A Leica PTX Reader for VTK

Release 0.00

David Doria

January 31, 2010

Rensselaer Polytechnic Institute, Troy NY

Abstract

Leica LiDAR scanners output points in the PTX file format. It is often useful to convert this format to a standard VTK point cloud or mesh format before processing. We propose a new class, vtkPTXReader, to provide this functionality.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3148]

Contents

1 Introduction

Leica LiDAR scanners output points in the PTX file format. It is often useful to convert this format to a standard VTK point cloud or mesh format before processing. We propose a new class, vtkPTXReader, to provide this functionality.

2 File Format

Leica scanners acquire points in a "strip wise" manner. That is, the points are acquired bottom-to-top, left-to-right. The file containing the scan points produced by the scanner is in a plain text format. The header of the file is 10 lines. It contains the number of columns in the scan (which we will call "Theta points"), the number of points per column ("Phi points") and an 8 line coordinate frame, which is almost always identity. This reader simply skips the 8 lines of the coordinate system.

After the header, every line contains a point's coordinate, the intensity of the return, and the RGB value of the image associated with the point.

An example file is as follows:

```
199
91
0 0 0
1 0 0
0 1 0
0 0 1
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
x1 y1 z1 I1 r1 g1 b1
x2 y2 z2 I2 r2 g2 b2
....
xn yn zn In rn gn bn
```

3 Options

3.1 Defaults

With no options set, the reader simply produces a point cloud containing the point coordinates, along with their associated color and intensity. An example point cloud is shown in Figure 3.1.

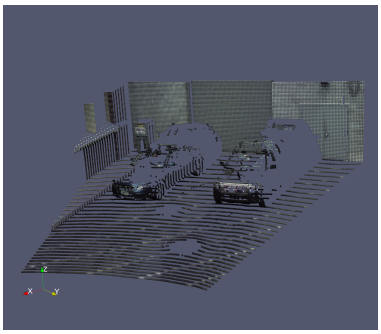

Figure 1: Colored points read using vtkPTXReader and default options

3.2 Triangulation

If the Triangulate variable is set to true, a triangulated mesh will be produced over the scan points. We exploid the grid structure of the scanner to produce this grid. The details are in the Algorithms section below. This method can (and usually does) leave gaps/holes in the surface due to missing/invalid points. A point can be missing if the laser hits a dark surface or reflects off of something such as a piece of glass and does not return to the scanner.

```
reader->SetTriangulate(true);
```

An example mesh is shown in Figure 3.2.

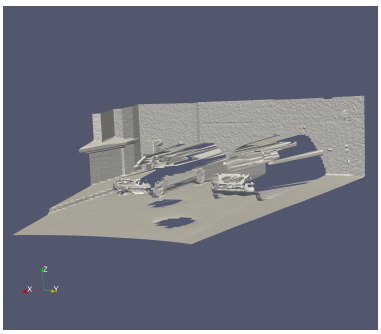

Figure 2: Triangulated mesh read using vtkPTXReader and Triangulate=true

3.3 Span Gaps

If the SpanGaps variable is set to true, the reader will connect the mesh across any missing points. The algorithm is described in the next section.

```
reader->SetSpanGaps(true);
```

An example mesh is shown in Figure 3.3.

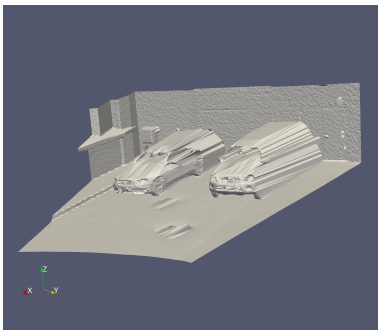

Figure 3: Triangulated mesh read using vtkPTXReader; Triangulate=true, SpanGaps=true

4 Algorithm

4.1 Triangulation

Since the point acquisition order is known, we know the "8-neighborhood" of every point. The trouble lies in the missing points. We construct the mesh in the following way:

- For every point p
- If p has a valid "left", "up", and "up-left" neighbor, create a quad on these four points
- Use the vtkTriangleFilter to convert the resulting set of quads into triangles

4.2 Span Gaps

Rather than compute a surface directly on the 3D points, we abstract the problem to triangulating a set of 2D points in the θ-ϕ coordinate space. We then copy the connectivity information (topology) from the 2D triangulation onto the 3D points. The result is automatically a surface with no holes.

The algorithm is as follows:

- Create a new, 2D, set of points which is simply $(\theta_n, \phi_n, 0)$ of each point p_n. These point coordinates have no physical significance, but their arrangement models the acquisition grid.
- Use vtkDelaunay2D to compute the optimal triangulation on this set of non-physical points.
- Copy the topology from the 2D points to the original 3D points.

5 Code Snippet

```
vtkSmartPointer<vtkPTXReader> reader =
vtkSmartPointer<vtkPTXReader>::New();
reader->SetFileName("file.ptx");
reader->SetTriangulate(true);
```

```
reader->SetSpanGaps(true);
reader->Update();

vtkPolyData* polydata = reader->GetOutput();
```

6 Paraview Plugin

For convenience, this reader is shipped with a Paraview reader plugin. Two checkboxes are provided to set the boolean options of the reader. A screenshot is shown in Figure 6.

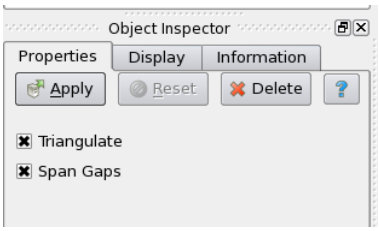

Figure 4: Paraview plugin screenshot