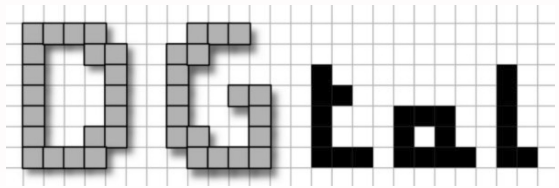
DGtal boards and viewers

Bertrand Kerautret

LORIA - Université de Lorraine

Journée DGtal, 7 Juin 2012 - Nantes
(mis à jour mars 2013 - DGtal v 0.6)



1. Visualisation: contexte et historique

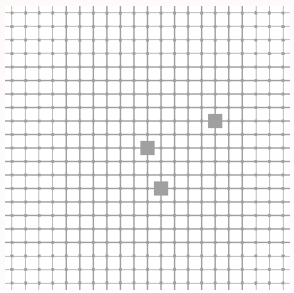
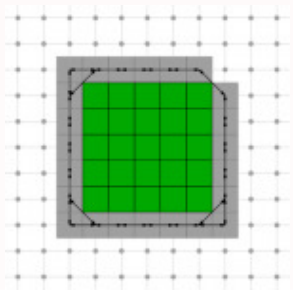
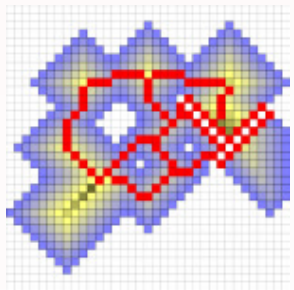
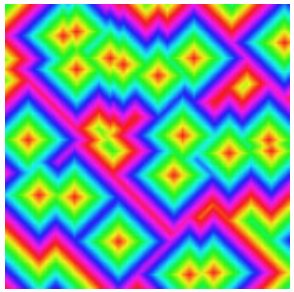
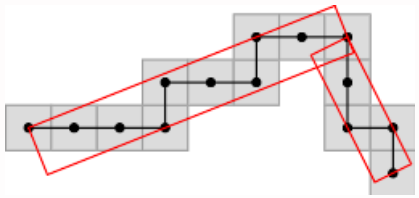
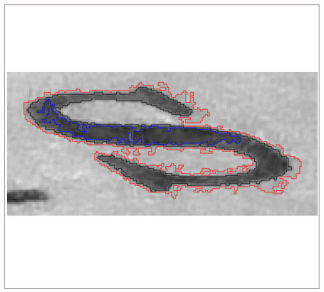
Premier *DGtal meeting* (4 janvier 2011)

- Outils de visualisation pour les primitives 2D.
- Mécanisme simple permettant de montrer des résultats liés à différents modules:
  - Topologie (Jacques-Olivier lachaud)
  - Géométrie (David Coeurjolly, Tristan Roussillon)
  - Domaine (Guillaume Damian)

1. Visualisation: contexte et historique

Premier *DGtal meeting* (4 janvier 2011)

- Outils de visualisation pour les primitives 2D.
- Mécanisme simple permettant de montrer des résultats liés à différents modules:
  - Topologie (Jacques-Olivier lachaud)
  - Géométrie (David Coeurjolly, Tristan Roussillon)
  - Domaine (Guillaume Damian)

1. Visualisation: contexte et historique

Premier *DGtal meeting* (4 janvier 2011)

- Outils de visualisation pour les primitives 2D.
- Mécanisme simple permettant de montrer des résultats liés à différents modules:
  - Topologie (Jacques-Olivier lachaud)
  - Géométrie (David Coeurjolly, Tristan Roussillon)
  - Domaine (Guillaume Damian)

Visualisation 3D (*DGtal meeting*, 29 aout-2 sept 2011)

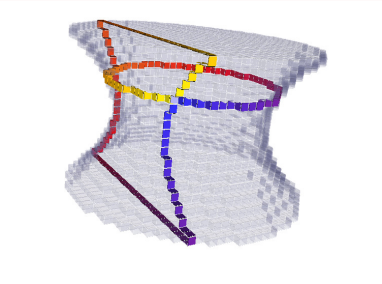
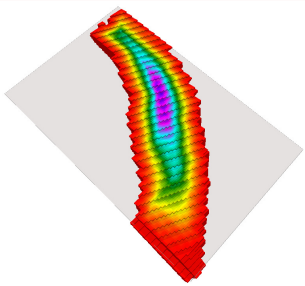
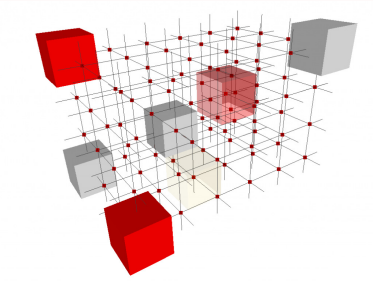
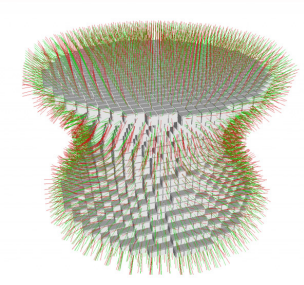
- Garder un système simple par flux (même primitive).
- Objectifs: visualisation, manipulation, édition d'objet 3D.
- Plusieurs directions:
  - Bertrand Kerautret: visualisation basée *OpenGL, LibQGLviewer*
  - Jacques-Olivier Lachaud: visualisation basée *Open Inventor SoQT*
  - Martial Tola: visualisation (2D/3D) basée sur *CAIRO* [a]

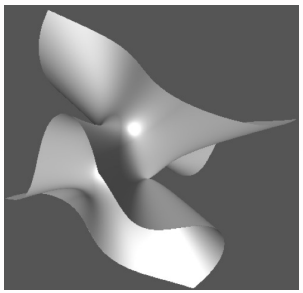
[a] Bibliothèque de rendu vectoriel: `http://cairographics.org`

1. Visualisation: contexte et historique

Premier *DGtal meeting* (4 janvier 2011)

- Outils de visualisation pour les primitives 2D.
- Mécanisme simple permettant de montrer des résultats liés à différents modules:
  - Topologie (Jacques-Olivier lachaud)
  - Géométrie (David Coeurjolly, Tristan Roussillon)
  - Domaine (Guillaume Damian)



2. Visualisation 2D: principe

Visualisation basée `Board2D` (initialement basée sur `LibBoard`[1])

Principe initial (mis à jour ensuite)

- Chaque primitive capable de s'auto dessiner.
- Vérifier le concept `CDrawableWithBoard2D` (concept vérifié avec *boost*).
- Mécanisme de "flux" avec l'opérateur <<.
- Exportation en différents formats (pdf, eps, fig, gif, png).

[1] (Copyleft, LGPL) 2007 Sébastien Fourey - GREYC ENSICAEN

2. Visualisation 2D: principe

Visualisation basée Board2D (initialement basée sur LibBoard[1])

Principe initial (mis à jour ensuite)

- Chaque primitive capable de s'auto dessiner.
- Vérifier le concept `CDrawableWithBoard2D` (concept vérifié avec *boost*).
- Mécanisme de "flux" avec l'opérateur <<.
- Exportation en différents formats (pdf, eps, fig, gif, png).

- `std::string styleName() const`
- `DrawableWithBoard2D* defaultStyle(const std::string & mode = "" ) const`
- `void selfDraw( Board2D & ) const`
- `BOOST_CONCEPT_ASSERT((CDrawableWithBoard2D<TDrawableWithBoard2D>));`

[1] (Copyleft, LGPL) 2007 Sébastien Fourey - GREYC ENSICAEN

2. Visualisation 2D: principe

Visualisation basée Board2D (initialement basée sur LibBoard[1])

Principe initial (mis à jour ensuite)

- Chaque primitive capable de s'auto dessiner.
- Vérifier le concept CDrawableWithBoard2D (concept vérifié avec *boost*).
- Mécanisme de "flux" avec l'opérateur <<.
- Exportation en différents formats (pdf, eps, fig, gif, png).

- `std::string styleName() const`
- `DrawableWithBoard2D* defaultStyle(const std::string & mode = "" ) const`
- `void selfDraw( Board2D & ) const`
- `BOOST_CONCEPT_ASSERT((CDrawableWithBoard2D<TDrawableWithBoard2D>));`

[1] (Copyleft, LGPL) 2007 Sébastien Fourey - GREYC ENSICAEN

2. Visualisation 2D: principe

Visualisation basée `Board2D` (initialement basée sur `LibBoard`[1])

Principe initial (mis à jour ensuite)

- Chaque primitive capable de s'auto dessiner.
- Vérifier le concept `CDrawableWithBoard2D` (concept vérifié avec *boost*).
- Mécanisme de "flux" avec l'opérateur <<.
- Exportation en différents formats (pdf, eps, fig, gif, png).

```
Board2D board;

board << object;
```

[1] (Copyleft, LGPL) 2007 Sébastien Fourey - GREYC ENSICAEN

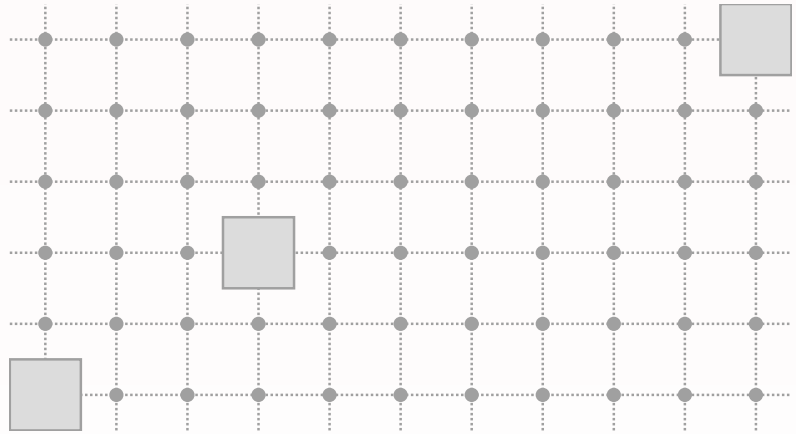
2. Visualisation 2D: principe

Visualisation basée `Board2D` (initialement basée sur `LibBoard`[2])

Principe initial (mis à jour ensuite)

- Chaque primitive capable de s'auto dessiner.
- Vérifier le concept `CDrawableWithBoard2D` (concept vérifié avec *boost*).
- Mécanisme de "flux" avec l'opérateur <<.
- Exportation en différents formats (pdf, eps, fig, gif, png).

```
using namespace DGTal::Z2i;

Point p1(-3, -2);
Point p2( 7, 3);
Point p3( 0, 0);
Domain domain (p1, p2);

Board2D board;
board << domain << p1 << p2 << p3 ;
board.saveSVG("dgtalboard-1-points.svg");
board.saveEPS("dgtalboard-1-points.eps");
board.saveTikZ("dgtalboard-1-points.tikz");
```



[2](Copyleft, LGPL) 2007 Sébastien Fourey - GREYC ENSICAEN

2. Visualisation 2D: principe

Nouvelle organisation (effectuée par Martial Tola, depuis la version 0.5)

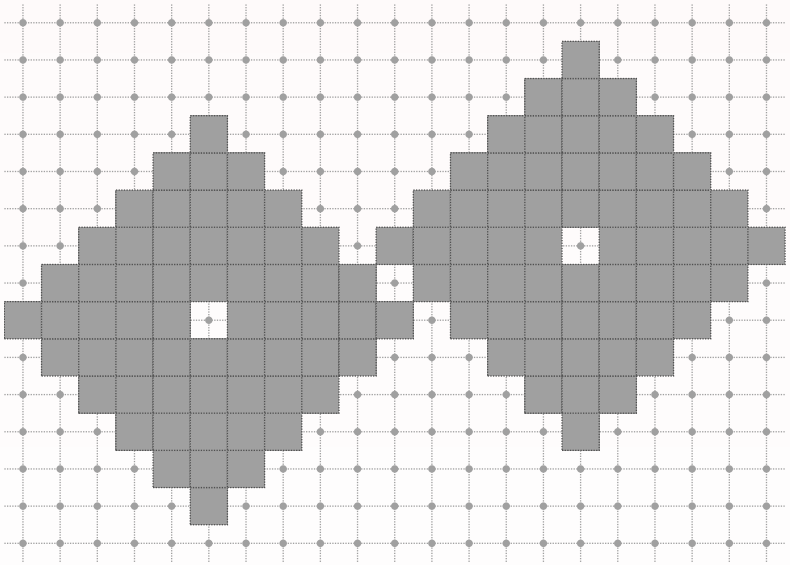
- Suppression du principe des primitives qui s'auto dessinent.
- Nouveau principe: `Display2DFactory`
- Pas de changement niveau utilisateur.
- Modification au niveau développeur.

2. Visualisation 2D: exemples

```
Point p1( -10, -7 );
Point p2( 10, 7 );
Domain domain( p1, p2 );
DigitalSet shape_set( domain );
Shapes<Domain>::addNorm1Ball( shape_set, Point( -5, -1 ), 5 );
Shapes<Domain>::addNorm1Ball( shape_set, Point( 5, 1 ), 5 );
shape_set.erase( Point( -5, -1 ) );
shape_set.erase( Point( 5, 1 ) );

Board2D board;
board << domain << shape_set; // display domain and set
board.saveSVG( "dgtalboard-2-sets-1.svg");
board.saveTikZ( "dgtalboard-2-sets-1.tikz");
```

2. Visualisation 2D: exemples

```
Point p1( -10, -7 );
Point p2( 10, 7 );
Domain domain( p1, p2 );
DigitalSet shape_set( domain );
Shapes<Domain>::addNorm1Ball( shape_set, Point( -5, -1 ), 5 );
Shapes<Domain>::addNorm1Ball( shape_set, Point( 5, 1 ), 5 );
shape_set.erase( Point( -5, -1 ) );
shape_set.erase( Point( 5, 1 ) );

Board2D board;
board << domain << shape_set; // display domain and set
board.saveSVG( "dgtalboard-2-sets-1.svg");
board.saveTikZ( "dgtalboard-2-sets-1.tikz");
```



2. Visualisation 2D: mode d'affichage

Changement du mode d'affichage

Possibilité de choisir un mode d'affichage spécifique:

```
board << SetMode( monObjet.className(), "DrawAdjacencies" )
```

Example (1):

```
Point p1( -10, -7 );
Point p2( 10, 7 );
Domain domain( p1, p2 );
DigitalSet shape_set( domain );
Shapes<Domain>::addNorm1Ball( shape_set, Point( -5, -1 ), 5 );
Shapes<Domain>::addNorm1Ball( shape_set, Point( 5, 1 ), 5 );
shape_set.erase( Point( -5, -1 ) );
shape_set.erase( Point( 5, 1 ) );

Board2D board;
Object4_8 shape( dt4_8, shape_set );
board << domain // display domain
      << SetMode( shape.className(), "DrawAdjacencies" )
      << shape; // and object with mode "DrawAdjacencies"
```

2. Visualisation 2D: mode d'affichage

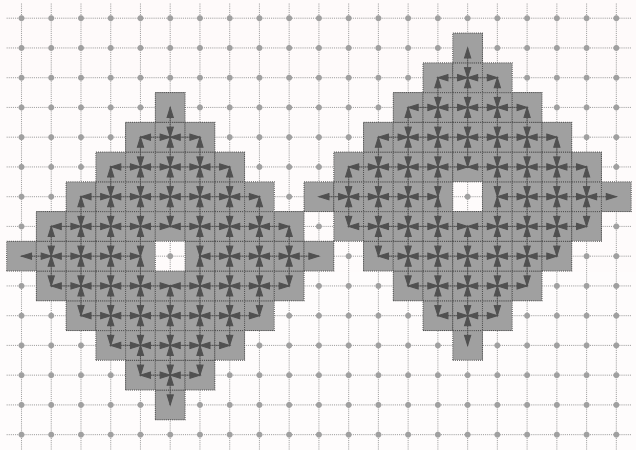
Changement du mode d'affichage

Possibilité de choisir un mode d'affichage spécifique:

```
board << SetMode( monObjet.className(), "DrawAdjacencies" )
```

Example (1):

```
Point p1( -10, -7 );
Point p2( 10, 7 );
Domain domain( p1, p2 );
DigitalSet shape_set( domain );
Shapes<Domain>::addNorm1Ball( shape_set, Point( -5, -1 ), 5 );
Shapes<Domain>::addNorm1Ball( shape_set, Point( 5, 1 ), 5 );
shape_set.erase( Point( -5, -1 ) );
shape_set.erase( Point( 5, 1 ) );

Board2D board;
Object4_8 shape( dt4_8, shape_set );
board << domain // display domain
      << SetMode( shape.className(), "DrawAdjacencies" )
      << shape; // and object with mode "DrawAdjacencies"
```



2. Visualisation 2D: mode d'affichage

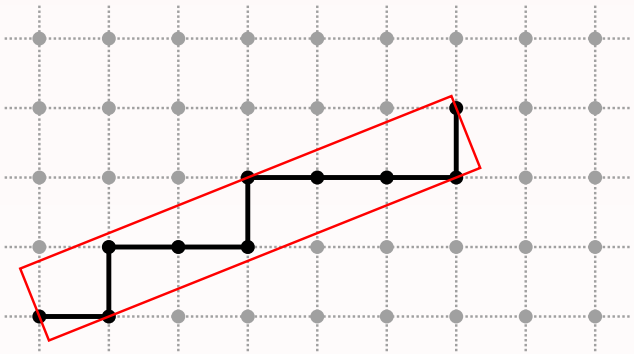
Changement du mode d'affichage

Possibilité de choisir un mode d'affichage spécifique:

```
board << SetMode( monObjet.className(), "DrawAdjacencies" )
```

Example (2):

```
// Draw the points of the DSS
board << SetMode("PointVector", "Grid")
<< SetMode(theDSS4.className(), "Points")
<< theDSS4;
// Draw the bounding box
board << SetMode(theDSS4.className(), "BoundingBox")
<< theDSS4;
```



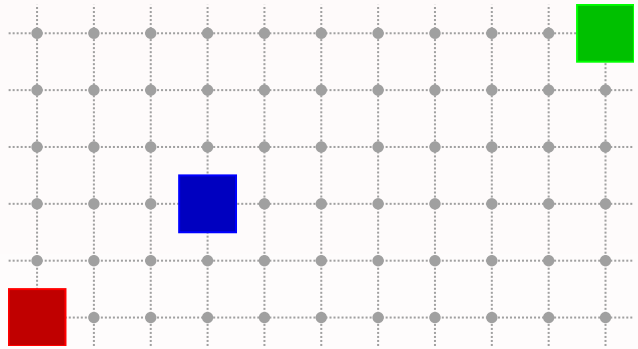
2. Visualisation 2D: mode d'affichage

Mode d'affichage personnalisé

- Possibilité de fabrique son propre style.
- Personnaliser des attributs de couleur/tracé.

Example:

```
Board2D board;
board << domain
      << CustomStyle(p1.className(), new CustomColors(red, dred))
      << p1
      << CustomStyle(p2.className(), new CustomFillColor(dgreen))
      << p2
      << CustomStyle(p3.className(),
                     new CustomPen( blue, dblue, 3.0,
                     Board2D::Shape::SolidStyle,
                     Board2D::Shape::RoundCap,
                     Board2D::Shape::RoundJoin ) )
      << p3;
```



3. Principe de la visualisation 3D

Mécanisme comparable en 3D:

- Classe semie abstraite: `Display3D`
- Même principe: opérateur << et concept `CDrawableWithDisplay3D`
- Deux classes héritent de `Display3D`: `Viewer3D` et `Board3DTo2D`

3. Principe de la visualisation 3D

Mécanisme comparable en 3D:

- Classe semie abstraite: `Display3D`
- Même principe: opérateur << et concept `CDrawableWithDisplay3D`
- Deux classes héritent de `Display3D`: `Viewer3D` et `Board3DTo2D`

Visualisation basée *OpenGL*: `Viewer3D`

- Utilisation de la bibliothèque *LibQLViewer* [a]
- Hérite à la fois de `Display3D` et `QGLViewer`.

\+ Visualisation 3D interactive (possibilité de sélectionner des objets 3D).

\+ Fonctionnement simple.

\+ Toujours maintenue depuis 2005 (dernière version mai 2012).

\- Dépendance avec *QT*.

\- Export vectoriel possible (théoriquement mais limité).

[a] `http://www.libqglviewer.com`

3. Principe de la visualisation 3D

Mécanisme comparable en 3D:

- Classe semie abstraite: `Display3D`
- Même principe: opérateur << et concept `CDrawableWithDisplay3D`
- Deux classes héritent de `Display3D`: `Viewer3D` et `Board3DTo2D`

Visualisation basée *OpenGL*: `Viewer3D`

- Utilisation de la bibliothèque *LibQLViewer* [a]
- Hérite à la fois de `Display3D` et `QGLViewer`.

\+ Visualisation 3D interactive (possibilité de sélectionner des objets 3D).

\+ Fonctionnement simple.

\+ Toujours maintenue depuis 2005 (dernière version mai 2012).

\- Dépendance avec *QT*.

\- Export vectoriel possible (théoriquement mais limité).

[a] `http://www.libqglviewer.com`

3. Principe de la visualisation 3D

Mécanisme comparable en 3D:

- Classe semie abstraite: `Display3D`
- Même principe: opérateur << et concept `CDrawableWithDisplay3D`
- Deux classes héritent de `Display3D`: `Viewer3D` et `Board3DTo2D`

Visualisation basée *CAIRO*: `Board3DTo2D` (Martial Tola)

- Même principe mais pas d'interaction.
+ Pas de dépendance *QT*.
+ Export qualité vectoriel.
- Positionnement manuel de la caméra.
- Potentiellement moins riche comparé à *OpenGL*.

3. Principe de la visualisation 3D

Classes vérifiant le concept `CDrawableWithDisplay3D`

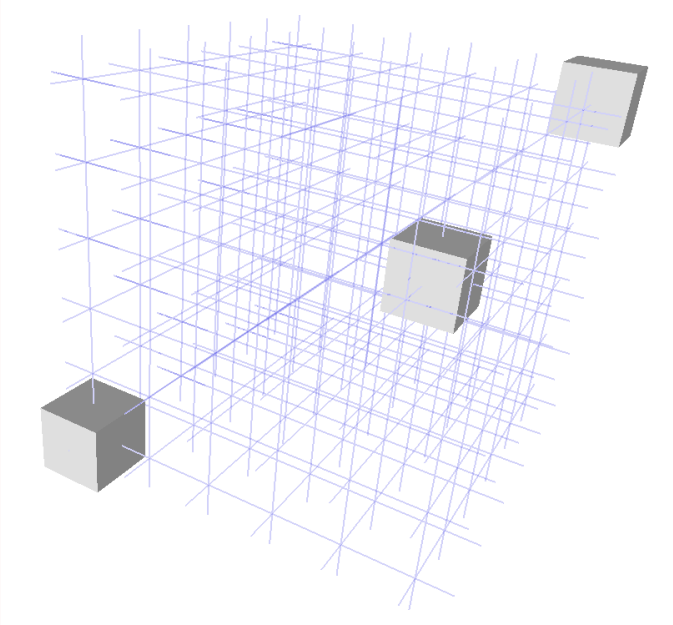
Classe	3DViewer	Board3DTo2D
`PointVector`	X	X
`DigitalSetBySTLSet`	X	X
`DigitalSetBySTLVector`	X	X
`Object`	X	X
`HyperRectDomain`	X	X
`KhalimskyCell`	X	X
`SignedKhalimskyCell`	X	X
`ArithmeticalDSS3d`	X	X
`MeshFromPoints`	X	-

X: depuis DGTal v. 0.5
X: depuis DGtal V. 0.6
X: partiellement

Exemple de visualisations 3D

Visualisation basée Viewer3D

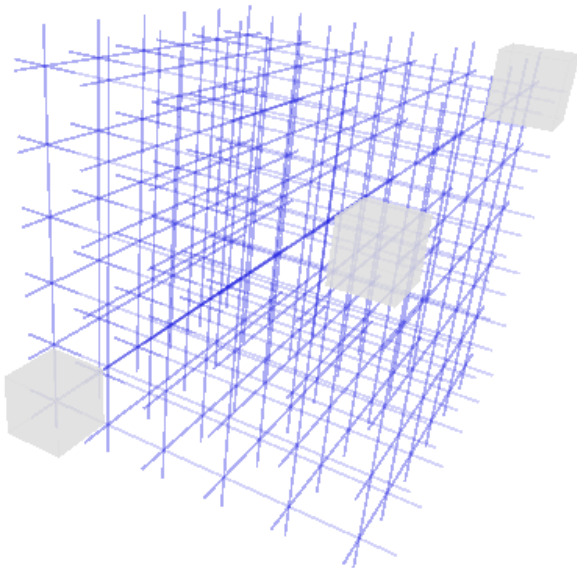
```
using namespace Z3i;

QApplication application(argc,argv);

Point p1( 0, 0, 0 );
Point p2( 5, 5 ,5 );
Point p3( 2, 3, 4 );
Domain domain( p1, p2 );

Viewer3D viewer;
viewer.show();
viewer << domain;
viewer << p1 << p2 << p3;
viewer<< Viewer3D::updateDisplay;

return application.exec();
```



Exemple de visualisations 3D

Visualisation basée Board3DTo2D

```
using namespace Z3i;

Point p1( 0, 0, 0 );
Point p2( 5, 5 ,5 );
Point p3( 2, 3, 4 );
Domain domain( p1, p2 );

Board3DTo2D viewer;

viewer << domain;
viewer << p1 << p2 << p3;

viewer << CameraPosition( -7.12609, 6.91577, 6.86312)
       << CameraDirection( 0.814587, -0.426381, -0.393252)
       << CameraUpVector(0.335923, 0.899486, -0.279428);
viewer << CameraZNearFar(3.9399, 18.9399);

viewer.saveCairo("dgtalCairo-1-points.pdf",
                 Board3DTo2D::CairoPDF, 600, 400);
```



4. Modes de visualisation 3D

Modification du style d'affichage

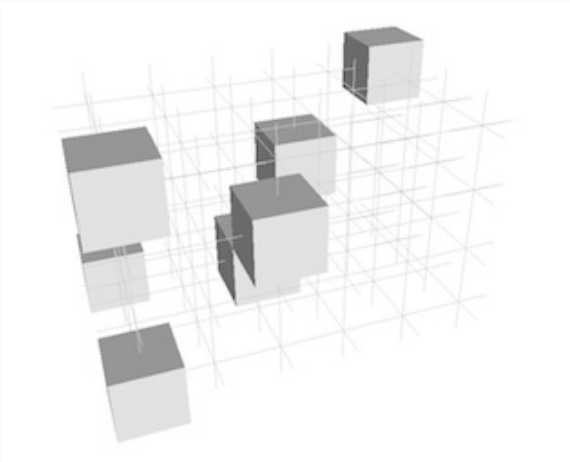
Définie pour chaque primitive:

```
viewer << SetMode3D( p1.styleName(), "Grid" );
```

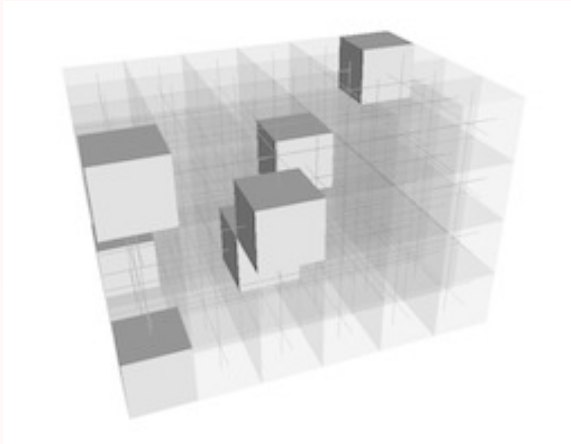
Différents modes possibles:

Classe	modes
PointVector	"Paving" (defaut), "Grid"
DigitalSetBySTLSet	"Paving" (default), "PavingTransp", "Grid"
DigitalSetBySTLVector	"Paving" (default), "PavingTransp", "Grid"
Object	"DrawAdjacencies", "PavingTransp"
HyperRectDomain	"Grid" (default), "Paving", "PavingPoints", "PavingGrids", "BoundingBox"
KhalimskyCell	"Highlighted" ,"Transparent", "Basic"
ArithmeticalDSS3d	"Points" ,"BoundingBox".
MeshFromPoints	"Faces".

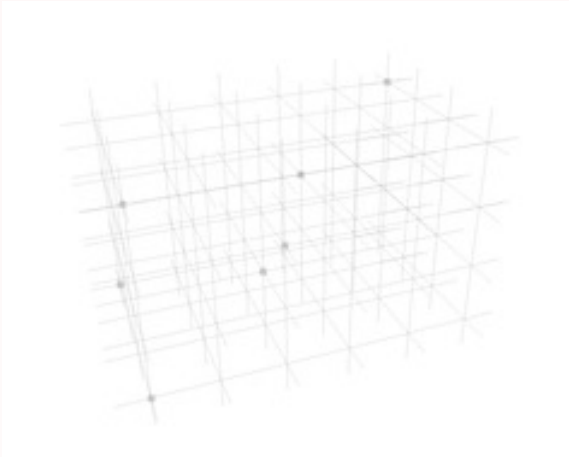
4. Modes de visualisation 3D



mode "" (Default)



mode "PavingGrids"

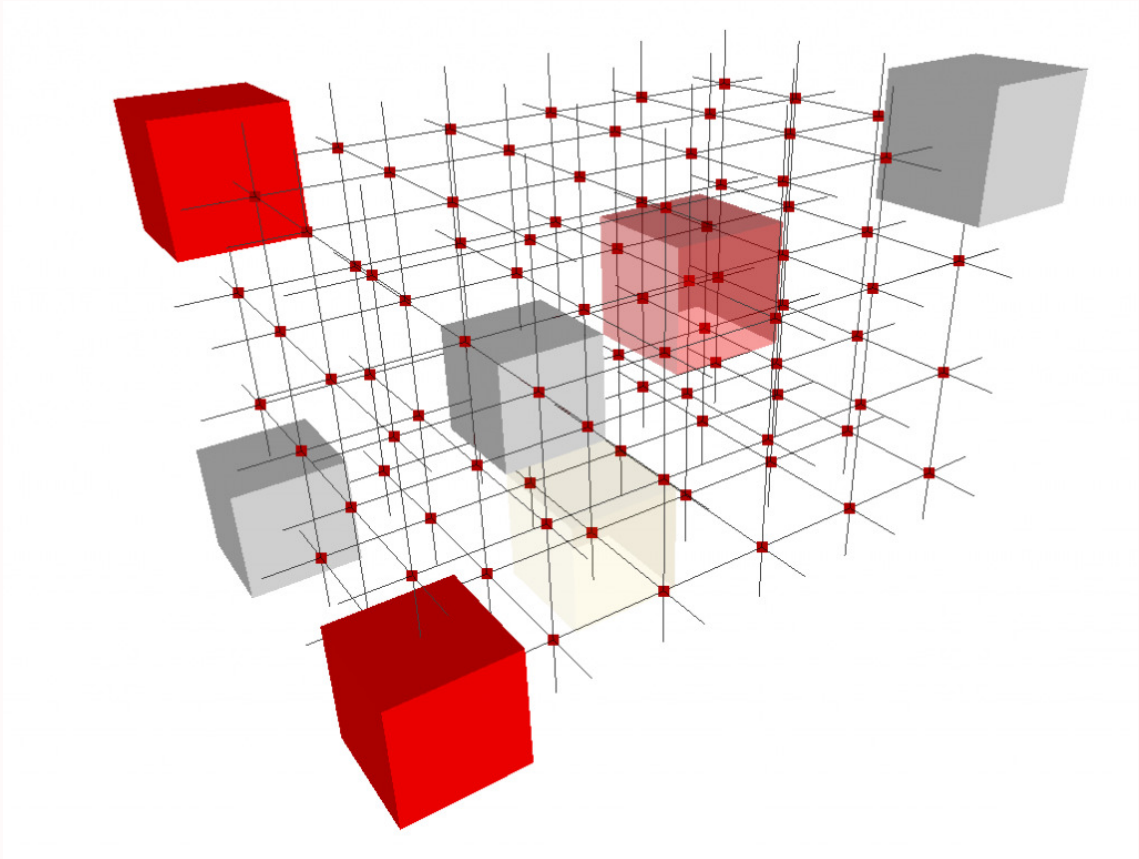


mode "Grid"

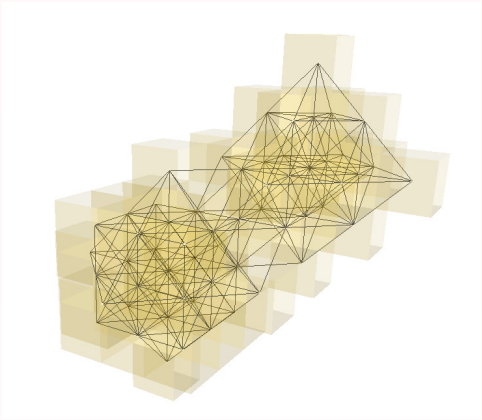
4. Modes de visualisation 3D

Visualisation personnalisée

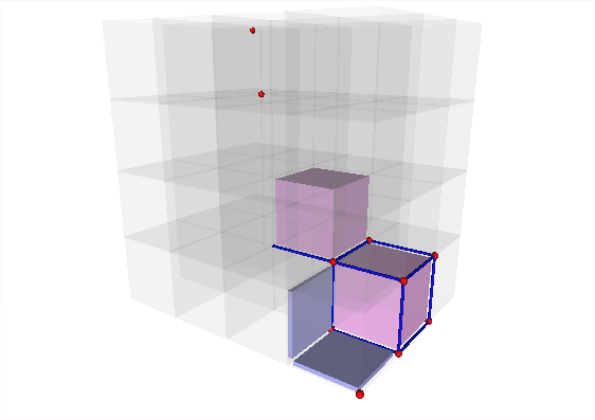
```
viewer << CustomColors3D(Color(250, 0,0),Color(250, 0,0));
```



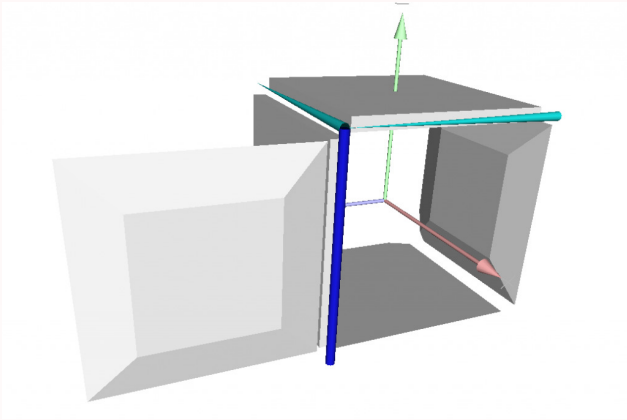
Exemples sur d'autres primitives



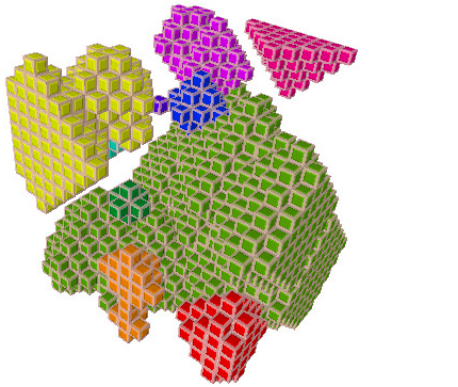
`Object` mode "DrawAdjacencies"



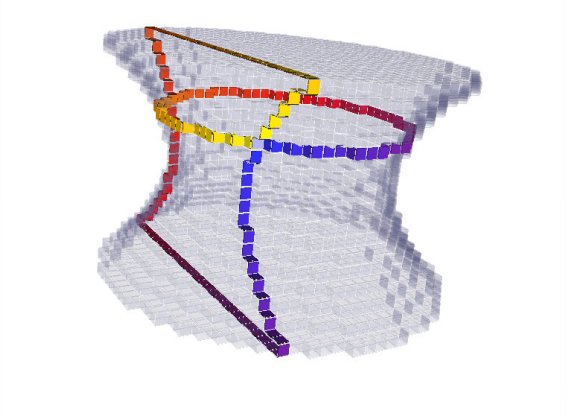
`Khalimsky` Cells



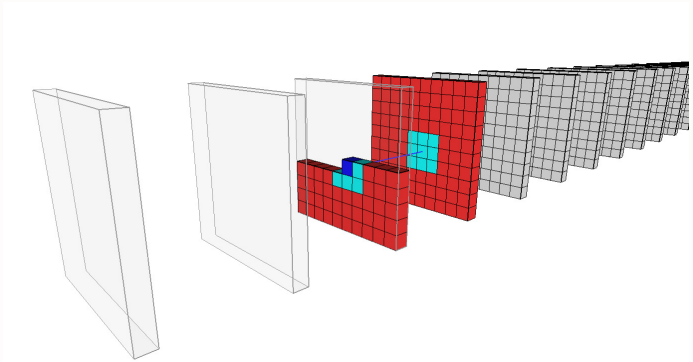
Signed `Khalimsky` Cells



Composantes connexes



Suivi de surfels

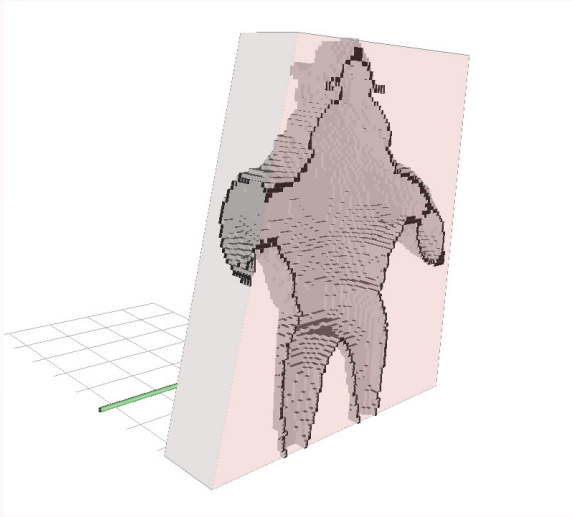


Autres fonctionnalités

- Clipping planes:

```
viewer << ClippingPlane(1,0,0,-4.9);
viewer << ClippingPlane(0,1,0.3,-10);
```

- Importation de données simplifiée (visu volumique en moins de 50 lignes)..
- Gestion de la transparence.

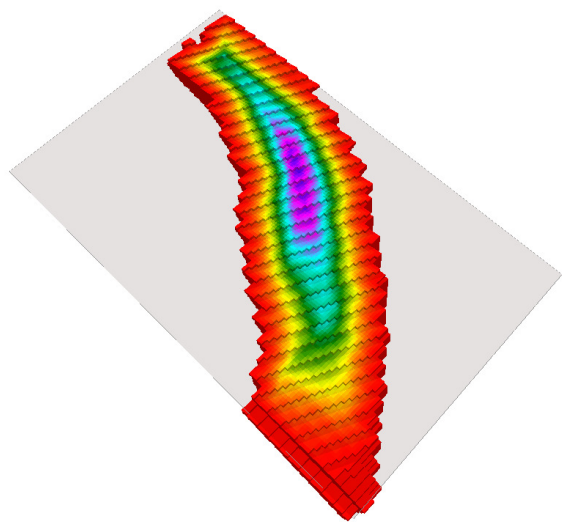


Autres fonctionnalités

- Clipping planes:

```
viewer << ClippingPlane(1,0,0,-4.9);
viewer << ClippingPlane(0,1,0.3,-10);
```

- Importation de données simplifiée (visu volumique en moins de 50 lignes)..
- Gestion de la transparence.

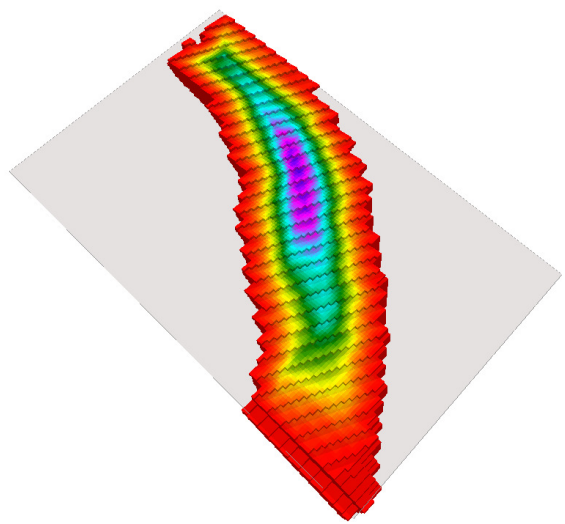


Autres fonctionnalités

- Clipping planes:

```
viewer << ClippingPlane(1,0,0,-4.9);
viewer << ClippingPlane(0,1,0.3,-10);
```

- Importation de données simplifiée (visu volumique en moins de 50 lignes)..
- Gestion de la transparence.

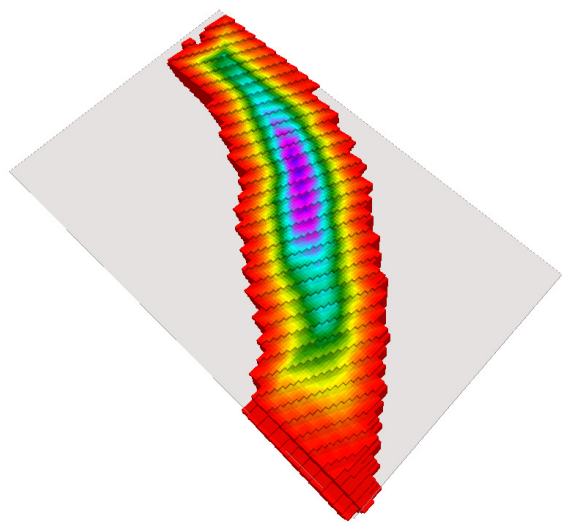


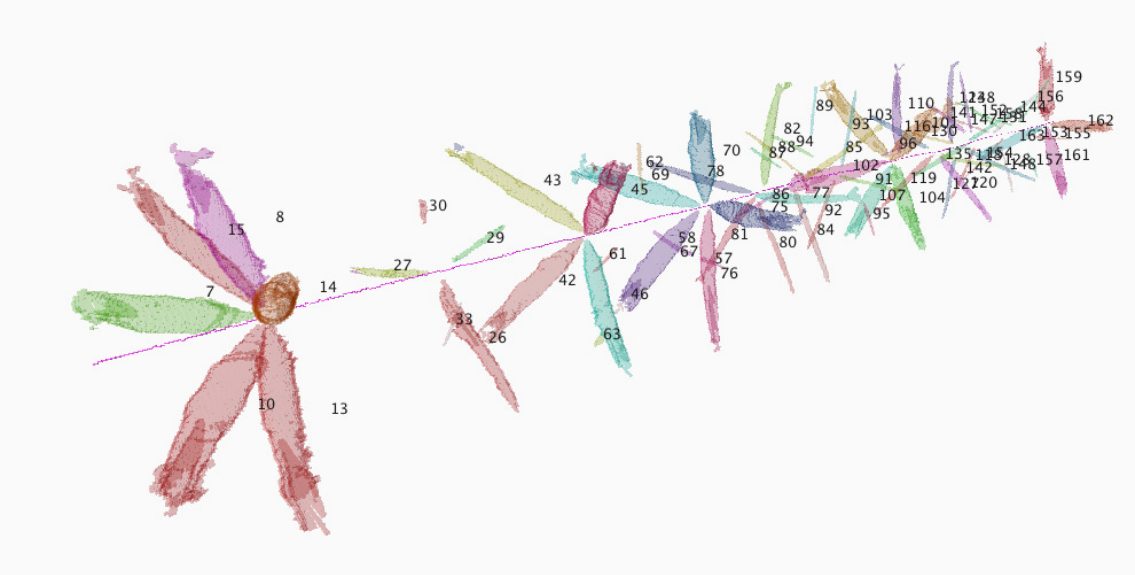
Autres fonctionnalités

- Clipping planes:

```
viewer << ClippingPlane(1,0,0,-4.9);
viewer << ClippingPlane(0,1,0.3,-10);
```

- Importation de données simplifiée (visu volumique en moins de 50 lignes)..
- Gestion de la transparence.





Exemple du code de l'outil de visualisation:

```
00079
00080  QApplication application(argc,argv);
00081  Viewer3D viewer;
00082  viewer.setWindowTitle("simple Volume Viewer");
00083  viewer.show();
00084
00085  typedef ImageSelector<Domain, unsigned char>::Type Image;
00086  Image image = VolReader<Image>::importVol( inputFilename );
00087
00088  trace.info() << "Image loaded: "<<image<< std::endl;
00089
00090  Domain domain(image.lowerBound(), image.upperBound());
00091  GradientColorMap<long> gradient( thresholdMin, thresholdMax);
00092  gradient.addColor(Color::Blue);
00093  gradient.addColor(Color::Green);
00094  gradient.addColor(Color::Yellow);
00095  gradient.addColor(Color::Red);
00096  for(Domain::ConstIterator it = domain.begin(), itend=domain.end(); it!=itend; ++it){
00097    unsigned char  val= image( (*it) );
00098
00099    Color c= gradient(val);
00100    if(val<=thresholdMax && val >=thresholdMin){
00101      viewer <<  CustomColors3D(Color((float)(c.red()), (float)(c.green()),(float)(c.blue()), transp),
00102                                Color((float)(c.red()), (float)(c.green()),(float)(c.blue()), transp));
00103      viewer << *it;
00104    }
00105  }
00106  //viewer << ClippingPlane(0,0,-1, 20);
00107  viewer << Viewer3D::updateDisplay;
00108  return application.exec();
```

6. Nouveautés depuis la version 0.6 (1)

- Import de maillages: `MeshReader` (format OFF, OFS).

```
#include "DGtal/io/readers/MeshReader.h"
#include <QtGui/qapplication.h>
#include "DGtal/io/Display3D.h"
#include "DGtal/io/viewers/Viewer3D.h"
...
std::string inputFilename = examplesPath + "samples/tref.off";
...
// Import du maillage:
MeshFromPoints<Display3D::pointD3D> anImportedMesh;
anImportedMesh << inputFilename;

// Visualisation:
viewer << anImportedMesh;
viewer << Viewer3D::updateDisplay;
```

- Export 3D (OBJ et OFF format) directement avec l'opérateur >>

```
Display3D viewer;
typedef ImageSelector < Z3i::Domain, int>::Type Image;
// Import d'un DGtalSet
Image image = VolReader<Image>::importVol(inputFilename);
Z3i::DigitalSet set3d (image.domain());
SetFromImage<Z3i::DigitalSet>::append<Image>(set3d, image, 0,255);
viewer << set3d ;
// Export en OFF format:
viewer >> "exportMeshToOFF.off";
```

6. Nouveautés depuis la version 0.6 (1)

- Import de maillages: `MeshReader` (format OFF, OFS).

```
#include "DGtal/io/readers/MeshReader.h"
#include <QtGui/qapplication.h>
#include "DGtal/io/Display3D.h"
#include "DGtal/io/viewers/Viewer3D.h"
...
std::string inputFilename = examplesPath + "samples/tref.off";
...
// Import du maillage:
MeshFromPoints<Display3D::pointD3D> anImportedMesh;
anImportedMesh << inputFilename;

// Visualisation:
viewer << anImportedMesh;
viewer << Viewer3D::updateDisplay;
```

- Export 3D (OBJ et OFF format) directem

```
Display3D viewer;
typedef ImageSelector < Z3i::Domain, int>::Type Image
// Import d'un DGtalSet
Image image = VolReader<Image>::importVol(inputFilena
Z3i::DigitalSet set3d (image.domain());
SetFromImage<Z3i::DigitalSet>::append<Image>(set3d, i
viewer << set3d ;
// Export en OFF format:
viewer >> "exportMeshToOFF.off";
```

6. Nouveautés depuis la version 0.6 (1)

- Import de maillages: `MeshReader` (format OFF, OFS).

```
#include "DGtal/io/readers/MeshReader.h"
#include <QtGui/qapplication.h>
#include "DGtal/io/Display3D.h"
#include "DGtal/io/viewers/Viewer3D.h"
...
std::string inputFilename = examplesPath + "samples/tref.off";
...
// Import du maillage:
MeshFromPoints<Display3D::pointD3D> anImportedMesh;
anImportedMesh << inputFilename;

// Visualisation:
viewer << anImportedMesh;
viewer << Viewer3D::updateDisplay;
```

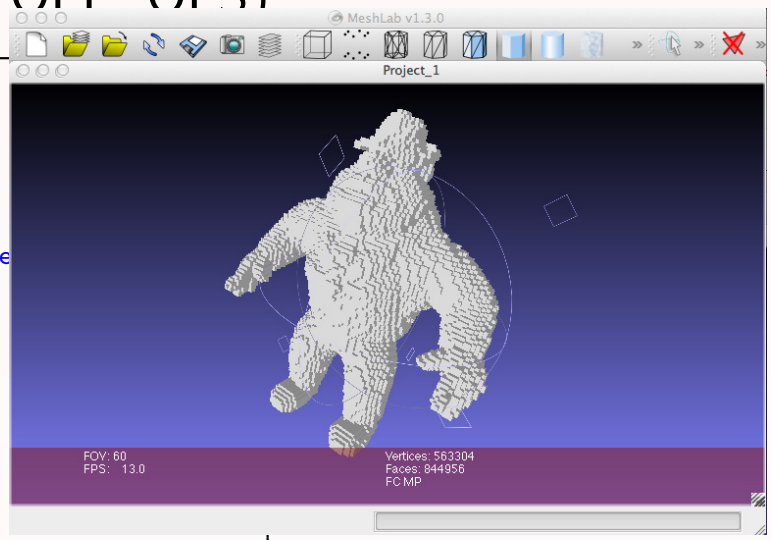
- Export 3D (OBJ et OFF format) directement avec l'opérateur >>

```
Display3D viewer;
typedef ImageSelector < Z3i::Domain, int>::Type Image;
// Import d'un DGtalSet
Image image = VolReader<Image>::importVol(inputFilename);
Z3i::DigitalSet set3d (image.domain());
SetFromImage<Z3i::DigitalSet>::append<Image>(set3d, image, 0,255);
viewer << set3d ;
// Export en OFF format:
viewer >> "exportMeshToOFF.off";
```

6. Nouveautés depuis la version 0.6 (1)

- Import de maillages: `MeshReader` (format OFF, OFS)

```
#include "DGtal/io/readers/MeshReader.h"
#include <QtGui/qapplication.h>
#include "DGtal/io/Display3D.h"
#include "DGtal/io/viewers/Viewer3D.h"
...
std::string inputFilename = examplesPath + "samples/tre
...
// Import du maillage:
MeshFromPoints<Display3D::pointD3D> anImportedMesh;
anImportedMesh << inputFilename;

// Visualisation:
viewer << anImportedMesh;
viewer << Viewer3D::updateDisplay;
```



- Export 3D (OBJ et OFF format) directement avec l'opérateur >>

```
Display3D viewer;
typedef ImageSelector < Z3i::Domain, int>::Type Image;
// Import d'un DGtalSet
Image image = VolReader<Image>::importVol(inputFilename);
Z3i::DigitalSet set3d (image.domain());
SetFromImage<Z3i::DigitalSet>::append<Image>(set3d, image, 0,255);
viewer << set3d ;
// Export en OFF format:
viewer >> "exportMeshToOFF.off";
```

6. Nouveautés depuis la version 0.6 (1)

- Import de maillages: `MeshReader` (format OFF, OFS).

```
#include "DGtal/io/readers/MeshReader.h"
#include <QtGui/qapplication.h>
#include "DGtal/io/Display3D.h"
#include "DGtal/io/viewers/Viewer3D.h"
...
std::string inputFilename = examplesPath + "samples/tref.off";
...
// Import du maillage:
MeshFromPoints<Display3D::pointD3D> anImportedMesh;
anImportedMesh << inputFilename;

// Visualisation:
viewer << anImportedMesh;
viewer << Viewer3D::updateDisplay;
```

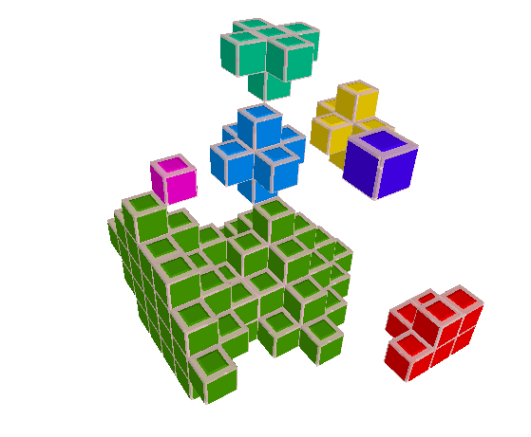
- Export 3D (OBJ et OFF format) directement avec l'opérateur >>

```
Display3D viewer;
typedef ImageSelector < Z3i::Domain, int>::Type Image;
// Import d'un DGtalSet
Image image = VolReader<Image>::importVol(inputFilename);
Z3i::DigitalSet set3d (image.domain());
SetFromImage<Z3i::DigitalSet>::append<Image>(set3d, image, 0,255);
viewer << set3d ;
// Export en OFF format:
viewer >> "exportMeshToOFF.off";
```

⇒ L'import/export ne nécessite aucune dépendance (ni QGLViewer/QT ni Cairo)

6. Nouveautés depuis la version 0.6 (2)

- Export possible (indirect) en u3d (génération pdf3D).

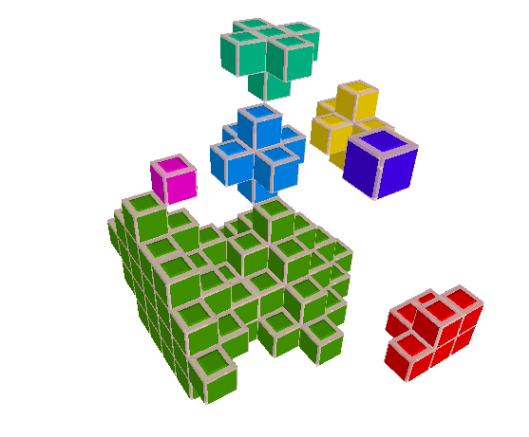


⇒ export OFF, puis u3d par *MeshLab* ou *jreality*

- Export en HTML5:

6. Nouveautés depuis la version 0.6 (2)

- Export possible (indirect) en `u3d` (génération pdf3D).



⇒ export OFF, puis `u3d` par *MeshLab* ou *jreality*

- Export en HTML5:
  `http://kerrecherche.iutsd.uhp-nancy.fr/TestWeb3D/test.html`

7. Nouveauté à intégrer dans une future version 0.7

- Rajouter des modèles de réflectance/éclairage dans `3DViewer` et `Board2DTo3D`.
- Finaliser la visualisation de MeshFromPoints dans `Board3DTo2D`.
- Visualisation basée OGRE (Pull Request de la branche OGRE3D de GitHub).