《VTK User's Guide 中文版》系列分享专栏

简介

这个系列是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8）争取每周都更新翻译。欢迎转载，另请转载时注明本文出处，谢谢合作！

文章

第01章-欢迎学习VTK

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本节对应原书中的第3页至第7页】

欢迎开启VTK之旅——《VTK用户指南》。VTK是一个开源的、面向对象的计算机图形、可视化和图像处理的软件系统。虽然VTK比较庞大、复杂，但是当你了解了它基本的面向对象的设计和实现的方法以后，你会发现VTK还是比较简单、比较容易入门的。这本《VTK用户指南》的目的就是帮助你学习、熟悉各种重要的VTK类。

如果你读过这本书的其他版本，你会发现我们已经开始使用"版次"（Edition Number）而不是VTK的"版本号"（VersionNumber）来区分更新这本书了。这本书是《VTK用户指南》的"第11版次"（11thEdition）。《VTK用户指南》的出版已经超过十一个年头，而这个版本的用户指南的出版距离VTK的面世也超过了十六个年头。虽然这本书是在VTK 5.6版本发布前出版的，我们可以很有信心地说，这本书中所列举的所有素材对VTK将来要发布的其他版本也将会是有效的。VTK把"向后兼容"（Backwards Compatability）这个特性当作重中之重，也许某些新的特性在后续的VTK版本中会增加进来，显然不会在本书中一一罗列，而且某个现有的特性保持一成不变也是很少有的。

VTK是一个庞大的系统。因此，不可能完整的将所有VTK的类以及它们的方法都在这个指南中详细介绍。不过，这个指南将会向你介绍一些重要的系统概念，带领你沿着学习曲线尽可能快速、高效地学习。一旦你掌握了这个基础，我们建议你利用好其他的学习资源，这些资源包括VTK Doxygen文档(http://www.vtk.org/doc/nightly/html/)以及VTK用户社区。

VTK是一个开源的软件系统。这意味着成千上万的开发人员和用户对这个系统做出了贡献。如果你发现VTK是一种有用的工具，我们鼓励你贡献出Bug修正代码、算法、想法或者是应用程序回馈到VTK社区中。你也可以联系像Kitware这样的商业公司来开发或者增加新的特性和工具。

1.1本书组织结构

这本用户指南共划分成三个部分，每一部分都细分为若干独立的章节。第一部分是VTK的概述，包括（下一章）如何在计算机上安装VTK，即安装预编译库和可执行程序以及直接从源代码编译。第一部分也介绍了基本的系统概念，包括系统架构概览、如何使用C++、TCL、Java和Python这些编程语言创建应用程序。从某种程度上讲，第二部分是本书的核心，因为这一部分通过众多的例子来说明重要的系统特性。第三部分是针对VTK高级用户的，介绍了如何创建自定义的类、扩展系统以及与各种视窗和GUI系统进行整合。第19章罗列了部分VTK类的继承图，通过这些继承图可以对VTK类库之间的关系有个大概的认识，同时也对部分过滤器（Filter）做了简要的概括，以及对VTK文件格式的描述。最后是本书的索引部分，借助它可以方便的检索本书的内容。

1.2如何学习VTK

VTK的用户主要分为两类，第一类是使用C++创建类的开发人员，第二类是使用C++类库来建立完整的应用程序的开发人员。类开发人员必须熟练掌握C++，如果你正在扩展或修改VTK，你必须同时熟悉VTK的内部结构和设计（第三部分所涵盖的内容）。应用程序开发人员可以使用C++或者不用C++，因为编译好的VTKC++类库已经"封装"了TCL、Python、VB和Java等解释性语言。不过，作为一名应用程序开发人员，你必须了解VTK对象的外部接口和他们之间的关系。

学习VTK的关键是熟悉它的对象并知道如何组合他们。如果你是一位VTK初学者，先从安装VTK软件系统开始学习。如果你是一位类开发人员，你可能会下载源代码并编译它们。应用程序开发人员可能仅仅需要预编译库和可执行程序。我们建议你通过研究例子程序（如果你是一位应用程序开发人员）和源代码（如果你是一位类开发人员）来学习这个软件系统。第三章中会对VTK软件系统中的一些关键概念作一个简要概览，然后在第二部分的例子中对这些内容进行回顾。你也可以运行随源代码发布的各种各样的例子，可以在源代码的VTK/Examples目录中找到（请查看文件VTK/Examples/README.txt来获取各个子目录下示例的描述）。在源代码发布目录中也有很多测试用例，如在VTK/Graphics/Testing/Tcl以及VTK/Graphics/Testing/Cxx，虽然大多数测试例子没有被载入文档的测试脚本，但是他们对了解VTK中的类如何组合使用是相当有用的。

1.3VTK软件系统组织结构

接下来简要描述一下各个源码目录中的内容，列举各个目录的软件功能、文档内容以及数据。

获取软件

可以通过以下两种方式访问VTK的源代码。

1. 从VTK网站(http://www.vtk.org)下载官方发布的源码。

2. 通过Git访问VTK的源码(gitclone http://vtk.org/VTK.git VTK)。

这本用户指南假定你选择的VTK源码是官网发布的版本。本书的写作时间是2009年9月份。在写作本书时我们考虑了VTK 5.4以及即将发布的VTK 5.6里的一些新特性。注意，本书的内容对VTK的后续版本也是同样适用。还有一点要注意的是，在过去的VTK发布的版本号中，我们使用了一个主识别号来代表版本的更新(如，VTK 4.4到VTK5.0)，某种程度上也表示版本的向后兼容性。但是随着版本的频繁发布，我们可能会面临着发布VTK 5.10版本(5.10可能会混淆某些用户，让人误会是5.2的前一个版本，实际上它是在5.8之后发布的)，或者会发布VTK6.0，而6.0会让人更加清楚版本的向后兼容性，所以将来我们会选择发布VTK 6.0而不是VTK5.10（译者：很显然，Kitware食言了，最后还是发布了VTK5.10，VTK 5的最后一个版本是VTK5.10.1）。当更新版本的VTK发布时，你还是可以阅读本书的内容，尽管某些最新的特性没有在本书中提到，但本书中的资源对于将来VTK发布的版本仍会是适用的。如果想了解将来要发布的VTK的新特性，可以关注VTK的邮件列表(http://www.vtk.org/VTK/help/mailing.html)或是Kitware内部发行的免费刊物《Source》(http://www.kitware.com/products/thesource.html)。

我们强烈建议你使用VTK 5.4或者更高的VTK官方发布版本。官方发布版本比Git版本稳定性、一致性要更好以及经过严格的测试。当然，如果你一定要用VTK的最新版本，就先查看一下VTK测试公告栏里的信息。VTK使用了Kitware的软件开发流程(SoftwareProcess)(http://www.kitware.com/solutions/softwareprocess.html)。在你更新Git仓库时，先确认测试公告栏是否为"绿色"，如果不是绿色，有可能所更新的版本就会不稳定。(见"Kitware'sQuality Software Process"一节，了解更多VTK代码质量控制公告栏等信息。)

目录结构

开始学习VTK之前，首先有必要对VTK的目录结构做一个整体的认识。即使你是选择预编译二进制文件的安装方式，了解这部分内容也有助你在VTK源码中更方便快捷地查找例子、代码以及文档文件。下面是VTK文档的组织结构：

l InfoVis– 包含了用于信息可视化的类。

l Views– 包含了对数据可视化的类，包括：过滤器(Filter)、可视化(Visualization)、交互(Interaction)和选择(Selection)。

l VTK/CMake– 用于跨平台编译的配置文件。

l VTK/Common– 核心的类

l VTK/Examples– 包含按主题归档的详细注释的例子。

l VTK/Filtering– 可视化管道中与数据处理有关的类。

l VTK/GenericFiltering– VTK与外部模拟包的接口适配框架。

l VTK/Geovis– 用于地形可视化的视图、数据源和其他对象。

l VTK/Graphics– 处理3D数据的过滤器(Filter)。

l VTK/GUISupport– VTK与MFC和Qt等用户图形界面开发工具的接口类。

l VTK/Hybrid– 同时要求使用图形学和图像处理功能的类。

l VTK/Imaging– 图像处理过滤器。

l VTK/IO– 用于读写数据的类。

l VTK/Parallel– 支持并行处理类，如MPI。

l VTK/Rendering– 用于渲染的类。

l VTK/Utilities– 支持像expat, png, jpeg, tiff和zlib等软件库。Doxygen目录包含了从源代码里生成Doxygen文档的脚本和配置文件。

l VTK/VolumeRendering– 用于体绘制的类。

l VTK/Widgets– 3D Widget类。

l VTK/Wrapping– 支持对Tcl, Python和Java的封装。

文档

除了这本用户指南以及《The Visualization Toolkit AnObject-Oriented Approach to 3D Graphics》，你还可以获取到其他的文档资源。

Doxygen文档。Doxygen文档是学习VTK非常重要的文档工具，每个web页上面都详细地描述了VTK里每个类的用法、数据成员、成员函数等。文档里也列出了类的继承图以及与该类协同工作的其他类的关系图表。每个文档都有链接到其他类以及源代码的超链接。Doxygen文档可通过http://www.vtk.org/doc/nightly/html/在线访问。注意你所用的VTK版本应该与Doxygen文档版本一致。

头文件。每一个VTK类都是由一个.h头文件以及.cxx的实现文件构成。所有在头文件中声明的每一个成员函数都是对该类所提供的方法的一种快速检索。(事实上，Doxygen文档也是通过这些头文件生成的)。

数据

VTK的程序用例以及测试例子里所用到的数据都可以在http://www.vtk.org上下载到，或者通过Git下载。如何使用Git可以访问VTK官方网页。

1.4其他资源

这本用户指南仅仅是学习VTK的可用资源之一，下面列出了一些在线资源、服务、应用程序以及其他的出版物，相信这些资源对你学习VTK会有很大的帮助。

l 与这本指南配套的教科书《The VisualizationToolkit An Object-Oriented Approach To 3D Graphics》深入讲解了许多VTK里使用到的算法、数据结构等。这本书是Kitware公司出版的，你可以在Kitware网站或者亚马逊网站上购买到。

l 《Source》是Kitware公司按季度发行的内部刊物，里面涵盖了Kitware公司的所有开源项目。一旦有新的功能加进VTK时，会有相关的文章发表在《Source》上。与VTK有关的其他有用的资源、入门等东西也会在季刊上发表。你可以在线访问《Source》(http://www.kitware.com/products/thesource.html)，或者通过邮箱注册请求发送每期的《Source》

季刊。

l VTK官网上含有大量的资源，如在线帮助文档、Wiki、常见问题解答、dashboard、bug跟踪以及vtkusers邮件列表的搜索引擎。不管是初学者还是经验丰富的开发者，Doxygen文档都是不可多得的好资源。

l vtkusers邮件列表可以让用户和开发者提问题以及接收别人的解答、发布更新、bug修复和改进的内容，以及提出改进系统的建议。请访问VTK官网查询如何加入vtkusers邮件列表(注册地址：http://public.kitware.com/mailman/listinfo/vtkusers)。

l Kitware公司开设了专业的培训课程。涵盖了Kitware公司的所有开源项目，包括VTK，ITK，CMake和ParaView等，每年在纽约北部地区举办两次。另外Kitware公司也可以根据你的开发团队提供一些定制的培训课程。更多信息请访问Kitware官网或者直接发邮件到courses@kitware.com咨询。

l Kitware公司也提供了商业支持和顾问。主要包括VTK专家辅助项目开发、根据你的项目详细说明书Kitware为你提供大规模的顾问等。你可以访问Kitware官网或者发送邮件到sales@kitware.com获取更多信息。

l ParaView是用VTK实现的针对科学可视化的应用程序。可从http://paraview.org上下载。使用ParaView来学习VTK是比较不错的选择，因为你可以通过用户图形界面来熟悉VTK的大部分常用的功能。这也是非常有参考价值的，你可以加载自己的数据，看看主要有哪些可视化技术是可以使用以及你自己希望具有什么样的性能等等。

l CMake是用于跨平台构建编译环境的开源工具。对于VTK初学者来说，只要掌握非常少的关于CMake的知识就可以在标准的Windows,Linux或Mac OSX平台上成功编译VTK；对于一些高级用户来说，他们会觉得CMake在开发过程中起到的作用是相当明显的，要把VTK移植到非标准的平台上时就需要掌握更多的关于CMake的知识。可以访问CMake官网http://cmake.org了解更多的信息。

l CDash是VTK采用的用于源码测试的开源平台。你可以在VTK官网上找到VTK的测试公告栏(由CDash提供技术支持)。公告栏显示了在不同的平台上经过测试的一些结果。对于一些在非标准的平台上进行开发的人员可以贡献他们的测试结果。10.8节中会有详细内容介绍Kitware的软件开发流程。

译者：国内目前学习VTK入门的中文资料，主要有【东灵工作室】撰写以及整理的VTK系列教程。希望大家能多多支持我们，一起推动VTK在中华地区的发展。

【第1章 翻译完毕】

第02章-VTK安装（1）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本节对应原书中的第9页至第10页】

这一章将详细介绍安装VTK的步骤。安装过程的难点依赖于若干因素。Windows平台下可先安装二进制文件vtk.exe，然后利用它可以运行Tcl脚本文件。使用Python和Java语言开发VTK应用程序时，可以链接VTK库到相应的应用程序中；而在其他非Windows平台下使用VTK时，则必须通过VTK源码来编译VTK（因为要保证VTK在不同平台下预编译文件的实时更新是一件工作量非常庞大的事情，所以我们力求让VTK在各种平台下的编译尽可能的简单）。如果你选择的是从源代码编译VTK，在多核处理器系统中可能只需花费半小时左右或者更快的时间；如果是内存有限的硬件系统，也许就要花费几个小时甚至更长的时间才能编译完成。另外，编译VTK所需时间的长短也取决于你要封装多少种解释性语言到VTK的C++内核中以及你的系统配置。

读者可以参考第三章"VTK系统架构"，这部分内容是关于VTK系统架构的概览，可能有助于你对接下来的编译过程的理解。如果在VTK的编译安装过程中遇到什么问题，读者可以在vtkusers邮件列表提问（参考第一章"其他资源"一节）。

2.1 概览

VTK可以在不同的计算机平台下编译和运行。对于不同的平台，我们必须要考虑不同的操作系统、硬件配置以及编译器等要素，由于这些要素的组合比较多，所以发布针对不同平台、编译环境等都可使用的VTK二进制版本的做法显然是不可取的。因此，建议通过VTK源代码来编译安装VTK，通过编译源代码生成VTK库文件以及可执行程序。

本章首先对CMake作一简要介绍，CMake是一个跨平台的工程构建工具，可用于各种操作系统配置编译环境。接着，本章内容根据不同类型的操作系统分为两个部分：Windows平台和UNIX平台的VTK安装（对于Macintosh OSX或Linux平台，可以参考UNIX平台的进行安装）。读者只需要选读相应的平台安装步骤说明进行VTK的编译安装。VTK不支持早期版本的Windows系统，如Windows 3.1等，同样也不支持版本老于OSX 10.2 (Jaguar)
的Macintosh操作系统。

2.2 CMake

CMake是一个开源的、跨平台的用于配置、管理工程构建过程的工具。简单地说，平台独立文件（CMakeLists.txt）描述了构建工程的一些细节以及所要依赖的库文件等。CMake运行时，它会根据所运行的操作系统以及编译器等环境创建本地编译文件。比如，在Windows平台下，使用Microsoft Visual Studio集成开发环境，CMake会自动生成项目（Solution）文件和工程（Project）文件。而在UNIX平台下，则是创建了makefile文件。通过这种方式，可以非常容易地在任何计算机平台下编译VTK，以及使用各种开发工具（如Editors,
Debuggers, Profiles, Compilers等）。想进一步了解更多关于CMake的内容可以访问http://www.cmake.org，或者参考Kitware公司出版的《Mastering
CMake》一书。最新版本的CMake可以从CMake网站下载。

运行CMake时，需要指定三方面的信息：使用何种编译器，源文件目录的路径和编译目录的路径。编译目录是指编译过程中产生的目标代码、库文件以及二进制文件等存放的位置。CMake会从源文件目录中读取最顶层的CMakeLists.txt文件，接着在编译目录中生成缓存文件CMakeCache.txt。值得注意的是，CMake可以很好地处理复杂的源文件目录结构，源文件目录树下的每个子目录里可以都放有一个CMakeLists.txt文件。

一旦这些基本的信息都设置好，用户就可以开始"配置"（Configure）。CMake会在这一步读取最顶层的CMakeLists.txt文件，以确定系统配置，定位系统资源以及遍历源文件目录中的每一层子目录。CMake运行时，会出现很多控制工程构建过程的CMake变量和标志（称为CMake的缓存条目"CMake Cache Entries"）。"配置"完成以后，这些条目会显示在CMake的GUI界面上，条目的显示风格可以通过CMake GUI上的视图选项进行切换。当改变某个条目的值时，要重新进行"配置"。直到所有的CMake条目都配置完成以后，就可以进入下一步，即"生成"（Generate）。"生成"阶段会根据指定的编译器生成项目文件、工程文件或者Makefiles文件。

接下来的两部分内容（Windows和UNIX）详细介绍了在不同平台下CMake的运行步骤。注意，上面提到的步骤对所有的平台都是适用的。CMake界面可能会依据不同的操作系统有所不同。但是CMake-gui从CMake 2.6开始及后续版本都是基于Qt的，所以在不同的平台下CMake界面都是大同小异的。建议选择安装CMake的预编译二进制文件，无须从CMake源文件中编译安装，除非你想要研究CMake的源代码。

第02章-VTK安装（2）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本节对应原书中的第10页至第18页】

2.3Windows XP, Vista及以上版本平台下安装VTK

Windows平台下有两种安装VTK的方式。第一种是二进制/可执行文件安装【译者：VTK的二进制安装文件可以从http://vtkchina.org/bbs/forum.php上下载，这个安装文件的VTK版本是5.0的，比较老，所以不建议用这种傻瓜式的安装方法安装VTK，同时建议安装本书后续内容提到的vtk-5.6.1-win32.exe文件。安装这个文件以后，可以在自己的计算机运行*.tcl的文件。另外也可以从http://vtkchina.org/bbs/forum.php下载到VTK-5.10.1的安装版本】，通过这种安装方式可以捆绑安装Tcl等工具；第二种方式是VTK源代码编译安装，通过编译生成C++库文件以及VTK封装代码（生成Java,Tcl和Python等的可执行文件）。两种安装方式相比较而言，二进制安装方式要简单得多，但源文件编译安装方式则更具优势，它可以跟踪、调试以及修改VTK的源代码，如果你是开发人员的话，这种安装方式将会非常适合你。当然，如果选择二进制方式安装VTK的话，同样可以通过不同的方式扩展VTK：创建自定义类（见"编写VTK的类：概览"一节），使用运行时可编程过滤器（Run-time
Programmable Filters）（见"可编程过滤器"一节）以及运行时使用自定义的VTK类取代VTK原有的类（见"对象工厂"一节）等方式同样可以扩展VTK。

二进制安装

安装vtk.exe（用于执行Tcl/Tk文件的可执行程序）。可从VTK官网下载，一般是vtk-X.X.X-win32.exe文件，如vtk-5.6.1-win32.exe，安装界面如图2-1所示。安装文件名中的"X.X.X"表示安装的VTK版本号。同时还可以下载相应版本号的VTK源代码的文件*.zip以及VTK数据文件VTKData。每当发布新版本的VTK时，我们会在VTK的官网上给出链接以供下载，用户可以关注vtkusers邮件列表上有关VTK版本发布的公告。

图2-1 Windows下vtk.exe程序安装界面

VTK源代码目录中包含很多*.tcl的脚本文件，借助这些文件可以了解VTK里的类是如何协同工作的。把下载到的vtk-X.X.X.zip和vtkdata-X.X.X.zip文件解压到硬盘中，在VTK源代码文件夹下，可以找到"Examples"文件夹，文件夹"Examples"里又有类似"GUI"，"MangleMesa"以及"Parallel"的文件夹，每个文件夹的都含有一个名为"Tcl"的文件夹，里面包含了大量的Tcl例子。除了"Examples"文件夹，还有大量的库文件夹【译者："库文件夹"是指VTK以这些文件夹的名字生成同名的库文件，如vtkGraphics.lib,vtkImaging.lib等】，像Graphics,
Imaging和Filtering。每个目录都会包含一个Testing/Tcl子目录，里面有对VTK逆行测试的例子，双击这些Tcl文件即可运行。初次双击后缀为.tcl的文件时，会弹出对话框，询问你用什么程序打开文件，这意味着你要把Tcl文件与vtk.exe可执行文件做一下关联，也就是在"打开为…"对话框中选择"浏览"，定位到VTK安装目录下，通常是"C:\ProgramFiles\VTK 5.6"或"C:\Program Files\Documents andSettings\<username>\My Documents\VTK
5.6"，然后选择bin目录，选择可执行文件vtk.exe，可以把"打开为…"对话框中的"使用此程序打开此种类型的文件"复选框选上，再点击"确定"按钮，以后双击.tcl后缀的文件就可以用VTK自动运行Tcl脚本文件了。

另外，如果你已经安装了Tcl/Tk程序的话，*.tcl的文件会自动与Tcl/Tk程序进行关联，这时你再双击*.tcl文件时要出现如下的错误提示：can't find package vtkwhile executing "package require vtk"。可以用上一段介绍的方法，把*.tcl文件与vtk.exe程序进行关联。

以上就是在Windows平台下安装VTK二进制文件的全部过程，在第三章里我们会详细介绍如何书写自己的C++类，以及Tcl, Java和Python应用程序。

源代码安装

如果准备用VTK开发C++应用程序以及扩展VTK的话，就需要采用源代码安装方式。这种安装方式可能是一件比较有挑战的事情，它可能需要占用你的计算机不少时间。首先要

确保你的机器是否适合编译所发布的VTK源代码，同时机器上也应该安装了Windows XP, Vista或以上版本的操作系统，另外必须安装了C++编译器，这个指南是以MicrosoftVisual Studio 2008【译者：原书是采用Microsoft VisualStudio 2005，我们在翻译时，将所有例子程序都在Microsoft VisualStudio 2008上做了测试，所以该翻译版本是基于Microsoft VisualStudio2008的。】及更高版本的编译器做测试的，VTK可以很好的用这些版本的编译器进行编译。同样也支持Borland C++编译器、Cygwin下的gcc编译器或MinGW，NMake，MicrosoftVisual C++免费版本以及Microsoft Visual C++ 2005等。如果机器上还没安装任何编译器的话，先安装以上所说的编译器之一【译者：推荐安装Microsoft VisualStudio 2008】。

接下来要考虑的是需要使用其他什么工具。假如准备用Java开发程序的话，必须下载安装Java JDK（可在Sun Microsystems:http://www.java.sun.com上下载）。如果打算用Tcl/Tk做开发，则须下载安装Tcl/Tk（源代码可从http://www.tcl.tk下载，或者从http://www.activestate.com/Products/ActiveTcl下载Tcl/Tk的二进制安装文件）。注意，Tcl/Tk 8.4版本可以与VTK 5.4.0版本协同工作。

安装CMake。要编译VTK，首先你必须先安装CMake。CMake的安装程序可以从http://www.cmake.org下载。

运行CMake。安装完C++编译器、CMake以及所需的其他工具后（如Tcl,Java, Python），就可以运行CMake程序。打开CMake程序，界面如图2-2所示。CMake会根据不同的编译器以及操作系统生成VTK项目文件。在“Where is the sourcecode”一栏中指定VTK源码所在的目录（也就是压缩包vtk-5.6.1.zip解压后所在的目录），在“Whereto build the binaries”一栏指定VTK编译过程中生成的文件的目录【译者：源文件目录和编译目录最好分开，可以建立一个空的文件夹，用于存放编译过程中生成的文件。】，这两个路径的设置可以通过右边的“Browse Source…”和“BrowseBuild…”按钮进行选择，或者手动输入。设置完这两个路径以后，就可以按“configure”按钮。初次配置时，会弹出对话框让用户选择编译器，根据机器已安装的编译器选择对应的编译器即可。接着CMake会自动填充变量列表的值，第一次运行configure以后，所有的变量条目会以红色显示，红色标示着缓存条目是自动生成的或者在上一次configure时值被作了修改。

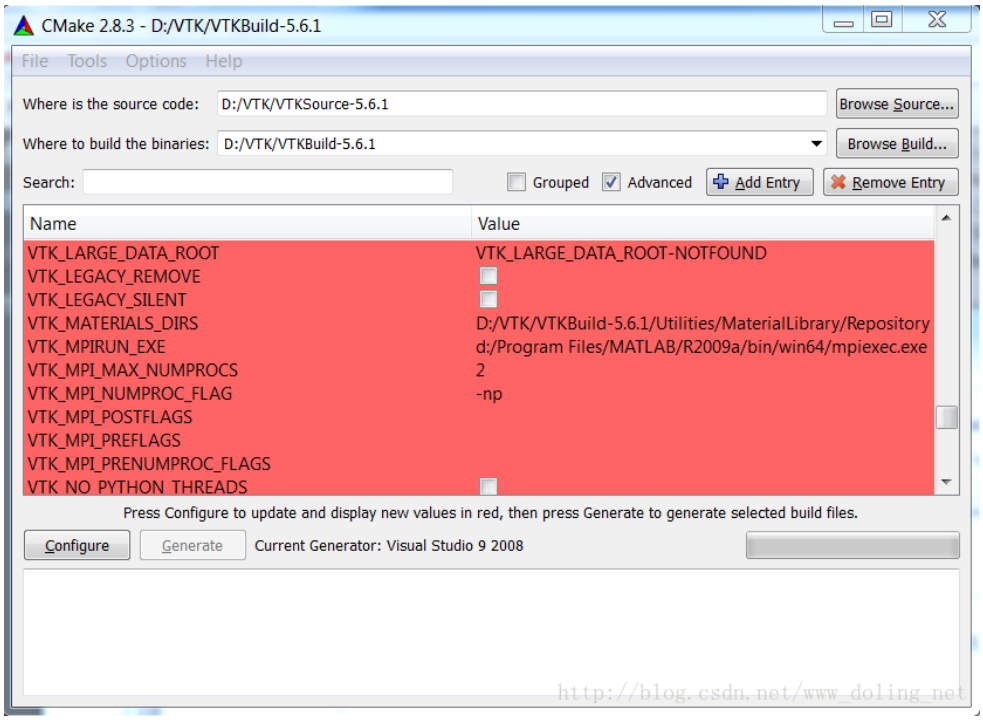

图2-2CMake会根据不同的编译器、操作系统生成工程文件、Makefiles等，CMake是跨平台的，图中所示为CMake的图形用户界面。

Configure结束后就可以定制VTK编译的选项。比如，如果想激活VTK的Tcl封装特性，就必须把VTK_WRAP_TCL这一项勾选上（设置其值为ON）。设置完这些选项的值以后，再次按Configure按钮，直到所有的选项都变成了灰色。如果安装了Tcl/Tk程序，关于Tcl/Tk的一些选项还是显示NOTFOUND的话，就必须手动设置它们的值。可点击每个条目最右边的按钮进行设置，根据各个条目类型的不同，条目编辑的交互形式也各不相同，有文件选择对话框、文本框、下拉菜单等。通过这些交互方式，可对条目的值进行修改。

下面列出VTK中一些重要的变量值：

l BUILD_SHARED_LIBS – 如果设置成yes（即勾上这个选项），VTK会编译成DLL动态库或者说共享库，如果设置成no，则编译成静态库。缺省值是no，即静态编译。某种程度上静态库可能比较好使用，因为在可执行文件运行时，不要求这些库文件在当前目录下或系统Path环境变量下，可执行文件自身会包含相关的库文件。这对于发布基于VTK的应用程序是比较有利的。

l VTK_WRAP_TCL – 这个选项决定是否编译Tcl的封装特性。

l VTK_WRAP_PYTHON – 决定是否编译Python的封装特性。

l VTK_WRAP_J***A – 决定是否编译Java的封装特性。

要获取关于这些选项的在线帮助，可以右击CMake上的变量，然后选择“Helpfor Cache Entry”，大多数默认的设置应该都是正确的。

设置完相关选项的值之后，再次按Configure按钮，直到所有的选项都变成了灰色。然后点击“Generate”（有些CMake版本可能是“OK”按钮），CMake会根据选择的编译器类型自动生成对应的工程文件。如果是Microsoft Visual Studio，就会在“Whereto build the binary”指定的目录下生成一些工程文件（Visual Studio 6.0是vtk.dsw文件，VS2005或更高版本的是vtk.sln文件）。在VisualStudio环境中加载工程文件后，选择编译版本（Debug,

Release,MinSizeRel, RelWithDebInfo等），然后选择ALL_BUILD工程，开始编译工程。如果是使用Borland编译器，生成的是Makefiles文件，则必须以命令行的形式给编译器指定合适的参数。Makefiles文件也是位于“Whereto build the binary”所指定的目录下。

当所有的VTK工程都编译完成后，所有的库文件以及可执行文件会出现在CMake里指定的编译目录下的bin子目录中（除非你修改了CMake选项EXECUTABLE_OUTPUT_PATH或LIBRARY_OUTPUT_PATH选项的值）。

注意：不要使用MSVC++中的“RebuildAll”菜单来编译源代码。需要重新生成每样内容的话，先清空VTK的编译目录，重新CMake，按以上的步骤重新生成工程文件，然后再编译。

如果选项BUILD_SHARED_LIBS的值设为ON的话，VTK程序运行时需要加载相应的DLL文件。这有不同的方法可以让VTK程序运行时加载DLL库文件，每种方法都有利有弊。最简单的方法就是确保VTK可执行程序和需要加载的VTK DLL库文件在同一个目录里，可以复制所有的VTK DLL库文件到应用程序所在的目录中，或者在CMakeLists文件中把应用程序的编译目录跟EXECUTABLE_OUTPUT_PATH的路径设置成一样的。一旦所有的VTK DLL文件和所执行的应用程序都位于同一目录，一切就会正常运行。这种做法的好处就是简单，不好的地方就是一旦更新了VTK DLL文件以后，就无法更新先前复制过去的VTK DLL文件，除非把重新编译的VTK DLL文件再复制一遍；而且一旦应用程序的编译目录与VTK的DLL文件混在一起的时候，可能很难确定哪个输出来自哪个项目。

另一种方法就是改变PATH环境变量，这样就算应用程序与VTK DLL库文件不在同一个目录也能正确找到所要加载的动态链接库。PATH环境变量的设置也有两种方法，第一种方法是在命令行窗口中通过set命令临时改变PATH环境变量，VTK应用程序的运行也是在同一个命令行窗口中；或者是直接改变系统的PATH环境变量，将包含VTKDLL文件的路径添加到PATH环境变量中。除非你需要在同一台计算机上运行两种不同版本的VTK动态库文件，否则建议用后一种方法【译者：也就是将VTK DLL文件所在的路径放在环境变量PATH里，右击“我的电脑”->“属性”->“高级”->“环境变量”进行设置。】设置PATH环境变量。

KWWidgets提供了一个使用批处理脚本文件设置PATH变量（不用去修改PATH系统环境变量）的很好的例子。可参考KWWidgets目录下的KWWidgetsSetupPaths.bat脚本文件，KWWidgets源代码文件可从http://www.kwwidgets.org下载。

如果想了解更多关于Windows下DLL文件加载方面的内容，可以参考Windows SDK文档里的LoadLibrary和LoadLibraryEx函数。

至此，就成功地在计算机上安装了VTK，因为VTK的复杂性以及大量的源代码文件，可能使得安装编译过程显得有一定的挑战性。请仔细阅读我们在这部分中所介绍的安装说明，如果在这个过程中遇到了什么问题，可以在vtkusers邮件列表上提问（参考第一章“其他资源”一节）以获取帮助或者从Kitware公司获取商业支持。

2.4UNIX平台下安装VTK

UNIX系统存在众多的版本，因此，我们不得不通过编译VTK源代码来生成相关的库文件和可执行程序。

源代码安装

这部分内容将带领你过一遍在UNIX平台下编译VTK的步骤。与Windows平台不同，我们很难获取到针对UNIX平台编译好的预编译二进制文件和可执行程序，因此我们必须自己编译VTK。（注意：查找一下vtkusers邮件列表及第一章“其他资源”一节介绍的资源，有些用户已经在Web上提供了编译好的二进制文件）。一般来说，这个过程还是相当简单的，根据机器的配置情况VTK的编译安装过程可能会花上一到四个小时【原书是说需要1到4小时，但是对于目前的计算机配置而言，VTK的编译时间不需这么长时间。】。对于高端、大内存、多处理器的机器来说，使用并行编译可以在十分钟内编译完C++和Tcl的二进制文件和可执行程序。绝大部分的时间都花在等待计算机编译源代码上。只需花费你10到30分钟的时间，第一步是确认是否有编译VTK所须的资源。为保险起见，最好空出300M以上的磁盘空间。对某些系统，如SGI，还得留出更多的空间，特别是如果你要编译VTK的调试版本。由于VTK是使用C++编写的，所以还需要C++编译器。一般的C++编译器如CC、g++或者ACC等。

如果想在Tcl/Tk、Python或者Java中使用VTK，还需要先下载并安装这些工具。Java JDK可以从Sun Microsystemshttp://ww.java.sun.com获得；如果打算使用Tcl/Tk，需要下载安装Tcl/Tk，可以从http://www.tcl.tk获得；Python可以从http://www.python.org下载。下面开始介绍如何在UNIX平台下编译VTK。

CMake

与Windows环境类似，在UNIX系统下编译VTK也需要借助CMake（见上一节关于CMake介绍部分）。CMake官网（http://www.cmake.org）上有用于不同版本UNIX系统的预编译二进制文件，可以直接下载使用，如果网站上没有提供的版本，就只有下载CMake的源代码，自行编译了。

安装CMake。下载CMake的预编译二进制文件（tar文件），解压到目标目录下（一般是/usr/local/）。确保“cmake”以及与它关联的可执行文件都在当前路径下，或者给出完整路径运行“cmake”。

编译CMake。如果无法下载到与自己的UNIX相对应的CMake文件，就需要自行编译安装CMake了。要编译安装CMake只要把CMake的源码包（可从http://www.cmake.org下载）解压到某个目录中，然后在那个路径下运行下面的命令即可：

```
./configure

make

makeinstall
```

如果没有root权限，可以省略上面的第三个步骤（也就是make install）。CMake可执行程序位于CMake/bin目录下。CMake提供了两种不同的可执行程序文件来配置VTK：ccmake提供了一个基于交互的终端，非常类似Windows平台下的cmake-gui程序；以及需要一步步回答一系列问题来完成整个配置过程cmake向导。

还得告诉CMake采用何种C++/C编译器。在多数UNIX系统中，可以通过下面的方式设置相关的信息：

```
setenvCXX /your/c++/compiler
```

setenvCC /your/c/compiler

或者：

exportCXX=/your/c++/compiler

exportCC=/your/c/compiler

否则CMake会自动检测编译器，假如机器上安装了多个编译器，CMake检测到编译器未必就是你想使用的。完成了这些设置后，在VTK源代码的同级目录中创建一个空的文件夹（比如叫VTK-bin），然后在这个目录下运行CMake，把刚才创建的目录以及VTK的源代码目录的路径作为参数传递给CMake，即：

cdVTK-bin

ccmake../VTK

或者

cdVTK-bin

cmake–i ../VTK

接下来两小节的内容简要介绍一下ccmake和cmake–i的区别。对配置脚本比较熟悉的UNIX开发人员会注意到CMake和configure在功能上是很相似的。但是用configure生成makefile文件时需要用命令行参数进行控制，而CMake则可以通过用户交互设置编译选项来生成makefile文件。

基于用户交互终端定制编译选项（ccmake）。ccmake有一个基于用户交互的简单的终端，允许用户根据不同的机器环境定制编译选项来设置VTK的编译环境。当用ccmake运行CMake时，有一系列选项可供选择、修改，以便用户根据不同的环境设置VTK的编译环境。大多数的选项CMake都可以设置合理的缺省值。可以通过键盘上的箭头按键选择某个选项，当某个选项呈高亮显示时按回车键即可修改该选项的值，如果选中的选项类型是布尔型的，按下回车键就可以更改该选项的值。选项的值修改完成后再按回车键可以确认该选项的值。当所有选项的值都设置完成后，按“c”键即可完成整个设置过程。接着CMake就会进入处理配置文件的过程，必要时CMake还会在顶端部分显示新的选项，比如，如果打开了选项VTK_WRAP_TCL，就会出现新的选项要求设置Tcl/Tk库文件以及包含的路径。新的选项出现时，应该重新设置它们的值，假如这些新的选项有合理的缺省值时，也可以采用这些缺省值。然后再按“c”键重新配置，直到没有新的选项出现。这个过程完成以后，就会有新的可用的命令出现：“Generate”和“Exit”。此时可以按“g”键让CMake生成makefile文件或者选择退出。以后如果还想重新改变CMake里某些选项的值的话，只需根据以上介绍步骤再次运行ccmake即可。

向导模式定制编译选项（cmake –i）。在某些平台下，ccmake提供的用户交互终端可能不好使，这时可以尝试运行带“-i”选项的cmake命令来激活设置向导。向导首先会询问是否使用“高级选项”，对于大部分用户来说很少会使用到“高级选项”。接着CMake会一步步地询问每个选项的取值，对于每一个选项都会列出它们的详细描述信息以及缺省值。大多数情况下，CMake会自动填充比较合理的缺省值。当然，有些时候CMake所填充的值未必都是正确的，比如OpenGL库文件没有位于预期的位置时，用户就必须手动设置相关选项的值。另外，默认情况下CMake不会创建对Tcl、Python和Java等语言进行封装的选项。如果想支持其中某种语言的话，应该打开选项VTK_WRAP_XXX，必要时还要告诉CMake相应库文件和头文件的位置。回答完CMake所有的问题以后，CMake就会自动生成makefile文件，VTK就可以开始进行编译了。以后如果想更改某些选项的取值的话，可以根据以上步骤再运行CMake向导。

编译源代码

运行完CMake后就会生成makefile文件，然后键入make命令，开始编译VTK。有些make工具，如GNUmake（gmake）支持并行编译（gmake–j）。尽可能使用并行编译，即使在单处理器系统，因为文件的读写操作是编译过程最大的瓶颈，处理器可以同时处理多个编译进程。如果在编译过程中遇到什么问题，可以到vtkusers邮件列表(参考第一章“其他资源”一节)寻求帮助，也向Kitware公司获取商业支持。

多平台下编译VTK

如果想在多个UNIX平台下编译VTK，可以给每个平台分别复制一份完整的VTK源文件目录，然后根据以上介绍的步骤进行编译；或者可以只复制一份VTK源文件目录，然后在另外一个目录下为每个平台都生成一份目标代码、库文件以及可执行文件。后一种方法需要为每个平台创建一个空的编译目录，比如vtk-solaris。假定VTK源文件的目录跟新创建的空的编译目录处于同一层目录结构中，改变当前路径（cd命令）到这个新建的目录中，按照类似以下的命令运行CMake：

cd/yourdisk

ls（输出是VTK vtk-solaris vtk-sgi）

cdvtk-solaris

cmake– i ../vtk

或者

ccmake../VTK

就会在目录vtk-solaris中生成makefile文件。这样就可以根据上面提到的步骤编译VTK。

安装VTK

现在VTK已经编译好了，可执行文件和库文件都在编译目录的bin/子目录中。如果打算在UNIX系统中和一个以上的开发者共享这些库文件，又有root权限的话，可以运行makeinst

all命令，如果没有修改CMake配置选项中的CMAKE_INSTALL_PREFIX的值，则这个命令会把VTK安装到/usr/local中。运行makeinstall命令会复制编译和运行VTK应用程序时所需的所有文件到一个其他用户都能使用的目录下。

这就是在UNIX平台下编译和安装VTK的完整步骤，如果你想了解更多关于CMake的内容，可以登录http://www.cmake.org或者购买Kitware公司出版的图书《MasteringCMake》。第3章将会详细讲解如何运行VTK的例子以及创建自己的VTK应用程序。

【第2章 完】

第03章-VTK系统概述（1）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本节对应原书中的第19页至第25页】

本章旨在介绍VTK系统的总体概述，并讲解运用C++、Java、Tcl和Python等语言进行VTK应用程序开发时所需掌握的基本知识。首先我们从VTK系统的基本概念和对象模型抽象开始进行介绍，并在本章最后通过例子演示这些概念以及介绍一下在构建VTK工程时所需要掌握的知识。

3.1系统结构

VTK系统由两个子系统组成：一个是编译的C++类库，另一个是解释性语言的封装层，以供Java、Tcl和Python等语言来操作该C++类库，系统结构如图3-1所示。

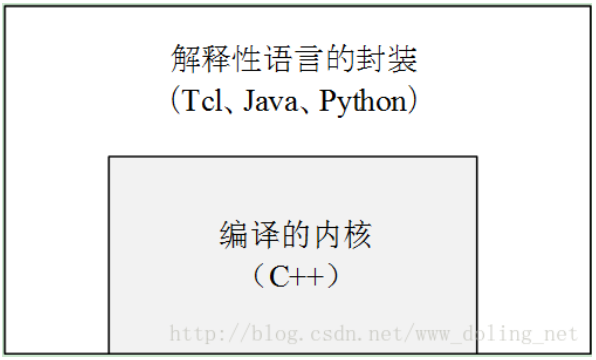

图3-1VTK系统由C++类库内核以及解释性语言（Java、Tcl、Python）封装层所组成。
该结构的优点在于保持解释性语言快速开发特性（避免编译、链接流程，工具简单而强大，同时又易于使用的GUI工具）的同时，用户可以利用编译的C++语言开发高效的算法（无论是CPU利用还是内存的利用）。当然，对于熟练掌握C++语言或者使用相应开发工具的用户来说，VTK应用程序可以完全采用C++语言进行开发。

由于VTK系统采用了面向对象的思想，因此利用VTK进行高效开发的关键在于深入理解系统内部的对象模型，以便消除用户在使用大量的VTK系统对象时的迷惑，并能更有效的对这些对象进行组合创建应用程序。此外，还需要了解系统中许多对象的功能，而这只能通过阅读例子代码和在线文档来获得。这本《VTK用户指南》中，我们将尽力介绍一些有用的VTK对象的组合以便用户可以将其应用到自己的应用程序中。

本章接下来将介绍VTK中的两个重要组件：可视化管线【译者：Visualization Pipeline，有些翻译成"可视化管道"，本书我们翻译成"可视化管线"】和渲染引擎。可视化管线主要负责数据获取或者创建，数据处理，然后将数据写入文件或者传递至渲染引擎中进行显示。而渲染引擎负责创建传递过来的数据的一个可视表达。注意这里所提到的组件并非VTK系统结构中具体的组件，而是一个抽象的概念组件。本章是在比较高的层次上进行阐述，但是当你将本章的内容和下一章节中具体的例子或者是VTK源文件中所提供的大量的示例程序结合在一起时，你将会对本章所介绍的概念有更深刻的理解。

底层对象模型

vtkObject是VTK对象继承关系树的根结点，几乎所有的VTK对象都继承自该类，除了一部分特殊对象继承自vtkObject的父类vtkObjectBase。所有的VTK对象都必须由对象的New()方法创建，并由对象的Delete()方法销毁。由于VTK对象中的构造函数被声明为受保护类型，因此VTK对象不能够在栈中分配空间。另外，VTK对象采用了共同基类和统一的创建、销毁方法，可以实现许多基本的面向对象操作。

引用计数。VTK对象内部显式地记录了引用本身的指针的个数。当一个对象通过静态的New()函数创建时，该对象内部的初始引用计数即为1，因为所创建的对象会有一个原始指针（Raw Pointer）指向该对象。

vtkObjectBase*obj = vtkExampleClass::New();

当指向某个对象的其他对象创建或者销毁时，引用计数会通过Register()和Unregister()函数进行相应的增加和减少。通常情况下系统会通过对象的API函数"Set()"自动完成。

otherObject->SetExample(obj);

这时对象obj的引用计数值为2。因为除了原始指针之外，还有另外一个对象otherObject内部的指针指向它。当原始存储对象的指针不再需要时，通过Delete()函数可以删除该引用，即：

obj->Delete();

这样再利用原始指针去访问该对象时将不再是安全的，因为该指针已经不再拥有对象的引用。因此为了保证对对象引用的有效管理，每次调用New()后都要进行Delete()，确保没有泄漏引用。

另外一种避免引用泄漏的方法是通过类模板vtkSmartPointer<>提供的智能指针来简化对象的管理操作，上述例子可以重写为如下代码：

vtkSmartPointer<vtkObjectBase>obj = vtkSmartPointer<vtkObjectBase>::New();

otherOject->SetExample(obj);

该例中，智能指针会自动管理对象的引用。当智能指针变量超出其作用域并且不再使用时，例如当其为函数内部局部变量，函数返回时，智能指针会自动通知对象减少引用计数。由于智能指针提供了内部静态New()函数，因此不需要原始指针来保存对象的引用，因此也不需要再调用Delete()函数。

运行时类型信息。在C++中对象的实际类型可能与该对象的索引指针的类型不一致。VTK中所有的类都有一个接口函数提供类名标识符，因此一个string类型足以标识这些类。运行时对象的实际类型可以通过GetClassName()函数获得：

constchar* type = obj->GetClassName();

通过IsA()函数可以判断一个对象是否是某一个类的实例，或者是该类的某个子类的实例：

if(obj->IsA("vtkExampleClass")) { … }

父类类型的指针可以通过静态函数SafeDownCast()安全地强制转换为子类类型的指针：

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

上述代码只有当对象example是obj的子类实例时，类型转换才成功；否则返回空指针。

对象状态显示。容易理解的对象的当前状态信息对于程序的调试是十分有用的。VTK对象的状态可以通过输出函数Print()获得：

obj->Print(cout);

渲染引擎

组成VTK渲染引擎的类主要负责接收可视化管线（Visualization Pipeline）的输出数据并将结果渲染到窗口中。该过程主要涉及到下述一些组件。注意，这些只是VTK渲染系统中比较常用的组件，并非全部。而每个子标题仅仅是代表一个对象类型的最高层VTK超类【译者：或者称为"父类"】，在许多情况下，这些超类只是定义了基本API函数的抽象类，而真正的实现则由其子类来完成。

vtkProp。渲染场景中数据的可视表达（Visible Depictions）是由vtkProp的子类负责。三维空间中渲染对象最常用的vtkProp子类是vtkActor和vtkVolume，其中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的子类负责确定场景中对象的位置、大小和方向信息。控制Prop【译者：Prop,Actor,
Mapper, Property等词，本书不作翻译。】位置信息的参数依赖于对象是否在渲染场景中，比如一个三维物体或者二维注释，它们的位置信息控制方式是有所区别的。三维的Prop如vtkActor和vtkVolume（vtkActor和vtkVolume都是vtkProp3D的子类，而vtkProp3D继承自vtkProp），既可以直接控制对象的位置、方向和放缩信息，也可以通过一个4×4的变换矩阵来实现。而对于二维注释功能的Props如vtkScalarBarActor，其大小和位置有许多的定义方式，其中包括指定相对于视口的位置、宽度和高度。Prop除了提供对象的位置信息控制之外，Prop内部通常还有两个对象，一个是Mapper对象，负责存放数据和渲染信息，另一个是Property（属性）对象，负责控制颜色、不透明度等参数。

VTK中定义了大量的功能细化的Prop（超过50个），如vtkImageActor（负责图像显示）和vtkPieChartActor（用于创建数组数据的饼图可视表示）。其中的有些Props内部直接包括了控制显示的参数和待渲染数据的索引，因此并不需要额外的Property和Mapper对象。vtkActor的子类vtkFollower可以自动的更新方向信息保持自身始终面向一个特定的相机，这样无论如何旋转渲染场景中的对象，vtkFellower对象都是可见的，适用于三维场景中的广告板（Billboards）或者是文本。vtkActor的子类vtkLodActor可以自动改变自身的几何表示来实现所要求的交互帧率，vtkProp3D的子类vtkLODProp3D则是通过从许多Mapper中进行选择来实现不同的交互性（可以是Volumetric
Mapper和GeometricMapper的集合）。vtkAssembly建立了Actor的等级结构以便在整个结构平移、旋转或者放缩时能够更合理的控制变换。

vtkAbsractMapper。许多Props如vtkActor和vtkVolume利用vtkAbstractMapper的子类来保存输入数据的引用以及提供真正的渲染功能。vtkPolyDataMapper是渲染多边形几何数据主要的Mapper类。而对于体数据，VTK提供了多种渲染技术。例如，vtkFixedPointVolumeRayCastMapper用来渲染vtkImageData类型的数据，vtkProjectedTetrahedraMapper则是用来渲染vtkUnstructuredGrid类型的数据。

vtkProperty和vtkVolumeProperty。某些Props采用单独的属性对象来存储控制数据外观显示的参数，这样不同的对象可以轻松的实现外观参数的共享。vtkActor利用vtkProperty对象存储外观（属性）参数，如颜色、不透明度、材质的环境光（Ambient）系数、散射光（Diffuse）系数和反射光（Specular）系数等。而vtkVolume则是采用vtkVolumeProperty对象来获取体对象的绘制参数，如将标量值映射为颜色和不透明度的传输函数（Transfer
Function）【译者：也有译成"传递函数"】。另外，一些vtkMapper提供相应的函数设置裁剪面以便显示对象的内部结构。

vtkCamera。vtkCamera存储了场景中的摄像机参数，换言之，如何来"看"渲染场景里的对象，主要参数是摄像机的位置、焦点、和场景中的上方向向量。其他参数可以控制视图变换，如平行投影或者透视投影，图像的尺度或者视角，以及视景体的远近裁剪平面等。

vtkLight。vtkLight对象主要用于场景中的光照计算。vtkLight对象中存储了光源的位置和方向，以及颜色和强度等。另外，还需要一个类型来描述光源相对于摄像机的运动。例如，HeadLight始终位于摄像机处，并照向焦点方向；而SceneLight则始终固定在场景中的某个位置。

vtkRenderer。组成场景的对象包括Prop，Camara和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，而这些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。

vtkRendererWindow。vtkRendererWindow将操作系统与VTK渲染引擎连接到一起。不同平台下的vtkRendererWindow子类负责本地计算机系统中窗口创建和渲染过程的管理。

当使用VTK开发应用程序时，只需要使用平台无关的vtkRendererWindow类，程序运行时，系统会自动替换为平台相关的vtkRendererWindow子类。vtkRendererWindow中包含了vtkRenderer的集合，以及控制渲染的参数，如立体显示（Stereo）、反走样、运动模糊（Motion Blur）和焦点深度（FocalDepth）。

vtkRenderWindowInteractor。vtkRenderWindowInteractor负责监听鼠标、键盘和时钟消息，并通过VTK中的Command/Observer设计模式进行相应的处理。vtkInteractorStyle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。vtkRenderWindowInteractor自动建立一个默认的3D场景交互器样式（InteractorStyle），当然你也可以选择一个二维图像浏览的交互器样式，或者是创建自定义的交互器样式。

vtkTransform。场景中的许多对象，如Prop、光源Light、照相机Camera等都需要在场景中合理的放置，它们通过vtkTransform参数可以方便的控制对象的位置和方向。vtkTransform能够描述三维空间中的线性坐标变换，其内部表示为一个4×4的齐次变换矩阵。vtkTransform对象初始化为一个单位矩阵，你可以通过管线连接的方式将变换进行组合来完成复杂的变换。管线方式能够确保当其中任一个变换被修改时，其后续的变换都会相应的进行更新。

vtkLookupTable，vtkColorTransferFunction和vtkPiecewiseFunction。标量数据可视化经常需要定义一个标量数据到颜色和不透明度的映射。在几何面绘制中用不透明度定义表面的透明程度，而体绘制中不透明度表示光线穿透物体时不透明度沿着光线的累积效果，两者都需要定义不透明度的映射。对于几何渲染可以使用vtkLookupTable来创建映射，体绘制中需要使用vtkColorTransferFunction和vtkPiecewiseFunction来建立映射。

一个简单的例子。下面的例子（摘自./VTK/Examples/Rendering/CXX/Cylinder.cxx）演示了怎样利用上述对象来指定和渲染场景。

```
vtkCylinderSource*cylinder = vtkCylinderSource::New();

vtkPolyDataMapper*cylinderMapper = vtkPolyDataMapper::New();
cylinderMapper->SetInputConnection(cylinder->GetOutputPort());

vtkActor*cylinderActor = vtkActor::New();
cylinderActor->SetMapper(cylinderMapper);

vtkRenderer*ren1 = vtkRenderer::New();
ren1->AddActor(cylinderActor);

vtkRenderWindow*renWin = vtkRenderWindow::New();
renWin->AddRenderer(ren1);

vtkRenderWindowInteractor*iren = vtkRenderWindowInteractor::New();
iren->SetRenderWindow(renWin);

renWin->Render();
iren->Start();
```

例子中我们直接创建了一个vtkActor，vtkPolyDataMapper，vtkRenderer，vtkRendererWindow和vtkRendererWindowInteractor。注意，vtkProperty会由vtkActor自动创建，而vtkLight和vtkCamera会由vtkRenderer自动创建。

第03章-VTK系统概述（2）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本节对应原书中的第25页至第29页】

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。例如，你可能从硬盘中读取一个3D体数据，经过处理生成体数据中一个等值面的三角面片的表示数据，然后将该几何数据写回到硬盘中。或者你可能创建了一些球体和圆柱用来表示原子和原子间的联系，然后传递到渲染引擎中显示。

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。

l vtkDataObject

l vtkAlgorithm

数据对象表示了不同类型的数据。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（数据集，vtkDataSet类）。图3-2显示了VTK中支持的DataSet对象。如图所示Dataset中包含几何结构和拓扑结构数据（点和Cell单元）；另外还有相应的属性数据，如标量或者向量数据。Dataset中的属性数据既可以关联到点，也可以关联到单元上。单元是点的拓扑组合，是构成Dataset结构的基本单位，常用来进行插值计算。图19-20和图19-21显示了VTK中支持的23种最常用单元类型。图3-3显示了VTK支持的属性数据。

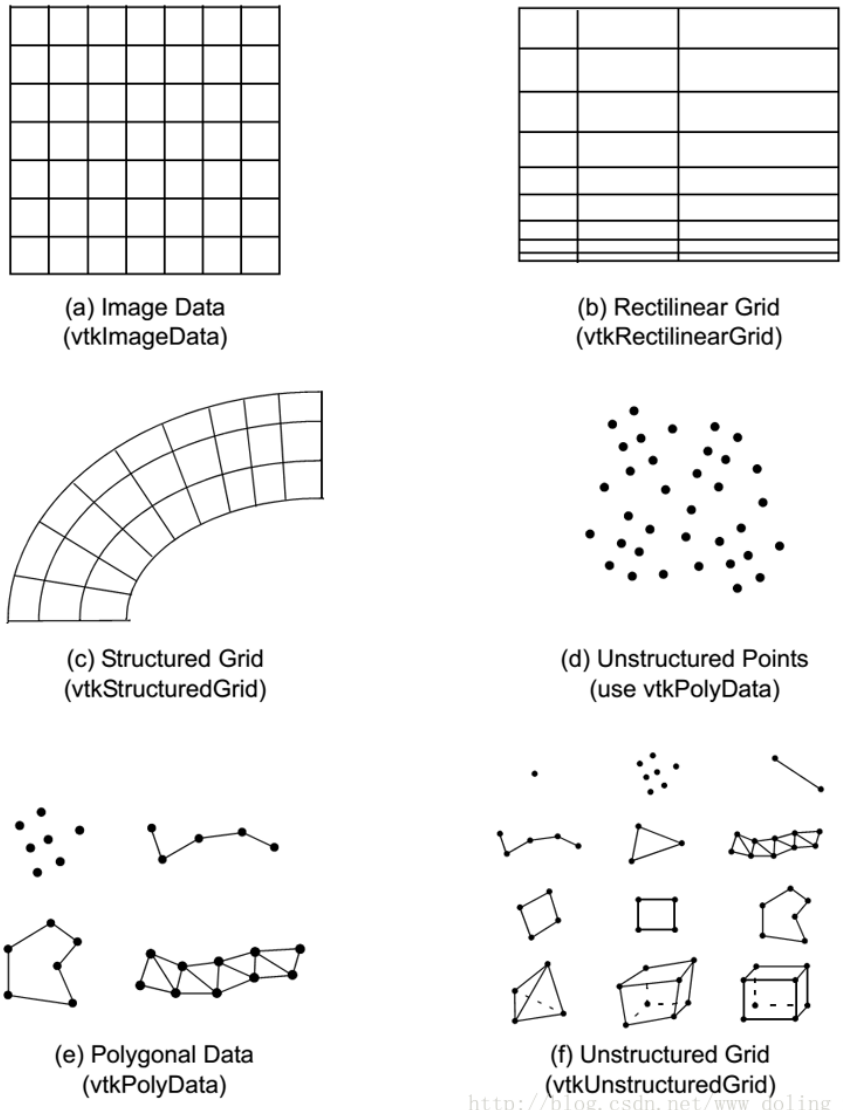

图3-2 VTK中的数据集类型。注意无结构点集（Unstructured Points）可以用多边形数据（Polygonal Data）或者无结构网格（Unstructured Grids）表示，因此VTK中没有该数据结构类型。

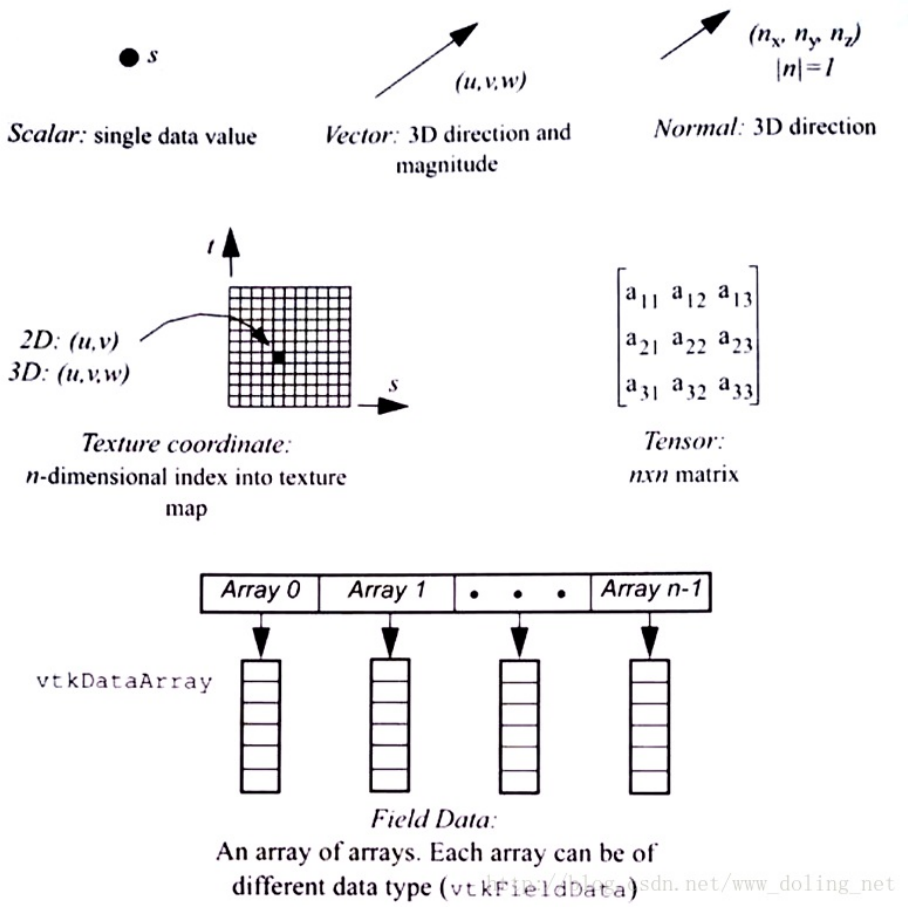

图3-3与数据集里的点和单元数据相关联的属性数据

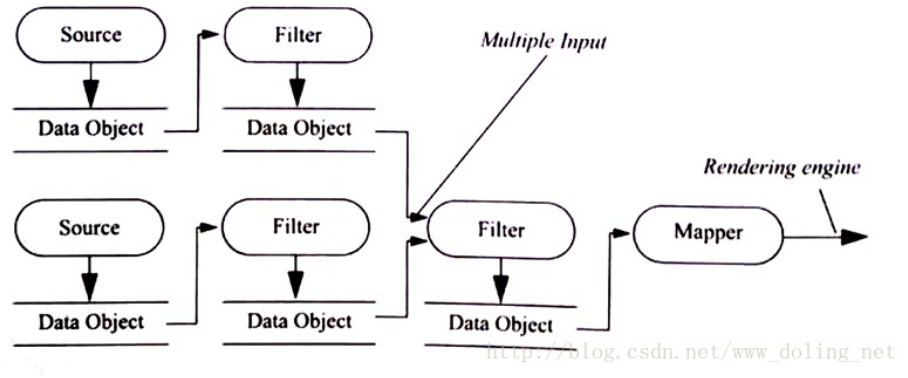

图3-4数据对象用算法（过滤器）连接形成可视化管线，图中箭头表示数据流动方向。
算法（Algorithms）常被称为过滤器（Filter），处理输入数据并产生新的数据对象。可视化管线是由算法和数据对象连接而成（例如，数据流网络）。图3-4描述了一个可视化管线。

图3-4和3-5中说明了一些重要的可视化概念。源算法通过读取（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。一个或者多个数据对象传入过滤器后，经过处理产生新的数据对象。Mappers（或者特殊情况下专门的Actors）接收数据并将其转换为可被渲染引擎绘制的可视化表达。Writer也可以看做是将数据写入文件或者流的Mapper类型。

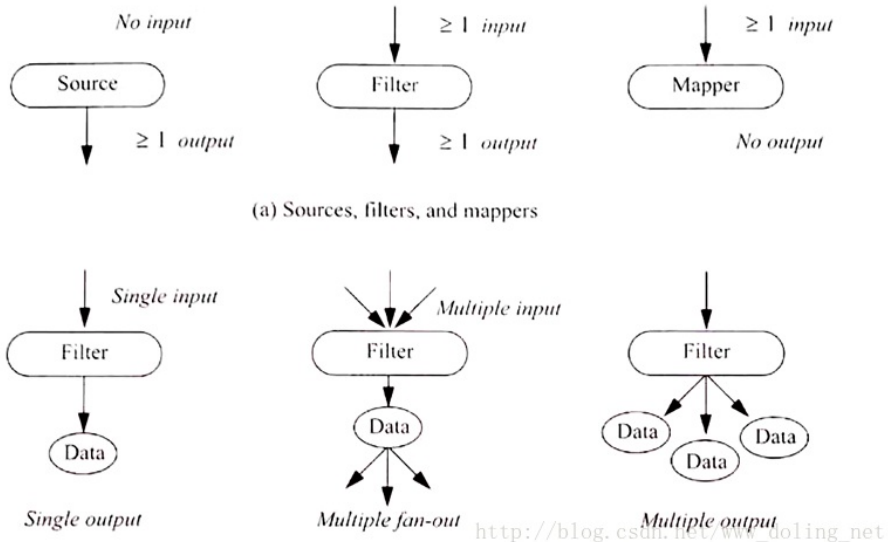

图3-5不同类型的算法，Filter可以处理一个或多个输入数据对象，并产生一个或多个输出。
接下来主要介绍可视化管线构建相关的几个重要问题。首先，可视化管线通过如下函数或者其变形来连接构建的：

aFilter->SetInputConnection(anotherFilter->GetOutputPort());

该函数将anotherFilter的输出作为aFilter的输入（有多个输入输出的Filters也有类似的方法）。第二，我们需要一种机制来控制管线的执行。我们只想执行管线中必要的部分来产生最新的输出。VTK采用了一种“惰性计算策略”（Lazy Evaluation Scheme）——只有需要数据的时候才进行计算，该机制是基于每个对象的内部修改时间（Modification Time）来实现的。

第三，管线的装配要求只有相互兼容的对象才能组装，使用的接口是SetInputConnection()和GetOutputPort()函数。如果运行时数据对象不兼容，则会产生错误。最后，当管线执行后，我们必须决定是否缓存或者保留数据对象。这对于一个成功的可视化工具应用程序来说十分重要，因此可视化数据集会非常的大。VTK提供了打开或者关闭缓存的方法，利用引用计数来避免数据拷贝，以及流数据分片方法来处理内存不能一次性容纳整个数据集的情况。（我们推荐您阅读《The VisualizationToolkit An Object-Oriented Approach to 3D Graphics》一书中VisualizationPipeline章节来获取更多相关信息。）

注意，算法（Algorithm）和数据对象（DataObject）都有许多不同的类型。图16-2中列出了当前版本VTK支持的最常用的数据对象类型。算法随着输入数据类型、输出数据类型的变化而变化，当然还有特定的算法实现。

管线的执行

前面章节我们讨论了管线执行控制的必要性。接下来，我们深入理解一些关于管线执行的重要概念。

如前所讲，VTK可视化管线只有当计算需要时才会执行（Lazy Evaluation 惰性计算）。思考一下下面的例子，该例子中我们初始化一个reader对象并且查询对象中点的个数。（示例采用Tcl语言编写）

```
vtkPlot3DReader reader
   reader SetXYZFileName$VTK_DATA_ROOT/Data/combxyz.bin
   [ reader GetOutput ] GetNumberOfPoints
```

即便数据文件中有上千个点，GetNumberOfPoints()函数返回为”0”。但是，当你添加Update()方法后，即：

```
readerUpdate
[reader GetOutput ] GetNumberOfPoints
```

函数会返回正确的点个数。前面例子中GetNumberOfPoints()并没有要求计算，因此返回当前的点个数：0。而在第二个例子中，Update()函数驱动管线执行，从而驱动reader执行，读取数据文件中的数据。一旦reader执行后，点的个数就被正确的赋值。

通常情况下，你不必手动调用Update()函数，因为filter在可视化管线中是连接在一起的。当Actor接收到渲染自己的请求（即Render()）后，它将向前传递Update()方法至相应的Mapper，Update()方法通过管线自动发出。图3-6显示了一个高层的可视化管线的执行流程。如图所示，Render()发出数据请求，通过管线向上传递。对于管线中已经过期的环境，filters会重新执行，从而得到最新的终端数据，并被Actor绘制。（详细的执行过程信息，请参考第15章ManagingPipeline Execution，第317页）

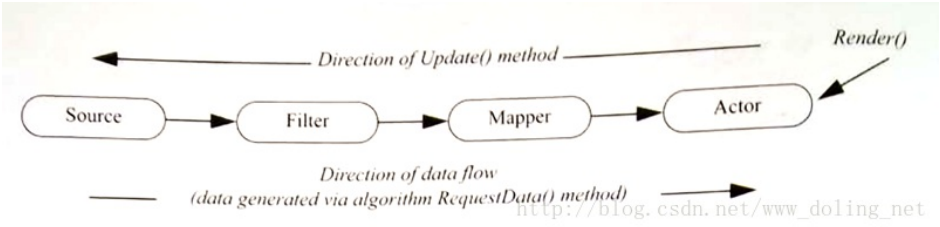

图3-6管线执行的概观

图像处理

VTK支持大量的图像处理和体绘制功能。VTK中无论是2D（图像）还是3D（volume，体）数据都可以看做为vtkImageData。一个图像数据集是一个沿坐标轴规则排列的数据数组；Volume（二维图像集合）是三维的图像数据集。

图像处理管线中算法的输入和输出通常为图像数据对象。由于图像数据简单而且规则的性质，图像处理管线还有一些其他的重要特征。体绘制算法用来可视化三维vtkImageData（参考139页“体绘制”一章），另外一些专门的图像Viewer【译者：如vtkImageViewer2等类】则是用来浏览二维vtkImageData。图像处理管线中几乎所有的算法都是多线程的，而且采用分片处理流数据以适应用户内存大小限制。运行时，Filter能够自动获取系统中处理器和核的个数，并创建相应的线程，同时，自动将数据进行分片。（参考325页“vtkStreamingDemandDrivenPipeline”）

以上是对VTK系统结构的总体概述。更多VTK中算法的详细细节请参阅《The Visualization Toolkit AnObject-Oriented Approach to 3D Graphics》一书。学习VTK中示例程序也是学习VTK的一个好方法。第4章到13章中包含了许多的示例程序来说明VTK的主要功能。另外，因为VTK代码是开源的，你也可以学习VTK源码目录中VTK/Examples中的例子。

通过以上简略的介绍之后，下面来看一下如何用C++，Tcl，Java，和Python来创建应用程序。

第03章-VTK系统概述（3）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本节对应原书中的第29页至第39页】

3.2创建VTK应用程序

本章内容包括利用Tcl，C++，Java和Python四种语言开发VTK应用程序的基本知识。阅读完引言后，你应该了解用你擅长的语言进行VTK开发的相关内容。为了指导你怎么去创建和运行一个简单的VTK程序，接下来内容都会针对不同的编程语言演示怎样使用Callback。

用户事件、观察者以及命令模式

Callback（又称用户方法UserMethod）采用Subject/Observer和Command设计模式进行设计。VTK中几乎所有的类都可以通过SetObserver()方法建立Callback。Observer观察者监听对象中的所有激活事件，一旦其中一个事件与其监听事件类型一致的话，则其相应的Command就会执行（如Callback）。例如，所有Filter在执行前都会激活StartEvent事件。如果为Filter添加一个Observer来监听StartEvent事件，那么每次Filter执行前，该Observer都会被调用响应Callback。下面的Tcl脚本中创建了一个vtkElevation的实例，并为添加一个Observer来监听StartEvent事件。当Observer监听到该事件时，则自动响应函数PrintStatus()。

```
proc PrintStatus {} {
puts “Starting to execute the elevation filter”
}
vtkElevationFilter foo
foo AddObserver StartEvent PrintStatus
```

VTK支持的所有语言都可使用这种类型的函数（Callback）。接下来每个小节中都会给出一个简单的例子来说明如何使用它。关于用户方法的深入探讨请参考421页“Integratingwith The Windowing System”（与窗口系统的整合）（该章中还涉及了用户接口整合问题）。

我们建议从VTK自带的例子开始学习如何创建应用程序。这些例子在VTK源文件VTK\Examples目录下。目录中根据不同的主题进行细分，在每个主题目录中会根据不同的语言再分为不同的子目录。

Tcl

学习使用VTK创建应用程序时，Tcl脚本语言是最简单的语言之一。VTK安装完毕后，即可执行VTK自带的Tcl例子。Unix系统下，根据“Unix平台下安装VTK”一节介绍，编译VTK时需要选择支持Tcl。而Windows系统下只需要编译安装自解压目录即可，参考第10页“WindowsXP, Vista及以上版本平台下安装VTK”。

Windows：Windows系统下只需双击Tcl脚本文件即可执行（如本例中Cone.tcl）。如果双击没有反应的话，脚本中可能存在错误，或者是相应的Tcl文件与vtk.exe有错误。如果要进一步检测具体问题，首先需要运行vtk.exe程序，该程序在vtk启动菜单中可以找到。执行后，会弹出一个命令提示行的控制台窗口，在提示行中，键入“cd”命令定位到Tcl文件目录，如下：

```
% cd“ c:/VTK/Examples/Tutorial/Step1/Tcl”
```

然后键入如下命令定位示例脚本：

```
%source Cone.tcl
```

Tcl会执行该脚本，这时所有的错误或者警告信息会输出来。

Unix：Unix下Tcl开发可以通过运行Binary编译目录下（如VTK-bin/bin/VTK，VTK-Solaris/bin/VTK）的可执行文件（编译完成后），然后将脚本文件作为第一个参数输入，如下所示。

```
unixmachine> cd VTK/Examples/Tutorial/Step1/Tcl

unixmachine> /home/ VTK-Solaris/bin/VTK Cone.tcl
```

用户方法使用如前所述。Examples/Tutorial/Step2/Tcl/Cone2.tcl中有相关的使用示例。下面仅列出了其中的关键部分。

```
Proc mycallback {} {

    Puts “Starting to render”

}


vtkRendererren1

ren1AddObserver StartEvent mycallback
```

你也可以直接给AddObserver()提供函数体。

```
vtkRenderer ren1

ren1 AddObserver StartEvent {puts “Starting to render”}
```

C++

相对于其他语言，C++应用程序体积更小，运行更快，而且容易部署安装。此外，采用C++语言开发VTK应用程序，你不需要编译额外的Tcl、Java和Python支持。本节中主要说明怎么在PC机上用MicrosoftVisual C++或者Unix下的编译器来用C++开发简单的VTK应用程序。

我们以Examples/Tutorial/Step1/Cxx下的Cone.cxx为例进行讲解。无论Windows平台还是Unix，都可以使用源代码编译版本或者是发布的可执行版本，两个版本都支持这些例子。

编译C++程序的第一步是利用CMake生成依赖于编译器的makefile或者是项目工程文件。Cone.txx目录下的CMakeLists.txt利用CMake的FindVTK和UseVTK模块来定位VTK目录并设置包含路径和链接库等。如果没有成功找到VTK的话，你需要手动设置CMake相关的参数并重新运行CMake。

```
PROJECT(Step1)

FIND_PACKAGE(VTKREQUIRED)

IF(NOTVTK_USE_RENDERING)

  MESSAGE(FATAL_ERROR "Example${PROJECT_NAME} requires VTK_USE_RENDERING.")

ENDIF(NOTVTK_USE_RENDERING)

INCLUDE(${VTK_USE_FILE})

ADD_EXECUTABLE(ConeCone.cxx)

TARGET_LINK_LIBRARIES(ConevtkRendering)
```

MicrosoftVisual C++：用CMake对Cone.cxx配置完成后，启动MicrosoftVisual C++并载入生成的解决方案。当前.net版本下的解决方案名字是Cone.sln。根据需要选择Release或者Debug版本编译程序。如果想把将VTK整合到其他不采用CMake的工程中，可以直接拷贝该工程的设置到相应的工程中。

下面看一个Windows应用程序示例。其创建过程与上述例子基本相同，除了我们建立的是一个Windows窗口程序而不是控制台程序。大部分代码都是Windows开发者熟悉的标准Windows窗口语言。该例子在VTK/Examples/GUI/Win32/SimpleCxx/Win32Cone.cxx目录下。注意CMakeLists.txt文件中的一个重要变化是ADD_EXECUTABLE命令的WIN32参数。

```
#include "windows.h"
#include "vtkConeSource.h"
#include "vtkPolyDataMapper.h"
#include "vtkRenderWindow.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkRenderer.h"

static HANDLE hinst;
LRESULTCALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

//define the vtk part as a simple c++ class
class myVTKApp
{
public:
  myVTKApp(HWND parent);
  ~myVTKApp();
private:
  vtkRenderWindow *renWin;
  vtkRenderer *renderer;
  vtkRenderWindowInteractor *iren;
  vtkConeSource *cone;
  vtkPolyDataMapper *coneMapper;
  vtkActor *coneActor;
};
```

程序开始包含进必须的VTK头文件。然后是两个原型声明，接下来定义了一个myVTKApp类。使用C++开发时，要尽量采用面向对象的编程方法，而不是像Tcl例子中的脚本语言样式。这里我们将VTK应用程序的组件封装到一个简单类中。

下面是myVTKApp类的构造函数。首先创建每个VTK对象并进行相应的设置，然后将各个组件对象连接形成可视化管线。除了vtkRenderWindow的代码外，其他都是很简洁的VTK代码。构造函数接收一个父窗口的HWND的句柄以便使父窗口包含VTK渲染窗口。然后利用vtkRenderWindow的SetParentId()函数设置父窗口，并创建自己的窗口作为父窗口的子窗口。

```
myVTKApp::myVTKApp(HWNDhwnd)
{
  // Similar toExamples/Tutorial/Step1/Cxx/Cone.cxx
  // We create the basic parts of a pipelineand connect them
  this->renderer = vtkRenderer::New();
  this->renWin = vtkRenderWindow::New();
 this->renWin->AddRenderer(this->renderer);

  // setup the parent window
  this->renWin->SetParentId(hwnd);
  this->iren =vtkRenderWindowInteractor::New();
 this->iren->SetRenderWindow(this->renWin);

  this->cone = vtkConeSource::New();
  this->cone->SetHeight( 3.0 );
  this->cone->SetRadius( 1.0 );
  this->cone->SetResolution( 10 );
  this->coneMapper =vtkPolyDataMapper::New();
 this->coneMapper->SetInputConnection(this->cone->GetOutputPort());
  this->coneActor = vtkActor::New();
  this->coneActor->SetMapper(this->coneMapper);

 this->renderer->AddActor(this->coneActor);
 this->renderer->SetBackground(0.2,0.4,0.3);
  this->renWin->SetSize(400,400);

  // Finally we start the interactor so thatevent will be handled
  this->renWin->Render();
}
```

析构函数中释放所有构造函数中创建的所有VTK对象。

```
myVTKApp::~myVTKApp()
{
    renWin->Delete();
    renderer->Delete();
    iren->Delete();
    cone->Delete();
    coneMapper->Delete();
    coneActor->Delete();
}
```

WinMain函数中都是标准的Windows编程语言，没有用到VTK引用。该应用程序中具有消息循环控制功能，消息处理由下面介绍的WinProc函数实现。

```
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR /* lpszCmdParam */,int nCmdShow)
{
  static char szAppName[] ="Win32Cone";
  HWND        hwnd ;
  MSG         msg ;
  WNDCLASS    wndclass ;

  if (!hPrevInstance)
    {
    wndclass.style         = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
    wndclass.lpfnWndProc   = WndProc ;
    wndclass.cbClsExtra    = 0 ;
    wndclass.cbWndExtra    = 0 ;
    wndclass.hInstance     = hInstance;
    wndclass.hIcon         = LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
    wndclass.lpszMenuName  = NULL;
    wndclass.hbrBackground =(HBRUSH)GetStockObject(BLACK_BRUSH);
    wndclass.lpszClassName = szAppName;
    RegisterClass (&wndclass);
    }

  hinst = hInstance;
  hwnd = CreateWindow ( szAppName,
                        "DrawWindow",
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        400,
                        480,
                        NULL,
                        NULL,
                        hInstance,
                        NULL);
  ShowWindow (hwnd, nCmdShow);
  UpdateWindow (hwnd);
  while (GetMessage (&msg, NULL, 0, 0))
    {
    TranslateMessage (&msg);
    DispatchMessage (&msg);
    }
  return msg.wParam;
}
```

WinProc是一个简单的消息处理函数。对于一个完整的应用程序来说，它可能要比这个复杂的多，但是关键部分都是相同的。函数开始定义了一个myVTKApp实例的静态引用变量。当处理WM_CREATE消息时，我们创建了一个Exit按钮，然后创建myVTKApp实例并传入当前窗口的句柄。vtkRendererWindowInteractor会为vtkRendererWindow处理所有的消息，因此这里不需要处理消息。你很可能想添加代码来处理resizing消息，这样窗口的大小就可以随着用户界面的改变而自动调整。如果没有设置vtkRendererWindow的ParentId的话，vtkRendererWindow就作为一个顶层的独立窗口显示。其他的部分则没有变化。

```
LRESULTCALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HWND ewin;
  static myVTKApp *theVTKApp;

  switch (message)
    {
    case WM_CREATE:
      {
      ewin =CreateWindow("button","Exit",
                         WS_CHILD | WS_VISIBLE| SS_CENTER,
                         0,400,400,60,
                         hwnd,(HMENU)2,
                        (HINSTANCE)vtkGetWindowLong(hwnd,vtkGWL_HINSTANCE),
                         NULL);
      theVTKApp = new myVTKApp(hwnd);
      return 0;
      }

    case WM_COMMAND:
      switch (wParam)
        {
        case 2:
          PostQuitMessage (0);
          if (theVTKApp)
            {
            delete theVTKApp;
            theVTKApp = NULL;
            }
          break;
        }
      return 0;

    case WM_DESTROY:
      PostQuitMessage (0);
      if (theVTKApp)
        {
        delete theVTKApp;
        theVTKApp = NULL;
        }
      return 0;
    }
  return DefWindowProc (hwnd, message, wParam,lParam);
}
```

Unix： Unix下主要通过运行CMake和make来创建应用程序。CMake生成一个makefile文件，该文件中设置了包含路径、链接路径（link lines）和依赖库等。然后make程序利用makefile文件编译应用程序，生成一个可执行文件。如果Cone.txx编译失败，则需要检查和改正代码中的错误。确保CMakeCache.txt文件中开始的变量值是有效的。如果可以编译，但是运行出错的话，则需要参考第二章中设置LD_LIBRARY_PATH变量。

C++中的用户方法：通过实例化vtkCommand的派生类，并重载Execute()函数，你可以添加自己的用户方法（利用Observer/Command设计模式）。下面例子截取自VTK/Exampl

es/Tutorial/Step2/Cxx/Cone2.cxx。

```
class vtkMyCallback : public vtkCommand
{
public:
  static vtkMyCallback *New()  { return new vtkMyCallback; }
  virtual void Execute(vtkObject *caller,unsigned long, void*)
    {
      vtkRenderer *renderer =reinterpret_cast<vtkRenderer*>(caller);
      cout <<renderer->GetActiveCamera()->GetPosition()[0] << " "
           <<renderer->GetActiveCamera()->GetPosition()[1] << " "
           <<renderer->GetActiveCamera()->GetPosition()[2] << "\n";
    }
};
```

Execute()函数会传递一个不常使用的caller对象。当你的确需要调用caller时，你需要使用SafeDownCast函数将其转换为其实际的类型。例如：

```
virtual void Execute (vtkObject *caller, unsigned long, void *callData)
{
  vtkRenderer *ren = vtkRenderer::SafeDownCast(caller);
  if (ren) { ren->SetBackground(0.2, 0.3,0.4); }
}
```

当创建了你的vtkCommand派生类对象时，你就可以添加一个Observer来监听某个事件。当事件发生时，就会通过Observer执行你的Command命令。如下所示。

```
//Hereis where we setup the observer
//wedo a new and ren1 will eventually free the observer
vtkMyCallback*mol = vtkMyCallback::New();
ren1->AddObserver(vtkCommand::StartEvent,mol);
```

上面代码创建了一个myCallback实例，然后为ren1添加了一个Observer来监听StartEvent事件。当ren1开始渲染时，vtkMyCallback的Execute方法就会被调用。ren1被删除后，Callback对象也会相应的删除。

Java

创建Java应用程序，首先需要一个Java开发环境。本小节主要为Windows或者Unix下利用sun公司JDK1.3或者后续版本开发VTK应用程序提供指导。当JDK和VTK都成功安装后，你需要设置CLASSPATH环境变量来包含VTK类库。微软Windows系统下右击“我的电脑”图标，选择“属性”选项卡，然后选择“高级”tab页，点击“环境变量”按钮。然后添加一个CLASSPATH环境变量，设置值vtk.jar文件路径、Wrapping/java路径及当前路径。Windows下其值如“C:\vtk-bin\bin\vtk.jar;C:\vtk-bin\Wrapping/java;.”。Unix系统下CLASSPATH应设置如“/yourdisk/vtk-bin/bin/vtk.jar;/yourdisk/vtk-bin/Wrapping/java;.”。

接着是字节编译（ByteCompile）Java应用程序。对于新手可以尝试编译（用javac命令）VTK自带例子VTK/Examples/Tutorial/Step1/Java/Cone.java。编译完成后，通过java命令即可运行生成的应用程序。该程序绘制了一个旋转360度的圆锥，然后退出。以该例子作为起点，可以开始创建自己的应用程序了。

```
public void myCallback()
{
  System.out.println("Starting torender");
}
```

设置Callback需要三个参数。首先是你关心的事件，第二是类的实例，第三是调用的函数名。在本例中我们设置监听StartEvent事件以调用me（Cone2的实例）的方法myCallback。为了避免错误，确保myCallback函数是Cone2的成员函数（该例子源码见VTK/Examples/Tutorial/Step2/Java/cone2.java）。

```
Cone2 me = new Cone2();
ren1.AddObserver("StartEvent",me, "myCallback");
```

Python

如果编译VTK时设置支持Python，那么编译后会生成一个vtkpython可执行文件。利用这个可执行文件，可以如下运行“Examples/Tutorial/Step1/Python/Cone.py”。

```
vtkpython Cone.py
```

以VTK自带的示例脚本作为参照来创建Python脚本会比较简单。用户方法通过定义一个函数并将其作为参数传入AddObserver来定义，如下：

```
def myCallback(obj, event)
   print "Starting to render"
ren1.AddObserver("StartEvent",myCallback)
```

完整的源代码见VTK/Examples/Tutorial/Step2/Python/Cone2.py。

3.3语言间的转换

VTK核心代码采用C++语言实现，然后用Tcl，Java和Python编程语言包装。这样在应用程序开发时，你可以有多种语言选择。语言选择主要依赖于个人的语言习惯，应用程序的特点，是否需要访问内部数据以及是否对性能有特殊要求。与其他语言相比，C++在访问内部数据结构、应用程序执行效率方***有更多的优势。但是，采用C++语言也带来了编译/链接循环的负担，从而降低了软件开发效率。

你可以使用解释性语言如Tcl来开发应用程序原型，然后再转换为C++程序。或者搜索示例代码（VTK目录或者其他用户）然后将其转换为最终的实现语言。

VTK代码的语言转换比较直接。各个语言中都采用相同的类名和方法名；不同的只是实现细节和GUI接口。例如：

C++语言：

```
anActor->GetProperty()->SetColor(red,green, blue)
```

Tcl语言则为：

```
[anActorGetProperty] Set Color $red $green $blue
```

Java语言为：

```
anActor.GetProperty().SetColor(red,green, blue);
```

Python为：

```
anActor->GetProperty().SetColor(red,green, blue);
```

由于指针操作问题，一些C++应用程序不能转换为其他三种语言，这也是语言转换时的一个主要限制。包装语言可以方便的获取或者设置对象的值，但是不能直接获取一个指针来快速遍历、检查或者修改大的数据结构。如果你的应用程序需要类似的操作，那你可以直接采用C++语言实现，或者利用C++扩展VTK生成你需要的类，然后使用你喜欢的解释性语言来使用新的类。

【第3章 翻译完毕】

第04章-VTK基础（1）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本小节内容对应原书的第41页至第45页】

这一章通过一系列典型的例子介绍VTK的功能。重点在于常用的方法和对象，以及对象间的组合。同时，介绍了VTK中的重要概念和应用。本章的目的是让读者对VTK有个大概的了解，并没有涵盖VTK所有的特性。读者可以参考在线文档或者类.h文件来学习每个类的其他功能。

本书的大多数例子是用Tcl程序语言实现的。这些例子可以用C++，Java和Python实现，这些编程语言之间可以直接地进行转换（参考"语言间的转换"一节）。由于C++在数据结构和指针操作方***有显著的优势，有关这方面的例子会采用C++语言实现。

这里列出的每个例子都包括了相关的代码和补充说明的图像。如果该例子在VTK源文件目录中存在的话，我们会列出该例子文件的名字，所以你不需要手动去输入这些代码。建议你自己去运行这些例子，理解里面的代码，用不同的参数进行实验。你也可以用不同的方法或者类去实现同样的功能。通常，VTK都会提供几种不同的方法来获得类似的结果。注意，脚本文件通常会因发布的源码的改变而作修改，这是为了简化概念或者删除多余的代码。

学习像VTK这样的面向对象的系统首先需要理解抽象编程，然后要熟悉对象库及其方法。因此，建议读者先复习一下第三章"VTK系统概述"中有关抽象编程的概念。这一章的例子可以让读者对VTK对象有一个很好的认识。

4.1创建简单模型

使用VTK的步骤一般为：读入/生成数据，对数据进行Filter操作，数据的渲染和交互。这部分的主要内容就是数据的读取和生成。

获取数据有两种方法，数据可能存在于读入系统内存的文件（或数据流）中，也可能由程序生成（通过算法或者数学表达式等形式）。在可视化管道中初始化数据处理的对象称为源对象（参考图3-5）。生成数据的对象称为程序（源）对象，读入数据的对象称为读（源）对象。

程序源对象(Procedural Source Object)

我们从绘制简单的圆柱开始。以下例子代码（VTK/Examples/Rendering/Tcl/Cylinder.tcl）展示了可视化管道和渲染引擎的一些基本概念。图4-1显示了该Tcl脚本的运行结果。

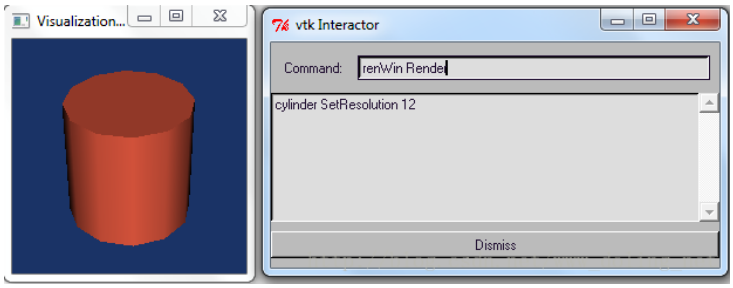

图4-1 使用Tcl/Tk编写的解释性应用程序

首先我们在该脚本程序的最开始调用了Tcl命令来加载VTK工具包（packagerequired vtk），并且创建了带用户交互界面的解释器（package requiredvtkinteraction），通过该交互界面可以让你在程序运行时输入命令。另外，脚本也加载了vtktesting模块，该模块定义了一系列的颜色，其中"番茄色"在该脚本中会使用到。

```
Package required vtk
Package required vtkinteraction
Package requiredvtktesting
```

接着，创建一个程序源对象：vtkCylinderSource，该类创建一个横截面为多边形的柱体，柱体的输出通过方法SetInputConnection()设置为vtkPolyDataMapper对象的输入。接着创建一个actor对象（要渲染的对象），设置定义几何信息的mapper到这个actor里。注意类对象在Tcl中的构造方式：类名后面紧跟对象的名字。

```
vtkCylinderSourcecylinder
  cylinder SetResolution 8
vtkPolyDataMappercylinderMapper
  cylinderMapper SetInputConnection [ cylinderGetOutputPort ]
vtkActor cylinderActor
   cylinderActor SetMapper cylinderMapper
   eval [ cyliderActor GetProperty ] SetColorStomato
      cylinderActor RotateX  30.0
      cylinderActor RotateY  -45.0
```

为了证明C++代码的实现与Tcl（及其他解释性语言）的相似性，与以上例子相同的C++实现代码也在下面列出，可以在VTK/Examples/Rendering/Cxx/Cylinder.cxx中找到该例子的C++代码。

```
vtkCylinderSource*cylinder = vtkCylinderSource::New();
cylinder->SetResolution(8);

vtkPolyDataMapper*cylinderMapper = vtkPolyDataMapper::New();
cylinderMapper->SetInputConnection(cylinder->GetOutputPort() );

vtkActor*cylinderActor = vtkActor::New();
cylinderActor->SetMapper(cylinderMapper );
cylinderActor->GetProperty()->SetColor( 1.0000, 0.3882, 0.2784 );
cylinderActor->RotateX(30.0 );
cylinderActor->RotateY(-45.0 );
```

回顾之前的内容可知，源对象位于可视化管线的开始，mapper对象（或者具有mapper功能的prop对象）处于管线的终端。因此，该例子的管线包含两种算法（即源source和mapper）。VTK的管线使用**惰性计算策略**，即使管线已经连接，如果没有请求获取数据，程序也不会生成数据及对数据进行处理。

接下来，为了渲染actor需要创建图形对象。vtkRenderer的实例ren1协调渲染窗口renWin的视口（viewport）的渲染过程。渲染窗口交互器实例iren是一个3D的widget，可以控制三维渲染场景的相机。

```
#Create the graphicsstructure
vtkRenderer ren1
vtkRenderWindow renWin
   renWin AddRenderer ren1
vtkRenderWindowInteractoriren
   iren SetRenderWindow renWin
```

注意，我们通过渲染窗口类的方法AddRenderer()把renderer和渲染窗口关联起来，接着使用renderer的方法AddActor()把要渲染的actor加入到renderer中去。

```
# Add the actors tothe renderer, set the background and size
ren1 AddActorcylinderActor
ren1 SetBackground 0.10.2 0.4
renWin SetSize 200 200
```

SetBackground()方法用取值范围为(0,1)的RGB（红，绿，蓝）值设置渲染窗口的背景色，SetSize()以像素为单位来确定窗口的大小。最后，将GUI交互器与用户自定义的渲染交互窗口交互器的方法关联起来。（当鼠标的焦点处在渲染窗口时，用户自定义的方法可以通过按击键盘u键激活。可参考本章的“使用VTK交互器”一节，或者参考第三章的“**用户事件、观察者以及命令模式**”一节）。调用Initialize()方法进入事件循环，Tcl/Tk的命令wmwithdraw .可以确保在程序开始运行时interacter
widget .vtkInteract不可见。

```
# Associate the “u”keypress with a UserEvent and start the event loop
iren  AddObserver UserEvent ( wm deiconify ,vtkInteract )
iren  Initialize

# suppress the tkwindow
wm withdraw
```

当以上脚本程序运行时，因为渲染引擎会请求数据，所以可视化管线就会执行。（窗口的expose事件会迫使渲染窗口自行渲染。）只有输入数据改变才能令管线执行的数据更新。读者可根据需要调用renWin Render手动控制管线的执行。

例子运行后，你可以尝试：首先，用鼠标与渲染窗口进行交互。然后，可以调用cylinder SetResolution 12改变柱体的分辨率（即柱体横截面的多边形的边数）。可以编辑脚本然后重新执行这个例子；也可以按击键盘u键，就会弹出图4-1所示的基于GUI的命令解释器，可以在上面输入命令。这些改变只有在你重新请求数据时才会生效，所以在输入cylinder SetResolution12命令以后，还必须输入renWin Renderer命令，或者用鼠标点击一下渲染窗口，所做的改变才会生效。

读取源对象ReaderSource Object

这个例子与前面的例子类似，除了用读取数据文件来生成数据替代由程序生成数据。立体成型(stereo-lithography)数据文件(后缀为.stl)是使用二进制STL数据格式来表达多边形数据。（参考图4-2和Tcl脚本VTK/Examples/Rendering/Tcl/CADPart.tcl）

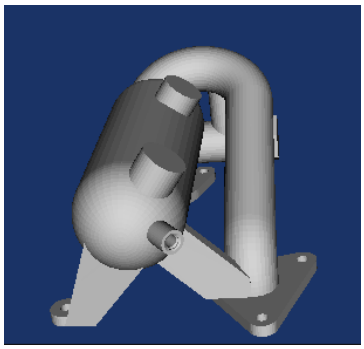

图4-2读取源对象

```
vtkSTLReader part
   part SetFileName$VTK_DATA_ROOT/Data/42400-IDGH.stl
vtkPolyDataMapperpartMapper
   partMapper SetInputConnecteion [ partGetOutputPort ]
vtkLODActor patrActor
   partActor SetMapper partMapper
```

注意类vtkLODActor的使用。为保证交互时的顺畅性，这种类型的actor会更改自身的显示形式，缺省情况下会创建一个带线框的点云替代交互时actor的显示。（参考本章“Level-Of-DetailActors”一节。）本例中所用到的模型数据量较小，在现在的大多数机器上运行的话，还是显示模型完整的形式（图4-2所示）。

文件读取类不会监听输入文件是否发生变化以及是否要重新执行管线。例如，如果文件42400-IDGH.stl作了改变，管线不会重新执行。可以手动调用方法Modified()以使这些更改生效。这个方法会使得该filter重新执行以及更新它后续的数据。

VTK限制了部分建模功能，如果你想用VTK编辑或者操作复杂的模型（例如，用solid modeler或其他建模工具建立的模型），一般使用文件读取（参考第十二章关于数据读取方面的内容）来导入数据。

第04章-VTK基础（2）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本小节内容对应原书的第45页至第48页】

4.2使用VTK交互器

对数据进行可视化时，通常都希望与这些数据进行交互，VTK提供多种数据交互的方法。第一种方法是使用vtkRenderWindowInteractor类，第二种方法是通过绑定特定的事件定制自己的交互类。如果是使用解释性语言的话，还可以在程序运行时输入命令与之交互。可以参考本章“拾取”（Picking）一节，了解如何从屏幕上选择数据。（注意：开发人员也可以选择窗口系统接口，详细内容请参考第18章。）

vtkRenderWindowInteractor

与数据进行交互最简单的方法就是实例化一个vtkRenderWindowInterator对象。该类会响应一系列预先设置的事件和动作，而且也提供了重载缺省动作的方法。通过这个类可以控制渲染场景的相机和Actor，包括两种交互风格：位置敏感型（position sensitive，即joystick模式）和动作敏感型（motionsensitive，即trackball模式）。（后面的章节会详细介绍交互风格方面的内容。）

vtkRenderWindowInterator可以响应以下的渲染窗口事件。（一个渲染窗口可以关联多个渲染器，其中渲染器绘制在渲染窗口的视口里，交互器支持一个渲染窗口拥有多个渲染器。）

按键j和按键t—— joystick（位置敏感型）和trackball（动作敏感型）模式之间的切换。joystick模式下，只要鼠标按下动作就持续发生；trackball模式下，鼠标按下且移动时动作才产生。
按键c和按键a——相机和Actor（对象）模式之间的切换。相机模式下，鼠标事件对相机的位置和焦点起作用；对象模式下，鼠标事件作用于鼠标指针下的actor。
鼠标左键—— 相机模式下绕焦点旋转相机；对象模式下绕原点旋转actor。旋转方向是从渲染器视口的中心指向鼠标所在的位置。joystick模式下，旋转速度取决于鼠标的位置与视口中心的距离大小。
鼠标中键—— 相机模式下滚动中键可以实现相机镜头的拉伸(Pan)；对象模式下中键可以平移actor。joystick模式下，镜头拉伸或平移的方由从视口中心指向鼠标所在的位置；trackball模式下，actor运动的方向与鼠标运动的方向一致。（注意：如果是两键鼠标，镜头的拉伸/对象的平移操作可以用按键<shift>+鼠标左键完成。）
鼠标右键—— 相机模式下可以实现相机镜头的拉伸(Zoom)；对象模式下可以实现actor的缩放。按下鼠标右键且朝视口上半部分移动鼠标时，就是镜头的拉近/actor的放大；按下鼠标右键且朝视口的下半部分移动鼠标时，就是镜头的拉远/actor的缩小。joystick模式下，镜头拉伸的快慢或actor缩放速度取决于鼠标所在的位置与视口水平中心线的距离。
按键3—— 激活或关闭渲染窗口的立体模式(stereo mode)。缺省情况下，会创建红-蓝立体对(red-bluestereo pairs)。对一些支持Crystal Eyes LCD立体眼镜的，必须调用方法SetStereoToCrystalEyes()实现该功能。
按键e/q—— 退出应用程序。
按键f—— 移动actor到光标的当前的位置。设置当前光标所在的位置为焦点，而且会以该点为旋转中心。
按键p—— 执行拾取（pick）操作。渲染窗口交互器内部有一个vtkPropPicker实例，可实现拾取功能。详细内容参考本章“拾取”一节。
按键r—— 沿当前视场的方向重置相机。移动相机和actor使所有的对象都可见。
按键s—— 改变所有actor的显示形式为表面模型。
按键u—— 调用用户自定义方法。（在Tcl/Tk脚本程序下）通常会弹出交互窗口，可以输入命令进行交互。
按键w—— 改变所有actor的显示形式为线框模型。

默认的交互风格是位置敏感型（即joystick模式），也就是说，只要鼠标按下就可以持续控制相机或actor以及渲染器。如果不喜欢这种缺省风格，你可以替换或者创建自己的交互风格。（参考第十八章了解有关编写交互风格的内容。）

vtkRenderWindowInteraction还有其他一些有用的特性。调用LightFollowCameraOn()方法(默认的行为)可以使相机位置和焦点与光源的位置和焦点同步（即创建“headlight”）。当然，也可以调用LightFollowCameraOff()关闭该功能。响应“u”键的回调函数可以通过AddObserver(UserEvent)方法进行设置。也可以设置一些与拾取有关的方法。AddObserver(StartPickEvent)定义了拾取之前调用的方法，而AddObserver(EndPickEvent)定义的是拾取完成后调用的方法。（详细内容可参考第三章“用户事件、观察者及命令模式”一节。）同样，你也可以实例化一个vtkAbstractPicker子类的对象，通过vtkRenderWindowInteractor里的方法SetPicker()设置拾取类型。（参考本章“拾取”一节。）

针对某一个Prop如果你想在交互速度与渲染质量方面取得平衡，可以通过交互器里的方法SetDesiredUpdateRate()设置期望的帧更新速率(Desiredframe rate)。正常情况下，这些会自动进行处理的。（当鼠标处于活动状态时，期望渲染帧率(Desired update rate)会提高；当鼠标松开时，期望渲染帧率会降回原来的值。）参考本章的“Level-Of-DetailActors”，“vtkLODProp3D”以及第七章“体绘制”方面的内容，了解更多的关于Prop以及与之相关联的mapper是如何调整渲染风格以达到期望的帧更新速率的。

前面已经介绍了vtkRenderWindowInteractor的使用方法，现摘要如下：

```
vtkRenderWindowInteractoriren
  iren SetRenderWindow renWin
  iren AddObserver UserEvent {wm deiconify.vtkInteract}
```

交互风格Interactor Styles

VTK有两种不同的方法可以控制交互风格。第一种是使用vtkInteractorStyle的子类，可以是VTK自带的或者是你自行编写的类。第二种是添加Observer监听vtkRenderWindowInteractor里的事件，定义一系列回调函数（或命令）来实现交互。（注意：3D Widget是另外一种与渲染场景中的数据进行交互的更加复杂的方式，可以参考本章的“3D Widget”一节了解更多的信息。）

vtkInteractorStyle. 类vtkRenderWindowInteractor可以支持不同的交互风格。交互时键入“t”或者“j”可以在trackball和joystick两种交互模式之间切换（见前面的内容）。vtkRenderWindowInteractor可以将特定的窗口系统事件（如鼠标按下，鼠标移动，键盘事件等）转换成MouseMoveEvent, StartEvent等VTK事件。（参考第三章“用户事件、观察者及命令模式”一节。）不同的交互风格可以监听特定的事件，然后执行与之相关联的动作。通过vtkRenderWindowInteractor::SetInteractorStyle()方法可以设置不同的交互风格，比如：

```
vtkInteractorStyleFlightfightStyle
vtkRenderWindowInteractoriren
  iren SetInteractorStyle flightStyle
```

注意：实例化vtkRenderWindowInteractor后，实际上就是实例化一个与特定的窗口系统相关的交互器实例，例如， UNIX系统下，当建立一个vtkRenderWindowInteractor实例时，实际上建立的是vtkXRenderWindowInteractor实例，而Windows系统下就是建立一个vtkWin32RenderWindowInteractor实例。

添加vtkRenderWindowInteractorObservers. 虽然VTK里已经有大量的交互风格可供使用，但你也可根据应用程序的需要自行定制其他的交互风格。C++里做法一般是从vtkInteractorStyle派生出子类（参考第十八章“vtkRenderWindowInteraction Style”的内容），但对于解释性语言（如Tcl, Python或Java）来说，这样做就比较困难。对于这些解释性语言，最简单的方法就是使用Observer来绑定特定的事件进行交互（参考第三章“用户事件、观察者及命令模式”一节）。这种绑定适用于VTK支持的任何语言，包括C++,Tcl, Python和Java，VTK/Examples/GUI/Tcl/CustomInteraction.tcl这个Tcl例子看到这种用法。下面的代码就是从这个例子摘抄出来的，看看这种绑定是如何进行的。

```
vtkRenderWindowInteractoriren
irenSetInteractorStyle ""
irenSetRenderWindow renWin

#Add the observers to watch for particular events. These invoke
#Tcl procedures.

setRotating 0
setPanning 0
setZooming 0
irenAddObserver LeftButtonPressEvent {global Rotating; set Rotating 1}
irenAddObserver LeftButtonReleaseEvent {global Rotating; set Rotating 0}
irenAddObserver MiddleButtonPressEvent {global Panning; set Panning 1}
irenAddObserver MiddleButtonReleaseEvent {global Panning; set Panning 0}
irenAddObserver RightButtonPressEvent {global Zooming; set Zooming 1}
irenAddObserver RightButtonReleaseEvent {global Zooming; set Zooming 0}
irenAddObserver MouseMoveEvent MouseMove
irenAddObserver KeyPressEvent Keypress

procMouseMove {} {
  …
  set xypos [iren GetEventPosition]
  set x [lindex $xypos 0]
  set y [lindex $xypos 1]
…
}

procKeypress {} {
  set key [iren GetKeySym]
    if { $key == "e" } {
        vtkCommand DeleteAllObjects
        exit
    }
…
}
```

这个例子关键的一步是调用SetInteractorStyle(“”)方法关闭缺省的交互风格，然后加入关联适当的Tcl过程的Observer来监听特定的事件。

这个例子是一个简单的介绍如何加入事件绑定的Tcl脚本程序，如果你想使用Tcl/Tk创建完整的包含用户图形界面的应用程序，可以使用类vtkTkRenderWidget，详细内容请参考第十八章。

第04章-VTK基础（3）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本小节内容对应原书的第48页至第52页】

4.3Filtering Data

4.1节例子里的管线只由一个Source和Mapper对象组成，管线里没有Filter。这一部分我们会演示如何在管线里增加一个Filter。

Filter的连接可以通过方法SetInputConnection()和GetOutputPort()。例如，我们可以在4.1节例子的基础上，对组成模型的多边形数据(Polygon)做收缩(Shrink)处理，下面列出了部分代码，完整的代码可以在VTK/Examples/Rendering/Tcl/FilterCADPart.tcl里找到。

```
vtkSTLReaderpart
    part SetFileName"$VTK_DATA_ROOT/Data/42400-IDGH.stl"
vtkShrinkPolyDatashrink
    shrink SetInputConnection [partGetOutputPort]
    shrink SetShrinkFactor 0.85
vtkPolyDataMapperpartMapper
    partMapper SetInputConnection [shrinkGetOutputPort]
vtkLODActorpartActor
    partActor SetMapper partMapper
```

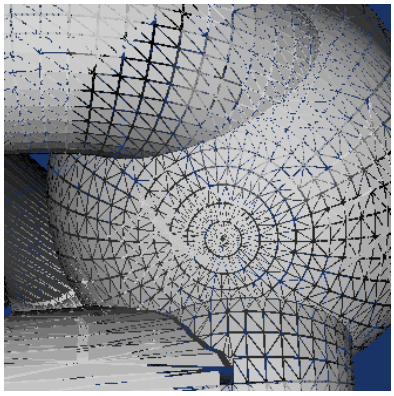

图4-3 Filtering Data, 这个例子我们使用了收缩Filter(Shrink Filter)，将多边形数据沿着模型的中心进行收缩。

正如你所见，创建可视化管线是比较简单的。只要选对正确的类，保证相互连接的Filter的输入和输出的类型匹配，以及设置必要的变量值即可。当管线里的Source或者Filter输出的数据类型与下一个Filter或Mapper的输入类型一致，就说输入和输出的类型是匹配的。输出的数据类型除了要与输入的匹配，还必须与输入的子类的类型匹配。可视化管线可以包含循环，但是一个Filter的输出不能直接地作为它本身的输入。

4.4控制相机

你可能已经注意到，以上所列的例子里，并没有实例化相机或光源对象。如果你对3D图形学比较熟悉的话，就应该知道相机和光源对于要渲染的对象来说是必不可少的。VTK里，如果没有直接地创建相机和光源对象的话，渲染器会自动地创建默认的相机和光源实例。

实例化相机

下面的Tcl脚本演示了如何实例化一个相机对象以及如何把相机与渲染器关联起来。

```
vtkCameracaml
  cam1 SetClippingRange 0.0475572 2.37786
  cam1 SetFocalPoint 0.052665 -0.129454-0.0573973
  cam1 SetPosition 0.327637 -0.116299 -0.256418
  cam1 ComputeViewPlaneNormal
  cam1 SetViewUp -0.0225386 0.999137 0.034901
ren1 SetActiveCamera cam1
```

如果你想访问一个已经存在的相机（比如，渲染器自动创建的相机实例），用Tcl脚本可以写为：

```
set cam1 [ren1 GetActiveCamera]
$cam1 Zoom 1.4
```

一起来看一下上面列出的有关相机的方法。SetClippingRange()函数接收两个参数，分别表示沿着视平面法向量的方向，相机与近剪裁平面(ClippingPlane)和远剪裁平面的距离。渲染时，所有位于这两个剪裁平面之外的图形元将会被剪切掉，所以，必须保证所有的你想观察的对象位于这两个剪裁平面之间。FocalPoint和Position变量（在世界坐标系里定义）分别控制相机的方向和位置。ComputeViewPlaneNormal()方法会以当前设置的相机位置和焦点重置视平面的法向量。如果视平面的法向量与视平面不垂直的话，可以得到其他的视觉效果，如错切(Shearing)等。ViewUp变量控置相机“向上”的方向。最后，Zoom()方法是通过改变视角(ViewAngle)(即SetViewAngle()方法)使得物体放大显示。同样，你也可以使用Dolly()方法沿着视平面的法向方向移动相机，或者放大、收缩渲染场景中可见的actor。

简单控制方法

以上介绍的方法并不是常用的控制相机的简便方法。如果相机已经“看着”你想要的那个点，即已经设置了相机的焦点(Focal Point)，你可以使用Azimuth()和Elevation()方法绕着焦点旋转相机。

```
cam1 Azimuth 150
cam1 Elevation 60
```

Azimuth()方法会在球坐标下以焦点为中心沿着经度方向(Longitude Direction)旋转指定的角度；Elevation()方法则是沿着纬度方向(LatitudeDirection)旋转。这两个方法都不会改变相机的View-up向量。但要注意一点，在球坐标的北极和南极时，View-up向量变成了与视平面法向量平行。如果要避免这种情况，可以调用方法OrthogonalizeViewUp()强制View-up向量与视向量正交。但是这样会改变相机的坐标系统，因此，如果你以水平或者View-up向量的方向绕着某个物体飞行（比如地形图），就数据而言，相机的控制就不再那么自然了。

控制视方向

相机的一个常用的功能是在一个特定的方向生成一个视方向。可以调用SetFocalPoint()，SetPosition()和ComputeViewPlanNormal()方法，紧接着调用渲染器的ResetCamera()方法把该相机设置到渲染器中。

```
vtkCameracam1
  cam1 SetFocalPoint 0 0 0
  cam1 SetPosition 1 1 1
  cam1 ComputeViewPlaneNormal
  cam1 OrthogonalizeViewUp
ren1 SetActiveCamera cam1
ren1 ResetCamera
```

相机的初始方向(视向量或视平面法向量)是通过焦点、相机位置及方法ComputeViewPlanNormal()来计算的。你也可以指定一个初始的View-up向量，然后将它与视向量正交而得到初始的方向。ResetCamera()方法可以沿着视向量移动相机，这样，渲染器里所有的Actor都会可见。

透视投影和正交投影

前面的例子，我们都假定相机是透视投影的，即通过相机的视角(View Angle)来控制渲染时Actor在视平面上的投影。透视投影产生更加自然的影像的同时，会引入形变的效果，这在某些应用程序中是不希望得到的效果。正交(或平行)投影是另外一种投影模式。正交投影模式下，视线都是平行的，物体的渲染就没有距离差别的效果。

相机的正交投影模式可以使用方法vtkCamera::ParallelProjectionOn()设置。正交(平行)投影模式下，相机的视角(ViewAngle)就不再控制对象的缩放了，可以用方法SetParallelScale()来代替控制对象的放大与缩小。

保存/恢复相机状态

另外一个应用程序普遍需要的功能是保存和恢复相机状态(也就是重置视图)。要保存相机状态，最起码需要保存剪裁范围、相机焦点、位置和View-up向量，计算视平面向量。然后可以用这些保存的信息实例化一个相机对象，恢复相机的状态，并把相机设置到适当的渲染器中(即SetActiveCamera())。

某些情况下，可能还需要保存其他的信息。比如，如果设置了相机的视角(View Angle) (或ParallelScale)，就需要保存这些信息。又或者你把相机用于立体视图，就需要设置及保存EyeAngle和Stereo标志。

4.5控制光照

相对于相机来说，光照的控制要简单一些。使用较多的方法主要有SetPosition()，SetFocalPoint()以及SetColor()。Position和FocalPoint分别控制光照的位置和方向。光照的Color是用RGB值来表示。光照可以通过方法SwitchOn()和SwitchOff()打开和关闭，光照的强度则是用方法SetIntensity()来设置。

缺省的vtkLight实例是方向光照(DirectionLight)，也就是说光照的位置和焦点所确定的向量与光照的光线是平行的，光照的位置位于无限远处。这意味着如果光照的焦点和位置做平移变换的话，照射在物体上的光照不会发生变化。

光照与渲染器的关联如下面代码所示。

```
vtkLightlight
  light SetColor 1 0 0
  light SetFocalPoint [cam1 GetFocalPoint]
  light SetPosition [cam1 GetPosition]
ren1 AddLight light
```

以上代码创建的是一个红色的前灯光照(Headlight)，即光照的位置和焦点与相机的一致。这也是一个非常有用的特技，主要应用于与渲染器交互时，当相机发生移动的时候可以重置光照的位置。

位置光照

用PositionnalOn()方法可以创建位置光照(即聚光灯)，这个方法通常与SetConeAngle()方法协同使用，用来控制聚光光照的照射范围。聚光锥角(Cone Angle)等于180度时意味着聚光光照不起作用。

第04章-VTK基础（4）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本小节内容对应原书的第52页至第63页】

4.6 控制3D Props

VTK中的渲染窗口渲染的对象通常称之为"Prop"("Prop"这个词来源于舞台剧，指的是出现在舞台上的东西)。VTK里有几种不同类型的Prop，包括vtkProp3D和vtkActor。其中vtkProp3D是一个抽象父类，表示的是三维场景中的对象；vtkActor是vtkProp3D的子类，用类似多边形(Polygon)和线(Lines)等基本数据来定义它的几何。

指定vtkProp3D的位置

我们已经知道了如何绕着一个对象来移动相机；反过来，也可以保持相机不动，而对Prop进行变换。下面的方法可以用于定义一个vtkProp3D(及其子类)对象的位置。

SetPosition(x, y, z)—— 指定vtkProp3D对象在世界坐标系中的位置。
AddPosition(deltaX,deltaY, deltaZ) —— 用指定的X、Y、Z三个方向的增量来平移Prop。
RotateX(theta),RotateY(theta), RotateZ(theta) —— 分别用指定的角度绕X、Y、Z轴旋转Prop。
SetOrientation(x, y,z) —— 通过先绕Z轴，然后绕X轴，最后绕Y轴旋转，从而来确定Prop的方向。
AddOrientation(a1, a2,a3) —— 在当前Prop方向增加a1,a2, a3增量。
RotateWXYZ(theta, x,y, z) —— 绕x, y, z指定的向量旋转theta角度。
SetScale(sx, sy, sz)—— 分别沿X、Y、Z三个方向缩放sx,sy, sz比例。
SetOrigin(x, y, z) —— 指定Prop的原点，Prop的原点指的是Prop旋转或缩放时的基准点。

以上这些方法联合使用，可以产生复杂的变换矩阵。最重要的一点是，以上方法使用时要注意它们的调用顺序，不同的顺序对Actor的位置有不同的影响。VTK里是用以下的顺序来应用这些变换的：

1.移动Prop到原点。

2.缩放。

3.绕Y轴旋转。

4.绕X轴旋转。

5.绕Z轴旋转。

6.从原点中移动回原来的位置。

7.平移。

第1步和第6步平移的大小分别是Origin的负值和正值。单纯的平移是由vtkProp3D的Position值来确定的。这些变换中最容易混淆的旋转操作。例如，将一个Prop先绕X旋转，再绕Y轴旋转，它的效果与先绕Y轴旋转，再绕X轴旋转的效果是完全不一样的(图4-4)。要了解更多关于Actor变换内容可以参考《VisualizationToolkit》一书。

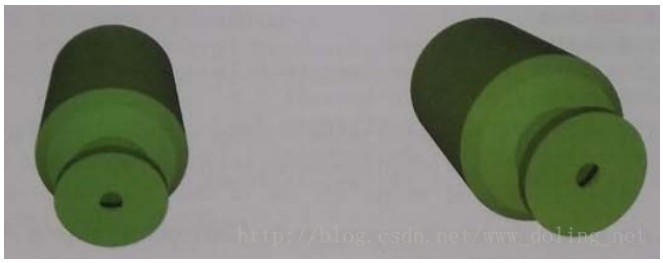

图4-4不同旋转顺序的效果。左边是先绕X轴旋转，再绕Y轴旋转的效果；右边是先绕Y轴旋转，再绕X轴旋转的效果
接下来，我们会介绍各种类型的vtkProp3D，其中VTK里最常用的类是vtkActor。在"控制vtkActor2D"一节里会介绍一下2DProp (也就是vtkActor2D)，这个类主要应用于标注(Annotation)及其他的二维操作。

Actors

Actor是最常的vtkProp3D类型，像其他的vtkProp3D子类一样，vtkActor提供了一组渲染属性，如表面属性(如环境光、散射光和镜面光颜色)，显示形式（Representation）(如表

面模型或线框模型)，纹理映射以及几何定义(即mapper)。

定义几何。前面的例子我们已经知道，一个Actor的几何是通过SetMapper()方法指定的：

```
vtkPolyDataMapper mapper
  mapper SetInputConnection [aFilter GetOutputPort]
vtkActor anActor
  anActor SetMapper mapper
```

在这个例子里，mapper的类型是vtkPolyDataMapper，也就是用类似点、线、多边形(Polygons)和三角形带(Triangle Strips)等几何图元进行渲染的。Mapper会结束可视化管线，在可视化子系统和图形子系统之间起到桥梁的作用。

Actor的属性。Actor里有一个类型为vtkProperty的实例，主要是用来控制Actor的显示属性。常用的属性是Actor的颜色，我们会在后面的内容详细描述。其他的重要属性有显示形式(点模型、线框模型或表面模型)、着色方法（平面着色或Gouraud着色）、Actor的不透明度(相对于透明度)以及环境光、散射光和镜面光颜色等相关参数。下面的脚本程序演示了如何设置这些变量。

```
vtkActor anActor
  anActor SetMapper mapper
  [anActor GetProperty] SetOpacity 0.25
  [anActor GetProperty] SetAmbient 0.25
  [anActor GetProperty] SetDiffuse 0.6
  [anActor GetProperty] SetSpecular 1.0
  [anActor GetProperty] SetSpecularPower 10.0
```

注意，我们通过方法GetProperty()间接引用Actor的属性。或者，我们也可以先实例化一个vtkProperty对象，然后把它设置到Actor中。

```
vtkProperty prop
  prop SetOpacity 0.25
  prop SetAmbient 0.5
  prop SetDiffuse 0.6
  prop SetSpecular 1.0
  prop SetSpecularPower 10.0
vtkActor anActor
  anActor SetMapper mapper
  anActor SetProperty prop
```

后一种方法的好处是，我们可以把多个Actor设置成同一种属性。

Actor的颜色。颜色可能是Actor里最重要的属性了。设置Actor的颜色最简单的方法莫过于调用SetColor()方法，该方法用RGB值来设置一个Actor的红、绿、蓝分量的颜色，每个分量的取值范围从0到1。

```
[anActor GetProperty] SetColor 0.1 0.2 0.4
```

或者，我们也可以通过设置环境光颜色、散射光颜色和镜面光颜色来控制Actor的颜色。

```
vtkActor anActor
anActor SetMapper mapper
[anActor GetProperty] SetAmbientColor .1 .1 .1
[anActor GetProperty] SetDiffuseColor .1 .2 .4
[anActor GetProperty] SetSpecularColor 1 1 1
```

以上代码把环境光颜色设置成深灰色，散射光颜色设置成蓝色的阴影，镜面光颜色设置成白色。(注意：SetColor()方法就是用指定的RGB值来设置环境光颜色、散射光颜色和镜面光颜色。)

重要：Actor的属性中关于颜色的设置只有当Actor的Mapper没有标量数据(ScalarData)时才起作用。缺省情况下，Mapper输入的标量数据会对Actor进行着色，而Actor的颜色设置会被忽略。如果要忽略这些标量数据，可以使用方法ScalarVisibilityOff()，如下面的Tcl脚本所示：

```
vtkPolyDataMapper planeMapper
  planeMapper SetInputConnection [CompPlaneGetOutputPort]
  planeMapper ScalarVisibilityOff
vtkActor planeActor
planeActor SetMapper planeMapper
  [planeActor GetProperty] SetRepresentationToWireframe
  [planeActor GetProperty] SetColor 0 0 0
```

Actor的透明度。很多情况下，调整Actor的透明度(或者不透明度)是很有用的。比如，如果你想显示一个病人图像的内部器官，而器官的外面包含着皮肤，这时你可以调整皮肤的

透明度，使得内部器官可见。可以使用vtkProperty::SetOpacity()方法，如下：

```
vtkActor popActor
  popActor SetMapper popMapper
  [popActor GetProperty] SetOpacity 0.3
  [popActor GetProperty] SetColor .9 .9 .9
```

要注意透明度的实现是使用渲染库里的α-Blending处理技术。这种处理技术要求以正确的顺序来渲染多边形(Polygons)。实际上，这是很难做到的，特别是当你有多个透明的Actor需要渲染时。要对多边形排序，应该把透明的Actor加到待渲染的Actor列表的最后。你也可以用vtkDepthSortPolyData这个Filter沿着视向量方向对多边形排序。关于这个Filter用法，可以参考VTK/Examples/VisualizationAlgorithm/Tcl/DepthSort.tcl这个例子。更多关于这方面的内容可以参考本章的"透明多边形几何"(Translucentpolygonal geometry)一节。

其他的属性。Actor还有其他一些重要的属性。你可以用方法VisibilityOn()/VisibilityOff()来控制Actor的可见与不可见。如果在拾取过程中，不想某个Actor被拾取，可以使用方法PickableOff()关闭拾取属性（关于拾取方面的内容，可以参考本章的"拾取"一节）。Actor有一个拾取事件(PickEvent)，当它们被拾取时就会调用这个事件。另外，方法GetBounds()可以获取与坐标轴对齐的Actor的包围盒(BoundingBox)。

Level-Of-DetailActors

图形系统一个主要的问题是，交互时有时会变得很慢。为了解决这个问题，VTK使用Level-of-detail技术，在与数据交互时，以低分辨率的显示形式（Representation）来表示Actor以求达到更快的渲染速度。

在本章的"读取源对象"(Reader SourceObject)一节，我们已经使用了vtkLODActor类。基本上，最简单的方法就是用vtkLODActor实例来替代vtkActor实例。另外，你也可以控制Level-of-detail的显示形式（Representation）。vtkLODActor缺省的做法是利用原始的Mapper创建另外两个低分辨率的模型。第一个是从定义Mapper输入的点采样得到的点云。你可以控制点云里点的个数(缺省是150个点)，如下所示。

```
vtkLODActor dotActor
  dotActor SetMapper dotMapper
  dotActor SetNumberOfCloudPoints 1000
```

Actor最低分辨率的模型就是一个包围盒。其他的level-of-detail也可以通过方法AddLODMapper()加入，由于复杂性问题，一般都没有必要加入。

为了控制渲染时Actor所选的level-of-detail，你可以设置渲染窗口的期望渲染帧率：

```
vtkRenderWindow renWin
  renWin SetDesiredUpdateRate 5.0
```

这样，就会以每秒5帧的速率进行渲染。vtkLODActor会自动地选择合适的Level-of-detail来达到请求的渲染速率。（注意：类似vtkRenderWindowInteractor等交互器，会自动地控制期望渲染帧率(DesiredUpdate Rate)，一般的做法是，当鼠标松开时，会把帧率设置得很低，而鼠标按下时，则会提高相应的帧率。这样就能保证相机运动时产生低分辨率/高帧率，而相机停止时产生高分辨率/低帧率的理想效果。如果你想了解更多关于Level-of-detail的控制方面的内容，可以参考本章的"vtkLODProp3D"一节，通过这个类，可以指定不同的Level。）

Assemblies

Actors有时也会组合在一起形成层次结构，当其中的某个Actor运动时，会影响到其他Actor的位置。例如，一个机械手臂可能由上臂、前臂、手腕和末端等部分通过关节连接起来。当上臂绕着肩关节旋转时，我们希望的是其他部分也会跟着运动。这种行为的实现就要用到Assembly，vtkAssembly是vtkActor的子类。下面的程序演示了如何使用vtkAssembly(摘自VTK/Examples/Rendering/Tcl/assembly.tcl)。

```
vtkSphereSource sphere
vtkPolyDataMapper sphereMapper
    sphereMapper SetInputConnection [sphere GetOutputPort]
vtkActor sphereActor
    sphereActor SetMapper sphereMapper
    sphereActor SetOrigin 2 1 3
    sphereActor RotateY 6
    sphereActor SetPosition 2.25 0 0
    [sphereActor GetProperty] SetColor 1 0 1

vtkCubeSource cube
vtkPolyDataMapper cubeMapper
    cubeMapper SetInputConnection [cube GetOutputPort]
vtkActor cubeActor
    cubeActor SetMapper cubeMapper
    cubeActor SetPosition 0.0 .25 0
    [cubeActor GetProperty] SetColor 0 0 1

vtkConeSource cone
vtkPolyDataMapper coneMapper
    coneMapper SetInputConnection [cone GetOutputPort]
vtkActor coneActor
    coneActor SetMapper coneMapper
    coneActor SetPosition 0 0 .25
    [coneActor GetProperty] SetColor 0 1 0

#top part of the assembly
vtkCylinderSource cylinder;
vtkPolyDataMapper cylinderMapper
    cylinderMapper SetInputConnection [cylinder GetOutputPort]
    cylinderMapper SetResolveCoincidentTopologyToPolygonOffset
vtkActor cylinderActor
    cylinderActor SetMapper cylinderMapper
    [cylinderActor GetProperty] SetColor 1 0 0

#Create the assembly and add the 4 parts to it. Also set the origin, position
#and orientation in space.
vtkAssembly assembly
    assembly AddPart cylinderActor
    assembly AddPart sphereActor
    assembly AddPart cubeActor
    assembly AddPart coneActor
    assembly SetOrigin 5 10 15
    assembly AddPosition 5 0 0
    assembly RotateX 15

#Create the Renderer, RenderWindow, and RenderWindowInteractor
#
vtkRenderer ren1
vtkRenderWindow renWin
    renWin AddRenderer ren1
vtkRenderWindowInteractor iren
    iren SetRenderWindow renWin

#Add the actors to the renderer, set the background and size
#
ren1 AddActor assembly
ren1 AddActor coneActor
```

注意是如何使用vtkAssembly里的方法AddPart()来建立层次结构的。只要不是自我嵌套，Assembly可以组合成任意层次深度的结构。vtkAssembly是vtkProp3D的子类，但没有与之相关联的Property以及Mapper。因此，vtkAssembly层次结构里的结点必须要包含关于材料的属性(如颜色等)及其他相关的几何信息。一个Actor可以用于多个Assembly(注意以上例子中的coneActor是如何作为一个单独的Actor以及作为Assembly里的一个节点的)。通过渲染器里的方法AddActor()只要加入最顶层的Assembly，而低层次的Assembly不用加入，因为它们会递归加入到渲染器中。

如果一个Actor加入到多个Assembly时(就像上面的例子)，你可以会想到如何去区分不同的Actor。（关于这一点，在类似“拾取”操作时是非常重要的，因为你必须要区分一下到底哪个vtkProp对象是处于选中的姿态。）我们会在后面的“拾取”一节，详细讨论这个问题，以及介绍类vtkAssemblyPath的用法。

Volumes

类vtkVolume主要用于体绘制，这个类与vtkActor非常类似。vtkVolume从vtkProp3D继承了对Volume进行定位和定向的方法。vtkVolume内部也有一个与它本身相关联的Property对象，即vtkVolumeProperty。请参考第七章“体绘制”了解更多的关于vtkVolume的应用以及体绘制方面的内容。

vtkLODProp3D

类vtkLODProp3D与vtkLODActor(参考本章“Level-Of-DetailActors”一节)类似，也是使用不同的显示形式（Representation）来表示它本身，以求达到更佳的交互渲染速率。与vtkLODActor不同的是，vtkLODProp3D只支持体绘制和面绘制。也就是说，你只能够在体绘制应用程序中使用vtkLODProp3D类来获取理想的交互帧率。以下的例子演示了这个类的使用。

```
vtkLODProp 3Dlod
  set level1 [lod AddLOD volumeMappervolumeProperty2 0.0]
  set level2 [lod AddLOD volumeMappervolumeProperty 0.0]
setlevel3 [lod AddLOD probeMapper_hres probeProperty 0.0]
setlevel4 [lod AddLOD probeMapper_lres probeProperty 0.0]
setlevel5 [lod AddLOD outlineMapper outlineProperty 0.0]
```

基本上，根据不同的渲染复杂度，会创建不同的Mapper，并把它设置到vtkLODProp3D里。AddLOD()方法可以接收Volume或几何类型的Mapper作为参数，可选的参数包括纹理映射、属性对象等。（根据提供的信息不同，该方法会有不同的函数签名(Signatures)。）AddLOD()方法的最后一个参数是渲染的估计时间。一般情况下都设置为0，即没有针对渲染的初始估计值。该方法返回一个整型的ID值，利用这个值就可以访问对应的LOD（可以用来选择某一Level或者删除某一Level）。

4.7 使用纹理

纹理映射是生成逼真的可视化效果的强大的图形工具。二维纹理映射的基本思想是在渲染过程中，图像可以“贴”到渲染对象的表面上，因此可以创建出细节更加丰富的渲染效果。纹理映射时需要提供三类信息：待贴纹理图的面、纹理映射(在VTK里，其实就是vtkImageData类型的数据，即2D图像)以及纹理坐标(控制纹理图在面上的位置)。

下面的例子演示了如何使用纹理映射(完整的程序代码见VTK/Examples/Rendering/Tcl/TPlane.tcl)。要注意纹理映射(类vtkTexture)是与Actor相联的，而纹理坐标则是由平面来定义(纹理坐标是由类vtkPlaneSource创建的)。

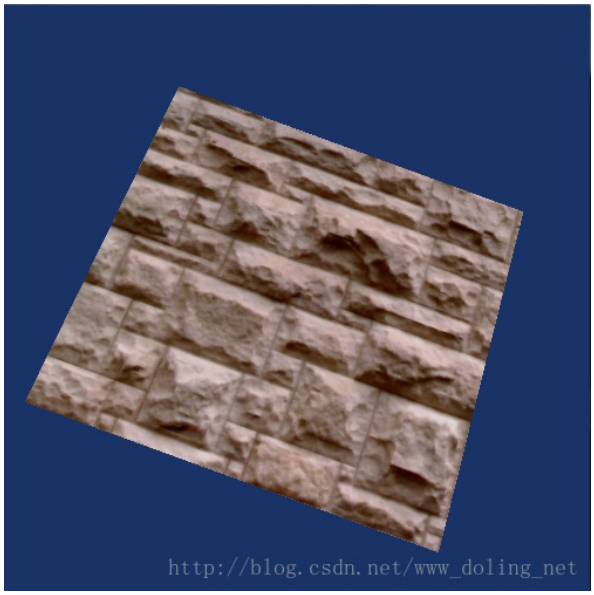

图4-5平面纹理映射

```
#Load in the texture map.
vtkBMPReader bmpReader
  bmpReader SetFileName"$VTK_DATA_ROOT/Data/masonry.bmp"
vtkTexture atext
  atext SetInputConnection [bmpReader GetOutputPort]
  atext InterpolateOn

#Create a plane source and actor.
vtkPlaneSource plane
vtkPolyDataMapper  planeMapper
  planeMapper SetInputConnection [plane GetOutputPort]
vtkActor planeActor
  planeActor SetMapper planeMapper
  planeActor SetTexture atext
```

很多时候，纹理坐标是获取不到的，因为这些纹理坐标不能在管线中生成。如果你需要生成纹理坐标，可以参考第五章“生成纹理坐标”一节。尽管一些老的图形卡在纹理贴图时会有所限制（比如要求所贴的纹理图必须是二维的，而且每维的大小必须小于1024），但VTK支持任意尺寸的纹理图。程序运行时，VTK会检索图形系统，确定这些图形系统的性能，然后会自动地对所设置的纹理图做采样，以求达到特定图形卡的要求。

4.8 拾取

拾取操作是可视化应用程序中常见的一种功能。拾取主要是用于选择数据和Actor或者获取底层的数据值。在显示位置(以像素为坐标值)中拾取时，就会调用vtkAbstractPicker的Pick()方法。依赖于所用的拾取类不同，拾取时返回的信息也不同，最简单的是返回一个x-y-z的全局坐标值，或者是单元（cell）的ID值，点的ID值，单元参数坐标(CellParametric Coordinates)，所拾取的vtkProp实例，以及Assemblypath。拾取方法的原型是：

```
Pick(selectionX,selectionY, selectionZ, Renderer)
```

注意Pick()方法需要一个渲染器作为参数。与渲染器相关联的Actor都是拾取的候选对象。另外，selectionZ通常都设置为0.0，它是与Z-buffer相关的值。（一般，Pick()这个方法都不会直接去调用它，用户使用vtkRenderWindowInteractor进行交互时，由这个类来管理拾取操作。这种情况下，用户只要选择一个拾取实例，让这个实例来控制拾取过程即可，后面的例子会演示如何使用。）

VTK支持多种不同功能的拾取类型（请参考图19-16，列出了与拾取相关的类的继承图）。类vtkAbstractPicker是所有拾取类的基类，它定义了一些公用的API，允许用户通过方法GetPickPosition()来获取拾取位置(全局坐标下)。

vtkAbstractPicker有两个直接子类。第一个是vtkWorldPointPicker，这是一种使用Z-buffer快速返回所拾取的位置的x-y-z全局坐标的类(基于硬件的)，但不会返回其他的信息(比如到底拾取了哪一个vtkProp实例等)。类vtkAbstractPropPicker是从vtkAbstractPicker中直接派生的另外一个子类。它定义了可以用于拾取某个vtkProp实例的API。下面列出一些比较方便的用于获取vtkProp实例的方法。

GetProp() —— 返回拾取的vtkProp实例指针。如果拾取了某个对象，就返回指向该vtkProp对象的指针，否则返回NULL。
GetProp3D() —— 如果拾取的是vtkProp3D对象，则返回该vtkProp3D对象的指针。
GetActor2D() —— 如果拾取的是vtkActor2D对象，则返回该vtkActor2D对象的指针。
GetActor() —— 如果拾取的是vtkActor对象，则返回该vtkActor对象的指针。
GetVolume() —— 如果拾取的是vtkVolume对象，则返回该vtkVolume对象的指针。
GetAssembly() —— 如果拾取的是vtkAssembly对象，则返回该vtkAssembly对象的指针。
GetPropAssembly() —— 如果拾取的是vtkPropAssembly对象，则返回该vtkPropAssembly对象的指针。

使用这些方法时需要特别注意，vtkAbstractPropPicker及其子类拾取时返回的是最顶层的Assembly路径（top level of theassembly path）。因此，如果有一个顶层类型是vtkAssembly的Assembly对象，其叶结点类型是vtkActor时，使用方法GetAssembly()返回的是指向vtkAssembly对象的指针，而使用方法GetActor()返回则是空指针。如果有一个包含Assembly、Actor及其他类型的Prop的复杂场景时，最安全的方法是使用GetProp()来确定所拾取的对象，再使用GetPath()方法。

类vtkAbstractPropPicker有三个直接子类，分别是：vtkPropPicker、vtkAreaPicker及vtkPicker。vtkPropPicker使用硬件拾取的策略来确定所拾取的vtkProp实例，包括拾取点的世界坐标系下的位置坐标。vtkPropPicker通常比vtkAbstractPropPicker的其他子类的速度要快，但是它不能返回所拾取对象的详细信息。

vtkAreaPicker及其基于硬件实现的子类vtkRenderedAreaPicker同样无法确定所拾取对象的详细信息，它们的作用是选择屏幕上的对象。vtkAreaPicker及其子类与其他的拾取类不同，前者可以确定哪些是位于屏幕上矩阵区域的像素的开始位置，而不仅仅确定哪些是位于某个像素的后方。这些类都有方法AreaPick(x_min, y_min,x_max, y_max, Renderer)，可以与标准的方法Pick(x,y,z,Renderer)一起使用。如果想获取更多的信息，比如确定位于某个区域后方的单元或点等信息，可以参考本节后续内容的介绍。

vtkPicker是一个基于软件实现的拾取类，具体实现是基于vtkProp对象的包围盒来拾取对象。该类在拾取时，会从相机的当前位置投射一条光线到拾取点，所投射的光线会与某个Prop3D对象的包围盒相交，当然，通过这种方式有可能会有多个的Prop3D对象被拾取到，最后返回的是所投射的光线与对象的包围盒相交最多的Prop3D。而方法GetProp3Ds()可以返回与投射光线相交的所有的Prop3D对象。vtkPicker拾取速度相对较快，但无法获取单一的拾取。

vtkPicker有两个子类，通过这两个子类可以获取所拾取对象更多详细的信息，比如，点的ID，单元的ID等。vtkPointPicker用于拾取单个点，返回其ID值和坐标值。拾取时，vtkPointPicker也是通过从相机当前位置投射一条光线至拾取点，然后将光线周围且位于容差（Tolerance）范围内的点投射至光线上，最后返回的是距离相机最近的点以及该点所对应的Actor对象。（注意：容差是用渲染窗口的对角线的长度作为分数的。）vtkPointPicker比vtkPicker拾取速度要慢，但比vtkCellPicker要快。因为引入的容差，所以vtkPointPicker可以返回单一的拾取对象。

vtkCellPicker用于拾取某个单元，并返回交点的信息，比如，交点所对应的单元ID、全局坐标以及参数化单元坐标（Parametric cellcoordinates）。与vtkPointPicker类似，vtkCellPicker拾取时也是投射一条光线至拾取点，在一定的容差范围内确定光线是否与Actor底层的几何相交，最后返回的就是沿着光线最靠近相机的单元及其对应的对象。（注意：在确定光线是否与单元相交时会使用到容差，可能需要多次实验才能获得满意的结果。）vtkCellPicker是所有拾取类中速度最慢的一个，但是所获取的信息也是最多的。通过指定容差，可以返回单一的拾取对象。

VTK定义了与拾取操作相关的几个事件。拾取操作发生之前会调用StartPickEvent事件，拾取完成后则调用EndPickEvent事件。当对象被拾取时会调用Picker类的PickEvent事件以及Actor类的PickEvent事件。注意：使用vtkWorldPointPicker类时，不会有PickEvent事件发生。

vtkAssemblyPath

在拾取包含不同类型的vtkProp对象的场景时，有必要理解类vtkAssemblyPath，特别是当场景中包含有vtkAssembly对象。vtkAssemblyPath简单地可以理解为包含vtkAssemblyNode的顺序列表，每个结点含有一个指向vtkProp对象的指针以及一个可选的vtkMatrix4x4对象。列表的顺序非常重要，列表的开始是根结点或者说是Assembly层次结构的顶层结点，列表的结尾表示Assembly层次结构的叶结点。结点的顺序会影响到与之关联的矩阵。每个矩阵是列表里结点的Prop所对应的矩阵与前一个矩阵的级联。因此，对于某个给定的vtkAssemblyNode，所关联的vtkMatrix4x4表示的是该结点的vtkProp对象的位置和方向（假设vtkProp对象初始状态是没有经过变换的）。

例子

通常，拾取是由vtkRenderWindowInteractor自动管理的(见“使用VTK交互器”一节了解更多关于交互器的内容)。比如，当按下P键时，vtkRenderWindowInteractor会调用内部的vtkPropPicker实例执行拾取操作。接着，可以通过vtkRenderWindowInteractor访问拾取器(Picker)或者其他信息。也可以给vtkRenderWindowInteractor指定一个从vtkAbstractPicker派生的拾取器。图4-6显示了对数据集拾取的结果，程序代码摘自VTK/Examples/Annotation/Tcl/annotationPick.tcl。

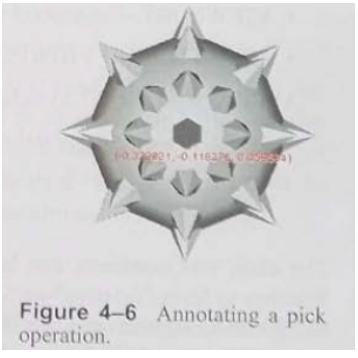

Figure 4–6 Annotating a pick operation.

图4-6 带标注信息的拾取操作

```
vtkCellPicker picker
    picker AddObserver EndPickEvent annotatePick

#Create a text mapper and actor to display the results of picking.
vtkTextMapper textMapper
settprop [textMapper GetTextProperty]
    $tprop SetFontFamilyToArial
    $tprop SetFontSize 10
    $tprop BoldOn
    $tprop ShadowOn
    $tprop SetColor 1 0 0
vtkActor2D textActor
    textActor VisibilityOff
    textActor SetMapper textMapper

#Create the Renderer, RenderWindow, and RenderWindowInteractor
#
vtkRenderer ren1
vtkRenderWindow renWin
    renWin AddRenderer ren1
vtkRenderWindowInteractor iren
    iren SetRenderWindow renWin
    iren SetPicker picker

#Create a Tcl procedure to create the text for the text mapper used to
#display the results of picking.
proc annotatePick {} {
    if { [picker GetCellId] < 0 } {
        textActor VisibilityOff

    } else {
        set selPt [picker GetSelectionPoint]
        set x [lindex $selPt 0]
        set y [lindex $selPt 1]
        set pickPos [picker GetPickPosition]
        set xp [lindex $pickPos 0]
        set yp [lindex $pickPos 1]
        set zp [lindex $pickPos 2]

        textMapper SetInput "($xp, $yp,$zp)"
        textActor SetPosition $x $y
        textActor VisibilityOn
    }

    renWin Render
}

#Pick the cell at this location.
picker Pick 85 126 0 ren1
```

这个例子使用vtkTextMapper在屏幕上绘制拾取点的世界坐标值。(参考“文本标注”一节了解更多信息)。注意到在这个例子中，我们注册了EndPickEvent事件，拾取操作完成后，就会调用annotatePick()过程。

4.9 vtkCoordinate和坐标系统

VTK支持多种不同类型的坐标系统，类vtkCoordinate管理这些坐标系统之间的变换。支持的坐标系统有：

DISPLAY —— X-Y坐标值定义在渲染窗口中，以像素为单位(注意vtkRenderWindow是vtkWindow的子类)。原点在窗口的左下角(这一点对于下面的二维坐标系统都是如此)。
NORMALIZED DISPLAY —— 窗口的X-Y坐标取值归一化，即(0,1)。
VIEWPORT —— X-Y坐标值定义在视口(Viewport)或者渲染器(Renderer，vtkRenderer是vtkViewport的子类)里。
NORMALIZED VIEWPORT ——视口里的X-Y坐标取值归一化，即(0, 1)。
VIEW —— X-Y-Z坐标值(取值范围为(-1,1))定义在相机坐标系统，Z表示深度。
WORLD —— X-Y-Z为全局坐标值。
USERDEFINED —— X-Y-Z定义在用户自定义的空间里。用户必须为自定义的坐标系统提供空间变换方法。请参考类vtkCoordinate了解更多信息。

类vtkCoordinate可以用于坐标系统之间的变换，也可以用于连接各个坐标系统以形成“相对”或者“偏移”等坐标值。请参考下一节内容了解vtkCoordinate的用法。

4.10 控制vtkActor2D

vtkActor2D与vtkActor很多功能都类似，除了vtkActor2D是在层叠（overlay）平面上绘制的以及没有与之相关联的4×4的变换矩阵。与vtkActor类似，vtkActor2D涉及到一个mapper(即vtkMapper2D)和属性对象(即vtkProperty2D)。使用vtkActor2D时，比较困难的部分是如何定位它的对象。定位vtkActor2D对象时会用到类vtkCoordinate(请参考上一部分“vtkCoordinate和坐标系统”一节)。下面的例子演示了如何使用vtkCoordinate对象。

```
vtkActor2D bannerActor
  bannerActor SetMapper banner
  [bannerActor GetProperty] SetColor 0 1 0
  [bannerActor GetPositionCoordinate]SetCoordinateSystemToNormalizedDisplay
  [bannerActor GetPositionCoordinate] SetValue0.5 0.5
```

在这个例子中，访问了坐标对象以及定义了它的坐标系统，然后设置了该坐标系统下合适的坐标值。这个例子中，使用了归一化显示坐标系统(Normalized DisplayCoordinate Sys

tem)，因此坐标范围定义为0到1，坐标值(0.5,0.5)就设置vtkActor2D对象在渲染窗口的中间位置。vtkActor2D也提供了一个方便的接口，SetDisplayPosition()可以设置坐标系统为DISPLAY，并且利用传入的参数(以像素为单位)设置vtkActor2D对象在渲染窗口中的位置。下一节的内容将会演示如何使用这个方法。

第04章-VTK基础（5）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本小节内容对应原书的第63页至第70页】

4.11 文本标注

VTK提供了两种方法用于标注图像。第一种是可以在三维图形窗口的顶层上绘制的文本(和图形)，通常涉及的是在层叠平面上绘制。第二种是创建三维的多边形数据的文本，可以像其他三维图形对象一样进行变换及显示。这两种类型的标注分别称为二维和三维标注，从图4-7中可以看出它们的区别。

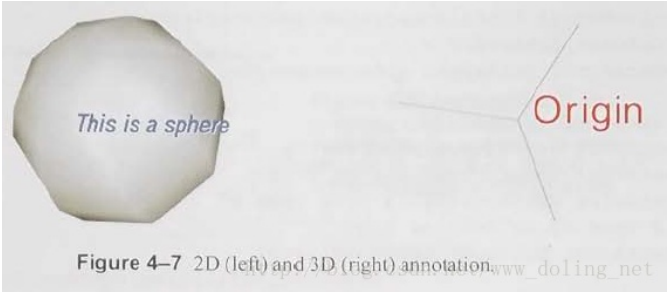

Figure 4–7 2D (left) and 3D (right) annotation.

图4-7 二维(左)和三维(右)标注

二维文本标注

使用二维文本标注时，需要用到二维的Actor(vtkActor2D或它的子类，如vtkScaledTextActor)和Mapper(vtkMapper2D或其子类vtkTextMapper)。二维的Actor和Mapper跟三维的类似，不同的是前者是在图形或图像的顶层层叠平面(overlay plane)上进行渲染的。下面的例子摘自VTK/Examples/Annotation/Tcl/TestText.tcl，结果如图4-7所示。

```
vtkSphereSource sphere

vtkPolyDataMapper sphereMapper
    sphereMapper SetInputConnection [sphere GetOutputPort]
    sphereMapper GlobalImmediateModeRenderingOn
vtkLODActor sphereActor
    sphereActor SetMapper sphereMapper

#Create a scaled text actor.
#Set the text, font, justification, and properties (bold, italics, etc.).
vtkTextActor textActor
    textActor SetTextScaleModeToProp
    textActor SetDisplayPosition 90 50
    textActor SetInput "This is asphere"

    # Set coordinates to match the oldvtkScaledTextActor default value
    [textActor GetPosition2Coordinate] SetCoordinateSystemToNormalizedViewport
    [textActor GetPosition2Coordinate] SetValue0.6 0.1

set tprop [textActor GetTextProperty]
    $tprop SetFontSize 18
    $tprop SetFontFamilyToArial
    $tprop SetJustificationToCentered
    $tprop BoldOn
    $tprop ItalicOn
    $tprop ShadowOn
    $tprop SetColor 0 0 1

#Create the Renderer, RenderWindow, RenderWindowInteractor
#
vtkRenderer ren1
vtkRenderWindow renWin
    renWin AddRenderer ren1
vtkRenderWindowInteractor iren
    iren SetRenderWindow renWin

#Add the actors to the renderer; set the background and size; zoom in;
#and render.
#
ren1 AddViewProp textActor
ren1 AddViewProp sphereActor
```

vtkTextProperty实例可以设置字体(Arial,Courier或者Times等)，文本颜色，粗体、斜体开关以及字体的阴影效果等。字体的阴影效果可以使标注的文本在复杂背景下可读性更强

。标注文本的位置和颜色则通过关联的vtkActor2D控制的。在这个例子中，文本的位置是用显示坐标或者像素坐标进行设置的。

vtkTextProperty也支持对齐(垂直和水平)及多行文本。方法SetJustificationToLeft()，SetJustificationToCentered()和SetJustificationToRight()可以控制水平对齐方向。SetVerticalJustificationToBottom()，SetVerticalJustificationToCentered()和SetVerticalJustificationToTop()等方法可以控制垂直对齐方向，缺省的对齐方向是左下角对齐。文本中嵌入“\n”字符可以支持多行文本。图4-8是对齐和多行文本的效果，该例子摘自VTK/Examples/Annotation/Tcl/multiLineText.tcl。以下是例子的主要代码：

```
vtkTextMapper textMapperL
    textMapperL SetInput "Thisis\nmulti-line\ntext output\n(left-top)"
    set tprop [textMapperL GetTextProperty]
    $tprop ShallowCopy multiLineTextProp
    $tprop SetJustificationToLeft
    $tprop SetVerticalJustificationToTop
    $tprop SetColor 1 0 0
vtkActor2D textActorL
    textActorL SetMapper textMapperL
    [textActorL GetPositionCoordinate]SetCoordinateSystemToNormalizedDisplay
    [textActorL GetPositionCoordinate] SetValue0.05 0.5
```

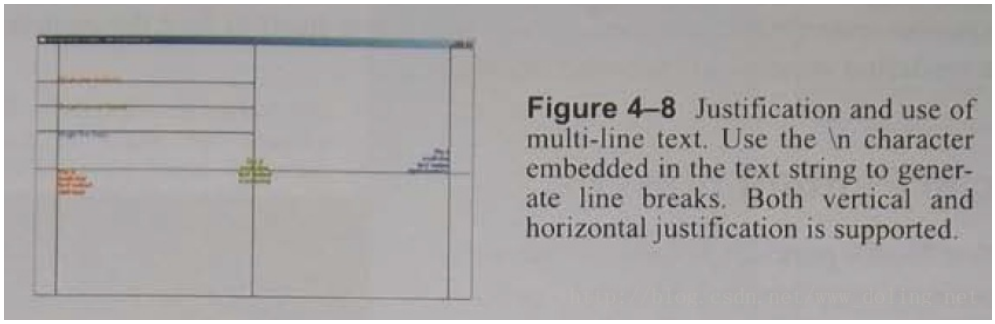

Figure 4–8 Justification and use of multi-line text. Use the \n character embedded in the text string to generate line breaks. Both vertical and horizontal justification is supported.

图4-8 文本对齐和多行文本。在文本里嵌入“\n”字符可以生成多行文本，支持水平和垂直对齐

注意，以上代码使用vtkCoordinate对象(调用方法GetPositionCoordinate()获得)控制Actor在NormalizedDisplay坐标系统下位置。参考“vtkCoordinate和坐标系统”一节，了解更多关于放置文本标注的信息。

三维文本标注与vtkFollower

三维文本标注的实现是使用vtkVectorText创建文本字符串的多边形表达形式(Polygonal Representation)，然后把它放置在渲染场景中。放置三维文本标注的类是vtkFollower。该类是vtkActor的子类，vtkFollower的对象总是面向渲染器的当前相机，因此可以保证这种文本标注都是可读、可见的。以下的代码演示了如何使用这个类，代码摘自VTK/Examples/Annotation/Tcl/textOrigin.tcl，运行结果如图4-7所示。这个例子创建了一个坐标轴和一个vtkVectorText实例，结合vtkFollower对象标注该坐标轴的原点。

```
vtkAxes axes
    axes SetOrigin 0 0 0
vtkPolyDataMapper axesMapper
    axesMapper SetInputConnection [axes GetOutputPort]
vtkActor axesActor
    axesActor SetMapper axesMapper

#Create the 3D text and the associated mapper and follower (a type of
#actor).  Position the text so it isdisplayed over the origin of the axes.
vtkVectorText atext
    atext SetText "Origin"
vtkPolyDataMapper textMapper
    textMapper SetInputConnection [atext GetOutputPort]
vtkFollower textActor
    textActor SetMapper textMapper
    textActor SetScale 0.2 0.2 0.2
    textActor AddPosition 0 -0.1 0
…etc…after rendering…
textActor SetCamera [ren1 GetActiveCamera]
```

当相机绕着坐标轴旋转时，vtkFollower对象会调整自己的方向，使其朝向相机。你可以试着在渲染窗口里，用鼠标移动相机，观察这种变化。

4.12 专用的绘图类

VTK提供了一些复合类用于实现绘图操作。包括绘制标量条、执行简单的X-Y平面绘图以及在三维空间中放置坐标轴等。

标量条

类vtkScalarBarActor用于创建与数值数据相关联的带关键颜色值的颜色条，如图4-9所示。这种标量条由三部分组成，分别是：一个带颜色的矩形条、标注和标题。使用vtkScalarBarActor时，必须含有如下信息：引用一个vtkLookupTable实例(该实例用于定义颜色和数值数据的范围)，在层叠平面（overlay plane）上放置颜色条的位置并指定其方向，标注的个数以及标量的文本字符串等。以下例子演示了该类的用法。

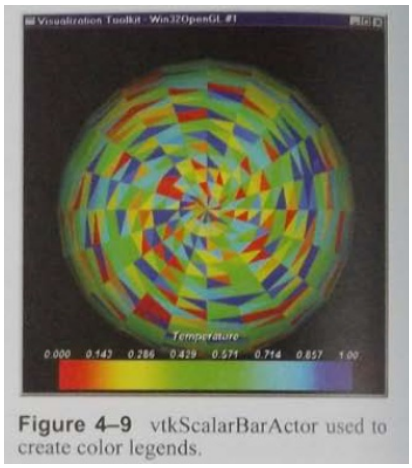

Figure 4–9 vtkScalarBarActor used to create color legends.

图4-9vtkScalarBarActor用于创建颜色条

```
vtkScalarBarActor scalarBar
  scalarBar SetLookupTable [mapper GetLookupTable]
  scalarBar SetTitle "Temperature"
  [scalarBar GetPositionCoordinate] SetCoordinateSystemToNormalizedViewport
  [scalarBar GetPositionCoordinate] SetValue0.1 0.1
  scalarBar SetOrientationToHorizontal
  scalarBar SetWidth 0.8
  scalarBar SetHeight 0.17
```

标量条的方向是通过方法SetOrientationToVertical()和SetOrientationToHorizontal()来指定的。位置(即标量条左下角)则是通过不同的坐标系统(允许使用任何坐标系统，见“vtkCoordinate和坐标系统”一节)设置，宽度和高度使用了NormalizedViewport坐标系下的坐标值来指定(或者也可以通过指定标量条右上角的位置，即设置Position2变量的值，间接来设置标量条的宽度和高度)。

X-Y平面绘图

类vtkXYPlotActor可根据输入的一个或多个数据集生成X-Y平面图形，如图4-10所示。该类在显示某个数据变量与一系列点之间的变化情况时非常有用，比如某个变量与探测线或者边界线之间的变化情况。

使用vtkXYPlotActor2D时，必须输入一个或多个数据集，指定坐标轴以及所绘制图形的标题、位置等。该类里的变量PositionCoordinate指定了X-Y图形左下角的位置(定义在Normalizedviewport坐标系下)，而变量Position2Coordinate则指定图形的右上角位置坐标。注意：Position2Coordinate是相对于PositionCoordinate而言的，因此可以通过设置PositionCoordinate在视口里移动vtkXYPlotActor。这两个位置确定了图形所在的矩形区域。以下例子演示了该类的用法(程序摘自VTK/Examples/Annotation/Tcl/xyPlot.tcl)

```
vtkXYPlotActor xyplot
    xyplot AddInput [probe GetOutput]
    xyplot AddInput [probe2 GetOutput]
    xyplot AddInput [probe3 GetOutput]
    [xyplot GetPositionCoordinate] SetValue 0.00.67 0
    [xyplot GetPosition2Coordinate] SetValue1.0 0.33 0;#relative to Position
    xyplot SetXValuesToArcLength
    xyplot SetNumberOfXLabels 6
    xyplot SetTitle "Pressure vs. ArcLength (Zoomed View)"
    xyplot SetXTitle ""
    xyplot SetYTitle "P"
    xyplot SetXRange .1 .35
    xyplot SetYRange .2 .4
    [xyplot GetProperty] SetColor 0 0 0
```

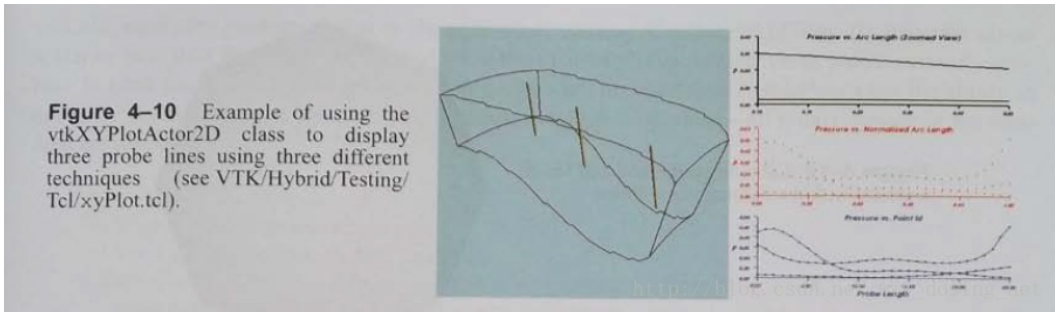

Figure 4–10 Example of using the vtkXYPlotActor2D class to display three probe lines using three different techniques (see VTK/Hybrid/Testing/Tcl/xyPlot.tcl).

图4-10使用类vtkXYPlotActor2D来显示分别用三种不同的技术所获取的探测线数据（见VTK/Hybrid/Testing/Tcl/xyPlot.tcl）

注意X轴的定义，缺省情况下，以输入数据集的点的索引作为X轴。也可以使用线的弧长或归一化弧长作为vtkXYPlotActor的输入来生成X值。

带坐标轴的包围盒（vtkCubeAxesActor2D）

另外一个复合的Actor类是vtkCubeAxesAxtor2D。该类可以用于标记相机所观察的空间位置，如图4-11所示。vtkCubeAxesActor2D可以沿着输入数据的包围盒的三个正交边显示X-Y-Z坐标值。当相机放缩时，坐标轴上的坐标值会自适应相机视口的变化而更新。用户可以控制显示的坐标值所用的字体属性以及字体大小等。字体大小可通过方法SetFontFactor()设置。下面的程序演示了这个类的用法(摘自VTK/Examples/Annotation/Tcl/cubeAxes.tcl)。

```
vtkTextProperty tprop
   tprop SetColor 1 1 1
   tprop ShadowOn
 vtkCubeAxesActor2D axes
   axes SetInput [normals GetOutput]
   axes SetCamera [ren1 GetActiveCamera]
   axes SetLabelFormat "%6.4g"
   axes SetFlyModeToOuterEdges
   axes SetFontFactor 0.8
   axes SetAxisTitleTextProperty tprop
   axes SetAxisLabelTextProperty tprop
```

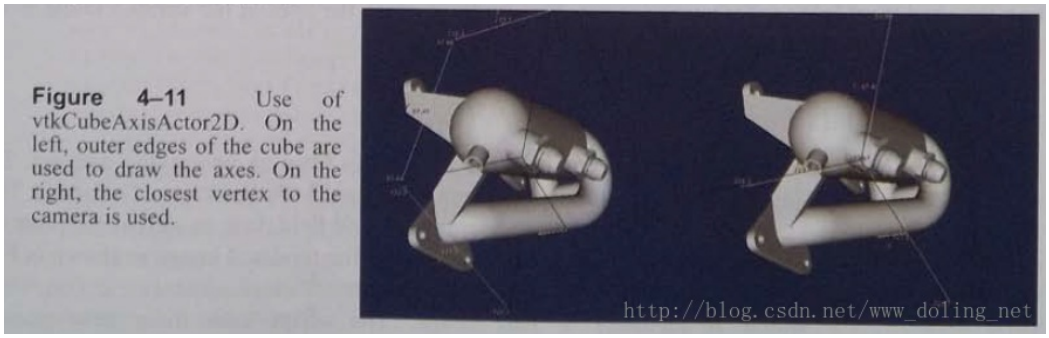

图4-11vtkCubeAxesActor2D类的使用。左图中立方体边框的外边缘用于绘制坐标轴，右图将坐标轴放置在距离相机最近的顶点位置上。
注意绘制坐标轴时有两种模式，缺省模式是SetFlyModeToOuterEdges()，即绘制在包围盒的边框外边缘；另一种模式是SetFlyModeToClosestTriad()，把坐标轴放置在距离相机最近的顶点位置上。

标记数据

在某些应用程序中可能需要根据底层的数据显示某些数值。vtkLabeledDataMapper就可以用于标记数据集里的点，包括标量、向量、张量、法向量、纹理坐标、域数据（Field Data）以及数据集里点的ID值。文本标记信息放置于所渲染图像的层叠平面上，如图4-12所示。该图是通过运行VTK/Examples/Annotation/Tcl/labeledMesh.tcl例子生成的，本节后面有该例子的部分代码。在这个例子里，用到了三个新类来标记球体的单元和点的ID值，分别是：vtkCellCenters、vtkIdFilter和vtkSelectVisiblePoints。其中vtkCellCenters用于在单元的中心位置生成点，vtkIdFilter则是根据单元和点的ID生成标量值或域数据，也就是说，点的属性数据标量值或域数据是从点的ID值生成的，单元的属性数据标量值或域数据则是根据单元的ID值生成的，vtkSelectVisiblePoints用于选择当前可见的点。另外vtkSelectVisiblePoints还可以在DISPLAY坐标系统下定义一个“窗口”来显示要标记的点的数值，并忽略窗口外的点使其不可见。

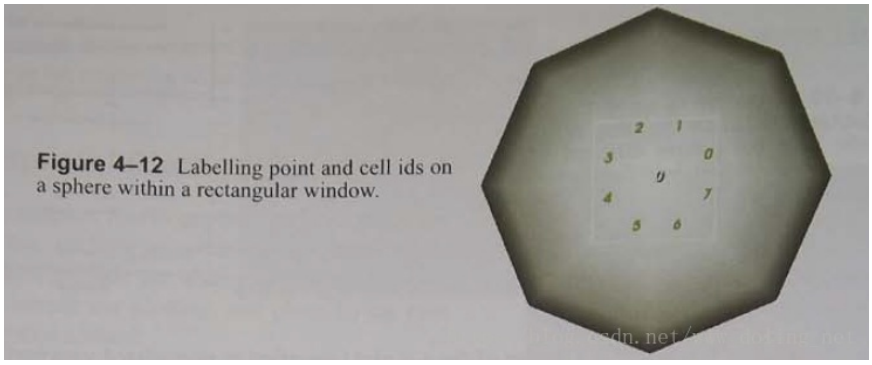

图4-12 在一个矩形窗口中标记其所对应的球体下的点和单元的ID值

```
#Create a sphere and its associated mapper and actor.
vtkSphereSource sphere
vtkPolyDataMapper   sphereMapper
    sphereMapper SetInputConnection [sphereGetOutputPort]
    sphereMapper GlobalImmediateModeRenderingOn
vtkActors phereActor
    sphereActor SetMapper sphereMapper

#Generate data arrays containing point and cell ids
vtkIdFilter ids
    ids SetInputConnection [sphere GetOutputPort]
    ids PointIdsOn
    ids CellIdsOn
    ids FieldDataOn

#Create the renderer here because vtkSelectVisiblePoints needs it.
vtkRenderer ren1

#Create labels for points
vtkSelectVisiblePoints visPts
    visPts SetInputConnection [idsGetOutputPort]
    visPts SetRenderer ren1
    visPts SelectionWindowOn
    visPts SetSelection $xmin [expr $xmin +$xLength] \
          $ymin [expr $ymin + $yLength]

#Create the mapper to display the point ids. Specify the
#format to use for the labels.  Alsocreate the associated actor.
vtkLabeledDataMapper ldm
    ldm SetInputConnection [visPtsGetOutputPort]
#    ldm SetLabelFormat "%g"
    ldm SetLabelModeToLabelFieldData
vtkActor2DpointLabels
    pointLabels SetMapper ldm

#Create labels for cells
vtkCellCenters cc
    cc SetInputConnection [ids GetOutputPort]
vtkSelectVisiblePoints visCells
    visCells SetInputConnection [ccGetOutputPort]
    visCells SetRenderer ren1
    visCells SelectionWindowOn
    visCells SetSelection $xmin [expr $xmin +$xLength] \
          $ymin [expr $ymin + $yLength]
#Create the mapper to display the cell ids. Specify the
#format to use for the labels.  Alsocreate the associated actor.
vtkLabeledDataMapper cellMapper
    cellMapper SetInputConnection [visCells GetOutputPort]
#    cellMapper SetLabelFormat "%g"
    cellMapper SetLabelModeToLabelFieldData
    [cellMapper GetLabelTextProperty] SetColor0 1 0
vtkActor2D cellLabels
    cellLabels SetMapper cellMapper

#Create the RenderWindow and RenderWindowInteractor
#
vtkRenderWindow renWin
    renWin AddRenderer ren1
vtkRenderWindowInteractor iren
    iren SetRenderWindow renWin

#Add the actors to the renderer; set the background and size;
#render
ren1AddActor sphereActor
ren1AddActor2D rectActor
ren1AddActor2D pointLabels
ren1AddActor2D cellLabels

ren1SetBackground 1 1 1
renWinSetSize 500 500
renWinRender
```

第04章-VTK基础（6）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本小节内容对应原书的第70页至第83页】

4.13 数据变换

在4.6节里的“Assemblies”一小节里，有以下一段内容：

“注意是如何使用vtkAssembly里的方法AddPart()来建立层次结构的。只要不是自我嵌套，Assembly可以组合成任意层次深度的结构。vtkAssembly是vtkProp3D的子类，但没有与之相关联的Property以及Mapper。因此，vtkAssembly层次结构里的结点必须要包含关于材料的属性(如颜色等)及其他相关的几何信息。一个Actor可以用于多个Assembly(注意以上例子中的coneActor是如何作为一个单独的Actor以及作为Assembly里的一个节点的)。通过渲染器里的方法AddActor()只要加入最顶层的Assembly，而低层次的Assembly不用加入，因为它们会递归加入到渲染器中。”

在世界坐标系中，我们可能需要确定vtkProp3D对象的位置及方向。某些应用程序中，有时希望将数据送入可视化管线进行可视化之前对其做某些空间变换操作。例如，使用平面对数据进行切割（见第98页的“切割”一节）或者裁剪某个对象（见第110页的“数据裁剪”一节）时，在管线里，用于切割的平面必须放置在正确的空间位置，而这不是通过Actor的变换矩阵来完成的。某些对象（特别是程序源对象）需要在空间的指定位置和方向创建，比如，vtkSphereSource类有中心点Center和半径Radius等变量，而vtkPlaneSource则有Origin、Point1和Point2等变量允许用户使用三个点来确定平面的空间位置。然而，大多数类不提供类似的方法来移动数据至适当的位置，在这种情况下，我们必须使用类vtkTransformFilter或者vtkTransformPolyDataFilter将其变换至合适的空间位置中。

vtkTransformFilter以vtkPointSet数据集对象作为输入。抽象类vtkPointSet的数据集子类是以显式的形式来表达点数据，也就是说，使用vtkPoints的实例来存储坐标信息。vtkTransformFilter应用一个变换矩阵到这些点集上，并产生出一个变换后的点集，而数据集结构（即单元拓扑）和属性数据（如标量值、向量值等）则保持不变。vtkTransformPolyDataFilter与vtkTransformFilter工作过程类似，在包含多边形数据的可视化管线中，使用前者则更为方便些。

以下例子使用vtkTransformPolyDataFilter来确定一个3D文本字符串的空间位置（摘自VTK/Examples/DataManipulation/Tcl/marching.tcl，程序如图4-13所示）。关于3D文本字符串的内容可以参考“三维文本标注与vtkFollower”一节。

Figure 4–13 Transforming data within the pipeline.

图4-13在管线里变换数据

```
#definethe text for the labels
vtkVectorText caseLabel
  caseLabel SetText “Case 12c – 11000101”
vtkTransform aLabelTransform
  aLabelTransform Identity
  aLabelTransform Translate -.2 0 1.25
  aLabelTransform Scale .05 .05 .05
vtkTransformPolyDataFilter labelTransform
  labelTransform SetTransform aLabelTransform
  labelTransform SetInputConnection [caseLabel GetOutputPort]
vtkPolyDataMapper labelMapper
  labelMapper SetInputConnection [labelTransform GetOutputPort]
vtkActor labelActor
  labelActor SetMapper labelMapper
```

需要注意的是vtkTransformPolyDataFilter需要指定一个vtkTransform实例。由前面的章节我们知道，vtkTransform是用于控制空间中Actor的位置及方向，vtkTransform还提供其他的方法，接下来列出该类的一些常用的方法：

RotateX(angle) ——绕X轴旋转angle角度（以度为单位）。
RotateY(angle) ——绕Y轴旋转angle角度（以度为单位）。
RotateZ(angle) ——绕Z轴旋转angle角度（以度为单位）。
RotateWXYZ(angle,x,y,z)—— 绕(x,y,z)向量所定义的轴旋转angle角度。
Scale(x,y,z) ——在X-Y-Z方向进行放缩。
Translate(x,y,z) ——平移。
Inverse() ——变换矩阵的逆。
SetMatrix(m) ——直接指定一个4×4的变换矩阵。
GetMatrix(m) ——获取一个4×4的变换矩阵。
PostMultiply() ——控制矩阵相乘的顺序，矩阵左乘。
PreMultiply() ——矩阵右乘。

以上介绍的最后两个方法告诉我们，变换矩阵的顺序对最终的变换结果有很大的影响。我们建议你多花点时间在这些方法的理解以及矩阵的变换顺序，以求能更加深刻地理解vtkTransform。

高级变换

对于高级用户来说，可能会使用到扩展的VTK变换层次结构（这部分内容大多数的工作是由David Gobbi完成的）。VTK变换的层次结构（如图19-17所示）提供了一系列的线性与非线性变换。

VTK变换层次结构一个非常棒的特性就是不同类型的变换可以在同一个Filter中使用，以产生不同的结果。例如，vtkTransformPolyDataFilter可以接受vtkAbstractTransform（或其子类）的变换类型作为其输入，包括线性变换、仿射变换（用4×4的矩阵表示）、非线性变换（如vtkThinPlateSplineTransform）。

3DWidgets

交互器样式（见第45页的“使用VTK交互器”一节）通常只是控制相机以及提供一些简单的面向键盘和鼠标事件的交互技术。交互器样式在渲染场景中并没有一种表达形式，也就是说，在交互时我们看不见交互器样式到底是什么样子的，用户在使用这些交互器样式时，必须事先知道哪些键盘和鼠标事件是控制哪些操作的。然而，渲染场景中大部分的交互操作都需要直接、简单。比如，如果某条线的端点是可以由用户自由放置的话，那么沿着某条线改变流线的倾斜度，就显得非常容易。

3DWidget就是针对这种功能需求而设计的。与类vtkInteractorStyle类似，3DWidget也是vtkInteractorObserver的子类。换言之，它们会监听由vtkRenderWindowInteractor所调用的事件。我们知道，vtkRenderWindowInteractor可以将基于具体操作系统的事件转换成VTK事件。但是与vtkInteractorStyle不同的是，3DWidget在渲染场景中可以用多种不同的表达形式，图4-14列出了一些VTK可见的3DWidget的样式。

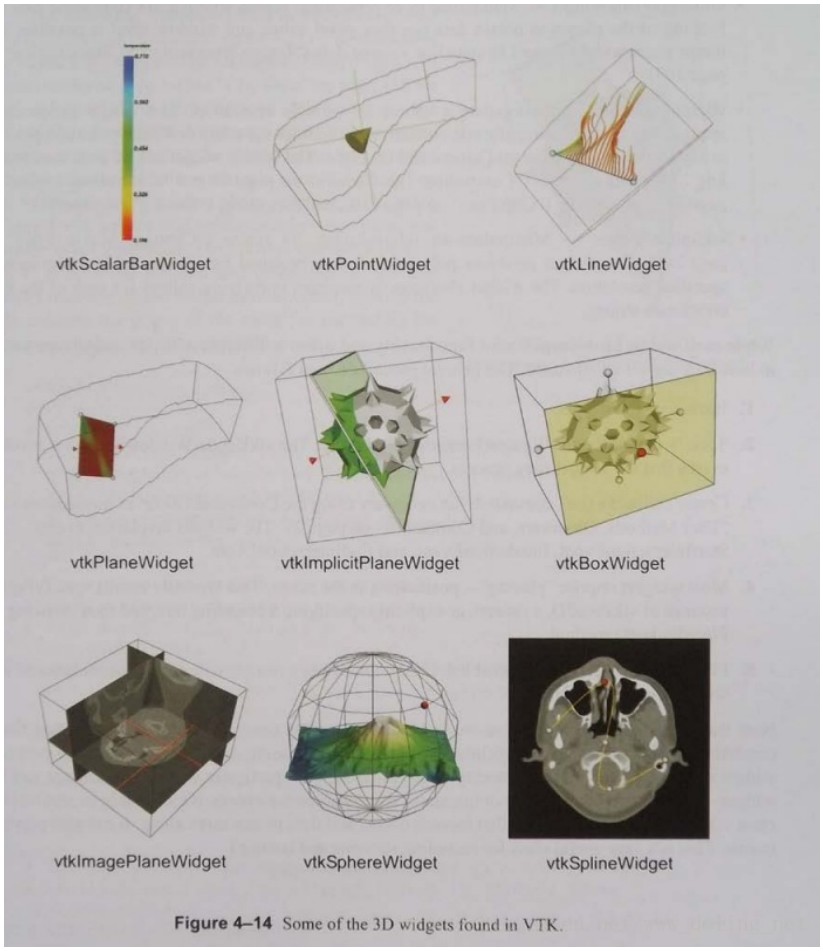

Figure 4–14 Some of the 3D widgets found in VTK.

图4-14VTK中可见的一些3D Widget
下面列出一些VTK中可见的比较重要的一些Widget，先简要描述一下这些Widget的特性与功能。需要注意的是，下面所提到的某些概念在本章中尚未详细讲解，读者可以查看第255页里的“交互、Widgets和选择”一章，了解相关的概念以及VTK中所实现的其他的Widget。

vtkScalarBarWidget ——管理一个vtkScalarBar，包括它的位置、缩放因子及方向等信息。关于vtkScalarBar可以参考第66页的“标题条”一节。
vtkPointWidget ——在3D空间中放置一个点。该Widget的输出是Polygonal类型的（vtkPolyData）。vtkPointWidget通常用于探测（见第100页“探测”一节）。
vtkLineWidget ——放置一条直线。该Widget的输出是Polygonal类型。主要应用于探测（见第100页“探测”一节）和绘图（见第66页“X-Y平面绘图”），或者是用于生成流线（见第95页“流线”一节）或流面（见第97页“流面”一节）。
vtkPlaneWidget ——放置一个有限范围的平面。平面的分辨率可以通过变量设置，该Widget生成一个隐函数和Polygon数据作为输出。主要用于探测（见第100页“探测”一节）和流线（seeding streamlines）（见第95页“流线”一节）。
vtkImplicitPlaneWidget—— 旋转一个无边缘的平面。该Widget生成一个隐函数和Polygon数据作为输出，其中Polygon数据是由平面与包围盒相切而生成的。主要用于探测（见第100页“探测”一节），切割（见第98页“切割”一节）以及数据的裁剪（见第110页“数据裁剪”一节）。
vtkBoxWidget ——放置一个包围盒。该Widget生成一个隐函数和变换矩阵。主要用于vtkProp3D及其子类的变换（见第70页“数据变换”一节），或者用于数据的切割与裁剪（见第98页“切割”和第110页“数据裁剪”一节）。
vtkImagePlaneWidget ——在一个三维体数据内操作三个正交的平面。平面的探测主要是为了得到数据位置、像素值以及窗宽窗位等信息。该Widget主要用于体数据的可视化（见第103页“图像处理及可视化”一章）。
vtkSphereWidget ——操作一个分辨率可设置的球体。该Widget生成一个隐函数和变换矩阵，另外还可以控制类似vtkCamera和vtkLight的焦点和位置等。主要用于控制光照和相机（见第49页“控制相机”和第51页“控制光照”等内容）以及数据的裁剪与切割等。
vtkSplineWidget ——操作一个插值的3D样条（见第248页“创建电影文件”一节）。该Widget生成用一系列分辨率可指定的分割线表达的Polygon数据作为输出。

虽然每个Widget都提供了不同的功能和API，但是这些3DWidget在创建和使用时都是相似的，其步骤通常为：

1. 初始化Widget；

2. 指定所要监听的vtkRenderWindowInteractor的实例，vtkRenderWindowInteractor会调用Widget要处理的事件。

3. 如果有必要的话，使用命令/观察者模式创建回调函数（见第29页“用户事件、观察者/命令模式”一节），Widget会调用StartInteractionEvent，InteractionEvent和EndInteractionEvent等事件。

4. 大多数Widget还需要放置在渲染场景中，通常需要指定一个vtkProp3D实例，一个数据集或者显式地指定一个包围盒，接着再调用方法PlaceWidget()。

5. 最后，必须激活Widget。缺省情况下，用户按键盘“i”键时，即可以激活Widget并让该Widget在场景中显示出来。

要注意的是，在某个时刻里有可能同时激活多个Widget，这些Widget可以与vtkInteractionStyle实例共同存在而各自的功能不受影响。如果场景中的鼠标事件不是作用在某个Widget上时，就会被vtkInteractorStyle所处理；如果作用于某个Widget时，当然只被该Widget所处理，其他的Widget或者vtkInteractorStyle对这个鼠标事件就是不可见的。（这里的一个例外是类vtkInteractorEventRecorder，该类可以记录用户事件，也可回放事件，这对于录制场景和测试等应用尤其有用）。

以下示例（摘自VTK/Examples/GUI/Tcl/ImplicitPlaneWidget.tct）演示了3DWidget的使用方法。类vtkImplicitPlaneWidget主要用于物体的切割，在该示例中，一个由vtkSphereSource和vtkConeSource所生成的类似狼牙棒（使用符号化的技术生成，见第94页“符号化”一节）的vtkProp3D实例是vtkImplicitPlaneWidget的切割对象，切割平面将其分成两部分，其中一部分用绿色进行着色。可以通过鼠标来控制vtkImplicitPlaneWidget的切割平面的位置和方向，具体就是调整平面的法向量，移动平面的原点等。

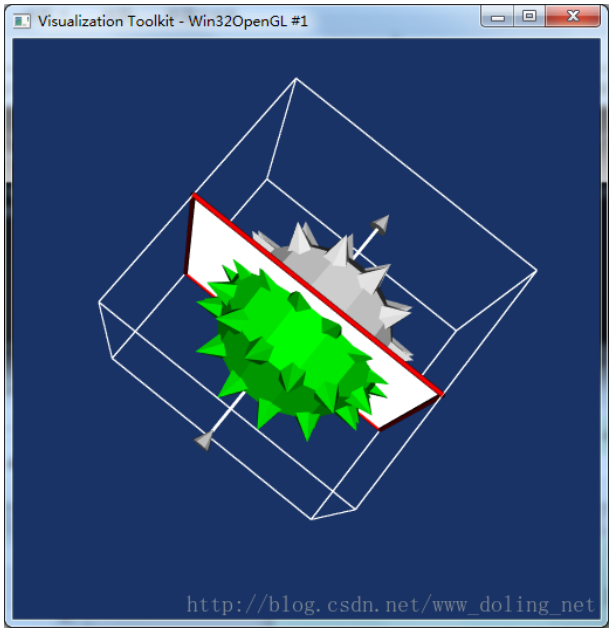

图4-15vtkImplicitPlaneWidget示例

```
vtkSphereSource sphere
vtkConeSource cone
vtkGlyph3D glyph
    glyph SetInputConnection [sphere GetOutputPort]
    glyph SetSourceConnection [cone GetOutputPort]
    glyph SetVectorModeToUseNormal
    glyph SetScaleModeToScaleByVector
    glyph SetScaleFactor 0.25

#The sphere and spikes are appended into a single polydata.
#This just makes things simpler to manage.
vtkAppendPolyData apd
    apd AddInputConnection [glyph GetOutputPort]
    apd AddInputConnection [sphere GetOutputPort]

vtkPolyDataMapper maceMapper
maceMapper SetInputConnection [apd GetOutputPort]

vtkLODActor maceActor
    maceActor SetMapper maceMapper
    maceActor VisibilityOn

#This portion of the code clips the mace with the vtkPlanes
#implicit function. The clipped region is colored green.
vtkPlane plane
vtkClipPolyData clipper
    clipper SetInputConnection [apd GetOutputPort]
    clipper SetClipFunction plane
    clipper InsideOutOn

vtkPolyDataMapper selectMapper
    selectMapper SetInputConnection [clipper GetOutputPort]

vtkLODActor selectActor
    selectActor SetMapper selectMapper
    [selectActor GetProperty] SetColor 0 1 0
    selectActor VisibilityOff
    selectActor SetScale 1.01 1.01 1.01

#Create the RenderWindow, Renderer and both Actors
#
vtkRendererren1
vtkRenderWindow renWin
    renWin AddRenderer ren1
vtkRenderWindowInteractor iren
    iren SetRenderWindow renWin

#Associate the line widget with the interactor
vtkImplicitPlaneWidget planeWidget
  planeWidget SetInteractor iren
  planeWidget SetPlaceFactor 1.25
  planeWidget SetInput [glyph GetOutput]
  planeWidget PlaceWidget
  planeWidget AddObserver InteractionEvent myCallback

ren1 AddActor maceActor
ren1 AddActor selectActor

#Add the actors to the renderer, set the background and size
#
ren1 SetBackground 1 1 1
renWin SetSize 300 300
ren1 SetBackground 0.1 0.2 0.4

#render the image
#
iren AddObserver UserEvent {wm deiconify .vtkInteract}
renWinRender

#Prevent the tk window from showing up then start the event loop.
wm withdraw .

proc myCallback {} {
    planeWidget GetPlane plane
    selectActor VisibilityOn
}
```

从上面的示例可以看到，首先是实例化一个vtkImplicitPlaneWidget对象，然后再放置该Widget。Widget的放置是相对于数据集而言的（Tcl脚本中“[glyph GetOutput]”返回的是vtkPolyData类型，该类型是vtkDataSet的子类）。PlaceFactor参数是调整Widget的相对大小，在这个示例中，Widget大小设置成比输入数据的包围盒大25%。示例中Widget的一个关键设置是添加观察者（Observer）来响应InteractionEvent事件。StartInteractionEvent和EndInteractionEvent分别是鼠标按下和鼠标弹起时所响应的事件，而InteractionEvent则是鼠标移动时响应的事件。在示例中，InteractionEvent是与Tcl的过程（Procedure）myCallback绑定在一起，该过程实现的功能主要是拷贝Widget所切割的平面到一个vtkPlane实例中，切割的过程主要是由一个隐函数完成的（见第213页“隐函数建模”一节）。

3DWidget是VTK里一个功能非常强大的特性，它可以快速用于任何应用程序中以完成复杂的交互。我们建议你看一下VTK源码中自带的与之相关的示例程序来学习这方面的知识，相关的例子在Examples/GUI和Hybrid/Testing/Cxx等目录中。

4.14 抗锯齿

VTK中有两种方法来开启抗锯齿的功能：基于图元或者是多重采样（perprimitive type or through multisampling），其中多重采样一般能获得更加理想的效果。

抗锯齿的方法都是通过控制vtkRenderWindow的API来完成的。当激活多重采样以及所用的图形卡支持该功能时，基于图元的抗锯齿功能的标志则会被忽略。抗锯齿的过程是在vtkRenderWindow对象创建之后，渲染场景初始化之前完成的。

要注意的是，VTK抗锯齿的结果与实际的OpenGL实现是有所区别的，OpenGL的实现可以是基于软件的，如Mesa，或者是由图形及其驱动来完成。

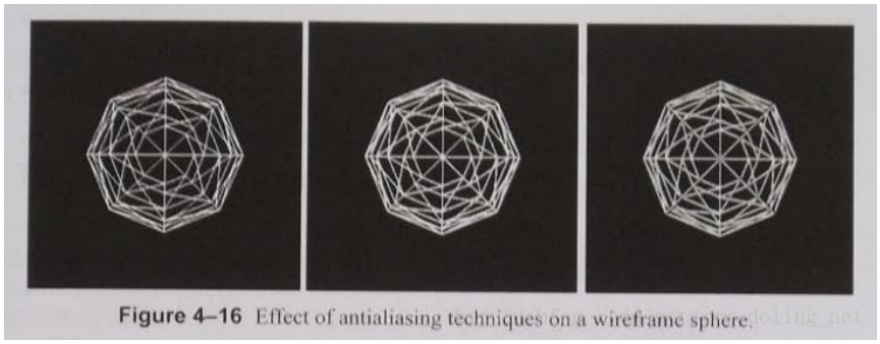

Figure 4–16 Effect of antialiasing techniques on a wireframe sphere.

图4-16针对球体线框模型的抗锯齿技术的效果

基于图元的抗锯齿

有三个标志来控制抗锯齿功能，一个标志分别控制一种图元类型：

PointSmoothing
LineSmoothing
PolygonSmoothing

初始状态下，这三个标志都是禁用的。下面列出激活点图元（Point Primitive）的抗锯齿功能所需的4个步骤：

1. vtkRenderWindow*w = vtkRenderWindow;:New();

2. w->SetMultiSamples(0);

3. w->SetPointSmoothing(1);

4. w->Render();

下面给出一个完整的示例程序，示例中使用点图元的抗锯齿功能来显示球体面片的顶点。

```
#include “vtkRenderWindowInteractor.h”
#include “vtkRenderWindow.h”
#include “vtkRenderer.h”
#include “vtkSphereSource.h”
#include “vtkPolyDataMapper.h”
#include “vtkProperty.h”

int main()
{
	vtkRenderWindowInteractor *i =vtkRenderWindowInteractor::New();
	vtkRenderWindow *w =vtkRenderWindow::New();
	i->SetRenderWindow(w);

	w->SetMultiSamples(0); //nomultisampling
	w->SetPointSmoothing(1); //poingantialiasing

	vtkRenderer *r = vtkRenderer::New();
	w->AddRenderer(r);

	vtkSphereSource *s =vtkSphereSource::New();
	vtkPolyDataMapper *m =vtkPolyDataMapper::New();
	m->SetInputConnection(s->GetOutputPort());

	vtkActor *a = vtkActor::New();
	a->SetMapper(m);
	vtkProperty *p = vtkProperty::New();
	p->SetRepresentationToPoints(); //wewant to see points
	p->SetPointSize(2.0); //big enough tonotice antialiasing
	p->SetLighting(0); //don’t be disturbby shading
	r->AddActor(a);

	i->Start();

	s->Delete();
	m->Delete();
	a->Delete();
	r->Delete();
	w->Delete();
	i->Delete();
}
```

以下几行代码是针对点图元的抗锯齿而设置的：

```
w->SetPointSmoothing(1);
p->SetRepresentationToPoints();
p->SetPointSize(2.0);
```

你可以设置成以下的代码，将点图元抗锯齿换成线图元的抗锯齿：

```
w->SetLineSmoothing(1);
p->SetRepresentationToWireframe();
p->SetLineSize(2.0);
```

同样，可以换成如下代码，将其换成多边形图元的抗锯齿：

```
w->PolygonSmoothing(1);
p->SetRepresentationToSurface();
```

多重采样

相对于图元的抗锯齿，多重采样可以获得更好的效果，多重采样默认是开启的，但是这个功能只要当图形卡支持时才起作用。目前，VTK只支持X窗口的多重采样功能。要禁用多重采用，只要设置变量MultiSamples（初始值是8）的值为0即可：

1. vtkRenderWindow*w = vtkRenderWindow::New();

2. w->SetMultiSamples(0);//disable multisampling.

3. w->Render();

回到上一小节的例子中，如果使用X11，只要禁用线的多重采样，就可以看到点、线或多边形的多重采样的效果。

4.15 透明多边形几何

对于可视化系统而言，将几何对象渲染成透明物体是一种非常强大、有用的功能。它可以让我们看到内部的数据，让我们可以将注意力集中在感兴趣的区域上，其中感兴趣区域可以渲染成不透明的物体，而其他的外层的物体则渲染成透明或半透明。

渲染透明物体也不是很繁琐：屏幕上显示的某个像素的颜色是由穿过该像素的所有可见的几何图元贡献的，像素的颜色是所有可见的几何图元的颜色做混合操作的结果。混合操作通常与顺序相关的，因此，要正确渲染透明物体，必须要有正确的深度（Depth sorting）排序。然而，深度排序一般都比较费时。

VTK提供三种方法来渲染透明物体，每种方法都权衡了正确性（质量）与效率（深度排序）。

快速但不准确。忽略之前所提的深度排序问题，这样就没有额外的计算时间占用，但是所渲染的对象则是不准确的。然而，根据应用程序的上下文关系，所渲染的结果也可能足够好。

较慢但基本准确。这种方法由两个Filter组成。首先是使用vtkAppendPolyData将所有的多边形几何数据整合在一起。然后将vtkAppendPolyData的输出做为vtkDepthSortPolyData的输入。由于深度排序是基于每个几何图元的质心而不是每个像素完成的。因此，这种方法不是足够准确但也能给出一个相对较好的结果。参考VTK/Hybrid/Testing/Tcl/depthSort.tcl示例。

非常慢但准确。如果图形显卡支持（只有NVidia的显卡）的话，可以使用"深度剥离"（Depth Peeling）技术。这种方法可以基于每个像素进行排序，但计算速度也是最慢的，同时结果也是最好的。首次渲染时，应该先设置vtkRenderWindow的Alpha位数：

```
vtkRenderWindow*w = vtkRenderWindow::New();
w->SetAlphaBitPlanes(1);

//禁用多重采样功能：
w->SetMultiSamples(0);
```

然后在渲染器中，开启深度剥离：

```
vtkRenderer*r = vtkRenderer::New();
r->SetUseDepthPeeling(1);<span style="line-height: 1.846153846; font-family: Arial; background-color: rgb(255, 255, 255);"> </span>
```

设置深度剥离的参数（MaximumNumberOfPeels和OcclusionRatio）：

```
r->SetMaximumNumberOfPeels(100);
r->SetOcclusionRatio(0.1);<span style="line-height: 1.846153846; font-family: Arial; background-color: rgb(255, 255, 255);"> </span>
```

渲染场景：

```
w->Render();
```

最后，应该检查一下图形卡是否支持深度剥离的功能：

```
r->GetLastRenderingUsedDepthPeeling();<span style="line-height: 1.846153846; font-family: Arial; background-color: rgb(255, 255, 255);"> </span>
```

深度剥离参数。要灵活使用深度剥离参数，需要对深度剥离算法有所了解。深度剥离算法是从前到后逐层剥离透明的几何物体，直到没有可渲染的物体为止。当达到所设置的最大迭代次数（即SetMaximumNumberOfPeels()所设置的参数），或者上一次剥离所修改的像素数目比预先设置的窗口的百分比区域对应的像素数还少的话，终止迭代。窗口的百分比是由SetOcclusionRatio()设置，如果设置为0.0，则表示希望获得精确的结果，设置成0.2比设置成0.1渲染速度要快。

对OpenGL的要求。如果OpenGL能达到以下要求，图形显卡可以支持深度剥离：

GL_ARB_depth_texture或OpenGL>= 1.4
GL_ARB_shadow或OpenGL>= 1.4
GL_EXT_shadow_funcs或OpenGL>= 1.5
GL_ARB_vertex_shader或OpenGL>= 2.0
GL_ARB_fragment_shader或OpenGL>= 2.0
GL_ARB_shader_objects或OpenGL>= 2.0
GL_ARB_occlusion_query或OpenGL>= 1.5
GL_ARB_multitexture或OpenGL>= 1.3
GL_ARB_texture_rectangle
GL_SGIS_texture_edge_clamp,GL_EXT_texture_edge_clamp或OpenGL >= 1.2

实际上，深度剥离只能运行于NVidia GeForce 6系列或更高版本的显卡，或者是Mesa（比如7.4），ATI的显卡不支持。

例子。以下是使用深度剥离技术的例子（可以查看VTK/Rendering/Tesing/Cxx目录里有关深度剥离的例子）。

```
#include"vtkRenderWindowInteractor.h"
#include"vtkRenderWindow.h"
#include"vtkRenderer.h"
#include"vtkActor.h"

#include"vtkImageSinusoidSource.h"
#include"vtkImageData.h"
#include"vtkImageDataGeometryFilter.h"
#include"vtkDataSetSurfaceFilter.h"
#include"vtkPolyDataMapper.h"
#include"vtkLookupTable.h"
#include"vtkCamera.h"

int main()
{
  vtkRenderWindowInteractor *iren=vtkRenderWindowInteractor::New();
  vtkRenderWindow *renWin =vtkRenderWindow::New();
  renWin->SetMultiSamples(0);

  renWin->SetAlphaBitPlanes(1);
  iren->SetRenderWindow(renWin);
  renWin->Delete();

  vtkRenderer *renderer = vtkRenderer::New();
  renWin->AddRenderer(renderer);
  renderer->Delete();
  renderer->SetUseDepthPeeling(1);
  renderer->SetMaximumNumberOfPeels(200);
  renderer->SetOcclusionRatio(0.1);

  vtkImageSinusoidSource*imageSource=vtkImageSinusoidSource::New();
  imageSource->SetWholeExtent(0,9,0,9,0,9);
  imageSource->SetPeriod(5);
  imageSource->Update();

  vtkImageData*image=imageSource->GetOutput();
  double range[2];
  image->GetScalarRange(range);

  vtkDataSetSurfaceFilter*surface=vtkDataSetSurfaceFilter::New();

 surface->SetInputConnection(imageSource->GetOutputPort());
  imageSource->Delete();

  vtkPolyDataMapper *mapper=vtkPolyDataMapper::New();
  mapper->SetInputConnection(surface->GetOutputPort());
  surface->Delete();

  vtkLookupTable *lut=vtkLookupTable::New();
  lut->SetTableRange(range);
  lut->SetAlphaRange(0.5,0.5);
  lut->SetHueRange(0.2,0.7);
  lut->SetNumberOfTableValues(256);
  lut->Build();

  mapper->SetScalarVisibility(1);
  mapper->SetLookupTable(lut);
  lut->Delete();

  vtkActor *actor=vtkActor::New();
  renderer->AddActor(actor);
  actor->Delete();
  actor->SetMapper(mapper);
  mapper->Delete();

  renderer->SetBackground(0.1,0.3,0.0);
  renWin->SetSize(400,400);

  renWin->Render();
 if(renderer->GetLastRenderingUsedDepthPeeling())
    {
    cout<<"depth peeling wasused"<<endl;
    }
  else
    {
    cout<<"depth peeling was notused (alpha blending instead)"<<endl;
    }
  vtkCamera*camera=renderer->GetActiveCamera();
  camera->Azimuth(-40.0);
  camera->Elevation(20.0);
  renWin->Render();

  iren->Start();
}
```

Painter机制：定制多边形数据的Mapper（Painter mechanism: customizing thepolydata mapper）。在渲染多边形数据时，我们有时需要有效地控制每个渲染步骤。VTK里可以使用Painter机制来实现，这要多亏工厂设计模式。以下的代码实际上创建一个vtkPainterPolyDataMapper对象。

```
vtkPolyDataMapper* m = vtkPolyDataMapper::New();
```

我们可以将m向下转型成vtkPainterPolyDataMapper，从而可以访问vtkPainterPolyDataMapper的API：

```
vtkPainterPolyDataMapper*m2= vtkPainterPolyDataMapper::SafeDownCast(m);
```


```
vtkPainterPolyDataMapper*m2= vtkPainterPolyDataMapper::SafeDownCast(m);
```


这个多边形数据的Mapper将渲染过程委托（Delegate）给vtkPainter对象。通过SetPainter()和GetPainter()可以访问这种委托。

vtkPainter只为具体的子类Painter提供抽象的API接口，每个具体的子类负责对各个阶段进行渲染。这种机制允许用户选择或者组合不同的渲染阶段。比如，vtkPolygonsPainter负责多边形数据的绘制；而vtkLightingPainter则负责光照参数的设置。不同Painter的组合可以形成一条Painter链，这条链中的每个Painer可以将渲染的某个执行部分委托给另外一个Painter。

大多数情况下，我们没有必要显式地去设置Painter链，因为vtkDefaultPainter已经为我们设置了一个标准的Painter链了。

编写自定义Painter。要编写自定义的Painter需要编写两个必要的类：一个是抽象基类vtkPainter的子类；另一个是用OpenGL实现具体的类。

我们先看看现在的Painter类：vtkLightingPainter。vtkLightingPainter派生自vtkPainter，这个类基本是空的。真正的实现是在vtkLightingPainter的子类vtkOpenGLLightingPainter中，在这个类中重载了保护成员方法RenderInternal()。

RenderInternal()方法的参数主要是Renderer和Actor。实现RenderInternal()方法时，主要是编写实际的渲染阶段的代码以及调用Painter链中的下一个Painter，即：this->Superclass::RenderInternal()。

第04章-VTK基础（7）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本小节内容对应原书的第83页至第87页】

4.16 动画

动画是可视化等系统的重要模块；
通过编写修改某个Filter和渲染器Render参数的循环，可以实现简单的动画效果。但是一旦有多个参数需要修改时，这种方法就会变得比较复杂。
VTK提供了一个由vtkAnimationCue和vtkAnimationScene等类所组成的框架，支持动画场景的创建和回放。
vtkAnimationCue对应一个可随时间改变的实体，比如Actor的空间位置；vtkAnimationScene则表示一个场景或者由vtkAnimationCue实例所组成的动画场景。

动画场景（vtkAnimationScene）

vtkAnimationScene表示一个场景或者是动画场景的建立。动画场景一般是通过渲染一系列的帧，在渲染每一帧时改变某些可视化参数来建立的。所渲染的每一帧都关联一个动画时间，这个时间用来确定动画中每一帧的位置。基于不同的播放模式，动画时间在动画播放过程中是一个计数不断增加的简单变量。
以下列出了类vtkAnimationScene中一些比较重要的方法：

SetStartTime()/SetEndTime()

设置动画场景的开始和结束时间，也是动画回放时的时间范围。

SetPlayMode()

用于控制动画回放的模式，也就是动画时间是如何改变的。有两种模式可以设置。

SequenceMode (PLAYMODE_SEQUENCE)

这种模式下，每帧的动画时间增加(1/frame-rate)的间隔，直至达到所设置的EndTime。因此，所渲染的总帧数是固定的，与渲染每一帧时所用的时间的长短无关。

RealTimeMode (PLAYMODE_REALTIME)

这种模式下，整个动画的运行时间大约是(EndTime-StartTime)秒，其中第n帧的运行时间是：第(n-1)帧的运行时间+渲染第(n-1)帧所用的时间。因此，所渲染的总帧数随着渲染每一帧所用时间的不同而不同。

SetFrameRate()

帧率是指单位时间内渲染的帧数。主要用于Sequence模式。

AddCue(),RemoveCue(), RemoveAllCue()

添加vtkAnimationCue实例到场景中或者从场景中移除vtkAnimationCue实例。

SetAnimationTime()

指定某一帧的动画时间。

GetAnimationTime()

动画回放时，获取动画的时钟时间。

Play()

开始播放动画。

SetLoop()

如果设置为true，则调用Play()方法后将进入动画循环。

AnimationCue (vtkAnimationCue)

vtkAnimationCue对应动画场景中随时间改变的实体。vtkAnimationCue实例本身并不知道与动画相关的参数是如果改变的。因此，用户必须从vtkAnimationCue中派生出子类或者使用观察者模式监听事件，来改变动画过程中需要改变的参数。

动画场景中的Cue实体都有一个开始时间（start-time）和结束时间（end-time）。动画回放时，当场景的动画时间是处于所指定的开始时间与结束时间的范围内时，Cue实体就会被激活。Cue实体一旦激活后，它本身会发出vtkCommand::StartAnimationCueEvent事件。而对于动画系列中的每一帧，则发出vtkCommand::AnimationCueTickEvent事件，当动画时间递增至Cue实体的结束时间时，会发出vtkCommand::EndAnimationCueEvent事件。以下是vtkAnimationCue类中一些比较重要的方法：

SetTimeMode

TimeMode定义了Cue实体的起始时间和结束时间是如何指定的，有两种模式可选。

Relative (TIMEMODE_RELATIVE)

这种模式下，动画场景中的Cue实体的时间是相对动画场景的开始时间来指定的。

Normalized (TIMEMODE_NORMALIZED)

这种模式下，Cue实体的开始时间和结束时间的取值范围为[0,1]，其中0对应动画场景的开始，1对应结束。

SetStartTime/SetEndTime

这两个方法主要是当Cue实体被激活时，标识动画时间的范围。当TimeMode取值为TIMEMODE_RELATIVE时，与动画场景的起始和结束时间有相同的单位。当TimeMode取值为TIMEMODE_NORMALIZED时，Cue实体的开始时间和结束时间的取值范围为[0,1]，其中0对应动画场景的开始，1对应结束。

GetAnimationTime()

用于vtkCommand::AnimationCueTickEvent事件的处理。处理事件时，可用来确定动画场景中的当前帧。其值依赖于TimeMode，如果TimeMode取值为TIMEMODE_RELATIVE时，其值是Cue实体激活后的时间单位的倍数；TimeMode取值为TIMEMODE_NORMALIZED时，Cue实体的开始时间和结束时间的取值范围为[0,1]，其中0对应动画场景的开始，1对应结束。

GetClockTime()

该方法与vtkAnimationScene::GetAnimationTime()中的动画时钟时间类似。只有当处理vtkCommand::AnimationCueTickEvent事件时才有效。

TickInternal (double currenttime, doubledeltatime, double clocktime)

正如前面所提的，我们可以派生一个vtkAnimationCue子类来代替编写处理动画事件的函数。子类派生时，需要重载该函数。其中该函数的参数分别对应GetAnimationTime()，GetDeltaTime()和GetClockTime()的返回值。

StartCueInternal(), EndCueInternal()

这两个方法可在子类中重载，主要做一些创建、清除等工作以及动画回放时控制Cue实体的开始和结束。用户也可以添加事件观察者，通过监听vtkCommand::StartAnimationCueEvent和vtkCommand::EndAnimationCueEvent事件来实现类似的功能。

以下的示例中，我们创建了一个简单的动画：vtkSphereSource的StartTheta随着动画时间而改变。示例中的Cue实体使用Normalized时间模式，因此，可以通过改变场景时间或者Cue实体的时间来改变StartTheta的值。

```
class vtkCustomAnimationCue : public vtkAnimationCue
{
public:
	static vtkCustomAnimationCue* New();
	vtkTypeRevisionMacro (vtkCustomAnimationCue,vtkAnimationCue);

	vtkRenderWindow *RenWin;
	vtkSphereSource *Sphere;

protected:
	vtkCustomAnimationCue()
	{
		this->RenWin = 0;
		this->Sphere = 0;
	}

	// Overridden to adjust the sphere'sradius depending on the frame we
	// are rendering. In this animation wewant to change the StartTheta
	// of the sphere from 0 to 180 over thelength of the cue.
	virtual void TickInternal(doublecurrenttime, double deltatime, double clocktime)
	{
		double new_st = currenttime * 180;
		// since the cue is in normalizedmode, the currenttime will be in the
		// range[0,1], where 0 is start ofthe cue and 1 is end of the cue.
		this->Sphere->SetStartTheta(new_st);
		this->RenWin->Render();
	}
};
vtkStandardNewMacro(vtkCustomAnimationCue);
vtkCxxRevisionMacro(vtkCustomAnimationCue,"$Revision$");

int main(int argc, char *argv[])
{
	// Create the graphics sturctures. Therenderer renders into the
	// render window.
	vtkRenderer *ren1 = vtkRenderer::New();
	vtkRenderWindow *renWin =vtkRenderWindow::New();
	renWin->SetMultiSamples(0);
	renWin->AddRenderer(ren1);

	vtkSphereSource *sphere =vtkSphereSource::New();
	vtkPolyDataMapper *mapper =vtkPolyDataMapper::New();
	mapper->SetInputConnection(sphere->GetOutputPort());
	vtkActor *actor = vtkActor::New();
	actor->SetMapper(mapper);
	ren1->AddActor(actor);

	ren1->ResetCamera();
	renWin->Render();

	//Create an Animation Scene
	vtkAnimationScene *scene =vtkAnimationScene::New();
	scene->SetModeToSequence();
	scene->SetFrameRate(30);
	scene->SetStartTime(0);
	scene->SetEndTime(60);

	// Create an Animation Cue to animate thecamera.
	vtkCustomAnimationCue *cue1 =vtkCustomAnimationCue::New();
	cue1->Sphere = sphere;
	cue1->RenWin = renWin;
	cue1->SetTimeModeToNormalized();
	cue1->SetStartTime(0);
	cue1->SetEndTime(1.0);
	scene->AddCue(cue1);
	scene->Play();
	scene->Stop();

	ren1->Delete();
	renWin->Delete();
	scene->Delete();
	cue1->Delete();
	return 0;
}
```

【第4章-VTK基础 全部翻译完成】

第05章-可视化技术（1）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本小节内容对应原书的第89页至第95页】

前面的章节中我们已经介绍了一些数据渲染和交互的基本工具。本章中我们主要介绍一下可视化技术。这些技术（实现为Filter）根据其操作数据类型来组织。其中部分Filter是能够处理任何数据类型的一般性Filter，他们能够接收vtkDataSet（或者任意的子类）。而更多的Filter则是依赖于其处理的数据类型（如vtkPolyData）的专门性Filter。其中处理vtkImageData（或者其子类vtkStructuredData）数据类型的一类Filter本章中暂不涉及，将在下一章中具体讨论（第103页"图像处理和可视化"）。

阅读本章时，有两点需要铭记在心。一是Filter会产生多种输出数据类型，而这些类型并不一定与输入数据类型一致。第二，Filter会通过组合来创建复杂的数据处理管线。从本章中的例子中可以看到一些常见的Filter组合。

5.1 vtkDataSet（及其子类）的可视化

本节主要讲解vtkDataSet数据对象的一些常用可视化操作。注意vtkDataSet是VTK中所有数据类型的超类类型（见图3-2）。因此，这里讲到的操作都适用于所有的数据类型。换句话说，所有接收vtkDataSet类型的Filter都可以接收vtkPolyData，vtkImageData，vtkStructured，vtkRectilinearGrid和vtkUnstructuredGrid类型。

数据属性操作（WorkingWith Data Attributes）

数据属性是与数据结构相关的信息（如25页"可视化管线"所述）。VTK中，属性数据分为点属性数据和单元属性数据。属性数据与结构数据一起被许多的Filter处理产生新的结构和属性。关于属性数据的内容不是本章的重点，这里给出了一个简单的例子来讲述基本的思想。（更多信息请参阅362页"场和属性数据"和图16-1。）

数据属性是简单的vtkDataArray类型，它可以是标量scalar，向量vector，张量tensor，法向normal，纹理坐标texture coordinate，全局id （global id，如标识大量元素），或者pedigreeids（用来追踪管线中的元素历史）。vtkDataSet中的一个点或者单元都有独立的属性数据。数据属性可以关联vtkDataSet中的点或者单元。与vtkDataSet关联的vtkDataArray都是vtkDataArray的一个具体子类，例如vtkFloatArray或者vtkIntArray。这些数据数组可以看做连续的、线性的内存块。在内存块中，数据数组可以看作由子数组或者"元组"（Tuple）组成。创建属性数据即是根据类型实例化数组内存，制定元组大小，插入数据并与数据集关联，如下面Tcl脚本所示。属性数据类型可以指定为标量，向量，张量，纹理坐标或者法向。例如：

```
vtkFloatArray scalars
    scalarsInsertTuple1 0 1.0
    scalarsInsertTuple1 1 1.2
    ...etc...

vtkDoubleArray vectors
    vectorsSetNumberOfComponents 3
    vectorsInsertTuple3 0 0.0 0.0 1.0
    vectorsInsertTuple3 1 1.2 0.3 1.1
    ...ect...

vtkIntArray justAnArray
   justAnArray SetNumberOfComponents 3
   justAnArray SetNumberOfTuples $numberOfPoints
   justAnArray SetName "Solution Attributes"
   justAnArray SetTuple2 0 1 2
   justAnArray SetTuple2 1 3 4
    ...etc...

vtkPolyData polyData; #A concrete type of vtkDataSet
    [polyDataGetPointData] SetScalars scalars
    [polyDataGetCellData] SetVectors vectors
    [polyDataGetPointData] AddArray justAnArray
```

这里创建了三种类型的属性数据，float，double和int。第一个数组scalars实例化后，默认的元组大小为1。InsertTuple1()方法用来向数组中插入数据（所有Insert___()方法负责分配足够的内存来保存数据）。下一个数组vectors的元组数为3，因此vectors定义为含有三个分量，InsertTuple3用来向数组中添加数据。最后创建的是元组数为2的数组，通过SetNumberOfTuples()分配内存。接着通过SetTuple2()添加数据；该方法使用的前提是内存已经分配，因此速度要明显快于Insert__()方法。当将属性数据关联到点数据或者单元数据时，注意区别设置类型的方法（SetScalars()和SetVectors()）。注意点属性个数必须与数据结构中的点个数一致，单元属性与数据结构的单元个数一致。

类似的，采用如下方法访问属性数据

```
set scalars [[polyData GetPointData] GetScalars]
set vectors [[polyData GetCellData] GetVectors]
```

许多的Filter需要专门的属性数据进行工作。例如，vtkElevationFilter依赖于相应的高度数据产生标量值。其他的Filter只依赖于结构数据，并忽略传来的属性数据。还有一些Filter

需要结构数据和属性数据来工作，如vtkMarchingCubes。它利用输入的标量属性数据和结构数据来产生轮廓结构。其他类型的属性数据，例如向量，在计算轮廓时进行差值计算并输出。

另一个与属性数据相关的重要问题是，有些Filter只输出一种类型的属性，忽略其他的类型。例如，当你想采用一个不能处理输入数据的属性数据类型的Filter来处理数据时，或者你想直接将其从一种类型转换至另一个类型。有2个Filter可以帮你实现：vtkPointDataToCellData和vtkCellDataToPointData。下面例子演示来怎样使用他们（摘自VTK/Examples/DataManipulation/Tcl/pointToCellData.tcl）。

```
vtkUnstructuredGridReader reader
    readerSetFileName "$VTK_DATA_ROOT/Data/blow.vtk"
    readerSetScalarsName "thickness9"
    readerSetVectorsName "displacement9"
vtkPointDataToCellData p2c
    p2cSetInputConnection [reader GetOutputPort]
    p2cPassPointDataOn
vtkWarpVector warp
    warpSetInputConnection [p2c GetOutputPort]
vtkThreshold thresh
    threshSetInputConnection [warp GetOutputPort]
    threshThresholdBetween 0.25 0.75
    threshSetAttributeModeToUseCellData<span style="font-size: 14px; line-height: 1.846153846; font-family: Arial; background-color: rgb(255, 255, 255);"> </span>
```

该例子演示了怎样转换属性数据类型，以及一个可以任意处理单元数据或者点数据的Filter（vtkThreshold）。PassPointDataOn()方法设置vtkPointDataToCellData来创建单元数据，并将点数据输出。SetAttributeModeToUseCellData()设置vtkThreshold使用单元数据来进行处理。

点数据和单元数据之间的转换采用的是一个平均算法。将一个给定单元的所有点的数据求平均值即可将点的属性数据转换为单元数据；而单元数据转换为点数据则是通过计算所有用到该点的单元的数据的平均值。

颜色映射（ColorMapping）

最常用的可视化技术可能是通过标量值或者颜色映射来对物体着色。着色技术的思想比较简单，将标量值映射到一个颜色查找表来获取颜色，然后在渲染时使用颜色来改变点或者单元的外观。在阅读本节前，请先理解怎样控制Actor的颜色（详见54页“Actor颜色”一节）。

VTK中颜色映射主要由用户生成或者数据文件中的标量数据和vtkMapper实例执行颜色映射使用的颜色查询表来控制。也可以使用任意的数据数组通过ColorByArrayComponent()方法来控制。如果没有指明，Mapper会生成一个默认的颜色查询表，你也可以自己创建（下例摘自VTK/Examples/Rendering/Tcl/Rainbow.tcl，运行结果如图5-1所示）。

```
vtkLookupTable lut
lut SetNumberOfColors 64
lut SetHueRange 0.0 0.667
lut Build
for {set i 0} {$i<16} {incr i 1} {
    eval lutSetTableValue [expr $i*16] $red 1
    eval lutSetTableValue [expr $i*16+1] $green 1
    eval lutSetTableValue [expr $i*16+2] $blue 1
    eval lutSetTableValue [expr $i*16+3] $black 1
}
vtkPolyDataMapper planeMapper
planeMapper SetLookupTable lut
planeMapper SetInputConnection [planeGetOutputPort]
planeMapper SetScalarRange 0.197813 0.710419
vtkActor planeActor
planeActor SetMapper planeMapper
```

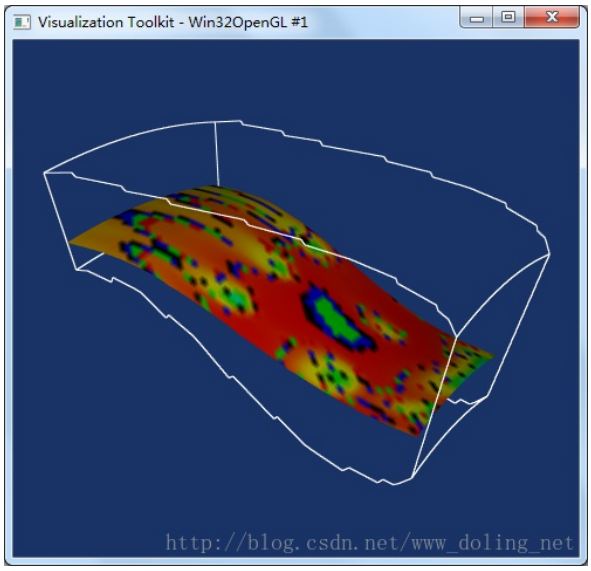

图5-1 颜色映射

如该例所示，操作查询表有两种方式。一是指定一个HSVA范围然后在HSVA空间中插值计算颜色表中颜色（实际由build()函数计算颜色）。第二种方法是在根据颜色表的位置人为指定颜色。注意颜色表中的颜色数目可以设置。本例中利用HSVA范围生成颜色表，然后利用SetTableValue()函数替换掉相应的颜色。

Mapper的SetScalarRange()函数控制标量值与颜色表的映射方式。大于最大值的所有标量值都被映射为最大值，小于最小值的所有标量值都映射为最小值。标量范围设置，可以通过在某一数据范围内映射更多的颜色来达到“扩展”的目的。

很多情况下标量数据就是颜色值，这样就不需要通过颜色查找表进行映射。映射器Mapper提供了多种方法来控制颜色映射。

SetColorModeToDefault()设置使用默认Mapper映射方法。默认方法直接将三个unsignedchar类型标量数据作为颜色值而不需要进行映射；而其他的类型数据则是通过颜色查找表映射。
SetColorModeToMapScalars()通过颜色查找表来映射所有类型的标量数据。如果每个元组由多于1个分量组成，那么通过第0个分量来执行颜色映射。
vtkMapper另一个重要特性就是控制执行颜色渲染使用的数据，例如点或者单元数据，或者是数据数组。该功能可以由如下方法完成。需要注意的是，这些方法会产生非常不同的渲染结果：在渲染时点标量数据模式下，几何体表面所有的点数据通过插值后映射颜色，而单元标量数据模式下则将每个单元渲染为一个颜色。
SetScalarModeToDefault()触发默认Mapper行为。Mapper默认采用点标量数据渲染；如果点标量数据不可用，则采用可用的单元数据进行渲染。
SetScalarModeToUsePointData()利用点标量数据进行颜色映射完成渲染。如果没有点标量数据可用，那么标量数据将对物体颜色没有任何影响。
SetScalarModeToUseCellData()利用单元标量数据进行颜色映射渲染物体。如果没有单元标量数据可用，那么标量数据将对物体颜色没有任何影响。
SetScalarModeToUsePointFieldData()利用点属性数据中的数据数组，而不是点标量数据和单元标量数据。该函数需要结合ColorByArrayComponent()函数来指定数据数组和作为标量数据的分量。
SetScalarModeToUseCellFieldData()利用单元属性数据中得场数据，而不是点或者单元标量数据。该方法也需要结合ColorByArrayComponent()函数来指定数据数组和作为标量数据的分量。

正常情况下Mapper默认行为都会正常工作，除非点和单元标量数据都可用。这时，你需要显式指明用点或者面标量数据来对物体着色。

轮廓（Contouring）

另一个常见的可视化技术是轮廓生成。轮廓是指具有相同标量数据的线或者面。VTK中vtkContourFilter执行轮廓生成，如下面的Tcl例子所示。示例代码摘自VTK/Examples/VisualizationAlgorithms/Tcl/VisQuad.tcl，运行结果如图5-2所示。

```
#Create 5 surfaces in range specified
vtkContourFilter contours
   contours SetInputConnection [ Sample \
            GetOutputPort]
   Contours GenerateValues 5 0.0 1.2
vtkPolyDataMapper contMapper
   contMapper SetInputConnection [contours GetOutputPort]
   contMapper SetScalarRange 0.0 1.2
vtkActor contActor
   contActor SetMapper contMapper
```

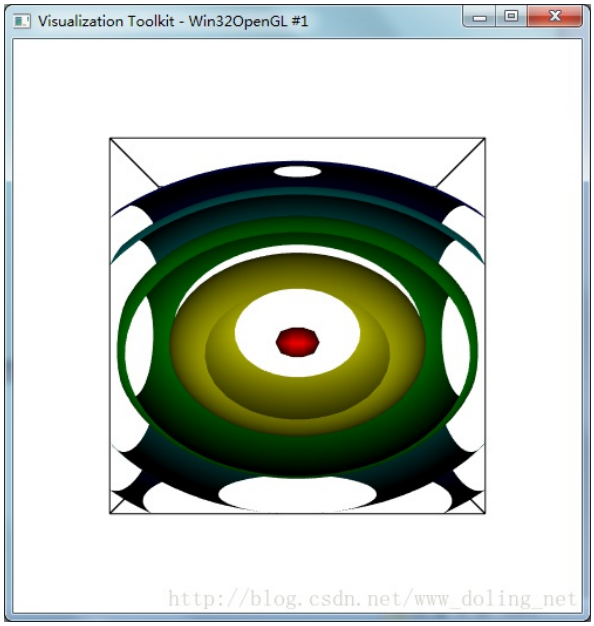

图5-2 生成轮廓

有两种方式来指定轮廓值。最简单的方式是通过SetValue()方法来指定轮廓数和对应的轮廓值（可以指定多个值）。

```
ContoursSetValue 0 0.5
```

示例代码中采用的是第二种方法，即GenerateValues()方法。通过该函数，你可以指定数据范围和该范围内要生成的轮廓个数（包含结束数据）。

注意在VTK中有多个对象来针对特定数据类型执行轮廓生成。如vtkSynchronizedTemplates2D和vtkSynchronizedTemplates3D。如果利用vtkContourFilter的话，不需要直接实例化这些类型的对象，该Filter会根据数据类型自动生成相应的轮廓生成函数。

符号化（Glyphing）

符号化是一种利用字符或字形来表示数据的可视化技术（图5-3）。这些符号可以简单，也可以复杂。简单的如利用有向锥体来表示向量数据，复杂的有多元字形，如Chernoff faces（由数据来控制表情的人脸字形表示）。VTK中利用vtkGlyph3D类来产生可以放缩、着色和具有方向的字形。输入数据的每个点都会拷贝一个字形。字形本身则是通过该Filter的第二个输入函数接收（接收vtkPolyData类型数据）。下面代码说明了vtkGlyph3D的使用。（代码摘自VTK/Examples/VisualizationAlgorithms/Tcl/spikeF.tcl）

```
vtkPolyDataReader fran
    franSetFileName "$VTK_DATA_ROOT/Data/fran_cut.vtk"
vtkPolyDataNormals normals
    normalsSetInputConnection [fran GetOutputPort]
    normalsFlipNormalsOn
vtkPolyDataMapper franMapper
   franMapper SetInputConnection [normals GetOutputPort]
vtkActor franActor
    franActor SetMapper franMapper
   eval[franActor GetProperty] SetColor 1.0 0.49 0.25
vtkMaskPoints ptMask
    ptMask SetInputConnection [normals GetOutputPort]
    ptMask SetOnRatio 10
    ptMask RandomModeOn
# In this case we are using a cone as a glyph. Wetransform the cone so
# its base is at 0,0,0. This is the point whereglyph rotation occurs.
vtkConeSource cone
    cone SetResolution 6
vtkTransform transform
    transformTranslate 0.5 0.0 0.0
vtkTransformPolyDataFilter transformF
   transformF SetInputConnection [cone GetOutputPort]
   transformF SetTransform transform
vtkGlyph3D glyph
    glyph SetInputConnection  [ptMask GetOutputPort]
    glyph SetSourceConnection [transformF GetOutputPort]
    glyph SetVectorModeToUseNormal
    glyph SetScaleModeToScaleByVector
    glyph SetScaleFactor 0.004
vtkPolyDataMapper spikeMapper
   spikeMapper SetInputConnection [glyph GetOutputPort]
vtkActor spikeActor
   spikeActor SetMapper spikeMapper
    eval[spikeActor GetProperty] SetColor 0.0 0.79 0.34
```

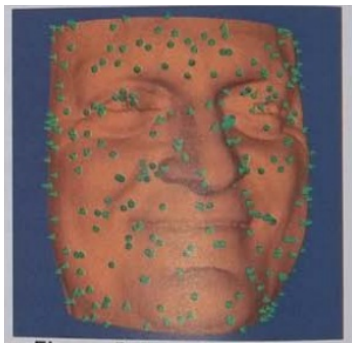

图5-3 在表面法向量位置上显示符号

这段代码演示了怎样用有向小锥体来表示曲面法向量。首先读入并显示一个数据集（由Cyberware激光数字化系统采集），然后vtkMastPoints用来对输入数据点进行降采样，采样结果（附带属性数据）作为vtkGlyph3D的输入。vtkConeSource用来生成字形实例。注意锥体采用vtkTransformPolyDataFilter平移来使原点位于(0, 0, 0)（因为vtkGlyph3D绕原点旋转）。

vtkGlyph3D对象glyph设置用点属性法向量作为方法向量。（SetVcetorModeToUseVector()设置使用向量数据来替代法向量）它利用向量模值来对锥体进行放缩。（当然也可以利用SetScaleModeToScaleByScalar()函数设置通过标量数据对字形进行放缩或者利用SetScaleModeToDataScalingOff()函数关闭放缩功能。）

当然也可以利用标量数据、向量数据或者放缩因子对字形进行着色。还可以创建一个字形表，采用标量或者向量数据来尽力索引。参考在线文档来获取更多信息。

第05章-可视化技术（2）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本小节内容对应原书的第95页至第105页】

流线（Streamlines）

流线可以看做无重量粒子在向量场（如速度场）中的移动路径。流线可以表达向量场的结构。通常可以创建多个流线来探索向量场中的感兴趣特征。如图5-4。流线可以通过数值积分来计算，因此只能近似的模拟真实的流线。

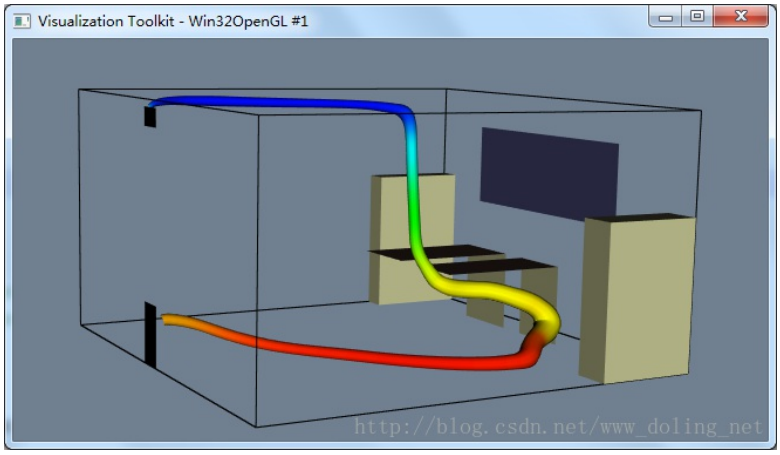

图5-4 被管道所包围的流线

创建流线需要指定起始点，方向（沿着或者反着流向），以及其他的控制前进的参数。下面代码说明了如何创建一条流线。该流线由一个管道表示，管道半径正比于速度模值的倒数。当流比较小时，管道会比较粗；反之亦然。Tcl代码摘自VTK/Exmaples/VisualizationAlgorithms/Tcl/OfficeTube.tcl。

```
vtkStructuredGridReader reader
    reader SetFileName"$VTK_DATA_ROOT/Data/office.binary.vtk"
    reader Update;#force a read to occur

vtkRungeKutta4 integ
vtkStreamTracer streamer
    streamer SetInputConnection [reader GetOutputPort]
    streamer SetStartPosition 0.1 2.1 0.5
    streamer SetMaximumPropagation 500
    streamerSetMaximumPropagationUnitToTimeUnit
    streamer SetInitialIntegrationStep 0.05
    streamer SetInitialIntegrationStepUnitToCellLengthUnit
    streamer SetIntegrationDirectionToBoth
streamer SetIntegrator integ

vtkTubeFilter streamTube
    streamTube SetInputConnection [streamer GetOutputPort]
    streamTube SetInputArrayToProcess 1 0 0vtkDataObject::FIELD_ASSOCIATION_POINTS vectors
    streamTube SetRadius 0.02
    streamTube SetNumberOfSides 12
    streamTube SetVaryRadiusToVaryRadiusByVector
vtkPolyDataMapper mapStreamTube
    mapStreamTube SetInputConnection[streamTube GetOutputPort]
    eval mapStreamTube SetScalarRange \
      [[[[reader GetOutput] GetPointData]GetScalars] GetRange]
vtkActor streamTubeActor
    streamTubeActor SetMapper mapStreamTube
[streamTubeActor GetProperty] BackfaceCullingOn
```

在该例中我们设置起始点为(0.1，2.1, 0.5)。也可以通过指定单元id，单元子id和参数坐标来指定起始坐标。MaximumPropagation变量用来控制流线的最大长度（度量单位由MaximumPropagationUnit变量指定）。如果需要更高的精度（代价是更多的计算时间），设置InitialIntegrationsetp变量为一个更小的值。（这里该参数指定为单元长度，也可以选择时间或者距离）另外，精度也可以通过选择一个vtkInitialValueProblemSolver的子类如vtkRungeKutta2或者vtkRungeKutta45（可以设置自适应的步长）来控制。默认情况下，流线追踪类利用vtkRungeKutta2来执行数值积分运算。积分方向可以由以下三个函数控制。

SetIntegrationDirectionToForward()
SetIntegrationDirectionToBackward()
SetIntergrationDirectionToBoth()

由于线通常难以观察，因为我们采用一个管道Filter来表示。管道Filter设置为半径正比于速度模值的倒数，通过函数SetVaryRadiusToVaryRadiusByVector()开启该功能。也可以通过SetVaryRadiusToVaryRadiusByScalar()设置根据标量控制半径，或者取消半径可变（SetVaryRadiusToVaryRadiusOff()）。注意需要通知管道filter利用哪个数组来控制半径，这里“vectors”数组通过SetInputArrayToProcess()函数通知管道filter。

有时候我们需要产生同步地产生许多流线。其中一个方法是利用SetSourceConnection()方法来指定一个vtkDataSet的实例，利用这个实例的点来追踪流线。下面是该应用的一个例子，代码摘自VTK/Examples/VisualizationAlgorithms/Tcl/OffcicesTubes.tcl。

```
vtkPointSource seeds
    seeds SetRadius 0.15
    eval seeds SetCenter 0.1 2.1 0.5
    seeds SetNumberOfPoints 6
vtkRungeKutta4 integ
vtkStreamTracer streamer
    streamer SetInputConnection  [reader GetOutputPort]
    streamer SetSourceConnection [seeds GetOutputPort]
    streamer SetMaximumPropagation 500
    streamer SetMaximumPropagationUnitToTimeUnit
    streamer SetInitialIntegrationStep 0.05
    streamer SetInitialIntegrationStepUnitToCellLengthUnit
    streamer SetIntegrationDirectionToBoth
    streamer SetIntegrator integ
```

注意该例中采用vtkPointSource对象来创建球状点云作为streamer的输入。对于输入的每一个点都会计算一条流线。

流面（StreamSurfaces）

高级用户可能想利用VTK的流面计算功能。流面计算分为两步，首先用一个有序点集生成一个流线序列，然后再由vtkRuledSurfaceFilter来创建流面。vtkRuledSurfaceFilter假设每条流线是有序排列的，而且每条流线与左右相邻流线在指定距离内，因此输入点集保持有序十分重要，否则计算结果将会非常糟糕。下面代码演示了如何创建一个流面（代码取自VTK/Examples/VisualizationAlgorithms/Tch/streamSurface.tcl，结果如图5-5）。

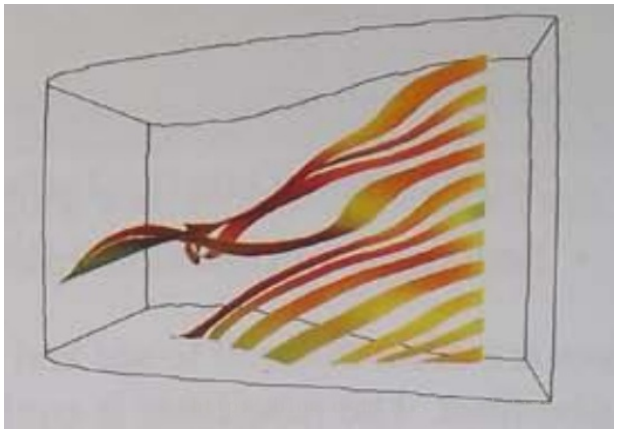

图5-5 流面

```
vtkLineSource rake
  rake SetPoint1 15 -5 32
  rake SetPoint2 15 5 32
  rake SetResolution 21
vtkPolyDataMapper rakeMapper
  rakeMapper SetInputConnection [rake GetOutputPort]
vtkActor rakeActor
  rakeActor SetMapper rakeMapper

vtkRungeKutta4 integ
vtkStreamTracer sl
  sl SetInputConnection [pl3d GetOutputPort]
  sl SetSourceConnection [rake GetOutputPort]
  sl SetIntegrator integ
  sl SetMaximumPropagation 0.1
  sl SetMaximumPropagationUnitToTimeUnit
  sl SetInitialIntegrationStep 0.1
  sl SetInitialIntegrationStepUnitToCellLengthUnit
  sl SetIntegrationDirectionToBackward

vtkRuledSurfaceFilter scalarSurface
  scalarSurface SetInputConnection [slGetOutputPort]
  scalarSurface SetOffset 0
  scalarSurface SetOnRatio 2
  scalarSurface PassLinesOn
  scalarSurface SetRuledModeToPointWalk
  scalarSurface SetDistanceFactor 30
vtkPolyDataMapper mapper
  mapper SetInputConnection [scalarSurface GetOutputPort]
  eval mapper SetScalarRange [[pl3d GetOutput] GetScalarRange]
vtkActoractor
  actor SetMapper mapper
```

vtkRuledSurfaceFilter一个优点是当输入多个流线时可以关闭带功能，这样有助于理解曲面的结构。

切割（Cutting）

VTK中切割或者切面化数据集意味着在数据集中利用任意类型的隐函数来创建交叉区域。例如我们可以采用一个平面来创建一个平面切面对数据集进行切面显示。切面进行切割时对数据进行插值，然后采用任何一个标准可视化技术来显示。切割结果通常是vtkPolyData类型（n维对象的切割结果是一个n-1维几何体。例如，切割一个四面体将产生一个三角形或者四边形）。

下面Tcl实例中演示了一个燃烧室被平面切割的结果，如图5-6。代码摘自VTK/Graphics/Tesing/Tcl/pro 。

```
vtkPlane plane
    eval plane SetOrigin [[pl3d GetOutput] GetCenter]
    plane SetNormal -0.287 0 0.9579
vtkCutter planeCut
    planeCut SetInputConnection [pl3d GetOutputPort]
    planeCut SetCutFunction plane
vtkProbeFilter probe
    probe SetInputConnection [planeCut GetOutputPort]
    probe SetSourceConnection [pl3d GetOutputPort]
vtkDataSetMappercutMapper
    cutMapper SetInputConnection [probe GetOutputPort]
    eval cutMapper SetScalarRange \
      [[[[pl3d GetOutput] GetPointData] GetScalars] GetRange]
vtkActor cutActor
cutActor SetMapper cutMapper
```

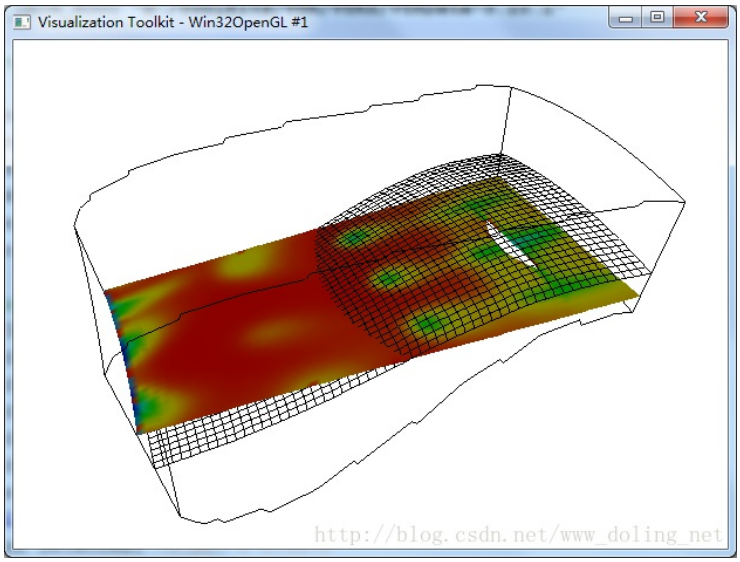

图5-6 燃烧室切割效果图

vtkCutter需要指定完成切割的隐函数。另外，你可能希望指定一个或者多个切割值，可以通过SetValue()或者GenerateValues()函数实现。这些数值指定了执行切割的隐式函数值。（默认情况下切割值为0 ，即切面精确地位于隐函数曲面，小于或者大于0值的曲面则位于该隐曲面的下面和上面。切割值可以看做是到隐曲面的“距离”。）

合并数据（MergingData）

到目前为止我们已经了解了简单的线性可视化管线。然后，管线还可以存在分支，合并甚至是循环情况。接下来我们介绍两个Filter，它们可以利用其他的数据集来构建新的数据集。现在从vtkMergeFilter开始。

vtkMergeFilter将多个数据集中的数据片段合并为一个新的数据集。例如，可以将一个数据集的结构（拓扑和几何），第二个数据集的标量数据，第三个数据集的向量数据合并为一个数据集。这里是该应用的一个实例。（代码摘自VTK/Examples/VisualizationAlgorithms/Tcl/imageWarp.tcl）。（主要关注vtkMergeFilter，其他的不熟悉的可以先忽略。在第106页我们将更加详细的进行描述。）

```
vtkBMPReader reader
  reader SetFileName $VTK_DATA_ROOT/Data/masonry.bmp
vtkImageLuminance luminance
  luminance SetInputConnection [reader GetOutputPort]
vtkImageDataGeometryFilter geometry
  geometry SetInputConnection [luminance GetOutputPort]
vtkWarpScalarwarp
  warp SetInputConnection [geometry GetOutputPort]
  warp SetScaleFactor -0.1

vtkMergeFilter merge
  merge SetGeometryConnection [warp GetOutputPort]
  merge SetScalarsConnection  [reader GetOutputPort]
vtkDataSetMapper mapper
  mapper SetInputConnection [merge GetOutputPort]
  mapper SetScalarRange 0 255
  mapper ImmediateModeRenderingOff
vtkActor actor
  actor SetMapper mapper<span style="font-size: 14px; line-height: 1.846153846; font-family: Arial; background-color: rgb(255, 255, 255);"> </span>
```

这里所做的是将vtkWarpScalar（输出是vtkPolyData类型）的输出与vtkBMPReader标量数据合并到一起。管线先分后合，因为几何结构需要利用标量数据来单独处理。

当合并数据时，数组中的元组数必须与点的个数一致，单元数据也是一样。

追加数据（AppendingData）

类似vtkMergeFilter，vtkAppendFilter以及vtkAppendPolyData通过追加数据集来产生新数据集。追加Filter接收一系列输入，这些输入的类型必须一致。在追加操作中，只有那些共同的数据数据才会合并在一块。下一节中演示了一个比较好的实例。

探测（Probing）

探测是利用其它数据集对数据集进行采样的过程。在VTK中，可以利用任意的数据集来作为探测几何，其点属性则由其它数据集映射而来。例如下例中创建了三个平面作为探测几何来对一个结构网格数据集进行采样。然后这些平面通过vtkContourFilter来计算等值线。代码摘自VTK/Examples/VisualizationAlgorithms/Tcl/probeComb.tcl。

```
vtkPLOT3DReader pl3d
    pl3d SetXYZFileName "$VTK_DATA_ROOT/Data/combxyz.bin"
    pl3d SetQFileName "$VTK_DATA_ROOT/Data/combq.bin"
    pl3d SetScalarFunctionNumber 100
    pl3d SetVectorFunctionNumber 202
pl3d Update

vtkPlaneSource plane
    plane SetResolution 50 50
vtkTransform transP1
    transP1 Translate 3.7 0.0 28.37
    transP1 Scale 5 5 5
    transP1 RotateY 90
vtkTransformPolyDataFilter tpd1
    tpd1 SetInputConnection [plane GetOutputPort]
    tpd1 SetTransform transP1
vtkOutlineFilter outTpd1
    outTpd1 SetInputConnection [tpd1 GetOutputPort]
vtkPolyDataMapper mapTpd1
    mapTpd1 SetInputConnection [outTpd1 GetOutputPort]
vtkActort pd1Actor
    tpd1Actor SetMapper mapTpd1
    [tpd1Actor GetProperty] SetColor 0 0 0

vtkTransform transP2
    transP2 Translate 9.2 0.0 31.20
    transP2 Scale 5 5 5
    transP2 RotateY 90
vtkTransformPolyDataFilter tpd2
    tpd2 SetInputConnection [plane GetOutputPort]
    tpd2 SetTransform transP2
vtkOutlineFilter outTpd2
    outTpd2 SetInputConnection [tpd2 GetOutputPort]
vtkPolyDataMapper mapTpd2
    mapTpd2 SetInputConnection [outTpd2 GetOutputPort]
vtkActor tpd2Actor
    tpd2Actor SetMapper mapTpd2
    [tpd2Actor GetProperty] SetColor 0 0 0

vtkTransform transP3
    transP3 Translate 13.27 0.0 33.30
    transP3 Scale 5 5 5
    transP3 RotateY 90
vtkTransformPolyDataFilter tpd3
    tpd3 SetInputConnection [plane GetOutputPort]
    tpd3 SetTransform transP3
vtkOutlineFilter outTpd3
    outTpd3 SetInputConnection [tpd3 GetOutputPort]
vtkPolyDataMapper mapTpd3
    mapTpd3 SetInputConnection [outTpd3 GetOutputPort]
vtkActort pd3Actor
    tpd3Actor SetMapper mapTpd3
    [tpd3Actor GetProperty] SetColor 0 0 0

vtkAppendPolyData appendF
    appendF AddInputConnection [tpd1 GetOutputPort]
    appendF AddInputConnection [tpd2 GetOutputPort]
appendF AddInputConnection [tpd3 GetOutputPort]
vtkProbeFilter probe
    probe SetInputConnection  [appendF GetOutputPort]
    probe SetSourceConnection [pl3d GetOutputPort]
vtkContourFilter contour
    contour SetInputConnection [probeGetOutputPort]
evalcontour GenerateValues 50 [[pl3d GetOutput] GetScalarRange]

vtkPolyDataMapper contourMapper
    contourMapper SetInputConnection [contour GetOutputPort]
evalcontourMapper SetScalarRange [[pl3d GetOutput] GetScalarRange]

vtkActor planeActor
planeActor SetMapper contourMapper
```

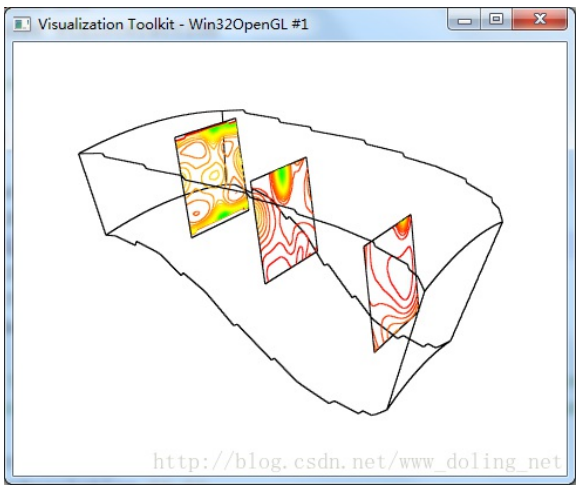

图5-7 数据探测
注意通过SetInputConnection()方法来设置探测器，而待探测数据集是通过SetSourceConnection()函数设置。

探测的另外一个应用是重采样数据。例如，如果你有一个无结构网格数据，而你想通过专门可视化vtkImageData的工具来显示（例如体绘制，139页）。那么可以采用vtkProbeFilter对无结构网格数据采用为一个体数据，然后再可视化。同样也可以采用直线来探测数据，并将结果绘制成X-Y曲线。

最后值得注意的是，切割和探测都能得到类似的结果，除了在分辨率上有所差别外。类似于98页的切割实例，vtkProbeFilter可以利用vtkPlaneSource来产生一个平面，而该平面的属性数据来自于结构网格数据。然而，切割产生的曲面分辨率要依赖于输入数据。而探测产生的曲面分辨率则独立于输入数据。因此在探测数据时，需要特别注意低采用或者过采样。低采样会导致显示错误，过采样则会占用过多的计算时间。

用其他标量着色等值面

计算等值面并用另一个标量进行着色是一个常见的可视化操作。可能你会想到用探测器，但是如果你等值面如果包含你想用来着色的数据时，会有一个更加有效的方法。因为vtkContourFilter（实际用来产生等值面）在计算过程中会对所有数据进行插值。插值后的数据在映射过程中用来进行着色。下面例子取自VTK/Examples/VisualizationAlgorithms/Tcl/ColorIsosurface.tcl。

```
vtkPLOT3DReader pl3d
    pl3d SetXYZFileName "$VTK_DATA_ROOT/Data/combxyz.bin"
    pl3d SetQFileName "$VTK_DATA_ROOT/Data/combq.bin"
    pl3d SetScalarFunctionNumber 100
    pl3d SetVectorFunctionNumber 202
    pl3d AddFunction 153
    pl3d Update
 vtkContourFilter iso
    iso SetInputConnection [pl3d GetOutputPort]
    iso SetValue 0.24

vtkPolyDataNormals normals
    normals SetInputConnection [iso GetOutputPort]
normals SetFeatureAngle 45

vtkPolyDataMapper isoMapper
    isoMapper SetInputConnection [normals GetOutputPort]
    isoMapper ScalarVisibilityOn
    isoMapper SetScalarRange 0 1500
    isoMapper SetScalarModeToUsePointFieldData
    isoMapper ColorByArrayComponent "VelocityMagnitude" 0

vtkLODActor isoActor
    isoActor SetMapper isoMapper
    isoActor SetNumberOfCloudPoints 1000
```

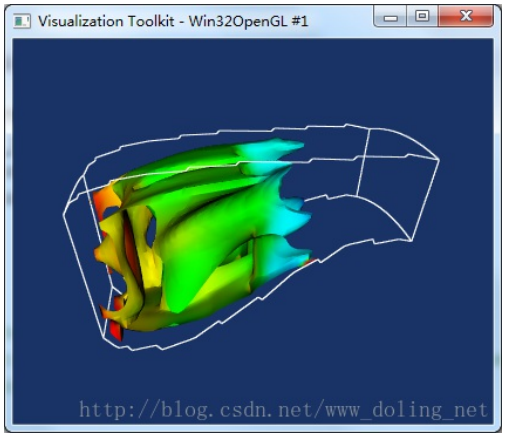

图5-8 用其他标题着色等值面
首先通过vtkPLOT3DReader读取数据集。这里我们添加一个要读的函数（函数号153），函数名字是“Velocity Magnitude”。计算等值面时，也会对所有输入数据包括速度场数据进行插值。然后我们调用SetScalarModeToUsePointFieldData()函数来利用速度模值来对等值面着色，ColorByArrayComponent()方法用来指定着色所用的数据数组。

提取单元子集

可视化数据的数据量一般都非常大，处理这样的数据往往要耗费大量的时间和内存。因此，提取部分数据功能显示非常重要。多数情况下，只有部分数据包含有意义的信息，或者在不影响精度的条件下对数据进行消减。

VTK中提供了许多工具来提取部分数据或者对数据降采样。我们已经了解到vtkProbeFilter可以用来进行降采样（见100页“Probing”）。其他工具包含降采样类，还有一些工具能够实现从空间区域中提取单元（降采样工具针对于特定数据，详见105页“降采样图像数据”，和113页“降采样规则网格数据”）。本节中我们主要讲述怎样在空间区域中提取数据片段。

vtkExtractGeometryl类提取数据集中位于隐函数曲面vtkImplicitFucntion内部或者外部的所有单元(注意，隐函数可以是多个隐式函数的二值组合)。下面代码创建了两个椭球的二值组合用来提取区域。vtkShrinkFilter用来对单元进行收缩以便能观察被提取的数据。（代码取自VTK/Examples/VisualizationAlgorithms/Tcl/ExtractionGeometry.tcl.）

```
vtkQuadric quadric
    quadric SetCoefficients .5 1 .2 0 .1 0 0 .20 0
vtkSampleFunction sample
    sample SetSampleDimensions 50 50 50
    sample SetImplicitFunction quadric
    sample ComputeNormalsOff
vtkTransform trans
    trans Scale 1 .5 .333
vtkSphere sphere
    sphere SetRadius 0.25
    sphere SetTransform trans
vtkTransform trans2
    trans2 Scale .25 .5 1.0
vtkSphere sphere2
    sphere2 SetRadius 0.25
    sphere2 SetTransform trans2
vtkImplicitBoolean union
    union AddFunction sphere
    union AddFunction sphere2
    union SetOperationType 0;#union

vtkExtractGeometryextract
    extract SetInputConnection [sampleGetOutputPort]
    extract SetImplicitFunction union
vtkShrinkFilter shrink
    shrink SetInputConnection [extract GetOutputPort]
    shrink SetShrinkFactor 0.5
vtkDataSetMapper dataMapper
    dataMapper SetInputConnection [shrink GetOutputPort]
vtkActor dataActor
dataActor SetMapper dataMapper
```

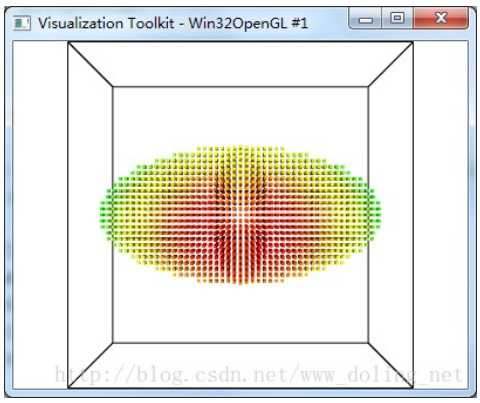

图5-9 提取单元的子集

vtkExtractGeometry的输出通常是vtkUnstructuredGrid类型。因为在提取过程中，数据集的拓扑结构往往被破坏，因此必须采用最普遍的数据格式来表示输出。

注意，隐式函数可以通过分配一个vtkTransform来进行变换。如果指定了变换，vtkTransform就被用来改变隐式函数的取值。你可能希望对该功能进行测试。

以多边形数据类型为输出提取单元

大部分数据类型不能直接被图形硬件或者图形库渲染。渲染系统中只有多边形数据（vtkPolyData）被广泛的支持。规则数据集，特别是图像以及体数据也可以被图形系统支持。而其他的数据则需要特别的处理才能被渲染。在VTK中，渲染非多边形数据的一个方法是将其转换为多边形数据。vtkGeometryFilter可实现该功能。

vtkGeometryFilter接收任意vtkDataSet类型数据并输出vtkPolyData数据。其执行转换时，遵循如下规则。所有二维及以下拓扑单元（如多边形，直线，顶点）直接传递至输出。位于数据集边界的三维单元面会被传递至输出结果中。（如果一个面只属于一个单元，那么这个面位于边界。）

vtkGeometryFilter常被用来进行数据格式转换。下面代码中利用vtkGeometryFilter来将二维不规则网格转换为多边形数据，这些多边形数据接下来会作为其他接收vtkPolyData的Filter所接收。代码取自VTK/Examples/DataManipulation/Tcl/pointToCellData.tcl。这里vtkConnectivityFilter提取vtkUnstructedGrid数据然后通过vtkGeometryFilter将其转换为多边形数据。

```
vtkConnectivityFilter connect2
    connect2 SetInputConnection [thresh GetOutputPort]
vtkGeometryFilter parison
    parison SetInputConnection [connect2 GetOutputPort]
vtkPolyDataNormals normals2
    normals2 SetInputConnection [parison GetOutputPort]
    normals2 SetFeatureAngle 60
vtkLookupTable lut
    lut SetHueRange 0.0 0.66667
vtkPolyDataMapper parisonMapper
    parisonMapper SetInputConnection [normals2 GetOutputPort]
    parisonMapper SetLookupTable lut
    parisonMapper SetScalarRange 0.12 1.0
vtkActor parisonActor
parisonActor SetMapper parisonMapper
```

实际上vtkDataSetMapper内部使用vtkGeometryFilter来将任意数据类型转换为多边形数据。（该Filter可以直接将输入的vtkPolyData传递到输出。）

另外vtkGeometryFilter提供了一系列函数来根据点id集合，单元ids集合或者判断是否位于一个特定的矩形空间区域中来提取数据单元。利用点或者面id集合提取数据片段时，使用的函数有PointClippingOn(),SetPointMinimum(),SetPointMaximum()和CellClippingOn(),SetCellMinimum(),SetCellMaximum()。最小值和最大值指定了提取的id 的范围。同样还可以指定一个空间矩形区域来限制提取的范围。用ExtentClippingOn()和SetExtent()来开始空间切割和指定范围。Extent包含了六个参数来定义一个包围盒-（
，）。可以使用这三种方式的任意组合来提取数据。在调试数据时会非常有用，或者当你只想观察部分数据时。

第05章-可视化技术（3）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本小节内容对应原书的第105页至第112页】

5.2 多边形数据可视化

多边形数据（vtkPolyData）是一种重要的可视化数据类型。因为它是图像硬件或者渲染引擎的几何接口。其他的数据必须转换为多边形数据才能被渲染，除了vtkImageData外。VtkImageData使用的时特别图像或者体绘制技术。可以参考104页"以多边形数据类型为输出提取单元"来了解怎样转换数据类型。

多边形数据主要由点和点集，直线和直线组合，三角形，四边形，多边形和三角形带组合而成。大部分接收vtkPolyData的Filter都会处理这些数据组合。当然也有一些Filter如vtkDecimatePro和vtkTubeFilter会只处理部分数据类型（三角网格和直线）。

手动创建vtkPolyData数据

多边形数据可以通过多种方式构造。通常需要创建一个vtkPoints来存储点集，然后创建4个vtkCellArrays来分别表示点，线，多边形和三角带连接。下面例子取自VTK/Examples/DataManipulation/Tcl/CreateStrip.tcl。它创建了一个具有一个三角形带的vtkPolyData数据。

```
vtkPoints points
    points InsertPoint 0 0.0 0.0 0.0
    points InsertPoint 1 0.0 1.0 0.0
    points InsertPoint 2 1.0 0.0 0.0
    points InsertPoint 3 1.0 1.0 0.0
    points InsertPoint 4 2.0 0.0 0.0
    points InsertPoint 5 2.0 1.0 0.0
    points InsertPoint 6 3.0 0.0 0.0
points InsertPoint 7 3.0 1.0 0.0

vtkCellArray strips
    strips InsertNextCell 8;#number of points
    strips InsertCellPoint 0
    strips InsertCellPoint 1
    strips InsertCellPoint 2
    strips InsertCellPoint 3
    strips InsertCellPoint 4
    strips InsertCellPoint 5
    strips InsertCellPoint 6
    strips InsertCellPoint 7
vtkPolyData profile
    profile SetPoints points
    profile SetStrips strips
vtkPolyDataMapper map
    map SetInput profile
vtkActor strip
    strip SetMapper map
[strip GetProperty] SetColor 0.3800 0.7000 0.1600
```

这里有另外一个C++例子来演示怎样创建一个立方体（VTK/Examples/DataManipulation/Cxx/Cube.cxx）。这次我们创建6个四边形和每个顶点对应的标量值。

```
int i;
  staticfloat x[8][3]={{0,0,0}, {1,0,0}, {1,1,0}, {0,1,0},
                       {0,0,1}, {1,0,1},{1,1,1}, {0,1,1}};
  staticvtkIdType pts[6][4]={{0,1,2,3}, {4,5,6,7}, {0,1,5,4},
                       {1,2,6,5}, {2,3,7,6},{3,0,4,7}};

  vtkPolyData *cube = vtkPolyData::New();
  vtkPoints *points = vtkPoints::New();
  vtkCellArray *polys = vtkCellArray::New();
  vtkFloatArray *scalars = vtkFloatArray::New();

// Load the point, cell, and data attributes.
  for (i=0;i<8; i++) points->InsertPoint(i,x[i]);
  for (i=0; i<6;i++) polys->InsertNextCell(4,pts[i]);
  for (i=0;i<8; i++) scalars->InsertTuple1(i,i);

// We now assign the pieces to the vtkPolyData.
 cube->SetPoints(points);
 points->Delete();
 cube->SetPolys(polys);
 polys->Delete();
 cube->GetPointData()->SetScalars(scalars);
 scalars->Delete();
```

vtkPolyData可以由点，线，多边形和三角形带的任意组合构建。同时，vtkPolyData支持一系列扩展的操作来对数据结构进行编辑和修改。参考345页“多边形数据”。

计算表面法向量

当你渲染多边形网格时，你可能会发现渲染图像中清晰的显示了网格面片（图5-10）。采用Gouraud着色技术可以提高渲染结果（见53页“Actor属性”）。然而Gouraud着色技术依赖于网格每个顶点的法向量。因此需要利用vtkPolyDataNoramals filter来计算网格顶点法向量。217页“Extrusion”、94页“Glyphing”和102页“用其他标量着色等值面”都是用了该类计算法向量。

图5-10 使用和不使用表面法向量的网格面片对比

该Filter有两个重要的参数，splitting和FeatureAngle。如果splitting开启，那么特征边被分裂（如果一条边任意一端的顶点法向与该边所成夹角大于或者等于特征角FeatureAngle的话，那么该边定义为特征边）。也就是沿着该条边复制顶点，网格在特征边的任意一边被分割。这样虽然产生了新的顶点，但是能够清晰的渲染尖锐角处。另外一个重要的参数是FlipNormals。调用函数FlipNormalsOn()将使法向量反向（同时多边形连接表也会反向）。

抽取（Decimation）

多边形数据，特别是三角形网格是很常用的图形数据格式。比如vtkContourFilter产生的结果就是三角形网格。但是这些三角形网格往往会比较大，因此在一些交互应用中难以快速的处理。抽取技术常被用来解决这个问题。抽取也被称作多边形消减，网格简化或者多分辨率建模。其在保持近似原始网格条件下对三角网格中的三角形数量进行消减。

VTK支持三种消减方法，vtkDecimatePro,vtkQuadricClustering和vtkQuadricDecimation。虽然每一种都有各自的优缺点，但在应用上基本都一致。

vtkDecimatePro相对来说速度比较快，而且在消减过程中可以修改拓扑结构。它采用边塌陷方法来去除顶点和三角形。它的误差度量是基于到平面或者边的距离。该方法的一个优点是可以完成任意程度的消减，因为该算法一开始就将网格分裂为小的碎片来完成消减目的（如果允许拓扑修改的话）。

vtkQuadricDecimation采用的Siggraph97论文“SurfaceSimplification Using Quadric Error Metrics”中提出的二次误差度量。它采用边塌陷方法来剔除顶点和三角面片。二次误差度量通常被认为是比较好的误差度量。

vtkQuadricClustering是最快的方法。它的思想基于Siggraph2000中的论文“Out-of-Core Simplification of Large Polygonal Models”。它能够快速的消减大网格模型，并且支持网格片段消减（利用StartAppend(), Append()和EndAppend()方法）。这样可以避免读取整个模型到内存中。对于大网格模型，该方法有较好的效果；但是当网格变小时，三角化过程效果不是很好（需要结合其他的算法会有较理想的效果）。

下面是关于类vtkDecimatePro的一个应用实例。代码取自VTK/Examples/VisualizationAlgorithms/Tcl/deciFram.tcl（如图5-11）。

```
vtkDecimatePro deci
    deci SetInputConnection [fran GetOutputPort]
    deci SetTargetReduction 0.9
    deci PreserveTopologyOn
vtkPolyDataNormals normals
    normals SetInputConnection [deci GetOutputPort]
    normals FlipNormalsOn
vtkPolyDataMapper franMapper
   franMapper SetInputConnection [normals GetOutputPort]
vtkActor franActor
    franActor SetMapper franMapper
eval [franActor GetProperty] SetColor 1.0 0.49 0.25
```

图5-11 三角网格面片经90%抽取后的效果对比（左为原始图，右为抽取后的图）
vtkDecimatePro有两个重要的参数：TargetReduction和PreserveTopology。TargetReduction是用户的消减量（例如0.9意味着我们希望减少网格中90%的面片。）根据你是否允许修改拓扑结构（PreserveTopologyOn/Off()）来得到相应的消减结果。

最后注意的是消减filter的输入为三角形数据。如果你的是多边形数据，那么需要利用vtkTriangleFilter 来进行转换。

网格平滑（SmoothMesh）

多边形网格往往由于噪声或者过于粗糙，从而影响渲染的结果。例如，低分辨率数据的等值面会有走样的效果。数据平滑是处理这种问题的方法之一。平滑是通过改变顶点位置来减少曲面噪声数据的的过程。

VTK中提供了两种平滑对象：vtkSmoothPolyDataFilter和vtkWindowSincPolyDataFilter。两者中，vtkWindowSincPolyDataFilter处理效果最好，而且处理速度也较快。下面例子（代码来自VTK/Examples/VisualizationAlgorithms/Tcl/smoothFran.tcl）演示了怎样利用该平滑filter。该例与上节例子内容相同，只是多了一个平滑filter处理。图5-12显示了简化模型的平滑效果。

```
vtkDecimatePro deci
    deci SetInputConnection [fran GetOutputPort]
    deci SetTargetReduction 0.9
    deci PreserveTopologyOn
vtkSmoothPolyDataFilter smoother
    smoother SetInputConnection [deci GetOutputPort]
    smoother SetNumberOfIterations 50
vtkPolyDataNormals normals
    normals SetInputConnection [smoother GetOutputPort]
    normals FlipNormalsOn
vtkPolyDataMapper franMapper
   franMapper SetInputConnection [normals GetOutputPort]
vtkActor franActor
    franActorSetMapper franMapper
    eval [franActor GetProperty] SetColor 1.0 0.49 0.25
```

图5-12 多边形网格平滑（右图为平滑后的效果）
两种平滑Filter的使用方法类似。他们提供了一些函数来控制特征边或者边界处的平滑效果。可查阅在线文档或者*.h头文件了解相关信息。

数据裁剪（ClipData）

裁剪类似于切割（98页“切割”）利用一个隐函数来定义裁剪面。裁剪将多边形模型分解为片段，如图5-13所示。裁剪在裁剪面的任意一边将多边形几何体分解为单独的部分。如切割一样，裁剪可以设置一个裁剪值来定义隐式裁剪函数的值。

下例中利用一个平面来裁剪一个牛的三维多边形模型。切割值用来定义切割平面沿着法向量移动距离的大小，这样便可以在不同的位置对模型进行裁剪。代码取自（VTK/Examples/VisualizationAlgorithms/Tcl/ClipCow.tcl）。

```
vtkBYUReader cow
  cow SetGeometryFileName "$VTK_DATA_ROOT/Data/Viewpoint/cow.g"
vtkPolyDataNormals cowNormals
  cowNormals SetInputConnection [cow GetOutputPort]

vtkPlane plane
    plane SetOrigin 0.25 0 0
    plane SetNormal -1 -1 0
vtkClipPolyData clipper
    clipper SetInputConnection [cowNormals GetOutputPort]
    clipper SetClipFunction plane
    clipper GenerateClipScalarsOn
    clipper GenerateClippedOutputOn
    clipper SetValue 0.5
vtkPolyDataMapper clipMapper
   clipMapper SetInputConnection [clipper GetOutputPort]
   clipMapper ScalarVisibilityOff
vtkProperty backProp
    evalbackProp SetDiffuseColor $tomato
vtkActor clipActor
    clipActorSetMapper clipMapper
    eval [clipActor GetProperty] SetColor $peacock
    clipActor SetBackfaceProperty backProp

vtkPolyDataMapper restMapper
  restMapper SetInputConnection [clipper GetClippedOutputPort]
  restMapper ScalarVisibilityOff
vtkActor restActor
  restActor SetMapper restMapper
  [restActor GetProperty] SetRepresentationToWireframe
```

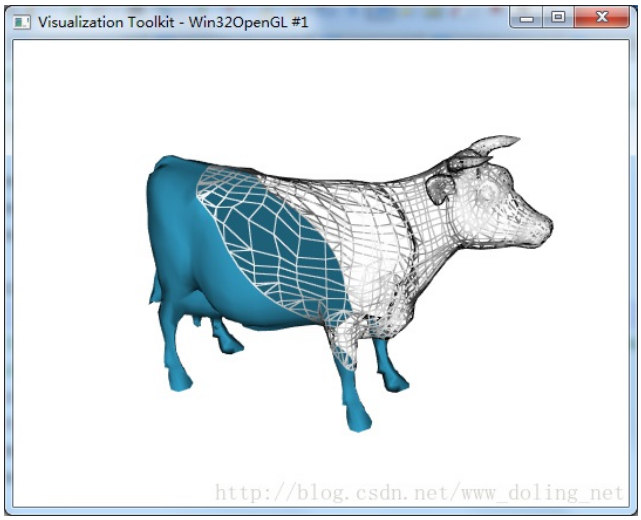

图5-13 模型裁剪

GenerateClippedOutputOn()函数计算裁剪过程中被裁剪掉得数据。在该例中这部分数据采用线框方式显示。如果通过SetValue()函数改变了裁剪值，那么裁剪面将移动至初始裁剪面前或者后并且平行与初始裁剪面的位置。（同样改变vtkPlane的定义也可实现同样的效果。）

计算纹理坐标

常用的计算纹理坐标的Filter是vtkTextureMapToPlane，vtkTextureMapToCylinder和vtkTextureMapToSphere。他们分别基于平面，圆柱和球面坐标系计算纹理坐标。另外，vtkTransformTextureCoordinates通过平移和缩放纹理坐标在曲面上定义纹理映射。下例中演示了利用vtkTextureMapToCylinder来为一个由vtkDelaunay3D计算产生的不规则网格数据计算纹理坐标（参考218页“Delaunay Triangulation”）。完整代码可以参考VTK/Examples/VisualizationAlgorithms/Tcl/GenerateTextureCoords.tcl。

```
vtkPointSource sphere
  sphere SetNumberOfPoints 25

vtkDelaunay3D del
  del SetInputConnection [sphere GetOutputPort]
  del SetTolerance 0.01

vtkTextureMapToCylinder tmapper
  tmapper SetInputConnection [del GetOutputPort]
  tmapper PreventSeamOn

vtkTransformTextureCoords xform
  xform SetInputConnection [tmapper GetOutputPort]
  xform SetScale 4 4 1

vtkDataSetMapper mapper
  mapper SetInputConnection [xform GetOutputPort]

vtkBMPReader bmpReader
  bmpReaderSetFileName "$VTK_DATA_ROOT/Data/masonry.bmp"
vtkTexture atext
  atext SetInputConnection [bmpReader GetOutputPort]
  atext InterpolateOn
vtkActor triangulation
 triangulation SetMapper mapper
 triangulation SetTexture atext
```

该例中首先对一个单位球上的随机点进行三角剖分，并为三角网格计算纹理坐标。接着在纹理坐标i-j方向上对纹理坐标进行缩放以便使纹理能够重复。最后读入纹理图像并指定到Actor上。

（注意，vtkDataSetMapper可以接收任意数据类型的输入。其内部vtkGeometryfilter对象将数据转换为多边形数据然后再传入到渲染引擎中。参考104页“以多边形数据为输出提取单元”）

如果需要深入学习纹理坐标，可以运行VTK/Examples/VisualizationAlgorithms/Tcl/TransformTexturecoords.tcl实例（如图5-14所示）。该例GUI可以选择多边形模型，纹理图像，纹理计算方法和纹理变换方法。

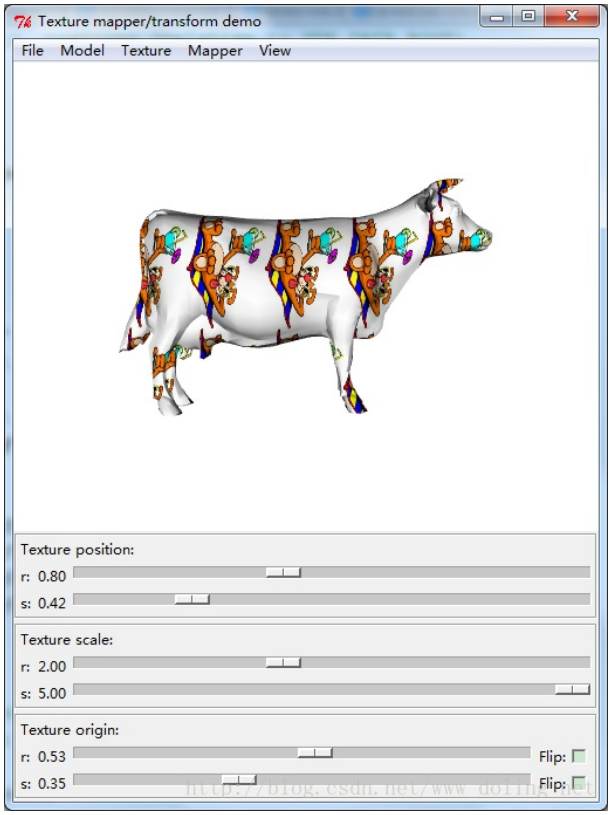

图5-14 变换以及应用不同的纹理

第05章-可视化技术（4）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本小节内容对应原书的第112页至第117页】

5.3 规则网格可视化

规则网格在拓扑结构上规则，而几何上是不规则的，如图3-2所示。规则网格常被用来进行数据分析（如流体动力学计算）。规则网格vtkStructuredGrid数据集主要有六面体（vtk Hexahedral）单元或者是四边形（vtkQuad）单元组成。

手动创建vtkStructedGrid

创建规则网格需要指定网格的维数（定义拓扑结构）和定义x-y-z坐标的vtkPoints对象（定义几何结构）。下面代码改自VTK/Examples/DataManiulation/Cxx/SGrid.cxx。

```
vtkPoints points

points InsertPoint 0 0.0 0.0 0.0

…etc…


vtkStructedGrid sgrid

sgrid SetDimensions 13 11 11

sgrid SetPoints points
```

主要vtkPoints对象中点的个数要与网格i，j和k三个方向的维数乘积保持一致。

提取计算平面

大多数情况下规则网格由接收vtkDataSet类型输入的filter来处理（见89页"可视化技术"）。而直接接收vtkStructuredGrid类型输入的其中一个filter是vtkStructuredGridGeometryFilter。该Filter根据其Extent变量值来提取网格中的局部点、线段或者平面（Extent是一个描述区域的六维向量（，））。

下例中我们读取了一个规则网格数据，提取其中的三个平面，并利用相关的向量数据对平面进行Warp处理（代码取自VTK/Examples/VisualizationAlgorithms/Tcl/warpComb.tcl）。

```
vtkPLOT3DReader pl3d

pl3d SetXYZFileName "$VTK_DATA_ROOT/Data/combxyz.bin"

pl3d SetQFileName "$VTK_DATA_ROOT/Data/combq.bin"

pl3d SetScalarFunctionNumber 100

pl3d SetVectorFunctionNumber 202

pl3d Update

vtkStructuredGridGeometryFilter plane
```

```
plane SetInputConnection [pl3d GetOutputPort]

plane SetExtent 10 10 1 100 1 100

vtkStructuredGridGeometryFilter plane2

plane2 SetInputConnection [pl3d GetOutputPort]

plane2 SetExtent 30 30 1 100 1 100

vtkStructuredGridGeometryFilter plane3

plane3 SetInputConnection [pl3d GetOutputPort]

plane3 SetExtent 45 45 1 100 1 100

vtkAppendPolyData appendF

appendF AddInputConnection [plane GetOutputPort]

appendF AddInputConnection [plane2 GetOutputPort]

appendF AddInputConnection [plane3 GetOutputPort]

vtkWarpScalar warp

warp SetInputConnection [appendF GetOutputPort]

warp UseNormalOn

warp SetNormal 1.0 0.0 0.0

warp SetScaleFactor 2.5

vtkPolyDataNormals normals

normals SetInputConnection [warp GetOutputPort]

normals SetFeatureAngle 60

vtkPolyDataMapper planeMapper

planeMapper SetInputConnection [normals GetOutputPort]

evalplaneMapper SetScalarRange [[pl3d GetOutput] GetScalarRange]

vtkActor planeActor

planeActor SetMapper planeMapper
```

规则网格数据降采样

如图像数据一样，规则网格数据也可以进行降采样（见105页“图像数据降采样”）。vtkExtractGrid用来执行数据降采样和提取。

```
vtkPLOT3DReader pl3d

pl3d SetXYZFileName "$VTK_DATA_ROOT/Data/combxyz.bin"

pl3d SetQFileName "$VTK_DATA_ROOT/Data/combq.bin"

pl3d SetScalarFunctionNumber 100

pl3d SetVectorFunctionNumber 202

pl3d Update
```

vtkExtractGrid extract

extract SetInputConnection [pl3d GetOutputPort]

extract SetVOI 30 30 -1000 1000 -1000 1000

extract SetSampleRate 1 2 3

extract IncludeBoundaryOn

在该例子中初始规则网格（维数为57X33X25）按照（1,2,3）的采样率进行采样，生成一个维数（1,17,9）的子集。函数IncludeBoundaryOn()保证了在没有选择边界的采样操作下也将边界提取出来。

5.4 线性网格可视化

线性网格在拓扑上规则，但是在几何上半规则（如图3-2（b））。线性网格也常用于数值分析。vtkRectilinearGrid由体素（vtkVoxel）或者像素（vtkPixel）单元组成。

手动创建vtkRectlinearGrid

线性网格创建需要指定网格维数定义网格拓扑，以及三个标量数组定义x-y-z三个方向坐标来定义几何。下面代码改自VTK/Examples/DataManipulation/Cxx/RGrid.cxx。

vtkFloatArray *xCoords = vtkFloatArray::New();

for (i=0;i<47; i++) xCoords->InsertNextValue(x[i]);

vtkFloatArray*yCoords = vtkFloatArray::New();

for (i=0;i<33; i++) yCoords->InsertNextValue(y[i]);

vtkFloatArray *zCoords = vtkFloatArray::New();

for (i=0;i<44; i++) zCoords->InsertNextValue(z[i]);

vtkRectilinearGrid *rgrid =vtkRectilinearGrid::New();

rgrid->SetDimensions(47,33,44);

rgrid->SetXCoordinates(xCoords);

rgrid->SetYCoordinates(yCoords);

rgrid->SetZCoordinates(zCoords);

确保x，y，z三个方向的标量个数要与拓扑i,j,k三个方向维数保持一致。

提取计算平面

大多数情况下线性网格由接收vtkDataSet类型输入的Filter来处理（见89页“可视化技术”）。而直接接收vtkRectilinearGrid类型输入的其中一个Filter是vtkRectilinearGridGeometryFilter。该Filter根据其Extent变量值来提取网格中的局部点、线段或者平面（Extent是一个描述区域的六维向量（，））。

下面例子还是紧接上面代码VTK/Examples/DataManipulation/Cxx/RGrid.cxx，我们如下提取一个平面：

vtkRectilinearGridGeometryFilter *plane =

vtkRectilinearGridGeometryFilter::New();

```
plane->SetInput(rgrid);

plane->SetExtent(0,46, 16,16, 0,43);
```

5.5 不规则网格可视化

不规则网格无论在拓扑上还是几何上都是不规则的（如图3-2（f））。不规则网格常用来进行数值分析（如有限元分析）。任何一中单元类型都可以用来表示不规则网格。

手动创建vtkUnstructuredGrid

不规则网格定义是通过vtkPoints来定义几何，通过插入单元来定义拓扑。（下面代码取自VTK/Examples/DataManipulation/Tcl/BuildUGrid.tcl.）

```
vtkPoints tetraPoints

tetraPoints SetNumberOfPoints 4

tetraPoints InsertPoint 0 0 0 0

tetraPoints InsertPoint 1 1 0 0

tetraPoints InsertPoint 2 .5 1 0

tetraPoints InsertPoint 3 .5 .5 1

vtkTetra aTetra

[aTetra GetPointIds] SetId 0 0

[aTetra GetPointIds] SetId 1 1

[aTetra GetPointIds] SetId 2 2

[aTetra GetPointIds] SetId 3 3

vtkUnstructuredGrid aTetraGrid

aTetraGrid Allocate 1 1

aTetraGrid InsertNextCell [aTetra GetCellType] [aTetra GetPointIds]

aTetraGrid SetPoints tetraPoints

…insertother cells if any…
```

在为vtkUnstructuredGrid实例插入单元数据前，必须先执行Allocate()函数来分配空间。该函数的两个参数分别表示初始分配的数据空间大小，和需要额外内存时扩展的内存空间大小。一般情况大比较大的值会有较好的效果（因为需要较少的内存重分配操作）。

提取局部网格

大部分情况下，不规则网格由接收vtkDataSet类型输入的Filter处理（见89页“可视化技术”）。直接接收vtkUnstructuredGrid类型输入的其中一个Filter是vtkExtractUnstructuredGrid。这个filter根据指定的点ids，单元ids和几何边界来提取局部网格（Extent参数定义一个包围盒）。下例代码取自VTK/Examples/VisualizationAlgorithms/Tcl/ExtractUGrid.tcl。

```
vtkDataSetReader reader

reader SetFileName "$VTK_DATA_ROOT/Data/blow.vtk"

reader SetScalarsName "thickness9"

reader SetVectorsName "displacement9"

vtkCastToConcrete castToUnstructuredGrid

castToUnstructuredGrid SetInputConnection [reader GetOutputPort]
```

```
vtkWarpVector warp
warp SetInput [castToUnstructuredGrid GetUnstructuredGridOutput]

vtkConnectivityFilter connect
connect SetInputConnection [warp GetOutputPort]
connect SetExtractionModeToSpecifiedRegions
connect AddSpecifiedRegion 0
connect AddSpecifiedRegion 1
vtkDataSetMapper moldMapper
moldMapper SetInputConnection [reader GetOutputPort]
moldMapper ScalarVisibilityOff
vtkActor moldActor
moldActorSetMapper moldMapper
[moldActor GetProperty] SetColor .2 .2 .2
[moldActor GetProperty] SetRepresentationToWireframe

vtkConnectivityFilter connect2
connect2 SetInputConnection [warp GetOutputPort]
connect2 SetExtractionModeToSpecifiedRegions
connect2 AddSpecifiedRegion 2
vtkExtractUnstructuredGrid extractGrid
extractGrid SetInputConnection [connect2 GetOutputPort]
extractGrid CellClippingOn
extractGrid SetCellMinimum 0
extractGrid SetCellMaximum 23
vtkGeometryFilter parison
parison SetInputConnection [extractGrid GetOutputPort]
vtkPolyDataNormals normals2
normals2 SetInputConnection [parison GetOutputPort]
normals2 SetFeatureAngle 60
vtkLookupTable lut
lut SetHueRange 0.0 0.66667
vtkPolyDataMapper parisonMapper
parisonMapper SetInputConnection [normals2 GetOutputPort]
```

parisonMapper SetLookupTable lut

parisonMapper SetScalarRange 0.12 1.0

vtkActor parisonActor

parisonActor SetMapper parisonMapper

该例中我们采用单元裁剪（利用单元id号）和一个连通Filter来提取局部网格。vtkConnectivityFilter(以及vtkPolyDataConnectivityFilter)用来提取数据中的连通部分（当单元共享点时，他们是连通的）。SetExtractionModeToSpecifiedRegions()方法设置提取哪个连通区域。连通filter默认提取最大的连通区域。也可以如该例中那样提取一个特定的区域，不过需要一些额外的实验来对应各个区域。

不规则网格等值线提取

vtkContourGrid用来提取不规则网格的等值线。与一般的vtkContourFilter类相比，这个类更加高效。通常你并不需要直接实例化该类，因为当vtkContourFilter识别到输入数据时vtkUnstructuredGrid类型时，vtkContourFilter内部会自动创建一个vtkContourGrid实例。

以上是可视化技术概述。接下来你可能希望学习图像处理和体绘制技术。也可以参考444页“Filter总结”部分了解VTK中的Filter。

【第5章-可视化技术 翻译完毕】

第06章-图像处理及可视化（1）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本节对应原书中的第119页至第125页】

第6章图像处理及可视化

图像数据，如图6-1所示，在拓扑上和几何上都是规则的，VTK中用vtkImageData表示。规则数据意味着可以用较少的参数如原点，间距和维数就可以隐式定义每个数据点的空间位置。医学及科学扫描设备如CT、MRI、超声仪和共聚焦显微镜等都是产生这种类型的数据。概念上讲，vtkImageData是由体素（vtkVoxel）或者像素（vtkPixel）单元组成。然而，这种数据的规则特性使得可以通过一个简单的数组来存储数据，而不是显示的定义vtkVoxel或者vtkPixel单元。

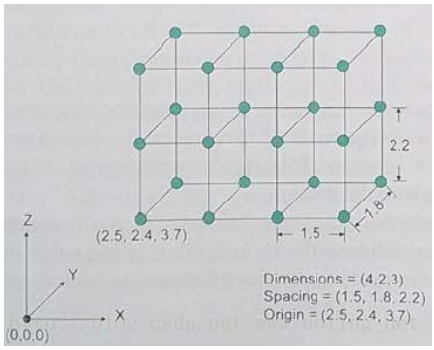

图6-1 vtkImageData结构根据图像的维数、像素间距和原点来定义。维数是每个轴上体素或者像素的个数。原点是第一页图像左下角点的世界坐标。像素间隔则是每个轴上相邻像素的距离。

在VTK中图像数据是一类比较特殊的数据类型，它可以由多种方式进行处理和渲染。虽然没有严格的定义，VTK中处理图像数据的大部分算子可以分为三类：图像处理，几何提取或者是渲染。图像处理Filter数据比较多，他们接收vtkImageData输入，输出的结果也是vtkImageData类型。几何提取Filter是将vtkImageData转换为vtkPolyData类型。例如，vtkContourFilter从图像数据中提取由三角面片组成的等值面。最后还有许多不同的Mapper和专门的Actor来渲染vtkImageData，从简单的二维图像渲染到体绘制。

本章中我们主要学习一些重要的图像处理技术。我们将讨论基本的图像显示、处理和高程图几何提取技术。其他的几何提取技术如等值面已在第5章中介绍。vtkImageData和vtkUnstructuredGrid数据的体绘制技术将在第7章中讲解。

6.1 手动创建vtkImageData

手动创建图像数据非常简单，只需要定义图像的维数，原点和间距。原点是图像左下角点的世界坐标系位置。维数是沿着三个主轴方向上体素或者像素的个数。间距则是体素的长、宽和高，或者是每个方向上相邻像素相邻像素的距离，这将区别与你将图像像素看做是同类的盒子或者是连续函数的采样点。

在第一个例子中我们假设一个包含size[0]*size[1]*size[2]个元素的数组"data"。该数据在VTK外部生成，现在我们需要将其读入到vtkImageData数据中以便用VTK Filter进行处理或者渲染操作。由于我们只是将数据内存指针传到VTK中，因此需要我们自己控制数据内存的释放。

```
vtkUnsignedCharArray*array = vtkUnsignedCharArray::New();

array->SetArray(data,size[0]*size[1]*size[2], 1);
```

接下来创建图像。我们必须保证各个数值的匹配-数据类型必须是标量类型，而且数据长度必须与图像的维数一致。

```
imageData= vtkImageData::New();

imageData->GetPointData()->SetScalars(array);

imageData->SetDimensions(size);

imageData->SetScalarType(VTK_UNSIGNED_CHAR);

imageData->SetSpacing(1.0,1.0, 1.0);

imageData->Set Origin(0.0, 0.0, 0.0);
```

由于图像维数、原点和间距隐式的定义图像数据的几何和拓扑结构，因此表示结构的数据存储需求就非常小。另外，由于数据的规则分布也使得该结构上的计算比较快速。真正需要内存的则是数据集上的属性数据。

下面例子中，我们将用C++语言来创建图像。这次不是直接创建数据数组并将其指定到一个图像中，而是用vtkImageData对象自动创建标量数据。这也将排除掉数据数组的大小与图像维数不一致的问题。

```
//Createthe image data

vtkImageData*id = vtkImageData::New();

id->SetDimensions(10,25, 100);

id->SetScalarTypeToUnsignedShort();

id->SetNumberOfScalarComponents(1);

id->AllocateScalars();

//Fillin scalar values

Unsignedshort *ptr = (unsigned shor*)id->GetScalarPointer();

for(int i=0; i<10*25*100; i++)

{

*ptr ++= i;

}
```

在这个例子中AllocateScalars()方法用来为图像分配内存。需要注意的是，这个方法调用之前，必须先设置标量类型（scalar type）和标量成分的个数（最多4个标量成分）。然后GetScalarPointer()方法，并将其返回的void*结果强制转换成unsigned short类型。我们只能在已知数据类型为unsigned short时才能这样做。RequestData()函数查询标量数据类型，然后然后根据类型选择模板函数来实现。VTK在设计时就尽量避免在公有接口中暴露标量类型为模板参数。这样就为那些缺少模板的语言如Tcl，Java和Python等提供了一个简单的接口函数。

6.2图像降采样

如103页“提取单元子集”，提取部分数据是常见的需求。vtkExtractVOI可以提取输入图像的部分数据。这个Filter也可以对图像降采样，虽然vtkImageReslice（后面会介绍）能够更方便的对数据重采样。其输出类型是vtkImageData。

VTK中有两个类似的Filter来执行裁剪功能：vtkExtractVOI和vtkImageClip。之所以分为两个Filter，是由于历史原因——图像管线与图形管线分开，在图像管线中vtkImageClip只处理vtkImageData数据，而在图形管线中vtkExtractVOI只处理vtkStructuredPoints数据。现在这些区别已经没有了，但是这两个Filter之间还有一些不同。vtkExtractVOI提取一个体数据的子区域，将其输出为一个vtkImageData。另外，vtkExtractVOI也可以在感兴趣区域VOI中进行重采样。另一方面，vtkImageClip默认情况下会保持输出图像数据和输入一致，除了图像范围信息。可以通过设置该filter的一个标志来强制产生精确地数据量，这种情况下对应区域也会直接拷贝到输出vktImageData中。vtkImageClip不能重采样图像。

下面Tcl实例（取自VTK/Examples/ImageProcessing/Tcl/Contours2D.tcl）说明了怎么样使用vtkExtractVOI。它提取输入图像的局部区域数据，然后对其重采样。输出数据再传递给vtkContourFilter。（你也许会像去掉vtkExtractVOI再比较结果。）

```
vtkQuadric quadric

quadricSetCoefficients .5 1 .2 0 .1 0 0 .2 0 0

vtkSampleFunction sample

sampleSetSampleDimensions 30 30 30

sampleSetImplicitFunction quadric

sampleComputeNormalsOff



vtkExtractVOI extract

extractSetInputConnection [sample GetOutputPort]

extractSetVOI 0 29 0 29 15 15

extractSetSampleRate 1 2 3
```

vtkContourFilter contours

contoursSetInputConnection [extract GetOutputPort]

contoursGenerateValues 13 0.0 1.2

vtkPolyDataMapper contMapper

contMapperSetInputConnection [contours GetOutputPort]

contMapperSetScalarRange 0.0 1.2

vtkActor contActor

contActorSetMapper contMapper

注意上面代码中通过指定一个感兴趣区域（0,29,0,29,15,15）来提取原始数据中的一个平面，而沿三个轴方向的采样率也是设置为不同的值。通过指定VOI大小，可以从数据中提取一个区域，甚至一条线或者一个点（VOI设置采用0偏移数值）。

6.3基于标量值的变形

图像数据的一个常见应用是存储高程值。这种图像常为称为范围图或者高程图。图像中每个像素的标量值表示一个高程值或者范围值。而可视化中一个常见的目的就是将这种图像显示成为一个精确地三维高程表示。图6-2中显示了一个根据高程值显示的图像。左边图像为原始图像，右边图像则是由原始图像产生的三维曲面。

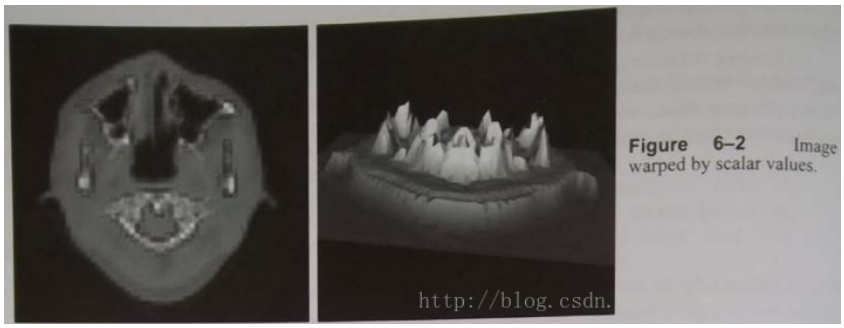

Figure 6–2 Image warped by scalar values.

图6-2 Image warped by scalar values

高程图可视化管线比较简单，但是有一个重要的概念需要理解。原始图像中隐藏了几何和拓扑结构。而图像经Warpping操作后会产生一个三维几何曲面。为了支持该操作，我们首先利用vtkImageDataGeometryFilter将图像转换成vtkPolyData类型，然后执行Warp操作并连接到mapper上。下面脚本中你会注意到我们用到了vtkWindowLevelLookupTable来提供一个灰度颜色查找表，取代默认的红到蓝颜色查找表。

vtkImageReader reader

readerSetDataByteOrderToLittleEndian

readerSetDataExtent 0 63 0 63 40 40

readerSetFilePrefix “$VTK_DATA_ROOT/Data/headsq/quarter”

readerSetDataMask 0x7fff

vtkImageDataGeometryFilter geometry

geometrySetInputConnection [reader GetOutputPort]

```
vtkWarpScalar warp

warpSetInputConnection [geometry GetOutputPort]

warpSetScalarFactor 0.005



vtkWindowLevelLookupTable wl



vtkPolyDataMapper mapper

mapperSetInputConnection [warp GetOutputPort]

mapperSetScalarRange 0 2000

mapperImmediateModeRenderingOff

mapperSetLookupTable wl



vtkActor actor

actorSetMapper mapper
```

这个例子经常会结合其他技术使用。如果你想使用图像的标量值进行Warp，然后使用其他的标量场对其进行着色。另一个常用的操作是对产生的曲面消减多边形个数。因为由图像产生的结果往往含有大量的多边形面片。可以使用vtkDecimatePro来消减面片数量。另外还可以考虑使用vtkTriangleFilter和vtkStripper来将多边形网格转换为三角形网格，这样可以提高渲染速度，减少内存占用。

6.4图像显示

VTK中显示图像有多种方式，本节中将介绍两种常用的方式。体绘制技术用来直接绘制三维图像，其细节将在第7章中介绍。

图像浏览器

vtkImageViewer2类取代了早期版本的vtkImageViewer，可以方便的显示图像。vtkImageViewer2类内部包含了vtkRenderWindow,vtkRenderer, vtkImageActor和vtkImageMapToWindowLevelColors对象，可以方便的在用户应用程序中调用。这个类也根据图像来实例化一个交互器类型（vtkInteractorStypeImage），提供了图像放缩、拉伸和交互式的窗宽/窗位调节（参考43页“交互器类型”和283页“vtkRenderWindow交互器类型”）。vtkImageViewer2（与vtkImageViewer不同）采用的是3D渲染和纹理映射技术将图像绘制到平面上，从而方便了快速渲染，放缩和拉伸。根据特定的图像切片所在的深度坐标，图像显示在三维空间中。每次调用SetSlice()函数都会更改显示的图像切片和对应的三维空间深度。该功能通过InteractorStyle中的AutoAdjustCameraClippingRange选项来控制。你还可以设置方向来显示XY，YZ或者XZ方向切片。

利用图像浏览器来显示三维图像切片的例子可以参考Widgets/Testing/CXX/TestingImageActorContourWidget.cxx。下面代码演示了该类的一个典型使用方法。

```
vtkImageViewer2 *ImageViewer =vtkImageViewer2::New();

ImageViewer->SetInput(shifter->GetOutput());

ImageViewer->SetColorLevel(127);

ImageViewer->SetColorWindow(255);

ImageViewer->SetupInteractor(iren);

ImageViewer->SetSlice(40);

ImageViewer->SetOrientationToXY();

ImageViewer->Render();
```

该类也支持图像和几何图形的混合显示，例如：

```
viewer->SetInput( myImage );

viewer->GetRenderer()->AddActor( myActor );
```

这样可以用一些边来注释图像或者高亮显示部分图像或者同时显示图像的一个切片和等值面等等。所有在当前显示的图像切片前面的几何图形都是可见的，而后面的部分则被遮挡。

Window-level传输函数定义如图6-3。Level值是位于window中央的值。窗宽（window）则是用来映射显示的数据范围。传输函数的斜率决定了最终图像的对比度。所有位于window以外的数据都会截取到windows边界值。

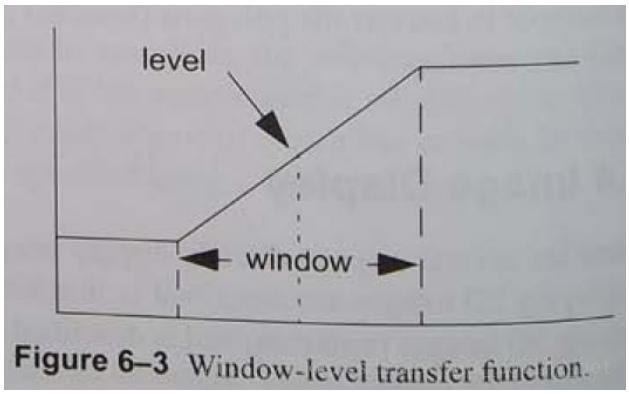

Figure 6–3 Window-level transfer function.

图6-3 窗宽窗位传输函数

图像Actor

当你想在一个窗口中显示图像以及一些简单的2D注释时，可以使用vtkImageViewer类。而当你想在3D渲染窗口中显示图像时，需要使用的是vtkImageActor。创建一个多边形来表示图像边界，然后通过纹理映射将图像复制到多边形上来显示显示图像。在大部分的平台下，在实时进行旋转、拉伸和放缩图像时需要对图像进行双线性插值。如果将其交互器替换为一个vtkInteractorStyleImage类型时，旋转操作就可以禁用，这样三维渲染窗口操作就如同二维图像浏览器。

vtkImageActor是一个包含Actor和Mapper对象的合成类。它的使用非常的简单，如下例所示。

```
vtkBMPReader bmpReader

bmpReader SetFileName “$VTK_DATA_ROOT/Data/masonry.bmp”

vtkImageActor imageActor

imageActor SetInput [ bmpReader GetOutput ]
```

图像Actor可以通过AddProp()函数添加至renderer中。vtkImageActor类期望输入的图像在一个方向上长度为1，而在其他两个方向上扩展。这样如果裁剪操作是沿着X或者Y轴时，就不需要重新组织数据，从而使vtkImageActor通过一个裁剪Filter与一个Volume连接。

vtkImagePlaneWidget

Widget会在255页“Interaction,widgets and Selections”中讲述。这个widget可以交互地放置到图像中，并通过该平面来显示重新生成的图像切片。生成切片数据时采用的插值选项包括最近邻、线性和立方插值。平面的位置和方向可以进行交互控制。当然也可以交互地控制新切片的窗宽窗位，并且可以选择性的显示窗宽窗位和位置注释。

```
vtkImagePlaneWidget* planeWidgetX =vtkImagePlaneWidget::New();

planeWidgetX->SetInteractor(iren);

planeWidgetX->RestrictPlaneToVolumeOn();
```

```
planeWidgetX->SetResliceInterpolateToNearestNeighbour();

planeWidgetX->SetInput(v16->GetOutput());

planeWidgetX->SetPlaneOrientationToXAxes();

planeWidgetX->SetSliceIndex(32);

planeWidgetX->DisplayTextOn();

planeWidgetX->On();
```

第06章-图像处理及可视化（2）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本节对应原书中的第125页至第138页】

6.5图像源

VTK中有些图像处理对象本身并不接收任何的数据对象，但是会有结果产生。他们被称为图像源，这里会介绍其中的一部分。参考444页"源对象"或者查阅Doxygen文档或者更完整的可用图像源列表。

ImageCanvasSource2D

vtkImageCanvasSource2D类根据指定的大小和数量类型生成一个二维的空白图像，并提供了函数完成在空白图像中绘制不同的几何图形。这些几何图形包括方形，线段和圆；同样还提供了一个填充操作函数。下面例子演示了利用该图像源来生成一个512*512像素的图像，并在图像中绘制了几个图形。结果如图6-4所示。

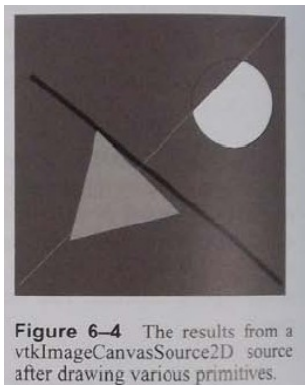

Figure 6–4 The results from a vtkImageCanvasSource2D source after drawing various primitives.

图6-4 用类vtkImageCanvasSource2D

```
#set up the size and type of the image canvas
vtkImageCanvasSource2D imCan
imCan SetScalarTypeToUnsignedChar
imCan SetExtent 0 511 0 511 0 0
#Draw various primitives
imCan SetDrawColor 86
imCan FillBox 0 511 0 511
imCan SetDrawColor 0
imCan FillTube 500 20 30 400 5
imCan SetDrawColor 255
imCan DrawSegment 10 20 500 510
imCan SetDrawColor 0
imCan DrawCircle 400 350 80.0
imCan SetDrawColor 255

imCan FillPixel 450 350
imCan SetDraw 170
imCan FillTriangle 100 100 300 150 150 300

#show the resulting image
vtkImageViewer viewer
viewer SetInputConnection [ imCan GetOutputoirt ]
viewer SetColorWindow 256
viewer SetColorLevel 127.5
```

ImageEllipsoidSource

如果你希望用一个模板函数来写图像，那么vtkImageEllipsoidSource会是一个好的起点。这个对象根据指定的中心点，每个轴的半径，椭圆内部和外部值来生成一个包含椭圆的二值图像。输出的图像数据类型也可以指定，这也是为什么采用模板函数的原因。一些图像filter内部会使用这个图像源，如vtkImageDilateErode3D。
如果你想创建一个vtkImageBoxSource对象，来生成一个二值盒状图。你可以先拷贝一份vtkImageEllipsoidSource源文件和头文件，并在里面搜索和替换。你很可能会改变变量radius为length，因为对于盒状图像源这个命名更有意义。最后，你需要替换掉模板函数vtkImageBoxSourceExecute来创建盒状图像而不是椭圆图像。（参考401页"多线程图像filter"）。

ImageGaussianSource

vtkGaussianSource对象根据指定的中心位置，最大值和标准差来生成一个像素值服从高斯分布的图像。其输出图像的像素类型为浮点型（如double）。

如果你想写一个仅仅生成一种类型图像的图像源，如float，那么这个类就是一个很好的例子。比较vtkImageGaussianSource和vtkImageEllipsoidSource，你会发现vtkImageGaussianSource实现代码是在RequestData()中，而vtkImageEllipsoidSource的RequestData()函数是通过调用了一个模板函数在完成功能。

ImageGridSource

如果你想采用一个二维网格标注图像，vtkImageGridSource来创建一个网格模式图像。如图6-5所示。下面代码演示了怎样将图像网格和CT图像一个切片进行融合。vtkImageReader读入了一个64×64大小的图像。

```
vtkImageGridSource imageGrid

imageGrid SetGridSpacing 16 16 0
imageGrid SetGridOrigin 0 0 0
imageGrid SetDataExtent 0 63 0 63 0 0
imageGrid SetLineValue 4095
imageGrid SetFillValue 0
imageGrid SetDataScalarTypeToShort

vtkImageBlend blend
blend SetOpacity 0 0.5
blend SetOpacity 1 0.5
blend AddInputConnection [reader GetOutputPort]
blend AddInputConnection [imageGrid GetOutputPort]
vtkImageViewer viewer
viewer SetInputConnection [blend GetOutputPort]
viewer SetColorWindow 1000
viewer SetColorLevel 500
viewer Render
```

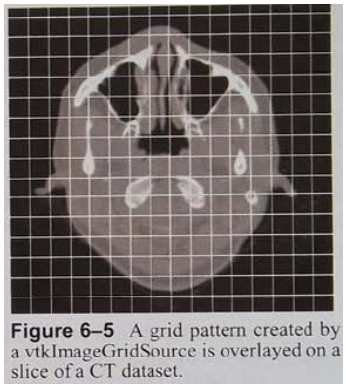

Figure 6–5 A grid pattern created by a vtkImageGridSource is overlayed on a slice of a CT dataset.

图6-5 用类vtkImageGridSource生成的网格覆盖在CT图像上

ImageNoiseSource

vtkImageNoiseSource用来生成一个由随机数填充的图像，随机数大小介于用户指定的最大和最小数之间。输出图像的数据类型是浮点型。

需要注意的是vtkImageNoiseSource每次都生成一个不同的图像。通常对于一个噪声源这是正常的。但是对于流水线中两次的Request会造成重叠区域数据发生变化。例如有这样一个流水线，vtkImageNoiseSource连接到了一个ImageMedianFilter，ImageMedianFilter再连接到一个vtkImageDataStreamer。如果你在streamer中指定一个内存限制使图像分为两半进行计算。第一个Request请求操作的是半幅图像。中值filter需要的图像要比半幅图像略大（由于核函数的范围）。当中值filter第二次执行时，两次需要的图像的重叠区域的值会发生变化，从而两次计算结果中重叠区域会变得不一致。

ImageSinusoidSource

ImageSinusoidSource对象可以用来生成图像，图像的大小由用户指定，而像素值则由sinusoid函数根据指定的方向、周期、相位和幅度来生成。

图6-6中为sinusoid图像源生成的图像转换为unsigned char类型后体绘制的效果图。该结果传递到一个轮廓filter中后来创建了一个图像包围盒。

```
vtkImageSinusoidSource ss
ss SetWholeExtent 0 99 0 99 0 99
ss SetAmplitude 63
ss SetDirection 1 0 0
ss SetPeriod 25
```

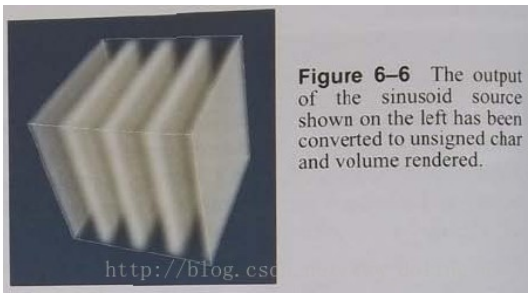

图6-6 由vtkImageSinusoidSource生成的数据转成unsigned char型并体绘制的效果

6.6图像处理

下面我们来看几个图像处理的例子。这里通过介绍部分图像处理Filter来演示怎样使用VTK的图像处理Filter。如果想了解更多相关信息，请参考Doxygen文档内容。另外，450页“图像Filters”章节中会有更完整的介绍。

标量类型转换

图像标量数据类型的转换是一种很常用的操作。例如，一些fitlers只能处理特点数据类型的图像，如float浮点型或者int整型。另外，你可能需要直接利用图像彩色值，而不是通过lookup颜色查找表映射。而这个操作要求图像数据类型必须是unsigned char。

VTK中有两种filter实现图像标量数据类型转换。第一个是vtkImageCast。该Filter允许用户指定输出标量类型。例如当已知一个图像中的像素灰度范围为0-255，而灰度数据的存储类型为无符号整型，此时可以采用vtkImageCast将其转换为unsigned char类型。需要注意该filter的ClampOverflow变量，如果该变量设置为on，那么超出输出数据类型范围的数据就会被截断。如图像中存在一个像素值为257，当ClampOverflow为on事，该像素值在输出图像中为255。如果ClampOverflow为off，那么输出值就为1。

当需要将一个数据范围为[-1，1]范围的浮点型图像转换为unsigned char类型时，vtkImageCast将不能满足要求，此时需要vtkImageShiftScale进行图像转换。该filter可以指定偏移和比例参数来对输入图像数据进行操作。在上例中，需要设置shift值为+1，比例系数设置为127.5。那么输入数据-1则映射为（-1+1）*127.5=0，+1则会映射为（+1+1）*127.5=255。

修改图像间距、原点或范围

VTK中修改图像的间距，原点或者范围常常令用户困惑。一个常采用的方法是获取filter的输出结果，然后将其调整到需要的大小。但是当管线update更新后这些改变又会恢复到原来的值。因此管线中需要一个filter来执行这些修改操作。这就是vtkImageChangeInformation。利用该filter，origin和图像间隔spacing、以及范围的起始点可以显示的指定。由于图像维数没有发生改变，因此范围的起点即决定了图像的范围。另外，vtkImageChangeInformation还定义了一些方法来方便的实现图像居中，平移图像范围，平移原点和缩放图像间隔操作。

下面例子中定义了vtkImageReader读取医学图像数据，将数据传递至一个三维vtkImageGradient中，然后将计算结果以彩色图像显示。

```
vtkImageReader reader
readerSetDataByteOrderToLittleEndian
reader SetDataExtent0 63 0 63 1 93
readerSetFilePrefix “$VTK_DATA_ROOT/Data/headsq/quarter”
readerSetDataMask 0x7fff

vtkImageGradient gradient gradient
gradientSetInputConnection [reader GetOutputPort]
gradientSetDimensionality 3

vtkImageViewer viewer
viewerSetInputConnection [gradient GetOutputPort]
viewerSetZSlice 22
viewerSetColorWindow 400
viewerSetColorLevel 0
```

图像合并

VTK中支持合并图像空间和合并图像成分。利用vtkImageAppend在空间上合并图像可以生成一个更大的图像，而vtkImageAppendComponents则可以将单独的单成分图像合并为一个RGB彩色图像。

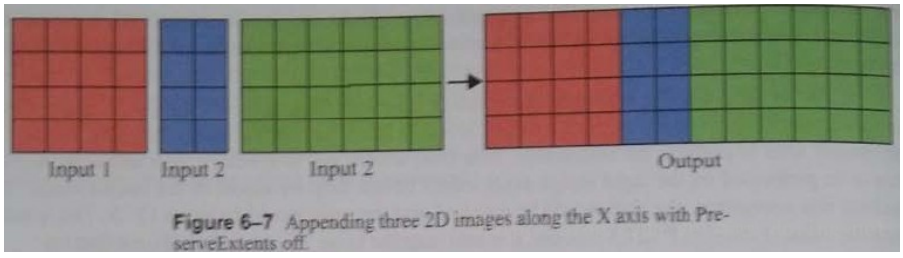

图6-7 沿着X方向将三个2D图像合并在一起

注意图像可以是一维、二维或者三维。当在空间上进行图像合并时，输出图像的维数可能会增加。例如，可以将一组一维图像组成一个二维图像，也可以将一组二维图像合并为一个三维图像。vtkImageAppend合并图像时，有两种策略。如果PreserveExtents变量关闭，那么就沿着AppendAxis变量指定的轴进行合并；所以图像必须要有相同的维数，相同的数据类型以及标量成分数目。输出图像的原点和间隔与第一幅图像相同。图6-7说明了将3个二维图像沿着X轴进行合并的过程。而图6-8中演示了将一组二维图像沿着Z轴合并

成一个三维图像的过程。

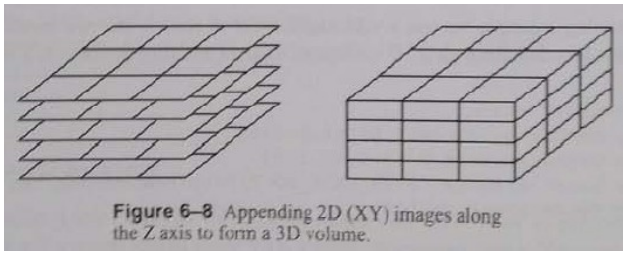

Figure 6–8 Appending 2D (XY) images along the Z axis to form a 3D volume.

图6-8 沿着Z轴方向将多个2D图像合并成3D图像

如同PreserveExtents变量设置为on，那么vtkImageAppend则会生成一个包含所有输入图像的图像，该图像的范围为所有图像范围的并集。输出图像的原点和间隔与输入的第一个图像相同，并且像素值初始化为0。然后每一个图像都拷贝到输出图像中。当两个输入图像存在覆盖的像素时，不会执行混合操作，而是根据图像的输入顺序决定该像素的数值大小，即顺序靠后面图像对应的像素大小。图6-9说明了PreserveExtents状态为on时图像合并的过程。

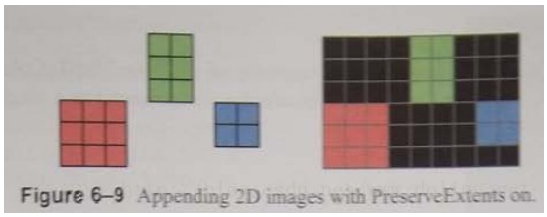

Figure 6–9 Appending 2D images with PreserveExtents on.

图6-9 PreserveExtents变量的状态为On时的2D图像合并效果

注意vtkImageAppend采用的是像素坐标而非世界坐标，所有输入图像如同具有相同的原点和间隔，因此输入图像之间的相对位置仅仅由图像的范围extent定义。

vtkImageAppendComponents可以将多个具有相同数据类型和维数的图像合并到一个多成分图像。输出图像的原点和间隔与输入的第一个图像一致，其成分数据与输入图像的个数相等。该fliter常用来将单成分的红、绿、蓝色图像合并为i一个RGB彩色图像。如图6-10。

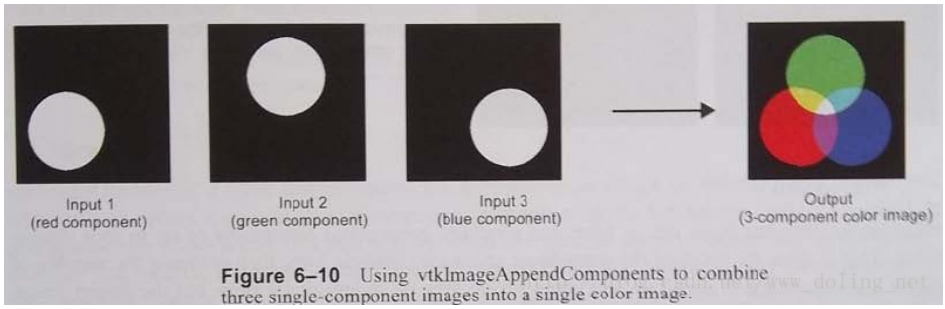

Figure 6–10 Using vtkImageAppendComponents to combine three single-component images into a single color image.

图6-10 用类vtkImageAppendComponets将单通道的图像组合成一个彩色图像

图像彩色映射

vtkImageMapToColors将灰度图像转换为彩色图像。如图6-11。该filter接收任意数据类型的图像作为输入，将用户选定的像素分量（通过vtkImageMapToColors的函数SetActiveComponent()）通过vtkScalarsToColors类实例进行映射彩色值，并存储至输出图像中。vtkImageMapToColors的子类vtkImageMapToWindowLevelColors在存储映射后的彩色值前，对彩色值通过一个窗宽-窗位函数规整。两个filter的输出图像的类型均为unsigned
char。

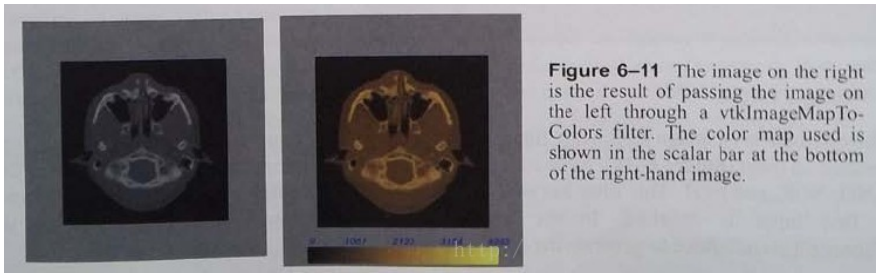

Figure 6–11 The image on the right is the result of passing the image on the left through a vtkImageMapToColors filter. The color map used is shown in the scalar bar at the bottom of the right-hand image.

图6-11 左图是vtkImageMapToColors类的输入，右图是其输出，右图下方是颜色映射的标量条

图像灰度映射

vtkImageLuminance执行与vtkImageMapToColors相反的操作，将一个RGB彩色图像转换为一个单成分的灰度图像。映射公式如下：

luminance = 0.3*R + 0.59*G + 0.11*B

该公式中，R为输入图像的第一分量（红色），G为第二分量（绿色），B为第三分量（蓝色）。这个计算结果计算一个RGB颜色的亮度。

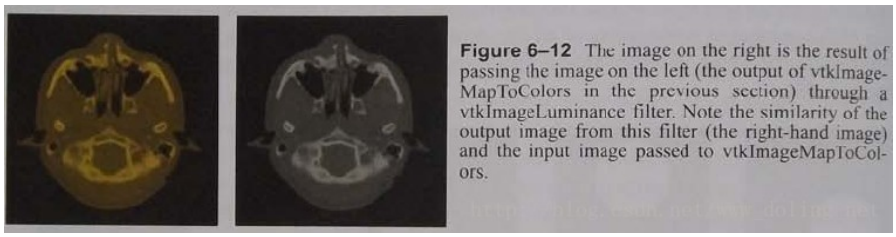

Figure 6–12 The image on the right is the result of passing the image on the left (the output of vtkImageMapToColors in the previous section) through a vtkImageLuminance filter. Note the similarity of the output image from this filter (the right-hand image) and the input image passed to vtkImageMapToColors.

图6-12 左图是vtkImageLuminance的输入，右图是其输出。要注意区别类vtkImageMapToColors的输入与类vtkImageLumiance的输出的相似性

直方图

vtkImageAccumulate计算一个图像的直方图，其维数最高可至四维。其计算原理是将图像的每个颜色分量空间划分为离散的区间，然后统计落入每个区域的像素个数。输入图像可以是任意的像素类型，但是输出结果通常为整型。如果一个图像只有一个颜色分量，那么其直方图就是一维的。如图6-13所示。（示例取自VTK/Examples/ImageProcessing/Tcl/Histogram.tcl）

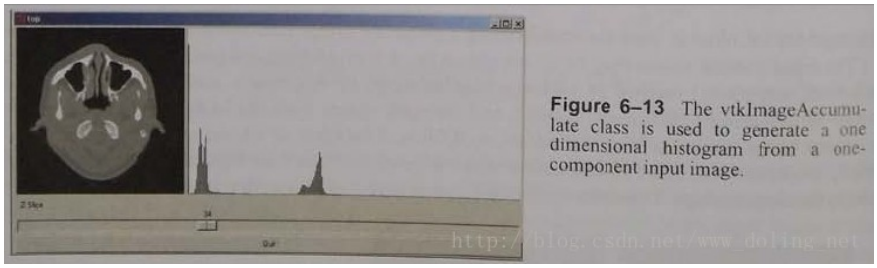

Figure 6–13 The vtkImageAccumulate class is used to generate a one dimensional histogram from a one-component input image.

图6-13 vtkImageAccumulate类主要应用是从单个通道的输入图像生成一维的直方图

图像逻辑运算

vtkImageLogic接收一个或者两个图像进行布尔逻辑运算。如图6-14。该Filter支持大部分的逻辑操作，包括AND，OR，XOR，NAND，NOR和NOT。它有两个输入，尽管一元操作只需要第一个输入。下面例子采用vtkImageEllipsoidSource来产生两个输入图像。

```
vtkImageEllipsoidSource sphere1

sphere1SetCenter 95 100 0
shpere1 SetRadius70 70 70

vtkImageEllipsoidSource sphere2
sphere2SetCenter 161 100 0
sphere2SetRadius 70 70 70

vtkImageLogic xor
xorSetInputConnection 0 \
[sphere1 GetOutputPort]
xorSetInputConnection 1 [sphere2 \
GetOutputPort]
xorSetOutputTrueValue 150
xorSetOperationToXor

vtkImageViewer viewer
viewerSetInput [xor GetOutput]
viewerSetColorWindow 255
viewerSetColorLevel 127.5
```

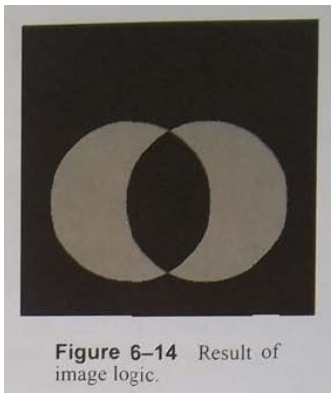

Figure 6–14 Result of image logic.

图6-14 图像逻辑运算的结果

梯度计算

vtkImageGradient计算图像梯度。通过SetDimensionality()函数设置要计算梯度的维数（二维或者三维）。根据设置的维数，该filter输出图像的每个像素都会对应有两个或者三个标量成分，分别对应于梯度向量的每个成分。如果仅仅计算梯度模值的话，可以使用vtkImageGradientMagnitude。
vtkImageGradient采用中间差分法计算梯度。当计算一个像素梯度时，需要利用该像素的左右相邻像素。由于在边界处像素会缺少其中一个像素，因此需要进行边界处理。改处理通过HandleBoundaries变量来控制。当该变量设置为on时，vtkImageGradient针对边界像素采用一种改进的计算方法。当该变量为off时，vtkImageGradient会忽略掉边界，因而输出图像要小于输入图像。

高斯平滑

基于高斯核的图像平滑类似于梯度计算。它可以控制用于卷积的高斯核的维数。vtkGaussianSmooth通过SetStandardDeviations()和SetRadiusFactors()方法控制高斯核的形状和截断半径大小。下面例子跟梯度计算比较类似。开始vtkImageReader连接到vtkImageGaussianSmooth，接下来再连接到vtkImageViewer。

```
vtkImageReader reader
readerSetDataByteOrderToLittleEndian
readerSetDataExtent 0 63 0 63 1 93
readerSetFilePrefix “$VTK_DATA_ROOT/Data/headsq/quarter”
readerSetDataMask 0x7fff

vtkImageGaussianSmooth smooth
smoothSetInputConnection [reader GetOutputPort]
smoothSetDimensionality 2
smoothSetStandardDeviations 2 10

vtkImageViewer2 viewer
viewerSetInputConnection [smooth GetOutputPort]
viewerSetSlice 22
viewerSetColorWindow 2000
viewerSetColorLevel 1000
```

图像翻转

vtkImageFlip实现图像的翻转，翻转轴由FilteredAxis变量决定。默认情况下，FlipAboutOrigin变量设置为0，此时该filter沿着FilteredAxis定义的轴向做关于图像中心的翻转（默认下为0，即X轴），输出图像的原点、间距和范围与输入图像一致。然而如果图像有自己的坐标系统，当需要将图像正的坐标值变换为负坐标值时，应该将图像做关于（0,0,0）的翻转，而不再是图像中心。如果FlipAboutOrigin变量设置为1，那么该filter就以（0,0,0）做翻转。图6-15中左边图像为输入图像，中间图像是FlipAboutOrigin为0时对输入图像沿着Y轴翻转的结果；最右边图像是FlipAboutOrigin为1时的翻转结果。

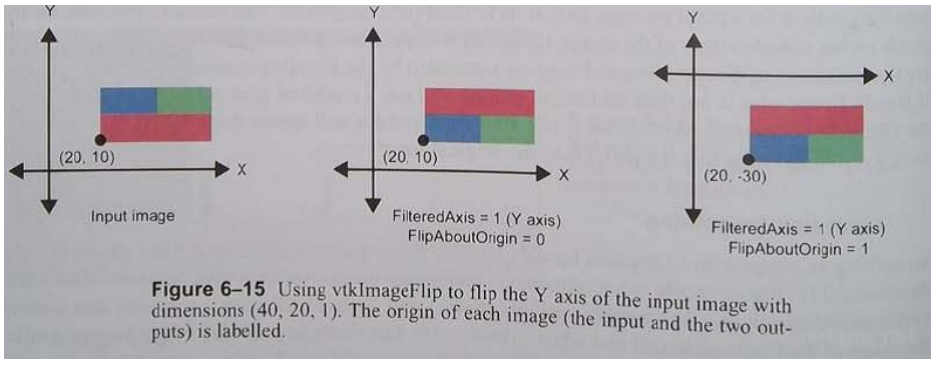

Figure 6–15 Using vtkImageFlip to flip the Y axis of the input image with dimensions (40, 20, 1). The origin of each image (the input and the two outputs) is labelled.

图6-15 使用类vtkImageFlip翻转图像

图像排列

vtkImagePermute可以实现输入图像的坐标值重排。如图6-16。通过FilteredAxes变量值来决定输入图像的哪个轴分别对应到输出图像的X轴、Y轴和Z轴。

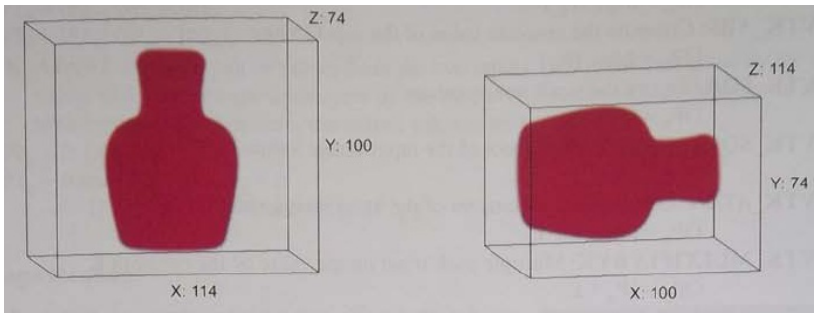

Figure 6–16 Using vtkImagePermute to reorder the axes of a volume dataset. The dimensions of the input volume, shown on the left, are (114, 100, 74). The FilteredAxes instance variable was set to (1, 2, 0), indicating that the Y axis be relabelled as X, Z be relabelled as Y, and X be relabelled as Z. As shown on the right, the dimensions of the output volume are (100, 74, 114).

图6-16 使用类vtkImagePermute重新排列体数据集的坐标轴。左图为输入的数据集，大小为(114,100,74)。FilteredAxes变量的值设置为(1,2,0)，表示Y轴重置成X轴，Z轴重置成Y，X轴重置成Z轴。输出结果如图所示，其大小了(100,74,114)

图像计算

vtkImageMathematics提供了基本的一元和二元数学操作。根据不同的操作，需要一个或者两个输入图像。当需要两个输入图像时，这两个图像必须有相同的像素数据类型，颜色分量，但是可以不是不同的图像范围。计算输出图像的范围为两个输入图像范围的并集。原点和间隔与第一个输入图像保持一致。

下面介绍下一元操作。注意IPn 为输入像素的第n个颜色分量值，OPn为输出像素第n个颜色分量值，C和K为常量。DivideByZeroToC变量用于处理除数为0的情况。如果该变量为On，那么当除数为0时计算结果为C。如果为off，那么计算结果将为当前标量类型所表示数值范围的最大值。

l **VTK_INVERT**: 图像取反。根据DivideByZeroToC的值来决定当除数为0时的计算结果。

如果IPn != 0，OPn = 1.0/ IPn

如果IPn == 0并且DivideByZeroToC为on，OPn = C

如果IPn == 0并且DivideByZeroToC为off，OPn = 数据类型范围的最大值

l **VTK_SIN**:对输入图像做正弦计算。

OPn = sin(IPn)

l **VTK_COS**: 对输入图像做余弦计算。

OPn = cos(IPn)

l **VTK_EXP**: 对输入图像做指数运算。底数为e，约等于2.71828。

OPn = exp(IPn)

l **VTK_LOG**: 计算图像的自然对数。

OPn = log(IPn)

l **VTK_ABS**: 对输入图像取绝对值。

OPn = fabs(IPn)

l **VTK_SQR**: 对输入图像计算平方。

OPn = IPn* IPn

l **VTK_SQRT**: 计算输入图像的平方根。

OPn = sqrt(IPn)

l **VTK_ATAN**: 对输入图像做反正切运算。

OPn = atan(IPn)

l VTK_MULTIPLYBYK：图像的每个像素值乘以常数K。

OPn = IPn*K

l VTK_ADDC: 图像每个像素值都加上常量C。

OPn = IPn+C

l VTK_REPLACECBYK: 将图像中所有等于C的像素值替换为K。

如果IPn=C，OPn = K

如果IPn!=C，OPn = IPn

l VTK_CONJUGATE: 该操作要求输入图像有两个标量数据。将两个标量值表示为共轭复数。

OP0 = IP0

OP1 = -IP1

图像重切（ImageReslice）

vtkImageReslice能够沿着任意方向对图像冲采样。输出图像的范围，原点和采样密度都可以进行设置。另外它还能实现其他的功能：图像排列，翻转，旋转，放缩，冲采样，图像填补及其组合功能。而图像斜切功能则是其他filters所不能实现的。下面代码演示了怎么使用vtkImageReslice。

```
vtkBMPReader reader
reader SetFileName“$VTK_DATA_ROOT/Data/masonry.bmp”
readerSetDataExtent 0 255 0255 0 0
readerSetDataSpacing 1 1 1
readerSetDataOrigin 0 0 0
readerUpdateWholeExtent

vtkTransform transform
transformRotateZ 45
transformScale 1.414 1.414 1.414
```

```
vtkImageReslice reslice
resliceSetInputConnection [reader GetOutputPort]
resliceSetResliceTransform transform
resliceSetInterpolationModeToCubic
resliceWrapOn
resliceAutoCropOutputOn

vtkImageViewer2 viewer
viewerSetpInputConnection [reslice \
GetOutputPort]
viewerSetSlice 0

viewerSetColorWindow 256.0
viewerSetColorLevel 127.5
viewerRender
```

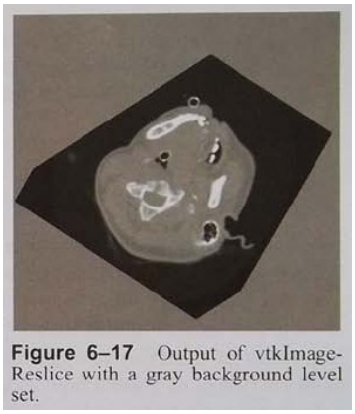

Figure 6–17 Output of vtkImage-Reslice with a gray background level set.

图6-17 vtkImageReslice的输出

例子中读入一个64*64*93大小的图像。定义一个transform来指定图像进行重切的位置，并且设置插值方式为三次插值。Wrap-pad为打开状态，而设置AutoCropOutput使得输出图像的范围足够大以至于不会被裁剪掉。默认情况下，输出图像的间距为1，原点和范围则会自动调整以包含输入图像。而viewer对象则用来沿着z轴方向显示计算结果。

迭代遍历图像

VTK中提供了迭代器来方便的访问、查找和设置图像像素值。vtkImageIterator实现该功能。该类是一个模板类，模板参数为图像的数据类型，而构造函数的参数为要迭代的图像范围。

```
int subRegion[6] = {10, 20, 10, 20, 10, 20};
vtkImageIterator< unsigned char > it(imagesubRegion);
while( !it.IsAtEnd())
{

unsigned char *inSI = it.BeginSpan();
unsignedchar *inSIEnd = it.EndSpan();
while( inSI!= inSIEnd )
{
*inSI = (255-*inSI);
++ inSI;
}
it.NextSpan();
}
```

【第6章 翻译完成】

第07章-体绘制（1）

【译者：这个系列教程是以Kitware公司出版的《VTK User's Guide -11th edition》一书作的中文翻译（出版时间2010年，ISBN: 978-1-930934-23-8），由于时间关系，我们不能保证每周都能更新本书内容，但尽量做到一周更新一篇到两篇内容。敬请期待^_^。欢迎转载，另请转载时注明本文出处，谢谢合作！同时，由于译者水平有限，出错之处在所难免，欢迎指出订正！】

【本节对应原书中的第139页至第142页】

第7章体绘制

体绘制是一种三维空间中而非三维空间中的二维曲面上绘制三维数据的技术。体绘制和几何绘制并没有严格的界限。两种技术经常会有相似的结果，有时两者又可认为是一种技术。例如，利用轮廓提取技术从图形数据中提取等值面，然后利用几何渲染技术来绘制等值面，也可以使用光线投射技术，在到达某个值时结束光线追踪来绘制等值面。这两种方法产生相似的效果。再比如，基于纹理映射的混合体绘制。该方法的适用数据是图像，因此可以看做是体绘制，也可以看做是几何绘制，因为它使用几何图元和标准的图形硬件。

为了根据渲染数据的属性进行定制，在VTK中两种渲染技术进行了区分。到目前为止看到的例子中，数据渲染都会用到vtkActor，vtkProperty以及vtkMapper的子类。vtkActor中保存了位置，方向以及缩放等信息，以及Property和mapper指针。vtkProperty中存储了数据渲染时的表面属性，例如环境光参数，阴影类型等。vtkMapper则负责数据渲染。对于体绘制，可以使用的类比较多。vtkVolume用来替代vtkActor表示空间对象。类似于vtkActor，vtkVolume中存储了数据的位置，方向和缩放参数。但是，其内部还有vtkVolumeProperty和vktAbstractVolumeMapper的引用。vtkVolumeProperty中存储了影像数据实际显示效果的参数，这些参数不同与几何绘制的参数。vktAbstractVolumeMapper负责数据体绘制和输入数据的合法性检查。

VTK中对于矩形网格数据（vtkImageData）和非规则网格（vtkUnstructuredGrid）都可以进行体绘制。根据具体vktAbstractVolumeMapper子类的SetInput()函数来设置相应的数据指针（vtkImageData或者vtkUnstructuredGrid）。注意，可以将非规则数据进行重采样为规则数据进行体绘制。（100页"Probing"）。另外也可以通过四面体化技术来产生非规则网格来进行体绘制。

对于每种支持的数据类型，都有多种不同的体绘制技术可以使用，接下来会进行具体的分析。然后介绍这些技术中都用到的一些对象和参数，详细分析每种技术细节。最后再讨论一下每个方法的效率问题。

7.1 体绘制支持的数据类型发展历程

VTK最开始仅仅支持的是基于vtkImageData体的绘制方法。vtkVolumeMapper类为这些体绘制方法定义了所有的API函数。后来，基于vtkUnstructuredGrid数据的体绘制方法加入进来。为了保持向前兼容，引入了一个抽象基类作为所有的体绘制方法类的父类。类似地，为vtkVolumeMapper（该类渲染vtkImageData）和vtkUnstructuredGridVolumeMapper（该类负责渲染vtkUnstructuredGrid）引入一个父类vtkAbstractVolumeMapper。

7.2 一个简单的例子

图7-1中展示了一个简单的体绘制效果（参考VTK/Examples/VolumeRendering/Tcl/SimpleRayCast.tcl）。该例子中对vtkImageData采用光线投射方法进行体绘制，代码中黑体部分是体绘制的相关代码。从该例可以看出体绘制相关代码可以使用的其他的mappers进行替换来实现体绘制，主要包括针对vtkImageData的纹理映射方法，针对vtkUnstructuredGrid数据的基于投影的体绘制方法。在当前例子中只需要该很小的改动，因为大多数的功能函数都在基类中定义，因此对于所有的体绘制方法子类是共有的。


Figure 7–1 Volume rendering.

图7-1 体绘制

```
# Create the reader for the data

vtkStructuredPointsReader reader

reader SetFileName"$VTK_DATA_ROOT/Data/ironProt.vtk"



# Create transfer mapping scalar value to opacity

vtkPiecewiseFunction opacityTransferFunction
```

```
opacityTransferFunction AddPoint 20 0.0

opacityTransferFunction AddPoint 255 0.2


# Create transfer mapping scalar value to color

vtkColorTransferFunction colorTransferFunction

colorTransferFunction AddRGBPoint 0.0 0.0 0.0 0.0

colorTransferFunctionAddRGBPoint 64.0 1.0 0.0 0.0

colorTransferFunction AddRGBPoint 128.0 0.0 0.0 1.0

colorTransferFunction AddRGBPoint 192.0 0.0 1.0 0.0

colorTransferFunction AddRGBPoint 255.0 0.0 0.2 0.0


# The property describes how the data will look

vtkVolumeProperty volumeProperty

volumeProperty SetColor colorTransferFunction

volumeProperty SetScalarOpacity opacityTransferFunction

volumeProperty ShadeOn

volumeProperty SetInterpolationTypeToLinear


# The mapper / ray cast function know how to renderthe data

vtkVolumeRayCastCompositeFunction compositeFunction

vtkVolumeRayCastMapper volumeMapper

volumeMapper SetVolumeRayCastFunction compositeFunction

volumeMapper SetInputConnection [readerGetOutputPort]


# The volume holds the mapper and the property and

# can be used to position/orient the volume

vtkVolume volume

volumeSetMapper volumeMapper

volume SetProperty volumeProperty


ren1 AddVolume volume

renWin Render
```

首先从硬盘上读取一个图像，接着为vtkVolumeProperty定义两个函数，分别负责将像素映射为不透明度和颜色。然后，定义一个专用于光线投射体绘制方法的vtkVolumeRayCastCompositeFunction对象。该对象负责合成投射光线上的采样点数据。另外还定义一个vtkVolumeRayCastMapper对象执行基本的光线投射操作，如空间变换和裁剪等。将读入的图像作为mappper对象的输入数据，并创建一个vtkVolume（该类是vtkProp3D的子类，与vtkActor功能类似）对象来粗存储mapper和property对象。最后，将vtkVolume对象添加

至renderer中实现场景渲染。

如果使用二维纹理映射方法来替代光线投射方法，那么黑体部分代码可以替换为：

vtkVolumeTextureMapper2D volumeMapper

volumeMapper SetInputConnection [readerGetOutputPort]

如果显卡支持三维纹理映射的话，那么上面代码还可以采用三维纹理映射替换如下：

vtkVolumeTextureMapper3D volumeMapper

volumeMapper SetInputConnection [readerGetOutputPort]

vtkFixedPointRayCastMapper也可以用来替换vtkVolumeRayCastMapper，并且在多数情况下推荐使用该mapper。vtkFixedPointRayCastMapper将所有数据类型都看做为多元数据，并使用定点计算和空间跳跃技术来实现高效计算。然后由于其混合操作采用的是硬编码，因此难于定制新的光线投射算法，可扩展性较差。上例中用vtkFixedPointRayCastMapper替换如下：

vtkFixedPointRayCastMapper volumeMapper

volumeMapper SetInputConnection [readerGetOutputPort]

如果使用非规则数据的体绘制方法进行替换的话，那么替换的代码就会稍微复杂一些，因为在设置mapper输入前，需要先将vtkImageData数据转换为vtkUnstructuredGrid类型数据。下面代码中使用了非规则网格体绘制技术，通过显卡将四面体网格数据进行投影。

#convert data to unstructured grid

vtkDataSetTriangleFilter tetraFilter

tetraFilter SetInputConnection [readerGetOutputPort]

#creates the objects specific to the projectedtetrahedral method

vtkProjectedTetrahedraMapper volumeMapper

volumeMapper SetInputConnection [tetraFilter GetOutputPort]

需要注意的是，不推荐将vtkImageData转换为vtkUnstructuredGrid数据。因为针对vtkImageData的mappers无论在时间效率上还是渲染效果上都要优于针对vtkUnstructuredGrid的mappers。