

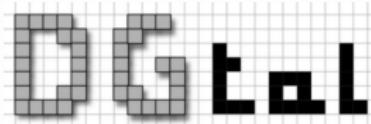
Flux, visualisation 3D

Bertrand Kerautret^{1,2}

¹LORIA - Nancy University

²LAMA - University of Savoie

Journées DGtal, 29 aout-2 septembre 2011



1. Visualisation 3D : contexte

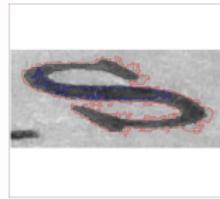
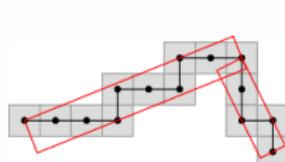
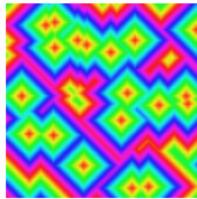
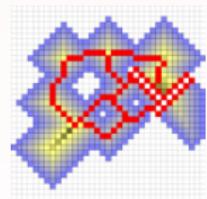
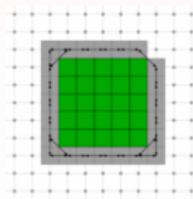
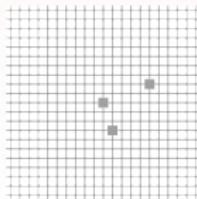
Premier *DGtal meeting* (4 janvier 2011)

- 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 3D : contexte

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 3D : contexte

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)

Objectifs pour la visualisation 3D

- Garder un système simple par flux (même primitive).
- 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 3D : objectif

Inspiré du mécanisme de Board2D (initialement basé sur LibBoard¹)

- 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. Visualisation 3D : objectif

Inspiré du mécanisme de Board2D (initialement basé sur LibBoard¹)

- 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. Visualisation 3D : objectif

Inspiré du mécanisme de Board2D (initialement basé sur LibBoard¹)

- 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. Visualisation 3D : objectif

Inspiré du mécanisme de Board2D (initialement basé sur LibBoard¹)

- 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. Visualisation 3D : objectif

Inspiré du mécanisme de Board2D (initialement basé sur LibBoard¹)

- 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. Visualisation 3D : objectif

Inspiré du mécanisme de Board2D (initialement basé sur LibBoard¹)

- 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 );

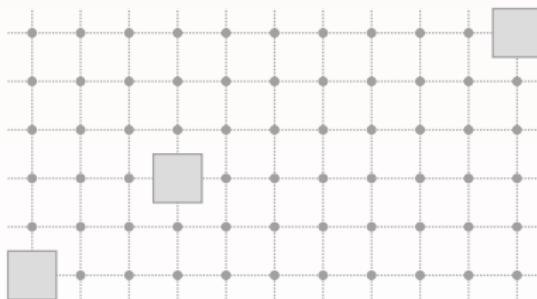
Board2D board;
board << domain << p1 << p2 << p3;
board.saveSVG("dgtalboard-1-points.svg");
board.saveEPS("dgtalboard-1-points.eps");
```

1. Visualisation 3D : objectif

Inspiré du mécanisme de Board2D (initialement basé sur LibBoard¹)

- 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");
```



2. Principe de la visualisation 3D

Mécanisme comparable en 3D :

- Classe semi abstraite : `Display3D`
- Même principe : opérateur << et concept `CDrawableWithDisplay3D`
- Deux classes héritent de `Display3D` : `Viewer3D` et `Board3D2D`

2. Principe de la visualisation 3D

Mécanisme comparable en 3D :

- Classe semi abstraite : `Display3D`
- Même principe : opérateur << et concept `CDrawableWithDisplay3D`
- Deux classes héritent de `Display3D` : `Viewer3D` et `Board3D2D`

Visualisation basée *OpenGL* : `Viewer3D`

- Utilisation de la bibliothèque *LibQGLViewer*^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 Juin 2010).
 - Dépendance avec *QT*.
 - Export vectoriel possible (théoriquement mais limité).

a. <http://www.libqglviewer.com>

2. Principe de la visualisation 3D

Mécanisme comparable en 3D :

- Classe semi abstraite : `Display3D`
- Même principe : opérateur << et concept `CDrawableWithDisplay3D`
- Deux classes héritent de `Display3D` : `Viewer3D` et `Board3D2D`

Visualisation basée *OpenGL* : `Viewer3D`

- Utilisation de la bibliothèque *LibQGLViewer*^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 Juin 2010).
- Dépendance avec *QT*.
- Export vectoriel possible (théoriquement mais limité).

a. <http://www.libqglviewer.com>

2. Principe de la visualisation 3D

Mécanisme comparable en 3D :

- Classe semi abstraite : `Display3D`
- Même principe : opérateur << et concept `CDrawableWithDisplay3D`
- Deux classes héritent de `Display3D` : `Viewer3D` et `Board3D2D`

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

- Même principe mais pas d'interaction.
- + Pas de dépendance *QT*/carte 3D.
- + Export qualité vectoriel.
 - Positionnement manuel de la caméra.
 - Potentiellement moins riche comparé à *OpenGL*.

2. Principe de la visualisation 3D

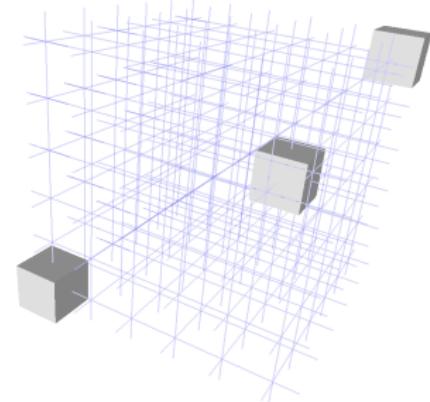
Classes vérifiant le concept `CDrawableWithDisplay3D`

Classe	3DViewer	Board3DTo2D
<code>PointVector</code>	X	X
<code>DigitalSetBySTLSet</code>	X	X
<code>DigitalSetBySTLVector</code>	X	X
<code>Object</code>	X	X
<code>HyperRectDomain</code>	X	X
<code>KhalimskyCell</code>	X	bientôt
<code>SignedKhalimskyCell</code>	X	bientôt

Exemple de visualisations 3D

Visualisation basée Viewer3D

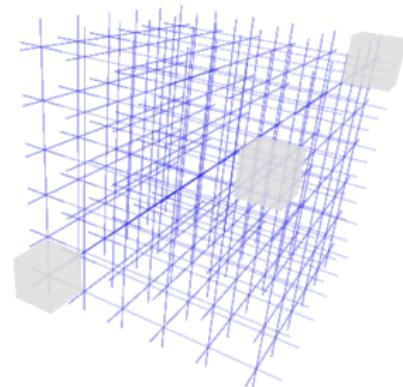
```
using namespace Z3i;  
  
QApplication application(argc, argv);  
  
Point p1( 0, 0, 0 );  
Point p2( 5, 5 ,5 );  
Point p3( 2, 3, 4 );  
Domain domain( p1, p2 );  
  
Viewer3D viewer;  
viewer.show();  
viewer << domain;  
viewer << p1 << p2 << p3;  
viewer << Viewer3D::updateDisplay;  
  
return 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);
```



3. Modes de visualisation 3D

Modification du style d'affichage

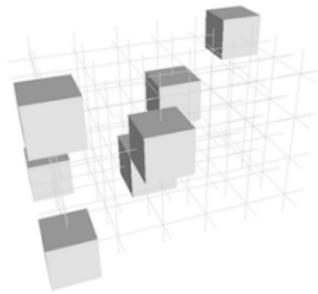
Définie pour chaque primitive :

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

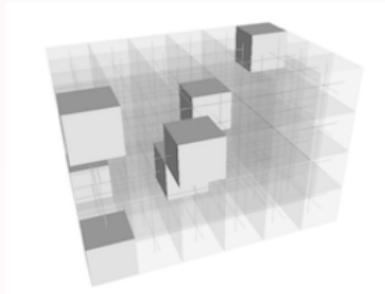
Différents modes possibles :

Classe	modes
PointVector	"Paving" (default), "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"
SignedKhalimskyCell	"Highlighted", "Transparent", "Basic".

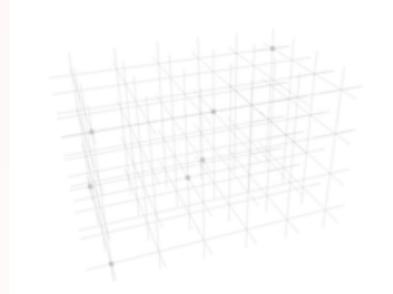
3. Modes de visualisation 3D



mode "" (Default)



mode "PavingGrids"

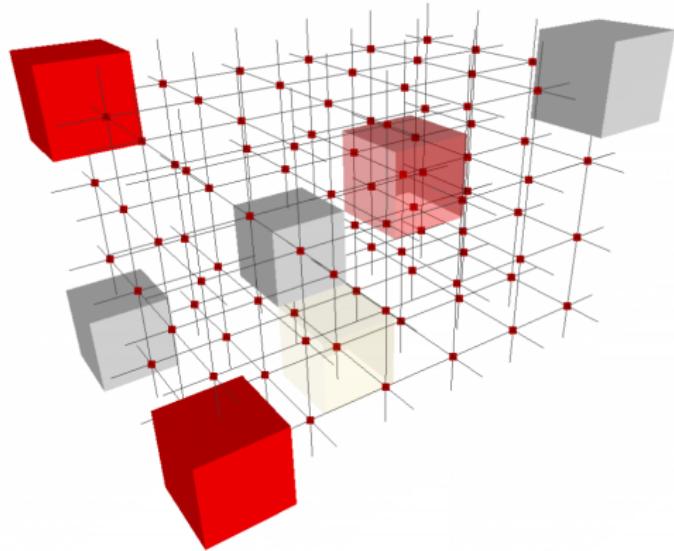


mode "Grid"

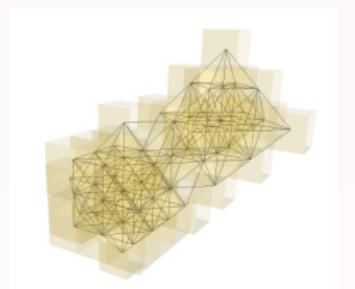
3. Modes de visualisation 3D

Visualisation personnalisée

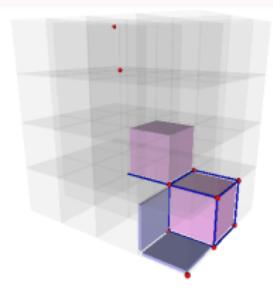
```
viewer << CustomColors3D(Color(250, 0,0),Color(250, 0,0));
```



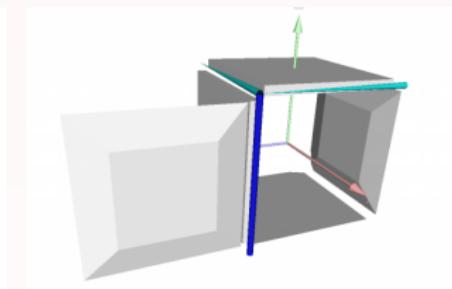
Exemples sur d'autres primitives



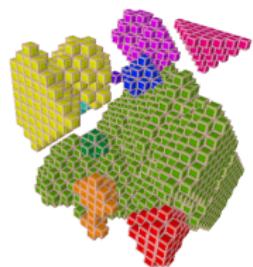
Object mode "DrawAdjacencies"



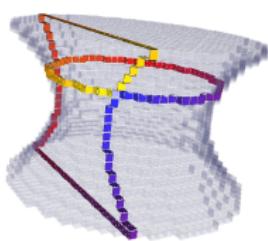
Khalimsky Cells



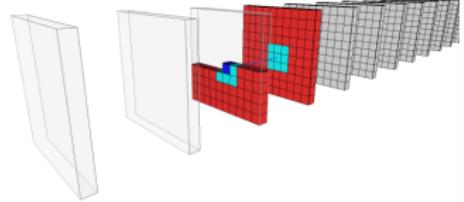
Signed Khalimsky Cells



Composantes connexes



Suivi de surfels

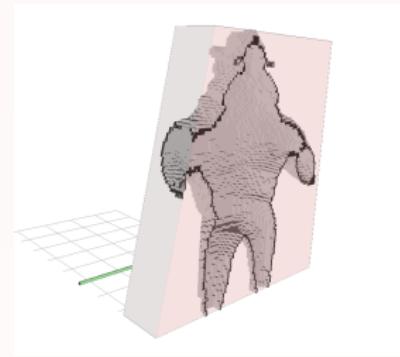


Autres fonctionnalités

- Clipping planes :

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

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

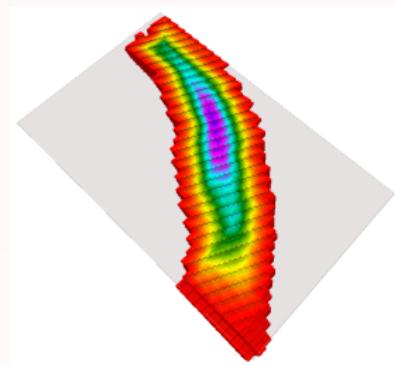


Autres fonctionnalités

- Clipping planes :

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

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

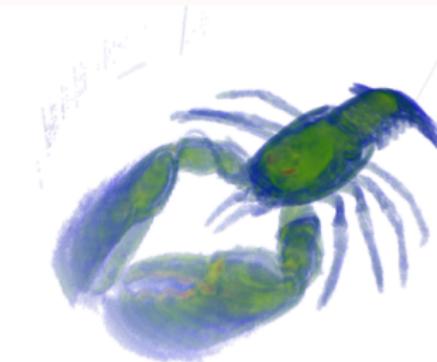
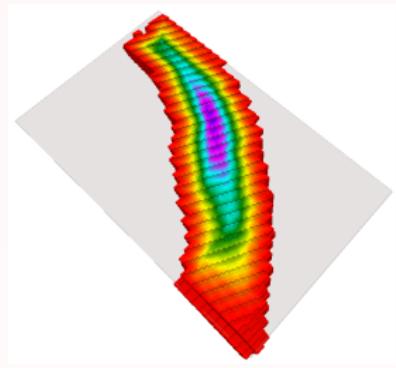


Autres fonctionnalités

- 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.

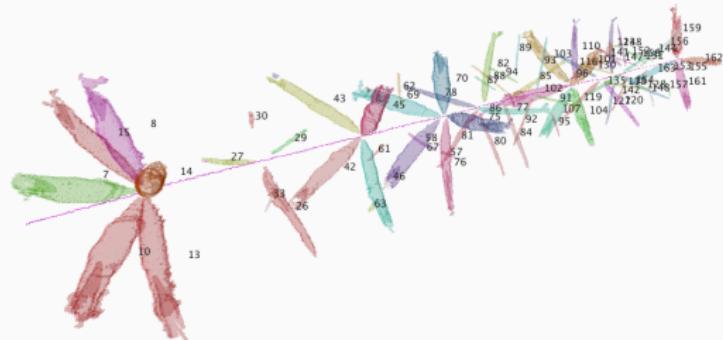
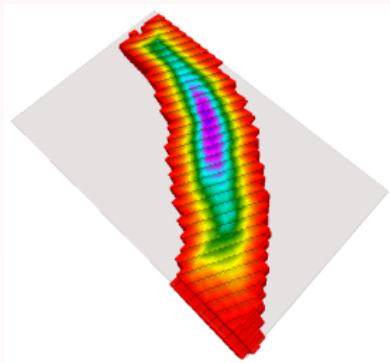


Autres fonctionnalités

- Clipping planes :

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

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



Exemple du code de l'outil de visualisation :

```
00079 QApplication application(argc,argv);
00080 Viewer3D viewer;
00081 viewer.setWindowTitle("simple Volume Viewer");
00082 viewer.show();
00083
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();
```

4. Travail restant

- Modifier le principe “SelfDraw()”.
- Terminer l’harmonisation entre `Board2DTo3D` et `Viewer3D`.
- Rajouter des modèles de réflectance dans `Board2DTo3D`.
- Corriger partiellement l’export vectoriel basé `Viewer3D` (de LibQGLViewer : bibliothèque VRRender de Cyril Sole).

4. Travail restant : visu en PDF 3D ? :(issue #35)

Export en format U3D :

(example.u3d : click to display)

4. Travail restant : visu en PDF 3D ? :(issue #35)

Export en format U3D :

```
poster,label=example.u3d,  
text=(example.u3d : click to display),  
3Daac=60.000000, 3Droll=0.000000, 3Dc2c=-3295.047852  
248.228210 -1028.702759, 3Droo=3460.807129,  
3Dcoo=-3.445007 -0.000031 1.349976,  
3Dlights=CAD,  
]3cm3cmexample.u3d
```