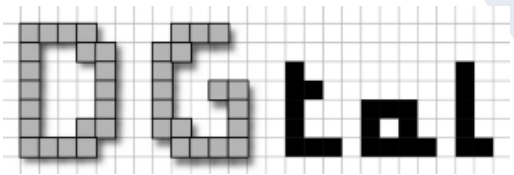
Digital surfaces in DGtal
Topology module (since 0.5)

Jacques-Olivier Lachaud

DGtal Meeting, june 2012

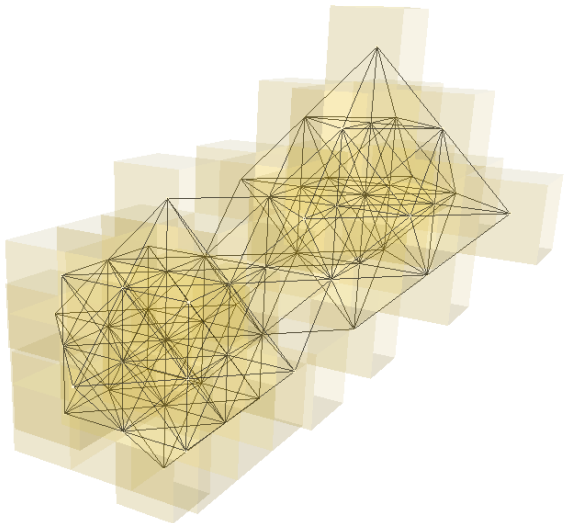


UMR 5127

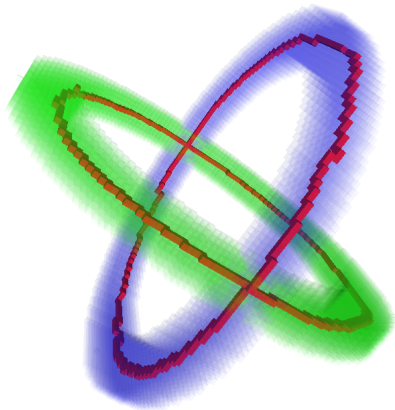
Package Topology, available in DGtal 0.4

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



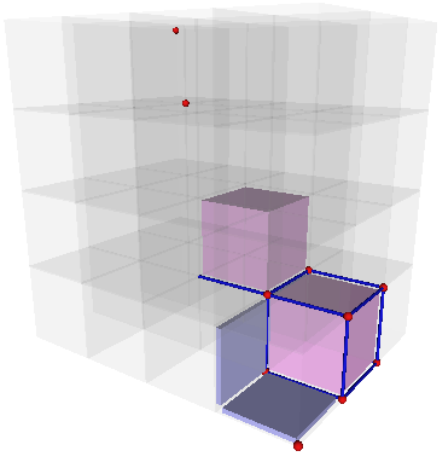
Adjacencies

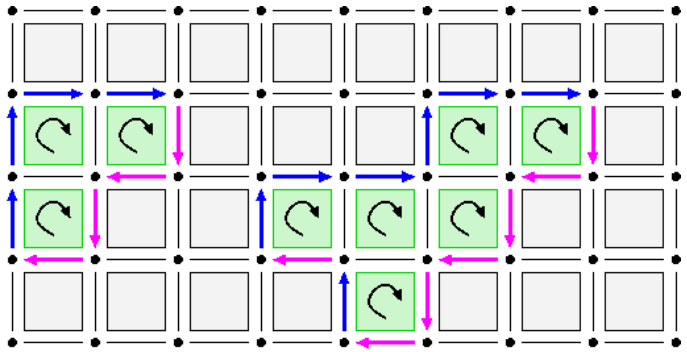


thinning in (6,26)

Package Topology, available in DGtal 0.4

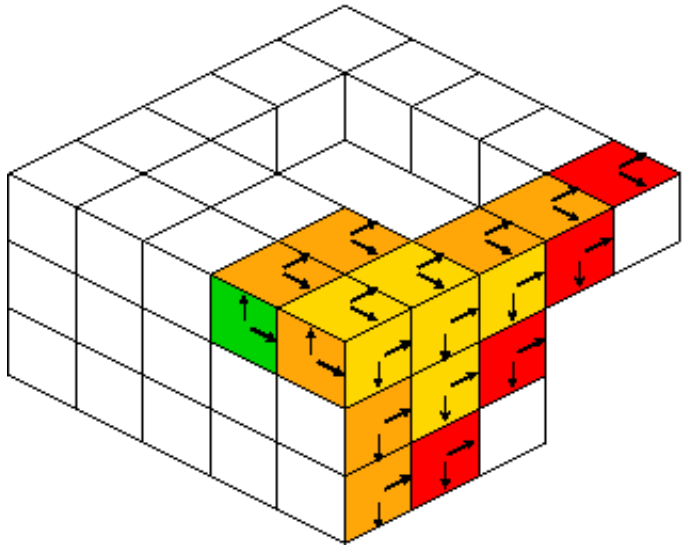
1. classical digital topology (*à la Rosenfeld*)
2. cubical cellular topology + algebraic topology
   - cells, adjacent and incident cells, faces and cofaces
   - signed cells, signed incidence, boundary operators

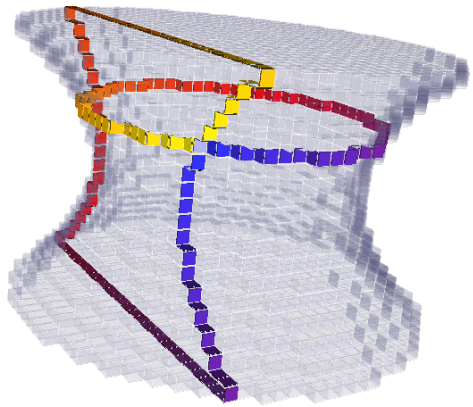




Package Topology, available in DGtal 0.4

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 *n*D



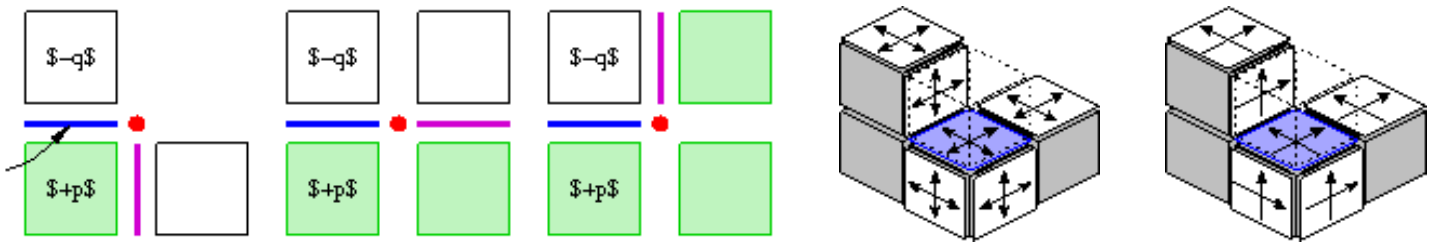


Package Topology, new in DGtal 0.5

Digital Surface

surfels / signed $n-1$-cells
\+ adjacencies between surfels

- kind of "dual" graph
- kind of manifold



Package Topology, new in DGtal 0.5

Digital Surface

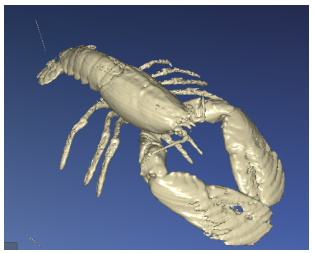
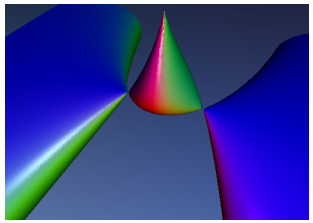
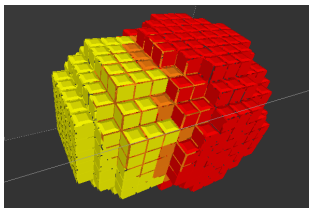
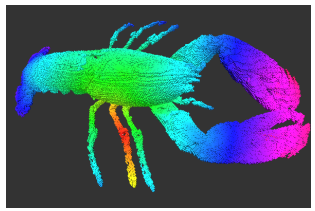
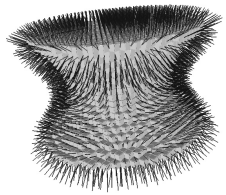
surfels / signed $n-1$-cells + adjacencies between surfels

- kind of "dual" graph
- 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 instanciations 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
```

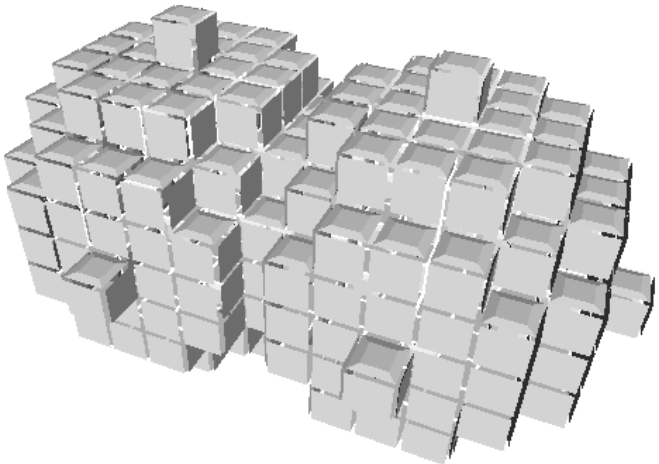
Using the digital surface (displays 518) :

```
1    cout << "- nb surfels/vertices = "
2         << digSurf.size() << endl;
```

How to use digital surfaces (I)

Just enumerating its elements...

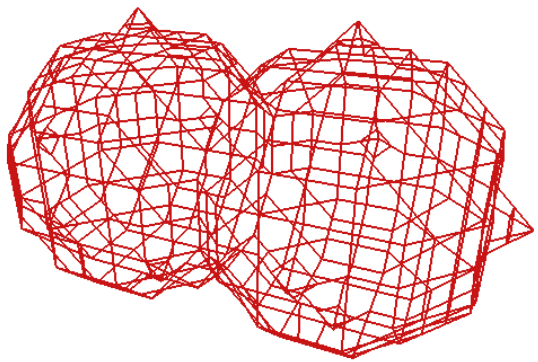
```
1    QApplication application( argc, argv );
2    Viewer3D viewer; // QGL viewer
3    viewer.show();
4    for( MyDigSurf::ConstIterator it = digSurf.begin(),
5     itend = digSurf.end(); it != itend; ++it )
6      viewer << *it;
7    viewer << Viewer3D::updateDisplay;
8    return application.exec();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



How to use digital surfaces (II)

Getting the neighbors and drawing the graph...

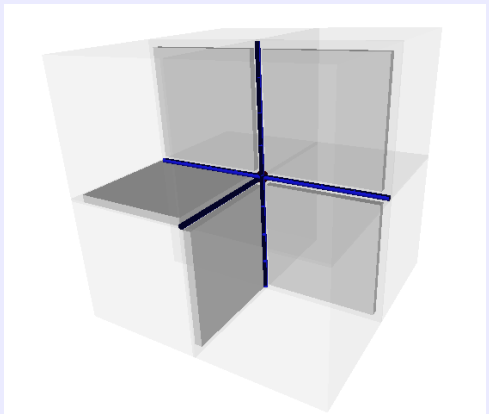
```
1   typedef std::vector<Vertex> Neighborhood;
2   for ( ConstIterator it = digSurf.begin(),
3           itend = digSurf.end(); it != itend; ++it )
4     {
5       Neighborhood N;
6       back_insert_iterator<Neighborhood> itN = back_inserter( N );
7       digSurf.writeNeighbors( itN , *it );
8       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



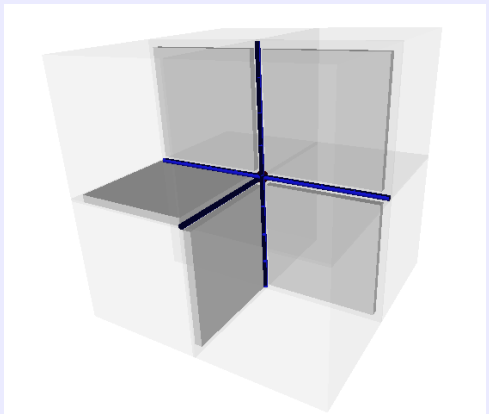
Inner types `Vertex, Arc, Face, xxxRange, xxxSet`

```
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() ...
```

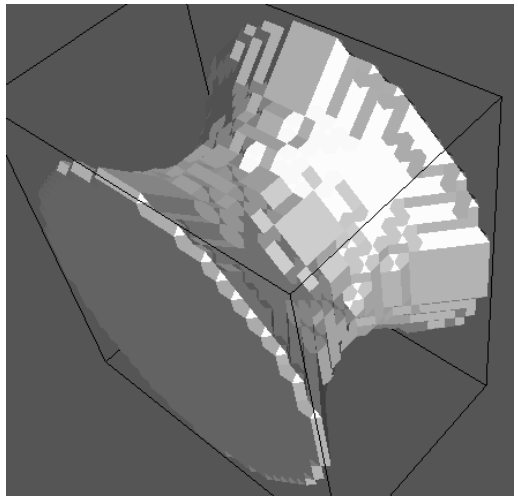
How to use digital surfaces (IV)

Getting the faces and outputing 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
        & out )
2
3   template <typename CellEmbedder>
4   void exportEmbeddedSurfaceAs3DOFF
5   ( std::ostream & out, const CellEmbedder
        & cembedder )
```

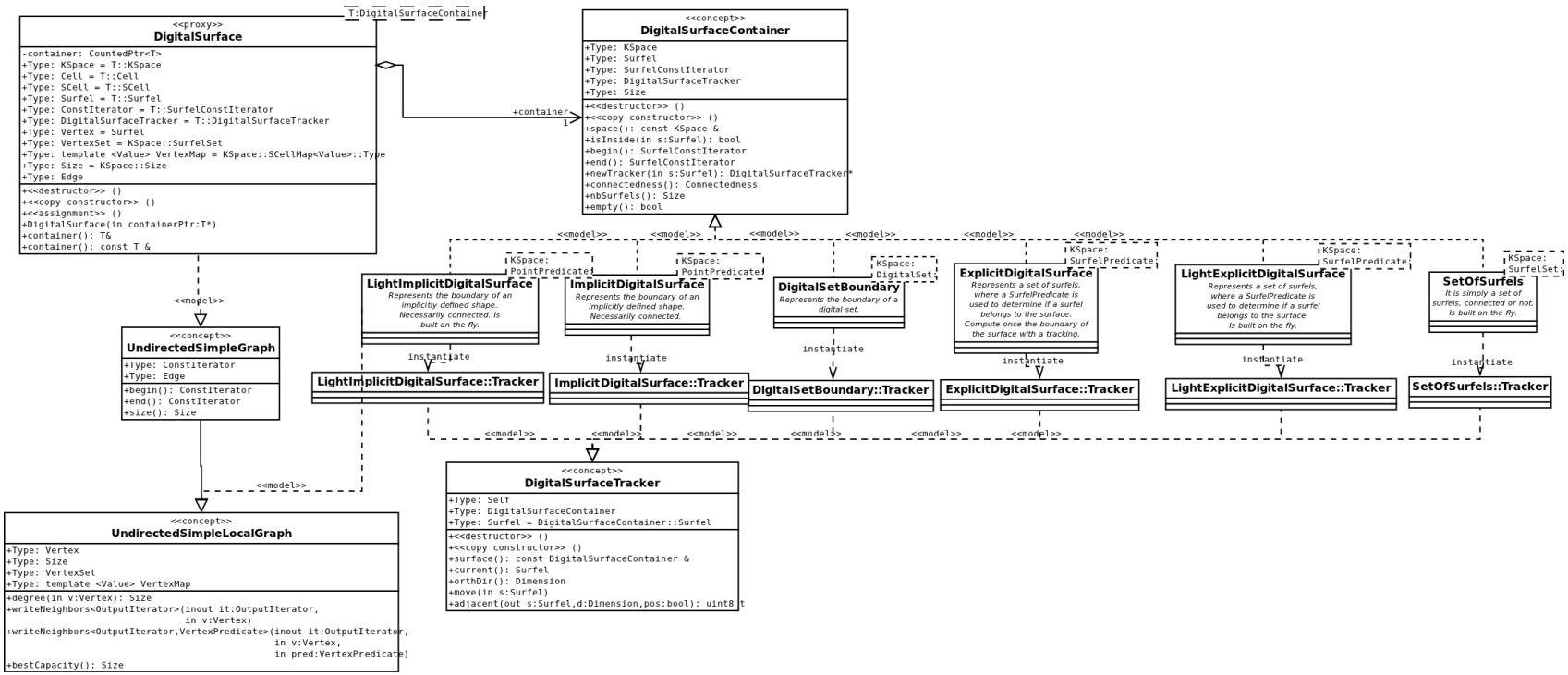


Diversity of digital surfaces

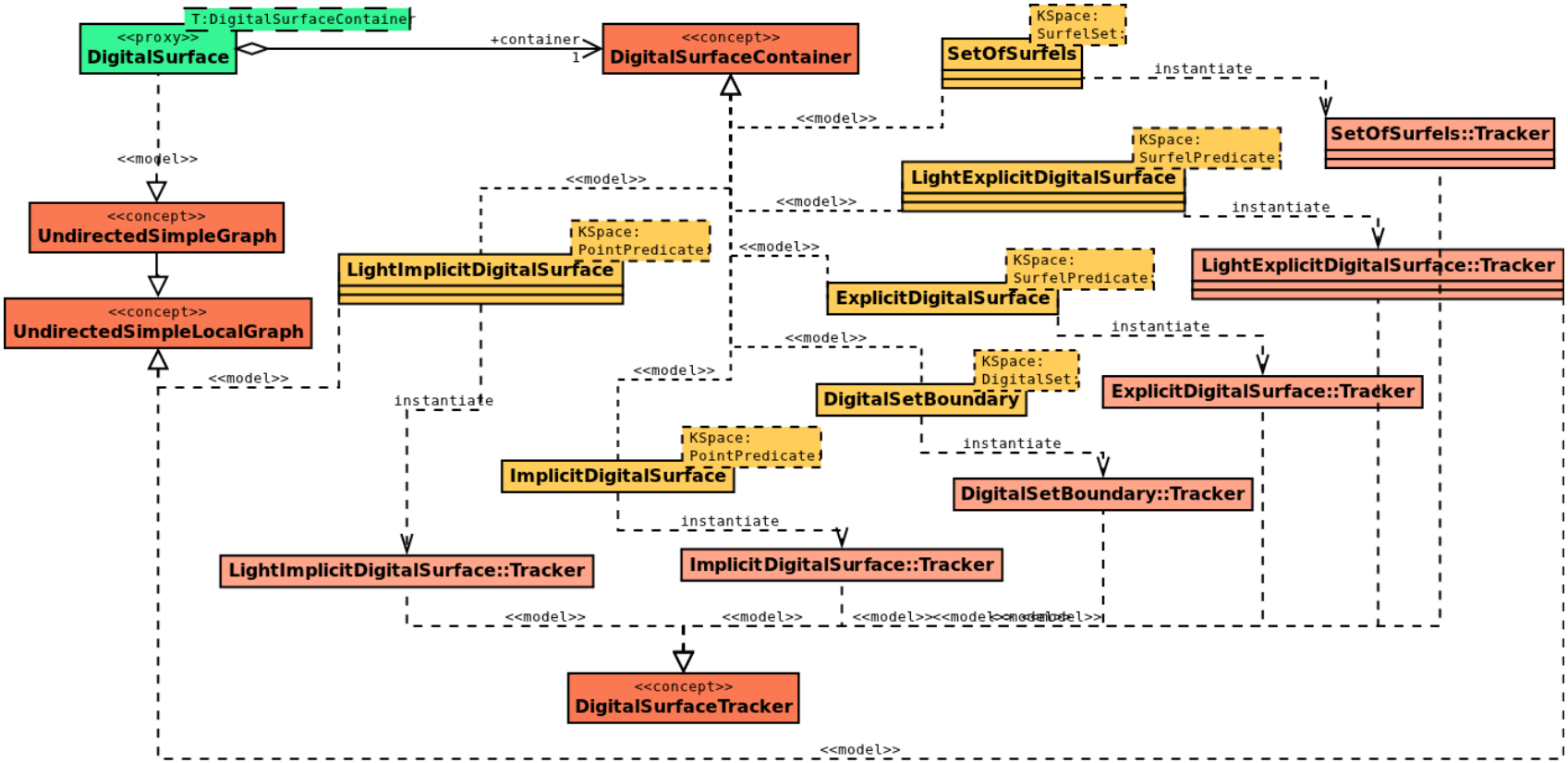
- may be open or closed
- may be connected or not
- may be defined explicitly with their surfels
- may be defined implicitly as the boundary of some shape
- the surfels may be listed or known only through a predicate
- the shape may be described by its points or known only through a predicate
- the surface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

You wish to process them with the same object : `DigitalSurface<T>`
T is a model of `CDigitalSurfaceContainer`

Partial architecture



Partial architecture



Digital surface containers

- `DigitalSetBoundary<KSpace,DigitalSet>` Represents the boundary of a digital set (a set of digital points, considered as the set of pixels/voxels/spels of the space).

$\Rightarrow$ interpixel boundary of a digital shape

Digital surface containers

- `DigitalSetBoundary<KSpace,DigitalSet>` Represents the boundary of a digital set (a set of digital points, considered as the set of pixels/voxels/spels of the space).

  $\Rightarrow$ interpixel boundary of a digital shape

- `ImplicitDigitalSurface<KSpace,PointPredicate>` Represents the (connected) boundary of shape defined implicitly by a predicate. + `Light` version.

  $\Rightarrow$ implicit surface computed once or on-the-fly

Digital surface containers

- `DigitalSetBoundary<KSpace,DigitalSet>` Represents the boundary of a digital set (a set of digital points, considered as the set of pixels/voxels/spels of the space).

  $\Rightarrow$ interpixel boundary of a digital shape

- `ImplicitDigitalSurface<KSpace,PointPredicate>` Represents the (connected) boundary of shape defined implicitly by a predicate. + `Light` version.

  $\Rightarrow$ implicit surface computed once or on-the-fly

- `SetOfSurfels<KSpace,SurfelSet>` Represents an arbitrary set of surfels stored explicitly.

  $\Rightarrow$ arbitrary known surface : add topology to a set

Digital surface containers

- `DigitalSetBoundary<KSpace,DigitalSet>` Represents the boundary of a digital set (a set of digital points, considered as the set of pixels/voxels/spels of the space).

  ⇒ interpixel boundary of a digital shape

- `ImplicitDigitalSurface<KSpace,PointPredicate>` Represents the (connected) boundary of shape defined implicitly by a predicate. + `Light` version.

  ⇒ implicit surface computed once or on-the-fly

- `SetOfSurfels<KSpace,SurfelSet>` Represents an arbitrary set of surfels stored explicitly.

  ⇒ arbitrary known surface : add topology to a set

- `ExplicitDigitalSurface<KSpace,SurfelPredicate>` Represents a (connected) set of surfels defined implicitly by a predicate. + `Light` version.

  ⇒ frontier between regions in images, computed once or on-the-fly

Example : frontiers between regions in image (I)

Creating the labelled image...

```
1   using namespace Z3i;
2   typedef ImageContainerBySTLVector<Domain,DGtal::
        uint8_t> Image;
3   Domain domain0( Point( 0,0,0 ), Point( 10, 10, 10 )
        );
4   Domain domain1( Point( 2,2,2 ), Point( 8, 8, 8 ) );
5   Domain domain2( Point( 2,4,4 ), Point( 8, 6, 6 ) );
6   Image image( domain0 );
7   fill( image, domain0, 0 ); // label 0
8   fill( image, domain1, 1 ); // label 1
9   fill( image, domain2, 2 ); // label 2
10  KSpace K; // creating cellular space
11  K.init( domain0.lowerBound(), domain0.upperBound(),
        true );
```

Example : frontiers between regions in image (II)

Creating the frontier between region 1 and region 0...

```
1   typedef SurfelAdjacency<KSpace::dimension> SurfAdj;
2   typedef FrontierPredicate<KSpace, Image> FSurfPred;
3   typedef ExplicitDigitalSurface<KSpace,FSurfPred>
        FrontierContainer;
4   typedef DigitalSurface<FrontierContainer> Frontier;
5   SurfAdj surfAdj( true ); // interior in all
        directions.
6   // frontier between label 1 and 0 (connected part
        with bel10)
7   SCell vox1  = K.sSpel( Point( 2,2,2 ), K.POS );
8   SCell bel10 = K.sIncident( vox1, 0, false );
9   FSurfPred surfPred10( K, image, 1, 0 );
10  Frontier frontier10 = // acquired
11    new FrontierContainer( K, surfPred10, surfAdj,
          bel10 );
```

Example : frontiers between regions in image (III)

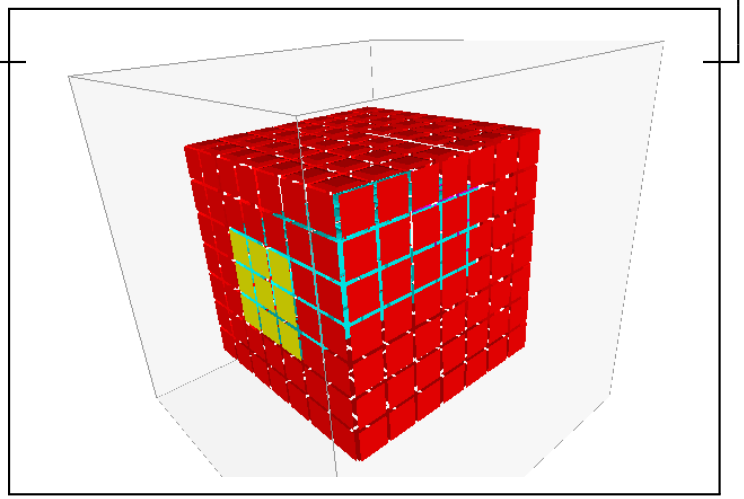
Idem for region 2 and 0 (two parts) and 2 and 1...

```
1     // frontier between label 2 and 0 (with bel20)
2     SCell vox2  = K.sSpel( Point( 2,4,4 ), K.POS );
3     SCell bel20 = K.sIncident( vox2, 0, false );
4     FFSurfPred surfPred20( K, image, 2, 0 );
5     Frontier frontier20 =
6       new FrontierContainer( K, surfPred20, surfAdj, bel20 );
7     // frontier between label 2 and 0 (with bel20bis)
8     SCell vox2bis  = K.sSpel( Point( 8,6,6 ), K.POS );
9     SCell bel20bis = K.sIncident( vox2bis, 0, true );
10    FFSurfPred surfPred20bis( K, image, 2, 0 );
11    Frontier frontier20bis =
12      new FrontierContainer( K, surfPred20bis, surfAdj, bel20bis );
13    trace.endBlock();
14    // frontier between label 2 and 1 (with bel21)
15    SCell bel21 = K.sIncident( vox2, 1, false );
16    FFSurfPred surfPred21( K, image, 2, 1 );
17    Frontier frontier21 =
18      new FrontierContainer( K, surfPred21, surfAdj, bel21 );
```

Example : frontiers between regions in image (III)

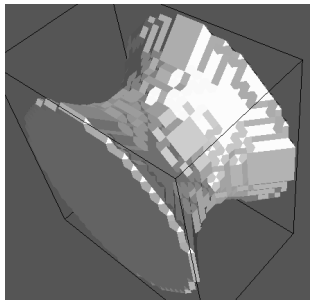
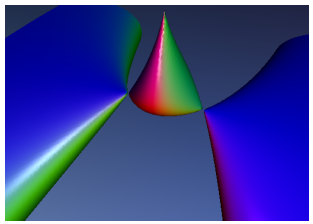
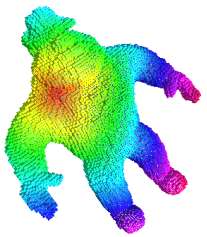
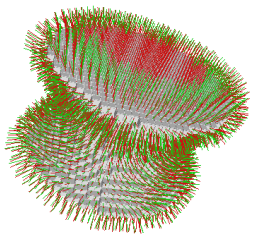
Displaying surfaces...

```
1   QApplication application(argc,argv);
2   Viewer3D viewer;
3   viewer.show();
4   viewer << SetMode3D( domain0.className(), "BoundingBox" )
5          << domain0;
6   Cell dummy;
7   // Display frontier between 1 and 0 in RED
8   unsigned int nbSurfels10 = 0;
9   viewer << CustomColors3D( Color::Red, Color::Red );
10  for ( Frontier::ConstIterator
11          it = frontier10.begin(), it_end = frontier10.end();
12        it != it_end; ++it, ++nbSurfels10 )
13    viewer << *it;
14  // Display frontier between 2 and 0 in MAGENTA and YELLOW
15  // Display frontier between 2 and 1 in CYAN
16    ...
```



More fun... current simple applications

```
1  cd examples/topology
2  # marching cubes algorithm
3  ./volMarchingCubes cat10.vol 1 255 0
4  geomview marching-cube.off
5  # tracking implicit polynomial surfaces
6  ./trackImplicitPolynomialSurfaceToOFF "3*x^2-2*y^2+z^3+5y^2*(z-1)*(z+1)
       " -2 -2 -2 2 2 2 0.05
7  geomview marching-cube.off
8  # breadth-first visiting on surfaces
9  ./volBreadthFirstTraversal A1.100.vol 0 255
10 # More elaborate example: estimating normals on surfaces
11 cd ../../tests/geometry/surfaces
12 ./testLocalConvolutionNormalVectorEstimator
```

Next objectives (from 0.4 to 0.5)

1. classical digital topology
   - other adjacencies (no)
   - Adjacency = unoriented graph, associated concepts (part)
   - make everything faster with specialization (especially simpleness) (part)
2. cubical cellular topology
   - cellular grid space concept (yes)
   - cubical complexes, interior, closure (no)
   - path, mapping (homotopy) (no)
   - chains, boundary operator, cochains, coboundary (no)
   - (co)homology (no)
3. digital surface topology
   - digital surface concept (yes)
   - digital surface graph and cograph (umbrellas) (yes)
   - digital surface map (part)

Topology package description (as of 0.5)

Content

- classical digital topology *à la* Rosenfeld
- cartesian cellular topology
- digital surface topology *à la* Herman
- base block of geometric algorithms

Examples

- adjacencies, connected components, simple points, thinning
- cells, boundary operators, incidence, opening, closing
- contours, surfel adjacency, surface tracking
- high-level manipulation of digital surfaces

Location

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