VoxGen

A C++ Program to Voxelize LAS Files

Megan Kress
kressmeg@msu.edu

Andrew Finley
finleya@msu.edu

April 2, 2015

Contents

1 Introduction to LAS Files

LAS files[5] contain data from LiDAR point clouds. Each point has a set of properties described by the LAS file:

- Classification: Points may or may not be classified. Classifications include unclassified, ground, low vegetation, medium vegetation, high vegetation, building, and water.
- Scan Angle

- Return Number: For each pulse there are 1-5 returns, so a point may be described by its return number for the pulse in which it was scanned.

2 Libraries

2.1 Boost

Boost[1] is a library that in VoxGen allows processing of an entire directory at once using its filesystem[10] class. A user can choose whether to run VoxGen on a directory or on a single file.

2.2 libLAS

libLAS[8] is a C/C++ library that can read and write LAS files.

2.3 kdtree

kdtree[4] is a simple C library that creates and iterates through kd-trees. Kd-trees are k-dimensional binary search trees. In this case, our kd-tree operates in three dimensions corresponding to the x, y, and z values for each point in the LAS file. The kdtree library has a function to add points to a kd-tree object

along with a function to determine the points within a given range of some point.

2.4 GDAL

GDAL[3] is " a translator library for raster and vector geospatial data formats." In this program, GDAL is used to read the .tif files providing metrics for the LiDAR data. The GDAL API[2] provides functions to open files and get data from the raster for an individual pixel.

3 Custom Classes

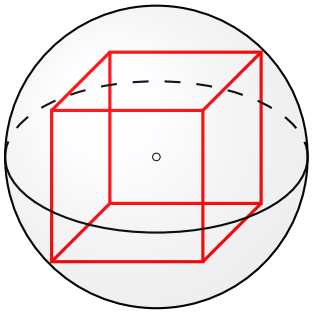
3.1 Point

The simple Point class contains the x, y, and z coordinates for a point.

3.2 Voxel

The Voxel class contains the x, y, and z coordinates for the center of the voxel along with the number of points in the voxel and a vector of the points. This

class also contains two functions: inVox() and trimVox(). inVox() takes in as arguments a point or 3D coordinates and returns whether or not that point is within the voxel. trimVox() is called on a voxel in VoxGen because the kdtree library's function to determine the points within a given range of a point returns points in a sphere around the voxel, and the Voxel class only wants points within the rectangular prism that is the voxel.



3.3 VoxCol

The VoxCol class simply contains the x and y center coordinates for a given voxel column and a vector of voxels (Voxel objects) that are within that column. The

VoxCol class also has a function to set the densities of each voxel using the MacArthur-Horn transformation.

3.4 VoxData

VoxData is a class that summarizes the voxel information for a given LAS file. It includes a vector of all the voxel columns (VoxCol objects), the number of points in all the voxels, the number of voxel columns, and the number of voxels in each voxel column. VoxData also includes methods to read and write files.

3.5 vgpar

vgpar requires a text document parameter file of relevant information for the VoxGen program (see below). The vgpar class constructor takes in a string of

the text file location, and the class includes functions getString() and getNum(), which utilize a map to obtain a value referenced in the paragraph. For example,

getString("filter") returns the string "n," which is then used as user input in VoxGen to determine that the user does not wish to filter points from the LAS file.

Example VoxGenPar Text File

```
##A paragraph of relevant information for VoxGen Program

fileORdir file

inFile /home/megan/Data/Duke/Duke_AM_24Oct2013_c0r0.las
inDir /home/megan/Data/Duke

#Filtering y or n?
filter n
```

```
#If filtering on, select parameters
#Classifications A: All, U: Unclassified, G: Ground,
#L: Low Vegetation, M: Medium Vegetation,
#H: High Vegetation, B: Building, W: Water
classes A
returns A
angle 20

#Voxel Dimensions
base 10
height 10

#Initial Coordinates(all 0s for min/max from file)
x1 0
x2 0

y1 0
y2 0

#Look for line intersections y or n?
lines n


#R output y or n?
routput y
outName testR
outDirectory /home/megan/Downloads

#VoxData File y or n? The file index number will be
#appended to the title in the given directory.
voxdata y
vdataTitle voxData
vdataDir /home/megan/Downloads
```

4 Functions

- **Voxelize**: VoxGen takes in user input in the form of a text file. Based on the LAS file and filters specified, VoxGen divides the points from the LAS file into voxels of specified dimensions.

- **Output R File**: If the user chooses to create an R file, VoxGen will create an R source file, which will input various matrices into the R workspace that contain voxel information.

- **Line Intersections**: A user can input two three-dimensional points, and VoxGen will return a vector of Voxel objects that indicates which voxels would be intersected by a line drawn between the two points.

- **VoxData File**: VoxGen reads and writes text files to create or save VoxData object information.

- **Voxel Column Metrics Flat File**: VoxGen writes a text file in which each row represents data corresponding to a voxel column's metrics.

- **Voxel Column Histogram Flat File**: VoxGen writes a text file in which each row has the number of points in each voxel column's voxels.

See appendix for descriptions of the two flat files.

5 VoxGenR

VoxGenR is an R package that can be used for basic analysis of voxel columns created in VoxGen. Note: VoxGenPar must have the R out file enabled in order to create an R file that can be imported by VoxGenR. The source package for

VoxGenR is located in the VoxGen/VoxGenR directory. It can be loaded into R by running R and entering the following commands:

```
> install.packages("/path/to/VoxGen/VoxGenR/VoxGenR_1.0.tar.gz",
    repos = NULL, type="source")
> library(VoxGenR)
> importData("/path/to/rOutFile.R")
```

R's environment now contains a list of matrices called VoxList, in which each index represents a four column voxel column matrix, wherein each row represents a voxel. Column 1 is the x center of the voxel, 2 the y center, 3 the z center, and 4 the number of points in the voxel. Here is an example of accessing a list index in VoxList:

```
 > VoxList[[83]]
         [,1]    [,2] [,3] [,4]
[1,] 363515.5 4306537 10.2   52
[2,] 363515.5 4306537 15.2    0
[3,] 363515.5 4306537 20.2    0
[4,] 363515.5 4306537 25.2    0
[5,] 363515.5 4306537 30.2    0
[6,] 363515.5 4306537 35.2    0
```

This shows that the first voxel in voxel column 83 contains 52 points.
The VoxGenR functions selectPoints() and heatMapAll() may now be run:

selectPoints(): This function displays the centers of all the voxel columns and allows the user to select certain points. The columns that are selected will be displayed as histograms individually along with a heat map of the voxel point densities for all columns selected.

heatMapAll(): This function displays a heat map of all the voxel columns next to each other to gain a basic understanding of the points' height distributions for the LAS file used in VoxGen.

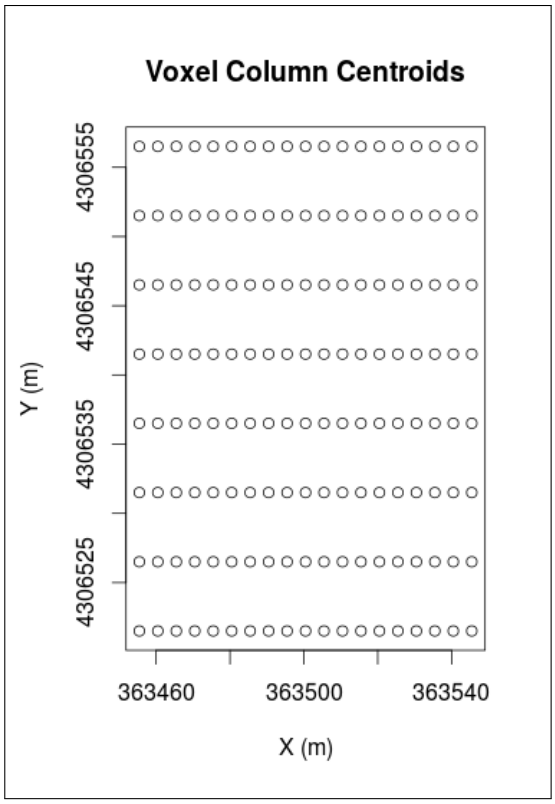


Figure 1: Voxel Column Centers Displayed after importData()

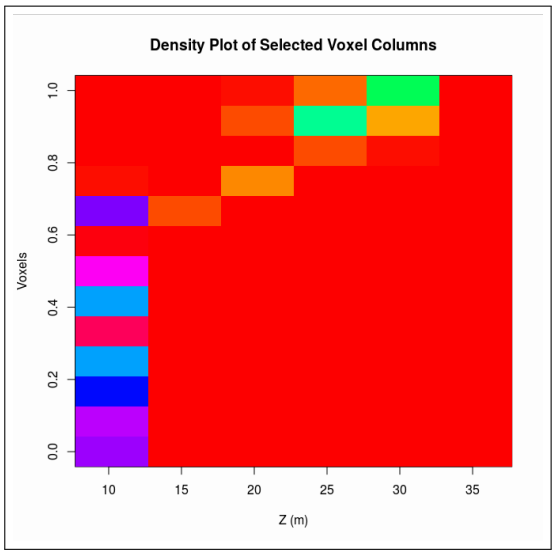


Figure 2: Rainbow Heat Map Displayed after selectPoints() - Red is Least Points

6 Quality Assurance

QGIS was used to check the accuracy of the output lidar metric flat files. The flat file was loaded into a spreadsheet program, and four metric raster layers were loaded into QGIS. A random number generated selected which rows from the flat file to examine. "ZoomToCoordinates" found the point in QGIS, and the "Identify Features" tool displayed the metric values at that point.

Acquisition: 20140712_brendon_1a
Type: mosaic.tif files
Metrics: tree_d6, ground_slope, shrub_mean, all_p50
Centroids: 60

Acquisition: 20140714_tanana_flats
Type: individual .tif files per segment
Metrics: pulse_density, tree_mean, all_d9
Centroids: 40

All of the 100 centroids had data in the flat file that matched the observed data in QGIS.

7 Appendix

7.1 VoxGen.cpp

```
1 /*
2 VoxGen, 2015
3
```

```
4  Megan Kress <kressmeg@msu.edu>
5
6  This program takes the input file VoxGenPar and its
7  user-defined categories.
8
9  The program may output multiple files:
10     - R File to be examined with VoxGenR library
11     - VoxData file
12     - Lidar Metrics Flat File
13     - Histogram Flat File
14             (see VoxGenFlatFileFormats text file)
15
16
17 */
18
19 #include <stdio.h>
20 #include <time.h>       /* time */
21 #include <vector>
22 #include <algorithm>
23 #include <iostream>
24 #include <cstddef>
25 #include <string>
26 #include <set>
27 #include <cmath>
28 #include <exception>
29 #include <fstream>  // std::ifstream
30 #include "liblas.hpp"
31 #include "kdtree.h"
32 #include <iomanip>
33 #include <map>
34 #include <math.h>
35 #include "vgpar.h"
36 #include "point.h"
37 #include "voxel.h"
38 #include "voxcol.h"
39 #include "voxdata.h"
40
41 #include "boost/filesystem/operations.hpp"
42 #include "boost/filesystem/fstream.hpp"
43 #include "boost/filesystem/path.hpp"
44
45
46 #include "gdal_priv.h"
47 #include "cpl_conv.h" // for CPLMalloc()
48
49 using namespace std;
50
51 namespace fs = boost::filesystem;
52
53 namespace std
```

```
54 {
55     template<> struct less<Point>
56     {
57        bool operator() (const Point& lhs, const Point&
          ↪  rhs)
58        {
59            return lhs.x < rhs.x || lhs.y < rhs.y ||
              ↪ lhs.z < rhs.z;
60        }
61     };
62
63 }
64
65 bool exists(string file)
66 {
67     ifstream infile (file.c_str());
68     return infile.good();
69 };
70
71 vector<Point> moveAlongLine(double x1, double y1,
   ↪ double z1, double x2, double y2, double z2)
72 {
73     vector<Point> result;
74
75     double diffX, diffY, diffZ;
76
77     diffX = x2 - x1;
78     diffY = y2 - y1;
79     diffZ = z2 - z1;
80
81     double x = x1;
82     double y = y1;
83     double z = z1;
84
85     double incX, incY, incZ;
86     incX = diffX / 1000.0;
87
88     incY = diffY / 1000.0;
89
90     incZ = diffZ / 1000.0;
91
92    for(int i = 0; i < 1000; ++i)
93     {
94         x += incX;
95         y += incY;
96         z += incZ;
97
98         Point p (x, y, z);
99
100        result.push_back(p);
```

```
101     }
102
103     return result;
104 }
105
106
107 vector<Voxel> voxelsIntersected(kdtree* VoxCenters,
        ↪ map<Point,Voxel> voxMap, double xi, double yi,
        ↪ double zi, double xf, double yf, double zf,
        ↪ double r)
108 {
109     Point p1 (xi, yi, zi);
110     Point p2 (xf, yf, zf);
111
112     vector<Voxel> VoxResults;
113
114     vector<Point> points = moveAlongLine(xi, yi, zi,
            ↪ xf, yf, zf);
115
116     Voxel vox;
117
118     double pos [3];
119
120
121     while(points.size() > 0)
122     {
123         Point p = points.back();
124
125         double pp [3];
126         pp[0] = p.x;
127         pp[1] = p.y;
128         pp[2] = p.z;
129
130
131         struct kdres * results = kd_nearest_range(
                ↪ VoxCenters, pp, r);
132
133         while( !kd_res_end( results ) ) {
134
135             /* get the data and position of the
                    ↪ current result item */
136             kd_res_item( results, pos );
137
138
139             Point voxCenter (pos[0], pos[1], pos[2]);
140             vox = voxMap[voxCenter];
141
142
143             if(vox.inVox(p)){
144
```

```
145                 if(find(VoxResults.begin(), VoxResults
                        ↪ .end(), vox) != VoxResults.end()
                        ↪ ) {
146                 } else {
147                     VoxResults.push_back(vox);
148                 }
149             }
150 
151 
152         /* go to the next entry */
153           kd_res_next( results );
154 
155       }
156 
157         points.pop_back();
158     }
159 
160     return VoxResults;
161 }
162 
163 void intersectVoxels(kdtree* VoxCenters, map<Point,
        ↪ Voxel> voxMap, double radius)
164 {
165     double xi, xf, yi, yf, zi, zf;
166     cout << "\nxi: ";
167     cin >> xi;
168     cout << "yi: ";
169     cin >> yi;
170     cout << "zi: ";
171     cin >> zi;
172 
173     cout << "\nxf: ";
174     cin >> xf;
175     cout << "yf: ";
176     cin >> yf;
177     cout << "zf: ";
178     cin >> zf;
179 
180     vector<Voxel> ret = voxelsIntersected(VoxCenters,
            ↪ voxMap, xi, yi, zi, xf, yf, zf, radius);
181 
182     cout << "\nNumber of voxels intersected: " << ret.
            ↪ size();
183 
184     for(unsigned int i = 0; i < ret.size(); ++i)
185     {
186         ret.at(i).print();
187     }
188 }
189 
```

```
190 void toR(vector<VoxCol> voxColumnsVector, int
        ↪ numVoxCols, int n, int numInCol, string outFile)
191 {
192
193     ofstream myfile;
194
195
196
197     myfile.open (outFile.c_str());
198     cout << "\nWriting data to a file...\n";
199
200     myfile << "\nVoxCol <- matrix(nrow = " <<
            ↪ numVoxCols << ", ncol = 3)";
201     myfile << "\nVoxels <- matrix(nrow = " << n << ",
            ↪ ncol = 4)";
202     myfile << "\nVoxList <- list()";
203
204     if(numVoxCols == 0) return;
205
206     int voxIndex = 1;
207
208     for(int i = 1; i <= numVoxCols; ++i)
209   {
210
211      VoxCol voxcol = voxColumnsVector.back();
212
213
214
215      myfile << "\nVoxCol[" << i << ", 1] <- " <<
            ↪ setprecision(20) << voxcol.xC;
216      myfile << "\nVoxCol[" << i << ", 2] <- " <<
            ↪ setprecision(20) << voxcol.yC;
217
218
219     myfile << "\nVoxColList <- matrix(ncol = 4, nrow =
            ↪  " << numInCol << ")";
220
221      int index = numInCol;
222
223     while(voxcol.voxels.size() > 0)
224     {
225
226
227       Voxel voxel = voxcol.voxels.back();
228
229       myfile << "\nVoxels[" << voxIndex << ", 1] <- "
            ↪ << setprecision(20) << voxel.cX;
230       myfile << "\nVoxels[" << voxIndex << ", 2] <- "
            ↪ << setprecision(20) << voxel.cY;
```

```
231       myfile << "\nVoxels[" << voxIndex << ", 3] <- "
              ↪ << setprecision(20) << voxel.cZ;
232       myfile << "\nVoxels[" << voxIndex << ", 4] <- "
              ↪ << setprecision(20) << voxel.pointNum;
233
234       myfile << "\nVoxColList[" << index << ", 1] <-"
              ↪ << setprecision(20) << voxel.cX;
235       myfile << "\nVoxColList[" << index << ", 2] <-"
              ↪ << setprecision(20) << voxel.cY;
236       myfile << "\nVoxColList[" << index << ", 3] <-"
              ↪ << setprecision(20) << voxel.cZ;
237       myfile << "\nVoxColList[" << index << ", 4] <-"
              ↪ << setprecision(20) << voxel.pointNum;
238
239       voxcol.voxels.pop_back();
240       voxIndex++;
241       index--;
242
243     }
244
245     myfile << "\nVoxList[[" << i << "]] <- VoxColList"
            ↪ ;
246
247
248     voxColumnsVector.pop_back();
249
250   }
251
252    cout << "Done.";
253
254    myfile.close();
255 };
256
257 //Suppress output: https://bbs.archlinux.org/viewtopic
       ↪ .php?id=79378
258
259 int main(int argc, char *argv[])
260 {
261     time_t timerI;
262
263     time(&timerI);
264
265     string vgfile;
266     bool  fileEntry = false;
267     string enteredFile;
268
269     std::streambuf* cout_sbuf;
270
271     if (argc > 3)
272     {
```

```
273         cout_sbuf = std::cout.rdbuf(); // save
                ↪ original sbuf
274         std::ofstream   fout("/dev/null");
275         std::cout.rdbuf(fout.rdbuf()); // redirect '
                ↪ cout' to a 'fout'
276 
277         fileEntry = true;
278         vgfile = argv[1];
279         enteredFile = argv[2];
280     }
281     else if (argc > 2)
282     {
283       fileEntry = true;
284       vgfile = argv[1];
285       enteredFile = argv[2];
286     }
287     else if (argc > 1)
288     {
289         vgfile = argv[1];
290     }else {
291         cout << "\nEnter /path/to/VoxGenPar: ";
292         cin >> vgfile;
293         vgfile = vgfile.c_str();
294     }
295 
296     cout << '\n' << vgfile << '\n';
297     vgpar par (vgfile);
298 
299     vector<string> filesInDir;
300 
301     string fORd = par.getString("fileORdir");
302 
303     if(fileEntry)
304     {
305       filesInDir.push_back(enteredFile);
306 
307     }else if(fORd[0] == 'd'){
308     //**********************************
309     // Boost Filesystem Stuff
310     // Adapted from
311     // http://www.boost.org/doc/libs/1_31_0/libs/
            ↪ filesystem/example/simple_ls.cpp
312     //**********************************
313 
314     string inDir = par.getString("inDir");
315 
316     fs::path full_path( inDir );
317 
318     unsigned long file_count = 0;
319     unsigned long dir_count = 0;
```

```
320
321     if ( !fs::exists( full_path ) )
322   {
323     std::cout << "\nDirectory not found";
324     return 1;
325   }
326
327     if ( fs::is_directory( full_path ) )
328   {
329     std::cout << "\nIn directory: "
330               << full_path << "\n\n";
331     fs::directory_iterator end_iter;
332     for ( fs::directory_iterator dir_itr( full_path );
333           dir_itr != end_iter;
334           ++dir_itr )
335     {
336       try
337       {
338         if ( fs::is_directory( *dir_itr ) )
339         {
340           ++dir_count;
341         }
342         else
343         {
344           filesInDir.push_back(dir_itr->path().c_str()
                ↪ );
345           ++file_count;
346         }
347       }
348       catch ( const std::exception & ex )
349       {
350         cout << "Error";
351       }
352     }
353     std::cout << "\n" << file_count << " files\n"
354               << dir_count << " directories\n";
355   }
356   else // must be a file
357   {
358     std::cout << "\nFound File (not directory): " <<
        ↪ full_path << "\n";
359   }
360
361     //*****************************
362     // End Boost Filesystem Stuff
363     //*****************************
364     }else
365     {
366         string selectedFile = par.getString("inFile");
367         filesInDir.push_back(selectedFile);
```

```
368     }
369
370     for(unsigned int ii = 0; ii < filesInDir.size();
            ↪ ++ii) {
371
372     cout << "\n**********************************\n";
373
374     string inFile = filesInDir.at(ii);
375
376     if(par.getString("routput")[0] == 'y') if(!exists(
            ↪ par.getString("outDirectory"))){
377         cout << "\nR output directory does not exist.
                ↪ Exiting...\n";
378         return 0;
379     }
380     if(par.getString("hff")[0] == 'y') if(!exists(par.
            ↪ getString("hffDir"))){
381         cout << "\nHistogram flat file output
                ↪ directory does not exist. Exiting...\n";
382         return 0;
383     }
384     if(par.getString("lmff")[0] == 'y') if(!exists(par
            ↪ .getString("lmffDir"))){
385         cout << "\nLidar metrics flat file output
                ↪ directory does not exist. Exiting...\n";
386         return 0;
387     }
388
389     ifstream ifs;
390     ifs.open( inFile.c_str(), ios::in | ios::binary );
391
392     liblas::Reader reader = liblas::Reader(ifs);
393
394     liblas::Header const& header = reader.GetHeader();
395
396     cout << '\n' << "Signature: " << header.
            ↪ GetFileSignature() << '\n';
397     cout << "Points count: " << header.
            ↪ GetPointRecordsCount() << '\n';
398
399     liblas::SpatialReference srs = header.GetSRS();
400
401     cout << "Spatial Reference: " << srs.GetWKT() << '
            ↪ \n';
402
403     cout << "\nMin X: " << setprecision(12) << header.
            ↪ GetMinX() << '\n';
404     cout << "Max X: " << setprecision(12) << header.
            ↪ GetMaxX() << '\n';
```

```
405     cout << "Min Y: " << setprecision(12) << header.
            ↪ GetMinY() << '\n';
406     cout << "Max Y: " << setprecision(12) << header.
            ↪ GetMaxY() << '\n';
407     cout << "Min Z: " << setprecision(12) << header.
            ↪ GetMinZ() << '\n';
408     cout << "Max Z: " << setprecision(12) << header.
            ↪ GetMaxZ() << '\n' << '\n';
409
410
411
412
413     string filter;
414
415     filter = par.getString("filter");
416
417     if(filter[0] == 'y' || filter[0] == 'Y')
418     {
419
420     // ************************************
421     // Filter by Classification
422     // ************************************
423
424     string classChoice;
425
426   std::vector<liblas::Classification> classes;
427
428   classChoice = par.getString("classes");
429
430   for(unsigned int i = 0; i < classChoice.length(); ++
        ↪ i)
431     {
432         if(classChoice[i] == 'A') break;
433         if(classChoice[i] == 'U') classes.push_back(
              ↪ liblas::Classification(1));
434         if(classChoice[i] == 'G') classes.push_back(
              ↪ liblas::Classification(2));
435         if(classChoice[i] == 'L') classes.push_back(
              ↪ liblas::Classification(3));
436         if(classChoice[i] == 'M') classes.push_back(
              ↪ liblas::Classification(4));
437         if(classChoice[i] == 'H') classes.push_back(
              ↪ liblas::Classification(5));
438         if(classChoice[i] == 'B') classes.push_back(
              ↪ liblas::Classification(6));
439         if(classChoice[i] == 'W') classes.push_back(
              ↪ liblas::Classification(9));
440     }
441
442     std::vector<liblas::FilterPtr> filters;
```

```
443     liblas::FilterPtr class_filter = liblas::FilterPtr
            ↪ (new liblas::ClassificationFilter(classes));
444
445     // eInclusion means to keep the classes that match
            ↪ .  eExclusion would
446     // throw out those that matched
447     class_filter->SetType(liblas::FilterI::eInclusion)
            ↪ ;
448     filters.push_back(class_filter);
449
450     // **********************************
451     // End Filter by Classification
452     // **********************************
453
454     // **********************************
455     // Filter by return
456     // **********************************
457
458     string returnChoice;
459     liblas::ReturnFilter::return_list_type returns;
460
461
462     returnChoice = par.getString("returns");
463
464     for(unsigned int i = 0; i < returnChoice.length();
            ↪  ++i)
465     {
466         if(classChoice[i] == 'A') break;
467         if(classChoice[i] == '1') returns.push_back(1)
                ↪ ;
468         if(classChoice[i] == '2') returns.push_back(2)
                ↪ ;
469         if(classChoice[i] == '3') returns.push_back(3)
                ↪ ;
470         if(classChoice[i] == '4') returns.push_back(4)
                ↪ ;
471         if(classChoice[i] == '5') returns.push_back(5)
                ↪ ;
472     }
473
474
475     liblas::FilterPtr return_filter = liblas::
            ↪ FilterPtr(new liblas::ReturnFilter(returns,
            ↪ false));
476     return_filter->SetType(liblas::FilterI::eInclusion
            ↪ );
477     filters.push_back(return_filter);
478
479     // **********************************
480     // End filter by return
```

```
481     // ************************************
482
483 //
484 //      ************************************
485 //      Filter by Scan Angle
486 //      ************************************
487
488     double angleChoice;
489     liblas::ContinuousValueFilter<double>::filter_func
        ↪  f;
490     liblas::ContinuousValueFilter<double>::
        ↪ compare_func c;
491     cout << "\n\nSelect Maximum Scan Angle.";
492     cout << "\nAngle: ";
493
494     //cin >> angleChoice;
495
496     angleChoice = par.getNum("angle");
497
498     f = &liblas::Point::GetScanAngleRank;
499     c = std::less_equal<double>();
500
501     liblas::FilterPtr angle_filterptr = liblas::
        ↪ FilterPtr(new liblas::ContinuousValueFilter<
        ↪ double>(f, angleChoice, c));
502
503     angle_filterptr->SetType(liblas::FilterI::
        ↪ eInclusion);
504
505     filters.push_back(angle_filterptr);
506
507     c = std::greater_equal<double>();
508     angleChoice *= -1;
509
510     liblas::FilterPtr angle_filterptr2 = liblas::
        ↪ FilterPtr(new liblas::ContinuousValueFilter<
        ↪ double>(f, angleChoice, c));
511
512     angle_filterptr2->SetType(liblas::FilterI::
        ↪ eInclusion);
513
514     filters.push_back(angle_filterptr2);
515
516
517     // ************************************
518     // End Filter by Scan Angle
519     // ************************************
520
521
522    reader.SetFilters(filters);
```

```
523     }
524
525
526     kdtree *kd = kd_create(3);
527     struct kdres *results;
528     cout << "\nAdding points to tree..." << '\n';
529
530    int totalPoints = 0;
531
532    if(par.getString("hff")[0] == 'y')
533     {
534   while (reader.ReadNextPoint())
535   {
536     liblas::Point const& p = reader.GetPoint();
537
538     double point[3];
539
540     point[0] = p.GetX();
541     point[1] = p.GetY();
542     point[2] = p.GetZ();
543
544
545     ++totalPoints;
546
547
548     kd_insert(kd, point, kd);
549   };
550
551     }
552
553   cout << "\nPoints in kdtree: " << totalPoints <<'\n'
       ↪ ;
554   //Set min and max values for each dimension
555   double x1 = header.GetMinX();
556   double x2 = header.GetMaxX();
557
558   double y1 = header.GetMinY();
559   double y2 = header.GetMaxY();
560
561   double z1 = header.GetMinZ();
562   double z2 = header.GetMaxZ();
563
564   if(par.getNum("z1") != 0) z1 = par.getNum("z1");
565
566   if(par.getNum("z2") != 0) z2 = par.getNum("z2");
567
568   double voxelsLo = 0;
569   double zStart = par.getNum("zmin");
570   double ht = par.getNum("height");
571   while(zStart < z1)
```

```
572   {
573     zStart += ht;
574     ++voxelsLo;
575   }
576
577   zStart -= ht;
578   z1 = zStart;
579
580   cout << "\n...Done creating tree.\n";
581
582
583
584   double b;
585   double h;
586   double xcoord;
587   double ycoord;
588   double xcoord2;
589   double ycoord2;
590
591
592   b = par.getNum("base");
593
594
595   h = par.getNum("height");
596
597
598   xcoord = par.getNum("x1");
599
600   if(xcoord == 0) xcoord = x1;
601
602 //  while(xcoord < x1 || xcoord > x2)
603 //  {
604 //    cout << "\nInitial x value: ";
605 //    cin >> xcoord;
606 //  }
607
608
609   xcoord2 = par.getNum("x2");
610
611   if(xcoord2 == 0) xcoord2 = x2;
612
613 //  while(xcoord2 < x1 || xcoord2 > x2 || xcoord2 <=
      ↪ xcoord)
614 //  {
615 //    cout << "\nFinal x value: ";
616 //    cin >> xcoord2;
617 //  }
618
619
620   ycoord = par.getNum("y1");
```

```
621
622   if(ycoord == 0) ycoord = y1;
623
624
625 //  while(ycoord < y1 || ycoord > y2)
626 //  {
627 //    cout << "\nInitial y value: ";
628 //    cin >> ycoord;
629 //  }
630
631
632   ycoord2 = par.getNum("y2");
633
634   if(ycoord2 == 0) ycoord2 = y2;
635
636 //
637 //  while(ycoord2 < y1 || ycoord2 > y2 || ycoord2 <=
      ↪ ycoord)
638 //  {
639 //    cout << "\nFinal y value: ";
640 //    cin >> ycoord2;
641 //  }
642
643   int numVoxX = ceil((xcoord2 - xcoord) / b);
644   int numVoxY = ceil((ycoord2 - ycoord) / b);
645
646   int numVoxCols = numVoxX * numVoxY;
647
648
649
650   vector<VoxCol> voxColumnsVector;
651
652   int numVox = 0;
653   double z = z1;
654   while(z < z2)
655   {
656     z += ht;
657     ++numVox;
658   }
659
660   z2 = z;
661
662   b = b/2.0;
663   h = h/2.0;
664
665
666   //Set Radius of Sphere
667   double r = sqrt(2*b*b + h*h);
668
669   double x = xcoord;
```

```
670   double y = ycoord;
671   double zcoord = z1;
672
673   double posX, posY, posZ;
674
675
676
677
678   int totalnumVox = numVoxCols * numVox;
679
680 //  vector<Voxel> allVoxels (totalnumVox);
681   vector<Voxel> allVoxels;
682
683   map<Point,Voxel> voxMap;
684
685   kdtree * kdVoxCenters = kd_create(3);
686
687   int kdPointCount = 0;
688
689
690   if(par.getString("hff")[0] == 'y')
691   {
692   cout << "\nNumber of Voxels: " << totalnumVox << '\n
        ↪ ';
693
694
695   cout << "Creating Voxels...\n";
696
697     int hundVox = totalnumVox/100.0;
698     int percent = 0;
699     int voxCount = 0;
700
701   int voxCt = 0;
702
703
704
705   time_t timer;
706
707   time(&timer);
708
709   for(int i = 0; i < numVoxY; ++i)
710   {
711     if(i == 0) y += b;
712
713     for(int j = 0; j < numVoxX; ++j)
714     {
715
716       if(j == 0) x += b;
717
718       VoxCol voxelColumn;
```

```
719
720         for(int k = 0; k < numVox; ++k)
721         {
722           if(k == 0) zcoord += h;
723           double pt[3];
724
725           pt[0] = x;
726           pt[1] = y;
727           pt[2] = zcoord;
728
729           results = kd_nearest_range(kd, pt, r);
730
731           int pointsNum = kd_res_size(results);
732
733           Voxel voxel;
734           voxel.pointNum = pointsNum;
735           voxel.cX = x;
736           voxel.cY = y;
737           voxel.cZ = zcoord;
738           voxel.d = b;
739           voxel.h = h;
740
741           double pos[3];
742
743           while( !kd_res_end( results ) ) {
744
745           /* get the data and position of the current
                 ↪ result item */
746           kd_res_item( results, pos );
747
748           Point point (pos[0], pos[1], pos[2]);
749           voxel.pointsInVox.push_back(point);
750
751
752           /* go to the next entry */
753           kd_res_next( results );
754
755         }
756
757          voxel.trimVox();
758          voxelColumn.voxels.push_back(voxel);
759
760           ++ voxCount;
761           if(hundVox > 0){
762          if(voxCount%hundVox == 0)
763           {
764               percent++;
765               cout << percent << "...\n";
766           }
767           }
```

```
768
769         zcoord += 2*h;
770       }
771
772         voxelColumn.xC = x;
773         voxelColumn.yC = y;
774
775 //        voxelColumn.setDensities();
776 //
777 //        for(unsigned int jj = 0; jj < voxelColumn.
    ↪ voxels.size(); ++jj)
778 //        {
779 //            allVoxels.at(voxCt++) = voxelColumn.
    ↪ voxels.at(jj);
780 //        }
781
782         voxelColumn.init(-99);
783         voxColumnsVector.push_back(voxelColumn);
784
785
786
787
788       zcoord = z1;
789       x = x + 2*b;
790
791     }
792       x = xcoord;
793       y = y + 2*b;
794   }
795
796
797
798
799   cout << "\nDone creating voxels.\n";
800
801    time_t timer2;
802
803   time(&timer2);
804
805     cout << "\nTime Elapsed Creating Voxels: " <<
        ↪ timer2 - timer;
806
807   } else {
808
809   for(int i = 0; i < numVoxY; ++i)
810   {
811     if(i == 0) y += b;
812
813     for(int j = 0; j < numVoxX; ++j)
814     {
```

```
815
816       if(j == 0) x += b;
817
818       VoxCol voxelColumn;
819
820        voxelColumn.xC = x;
821        voxelColumn.yC = y;
822
823        voxelColumn.init(-99);
824        voxColumnsVector.push_back(voxelColumn);
825
826
827       x = x + 2*b;
828
829     }
830       x = xcoord;
831       y = y + 2*b;
832   }
833
834   }
835
836     // ***************************************
837     // GDAL http://www.gdal.org/gdal_tutorial.html
838     // ***************************************
839
840
841     cout << "\nReading metrics...\n";
842
843     string fileName = inFile;
844
845     string delimiter = "/";
846     string delimiter1 = "_";
847     string delimiter2 = ".";
848     string delimiter3 = "-";
849
850     string ac, seg, splitNum;
851
852
853     size_t pos = 0;
854     size_t pos1 = 0;
855     size_t pos2 = 0;
856
857     while ((pos = fileName.find(delimiter)) != string
        ↪ ::npos) {
858         fileName.erase(0, pos + delimiter.length());
859     }
860
861     ac = par.getString("acquisition");
862
863     if(par.getString("split")[0] == 'y')
```

```
    {
        pos = fileName.find(delimiter1);
        seg = fileName.substr(0, pos);
        fileName.erase(0, pos + delimiter1.length());
        pos = fileName.find(delimiter2);
        splitNum = fileName.substr(0, pos);
//        string identify;
//
//        int word = 0;
//        while ((pos = fileName.find(delimiter1,
    ↪ fileName.find(delimiter1))) != string::npos)
//        {
//            pos = fileName.find(delimiter1);
//            if(word != 0) identify += "_";
//            identify += fileName.substr(0, pos);
//            fileName.erase(0, pos + delimiter1.
    ↪ length());
//            ++word;
//        }
//
//
//        word = 0;
//        while ((pos = identify.find(delimiter1)) !=
    ↪ string::npos) {
//            pos = identify.find(delimiter1);
//            if(word != 0) ac += "_";
//            ac += identify.substr(0, pos);
//            identify.erase(0, pos + delimiter1.
    ↪ length());
//            ++word;
//        }
//
//        seg = identify;
//
//        pos = fileName.find(delimiter2);
//        splitNum = fileName.substr(0, pos);
    } else {

        pos = 0;

        int word = 0;
        while ((pos = fileName.find(delimiter1)) !=
            ↪ string::npos) {
            pos = fileName.find(delimiter1);
            if(word != 0) ac += "_";
            ac += fileName.substr(0, pos);
            fileName.erase(0, pos + delimiter1.length
                ↪ ());
            ++word;
        }
```

```
908
909         pos = fileName.find(delimiter2);
910         seg = fileName.substr(0, pos);
911         splitNum = "";
912     }
913
914
915     vector<string> filevect;
916     vector<int> metricvect;
917     vector<string> keyvgpar;
918
919     keyvgpar.push_back("all_d0");
920     keyvgpar.push_back("all_d1");
921     keyvgpar.push_back("all_d2");
922     keyvgpar.push_back("all_d3");
923     keyvgpar.push_back("all_d4");
924     keyvgpar.push_back("all_d5");
925     keyvgpar.push_back("all_d6");
926     keyvgpar.push_back("all_d7");
927     keyvgpar.push_back("all_d8");
928     keyvgpar.push_back("all_d9");
929     keyvgpar.push_back("all_kurt");
930     keyvgpar.push_back("all_mean");
931     keyvgpar.push_back("all_p10");
932     keyvgpar.push_back("all_p20");
933     keyvgpar.push_back("all_p30");
934     keyvgpar.push_back("all_p40");
935     keyvgpar.push_back("all_p50");
936     keyvgpar.push_back("all_p60");
937     keyvgpar.push_back("all_p70");
938     keyvgpar.push_back("all_p80");
939     keyvgpar.push_back("all_p90");
940     keyvgpar.push_back("all_qmean");
941     keyvgpar.push_back("all_refl_kurt");
942     keyvgpar.push_back("all_refl_mean");
943     keyvgpar.push_back("all_refl_skew");
944     keyvgpar.push_back("all_refl_stdev");
945     keyvgpar.push_back("all_skew");
946     keyvgpar.push_back("all_stdev");
947     keyvgpar.push_back("ground_aspect");
948     keyvgpar.push_back("ground_elev_mean");
949     keyvgpar.push_back("ground_refl_kurt");
950     keyvgpar.push_back("ground_refl_mean");
951     keyvgpar.push_back("ground_refl_skew");
952     keyvgpar.push_back("ground_refl_stdev");
953     keyvgpar.push_back("ground_slope");
954     keyvgpar.push_back("pulse_density");
955     keyvgpar.push_back("pulse_scan_angle");
956     keyvgpar.push_back("returns_per_pulse");
957     keyvgpar.push_back("shrub_mean");
```

```
958     keyvgpar.push_back("shrub_refl_mean");
959     keyvgpar.push_back("shrub_refl_stdev");
960     keyvgpar.push_back("shrub_stdev");
961     keyvgpar.push_back("tree_aad");
962     keyvgpar.push_back("tree_crr");
963     keyvgpar.push_back("tree_d0");
964     keyvgpar.push_back("tree_d1");
965     keyvgpar.push_back("tree_d2");
966     keyvgpar.push_back("tree_d3");
967     keyvgpar.push_back("tree_d4");
968     keyvgpar.push_back("tree_d5");
969     keyvgpar.push_back("tree_d6");
970     keyvgpar.push_back("tree_d7");
971     keyvgpar.push_back("tree_d8");
972     keyvgpar.push_back("tree_d9");
973     keyvgpar.push_back("tree_fcover");
974     keyvgpar.push_back("tree_fract_all");
975     keyvgpar.push_back("tree_iqr");
976     keyvgpar.push_back("tree_kurt");
977     keyvgpar.push_back("tree_mad");
978     keyvgpar.push_back("tree_mean");
979     keyvgpar.push_back("tree_p10");
980     keyvgpar.push_back("tree_p20");
981     keyvgpar.push_back("tree_p30");
982     keyvgpar.push_back("tree_p40");
983     keyvgpar.push_back("tree_p50");
984     keyvgpar.push_back("tree_p60");
985     keyvgpar.push_back("tree_p70");
986     keyvgpar.push_back("tree_p80");
987     keyvgpar.push_back("tree_p90");
988     keyvgpar.push_back("tree_qmean");
989     keyvgpar.push_back("tree_mean");
990     keyvgpar.push_back("tree_refl_kurt");
991     keyvgpar.push_back("tree_reflt_mean");
992     keyvgpar.push_back("tree_refl_skew");
993     keyvgpar.push_back("tree_refl_stdev");
994     keyvgpar.push_back("tree_rugosity");
995     keyvgpar.push_back("tree_skew");
996     keyvgpar.push_back("tree_stdev");
997     keyvgpar.push_back("tree_vdr");
998     keyvgpar.push_back("refl_max");
999     keyvgpar.push_back("ground_refl_max");
1000    keyvgpar.push_back("tree_refl_max");
1001    keyvgpar.push_back("all_p100");
1002    keyvgpar.push_back("shrub_refl_max");
1003    keyvgpar.push_back("tree_p100");
1004
1005    if(par.getString("namconv")[0]!='y' && par.
          ↪ getString("namconv")[0]!='Y')
1006    {
```

```
1007
1008            for (unsigned int vg = 0; vg < keyvgpar.size()
                    ↪ ; ++vg)
1009            {
1010                string key = keyvgpar.at(vg);
1011                if(par.getString(key)[0] != '0')
1012                {
1013                    filevect.push_back(par.getString(key))
                            ↪ ;
1014                    metricvect.push_back(vg);
1015                }
1016            }
1017        } else {
1018            if (par.getString("AKnam")[0]=='y' || par.
                    ↪ getString("AKname")[0]=='Y')
1019        {
1020
1021
1022            if(par.getString("mapNam")[0] == 'y')
1023            {
1024                string metDirect = par.getString("mnDir");
1025                if(metDirect[metDirect.length() - 1] != '/
                        ↪ ') metDirect += '/';
1026                for (unsigned int vg = 0; vg < keyvgpar.
                        ↪ size(); ++vg)
1027                {
1028                    if(par.getNum(keyvgpar.at(vg)))
1029                    {
1030                        string metFile = metDirect +
                                ↪ keyvgpar.at(vg) + '/' + "
                                ↪ mosaic.tif";
1031
1032                        filevect.push_back(metFile);
1033                        metricvect.push_back(vg);
1034
1035                    }
1036                }
1037            } else {
1038            string metDirect = par.getString("AKmetricsDir
                    ↪ ");
1039            if(metDirect[metDirect.length() - 1] != '/')
                    ↪ metDirect += '/';
1040            for (unsigned int vg = 0; vg < keyvgpar.size()
                    ↪ ; ++vg)
1041            {
1042                if(par.getNum(keyvgpar.at(vg)))
1043                {
1044                    string metFile = metDirect + keyvgpar.
                            ↪ at(vg) + '/' + seg + '_' +
                            ↪ keyvgpar.at(vg) + ".tif";
```

```
1045
1046                filevect.push_back(metFile);
1047                metricvect.push_back(vg);
1048
1049            }
1050        }
1051        }
1052    } else {
1053        string metDirect = par.getString("metricsDir")
            ↪ ;
1054        if(metDirect[metDirect.length() - 1] != '/')
            ↪ metDirect += '/';
1055        for (unsigned int vg = 0; vg < keyvgpar.size()
            ↪ ; ++vg)
1056        {
1057            if(par.getNum(keyvgpar.at(vg)))
1058            {
1059                string metFile = metDirect + ac + '_'
                    ↪ + keyvgpar.at(vg) + ".tif";
1060
1061                filevect.push_back(metFile);
1062                metricvect.push_back(vg);
1063
1064            }
1065
1066        }
1067    }
1068    }
1069
1070    for(unsigned int ss = 0; ss < filevect.size(); ++
        ↪ ss)
1071    {
1072        string tifFile = filevect.at(ss);
1073
1074        int metricNum = metricvect.at(ss);
1075
1076        GDALDataset  *mDataset;
1077
1078        GDALAllRegister();
1079
1080
1081        mDataset = (GDALDataset *) GDALOpen( tifFile.
            ↪ c_str(), GA_ReadOnly );
1082
1083        double        adfGeoTransform[6];
1084        int         nXBlockSize, nYBlockSize;
1085
1086        GDALRasterBand  *mBand;
1087        mBand = mDataset->GetRasterBand( 1 );
1088
```

```
1089
1090
1091        float      *mData;
1092
1093        mBand->GetBlockSize( &nXBlockSize, &
               ↪ nYBlockSize );
1094
1095        mData = (float*) CPLMalloc(sizeof(float)*
               ↪ nXBlockSize * nYBlockSize);
1096
1097        mDataset->GetGeoTransform( adfGeoTransform ) ;
1098        for(unsigned int cc = 0; cc < voxColumnsVector
               ↪ .size(); ++cc)
1099        {
1100
1101            double xx = voxColumnsVector.at(cc).xC;
1102            double yy = voxColumnsVector.at(cc).yC;
1103
1104            int xOffset, yOffset;
1105
1106            xOffset = static_cast <int> ((xx -
                   ↪ adfGeoTransform[0]) /
                   ↪ adfGeoTransform[1]);
1107            yOffset = static_cast <int> ((yy -
                   ↪ adfGeoTransform[3]) /
                   ↪ adfGeoTransform[5]);
1108
1109            mBand->RasterIO( GF_Read, xOffset, yOffset
                   ↪ , 1, 1,
1110                             mData, 1, 1, GDT_Float32
                                   ↪ ,
1111                             0, 0 );
1112
1113
1114           if(!isnan(*mData)) {
1115            voxColumnsVector.at(cc).inRaster = true;
1116            }
1117
1118            switch (metricNum)
1119            {
1120                case 0: voxColumnsVector.at(cc).d0 = *
                       ↪ mData;
1121                    break;
1122                case 1: voxColumnsVector.at(cc).d1 = *
                       ↪ mData;
1123                    break;
1124                case 2: voxColumnsVector.at(cc).d2 = *
                       ↪ mData;
1125                    break;
```

```
1126                    case 3: voxColumnsVector.at(cc).d3 = *
                           ↪ mData;
1127                     break;
1128                    case 4: voxColumnsVector.at(cc).d4 = *
                           ↪ mData;
1129                     break;
1130                    case 5: voxColumnsVector.at(cc).d5 = *
                           ↪ mData;
1131                     break;
1132                    case 6: voxColumnsVector.at(cc).d6 = *
                           ↪ mData;
1133                     break;
1134                    case 7: voxColumnsVector.at(cc).d7 = *
                           ↪ mData;
1135                     break;
1136                    case 8: voxColumnsVector.at(cc).d8 = *
                           ↪ mData;
1137                     break;
1138                    case 9: voxColumnsVector.at(cc).d9 = *
                           ↪ mData;
1139                     break;
1140                    case 10: voxColumnsVector.at(cc).kurt
                           ↪ = *mData;
1141                     break;
1142                    case 11: voxColumnsVector.at(cc).mean
                           ↪ = *mData;
1143                     break;
1144                    case 12: voxColumnsVector.at(cc).p10 =
                           ↪  *mData;
1145                     break;
1146                    case 13: voxColumnsVector.at(cc).p20 =
                           ↪  *mData;
1147                     break;
1148                    case 14: voxColumnsVector.at(cc).p30 =
                           ↪  *mData;
1149                     break;
1150                    case 15: voxColumnsVector.at(cc).p40 =
                           ↪  *mData;
1151                     break;
1152                    case 16: voxColumnsVector.at(cc).p50 =
                           ↪  *mData;
1153                     break;
1154                    case 17: voxColumnsVector.at(cc).p60 =
                           ↪  *mData;
1155                     break;
1156                    case 18: voxColumnsVector.at(cc).p70 =
                           ↪  *mData;
1157                     break;
1158                    case 19: voxColumnsVector.at(cc).p80 =
                           ↪  *mData;
```

```
1159                    break;
1160                case 20: voxColumnsVector.at(cc).p90 =
                        ↪  *mData;
1161                    break;
1162                case 21: voxColumnsVector.at(cc).qmean
                        ↪  = *mData;
1163                    break;
1164                case 22: voxColumnsVector.at(cc).
                        ↪ refl_kurt = *mData;
1165                    break;
1166                case 23: voxColumnsVector.at(cc).
                        ↪ refl_mean = *mData;
1167                    break;
1168                case 24: voxColumnsVector.at(cc).
                        ↪ refl_skew = *mData;
1169                    break;
1170                case 25: voxColumnsVector.at(cc).
                        ↪ refl_stdev = *mData;
1171                    break;
1172                case 26: voxColumnsVector.at(cc).skew
                        ↪ = *mData;
1173                    break;
1174                case 27: voxColumnsVector.at(cc).stdev
                        ↪  = *mData;
1175                    break;
1176                case 28: voxColumnsVector.at(cc).
                        ↪ ground_aspect = *mData;
1177                    break;
1178                case 29: voxColumnsVector.at(cc).
                        ↪ ground_elev_mean = *mData;
1179                    break;
1180                case 30: voxColumnsVector.at(cc).
                        ↪ ground_refl_kurt = *mData;
1181                    break;
1182                case 31: voxColumnsVector.at(cc).
                        ↪ ground_refl_mean = *mData;
1183                    break;
1184                case 32: voxColumnsVector.at(cc).
                        ↪ ground_refl_skew = *mData;
1185                    break;
1186                case 33: voxColumnsVector.at(cc).
                        ↪ ground_refl_stdev = *mData;
1187                    break;
1188                case 34: voxColumnsVector.at(cc).
                        ↪ ground_slope = *mData;
1189                    break;
1190                case 35: voxColumnsVector.at(cc).
                        ↪ pulse_density = *mData;
1191                    break;
```

```
1192                    case 36: voxColumnsVector.at(cc).
                           ↪ pulse_scan_angle = *mData;
1193                     break;
1194                    case 37: voxColumnsVector.at(cc).
                           ↪ returns_per_pulse = *mData;
1195                     break;
1196                    case 38: voxColumnsVector.at(cc).
                           ↪ shrub_mean = *mData;
1197                     break;
1198                    case 39: voxColumnsVector.at(cc).
                           ↪ shrub_refl_mean = *mData;
1199                     break;
1200                    case 40: voxColumnsVector.at(cc).
                           ↪ shrub_refl_stdev = *mData;
1201                     break;
1202                    case 41: voxColumnsVector.at(cc).
                           ↪ shrub_stdev = *mData;
1203                     break;
1204                    case 42: voxColumnsVector.at(cc).
                           ↪ tree_aad = *mData;
1205                     break;
1206                    case 43: voxColumnsVector.at(cc).
                           ↪ tree_crr = *mData;
1207                     break;
1208                    case 44: voxColumnsVector.at(cc).
                           ↪ tree_d0 = *mData;
1209                     break;
1210                    case 45: voxColumnsVector.at(cc).
                           ↪ tree_d1 = *mData;
1211                     break;
1212                    case 46: voxColumnsVector.at(cc).
                           ↪ tree_d2 = *mData;
1213                     break;
1214                    case 47: voxColumnsVector.at(cc).
                           ↪ tree_d3 = *mData;
1215                     break;
1216                    case 48: voxColumnsVector.at(cc).
                           ↪ tree_d4 = *mData;
1217                     break;
1218                    case 49: voxColumnsVector.at(cc).
                           ↪ tree_d5 = *mData;
1219                     break;
1220                    case 50: voxColumnsVector.at(cc).
                           ↪ tree_d6 = *mData;
1221                     break;
1222                    case 51: voxColumnsVector.at(cc).
                           ↪ tree_d7 = *mData;
1223                     break;
1224                    case 52: voxColumnsVector.at(cc).
                           ↪ tree_d8 = *mData;
```

```
1225                    break;
1226                case 53: voxColumnsVector.at(cc).
                       ↪ tree_d9 = *mData;
1227                    break;
1228                case 54: voxColumnsVector.at(cc).
                       ↪ tree_fcover = *mData;
1229                    break;
1230                case 55: voxColumnsVector.at(cc).
                       ↪ tree_fract_all = *mData;
1231                    break;
1232                case 56: voxColumnsVector.at(cc).
                       ↪ tree_iqr = *mData;
1233                    break;
1234                case 57: voxColumnsVector.at(cc).
                       ↪ tree_kurt = *mData;
1235                    break;
1236                case 58: voxColumnsVector.at(cc).
                       ↪ tree_mad = *mData;
1237                    break;
1238                case 59: voxColumnsVector.at(cc).
                       ↪ tree_mean = *mData;
1239                    break;
1240                case 60: voxColumnsVector.at(cc).
                       ↪ tree_p10 = *mData;
1241                    break;
1242                case 61: voxColumnsVector.at(cc).
                       ↪ tree_p20 = *mData;
1243                    break;
1244                case 62: voxColumnsVector.at(cc).
                       ↪ tree_p30 = *mData;
1245                    break;
1246                case 63: voxColumnsVector.at(cc).
                       ↪ tree_p40 = *mData;
1247                    break;
1248                case 64: voxColumnsVector.at(cc).
                       ↪ tree_p50 = *mData;
1249                    break;
1250                case 65: voxColumnsVector.at(cc).
                       ↪ tree_p60 = *mData;
1251                    break;
1252                case 66: voxColumnsVector.at(cc).
                       ↪ tree_p70 = *mData;
1253                    break;
1254                case 67: voxColumnsVector.at(cc).
                       ↪ tree_p80 = *mData;
1255                    break;
1256                case 68: voxColumnsVector.at(cc).
                       ↪ tree_p90 = *mData;
1257                    break;
```

```
1258                    case 69: voxColumnsVector.at(cc).
                           ↪ tree_qmean = *mData;
1259                     break;
1260                    case 70: voxColumnsVector.at(cc).
                           ↪ tree_mean = *mData;
1261                     break;
1262                    case 71: voxColumnsVector.at(cc).
                           ↪ tree_refl_kurt = *mData;
1263                     break;
1264                    case 72: voxColumnsVector.at(cc).
                           ↪ tree_reflt_mean = *mData;
1265                     break;
1266                    case 73: voxColumnsVector.at(cc).
                           ↪ tree_refl_skew = *mData;
1267                     break;
1268                    case 74: voxColumnsVector.at(cc).
                           ↪ tree_refl_stdev = *mData;
1269                     break;
1270                    case 75: voxColumnsVector.at(cc).
                           ↪ tree_rugosity = *mData;
1271                     break;
1272                    case 76: voxColumnsVector.at(cc).
                           ↪ tree_skew = *mData;
1273                     break;
1274                    case 77: voxColumnsVector.at(cc).
                           ↪ tree_stdev = *mData;
1275                     break;
1276                    case 78: voxColumnsVector.at(cc).
                           ↪ tree_vdr = *mData;
1277                     break;
1278                    case 79: voxColumnsVector.at(cc).
                           ↪ refl_max = *mData;
1279                     break;
1280                    case 80: voxColumnsVector.at(cc).
                           ↪ ground_refl_max = *mData;
1281                     break;
1282                    case 81: voxColumnsVector.at(cc).
                           ↪ tree_refl_max = *mData;
1283                     break;
1284                    case 82: voxColumnsVector.at(cc).p100
                           ↪ = *mData;
1285                     break;
1286                    case 83: voxColumnsVector.at(cc).
                           ↪ shrub_refl_max = *mData;
1287                     break;
1288                    case 84: voxColumnsVector.at(cc).
                           ↪ tree_p100 = *mData;
1289                     break;
1290
1291
```

```
1292                 default:
1293                     break;
1294             }
1295
1296         }
1297
1298
1299         CPLFree(mData);
1300
1301         GDALClose(mDataset);
1302     }
1303
1304     cout << "Done reading metrics.\n";
1305
1306     // ************************************
1307     // End GDAL Stuff
1308     // ************************************
1309
1310
1311
1312
1313
1314
1315     string response;
1316
1317     response = par.getString("lines");
1318
1319     if (response[0] == 'y'){
1320          while(allVoxels.size() > 0)
1321           {
1322               Voxel v = allVoxels.back();
1323
1324
1325               kd_insert3(kdVoxCenters, v.cX, v.cY, v.
                        ↪ cZ, kdVoxCenters);
1326
1327               Point p (v.cX, v.cY, v.cZ);
1328
1329               voxMap[p] = v;
1330
1331               allVoxels.pop_back();
1332
1333           }
1334         intersectVoxels(kdVoxCenters, voxMap, r);
1335         cout << "\n\nCheck for voxel intersection by a
                ↪  line? ";
1336         cin >> response;
1337     }
1338
1339     while (response[0] == 'y'){
```

```
1340            intersectVoxels(kdVoxCenters, voxMap, r);
1341            cout << "\n\nCheck for voxel intersection by a
                    ↪  line? ";
1342            cin >> response;
1343        }
1344
1345
1346
1347        string outFile;
1348
1349
1350
1351        // ************************************
1352        // Set up end of file names (based on id)
1353        // ************************************
1354
1355
1356        VoxData data;
1357        data.numCols = numVoxCols;
1358        data.numHi = numVox;
1359        data.numPts = kdPointCount;
1360        data.voxcols = voxColumnsVector;
1361
1362
1363
1364        data.acqid = ac;
1365
1366        data.segid = seg;
1367
1368        if(splitNum.compare("") != 0)
1369        {
1370            data.splitid = splitNum;
1371            splitNum = "-" + splitNum;
1372        }else {
1373            data.splitid = "1";
1374        }
1375
1376
1377        string id = "_" + ac + "_" + seg + splitNum;
1378
1379        // ************************************
1380        // VoxData file creation
1381        // ************************************
1382
1383        string vDataFile = par.getString("voxdata");
1384        if(vDataFile[0] == 'y'){
1385            cout << "\n\nWriting VoxData to file...\n";
1386            string vdatadir = par.getString("vdataDir") +
                    ↪ "/" + par.getString("vdataTitle") + id +
                    ↪  ".voxdata";
```

```
1387            data.toFile(vdatadir.c_str());
1388            cout << "Done writing VoxData to file.\n";
1389        }
1390
1391
1392        // ************************************
1393        // R file creation
1394        // ************************************
1395
1396
1397
1398        string outDir = par.getString("outDirectory");
1399
1400        string outName = par.getString("outName");
1401
1402         outFile = outDir + "/" + outName + id + ".R";
1403
1404        response = par.getString("routput");
1405
1406        if (response[0] == 'y') {
1407            cout << "\n\nWriting data to R file...\n";
1408            toR(voxColumnsVector, numVoxCols, totalnumVox,
                    ↪  numVox, outFile);
1409            cout << "Done writing data to R file.\n";
1410        }
1411
1412
1413        // ************************************
1414        // Histogram flat file creation
1415        // ************************************
1416
1417
1418        double zmax = par.getNum("zmax");
1419        int voxelsHi = 0;
1420        while(z2 < zmax)
1421        {
1422            z2 += ht;
1423            ++voxelsHi;
1424        }
1425
1426        cout << "Voxels Above: " << voxelsHi << '\n';
1427        cout << "Voxels Below: " << voxelsLo << '\n';
1428        cout << "Filled Voxels: " << numVox << '\n';
1429
1430        string hff = par.getString("hff");
1431        if(hff[0] == 'y')
1432        {
1433            cout << "\n\nWriting data to histogram flat
                    ↪ file...\n";
```

```
1434          string hffdir = par.getString("hffDir") + "/"
                  ↪ + par.getString("hffTitle") + id;
1435
1436
1437
1438          data.toHistFF(hffdir, voxelsHi, voxelsLo - 1,
                  ↪ numVox);
1439          cout << "Done writing data to histogram flat
                  ↪ file.\n";
1440      }
1441
1442      // ************************************
1443      // Lidar Metrics flat file creation
1444      // ************************************
1445
1446
1447      string lmff = par.getString("lmff");
1448      if(lmff[0] == 'y')
1449      {
1450          cout << "\n\nWriting data to lidar metrics
                  ↪ flat file...\n";
1451          string lmffdir = par.getString("lmffDir") + "/
                  ↪ " + par.getString("lmffTitle") + id;
1452
1453
1454          data.toLidarMetrics(lmffdir);
1455          cout << "Done writing data to lidar metrics
                  ↪ flat file.\n";
1456      }
1457
1458      // ************************************
1459      // Percentiles flat file creation
1460      // ************************************
1461      string pff = par.getString("pff");
1462      if(pff[0] == 'y')
1463      {
1464          string pffdir = par.getString("pffDir") + "/"
                  ↪ + par.getString("pffTitle") + id;
1465          data.toPercMetrics(pffdir.c_str());
1466      }
1467
1468  //    vector<VoxCol>::iterator q;
1469  //    for(q = voxColumnsVector.begin(); q !=
      ↪ voxColumnsVector.end(); ++q)
1470  //    {
1471  //        delete &q;
1472  //    }
1473  //    voxColumnsVector.clear();
1474  //
1475  //    data.clearData();
```

```
1476
1477      ifs.close();
1478
1479
1480      }
1481
1482
1483      time_t timerF;
1484
1485      time(&timerF);
1486
1487      cout << "\n\nTotal Time Elapsed: " << timerF -
             ↪ timerI;
1488      cout << '\n' << '\n';
1489      cout << '\a';
1490
1491
1492    if (argc > 3) std::cout.rdbuf(cout_sbuf); // restore
        ↪  the original stream buffer
1493    return 0;
1494  };
```

7.2 Flat File Format Descriptions

```
*****************************************

VoxGen Flat File Formats

*****************************************

Lidar Metric File Format

VoxGen outputs a lidar metric flat file in the
        following format by row per voxel column
        (if there is no file containing the metric,
        a default value of -99 is used):

x coordinate vox column center, y coordinate vox
        column center, acquisition id, segment id,
        pulse_density, pulse_scan_angle,
        returns_per_pulse, mean, qmean, stdev,
        skew, kurt, p10, p20, p30, p40, p50, p60,
        p70, p80, p90, p100, d0, d1, d2, d3, d4,
        d5, d6, d7, d8, d9, refl_max, shrub_mean,
        shrub_stdev, shrub_refl_max, tree_mean,
        tree_qmean, tree_stdev, tree_rugosity,
        tree_skew, tree_kurt, tree_fcover,
        tree_fract_al, tree_p10, tree_p20, tree_p30,
        tree_p40, tree_p50,tree_p60, tree_p70, tree_p80,
```

```
        tree_p90, tree_p100, tree_d0, tree_d1, tree_d2,
        tree_d3, tree_d4, tree_d5, tree_d6, tree_d7,
        tree_d8, tree_d9, tree_iqr, tree_vdr, tree_mad,
        tree_aad, tree_crr, tree_refl_max,
        ground_elev_mean, ground_slope, ground_aspect,
        ground_refl_max


*******************************************

Histogram File Format

VoxGen outputs a histogram flat file in the following
        format by row per voxel column:

x coordinate vox column center, y coordinate vox column
        center, acquisition id, segment id,
        bin1 number of points, ..., binN number of points
```

References

[1] *Boost Background Information*. boost. http://www.boost.org/users/. 2005.

[2] *GDAL API Tutorial*. GDAL. http://www.gdal.org/gdal_tutorial.html. 20 November 2013.

[3] *GDAL - Geospatial Data Abstraction Library* . GDAL. http://www.gdal.org/.

[4] *kdtree: A simple C library for working with KD-Trees*. Google Code. https://code.google.com/p/kdtree/. Nov. 2011.

[5] *LAS Specification*. asprs. The American Society for Photogrammetry & Remote Sensing. http://www.asprs.org/a/society/committees/standards/LAS_1_4_r13.pdf. 15 July 2013.

[6] *libLAS API Reference Documentation*. libLAS. http://www.liblas.org/doxygen/index.html.

[7] Loskot, Mateusz & Butler, Howard. *C++ Tutorial*. libLAS. http://www.liblas.org/tutorial/cpp.html. 22 February 2011.

[8] Loskot, Mateusz & Butler, Howard. *libLAS*. libLAS. http://www.liblas.org/index.html. 22 July 2014.

[9] R Core Team. *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org/. 2014.

[10] *Simple ls Program*. boost. http://www.boost.org/doc/libs/1_31_0/libs/filesystem/example/. 2002.