

VTk 简明教程

csuzhangxc

Published
with GitBook



Table of Contents

- 1. 前言
- 2. 开始使用 VTK
 - i. 编译与安装
 - ii. 第一个 VTK 程序
 - iii. QVTKWidget 集成
- 3. VTK 基础
 - i. VTK 可视化流程
 - ii. 主要 class
 - iii. 可视化管线
 - iv. 智能指针

VTK 简明教程

快速上手 VTK。

教程中示例程序完整代码见 `_examples` 文件夹，直接使用时请注意修改 `.pro` 文件中的 `INCLUDEPATH` 、 `LIBS` 。

GitHub 维护地址：<https://github.com/csuzhangxc/vtk-simple-tutorial>

目录

1. 开始使用 VTK
 - 1.1 编译与安装
 - 1.2 第一个 VTK 程序
 - 1.3 QVTKWidget 集成
2. VTK 基础
 - 2.1 VTK 可视化流程
 - 2.2 主要 class
 - 2.3 可视化管线
 - 2.4 智能指针

开始使用 VTK

本章内容

1. 编译与安装
2. 第一个 VTK 程序
3. QVTKWidget 集成

编译与安装

Qt5（MinGW）使用 CMake 编译与使用 VTK6

本文使用的各软件版本

- Qt : 5.4.0
- MinGW : 4.9.1
- CMake : 3.1.1
- VTK : 6.1.0

各软件下载地址

- Qt with MinGW : <http://www.qt.io/download-open-source/>
- CMake : <http://www.cmake.org/download/>
- VTK : <http://vtk.org/VTK/resources/software.html>

编译VTK

安装 Qt（安装过程中注意勾选安装MinGW）与 CMake，并添加与 Qt 集成安装的 MinGW 可执行程序路径（Qt***/Tools/mingw***/bin，即 mingw32-make.exe 所在路径）到系统环境变量 *PATH*。解压 VTK，建议解压后路径不要包含中文与空格。

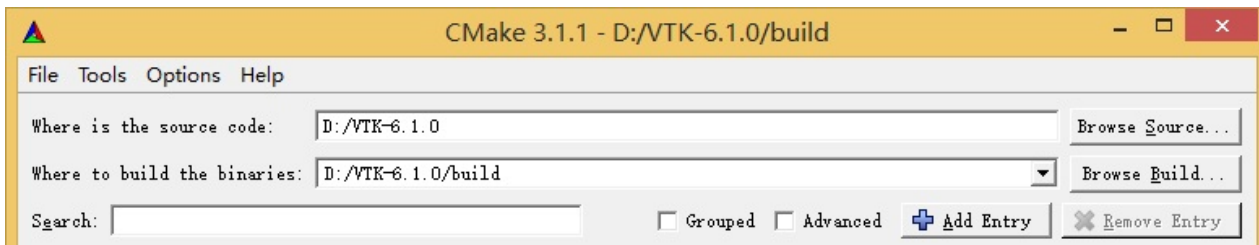
打开 VTK 目录下的 **CMakeLists.txt** 文件，在约第219行，将 `set(VTK_USE_WIN32_THREADS 1)` 修改为 `set(VTK_USE_PTHREADS 1)`，该处原始上下文为：

```
include(FindThreads)
set(VTK_USE_WIN32_THREADS 0)
set(VTK_USE_PTHREADS 0)
set(VTK_HP_PTHREADS 0)
set(VTK_USE_SPROC 0)
if(CMAKE_USE_WIN32_THREADS_INIT)
    set(VTK_USE_WIN32_THREADS 1)
    set(CMAKE_THREAD_LIBS_INIT "")
elseif(CMAKE_USE_PTHREADS_INIT)
    set(VTK_USE_PTHREADS 1)
    if(CMAKE_HP_PTHREADS_INIT)
        set(VTK_HP_PTHREADS 1)
    endif()
elseif(CMAKE_USE_SPROC_INIT)
    set(VTK_USE_SPROC 1)
endif()
```

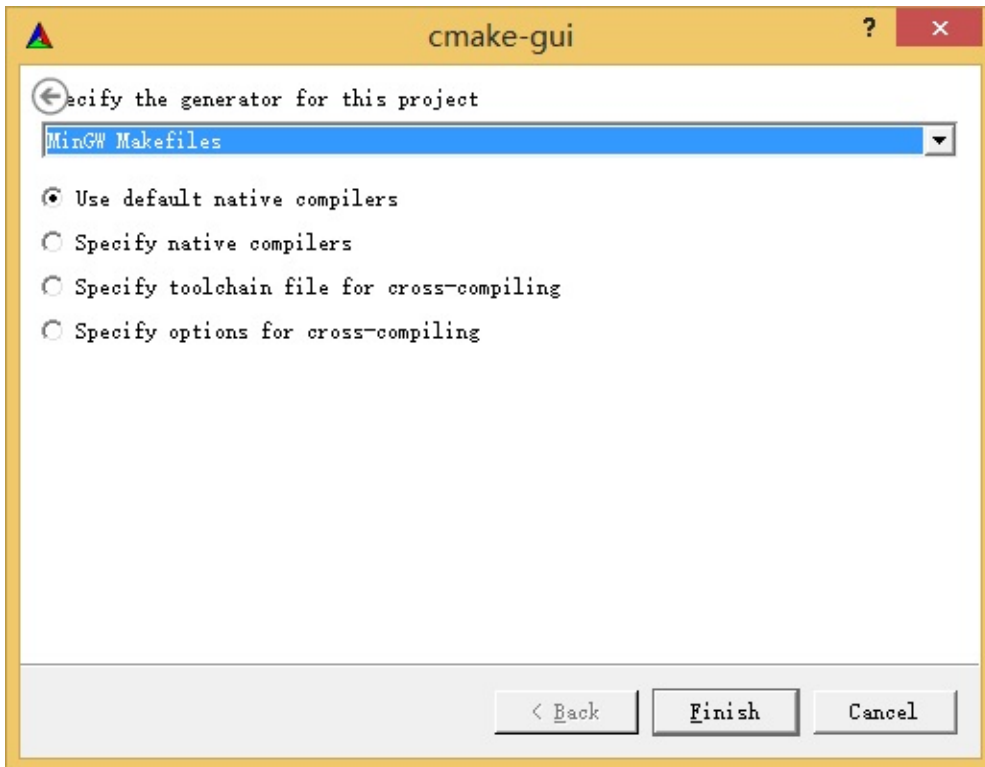
打开 VTK目录/ThirdParty/libxml2/vtklibxml2/threads.c，在文件开头 `#include "libxml.h"` 后添加行：`#undef HAVE_WIN32_THREADS`，修改后上下文为：

```
#define IN_LIBXML
#include "libxml.h"
#undef HAVE_WIN32_THREADS
#include <string.h>
```

启动 CMake，并指定源代码路径和生成路径：



配置生成器为 MinGW：



点击 CMake 中的 **Configure** 进行第一次配置，配置完成后，勾选 **Search** 框后的 **Grouped** 与 **Advanced**。

在 **CMAKE** 下设置 **CMAKE_INSTALL_PREFIX** 为编译完成后 VTK 期望的安装路径，如 **D:/VTK-6.1.0/MinGW**。

在 **Module** 下勾选 **Module_vtkGUISupportQt**、**Module_vtkGUISupportQtOpenGL**、**Module_vtkGUISupportQtSQL**、**Module_vtkGUISupportQtWebkit**、**Module_vtkRenderingQt**、**Module_vtkViewsQt**。

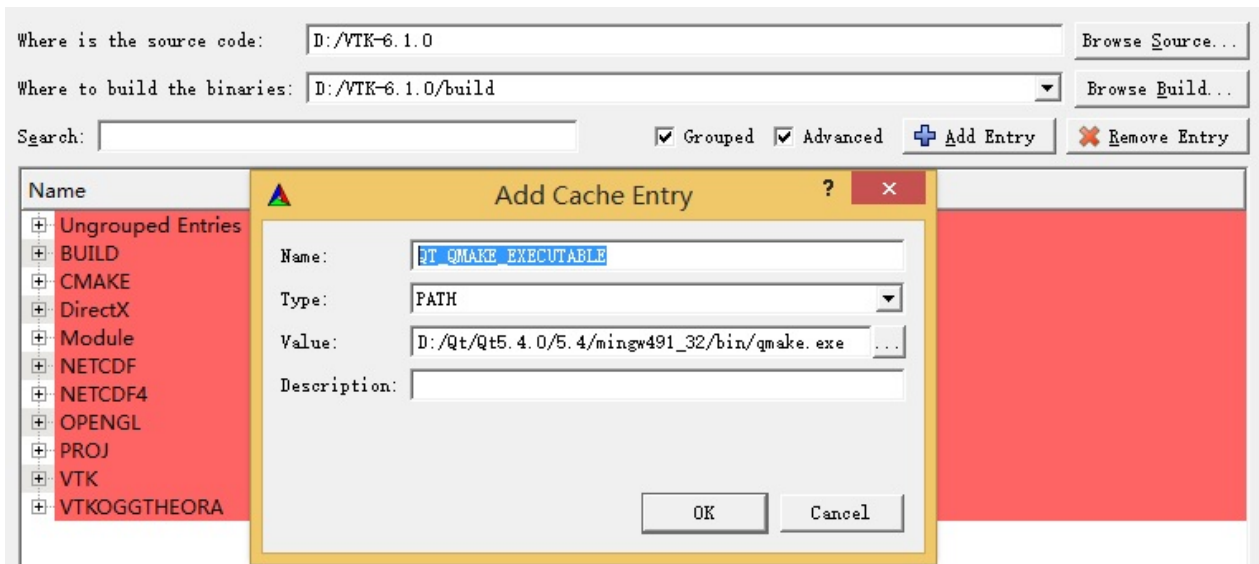
在 **VTK** 下勾选 **VTK_Group_Qt**。

如果需要编译为静态链接库，在 **BUILD** 下取消勾选 **BUILD_SHARED_LIBS**（如编译为静态链接库，使用时可能会遇到 lib 文件循环依赖问题）。

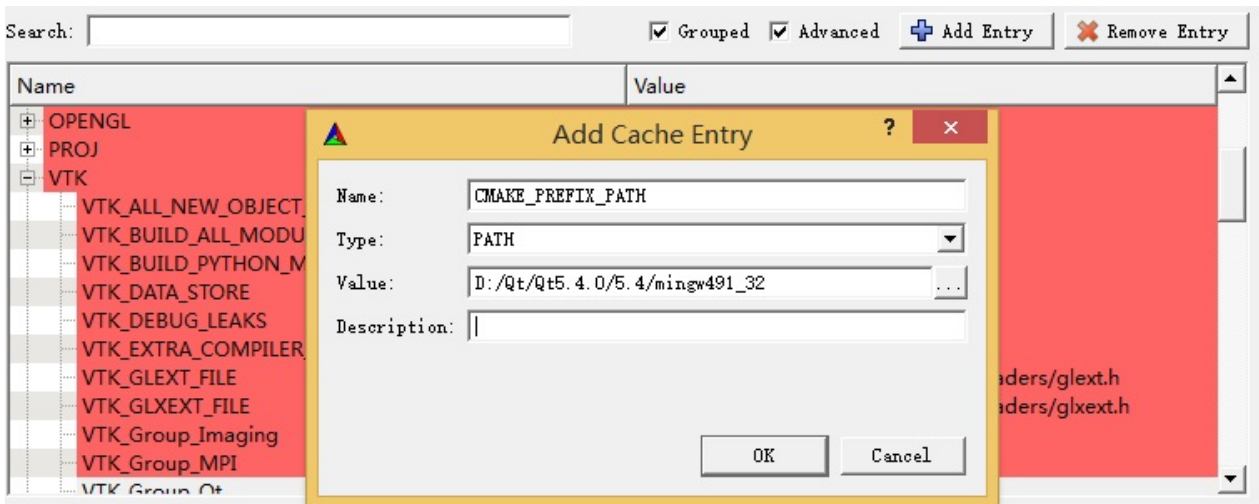
如果需要编译为 Release，在 **CMAKE** 下修改 **CMAKE_BUILD_TYPE** 为 **Release**。

如果需要在 Debug 模式下编译生成的库文件带 **d** 后缀，点击 **Add Entry**，手动添加后缀项，**Name** 为 **CMAKE_DEBUG_POSTFIX**，**Type** 为 **STRING**，**Value** 为 **d**。

点击 **Add Entry**，手动添加 qmake 所在路径，**Name** 为 **QT_QMAKE_EXECUTABLE**，**Type** 为 **PATH**，**Value** 为 **qmake.exe** 所在完整路径：

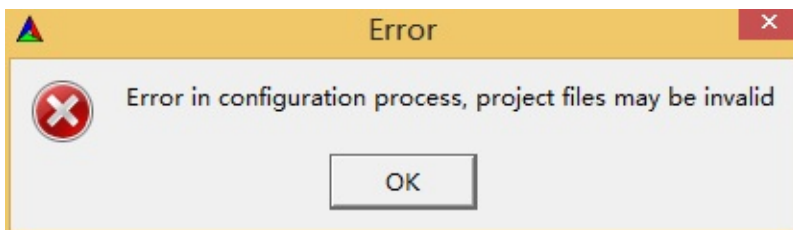


点击 **Add Entry**，手动添加 Qt 安装目录，Name 为 `CMAKE_PREFIX_PATH`，Type 为 `PATH`，Value 为 Qt 安装目录（应为包含 `qmake.exe` 的 `bin` 的父目录，可对照上下两图）：



再次点击 **Configure**。

此时将弹出错误：

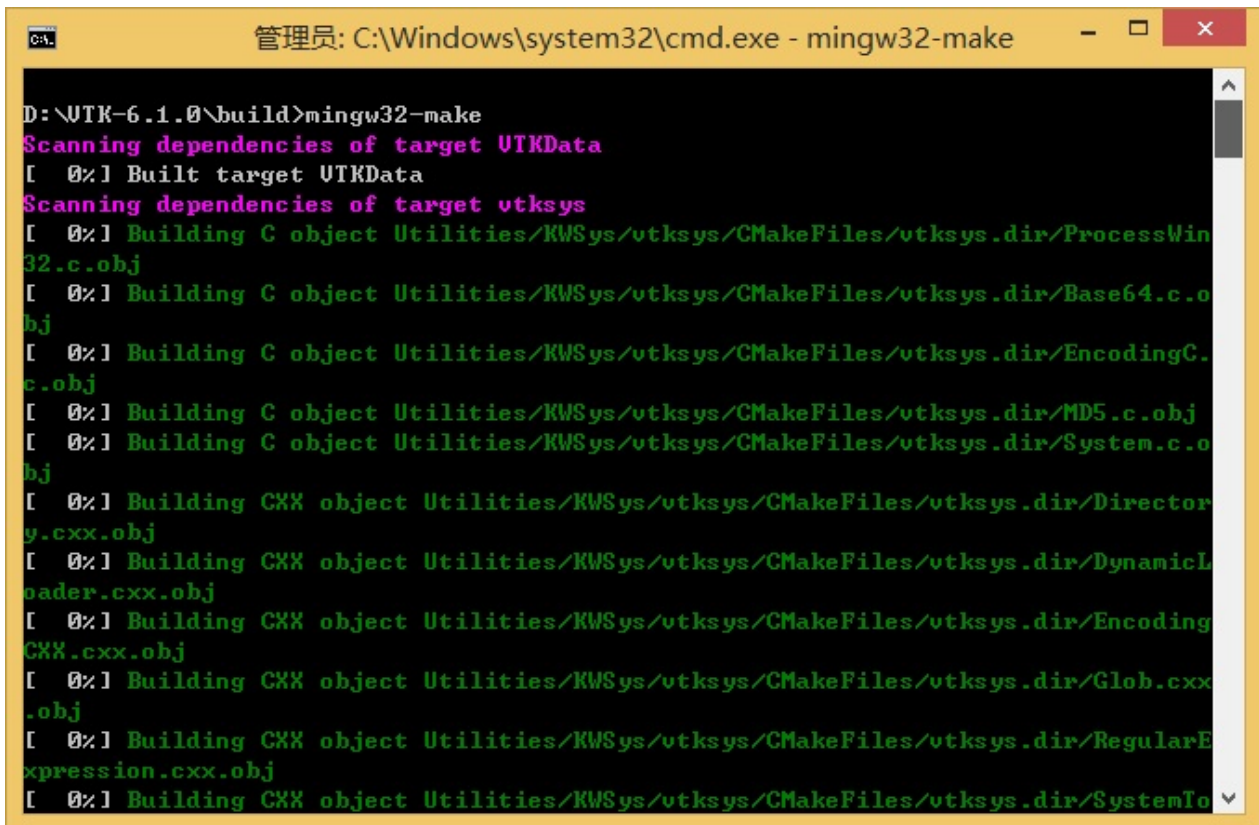


在 *Ungrouped Entries* 下将 `QT_VTK_VERSION` 修改为 5，再次点击 **Configure**。

此时在界面下方的消息输出窗口中，将输出 *Configuring done*，点击 **Generate**。

生成完成后，将输出 *Generating done*。

在最开始在CMake中指定的生成路径中打开命令行窗口，输入 `mingw32-make`，开始编译生成：



```
D:\VTK-6.1.0\build>mingw32-make
Scanning dependencies of target VTKData
[ 0%] Built target VTKData
Scanning dependencies of target vtksys
[ 0%] Building C object Utilities/KWSys/vtksys/CMakeFiles/vtksys.dir/ProcessWin
32.c.obj
[ 0%] Building C object Utilities/KWSys/vtksys/CMakeFiles/vtksys.dir/Base64.c.o
bj
[ 0%] Building C object Utilities/KWSys/vtksys/CMakeFiles/vtksys.dir/EncodingC.
c.obj
[ 0%] Building C object Utilities/KWSys/vtksys/CMakeFiles/vtksys.dir/MD5.c.obj
[ 0%] Building C object Utilities/KWSys/vtksys/CMakeFiles/vtksys.dir/System.c.o
bj
[ 0%] Building CXX object Utilities/KWSys/vtksys/CMakeFiles/vtksys.dir/Director
y.cxx.obj
[ 0%] Building CXX object Utilities/KWSys/vtksys/CMakeFiles/vtksys.dir/DynamicL
oader.cxx.obj
[ 0%] Building CXX object Utilities/KWSys/vtksys/CMakeFiles/vtksys.dir/Encoding
CXX.cxx.obj
[ 0%] Building CXX object Utilities/KWSys/vtksys/CMakeFiles/vtksys.dir/Glob.cxx
.obj
[ 0%] Building CXX object Utilities/KWSys/vtksys/CMakeFiles/vtksys.dir/RegularE
xpression.cxx.obj
[ 0%] Building CXX object Utilities/KWSys/vtksys/CMakeFiles/vtksys.dir/SystemTo
```

编译完成后，执行 `mingw32-make install`，将编译生成文件输出到已配置的 `CMAKE_INSTALL_PREFIX` 目录中，此时该目录中将有 `bin`、`include`、`lib`、`plugins`、`share` 文件夹。

如果在添加了 `d` 后缀进行编译后执行 `mingw32-make install` 时，提示 `libQVTKWidgetPlugin.dll` 文件复制错误，可手动修改生成目录 `/GUISupport/Qt/PluginInstall.cmake` 文件，将其中的 `libQVTKWidgetPlugin.dll` 修改为 `libQVTKWidgetPlugin.dll`。

参考

1. http://www.vtk.org/Wiki/VTK/Configure_and_Build
2. <http://vtk.1045678.n5.nabble.com/vtk-users-VTK-6-0-Compile-issue-with-MinGW64-on-Windows-7-x64-td5724152.html#a5726939>

第一个 VTK 程序

在 Qt 中使用 VTK

ImageViewer 程序

说明：本示例程序修改自 **VTK-6.1.0/Examples/GUI/Qt/ImageViewer**。

新建 **Qt Widgets** 程序，将 **.pro** 文件修改为类似如下：

```
/* .pro文件 */
QT      += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = ImageViewer
TEMPLATE = app

INCLUDEPATH += D:/VTK-6.1.0/MinGW/include/vtk-6.1

LIBS += -LD:/VTK-6.1.0/MinGW/lib/ \
        -lvtkGUISupportQt-6.1d \
        -lvtkIOImage-6.1d \
        -lvtkInteractionImage-6.1d \
        -lvtkRenderingCore-6.1d \
        -lvtkCommonExecutionModel-6.1d \
        -lvtkCommonCore-6.1d \
        -lvtkRenderingOpenGL-6.1d \
        -lvtkInteractionStyle-6.1d

SOURCES += main.cpp

HEADERS +=
```

其中 `INCLUDEPATH += D:/VTK-6.1.0/MinGW/include/vtk-6.1` 为添加 VTK 头文件所在路径；`LIBS += -LD:/VTK-6.1.0/MinGW/lib/` 为添加 VTK 库文件所在路径（对于 MinGW 编译为 `.a` 文件所在路径，对于 VS 编译为 `.lib` 文件所在路径）；`LIBS += -lvtk***-6.1d` 等为添加具体的库文件（请注意实际使用到的库文件版本及是否需要添加 `d` 后缀）。

将 **main.cpp** 修改为类似如下：

```
/* main.cpp */
#include <QApplication>

#include <vtkAutoInit.h>
VTK_MODULE_INIT(vtkRenderingOpenGL)
VTK_MODULE_INIT(vtkInteractionStyle)

#include "vtkImageViewer.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkRenderer.h"
#include "vtkPNGReader.h"
#include "QVTKWidget.h"
#include "vtkImageData.h"

int main(int argc, char** argv)
{
    QApplication app(argc, argv);

    QVTKWidget widget;

    char filename[] = "D:/test.png";
    vtkPNGReader* reader = vtkPNGReader::New();
    reader->SetFileName(filename);
    reader->Update();

    vtkImageViewer* imageView = vtkImageViewer::New();
    imageView->SetInputConnection(reader->GetOutputPort());
```

```

widget.SetRenderWindow(imageView->GetRenderWindow());
imageView->SetupInteractor(widget.GetRenderWindow()->GetInteractor());
imageView->SetColorLevel(138.5);
imageView->SetColorWindow(233);

int *dims = reader->GetOutput()->GetDimensions();
widget.resize(dims[0], dims[1]);
widget.show();

app.exec();

imageView->Delete();
reader->Delete();
return 0;
}

```

将其中的 `char filename[] = "D:/test.png"`；指定为需要通过使用 VTK 显示的图片的路径（如果不是 *png* 图片，需要修改对应的 *Reader* 对象）。

编译运行后（为能在 QtCreator 中直接启动程序，可能需要将 VTK 编译安装后的 *bin* 路径添加到系统环境变量 *PATH*），程序将在窗口中显示程序中指定文件名对应的图片（如上面代码中的 *D:/test.png*）。

注意

main.cpp 文件中如果不正确设置如下的模块初始化：

```

#include <vtkAutoInit.h>
VTK_MODULE_INIT(vtkRenderingOpenGL)
VTK_MODULE_INIT(vtkInteractionStyle)

```

可能会报 `Error: no override found for '*****'` 错误。

如果不添加 `VTK_MODULE_INIT(vtkInteractionStyle)`，可能会在运行时产生警告，并在调试状态关闭程序时发生崩溃（*segmentation fault*）。

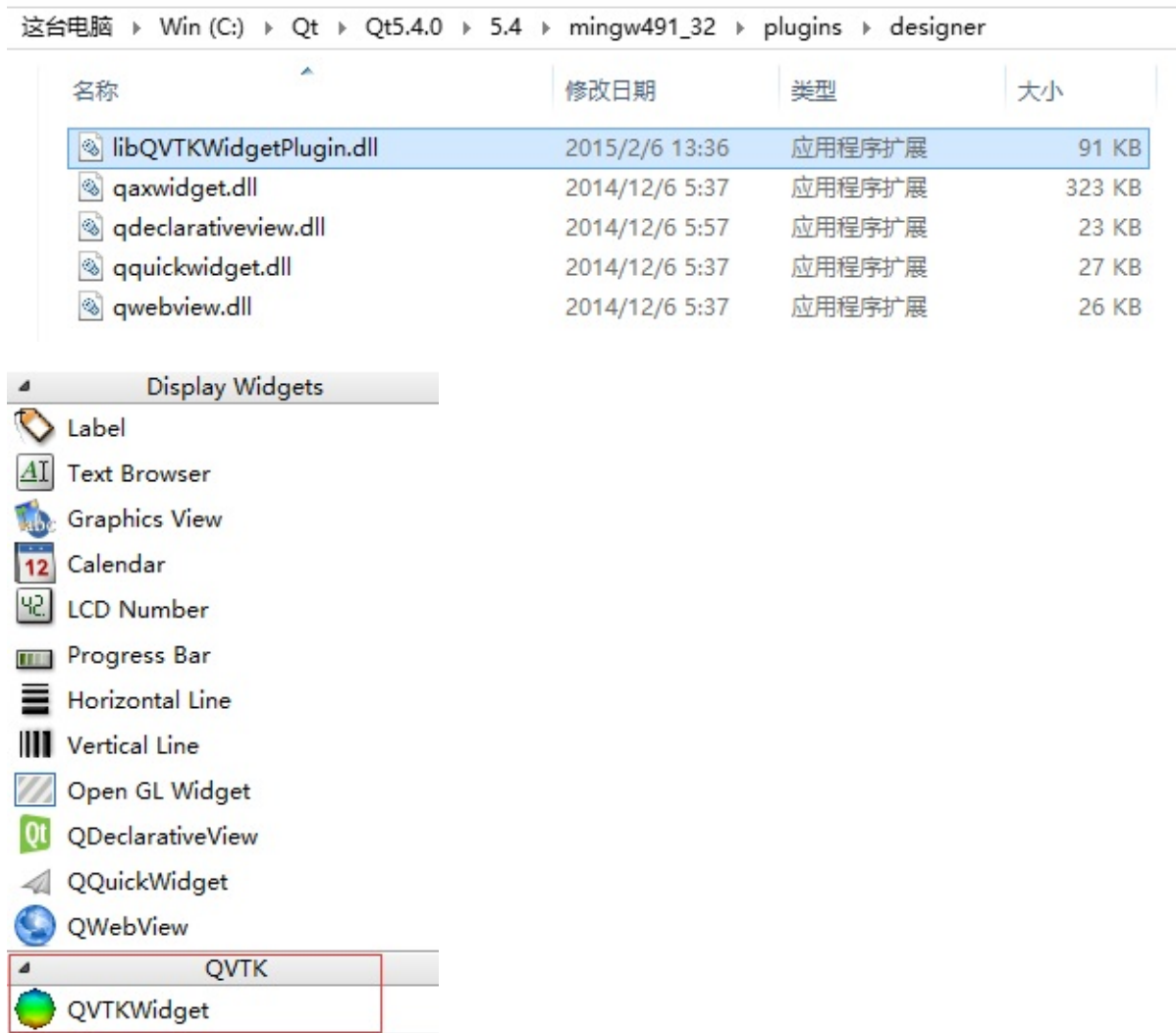
参考

1. http://www.vtk.org/Wiki/VTK/VTK_6_Migration/Factories_now_require_defines
2. <http://public.kitware.com/pipermail/vtkusers/2014-July/084396.html>

QVTKWidget 集成

在 Qt Designer 中集成 QVTKWidget

在 **Release** 模式下编译完成并执行 `mingw32-make install` 后，复制 `CMAKE_INSTALL_PREFIX/plugins/designer` 目录下的 `libQVTKWidgetPlugin.dll` 到 `Qt/Qt***/***/mingw***/plugins/designer` 目录（此目录应已有 `qaxwidget.dll` 等文件），打开 Qt Designer（非 QtCreator 内的 Designer，而是独立的 Qt Designer）后，即可发现增加了 QVTK 分类，并在其下有 QVTKWidget 控件。



在 QtCreator 中集成 QVTKWidget

如需要在 QtCreator 内集成的 Designer 中使用 QVTKWidget，应把 `libQVTKWidgetPlugin.dll` 复制到 `Qt/Qt***/Tools/QtCreator/bin/plugins/designer` 目录。

但由于编译生成 QtCreator 的编译器版本与生成 `libQVTKWidgetPlugin.dll` 的版本可能并不兼容，将导致无法正常加载插件。

如果无法直接使用 `libQVTKWidgetPlugin.dll` 插件，可在 Designer 中添加普通的 `QWidget` 控件后，手动将其提升为 `QVTKWidget`（对应的头文件为 `QVTKWidget.h`，注意在 `.pro` 中设置正确的 `INCLUDEPATH`）。

参考

1. <http://www.vtk.org/Wiki/VTK/Tutorials/QtSetup>

VTK 基础

本章内容

1. VTK 可视化流程
2. 主要 class
3. 可视化管线
4. 智能指针

VTK 可视化流程

VTK 两大重要模块

每个 VTK 程序，均包含两个最重要的模块：

- 可视化管线（Visualization Pipeline）
- 渲染引擎（Rendering Engine）

其中，可视化管线用于获取或创建数据、加工处理数据、把数据写入文件或者把数据传递给渲染引擎，渲染引擎负责数据的可视化表达。

Cylinder 程序

说明：本示例程序修改自 **VTK-6.1.0/Examples/Rendering/Cxx/Cylinder.cxx**。

```
#include <vtkAutoInit.h>
VTK_MODULE_INIT(vtkRenderingOpenGL)
VTK_MODULE_INIT(vtkInteractionStyle)

#include "vtkCylinderSource.h"
#include "vtkPolyDataMapper.h"
#include "vtkActor.h"
#include "vtkRenderer.h"
#include "vtkRenderWindow.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkProperty.h"
#include "vtkCamera.h"
#include "vtkInteractorStyleTrackballCamera.h"

int main()
{
    // 圆柱体数据模型
    vtkCylinderSource *cylinder = vtkCylinderSource::New();
    cylinder->SetResolution(8);

    // 将几何数据转换为可被渲染引擎绘制的可视化表达
    vtkPolyDataMapper *cylinderMapper = vtkPolyDataMapper::New();
    cylinderMapper->SetInputConnection(cylinder->GetOutputPort());

    // 需要被渲染绘制的对象
    vtkActor *cylinderActor = vtkActor::New();
    cylinderActor->SetMapper(cylinderMapper);
    cylinderActor->GetProperty()->SetColor(1.0000, 0.3882, 0.2784);
    cylinderActor->RotateX(30.0);
    cylinderActor->RotateY(-45.0);

    // 渲染器 渲染窗口 交互器
    vtkRenderer *renderer = vtkRenderer::New();
    vtkRenderWindow *renWin = vtkRenderWindow::New();
    renWin->AddRenderer(renderer);
    vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
    iren->SetRenderWindow(renWin);

    // 交互方式
    vtkInteractorStyleTrackballCamera *style = vtkInteractorStyleTrackballCamera::New();
    iren->SetInteractorStyle(style);

    // 将需要被渲染的对象添加到渲染器
    renderer->AddActor(cylinderActor);
    renderer->SetBackground(0.1, 0.2, 0.4);
    renWin->SetSize(200, 200);

    // 设置用于观察场景的相机
    renderer->ResetCamera();
    renderer->GetActiveCamera()->Zoom(1.5);
    renWin->Render();

    // 开始渲染 进入事件循环
```

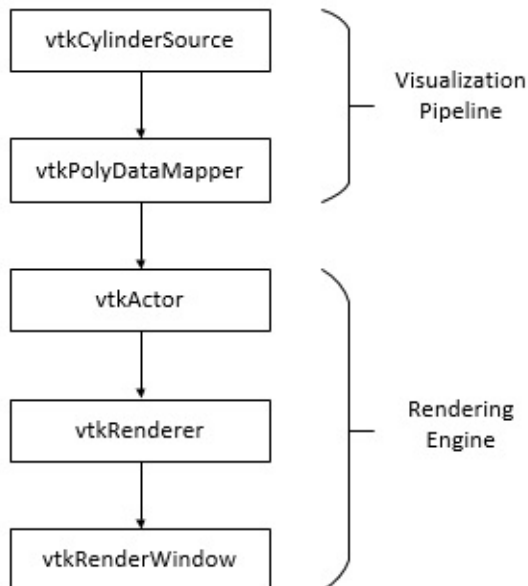
```

    iren->Start();

    // 清除实例
    cylinder->Delete();
    cylinderMapper->Delete();
    cylinderActor->Delete();
    renderer->Delete();
    renWin->Delete();
    iren->Delete();
    style->Delete();

    return 0;
}

```



Contour2D 程序

说明：本示例程序完成代码见 `_examples/Contour2D`。

```

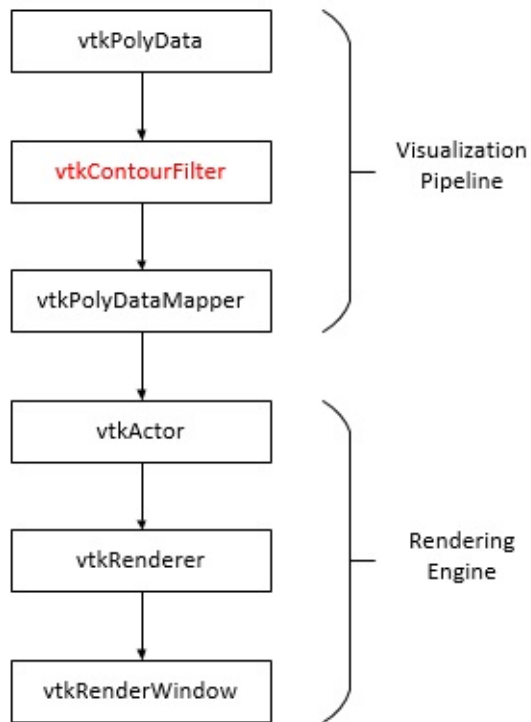
/* 二维等值线提取与可视化代码片段 */
// 等值线 Filter
vtkContourFilter *contourFilter = vtkContourFilter::New();
contourFilter->SetValue(0, ui->selectedValueLabel->text().toDouble());
contourFilter->SetInputData(polyData);

// 将几何数据转换为可被渲染引擎绘制的可视化表达
vtkPolyDataMapper *contourMapper = vtkPolyDataMapper::New();
contourMapper->SetInputConnection(contourFilter->GetOutputPort());
contourMapper->ScalarVisibilityOff();

// 需要被渲染绘制的对象
contourActor = vtkActor::New();
contourActor->SetMapper(contourMapper);
contourActor->GetProperty()->SetColor(1.0, 0.0, 0.0);
contourActor->GetProperty()->SetLineWidth(2.0);

// 添加到渲染器
renderer->AddActor(contourActor);

```



相对于 Cylinder 程序，Contour2D 主要区别在于其在增加了 `vtkContourFilter` 过滤器。

在进行数据可视化时，通过在可视化管线中连接一个或多个 Filter，可以执行相关可视化算法，实现对数据的加工处理并影响最终的可视化显示。

参考

1. http://blog.csdn.net/www_doling_net/article/details/8541436
2. <http://lzchenheng.blog.163.com/blog/static/8383353620108130751672/>

主要 class

vtkAlgorithm & vtkPolyDataAlgorithm

`vtkAlgorithm` 通过 `SetInputConnection` 、 `AddInputConnection` 、 `GetOutputPort` 进行管线的连接。

`vtkPolyDataAlgorithm` 继承自 `vtkAlgorithm` ，以多边形数据作为输出。Cylinder 程序中的 `vtkCylinderSource` 继承自 `vtkPolyDataAlgorithm` 。

`vtkPolyDataAlgorithm` 通过 `GetOutput` 可以直接获得 `vtkPolyData*` 。

VTK 中所有 sources、filters 均直接或间接继承自 `vtkAlgorithm` 。

vtkAbstractMapper & vtkPolyDataMapper

`vtkAbstractMapper` 相关子类用于将输入数据转换为几何图元（2D / 3D）。

```
vtkAbstractMapper <- vtkAbstractMapper3D <- vtkMapper <- vtkPolyDataMapper
```

`vtkAbstractMapper` 也继承自 `vtkAlgorithm` 。

vtkProp & vtkActor

`vtkProp` 用于渲染场景数据的可视化表示（2D / 3D）。

`vtkProp` 的子类负责确定场景中对象的位置、大小和方向信息。

Prop 依赖于两个对象，一个是 Mapper（`vtkMapper`）对象，负责存放数据和渲染信息；另一个是属性（`vtkProperty`）对象，负责控制颜色、不透明度等参数。

在 `vtkProp` 的子类中，`vtkActor` 用于表示场景中的几何数据（Geometry Data），`vtkVolume` 表示场景中的体数据（Volumetric Data），`vtkActor2D` 常用来表示二维空间中的数据。

```
vtkProp <- vtkProp3D <- vtkActor
```

`vtkProp` 也用于实现对象的拾取（picking）、拖动（dragging）。

vtkRenderer

组成场景的对象包括 Prop、Camera、Light 都被集中在一个 `vtkRenderer` 对象中。

`vtkRenderer` 负责管理场景的渲染过程。一个 `vtkRenderWindow` 中可以有多个 `vtkRenderer` 对象，这些 `vtkRenderer` 可以渲染在窗口中不同的矩形区域中（视口），甚至可以是覆盖的区域。

`vtkRenderer` 继承自 `vtkViewport` 。

vtkRendererWindow

`vtkRendererWindow` 将操作系统与 VTK 渲染引擎连接到一起。不同平台下的 `vtkRenderWindow` 子类负责本地计算机系统中窗

口创建和渲染过程管理。当使用 VTK 开发应用程序时，只需要使用平台无关的 `vtkRendererWindow` 类，运行时系统会自动替换为平台相关的 `vtkRenderWindow` 子类。

对于 `QVTKWidget`，可以通过 `GetRenderWindow` 获取到 `vtkRenderWindow*`。

vtkRenderWindowInteractor

`vtkRenderWindowInteractor` 提供平台独立的响应鼠标、键盘和时钟事件的交互机制，通过 VTK 的 Command / Observer 设计模式将监听的特定平台的鼠标、键盘和时钟事件交由 `vtkInteractorObserver` 或其子类，如 `vtkInteractorStyle` 进行处理。

`vtkInteractorStyle` 等监听这些消息并进行处理以完成旋转、拉伸和放缩等运动控制。

参考

1. <http://www.vtk.org/doc/nightly/html/classvtkAlgorithm.html>
2. <http://www.vtk.org/doc/nightly/html/classvtkPolyDataAlgorithm.html>
3. <http://www.vtk.org/doc/nightly/html/classvtkAbstractMapper.html>
4. <http://www.vtk.org/doc/nightly/html/classvtkProp.html>
5. <http://www.vtk.org/doc/nightly/html/classvtkRenderer.html>
6. <http://www.vtk.org/doc/nightly/html/classvtkRenderWindow.html>
7. <http://www.vtk.org/doc/nightly/html/classvtkRenderWindowInteractor.html>
8. http://blog.csdn.net/www_doling_net/article/details/8536376

可视化管线

可视化管线

VTK 可视化管线主要负责读取或者生成数据，分析或生成数据的衍生版本，写入硬盘文件或者传递数据到渲染引擎进行显示。

VTK 中采用数据流的方法将信息转换为几何数据，主要涉及到两种基本的对象类型：

- `vtkDataObject`
- `vtkAlgorithm`

`vtkDataObject`

`vtkDataObject` 可以看做一般的数据集合，有规则结构的数据称为一个 `Dataset`（`vtkDataSet`，继承自 `vtkDataObject`）。

`vtkDataSet` 中包括几何结构、拓扑结构及属性数据。属性数据既可以关联到点，也可以关联到单元上。

单元（cell）是点的拓扑组合，是构成 `Dataset` 结构的基本单位，常用来进行插值计算。

`vtkDataObject` 与 `vtkDataSet` 支持的具体数据对象可参考各自类文档。

`vtkAlgorithm`

`vtkAlgorithm` 包含各类 sources（源算法）、filters（过滤器）、mappers（映射器）。

Sources 通过读取（Reader 对象）或者创建数据对象（程序源对象）两种方式来产生数据。

Filters 处理输入数据并产生新的数据对象。

Mappers 接收数据并将其转换为可被渲染引擎绘制的可视化表达。

管线的连接与执行

可视化管线主要通过 `SetInputConnection`、`AddInputConnection`、`GetOutputPort`（均为 `vtkAlgorithm` 成员函数）进行连接。连接时，可以有多个 Input 或 Output。

管线连接的起点为 Sources，终点为 Mappers，中间可以有 0 个、1 个或多个 Filters。

管线连接时，必须确保参与连接的对象相互兼容。

管线只有当计算需要时才会执行（Lazy Evaluation，惰性计算）。

当管线中的 `DataSet` 或 `Algorithm` 发生改变时，管线需要重新执行。`vtkObject` 内通过成员变量 `MTime` 记录最后修改时间（Modification Time）。

参考

1. http://blog.csdn.net/www_doling_net/article/details/26690929
2. <http://www.vtk.org/doc/nightly/html/classvtkDataObject.html>
3. <http://www.vtk.org/doc/nightly/html/classvtkDataSet.html>

智能指针

对象创建与销毁

`vtkObject` 将构造函数声明为 `protected`，因此不能使用构造函数创建对象。

`vtkObject` 通过静态函数 `New()` 创建对象，通过函数 `Delete()` 销毁对象。

引用计数

在引用计数中，每一个对象负责维护对象所有引用的计数值。

使用引用计数，可以实现数据之间的共享而不用拷贝，同时也可以实现简单的垃圾回收。

智能指针

智能指针会自动管理引用计数的增加与减少，如果检测到某对象的引用计数值减少为 0，则会自动地释放该对象的资源，从而达到自动管理内存的目的。

在 VTK 中，智能指针通过模板类 `vtkSmartPointer` 实现，使用前需 `#include <vtkSmartPointer.h>`。

```
/* 使用智能指针管理对象 无需要调用 Delete */
vtkSmartPointer<vtkObject> MyObject = vtkSmartPointer<vtkObject>::New();
```

```
/* 将已创建对象交给智能指针进行管理 */
vtkPolyData *pd = vtkPolyData::New();
// ...

// 方法 1
vtkSmartPointer<vtkPolyData> MyObject;
MyObject.TakeReference(pd);           // TakeReference 为成员函数

// 方法 2
vtkSmartPointer<vtkPolyData> MyObject
    = vtkSmartPointer<vtkPolyData>::Take(pd); // Take 为静态成员函数
```

```
/* 使用智能指针获取对象 */
vtkSmartPointer<vtkXMLPolyDataReader> Reader = vtkSmartPointer<vtkXMLPolyDataReader>::New();

// 不使用智能指针 Reader 超出作用域后数据将被释放
vtkPolyData* pd = Reader->GetOutput();

// 使用智能指针 Reader 与 pd 均超出作用域后数据才被释放
vtkSmartPointer<vtkPolyData> pd = Reader->GetOutput();
```

使用 `GetPointer` 可以从智能指针中获取普通指针对象。

智能指针与普通指针均可以传递给接收智能指针或普通指针为参数的函数。

智能指针函数返回值必须传递给智能指针对象。

智能指针成员变量在析构时自动释放其拥有的对象，无需调用 `Delete`。

不要修改智能指针指向新的对象。

```
/* 使用示例 */
#include <vtkFloatArray.h>
#include <vtkSmartPointer.h>

void WithSmartPointers();
void WithoutSmartPointers();

int main(int argc, char *argv[])
{
    WithSmartPointers();
    WithoutSmartPointers();

    return 0;
}

void WithSmartPointers()
{
    vtkSmartPointer<vtkFloatArray> Distances = vtkSmartPointer<vtkFloatArray>::New();
}

void WithoutSmartPointers()
{
    vtkFloatArray* Distances = vtkFloatArray::New();
    Distances->Delete();
}
```

参考

1. <http://www.vtk.org/Wiki/VTK/Tutorials/SmartPointers>
2. <http://www.vtk.org/doc/nightly/html/classvtkSmartPointer.html>
3. http://blog.csdn.net/www_doling_net/article/details/8540242