
+++ gpu/kinfu_large_scale/tools/tsdf_volume.h    (working copy)
@@ -43,9 +43,9 @@
#include <pcl/console/print.h>


-#define DEFAULT_GRID_RES_X 512  // pcl::device::VOLUME_X ( and _Y, _Z)
-#define DEFAULT_GRID_RES_Y 512
-#define DEFAULT_GRID_RES_Z 512
+#define DEFAULT_GRID_RES_X 128  // pcl::device::VOLUME_X ( and _Y, _Z)
+#define DEFAULT_GRID_RES_Y 128
+#define DEFAULT_GRID_RES_Z 128

#define DEFAULT_VOLUME_SIZE_X 3000
#define DEFAULT_VOLUME_SIZE_Y 3000
```

3) The CUDA nvcc compiler does not accept the -Wno-unused-but-set-variable flag. I quickly removed it in two places below. This needs more work.

```
Index: gpu/CMakeLists.txt
===================================================================
=
--- gpu/CMakeLists.txt    (revision 6954)
+++ gpu/CMakeLists.txt    (working copy)
@@ -9,7 +9,8 @@
   if(CMAKE_COMPILER_IS_GNUCXX)
     string(REPLACE "-Wold-style-cast" "" CMAKE_CXX_FLAGS "$
{CMAKE_CXX_FLAGS}")
```

```
      string(REPLACE "-Wno-invalid-offsetof" "" CMAKE_CXX_FLAGS "$
{CMAKE_CXX_FLAGS}")
-       set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wno-conversion -Wno-
unused-but-set-variable -Wno-unused-parameter -Wno-unused-variable -Wno-unused-
function")
+#       set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wno-conversion -Wno-
unused-but-set-variable -Wno-unused-parameter -Wno-unused-variable -Wno-unused-
function")
+       set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wno-conversion -Wno-
unused-parameter -Wno-unused-variable -Wno-unused-function")
  endif()

  collect_subproject_directory_names(${CMAKE_CURRENT_SOURCE_DIR}
"CMakeLists.txt" PCL_GPU_MODULES_NAMES PCL_GPU_MODULES_DIRS)
Index: cuda/CMakeLists.txt
===================================================================
=
--- cuda/CMakeLists.txt    (revision 6954)
+++ cuda/CMakeLists.txt    (working copy)
@@ -9,7 +9,8 @@
   if(CMAKE_COMPILER_IS_GNUCXX)
       string(REPLACE "-Wold-style-cast" "" CMAKE_CXX_FLAGS "$
{CMAKE_CXX_FLAGS}")
       string(REPLACE "-Wno-invalid-offsetof" "" CMAKE_CXX_FLAGS "$
{CMAKE_CXX_FLAGS}")
-       SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wno-conversion -Wno-
unused-but-set-variable -Wno-unused-parameter -Wno-unused-variable -Wno-unused-
function")
+#       SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wno-conversion -Wno-
unused-but-set-variable -Wno-unused-parameter -Wno-unused-variable -Wno-unused-
function")
+       SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wno-conversion -Wno-
unused-parameter -Wno-unused-variable -Wno-unused-function")
   endif()

   collect_subproject_directory_names(${CMAKE_CURRENT_SOURCE_DIR}
"CMakeLists.txt" PCL_CUDA_MODULES_NAMES PCL_CUDA_MODULES_DIRS)
```

Output of PCL CMake

```
-- The C compiler identification is GNU 4.2.1
-- The CXX compiler identification is GNU 4.2.1
-- Checking whether C compiler has -isysroot
-- Checking whether C compiler has -isysroot - yes
-- Checking whether C compiler supports OSX deployment target flag
-- Checking whether C compiler supports OSX deployment target flag - yes
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Checking whether CXX compiler has -isysroot
-- Checking whether CXX compiler has -isysroot - yes
-- Checking whether CXX compiler supports OSX deployment target flag
-- Checking whether CXX compiler supports OSX deployment target flag - yes
-- Check for working CXX compiler: /usr/bin/g++
-- Check for working CXX compiler: /usr/bin/g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- GCC < 4.3 found, enabling -Wno-deprecated
-- Performing Test HAVE_MM_MALLOC
-- Performing Test HAVE_MM_MALLOC - Success
-- Performing Test HAVE_POSIX_MEMALIGN
-- Performing Test HAVE_POSIX_MEMALIGN - Success
-- Performing Test HAVE_SSE4_1_EXTENSIONS
-- Performing Test HAVE_SSE4_1_EXTENSIONS - Success
-- Performing Test HAVE_SSE3_EXTENSIONS
-- Performing Test HAVE_SSE3_EXTENSIONS - Success
-- Performing Test HAVE_SSE2_EXTENSIONS
-- Performing Test HAVE_SSE2_EXTENSIONS - Success
-- Performing Test HAVE_SSE_EXTENSIONS
-- Performing Test HAVE_SSE_EXTENSIONS - Success
-- Found SSE4.1 extensions, using flags:  -msse4.1 -mfpmath=sse
-- Not found OpenMP
-- Boost version: 1.50.0
-- Found the following Boost libraries:
--   system
--   filesystem
--   thread
--   date_time
--   iostreams
-- checking for module 'eigen3'
--   found eigen3, version 3.1.1
-- Found Eigen: /usr/local/include/eigen3
-- Eigen found (include: /usr/local/include/eigen3)
```

```
-- checking for module 'flann>=1.7.0'
--   found flann, version 1.7.1
-- Found FLANN: /usr/local/lib/libflann_cpp.dylib (Required is at least version "1.7.0")
-- FLANN found (include: /usr/local/include, lib: optimized;/usr/local/lib/
libflann_cpp.dylib;debug;/usr/local/lib/libflann_cpp-gd.dylib)
-- Found LIBUSB_1: /usr/local/lib/libusb-1.0.dylib
-- checking for module 'libusb-1.0'
--   found libusb-1.0, version 1.0.9
-- Found USB_10: /usr/local/lib/libusb-1.0.dylib
-- Found OpenNI: /usr/lib/libOpenNI.dylib
-- OpenNI found (include: /usr/include/ni, lib: /usr/lib/libOpenNI.dylib)
-- Found ZLIB: /usr/lib/libz.dylib (found version "1.2.5")
-- Could NOT find PNG (missing:  PNG_LIBRARY PNG_PNG_INCLUDE_DIR)
-- Found Qhull: /usr/local/lib/libqhull.dylib
-- QHULL found (include: /usr/local/include, lib: optimized;/usr/local/lib/
libqhull.dylib;debug;/usr/local/lib/libqhull.dylib)
-- Found CUDA Toolkit v5.0
-- CUDA NVCC target flags: -gencode;arch=compute_20,code=sm_20;-
gencode;arch=compute_20,code=sm_21;-gencode;arch=compute_30,code=sm_30
-- Could NOT find Qt4 (missing:  QT_QMAKE_EXECUTABLE QT_MOC_EXECUTABLE
QT_RCC_EXECUTABLE QT_UIC_EXECUTABLE QT_INCLUDE_DIR
QT_LIBRARY_DIR QT_QTCORE_LIBRARY)
-- Could NOT find QVTK (missing:  QVTK_LIBRARY QVTK_INCLUDE_DIR)
-- VTK found (include: /usr/local/include/vtk-5.10, lib: /usr/local/lib/vtk-5.10)
-- Could NOT find Doxygen (missing:  DOXYGEN_EXECUTABLE)
-- Found OpenGL: /System/Library/Frameworks/OpenGL.framework
-- [pcl_keypoints] SSE 4.1 status: found. Enabling BriskKeypoint2D.
-- VTK found (include: /usr/local/include/vtk-5.10, lib: /usr/local/lib/vtk-5.10)
-- PCL_EXAMPLES_SUBDIRS /Users/ken/dev/pcl-trunk/examples/common/;/Users/
ken/dev/pcl-trunk/examples/features/;/Users/ken/dev/pcl-trunk/examples/filters/;/Users/
ken/dev/pcl-trunk/examples/geometry/;/Users/ken/dev/pcl-trunk/examples/keypoints/;/
Users/ken/dev/pcl-trunk/examples/segmentation/;/Users/ken/dev/pcl-trunk/examples/
surface/
-- DOXYGEN_FOUND NO
-- HTML_HELP_COMPILER
-- checking for module 'sphinx-build'
--   package 'sphinx-build' not found
-- Found PythonInterp: /usr/bin/python (found version "2.7.2")
-- Could NOT find Sphinx (missing:  SPHINX_EXECUTABLE)
-- The following subsystems will be built:
--   common
--   octree
--   io
--   kdtree
--   search
--   sample_consensus
```

```
--   filters
--   2d
--   geometry
--   features
--   ml
--   segmentation
--   visualization
--   surface
--   registration
--   keypoints
--   tracking
--   recognition
--   apps
--   3d_rec_framework
--   cuda_common
--   cuda_io
--   cuda_features
--   cuda_segmentation
--   cuda_sample_consensus
--   cuda_apps
--   examples
--   gpu_containers
--   gpu_utils
--   gpu_octree
--   gpu_features
--   gpu_kinfu
--   gpu_kinfu_large_scale
--   gpu_segmentation
--   outofcore
--   stereo
--   global_tests
--   tools
-- The following subsystems will not be built:
--   modeler: VTK was not built with Qt support.
--   cloud_composer: Cloud composer requires QVTK
--   in_hand_scanner: Qt4 was not found.
--   gpu_people: Disabled by default.
--   gpu_surface: Disabled by default.
--   gpu_tracking: Disabled by default.
--   simulation: Disabled by default.
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/ken/dev/pcl-trunk/build
```

PCL Trunk Build with Apple c++ (clang) Compiler

There are a few remaining compile errors using the Apple c++ (clang) compiler.

This section will be updated once these errors are resolved.

PCL Applications

The following applications have been successfully tested for basic operation:

1) pcl_openni_image.app/Contents/MacOS/pcl_openni_image

Approximately 70 seconds after the app is invoked, this warning message is logged to the console:

> Warning: USB events thread - failed to set priority. This might cause loss of data...

The windows appear approximately 75 seconds after the app is invoked. The delay in startup appears to occur in OpenNI/Sensor.

The average framerate is 30 Hz.

2) *pcl_openni_viewer.app/Contents/MacOS/pcl_openni_viewer*

Exhibits similar behavior as pcl_openni_image - long startup and the warning message.

The average framerate for drawing cloud is 15-16 Hz.

3) *kinfu_app.app/Contents/MacOS/kinfu_app*

Exhibits similar behavior as pcl_openni_image - long startup and the warning message.

The average framerate is 7.5 Hz.

4) pcl_kinfu_largeScale.app/Contents/MacOS/pcl_kinfu_largeScale

Exhibits similar behavior as pcl_openni_image - long startup and the warning message.

The average framerate is 7 Hz.

It runs, but experiencing an intermittent error (works on one invocation, fails the next) just after the windows appear:

> Error: out of memory /Users/ken/dev/pcl-trunk/gpu/containers/src/device_memory.cpp:260

FYI ... I found the gfxCardStatus app at http://codykrieger.com/gfxCardStatus useful to toggle the discrete GPU on and off. Toggling the GPU seemed to free up the memory after the out of memory error.