Boost's Douglas-Peucker Contour Simplification Wrapper for ITK and VTK

Release 0.00

David Doria

July 29, 2011

Rensselaer Polytechnic Institute

Abstract

This document presents wrappers (ITK and VTK) of the Boost Geometry Library algorithm to find a low edge-count approximation of a complex, discrete, 2D contour. The contour can be open or closed. The code is available here: `https://github.com/daviddoria/DouglasPeuckerPolylineSimplification`

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3302]

Contents

1 Introduction

This document presents ITK and VTK wrappers to an algorithm to find a low edge-count approximation of a complex, discrete, 2D contour. Our goal is to represent the outline of an object in a simple fashion.

2 Algorithm

The algorithm is explained very well here: http://www.softsurfer.com/Archive/algorithm_0205/algorithm_02

We include excerpts below to make this document self contained. The algorithm consists of two steps, vertex reduction and then Douglas-Peucker simplification.

2.1 Vertex Reduction

"Vertex reduction is the brute-force algorithm for polyline simplification. For this algorithm, a polyline vertex is discarded when its distance from a prior initial vertex is less than some minimum tolerance $\varepsilon > 0$. Specifically, after fixing an initial vertex V_0, successive vertices V_i are tested and rejected if they are less than ε away from V_0. But, when a vertex is found that is further away than ε, then it is accepted as part of the new simplified polyline, and it also becomes the new initial vertex for further simplification of the polyline."

2.2 Douglas-Peucker Simplification

"Whereas vertex reduction uses closeness of vertices as a rejection criterion, the Douglas-Peucker (DP) algorithm uses the closeness of a vertex to an edge segment. This algorithm works from the top down by starting with a crude initial guess at a simplified polyline, namely the single edge joining the first and last vertices of the polyline. Then the remaining vertices are tested for closeness to that edge. If there are vertices further than a specified tolerance, $\varepsilon > 0$, away from the edge, then the vertex furthest from it is added the simplification. This creates a new guess for the simplified polyline. Using recursion, this process continues for each edge of the current guess until all vertices of the original polyline are within tolerance of the simplification."

3 Parameters

The only parameter to the algorithm is the "approximation tolerance", which is a distance value that determines how simplified the contour becomes. The larger the tolerance, the more simplification will take place.

4 VTK Wrapper

The VTK representation of a contour is a `vtkPolyData` whose CellArray contains a list of ordered vtkLine objects. We therefore provide a wrapper with signature:

```
void SimplifyPolyline( vtkPolyData* input, float tolerance, vtkPolyData* output)
```

4.1 ITK Wrapper

The ITK representation of a contour is a `itk::PolyLineParametricPath< 2 >`. We therefore provide a wrapper with signature:

```
void SimplifyPolyline( itk::PolyLineParametricPath< 2 >::Pointer input,
                       float tolerance,
                       itk::PolyLineParametricPath< 2 >::Pointer output);
```

5 Demonstration

A typical case of the simplification of a contour is shown in Figure 1.

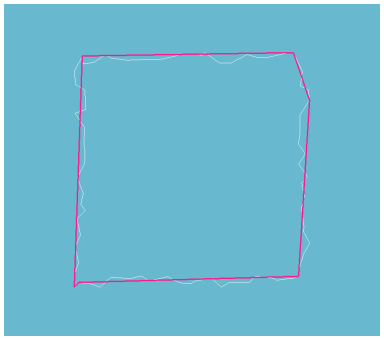

Figure 1: A demonstration of the simplification algorithm. The original contour is shown in white, where the simplified contour (tolerance = 1.0) is shown in red.