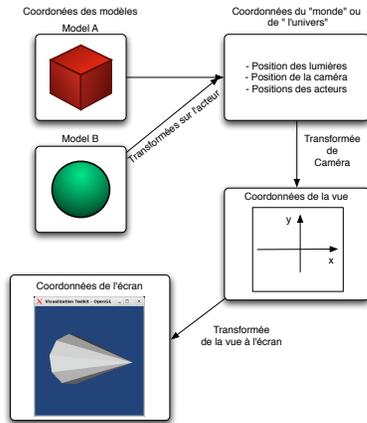
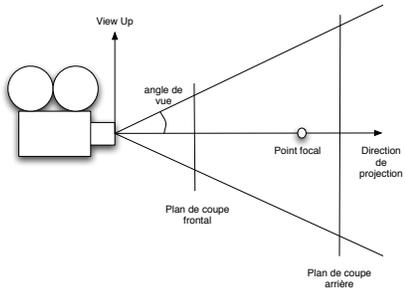


Caméra et systèmes de coordonnées



Modèle graphique

7

Spécificités de la visualisation scientifique

- ▶ Spécificités fonctionnelles
- ▶ Contraintes techniques

8

Bibliothèques 3D de bas niveau

- ▶ OpenGL / MesaGL : Offre l'ensemble des primitives 3D de base:
 - ▶ Polygones, surfaces, primitives géométriques...
 - ▶ Placage de couleurs et de matières
 - ▶ Gestion de la scène 3D (position des objet, gestion de la camera...)
 - ▶ Interaction entre objets
 - ▶ Calcul du rendu (gestion bas niveau)
 - ▶ Moteur de rendu 3D de la majorité des logiciels 3D
 - ▶ Initialement développé par et pour SGI, implémentation libre MesaGL
- ▶ DirectX : équivalent MS Windows d'OpenGL
- ▶ Java 3D : Wrapping (sur-couche) Java à OpenGL et DirextX
 - ▶ Simplification du déploiement
 - ▶ API plus haut niveau (scénographe)
 - ▶ Amélioration des capacités d'interactivité et d'import/export vers les formats de réalité virtuelle (VRML, X3D...)

13

Bibliothèque de traitements libre

Root : Bibliothèque de traitement de données, graphe 2D/3D

- ▶ Initialement développée par le CERN
- ▶ C/C++, X11
- ▶ Intègre des modules de CAD simplifiés
- ▶ Encore utilisée par de nombreux projets, en particulier dans le domaines des radiation (projet GEANT-40)
- ▶ Accessible à <http://root.cern.ch>

14

Outils de traitements propriétaires

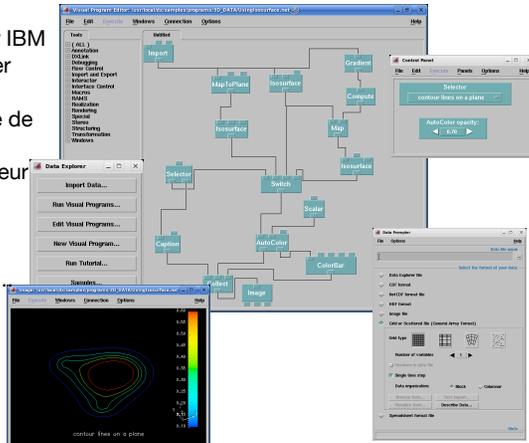
- ▶ AVS
- ▶ Explorer
- ▶ MatLab
- ▶ Enight
- ▶ IDL, PV-WAVES
- ▶ ...

15

Outil de traitement libre

OpenDX

- ▶ Initialement développé par IBM
- ▶ Version libre d'IBM explorer
- ▶ Intègre un outil graphique avancé d'édition de pipeline de traitement
- ▶ Intègre un puissant importeur de formats de données
- ▶ C/C++
- ▶ Forte vie communautaire
- ▶ Mise en oeuvre à la NASA...
- ▶ Accessible à:
<http://www.opendx.org>

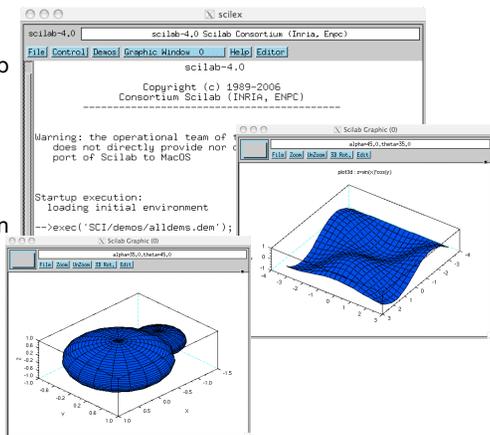


16

Outil de traitement libre

SciLab

- ▶ Environnement de modélisation, "clone" de MatLab
- ▶ Puissant langage de script
- ▶ Nombreuses boîtes à outils scientifiques
- ▶ C/Fortran, X11
- ▶ Forte dynamique communautaire, supporté par un consortium central
- ▶ Licence Scilab
- ▶ Accessible à : <http://www.inria.fr>

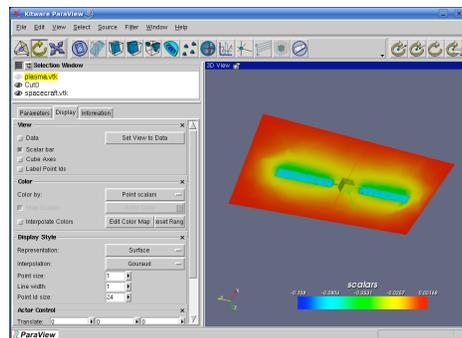


17

Outil de traitement libre

Paraview

- ▶ Visualiseur développé en surcouche à VTK
- ▶ Initialement développé par Kitware
- ▶ Déployable en mode client-serveur
- ▶ Utilise les fonctionnalités de parallélisation de VTK pour le traitement des gros volumes de données
- ▶ En test dans plusieurs grands comptes
- ▶ Paraview 2.x : Tcl/Tk, C++ (VTK)
- ▶ Paraview 3.x : Qt 4, C++
- ▶ Accessible à : <http://www.paraview.org/>

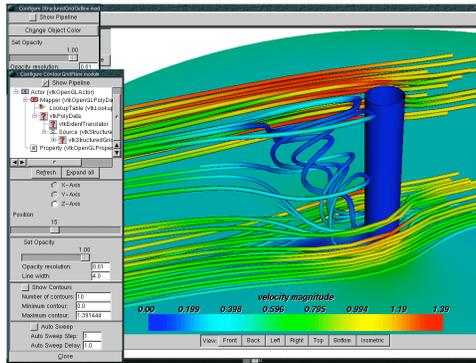


18

Outil de traitement libre

Mayavi

- ▶ initialement développé par P. Ramachandra
- ▶ Python et Tcl/Tk
- ▶ Intègre un visualiseur de pipeline
- ▶ Accessible à <http://mayavi.sourceforge.net>

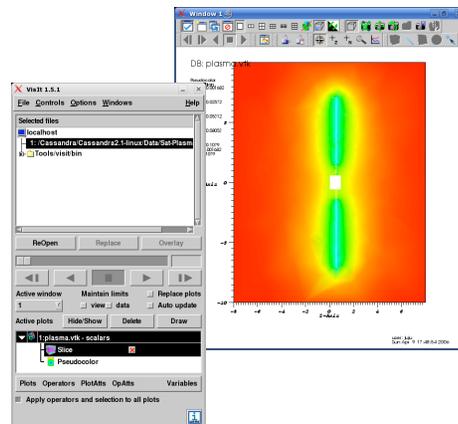


19

Outil de traitement libre

Visit

- ▶ Initialement développé par le Lawrence Livermore Laboratory
- ▶ C/C++, avec interfaces ouvertes vers Python et Java
- ▶ Orienté "plots pré-définis"
- ▶ Repose sur une architecture client-serveur
- ▶ Utilise les capacités parallèles de VTK pour le traitement des gros volumes de données
- ▶ Accessible à <http://www.llnl.gov/Visit/home.html>

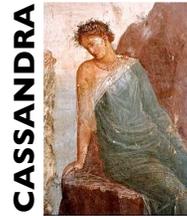
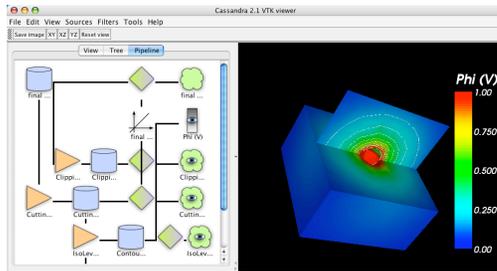


20

Outil de traitement libre

Cassandra

- ▶ Fusion des approches VTK et OpenDX
- ▶ Java/Jython, VTK
- ▶ Utilisé à l'ESA, CNES, ONERA, EDF...
- ▶ Framework d'apprentissage et de développement VTK
- ▶ Licence QPL
- ▶ Accessible à : <http://dev.artenum.com/projects/cassandra>



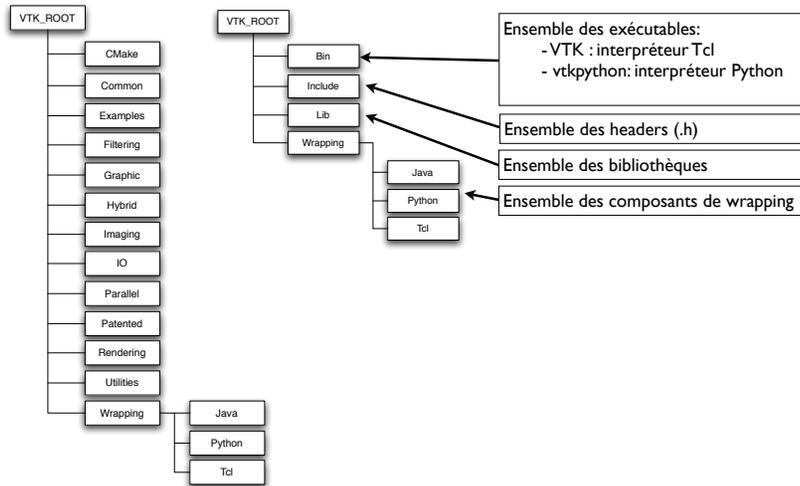
21

Outils de traitement libres

- ▶ Gnuplot : bien connu...
- ▶ Xmgrace (2D) : outils de graphes 2D et d'analyses de données
- ▶ ImageJ : Outil Java d'analyse de données et d'images.
 - ▶ Forte dynamique communautaire
 - ▶ Nombreux plug-in
 - ▶ Un plugin avec VTK (analyse d'image)

22

Structure du package VTK



27

Où trouver l'information

► Site Web Kitware / VTK: <http://public.kitware.com/VTK>

► Documentation d'API: <http://www.vtk.org/doc/release/5.0/html/>

► Documentation technique (formats de fichier, articles théoriques):

<http://public.kitware.com/VTK/documents.php>

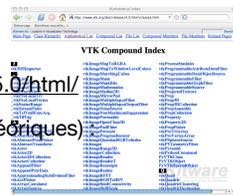
► Mailing listes: <http://public.kitware.com/mailman/listinfo/vtkusers>

► Livres et documents:

► W. Schroeder, K. Martin, B. Lorensen, The Visualization

► ToolKit, 3rd Edition, Kitware (commande en ligne, Amazone...)

► The VTK User's Guide, Kitware (commande en ligne)



28

Où trouver l'information

- ▶ Paraview: <http://www.paraview.org>
- ▶ Mayavi: <http://mayavi.sourceforge.net>
- ▶ Cassandra: <http://dev.artenum.com/projects/cassandra>
 - ▶ Tutorial VTK-JAVA-Eclipse
 - ▶ Exemples de scripts python pour Cassandra
 - ▶ Forum de discussions
 - ▶ VTK compilé avec Wrapping Java

29

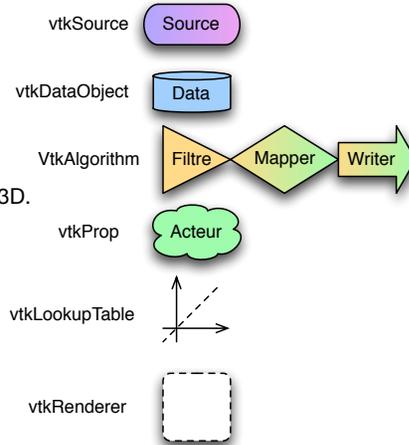
Le passé, le présent et le futur de VTK

- ▶ **Le passé**
 - ▶ VTK 4.2
 - ▶ Format de fichier Legacy
- ▶ **Le présent**
 - ▶ VTK 5.x
 - ▶ Un pipeline extensible et plus souple structurant ses données à la demande
 - ▶ Format de fichiers multi-block, parallèle
 - ▶ Méta-données XML + contenu binaire
- ▶ **Le futur**
 - ▶ Optimisation des traitements avec utilisation poussée du streaming
 - ▶ Meilleures performances
 - ▶ Gestion du temps dans les informations du pipeline
 - ▶ Possible d'avoir des algorithmes effectuant des opérations temporelles (timeshift, delta de scalaires entre deux pas de temps...)
 - ▶ **ATTENTION : Nécessité d'avoir utilisé le nouveau pipeline**

30

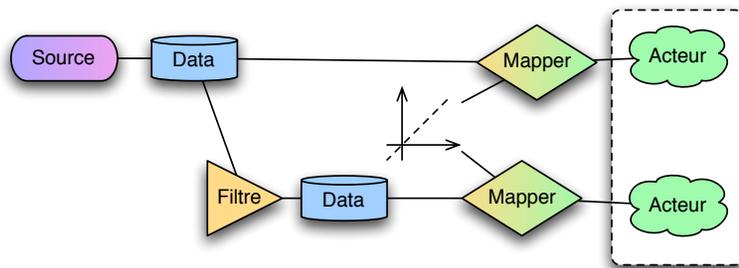
Eléments composant un pipeline

- ▶ **vtkSource**
 - ▶ Lecteur de fichiers, géométrie de base (cône, sphère, box...)
- ▶ **vtkDataObject**
 - ▶ grille, nuage de points...
- ▶ **vtkAlgorithm**
 - ▶ Filtres : dataset => dataset
 - ▶ Mapper : dataset => représentation 3D.
 - ▶ Writer : dataset => autre chose
- ▶ **vtkActor**
 - ▶ Propriété 3D
 - ▶ type de représentation
 - ▶ algorithme de rendu...
- ▶ **vtkLookupTable**
 - ▶ Conversion scalaire/couleur
- ▶ **vtkRenderer** (Scène 3D)



33

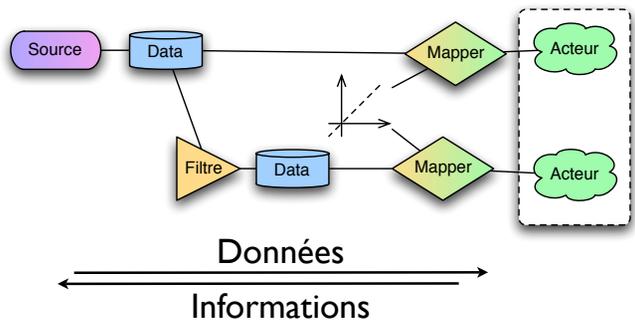
Exemple de pipeline de traitement



34

Rôle et fonction du pipeline

- ▶ Propagation de l'information et des données (maintient de cohérence)
 - ▶ Informations : requête de visualisation (Render, Update, UpdateData)
 - ▶ Données : Modification d'un filtre, génération d'une nouvelle sortie



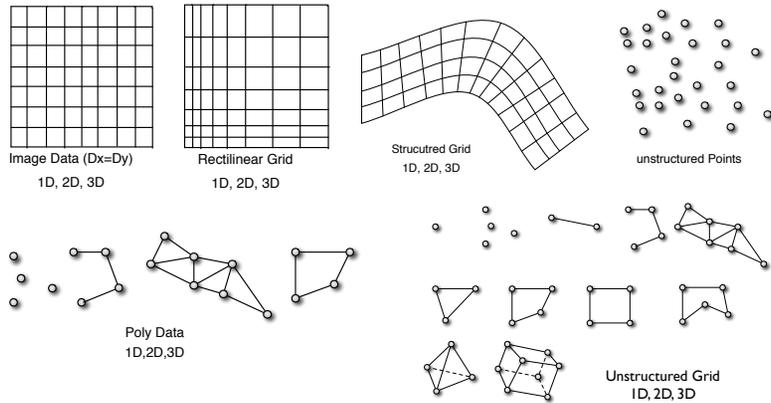
35

Structures de données

- ▶ Structures de données et type de grilles
- ▶ Type de cellules
- ▶ Hiérarchie de classes des différents VTK datasets
- ▶ Composition d'un dataset

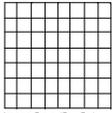
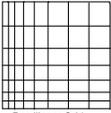
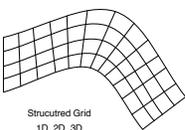
36

Structure de données et type de grilles



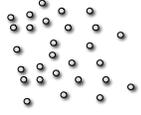
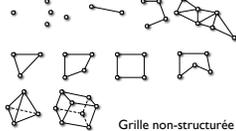
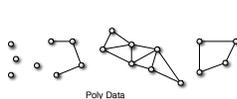
37

Caractéristiques des grilles structurées

 Image Data (Dx=Dy) 1D, 2D, 3D	 Rectilinear Grid 1D, 2D, 3D	 Structured Grid 1D, 2D, 3D
<ul style="list-style-type: none"> - Noeuds répartis régulièrement et parallèlement au système d'axes (repérage par indices) - Pas constant, cellules régulières - Représentation implicite de la topologie de cellule & connectivité - $n_x \times n_y \times n_z$ noeuds; - $(n_x - 1) \times (n_y - 1) \times (n_z - 1)$ cellules - Données localisées sur les noeuds ou sur les cellules - Coût mémoire optimum - Coût CPU des traitements optimum 	<ul style="list-style-type: none"> - Topologie de cellules régulière (i.e Pixel en 2D, Voxel en 3D) - Noeuds répartis régulièrement et parallèlement au système d'axes - Coordonnées des noeuds (x, y, z) doivent être explicitées dans trois listes séparées (repérage par indices dans ces trois listes). - Données localisées sur les noeuds ou sur les cellules 	<ul style="list-style-type: none"> - Topologie de cellules régulière, géométrie irrégulière (i.e quadrilatère en 2D, hexaèdre en 3D) - La grille peut suivre tout type de configuration (forme) sous réserve que les cellules ne se superposent pas ou ne se coupent pas elles mêmes. - Données localisées sur les noeuds ou sur les cellules - Nécessite un tableau de coordonnées des noeuds - Repérage implicite par indice
<ul style="list-style-type: none"> - Différences finies sur maillage Cartésien régulier 	<ul style="list-style-type: none"> - Différences finies sur maillage Cartésien adaptatif 	<ul style="list-style-type: none"> - Différences finies sur maillage irrégulier

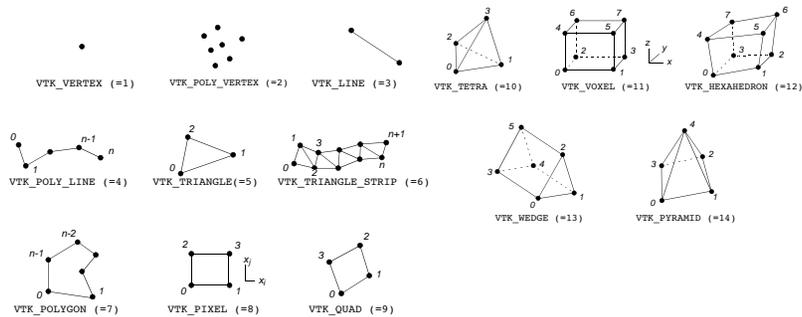
38

Caractéristiques des grilles non structurées

 <p>unstructured Points</p>	 <p>Grille non-structurée</p>	 <p>Poly Data 1D,2D,3D</p>
<ul style="list-style-type: none"> - Noeuds répartis irrégulièrement - Pas de cellules - Localisation des données sur les noeuds 	<ul style="list-style-type: none"> - Topologie de cellules irrégulières - Coordonnées des noeuds (x, y, z) - Liste de types de cellules - Liste des noeuds pour chaque cellule - Repérage par indices dans les trois listes. - Données pouvant être localisées sur les noeuds et/ou les cellules - Différents types de cellules peuvent être présentes dans une même grille 	<ul style="list-style-type: none"> - Topologie de cellule régulière, géométrie irrégulière (i.e quadrilatère en 2D, hexaèdre en 3D) - La grille peut suivre tout type de configuration (forme) sous réserve que les cellules ne se superposent pas ou ne se coupent pas elles mêmes. - Nécessite un tableau de coordonnées des noeuds - Repérage implicite par indice
<ul style="list-style-type: none"> - Nuage de valeurs discrètes 	<ul style="list-style-type: none"> - Éléments finis 	<ul style="list-style-type: none"> - Éléments finis - Rendu optimisé - Contraintes de calcul

39

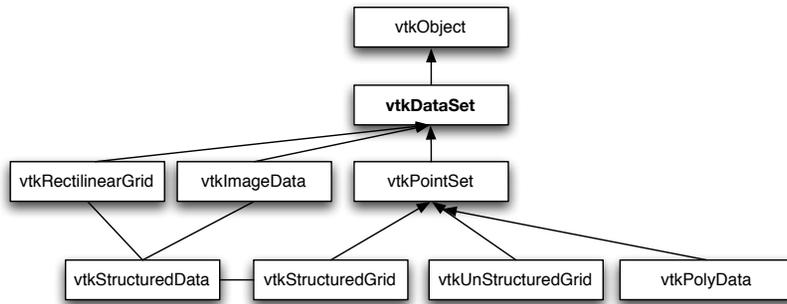
Types de cellules (grilles non structurées)



► **Remarque** : VTK supporte 19 types différents de cellules.

40

Hiérarchie de classes des ≠ datasets



41

Composition d'un dataset

- ▶ Généralement, un tableau de points (ou noeuds), stocké dans une liste ordonnée `vtkPoints`
- ▶ Généralement, une liste indexée de cellules
- ▶ Un tableau de valeurs
- ▶ Plusieurs données peuvent être attribuées à un data set et un nom peut être attribué à chacune de ces données
- ▶ Les données peuvent être scalaires, vectorielles, matricielles, selon le type de tuple de valeurs défini
- ▶ Selon le type de grille, les données peuvent être attribuées aux noeuds ou aux cellules
- ▶ Des données pouvant être stockées dans un `vtkFloatArray`

42

vtkCutter (Après)

```

# -----
# Creation du filtre de traitement
# -----
# definition de la fonction de coupe
plane = vtkPlane()
plane.SetOrigin(0.0, 0.0, 0.0)
plane.SetNormal(0, 0, 1.0)

#creation du filtre
planeCut = vtkCutter()
planeCut.SetCutFunction(plane)
planeCut.SetInputConnection(outputPort)

# -----
# fin de la creation du filtre
# -----

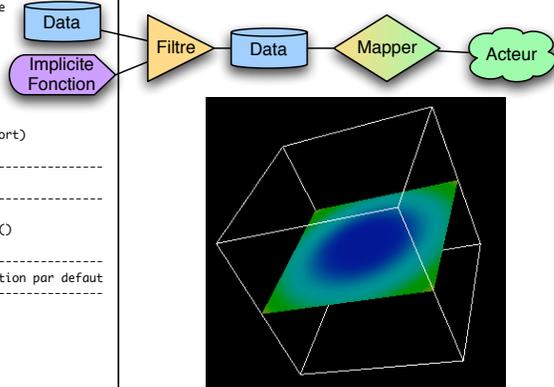
outputPort = planeCut.GetOutputPort()

# -----
# creation du pipeline de visualisation par default
# -----

mapper = vtkDataSetMapper()
mapper.SetInputPort(outputPort)

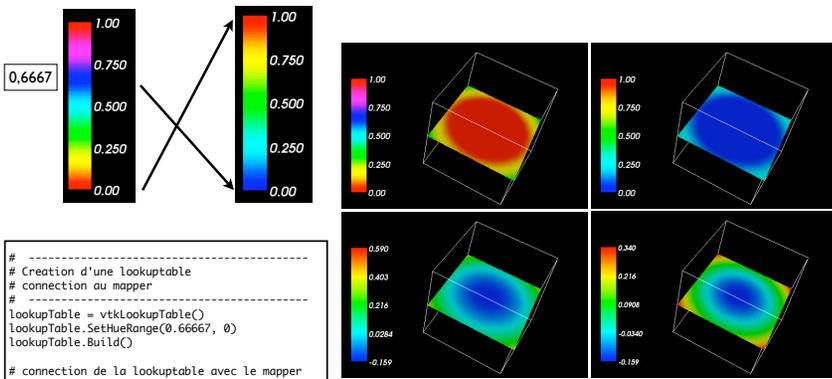
actor = vtkActor()
actor.SetMapper(mapper)

```



Gestion du placage de couleurs

► Conversion donnée scalaire => couleur



```

# -----
# Creation d'une lookuptable
# connection au mapper
# -----
LookupTable = vtkLookupTable()
LookupTable.SetHueRange(0.66667, 0)
LookupTable.Build()

# connection de la lookuptable avec le mapper
mapper.SetLookupTable(LookupTable)
mapper.SetScalarRange(dataset.GetScalarRange())

```

Gestion de la scène 3D

- ▶ Objets de la scène 3D
- ▶ Intégration de scènes 3D dans une fenêtre
- ▶ Manipulation des acteurs 3D
- ▶ Gestion des événements de la scène 3D

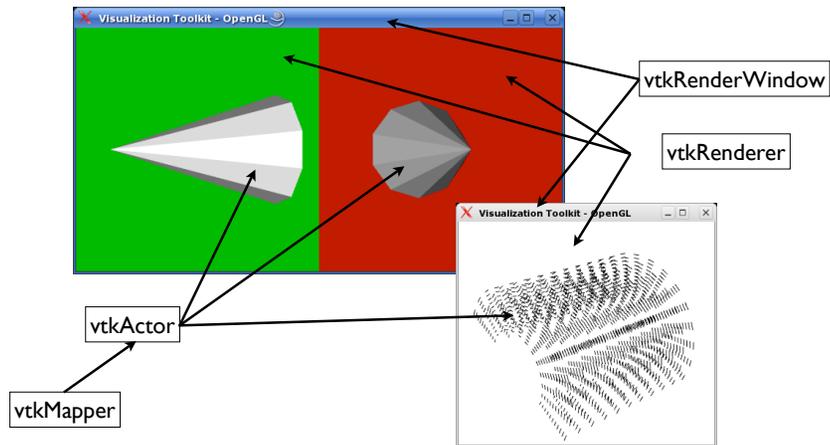
49

Objets de la scène 3D

- ▶ **vtkRenderWindow** : gère la fenêtre de rendu dans le contexte graphique (X11, OpenGL, Windows...); peut intégrer un ou plusieurs vtkRenderers.
- ▶ **vtkRenderer** : Gère les caractéristiques de la scène 3D (caméra, lumière, orientation, ...); gère le type de rendu (perspective, stéréo, anaglyph...)
- ▶ **vtkLight** : gère la/les lumières de la scène
- ▶ **vtkCamera** : définit la position de la vue, du point focal...
- ▶ **vtkActor** : acteur de représentation 3D des données affichées
- ▶ **vtkProperty** : définit les propriétés de surface des acteurs 3D
- ▶ **vtkMapper** : définit la géométrie de l'objet 3D

50

Intégration de scènes 3D



51

Manipulation des acteurs 3D

► **vtkActor** : représentation des objets dans la scène 3D

► Propriété de l'acteur, **vtkProperty** : définir les propriétés de rendu des acteurs comme:

- Couleur
- Transparence
- Type de rendu / interpolation
- Rendu
- ...

```
actor.GetProperty().SetColor(1.0, 0.0, 0.3)

actor.GetProperty().SetRepresentationToWireframe()
actor.GetProperty().SetRepresentationToPoints()
actor.GetProperty().SetPointSize(10.0)

# gestion des interpolations (si donnees mappees)
actor.GetProperty().SetInterpolationToFlat()
actor.GetProperty().SetInterpolationToGouraud()
actor.GetProperty().SetRepresentationToWireframe()
```

52

Gestion des événements de la scène

- ▶ Gère les événements de la scène 3D
 - ▶ Rotation, translation, zoom, picking...
 - ▶ Attribution possible d'un style
 - ▶ Attribution possible d'un observateur

Instanciation de l'interacteur d'événements

Attribution de l'interacteur à la fenêtre

Définition du style d'interaction

Attribution du style

Démarrage de la boucle d'événements et du rendu

Test : Commenter les lignes impliquant le `vtkRenderWindowInteractor` et activer le rendu par :

```
renWin.Render()  
while 1:  
    pass
```

```
import vtk  
cone = vtk.vtkConeSource()  
cone.SetHeight( 3.0 )  
cone.SetRadius( 1.0 )  
cone.SetResolution( 10 )  
  
coneMapper = vtk.vtkPolyDataMapper()  
coneMapper.SetInput(cone.GetOutput())  
  
coneActor = vtk.vtkActor()  
coneActor.SetMapper(coneMapper)  
  
ren1 = vtk.vtkRenderer()  
ren1.AddActor(coneActor)  
ren1.SetBackground(0.1, 0.2, 0.4)  
  
renWin = vtk.vtkRenderWindow()  
renWin.AddRenderer(ren1)  
renWin.SetSize(300, 300)  
  
iren = vtk.vtkRenderWindowInteractor()  
iren.SetRenderWindow(renWin)  
  
style = vtk.vtkInteractorStyleTrackballCamera()  
#style = vtk.vtkInteractorStyleFlight()  
#style = vtk.vtkInteractorStyleImage()  
iren.SetInteractorStyle(style)  
  
iren.Initialize()  
iren.Start()
```

53

Lecture et écriture

- ▶ Formats de fichiers VTK
- ▶ Différence entre Reader-Importer et Writer-Exporter
- ▶ Lecture et écriture de fichiers VTK
- ▶ Import / export

54

Formats de fichiers VTK

► **Remarque** : Pour plus d'informations, voir File Formats for VTK (extrait de The VTK User's Guide), <http://public.kitware.com/VTK/documents.php>

Simple Legacy Format		XML Format	
ASCII	BINAIRE	Serial	Parallèle
- Identifiant de version - Entête - Format de fichier - Structure du dataSet - Attributs du dataSet		• ImageData (.vti) • PolyData (.vtp) • RectilinearGrid (.vtr) • StructuredGrid (.vts) • UnstructuredGrid (.vtu)	p + XXX
- Facile à écrire (à la main ou avec des moulinettes) - Portable (ASCII) - Compacte (BINAIRE)		- Plus de capacités que le format classique - Facilite le streaming et les E/S parallèles - Plus complexe - Compacte (BINAIRE)	

55

Import/Export ≠ Reader/Writer

► **Les Imports / Exports agissent sur la scène 3D dans sa globalité**

- Export 2D : Image 2D de la scène
- Export 3D : Exporte la géométrie, les lumières, la caméra...
- Import 3D : Charge une scène complète

► **La lecture / Ecriture s'effectue sur des jeux de données**

- Writer : vtkDataset, autre structure de données
- Reader : Lit des données et construit un vtkDataset

► **Remarque**

- Un mapper est un writer à sa manière. Un mapper convertit un dataset en une structure 3D OpenGL auquel il externalise des propriétés 3D (vtkActor).

56

Lecture et écriture de fichiers VTK

- ▶ VTK offre une série de classes de lecture et d'écriture fichiers
 - ▶ vtkReader, module générique de lecture du format VTK Legacy
 - ▶ vtkWriter, module générique d'écriture au format VTK Legacy
 - ▶ Plusieurs modules de lecture/écriture des format VTK XML

```
# -----  
# Lecture du dataset  
# -----  
reader = vtkDataSetReader()  
reader.SetFileName(nom_du_fichier)  
reader.Update()  
reader.CloseVTKFile()  
  
# -----  
# fin de la lecture  
# -----  
dataset = reader.GetOutput()  
outputPort = reader.GetOutputPort()
```

```
# -----  
# Ecriture du dataset dans un fichier  
# -----  
nom_du_fichier = "test.vtk"  
  
writer = vtkDataSetWriter()  
writer.SetFileName(nom_du_fichier)  
writer.SetInput(dataset)  
writer.Update()
```

Remarque: Pour plus d'informations, voir la documentation d'API et les mailing listes.

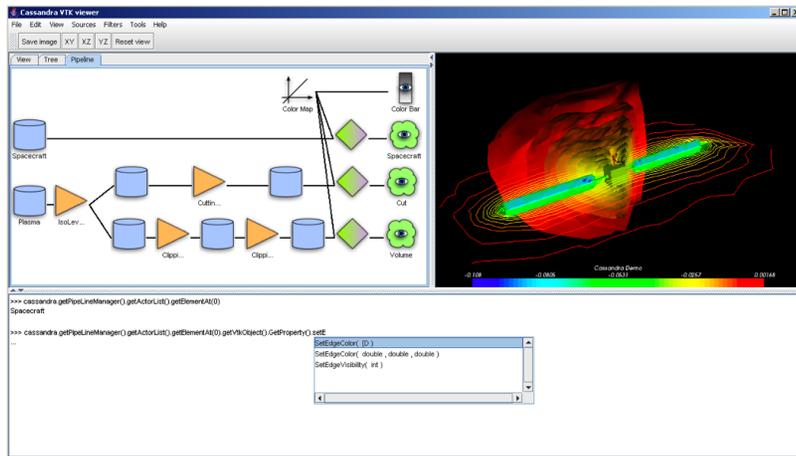
57

Import / Export de la scène

- ▶ VTK offre un large ensemble de modules d'import
 - ▶ Données scientifiques : Gambit, Enight, AVS (UCD), Exodus, Plot3D, OBJ, SLC, STL, DEM, BYU...
 - ▶ De données 3D : RIB, 3DS, VRML...
 - ▶ De formats d'image : JPEG...
- ▶ Et de modules d'export....

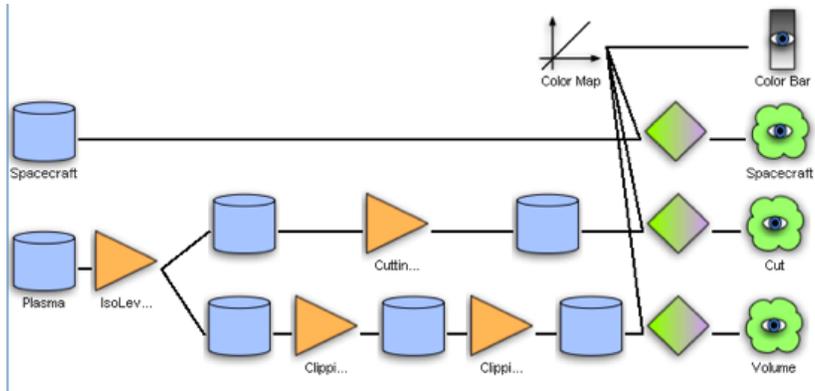
58

Présentation de Cassandra



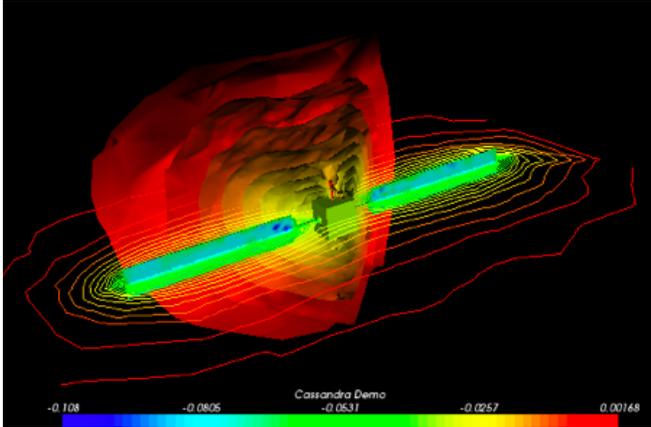
61

Editeur graphique de pipeline VTK



62

Vue 3D



63

Console Python avec completion objet

► La console intégrée embarque une référence vers Cassandra. Cette référence donne accès à l'ensemble des méthodes accessibles du framework.

```
>>> cassandra.getPipelineManager().getActorList().getElementAt(0)  
Spacecraft  
  
>>> cassandra.getPipelineManager().getActorList().getElementAt(0).getVtkObject().GetProperty().setE  
...  
SetEdgeColor( [D ] )  
SetEdgeColor( double , double , dou  
SetEdgeVisibility( int )
```

64

Accès aux objets de Cassandra

- ▶cassandra
 - ▶getPipelineManager()
 - ▶getDataSetList() / getMapperList() / getActorList() / ...
 - ▶getElementAt(int index) => VtkObject
 - ▶getVtkObject() => vtkDataset/vtkMapper/vtkActor/
 - ▶getSize()
 - ▶getPluginManager()
 - ▶getElementAt(int index) => CassandraPlugin
 - ▶getId(), getName()...
 - ▶updateXXX(...)
 - ▶getSize()
 - ▶getCassandraView()
 - ▶validateAndWait()
 - ▶validateAndGo()
 - ▶rotate(double X, double Y)
 - ▶zoom(double zoomFactor)

65

Ajout d'objets dans Cassandra

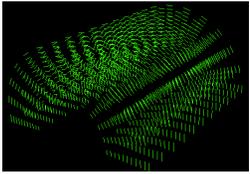
- ▶cassandra
 - ▶getPipelineManager()
 - ▶addVtkFile(File vtkFile)
 - ▶setActorVisible(VtkObject actor, boolean viewActor)
 - ▶addActor(vtkActor actorVtk, String actorName)
 - ▶addMapper(vtkMapper mapperVtk, String mapperName)
 - ▶addDataSet(vtkDataset datasetVtk, String datasetName)
 - ▶addScalarBar(vtkScalarBar scalarBarVtk, String name)
 - ▶addLookupTable(vtkLookupTable lTable, String name)
 - ▶addTxtActor(vtkTextActor txtActor, String name)
- ▶VtkObject
 - ▶getId(), getName(), getType()
 - ▶getVtkObject() => get real VTK object
 - ▶getMetaData() => HashTable

66

Construire une grille structurée

```
x = [0,0,0]
v = [0, 0,0]
rMin = 0.5
rMax = 1.0
dims = [13, 11, 11]
nbTuples = dims[0]*dims[1]*dims[2]
# Create the structured grid.
sgrid = vtkStructuredGrid()
sgrid.SetDimensions(dims);

# We also create the points and
# vectors. The points form a
# hemi-cylinder of data.
vectors = vtkFloatArray()
vectors.SetNumberOfComponents(3)
vectors.SetNumberOfTuples(nbTuples)
points = vtkPoints()
```



```
deltaZ = 2.0 / (dims[2]-1)
deltaRad = (rMax-rMin) / (dims[1]-1)
v[2]=0.0
for k in range(dims[2]):
    x[2] = -1.0 + k*deltaZ
    kOffset = k * dims[0] * dims[1]
    for j in range(dims[1]):
        radius = rMin + j*deltaRad
        jOffset = j * dims[0]
        for i in range(dims[0]):
            theta = i * 15.0 * math.pi/180.0
            #vtkMath.DegreesToRadians()
            x[0] = radius * cos(theta)
            x[1] = radius * sin(theta)
            v[0] = -x[1]
            v[1] = x[0]
            offset = i + jOffset + kOffset
            points.InsertPoint(offset,x)
            vectors.InsertTuple3(offset,v[0], v[1], v[2])

sgrid.SetPoints(points)
sgrid.GetPointData().SetVectors(vectors)
```

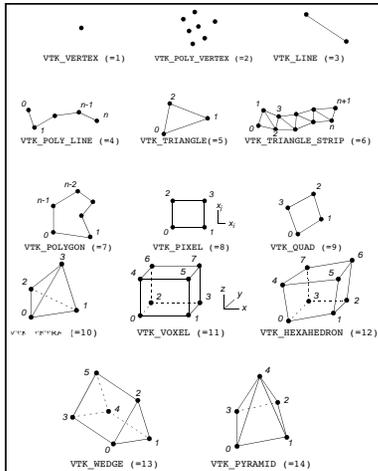
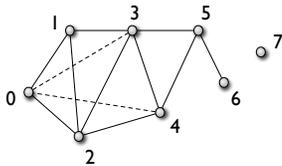
69

Construire une grille non-structurée

```
nbpoints = 8
# Definition des noeuds
pointCoord = range(nbpoints)
pointCoord[0]=(0.0, 0.0, 0.0)
pointCoord[1]=(0.3, 0.1, 1.0)
pointCoord[2]=(0.7, 0.1, -0.2)
pointCoord[3]=(0.5, 1.2, 1.1)
pointCoord[4]=(0.9, 2.0, -0.3)
pointCoord[5]=(-1.5, 2.5, 1.4)
pointCoord[6]=(0.2, 3.4, -1.2)
pointCoord[7]=(2, 4, -1.2)

# Definition des valeurs correspondantes
ptData = range(nbpoints)
ptData[0] = 1.0
ptData[1] = 1.2
ptData[2] = 1.6
ptData[3] = 2.0
ptData[4] = 2.5
ptData[5] = 3.4
ptData[6] = 5.7
ptData[7] = 7.0

VTK_TETRA = 10
VTK_TRIANGLE = 5
VTK_LINE = 3
VTK_VERTEX = 1
```



70

Construire une grille non-structurée

```

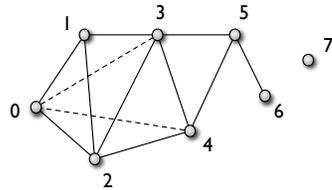
unstructuredGrid = vtkUnstructuredGrid()
points = vtkPoints()

for i in range(nbpoints):
    points.InsertNextPoint(pointCoord[i])

data = vtkFloatArray()
data.SetName("My data name")
data.SetNumberOfTuples(1)
data.SetNumberOfValues(nbpoints)

# 1er tetraedre
idList = vtkIdList()
idList.InsertNextId(0)
idList.InsertNextId(1)
idList.InsertNextId(2)
idList.InsertNextId(3)
unstructuredGrid.InsertNextCell( VTK_TETRA, idList)

# 2eme tetraedre
idList = vtkIdList()
idList.InsertNextId(0)
idList.InsertNextId(2)
idList.InsertNextId(3)
idList.InsertNextId(4)
unstructuredGrid.InsertNextCell( VTK_TETRA, idList)
    
```



71

Construire une grille non-structurée

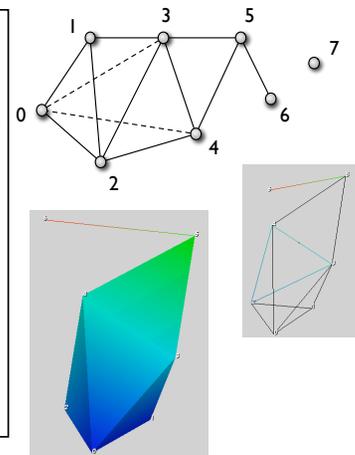
```

# le triangle
idList = vtkIdList()
idList.InsertNextId(3)
idList.InsertNextId(4)
idList.InsertNextId(5)
unstructuredGrid.InsertNextCell( VTK_TRIANGLE, idList)

# l'arrete
idList = vtkIdList()
idList.InsertNextId(5)
idList.InsertNextId(6)
unstructuredGrid.InsertNextCell( VTK_LINE, idList)

# le dernier noeud en tant que simple vertex
idList = vtkIdList()
idList.InsertNextId(7)
unstructuredGrid.InsertNextCell( VTK_VERTEX, idList)

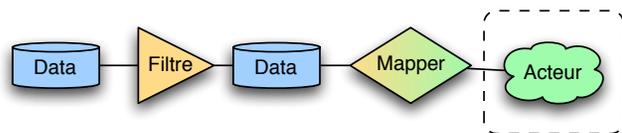
# attribution des valeurs aux noeuds
for i in range(nbpoints):
    data.InsertTuple1(i, ptData[i])
    
```



72

Appliquer un Iso-Level sur l'image data

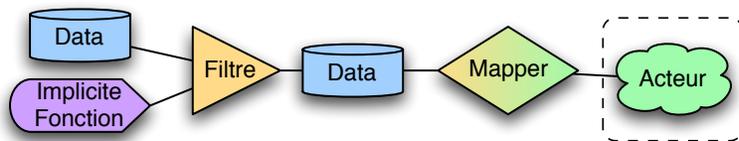
- ▶ Filtre VTK à utiliser :
 - ▶ vtkContourFilter
- ▶ Un simple IsoLevel
 - ▶ SetNumberOfContour(1)
 - ▶ SetValue(xxx)
- ▶ Un multi-iso-level
 - ▶ GenerateValues(nbLevel, min, max)



75

Appliquer un cutter sur l'image data

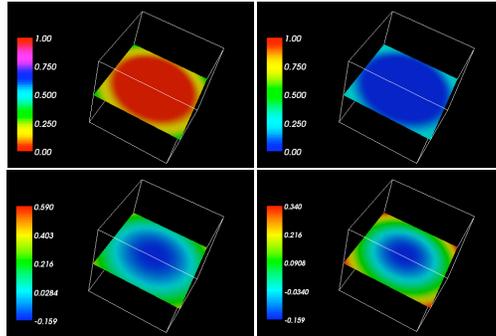
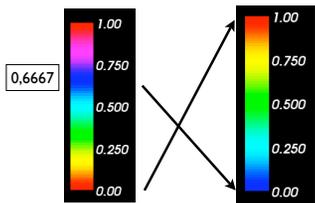
- ▶ Objets VTK à utiliser :
 - ▶ vtkCutter
 - ▶ vtkImplicitFunction : vtkPlane, vtkCone, vtkSphere...
- ▶ Définition de la fonction de coupe
 - ▶ SetCutFunction(plane)
- ▶ Essayer différentes fonctions



76

Utiliser un placage de couleurs correct

- ▶ Objets VTK à utiliser :
 - ▶ vtkLookupTable
- ▶ Opérations à effectuer :
 - ▶ Sur la lookupTable
 - ▶ SetHueRange
 - ▶ Build
 - ▶ Sur le mapper
 - ▶ SetLookupTable
 - ▶ SetScalarRange



77

Plan du cours (2/3)

- ▶ Contrôle et manipulation des données
 - ▶ Sélection d'une donnée dans un dataset contenant plusieurs données
 - ▶ Effectuer un traitement sur des données (vtkProgrammableFilter)
 - ▶ Exercices
- ▶ Contrôle et manipulation de la scène
 - ▶ Manipulation, transformation, agrégation et optimisation des acteurs 3D
 - ▶ Ajouter des données 2D à la scène (Texte, ScalarBar)
 - ▶ Gestion de textures
 - ▶ Interaction directe avec les vtkWidgets
 - ▶ Exercices
- ▶ Introduction à la gestion de gros volumes de données et au parallélisme
 - ▶ Streaming de données
 - ▶ Parallélisation des tâches
 - ▶ Parallélisation du pipeline
 - ▶ Parallélisation des données

78

Contrôle et manipulation des données

- ▶ Sélection d'une donnée dans un dataset contenant plusieurs données
- ▶ Effectuer un traitement sur des données (vtkProgrammableFilter)
- ▶ Exercices

79

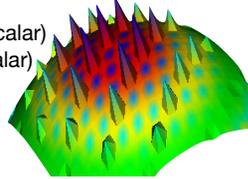
Sélection d'une donnée

- ▶ Un filtre ne doit pas modifier son dataset d'entrée !
 - ▶ Comment effectuer un traitement sur une donnée spécifique ?
 - ▶ Utiliser un filtre qui permet de sélectionner une donnée
 - ▶ vtkAssignAttribute
 - ▶ Assign(ArrayName, ArrayType, DataLocation)
 - ▶ Update()
 - ▶ ArrayType : SCALARS, VECTORS, TENSORS
 - ▶ DataLocation : POINT_DATA, CELL_DATA
- ▶ Comment plaquer des couleurs sur la base de données différentes de celles qui sont actives ?
 - ▶ Utiliser les méthodes du mapper
 - ▶ ColorByArrayComponent
 - ▶ SetScalarModeToUsePointData, SetScalarModeToUseCellData, SetScalarModeToUsePointFieldData, SetScalarModeToUseCellFieldData
 - ▶ ScalarVisibilityOn()
 - ▶ Update()

80

vtkProgrammable Filters

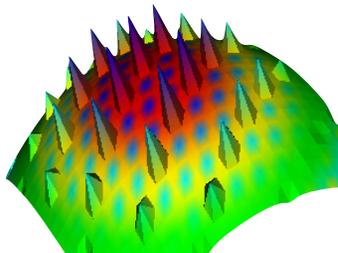
- ▶ Implémentation d'un filtre personnalisé par héritage d'un vtkProgrammableFilter
 - ▶ Définir la méthode à exécuter afin d'effectuer le processing du filtre.
 - ▶ SetExecuteMethod(instance , nomDeLaMethode)
 - ▶ Méthodes nécessaires pour récupérer l'input du filtre
 - ▶ GetXXXInput()
 - ▶ Méthodes nécessaires pour définir la sortie du filtre
 - ▶ GetXXXOutput().CopyStructure(GetXXXInput())
 - ▶ GetXXXOutput().GetPointData().PassData(GetXXXInput().GetPointData())
 - ▶ GetXXXOutput().GetCellData().PassData(GetXXXInput().GetCellData())
 - ▶ GetXXXOutput().SetPoints(newPoints)
 - ▶ GetXXXOutput().GetPointData().SetScalars(newScalar)
 - ▶ GetXXXOutput().GetCellData().SetScalars(newScalar)



81

Exercices

- ▶ Réaliser un filtre qui modifie la géométrie
 - ▶ Implémenter l'équivalent du filtre vtkWarpScalar
- ▶ Réaliser un filtre qui modifie les données
 - ▶ Afin d'obtenir un placage de couleurs en quadrillage



82

Contrôle et manipulation de la scène

- ▶ Manipulation, transformation, agrégation et optimisation des acteurs 3D
- ▶ Ajouter des données 2D à la scène (Texte, ScalarBar)
- ▶ Gestion de textures
- ▶ Interaction directe avec les vtkWidgets
- ▶ Exercices

83

Transformation d'acteur 3D

- ▶ Des transformations peuvent être appliqués sur les acteurs 3D
 - ▶ Translation (SetUserTransform)
 - ▶ Rotation (SetUserTransform)
 - ▶ Echelle (SetScale)
- ▶ L'interactivité de la vue 3D peut être accrue par l'utilisation d'acteurs LOD
 - ▶ vtkLODActor : Level Of Details
- ▶ Agrégation d'acteurs par la construction d'un vtkAssembly
 - ▶ AddPart(vtkActor)
 - ▶ SetOrigine(x,y,z)
 - ▶ ...
- ▶ Propriétés des acteurs
 - ▶ Transparence (Opacity)
 - ▶ Couleur : Désactiver sur le mapper le placage de couleurs sur les scalaires `ScalarVisibilityOff()`
 - ▶ Représentation filaire, point ou surfacique (SetRepresentationToXXX)
 - ▶ Taille des arrêtes, points...

84

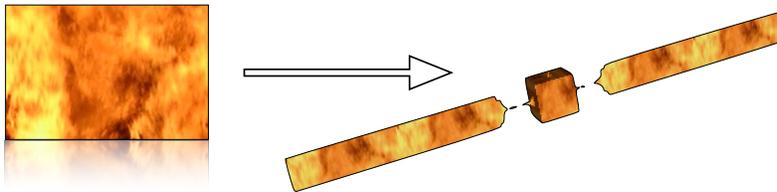
Données 2D dans la scène 3D

- ▶ Acteurs 2D sont des acteurs ajoutés dans une scène 3D mais présentés toujours face à la caméra.
- ▶ Une méthode spécifique est utilisée pour les ajouter au renderer.
 - ▶ AddActor2D
- ▶ Liste d'acteurs 2D
 - ▶ vtkTextActor
 - ▶ vtkScalarBar

85

Gestion des textures

- ▶ Le placage d'une image sur une géométrie nécessite
 - ▶ Un lecteur de fichier image (vtkBMPReader, vtkPNGReader...)
 - ▶ Une texture (vtkTexture)
 - ▶ Application de la texture sur un acteur
- ▶ Utilisation de filtres sur les données pour la gestion du Mapping de texture
 - ▶ vtkTextureMapToCylinder, vtkTextureMapToPlane, vtkTextureMapToXXX
 - ▶ vtkTransformTextureCoords

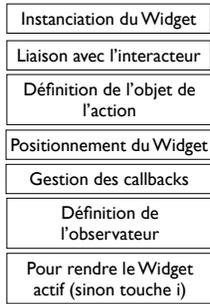
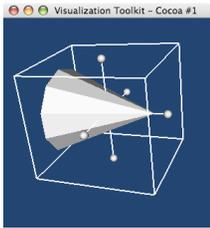


86

Widgets VTK

► Permet la manipulation des objets 3D en utilisant le gestionnaire d'événements

- vtkBoxWidget, boîte de contrôle
- vtkSphereWidget, sphère de contrôle
- vtkPointWidget, point de sonde
- vtkPlaneWidget, vtkImplicitPlaneWidget, scalarBarWidget



```
coneActor = vtk.vtkActor()
coneActor.SetMapper(coneMapper)

ren1 = vtk.vtkRenderer()
ren1.AddActor(coneActor)
ren1.SetBackground(0.1, 0.2, 0.4)

renWin = vtk.vtkRenderWindow()
renWin.AddRenderer(ren1)
renWin.SetSize(300, 300)

iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)

style = vtk.vtkInteractorStyleTrackballCamera()
iren.SetInteractorStyle(style)

boxWidget = vtk.vtkBoxWidget()
boxWidget.SetInteractor(iren)
boxWidget.SetPlaceFactor(1.25)

boxWidget.SetProp3D(coneActor)
boxWidget.PlaceWidget()

def myCallback(widget, event_string):
    t = vtk.vtkTransform()
    boxWidget.GetTransform(t)
    boxWidget.GetProp3D().SetUserTransform(t)

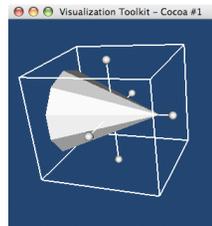
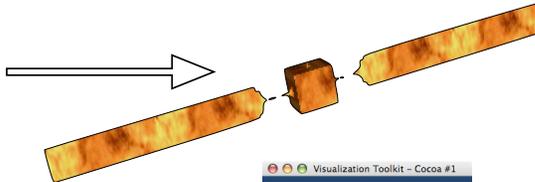
boxWidget.AddObserver("InteractionEvent", myCallback)
boxWidget.On()

iren.Initialize()
iren.Start()
```

87

Exercices

► Mise en pratique des éléments pertinents pour vous !



88

Introduction à la gestion de gros volumes de données et au parallélisme

- ▶ Streaming de données
- ▶ Parallélisation des tâches
- ▶ Parallélisation du pipeline
- ▶ Parallélisation des données

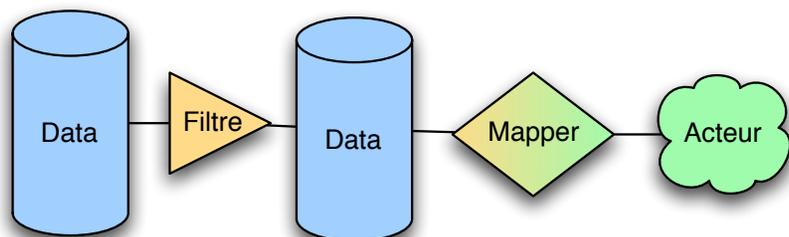
89

Types de parallélisme

- ▶ Data streaming
- ▶ Parallélisation des tâches
- ▶ Parallélisation du pipeline
- ▶ Parallélisation des données

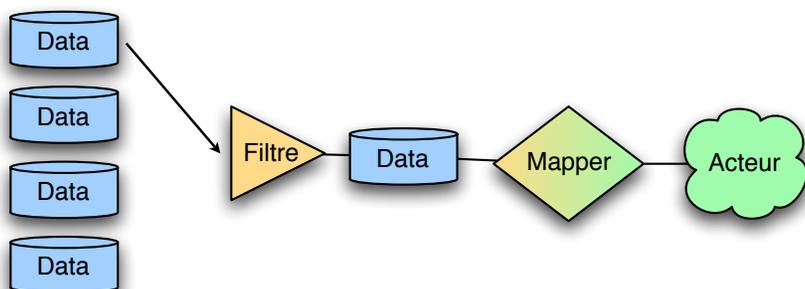
90

Sans streaming



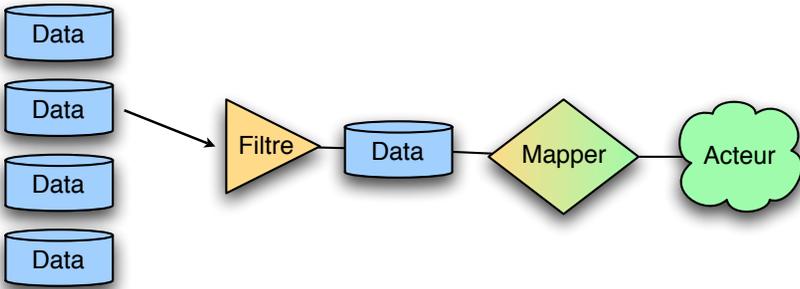
93

Avec streaming



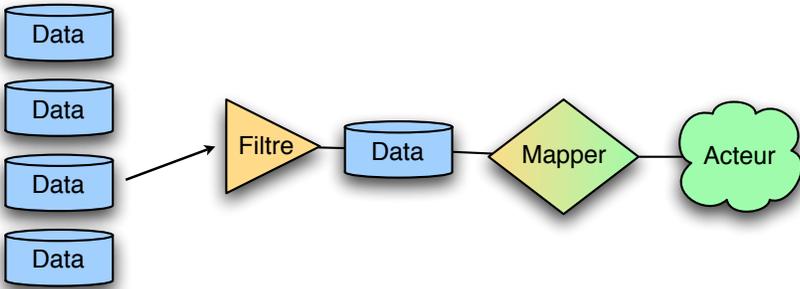
94

Avec streaming



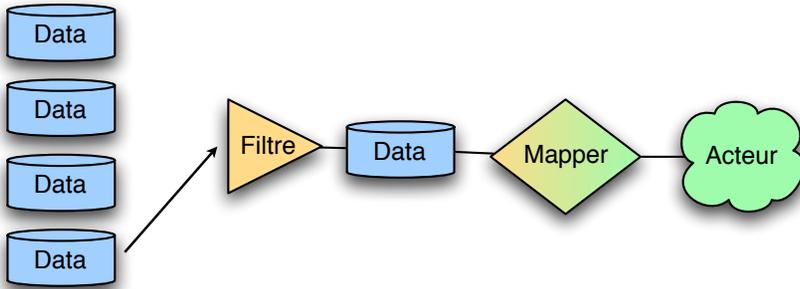
95

Avec streaming



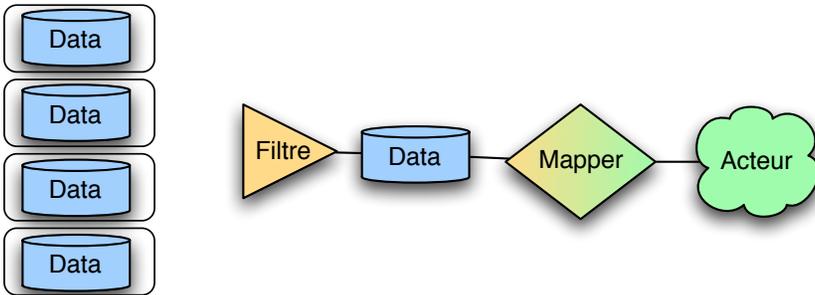
96

Avec streaming

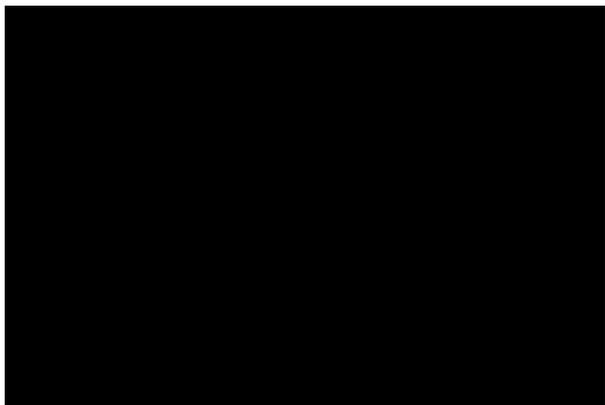


97

Avec streaming



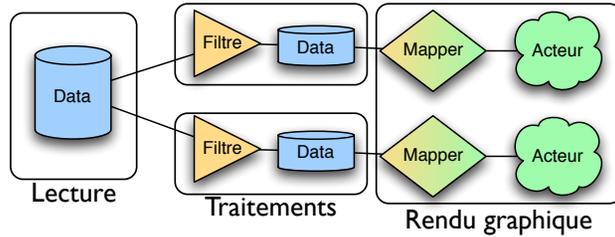
98



Parallélisation des tâches

Parallélisation des tâches

► Les modules indépendants s'exécutent en parallèle (Local = Multi-threading)



► Avantages

► Adapté pour la parallélisation des tâches associées à la visualisation

► Inconvénients

► Le nombre d'éléments parallélisés se limite au nombre de tâches

► Pas de répartition de charge

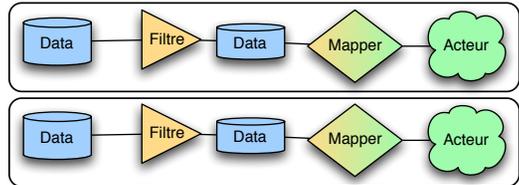
101

Parallélisation du pipeline

102

Parallélisation du pipeline

▶ Les pipelines de traitement indépendants s'exécutent en parallèle



▶ Avantages

- ▶ Adapté lorsqu'une multitude de tâches indépendantes existe
- ▶ Utilisation optimum de toutes les ressources parallèle.

▶ Inconvénients

- ▶ Pas de répartition de charge sur les pipelines
- ▶ Limité par l'élément du pipeline le plus lent
- ▶ Limité par le nombre de pipeline indépendants.

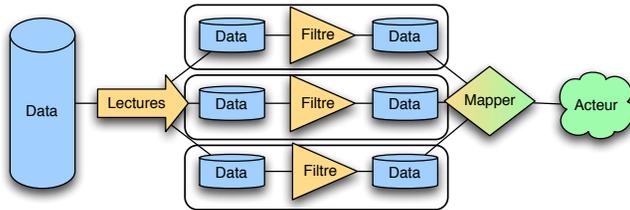
103

Parallélisation des données

104

Parallélisation des données

▶ Similaire au streaming dans le principe, mais avec duplication des traitements.



▶ Avantages

- ▶ Parallélisation avantageuse dans le cas de traitements indépendants et locaux.
- ▶ Supporte la mise à l'échelle.

▶ Inconvénients

- ▶ Coût de communication lors d'un déploiement cluster.
- ▶ Opération coûteuse dans sa globalité.

105

Plan du cours (3/3)

▶ Informations techniques et développement logiciel

- ▶ Compilation, installation et déploiement
- ▶ Mise en Oeuvre C++/Python/Java

▶ VTK et son wrapping Java

- ▶ Pourquoi Java dans la visualisation scientifique ?
- ▶ Utilisation du wrapping Java
- ▶ Configuration du Wrapping Java
- ▶ Outils de développement Java

106

Informations techniques et développement logiciel

- ▶ Compilation, installation et déploiement
- ▶ Mise en oeuvre C++
- ▶ Mise en oeuvre Python
- ▶ Mise en oeuvre Java

107

Pré-requis et dépendances

- ▶ Compilation :
 - ▶ Base : ar, make, gcc/g++, X11, ranlib, OpenGL/MesaGL
 - ▶ Wrapping : java, javac, python, tcl...
 - ▶ Export : libtiff, postscript...
 - ▶ Avancé : MPI...
- ▶ Utilisation :
 - ▶ Base : X11, OpenGL/MesaGL
 - ▶ Wrapping : java, python, tcl...
 - ▶ Export : libtiff...
 - ▶ Avancé : MPI...

Remarque: Par défaut sous Linux/Unix, VTK est compilé en mode liens dynamiques. Les bibliothèques externes sont nécessaires à l'exécution et lors du link de codes utilisant VTK.

Remarque: Sous Linux/UNIX, VTK est assez dépendant des bibliothèques bas niveau, dont glibc.

Remarque: VTK repose sur OpenGL et, indirectement, sur les fonctions avancées de rendu 3D du pilote de/des carte(s) vidéo mise(s) en oeuvre. La stabilité du système dans son ensemble peut être très fortement dépendante du réglage / installations correctes de tous ces éléments (pilotes, accélération 3D...).

108

Compilation et installation de VTK

- ▶ Téléchargement de VTK : <http://public.kitware.com/VTK/get-software.php#latest>
 - ▶ Binaires Windows (avec wrapping partiel), sources multi plate-forme
 - ▶ Miroir (partiel): <http://dev.artenum.com/projects/cassandra> (binaires avec wrapping java pour Linux, Windows)
- ▶ Configuration: cmake /ccmake
 - ▶ cmake : générateur multi plate-forme de makefile, interface texte
 - ▶ ccmake : version avancée de cmake avec interface test plus avancée, mais parfois moins robuste
 - ▶ Avec cmake (<http://www.cmake.org/HTML/Index.html>) :
 - ▶ Première configuration automatique : `cmake $VTK_ROOT`
 - ▶ Deuxième configuration manuelle : `cmake -i $VTK_ROOT`
 - ▶ + Choisir “options avancées”

109

Compilation de VTK (1/2)

- ▶ Il est recommandé de faire une configuration/compilation étape par étape:
 - ▶ Couche C/C++ (réglages de base) (1)
 - ▶ Couche wrapping Tcl (2)
 - ▶ Autres couches wrapping (Python, Java) (2) (3)
 - ▶ Modules d’export (e.g tiff, png, postscript...)
 - ▶ Modules avancés (MPI...)

(1) Suffisant pour utilisation directe de la couche C,VTK_RENDERERING à ON pour visualisation

(2) Nécessite BUILD_SHARED_LIBS à ON

(3) Avec VTK 4.XX, nécessitent VTK_USE_HYBRID,VTK_USE_PARALLEL,VTK_USE_PATENTED à ON

110

Mise en oeuvre de scripts Python

- ▶ Réglage des variables d'environnement
 - ▶ Chargement des librairies dynamiques
 - ▶ Linux : export LD_LIBRARY_PATH=\$VTK_ROOT/lib
 - ▶ Windows : PATH=%VTK_ROOT%\bin;%VTK_ROOT%\lib
 - ▶ Chargement des modules Python (avec interpréteur python standard)
 - ▶ Linux/Windows : PYTHONPATH= \$VTK_ROOT/
 - ▶ Les interpréteurs
 - ▶ python (interpréteur standard): utilisation standard, meilleure intégration avec bibliothèques externes (i.g. PyQt...)
 - ▶ vtkpython (interpréteur fourni par VTK): Meilleures performances, meilleure intégration avec l'ensemble des composants VTK

113

Mise en oeuvre de codes Java

- ▶ Réglage des variables d'environnement
 - ▶ Chargement des librairies dynamiques
 - ▶ Linux : LD_LIBRARY_PATH=\$VTK_ROOT/lib
 - ▶ Windows : PATH=%VTK_ROOT%\bin;%VTK_ROOT%\lib

114

VTK et son wrapping Java

- ▶ Pourquoi Java dans la visualisation scientifique ?
- ▶ Utilisation du wrapping Java
- ▶ Configuration du Wrapping Java
- ▶ Outils de développement Java

115

Intérêt de Java

- ▶ Java permet le développement d'applications graphiques multi plate-forme simplement.
- ▶ La rigueur du langage permet de réduire les coûts de développement et augmente la pérennité du code.
- ▶ Un grand nombre de bibliothèques Java est disponible. (Apache...)
- ▶ Java offre des possibilités intéressantes de par ses possibilités de chargement à chaud de composants et de par la réflexivité assurée par le langage.
- ▶ Ouvre des possibilités étendues de déploiement. (Web, 3d...)

116

VTK et les autres langages

- ▶ VTK est une bibliothèque native écrite en C++
- ▶ Cependant VTK peut être manipulé par d'autres langages comme...
 - ▶ Tcl/Tk : Le seul wrapping fournit par défaut
 - ▶ Python : Accessible après recompilation de VTK
 - ▶ Java : Accessible après recompilation de VTK

117

Java et les codes natifs

- ▶ Java Native Interface permet de faire un pont entre des codes natifs C/C++ et du Java. (Dans les 2 sens)
- ▶ Contraintes et avantages
 - ▶ Le code Java généré reste générique et indépendant de la plate-forme hôte.
 - ▶ Les bibliothèques natives sont dépendantes de la machine hôte.
 - ▶ Une variable système doit être définie pour assurer le chargement des codes natifs par la JVM.

118

JNI et VTK

- ▶ Après une compilation spécifique, VTK fournit un fichier “vtk.jar” qui encapsule de manière transparente toute la couche native de VTK.
- ▶ L'ensemble des objets VTK peuvent être manipulés comme n'importe quel objet Java.
- ▶ Cependant JNI, impose malgré tout une petite contrainte. Une variable système doit pointer vers les différentes couches natives appelées.

119

Contraintes de JNI

- ▶ Pour Linux
 - ▶ export LD_LIBRARY_PATH=\$VTK_HOME/lib
- ▶ Pour Windows
 - ▶ SET PATH=%PATH%;%VTK_HOME%/lib
- ▶ Pour Mac OS X
 - ▶ export DYLD_LIBRARY_PATH=\$VTK_HOME/lib

120

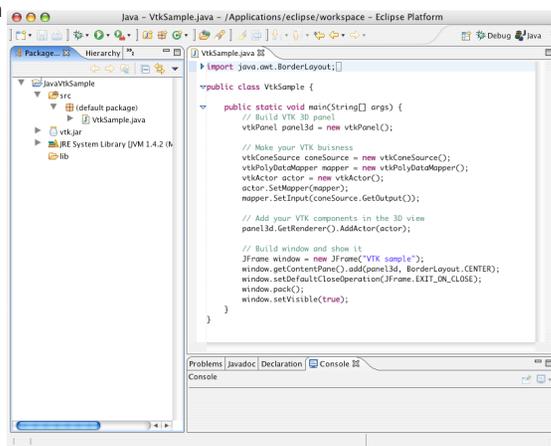
Programmer du VTK avec eclipse

- ▶ Eclipse est un IDE avancé pour Java avec un grand nombre de fonctionnalités
 - ▶ Framework de plug-ins (UML, SwingBuilder...)
 - ▶ Complétion objet
 - ▶ Compilation incrémentale
 - ▶ Débugueur
 - ▶ Gratuit...

121

VTK et Eclipse comme IDE

- ▶ Exemple de l'utilisation de VTK dans Eclipse



122

Exemple VTK de base

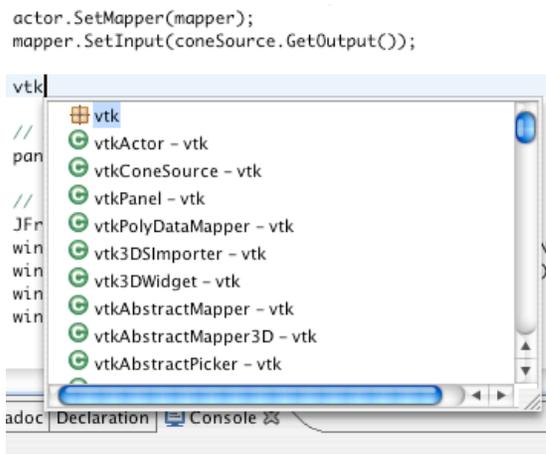
► Du code Java standard

```
public class VtkSample {  
    public static void main(String[] args) {  
        // Build VTK 3D panel  
        vtkPanel panel3d = new vtkPanel();  
  
        // Make your VTK buisness  
        vtkConeSource coneSource = new vtkConeSource();  
        vtkPolyDataMapper mapper = new vtkPolyDataMapper();  
        vtkActor actor = new vtkActor();  
        actor.SetMapper(mapper);  
        mapper.SetInput(coneSource.GetOutput());  
  
        // Add your VTK components in the 3D view  
        panel3d.GetRenderer().AddActor(actor);  
  
        // Build window and show it  
        JFrame window = new JFrame("VTK sample");  
        window.getContentPane().add(panel3d, BorderLayout.CENTER);  
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        window.pack();  
        window.setVisible(true);  
    }  
}
```

123

Complétion sur les objets VTK

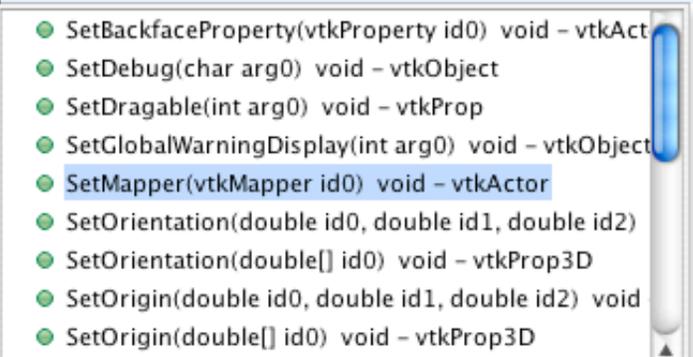
► La complétion de manière transparente



124

... avec le type des paramètres

```
vtkActor *actor = vtkActor();  
actor.SetMapper(mapper);  
  
// Add your panel3d.G  
  
// Build  
JFrame window;  
window.getContentPane().add(actor);  
window.setVisible(true);
```



- SetBackfaceProperty(vtkProperty id0) void - vtkActor
- SetDebug(char arg0) void - vtkObject
- SetDragable(int arg0) void - vtkProp
- SetGlobalWarningDisplay(int arg0) void - vtkObject
- SetMapper(vtkMapper id0) void - vtkActor**
- SetOrientation(double id0, double id1, double id2)
- SetOrientation(double[] id0) void - vtkProp3D
- SetOrigin(double id0, double id1, double id2) void
- SetOrigin(double[] id0) void - vtkProp3D

125

Discussion ouverte

Sébastien Jourdain : jourdain@artenum.com
Julien Forest : j.forest@artenum.com

126
