Digital surfaces in DGtal
Topology module (since 0.5)

Jacques-Olivier Lachaud

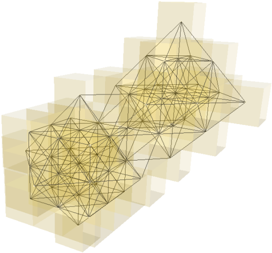
DGtal Meeting, june 2012

UMR 5127

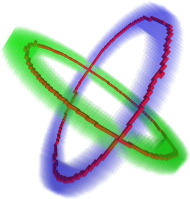
Package Topology, available in DGtal 0.4

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



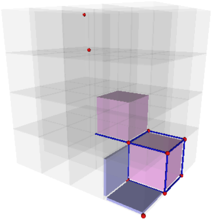
Adjacencies

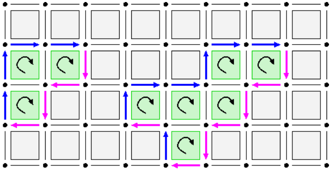


thinning in (6,26)

Package Topology, available in DGtal 0.4

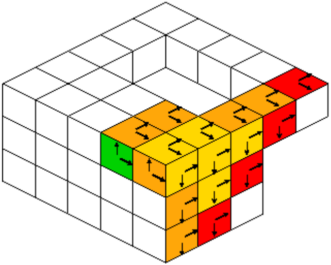
1. classical digital topology (*à la Rosenfeld*)
2. cubical cellular topology + algebraic topology
   - cells, adjacent and incident cells, faces and cofaces
   - signed cells, signed incidence, boundary operators

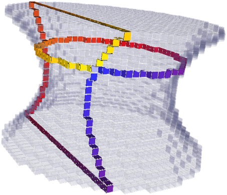




Package Topology, available in DGtal 0.4

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





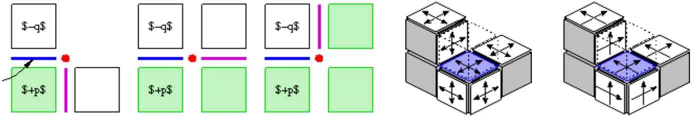
Package Topology, new in DGtal 0.5

Digital Surface

surfels / signed $n-1$-cells

$+$ adjacencies between surfels $\Big\}$

- kind of "dual" graph
- kind of manifold



Package Topology, new in DGtal 0.5

Digital Surface

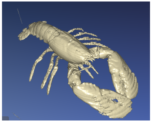
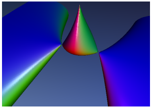
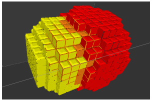
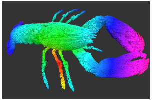
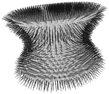
$$\left.\begin{array}{rl} & \text{surfels / signed } n-1\text{-cells} \\ + & \text{adjacencies between surfels} \end{array}\right\}$$

- 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

```
typedef Z3i::Point Point; // 3D digital point
typedef Z3i::Domain Domain;
typedef Z3i::DigitalSet DigitalSet; // a set of digital points
typedef Z3i::KSpace KSpace; // 3D cellular grid space
typedef SurfelAdjacency<3> SAdj; // surfel adjacency.
typedef DigitalSetBoundary<KSpace,DigitalSet> Container; // kind of surface container
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

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

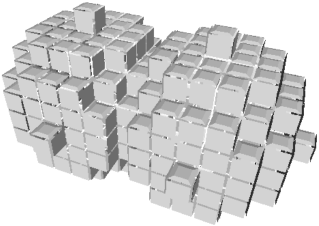
Using the digital surface (displays 518) :

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

How to use digital surfaces (I)

Just enumerating its elements...

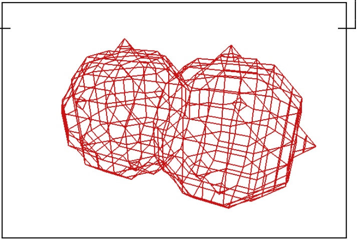
```
QApplication application( argc, argv );
Viewer3D viewer; // QGL viewer
viewer.show();
for( MyDigSurf::ConstIterator it = digSurf.begin(),
 itend = digSurf.end(); it != itend; ++it )
  viewer << *it;
viewer << Viewer3D::updateDisplay;
return application.exec();
```



How to use digital surfaces (II)

Getting the neighbors and drawing the graph...

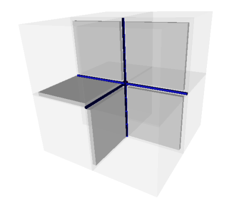
```
typedef std::vector<Vertex> Neighborhood;
for ( ConstIterator it = digSurf.begin(),
        itend = digSurf.end(); it != itend; ++it )
  {
    Neighborhood N;
    back_insert_iterator<Neighborhood> itN = back_inserter( N );
    digSurf.writeNeighbors( itN , *it );
    Point p = K.sKCoords( *it );
    for ( unsigned int i = 0; i < N.size(); ++i )
      {
        Point q = K.sKCoords( N[ i ] );
        viewer.addLine ( p[0]/2.0, p[1]/2.0, p[2]/2.0,
                         q[0]/2.0, q[1]/2.0, q[2]/2.0,
                         DGtal::Color ( 200,20,20 ), 2.0 );
      }
  }
```



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



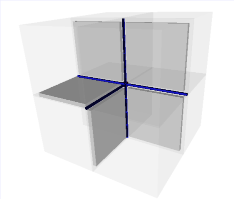
Inner types `Vertex, Arc, Face, xxxRange, xxxSet`

```
FaceRange   facesAroundVertex( const Vertex & v )
VertexRange verticesAroundFace( const Face & f )
FaceRange   facesAroundArc( const Arc & a )
FaceSet     allFaces()
FaceSet     allClosedFaces()
FaceSet     allOpenFaces() ...
```

How to use digital surfaces (III)

Digital surfaces are combinatorial surfaces

- in 3-D
- vertices = surfels
- edges ≈ linels
- faces = umbrellas



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

```
FaceRange   facesAroundVertex( const Vertex & v )
VertexRange verticesAroundFace( const Face & f )
FaceRange   facesAroundArc( const Arc & a )
FaceSet     allFaces()
FaceSet     allClosedFaces()
FaceSet     allOpenFaces() ...
```

How to use digital surfaces (IV)

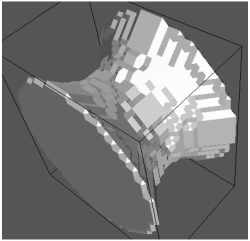
Getting the faces and outputing their vertices

```
typedef typename FaceSet::const_iterator FaceSetIter;
typedef typename VertexRange::const_iterator VertexRangeIter;
FaceSet faces = digSurf.allClosedFaces();
for ( FaceSetIter itf = faces.begin(),
        itf_end = faces.end(); itf != itf_end; ++itf )
  {
    Face face = *itf;
    out << face.nbVertices;
    VertexRange vtcs = digSurf.verticesAroundFace( face );
    for ( VertexRangeIter itv = vtcs.begin(),
            itv_end = vtcs.end(); itv != itv_end; ++itv )
      out << "␣" << index[ *itv ];
    out << std::endl;
  }
```

e.g. export in OFF format

```
void exportSurfaceAs3DOFF ( std::ostream
     & out )

template <typename CellEmbedder>
void exportEmbeddedSurfaceAs3DOFF
( std::ostream & out, const CellEmbedder
     & cembedder )
```

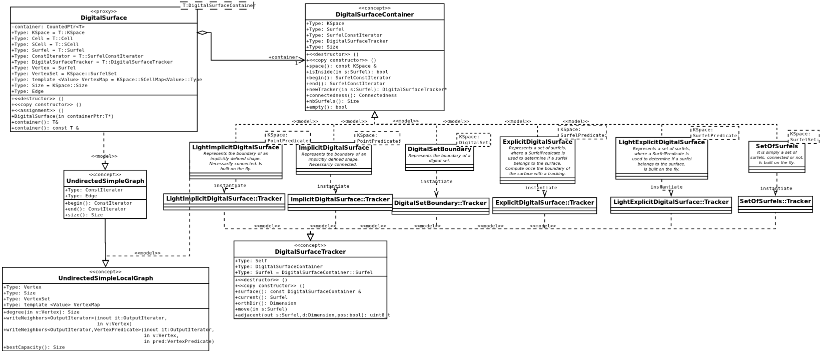


Diversity of digital surfaces

- 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



```
<<proxy>>
DigitalSurface
-container: CountedPtr<T>
+Type: KSpace = T::KSpace
+Type: Cell = T::Cell
+Type: SCell = T::SCell
+Type: Surfel = T::Surfel
+Type: ConstIterator = T::SurfelConstIterator
+Type: DigitalSurfaceTracker = T::DigitalSurfaceTracker
+Type: Vertex = Surfel
+Type: VertexSet = KSpace::SurfelSet
+Type: template <Value> VertexMap = KSpace::SCellMap<Value>::Type
+Type: Size = KSpace::Size
+Type: Edge
+<<destructor>> ()
+<<copy constructor>> ()
+<<assignment>> ()
+DigitalSurface(in containerPtr:T*)
+container(): T&
+container(): const T &
```

```
<<concept>>
DigitalSurfaceContainer
+Type: KSpace
+Type: Surfel
+Type: SurfelConstIterator
+Type: DigitalSurfaceTracker
+Type: Size
+<<destructor>> ()
+<<copy constructor>> ()
+space(): const KSpace &
+isInside(in s:Surfel): bool
+begin(): SurfelConstIterator
+end(): SurfelConstIterator
+newTracker(in s:Surfel): DigitalSurfaceTracker*
+connectedness(): Connectedness
+nbSurfels(): Size
+empty(): bool
```

```
<<concept>>
UndirectedSimpleGraph
+Type: ConstIterator
+Type: Edge
+begin(): ConstIterator
+end(): ConstIterator
+size(): Size
```

```
<<concept>>
UndirectedSimpleLocalGraph
+Type: Vertex
+Type: Size
+Type: VertexSet
+Type: template <Value> VertexMap
+degree(in v:Vertex): Size
+writeNeighbors<OutputIterator>(inout it:OutputIterator,
                                in v:Vertex)
+writeNeighbors<OutputIterator,VertexPredicate>(inout it:OutputIterator,
                                                in v:Vertex,
                                                in pred:VertexPredicate)
+bestCapacity(): Size
```

LightImplicitDigitalSurface (KSpace: PointPredicate)
Represents the boundary of an implicitly defined shape. Necessarily connected. Is built on the fly.

ImplicitDigitalSurface (KSpace: PointPredicate)
Represents the boundary of an implicitly defined shape. Necessarily connected.

DigitalSetBoundary (KSpace: DigitalSet)
Represents the boundary of a digital set.

ExplicitDigitalSurface (KSpace: SurfelPredicate)
Represents a set of surfels, where a SurfelPredicate is used to determine if a surfel belongs to the surface. Compute once the boundary of the surface with a tracking.

LightExplicitDigitalSurface (KSpace: SurfelPredicate)
Represents a set of surfels, where a SurfelPredicate is used to determine if a surfel belongs to the surface. Is built on the fly.

SetOfSurfels (KSpace: SurfelSet)
It is simply a set of surfels, connected or not. Is built on the fly.

LightImplicitDigitalSurface::Tracker

ImplicitDigitalSurface::Tracker

DigitalSetBoundary::Tracker

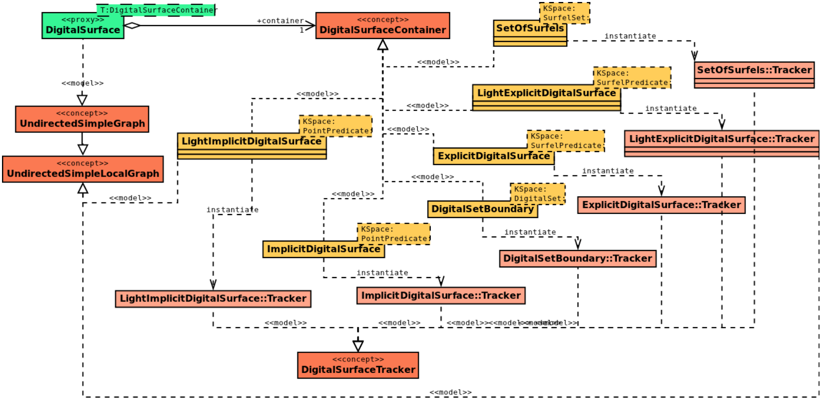
ExplicitDigitalSurface::Tracker

LightExplicitDigitalSurface::Tracker

SetOfSurfels::Tracker

```
<<concept>>
DigitalSurfaceTracker
+Type: Self
+Type: DigitalSurfaceContainer
+Type: Surfel = DigitalSurfaceContainer::Surfel
+<<destructor>> ()
+<<copy constructor>> ()
+surface(): const DigitalSurfaceContainer &
+current(): Surfel
+orthDir(): Dimension
+move(in s:Surfel)
+adjacent(out s:Surfel,d:Dimension,pos:bool): uint8_t
```

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).

$\boxed{\Rightarrow \text{interpixel boundary of a digital shape}}$

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).

  ⇒ 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

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

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...

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

Example : frontiers between regions in image (II)

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

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

Example : frontiers between regions in image (III)

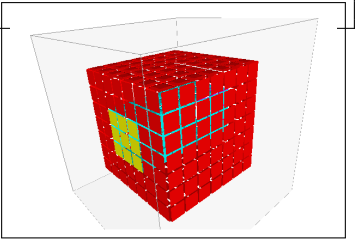
Idem for region 2 and 0 (two parts) and 2 and 1...

```
// frontier between label 2 and 0 (with bel20)
SCell vox2  = K.sSpel( Point( 2,4,4 ), K.POS );
SCell bel20 = K.sIncident( vox2, 0, false );
FFSurfPred surfPred20( K, image, 2, 0 );
Frontier frontier20 =
  new FrontierContainer( K, surfPred20, surfAdj, bel20 );
// frontier between label 2 and 0 (with bel20bis)
SCell vox2bis  = K.sSpel( Point( 8,6,6 ), K.POS );
SCell bel20bis = K.sIncident( vox2bis, 0, true );
FFSurfPred surfPred20bis( K, image, 2, 0 );
Frontier frontier20bis =
  new FrontierContainer( K, surfPred20bis, surfAdj, bel20bis );
trace.endBlock();
// frontier between label 2 and 1 (with bel21)
SCell bel21 = K.sIncident( vox2, 1, false );
FFSurfPred surfPred21( K, image, 2, 1 );
Frontier frontier21 =
  new FrontierContainer( K, surfPred21, surfAdj, bel21 );
```

Example : frontiers between regions in image (III)

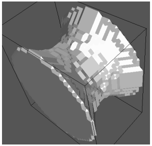
Displaying surfaces...

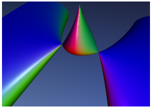
```
QApplication application(argc,argv);
Viewer3D viewer;
viewer.show();
viewer << SetMode3D( domain0.className(), "BoundingBox" )
       << domain0;
Cell dummy;
// Display frontier between 1 and 0 in RED
unsigned int nbSurfels10 = 0;
viewer << CustomColors3D( Color::Red, Color::Red );
for ( Frontier::ConstIterator
        it = frontier10.begin(), it_end = frontier10.end();
      it != it_end; ++it, ++nbSurfels10 )
  viewer << *it;
// Display frontier between 2 and 0 in MAGENTA and YELLOW
// Display frontier between 2 and 1 in CYAN
  ...
```

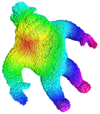


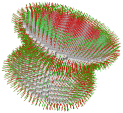
More fun... current simple applications

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