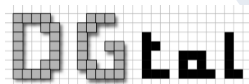


Digital surfaces in DGtal Topology module (since 0.5)

Jacques-Olivier Lachaud

DGtal Meeting, june 2012

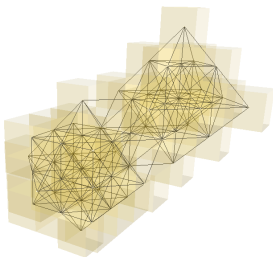


UMR 5127

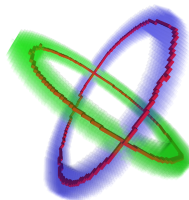
Package Topology, available in DGtal 0.4

1. classical digital topology (*à la Rosenfeld*)

- ▶ Arbitrary adjacencies in \mathbb{Z}^n , but also in subdomains
- ▶ Digital topology = couple of adjacencies (Rosenfeld)
- ▶ Object = Topology + Set
- ▶ Operations : neighborhoods, border, connectedness and connected components, decomposition into digital layers, simple points



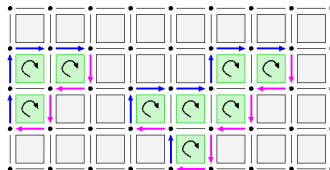
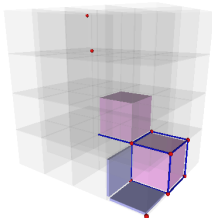
Adjacencies



thinning in (6,26)

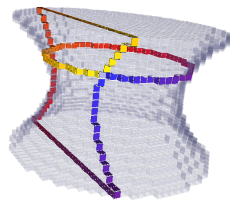
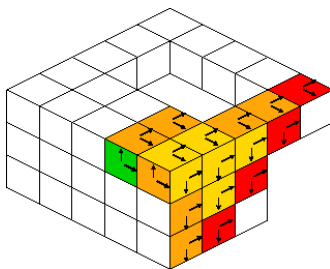
Package Topology, available in DGtal 0.4

1. classical digital topology (*à la Rosenfeld*)
2. cubical cellular topology + algebraic topology
 - ▶ cells, adjacent and incident cells, faces and cofaces
 - ▶ signed cells, signed incidence, boundary operators



Package Topology, available in DGtal 0.4

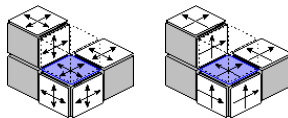
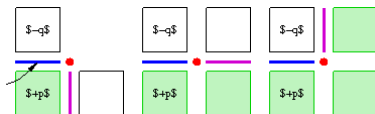
1. classical digital topology (*à la Rosenfeld*)
2. cubical cellular topology + algebraic topology
3. digital surface topology (*à la Herman*)
 - ▶ surfels, surfel adjacency, surfel neighborhood
 - ▶ surface tracking (normal, fast), contour tracking in nD



Package Topology, new in DGtal 0.5

Digital Surface

- | | | |
|---------------------------------|---|------------------------|
| surfels / signed $n - 1$ -cells | } | • kind of "dual" graph |
| + adjacencies between surfels | | • kind of manifold |



Package Topology, new in DGtal 0.5

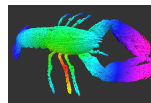
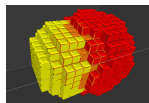
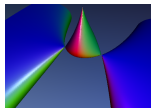
Digital Surface

surfels / signed $n - 1$ -cells	}	• kind of "dual" graph
+ adjacencies between surfels		• kind of manifold

1. High-level `DigitalSurface` class for representing any kind of digital surface
2. Many container classes for digital surfaces
 - ▶ boundary of digital shape
 - ▶ boundary of implicitly defined shape
 - ▶ set of surfels
 - ▶ implicitly defined set of surfels
 - ▶ light containers
3. a `DigitalSurface` is a graph
4. a `DigitalSurface` is a combinatorial surface (with umbrellas)

Direct applications

- marching cubes algorithm
- tracking implicit polynomial surfaces
- representing boundary of regions and frontier between regions
- breadth-first visiting on surfaces
- estimating normals on surfaces



Necessary concepts and classes for digital surfaces

One must choose

- the representation of cellular grid space : model of `CCellularGridSpaceND`
e.g. `KhalimskySpaceND< N, int >, Z2i::KSpace, Z3i::KSpace`
- the kind of adjacency between surfels, `SurfelAdjacency< N >`
- the kind of surface container : model of `CDigitalSurfaceContainer`

```
1  typedef Z3i::Point Point; // 3D digital point
2  typedef Z3i::Domain Domain;
3  typedef Z3i::DigitalSet DigitalSet; // a set of
   digital points
4  typedef Z3i::KSpace KSpace; // 3D cellular grid space
5  typedef SurfelAdjacency<3> SAdj; // surfel adjacency.
6  typedef DigitalSetBoundary<KSpace,DigitalSet>
   Container; // kind of surface container
7  typedef DigitalSurface<Container> MyDigSurf; //
   concrete digital surface
```


Concrete instantiations for digital surfaces

Then, the chosen types are instantiated. Here

digital surface = boundary of two intersecting balls

```
1 Point p1( -20, -20, -20 ), p2( 20, 20, 20 );
2 KSpace K; K.init( p1, p2, true ); // init space
3 DigitalSet someShape( Domain( p1, p2 ) );
4 Shapes<Domain>::addNorm2Ball( someShape, Point
    (-3,0,0), 4 );
5 Shapes<Domain>::addNorm2Ball( someShape, Point
    (3,0,0), 4 );
6 SAdj surfAdj( true ); // the adjacency
7 Container surfContainer( K, someShape, surfAdj );
8 MyDigSurf digSurf( surfContainer ); // digital
    surface
```

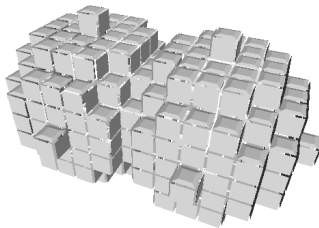
Using the digital surface (displays 518) :

```
1 cout << "-_nb_surfels/vertices_="
2      << digSurf.size() << endl;
```

How to use digital surfaces (I)

Just enumerating its elements...

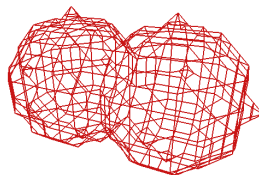
```
1  QApplication application( argc, argv );
2  Viewer3D viewer; // QGL viewer
3  viewer.show();
4  for( MyDigSurf::ConstIterator it = digSurf.begin(),
5      itend = digSurf.end(); it != itend; ++it )
6      viewer << *it;
7  viewer << Viewer3D::updateDisplay;
8  return application.exec();
```



How to use digital surfaces (II)

Getting the neighbors and drawing the graph...

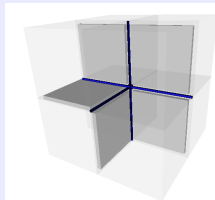
```
1     typedef std::vector<Vertex> Neighborhood;
2     for ( ConstIterator it = digSurf.begin(),
3           itend = digSurf.end(); it != itend; ++it )
4     {
5         Neighborhood N;
6         back_insert_iterator<Neighborhood> itN = back_inserter( N );
7         digSurf.writeNeighbors( itN , *it );
8         Point p = K.sKCoords( *it );
9         for ( unsigned int i = 0; i < N.size(); ++i )
10        {
11            Point q = K.sKCoords( N[ i ] );
12            viewer.addLine ( p[0]/2.0, p[1]/2.0, p[2]/2.0,
13                            q[0]/2.0, q[1]/2.0, q[2]/2.0,
14                            DGtal::Color ( 200,20,20 ), 2.0 );
15        }
16    }
```



How to use digital surfaces (III)

Digital surfaces are combinatorial surfaces

- in n -D
- vertices = $n - 1$ -cells
- edges $\approx n - 2$ -cells
- faces $\approx n - 3$ -cells



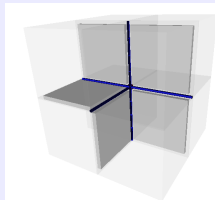
Inner types `Vertex`, `Arc`, `Face`, `xxxRange`, `xxxSet`

```
1 FaceRange facesAroundVertex( const Vertex & v )
2 VertexRange verticesAroundFace( const Face & f )
3 FaceRange facesAroundArc( const Arc & a )
4 FaceSet allFaces()
5 FaceSet allClosedFaces()
6 FaceSet allOpenFaces() ...
```

How to use digital surfaces (III)

Digital surfaces are combinatorial surfaces

- in **3-D**
- vertices = surfels
- edges \approx linels
- faces = **umbrellas**



Inner types **Vertex**, **Arc**, **Face**, **xxxRange**, **xxxSet**

```
1 FaceRange facesAroundVertex( const Vertex & v )
2 VertexRange verticesAroundFace( const Face & f )
3 FaceRange facesAroundArc( const Arc & a )
4 FaceSet allFaces()
5 FaceSet allClosedFaces()
6 FaceSet allOpenFaces() ...
```

How to use digital surfaces (IV)

Getting the faces and outputting their vertices

```

1     typedef typename FaceSet::const_iterator FaceSetIter;
2     typedef typename VertexRange::const_iterator VertexRangeIter;
3     FaceSet faces = digSurf.allClosedFaces();
4     for ( FaceSetIter itf = faces.begin(),
5           itf_end = faces.end(); itf != itf_end; ++itf )
6     {
7         Face face = *itf;
8         out << face.nbVertices;
9         VertexRange vtcs = digSurf.verticesAroundFace( face );
10        for ( VertexRangeIter itv = vtcs.begin(),
11              itv_end = vtcs.end(); itv != itv_end; ++itv )
12            out << " " << index[ *itv ];
13        out << std::endl;
14    }

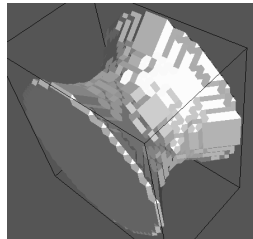
```

e.g. export in OFF format

```

1     void exportSurfaceAs3DOFF ( std::ostream
2                                & out )
3
4     template <typename CellEmbedder>
5     void exportEmbeddedSurfaceAs3DOFF
6     ( std::ostream & out, const CellEmbedder
7       & cembedder )

```



Diversity of digital surfaces

- may be open or closed
- may be connected or not
- may be defined explicitly with their surfels
- may be defined implicitly as the boundary of some shape
- the surfels may be listed or known only through a predicate
- the shape may be described by its points or known only through a predicate
- the surface may be big or infinite so that only lazy extraction is reasonable

You can process them with the same object `DigitalSurface<T>`

Package description

Should contain

- classical digital topology \tilde{A} /a Rosenfeld
- cartesian cellular topology
- digital surface topology \tilde{A} /a Herman
- must be the base block of geometric algorithms

Examples

- adjacencies, connected components, simple points, thinning
- cells, boundary operators, incidence, opening, closing
- contours, surfel adjacency, surface tracking
- topological invariants

Location

- `{DGtal}/src/DGtal/topology`
- `{DGtal}/src/DGtal/helpers`
- `{DGtal}/tests/topology`