**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。

# 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

colorTransferFunction AddRGBPoint 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 render the data

vtkVolumeRayCastCompositeFunction compositeFunction

vtkVolumeRayCastMapper volumeMapper

volumeMapper SetVolumeRayCastFunction compositeFunction

volumeMapper SetInputConnection [reader GetOutputPort]

# The volume holds the mapper and the property and

# can be used to position/orient the volume

vtkVolume volume

volume SetMapper 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 [ reader GetOutputPort ]

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 [ reader GetOutputPort ]

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 [ reader GetOutputPort ]

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 [ reader GetOutputPort ]

#creates the objects specific to the projected tetrahedral 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。

7.3 为什么需要多种体绘制技术？

从上面简单的例子中可以看出，不同渲染方法的代码的主要区别在于实例化的mappper和每个渲染此方法特定的参数，例如光线投射法需要的光线投射函数。因此读者就会产生疑问：为什么VTK中会有多个不同的体绘制方法？为什么不直接选择一种最好的策略？首先，那种方法最好是难以确定的。当图像较小，有较多处理器可用，或者图形硬件限制渲染速率时，光线投射方法要优于纹理映射方法。这些参数在不同平台之间都是不同的，甚至在运行时都会改变。第二，由于计算复杂度限制，体绘制只能产生一种近似的效果。例如，图像内部采样并通过alpha混合函数对样本进行合成的技术，仅仅是对图像内部属性的近似。不同的环境下，不同的渲染技术在渲染质量和速度上都会有较大的差别。另外，有些技术仅仅在特定的条件下工作。例如，有些渲染技术仅仅支持unsigned char或者unsigned short类型的标量数据，而其他技术则支持任意的标量或者多元数据。“最好的方法”依赖于的要处理的数据，系统的硬件配置等因素。事实上，“最好的”技术实际上是一些技术的组合。本章主要讲述的是跨平台的交互体绘制技术，该技术主要使用的是细节层次技术（LOD）。

7.4 创建vtkVolume

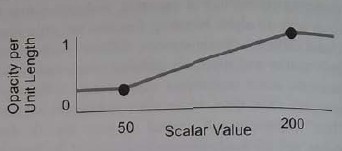
vtkVolume继承自vtkProp3D，用来实现体绘制。类似于几何渲染中的vtkActor，vtkVolume中存储了变换信息，如位置，方向和缩放比例，以及mapper和property指针。关于如何控制变换信息，参考52页“控制3D属性”。

vtkVolume中SetMapper函数接收vtkAbstractVolumeMapper的子类类型数据，SetProperty()函数接收vtkVolumeProperty类型数据。vtkActor和vtkProperty是两种不同的对象来驱动各自不同的mappers和properties。因此几何绘制中的参数在体绘制中可能毫无意义，反之亦然。例如，vtkProperty中的SetRepresentationToWireframe()方法在几何渲染中是没有意义的。

7.5 vtkPiecewiseFunction函数使用

VTK中定义了多个映射函数来控制三维图形标量属性的显示。一般来说，所有的体绘制方法都需要定义两个映射函数。第一个叫做标量不透明度映射函数，即将图形灰度值映射为一个不透明度值。第二个称为颜色变换函数，将图像灰度值映射为一个彩色值。另外，有的体绘制方法还用到一个梯度不透明度映射函数。该函数将灰度的梯度值映射为一个不透明度的乘子。这些映射函数都是一对一的映射，因此可以使用vtkPiecewiseFunction表示。对于灰度值向彩色值的映射，则可以使用vtkColorTransferFunction。

从用户的角度看，vtkPiecewiseFucntion有两个方法，一个是添加映射信息。另一个是清除映射信息。当映射信息添加后，那么就可以通过插值方法计算介于添加的两个样本点之间的任意点的映射值。例如，下面一段脚本生成的映射函数如右侧图所示：

vtkPiecewiseFunction tfun

tfun AddPoint 50 0.2

tfun AddPoint 200 1.0

该例中标量值50和200分别映射为0.2和1.0，通过线性插值方法即可获得两者之间任意标量值的映射值。如果Clamping设置为on状态（默认），那么小50的值都被映射为0.2，大于200的任意值将被映射为1.0。如果设置Clamping为Off，那么50-200范围以外的值都被映射为0。

任意时刻都可以为vtkPiecewiseFunction添加映射点。当映射点重新定义后，就取代当前的映射点。除了添加一个映射点，还可以添加一个映射线段，并清除该线段内的所有映射点。下面代码说明了该功能。

tfun RemovePoint 50

tfun AddPoint 50 0.0

tfun AddSegment

100 0.8 150 0.2

tfun AddPoint 50 0.2

tfun AddSegment

60 0.4 190 0.8

tfun ClampingOff

首先移除映射点50，并重新添加该映射点，接着添加一个映射线段。在第二步中，直接添加了映射点50，而没有将其先移除，同样可以达到修改该映射点的效果。同时，添加了一个新的映射线段，添加后会移除所有位于该线段内的映射点，即100和150。最后关闭Clamping功能。

7.6 vtkColorTransferFunction使用

vtkColorTransferFunction用来完成标量值和RGB或者HSV彩色值之间的映射。其定义的方法类似于vtkPiecewiseFunction，但是分别定义了两个版本。例如，AddRGBPoint()和AddHSVPoint ()都是为映射函数添加一个映射点，但是前者接收RGB值，后者接收HSV值。

下面的Tcl代码演示了怎么定义一个由红到绿再到蓝的映射函数。

vtkColorTransferFunction ctfn

ctfun SetColorSpaceToRGB

ctfun AddRGBPoint 0 1 0 0

ctfun AddRGBPoint 127 0 1 0

ctfun AddRGBPoint 255 0 0 1

7.7 使用vtkVolumeProperty控制颜色/不透明度

在前面两节中我们讨论了怎样创建传输函数，但是没有说明这些传输函数怎样来控制体绘制的效果。通常定义传输函数是有效实现体绘制的一个难点，因为这是一个分类操作，需要用户能够理解数据潜在的意义。

渲染技术是将一个像素点映射到一个空间位置上（例如等值面绘制或者最大灰度投影），通过不透明度函数将灰度值映射为不透明度。当使用合成技术时，不透明度函数将灰度值映射为该灰度值对应匀质区域中单位长度上不透明度的累积值。然后由专门的mapper将连续变化的颜色值和不透明度进行累积从而形成最终的颜色和不透明度，并存储到相应的像素中。

不透明度传输函数和颜色传输函数用来对数据进行简单的分类。将背景或者噪声像素的不透明度值映射为0.0，从而将他们从图像内容中消除。剩余的像素值分类为不同的材料，他们之间具有不同的不透明度和颜色值。例如，如同7-2，根据图像强度值，可以将CT数据分为空气，软组织和骨头。空气对应像素的不透明度应该映射为0.0，软组织应该设置颜色为红色-棕色之间，而骨头颜色应该为白色。通过改变后两种材料的不透明度，就可以对皮肤表面或者骨头表面进行可视化。决定数据中不同材料之间的临界值会非常的繁琐，甚至有些情况下不可能的，仅仅根据原始数据。例如，肝脏和肾脏强度值非常接近。这种情况下，需要分割算法来改变原始图像的数据以便能够正常的对材料进行分类，或者提取出相应的材料。这些分割操作还需要借助额外的信息，如位置，以及参考图像等。



这里给出了两个例子，这两个例子都只利用vtkProperty中定义的传输函数进行分割，分别是躯干数据分割（图7-2）和头颅数据分割（图7-3）。两个例子中都用到了另外一个传输函数，该函数将梯度模值映射为一个不透明度乘子，从而增强过渡区域的显示效果。例如，在不同材料的临界区域，如空气到软组织，或者软组织到骨头的临界区，梯度值会比较大，而材料的内部梯度值则会相对比较小。下面代码定义了一个8位无符号数据的梯度不透明度传输函数。

vtkPiecewiseFuction gtfun

gtfun AddPoint 0 0.0

gtfun AddPoint 3 0.0

gtfun AddPoint 6 1.0

gtfun AddPoint 255 1.0

该函数将所有梯度值小于3的区域的不透明度乘子定义为0，从而消除了匀质区域。在梯度值介于3和6之间的区域，该乘子值通过0和1之间的一个线性映射计算。而当梯度值大于6的区域，不透明度则保持不变。对于噪声数据则需要更有效的边缘检测（因此3和6两个值应该取更大一些）。注意，梯度传输函数只是在渲染vtkImageData的mapper中用到。而渲染vtkUnstructuredGrid数据时，由于梯度没有计算，因此无论是梯度传输函数还是阴影都没有使用。

vtkVolumeProperty中有几个与颜色和不透明度传输函数有关的函数。SetColor()函数可以接收vtkPiecewiseFunction（当颜色函数只是定义灰度值时）或者vtkColorTransferFunction。可以通过GetColorChannels()函数获取颜色通道个数。当设置为vtkPiecewiseFunction时，返回值为1，而当指定了vtkColorTransferFunction时，返回值为3。一旦知道颜色通道数目时，就可以调用GetGrayTransferFunction()或者GetRGBTransferFunction()来得到相应的传输函数。

SetScalarOpacity()函数接收vtkPiecewiseFucntion传输函数对象来定义标量不透明度传输函数，而通过GetScalorOpacity()则可以获取该传输函数。同样的，还有两个函数来设置和获取梯度不透明度传输函数：SetGradientOpacity()和GetGradientOpacity()。

目前只是讨论了单成分标量数据的传输函数定义，而多成分数据的传输函数可以通过以下两种方式定义。如果成分之间是相互独立的，那么就可以为每一种成分定义相应的传输函数。例如，通过模拟产生的不规则网格，网格每个节点都具有温度和密度数据。另外一个例子是，当使用显微镜对样本进行多次扫描，而每次扫描采用不同的荧光粉来区分样本的内部结构，这样得到的多成分数据也是独立的。当渲染相互独立的多成分数据时，必须为每个成分定义合适的参数。SetColor()，SetScalarOpacity()和SetGradientOpactiy()方法的第一个参数为成分索引，通过给定成分索引，就可以为每个成分设置相应的传输函数。

有的多元数据，各个成分之间不是独立的，多个成分定义了一种属性。例如，当使用物理切片技术时，可能会得到三个或者四个成分数据来表示RGB或者RGBA。也或者有两个成分数据，分别表示亮度和alpha值。对于支持多元数据的体绘制mapppers来说，有两种方式来处理多元非独立数据。第一种方式是将二元数据的第一个成分传递给标量不透明度函数来定义样本alpha值。第二种则是将四元数据的前三个成分数据直接定义为RGB值，第四个成分传递给标量不透明度函数来定义alpha值。这两种情况都是利用第四个成分计算梯度并用来控制梯度不透明度传输函数。

注意并不是所有的mapper都支持多元数据，请参考本章后续mapper相关内容来获取更多信息。而对于支持多元数据的mapper来说，通常要求是四元数据。

7.8 利用vtkVolumeProperty控制阴影

利用vtkVolumeProperty控制阴影，类似于使用property来控制几何Actor的阴影（见53页“Actor Properties”和54页“Actor Color”）。阴影控制主要用到阴影标志变量和四个基本参数：环境光系数，漫反射光系数，镜面光系数和高光强度。一般来说，前三组的系数和为1，但是在体绘制中为了提高亮度，有时也会超出1。这些参数的解释依赖于特定体绘制方法的光照方程。通常如果环境光占主导的话，物体显示会没有阴影效果，如果漫反射占主导，物体看起来会比较粗糙（如水泥面）；而如果镜面反射占主导，物体就会表现的较为平滑（如玻璃）。高光强度主要控制了物体表面的平面程度（如拉丝金属对比抛光金属）。

阴影是默认关闭的，可以调用ShadeOn()函数来打开阴影设置。关闭阴影效果等价于设置环境光系数为1，漫反射系数为0，镜面反射系数为0。注意当前用于渲染的vtkUnstructuredGrid数据的体绘制mapper是不支持阴影的。另外，部分渲染vtkImageData的体绘制技术，如基于最大强度光线函数的光线投射体绘制方法，会忽略阴影系数。

开启阴影的体绘制效果不仅依赖于vtkVolumeProperty中的阴影参数，而且也依赖于renderer中的光源及其属性。体绘制效果会依赖于场景中的光源个数，位置和颜色。

如果可能，体绘制将尝试重新生成由OpenGL定义的光照方程。参考下面例子。



#create a geometric sphere

vtkSphereSource sphere

sphere SetRadius 20

sphere SetCenter 70 25 25

sphere SetThetaResolution 50

sphere SetPhiResolution 50

vtkPolyDataMapper mapper

mapper SetInput [ sphere GetOutput ]

vtkActor actor

actor SetMapper mapper

[actor GetProperty] SetColor 1 1 1

[actor GetProperty] SetAmbient 0.01

[actor GetProperty] SetDiffuse 0.7

[actor GetProperty] SetSpecular 0.5

[actor GetProperty] SetSpecularPower 70.0

#Read in a volumetric sphere

vtkSLCReader reader

reader SetFileName “$VTK\_DATA\_ROOT/Data/Sphere.slc”

#Use this tfun for both opacity and color

vtkPiecewiseFunction opacityTransferFunction

opactiyTransferFunction AddSegment 0 1.0 255 1.0

#Make the volume property match the geometric one

vtkVolumeProperty volumeProperty

volumeProperty SetColor opacityTransferFunction

volumeProperty SetScalarOpactiy tfun

volumeProperty ShadeOn

volumeProperty SetInterplationTypeToLinear

volumeProperty SetDiffuse 0.7

volumeProperty SetAmbient 0.01

volumeProperty SetSpecular 0.5

volumeProperty SetSpecularPower 70.0

vtkVolumeRayCastCompositeFunction compositeFunction

vtkVolumeCastMapper volumeMapper

volumeMapper SetInput [reader GetOutput]

volumeMapper SetVolumeRayCastFunction compositeFunction

vtkVolume volume

volume SetMapper volumeMapper

volume SetProperty volumeProperty

#Add both the geometric and volumemetric spheres to the renderer

ren1 AddProp volume

ren1 AddProp actor

#Create a red, green, and blue light

vtkLight redlight

redlight SetColor 1 0 0

redlight SetPosition 1000 25 25

redlight SetFocalPoint 25 25 25

redlight SetIntensity 0.5

vtkLight greenlight

greenlight SetColor 0 1 0

greenlight SetPosition 25 1000 25

greenlight SetFocalPoint 25 25 25

greenlight SetIntensity 0.5

vtkLight bluelight

bluelight SetColor 0 0 1

bluelight SetPosition 25 25 1000

bluelight SetFocalPoint 25 25 25

bluelight SetIntensity 0.5

#Add the lights to the renderer

ren1 AddLight redlight

ren1 AddLight greenlight

ren1 AddLight bluelight

#Render it!

renWin Render

图7-4中左边的球采用光线投射体绘制方法绘制，而右边的球则是利用OpenGL面绘制技术绘制。由于vtkActor中的vtkProperty，以及vtkVolume中的vtkVolumeProperty都设置了同样的环境光系数，漫反射光系数，镜面反射光系数和高光强度，所以两个球都是白色，都有相似的外观。

当渲染多元独立成分数据时，需要为每个成分设置参数。SetAmbient()，SetDiffuse()，SetSpecular()和SetSpecularPower()函数的提供了第一个可选参数，该参数可以指定要设置的成分索引。虽然vtkVolumeProperty的API函数运行打开或者关闭每个成分的阴影，但是没去的体绘制mapper并不支持。因此索引的阴影变量都必须设置为on或者off。

7.9 创建体绘制mapper

所有体绘制mapper的抽象基类是vtkAbstractVolumeMapper，不能直接创建该类对象，而是根据具体的需要创建相应子类的对象。在VTK5.4中，适用于vtkImageData的mapper类有vtkVolumeRayCastMapper，vtkVolumeTextureMapper2D，vtkFixedPointVolumeRayCastMapper，vtkVolumeTextureMapper3D，和vtkVolumeProVP1000Mapper。而适用于vtkUnstructuredGrid数据的mapper类有vtkUnstructuredVolumRayCastMapper，vtkUnstructuredGridZSweepMapper，vtkProjectedTetrahedraMapper，和VTKHAVSVolumeMapper。

所有的mapper类都提供SetInput()函数来接收vtkImageData或者vtkUnstructuredGrid类型的对象指针。对于vtkImageData的体绘制mapper，每个体绘制方法都支持部分类型的vtkImageData数据。例如，vtkVolumeRayCastMapper和vtkVolumeTextureMapper2D只能支持单成分VTK\_UNSIGNED\_CHAR和VTK\_UNSIGNED\_SHORT类型数据。而vtkVolumeTextureMapper3D则支持任意类型数据，但是必须是单成分数据，或者是多元独立数据。vtkFixedPointVolumeRayCastMapper灵活性最高，可以支持所有类型数据，最高至四元数据。

7.10 三维修剪

由于体绘制渲染的图像数据大，而且复杂图像的体绘制渲染结果也不易于理解，因此通常只渲染部分数据。用于限定数据渲染范围的两种方法分别是：cropping和clipping。

Cropping技术是通过6个剪切平面定义数据的渲染区域，每个主轴方向上两个平面。Cropping只适用于渲染vtkImageData数据的体绘制mappers。而clipping技术则没有这个限制，可用于渲染vtkImageData和vtkUnstructuredGrid类型的mappers。6个裁剪平面定义在数据的坐标系统下，因此依赖于数据的原点和间隔，但是不受任何作用于数据的变换的影响。最常用的方式是定义个用户感兴趣的子区域，如右图所示。

当绘制指定区域时，必须打开Cropping功能，设置cropping区域标识和设置剪切平面，如下所示。

set xmin 10.0

set xmax 50.0

set ymin 0.0

set ymax 33.0  
set zmin 21.0

set zmax 47.0

vtkVolumeRayCastMapper mapper

mapper CroppingOn

mapper SetCroppingRegionPlanes $xmin $xmax $ymin $ymax $zmin $zmax

mapper SetCroppingRegionFlagsToSubVolume

注意该例子中使用的vtkVolumeRayCastMapper，不过其他所有的vtkVolumeMapper子类都可以替换，因为Cropping函数是定义在父类中。

六个裁剪平面将物体分为27个区域（3X3的网格）。CroppingRegionFlags是一个27位长度的数字，每一位代表一个区域。如果某一位上数字为1，说明其指向的区域需要被渲染，而如果是0，那么说明该区域要被裁剪掉。第一位代表的是坐标小于xmin，ymin，zmin的区域，并按照先沿着x轴，再y轴，最后z轴的顺序排列。

SetCroppingRegionFlagsToSubVoume()函数用来设置显示区域标识，当设置为0x0002000时，表示只有中间区域显示。虽然任何一个27位数字都可以定义一个裁剪区域，实际上只有几个可用。另外还有4个函数可以设置：SetCroppingFlagsToFence()，SetCroppingRegionFlagsToInvertedFence()，SetCroppingRegionFlagsToCross()，SetCroppingRegionFlagsToInvertedCross()，如图7-5所示。

7.11 三维裁剪 Clipping a Volume

除了vtkVolumeMapper提供修剪功能外，vtkAbstractMapper3D支持任意平面裁剪功能。vtkAbstractMapper3D的子类利用OpenGL在硬件中实现裁剪，例如vtkPolyDataMapper，vtkVolumeTextureMapper2D和vtkProjectedTetrahedraMpaper。当使用超过OpenGL支持的最大裁剪平面个数（一般为6个）时，会给出错误信息。而软件渲染技术如vtkVolumeRayCastMapper能支持任意个数的裁剪平面。而vtkVolumeProMapper虽然类中定义了指定由一个平面和一个厚度值组成的裁剪盒的函数，但是该类并不直接支持裁剪平面。

裁剪平面由vtkPlane定义，并指定系那个有的参数，然后将该平面通过AddClippingPlane()函数添加到mapper中。体绘制中裁剪平面的一个常用用法是指定两个互相平行的片面来进行厚度重新格式化操作，如图7-6为CT数据进行裁剪的结果。对于非规则数据，裁剪平面主要用来完成修剪功能来渲染数据的子区域，这方便了观察复杂数据的内部结构。

7.12 法向量编码

标准光照方程依赖于表面法向量来计算漫反射光系数和镜面反射光系数。渲染vtkImageData时，图像一个位置的梯度方向与该点的表面法向量相反。图像中常常利用有限差分技术计算梯度，但是如果采用这个方法计算每条光线上所有采样点的梯度，将会带来巨大计算量，并降低体绘制速度。

避免计算量的一个方法是预先计算出图像中所有网格节点的法线，而网格节点之间的法线则通过某种插值方法计算。如果这么做的话，每个网格节点需要三个浮点数，另外还需要一次平方根计算来得到梯度模值。梯度也可以存储起来，那么每个法向量就需要四个浮点数。由于体绘制的数据量往往都比较巨大，而这种方法就会需要非常多的内存，因为必须将法向量量化为更小的字节数。

对于一些渲染vtkImageData的mappers，我们将法向量的方向量化为2个字节，模值量化为1个字节。法向量的计算由vtkEncodedGradientEstimator的子类完成（目前该类是vtkFiniteDifferenceGradientEstimator），然后通过vtkDirectionEncoder的子类（当前有vtkRecursiveSphere-DriectionEncoder和VTKSphericalDirectionEncoder）将法向量方向量化为两个字节。对于支持法向量编码的mapper来说，其内部都自动定义了这些对象，因此用户不用关心这些对象的创建。而如果一个数据被多个mapper绘制到同一个图像中时，定义一个梯度算子供所有mapper使用会是一个比较有效的方法。如果不这样的话，每个mapper都会有一个法向量数据拷贝，从而耗费了内存空间，并增大了计算量。下面是一个该应用的代码片段：

#Create the gradient estimator

vtkFiniteDifferenceGradientEstimator gradientEstimator

#Create the first mapper

vtkVolumeRayCastMapper volumeMapper1

volumeMapper1 SetGradientEstimator gradientEstimator

volumeMapper1 SetInput [ reader GetOutput ]

#Create gthe second mapper

vtkVolumeRayCAstMapper volumeMapper2

volumeMapper2 SetGradientEstimator gradientEstimator

volumeMapper2 SetInput [ reader GetOutput ]

如果在两个不同的mapper对象中设置同样的梯度算子，那么这两个mapper的数据输入必须一致。否则，每次mapper需要法向量时，梯度算子的数据就会过时，当每次绘制时，梯度就会重新计算。在上面的例子中，法向量方向编码对象并没有显示的定义，因为在每个梯度算子内部自动定义该对象。该对象没有大量的内存需求，这种方式是可以接受的。也可以通过定义一个vtkRecursiveSphereDirectionEncoder对象，通过SetDirectionEncoder()方法设置到每个梯度算子中。