Lecture 21
ITK’s Path Framework

Methods in Medical Image Analysis - Spring 2015
BioE 2630 (Pitt) : 16-725 (CMU RI)
18-791 (CMU ECE) : 42-735 (CMU BME)
Dr. John Galeotti

Preface

- This is based on the slides I presented at MICCAI 05’s ITK workshop.
- They discuss the motivation and usage for the unified path framework I added to ITK.
- You can see the related Insight Journal article at http://hdl.handle.net/1926/40
  (Note: It used to be one of the top-rated journal articles until I.J. was redone, and all the old reviews were scrapped.)
Introduction

- The need for paths in ITK
- Basic concepts and path types
- Implementation details
- Example usage

The Need for Paths in ITK

- A path is a curve that maps a scalar value to a point in n-dimensional space
The Need for Paths in ITK

- Paths are useful for:
  - Segmentation algorithms
  - Active contours, snakes, LiveWire
  - Ridge tracking
  - Path planning
  - User interaction

- Implementation of the above in ITK can be simplified by having a common, existing path framework.

Unfortunately, the ITK pipeline was originally designed to operate on image and mesh data types
- Neither images nor meshes are well suited for path representation
Basic Concepts and Path Types

- Two common types of paths:
  - Chain codes are a type of discrete curve
  - Parametric curves are continuous
  - Other types of paths are also possible

Chain Codes

- Represent a path as a sequence of offsets between adjoining voxels
- Provide efficient incremental access and comparatively poor random index access
2D Chain Code Example:
Freeman Code

![Diagram of Freeman Code Example]

Parametric Curves

- Represent a path as an algebraically defined curve parameterized over a scalar input
- Provide efficient random access and comparatively poor incremental index access
  - Difficult to know how much to increment the parameterized input to reach the next voxel
Implementation Details

- Necessary Functionality
- Path class hierarchy
- Path iterators
- Path filter hierarchy

Necessary Functionality

- Efficiency
- Handle open, closed, & self-crossing paths
- Iterate along a path through an image
- Examine image data in an arbitrarily defined neighborhood surrounding any given point along a path
Necessary Functionality

- Create and modify arbitrary chain codes
- Smooth paths in continuous index space
- Find exact normals to smooth paths
- Distort paths by orthogonal offsets at regular spacing
- Support user interaction

Path Class Hierarchy

```
DataObject
  Functi<sup>1</sup>onBase<sup>(API)</sup><sub>TInput,TOutput</sub>
  Path<sub>TInput,TOutput,VDimension</sub>
  Path<int,Offset<sub>VDimension</sub>,VDimension>
  ChainCodePath<sub>VDimension</sub>
  ChainCodePath2D
  ParametricPath<sub>VDimension</sub>
  FourierSeriesPath<sub>VDimension</sub>
  OrthogonallyCorrected2DParametricPath
  PolyLineParametricPath<sub>VDimension</sub>
```

Key:
- Abstract Base Class
- Instantiatable Class
PolyLineParametricPath

- Represents a path as a series of vertices connected by line segments
- Provides a simple means of creating a path that can then be converted to other path types

FourierSeriesPath

- Represents a closed path by its Fourier coefficients in each dimension
- Has continuous well-defined derivatives with respect to its input
  - At all points along the curve, the normal direction is well-defined and easy to compute.
Orthogonally Corrected Path

Path Iterators

- Iterators traverse paths through images
  - Allows const paths
  - Necessary for path inputs in pipeline
  - Implemented a universal path iterator

PathConstIterator\(<TImage, TPath>\)
PathIterator\(<TImage, TPath>\)
Path Iterators: Implementation

- Iterators traverse paths through images
  - Paths do not store a current position; iterators do
  - Allows const paths with many concurrent positions
  - The path iterator is able to traverse any type of path
- Path iterators are supported by the Path:
  IncrementInput(InputType & Input) function
  - All paths must know how much to increment a given path input to advance the path output to the next neighboring voxel along the path
  - For efficiency, IncrementInput() returns the offset resulting from its modification of Input

Current Base Class API

- Path<TInput, TOutput, VDimension>
  - virtual InputType StartOfInput() const
  - virtual InputType EndOfInput() const
  - virtual OutputType Evaluate(InputType) const = 0
  - virtual IndexType EvaluateToIndex(InputType) const = 0
  - virtual OffsetType IncrementInput(InputType) const = 0

- PathConstIterator<TImage, TPath>
  - GoToBegin()
  - bool IsAtEnd()
  - operator++()
  - IndexType GetIndex()
  - PathInputType GetPathPosition()
Subclass API Extensions

- `ChainCodePath<VDimension>`
  - `SetStart(IndexType)`
  - `IndexType GetStart() const`
  - `unsigned NumberOfSteps() const`
  - `InsertStep(InputType position, OffsetType step)`
  - `ChangeStep(InputType position, OffsetType step)`
  - `Clear()`

- `ParametricPath<VDimension>`
  - `VectorType EvaluateDerivative(InputType) const`

- `FourierSeriesPath<VDimension>`
  - `AddHarmonic(VectorType CosCoef, VectorType SinCoef)`
  - `Clear()`

Path Filter Hierarchy

```
PathSource<TOutputPath>  ImageSource<TOutputImage>
                         |__________|
                         |         |
PathToPathFilter      PathToImageFilter  ImageToImageFilter
  <TInputPath, TOutputPath>  <TInputPath, TOutputImage>  <TInputImage, TOutputImage>
                        |                  |
PathToChainCodePathFilter ChainCodeToFourierSeriesPathFilter
  <TInputPath, TOutputChainCodePath>  <TInputChainCodePath, TOutputFourierSeriesPath>
                        |                  |
PathAndImageToPathFilter
  <TInputPath, TInputImage, TOutputPath>
                        |                  |
OrthogonalSwath2DPathFilter
  <TFourierSeriesPath, TSwathMeritImage>
                        |                  |
ImageAndPathToImageFilter
  <TInputImage, TInputPath, TOutputImage>
                        |                  |
ExtractOrthogonalSwath2DImageFilter<TImage>
```

Conversion Filters

Path
  ↓
PathToChainCodePathFilter
  ↓
ChainCode

AllowDiagonalSteps (bool=true)

ChainCode
  ↓
ChainCodeToFourierSeriesPathFilter
  ↓
FourierSeriesPath

SetNumHarmonics (int=8)

Philosophical Comparison with Spatial Objects

- Spatial Objects represent geometric shapes (and therefore their associated boundaries)
  - A Spatial Object’s interior is well defined
- Paths represent sequences of connected indices
  - A path may not be closed (no interior defined)
  - A closed path’s interior is difficult to compute
Empirical Comparison with Spatial Objects

- Spatial Objects are well suited to rendering, analysis, and data interchange
- Paths are well suited to computation, optimization, and iterator direction control
- ITK could be extended to enable simple conversion by:
  - Making a Spatial Object that uses one or more paths as an internal representation
  - Making a Path that uses one or more intersecting spatial objects as an internal representation

Example Usage

- Implementation of a published 2D active contour algorithm
  - Finds optimal orthogonal offsets at evenly spaced points along an initial path
  - Requires that neighboring offsets differ in value by at most one
- Added to ITK, including demonstration test code
  - `Modules/Filtering/Path/test/ itkOrthogonalSwath2DPathFilterTest.cxx`
OrthogonalSwath2DPathFilter

PolyLineParametricPath (closed)

PathToChainCodePathFilter

ChainCodeToFourierSeriesPathFilter

ExtractOrthogonalSwath2DImageFilter

DerivativeImageFilter (vertical)

OrthogonalSwath2DPathFilter

OrthogonalCorrected2DParametricPath

Conclusion

- Added user-extensible path support to ITK
  - Data type hierarchy
  - Iterators
  - Filter hierarchy
  - Example implementation in test code
- New core data types can be added to ITK!