A Conditional Mesh Front Iterator for VTK

Release 0.00

David Doria

July 26, 2011

Rensselaer Polytechnic Institute

Abstract

Region growing is a technique that can be used to propagate information over a mesh. In a previous submission, “A Mesh Front Iterator for VTK”, we introduced an iterator that can be used with *vtkPointSet* subclasses to traverse a mesh. It is sometimes useful to visit only vertices that are “similar” to their neighbors by some definition. This concept can be used to select many types of common regions, including planar regions or regions with similar color, to name a few. In this paper, we propose an iterator which propagates on a mesh using a condition test which can be easily modified.

Contents

1 Introduction

Region growing is a technique that can be used to propagate information over a mesh. In a previous submission, “A Mesh Front Iterator for VTK” (`http://www.midasjournal.org/browse/publication/724`), we introduced an iterator that can be used with *vtkPointSet* subclasses to traverse a mesh. It is sometimes useful to visit only vertices that are “similar” to their neighbors by some definition. This concept can be used to select many types of common regions, including planar regions (similar normals) or regions with similar color, to name a few. In this paper, we present an iterator which propagates on a mesh to regions which pass one or more conditional tests. We provide several such tests, and provide a simple abstract interface which can be used to implement custom comparison functions.

2 Algorithm

Front propagation is essentially a breadth first search. The initial “front” is the set of immediate neighbors (connected by an edge) of the seed vertex. To propagate the front, we perform the procedure described below.

2.1 Initialization

- Add the seed vertex to a queue.

2.2 Iteration

- Get the vertex in the front of the queue. Set *NextId* to this value.
- Add all of the vertices that are connected to *NextId* and pass all of the condition tests to the back of the queue, unless they have already been visited or are already in the queue.
- Mark *NextId* as visited.
- Return *NextId*.

3 Condition Interface

An abstract *Condition* class can be subclassed to provide any type of vertex comparison function.

The function which must be implemented to determine the comparison function is:

```
virtual bool IsSimilar(const int a, const int b) = 0;
```

where a and b are the ids of two vertices. This function returns true if the comparison function determines that the two vertices are "close enough" to each other with the specified metric.

This comparison will always reference a mesh, which is set with:

```
void SetMesh(vtkPolyData* m) {this->Mesh = m;}
```

We suggest overriding this function in each subclass to ensure that the attributes on which the comparison function operates are present in the supplied mesh.

4 Provided Conditions

We have included several Condition subclasses for common comparison operations.

4.1 NormalCondition

This Condition should be used to detect planar regions, or gradually varying regions of a mesh. The `AcceptableAngle` parameter controls how similar adjacent normals must be to be considered "close enough" to pass the test.

4.2 ColorCondition

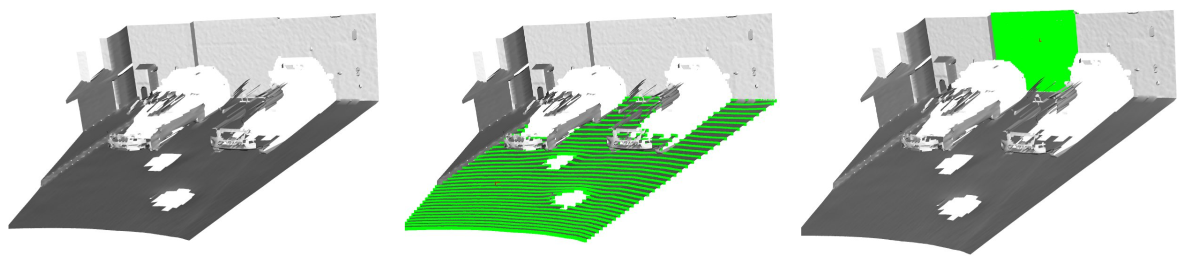
This Condition should be used to regions of a mesh with similar colors. The `AcceptableDifference` parameter controls how similar adjacent colors must be to be considered "close enough" to pass the test.

4.3 DistanceCondition

This Condition should be used to "connected" regions of a mesh. The `AcceptableDistance` parameter controls how close adjacent points must be to be considered "close enough" to pass the test.

5 Demonstration

Figure 1 shows two selections on a typical LiDAR data set. Figure 1(a) shows the input mesh. The green points in Figure 1(b) shows how the ground can be selected by selecting a single seed point (the red point). Figure 1(c) shows how unlike many ground detection algorithms (which rely on the ground plane having a normal of approximately $(0,0,1)$), this region growing approach is much more general and can also be used to select a wall. These examples use the `vtkMeshNormalConditionFrontIterator` with the `NormalCondition`.



(a) Input mesh (b) Selected ground (c) Selected wall

Figure 1: Demonstration of growing regions conditioned on normal similarity.

6 Extracting the region that has been grown

7 Code Snippet

The interface is straight forward:

```
DistanceCondition* condition = new DistanceCondition;
condition->SetAcceptableDistance(1.);

vtkSmartPointer<vtkMeshConditionalFrontIterator> iterator =
  vtkSmartPointer<vtkMeshConditionalFrontIterator>::New();
iterator->SetCondition(condition);
iterator->SetMesh(normalsFilter->GetOutput());
iterator->SetStartVertex(0);
iterator->Initialize();

// Inspect the order of the growing
while(iterator->HasNext())
  {
  vtkIdType nextVertex = iterator->Next();
  std::cout << "Next vertex: " << nextVertex << std::endl;
  }

// Extracted the region that was reached
vtkSmartPointer<vtkPolyData> region = vtkSmartPointer<vtkPolyData>::New();
iterator->GetSelectedRegion(region);

// Mark the region that was reached by adding a membership array to the input mesh
iterator->MarkSelectedRegion();
```

Note that if all of the Conditions are not passed to the iterator before the `vtkMeshConditionalFrontIterator::SetMesh` function is called, that the Conditions will not have access to the mesh unless the user had manually called `SetMesh` on the condition directly.