

The Lidar FXYZ Convertor Tool

Scott D. Brown
Digital Imaging and Remote Sensing Laboratory
Rochester Institute of Technology
Rochester, NY 14623

May 23, 2006

Contents

1 Overview	1
1.1 Document Version	1
2 Custom Data Types	2
2.1 Task Header Data	2
2.2 Pulse Header Data	2
3 Source file	3
4 Revision History	11

1 Overview

This document describes a program called `fxyz_convertor`. The purpose of this program is to convert the DIRSIG4 LIDAR simulation dataset into a common FXYZ file used for 3D visualization.

1.1 Document Version

The following chunk stores the current document version. This is accessible through the class namespace so that the programmer can set up query mechanisms from the application to track the versions used in a specific build.

```
<Document version>≡  
const char * version = "$Revision: 1.9 $";
```

2 Custom Data Types

2.1 Task Header Data

This structure matches the `TaskDataHeader` defined in the `CDLidarCapture` method and allows us to easily read the header at the start of a task data file which may contain a one or more pulses.

```

(Data types)≡
typedef struct _TaskDataHeader {
    char            idString[ 64 ];
    unsigned int    endianFlag;
    char            generationDate[ 32 ];
    double          focalLength;
    double          xPixelPitch;
    double          yPixelPitch;
    unsigned int    xPixelCount;
    unsigned int    yPixelCount;
    double          pulseRate;
    double          pulseDuration;
    double          pulsePower;
    double          spectralPeak;
    double          spectralWidth;
    double          timeGateStart;
    double          timeGateStop;
    double          timeGateDelta;
    unsigned int    timeBinCount;
    unsigned int    pulseCount;
} TaskDataHeader;

```

2.2 Pulse Header Data

This structure matches the `PulseDataHeader` defined in the `CDLidarCaptureMethod` class and can be used to read the header data at the start of each pulse.

```

(Data types)+≡
typedef struct _PulseDataHeader {
    double          pulseTime;
    double          xPlatformPosition;
    double          yPlatformPosition;
    double          zPlatformPosition;
    double          xPlatformAngle;
    double          yPlatformAngle;
    double          zPlatformAngle;
    double          alongTrackAngle;
    double          acrossTrackAngle;
} PulseDataHeader;

```

3 Source file

The following target collects the contents of the `fxyz_convertor.cpp` source file.

```

<fxyz_convertor.cpp>≡
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include "version.cpp"

#include "CDPoint.h"
#include "CDVector.h"
#include "CDMatrix.h"

<Data types>

void printUsage( void )
{
    std::cerr
        << "usage: fxyz_convertor"
        << " --threshold=<value>"
        << " --input_file=<filename>"
        << " --output_file=<filename>"
        << std::endl;
    return;
}

double degreesToRadians( double angle )
{
    return M_PI / 180.0 * angle;
}

int
main( int argc, char * argv[] )
{
    const double speedOfLight = 2.99792458e+08;

    std::cout << "DIRSIG: LIDAR FXYZ Conversion Tool" << std::endl
        << "Version " << version << std::endl
        << "Build Date: " << __DATE__ << " " << __TIME__

```

```
<< std::endl << std::endl;

// check the command line
if( argc != 4 ) {
    printUsage();
    return 1;
}

// extract the data filename to a string
double threshold;
std::string inFilename, outFilename;
std::string argument, option;
for( int ii = 1; ii < argc; ii++ ) {

    // get the argument
    argument = argv[ ii ];

    // search for the "="
    std::string::size_type idx = argument.find( '=' );

    // check if the "=" was missing
    if( idx == std::string::npos ) {
        printUsage();
        return 1;
    }

    // extract the option name
    option = argument.substr( 0, idx );

    if( option == "--threshold" ) {
        threshold = atof( argument.substr( idx+1 ).c_str() );
    }
    else if( option == "--input_file" ) {
        inFilename = argument.substr( idx+1 );
    }
    else if( option == "--output_file" ) {
        outFilename = argument.substr( idx+1 );
    }
    else {
        printUsage();
        return 1;
    }
}

// open the input file with a file descriptor
int inFd = open( inFilename.c_str(), O_RDONLY );
```

```
if( inFd < 0 ) {
    std::cerr << "Could not open input data file!" << std::endl
    << "    Filename = " << inFilename << std::endl;
    return 1;
}

// open the output file as a file stream
std::ofstream outFile( outFilename.c_str(), std::ios::out );
if( outFile.fail() ) {
    std::cerr << "Could not open output file!" << std::endl
    << "    Filename = " << outFilename << std::endl;
    return 1;
}
outFile << std::fixed << std::showpoint << std::setprecision( 4 );

// read in the task header
TaskDataHeader taskHeader;
ssize_t readSize;
read( inFd, ( void * )&taskHeader.idString,
      sizeof( taskHeader.idString ) );
read( inFd, ( void * )&taskHeader.endianFlag,
      sizeof( taskHeader.endianFlag ) );
read( inFd, ( void * )&taskHeader.generationDate,
      sizeof( taskHeader.generationDate ) );
read( inFd, ( void * )&taskHeader.focallLength,
      sizeof( taskHeader.focallLength ) );
read( inFd, ( void * )&taskHeader.xPixelPitch,
      sizeof( taskHeader.xPixelPitch ) );
read( inFd, ( void * )&taskHeader.yPixelPitch,
      sizeof( taskHeader.yPixelPitch ) );
read( inFd, ( void * )&taskHeader.xPixelCount,
      sizeof( taskHeader.xPixelCount ) );
read( inFd, ( void * )&taskHeader.yPixelCount,
      sizeof( taskHeader.yPixelCount ) );
read( inFd, ( void * )&taskHeader.pulseRate,
      sizeof( taskHeader.pulseRate ) );
read( inFd, ( void * )&taskHeader.pulseDuration,
      sizeof( taskHeader.pulseDuration ) );
read( inFd, ( void * )&taskHeader.pulsePower,
      sizeof( taskHeader.pulsePower ) );
read( inFd, ( void * )&taskHeader.spectralPeak,
      sizeof( taskHeader.spectralPeak ) );
read( inFd, ( void * )&taskHeader.spectralWidth,
      sizeof( taskHeader.spectralWidth ) );
read( inFd, ( void * )&taskHeader.timeGateStart,
      sizeof( taskHeader.timeGateStart ) );
```

```
read( inFd, ( void * )&taskHeader.timeGateStop,
      sizeof( taskHeader.timeGateStop ) );
read( inFd, ( void * )&taskHeader.timeGateDelta,
      sizeof( taskHeader.timeGateDelta ) );
read( inFd, ( void * )&taskHeader.timeBinCount,
      sizeof( taskHeader.timeBinCount ) );
readSize = read( inFd, ( void * )&taskHeader.pulseCount,
                sizeof( taskHeader.pulseCount ) );

// check if these reads have been failing
if( readSize <= 0 ) {
    std::cerr << "Could not read task header!" << std::endl
              << "    Filename = " << inFilename << std::endl;
    return 1;
}

// check if the ID string looks good
if( strncmp( taskHeader.idString, "DIRSIG Lidar Capture", 20 ) != 0 ) {
    std::cerr << "Not a file created by the 'lidar' capture method!"
              << std::endl
              << "    Filename = " << inFilename << std::endl;
    return 1;
}

// compute the size of each pulse data block
unsigned int pulseDataCounts =
    taskHeader.xPixelCount * taskHeader.yPixelCount
    * taskHeader.timeBinCount;
ssize_t pulseDataSize = pulseDataCounts * sizeof( double );

// allocate a buffer for the pulse data
double * pulseData = new double[ pulseDataCounts ];
if( pulseData == 0 ) {
    std::cerr << "Could not allocate buffer for pulse data!" << std::endl;
    return 1;
}

// create a point to hold the platform location
CDPoint platformLocation;

// create a transform matrix for the platform orientation
CDMatrix platformMatrix;
CDMatrix xRotation;
CDMatrix yRotation;
CDMatrix zRotation;
```

```

// create a transform matrix for the platform relative pointing
CDMatrix mountMatrix;

// talk to the users
std::cout
    << "Input data contains "
    << taskHeader.pulseCount << " pulses" << std::endl;

// walk through all the pulses in the file to compute the pixel means
for( unsigned int pulseIndex = 0; pulseIndex < taskHeader.pulseCount;
    ++pulseIndex ) {

    // talk to the friendly people
    std::cout
        << "Processing pulse #" << pulseIndex + 1 << std::endl;

    // read in the pulse header
    PulseDataHeader pulseHeader;
    read( inFd, ( void * )&pulseHeader.pulseTime,
        sizeof( pulseHeader.pulseTime ) );
    read( inFd, ( void * )&pulseHeader.xPlatformPosition,
        sizeof( pulseHeader.xPlatformPosition ) );
    read( inFd, ( void * )&pulseHeader.yPlatformPosition,
        sizeof( pulseHeader.yPlatformPosition ) );
    read( inFd, ( void * )&pulseHeader.zPlatformPosition,
        sizeof( pulseHeader.zPlatformPosition ) );
    read( inFd, ( void * )&pulseHeader.xPlatformAngle,
        sizeof( pulseHeader.xPlatformAngle ) );
    read( inFd, ( void * )&pulseHeader.yPlatformAngle,
        sizeof( pulseHeader.yPlatformAngle ) );
    read( inFd, ( void * )&pulseHeader.zPlatformAngle,
        sizeof( pulseHeader.zPlatformAngle ) );
    read( inFd, ( void * )&pulseHeader.alongTrackAngle,
        sizeof( pulseHeader.alongTrackAngle ) );
    readSize = read( inFd, ( void * )&pulseHeader.acrossTrackAngle,
        sizeof( pulseHeader.acrossTrackAngle ) );

    // check if these reads have been failing
    if( readSize <= 0 ) {
        std::cerr << "Could not read pulse header!" << std::endl
            << "    Pulse #" << pulseIndex << std::endl
            << "    Filename = " << inFilename << std::endl;
        return 1;
    }
}

// setup the mount transform for all the pixels

```

```

mountMatrix.makeYRotation(
    degreesToRadians( pulseHeader.acrossTrackAngle ));

// setup the platform transform for all the pixels (Z, then Y, then X)
xRotation.makeXRotation(
    degreesToRadians( pulseHeader.xPlatformAngle ));
yRotation.makeYRotation(
    degreesToRadians( pulseHeader.yPlatformAngle ));
zRotation.makeZRotation(
    degreesToRadians( pulseHeader.zPlatformAngle ));
platformMatrix = xRotation * yRotation * zRotation;

// extract the platform location
platformLocation.set(
    pulseHeader.xPlatformPosition,
    pulseHeader.yPlatformPosition,
    pulseHeader.zPlatformPosition );

// read the pulse data
ssize_t readSize = read( inFd, ( void * )pulseData, pulseDataSize );
if( readSize != pulseDataSize ) {
    std::cerr << "Could not read pulse data!" << std::endl
        << "    Pulse #" << pulseIndex << std::endl
        << "    Filename = " << inFilename << std::endl;
    return 1;
}

// walk through each pixel
for( unsigned int yIndex = 0; yIndex < taskHeader.yPixelCount;
    yIndex++ ) {

    // compute the offset to the start of this set of y pixels
    unsigned int yOffset =
        yIndex * taskHeader.xPixelCount * taskHeader.timeBinCount;

    for( unsigned int xIndex = 0; xIndex < taskHeader.xPixelCount;
        xIndex++ ) {

        // compute the offset to the start of this set of x pixels
        unsigned int xOffset = xIndex * taskHeader.timeBinCount;

        // initialize the time of flight
        double timeOfFlight = -1.0;

        // initialize the number of photons received in this pixel
        double photonCounts = 0.0;
    }
}

```

```
// walk through all the times, looking for a hit
for( unsigned int tIndex = 0; tIndex < taskHeader.timeBinCount;
    tIndex++ ) {

    // extract the number of photon counts at this time
    photonCounts += pulseData[ yOffset + xOffset + tIndex ];

    // check if the photon counts exceeds the threshold
    if( photonCounts > threshold ) {

        // record the time of flight
        timeOfFlight = taskHeader.timeGateStart
            + ( tIndex * taskHeader.timeGateDelta );

        // get out of dodge
        break;
    }
}

// check if we hit something
if( timeOfFlight >= 0 ) {

    // compute the pixel's pointing vector from the focal plane
    CDVector pixelVector;
    pixelVector.setX( taskHeader.xPixelPitch
        * ( xIndex - ( taskHeader.xPixelCount / 2.0 ) ));
    pixelVector.setY( taskHeader.yPixelPitch
        * ( yIndex - ( taskHeader.yPixelCount / 2.0 ) ));
    pixelVector.setZ( -taskHeader.focalLength );

    // make the vector unit length
    pixelVector.setUnitLength();

    // translate the time of flight into a range [m]
    double range = timeOfFlight * speedOfLight / 2.0;

    // compute the projected point at range
    CDPoint hitLocation( pixelVector * range );

    // rotate the hit from camera into the platform space
    hitLocation = mountMatrix * hitLocation;

    // rotate the hit from platform space into world space
    hitLocation = platformMatrix * hitLocation;
}
```

```
        // translate the hit by the platform location
        hitLocation += platformLocation;

        // write this puppy out
        outFile
            << hitLocation.getX() << " "
            << hitLocation.getY() << " "
            << hitLocation.getZ() << std::endl;
    }
}

// close the input and output files
close( inFd );
outFile.close();

return 0;
}
```

4 Revision History

<Revision History>≡

Revision 1.9 2006/04/14 12:39:22 sdbpci

Fixed an error in reading the pulse header where the alongTrack angle was not being read in.

Revision 1.8 2006/04/06 13:02:45 sdbpci

Fixed the error message for the check to see if the file was created by the CDLidarCaptureMethod. Before, it printed the name of the output file as being the broken file, not the input file.

Revision 1.7 2006/03/23 16:51:01 sdbpci

The task and pulse header blocks are now read in element by element to avoid struct padding issues on different machines.

Revision 1.6 2006/03/07 15:33:20 sdbpci

Fixed an X & Y axis flip issue in the XYZ data by making the pixel vector point "down" through the focal point rather than "up" to the focal point and then flipping the whole vector (which also flipped X & Y).

Revision 1.5 2006/03/07 14:40:05 sdbpci

Made the CDGeoBase::degreesToRadians() function a small function inside this program in an effort to make this code as stand-alone as possible since we will make the source code available to modify and compile.

Revision 1.4 2006/03/06 20:48:58 sdbpci

Added the code to check the ID string so we know that the file we were given is actually the correct format.

Revision 1.3 2006/02/25 00:01:36 sdbpci

Added the "real" speed of light to this program too.
Yes, we should have this in one central location.

Revision 1.2 2006/01/26 20:47:54 sdbpci

First working version.
Improved the command line interface and fixed a logic error in the computation of the pixel pointing vector.

Revision 1.1 2006/01/25 22:12:21 sdbpci

Introduced this little gem of a program to convert a DIRSIG lidar dataset (currently only ones produced by the "lidar" capture method) into a fxyz formatted file.

At this point, untested but it compiles.