

Visualisation :
présentation
de VTK et
ParaView

Sylvain Faure

Plan de
l'exposé

Introduction

VTK

Introduction
Les formats de
fichiers VTK
Extraction de
primitives
Pipeline de
visualisation
Objets VTK de
base
Applications

Paraview

Introduction
Installation et
lancement
Chargement des
données
Manipulation
des données
Démonstrations

Références

Visualisation : présentation de la librairie VTK (Visualization ToolKit) et de l'application Paraview (Parallel Visualization Application)

Ecole d'Automne Informatique Scientifique (module 2)

Sylvain Faure

Université Paris-Sud, Laboratoire de Mathématiques

5 décembre 2008

1 Introduction

2 VTK

Introduction

Les formats de fichiers VTK

Extraction de primitives

Pipeline de visualisation

Objets VTK de base

Applications

3 Paraview

Introduction

Installation et lancement

Chargement des données

Manipulation des données

Démonstrations

4 Références

Contexte

Deux façons de visualiser des résultats numériques :

- Sur sa machine de bureau : tout est local, le rendu, l'affichage... C'est valable pour des données de petites tailles.
- Sur un cluster graphique dédié : le rendu est calculé en parallèle sur un supercalculateur graphique ou sur une grappe de PC équipée de cartes graphiques performantes. L'image est ensuite compressée et envoyée sur une machine de visualisation, un mur d'image, le PC du bureau du chercheur ou le PC de son domicile...

Cahier des charges

Afin de pouvoir manipuler des données volumiques, il faut pouvoir :

- Bouger ce que l'on visualise : manipulation interactive obligatoire pour des données 3D.
- Adapter la qualité de l'image en fonction du matériel utilisé : représentation multirésolution.
- Paralléliser le rendu dans le processus de visualisation.

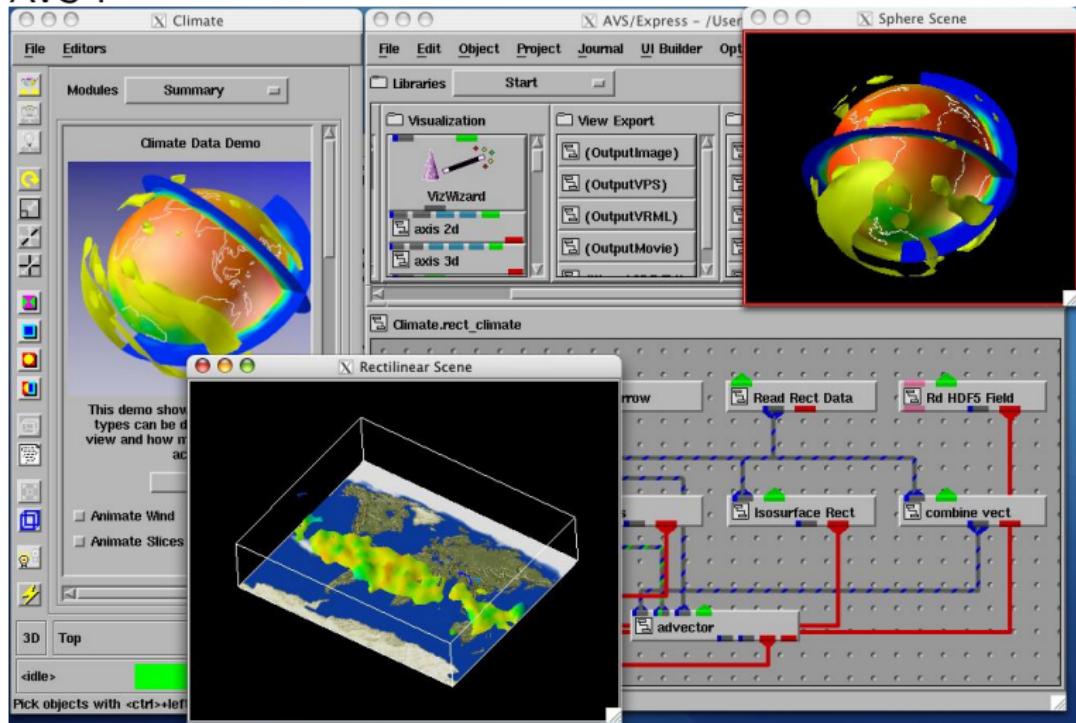
Logiciels de visualisation scientifique

Les logiciels propriétaires et commerciaux :

- AVS/EXPRESS (v7.2.1) : Fondé en 1991, longtemps leader du domaine. C'est un atelier de développement avec modèle d'exécution par flots de données, programmable via une interface regroupant des fonctions de lecture, de traitement, de représentations graphiques et d'affichage de données sous la forme de modules que l'on relie entre eux.
- CEI EnSight (v9.0.2) : Fondé en 1994, devenu le leader du domaine. EnSight est capable de lire directement les fichiers de solutions de nombreux solveurs couramment utilisés : StarCD, Ansys, Fluent, Marc, CFX,... Son interface, réputée très intuitive, est simple, mais très puissante et permet de mélanger des données structurées ou non structurées.

Logiciels de visualisation scientifique

AVS :



Plan de l'exposé

Introduction

VTK

Introduction

Les formats de fichiers VTK

Chapitre 1
Fichiers VTK

Pipeline de visualisation

Visualisation
Objets VTK de base

Base Applications

Paraview

Introduction

Installation et
lancement

Chargement des données

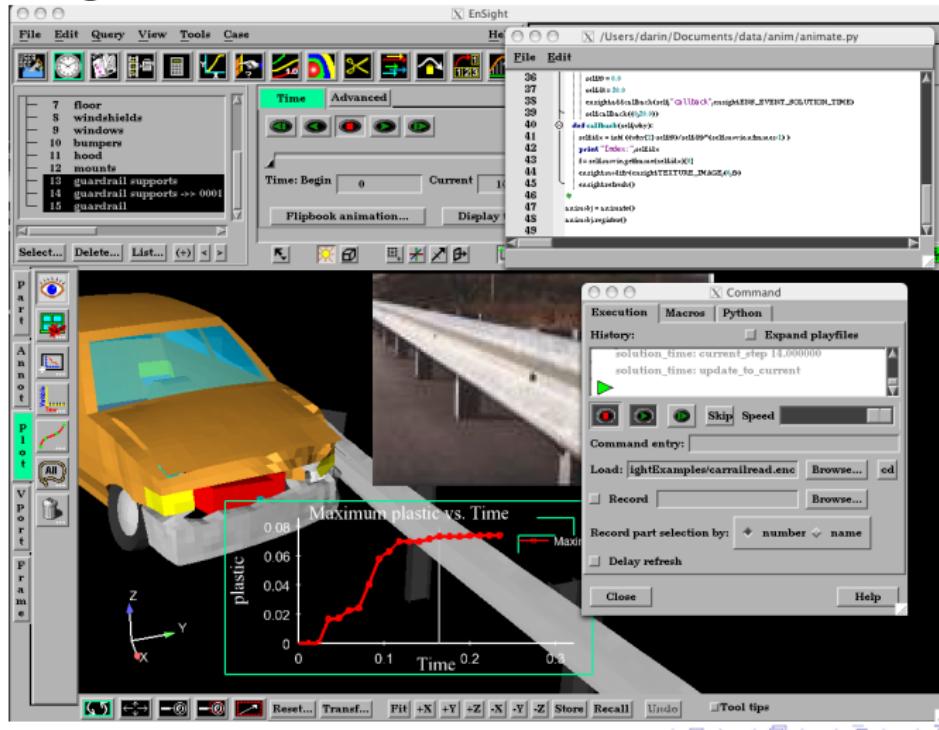
Manipulation des données

des données
Démonstrations

Références

Logiciels de visualisation scientifique

Ensight :



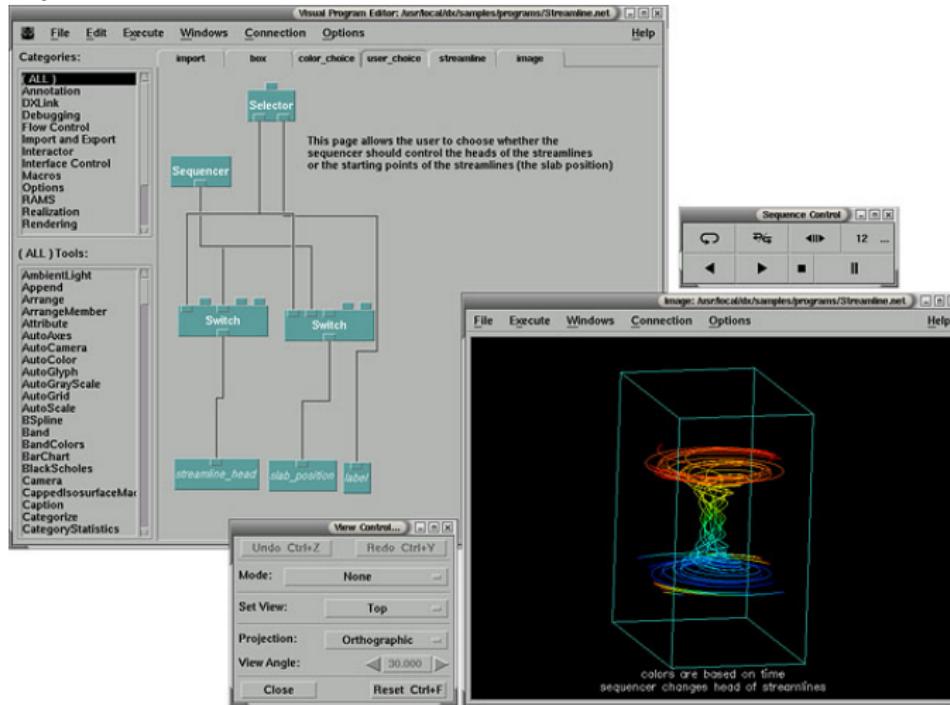
Logiciels de visualisation scientifique

Les logiciels libres :

- OpenDX (v4.4.4) : Produit à l'origine par IBM, DX (Data Explorer) est devenu un logiciel libre sous le nom d'OpenDX. Egalement "data flow" (exécution par flots de données) avec un éditeur visuel de programmes.
- VTK (v5.2.0) : Librairie de visualisation écrite en C++. C'est un système de type "data-flow". Le processus de visualisation utilisé par VTK est le suivant : les données brutes, introduites en début de pipeline, sont transformées et traitées par celui-ci de manière à donner une image. Deux principales interfaces graphiques : Paraview (v3.4.0) et MayaVi2.

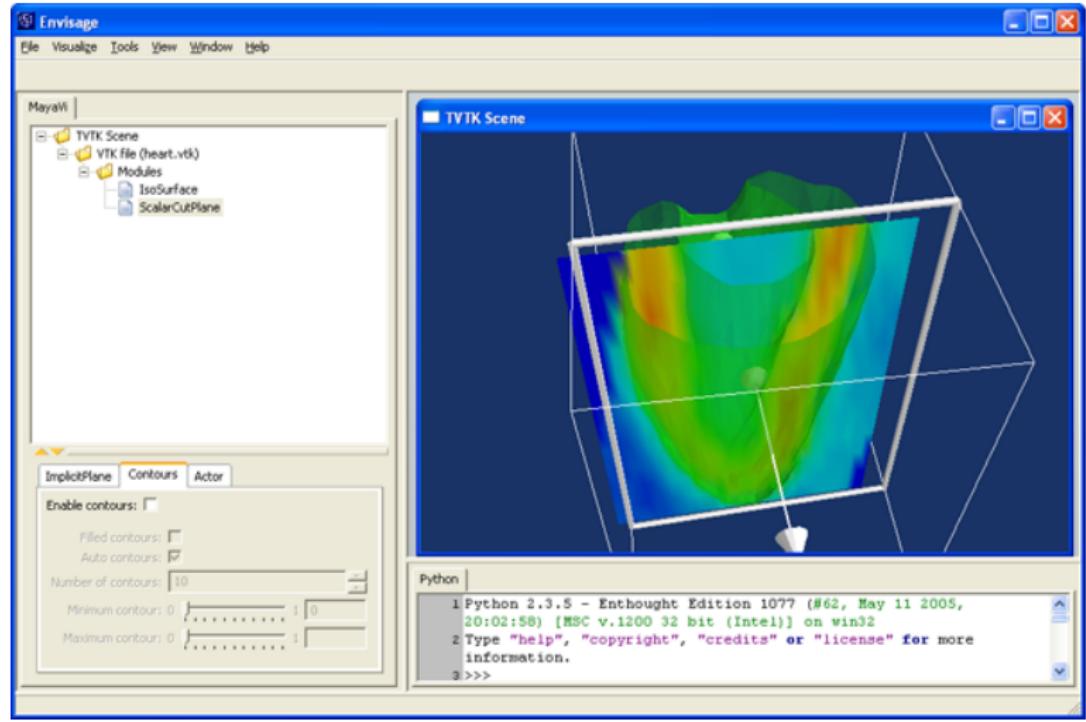
Logiciels de visualisation scientifique

OpenDX :



Logiciels de visualisation scientifique

Mayavi 2 :

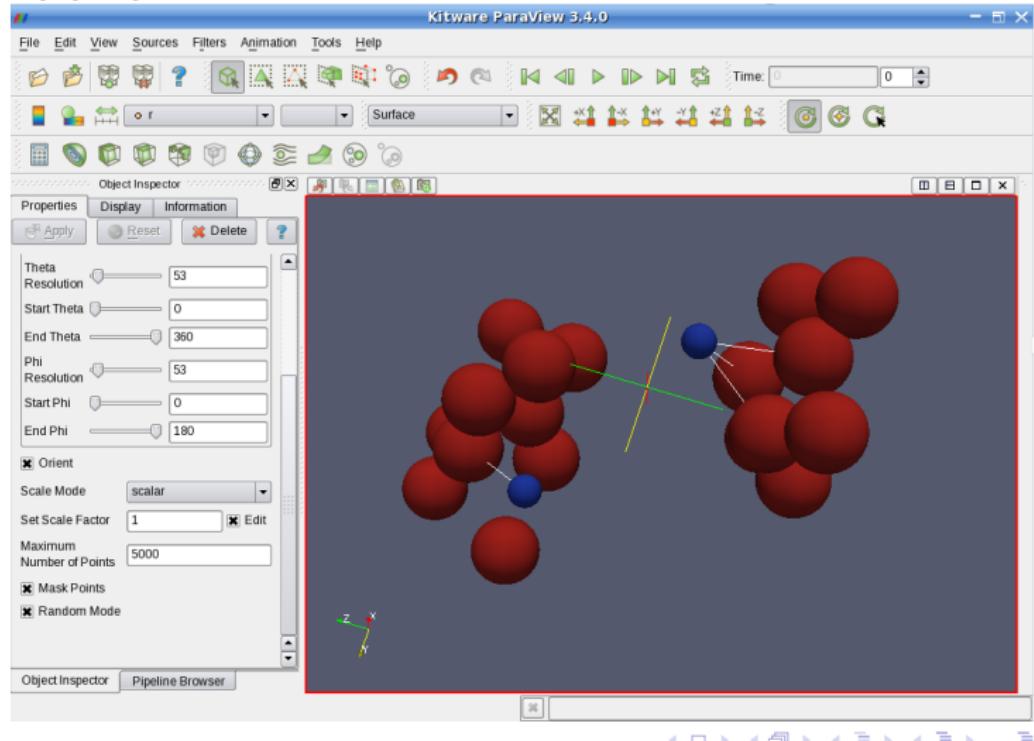


Introduction
Les formats de
fichiers VTK
Extraction de
primitives
Pipeline de
visualisation
Objets VTK de
base
Applications

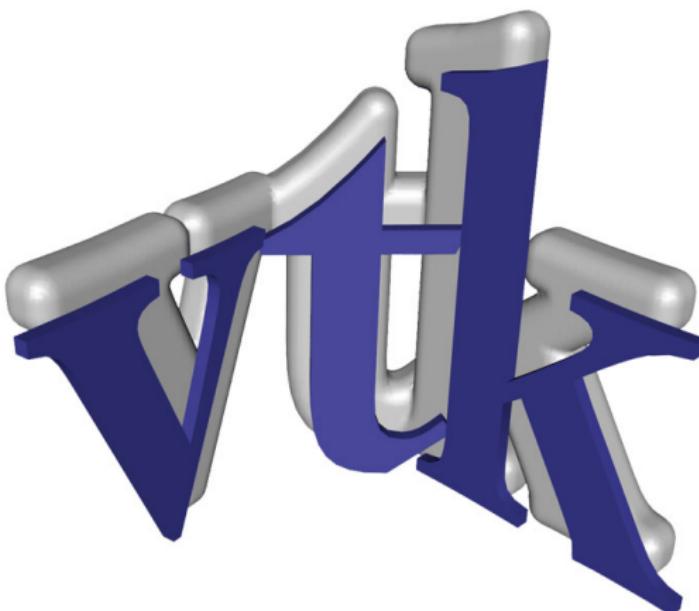
Introduction
Installation et
lancement
Chargement des
données
Manipulation
des données
Démonstrations

Logiciels de visualisation scientifique

ParaView :



Première partie



Qu'est-ce ?

VTK est une bibliothèque orientée objet de visualisation scientifique écrite en *C++* permettant :

- Le traitement de données 2D et 3D (de très nombreux algorithmes de visualisation sont implantés).
- Une visualisation parallèle. Parallélisme de tâche, de pipeline et de données.
- Une interactivité.
- Le développement d'applications spécifiques, voir de nouvelles classes.
- Un accès simplifié grâce aux interfaces Python, TcL/Tk, Java.

Compilation sous Unix

Ce qui est nécessaire au bon fonctionnement de *VTK* :

- *cmake* pour la compilation.
- *OpenGL* (librairie liée à la carte graphique)
- Des librairies *X11* récentes...

Ce qui est optionnel :

- *Tcl/Tk*
- *Python*
- *Java*
- *MPI*

Compilation sous Unix

Procédure d'installation :

- Télécharger l'archive :
`http://public.kitware.com/VTK/files`
- Décompresser l'archive `vtk.tar.gz` dans un répertoire
`/usr/local/src/vtk/`
- Avec l'utilitaire `cmake`, générer les fichiers `Makefile` :
`[/usr/local/vtk/]$ ccmake .../src/vtk/`
(Taper `[t]` pour avoir toutes les options)
Lancer la configuration `[c]`, vérifier les options, relancer éventuellement la configuration `[c]`. Enfin, générer les fichiers `Makefile` avec `[g]`.
- Compiler :
`[/usr/local/vtk/]$ make`
- En tant que `root`, finir l'installation :
`[/usr/local/vtk/]$ make install`

Compilation sous Unix

```
root@localhost: /usr/local/vtk-5.2
Fichier Édition Affichage Terminal Onglets Aide
Page 1 of 1

BUILD_EXAMPLES OFF
BUILD_SHARED_LIBS ON
BUILD_TESTING ON
CMAKE_BACKWARDS_COMPATIBILITY 2.4
CMAKE_BUILD_TYPE RELEASE
CMAKE_INSTALL_PREFIX /usr/local
VTK_DATA_ROOT /usr/local/src/VTKData
VTK_USE_INFOVIS ON
VTK_USE_PARALLEL OFF
VTK_USE_RENDERING ON
VTK_USE_VIEWS ON
VTK_WRAP_JAVA OFF
VTK_WRAP_C C
VTK_WRAP_CXX C++
VTK_WRAP_TCL OFF
vtkMPEG2Encode_INCLUDE_PATH /usr/local/src/vtkmpeg2encode/
vtkMPEG2Encode_LIBRARIES /usr/local/vtkmpeg2encode/libvtkMPEG2Encode.a

BUILD EXAMPLES: Build VTK examples.
Press [enter] to edit option
Press [c] to configure      Press [g] to generate and exit
Press [h] for help          Press [q] to quit without generating
Press [t] to toggle advanced mode (currently off)
CMake Version 2.6 - patch 1
```

Compilation sous Unix

Options pour une compilation basique :

- *CMAKE_INSTALL_PREFIX=/usr/local*
- *BUILD_SHARED_LIBS=on*
- *BUILD_EXAMPLES=on*
- *VTK_USE_PARALLEL=off*
- *VTK_DATA_ROOT=/usr/local/VTKData*

Compilation sous Unix

Options supplémentaires pour une compilation avec Python :

- *VTK_WRAP_PYTHON=on*
- *PYTHON_INCLUDE_PATH=/usr/include/python*
- *PYTHON_LIBRARY=/usr/lib/python/config*

Vérifier que le répertoire */usr/lib/python* contient un répertoire *vtk* (correspondant aux modules Python compilés dans */usr/local/vtk/Wrapping/Python/vtk*).

Processus de visualisation scientifique

Des données à l'affichage, on a les étapes suivantes :

- 1. Conversion des données dans un format exploitable.
- 2. Extraction des primitives graphiques.
- 3. Rendu graphique : calcul d'une image à partir des primitives.
- 4. Affichage de l'image rendue.

Formats de fichiers VTK

Il existe deux familles de fichiers de données :

- Les formats de fichiers hérités du passé (“simple legacy formats”).
- Les formats *XML* (eXtensible Markup Language, successeur de l’*HTML*, c’est un métalangage permettant d’inventer à volonté de nouvelles balises).

Formats de fichiers VTK "classiques"

Ils sont composés de 5 parties :

- *# vtkDataFile Version 3.1.*
- L'en-tête (256 caractères maximum se terminant par \n).
- Le format des données : *ASCII* ou *BINARY*.
- La structure des données, *DATASET* :
STRUCTURED_POINTS, *STRUCTURED_GRID*,
UNSTRUCTURED_GRID, *POLYDATA*,
RECTILINEAR_GRID, *FIELD*.
- Les valeurs (*POINT_DATA*, *CELL_DATA*).

Visualisation : présentation de VTK et ParaView

Sylvain Faure

Plan de
l'exposé

Introduction

VTK

Introduction

Les formats de
fichiers VTK

Extraction de
primitives

Pipeline de
visualisation

Objets VTK de
base

Applications

Paraview

Introduction

Installation et
lancement

Chargement des
données

Manipulation
des données

Démonstrations

Références

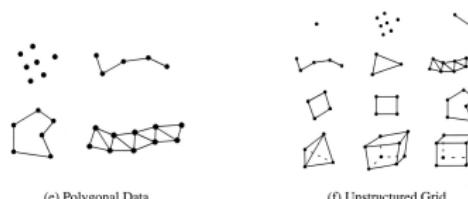
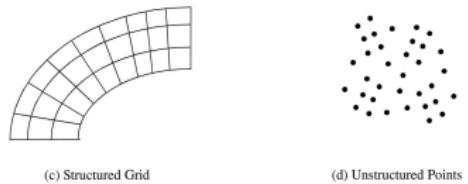
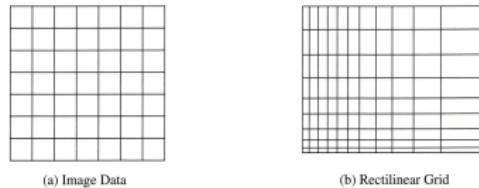


Figure 5-7 Dataset types. The unstructured grid consists of all cell types.

Formats de fichiers VTK

"classiques" : exemple

```
# vtk DataFile Version 3.1
```

```
Premier exemple \n
```

```
ASCII
```

```
DATASET UNSTRUCTURED_GRID
```

```
POINTS 9 FLOAT
```

```
0 0 0
```

```
0 1 0
```

```
0 2 0
```

```
1 2 0
```

```
1 1 0
```

```
1 0 0
```

```
2 0 0
```

```
2 1 0
```

```
2 2 0
```

Type de données
Coordonnées des
points nécessaires
pour ce DATASET

Formats de fichiers VTK

"classiques" : exemple

CELLS 4 20

4 0 5 4 1

4 1 4 3 2

4 5 6 7 4

4 4 7 8 3

CELL_TYPES 4

9 9 9 9

Nécessaire pour ce DATASET

Nombre de cellules : 4

Taille de la liste les définissant : 20

Pour chaque cellule : Nb_pts Liste_pts

Nécessaire pour ce DATASET

Type de chaque cellule : 9=VTK_QUAD

Visualisation : présentation de VTK et ParaView

Sylvain Faure

Plan de l'exposé

Introduction

VTK

Introduction
Les formats de fichiers VTK
Extraction de primitives
Pipeline de visualisation
Objets VTK de base
Applications

Paraview

Introduction
Installation et lancement
Chargement des données
Manipulation des données
Démonstrations

Références

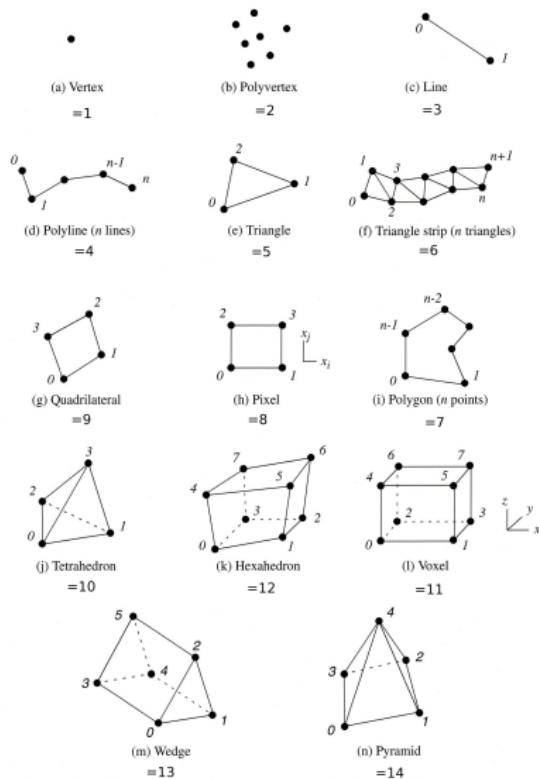


Figure 5–2 Linear cell types found in VTK. Numbers define ordering of the defining points.

VTK

Introduction
Les formats de fichiers VTK
Extraction de primitives
Pipeline de visualisation
Objets VTK de base
Applications

Paraview

- Introduction
- Installation et lancement
- Chargement des données
- Manipulation des données
- Démonstrations

Références

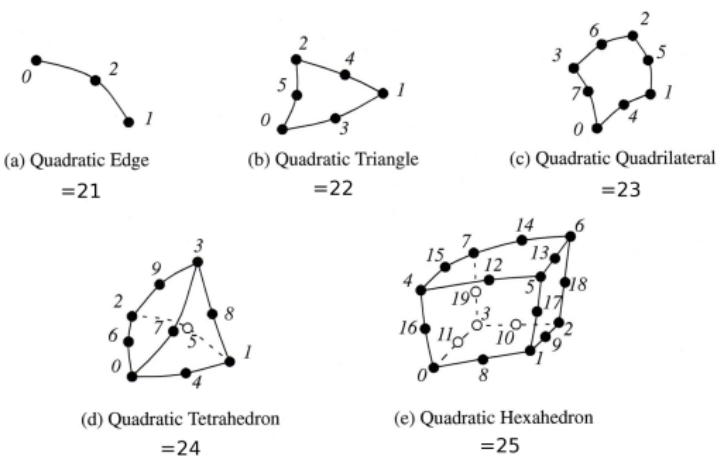


Figure 5–3 Non-linear cell types found in VTK.

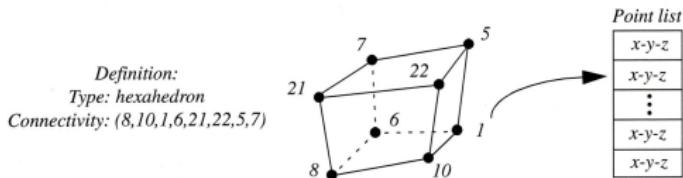


Figure 5-4 Example of a hexahedron cell. The topology is implicitly defined by the ordering of the point list.

Formats de fichiers *VTK*

"classiques" : exemple

```
POINT_DATA 9
SCALARS Pression FLOAT
LOOKUP_TABLE default
0
0
0
1
2
1
0
0
0
```

Données aux points : 2 scalaires
(Table des couleurs)

Formats de fichiers *VTK*

"classiques" : exemple

SCALARS Temperature FLOAT

LOOKUP_TABLE default

1

1.1

1.2

2.1

2.5

3.5

1.3

1.4

1.6

Formats de fichiers VTK "classiques" : exemple

CELL_DATA 4

SCALARS Cell_Temperature FLOAT
LOOKUP_TABLE default

0

1

2

1

Données cellules :
un scalaire

Fichiers VTK au format XML

Avantages/inconvénients :

- Plus délicats à utiliser.
- Permettent le chargement sur une architecture parallèle.
- Permettent de compresser les données.

Deux catégories :

- Structurés : `ImageData (.vti)`, `RectilinearGrid (.vtr)`,
`StructuredGrid (.vts)`, `PlImageData (.pvti)`,
`PRectilinearGrid (.pvtr)`, `PStructuredGrid (.pvti)`.
- Non structurés : `PolyData (.vtp)`, `UnstructuredGrid (.vtu)`,
`PPolyData (.pvti)`, `PUnstructuredGrid (.pvtu)`.

Fichiers VTK au format XML : *ImageData*

Les points et cellules sont définis implicitement.

```
<?xml version = "1.0"?>
< VTKFile type = "ImageData" version = "0.1" >
< imageData WholeExtent = "x1 x2 y1 y2 z1 z2"
  Origin = "x0 y0 z0" Spacing = "dx dy dz" >
< Piece Extent = "x1 x2 y1 y2 z1 z2" >
< PointData Vectors = "velocity" Scalars = "pressure" >
< dataArray type = "Float32" Name = "velocity"
  NumberOfComponents = "3" format = "ascii" >
  u0 v0 w0
  ...
< /dataArray >
```

Fichiers VTK au format XML : *ImageData*

```
< DataArray type = "Float32" Name = "pressure" format = "ascii" >
  p0
  ...
< /DataArray >
< /PointData >
< CellData >
  ...
< /CellData >
< /Piece >
< /ImageData >
< /VTKFile >
```

Fichiers VTK au format XML : *RectilinearGrid*

Les points sont décrits par leurs coordonnées < *Coordinates* >.

```
<?xml version = "1.0"?>
< VTKFile type = "RectilinearGrid" version = "0.1" >
< RectilinearGrid WholeExtent = "x1 x2 y1 y2 z1 z2" >
< Piece Extent = "x1 x2 y1 y2 z1 z2" >
< PointData > ... < /PointData >
< CellData > ... < /CellData >
< Coordinates > ... < /Coordinates >
< /Piece >
< /RectilinearGrid >
< /VTKFile >
```

Fichiers VTK au format XML : *StructuredGrid*

Les points sont décrits par leurs coordonnées < *Points* >.

```
<?xml version = "1.0"?>
< VTKFile type = " StructuredGrid" version = "0.1" >
< StructuredGrid WholeExtent = " x1 x2 y1 y2 z1 z2" >
< Piece Extent = " x1 x2 y1 y2 z1 z2" >
< PointData > ... < /PointData >
< CellData > ... < /CellData >
< Points > ... < /Points >
< /Piece >
< /StructuredGrid >
< /VTKFile >
```

Fichiers VTK au format XML : *UnstructuredGrid*

Les points sont définis explicitement par *<Points>*. Les cellules sont définies par *<Cells>*.

```
<?xml version = "1.0"?>
<VTKFile type = "UnstructuredGrid" version = "0.1" >
<UnstructuredGrid>
<Piece NumberOfPoints = "#" NumberOfCells = "#" >
<PointData> ... </PointData>
<CellData> ... </CellData>
<Points> ... </Points>
<Cells> ... </Cells>
</Piece>
</UnstructuredGrid>
</VTKFile>
```

Fichiers VTK au format XML :

PolyData

Les points sont définis explicitement par `<Points>`. Les cellules sont définies par `<Verts>`, `<Lines>`, `<Strips>` et `<Polys>`.

```
<?xml version = "1.0"?>
<VTKFile type = "PolyData" version = "0.1">
<Piece NumberOfPoints = "#" NumberOfVerts = "#" 
NumberOfLines = "#" NumberOfStrips = "#" 
NumberOfPolys = "#" >
<PointData> ... </PointData>
<CellData> ... </CellData>
<Points> ... </Points>
<Verts> ... </Verts>
<Lines> ... </Lines>
<Strips> ... </Strips>
<Polys> ... </Polys>
</Piece>
</PolyData>
</VTKFile>
```

Fichiers VTK au format XML : *PlImageData*

Le fichier ci-dessous (.pvti) recolle les morceaux (.vti)

```
<?xml version = "1.0"?>
< VTKFile type = " PlImageData" version = "0.1" >
< PlImageData WholeExtent = " 0 32 0 32 0 1"
  GhostLevel = "0" Origin = " 0 0 0"
  Spacing = " 0.03125 0.03125 0.0001" >
< PPointData >
< /PPointData >
< PCellData Vectors = " velocity" Scalars = " pressure" >
< dataArray type = " Float32" Name = " velocity"
  NumberOfComponents = "3" format = " ascii" / >
< dataArray type = " Float32" Name = " pressure"
  format = " ascii" / >
< /PCellData >
```

Fichiers VTK au format XML :

PlImageData

```
< Piece Extent = "0 16 0 16 0 1" Source = "Gr4 - 000.001.vti" / >
< Piece Extent = "16 32 0 16 0 1" Source = "Gr4 - 001.001.vti" / >
< Piece Extent = "0 16 16 32 0 1" Source = "Gr4 - 002.001.vti" / >
< Piece Extent = "16 32 16 32 0 1" Source = "Gr4 - 003.001.vti" / >
< /PlImageData >
< /VTKFile >
```

Fichiers VTK : Exemples...

- moulinette-1 : Conversion de fichiers de données de types reconnus par *VTK*. On passe d'un .case (ASCII) à un .vtu (XML, binaire donc utilisable et performant sur une architecture parallèle).
- moulinette-2 : Construction "manuelle" d'un fichier de données de type .vtu (XML, binaire).
- cellules (exemple fourni par *VTK*) : Construction "manuelle" de tous les types de cellules de maillages non structurés.

Une fois les données converties du format natif vers un format exploitable, on en extrait les caractéristiques les plus importantes sous forme de primitives :

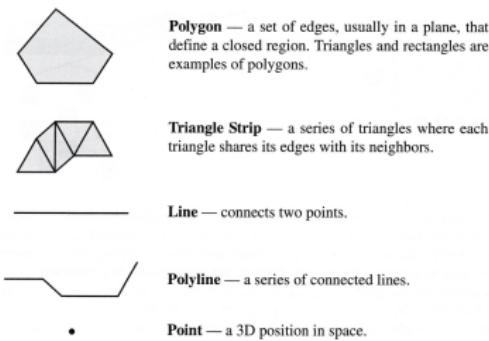


Figure 3–19 Graphics primitives.

Cela évite de regarder les images pixels par pixels ! OpenGL (Open Graphics Library) est une interface qui regroupe plusieurs centaines de fonctions différentes permettant d'afficher des scènes tridimensionnelles complexes à partir de simples primitives. Elle est généralement implantée par les constructeurs de matériels graphiques et y est étroitement liée. Le rendu : calcul d'une image à partir de primitives extraites

Programmation orientée objet

La librairie VTK est écrite en C++ en respectant de façon rigoureuse les concepts de la programmation orientée objet. C'est vital pour une librairie qui n'est utilisable qu'en parcourant l'arborescence des différentes classes la composant : <http://public.kitware.com/VTK/doc/release/5.2/html/>

- Notion d'héritage (en anglais inheritance) : permet de créer une nouvelle classe à partir d'une classe existante. La classe dérivée (i.e. la classe nouvellement créée) contient les attributs et les méthodes de sa superclasse (i.e. la classe dont elle dérive).
- Intérêt majeur : permet de définir de nouvelles méthodes et de nouveaux attributs pour la classe dérivée, qui viennent s'ajouter à celles et ceux hérités. On crée donc une hiérarchie de classes de plus en plus spécialisées.

Programmation orientée objet

Sylvain Faure

Plan de
l'exposé

Introduction

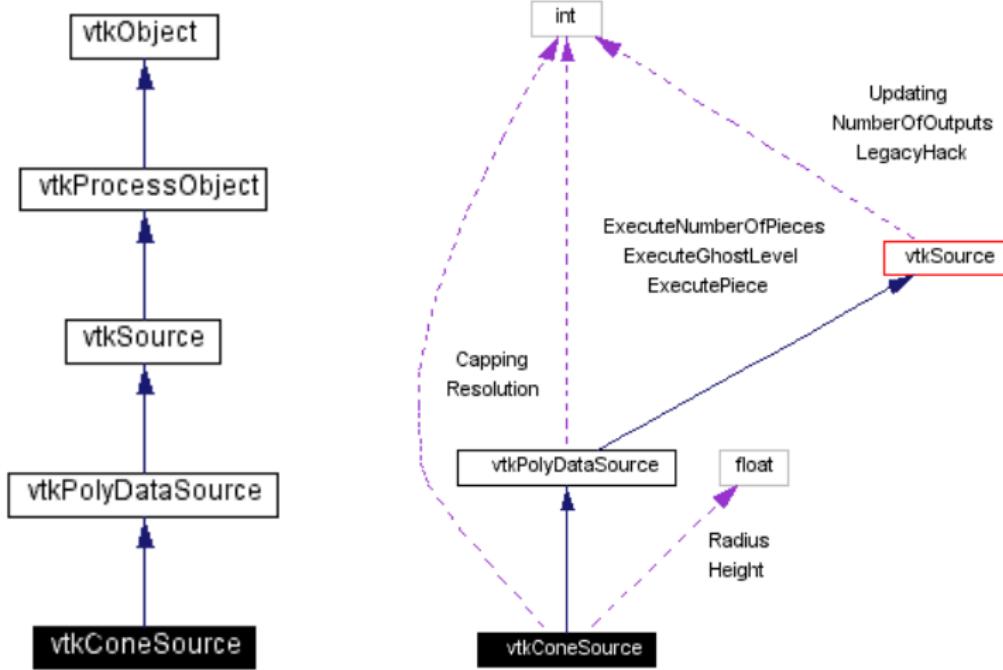
VTK

Introduction
Les formats de
fichiers VTK
Extraction de
primitives
Pipeline de
visualisation
Objets VTK de
base
Applications

Paraview

Introduction
Installation et
lancement
Chargement des
données
Manipulation
des données
Démonstrations

Références



Programmation orientée objet

vtkConeSource

Public Methods

```
virtual const char * GetClassName ()  
virtual int IsA (const char *type)  
void PrintSelf (ostream &os, vtkIndent indent)  
  
virtual void SetHeight (float)  
virtual float GetHeight ()  
  
virtual void SetRadius (float)  
virtual float GetRadius ()  
  
virtual void SetResolution (int)  
virtual int GetResolution ()  
  
void SetAngle (float angle)  
float GetAngle ()  
  
virtual void SetCapping (int)  
virtual int GetCapping ()  
virtual void CappingOn ()  
virtual void CappingOff ()
```

vtkObject

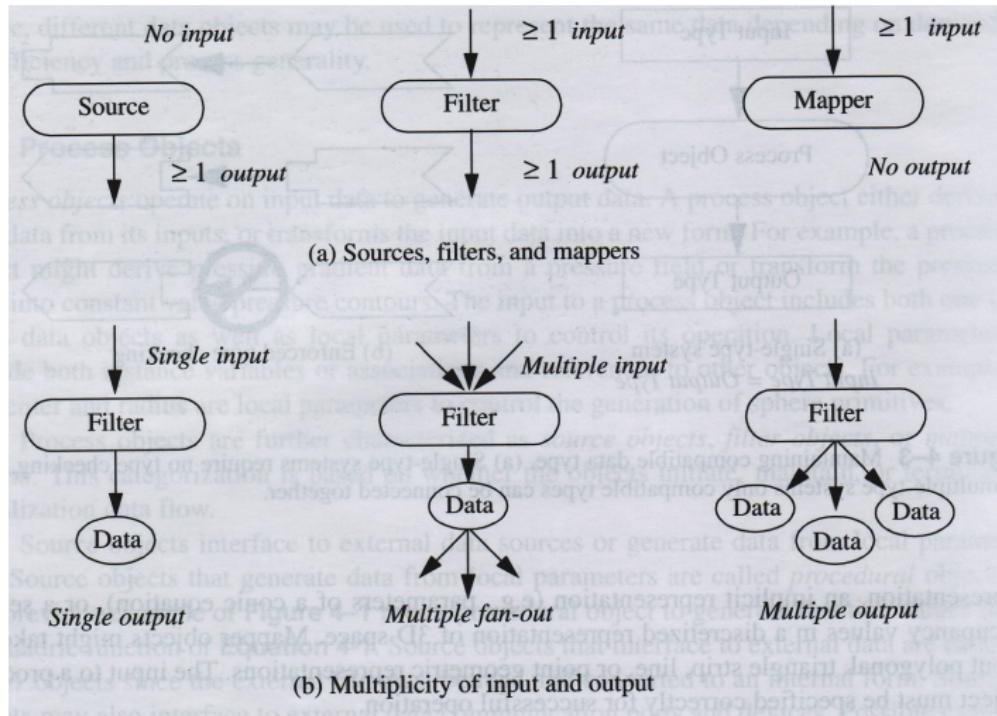
Public Methods

```
virtual const char * GetClassName ()  
virtual int IsA (const char *name)  
virtual void Delete ()  
virtual void DebugOn ()  
virtual void DebugOff ()  
unsigned char GetDebug ()  
void SetDebug (unsigned char debugFlag)  
virtual void Modified ()  
virtual unsigned long GetMTime ()  
void Print (ostream &os)  
void Register (vtkObject *o)  
virtual void UnRegister (vtkObject *o)  
void SetReferenceCount (int)  
  
virtual void PrintSelf (ostream &os, vtkIndent indent)  
virtual void PrintHeader (ostream &os, vtkIndent indent)  
virtual void PrintTrailer (ostream &os, vtkIndent indent)  
  
int GetReferenceCount ()  
  
unsigned long AddObserver (unsigned long event, vtkCommand *)  
unsigned long AddObserver (const char *event, vtkCommand *)  
vtkCommand * GetCommand (unsigned long tag)  
void InvokeEvent (unsigned long event, void *callData)  
void InvokeEvent (const char *event, void *callData)  
void RemoveObserver (unsigned long tag)  
int HasObserver (unsigned long event)  
int HasObserver (const char *event)
```

Pipeline et filtres : définitions

- Chaque classe de VTK représente un filtre avec entrée(s) et sortie(s).
- Le pipeline de visualisation est une succession cohérente de filtres reliés les uns aux autres permettant d'aboutir à une image.
- Exécution du pipeline : chaque filtre possède une méthode *Execute()* utilisée manuellement ou le plus souvent automatiquement. Il existe également une méthode *Update()* qui vérifie si les données intérieures au filtre sont bien à jour et les met à jour si nécessaire.
- Conséquence : un objet filtre ne doit jamais être effacé de la mémoire brutalement avec la commande *delete* du C++. Il faut toujours utiliser la méthode *Delete()* implantée dans le filtre.

Pipeline et filtres : définitions



Réf. : *The Visualization Toolkit An Object-Oriented Approach To 3D Graphics 3rd Edition (Kitware)*.

Connecter les filtres pour former le pipeline

Pour construire le pipeline, on doit posséder des méthodes permettant de connecter les filtres :

- Pour connecter une entrée :
SetInputConnection(port, input)
- Pour ajouter une entrée :
AddInputConnection(port, input)
- Pour enlever une entrée :
RemoveInputConnection(port, input)
- Pour récupérer une sortie :
GetOutputPort(port)
- Remarque : on peut réaliser une connexion même si la sortie n'est pas encore executable (il peut par exemple manquer des données...).
- Usage classique :
filtre2 -> SetInputConnection(0, filtre1 -> GetOutputPort(0))

Connecter les filtres pour former le pipeline : ancien style

Une ancienne façon de connecter les filtres est encore bien présente dans de nombreux exemples :

filtre2 –> SetInput(filtre1 –> GetOutput())

Principal inconvénient : nécessite la connaissance des types de données au moment de la connection. C'est problématique quand un filtre produit une multitude de sorties car cela oblige de reformer le pipeline à chaque changement de type de sortie.

Comparaison :

- *void vtkPolyDataAlgorithm :: SetInput(int, vtkDataObject *)*
- *virtual void vtkAlgorithm :: SetInputConnection(int, vtkAlgorithmOutput *)*

Types de filtres

Il existe deux types d'objets en VTK :

- “Data Objects” : pour charger les données à partir de multiples types de fichiers par exemple (voir `vtkDataSetSource`)
- “Process Objects” : pour modifier les données entrées et donc les utiliser !

Il est bien entendu possible d'implanter ses propres filtres à condition de définir au minimum les méthodes `SetInputConnection`, `GetOutputPort`, et `Execute...`

Exécution et mise à jour du pipeline

L'exécution du pipeline est contrôlée de façon implicite par VTK : chaque fois qu'une sortie est requise par un filtre. Les filtres se mettent à jour d'eux mêmes ("en remontant le pipeline et récursivement") et du coup tout le pipeline est à jour.

Avantages :

- L'utilisateur n'a pas besoin de gérer l'exécution du pipeline. Dans certains cas, il peut cependant être amené à utiliser la méthode *Update*.
- Les filtres sont dynamiques : la parallélisation est rendue plus facile.
- La mise à jour des filtres comporte des étapes qui peuvent être asynchrones

Remarque : on peut boucler un pipeline, dans ce cas il faudra utiliser *Update* pour exécuter la boucle.

Exécution et mise à jour du pipeline

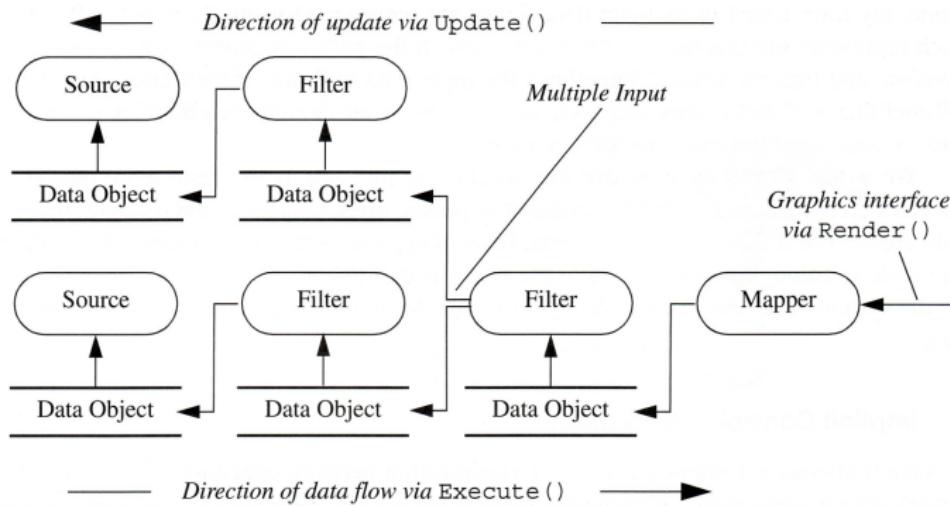
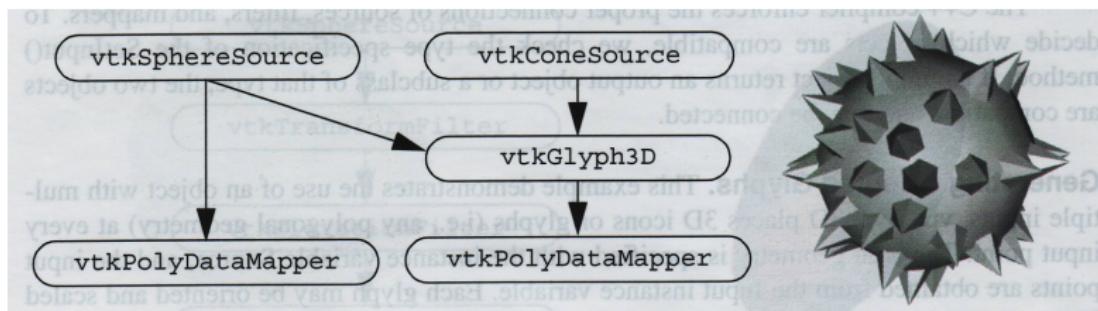
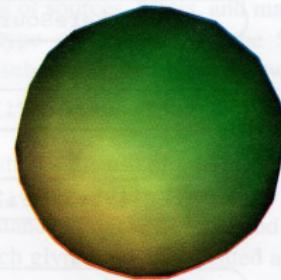
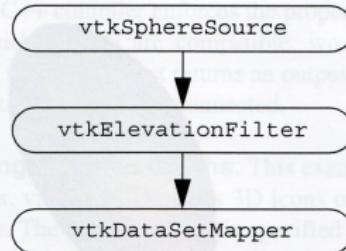


Figure 4-12 Description of implicit execution process implemented in VTK. The `Update()` method is initiated via the `Render()` method from the actor. Data flows back to mapper via `Execute()` method. Arrows connecting objects indicate direction of `Update()` process.

Exemples



Exemples



```
vtkSphereSource *sphere = vtkSphereSource::New();
    sphere->SetPhiResolution(12); sphere->SetThetaResolution(12);

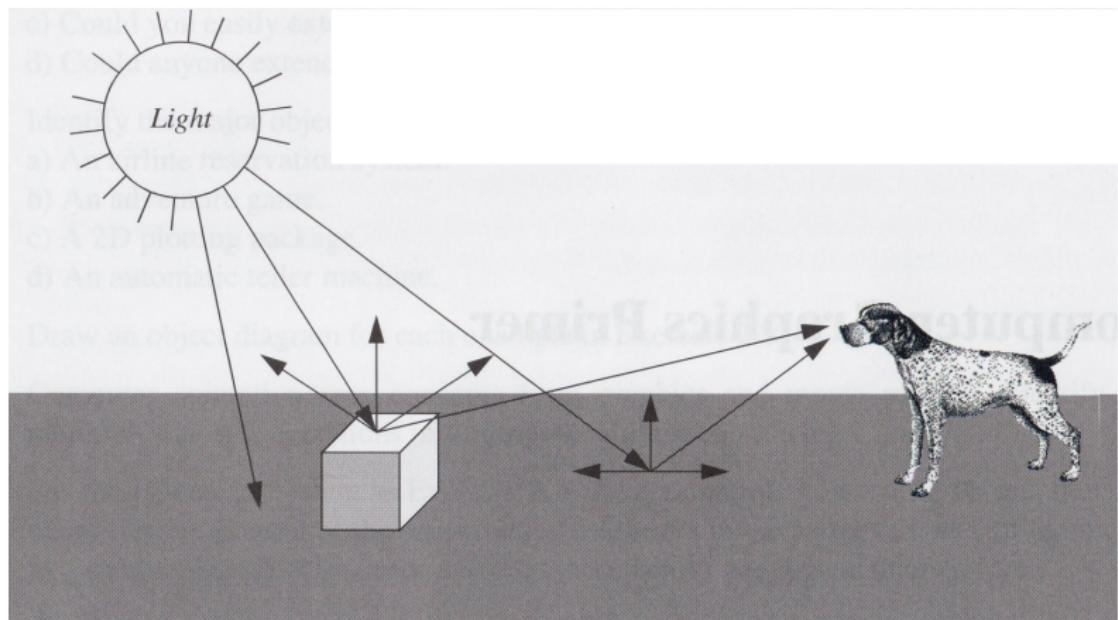
vtkElevationFilter *colorIt = vtkElevationFilter::New();
    colorIt->SetInput(sphere->GetOutput());
    colorIt->SetLowPoint(0,0,-1);
    colorIt->SetHighPoint(0,0,1);

vtkDataSetMapper *mapper = vtkDataSetMapper::New();
    mapper->SetInput(colorIt->GetOutput());

vtkActor *actor = vtkActor::New();
    actor->SetMapper(mapper);
```

Figure 4–13 A simple sphere example (ColorSph.cxx).

Comme dans un film



Les objets indispensables

- *vtkRenderWindow* : gère la (ou les) fenêtre(s) d'affichage.
- *vtkRenderer* : coordonne le rendu en utilisant les sources de lumières, les caméras et bien sûr les acteurs.
- *vtkLight* : définit une source de lumière illuminant la scène.
- *vtkCamera* : définit une source de lumière illuminant la scène.
- *vtkActor* : définit un acteur de la scène.
- *vtkProperty* : définit les propriétés d'apparences d'un acteur : couleur, transparence, comportement par rapport aux lumières.
- *vtkMapper* : définit la représentation géométrique d'un ou de plusieurs acteurs.

Les objets indispensables

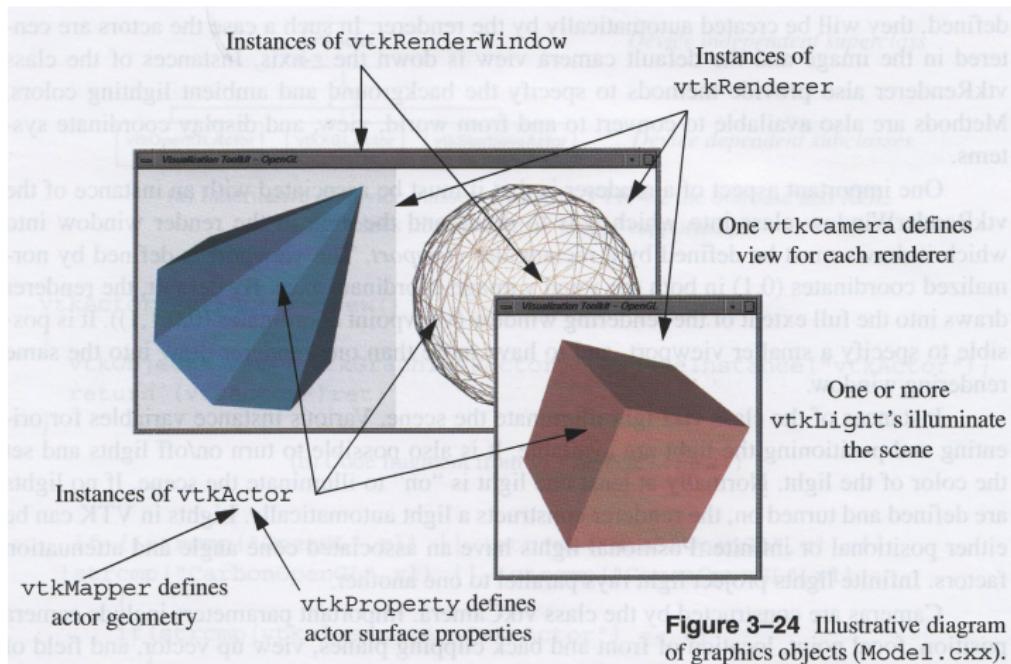


Figure 3–24 Illustrative diagram of graphics objects (`Model.cxx`).

vtkRenderWindow

C'est un objet abstrait spécifiant le comportement de la fenêtre d'affichage, ses principales caractéristiques sont :

- sa taille.
- sa résolution (bits par pixel).
- sa position.
- son nom.

⇒ Exemple *vtkRenderWindow.py*.

vtkRenderer

Prend en compte les lumières, caméras et acteurs afin de faire le rendu et de produire une image.

- Au moins un acteur doit être défini (lumières et caméras seront créées automatiquement si elles ne sont pas définies explicitement).
- Permet également de spécifier le fond de l'image et la lumière d'ambiance.
- Nécessite évidemment une fenêtre d'affichage.

⇒ Exemple *vtkRenderer.py*.

Positionne et oriente la caméra.

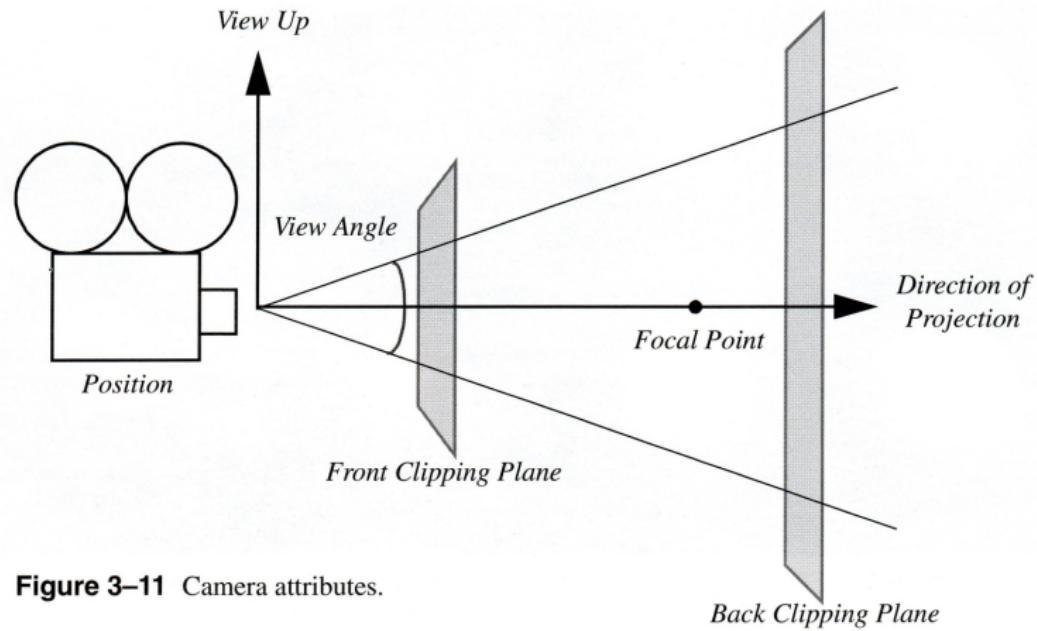


Figure 3–11 Camera attributes.

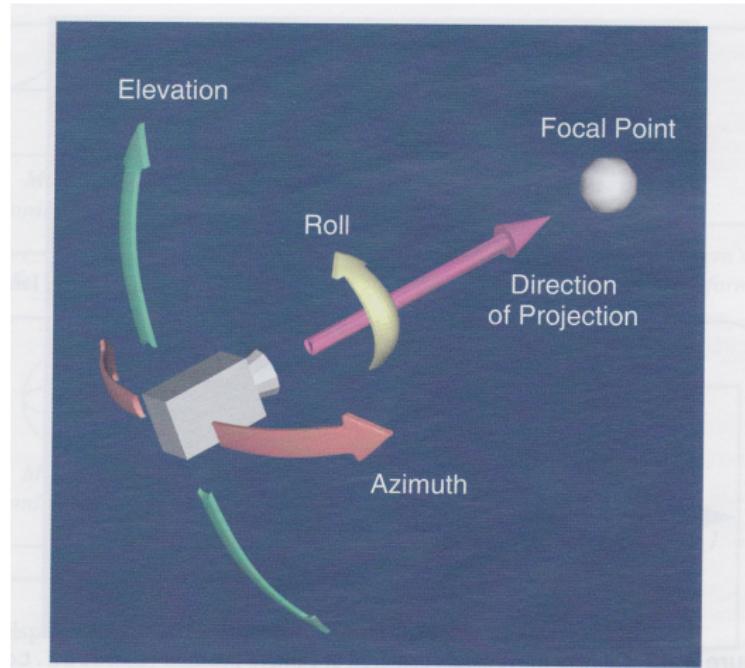


Figure 3–12 Camera movements around focal point (`camera.tcl`).

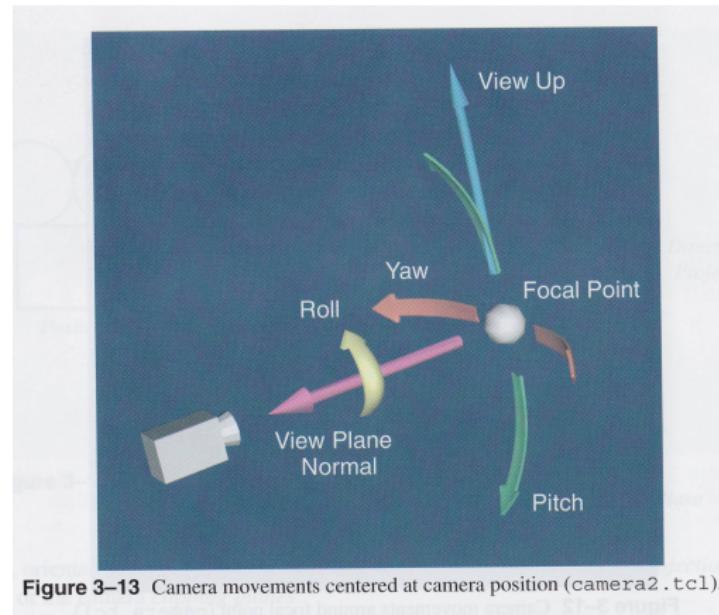


Figure 3–13 Camera movements centered at camera position (`camera2.tcl`).

⇒ Exemple *vtkCamera.py*.

vtkLight

Nécessaire pour éclairer la scène, par défaut il y a une lumière d'ambiance. Il existe deux types de lumière : une lumière infinie (tous les rayons sont parallèles) et un spot de lumière (i.e. un cône de lumière). Les principaux réglages sont :

- l'orientation.
- la position.
- la possibilité d'allumer et d'éteindre.
- la couleur de la lumière.
- l'angle et le coefficient d'atténuation (pour un spot de lumière).

⇒ Exemple *vtkLight.py*.

vtkRenderWindowInteractor

Permet d'interagir avec la figure en contrôlant la caméra et les acteurs. Les principales touches permettant d'interagir :

<p>e : ferme la fenêtre</p> <p>s : mode "surface"</p>	<p>r : revient à la position initiale</p> <p>w : mode "wireframe" (pour voir le maillage)</p>
---	---

On bouge la caméra (mode caméra, touche c) ou les acteurs (mode acteurs, touche a).

Il y a aussi deux modes pour utiliser la souris : touche j pour le mode "joystick" (mouvement continu durant aussi longtemps que l'on appuie sur le bouton) et t pour le mode "trackball" (mouvement uniquement quand on appuie sur le bouton et que le pointeur bouge).

Remarque : on peut définir son propre interacteur.
⇒ Exemple *vtkRenderWindowInteractor.py*.

vtkLODActor au lieu de *vtkActor*

Ajuste dynamiquement le niveau de détails de l'image (Level Of Details). Usage basique : *vtkActor* devient *vtkLODActor* !

Initialement, une méthode très simple, basée sur

Temps_Rendu/Nb_Acteurs, était utilisée pour déterminer quel niveau de détails doit être utilisé.

Il y a par défaut trois niveaux de détails :

- le niveau maximal.
- le niveau minimal : une simple boîte délimitant l'acteur (*vtkOutlineFilter*).
- le niveau intermédiaire : on fixe un nombre de points (positionnés aléatoirement) maximal. (*vtkMaskPoints*).

Il est possible de définir des niveaux de détails supplémentaires avec la méthode *AddLODMapper*.

⇒ Exemple *vtkLODActor.py*.

vtkAVIWriter ou vtkMPEG2Writer

Permet de faire des animations aux formats *avi* ou *mpeg2*.

- On spécifie le nom du fichier.
- La qualité de l'image.
- *vtkWindowToImageFilter* doit être utilisé afin de récupérer le contenu de la fenêtre d'affichage sous forme d'une image. Cette image sera ajoutée ensuite au film via la méthode *Write* de *vtkAVIWriter* ou de *vtkMPEG2Writer*.
- Attention : *vtkWindow* ne se comporte pas tout à fait comme les autres morceaux du pipeline. Dans une boucle, il faut forcer la mise à jour de *vtkWindowToImageFilter* via la méthode *Modified*.

⇒ Exemple *movie.cpp*.

Derniers exemples...

- \Rightarrow *particules.cpp* : particules affichées en direct i.e. au fur et à mesure que le calcul.
- \Rightarrow *ImagePlaneWidget.py* : boutons et "widget".
- \Rightarrow *medical1.py* et *medical2.py* : visualisation d'une image 3D, *LOD* et *vtkVolumeRayCastMapper*.
- \Rightarrow *ParallelVTK* : VTK et le parallélisme.

Deuxième partie



Motivations

Paraview est une application destinée à permettre la visualisation de gros volumes de données.

Principales caractéristiques :

- Logiciel libre multi-plateformes.
- Fonctionne sur un ou plusieurs processeurs.
- Construit sur *VTK* (*Visualization ToolKit*, écrit en *C++*).
VTK utilise *OpenGL* (librairie graphique 3D) pour le rendu des images.
- Interface utilisateur : *Tcl*, *Qt*.
- *http://www.paraview.org*

Compilation sous Unix

Ce qui est nécessaire au bon fonctionnement de *Paraview* :

- *OpenGL* (lié à la carte graphique)
- *Qt* (Depuis ParaView 3, peut être long à compiler...)
- *Tcl* (inclus dans l'archive de *Paraview*)
- *VTK* (inclus dans l'archive de *Paraview*)
- *MPI* (facultatif)
- Des librairies *X11* récentes...

Compilation sous Unix

Procédure d'installation :

- Téléchargement de l'archive :
`http://www.paraview.org/files/`
- [user /tmp]\$ tar xvfz paraview.tar.gz
- [user /home/user/paraview]\$ ccmake /tmp/paraview
Taper [t] pour avoir un menu très complet.
Lancer la configuration [c], vérifier les options
(*BUILD_SHARED_LIBS=on*), relancer éventuellement la
configuration puis générer les fichiers *Makefile* avec [g].
- [user /home/user/paraview]\$ make
- [root /home/user/paraview]\$ make install

Lancement de *Paraview*

Sur un seul processeur :

- [user]\$ paraview

Sur plusieurs processeurs dans le cas où *Paraview* a été compilé
avec *MPI* :

- [user]\$ mpirun -np 4 /usr/local/bin/paraview

Formats compatibles

Paraview (.pvda)
EnSight (.case .sos)
BYU (.g)
Plot3D (.xyz)
HDF5 (.h5)
VRML2 (.wrl)
Protein DB (.pdb)
STL (.stl)
AVS UCD (.inp)
Facet (.facet)
Gaussian Cube (.cube)
POP Ocean (.pop)

VTK (.pvtk .pvtpl .pvti .pvt...)
Exodus (.g .e .ex2 .ex2v2 .exo .gen ...)
XDMF (.xmf .xdmf)
SpyPlot CTH (.spcth)
DEM (.dem)
PLY (.ply)
XMol (.xyz)
Raw (.raw)
Meta Image (.mhd .mha)
PNG (.png)
SAF

Filtres et pipelines

Sous *Paraview* les données sont manipulées par l'intermédiaire de filtres. La liste des filtres disponibles dépend du type des données. On parle de “pipeline” de visualisation :

Données initiales → *Filtre 1* → *Filtre 2* → ...

Après chaque filtre les données changent de type et on peut ainsi avoir accès à plus ou moins de filtres.

Visualisation : présentation de VTK et ParaView

Sylvain Faure

Plan de
l'exposé

Introduction

VTK

Introduction
Les formats de
fichiers VTK
Extraction de
primitives
Pipeline de
visualisation
Objets VTK de
base
Applications

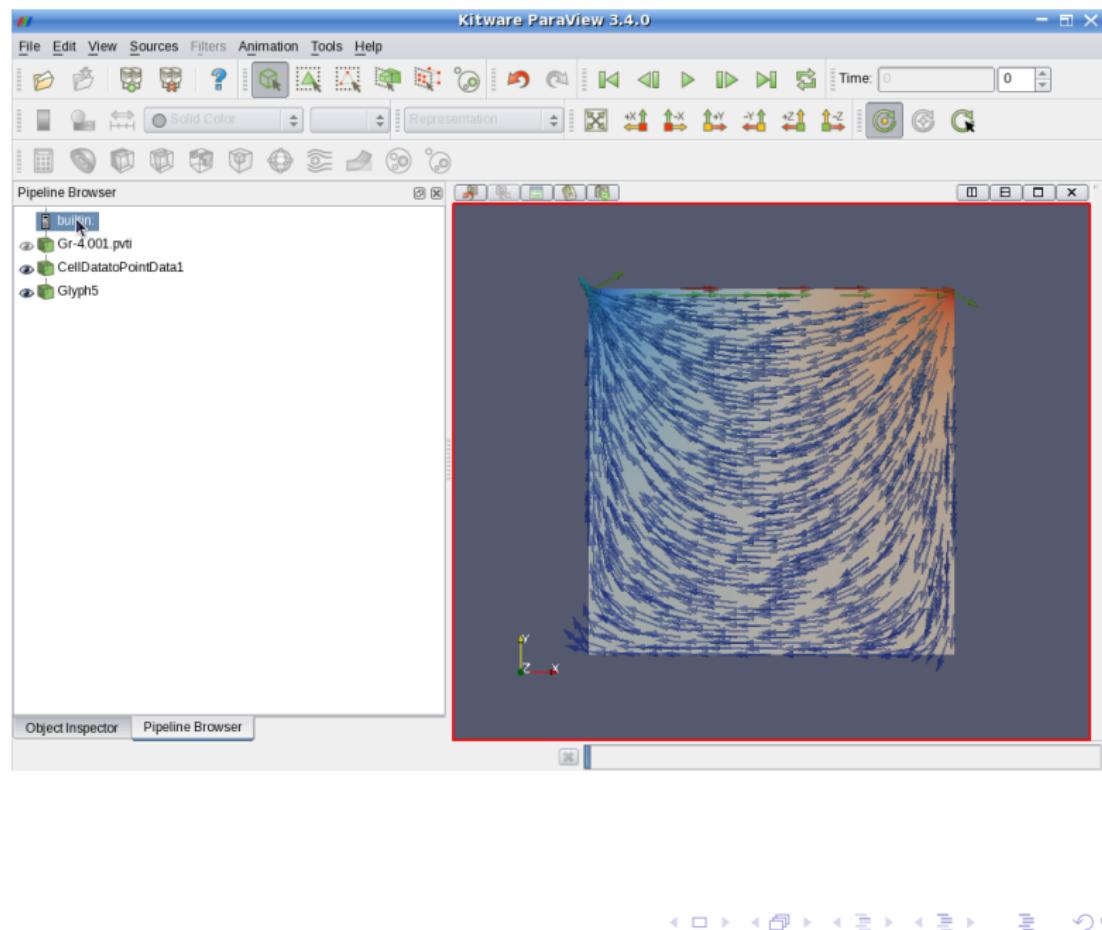
Paraview

Introduction
Installation et
lancement
Chargement des
données

Manipulation des données

Démonstrations

Références



Visualisation : présentation de VTK et ParaView

Sylvain Faure

Plan de l'exposé

Introduction

VTK

Introduction
Les formats de fichiers VTK
Extraction de primitives
Pipeline de visualisation
Objets VTK de base
Applications

Paraview

Introduction
Installation et lancement
Chargement des données

Manipulation des données

Démonstrations

Références

Annotate Time	Generate Surface Normals	Resample with dataset
Append Attributes	Glyph	Ribbon
Append Datasets	Glyph (Custom Source)	Rotational Extrusion
Append Geometry	Gradient	Scatter Plot
Block Scalars	Gradient (Unstructured)	Shrink
Calculator	Group Datasets	Slice
Cell Centers	Histogram	Smooth
Cell Data to Point Data	Integrate Variables	Stream Tracer
Clean	Intersect CTH Fragments	Stream Tracer (Custom Source)
Clean to Grid	Level Scalars	Subdivide
Clip	Linear Extrusion	Surface Flow
Compute Derivatives	Loop Subdivision	Surface Vectors
Connectivity	Mask Points	Temporal Cache
Contour	Median	Temporal Interpolator
Curvature	Merge Blocks	Temporal Shift Scale
D3	Mesh Quality	Temporal Snap-to-Time-Step
Decimate	Normal Glyphs	Temporal Statistics
Delaunay 2D	Octree Depth Limit	Tessellate
Delaunay 3D	Octree Depth Scalars	Tetrahedralize
Elevation	Outline	Texture Map to Cylinder
Extract Block	Outline (curvilinear)	Texture Map to Plane
Extract CTH Fragments	Outline Corners	Texture Map to Sphere
Extract CTH Parts	ParticleTracer	Threshold
Extract Cells by Region	Plot Global Variables over Time	Transform
Extract Datasets	Plot Over Line	Triangle Strips
Extract Edges	Plot Selection Over Time	Triangulate
Extract Level	Point Data to Cell Data	Tube
Extract Selection	Probe Location	Warp (scalar)
Extract Subset	Process Id Scalars	Warp (vector)
Extract Surface	Quadric Clustering	
Feature Edges	Random Vectors	
Generate Ids	Reflect	



Rapide description de quelques filtres

- *Append Attributes* : Regroupe plusieurs attributs de données définis sur la même géométrie (voir aussi *Group Parts*).
- *Append Datasets* et *Append Geometry* : Regroupe des attributs de données communs définis sur des géométries différentes (voir aussi *Group Parts*).
- *Calculator* : Agit comme une calculatrice scientifique sur les données : +, -, /, *, cos,...
- *Cell Centers* : Crée un point (centre de gravité) au centre de chaque cellule.
- *Cell Data to Point Data* : Calcule les valeurs aux points à partir des valeurs aux cellules.
- *Clean* : Dans le cas de données polygonales, efface les points non utilisés, gère les doublons, modifie les cellules dégénérées.

Rapide description de quelques filtres

- *Clean to grid* : Fusionne les points qui sont proches. Peut être utilisé pour convertir des données sur une grille non structurée.
- *Clip* : Fait une coupe à l'aide d'un plan, d'une boîte, d'une sphère ou d'un seuil. Ne réduit pas la dimension de l'ensemble de données. Renvoie toujours des données sur une grille non structurée.
- *Connectivity* : Marque les régions connectées.
- *Contour* : Calcule les isolignes ou les isosurfaces reliées à un scalaire.
- *Curvature* : Pour des données polygonales, calcule la courbure à chaque point.
- *Cut* : Extrait un plan ou une sphère d'une figure.
- *D3* : Divise les données en régions contigües afin de les répartir sur plusieurs processeurs. Utile quand *Paraview* fonctionne en parallèle.

Rapide description de quelques filtres

- *Decimate* : Réduit le nombre de triangles dans un ensemble de données polygonales.
- *Delaunay2D* : Crée une triangulation 2D de Delaunay à partir des points entrés.
- *Elevation* : Génère aux points des scalaires déterminés par la projection orthogonale de ces points sur une ligne.
- *Extract CTH Parts* : Pour visualiser des données provenant d'une simulation CTH.
- *Extract Datasets* : Extrait un ensemble de données parmi plusieurs.
- *Extract Edges* : Extrait les côtés de cellules 2D et 3D. Renvoie des lignes. Utile par exemple pour visualiser le maillage.
- *Extract Surface* : Extrait la surface 2D représentant la frontière.

Rapide description de quelques filtres

- *Feature Edges* : Extrait des sous-ensembles de côtés des données.
- *Glyph* : Permet de représenter un champ de vecteurs.
- *Gradient* : Calcule le gradient à chaque point d'une image.
- *Group Datasets* : Permet de regrouper des données.
- *Integrate Variables* : Intègre les valeurs aux cellules ou aux points.
- *Level Scalars* : Utilise des couleurs pour montrer les différents niveaux d'un ensemble de données hiérarchisé.
- *Linear Extrusion* : Extrusion d'une surface dans une direction donnée.
- *Loop Subdivision* : Divise itérativement chaque triangle en 4 triangles.

Rapide description de quelques filtres

- *Mask Points* : Réduit le nombre de points. Peut éventuellement être utilisé avant de tracer un champ de vecteurs.
- *Median* : Remplace pour chaque pixel une valeur scalaire par la valeur moyenne obtenue avec un nombre spécifié de voisins (peut réduire le bruit).
- *Mesh Quality* : Calcule la qualité de cellules (triangulaires, quadrilatérales et trétrahédriques) d'un maillage.
- *Outline* : Dessine le contour de la boîte dans laquelle sont représentées les données.
- *Outline (curvilinear)* : Dessine plus précisément (qu'avec *Outline*) le contour des données.
- *Outline Corners* : Dessine les angles de la boîte dans laquelle sont représentées les données.

Rapide description de quelques filtres

- *Point Data to Cell Data* : Calcule les valeurs aux cellules à partir des valeurs aux points.
- *Probe Location* : Renvoie les valeurs le long d'une ligne (avec la possibilité de tracer la courbe obtenue) ou à un point.
- *Process Id Scalars* : Attribut un scalaire à chaque *piece* des données. Utile avec les formats *.pvti*,... *Paraview* doit être lancé en parallèle.
- *Quadric Clustering* : Produit une approximation polygonale avec une résolution réduite (c'est ce filtre qui est par *Paraview* utilisé pour le niveau de détails LOD).
- *Random Vectors* : Génère aléatoirement un nouveau vecteur 3D aux points.

Rapide description de quelques filtres

- *Reflect* : Réflexion planaire des données.
- *Ribbon* : Génère des rubans à partir de lignes. Cela peut être utile pour voir les lignes de courant.
- *Rotational Extrusion* : Fait tourner une surface autour de l'axe des Z.
- *Shrink* : Rétrécit les cellules. Elles deviennent distinctes.
- *Smooth* : Ajuste la position des points pour des données polygonales.
- *Stream Tracer* : Trace les lignes de courant d'un champ vectoriel.
- *Subdivide* : Divise itérativement chaque triangle en 4 petits triangles.
- *Surface Flow* : Intègre un écoulement à travers une surface.

Rapide description de quelques filtres

- *Surface Vectors* : Contraint les vecteurs à se trouver sur une surface.
- *Tessellate* : Approche une forme géométrique d'ordre élevé par des formes géométriques plus simples (tétraèdres en 3D, triangles en 2D, segments en 1D).
- *Tetrahedralize* : Convertit des cellules 3D de n'importe quelles sortes en tétraèdres. Convertit des cellules 2D de n'importe quelles sortes en triangles.
- *Threshold* : Extrait la zone où une valeur scalaire est comprise entre 2 valeurs.
- *Transform* : Applique une transformation sur des polygones (translation, rotation,...)
- *Triangle Strips* : Regroupe les triangles en triangles “collés bout à bout” i.e. partageant toujours 2 sommets. Regroupe les lignes en lignes polygonales.

Rapide description de quelques filtres

- *Triangulate* : Décompose des polygones en triangles, lignes et points.
- *Tube* : Remplace des lignes par des tubes afin d'offrir une meilleure lisibilité (des lignes de courant par exemple).
- *Warp (scalar)* : Translate tous les points suivants un même vecteur.
- *Warp (vector)* : Translate chaque point suivant un vecteur qui lui est propre.

Niveaux de détails (LOD)

On veut utiliser *Paraview* afin de manipuler de gros volumes de données. Il est donc important de pouvoir adapter l'outil de visualisation à la puissance de l'ordinateur utilisé. Pour cela on dispose de quelques paramètres permettant d'arriver à un compromis entre les performances d'affichage (vitesse de rotation d'une image 3D,...) et la qualité des images (résolution).

Visualisation :
présentation
de VTK et
ParaView

Sylvain Faure

Plan de
l'exposé

Introduction

VTK

Introduction
Les formats de
fichiers VTK
Extraction de
primitives
Pipeline de
visualisation
Objets VTK de
base
Applications

Paraview

Introduction
Installation et
lancement
Chargement des
données
Manipulation
des données

Démonstrations

Références

Démonstrations et travaux pratiques !

Concernant Paraview

- *Paraview Guide* :
<http://www.kitware.com/products/paraviewguide.html>
- <http://www.paraview.org>
- Mailing list
<http://www.paraview.org/HTML/MailingList.html>
- Wiki (site web dynamique) :
<http://www.paraview.org/Wiki/ParaView>
- FAQ : <http://www.paraview.org/Wiki/ParaViewFAQ>

Concernant VTK

- *Visualization Toolkit Book et VTK User's Guide :*
http://www.vtk.org/buy-books.php
- *http://www.vtk.org*
- *Mailing list :*
http://public.kitware.com/mailman/listinfo/vtkusers
- *Wiki : http://www.vtk.org/Wiki/VTK*
- *FAQ : http://www.vtk.org/Wiki/VTK_FAQ*
- *Documents techniques (formats des fichiers de
données,...) : http://www.vtk.org/documents.php*