DGtal: Digital Geometry Tools and Algorithms Library
1D Geometry

Tristan Roussillon
Objectives

Tools that help in analysing any one-dimensional discrete structures in a generic framework.

Examples in digital geometry

- digital curves
  - 2d, 3d, nd
  - 4-connected, 8-connected, disconnected
  - pixels, interpixels, points
  - open or closed
- chain codes

Constant structures, not mutable
Structures

2 characteristics
- discrete
- one-dimensional

2 notions
- element
- local order (next and previous element)
Iterators

Iterator

- `operator*` (to get the element)
- `operator++`, `operator--` (to point to the next and previous element)

Reachability

An iterator `j` is reachable from an iterator `i` if and only if `i` can be made equal to `j` with finitely many applications of the operator `operator++`.

Range

If `j` is reachable from `i`, one can iterate over the range of elements bounded by `i` and `j`, from the one pointed to by `i` and up to but not including the one pointed to by `j`. Such a range is valid and is denoted by `[i,j)`. 
Open/Linear structures

Classic iterator

- past-the-end value
- \([\text{begin}, \text{end})\) is the whole range
- \([i, j)\) is not always valid
- \([i, i)\) is the empty range
Closed/Circular structures

CGAL circular iterator (circulator)

- no past-the-end value
- \([i, j]\) is always valid
- \([i, i]\) is the whole range
- As long as \(i \neq j\), the range \([i, j]\) behaves like a classic iterator range.
Scanning backward

Reverse iterator

A reverse iterator is an adaptor for scanning backward. The operator++ of the adaptor calls the operator– of the underlying (circular) iterator and conversely. You can use the STL reverse iterator.

Tricky part

Operator* of the adaptor calls operator-- of the underlying (circular) iterator before calling its operator*.
GridCurve is an (open or closed) n-dimensional oriented grid curve. It stores a list of alternated (signed) 0-cells and 1-cells.
GridCurve

Ranges

GridCurve provides many ranges as nested types to iterate over different kinds of elements:

- nd
  - SCellsRange
  - PointsRange
  - MidPointsRange
  - ArrowsRange

- 2d TODO
  - InnerPointsRange
  - OuterPointsRange
  - IncidentPointsRange
  - CodesRange
GridCurve

Ranges

GridCurve provides many ranges as nested types to iterate over different kinds of elements:

- nd
  - SCellsRange
  - PointsRange
  - MidPointsRange
  - ArrowsRange

- 2d TODO
  - InnerPointsRange
  - OuterPointsRange
  - IncidentPointsRange
  - CodesRange
GridCurve

Ranges

GridCurve provides many ranges as nested types to iterate over different kinds of elements:

- nd
  - SCellsRange
  - PointsRange
  - MidPointsRange
  - ArrowsRange

- 2d TODO
  - InnerPointsRange
  - OuterPointsRange
  - IncidentPointsRange
  - CodesRange
GridCurve

Ranges

GridCurve provides many ranges as nested types to iterate over different kinds of elements:

- nd
  - SCellsRange
  - PointsRange
  - MidPointsRange
  - ArrowsRange

- 2d TODO
  - InnerPointsRange
  - OuterPointsRange
  - IncidentPointsRange
  - CodesRange
GridCurve

Ranges

GridCurve provides many ranges as nested types to iterate over different kinds of elements:

- nd
  - SCellsRange
  - PointsRange
  - MidPointsRange
  - ArrowsRange

- 2d TODO
  - InnerPointsRange
  - OuterPointsRange
  - IncidentPointsRange
  - CodesRange
GridCurve

Ranges

GridCurve provides many ranges as nested types to iterate over different kinds of elements:

- **nd**
  - SCellsRange
  - PointsRange
  - MidPointsRange
  - ArrowsRange

- **2d TODO**
  - InnerPointsRange
  - OuterPointsRange
  - IncidentPointsRange
  - CodesRange
GridCurve

Ranges

GridCurve provides many ranges as nested types to iterate over different kinds of elements:

- nd
  - SCellsRange
  - PointsRange
  - MidPointsRange
  - ArrowsRange

- 2d TODO
  - InnerPointsRange
  - OuterPointsRange
  - IncidentPointsRange
  - CodesRange
GridCurve

Ranges

GridCurve provides many ranges as nested types to iterate over different kinds of elements:

- nd
  - SCellsRange
  - PointsRange
  - MidPointsRange
  - ArrowsRange

- 2d TODO
  - InnerPointsRange
  - OuterPointsRange
  - IncidentPointsRange
  - CodesRange
FreemanChain

FreemanChain is 2-dimensional and 4-connected digital curve stored as a string of codes 0,1,2,3. As GridCurve, it provides a CodesRange.

Conversion between FreemanChain and GridCurve

TODO
A segment is a valid and not empty range. The concept CSegment is such that:

**Types**
- Self
- ConstIterator

**Methods**
- begin()
- end()
Class of segments

A class of segments can be defined from a valid property $P$. $P$ is valid iff $P$ is true for any range of only one element and for any not empty range of any segment.

Examples

- to be a DSS
- to be a balanced word
- $x$ to contain at least $k$ elements ($k > 1$)
Segment computer

Detection problem

Deciding whether a given segment belongs to a class of segments defined from a valid property P or not. If P is valid, the detection of a segment can be performed in an incremental way: a segment is initialized at a starting element and then can be extended to the neighbors elements if the property P still holds.

Segment computer

Segment that can control its own extension (so that the property P remains true)
The CTrivialSegmentComputer class is a refinement of the CSegment class that provides additional methods:

- **void init (const ConstIterator& it)**: Set the segment to the element pointed to by it.
- **bool isExtendable()**: Return 'true' if the segment can be extended to the element pointed to by `end()` and 'false' otherwise (no extension is performed).
- **bool extend()**: Return 'true' and extend the segment to the element pointed to by `end()` if it is possible, return 'false' and does not extend the segment otherwise.

### Detection of a segment
```
// s is a segment computer
// [begin, end) is a range
s.init(beg);  
while ((s.end() != end) && (s.extend())) {}  
```

### Avoiding infinite loops with circulators
```
// s is a segment computer
// c is a circulator
s.init(c);  
while ((s.end() != s.begin()) && (s.extend())) {}  
```
List of segment computers

- ArithmeticalDSS
- ArithmeticalDSS3d
- CombinatorialDSS
- GeometricalDSS
- GeometricalDCA
- ThickSegment
- ConvexPart
- ...
- other based on linear programming
Useful functions

The code can be different if an iterator or a circulator is used as the nested ConstIterator type. Moreover, some tasks can be made faster for a given kind of segment computer than for another kind of segment computer. That’s why many generic functions are provided in SegmentComputerUtils.h:

- maximalExtension, oppositeEndMaximalExtension, maximalSymmetricExtension,
- maximalRetraction, oppositeEndMaximalRetraction,
- longestSegment (init the segment computer),
- firstMaximalSegment, lastMaximalSegment, mostCenteredMaximalSegment,
- previousMaximalSegment, nextMaximalSegment,
### Segmentation

**Definition**

A given range contains a finite set of segments verifying a valid property $P$. A segmentation is a subset of the whole set of segments, such that:

1. each element of the range belongs to a segment of the subset
2. no segment contains another segment of the subset

Due to (2), the segments of a segmentation can be ordered without ambiguity (according to the relative position of their first element for instance).

**Types**

**SegmentComputerIterator**

- dereference operator: return an instance of a segment computer.
- `intersectPrevious()`, `intersectNext()`: return ‘true’ if the current segment intersects, respectively, the previous and the next one (when they exist), ‘false’ otherwise.

**Methods**

`init` method taking as input parameters:

- `begin/end (circular)iterators` of the range to be segmented
- an instance of segment computer
Greedy segmentation

```cpp
// types definition
typedef PointVector<2, int> Point;
typedef std::vector<Point> Range;
typedef Range::const_iterator ConstIterator;
typedef ArithmeticalDSS<ConstIterator, int, 8> SegmentComputer;
typedef GreedySegmentation<SegmentComputer> Segmentation;

Range curve;
... // create curve

// Segmentation
SegmentComputer recognitionAlgorithm;
Segmentation theSegmentation(curve.begin(), curve.end(), recognitionAlgorithm);

Segmentation::SegmentComputerIterator i = theSegmentation.begin();
Segmentation::SegmentComputerIterator end = theSegmentation.end();
for ( ; i != end; ++i) {
    SegmentComputer current(*i);
    trace.info() << current << std::endl;  // standard output
}
Greedy segmentation

```cpp
... 
typedef Range::const_reverse_iterator ConstIterator;
...
Segmentation theSegmentation(curve.rbegin(), curve.rend(), recognitionAlgorithm);
... 
```
Saturated segmentation

```cpp
typedef SaturatedSegmentation<SegmentComputer> Segmentation;
```
...
Segmentation of subranges

```cpp
theSegmentation.setSubRange(beginIt, endIt);
theSegmentation.setMode("myMode");
```

- **greedy**
  - "Truncate" (default)
  - "Truncate+1"
  - "DoNotTruncate"

- **saturated**
  - "First",
  - "MostCentered" (default)
  - "Last"