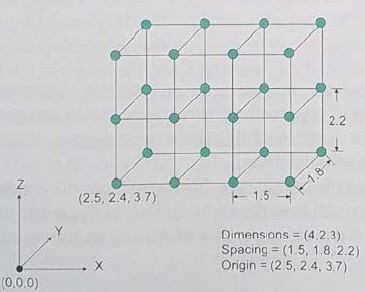
**图像处理&可视化**

图像数据，如图6-1所示，在拓扑上和几何上都是规则的，在VTK中用vtkImageData表示。规则数据意味着通过很少的参数如原点，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同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。

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Filters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filters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mappers和专门的actors来渲染vtkImageData，从简单的二维图像渲染到体绘制。

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。

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五章中介绍。VtkImageData和vtkUnstructuredGrid数据的体绘制技术将在第七章中讲解。

**6.1 人工创建vtkImageData**

直接创建图像数据非常直接，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 Filters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。

//Create the image data

vtkImageData \*id = vtkImageData::New();

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

id->SetScalarTypeToUnsignedShort();

id->SetNumberOfScalarComponents(1);

id->AllocateScalars();

//Fill in scalar values

Unsigned short \*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——图像管线要与图形管线分开，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filters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

quadric SetCoefficients .5 1 .2 0 .1 0 0 .2 0 0

vtkSampleFunction sample

sample SetSampleDimensions 30 30 30

sample SetImplicitFunction quadric

sample ComputeNormalsOff

vtkExtractVOI extract

extract SetInputConnection [sample GetOutputPort]

extract SetVOI 0 29 0 29 15 15

extract SetSampleRate 1 2 3

vtkContourFilter contours

contours SetInputConnection [extract GetOutputPort]

contours GenerateValues 13 0.0 1.2

vtkPolyDataMapper contMapper

contMapper SetInputConnection [contours GetOutputPort]

contMapper SetScalarRange 0.0 1.2

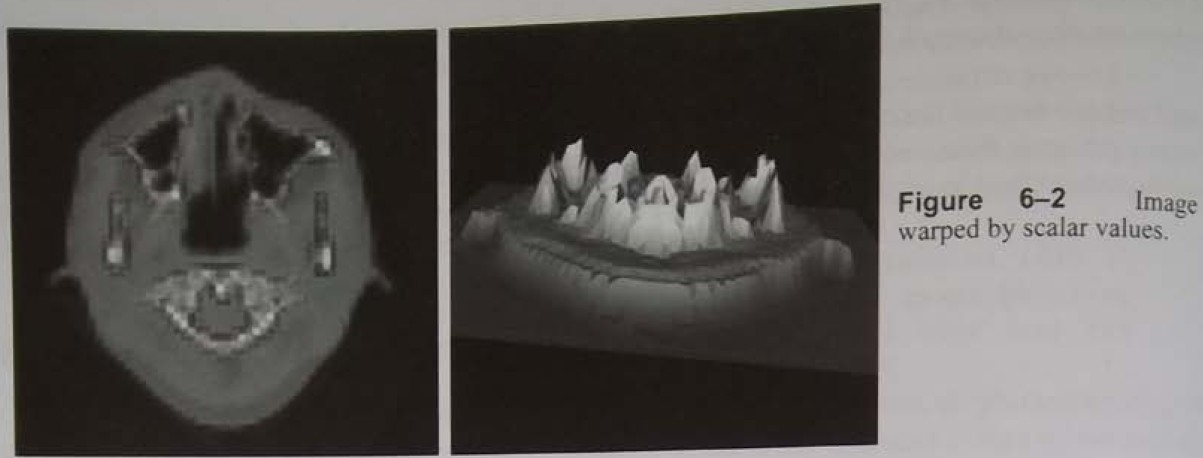
vtkActor contActor

contActor SetMapper 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 Warp Based On Scalar Values基于标量值的Warp？？？？**

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。



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

reader SetDataByteOrderToLittleEndian

reader SetDataExtent 0 63 0 63 40 40

reader SetFilePrefix “$VTK\_DATA\_ROOT/Data/headsq/quarter”

reader SetDataMask 0x7fff

vtkImageDataGeometryFilter geometry

geometry SetInputConnection [reader GetOutputPort]

vtkWarpScalar warp

warp SetInputConnection [geometry GetOutputPort]

warp SetScalarFactor 0.005

vtkWindowLevelLookupTable wl

vtkPolyDataMapper mapper

mapper SetInputConnection [warp GetOutputPort]

mapper SetScalarRange 0 2000

mapper ImmediateModeRenderingOff

mapper SetLookupTable wl

vtkActor actor

actor SetMapper 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。

**Image Viewer 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();

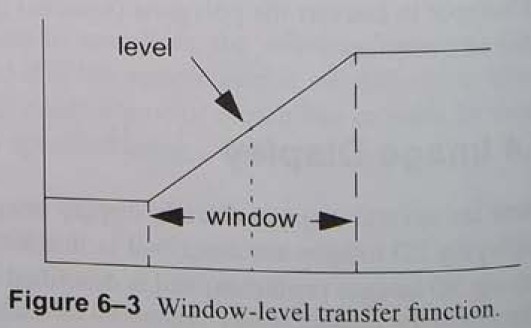
ImageViewer->Render();

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。

**Image Actor 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();

**6.5 Image Sources 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。

#set up the size and type of the image canvas

vtkImageCanvasSource2D imCan

imCan SetScalarTypeToUnsignedChar

imCan SetExtent 0 511 0 511 0 0

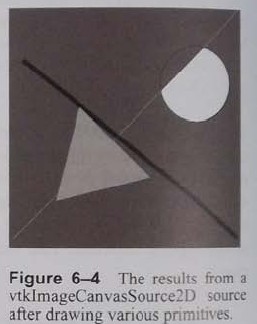
#Draw various primitives

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”）。

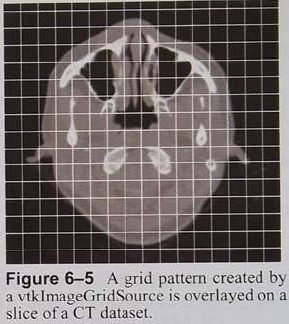
**ImageGaussianSource**

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64X64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

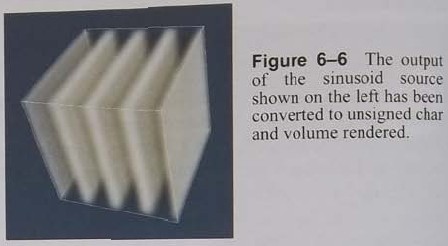
viewer SetColorLevel 500

viewer Render

**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

**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，原点或者范围**

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

reader SetDataByteOrderToLittleEndian

reader SetDataExtent 0 63 0 63 1 93

reader SetFilePrefix “$VTK\_DATA\_ROOT/Data/headsq/quarter”

reader SetDataMask 0x7fff

vtkImageGradient gradient gradient

gradient SetInputConnection [reader GetOutputPort]

gradient SetDimensionality 3

vtkImageViewer viewer

viewer SetInputConnection [gradient GetOutputPort]

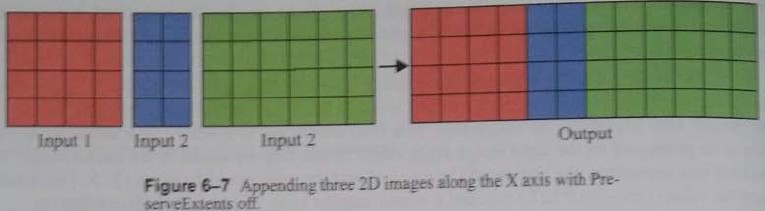
viewer SetZSlice 22

viewer SetColorWindow 400

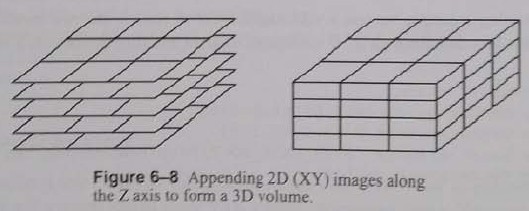
viewer SetColorLevel 0

**图像合并**

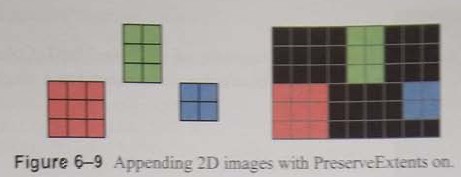
VTK中支持合并图像空间和合并图像成分。利用vtkImageAppend在空间上合并图像可以生成一个更大的图像，而vtkImageAppendComponents则可以将单独的单成分图像合并为一个RGB彩色图像。



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。

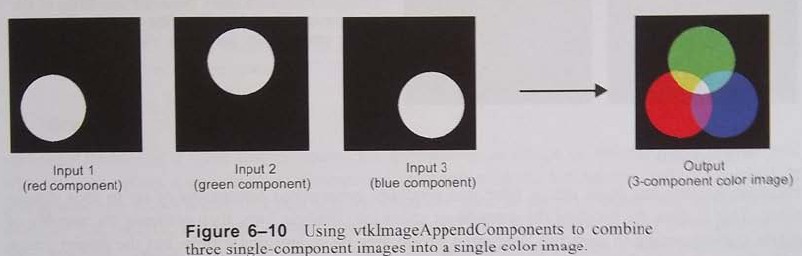


如同PreserveExtents变量设置为on，那么vtkImageAppend则会生成一个包含所有输入图像的图像，该图像的范围为所有图像范围的并集。输出图像的原点和间隔与输入的第一个图像相同，并且像素值初始化为0。然后每一个图像都拷贝到输出图像中。当两个输入图像存在覆盖的像素时，不会执行混合操作，而是根据图像的输入顺序决定该像素的数值大小，即顺序靠后面图像对应的像素大小。图6-9说明了PreserveExtents状态为on时图像合并的过程。



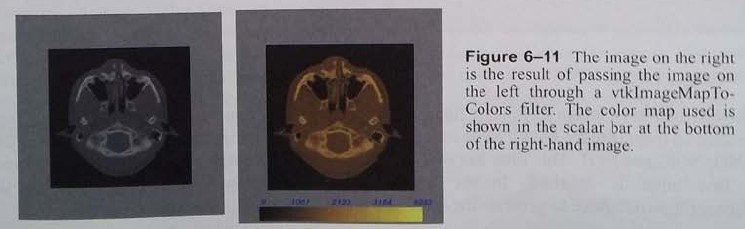
注意vtkImageAppend采用的是像素坐标而非世界坐标，所有输入图像如同具有相同的原点和间隔，因此输入图像之间的相对位置仅仅由图像的范围extent定义。

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。



**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SetActiveCompnent()）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。

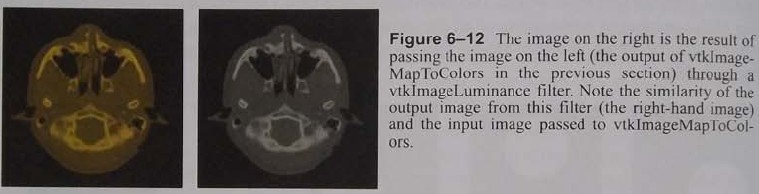


**图像灰度映射**

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：

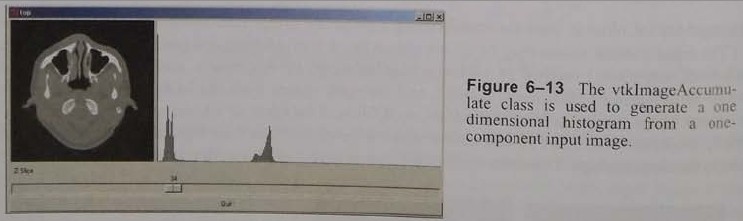
luminance = 0.3\*R + 0.59\*G + 0.11\*B

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。



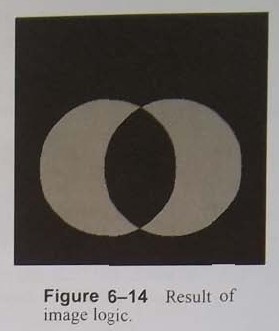
**直方图**

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）



**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

sphere1 SetCenter 95 100 0

shpere1 SetRadius 70 70 70

vtkImageEllipsoidSource sphere2

sphere2 SetCenter 161 100 0

sphere2 SetRadius 70 70 70

vtkImageLogic xor

xor SetInputConnection 0 \

[sphere1 GetOutputPort]

xor SetInputConnection 1 [sphere2 \

GetOutputPort]

xor SetOutputTrueValue 150

xor SetOperationToXor

vtkImageViewer viewer

viewer SetInput [xor GetOutput]

viewer SetColorWindow 255

viewer SetColorLevel 127.5

**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

reader SetDataByteOrderToLittleEndian

reader SetDataExtent 0 63 0 63 1 93

reader SetFilePrefix “$VTK\_DATA\_ROOT/Data/headsq/quarter”

reader SetDataMask 0x7fff

vtkImageGaussianSmooth smooth

smooth SetInputConnection [reader GetOutputPort]

smooth SetDimensionality 2

smooth SetStandardDeviations 2 10

vtkImageViewer2 viewer

viewer SetInputConnection [smooth GetOutputPort]

viewer SetSlice 22

viewer SetColorWindow 2000

viewer SetColorLevel 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。



**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。



**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。

**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

**VTK\_SIN:**对输入图像做正弦计算。

OPn = sin(IPn)

**VTK\_COS:** 对输入图像做余弦计算。

OPn = cos(IPn)

**VTK\_EXP:** 对输入图像做指数运算。底数为e，约等于2.71828。

OPn = exp(IPn)

**VTK\_LOG:** 计算图像的自然对数。

OPn = log(IPn)

**VTK\_ABS:** 对输入图像取绝对值。

OPn = fabs(IPn)

**VTK\_SQR:** 对输入图像计算平方。

OPn = IPn\* IPn

**VTK\_SQRT:** 计算输入图像的平方根。

OPn = sqrt(IPn)

**VTK\_ATAN:** 对输入图像做反正切运算。

OPn = atan(IPn)

**VTK\_MULTIPLYBYK：**图像的每个像素值乘以常数K。

OPn = IPn\*K

**VTK\_ADDC:** 图像每个像素值都加上常量C。

OPn = IPn+C

**VTK\_REPLACECBYK:** 将图像中所有等于C的像素值替换为K。

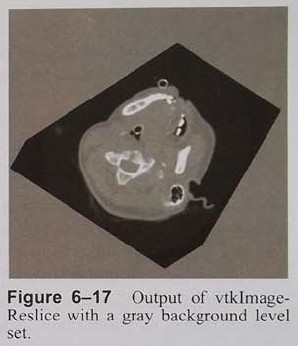
如果IPn=C，OPn = K

如果IPn!=C，OPn = IPn

**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 Image Reslice**

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”

reader SetDataExtent 0 255 0255 0 0

reader SetDataSpacing 1 1 1

reader SetDataOrigin 0 0 0

reader UpdateWholeExtent

vtkTransform transform

transform RotateZ 45

transform Scale 1.414 1.414 1.414

vtkImageReslice reslice

reslice SetInputConnection [reader GetOutputPort]

reslice SetResliceTransform transform

reslice SetInterpolationModeToCubic

reslice WrapOn

reslice AutoCropOutputOn

vtkImageViewer2 viewer

viewer SetpInputConnection [reslice \

GetOutputPort]

viewer SetSlice 0

viewer SetColorWindow 256.0

viewer SetColorLevel 127.5

viewer Render

图6-17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(image subRegion);

while( !it.IsAtEnd())

{  
 unsigned char \*inSI = it.BeginSpan();

unsigned char \*inSIEnd = it.EndSpan();

while( inSI != inSIEnd )

{

\*inSI = (255-\*inSI);

++ inSI;

}

it.NextSpan();

}