版本：1.0

VTK调研入门

成都邦飞科技有限公司

20年 11月

VTK调研入门

编 写：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

审 核：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

会 签：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

批 准：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 内容 | 修改人 | 日期 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目 录

[1 什么是VTK 1](#_Toc61451696)

[1.1 了解VTK 1](#_Toc61451697)

[1.2 VTK的特点 1](#_Toc61451698)

[2 安装VTK 2](#_Toc61451699)

[2.1 生成对应的VS版本的工程文件 2](#_Toc61451700)

[2.2 进入工程目录，打开VTK.sln文件 3](#_Toc61451701)

[2.3 创建一个测试工程 4](#_Toc61451702)

[3 VTK的结构 9](#_Toc61451703)

[3.1 图形模型 9](#_Toc61451704)

[3.2 VTK绘制管线 10](#_Toc61451705)

[4 编码须知 11](#_Toc61451706)

[4.1 低级对象模型 11](#_Toc61451707)

[4.1.1 引用计数 11](#_Toc61451708)

[4.1.2 运行时类型信息 11](#_Toc61451709)

[4.1.3 对象状态显示 11](#_Toc61451710)

[4.2 渲染引擎 12](#_Toc61451711)

[4.2.1 vtkProp(道具类) 12](#_Toc61451712)

[4.2.2 vtkAbstractMapper（映射器类） 13](#_Toc61451713)

[4.2.3 vtkProperty and vtkVolumeProperty.（属性类） 13](#_Toc61451714)

[4.2.4 vtkCamera（相机类） 13](#_Toc61451715)

[4.2.5 vtkLight（灯光类） 13](#_Toc61451716)

[4.2.6 vtkRenderer(渲染器类) 14](#_Toc61451717)

[4.2.7 vtkRenderWindow（渲染窗口类） 14](#_Toc61451718)

[4.2.8 vtkRenderWindowInteractor（渲染窗口交互器类） 14](#_Toc61451719)

[4.2.9 vtkTransform（转换类） 14](#_Toc61451720)

[4.2.10 vtkLookupTable, vtkColorTransferFunction, and vtkPiecewiseFunction 14](#_Toc61451721)

[5 Qt VTK应用 15](#_Toc61451722)

[5.1 实现一个JPG图像浏览器 15](#_Toc61451723)

[5.1.1 vtkOutputWindow.h 16](#_Toc61451724)

[5.1.2 vtkOutputWindow.cpp 16](#_Toc61451725)

[5.1.3 main.cpp 19](#_Toc61451726)

[5.1.4 QtWidgetsVTKTest.ui 19](#_Toc61451727)

[5.2 模型文件的导入和导出 21](#_Toc61451728)

[5.2.1 导入 21](#_Toc61451729)

[5.2.2 导出 22](#_Toc61451730)

[6 引用 22](#_Toc61451731)

# 什么是VTK

## 了解VTK

VTK，全称是Visualization Toolkit，即可视化工具包。是一个开源、跨平台、可自由获取、支持并行处理的图形应用函数库。

VTK最早是作为Prentice Hall在1993年出版的《The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics》一书的附件出现的。该书及相应的VTK软件由美国GE公司的三位研究人员：Ken Martin、Will Schroeder和Bill Lorensen用其闲暇时间合作编著与开发的，因此该软件的授权完全由这三位决定。由于其开放源码式的授权，该书一上市后，很快就建立起VTK的使用者及开发者社区交流平台，同时GE（特别是GE医疗系统）与其他数家公司也开始提供对VTK的支持。1998年，Will Schroeder和Ken Martin离开GE创立了Kitware公司（[http://www.kitware.com](http://www.kitware.com/)）。有了Kitware的资金支持，VTK社区快速地成长，其在学术研究及商业应用领域都受到重用，例如Slicer生物医学计算软件使用VTK作为其核心，许多讨论研究VTK的IEEE论文出现。VTK也是许多大型研究机构，如Sandia，Los Alamos及Livermore国家实验室与Kitware的合作基础，这些研究中心使用VTK作为数据可视化处理工具。VTK同时也是美国国家卫生研究院（National Institutes of Health，NIH）创立的美国国家医学影像计算合作联盟（National Alliance for Medical Image Computing，NA-MIC，[http://www.na-mic.org](http://www.na-mic.org/)）的关键计算工具。

## VTK的特点

* 强大的三维图形功能。VTK既支持基于体素的体绘制（Voxel-Based Volume Rendering），又保留了传统的面绘制（Surface Rendering），从而在极大的改善可视化效果的同时又可以充分利用现有的图形库和图形硬件。
* VTK的体系结构使其具有非常好的流Streaming和高速缓存Caching的能力，在处理大量的数据时不必考虑内存资源的限制。
* VTK能够更好的支持基于网络的工具，比如Java和VRML。随着Web和Internet技术的发展，VTK有着很好的发展前景。
* 能够支持多种着色，如OpenGL等。
* VTK具有设备无关性，使其代码具有良好的可移植性。
* VTK中定义了许多宏，这些宏极大的简化了编程工作并且加强了一致的对象行为。

# 安装VTK

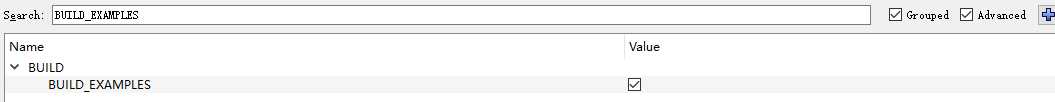
## 生成对应的VS版本的工程文件

* 下载源码，打开CMAKE工具，配好源码路径和工程生成路径

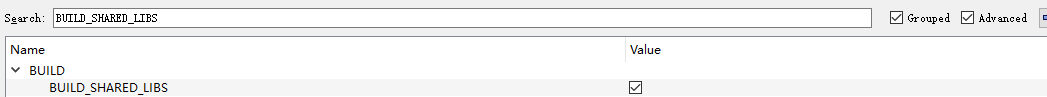


* 单击configure等待，等待完成后开始配置：

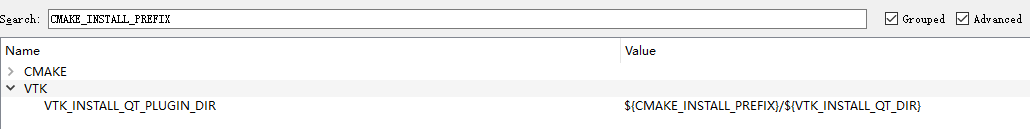
1. 包含例程（可选）BUILD\_EXAMPLES



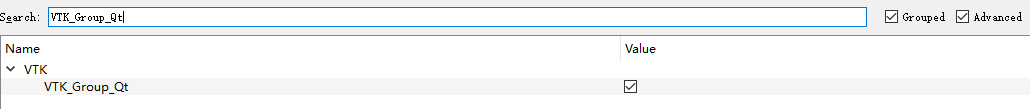
1. 生成动态共享库DLL（必选）BUILD\_SHARED\_LIBS



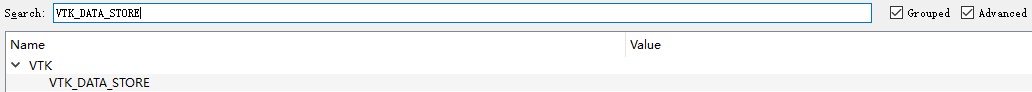
1. VTK库要安装的位置(文件深度不能太深，不能有中文，特殊字符) CMAKE\_INSTALL\_PREFIX



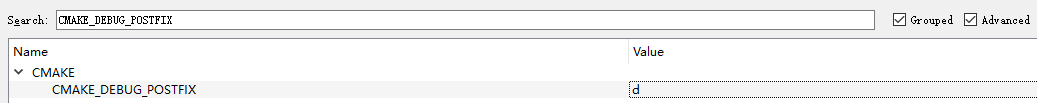
1. 使用Qt（必选）VTK\_Group\_Qt



1. 选择VTK-8.X.X-Data所在的路径（可选）VTK\_DATA\_STORE



1. 区别release和debug版（可选）CMAKE\_DEBUG\_POSTFIX



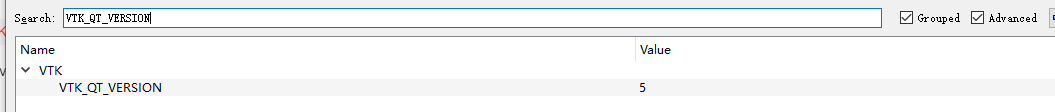
* 再次单击configure等待

1. 选择Qt安装路径（必选）**Qt\_DIR**

例如(E:\install\qt\5.14.2\msvc2017\_64\lib\cmake\Qt5)



1. 确认Qt版本 VTK\_QT\_VERSION

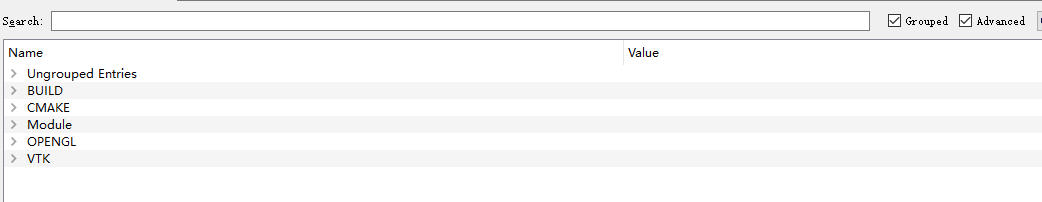


1. 勾选



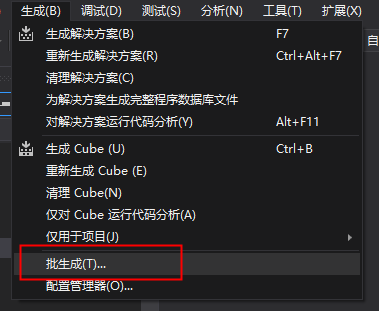
* 再次单击configure等待

没有红色则成功，点击生成，否则请重新检查配置

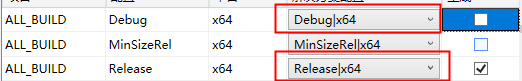


## 进入工程目录，打开VTK.sln文件

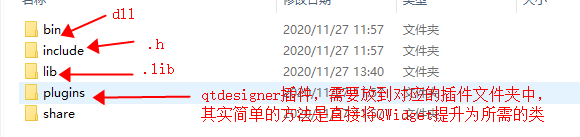
1. 打开批生成



Release和Debug选择一个开始生成，等待约1小时

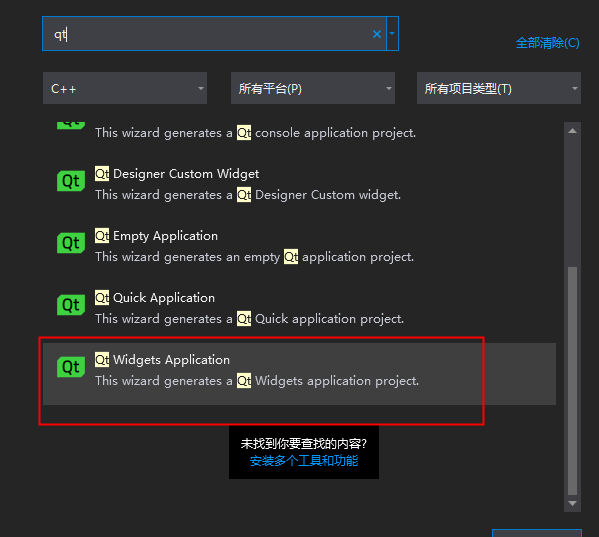


* 成功后到先前的安装目录

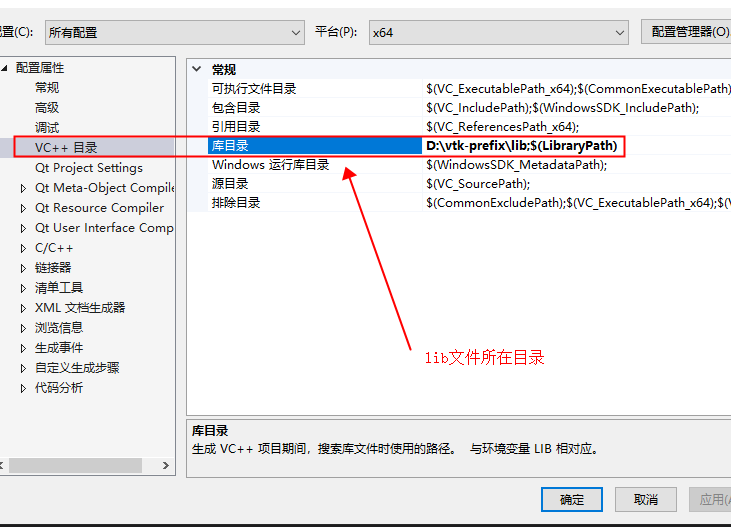


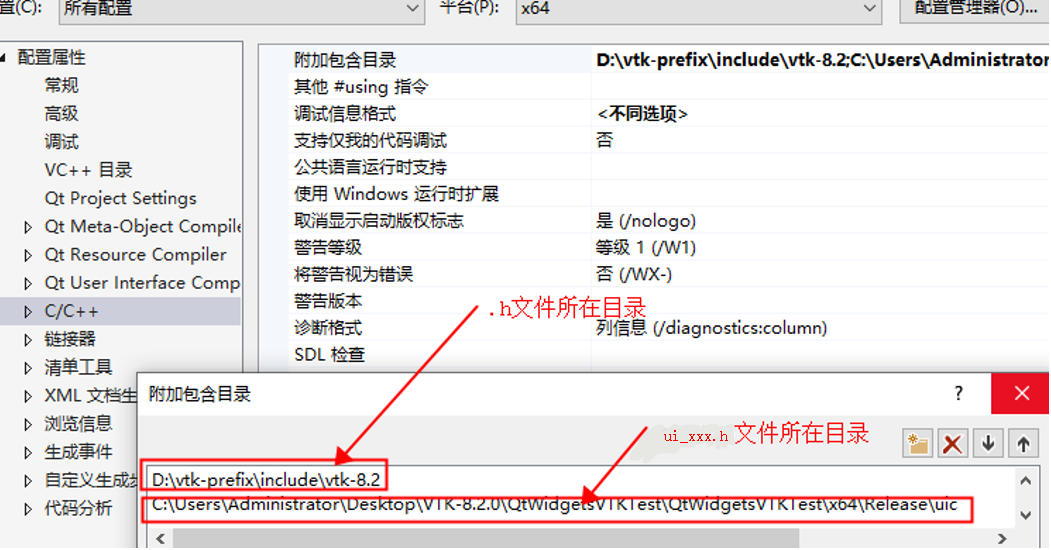
## 创建一个测试工程

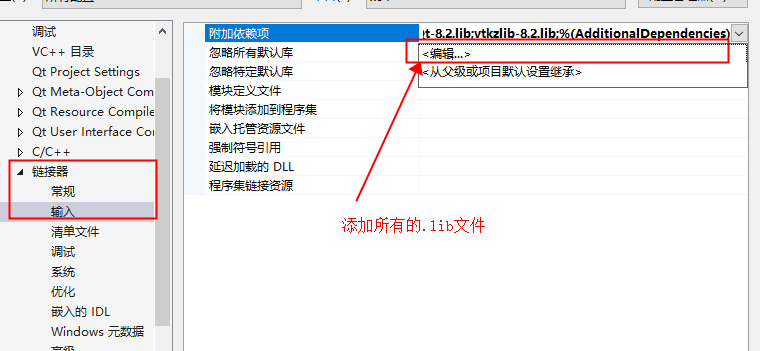
1. 使用VS创建Qt gui项目(事先安装配置配置好VS Qt开发插件)，



1. 打开项目属性做如下配置

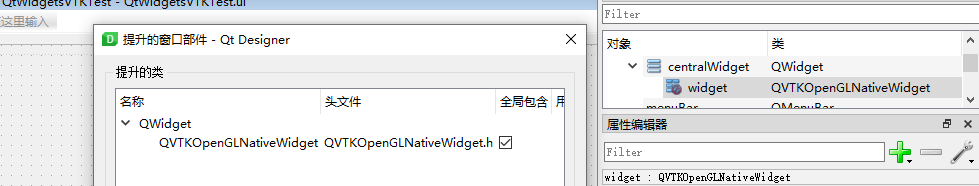








使用如上脚本可快速获得所有文件名（DIR \*.lib /B >LIBLIST.TXT）

1. 打开.ui文件拖出一个qwidget将其提升为QVTKOpenGLNativeWidget(这个性能好)

另外还有QVTKWidget/QVTKWidget2/QVTKOpenGLWidget三种组件适用于过去的版本；

在Qt5.4以前，在Qt中使用OpenGL一般是使用QGLWidget，相对应的VTK控件只能选择QVTKWidget/QVTKWidget2。其中QVTKWidget2是QVTKWidget的升级，不过也是十分古老了，这两个类都不建议使用。

Qt5.4以后，引入了QOpenGLWidget旨在代替老旧的QGLWidget，但是QGLWidget仍然是被保留的。对应的VTK控件为QVTKOpenGLWidget/QVTKOpenGLNativeWidget。

1. 保存编译

在mainwindows的cpp文件中添加：

#include <vtkAutoInit.h>

VTK\_MODULE\_INIT(vtkRenderingOpenGL2)

VTK\_MODULE\_INIT(vtkInteractionStyle)

完成后拷贝所有先前生成的vtk目录中bin下的所有dll到可执行文件目录下(也可以自行配置环境变量)。

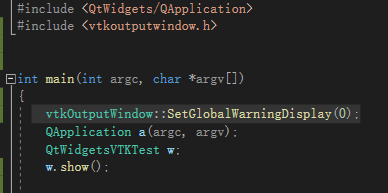
1. 编译运行

可以看到弹出了两个窗口一个是Qt的mainwindow另一个是vtk的打印窗口

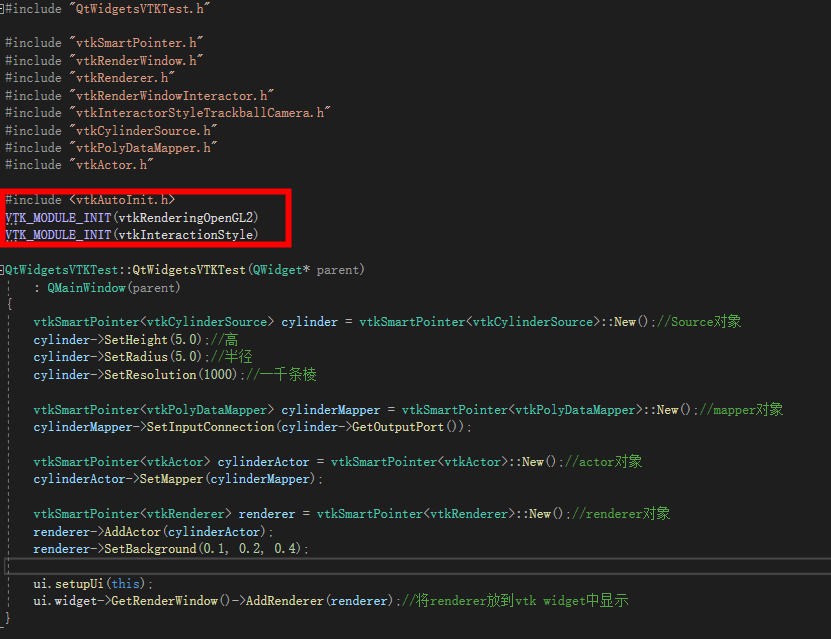
在main.cpp中包含

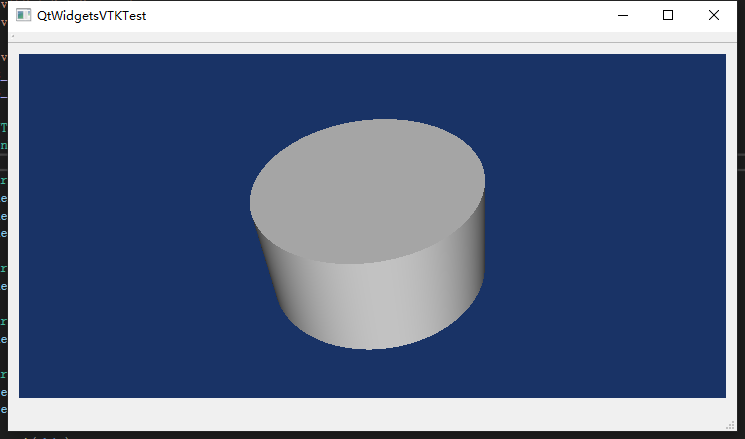
#include <vtkoutputwindow.h>

并在第一行调用vtkOutputWindow::SetGlobalWarningDisplay(0);可隐藏vtk打印窗口



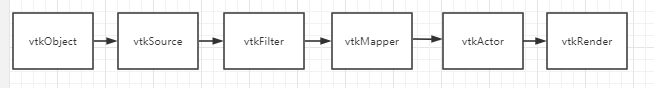
1. 画一个圆柱体





调用renderwindow的addRenderer方法将renderer指针设入即可完成显示

# VTK的组成



VTK是在三维函数库OpenGl的基础上，采用面向对象的设计方法发展起来的。

它有两种不同的方式:图形模型和可视化模型

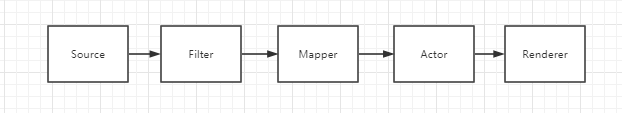
## 图形模型

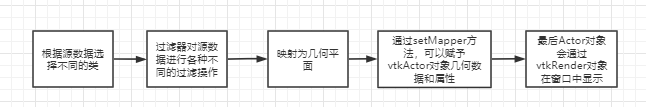
图形模型就是3D图形的抽象，如上一章中画的圆柱体，整个图形模型表现了3d图形系统的本质特征，主要有9类基本对象：

* Render master(渲染控制器)定义与设备无关的坐标计算方法，创建渲染窗口
* render window（渲染窗口） 管理显示设备上的窗口，一个或多个绘制方法可在渲染窗口上创建一个场景；渲染窗口是用户图形界面，其中包括了设置渲染窗口的大小，产生立体显示效果等方法：
* Renderer（渲染器） 是管理光源照相机和绘制对象等的位置、属性等，提供了世界坐标系，观察坐标系及显示坐标系之间的转换；
* Lights（灯光） 可在场景中照亮绘制对象，可通过调用参数改变控制灯光的状态、照射角度、照射强度、颜色等，并支持点光源和平行光源。
* Camera（照相机） 是定义观察者的位置、聚焦点和其他有关属性，参数可由调用者根据需要设置。
* Actor（角色） 代表渲染场景中的绘制对象实体，通过参数调节可以设置角色的位置、方向、渲染特性、引用、纹理影射等属性，并可对角色进行缩放。
* Property（属性） 是说明几何物体的一些特性，实现三维图形真实感。
* Transform（变换） 是一个放置4×4变换矩阵的堆栈，可以进行各种操作。图形模型支持3D几何数据绘制、3D体数据绘制、2D几何文字、图像绘制。
* Mapper（映射、制图人） 指定了渲染数据和图形库中基本图元之间的联系，一个或多个角色可以使用相同的映射，有多个参数对其进行控制。

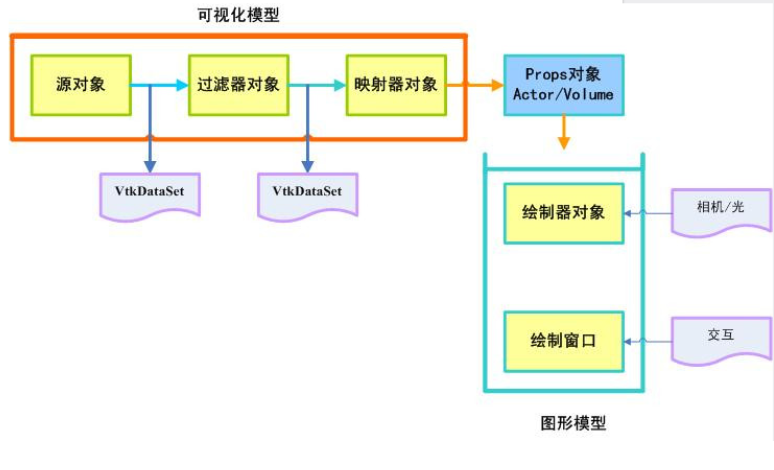
可视化模型

## VTK绘制管线





1. vtkObject提供整个可视化流程的基本方法；vtkSource派生于vtkObject，为整个可视化流程的开始比如读取数据等定义具体的行为和接口；
2. vtkFilter派生于vtkSource对数据进行各种处理，将原始数据经过各种过滤器的处理后，转换为可以直接用某种算法模块对其进行处理的形式；
3. vtkMapper派生于vtkObject，将经过各种filter处理后的应用数据映射为几何数据，为源数据与图像数据之间定义了接口
4. vtkActor角色对象将可视化的数据在窗口中表现出来，在Visualization Toolkit中任何可视化的数据都是通过角色在窗口中表现出来的。
5. 最后通Actor通过vtkRender将结果在窗口中显示出来。



# 编码须知

## 低级对象模型

### 引用计数

几乎所有的VTK类都派生于vtkObjectBase类，基于此类的派生类都必须使用其静态方法New创建实例，使用实例的Delete方法释放改对象；

* vtkObjectBase\* obj = vtkExampleClass::New();

当通过首次静态方法New创建实例时，会将该类对象的引用计数初始化为1，当创建或删除该类对象的时候会调用Register和UnRegister自动增减其引用计数；通常创建的句柄会提供各种各样的”set”方法；

* otherObject->SetExample(obj);

此时引用计数为2，因为原始指针和该原始执政的引用指针都存储在另一个对象中，当该otherObject中存储的指针不在需要使用时使用Delete方法；

* obj->Delete();

请确保每次掉用New后使用Delete释放以免造成内存泄露。

另外vtksmartpoiner类模板提供了智能指针的功能，以上例子可以写成。

* vtkSmartPointer<vtkExampleClass> obj

= vtkSmartPointer<vtkExampleClass>::New();

* otherObject->SetExample(obj);

### 运行时类型信息

由于在实际运行过程中，指针的实际类型可能和当前取到的类型不一致，所以所有的类（除了模板类）都提供了GetClassName方法；

* const char\* type = obj->GetClassName();

也可以使用isA方法来判断该类一个其他类还是某一个子类；

* if(obj->IsA("vtkExampleClass")) { ... }

一个超级指针可以通过DownCast方法安全地转换为其派生类型；

* vtkExampleClass\* example = vtkExampleClass::SafeDownCast(obj)

只有当对象确实是派生类型的实例，此操作才会运行成功，否则将返回空指针。

### 对象状态显示

调试时需要打印当前对象的状态，对于vtk对象可以调用其Print方法

* obj->Print(cout);

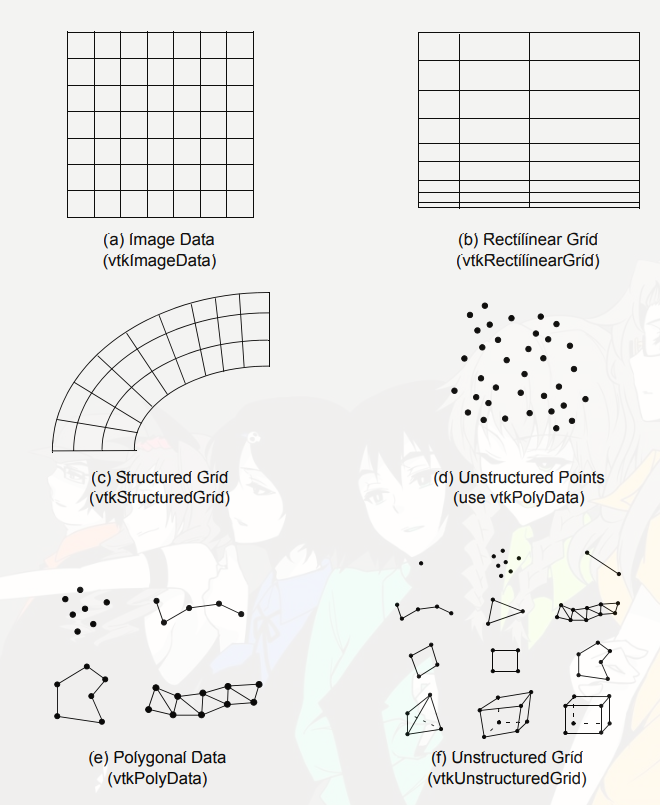
## 渲染引擎介绍

渲染引擎由以下一些基本类构成，这不是完整的列表，但是是最常使用的。

### vtkProp(道具类)

场景中存在的数据可视描述由vtkProp的子类表示，vtkProp最常用的子类是vtkActor(用于表示场景中的几何数据)和vtkVolume(用于表示场景中的体积数据)；还有一些表示2D数据的属性，比如vtkActor2D。

VtkProp子类通常负责记录其对象在场景中的位置、大小、方向，这些参数用于控制该道具的放置，通常依赖该道具是否是场景中的3d对象或2d注释；



对于3D道具像vtkActor和vtkVolume(继承于vtkProp3D(继承于vtkProp))，我们也可以直接控制其位置，方向，比例等参数，也可以使用4x4变换矩阵。

对于2D提供注释的道具，如vtkScalarBarActor，批注的大小位置可以通过多种方式i定义，包括相对于指定视口的大小。

另外，除了提供放置控制，道具通常由一个映射器对象，该对象保存数据并且知道如何呈现数据，以及控制其参数，比如颜色和透明度。

有超过50种专业道具如vtkImageActor（用于显示图片）和vtkPieChartActor（用于创建数值数组的饼图可视化显示）。有一些道具直接包含控制外观的参数，以及直接引用要呈现的输入数据，因此不需要使用属性或映射器。

vtkFollower是vtkActor的一个特殊子类，它将自动更新其方向，以便持续地面向指定的摄像机。

vtkLODActor也是vtkActor的一个特殊子类，它可以自动的改变几何表示以便维持交互频率。

vtkLODProp3D是vtkProp3D的一个子类，它可以在多个mapper之间选择，至是体积贴图和几何贴图的混合，以提供交互性。

vtkAssermbly允许层次化所有的道具参与者，当层次结构被转换旋转或缩放时，能够正确的管理转换。

### vtkAbstractMapper（映射器类）

有些道具如vtkActor和vtkVolume通过使用vtkAbstractMapper的子类对象的引用来输入数据和提供实际的渲染功能。VtkPolyDataMapper是渲染多边形几何体的主要映射器。

对于体积对象vtk提供了几种渲染计数包括用于渲染vtkImageData的vtkFixedPointVolumeRayCastMapper，以及vtkProjectedTetrahedra mapper用于呈现vtkUnstructuredGrid数据。

### vtkProperty and vtkVolumeProperty.（属性类）

有些道具使用单独的属性对象来保存控制数据外观的各种参数，这样可以更轻松地在场景中的不同对象之间共享外观设置。VtkActor对象使用vtkProperty来存储诸如颜色、不透明度、以及材质的环境光、漫反射和镜面反射系数等参数。

vtkVolume对象使用vtkVolume属性来捕获适用的参数到体积对象，例如将标量值映射到颜色和不透明度的传递函数。许多mapper还提供了可用于显示内部裁剪平面结构的功能设置。

### vtkCamera（相机类）

vtkCamera包含控制如何查看场景的参数。VtkCamera有一个位置、一个焦点和一个定义场景中“向上”的向量。其他参数恐怖告知特定的视图变换（平行或透视）、比例或视图角度，以及视锥的近、源裁剪平面。

### vtkLight（灯光类）

为场景计算照明时，需要一个或多个vtkLight对象。VtkLight对象存储灯光的位置和方向，以及颜色和强度。灯还有一个类型，它描述灯光相对于相机的位置并照射在相机的焦点上，而场景灯位于场景中的固定位置。

### vtkRenderer(渲染器类)

构成场景的物体包括道具相机灯光都被收集在vtkrenderer中。vtkrenderer负责管理渲染过程。在vtkRender窗口中可以同时使用多个对象。这些渲染器可以渲染到渲染窗口的不同矩形区域(视口)，或是重叠。

### vtkRenderWindow（渲染窗口类）

vtkRenderWindow提供操作系统之间的连接，以及VTK渲染引擎。vtkRenderWindow的特定于平台的子类负责在计算机上的本机窗口系统中打开窗口并管理显示过程。使用VTK进行开发时，只需使用独立于平台的vtkRenderWindow它在运行时被自动替换为正确的特定于平台的子类。vtkRenderWindow包含vtkrender的集合，以及控制渲染功能的参数，例如作为立体，抗锯齿，运动模糊和焦深。

### vtkRenderWindowInteractor（渲染窗口交互器类）

vtkrenderWindowInteractior负责处理鼠标、键和计时器事件，并通过VTK的命令实现路由这些事件/观察者设计模式。vtkInteractorStyle侦听这些事件并处理它们，以便提供运动控制，如旋转、平移和缩放。vTkrenderWindowInteractior自动创建适用于3D场景的默认交互器样式，但您可以更改为例如，选择一个用于二维图像查看，或创建您自己的自定义交互器样式。

### vtkTransform（转换类）

场景中需要放置的许多对象，例如道具、灯光和相机有一个vtkTransform参数，可用于轻松操纵对象的位置和方向。vtkTransform可用于描述整个线性范围（也称为仿射）三维坐标变换，内部表示为4x4齐次变换矩阵。vtkTransform对象将以默认的标识矩阵开始或者，可以以流水线方式将转换链接在一起，以创建复杂的行为。管道机制确保，如果修改管道中的任何转换，则所有后续的转换相应地更新。

### vtkLookupTable, vtkColorTransferFunction, and vtkPiecewiseFunction

可视化标量数据通常需要定义从标量值到颜色和不透明度的映射。无论是在几何曲面渲染（其中不透明度将定义曲面的半透明性）中，还是在体积中都是如此在渲染中，不透明度将表示沿穿过体积的某段光线累积的不透明度。对于几何渲染，此映射通常使用vtkLookupTable创建，而在体绘制中，vtkColortTransferFunction和vtkPiecewiseFunction将使用vtkPiecewise函数。

## 用户输入

可以使用任何GUI开发框架来处理用户输入，但是如果不进行设置vtk应用程序会有默认的interactor，所以通常需要自定义设置一下我们所关注输入。

### 观察者

//鼠标点击 观察者

void ClickCallbackFunction(vtkObject\*, long unsigned int, void\*, void\*)

{

std::cout << "Click callback" << std::endl;

// We can get the calling object like this:

// vtkRenderWindowInteractor \*iren =

// static\_cast<vtkRenderWindowInteractor\*>(caller);

} vtkSmartPointer<vtkCallbackCommand> clickCallback = vtkSmartPointer<vtkCallbackCommand>::New();

clickCallback->SetCallback(ClickCallbackFunction);

iren->AddObserver(vtkCommand::LeftButtonPressEvent,clickCallback);

### 默认样式

vtkSmartPointer<vtkInteractorStyleTrackballCamera> style2 =

vtkSmartPointer<vtkInteractorStyleTrackballCamera>::New();

iren->SetInteractorStyle(style2);

### 重写样式事件函数

class KeyPressInteractorStyle : public vtkInteractorStyleTrackballCamera

{

public:

static KeyPressInteractorStyle\* New();

vtkTypeMacro(KeyPressInteractorStyle,

vtkInteractorStyleTrackballCamera);

virtual void OnKeyPress() override

{

// Get the keypress

vtkRenderWindowInteractor\* rwi = this->Interactor;

std::string key = rwi->GetKeySym();

// Output the key that was pressed

std::cout << "Pressed " << key << std::endl;

// Handle an arrow key

if (key == "Up")

{

std::cout << "The up arrow was pressed." << std::endl;

}

// Handle a "normal" key

if (key == "a")

{

std::cout << "The a key was pressed." << std::endl;

}

// Forward events

vtkInteractorStyleTrackballCamera::OnKeyPress();

}

};

vtkStandardNewMacro(KeyPressInteractorStyle);

//应用

vtkSmartPointer<KeyPressInteractorStyle> style2 = vtkSmartPointer<KeyPressInteractorStyle>::New();

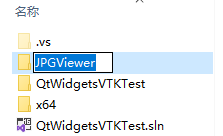
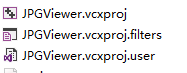
iren->SetInteractorStyle(style2);

style2->SetCurrentRenderer(m\_renderer);

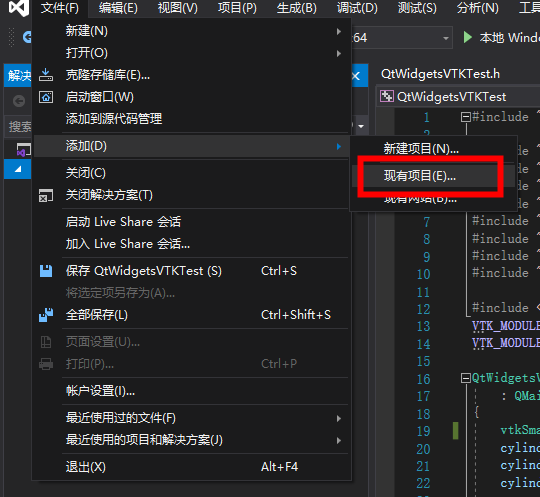
# Qt VTK应用

如第二章中的测试程序，能够选择QVTKWidget/QVTKWidget2/QVTKOpenGLWidget/QVTKOpenGLNativeWidget四种widget用来显示renderer对象。性能由低到高排列。Qt5.4以前使用前两种，Qt5.4以后使用后两种。

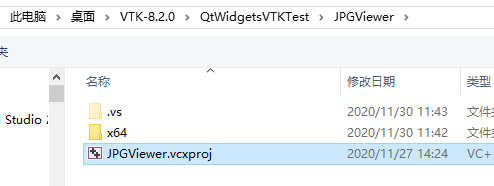
## 实现一个JPG图像浏览器

直接拷贝上一个工程并重命名为JPGViewer，然后打开解决方案，添加现有项目；



选择JPGViewer.vcxproj这个项目



随后打开

### vtkOutputWindow.h

#pragma once

#include <QtWidgets/QMainWindow>

#include "ui\_QtWidgetsVTKTest.h"

class vtkImageViewer2;

class vtkRenderer;

class QtWidgetsVTKTest : public QMainWindow

{

Q\_OBJECT

public:

QtWidgetsVTKTest(QWidget \*parent = Q\_NULLPTR);

private slots:

//响应打开图像文件的槽函数

void onOpenSlot();

private:

Ui::QtWidgetsVTKTestClass ui;

vtkSmartPointer< vtkImageViewer2 > m\_pImageViewer;

vtkSmartPointer< vtkRenderer > m\_pRenderder;

};

### vtkOutputWindow.cpp

#include "QtWidgetsVTKTest.h"

#include "vtkSmartPointer.h"

#include "vtkRenderWindow.h"

#include "vtkRenderer.h"

#include "vtkRenderWindowInteractor.h"

#include "vtkInteractorStyleTrackballCamera.h"

#include "vtkCylinderSource.h"

#include "vtkPolyDataMapper.h"

#include "vtkActor.h"

#include <vtkAutoInit.h>

#include <QDir>

#include <QFileDialog>

#include <vtkImageViewer2.h>

#include <vtkSmartPointer.h>

#include <vtkImageViewer2.h>

#include <vtkJPEGReader.h>

#include <vtkImageActor.h>

#include <qdebug.h>

VTK\_MODULE\_INIT(vtkRenderingOpenGL2)

VTK\_MODULE\_INIT(vtkInteractionStyle)

QtWidgetsVTKTest::QtWidgetsVTKTest(QWidget\* parent)

: QMainWindow(parent)

{

ui.setupUi(this);

m\_pImageViewer = vtkSmartPointer< vtkImageViewer2 >::New();

m\_pRenderder = vtkSmartPointer< vtkRenderer >::New();

// 设置m\_QVTKWidget的渲染器

ui.widget->GetRenderWindow()->AddRenderer(m\_pRenderder);

connect(ui.action\_JPG, &QAction::triggered,

this, &QtWidgetsVTKTest::onOpenSlot);

}

void QtWidgetsVTKTest::onOpenSlot()

{

QString filter;

filter = "JPEG image file (\*.jpg \*.jpeg)";

QDir dir;

QString fileName = QFileDialog::getOpenFileName(this,

QString::fromLocal8Bit("打开图像"), dir.absolutePath(), filter);

if (fileName.isEmpty() == true) return;

// 支持带中文路径的读取

QByteArray ba = fileName.toLocal8Bit();

const char\* fileName\_str = ba.data();

qDebug() << fileName\_str;

// printf("%s", fileName\_str);

// 用vtkJPEGReader读取JPG图像

vtkSmartPointer<vtkJPEGReader> reader = vtkSmartPointer<vtkJPEGReader>::New();

reader->SetFileName(fileName\_str);

reader->Update();

// 将reader的输出作为m\_pImageViewer的输入，并设置m\_pImageViewer与渲染器m\_pRenderer的关联

m\_pImageViewer->SetInputData(reader->GetOutput());

m\_pImageViewer->UpdateDisplayExtent();

m\_pImageViewer->SetRenderWindow(ui.widget->GetRenderWindow());

m\_pImageViewer->SetRenderer(m\_pRenderder);

//m\_pImageViewer->SetupInteractor(ui.widget->GetRenderWindow()->GetInteractor());

m\_pImageViewer->SetSliceOrientationToXY();//默认就是这个方向的

m\_pImageViewer->GetImageActor()->InterpolateOff();

m\_pRenderder->ResetCamera();

m\_pRenderder->DrawOn();

m\_pRenderder->SetBackground(1.0, 1.0, 1.0);

ui.widget->GetRenderWindow()->Render();

}

### main.cpp

#include "QtWidgetsVTKTest.h"

#include <QtWidgets/QApplication>

#include <vtkoutputwindow.h>

int main(int argc, char \*argv[])

{

vtkOutputWindow::SetGlobalWarningDisplay(0);

QApplication a(argc, argv);

QtWidgetsVTKTest w;

w.show();

return a.exec();

}

### QtWidgetsVTKTest.ui

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>QtWidgetsVTKTestClass</class>

<widget class="QMainWindow" name="QtWidgetsVTKTestClass">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>729</width>

<height>400</height>

</rect>

</property>

<property name="windowTitle">

<string>QtWidgetsVTKTest</string>

</property>

<widget class="QWidget" name="centralWidget">

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QVTKOpenGLNativeWidget" name="widget" native="true"/>

</item>

</layout>

</widget>

<widget class="QMenuBar" name="menuBar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>729</width>

<height>23</height>

</rect>

</property>

<widget class="QMenu" name="menu\_JPG">

<property name="title">

<string>文件</string>

</property>

<addaction name="action\_JPG"/>

</widget>

<addaction name="menu\_JPG"/>

</widget>

<widget class="QToolBar" name="mainToolBar">

<attribute name="toolBarArea">

<enum>TopToolBarArea</enum>

</attribute>

<attribute name="toolBarBreak">

<bool>false</bool>

</attribute>

</widget>

<widget class="QStatusBar" name="statusBar"/>

<action name="action\_JPG">

<property name="text">

<string>打开JPG</string>

</property>

</action>

</widget>

<layoutdefault spacing="6" margin="11"/>

<customwidgets>

<customwidget>

<class>QVTKOpenGLNativeWidget</class>

<extends>QWidget</extends>

<header location="global">QVTKOpenGLNativeWidget.h</header>

<container>1</container>

</customwidget>

</customwidgets>

<resources>

<include location="QtWidgetsVTKTest.qrc"/>

</resources>

<connections/>

</ui>

## 模型文件的导入和导出

Vtk提供了许多类诸如xxReader，xxImporte可以用来导入顶点信息，具体要导入什么模型可以在<https://kitware.github.io/vtk-examples>找到所需的Demo。同样的导出也有许多的xxWriter类可以使用。

### 导入

例如使用以下代码即可导入一个stl模型：

…

vtkOBJReader \* reader = vtkOBJReader::New();//创建资源对象

reader->SetFileName(filePath.c\_str());//设置文件绝对路径

reader->Update();//更新资源对象

m\_cylinderMapper = vtkSmartPointer<vtkPolyDataMapper>::New();//创建映射对象

m\_cylinderMapper->SetInputConnection(reader->GetOutputPort());//连接资源对象的输出端口

…

reader->Delete();//若将其智能作为成员变量，析构时则可省略该语句

### 导出

使用如下代码导出一个obj模型文件：

vtkNew<vtkOBJWriter> objWriter;

objWriter->SetFileName(filePath);//设置文件名绝对路径

objWriter->SetInputConnection(anyModelSoure->GetOutputPort());//连接任意资源对象的输出端口

objWriter->Write();//创建文件并写入数据

# 引用

《User’s Guide》 <https://www.kitware.com/products/books/VTKUsersGuide.pdf>（官方指南）

<https://blog.csdn.net/www_doling_net/article/details/8763686>（东灵工作室出的教程说的非常详细）

<https://kitware.github.io/vtk-examples/>（非常多的demo）