

Introduction à VTK

par Julian Ibarz ([Mon site perso](#)) ([Blog](#))

Date de publication : Lundi 28 juillet 2008

Dernière mise à jour : Vendredi 15 août 2008

Cet article a pour but de vous faire une introduction sur la bibliothèque de visualisation de données scientifiques VTK. Il vous présentera mon point de vue sur cette dernière, ses avantages et inconvénients ainsi que ses performances.

I - Qu'est-ce que VTK ?.....	3
II - Installation de la bibliothèque.....	3
II-A - Prérequis.....	3
II-B - Création des fichiers de compilation.....	4
II-C - Compilation.....	4
II-D - Linkage.....	4
II-E - Programme minimaliste.....	5
III - Architecture de VTK.....	5
IV - Remerciements.....	7
V - Références.....	7

I - Qu'est-ce que VTK ?

VTK est une bibliothèque portable de visualisation de données scientifiques écrite en C++ et orientée objet. Elle permet de faire des traitements sur ces données en créant très simplement une chaîne d'algorithmes pour produire au final une image 2D/3D. Elle est utilisée par de nombreuses universités et entreprises.

VTK a de nombreux bindings et peut être ainsi utilisée dans les langages Java, Python et Tcl. De plus, elle peut être intégrée très facilement dans de nombreuses bibliothèques de GUI : Qt, Swing, Tk, FLTK.

Son grand avantage est son large spectre d'applications (du traitement de l'image à la visualisation de données volumiques). De plus, un logiciel de visualisation nommé [ParaView](#) permet de faire de la visualisation de données au format interne VTK et ceci sur de simples machines ou des clusters graphiques et même du post processing en appliquant directement certains algorithmes de VTK.

Ceci est donc un grand avantage par rapport à ses concurrents car il permet aux programmeurs d'une entreprise de se concentrer uniquement sur la conversion de ses données vers le format VTK et de laisser les traitements de post processing aux non-informaticiens, j'entends par là les mathématiciens, physiciens, etc.

VTK a aussi un système de picking et d'interaction avec les objets 3D très très avancé et utilisant divers algorithmes (sélection par frustum, pick color, etc.). Vous pouvez sélectionner des objets entiers ou des primitives graphiques (triangles, points, lignes, etc.).

Son principal défaut est qu'il n'a pas de système de graphe de scène. Ne comptez pas utiliser cette bibliothèque si vous avez besoin de ce type de structure de données, orientez-vous plutôt vers [OpenSceneGraph](#) ou en java vers [jmonkey](#). De plus les performances de VTK s'effondrent s'il contient plus de 1000 objets 3D (j'entends par là des *vtkActor*). Si votre application dépasse ce nombre je vous déconseille son utilisation. Ceci est dû à l'architecture de VTK qui donne une très grande généricité d'utilisation mais détruit les performances pour un très grand nombre d'objets. Néanmoins avec peu d'acteurs les performances de VTK sont très correctes (utilisant des **display list** donc l'affichage d'objets de plusieurs millions de triangles est possible). Encore une fois, je vous conseille de tester ParaView pour voir ce que VTK a dans le ventre au niveau performance.

Si vous voulez des exemples d'utilisation de VTK je vous conseille d'aller voir les [références](#). Pensez que tous les exemples qui sont donnés font en moyenne 100 lignes de code. Vous pouvez donc faire de la visualisation très avancée avec très peu d'effort grâce à VTK.

II - Installation de la bibliothèque

II-A - Prérequis

Pour compiler et installer VTK il va vous falloir plusieurs bibliothèques :

- CMake pour créer les fichiers de compilation.
- GCC pour Linux, Visual C++ ou Code Blocks sous Windows pour compiler.
- La bibliothèque OpenGL.

Procurez-vous les sources de VTK accessibles sur le site officiel. Il est à noter que ce cours se base sur la version 5.2 de VTK actuellement disponible qu'en version CVS. Je vous conseille donc fortement de vous procurer cette version sur le CVS (tag 5.2).

II-B - Création des fichiers de compilation

Pour les linuxiens allez dans le dossier racine des sources et tapez :

```
ccmake .
```


Ce que vous faites n'est ni plus ni moins que d'exécuter ccmake dans le dossier courant.

Pour ceux qui sont sur Windows lancez le GUI ccmake. Il vous demande où sont les sources, indiquez où vous avez les sources de VTK et dites-lui de construire les binaires au même endroit.

Dans les deux cas vous aurez une option Show Advanced Values. Activez cette option. Lancez la configuration. ccmake va se renseigner sur votre système (quel compilateur, etc.). Sous windows vous aurez à choisir votre compilateur. Choisissez Visual Studio 9 2008 ou CodeBlocks Mingw Makefile si vous avez cette version. Attention pour la version mingw, vous devrez peut-être rajouter le dossier des binaires de mingw dans le PATH sans quoi CMake ne voudra pas s'exécuter. Une fois que la configuration préliminaire est effectuée vous allez devoir configurer VTK lui-même. Si les options `VTK_USE_GEOVIS` et `VTK_USE_INFOVIS` sont activées, désactivez-les (à moins que vous vouliez utiliser ces modules...). Si vous voulez utiliser VTK en Java activez le wrapping Java `VTK_WRAP_JAVA` et sous linux vous devrez indiquer les include du JDK Java. Sur ma machine, les répertoires à indiquer sont :

```
//Path to a file.  
JAVA_INCLUDE_PATH:PATH=/usr/lib/jvm/java-6-sun/include  
  
//Path to a file.  
JAVA_INCLUDE_PATH2:PATH=/usr/lib/jvm/java-6-sun/include/linux
```

Lancez la configuration. Une fois ceci effectué sous Windows appuyez sur OK pour lancer la génération des fichiers de compilation. Sous linux appuyez sur la touche g pour faire la génération.

 Si ccmake vous demande de rentrer des informations manquantes vous avez deux possibilités : rentrer ces valeurs dans le fichier `CMakeCache.txt` ou alors supprimer le cache et recommencer la saisie de la configuration depuis le début.

II-C - Compilation

Sous linux il vous suffira de faire un simple make. Ne faites pas de make install car cette option n'est pas très bien gérée.

Sous Windows vous n'avez qu'à ouvrir le fichier solution (`vtk.sln`) pour Visual et lancer la compilation ou le projet Code Blocks (`vtk.cbp`). Attention, si vous utilisez la version 2.6 de CMake, une erreur surviendra lors de la compilation avec mingw. Pour résoudre ce problème, éditez le fichier `vtkConfigure.h` qui se trouve à la racine du répertoire de build et supprimez la ligne :

```
#define VTK_TYPE_USE___INT64
```

II-D - Linkage

Pour lier la bibliothèque vous devrez indiquer à votre compilateur le dossier include et le dossier bin pour les fichiers de linkage et aussi rajouter le dossier bin dans le répertoire PATH de Windows et dans le `LD_LIBRARY_PATH` de Linux.

II-E - Programme minimaliste

Voici un programme en java (vous avez la version C++ sur le site officiel) qui vous permettra de tester l'installation de la bibliothèque :

```
import vtk.*;

public class test {

    // Constructeur statique qui charge la bibliothèque VTK.
    // Les bibliothèques doivent être dans le path de votre système.
    static {
        System.loadLibrary("vtkCommonJava");
        System.loadLibrary("vtkFilteringJava");
        System.loadLibrary("vtkIOJava");
        System.loadLibrary("vtkImagingJava");
        System.loadLibrary("vtkGraphicsJava");
        System.loadLibrary("vtkRenderingJava");
    }

    // Fonction principale
    public static void main (String[] args)
    {
        // Créer une géométrie sphérique
        vtkSphereSource sphere = new vtkSphereSource();
        sphere.SetRadius(1.0);
        sphere.SetThetaResolution(18);
        sphere.SetPhiResolution(18);

        // Transforme la géométrie en primitives graphiques (OpenGL dans notre cas)
        vtkPolyDataMapper map = new vtkPolyDataMapper();
        map.SetInput(sphere.GetOutput());

        // L'acteur représente l'entité géométrique.
        // Il permet de définir sa position, son orientation, sa couleur, etc.
        vtkActor aSphere = new vtkActor();
        aSphere.SetMapper(map);
        aSphere.GetProperty().SetColor(0,0,1); // color blue

        // Nous créons un renderer qui va faire le rendu de notre entité.
        vtkRenderer ren1 = new vtkRenderer();
        ren1.AddActor(aSphere);
        ren1.SetBackground(1,1,1); // background color white

        // Nous créons une fenêtre de rendu
        vtkRenderWindow renWin = new vtkRenderWindow();
        renWin.AddRenderer(ren1);
        renWin.SetSize(300,300);

        // Nous créons un interactor qui permet de bouger la caméra.
        vtkRenderWindowInteractor iren = new vtkRenderWindowInteractor();
        iren.SetRenderWindow(renWin);

        // Nous lançons le rendu et l'interaction
        renWin.Render();
        iren.Start();
    }
}
```

Vous devriez voir une fenêtre apparaître avec une sphère.

III - Architecture de VTK

Lors de la présentation, nous avons insisté sur le large spectre d'application de VTK (affichage 2D, 3D et volumique). Nous nous concentrerons ici sur la partie 3D de VTK.

VTK divise en 3 grandes parties la construction et l'affichage d'un objet 3D.


Tout d'abord nous avons les données elles-mêmes qui sont créées avec un objet **source** ou lues dans un fichier avec un objet **reader** ou alors directement créées par programmation. Dans notre exemple nous utilisons un objet de type `vtkSphereSource` :

```
// Créer une géométrie sphérique
vtkSphereSource sphere = new vtkSphereSource();
sphere.SetRadius(1.0);
sphere.SetThetaResolution(18);
sphere.SetPhiResolution(18);
```

Vient ensuite la génération des primitives OpenGL qui est faite à l'aide d'un objet nommé **mapper**. Il va transformer la structure de données en primitives OpenGL (`GL_TRIANGLE`, etc.) :

```
// Transforme la géométrie en primitives graphiques (OpenGL dans notre cas)
vtkPolyDataMapper map = new vtkPolyDataMapper();
map.SetInput(sphere.GetOutput());
```

Il existe plusieurs types de mappers adaptés à chaque structure de données fournie par VTK. Dans notre cas, comme nous avons une structure de données `vtkPolyData` (la sortie donnée par `GetOutput` du `vtkSphereSource` est un `vtkPolyData`), nous utiliserons le mapper de type `vtkPolyDataMapper`. Il faut savoir qu'il existe plusieurs types de données disponibles dans VTK mais le type le plus usité pour l'affichage 3D est le `vtkPolyData`.

 *Dans ce code vous voyez le mécanisme de VTK pour créer l'association entre les différents algorithmes : le mapper reçoit en entrée (`SetInput`) la sortie de l'objet source (`GetOutput`). Vous pouvez faire cela plusieurs fois d'affilée et ainsi créer un graphe complexe (certains algorithmes reçoivent plusieurs entrées ou donnent plusieurs sorties). Vous n'êtes donc limité que par votre imagination et tout ce qu'il faut savoir en utilisant VTK c'est quelles sont les associations d'algorithmes qui vont vous permettre d'obtenir le résultat escompté.*

La dernière partie est l'association du mapper à l'objet 3D VTK lui-même qui est un `vtkActor` :

```
// L'acteur représente l'entité géométrique.
// Il permet de définir sa position, son orientation, sa couleur, etc.
vtkActor aSphere = new vtkActor();
aSphere.SetMapper(map);
aSphere.GetProperty().SetColor(0,0,1); // color blue
```

`vtkActor` nous sert entre autre à positionner et orienter l'objet, à le colorer, choisir des propriétés graphiques (avec son membre `vtkProperty` obtenu avec la méthode `GetProperty`).

Enfin il y a la partie sur le rendu lui-même qui consiste en la création d'un objet de type `vtkRenderer` qui va s'occuper de lancer les algorithmes : exécution du `vtkSphereSource` qui va engendrer un `vtkPolyData` contenant une sphère qui va être envoyé au `vtkPolyDataMapper` qui va créer des primitives OpenGL qui seront ensuite affichées dans la fenêtre associée au `vtkRenderer`. Nous lui ajoutons notre objet 3D pour qu'il s'occupe de son rendu :

```
// Nous créons un renderer qui va faire le rendu de notre entité.
vtkRenderer ren1 = new vtkRenderer();
ren1.AddActor(aSphere);
ren1.SetBackground(1,1,1); // background color white
```

Nous créons ensuite la fenêtre qui affichera le rendu :

```
// Nous créons une fenêtre de rendu
vtkRenderWindow renWin = new vtkRenderWindow();
renWin.AddRenderer(ren1);
renWin.SetSize(300, 300);
```

Et pour que l'utilisateur puisse bouger la caméra à l'aide de la souris nous créons un objet nommé **interactor** de type *vtkRenderWindowInteractor* et lançons l'affichage de la fenêtre et l'interaction avec l'utilisateur (la boucle événementielle de VTK) :

```
// Nous créons un interactor qui permet de bouger la caméra.
vtkRenderWindowInteractor iren = new vtkRenderWindowInteractor();
iren.SetRenderWindow(renWin);

// Nous lançons le rendu et l'interaction
renWin.Render();
iren.Start();
```






L'introduction s'achève ici. J'espère vous avoir donné l'eau à la bouche pour utiliser cette bibliothèque pas comme les autres.

IV - Remerciements

Merci à **diogene** pour sa relecture et à toute l'équipe 2D/3D/Jeux.

V - Références

Voici différents liens qui pourraient vous être utiles :

-  [Site officiel de VTK.](#)
-  [Site officiel du logiciel Paraview.](#)
-  [Exemples d'utilisations de VTK.](#)
-  [Site officiel du scene graph java jmonkey.](#)
-  [Site officiel de OpenSceneGraph.](#)