Integration of the Urban Robotics Out of Core Octree to PCL: URCS Final Report

Stephen Fox

September 29, 2012

Contents

This document summarizes the contributions made throughout the Urban Robotics Code Sprint for the Point Cloud Library. [1] The primary goal of this sprint was to integrate the Urban Robotics out of core octree into the Point Cloud Library code base. The project has been successful with the addition of **pcl_outofcore** to the trunk of PCL. The library now includes a command-line tool for creating out of core octrees from a set of PCD files, as well as a VTK-based viewer developed by Justin Rosen capable of displaying the tree at various resolutions or voxel sizes. Another member of the PCL community, Adam Stambler, has shared an open scene graph-powered out of core viewer for use with **pcl_outofcore** format octrees. [4]

The remainder of this document begins with details regarding the transformation of the original Urban Robotics source code to its current state. It is then followed with a review of available features with tutorial instructions. This should serve both as a guide to understand how things have changed, and as a reference for using the new **pcl_outofcore** library.

1 Overview

The integration of the out of core octree was sponsored by Urban Robotics, a company who has been developing dense aerial topological maps. For any reader unfamiliar with their work in geospatial visualization, check out their website for some videos of 3D-reconstructed scenes. [3] One of their focuses has been developing very large 3D geospatial maps spanning hundreds of square kilometers. Once the data have been processed into a point cloud format, the data are organized into the out of core octree data structure. Spatially sorting the data into an out of core octree allows adaptive rendering of the data by loading only the relevant information from the secondary data storage area. The on-disk data structure is a directory tree representing these octants storing binary point data along with the metadata describing the spatial characteristics of the voxel. The purpose of this project was to integrate Urban Robotics' implementation of this Out of Core Octree software into PCL.

We have finished the initial integration of the Urban Robotics Out of Core Octree Library into PCL which provides access to all of PCL's algorithms for pre-processing. The code is now fully operational giving access to other features of PCL, and is available in trunk/outofcore. The automatically generated API reference is located in the **pcl_outofcore** library documentation section. [2] The code can be used with any version of *Boost* supported by PCL ≥ 1.7 (i.e., trunk as of the date this document was written). The **cJSON** dependency is still bundled as a sublibrary in pcl_outofcore, but it is built and linked to automatically when linking to **pcl_outofcore**. Access to and from metadata have been encapsulated into two independent classes, **OutofcoreOctreeNodeMetadata**, and **OutofcoreOctreeMetadata**. Each of these two classes is inherited from an abstract interface, so changing the metadata to XML, for example, requires inheriting each class and overriding the **serializeMetadataToDisk** and **loadMetadataFromDisk** methods. Some care may need to be taken to multiplex the metadata format for cross compatibility, but it should be straight forward to migrate to another format if desired.

The out of core octree can be used with any point type containing "x", "y" and "z" fields. However, **pcl_outofcore** *does not support multiple point types in a single tree*. No internal checking is done to verify this. On the other hand, point clouds do not need to be filtered for NaN entries; the library will automatically ignore NaN points in the insertion methods.

2 New pcl_outofcore Library

Description of the Original Urban Robotics Code

The original Urban Robotics out of core octree source that PCL received at the beginning of the URCS/TRCS code sprints contained four files in the outofcore octree library:

1. octree.hpp
2. octree2.hpp

3. octree_ram_container.hpp
4. octree_disk_container.hpp

Furthermore, it contained a custom C-struct, `PointCloudTools::point`, located in `sdks/pointCloudTools.h`. These files have been split into separate header and implementation files in the pcl_octree library. PCL keeps its code in one of three types of files. The declaration of a class and of its members and methods belong in the header file, for **pcl_outofcore** in `outofcore/include/pcl/outofcore/*.h`. Templated implementation lives in `outofcore/include/pcl/outofcore/impl/*.hpp`, and non-templated implementation in `outofcore/src/*.cpp`. The four original files have now been split into separate files for each class:

1. Header Files
   - (a) octree_base.h
   - (b) octree_base_node.h
   - (c) octree_abstract_node_container.h*
   - (d) metadata.h*: an abstract class for metadata
   - (e) outofcore_node_data.h*: interface for node-level outofcore metadata files (tree.oct_idx)xsy
   - (f) outofcore_base_data.h*: interface for high-level outofcore octree metadata (tree.octree JSON file)
   - (g) OutofcoreDepthFirstIterator.h*: depth first iterator for the outofcore octree, with direct access to the outofcore node class
   - (h) OutofcoreIteratorBase.h*: abstract iterator class based directly on the iterator in pcl_octree.
   - (i) octree_disk_container.h: I/O to the disk container
   - (j) octree_ram_container.h: an in-core data structure for the outofcore octree (this is no longer necessary since a query can easily be converted to a pcl_octree)
   - (k) outofcore_node_data.h: contains the main node data structure along with recursive methods for insertion and query
   - (l) boost.h: contains all boost headers needed in pcl_outofcore
   - (m) cJSON.h: minor modifications have been made for compatibility with the PCL build system
2. Templated Implementation Files
   - (a) octree_base.hpp
   - (b) octree_base_node.hpp
   - (c) octree_disk_container.hpp
   - (d) octree_ram_container.hpp
   - (e) OutofcoreDepthFirstIterator.hpp
   - (f) OutofcoreIteratorBase.hpp
3. Source Files
   - (a) cJSON.cpp
   - (b) outofcore_node_data.cpp
   - (c) outofcore_base_data.cpp
4. Library include Files (includes headers for the entire pcl_outofcore library)
   - (a) outofcore.h
   - (b) outofcore_impl.h

* denotes a file which contains a new class.

Most of the unused code has been stripped from the source files. In particular to note,

1. The original C-struct based point type has been deprecated in favor of any PCL point type; Templating is currently available for insertion, but **it is recommended to use only PointCloud2 methods for querying.**
2. All exceptions have been converted to the official PLCException class.
3. The Ram Container class is not necessary any longer. Using the PointCloud2 result of a query, the user can construct a **pcl_octree**-based data structure. There are many powerful options available for in-memory representations of octrees built into PCL.

Furthermore, many functions and variables have been refactored to meet the PCL style policies. The library is now packageable as a shared object library, and fits the need of the developer with a full, cross-platform `cmake` build system, with accompany doxygen documentation. The html version of the doxygen documentation is available at docs.pointclouds.org. A LaTeXpdf can also be produced for a convenient off-line resource with `make doc` after configuring your build.

An Important Note on Templating: Use PointCloud2(!) but template with pcl::PointXYZ

Currently, the outofcore classes are still templated against:

1. Container Type
2. Point Type

However, we plan to deprecate the templating on PointT in pcl_outofcore. For this reason, it is best to use query and insertion methods that use the PointCloud2 objects. Since the PointT templating has not officially been stripped yet, it is best to instantiate the OutofcoreOctreeBase against pcl::PointXYZ, **but use the data type you prefer packed into a PointCloud2 object** for insertion.

The issue that arose with templating is the requirement to know the desired Point Type at compile time, or to deal with many hard-coded cases for each templated type. Justin, who has been developing a VTK-powered out of core viewer within PCL for the TRCS, and I decided that using `sensor_msgs::PointCloud2` would provide more flexibility for the out of core library. In particular, the PointCloud2 data structure, albeit messier because of the need to deal with the binary data directly, is a more dynamic data type. The simplest example of this is the following:

Bare Bones Example

```
typedef OutofcoreOctreeBase<OutofcoreOctreeDiskContainer<pcl::PointXYZ>, pcl::PointXYZ>
    OctreeDisk;
typedef OutofcoreOctreeBaseNode<OutofcoreOctreeDiskContainer<pcl::PointXY>, pcl::PointXYZ>
    OctreeDiskNode;

int depth = 3;

//specify a 20x20x20 m bounding box; all points inserted should fall into this bounding box
Eigen::Vector3d min (-10.0, -10.0, -10.0);
Eigen::Vector3d max (10.0, 10.0, 10.0);

//specify the root node metadata path
boost::filesystem::path file_location ("tree/tree.oct_idx");

//Create the outofcore octree; if the bounding box doesn't have square sides, it will be
    expanded to a cube
```

```
OctreeDisk* octree;
octree = new OctreeDisk (depth, min, max, file_location, "ECEF");

sensor_msgs::PointCloud2::Ptr cloud (new sensor_msgs::PointCloud2 ());

//read in an arbitrary PCD file into a PointCloud2
pcl::io::loadPCDFile (argv[1], *cloud);
//insert it into the leaves of the outofcore octree
octree->addPointCloud (cloud, false);

//read in another arbitrary PCD file into a PointCloud2, but ASSUME it has the same fields
pcl::io::loadPCDFile (argv[2], *cloud);
//insert it into the leaves of the outofcore octree
octree->addPointCloud (cloud, false);

//configure the sampling percentage
octree->setSamplePercent (0.125);
octree->buildLOD ();
```

Conversion of Documentation

Documentation has been integrated from the original `octree_doc.docx` file directly into the source using *doxygen* tags. Most doxygen documentation can be found in the `*.h` header files. We have also introduced more comments in the implementation where needed. An API reference is automatically generated by the PCL build system and can be updated and maintained by any PCL developer, or via a patch submitted by community users.

No More External Dependencies

The original Urban Robotics Octree came distributed with two external dependencies:

1. Boost 1.47
2. cJSON

The Boost libraries are already a required dependency of PCL. Following the normal installation instructions of PCL will satisfy this requirement. Since cJSON is not currently a dependency of PCL, we have kept the cJSON.h/cJSON.cpp library for writing and parsing the metadata. The possibility of developing/integrating a web viewer may be of interest to some in the PCL community, so we decided to keep this format of the meta-data.

Outofcore Node Format

On disk, each interior node or leaf node can contain up to two files:

***.oct_idx** a JSON metadata file with the format:

```
{
  "version":  3,
  "bb_min":   [xxx,yyy,zzz],
  "bb_max":   [xxx,yyy,zzz],
  "bin":      "path_to_data.pcd"
}
```

This metadata can be accessed directly using the OutofcoreOctreeNodeMetadata class. This class abstracts the interface to the on-disk metadata, so in theory the format can easily be changed to XML, YAML, or some other desired format.

***.pcd** the standard format (v7+) PCD file containing any points associated with that node. This file exists for all leaves, but only exists for interior ("Branch") nodes if the LOD have been built. Access to this file occurs via the OutofcoreOctreeDiskContainer class.

The root node contains one additional file:

***.octree** which contains the high-level information about the structure of the octree. The format is:

```
{
   "name": "test ,
   "version": 3,
   "pointtype": "urp",                  #(needs to be changed*)
   "lod": 3,                            #depth of the tree
   "numpts":  [X0, X1, X2, ..., XD],    #total number of points at each LOD
   "coord_system": "ECEF"               #the tree is not affected by this value
}
```

Note that # denotes a comment for purposes of this document, but should not be used in the JSON files.

Compressed Binary Data

The format of the PCD files is now *binary compressed*, which provides space saving comapred to the original linear binary dump of the C-structs. Furthermore, as PCL grows and evolves, access to other on-disk formats will be immediately available. This is not publicly parameterized, so if you would prefer ASCII data in your outofcore tree, you will have to modify the `insert` family of methods to construct the outofcore octree. There should be no complications in reading other PCD formats, so no modifications of the read methods would be necessary.

If you would like to convert the data from a particular node to ASCII PCD file, it is simple to query the bounding box of interest and save the result as an ASCII PCD file.

Unit Test Development

The original unit tests for the Urban Robotics out of core library were written in the Boost testing framework. Since PCL uses Google test, we have converted all unit tests to match our framework. With regard to unit tests, Justin and I did the following:

1. Converted from Boost to GTest framework (Thanks Justin!)
2. Analyzed the expected behavior of the code and make the unit tests more robust
3. Wrote new unit tests for new features in the out of core library as we started integrating support for the PCL-based point cloud data types

I have made an effort to test the public interface of OutofcoreOctreeBase well, since this is PCL policy, and the class the user interacts with primarily. One major change that I made from the original UR code base is to standardize the constructors. All constructors now pass through common initialization routines with consistent input validation, which should smooth out some former inconsistencies.

Cross Platform Building

I have done my development on a Debian system (i7, 8 GB of RAM). The Point Cloud Library has an automated build system that tests every commit on all of our supported hardware systems (x86, x86-64) in Ubuntu, Mac OS X, and Windows 7 environments. With these builds, whose results are available publicly at build.pointclouds.org, we can monitor the ability to build and run our software in different environments. As with all PCL libraries, we have taken care to write cross-platform code, and consequently it builds, executes, and successfully passes our unit tests in all of our supported run-time environments.

3 Using the Out-of-core libraries with PCL

To develop directly with the Out of core Libraries, two header files are enough:

```
#include <pcl/outofcore/outofcore.h>
#include <pcl/outofcore/outofcore_impl.h>
```

Using PCL's CMake build system, link to outofcore as well. For more information on PCL's CMake build system, see Using PCL Config.

4 Features

Insertion

Methods: **addPointCloud**, **addPointCloud_and_genLOD**
Public access to inserting a cloud into an outofcore octree is available using the **addPointCloud** method. NaN points will be ignored, and all points will be inserted at full density to the leaves. Multiple point clouds can be inserted into the outofcore octree by calling **addPointCloud** iteratively. **I recommend using only PointCloud2 methods** at this point in time.

Query

Methods: **queryBoundingBox**
queryBoundingBox is the public interface provided to query an outofcore tree. The method is overloaded and depending on the arguments passed, will return either all of the points falling within the queried bounding box at the specified depth, or a list of all PCD files whose union will contain all points within the queried bounding box. These are the PCD files from any node whose bounding box intersects the queried bounding box region. These methods wrap what was formerly called **queryBBIncludes**, **queryBBIncludes_subsample** and **queryBBIntersects**. They are still available in the public interface, but the legacy methods will be deprecated to streamline the interface.

Levels of Depth: Multiresolution Out of core Octrees

Methods: **buildLOD**, **addPointCloud_and_genLOD**
One of the key features of an out of core octree is the so-called "levels of depth," or LOD for short. By convention calling the root of the octree "level 0" (N.B. this departs from some other octree nomenclature), each level i is an eight-fold subdivision of leve $i-1$. The Level of Depth is constructed by randomly downsampling the number of points at each level. This percentage is accessible via the `setSamplePercent` method in the `OutofcoreOctreeBase` class. This allows for creating multi-resolution representations of the point cloud as a pre-processing step, which can quickly be queried for rendering. During rendering, the desired resolution (accessed by "level" or "depth" and bounding box) of the point cloud can be accessed based on some heurstic, such as the voxel's distance from the viewpoint. Rendering is not part of the out of core back-end. These tasks are left for the two compatible out of core viewers currently available.

buildLOD re-builds the LOD of an outofcore octree using a multi-resolution approach. Each branch node is a union of subsamplings of its child leaves. The percentage of subsampling is determined by **setSamplePercent** and is by default $0.125^{depth-maxdepth}$.

addPointCloud_and_genLOD Using this method builds the LOD as a point cloud is inserted. This builds a different style of LOD, where no data are repeated. Thus to render this, rather than re-rendering ALL data at a particular level, the new data should be added to the previous level and re-rendered. This feature is still experimental. No access to the parameters are available here at the moment.

For more information on LOD algorithms, see [5].

Breadth First Iterator

I have converted Julius' implementation of his depth first iterator for the octree in pcl_octree. While doing this, I tried to add some methods to the pcl_outofcore classes that drives the pcl_outofcore interface toward Julius' abstarct interface in pcl_octree. This could conceivably allow our classes to inherit his at some point. I have held off on making this change because there are still some structural incompatibilities between the pcl_octree and pcl_outofcore libraries.

```
//load an octree from disk
OctreeDisk octree2 (file_location , true);

//iterate over the octree, depth first
OutofcoreDepthFirstIterator<pcl::PointXYZ, pcl::outofcore::OutofcoreOctreeDiskContainer<
    pcl::PointXYZ> > it (octree2);
OctreeDisk::Iterator myit (octree2);

while ( *myit !=0 )
{
  octree2.printBoundingBox (**myit);
  myit++;
}
```

Command line Tool for Building Out of Core Octrees

Justin and I have been maintaining a useful tool for builting out of core octrees from an arbitrary number of PCD files in trunk/outofcore/tools called `pcl_outofcore_process`. Assuming PCD files contain the same point types, an out of core octree can be constructed from these files. The construction process can be parameterized either by *depth* or by *resolution*. If the resolution (i.e. the maximum leaf size) is specified, the depth will automatically be calculated. Beware of having too deep of a tree: the LOD building can take quite a long time. Recommended uses for `pcl_outofcore_process` are given below.
Suppose we have two large PCD files with fields: X Y Z I, named data01.pcd and data02.pcd respectively, with 50-100 million points each. Viewing these clouds with pcl_viewer will push the limits of the pcl_viewer on most systems. In my experience, these clouds will probably render, but the interactivity is extremely poor (time between re-rendering views is high). To create an outofcore octree with multi-resolution LOD built, use the pcl_outofcore_process command line tool.

Parameterizing by Depth

To create a multi-resolution outofcore octree with depth 4,

```
pcl_outofcore_process -depth 4 -gen_lod data01.pcd data02.pcd my_outofcore_tree
```

You can navigate manually through the directories to examine the data and view individual node PCD files, or several node PCD files simultaneously, using `pcl_viewer`. The outofcore tree output to disk is compatible with Adam Stambler's OSGPCL viewer, and with the pcl_outofcvore_viewer.

Parameterizing by Resolution

To create a multi-resolution outofcore octree with voxel size of leaf nodes at most 50 cm, specify the following:

```
pcl_outofcore_process -resolution 0.50 -gen_lod data01.pcd data02.pcd
    my_other_outofcore_tree
```

Out of core Octree Viewers

There are two viewers available for use with the new pcl_outofcore library.

PCL's VTK-based Outofcore Viewer: pcl_outofcore_viewer

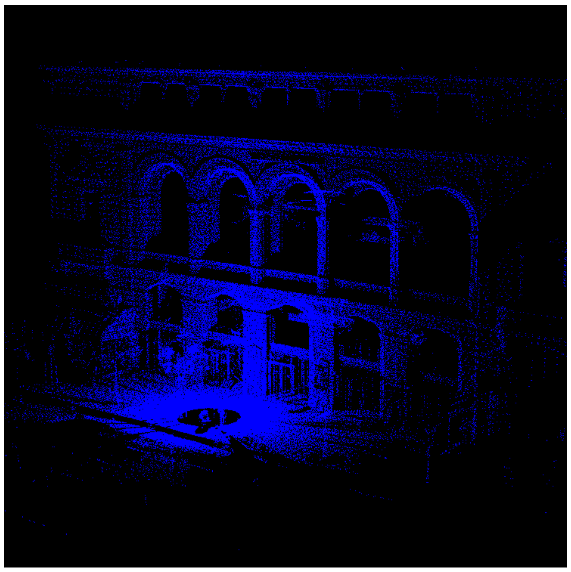
Justin Rosen, who is working on the TRCS, has been developing a viewer for pcl_outofcore. It should be noted that *this is still under heavy development*. The code is available in trunk/outofcore/tools. This can be used to view a single depth of the tree, or bounding voxel representation of the outofcore octree. Type `pcl_outofcore_viewer -h` to see the available options.

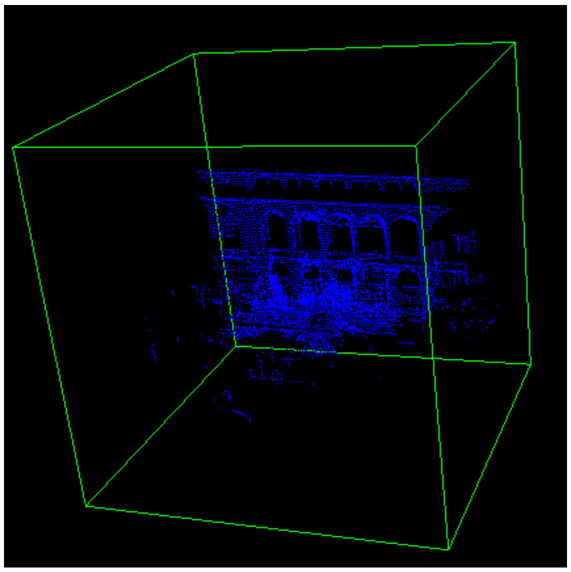
On my Debian-based system with an ATI Radeon HD 5450 graphics card, my graphics card driver is not compatible with VTK 5.8. If trouble is encountered, try building the viewer against VTK-5.10 rather than VTK-5.8. This solved OpenGL incompatibility for my system.

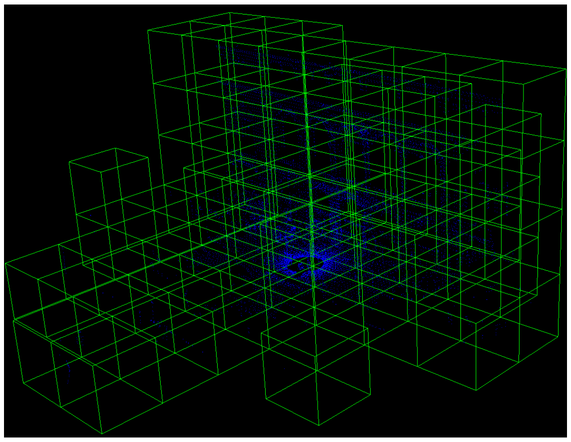
The "outofcore" aspects (related to caching and paging) have not yet been implemented for this viewer, but it serves as a useful tool for viewing the LOD of an outofcore octree. In particular, it allows the points to be rendered from arbitrary point types and uses some of the newer features of VTK+OpenGL (in particular, vertex buffer objects). Be sure to follow his blog for continuing development notes.

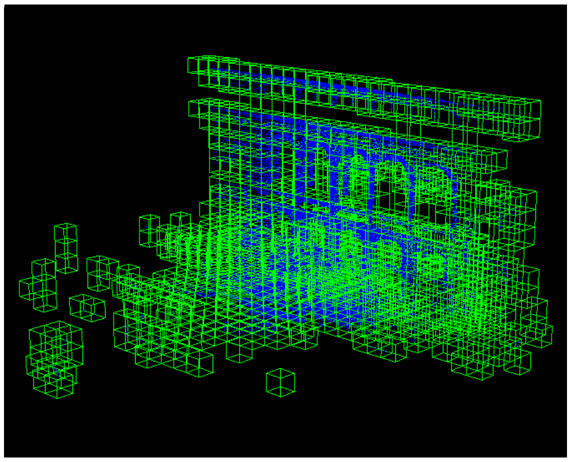
The two trees below have been built with large point clouds in the Trimble data set. `data/Trimble/Outdoor1` contains many large point cloud files (yet these are relatively modest compared to the amount of data our sensors are producing). On my machine, viewing these PCD files (with 30 million points+) causes the pcl_viewer to re-render very slowly. The cloud renders, but it is tedious to navigate and view. Creating an outofcore octree allows these to be rendered seamlessly, and with Adam's viewer, adaptively.

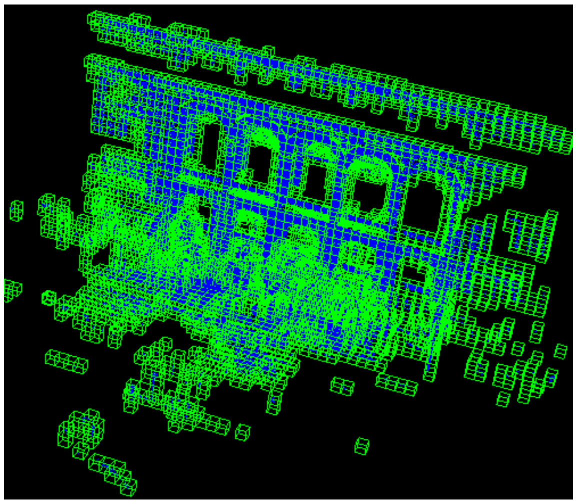
The following five screen shots come from Trimble/Outdoor1/Theatre_1.pcd converted to an outofcore octree, and viewed with `pcl_outofcore_viewer` at various depths.



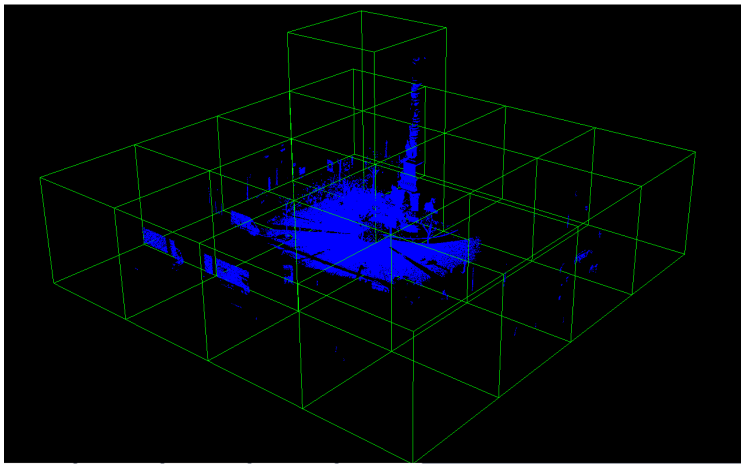


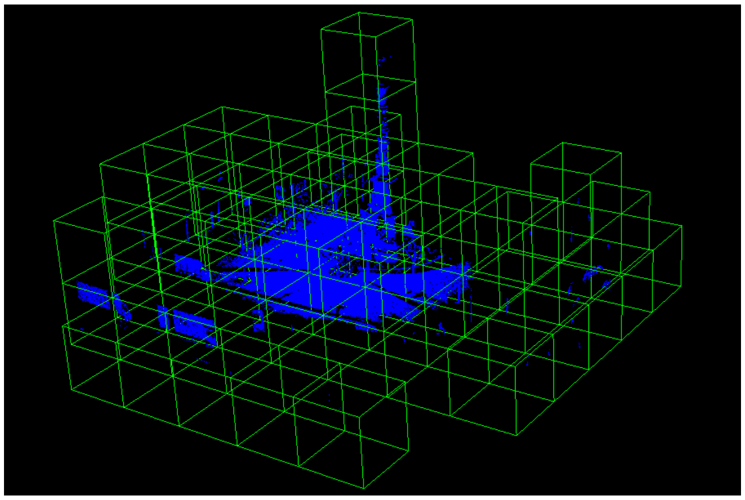


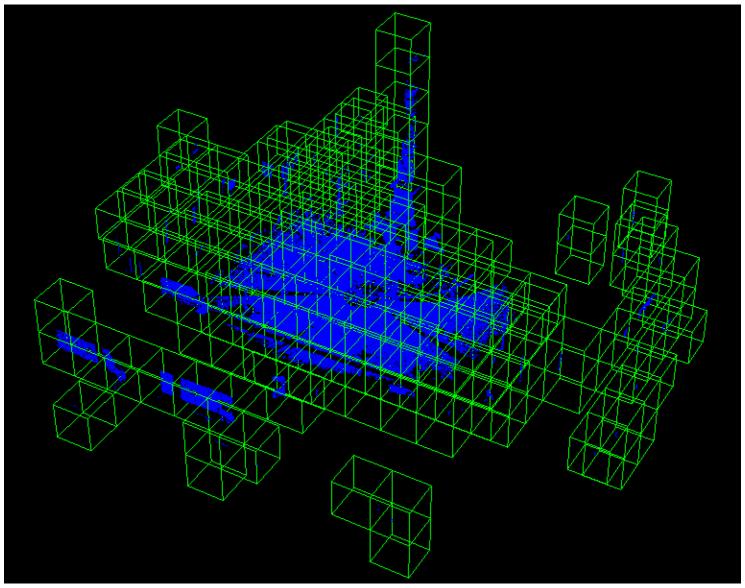




The screen shots below come from Fontaine_2.pcd converted to an outofcore tree with `pcl_outofcore_process` and viewed with `pcl_outofcore_viewer` at various depths.





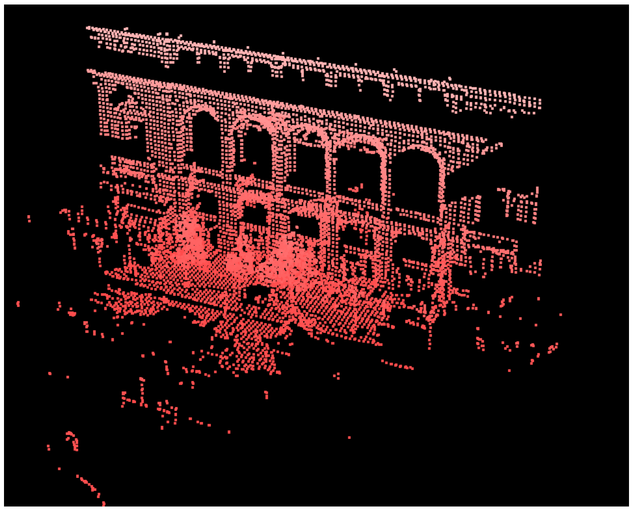


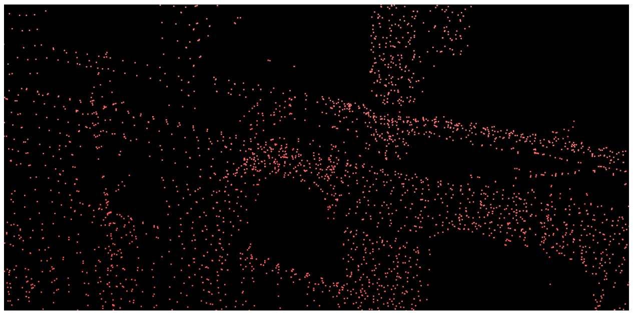
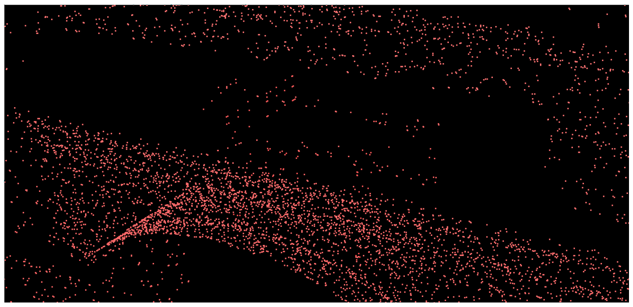
OSGPCL: A contribution from Adam Stambler

Adam Stambler, a member of the PCL community, has shared an out of core octree viewer based on the open scene graph library. The viewer is not officially supported by PCL and is not included in the PCL library, but it can be accessed at his github account located at https://github.com/adasta/osgpcl. Up to date instructions on how to use the viewer are available in the `README`. It automatically handles paging of the LODs based on perspective and distance from the cloud, and can easily render very large pcl_outofcore octrees if the LOD are built.

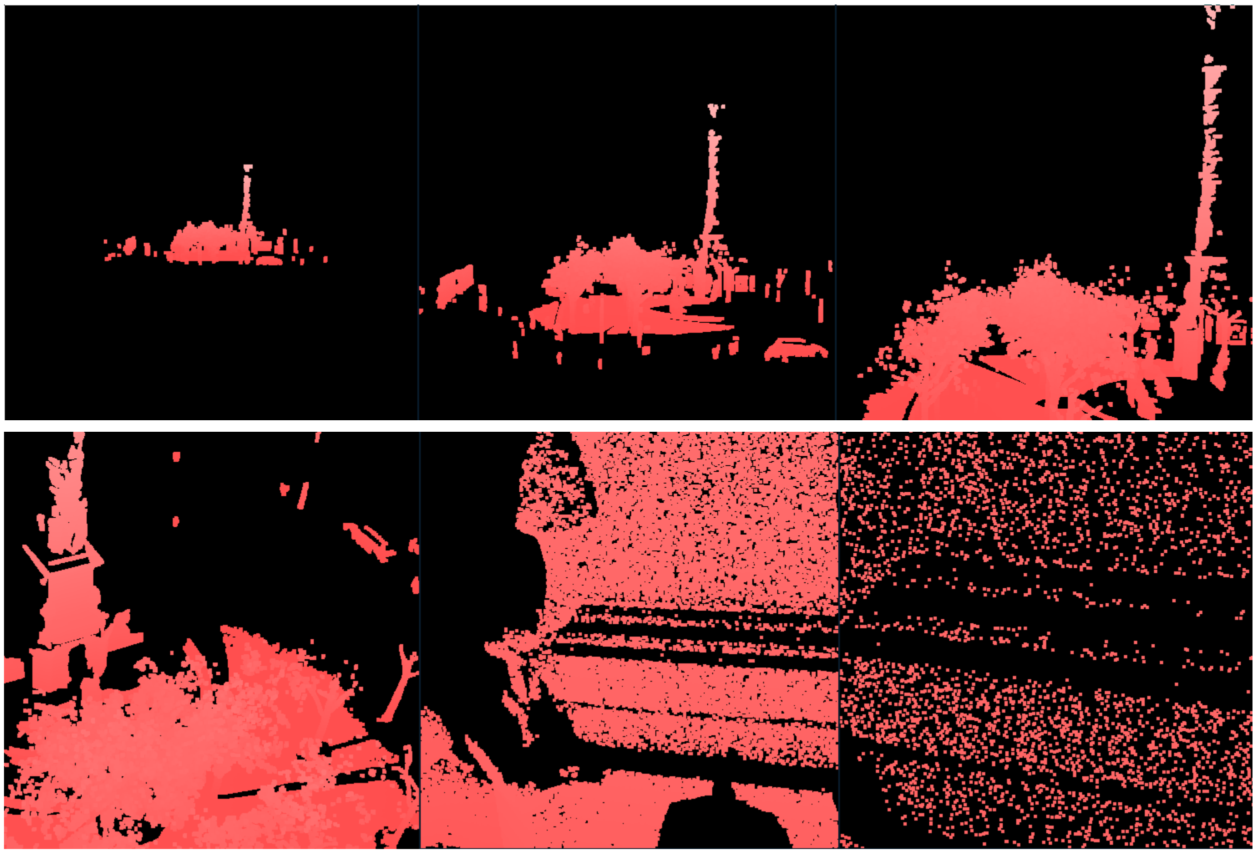
One particularly nice feature of this viewer is that the point cloud re-renders at higher and lower resolution as you zoom in and out to certain parts of the point cloud. This is difficult to demonstrate with screen shots, so I encourage you to try this yourself!

These images are renderings of the Theatre using Adam's viewer. I've tried to show the effect of zooming into a particular region, but it is difficult to show.



Here is the fountain rendered at various viewpoints using the osgpcl viewer. I had three windows open for viewing the same outofcore with no trouble.



5 Tips for Isolating Bugs

I have added a file, trunk/test/octants.pcd, which contains only 8 points. The octree that should be generated from these points should be a tree of depth 3 with branches to the 0 and 1 octants. If you encounter a bug, please try to reproduce it with this file or one of equal simplicity. Furthermore, building with depth 0 (only root node) should essentially copy your data to the root node.

6 Conclusions and Future Work

The outofcore library is fully functional at this point, but needs further optimization and some direction by those interested in using the library. I focused a lot of effort into ensuring the basic functionality by carefully writing new unit tests. If a bug has slipped through the unit tests, I have done my best to write a new unit test to catch it in the event it arises again.

I have tried to make sound design decisions while maintaining as much lateral compatibility with the Urban Robotics octrees as possible. Ultimately, I hope to streamline the interface more and watch it evolve with more feedback from the PCL community as it matures into a valuable visualization tool capable of wrangling the explosion of data that is upon us.

7 Acknowledgments

I would like to thank Jacob Schloss, who wrote the initial code, for all of the information and insights he shared with me during the pcl_outofcore sprint. Furthermore, thank you Justin, Julius and Radu for your collaborative spirit. This project has been an unrivaled learning experience for me, and it has been a pleasure and honor working with you.

References

[1] PCL Urban Robotics Code Sprint Blog. http://www.pointclouds.org/blog/urcs.

[2] Point Cloud Library Documentation: Module outofcore. https://docs.pointclouds.org/trunk/group__outofcore.html.

[3] Urban Robotics. http://www.urbanrobotics.net.

[4] Adam Stambler. osgpcl: OpenSceneGraph Point Cloud Library Cloud Viewer. https://github.com/adasta/osgpcl, 2012.

[5] Claus Scheiblauer and Michael Wimmer. Out-of-core selection and editing of huge point clouds. *Computers & Graphics*, 35(2):342–351, April 2011.