3DMed

用户手册

3DMed 用户手册

version 2.0

中国科学院自动化医学图像处理研究室
网址: http://www.3dmed.net
联系人: 田捷教授
E-mail: tian@doctor.com

目录

插图

第 1 章

功能概述

3DMed 是 3D Medical Image Processing and Analyzing System（三维医学图像处理及分析系统）的简称，由中国科学院自动化研究所医学图像处理研究室研制开发。本软件系统的开发为医学影像领域的从业人员提供了一款实用软件，方便了医学图像数据的分析和处理。软件目前在网站 www.mitk.net 提供免费下载试用.

本软件功能强大，集数据采集、数据格式转换、二维图像处理、面绘制、体绘制、图像分割、图像配准、三维虚拟切割和三维测量等功能于一身，其中分割和配准功能是依靠有关插件提供的。

详细内容欢迎登陆访问我们的主页 http://www.3dmed.net 和 http://www.mitk.net。

1.1 主界面

3DMed 的主界面如图 1.1 所示：

1.1.1 菜单栏

菜单栏主要是显示了 3DMed 的所有功能菜单选项，您可以通过点击一个菜单选项来调用其功能。其中“滤波算法”，“分割算法”，“配准算法”，“可视化算法”菜单都是动态的，他们的内容依赖于动态加载的相关插件。

1.1.2 工具栏

工具栏提供了一些常用的基本功能，您可以通过点击图标方便快捷地实现所需功能。

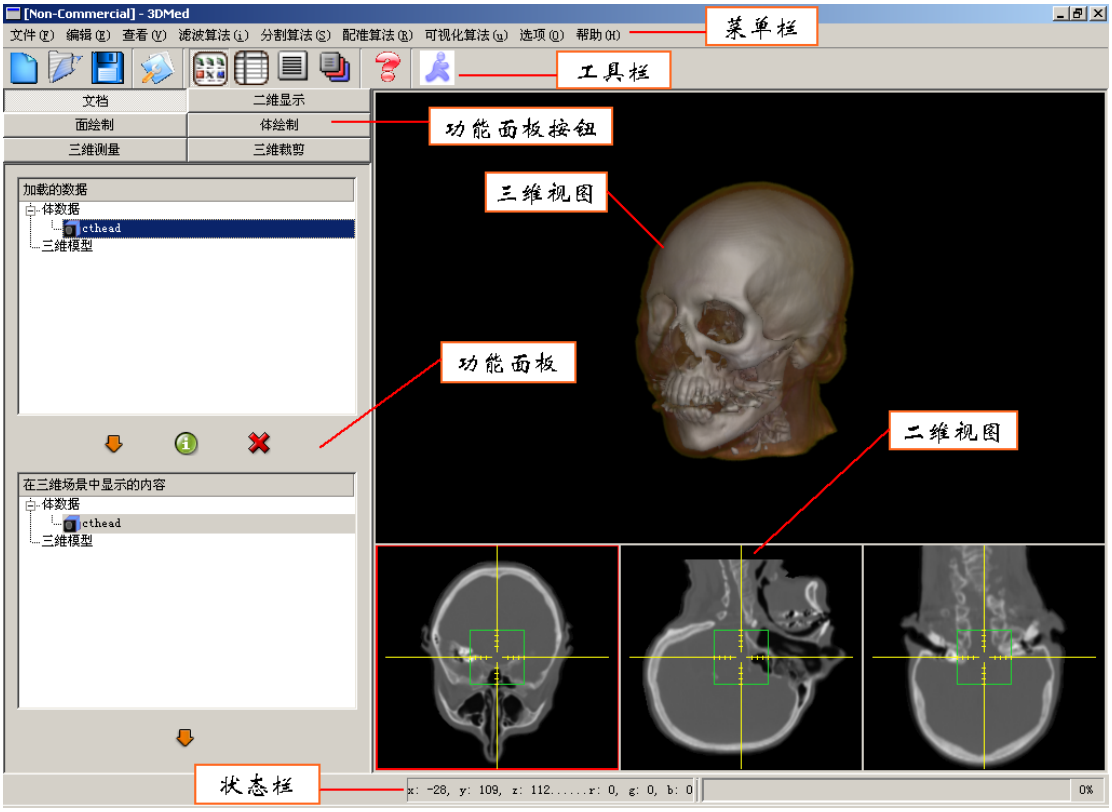


图 1.1 3DMed 主界面

1.1.3 功能按钮和功能面板

功能按钮汇集了3DMed所提供的各种功能，每个按钮对应一个相关功能平台，从而为用户提供了一个方便快捷的界面实现各种功能。

文档按钮及其对应面板提供对输入数据和“三维显示”所展示内容的管理功能。

“二维显示”按钮及其对应面板提供了二位图像处理的功能，详情可参阅第 2 章。

“面绘制”按钮及其对应面板提供了表面绘制功能，详细内容可参阅第 3 章。

“体绘制”按钮及其面板提供了体绘制功能，详情请参阅第 5 章。

“三维测量”按钮及其对应面板提供了三维测量功能，详情请参阅第 4 章。

“三维裁剪”及其对应面板为体绘制提供了三维虚拟切割功能，详情请参阅 5.4 节。

3DMed 还提供了分割和配准功能，菜单栏下的“分割”选项可以提供图像分割功能。系统将在启动时加载分割插件。如果插件已经存在，那么分割菜单下将增加下拉菜单。详细情况请参阅第 6 章。在此次的 3DMed 2.0 beta 版中并未提供配准插件，因此“配准算法”菜单下没有下拉菜单。

1.1.4 三维视图

“三维视图”显示了面绘制或体绘制的结果，详细内容可参阅本书第 3 章和第 5 章。

1.1.5 二维视图

“二维视图”可以一层一层显示医学数据集，详情请参阅第 2 章。

1.1.6 状态栏

状态栏中显示了诸如鼠标位置处在图像空间的坐标、像素值以及任务进程等信息。详情请参阅 2.10 节。

1.2 数据管理

3DMed 提供了多种数据格式，并且可以通过加载插件引入新的格式。

若要向 3DMed 中加载医学数据文件，请选择“文件”→“加载体数据”，在出现的子菜单中选择所需加载的数据文件格式。目前 3DMed 可以支持 BMP、JPEG、TIFF、DICOM、RAW、IM0 格式的体数据加载，如图 1.2 所示。

若要向 3DMed 中加载三维模型（面片）数据，请选择“文件”→“加载三维模型”，在出现的子菜单中选择所需加载的数据文件格式。目前 3DMed 支持PLY格式的三维模型数据加载，如图 1.3 所示。

对加载数据的管理集中在“文档”面板。该面板由上下两个列表框和一些按钮组成。上面的列表框显示当前加载的数据列表，下面的列表框显示加入三维场景中进行三维可视化的数据。如图 1.4 所示。

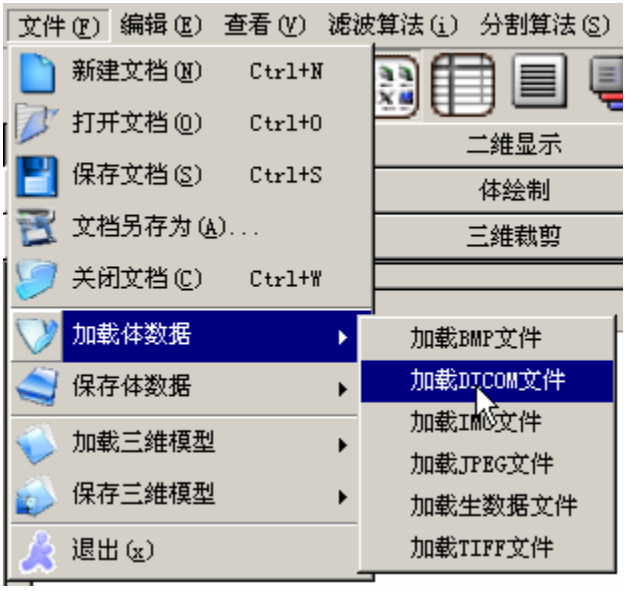


图 1.2　体数据加载菜单

当您将数据集加载进了 3DMed，按面板中部的  铵钮可以对“加载的数据”列表中当前选中数据进行三维可视化，同时将其加入下面的列表中。如果当前选中的是体数据，则对该数据集进行体绘制，具体参数调节请参照第 5 章；如果当前选中的是三维模型，则进行面绘制，具体参数调节请参考第 3 章。

通过面板底部的  按钮可以将“在三维场景中显示的内容”列表中选中的数据从三维场景中移除（即不进行绘制）。

按下  铵钮可以弹出对话框显示载入数据的一些信息，如图 1.5 所示。

 铵钮可以把在“加载的数据”列表中当前选中的数据从内存清除。

另外，“文档”面板还支持鼠标右键操作。用鼠标右键点击“体数据”列表项，可以在弹出的下拉菜单中所要加载的体数据类型。如图 1.6 所示。

类似的，用鼠标右键点击“三维模型”列表项可以利用弹出菜单加载三维模型数据。

若要将处理后的数据保存，则选择“文件”→“保存体数据”或“保存三维模型”，在弹出的子菜单中选择数据保存的格式，可以将当前在文档面板“加载的数据”列表中选中的体数据或三维模型保存至磁盘。目前，体数据的保存格式有 BMP、JPEG、TIFF、DICOM、RAW 和 IM0，三维模型数据的保

图 1.3 三维模型加载菜单

存格式有 PLY （ASCII/Binary）和 STL（ASCII/Binary）。

当然，在文档面板“加载的数据”列表中选中某一个数据点鼠标右键可以实现相同的功能，如图 1.7 所示。

1.3 图像格式转换

3DMed 支持许多数据格式，所以您可以将 3DMed 作为一个图像格式转换器使用。您可以加载一种 3DMed 支持的数据格式，然后将其保存为另一种 3DMed 支持的数据格式。随着不断加入新的数据输入/输出插件， 3DMed 会支持越来越多的数据格式，其数据转换功能也必将得到越来越多的应用。

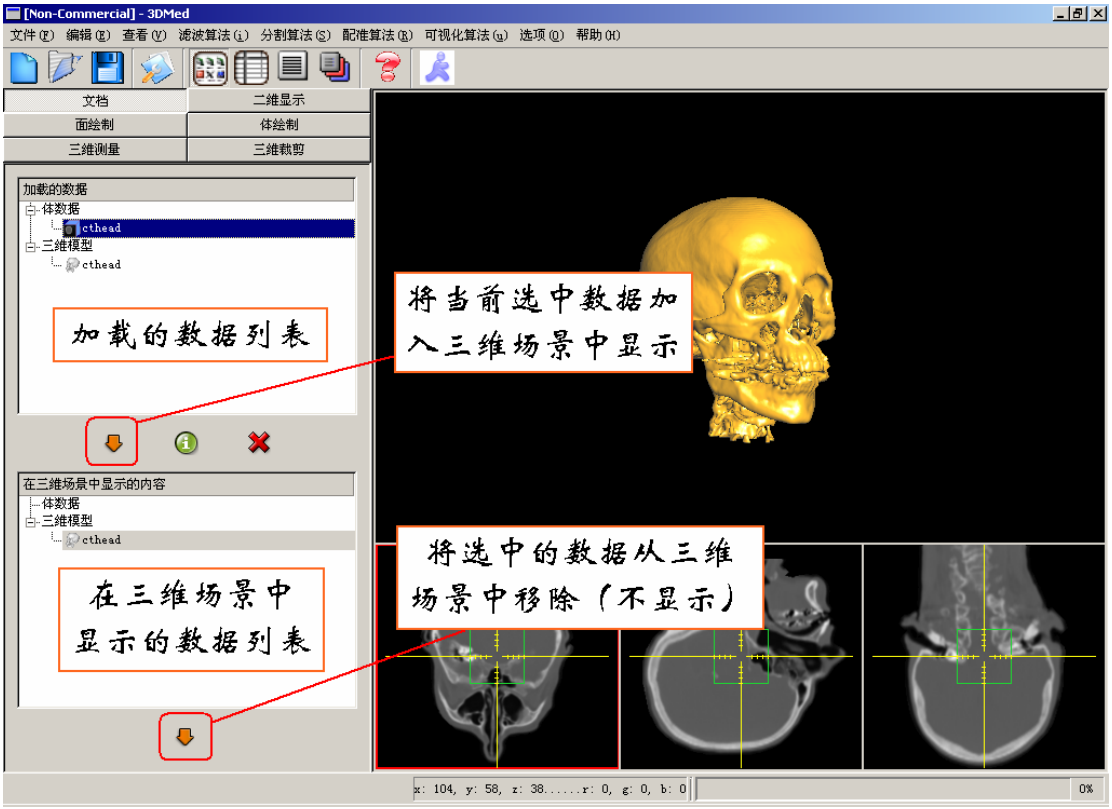


图 1.4 管理加载的数据

图 1.5 显示加载数据信息的对话框

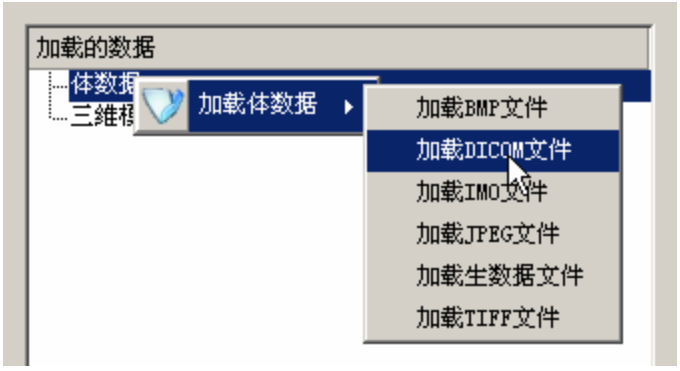


图 1.6　用鼠标右键加载体数据

图 1.7　用鼠标右键导出体数据

第 2 章

二维图像处理

二维图像处理功能包括沿着 Z 轴， X 轴和 Y 轴浏览所有切片（X-Y 断面图像，Y-Z 断面图像和 X-Z 断面图像）；连续显示切片，调整所选图像的窗宽和窗位；显示光标十字指针，放大坐标十字指针中心区域；改变视图布局等。

加载完体数据后，点击“二维显示”就可以进入二维图像处理功能面板，如图 2.1 所示。

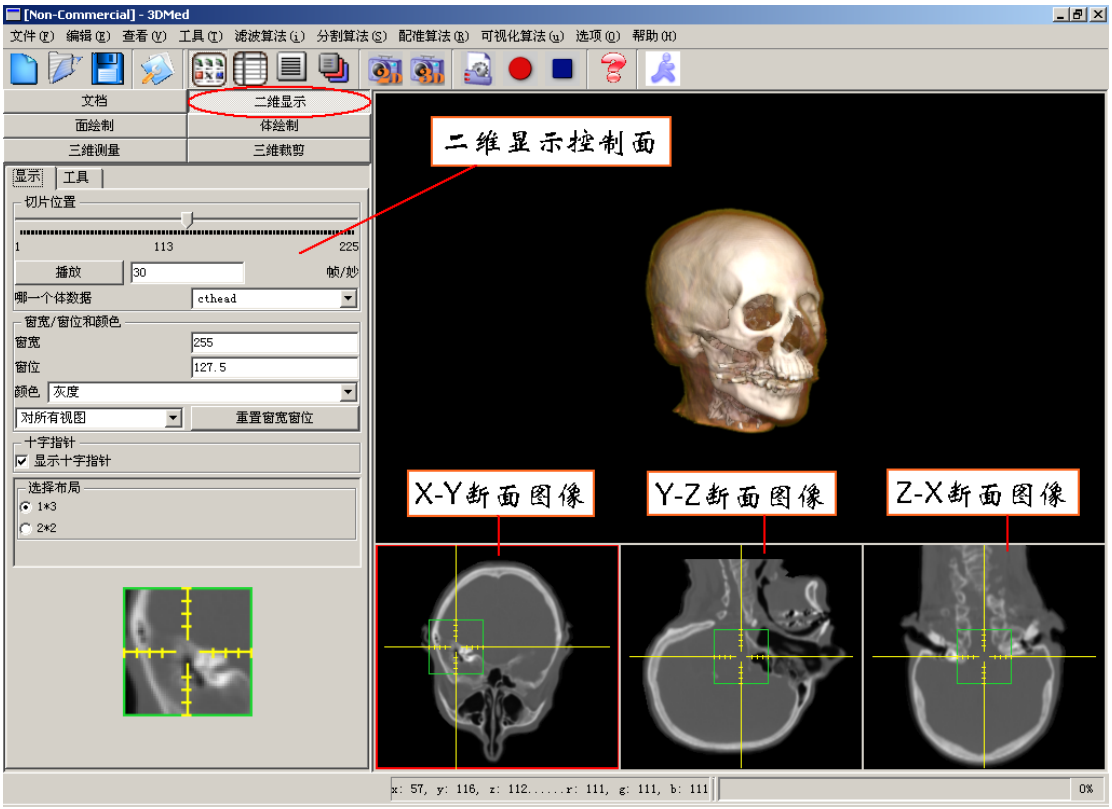
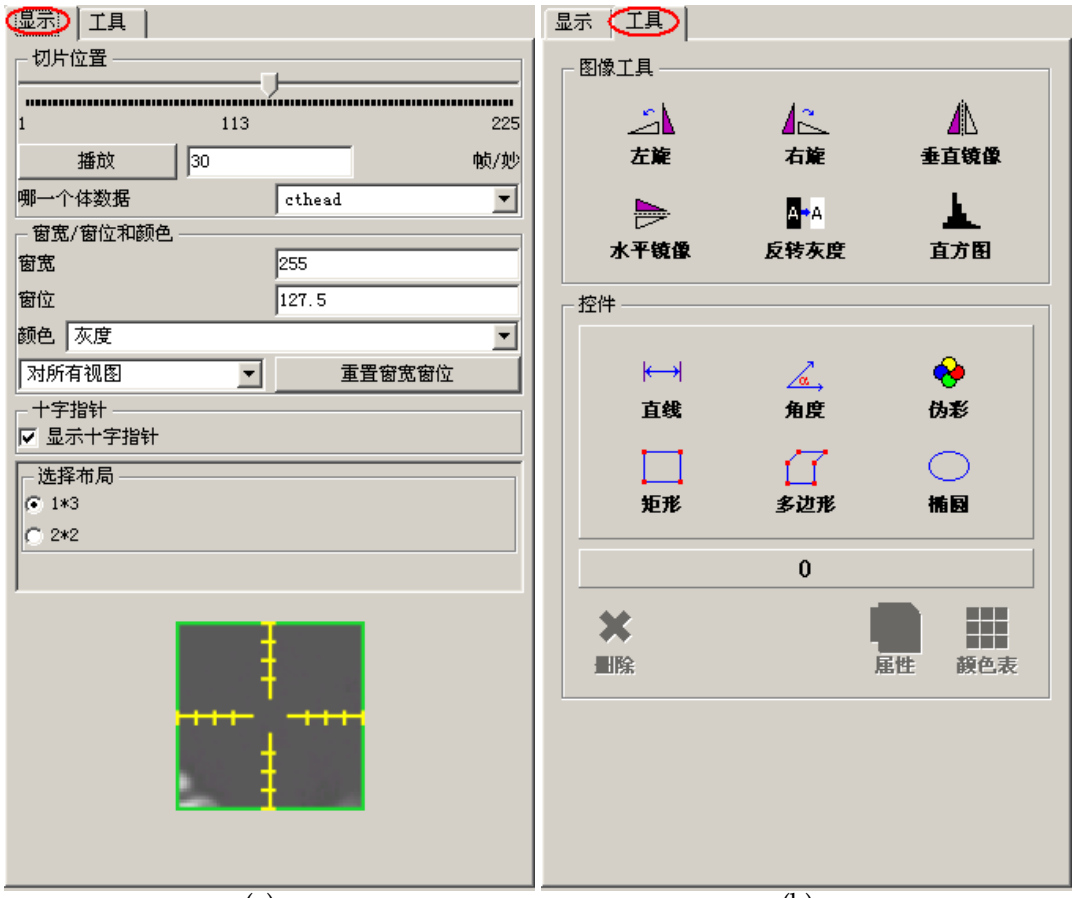


图 2.1　二维图像显示控制

在主窗口的右下方，显示了供您浏览体数据的三种视角（X-Y 断面图像，Y-Z 断面图像和 X-Z 断面图像），其中被红框框住的图像为当前选中的二

维视图（可用鼠标左键点击相应视图进行选择）。

“二维显示”面板包括两个部分：“显示”和“工具”，通过上方的标签按钮可以进行切换，如图 2.2 所示。



(a) (b)

图 2.2 “二维显示”面板

2.1 浏览切片

“切片位置”组用于控制当前选中二维视图所显示的切片，如图 2.3 所示。

其中，拖动滑块可以对所选图像的切片进行定位。滑块下中心处的数字显示了当前所在的切片位置，右边的数显示了所选视图的总切片数。

点击一下“播放”按钮可以连续地显示所有切片。您可以通过修改“播放”按钮旁边编辑框中的数值来改变播放的帧速。再点击一下“播放”就可以

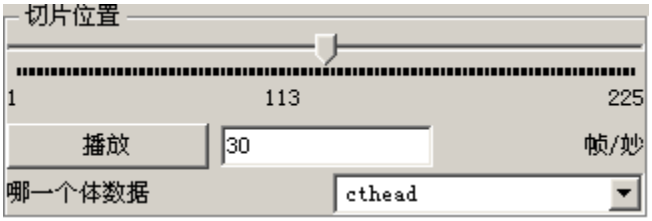


图 2.3 当前切片位置控制

停止播放。

另外左边标有“哪一个体数据”的下拉组合框可以用于选择当前在二维视图中显示哪组体数据。

2.2 调整窗宽/窗位

有两种方法调整显示图像的窗宽（Window Width）和窗位（Window Center），第一种方法是通过鼠标交互，详见 2.9 节；第二种方法是用键盘在“显示”面板的“窗宽/窗位和颜色”组中响应文本编辑框内输入调整后数值，按回车键使更改生效，如图 2.4 。另外，左下角的下拉框用于选择窗宽/窗位的更改仅对选中视图（有红框）还是对所有视图有效，右下角的“重置窗宽窗位”按钮用于恢复窗宽窗位的初始值。

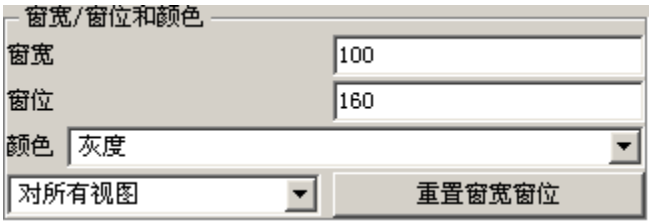


图 2.4 窗宽/窗位的调整

2.3 显示/隐藏十字指针

如图 2.5 所示，通过点击“十字指针”组中的选择按钮可以显示或隐藏二维视图上面覆盖的跟随鼠标移动的十字指针。

图 2.5 显示或隐藏十字指针

2.4 放大镜

当鼠标指针在图像上移动时，切片上当前鼠标所在位置附近的矩型区域会被放大，并显示在“显示”面板的下方，如图 2.6 所示。

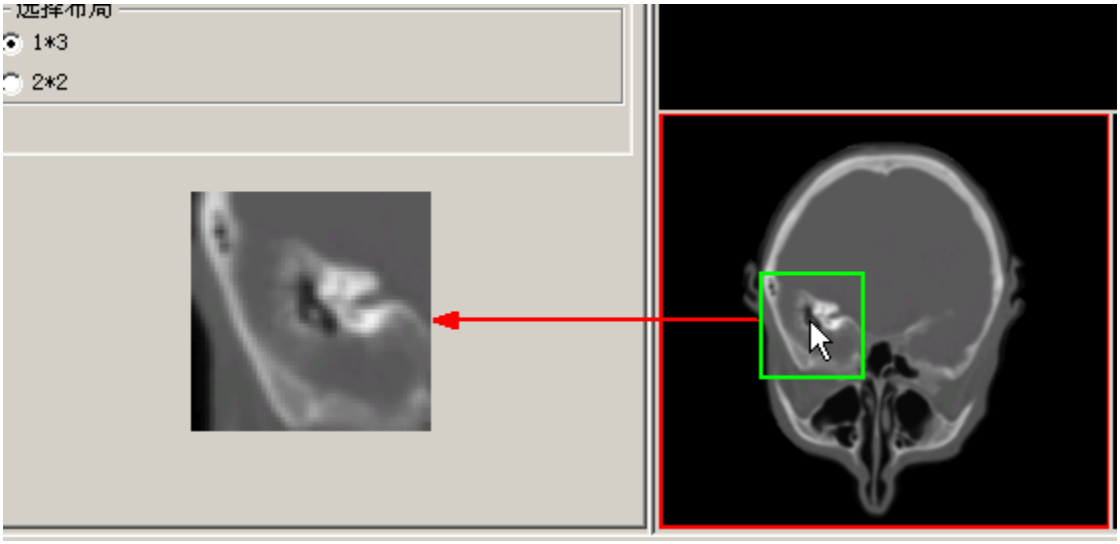


图 2.6 放大器

2.5 窗口布局选择

点击“布局选择”组中的单选按钮来选择各视图窗口的布局（即三个二维视图和一个三维视图的布局）。选择“1*3”时的布局如图 2.7 所示；选择“2*2”时的布局如图 2.8 所示。

2.6 二维图像操作

“图像工具”组（如图 2.9 所示）对二维图像提供一些简单的操作。

其中“左旋”和“右旋”按钮可将选中二维视图中的图像向左（逆时针）或向右（顺时针）旋转 90°；“垂直镜像”和“水平镜像”按钮可将选中二维视图中的图像垂直或水平翻转；“反转灰度”按钮可将显示图像的灰度值反

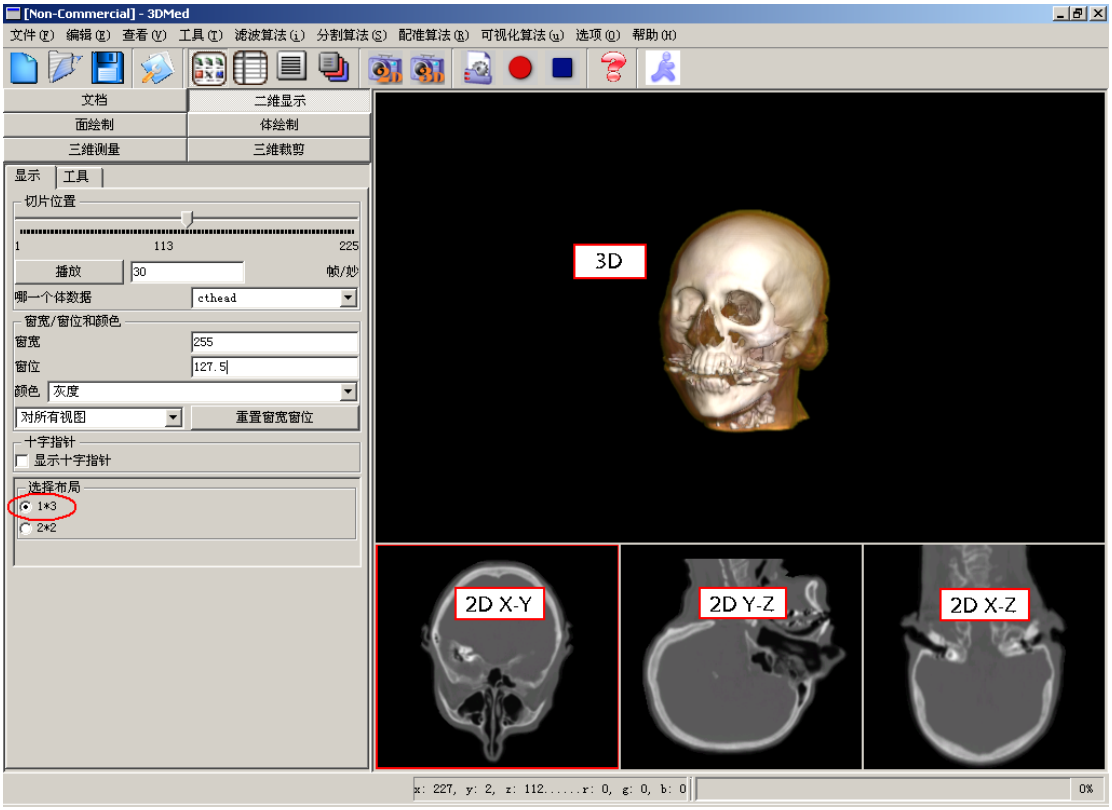


图 2.7　1*3布局

转，产生负片效果；点击“直方图”按钮会弹出一个对话框，显示选中二维视图中各切片图像的直方图和一些统计信息，如图 2.10 所示。

在直方图对话框中，通过改变“切片序号”框中的值可以显示不同切片图像的直方图和统计信息；对于多通道图像，可以通过标有“通道”的下拉框选择显示不同通道的直方图和统计信息。对于具有大面积背景的图像，其直方图中在背景像素值附近将出现一个非常陡的峰，这时候直方图其他部分趋于平坦，以致无法区分感兴趣部分的灰度分布，之时可以通过纵向拉伸对话框来改善显示，或者也可以选中“去除背景”选择按钮，这时候在计算直方图时将忽略灰度从最小值起 5%总灰度范围之内的像素，因为医学图像背景像素大多集中低灰度区域，因此该措施可以有效地改善直方图的显示。此外，当鼠标在直方图显示区域移动时，其指针会变成十字形，同时下面的显示区将显示当前鼠标位置的像素值（横坐标）以及具有该像素值的像素数目（纵坐标）及其在总像素数目中所占的百分比。

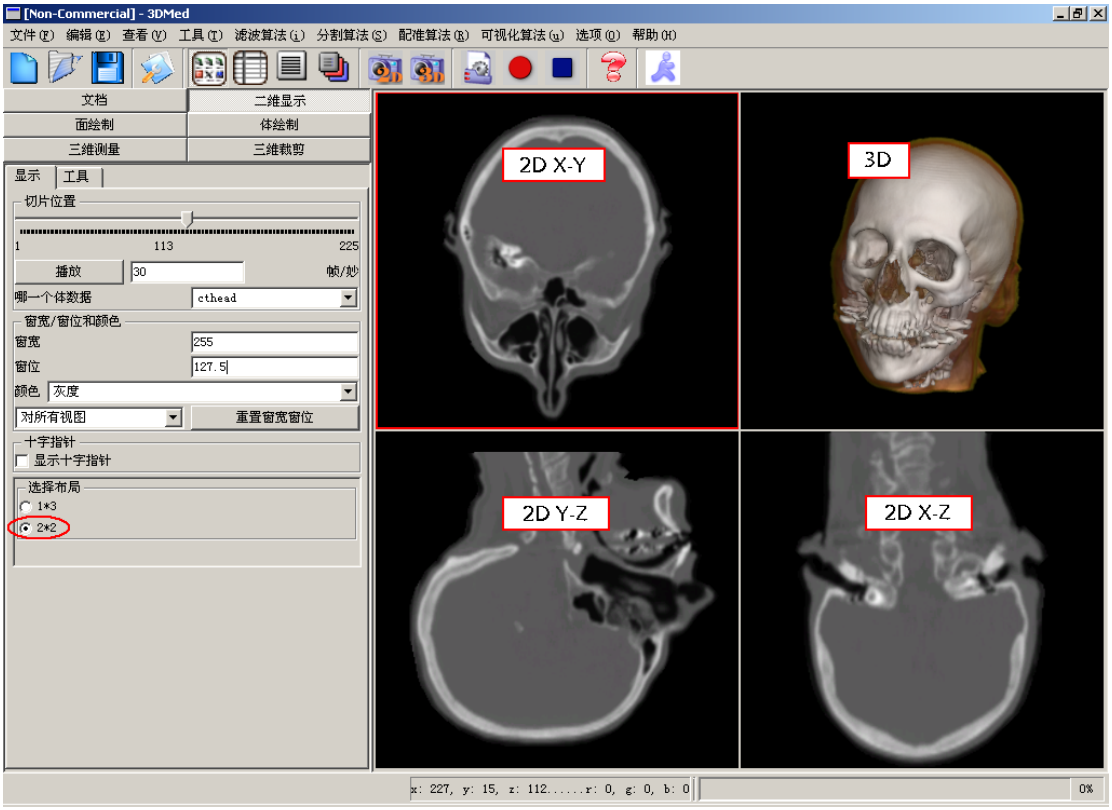


图 2.8 2*2布局

2.7 二维测量和伪彩显示

如图 2.11 所示，在“控件”组提供一组工具按钮对二维图像进行测量和伪彩显示，其中每个按钮都有两种状态：按下和弹起。当处于“按下”状态时可以在当前选中的视图中连续添加多个该按钮对应的控件，再次点击按钮使其恢复成“弹起”状态可以中止添加控件的操作，所有添加到视图中的控件均可以直接用鼠标对其进行控制。下方的“删除”按钮可以删除当前处于选中（激活）状态的控件（如当前无控件处于选中状态，则“删除”按钮为灰色，不可用）。

2.7.1 距离测量控件

距离测量控件实现对同一幅图图像中两点间的物理距离（在被测对象体内的实际距离,而不是在屏幕上显示的距离）的测量。添加该控件的步骤如下：

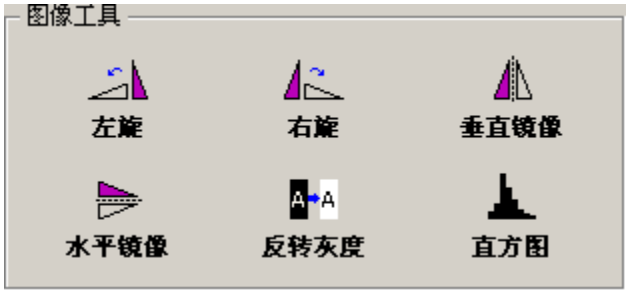


图 2.9 图像工具

图 2.10 直方图对话框

1. 按下“直线”按钮；
2. 在当前选中二维视图内将鼠标移至测量起点，按下鼠标左键确定起点位置；
3. 拖动鼠标至测量终点，松开鼠标左键确定终点位置，即添加了一个距离测量控件（两端带箭头的线段），如图 2.12 所示；

在此期间，“直线”按钮将保持按下状态，可连续添加多条测量线段。所需控件添加完成后，再次单击“直线”按钮使其弹起，从而使添加的控件能够响应鼠标的控制。

在非选中状态下，整个控件呈黄色。用鼠标左键单击线段的任一部位可使线段处于选中状态。当点击的是线段中部直线部分时，直线部分呈绿色，两端

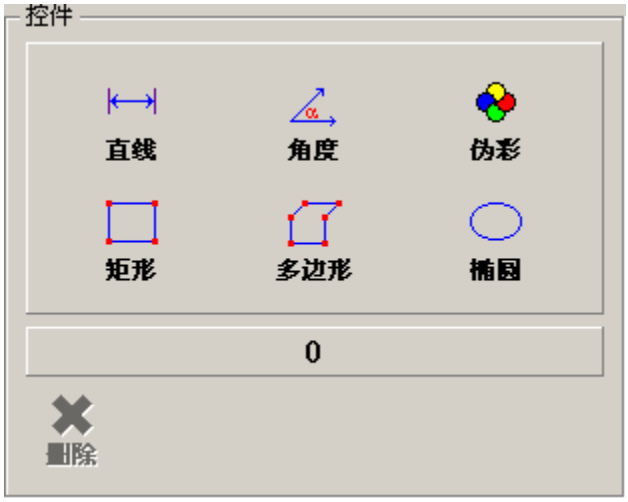


图 2.11　二维测量和伪彩显示

箭头呈红色，这时保持鼠标左键按下并拖动鼠标可以使整条线段随鼠标平移；当点击的是箭头时，点中的箭头呈红色，其余部分呈白色，这时保持鼠标左键按下并拖动鼠标可以使该箭头随鼠标移动以改变端点位置。松开鼠标左键，在视图空白区域（没有控件的区域）单击可取消对控件的选择，这时控件恢复为黄色。

在上述控制过程中，测量线段的当前长度将显示在左边面板的状态显示区，如图 2.12 中红框部分所示。

可点击“删除”按钮将当前选中的控件删除。

2.7.2 角度测量控件

角度测量控件实现对同一幅图图像中三点所形成的两个线段之间的夹角的测量。添加该控件的步骤如下：

1. 按下“角度”按钮；
2. 在当前选定二维视图中将鼠标移至测量角度的顶点的预定位置，单击鼠标左键确定顶点位置；
3. 松开鼠标左键，移动鼠标至测量角度的一条边的端点，单击鼠标左键确定端点位置；

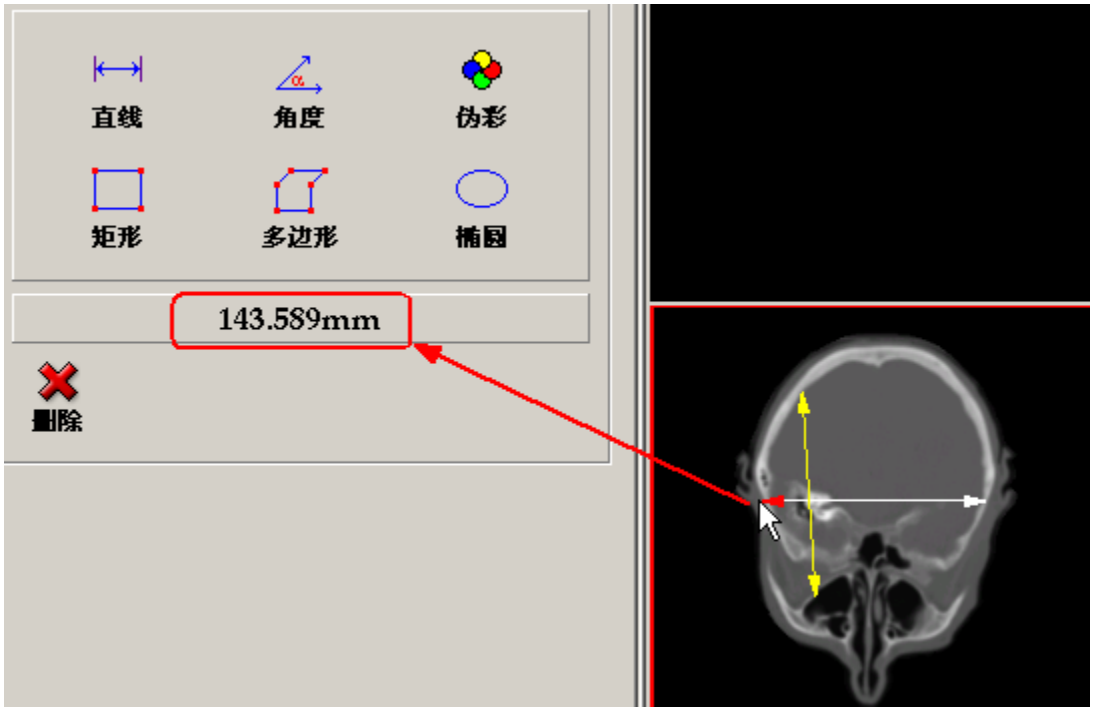


图 2.12　二维距离测量控件

4. 松开鼠标左键，移动鼠标至测量角度的另一条边的端点，单击鼠标左键定此端点位置，即添加了一个角度测量控件，如图 2.13 所示。

在此期间，“角度”按钮将保持按下状态，可连续添加多个角度测量控件。所需控件添加完成后，再次单击“角度”按钮使其弹起，从而使添加的控件能够响应鼠标的控制。

在非选中状态下，整个控件呈黄色。用鼠标左键单击控件的任一部位可使控件处于选中状态。当点击的是线段部分时，线段部分呈绿色，顶点及两个端点箭头呈红色，这时保持鼠标左键按下并拖动鼠标可以使整个控件随鼠标平移；当点击的是顶点或端点箭头时，点中的部分呈红色，其余部分呈白色，这时保持鼠标左键按下并拖动鼠标可以使选中的顶点或端点箭头随鼠标移动以改变该点的位置。松开鼠标左键，在视图空白区域（没有控件的区域）单击可取消对控件的选择，这时控件恢复为黄色。

在上述控制过程中，当前的测量角度将显示在左边面板的状态显示区，如图 2.13 中红框部分所示。

可点击“删除”按钮将当前选中的控件删除。

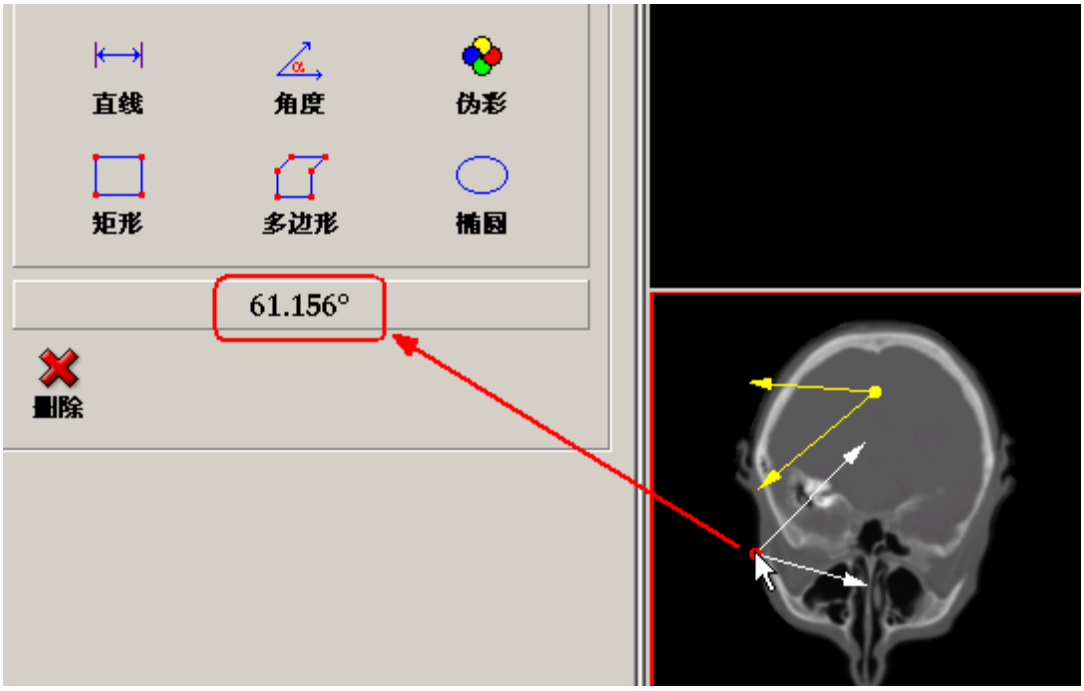


图 2.13　二维角度测量控件

2.7.3 伪彩控件

伪彩控件实现将图像中指定矩形区域用伪彩色显示。添加该控件的步骤如下：

1. 按下“伪彩”按钮；
2. 在当前选定二维视图中按住鼠标左键拖动，可以拉出一个黄色虚线框住的矩形，矩形内部以伪彩显示；
3. 得到合适大小的矩形后释放鼠标左键，即添加了一个伪彩控件，如图 2.14 所示。

在此期间，“伪彩”按钮将保持按下状态，可连续添加多个伪彩控件。所需控件添加完成后，再次单击“伪彩”按钮使其弹起，从而使添加的控件能够响应鼠标的控制。

在非选中状态下，整个控件呈无边框的伪彩窗口。用鼠标左键单击控件的任一部位可使控件处于选中状态。当点击的是伪彩窗部分时，边框线段及端点

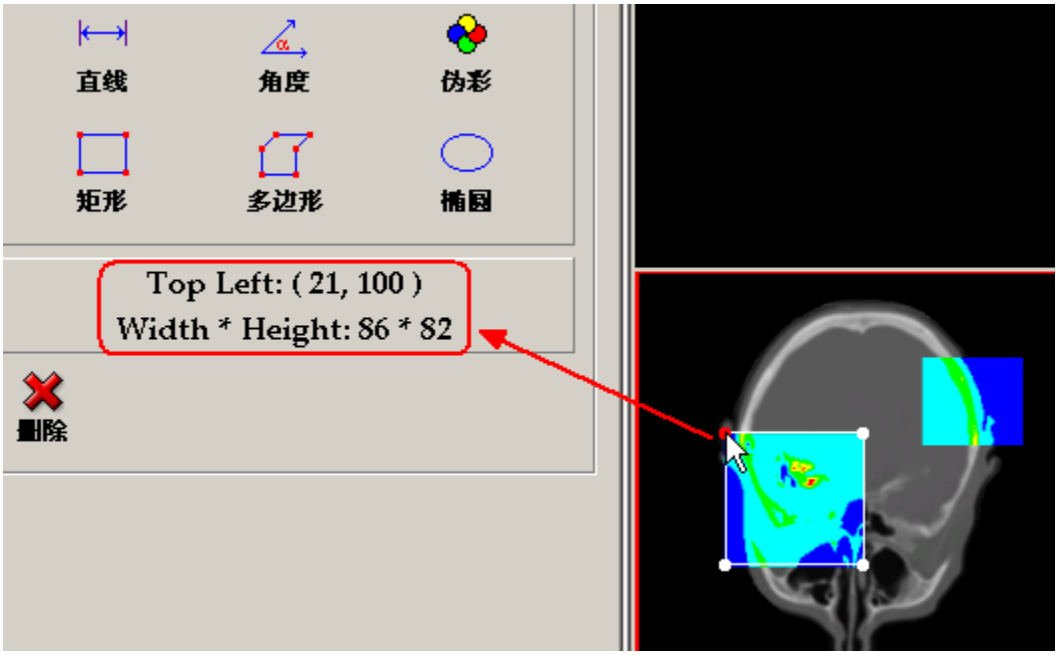


图 2.14　二维伪彩控件

均呈白色，这时保持鼠标左键按下并拖动鼠标可以使整个控件随鼠标平移；当点击的是端点时，点中的端点呈红色，这时保持鼠标左键按下并拖动鼠标可以使选中的端点随鼠标移动以改变该点的位置；当点击的是边框时，点中的边框线段呈绿色，这时保持鼠标左键按下并拖动鼠标可以使选中的边框在垂直该边框的方向上平移（同时改变两个端点的位置）。松开鼠标左键，在视图空白区域（没有控件的区域）单击可取消对控件的选择，这时控件恢复为无边框的伪彩窗口。

在上述控制过程中，当前伪彩窗口的左上角坐标和宽/高（在原图像空间以像素为单位）将显示在左边面板的状态显示区，如图 2.14 中红框部分所示。

当二维视图中有伪彩控件处于选中状态时，“颜色表”按钮被激活，单击该按钮弹出如图 2.15 所示的“设定颜色表”对话框。该对话框上半部分显示了当前切片的直方图，为设定合适的伪彩映射表提供参考。中部的色带区域是颜色表设定区：色带上的每个色块对应于直方图中该灰度区域范围内像素的对应颜色，用鼠标单击色块弹出颜色选择对话框，用于选择该色块的颜色；色带下的一系列三角形控件对应色块直接的分隔位置，可由鼠标进行调整，鼠标左键点中的三角形控件由空心变为实心，按住鼠标左键拖动可改变该三角形控件的位置，从而改变色块所对应的灰度范围；鼠标左键点击控制带的空白区域可

取消对当前三角形控件的选择；鼠标右键在空白区域点击可增加一个三角形控件，同时增加的色块区域初始化为黑色；鼠标右键点击三角形控件可将该控件删除，其对应色块与前一色块合并。对话框下部的“预设值”下拉框提供了一些预先设定的颜色表模式以方便初始颜色表的设置，可供选择的项包括“原始颜色表”（打开对话框之前伪彩控件原有的颜色表）、“彩虹 16 等分”（将灰度值范围等分成 16 份，颜色由蓝色 (0, 0, 255) 到红色 (255, 0, 0) 线性过渡）、“彩虹 32 等分”（将度值范围等分成 32 份，颜色同上）。“像素值”编辑框显示当前选中的三角形控件所对应的像素值。“微调”滚轮用于对三角形控件进行微调，以精确控制三角形控件所对应的像素值。

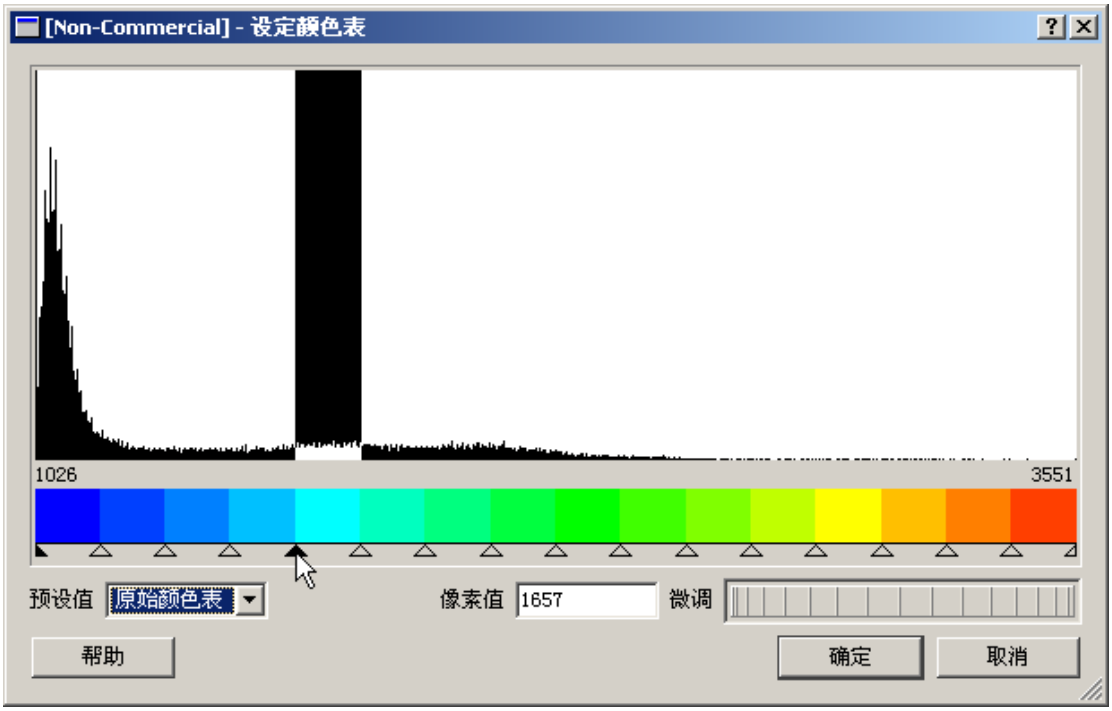


图 2.15　设定颜色表

可点击“删除”按钮将当前选中的控件删除。

2.7.4 矩形控件

矩形控件用于在图像中选定一个矩形区域，如图 2.16 所示。按“矩形”按钮添加，其添加步骤和控制方式同伪彩控件。

在操纵矩形控件时，其物理尺寸将显示在左边面板的状态显示区。

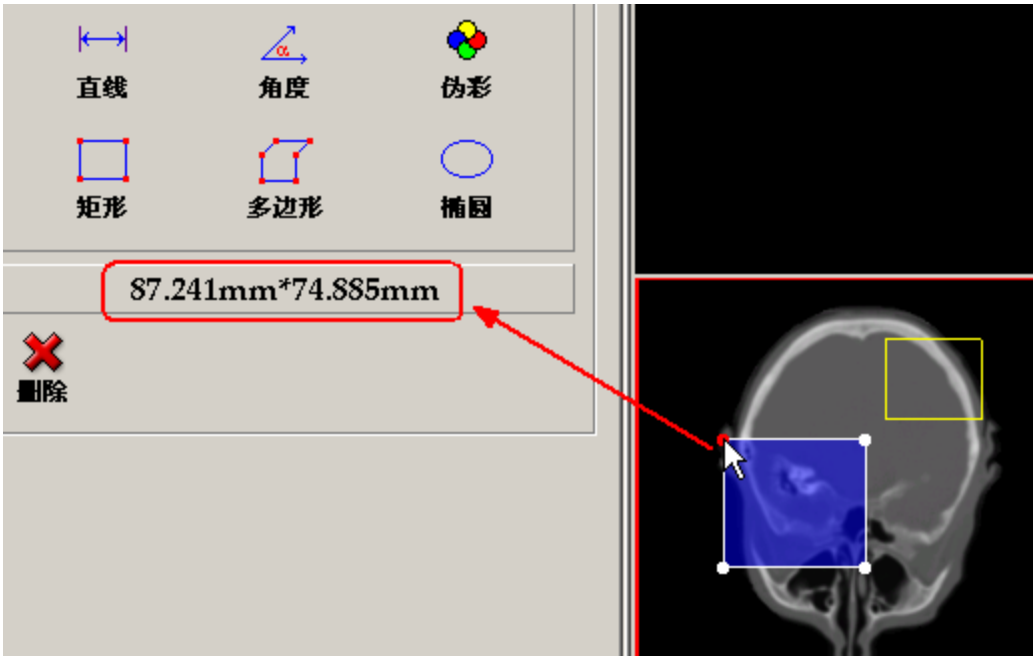


图 2.16　二维矩形控件

2.7.5 多边形控件

多边形控件用于在图像中选定一个多边形区域。添加该控件的步骤如下：

1. 按下“多边形”按钮；
2. 在当前选定二维视图中单击鼠标左键添加一个顶点；
3. 移动鼠标到下一个顶点位置，重复第 2 步添加顶点；
4. 单击鼠标右键，最后一个顶点位置确定，即添加了一个多边形控件，如图 2.17 所示。

在此期间，“多边形”按钮将保持按下状态，可连续添加多个多边形控件。所需控件添加完成后，再次单击“多边形”按钮使其弹起，从而使添加的控件能够响应鼠标的控制。

在非选中状态下，整个控件呈黄色边框的多边形。用鼠标左键单击控件的任一部位可使控件处于选中状态。当点击的是多边形内部时，边框线段及端点均呈白色，这时保持鼠标左键按下并拖动鼠标可以使整个控件随鼠标平移；当

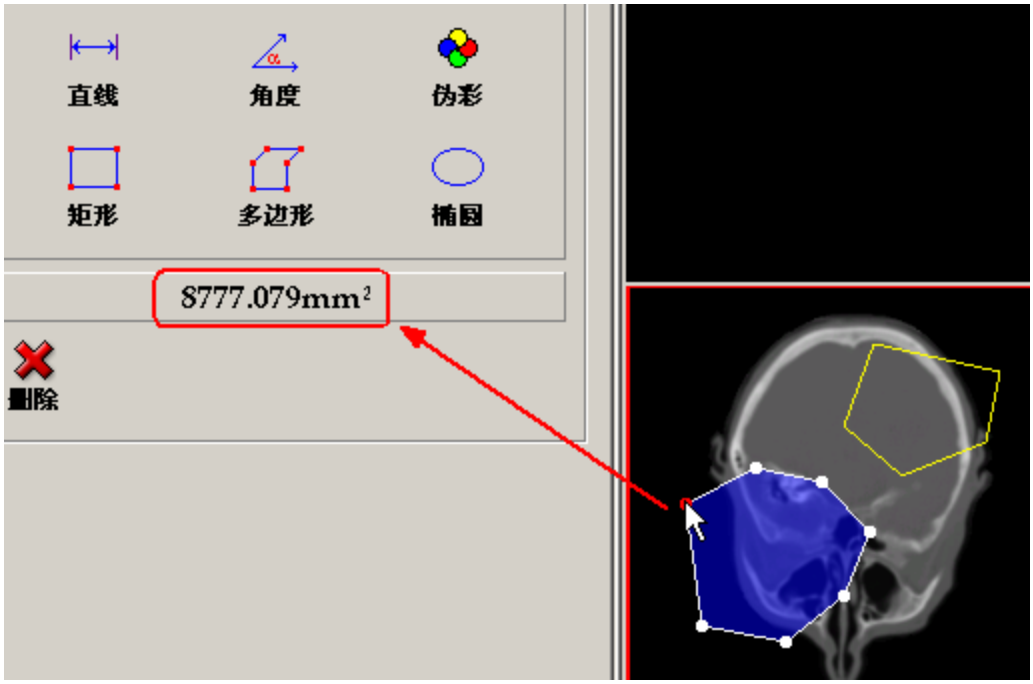


图 2.17　二维多边形控件

点击的是端点时，点中的端点呈红色，这时保持鼠标左键按下并拖动鼠标可以使选中的端点随鼠标移动以改变该点的位置；当点击的是边框时，点中的边框线段呈绿色，这时保持鼠标左键按下并拖动鼠标可以使选中的边框线段随鼠标平移（同时改变两个端点的位置）。松开鼠标左键，在视图空白区域（没有控件的区域）单击可取消对控件的选择，这时控件恢复为无边框的伪彩窗口。

当某边框线段处于选中状态时，单击鼠标右键可在该边框线段的两端点之间插入一个端点，新插入端点的位置是鼠标指针当前位置；当某个端点处于选中状态时，单击鼠标右键可删除该端点。

在上述控制过程中，当前多边形的物理面积将显示在左边面板的状态显示区，如图 2.17 中红框部分所示。

可点击“删除”按钮将当前选中的控件删除。

2.7.6 椭圆控件

椭圆控件用于在图像中选定一个椭圆区域。添加该控件的步骤如下：

1. 按下“椭圆”按钮；

2. 在当前选定二维视图中按住鼠标左键拖动将产生一个内切于鼠标拉出矩形的椭圆；

3. 当产生的椭圆符合要求后，释放鼠标左键，即添加了一个椭圆控件，如图 2.18 所示。

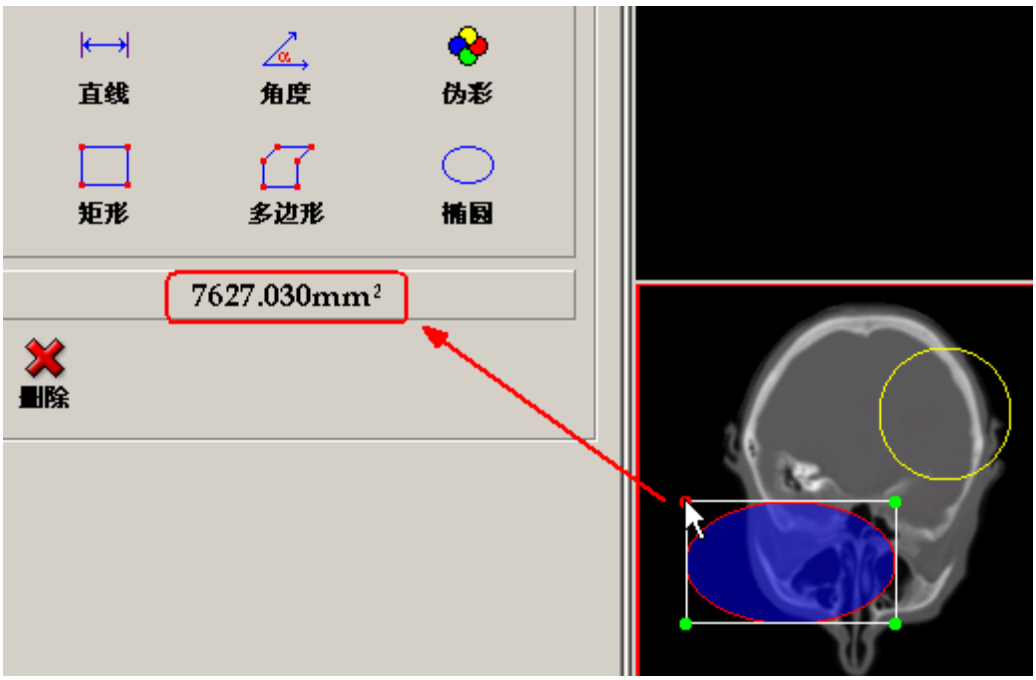


图 2.18　二维椭圆控件

在上述过程中，如果保持键盘“Ctrl”键按下拖动鼠标，则产生的椭圆始终以鼠标左键按下的位置为中心；如果保持键盘“Shift”键按下拖动鼠标则产生规则的圆；如果同时保持这两个键按下拖动鼠标，则产生以鼠标左键按下位置为圆心的规则圆。需要注意的是，务必在鼠标左键释放之后再释放键盘上这两个键，否则不能保证产生的椭圆符合上述要求。

在此期间，“椭圆”按钮将保持按下状态，可连续添加多个椭圆控件。所需控件添加完成后，再次单击“椭圆”按钮使其弹起，从而使添加的控件能够响应鼠标的控制。

在非选中状态下，整个控件呈黄色边框的椭圆。用鼠标左键单击控件的任一部位可使控件处于选中状态。处于选中状态下的椭圆呈蓝色半透明并带红色边框，同时外切一个控制矩形，可以通过调节这个控制矩形来调节椭圆的形状，控制矩形的调节方式同矩形控件。

在上述控制过程中，当前椭圆的物理面积将显示在左边面板的状态显示区，如图 2.18 中红框部分所示。

2.8 二维控件的属性

当二维视图中有二维控件（除角度测量控件外）处于选中状态时，“工具”标签页“控件”组的“属性”按钮被激活，这时鼠标单击“属性”按钮会弹出响应的属性对话框。

如果当前选中的是“直线”控件，则弹出如图 2.19 所示的“线段属性对话框”，该对话框上半部分显示了线段上的像素值分布曲线，当鼠标移进曲线区域时，指针变为十字形，同时在曲线区域下部显示出当前鼠标位置对应于线段上点的坐标以及像素值；对话框的下半部分则显示一些线段的基本信息，如端点坐标、线段长度等。

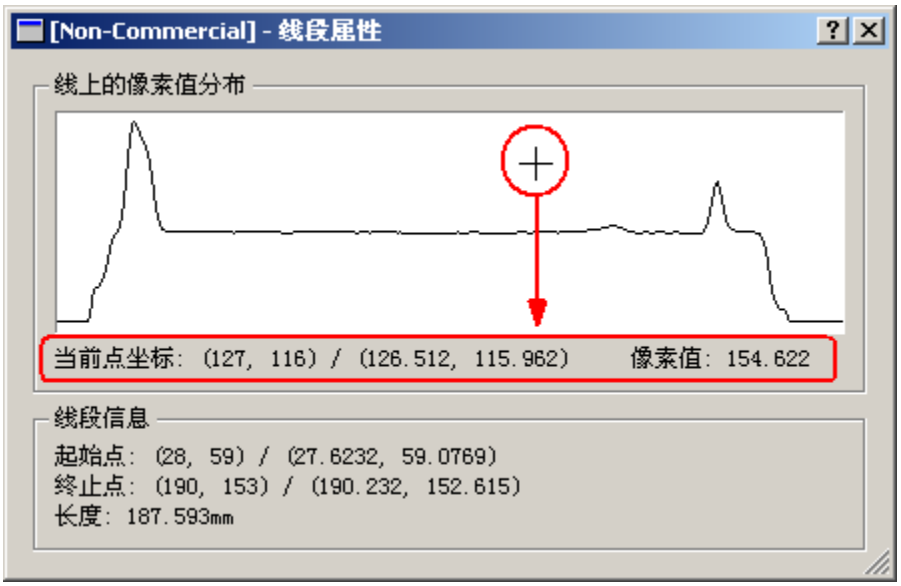


图 2.19　线段属性对话框

如果当前选中的是“区域”控件（如“伪彩”、“矩形”、“多边形”、“椭圆”等），则弹出如图 2.20 所示的“区域属性对话框”。该对话框上半部分显示控件区域的灰度直方图，其显示信息及功能与图 2.10 所示的类似；对话框下半部分则显示所选控件的形状信息，如“矩形”控件则显示矩形左上角坐标和宽、高等信息。

注：在上述对话框所显示的坐标或长度信息中如果包含用“/”分割的两

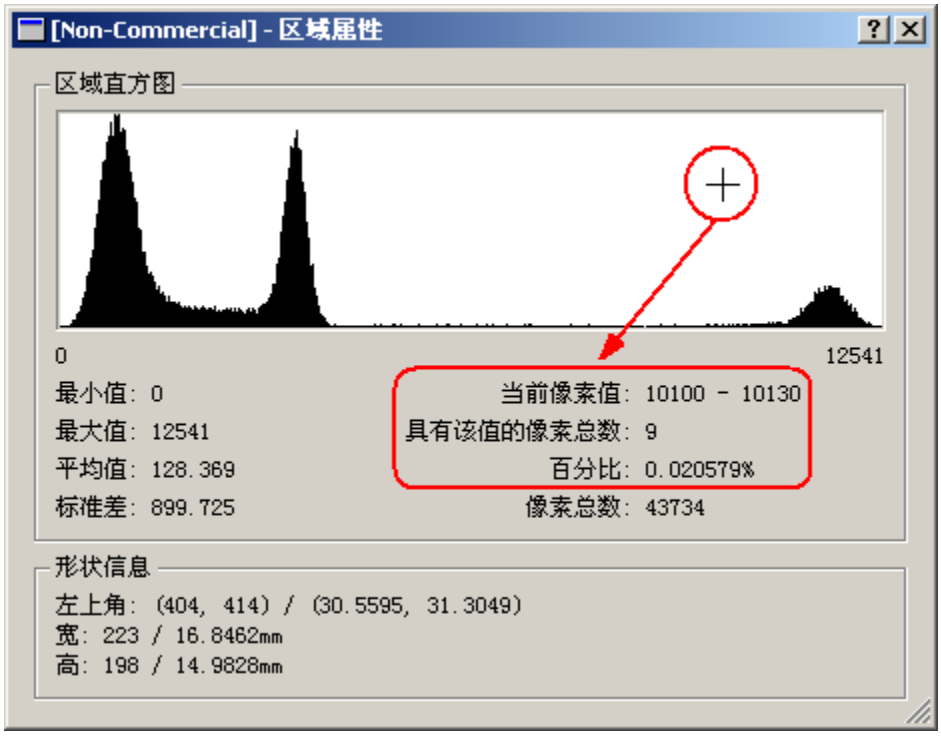


图 2.20　区域属性对话框

部分值，如“(404, 414) / (30.5595, 31.3049)”，则前一部分总是表示在图像空间的坐标（整型数值），以像素为单位，而后一部分则是真实物理空间中的坐标（考虑了像素间距，数值为浮点型）。

2.9 二维图像视图的鼠标操作

平移 按住鼠标左键在二维图像视图区域内拖动；

缩放 按住鼠标右键在二维图像视图区域内拖动；

调整窗宽/窗位 按住鼠标中键在二维图像视图区域内拖动（纵向移动调整窗位，横向移动调整窗宽）。

2.10 状态栏中的信息

当鼠标在二维视图区域内移动时，主窗口下方的状态栏将显示当前鼠标指针位置在图像空间的坐标（以像素为单位）和该点像素值的 R、G、B 分量值，如果是单通道图像，三分量值相等，如图 2.21 红框部分所示。

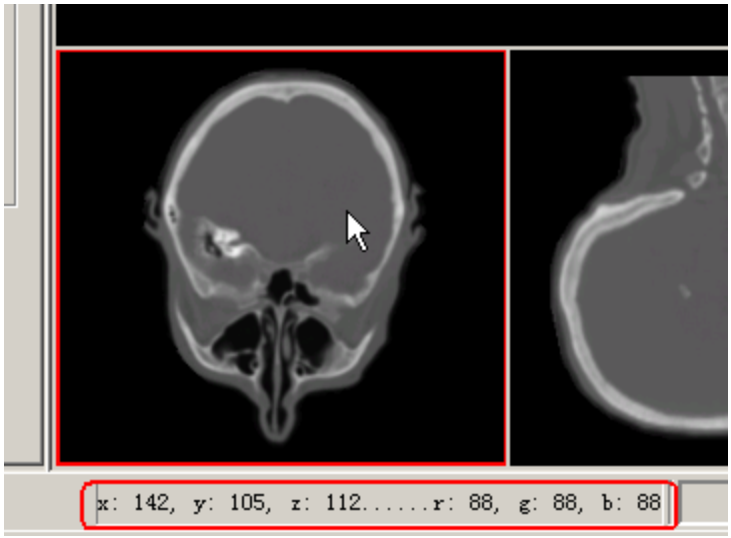


图 2.21　状态栏

第 3 章

面绘制

面绘制作为分割后的一个过程，可以利用分割后的结果和源图像体数据重建出一个三维立体表面模型，进而将此表面模型在三维视图中绘制出来。

例如采用阈值分割方法，分割过程结束后点击“确定”开始表面重建，如图 3.1 所示。

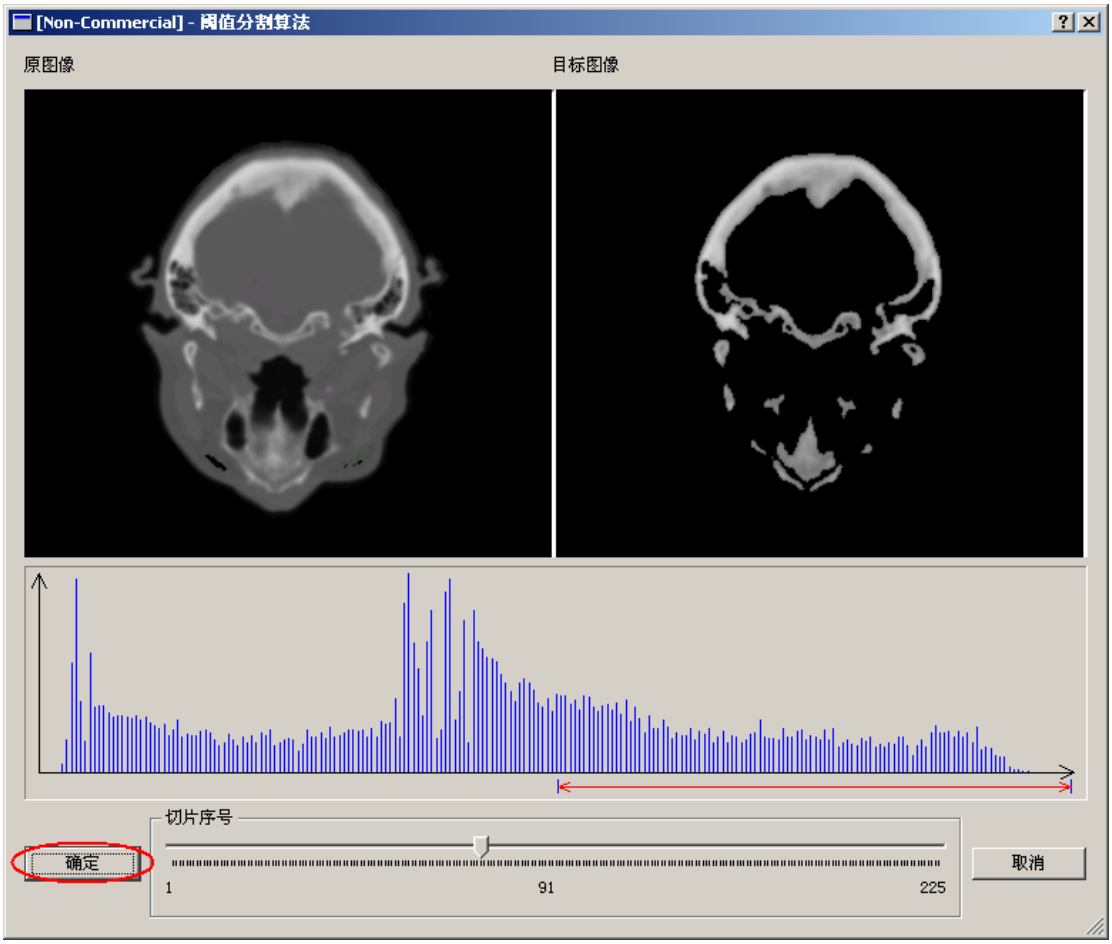


图 3.1　采用阈值分割

如果重建过程成功的话，将可以在主窗口中看到下图所示三维显示结果，如图 3.2 所示。然后在左边的功能面板区按下“面绘制”按钮，激活面绘制的控制面板。利用这个面板再辅之以鼠标，可以方便地操纵表面模型的绘制。

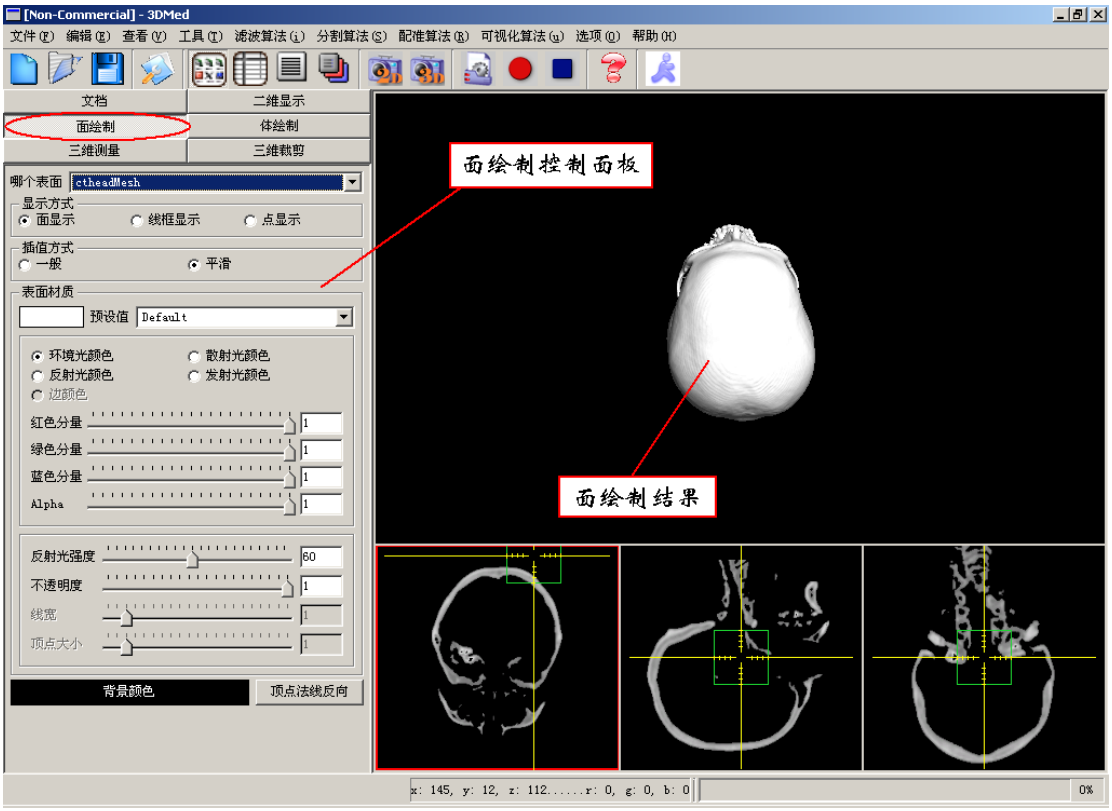


图 3.2　面绘制结果及控制面板

3.1 旋转、平移和缩放的鼠标操作

旋转 在三维视图区域内按住鼠标左键拖动；

平移 在三维视图区域内按住鼠标中键拖动；

缩放 在三维视图区域内按住鼠标右键拖动。

注意： 鼠标操作将同时影响三维视图区域内绘制的所有模型。

3.2 选择表面模型

“面绘制”面板上部标有“哪个表面”的下拉框用于选择当前对面板上绘制参数的调节将应用于加入三维视图中绘制的哪个表面模型，如图 3.3 所示。

图 3.3　选择表面模型

3.3 选择表面模型显示方式

“显示方式”组用于选择表明模型的显示方式，目前提供了模型的三种显示方式：面显示，线框显示和点显示。通过点击三个单选按钮之一选择对应的显示方式，如图 3.4 所示。

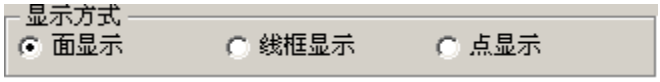


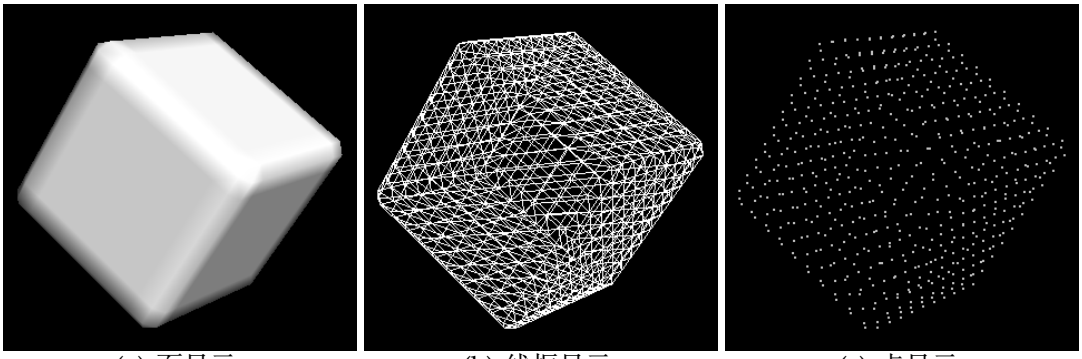
图 3.4　选择表面模型的显示方式

各种显示方式下的面绘制结果如图 3.5 所示。这里采用一个简单的立方体模型以清楚地表示出各种显示方式的不同。其中，面显示是模型表面的默认显示方式，可以把它当作面绘制的最终结果；在线框显示方式下，构成表面模型的所有三角面片的边框都会在场景中显现出来；在点显示方式下，表面模型的所有顶点会在场景中显示出来（图中显示的结果顶点大小取1.5）。

3.4 选择面显示的插值方式

在面显示的绘制方式下，有两种插值方式可供选择：一般方式和平滑方式，通过“插值方式”组的单选按钮进行选择，如图 3.6 所示。

一般插值方式使用了一种简单的插值方法进行面绘制，和平滑方式相比，其结果比较粗糙但绘制的过程会快一点；平滑插值方式使用了一种复杂的插值方法来进行表面绘制，其结果相对平滑但速度较慢。两种方式的显示结果如图 3.7 所示。



(a) 面显示 (b) 线框显示 (c) 点显示

图 3.5 不同显示方式下的绘制结果

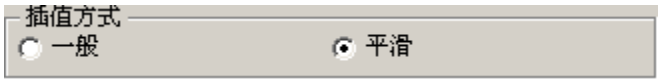


图 3.6 选择面显示的插值方式

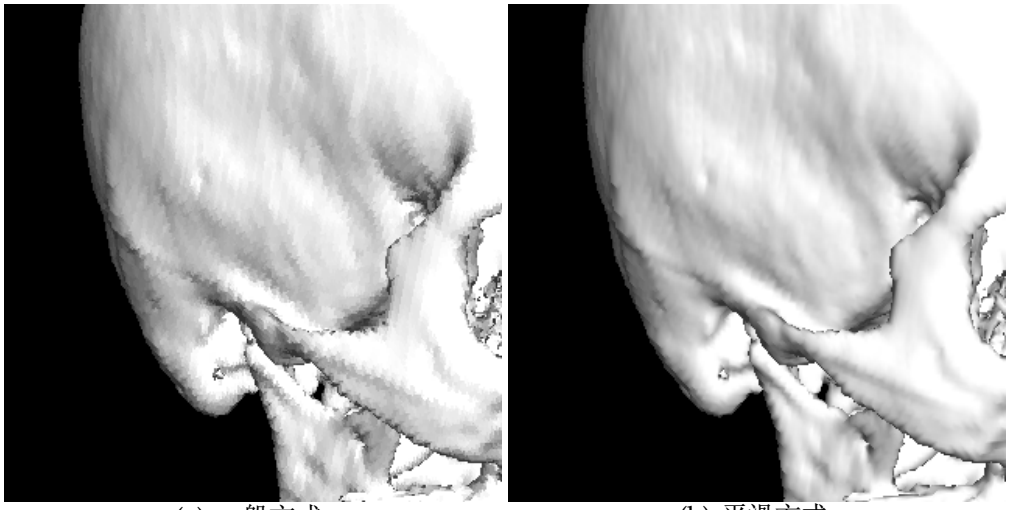
3.5 调整表面材质

“表面材质属性”组用于调节绘制模型的表面材质参数，如图 3.8 所示。材质属性包括：环境颜色、散射颜色、镜面反射颜色和强度、表面放射光颜色等，分别对应 OpenGL 中的相关概念。

在这个区域里，可以自由调整表面模型的材质属性（颜色），调节方法如下：

1. 用鼠标点击对应单选按钮选择要调节的材质属性；
2. 用下面 4 个滑动条分别调节材质属性颜色的 RGB 和 Alpha 分量，释放滑动条后更改生效；
3. 或者在滑动条右边的编辑框内直接用键盘输入分量值（取值范围在 0.0̃1.0 之间），按“Enter”键使改动生效；
4. 或者单击“预设值”左边的颜色按钮，直接在弹出的如图 3.9 所示的颜色对话框中选择所需颜色，该颜色按钮会显示当前选定的颜色。

在四种材质属性中，只有散射颜色的 Alpha 分量是有实际意义的，它对应于表面的不透明度，0.0 表示完全透明，1.0 表示完全不透明。同时，当选择调



(a) 一般方式 (b) 平滑方式

图 3.7 不同插值方式下的面显示结果

节散射颜色时，Alpha 分量的滑动条和下面的“不透明度”滑动条是联动的，两者调节的实际上是同一个参数。

“镜面反射强度”调节的是镜面反射高光区域的大小和亮度，范围为 [0.0, 128.0]，其数值越大，高亮区越小、亮度越高。该参数可以通过滑动条调节，释放滑动条使更改生效；也可通过键盘直接在右边的编辑框输入，输入完成按“Enter”键使改动生效。

当显示方式是线框显示时，点击上部“边颜色”单选按钮，然后可以像调节材质属性颜色一样调节边框颜色；同时，通过下部的“线宽”滑动条和编辑框可以调节边框的粗细。

当显示方式是点显示时，可以通过底部的“顶点大小”滑动条和编辑框调节显示顶点的尺寸。

另外，“表面材质属性”面板顶部“预设值”右边的下拉框提供一些预设的材质属性，包括 Default、Bone、Skin 和 Muscle，通过它可以选择一些预定义的材质属性，简化调节步骤。

如图 3.10 所示是相同的表面模型在不同的材质属性下的绘制结果。

提示：表面模型最终展现的颜色很大程度地受到散射光的影响，它取决于入射光中的散射光颜色和入射光相对于表面法线的角度。同时，环境光反射也会影响到最终表面模型的绘制颜色。因此调节表面模型的材质属性主要就是调

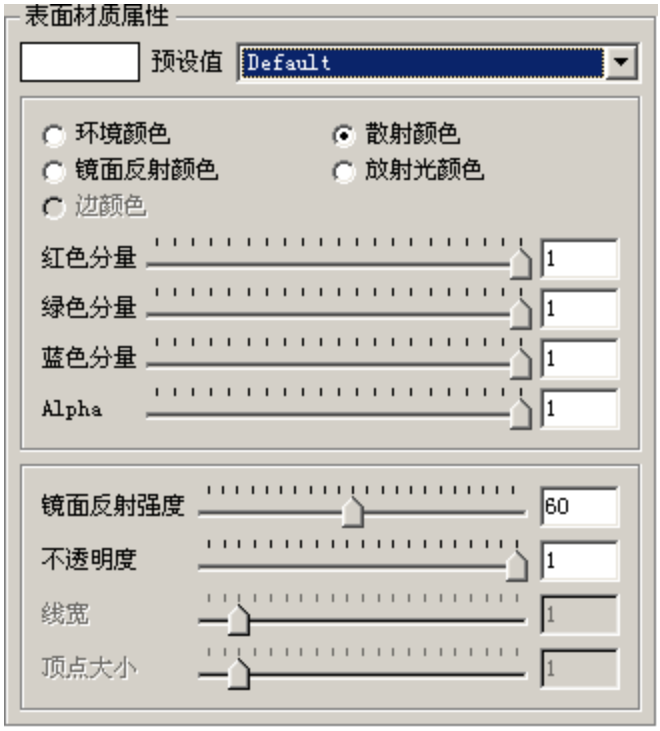


图 3.8　表面材质调节面板

节散射颜色和环境颜色，二者通常取相同的值。

3.6 背景颜色设置

单击面板底部的“背景颜色”按钮可在弹出的颜色选择对话框中选择三维视图的背景颜色，同时该按钮的底色会显示所选择的颜色。

3.7 顶点法线反向

面板底部的“顶点法线反向”按钮可用于修正错误的顶点法向。有时重建算法计算得出的方向可能与正确的方向相反，其结果显示异常。这时点击此按钮就能得到合适的显示结果，如图 3.11 所示。

3.8 同时显示多个表面模型

三维视图支持同时显示多个表面模型，当前在三维视图中显示的表面模型名称列于“文档”面板下部“在三维场景中显示的内容”列表中的“三维模型”子列表中，可通过“文档”面板中部的 铵钮将“加载的数据”列表

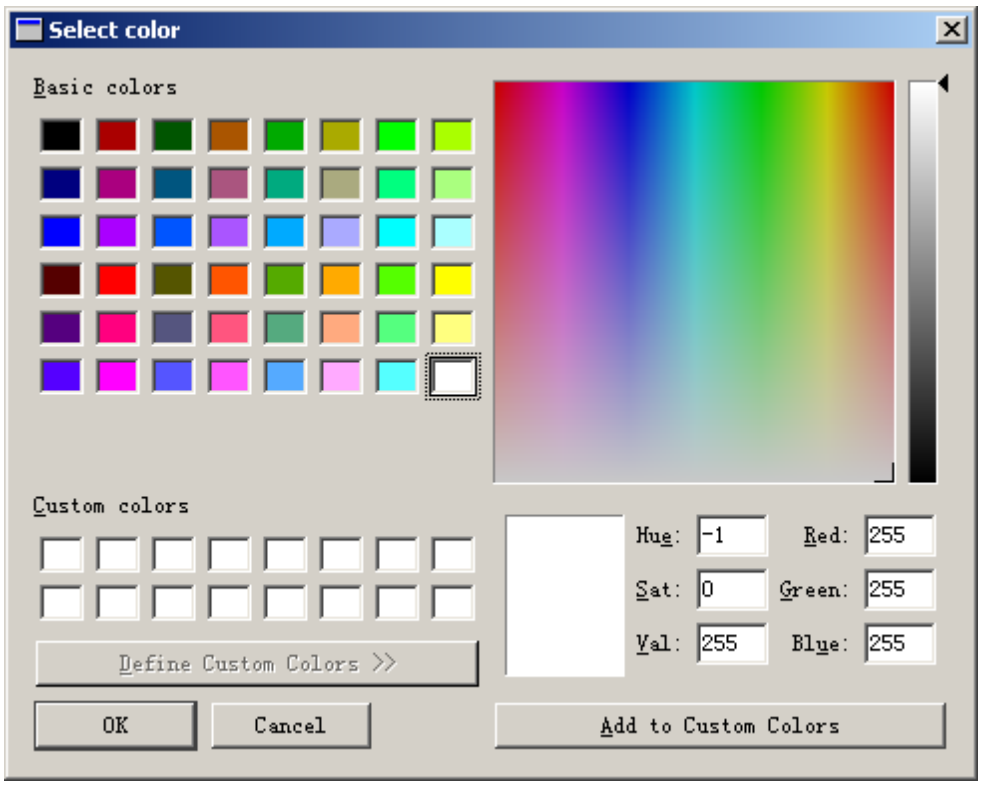
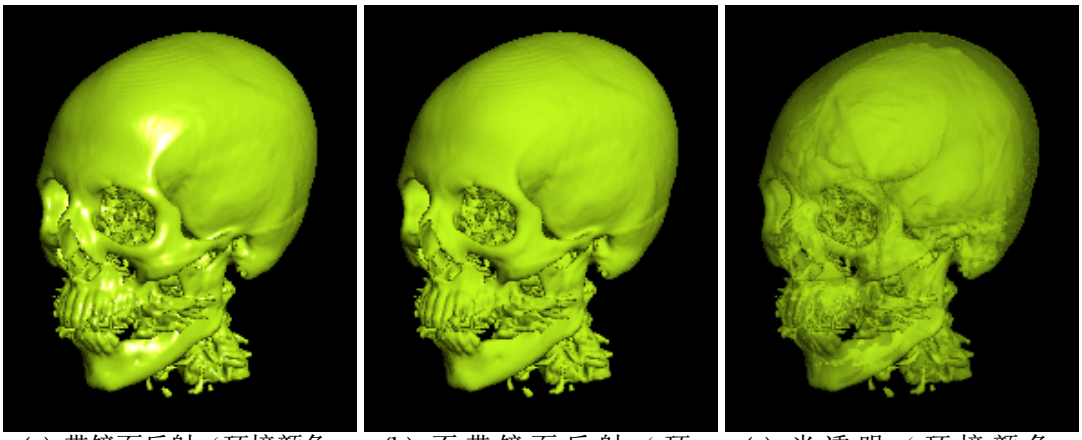


图 3.9　颜色选择对话框

中当前选中的三维模型添加到三维视图中进行绘制，通过“文档”面板底部的 铵钮将“在三维场景中显示的内容”列表中当前选中的三维模型从三维视图中移除（并不从内存中删除该三维模型），如图 3.12 所示。

图 3.13 给出一个多表面叠加显示的例子，皮肤和骨骼的表面模型是分别采用不同阈值对同一体数据进行分割、重建所得的结果。

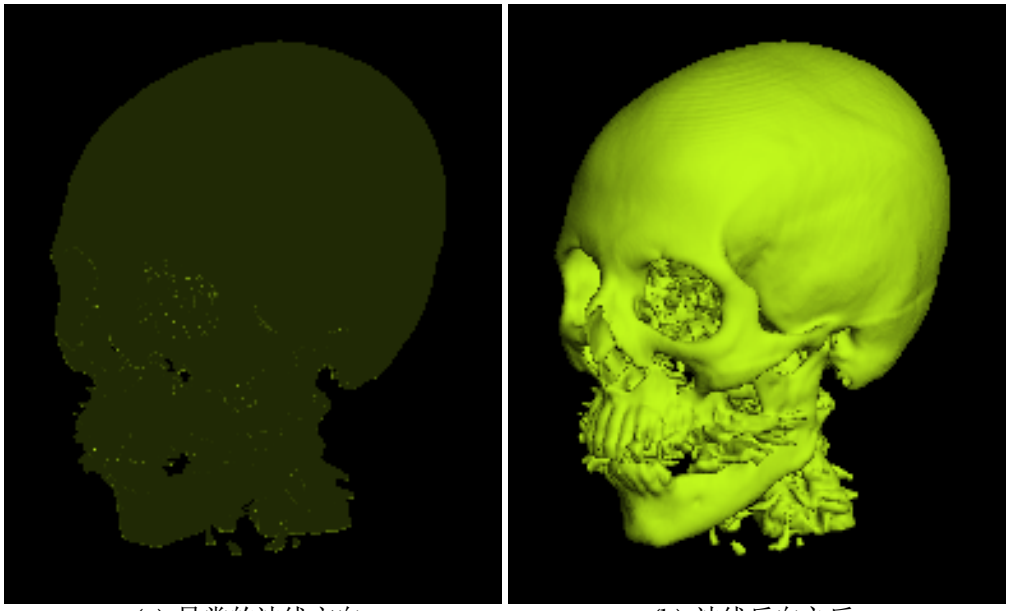


(a) 带镜面反射（环境颜色（0.63, 0.81, 0.09, 1.00），散射颜色（0.63, 0.81, 0.09, 1.00），反射颜色（1.0, 1.0, 1.0, 1.0），反射强度 91）

(b) 不带镜面反射（环境颜色（0.63, 0.81, 0.09, 1.00），散射颜色（0.63, 0.81, 0.09, 1.00），反射颜色（0.0, 0.0, 0.0, 1.0））

(c) 半透明（环境颜色（0.63, 0.81, 0.09, 1.00），散射颜色（0.63, 0.81, 0.09, 0.50），反射颜色（0.0, 0.0, 0.0, 1.0））

图 3.10　不同材质属性的绘制结果（放射光颜色均为（0.0, 0.0, 0.0, 1.0））



(a) 异常的法线方向　　(b) 法线反向之后

图 3.11　用“顶点法线反向”按钮修正错误的顶点法线方向

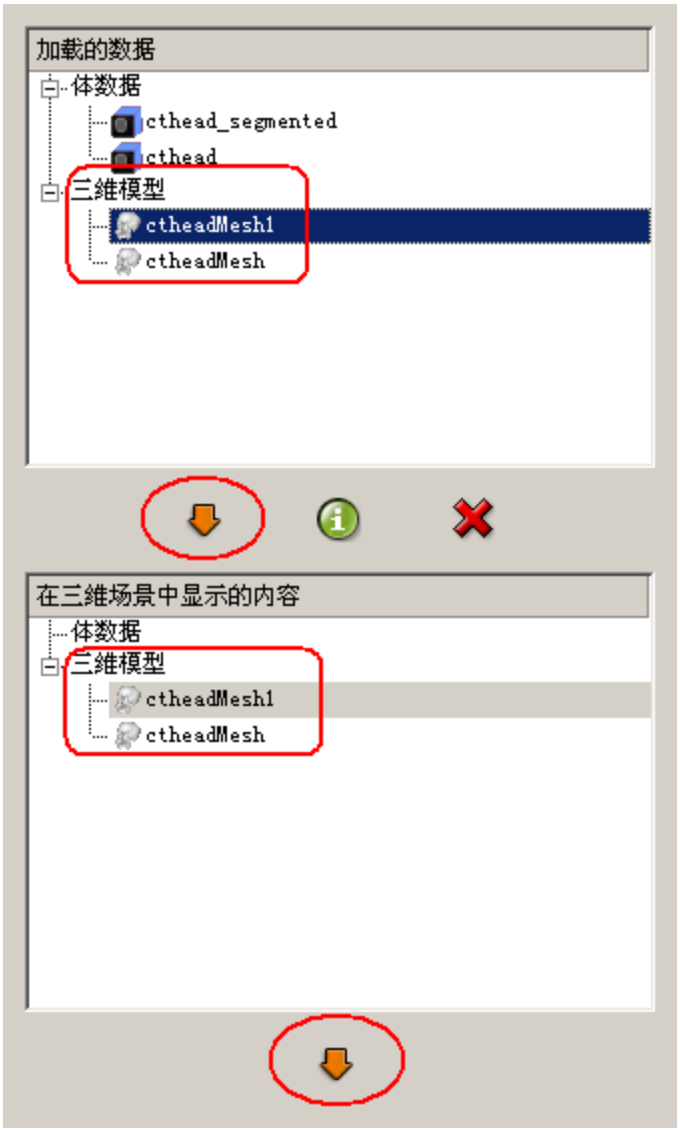
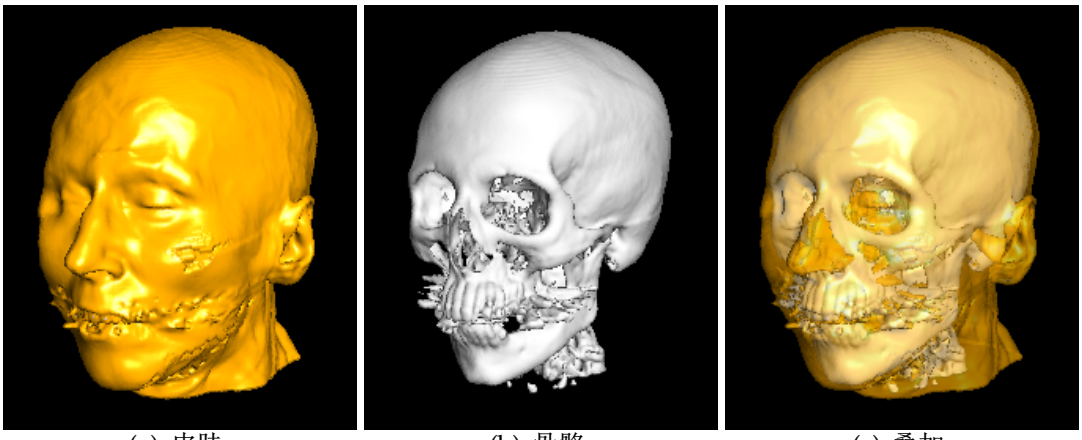


图 3.12　在“文档”面板对绘制的模型进行管理



(a) 皮肤 (b) 骨骼 (c) 叠加

图 3.13　多个表面模型叠加显示，皮肤采用半透明绘制

第 4 章

三维测量和切片重组

“三维测量”面板提供了三维测量和切片重组等功能，目前该面板的功能仅对面绘制有效，仅当三维视图中有表面模型被绘制时，该面板上的控件才是可用的，如图 4.1 所示。

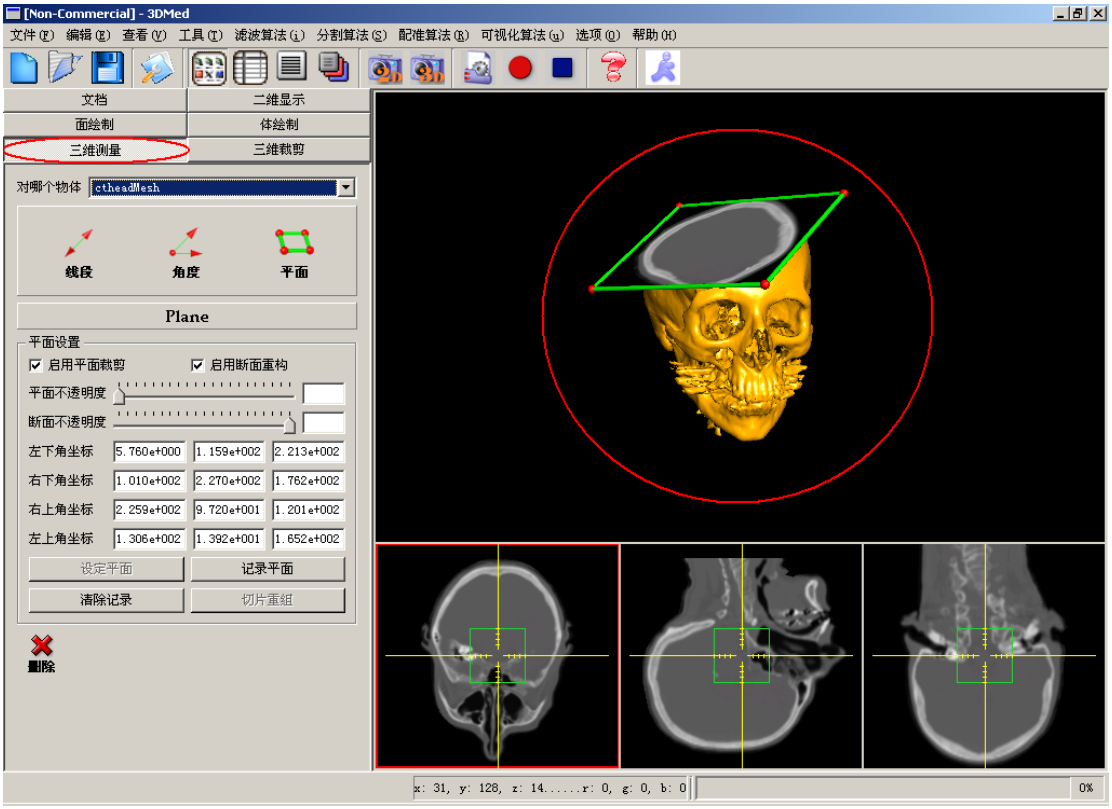


图 4.1　三维测量和切片重组

其中，“平面设置”组只有当视图中有“平面”控件并且处于选中状态时可用。

4.1 选择测量对象

“三维测量”面板上部标有“对哪个物体”的下拉框用于选择测量对象，里面所列为当前显示在三维视图中的表面模型，如图 4.2 所示。

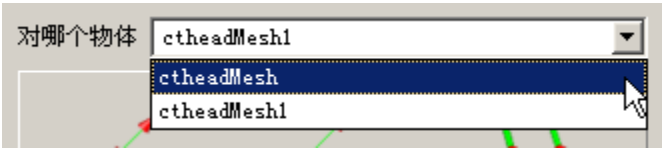


图 4.2　选择测量对象

4.2 测量长度

“三维测量”面板上的“线段”控件用于测量对象空间中任意两点之间距离的测量。鼠标单击“线段”按钮可在三维视图中添加一个“线段”控件，添加线段的初始状态是测量对象包围盒的对角线，如图 4.3 所示。

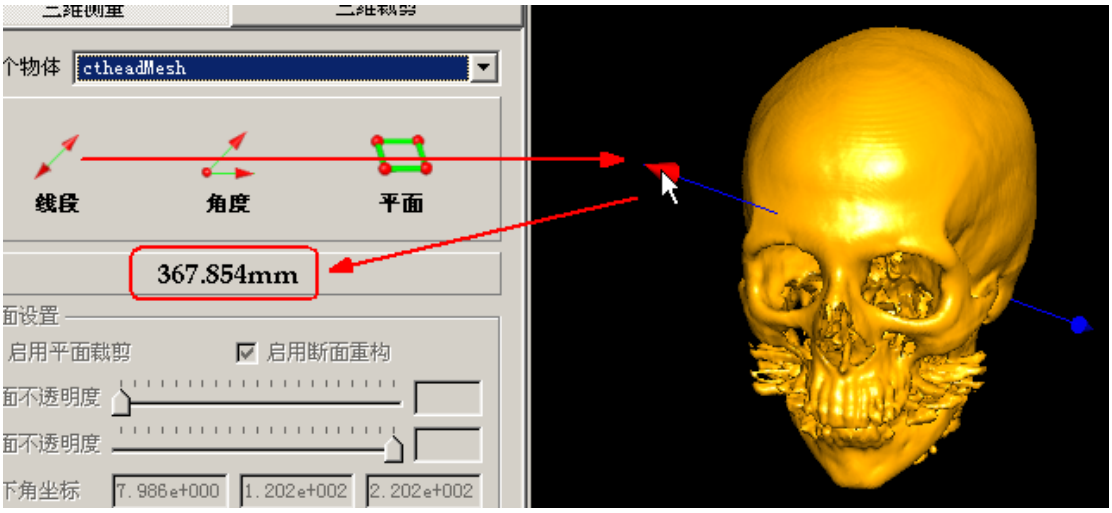


图 4.3　三维距离测量控件

在非选中状态下，整个控件呈蓝色，由一条线段和两端两个锥形箭头组成。用鼠标左键单击线段的任一部位可使线段处于选中状态。当点击的是线段中部直线部分时，直线部分呈绿色，两端箭头呈红色，这时保持鼠标左键按下并拖动鼠标可以使整条线段随鼠标平移；当点击的是箭头时，点中的箭头呈红色，其余部分呈蓝色，这时保持鼠标左键按下并拖动鼠标可以使该箭头随鼠标

移动以改变端点位置。松开鼠标左键，在视图空白区域（没有控件的区域）单击可取消对控件的选择，这时控件恢复为蓝色。

在上述控制过程中，测量线段的当前长度将显示在左边面板的状态显示区，如图 4.3 中红框部分所示。

可点击“删除”按钮将当前选中的“线段”控件删除。

4.3 测量角度

“三维测量”面板上的“角度”控件用于测量对象空间中任意三维角度的测量。鼠标单击“角度”按钮可在三维视图中添加一个“角度”控件，添加角度的初始状态由测量对象包围盒的一条体对角线和相接的一条面对角线组成，如图 4.4 所示。

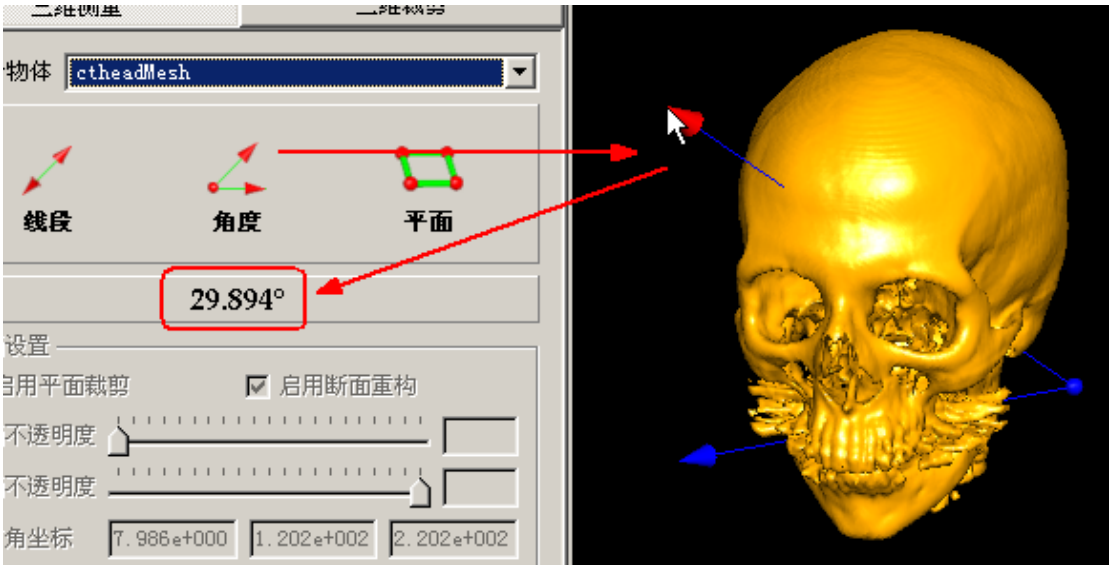


图 4.4 三维角度测量控件

在非选中状态下，整个控件呈蓝色，由两条边线段、一个球形端点和两个锥形箭头端点组成。用鼠标左键单击控件的任一部位可使线段处于选中状态。当点击的是线段部分时，直线部分呈绿色，球形端点和锥形箭头端点均呈红色，这时保持鼠标左键按下并拖动鼠标可以使整个控件随鼠标平移；当点击的是球形端点或锥形箭头端点时，点中的端点呈红色，其余部分呈蓝色，这时保持鼠标左键按下并拖动鼠标可以使该端点随鼠标移动以改变端点位置。松开鼠标左键，在视图空白区域（没有控件的区域）单击可取消对控件的选择，这时

控件恢复为蓝色。

在上述控制过程中，当前的测量角度将显示在左边面板的状态显示区，如图 4.4 中红框部分所示。

可点击“删除”按钮将当前选中的“角度”控件删除。

4.4 切片重组

“三维测量”面板上的“平面”控件可用于对三维表面模型的显示进行裁减及断面的重组。鼠标单击“平面”按钮可在三维视图中添加一个“平面”控件，其初始位置在表面模型中部，法向为 (0, 0, 1)，即沿 z 轴正方向。

在非选中状态下，整个控件的边框（4 个细圆柱）和顶点（4 个小球）呈蓝色，矩形框中部显示的是蓝色半透明的裁减平面或者重构的断面（由启用的平面功能决定）。用鼠标任一键单击控件的任一部位可使控件处于选中状态。当点击的是边框或内部平面时，边框圆柱呈绿色，顶点圆球呈红色，这时，若是鼠标左键点中并按住拖动，可使矩形框随鼠标在平面内平移；若是鼠标中键点中并按住拖动，可改变矩形框的尺寸；若是鼠标右键点中并按住拖动，可使矩形框随鼠标在垂直于平面的方向上平移。当点击的是顶点上的小球时，点中的小球呈红色，其余边框和顶点呈蓝色，这时，若是鼠标左键点中并按住拖动，可使矩形框随鼠标在空间做任意旋转；鼠标其余二键同上。“平面”控件的使用状态如图 4.5 所示。

在上述控制过程中，当前平面矩形的尺寸即 4 个顶点的坐标将显示在左边面板的状态显示区，如图 4.5 中红框部分显示。此外，如果断面重构的功能启用，一般情况下，在平面调节结束后才进行断面重构并显示断面，但在鼠标操作过程中如果按住键盘 Ctrl 键可使重构的断面随平面改变而即时更新，然而由于计算量较大，可能产生控制不连贯的情况。

“平面”控件的一些功能和参数设定都集中在“平面设置”组，如图 4.6 所示。

单击“启用平面裁减”选择按钮可启用平面裁减功能，这时三维视图中在沿平面法向一边的表面模型被保留，其余部分将被裁减掉（不显示），在没有启用断面重构的情况下，裁减平面呈蓝色半透明状态，其不透明度可由下方对应的滑动条和编辑框设定，取值范围为 [0.0, 1.0]，如图 4.7 所示。

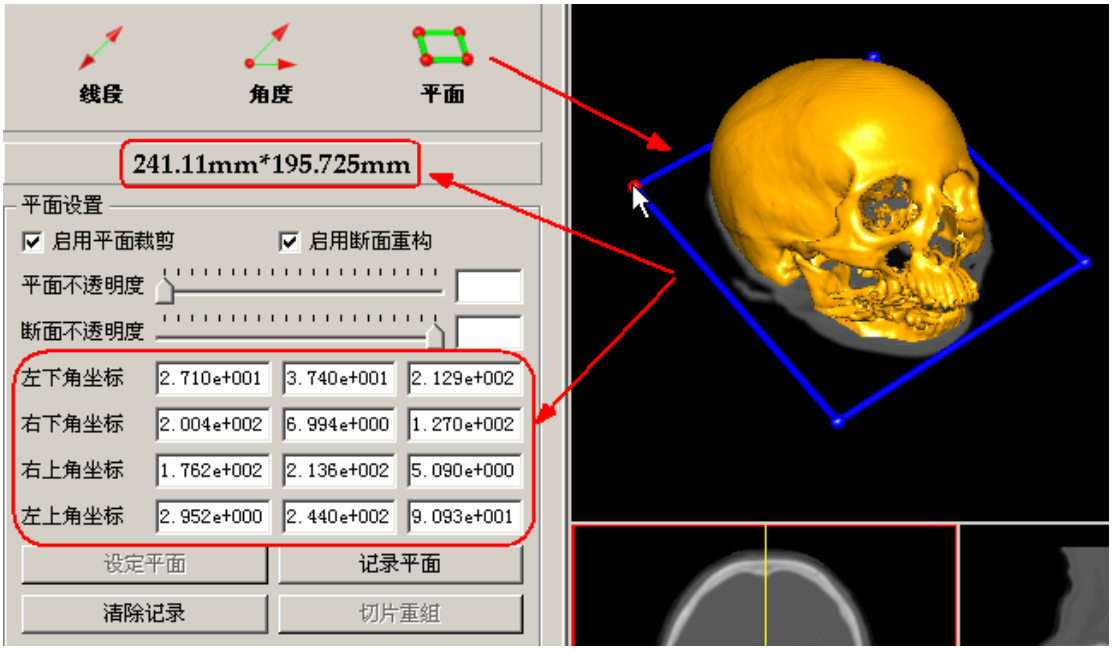


图 4.5 三维模型裁减和切片重组控件

单击“启用断面重构”选择按钮可启用断面重构功能，这时裁减平面上将显示根据原始体数据重构的断面图像。断面图像的不透明度同样可以通过下方对应的滑动条和编辑框设定，取值范围为 [0.0, 1.0]，如图 4.8 所示。

一般情况下，当启用断面重构时，平面裁减也是同时启用的，否则重构的断面图像可能被表面模型阻挡而观察不到。

“平面设置”组中部的 4 组编辑框显示了平面矩形 4 个顶点的当前坐标，同时其中的数值可由键盘重新设定，然后按“设定平面”按钮使更改生效，但需注意输入的 4 组顶点坐标必须构成三维空间中的一个矩形，否则程序会报错，并且可能导致平面显示不正常。

余下的三个按钮可以完成对原始体数据进行切片重组的功能。进行切片重组，首先需要输入切片重组的范围，该范围由两个平面指定：起始平面和终止平面；其次需要输入重组切片的参数：重组切片组的尺寸（包括宽、高和重组切片数，以像素为单位）或三个轴向的像素间距。这些参数可通过单击“切片重组”按钮弹出的对话框输入，如图 4.9 所示。

其中，起始平面和终止平面的位置一般由“平面”控件通过鼠标交互操作设定，如果直接输入的话必须保证输入的 4 组顶点坐标构成三维空间中的一个矩形，所以一般不推荐直接输入。切片重组的范

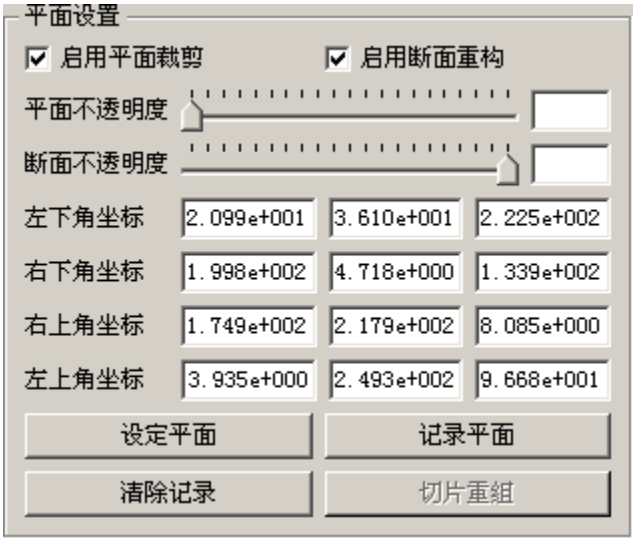


图 4.6 平面设置

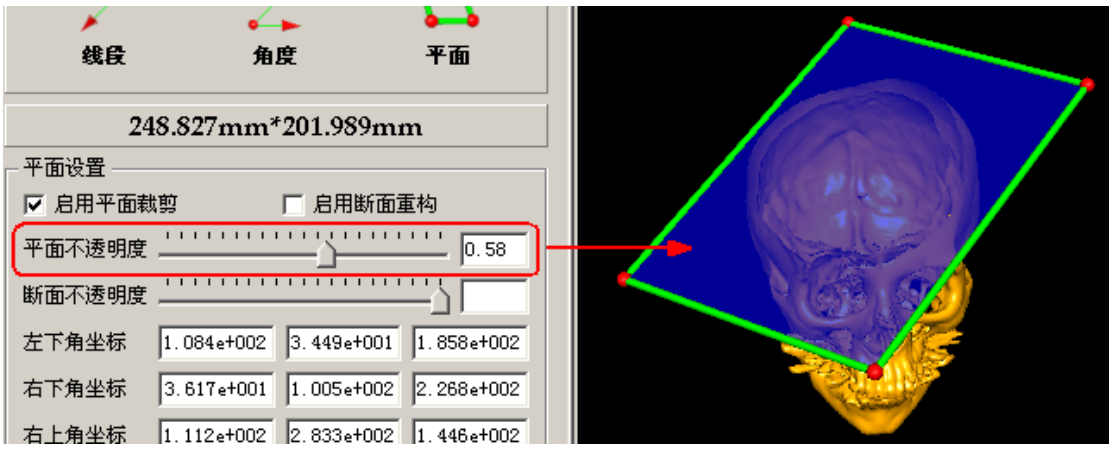


图 4.7 平面裁减

围确定了重组切片的物理区域，这时如果指定重组切片的尺寸，就可以根据物理区域和切片尺寸算出像素间距；同样，知道像素间距也可以算出切片尺寸。比如以单张切片来说，若切片断面矩形尺寸为 100mm×100mm，如果指定重组切片的尺寸为 200×100（以像素为单位），则重组的切片像素间距为 100mm/200×100mm/100=0.5mm×1.0mm；如果指定重组切片像素间距为 1.0mm×0.5mm，则重组切片的尺寸为 100mm/1.0mm×100mm/0.5mm=100×200。因此，对话框中下部“切片参数设置”中只需设定一组参数，如果“优先考虑像素间距”被选中，则下面一行编辑框被激活，可输入三个轴向的像素间距；否则，上面一行编辑框被激

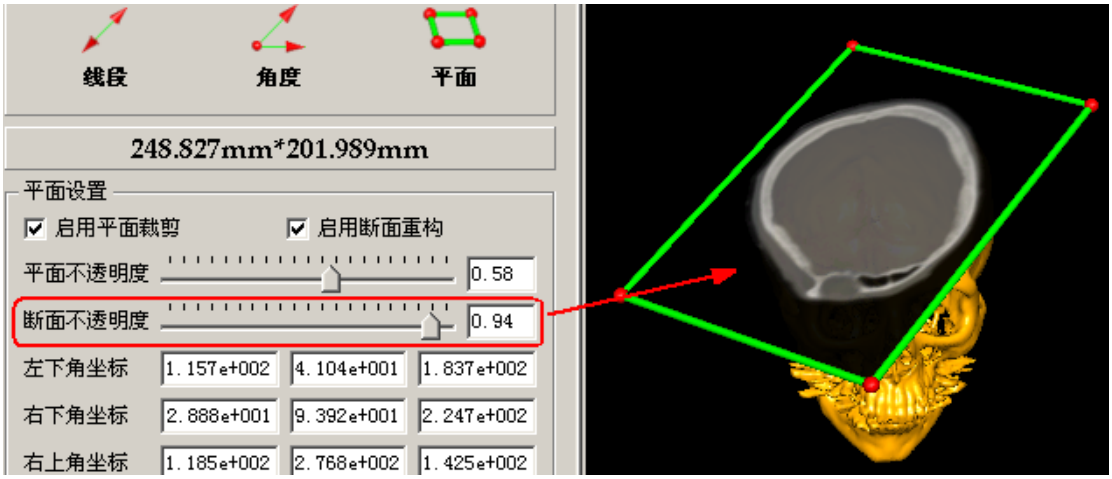


图 4.8　断面重构

活，可输入重组切片的尺寸。

最后，切片重组的步骤如下：

1. 根据上面所说的操作方法调节“平面”控件到所需位置和角度；
2. 单击“记录平面”按钮，记录切片重组的起始平面，若记录成功，“平面设置”组上方的状态显示区将显示“Start plane recorded!”；
3. 用鼠标右键点击平面上的小球控制点，然后将平面沿垂直于平面方向移动到新的位置（切片重组的终止平面位置）；
4. 再次单击“记录平面”按钮，记录切片重组的终止平面，若记录成功，状态显示区将显示“Stop plane recorded!”；
5. “切片重组”按钮这时将被激活，单击该按钮在弹出的对话框中输入重组参数，按“确定”按钮开始切片重组。

重组后的切片组够成一个新的体数据，显示在二维视图区，同时加入“文档”面板上部的“加载的数据”-“体数据”列表中，其名称以“_resliced”为后缀，如图 4.10 所示，对其可以进行与其他体数据一样的操作。

注意：上述断面重构和切片重组功能的使用有一个前提，就是“平面”控件所操纵的表面模型必须是通过本次 3DMed 重建过程直接得来，而非从 PLY 文件导入以前重建好的表面模型，否则，断面重构和切片重组的所有功能都将因为找不到表面模型所对应的原始体数据而全部失效。

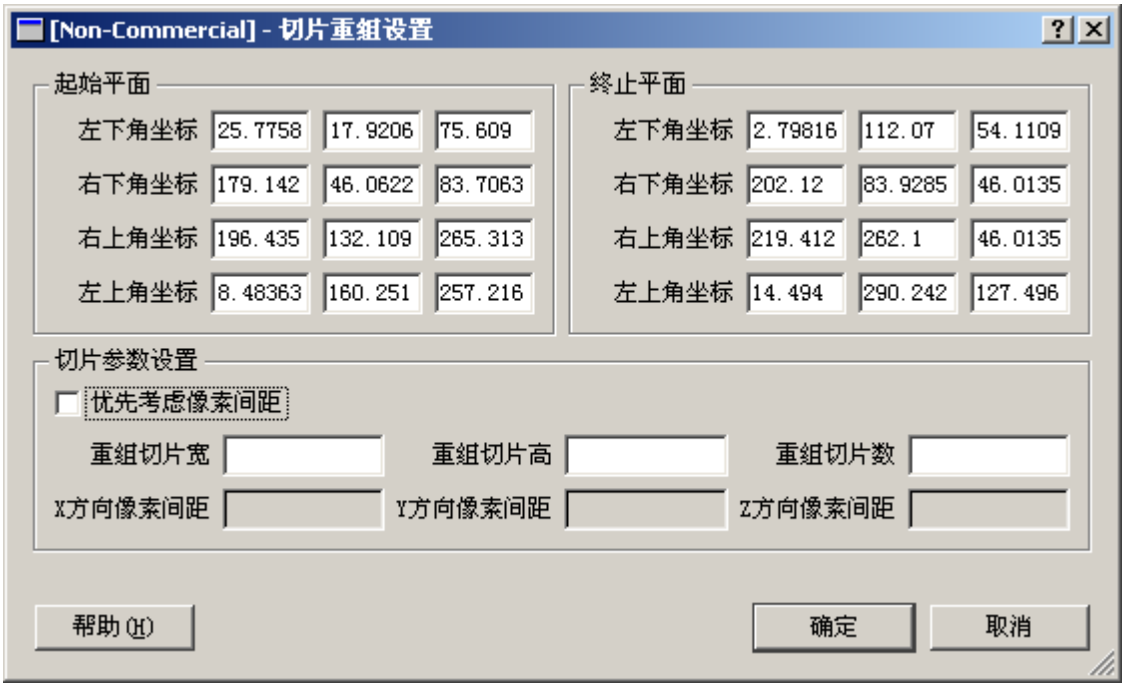


图 4.9 切片重组参数设定

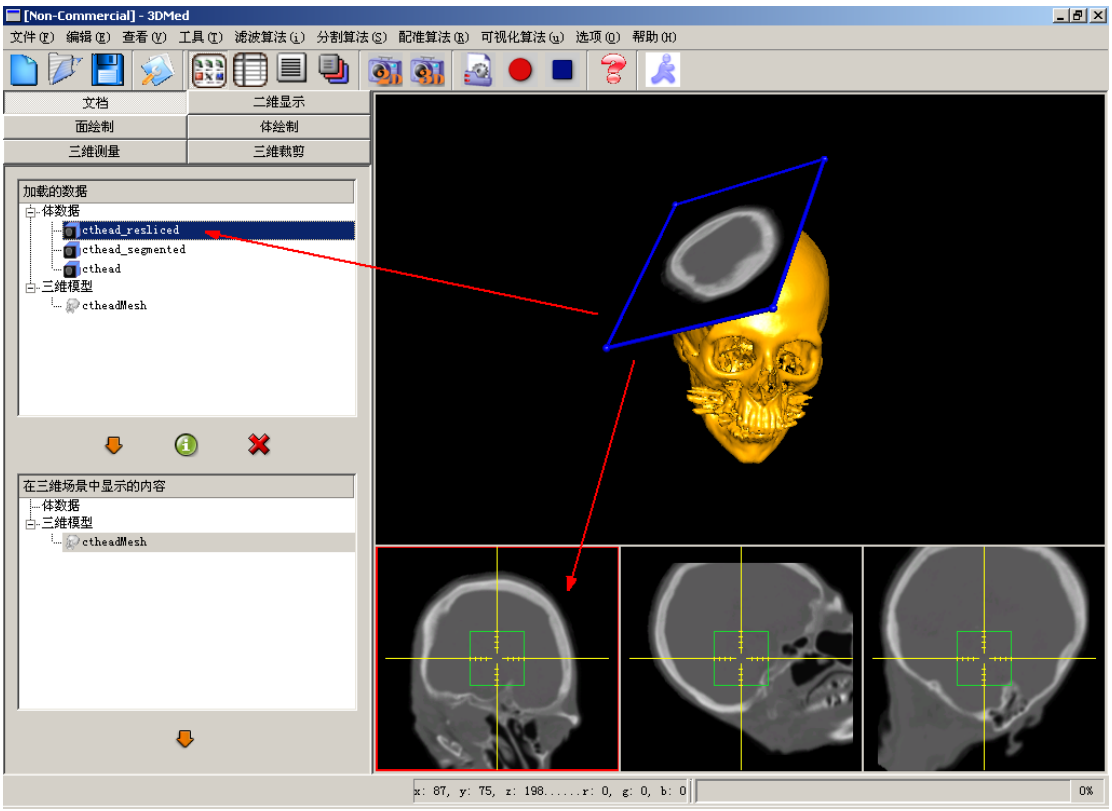


图 4.10 重组后的切片组

第 5 章

体绘制

体绘制是一个直接对体数据进行绘制的过程。当体数据载入 3DMed 后，可通过“文档”面板中部的  按钮对上部“加载的数据”列表中选中的体数据进行体绘制，同时该体数据名称将出现在下部“在三维场景中显示的内容”列表中；通过“文档”面板底部的  按钮可将“在三维场景中显示的内容”列表中当前选中的体数据从三维视图中移除（即不进行绘制），如图 5.1 所示。

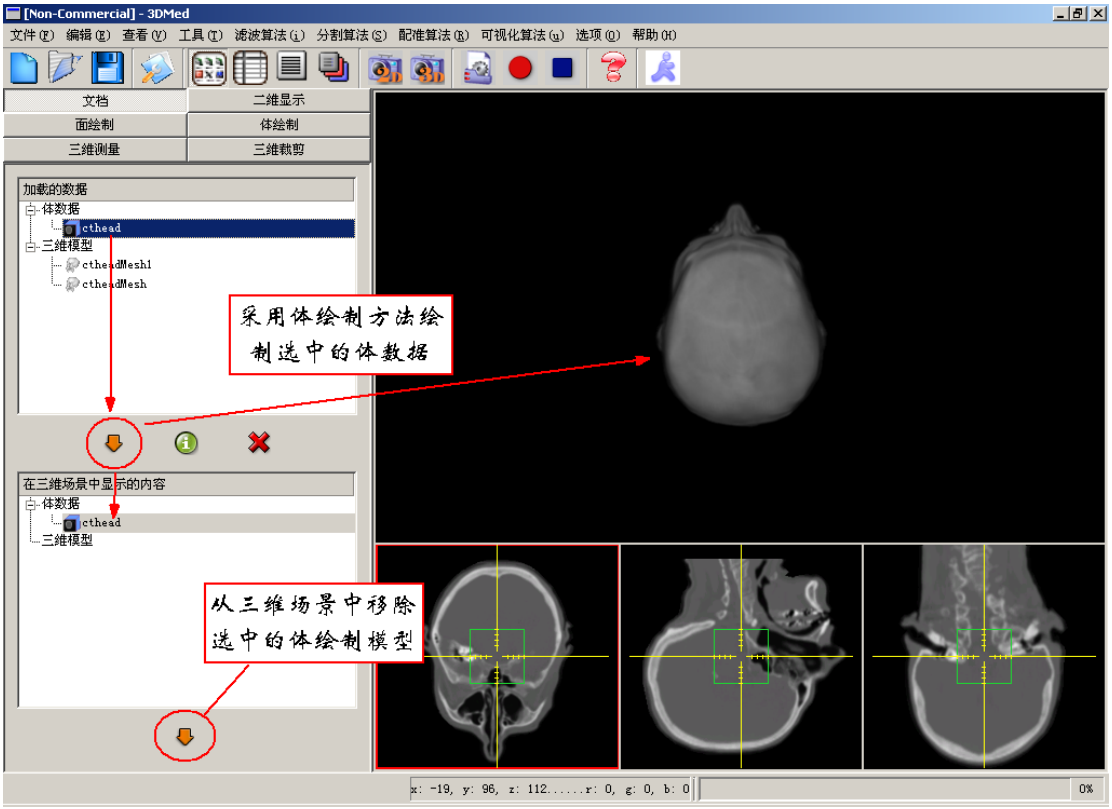


图 5.1　开始和结束体绘制

注意：和面绘制不同，体绘制不能同时绘制多个体数据。添加新的体数据到三维视图中时将把原先在三维视图中绘制的体数据移除。

然后，点击功能按钮区的“体绘制”可以激活体绘制控制面板，在该面板内调节体绘制的各项参数，如图 5.2 所示。

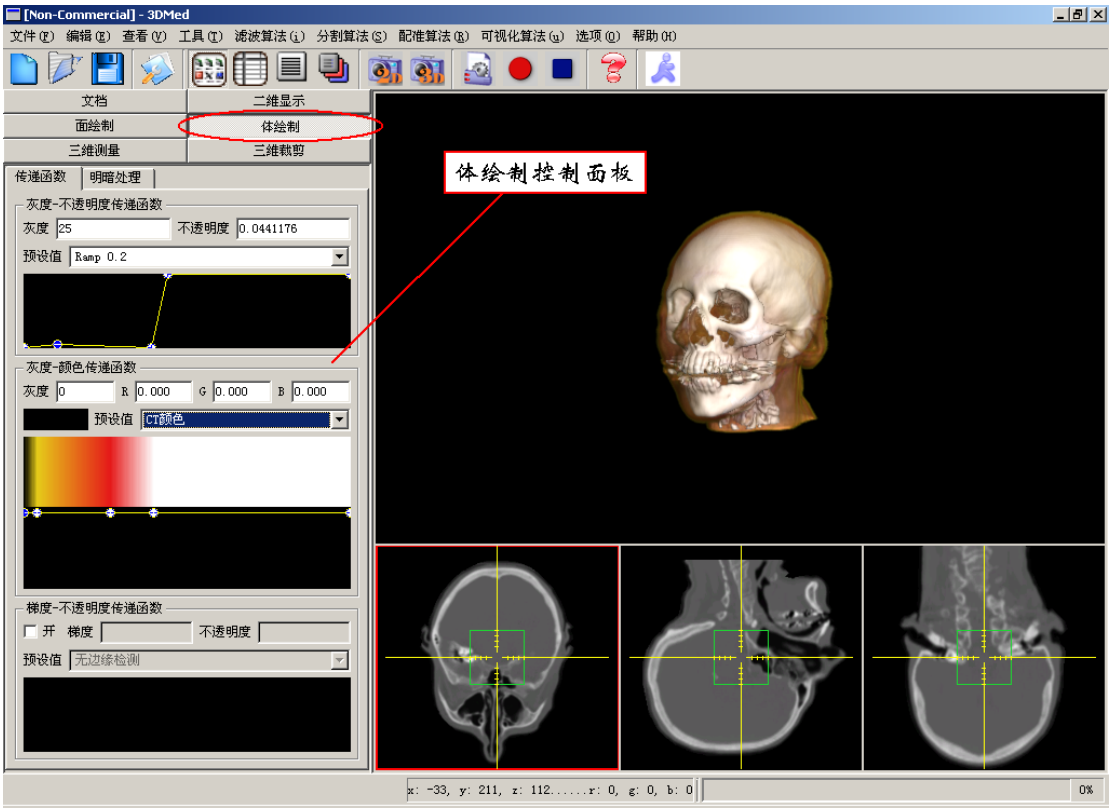


图 5.2　体绘制控制面板

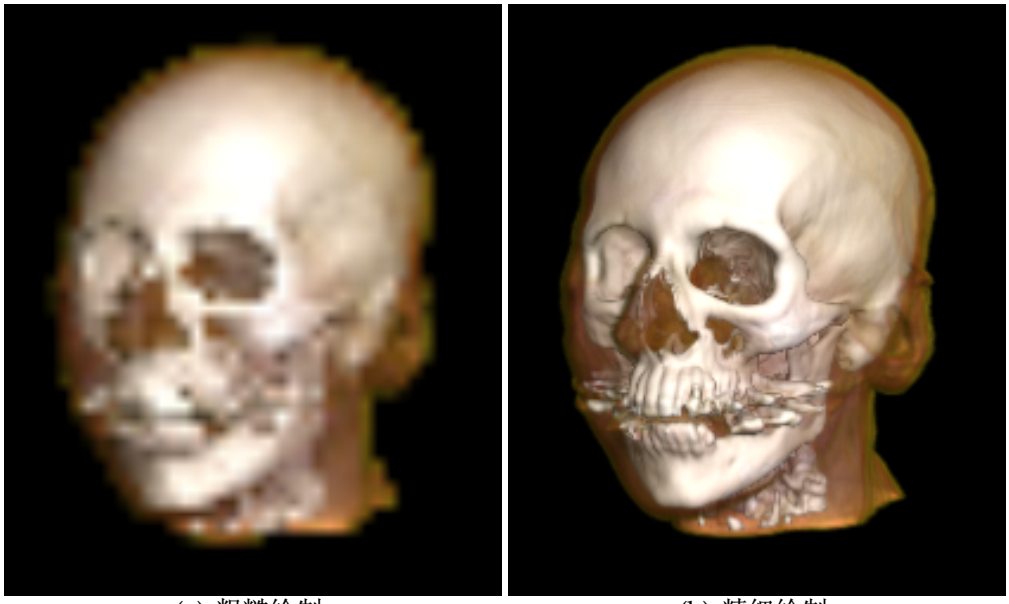
5.1 旋转、平移和缩放的鼠标操作

旋转 在三维视图区域内按住鼠标左键拖动；

平移 在三维视图区域内按住鼠标中键拖动；

缩放 在三维视图区域内按住鼠标右键拖动。

由于体绘制计算量比较大，绘制速度要比面绘制慢得多，所以在响应鼠标消息期间将以一种粗糙的方式绘制体数据，以求得对鼠标操作的快速响应；当鼠标操作停止后才对体数据进行精细绘制，这时需要等待一段时间才能看到最终的绘制效果，如图 5.3 所示。



(a) 粗糙绘制 (b) 精细绘制

图 5.3 两种绘制方式

5.2 传递函数的调节

体绘制的最终效果很大程度上是由传递函数决定的，在 3DMed 中通过体绘制控制面板的“传递函数”标签页可对三类传递函数进行调节：灰度-不透明度（阻光度）传递函数、灰度-颜色传递函数和梯度-不透明度传递函数，如图 5.4 所示。

图 5.4 传递函数调节面板

5.2.1 灰度-不透明度传递函数

灰度-不透明度传递函数用于设定具有某灰度值的体素的不透明度（阻光度），该传递函数在“灰度-不透明度传递函数”组的黑色区域显示为一条折线，如图 5.5 所示，横坐标是灰度值，纵坐标是对应的不透明度，取值范围在 [0.0, 1.0] 内，该折线由显示为黄色圆圈的一些控制点控制，用鼠标左键点中控制点拖动可以移动控制点的位置，将其拖出黑色区域可将该点删除；在空白处单击鼠标左键可以在单击位置添加一个控制点；当前点中的控制点显示为蓝

色，其对应灰度值（横坐标）和不透明度（纵坐标）分别显示在上方两个编辑框内，用键盘在编辑框内可直接输入当前点中的控制点的坐标，按“Enter”键生效。此外，“预设值”下拉框提供一组预设值以方便用户调节。

图 5.5　调节灰度-不透明度传递函数

5.2.2 灰度-颜色传递函数

灰度-颜色传递函数用于设定具有某灰度值的体素的颜色，该传递函数在“灰度-颜色传递函数”组的黑色区域显示为一条直线，上面有一系列的控制点，其对应颜色显示在黑色区域上半部分的色带内，如图 5.6 。相邻控制点之间的颜色通过线性插值产生。用鼠标左键点中控制点拖动可以移动控制点的位置，将其拖出黑色区域可将该点删除；空白处单击鼠标左键可以在单击位置添加一个控制点；当前点中的控制点显示为蓝色，其对应的灰度值和颜色分别显示在上方几个编辑框内，可通过键盘输入改变其值，按“Enter”键生效；也可以按“预设值”左边的颜色选择按钮直接在颜色选择对话框中选择特定颜色。此外，“预设值”下拉框提供一组预设值以方便用户调节。

5.2.3 梯度-不透明度传递函数

梯度-不透明度传递函数用于设定具有某梯度值的体素的不透明度，其调节区域如图 5.7 所示。使用该传递函数可以增强边缘（高梯度）区域的显示效果，但是需要大量额外计算，从效率上考虑，缺省情况下并不打开该传递函数，可以通过单击左上角的选择按钮开启该传递函数。其调节方法同灰度-不透明度传递函数。

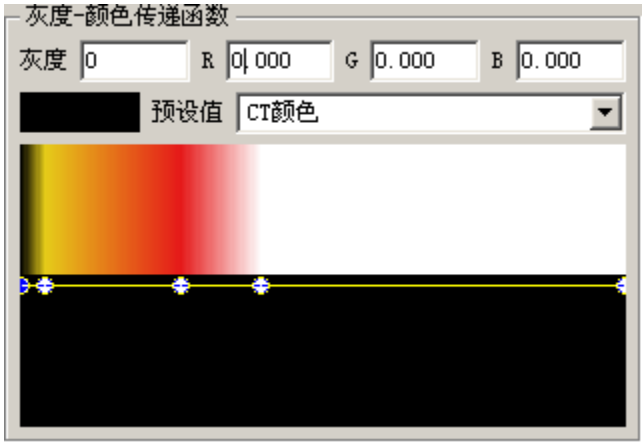


图 5.6 调节灰度-颜色传递函数

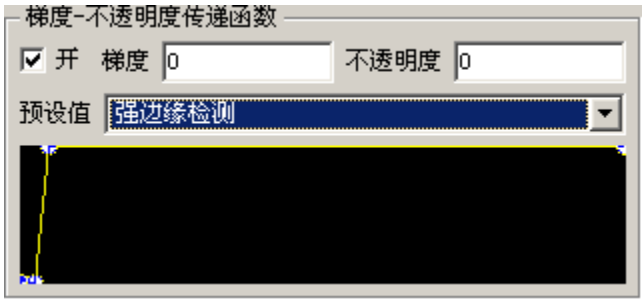


图 5.7 调节梯度-不透明度传递函数

5.3 明暗处理

明暗处理（Shading）可以有效地加强体绘制的立体感，在 3DMed 中通过体绘制控制面板的“明暗处理”标签页对其参数进行调节，如图 5.8 所示。由于其需要大量运算，所以缺省状态下未开启，可以通过点击该面板上部的选择按钮开启明暗处理。

明暗处理可调节的参数包括：环境光亮度、散射光亮度、反射光亮度及强度，通过面板上的一组滑动条进行调节。此外“预设值”下拉框提供了一组明暗处理参数的预设值，可简化调节步骤。

图 5.9 显示了开启和关闭明暗处理体绘制效果的不同。

5.4 体绘制的裁减

点击功能按钮区中的“三维裁剪”按钮，可以激活体绘制的三维裁剪控制

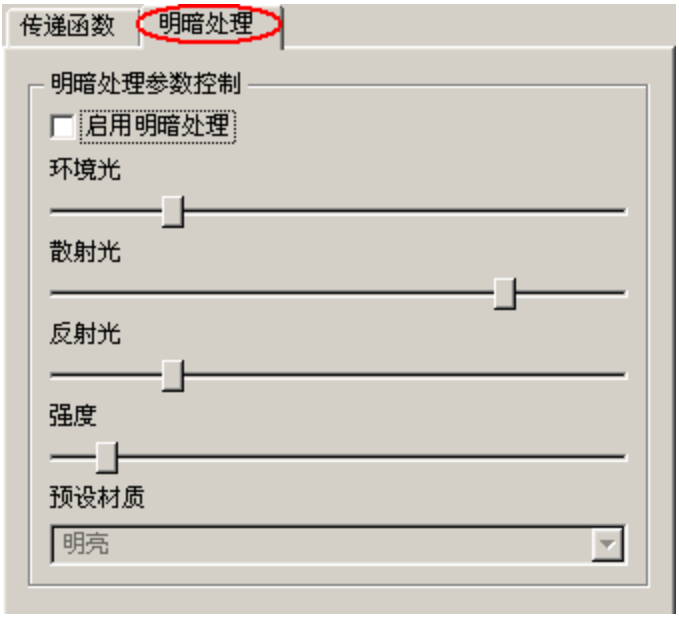


图 5.8 明暗处理调节面板

面板，如图 5.10 所示。

通过点击该面板上的“启用裁剪”选择按钮可开启裁剪功能，然后通过点击“平面裁剪”或“立方体裁剪”单选按钮可开启平面裁剪或立方体裁剪功能，也可以两个同时开启。

5.4.1 平面裁剪

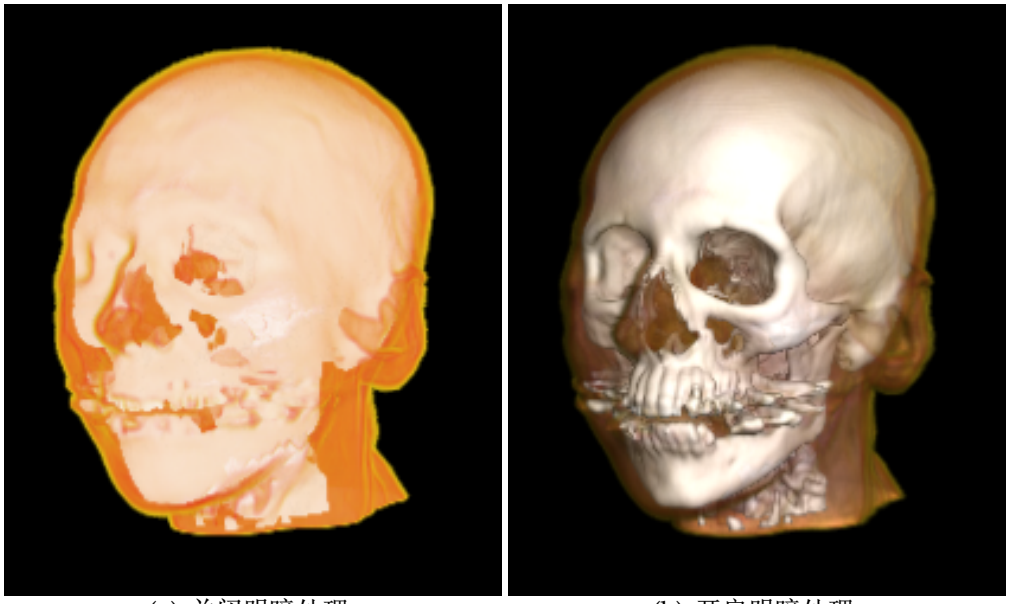
当开启了“平面裁剪”，您就可以点击“添加”按钮来添加裁剪平面。裁剪的初始方向可以在“初始方向”下拉框中加以选择。如图 5.11 所示。

可以连续添加多个裁剪平面，最多支持六个裁剪平面。添加裁剪平面后可通过“操纵裁剪平面”组内的控件来操纵选定的裁剪平面，如图 5.12 所示。首先在“哪个裁剪平面”下拉框中选择要操纵的平面，然后通过下面的一组滑动条调整裁剪平面的位置，单击“删除”按钮可删除选定的裁剪平面，单击“反向”按钮可使裁剪方向反向。

图 5.13 显示了一个 YoZ 平面裁剪的实例。

5.4.2 立方体裁剪

当开启了“立方体裁剪”选项后，可以通过该单选按钮右边的下拉框选择



(a) 关闭明暗处理 (b) 开启明暗处理

图 5.9 关闭和开启明暗处理进行体绘制的不同效果

采用“保留立方体”还是“挖除立方体”的方式进行裁剪。然后可通过“操纵裁剪立方体”组内的一组滑动条来调节裁剪立方体六个面的位置，如图 5.14 所示。

图 5.15 给出了保留立方体裁剪和挖除立方体裁剪的例子。

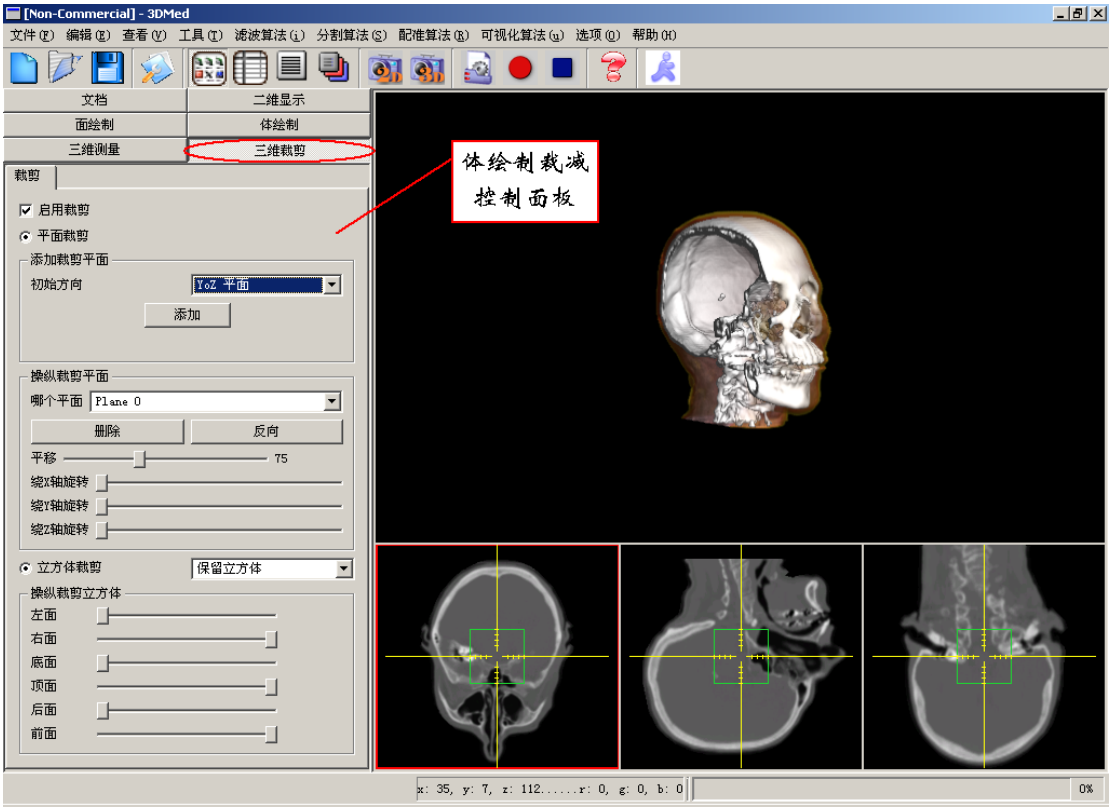


图 5.10 开启体绘制裁剪

图 5.11 开启体绘制裁剪

图 5.12 操纵裁剪平面

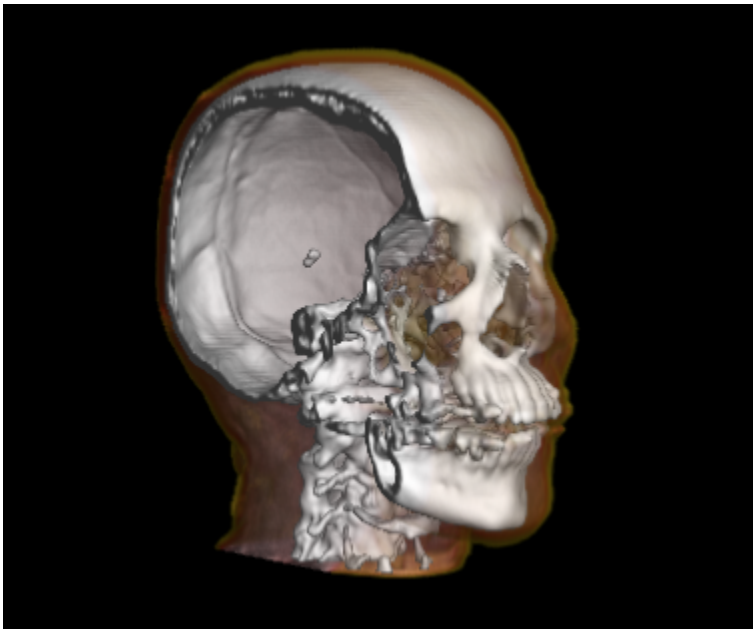


图 5.13 YoZ 平面裁剪

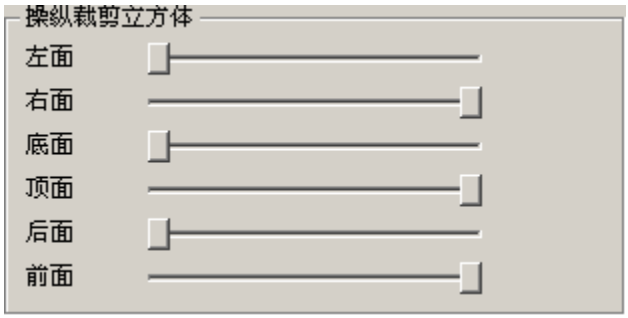
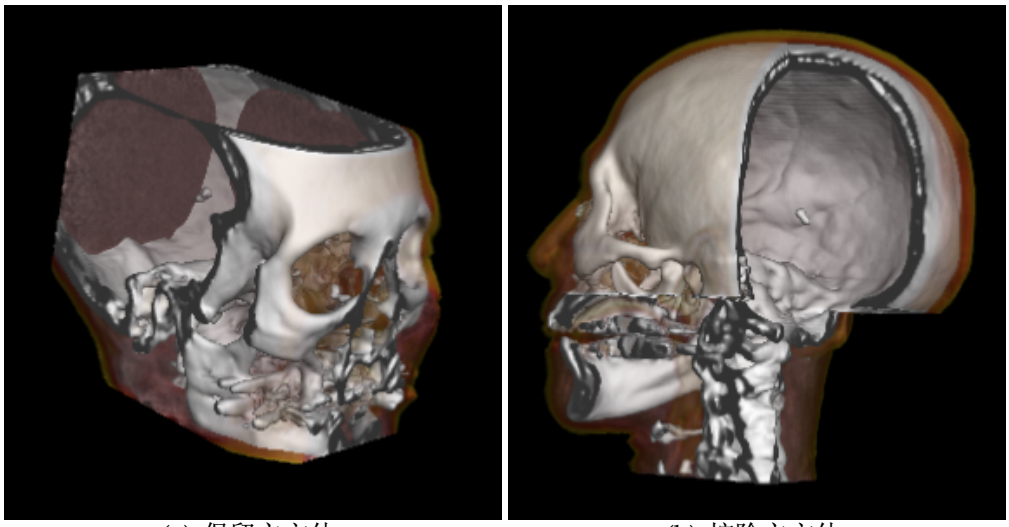


图 5.14 操纵裁剪立方体



(a) 保留立方体 (b) 挖除立方体

图 5.15 立方体裁剪

第 6 章

分割

加载完体数据后，您就可以点击主菜单中的“分割算法”选项开始分割。在“分割算法”下拉菜单中有许多选项，每个选项都各自代表了一种不同的分割方法，如图 6.1 所示。

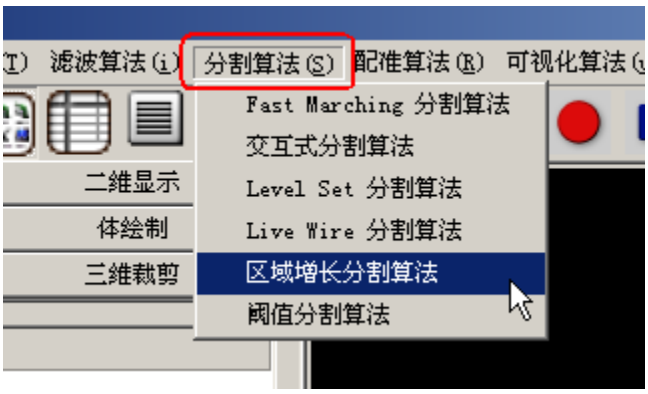


图 6.1　分割算法菜单

所有的分割算法都是通过插件的形式动态加载到 3DMed 中去的，每个算法插件对应一个动态链接库（DLL）文件，位于 3DMed 安装目录的 Plugins 子目录下。目前 3DMed 提供 6 种分割算法插件，包括阈值分割算法、区域增长分割算法、交互式分割算法、Live Wire 分割算法、Fast Marching 分割算法和 Level Set 分割算法。

这些分割算法插件的输出结果将添加到主界面的“体数据”列表中，其名称以“_segmented”为后缀，其切片图像显示在二维视图区域；与此同时还将对分割结果进行三维重建，重建结果加入“三维模型”列表并显示在主界面的三维视图区域。如图 6.2 所示。

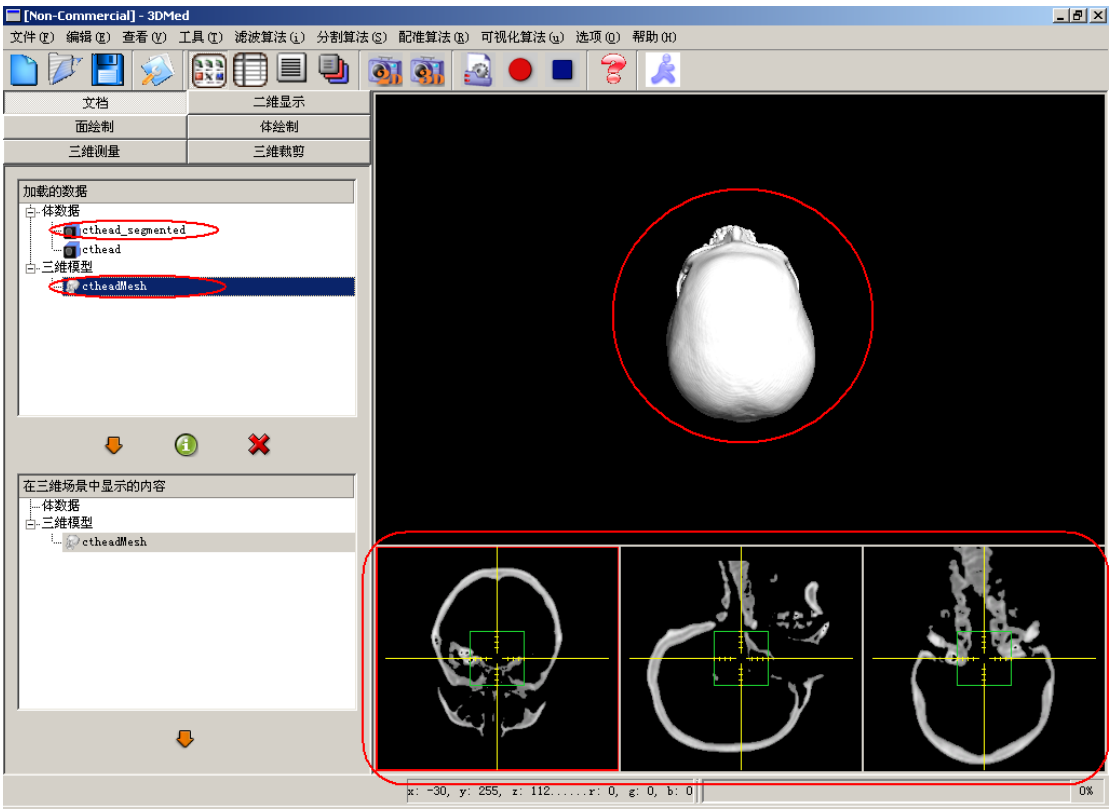


图 6.2 分割算法的输出结果及对其进行三维重建的结果

6.1 阈值分割

阈值分割是最常见的分割方法，其优点是简单，同时对于不同类的物体灰度值或其他特征值相差很大时，它能很有效的对图像进行分割。阈值分割通常作为预处理，在其后应用其他一系列分割方法进行处理，它常被用于 CT 图像中皮肤、骨骼的分割。其缺点是不适用于多通道图像和特征值相差不大的图像，对于图像中不存在明显的灰度差异或各物体的灰度值范围有较大重叠的图像分割问题难以得到准确的结果。另外，由于它仅仅考虑了图像的灰度信息而不考虑图像的空间信息，阈值分割对噪声和灰度不均匀很敏感。

点击“分割算法”菜单栏的“阈值分割算法”菜单项弹出如图 6.3 所示的对话框，阈值分割操作在此对话框中进行。

对话框的顶部显示两幅图像。左边的是“原图像”，显示了分割前的原始图像。右边的是“目标图像”，显示了分割后的结果。对话框的中部是“直方图”，显示了原始图像的灰度分布。对话框的底部“切片序号”组内的滑动条

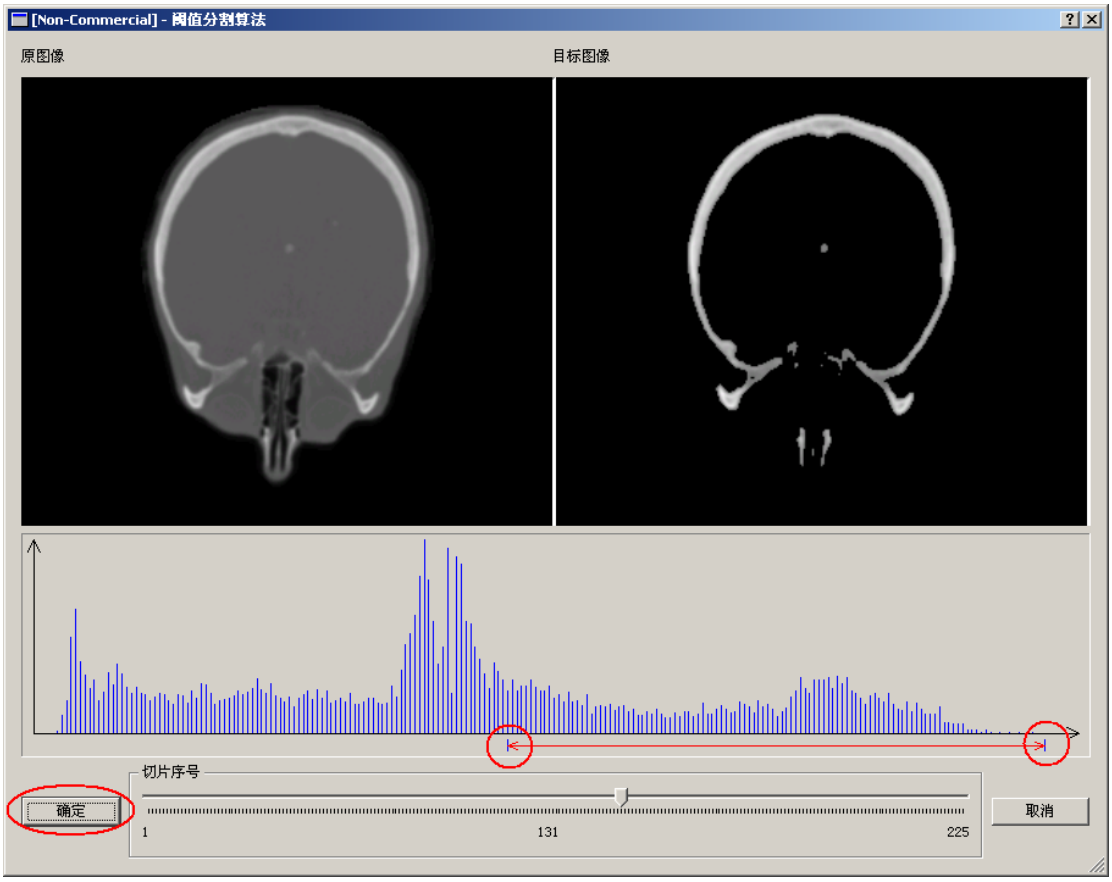


图 6.3　阈值分割对话框

可以用来改变当前当前显示的切片图像。

通过拖动“直方图”下方的红色箭头来选定分割阈值，左箭头设定低阈值，右箭头设定高阈值。当设定阈值之后，“目标图像”中就会显示出分割后的结果。

如果对分割结果满意，可单击“确定”按钮保存分割结果并回到主界面。单击“取消”按钮则不做任何操作直接返回主界面。

6.2 区域增长分割

区域生长是典型的串行区域分割方法，其特点是将分割过程分解为多个顺序的步骤，其中后续步骤要根据前面步骤的结果进行判断而确定。

区域生长的基本思想是将具有相似性质的像素集中起来构成区域，该方法需要先选取一个种子点，然后依次将种子像素周围的相似像素合并到种子像素

所在的区域中。

区域生长算法的优点是计算简单，特别适用于分割小的结构如肿瘤和伤疤。缺点是需要人工交互以获得种子点，这样使用者必须在每个需要抽取出的区域中植入一个种子点。同时，区域生长方法也对噪声敏感，导致抽取出的区域有空洞或者在局部体效应的情况下将原本分开的区域连接起来。

点击“分割算法”菜单栏的“区域增长分割算法”菜单项弹出如图 6.4 所示的对话框，区域增长分割操作在此对话框中进行。

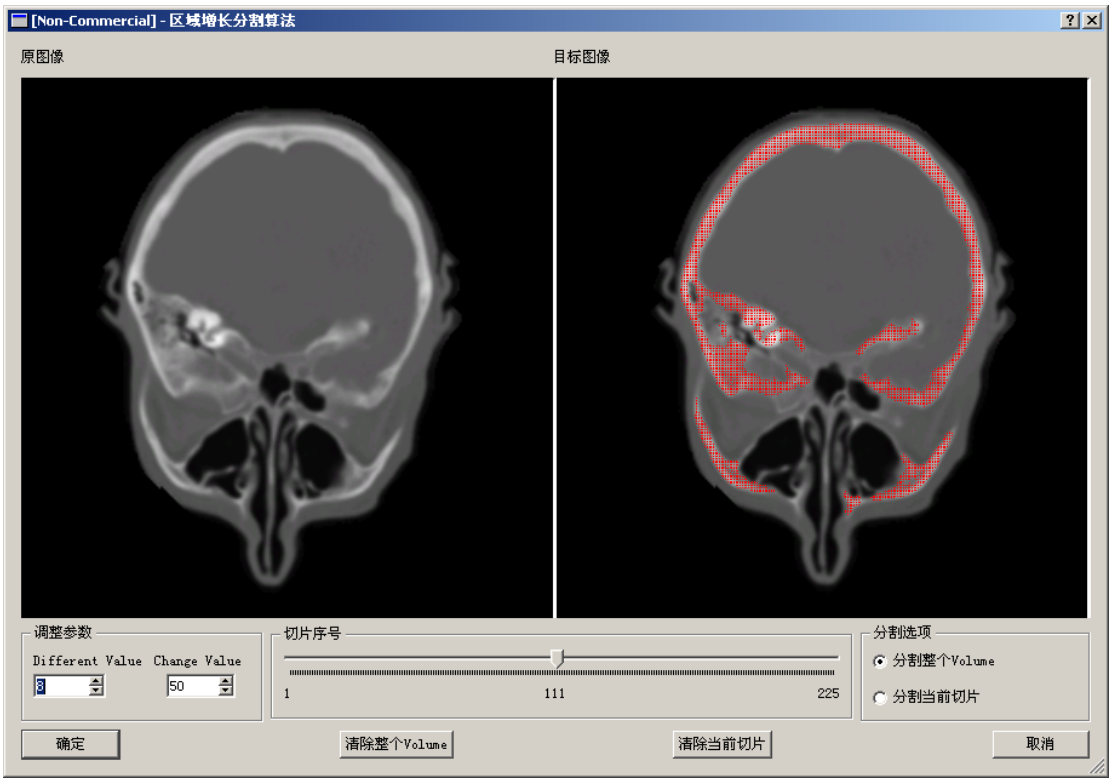


图 6.4　区域增长分割对话框

对话框的顶部显示两幅图像。左边的是“原图像”，显示了分割前的原始图像。右边的是“目标图像”，显示了分割后的结果。对话框的中部是“直方图”，显示了原始图像的灰度分布。对话框的下部“切片序号”组内的滑动条可以用来改变当前当前显示的切片图像。

“调整参数”组中的两个编辑框用于输入控制区域增长的两个参数：Different Value（dv）和 Change Value（cv）。可通过键盘输入或鼠标点击编辑框右边的上下箭头改变这两个参数的值，然后用在“原图像”区域单

击鼠标左键选定种子点位置，分割算法即开始运行，结果显示在“目标图像”区域。

该算法区域增长的规则如下：假设当前处理的区域中的点灰度值为 gc，其相邻点灰度值为 gn，用户选定的种子点灰度值为 gs，则当相邻点灰度值满足条件 $|gn - gc| < dv$ 和 $|gn - gs| < cv$ 时认为该相邻点也属于分割区域而将其合并到区域中。

对话框右下“分割选项”组的两个单选按钮用于选择分割操作紧对当前切片还是对整个体数据进行；对话框底部的“清除整个Volume”和“清除当前切片”按钮用于清除当前的分割结果（对整个体数据或仅对当前切片）。

如果对分割结果满意，可单击“确定”按钮保存分割结果并回到主界面。单击“取消”按钮则不做任何操作直接返回主界面。

第 7 章

其他辅助功能

主界面的工具栏按钮提供了一些辅助功能，主要包括改变视图区布局、对二维和三维视图进行屏幕截图、将三维视图中的变化记录为 avi 视频文件等，如图 7.1 所示。

图 7.1　辅助功能按钮

7.1 改变视图区布局

单击工具栏  按钮可实现二维和三维视图混合显示，如图 7.2 所示。在该种布局模式下还可以在“二维显示”面板进一步选择“1*3”布局或“2*2”布局，详见 2.5 节。

单击工具栏  按钮可实现单一三维视图的显示，如图 7.3 所示。

单击工具栏  按钮可实现单一二维视图（X-Y 断面）显示，如图 7.4 所示。

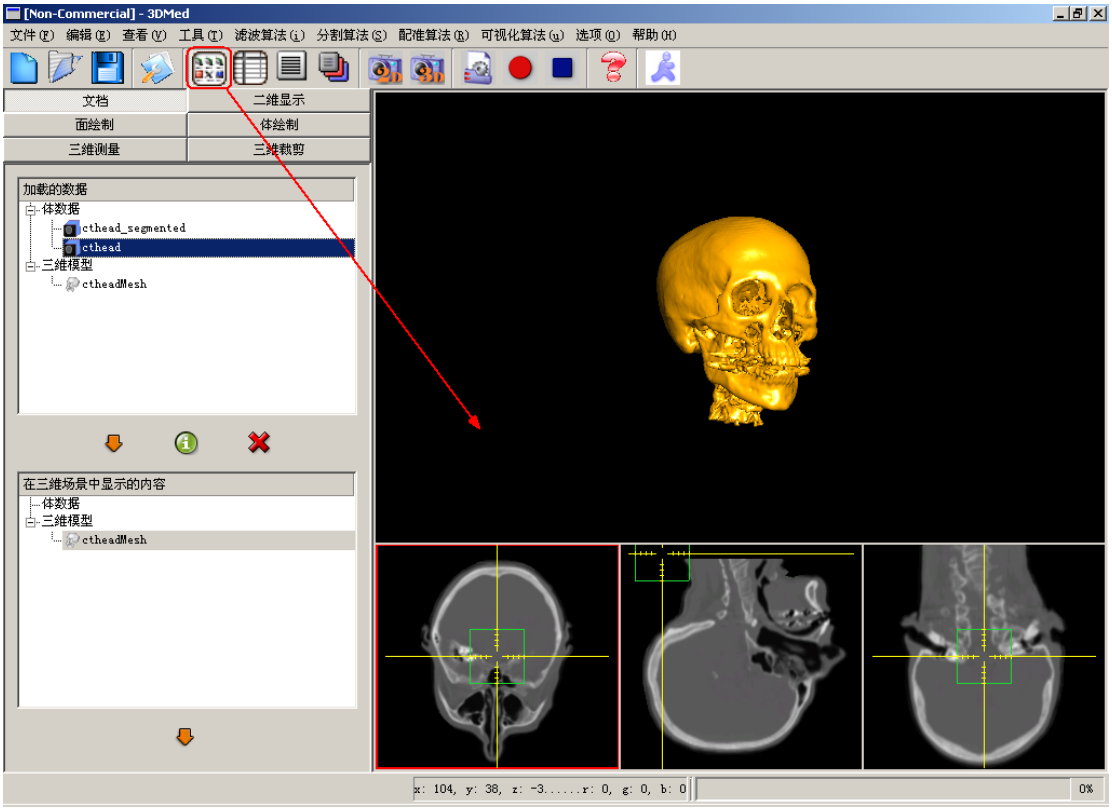


图 7.2 二维和三维视图混合显示

上述三个按钮对应于“查看”菜单栏下的相应菜单项，可以通过点击相应菜单项完成相同功能。

注意：在切换视图布局后，先前对视图中显示参数的调节将全部丢失，视图的显示将恢复到初始状态。

7.2 屏幕截图

单击工具栏  按钮可以截下当前选中二维视图区显示的内容，其结果被当作一个只有一张切片的体数据加入“体数据”列表，其名称以“snapshot”开头，后跟截图时间，如图 7.5 所示。然后可以使用导出体数据的功能将其另存为所需要的文件格式。

单击工具栏  按钮可以截下三维视图区当前显示的内容，其结果被当作一个只有一张切片的体数据加入“体数据”列表，其名称以“snapshot”开头，后跟截图时间，如图 7.6 所示。然后可以使用导出体数据的功能将其另存

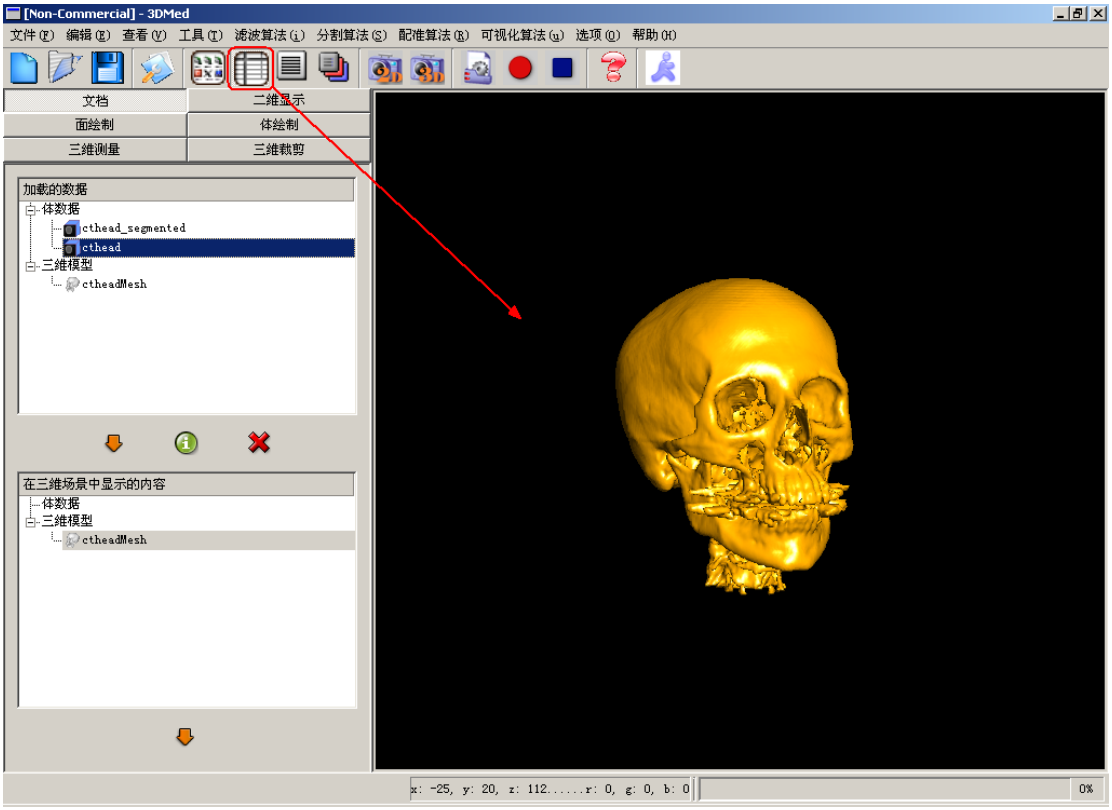


图 7.3　单一三维视图显示

为所需要的文件格式。

这两个按钮对应于菜单栏“工具”→“截图”下的菜单项，可以通过点击相应的菜单项来完成相同的功能。

7.3 屏幕录像

屏幕录像功能可以将三维视图区的变化记录成 avi 格式的视频文件。

首先，单击工具栏  按钮，会弹出一个如所示的对话框，在该对话框中输入录制参数。

在该对话框中，若选中“自动旋转”，则录制时三维视图中的物体将自动按设定的参数进行旋转，每转动一个角度就记录一帧，这时可通过下方“旋转参数设置”组中的编辑框输入旋转轴、每次旋转的角度以及总共旋转的角度，在这种情况下，录制时主界面将不响应鼠标输入。若取消“自动旋转”的选择，则录制时将记录三维视图区鼠标操作所带来的每一次显示更新（视图区的

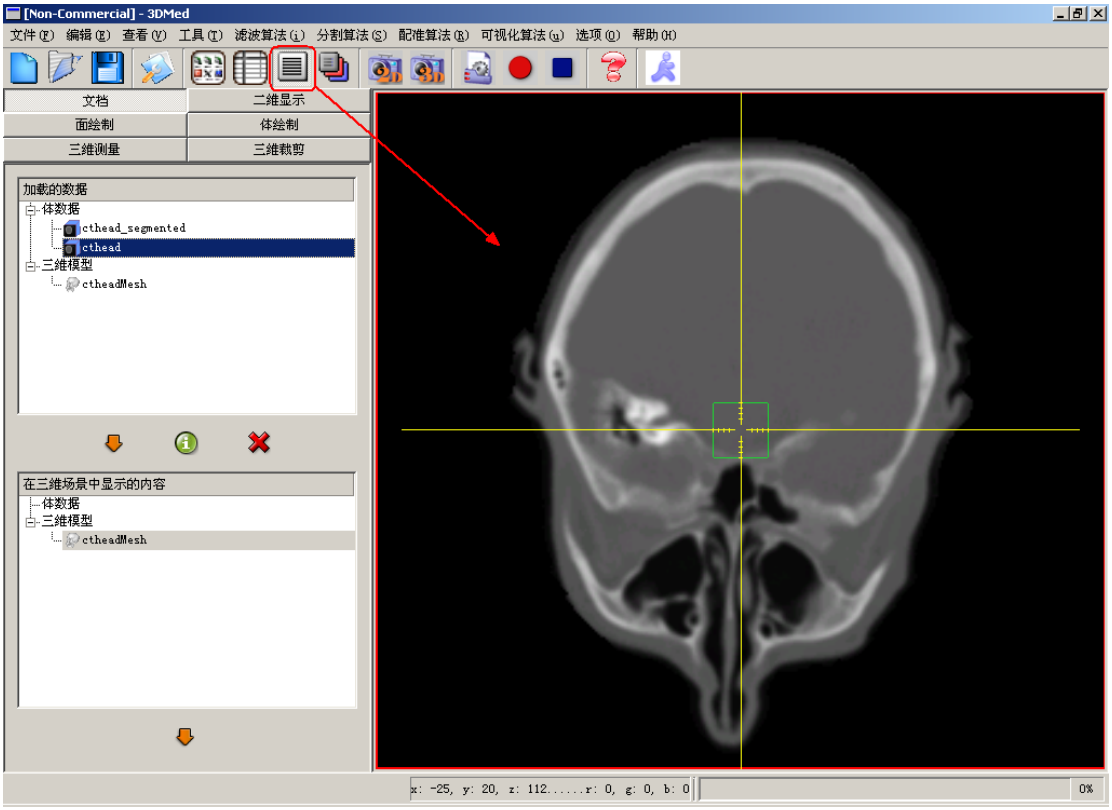


图 7.4 单一二维视图显示

每一次更新将记录为一帧）。

在该对话框的“AVI 参数设置”组可以设定输出的 avi 文件的相关参数，包括帧尺寸、帧速等。帧尺寸通过 4 个参数来控制，左下角坐标（X 方向偏移和 Y 方向偏移）、帧宽和帧高，设定后在录制过程中，将记录三维视图区从左下角坐标开始，宽和高分别为帧宽和帧高的矩形区域中的内容。初始状态下记录区域是整个三维视图区，帧速为 15 fps，可通过键盘直接在对应编辑框内输入数值来改变这些参数的设定，同时可在右边的显示区看到对帧尺寸设定的预览效果，如图 7.8 所示。

完成设定后，按“确定”按钮将保存录制参数，按“取消”按钮则丢弃对录制参数的更改。

然后，单击工具栏  按钮，这时先弹出一个保存文件的对话框，输入 avi 文件的文件名，如所示，按“保存”之后弹出第二个对话框，选择视频流编码器，选定之后按“确定”即开始录制。在录制过程中  按钮保持按下状态，  按钮不可用。录制完成后，单击工具栏  按钮可停止录制（对非自动旋转

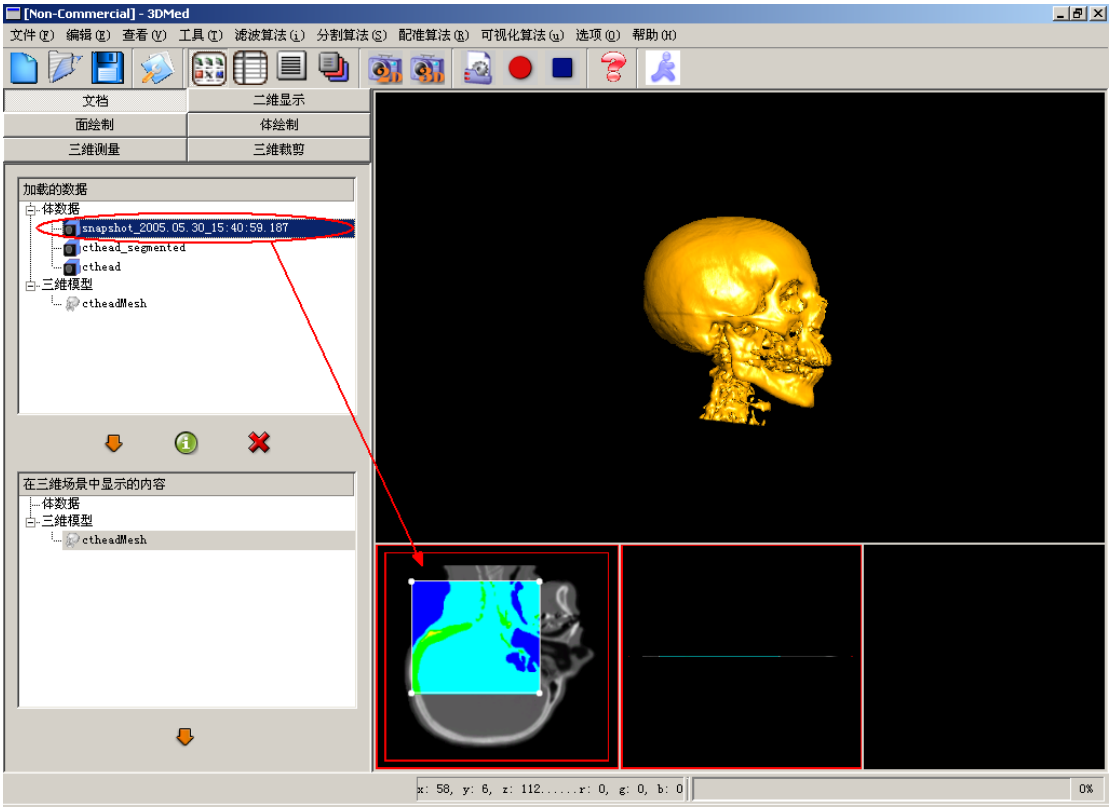


图 7.5 截取二维视图

录制而言）。

上述按钮对应于菜单栏“工具”→“录像”下的菜单项，可以通过点击相应的菜单项来完成相同的功能。

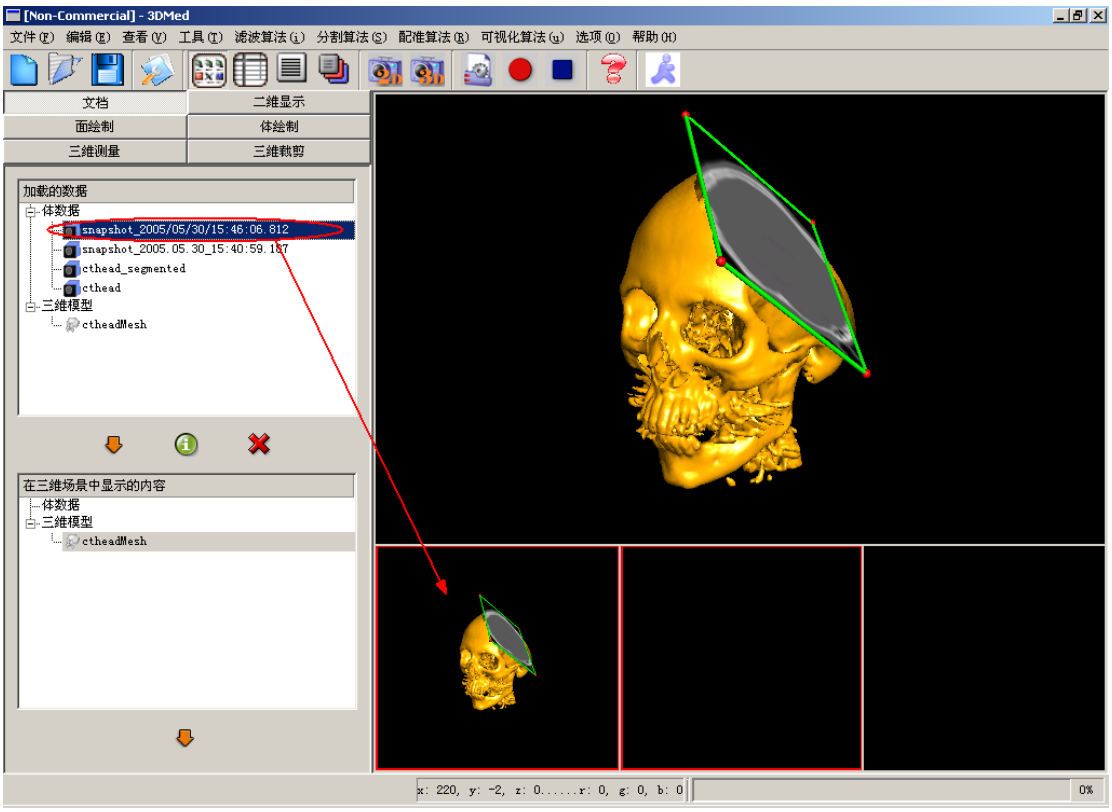

[Non-Commercial] - 3DMed
文件(F) 编辑(E) 查看(V) 工具(T) 滤波算法(i) 分割算法(S) 配准算法(R) 可视化算法(u) 选项(O) 帮助(H)
文档
二维显示
面绘制
体绘制
三维测量
三维裁剪
加载的数据
体数据
snapshot_2005/05/30/15:46:06.812
snapshot_2005.05.30_15:40:59.187
ctheadMesh
在三维场景中显示的内容
体数据
三维模型
ctheadMesh

图 7.6 截取三维视图

图 7.7 录制参数设定

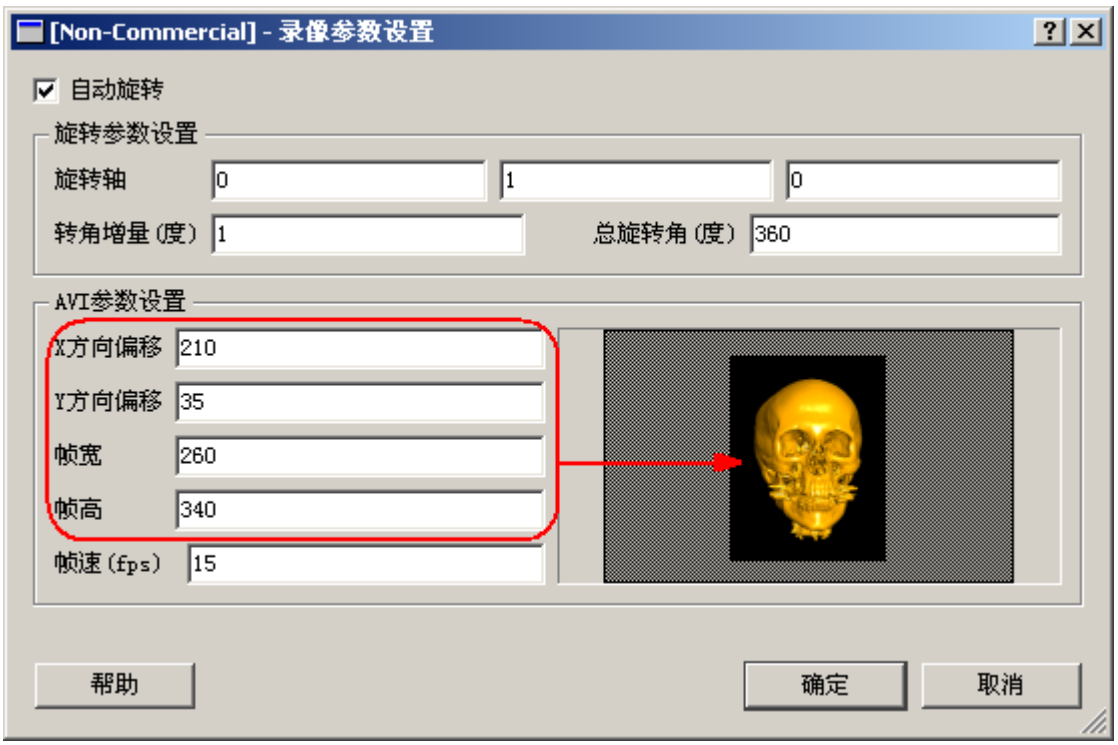


图 7.8 预览帧尺寸改变的效果

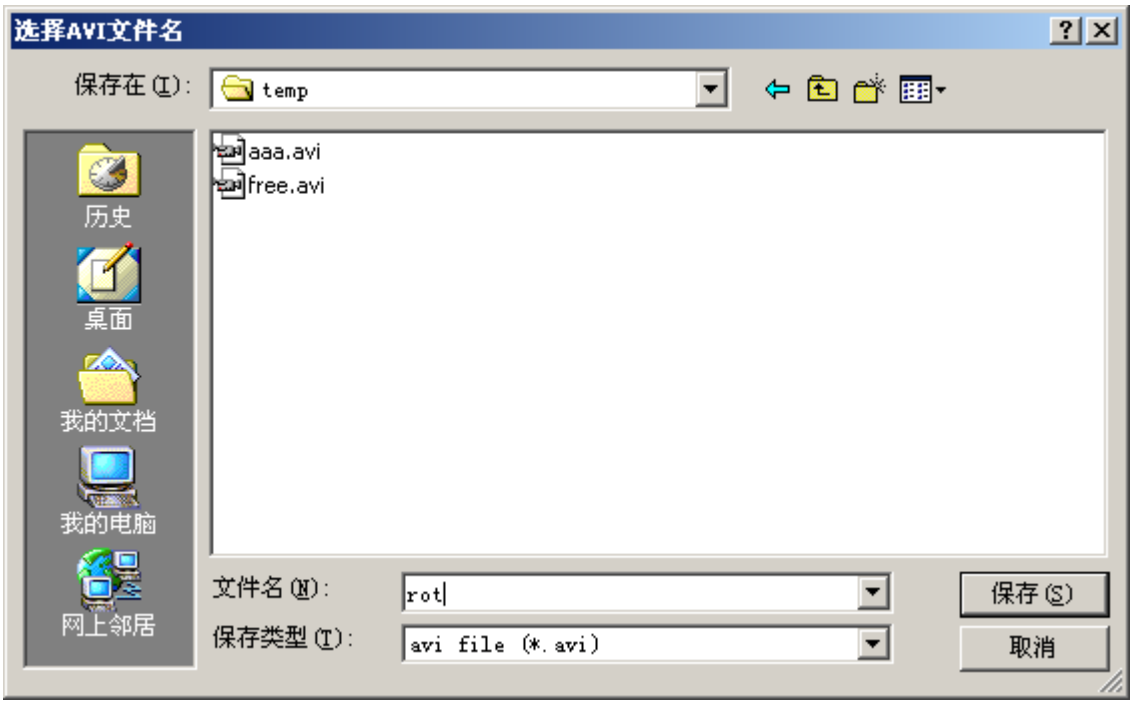


图 7.9　选择或输入保存 avi 文件的文件名

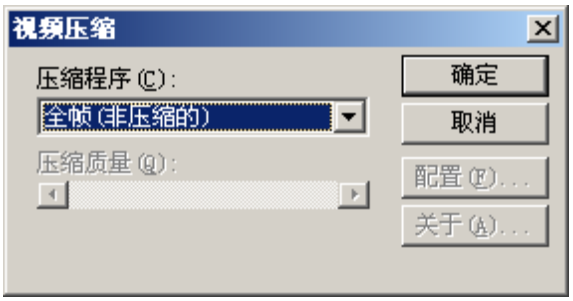


图 7.10　选择视频流编码器

第 8 章

开发 3DMed 的 Plugin

3DMed 采用一种开放的架构，通过 Plugin 的方式动态加载分割、配准、可视化等算法模块。按照软件配套提供的 PluginsSDK 接口规范，用户可以自行设计实现自己算法的 Plugin 加入到 3DMed 中去。下面以 Microsoft Visual C++ 6.0 作为编程环境，演示如何开发 3DMed 的 Plugin。

8.1 概述

在 3DMed 中，所有的 Plugins 分为五个大类（ I/O Plugin, Filter Plugin, Registration Plugin, Segmentation Plugin 和 Visualization Plugin ），其中 I/O Plugin 又分为 medVolumeImportPlugin, medVolumeExportPlugin, medMeshImportPlugin 和 medMeshExportPlugin 四个小类，Filter Plugin 又分为 medVolumeFilterPlugin 和 medMeshFilterPlugin 两个小类。这五大类九小类 Plugin 定义了编写 3DMed Plugin 时必须遵循的接口，是整个 Plugin 机制的基础。要编写一个 3DMed 的 Plugin，一般的步骤如下所示：

1. 先判断自己要完成的Plugin在功能上属于上面所讲的五大类中哪一类的，然后再找出从属的具体的某个小类；
2. 写自己的Plugin，从第一步中选择的那个基类中公有继承；
3. 实现虚函数 bool Show()，在这个函数里面完成Plugin的具体功能。

需要注意的是，在这个函数里面你可以作出完整的GUI图形界面来完成某项功能，也可以不要图形界面，只是在后台完成功能。这个接口规定得相当宽松，所以实现 GUI 图形界面时所用的编程工具也并没有限制，本章中的例子使用 Microsoft 的 MFC 来作为 GUI 开发工具，而 3DMed 里面自带的所有的 Plugin 都是使用跨平台的 GUI 类库 Qt 来开发的，你甚至可以使用 Windows API，或者 Borland 的 C++ Builder，所有这些工具作出来的 Plugin

都可以很好地被 3DMed 的主程序所管理和调用。另外， 3DMed 通过两个数据类：medVolume 和 medMesh 把自己内部的数据暴露给 Plugin 的开发者，所以要开发一个 3DMed 的 Plugin，还必须对这两个数据类所提供的 API 函数有所了解，medVolume 所提供的主要的 API 函数如下：

```
//设置Volume 的宽度（Pixel 为单位）
void SetWidth(int w);
//得到Volume 的宽度（Pixel 为单位）
int  GetWidth();

//设置Volume 的高度（Pixel 为单位）
void SetHeight(int h);
//得到Volume 的高度（Pixel 为单位）
int  GetHeight();

//设置Volume 的切片张数
void SetImageNum(int s);
//得到Volume 的切片张数
int  GetImageNum();

//设置像素间的水平间距（mm 为单位）
void  SetSpacingX(float px);
//得到像素间的水平间距（mm 为单位）
float GetSpacingX();

//设置像素间的竖直间距（mm 为单位）
void  SetSpacingY(float py);
//得到像素间的竖直间距（mm 为单位）
float GetSpacingY();

//设置切片间的间距（mm 为单位）
void  SetSpacingZ(float pz);
//得到切片间的间距（mm 为单位）
float GetSpacingZ();

//设置Volume 的通道数（1 为灰度图像，3 为RGB 彩色图像）
void SetNumberOfChannel( int n );
//得到Volume 的通道数（1 为灰度图像，3 为RGB 彩色图像）
int  GetNumberOfChannel();
```

```
//得到实际的数据指针，必须依据GetDataType 返回的数据类型，
//将其强制转换成相应的指针类型才能使用
void* GetRawData();

//得到某一张切片的数据指针，必须依据
//GetDataType 返回的数据类型，
//将其强制转换成相应的指针类型才能使用
void* GetSliceData(int sliceNum);

//得到此Volume的数据类型，返回的值及其代表的意义如下：
// MED_CHAR -- c 语言中char 类型
// MED_UNSIGNED_CHAR -- c 语言中unsigned char 类型
// MED_SHORT -- c 语言中short 类型
// MED_UNSIGNED_SHOR -- c 语言中unsigned short 类型
// MED_INT -- c 语言中int 类型
// MED_UNSIGNED_INT -- c 语言中unsigned int 类型
// MED_LONG -- c 语言中long类型
// MED_UNSIGNED_LONG -- c语言中unsigned long 类型
// MED_FLOAT -- c 语言中float 类型
// MED_DOUBLE -- c 语言中double 类型
int  GetDataType();

//设置Volume 的数据类型，变量type 的取值及意义参见GetDataType
void SetDataType(int type);
//设置Volume 的数据类型为c 语言中的float 类型
void SetDataTypeToFloat();
//设置Volume 的数据类型为c 语言中的double 类型
void SetDataTypeToDouble();
//设置Volume 的数据类型为c 语言中的int 类型
void SetDataTypeToInt();
//设置Volume 的数据类型为c 语言中的unsigned int 类型
void SetDataTypeToUnsignedInt();
//设置Volume 的数据类型为c 语言中的long 类型
void SetDataTypeToLong();
//设置Volume 的数据类型为c 语言中的unsigned long 类型
void SetDataTypeToUnsignedLong();
//设置Volume 的数据类型为c 语言中的short 类型
void SetDataTypeToShort();
//设置Volume 的数据类型为c 语言中的unsigned short 类型
void SetDataTypeToUnsignedShort();
//设置Volume 的数据类型为c 语言中的char 类型
```

```
void SetDataTypeToUnsignedChar();
//设置Volume 的数据类型为c 语言中的unsigned char 类型
void SetDataTypeToChar();

//根据预先设置好的Width, Height, Slice Number,
//Data Type 和Channel Number 来计算实际数据所需的
//内存大小，并分配内存，最后返回首地址。
//需要注意的是，在调用此函数之前，必须确保已经调用了
//SetWidth, SetHeight, SetImageNum, SetNumberOfChannel,
//SetDataType 设置好了相关的信息。
void* Allocate();

//返回数据所占用的内存的大小，单位是字节。
unsigned long GetActualMemorySize();
```

medMesh 提供的主要的 API 函数如下：

```
//设置此Mesh 的顶点个数
void SetVertexNumber(int number);
//得到此Mesh 的顶点个数
int GetVertexNumber();
//设置此Mesh 的三角片个数
void SetFaceNumber(int number);
//得到此Mesh 的三角片个数
int GetFaceNumber();
//得到此Mesh 的顶点数据指针
float* GetVertexData();
//得到此Mesh 的三角面片数据指针
unsigned int* GetFaceData();

//得到此Mesh 的包围盒，返回一个指向六个浮点数的指针.
//六个浮点数的顺序及意义为：最小x ，最大x ，
//最大y ，最小y ，最小z ，最大z 。
float* GetBoundingBox(void);
//设置此Mesh 的包围盒，参数意义见GetBoundingBox
void   SetBoundingBox(float minX, float maxX,
                      float minY, float maxY,
                      float minZ, float maxZ);

//得到顶点数据和三角面片数据所占的内存大小，单位是字节。
unsigned long GetActualMemorySize();
```

有了 medVolume 和 medMesh 提供的 API 函数以后，Plugin 的开发者就可以通过它们访问 3DMed 的内部数据.

在编写具体的 Plugin 时，首先，在一开始创建工程的时候就要选择正确的工程类型（每一个 Plugin 都是一个动态链接库）；其次，所有 Plugin 的开发都要用到 PluginsSDK，它里面提供了 medVolume、medMesh 以及九个小类的 Plugin 的定义，PluginsSDK 是随着 3DMed 一起分发的，提供了必要的头文件和库文件；最后，Plugin 的开发可以分为两种，一种使用 MITK 作为底层算法库，一种可以完全不使用 MITK，纯粹加入自己的算法，如果使用 MITK 的话，那么开发 Plugin 时还需要 MITK 的头文件以及相应的库文件。

下面将用两个具体的实例来演示 3DMed Plugin 的开发，其中第一个实例使用 MITK，而第二个实例不使用 MITK。

8.2 实例 1：使用 MITK

本实例使用 Microsoft Visual C++ 6.0 作为工具，从如何建立工程，到修改工程的设置，以及具体的编程实现，一直到最后集成进 3DMed，给出了一个完整的例子，演示如何在使用 MITK 的情况下开发 3DMed 的 Plugin。为了清晰和简化起见，这个 Plugin 的功能比较简单，只是读入一系列 BMP 格式的文件，并送给 3DMed 处理。

8.2.1 工程的建立及设置

在 Microsoft Visual C++ 6.0 的 IDE 环境下，新建一个工程 IOPlugin，工程的类型选择“MFC AppWizard (dll)”（如图 8.1 所示），因为我们要使用 MFC 作 GUI 图形界面，并且目标是生成一个动态链接库。

在接下来的一步中，选择 DLL 类型为“Regular DLL using shared MFC DLL”，如图 8.2 所示。此时可以单击“Finish”按钮完成工程的创建。

创建完工程以后，选择菜单栏“Projcet”→“Settings”菜单，在弹出的“Project Settings”对话框中选择“C/C++”标签页，在“Category”列表框里面选择“Preprocessor”，然后在“Additional Include directories”编辑框中输入 PluginsSDK 和 MITK 的头文件路径，如图 8.3 所示。需要注意的是这里实际的路径依赖于你的机器上 PluginsSDK 和 MITK 头文件的放置路径，请根

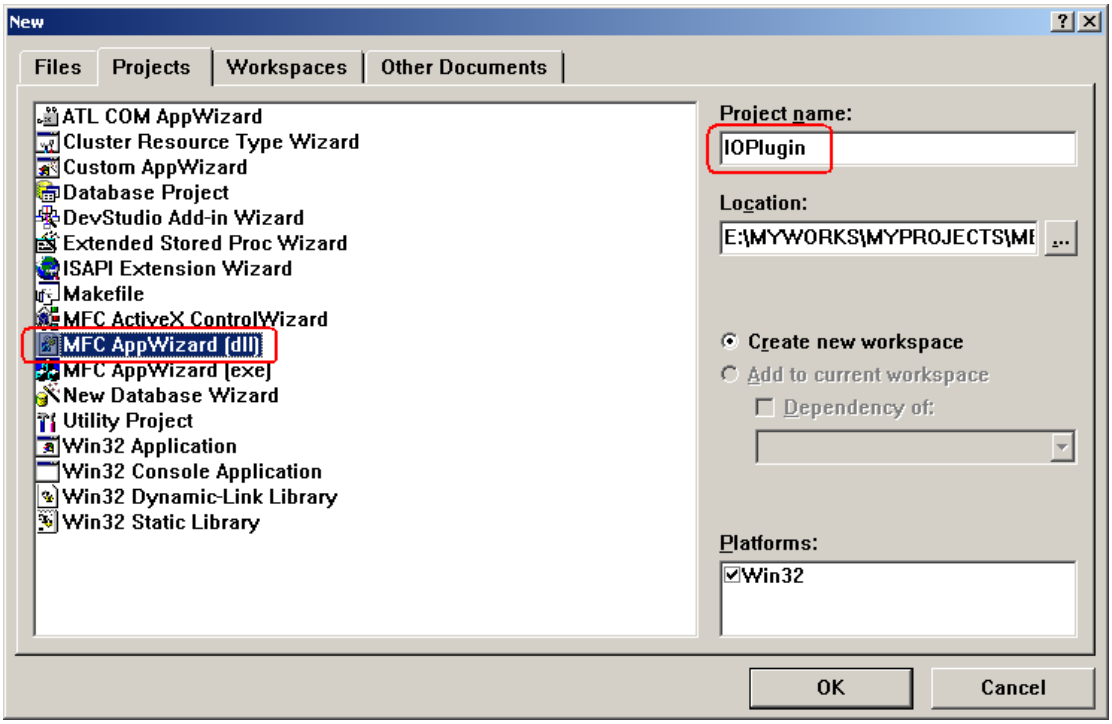


图 8.1 新建 IOPlugin 工程

据实际情况设置。

设置完头文件路径以后，接下来要设置的是必要的导入库文件的路径。还是在“Projcet Settings”对话框中，选择“Link”标签页，在“Category”列表框中选择“Input”，然后在“Object/library modules”编辑框中输入“PluginsSDK.lib Mitk_dll.lib”（之间用空格格开），表示要使用这两个导入库，并且还需要在“Additional library path”编辑框中输入这两个导入库文件所在的路径，如图 8.4 所示。需要注意的是这里实际的路径依赖于你的机器上的 PluginsSDK 和 MITK 库文件的放置路径，请根据实际情况设置。

8.2.2 实例制作

设置好工程选项以后，就可以进入实质部分了。下面创建我们的 Plugin 的类，选菜单栏“Insert”→“New Class”创建一个新类，并将其命名为 CMyIOPlugin，如图 8.5 所示。

因为我们的 Plugin 功能是读入一系列 BMP 格式的文件，所以很显然是属于 I/O Plugin 中的 medVolumeImportPlugin，故 CMyIOPlugin 应该从

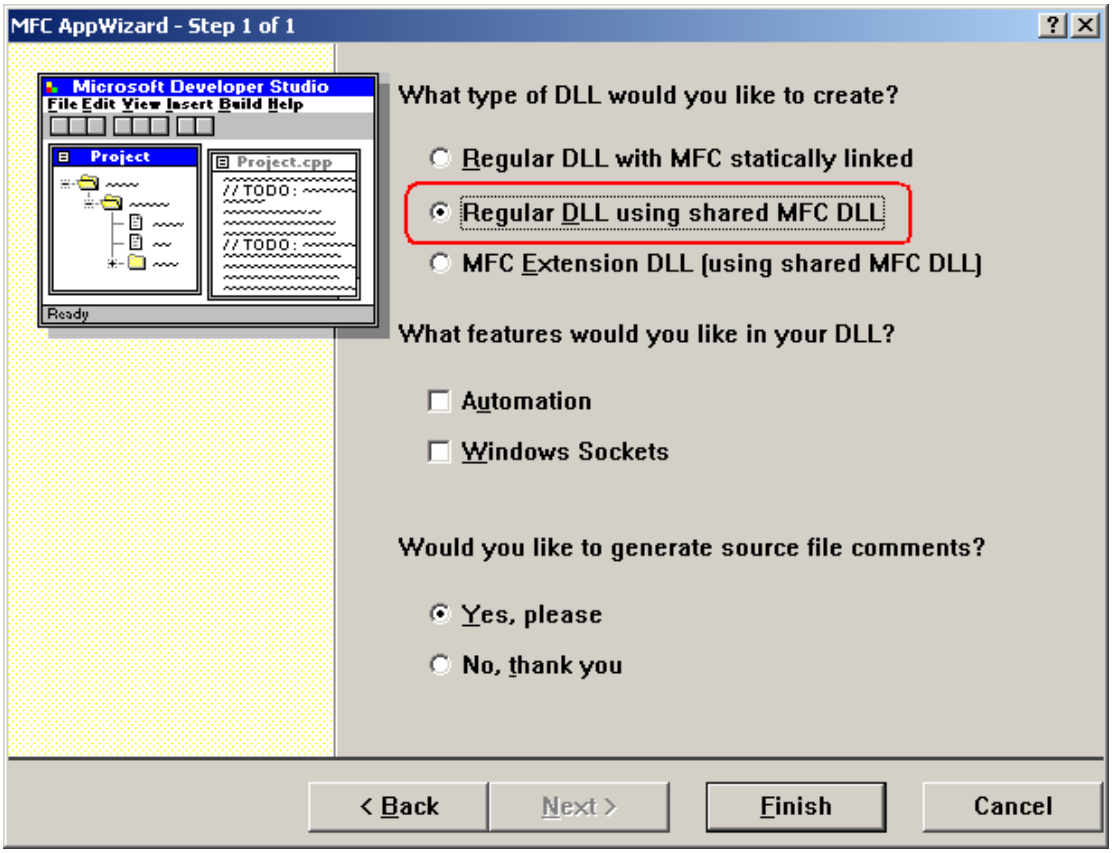


图 8.2 选择 DLL 类型

medVolumeImportPlugin 公有继承，并应该重载 Show 函数，其类声明（在“MyIOPlugin.h”文件中）如下所示：

```
class CMyIOPlugin : public medVolumeImportPlugin
{
public:
    CMyIOPlugin();
    virtual ~CMyIOPlugin();

    virtual bool Show(void);

};
```

当然，在其前面应该包含 PluginsSDK 中的 medPlugin.h 文件，里面定义了 medVolumeImportPlugin，如下所示：

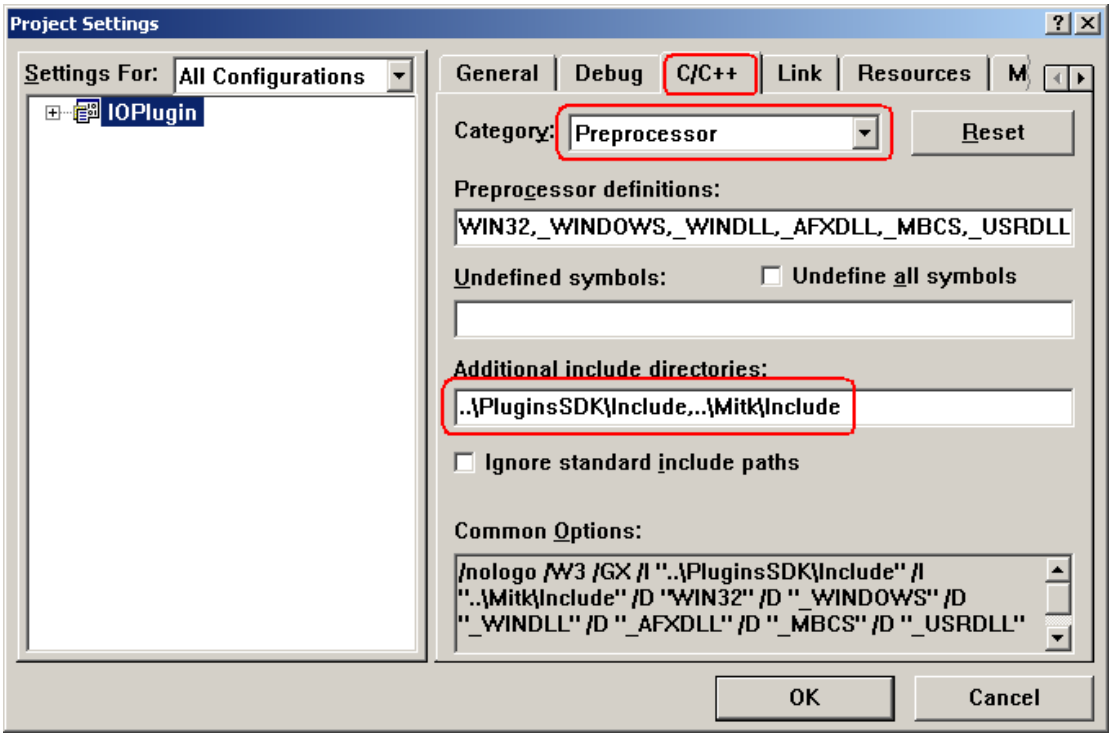


图 8.3　设置必要的头文件路径

```
#include "medPlugin.h"
```

整个 Plugin 的功能都在最重要的 Show 函数里面完成，它产生一个打开文件对话框，供用户选择多个 BMP 文件，并打开这些文件，形成一个 Volume 传给 3DMed 去处理。因为此例子使用了 MITK，所以这里的代码很简单，直接使用了 MITK 中提供的 mitkBMPReader，整个 Show 函数的代码（在“MyIOPlugin.cpp”文件中）如下所示：

```
bool CMyIOPlugin::Show(void)
{
    //MFC 的规定，必须先调用这个宏.
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    //使用MFC 提供的打开文件对话框.
    CFileDialog dlg(TRUE,
                    ".bmp",
                    NULL,
                    OFN_HIDEREADONLY |
```

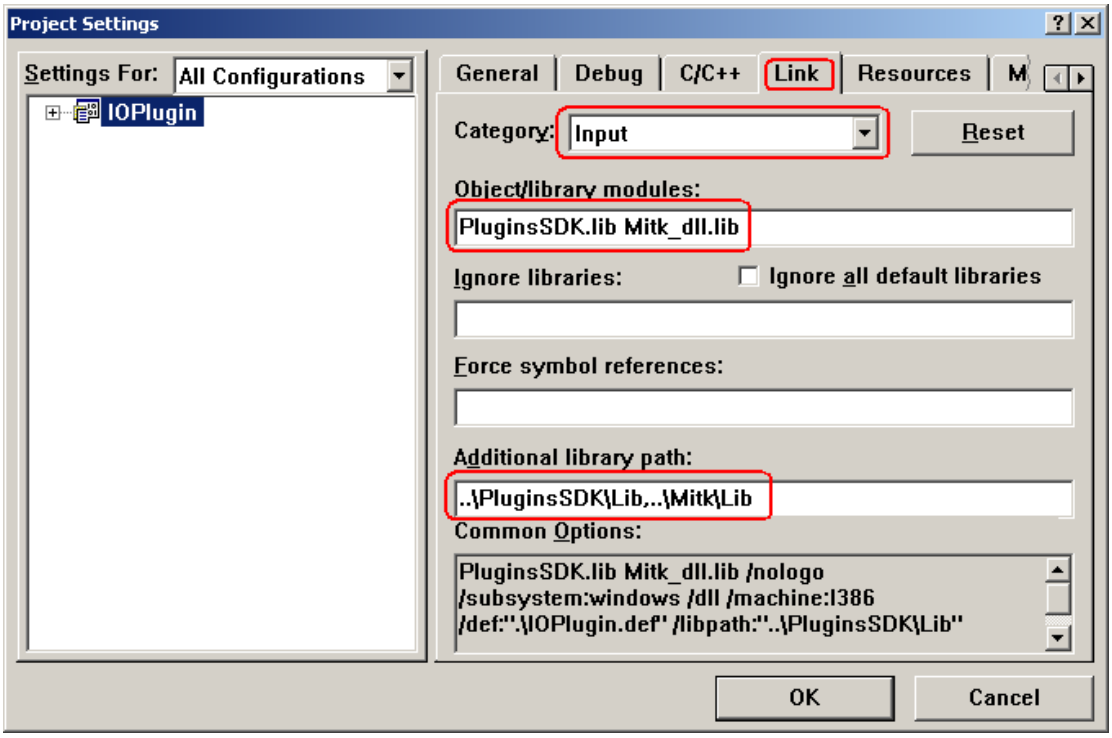


图 8.4 设置必要的库文件路径

```
                OFN_OVERWRITEPROMPT |
                OFN_ALLOWMULTISELECT,
                "BMP文件(*.bmp)|*.bmp||",
                NULL);

//CFileDialog 用于存储文件名的默认缓冲区
//太小，在选择多文件时可能溢出，因此需要
//设置额外的缓冲区来存储文件名
char *fileNamesBuf = new char[512 * _MAX_PATH];
fileNamesBuf[0] = '\0';
DWORD maxBytes = 512 * _MAX_PATH;
dlg.m_ofn.lpstrFile = fileNamesBuf;
dlg.m_ofn.nMaxFile = maxBytes;

//如果用户选择了文件并点“确定”按钮.
if(dlg.DoModal() == IDOK)
{
    mitkBMPReader *aReader = new mitkBMPReader;

    //循环得到用户选中的每个文件名,
```

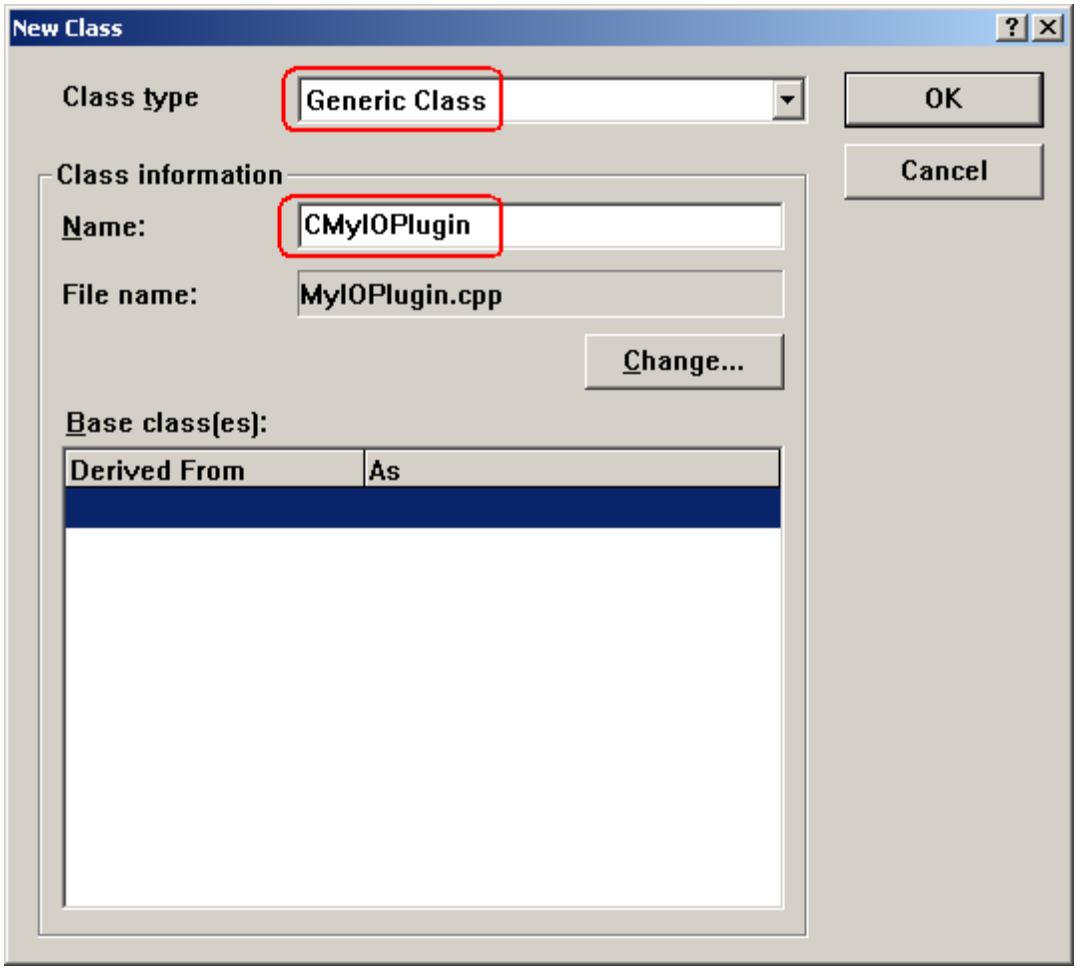


图 8.5 创建 CMyIOPlugin 类

```
//并将其加入到mitkBMPReader 中.
POSITION pos = dlg.GetStartPosition();
CString szFileName;
while(pos)
{
    szFileName = dlg.GetNextPathName(pos);
    aReader->AddFileName(szFileName);
}

//像素间距应该是在读取文件之前由用户输入的，
//这里为简化起见直接设置为1.0.
aReader->SetSpacingX(1.0f);
aReader->SetSpacingY(1.0f);
aReader->SetSpacingZ(1.0f);
```

```
            //运行mitkBMPReader 读取文件.
            aReader->Run();

            //生成输出的数据.
            m_Data = new medVolume;
            m_Data->SetData(aReader->GetOutput());
            m_Data->SetFileName(szFileName);
            m_Data->SetName("My BMP Files");

            //删除mitkBMPReader 对象.
            aReader->Delete();
            return true;
        }

        return false;
    }
```

重载完了 Show 函数以后，还剩下最后一件事情，就是加入一些导出函数，这一切只需在“MyIOPlugin.cpp”文件中所有函数体**之外**调用下面的宏即可实现，其中第一个参数是我们所写的 Plugin 的类名，这里当然是 CMyIOPlugin 了，第二个参数是我们所希望在 3DMed 菜单里面所看到的此 Plugin 对应的菜单项文字。每一类 Plugin 都有各自的宏来生成导出函数，具体宏名称及定义可参考“medPlugin.h”。

```
IMPLEMENT_VOLUME_IMPORT(CMyIOPlugin, "My BMP Plugin")
```

最后不要忘了在“MyIOPlugin.cpp”文件前面加上必要的头文件，如下所示：

```
//PluginsSDK头文件
#include "medVolume.h"

//MITK头文件
#include "mitkBMPReader.h"
#include "mitkVolume.h"
```

8.2.3 将生成的 Plugin 加入 3DMed

经过了上面的步骤以后，现在可以编译整个工程，得到 IOPlugin.dll 文件，将其拷贝到 3DMed 安装目录的 Plugins 子目录下，这时运行 3DMed 主程序，加载完所有的插件以后，点击“文件”菜单下的“加载体数据”菜单时，就会发现我们的“My BMP Plugin”也在其中，如图 8.6 所示。

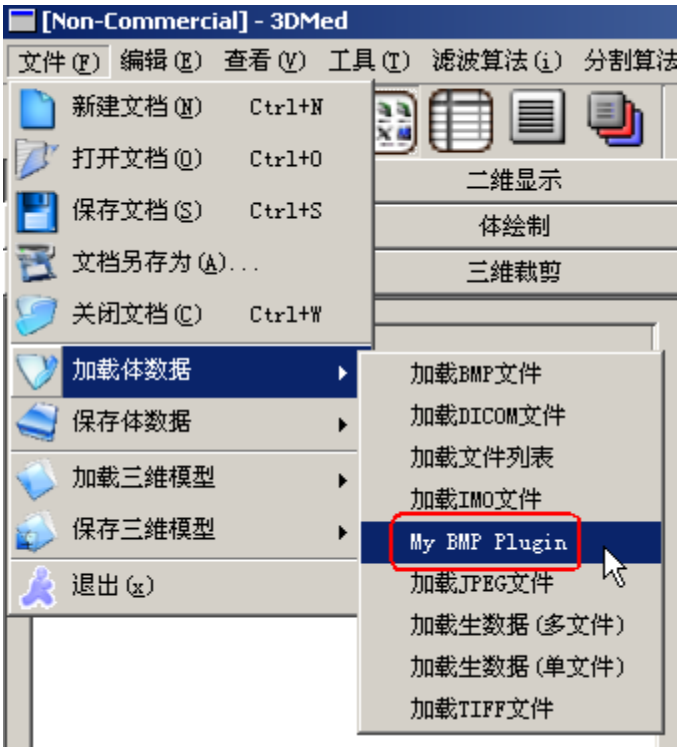


图 8.6 成功加载的 IOPlugin

当点击“My BMP Plugin”后，就会有打开文件对话框弹出来，让用户选择 BMP 文件，如图 8.7 所示。选择完以后这些数据就会在 3DMed 中打开并显示，一切都如预期的一样。

8.3 实例 2：不使用 MITK

在有些情况下，用户只对 3DMed 感兴趣，而并不想学习并使用 MITK，那么照样可以撰写 3DMed 的 Plugin。本节的实例就是演示如何在没有 MITK 的情况下，来编写有效的 Plugin 并集成到 3DMed 中去。本实例仍然使用

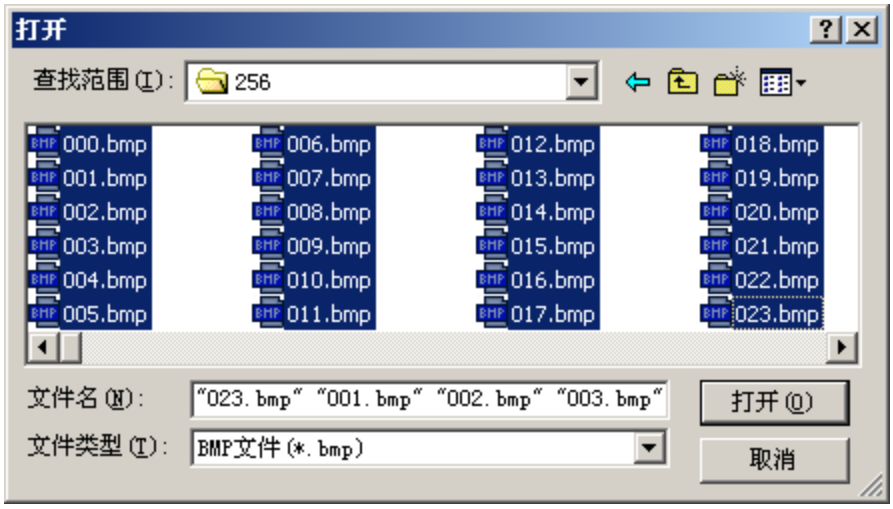


图 8.7 运行 IOPlugin 弹出的打开文件对话框

Microsoft Visual C++ 6.0 作为工具，为了清晰和简化起见，这个 Plugin 的功能比较简单，实现了对任意数据类型体数据的阈值分割算法，支持设置高阈值和低阈值，提供了一个简单的对话框来设置参数。

8.3.1 工程的建立及设置

在 Microsoft Visual C++ 6.0 的 IDE 环境下，新建一个工程 SegPlugin，工程的类型选择“MFC AppWizard (dll)”，因为我们要使用 MFC 作 GUI 图形界面，并且目标是生成一个动态链接库。在接下来的一步中，选择 DLL 类型为“Regular DLL using shared MFC DLL”，如此时可以单击“Finish”按钮完成工程的创建。（相关图示可参考 8.2 节。

创建完工程以后，选择菜单栏“Projcet”→“Settings”菜单，在弹出的“Project Settings”对话框中选择“C/C++”标签页，在“Category”列表框里面选择“Preprocessor”，然后在“Additional Include directories”编辑框中输入 PluginsSDK 的头文件路径（这里不再需要MITK），如图 8.8 所示。需要注意的是这里实际的路径依赖于你的机器上 PluginsSDK 头文件的放置路径，请根据实际情况设置。

设置完头文件路径以后，接下来要设置的是必要的导入库文件的路径。还是在“Projcet Settings”对话框中，选择“Link”标签页，在“Category”列表框中选择“Input”，然后在“Object/library modules”编辑框中输入

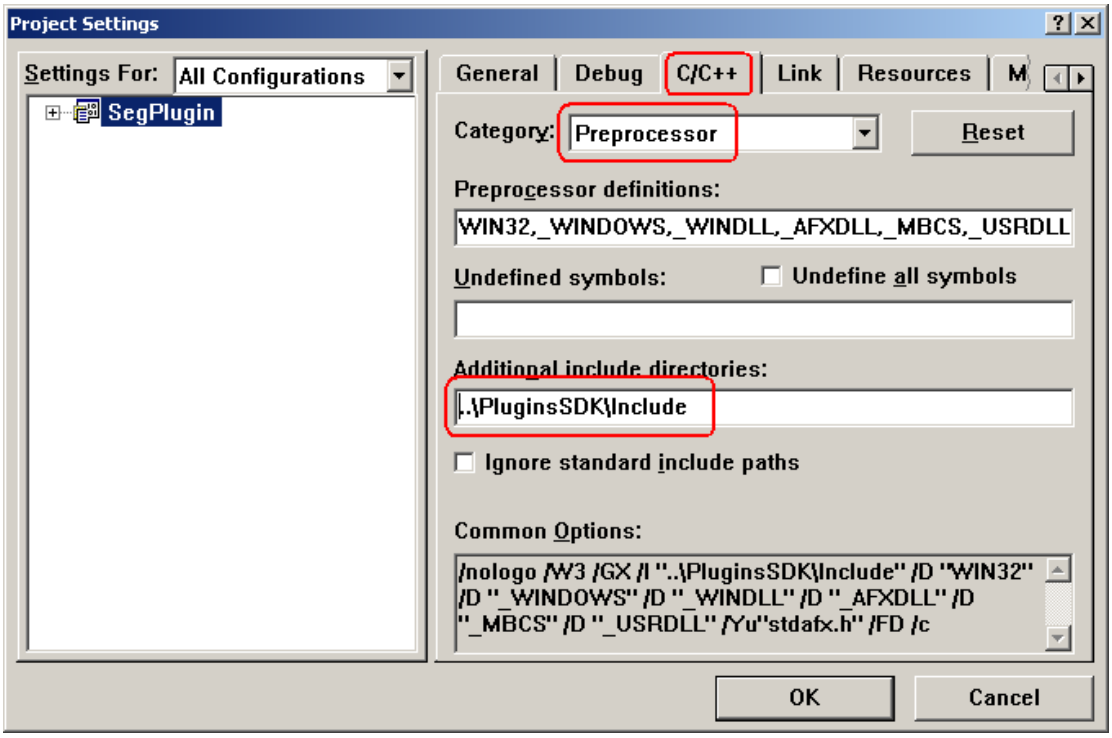


图 8.8 设置必要的头文件路径

“PluginsSDK.lib”（不再需要 Mitk_dll.lib ），并且还需要在“Additional library path”编辑框中输入这个导入库文件所在的路径，如图 8.9 所示。需要注意的是这里实际的路径依赖于你的机器上的 PluginsSDK 库文件的放置路径，请根据实际情况设置。

8.3.2 实例制作

置好工程选项以后，就可以进入实质部分了。下面创建我们的 Plugin 的类，选菜单栏“Insert”→“New Class”创建一个新类，并将其命名为 CMyIOPlugin ，如图 8.10 所示。

这个 Plugin 的功能是进行阈值分割，所以很显然是属于 medSegmentationPlugin，故 CMySegPlugin 应该从 medSegmentationPlugin 公有继承，并应该重载 Show 函数，其类声明（在“MySegPlugin.h”文件中）如下所示，同时不要忘了在前面包含“medPlugin.h”头文件：

```
class CMySegPlugin
```

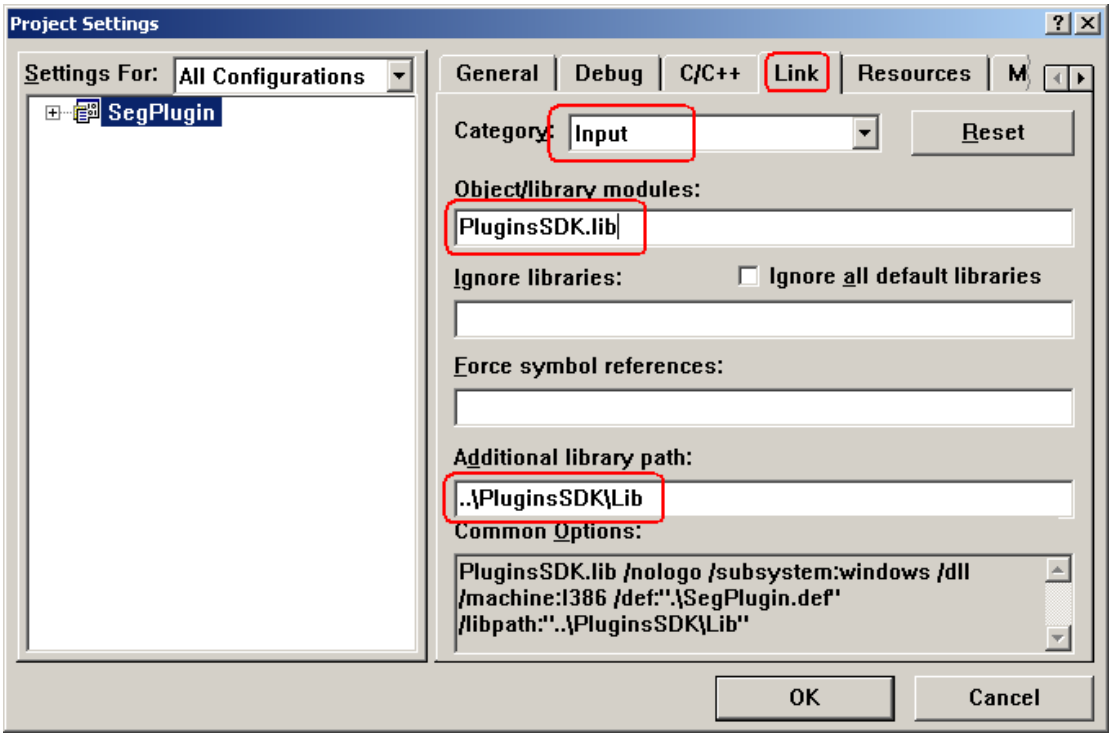


图 8.9 设置必要的库文件路径

```
{
public:
    CMySegPlugin();
    virtual ~CMySegPlugin();

    virtual bool Show(void);

private:
    bool doSegmentation(float lowThre, float highThre);
};
```

其中的私有成员函数 doSegmentation 用来完成实际的分割工作，它在 Show 函数里面被调用。

下面需要作的是 GUI 图形界面工作了，我们需要制作一个对话框，在“Resource View”面板上点右键，选“Insert”菜单，插入一个新的对话框资源。在“Dialog Properties”对话框中将其 ID 设置为“IDD_DIALOG_PARAMETER”，Caption 设置为“设置阈值”，如图 8.11 所示。然后在对话框上放置两个静态文本控件、两个编辑框控件和“确定”、

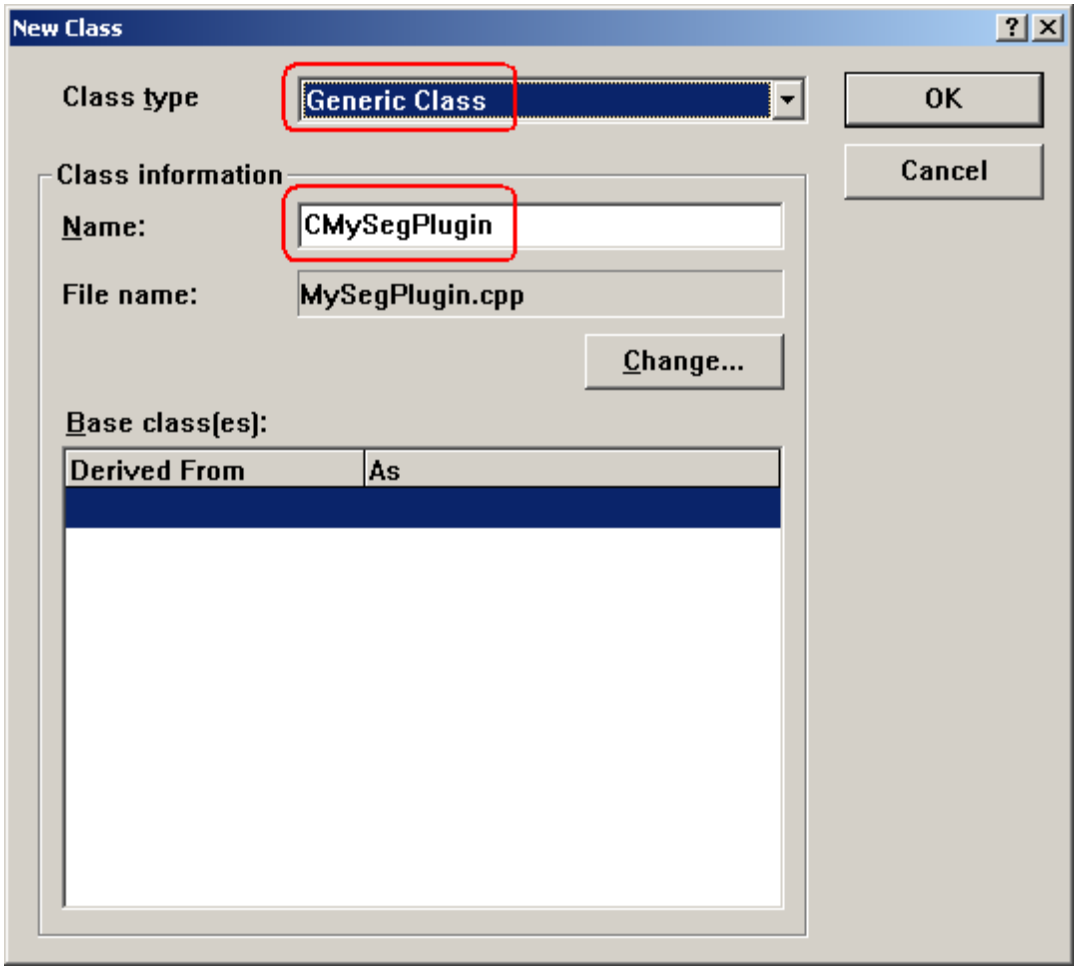


图 8.10 创建 CMySegPlugin 类

“取消”按钮，其界面如图 8.12 所示。

界面创建完以后，使用 ClassWizard 为这个对话框创建一个新类，类的名字叫 CDialogParameter，如图 8.13 所示。然后再使用 ClassWizard 对话框中的“Member Variables”标签页，将对话框中的两个编辑框设置成 CDialogPapameter 类的成员变量，类型均为 float 型，如图 8.14 所示。

最后剩下的是添加成员函数来读取和设置高、低域值这两个成员变量了，分别是 SetLowThre 、 SetHighThre 、 GetLowThre 和 GetHighThre 函数，整个 CDialogParameter 的程序代码如下所示：

```
class CDialogParameter : public CDialog
{
```

图 8.11 对话框属性设置

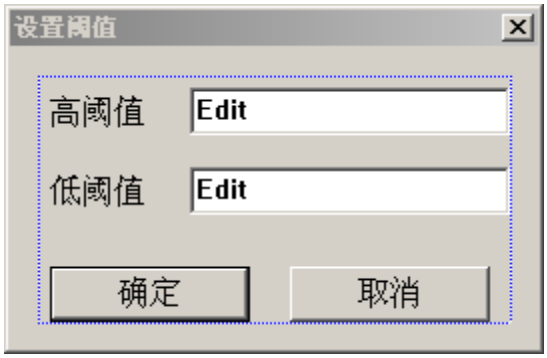


图 8.12 对话框界面设计

```
// Construction
public:
    CDialogParameter(CWnd* pParent = NULL);

    void SetLowThre(float lowThre) { m_LowThre = lowThre; }
    void SetHighThre(float highThre) { m_HighThre = highThre; }

    float GetLowThre() { return m_LowThre; }
    float GetHighThre() { return m_HighThre; }

// Dialog Data
    //{{AFX_DATA(CDialogParameter)
    enum { IDD = IDD_DIALOG_PARAMETER };
    float   m_HighThre;
    float   m_LowThre;
    //}}AFX_DATA

// Overrides
```

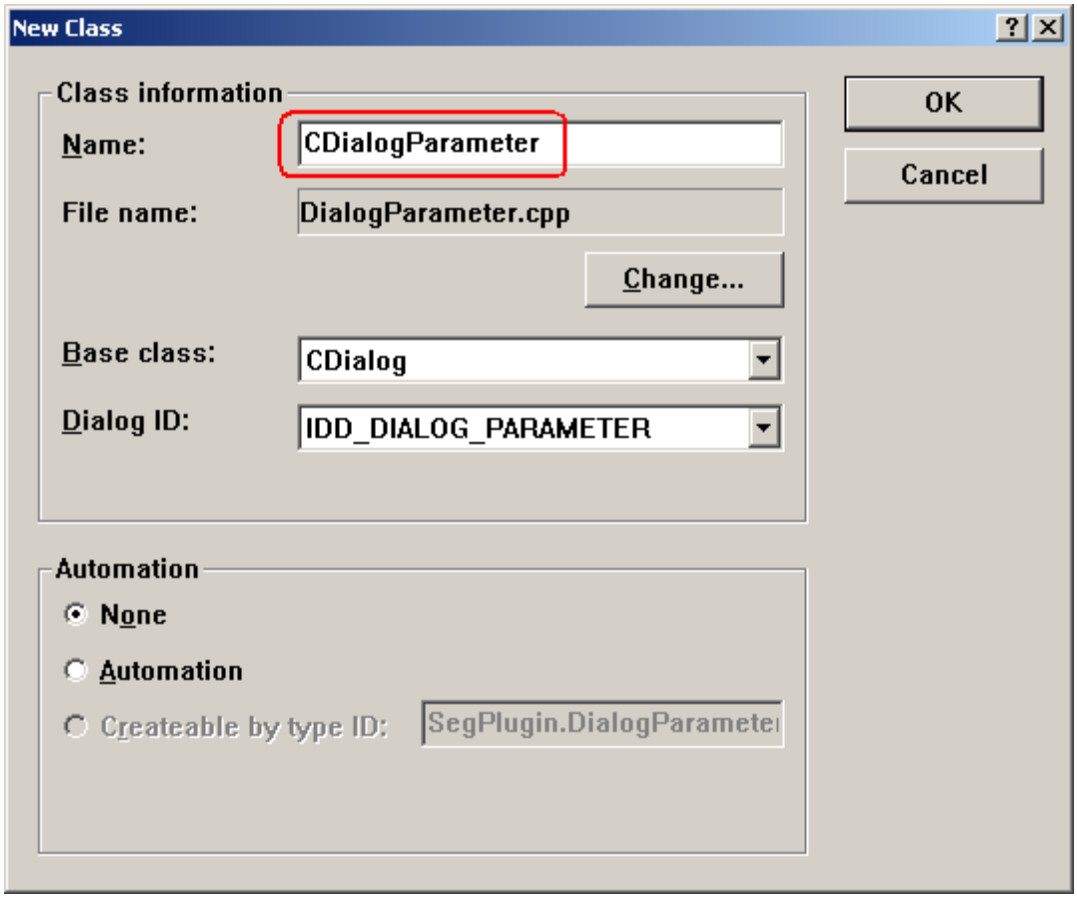


图 8.13 创建 CDialogParameter 类

```
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDialogParameter)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CDialogParameter)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

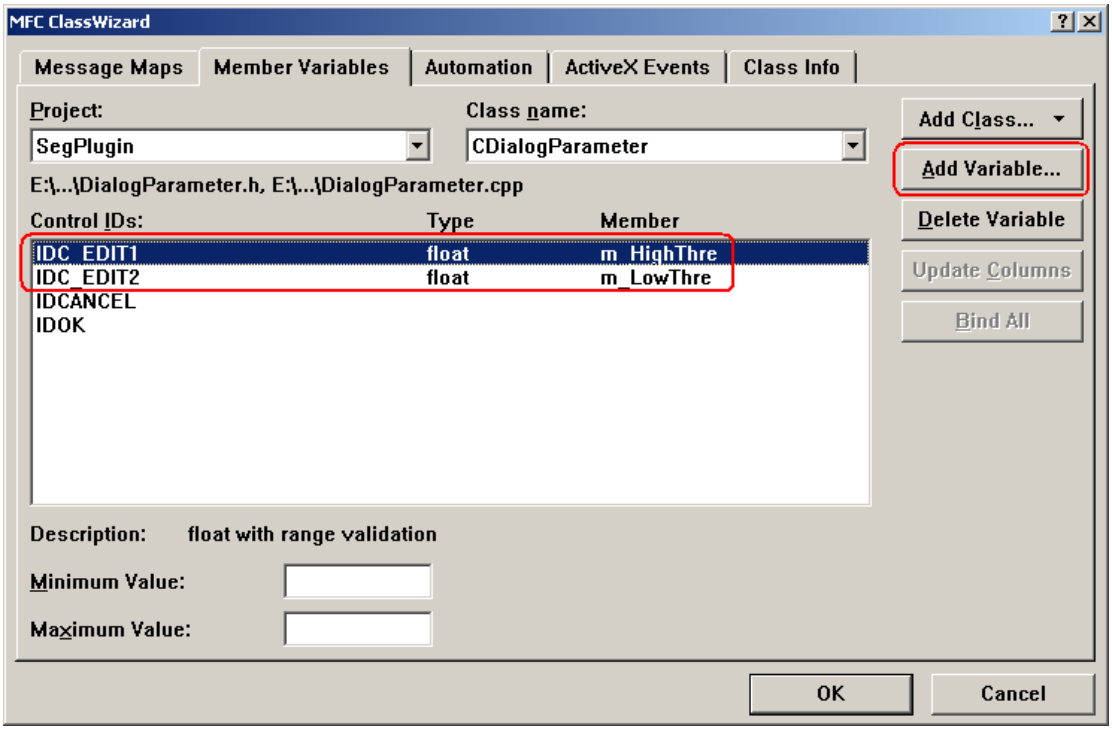


图 8.14 为编辑框控件添加成员变量

完成了烦琐的 GUI 图形界面工作，也有了 CDialogParameter 的辅助，下一步就是实现最重要的 CMySegPlugin 类中的 Show 函数了，其实现代码如下所示：

```
bool CMySegPlugin::Show(void)
{
    //MFC 的规定，必须先调用这个宏.
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    //弹出域值选择对话框.
    CDialogParameter paraDialog;
    paraDialog.SetLowThre(0.0f);
    paraDialog.SetHighThre(255.0f);

    if(paraDialog.DoModal() == IDOK)
    {
        //作实际的域值分割工作.
        return this->doSegmentation(paraDialog.GetLowThre(),
                                    paraDialog.GetHighThre());
```

```
    }

    return false;
}
```

里面用到了我们刚完成的 CDialogParameter 类，弹出一个域值设置对话框，如果用户设置完高、低域值并单击“确定”按钮以后， Show 函数内部调用 CMySegPlugin 类的私有成员函数 doSegmentation，把用户设置的高、低域值传进来，具体的工作由 doSegmentation 函数来完成，其实现代码如下所示：

```
bool CMySegPlugin::doSegmentation(float lowThre,
                                  float highThre)
{
    if(lowThre >= highThre)
        return false;

    medVolume *inVolume = this->GetInput();

    if(inVolume == NULL)
    {
        AfxMessageBox("输入数据为空");
        return false;
    }

    if(inVolume->GetNumberOfChannel() != 1)
    {
        AfxMessageBox("对不起，只支持单通道的数据");
        return false;
    }

    //生成输出数据（即分割后数据）.
    m_OutData = new medVolume;

    //设置分割后数据的属性，
    //和原始数据大部分一致.
    medVolume *outVolume = this->GetOutput();
    outVolume->SetWidth(inVolume->GetWidth());
    outVolume->SetHeight(inVolume->GetHeight());
    outVolume->SetImageNum(inVolume->GetImageNum());
```

```
outVolume->SetSpacingX(inVolume->GetSpacingX());
outVolume->SetSpacingY(inVolume->GetSpacingY());
outVolume->SetSpacingZ(inVolume->GetSpacingZ());
outVolume->SetNumberOfChannel(
                inVolume->GetNumberOfChannel());

//分割后的数据为二值数据,
//因此这里将数据类型设置为unsigned char (8bit).
outVolume->SetDataType(MED_UNSIGNED_CHAR);

//为分割后的数据分配实际内存.
unsigned char *outData
    = (unsigned char*) outVolume->Allocate();

//得到输入数据的内存指针.
void *inData = inVolume->GetRawData();

//根据输入数据的类型,
//使用模板函数来完成实际分割过程.
switch(inVolume->GetDataType())
{
    case MED_CHAR:
        t_ExecuteSegmentation((char*) inData,
                              outData,
                              inVolume,
                              lowThre,
                              highThre);
        break;

    case MED_UNSIGNED_CHAR:
        t_ExecuteSegmentation((unsigned char*) inData,
                              outData,
                              inVolume,
                              lowThre,
                              highThre);
        break;

    case MED_SHORT:
        t_ExecuteSegmentation((short*) inData,
                              outData,
                              inVolume,
```

```
                            lowThre,
                            highThre);
        break;

    case MED_UNSIGNED_SHORT:
        t_ExecuteSegmentation((unsigned short*) inData,
                            outData,
                            inVolume,
                            lowThre,
                            highThre);
        break;

    case MED_INT:
        t_ExecuteSegmentation((int*) inData,
                            outData,
                            inVolume,
                            lowThre,
                            highThre);
        break;

    case MED_UNSIGNED_INT:
        t_ExecuteSegmentation((unsigned int*) inData,
                            outData,
                            inVolume,
                            lowThre,
                            highThre);
        break;

    case MED_LONG:
        t_ExecuteSegmentation((long*) inData,
                            outData,
                            inVolume,
                            lowThre,
                            highThre);
        break;

    case MED_UNSIGNED_LONG:
        t_ExecuteSegmentation((unsigned long*) inData,
                            outData,
                            inVolume,
                            lowThre,
```

```
                                highThre);
            break;

        case MED_FLOAT:
            t_ExecuteSegmentation((float*) inData,
                                outData,
                                inVolume,
                                lowThre,
                                highThre);
            break;

        case MED_DOUBLE:
            t_ExecuteSegmentation((double*) inData,
                                outData,
                                inVolume,
                                lowThre,
                                highThre);
            break;

        default:
            AfxMessageBox("不支持的数据类型!");
            return false;
    }

    return true;
}
```

由于这个例子不使用 MITK，因此许多底层细节必须处理，所以程序稍微烦琐。为简化起见，这个例子只处理单通道的数据，如果是多通道的数据，可以先经过预处理，转换成单通道的数据。另外，这个地方大量地使用了 medVolume 所提供的 API 函数，请参考 8.1 节的介绍。在整个代码中，因为要处理不同数据类型的输入数据，所以最烦琐的地方在那个大的 switch-case 结构语句中，判断输入数据的类型，并据此将输入数据的内存指针强制转换成对应的指针类型，交给模板函数 t_ExecuteSegmentation 来处理，而 t_ExecuteSegmentation 的第一个参数是模板参数，可以被实例化成各种不同的指针类型，这也大大简化了编程的工作量。需要注意的是 t_ExecuteSegmentation 并不是 CMySegPlugin 的成员函数，而只是一个普通函数，所以它必须在 doSegmentation 函数前面被声明，其整个实现代码如下所

示：

```
template <class T>
void t_ExecuteSegmentation(T *inData, unsigned char *outData,
                           medVolume *inVolume,
                           float lowThresh, float highThresh)
{
    //得到图像的一些属性信息.
    int imageWidth = inVolume->GetWidth();
    int imageHeight = inVolume->GetHeight();
    int imageNum = inVolume->GetImageNum();
    int i, j, k;

    //循环遍历整个数据.
    for(k = 0; k < imageNum; k++)
    {
        for(j = 0; j < imageHeight; j++)
        {
            for(i = 0; i < imageWidth; i++)
            {
                //如果数据值在指定的域值范围内，
                //则输出的数据二值化为255.
                if(*inData>=lowThresh && *inData<=highThresh)
                {
                    *outData = 255;
                }
                //否则为0.
                else
                {
                    *outData = 0;
                }

                //输入数据和输出数据指针前移.
                inData++;
                outData++;
            }
        }
    }
}
```

最后，在“MySegPlugin.cpp”文件中所有函数体**之外**调用下面的宏生成几个必要的导出函数：

```
IMPLEMENT_SEGMENTATION(CMySegPlugin, "My Segmentation Plugin")
```

另外，不要忘记在“MySegPlugin.cpp”文件前面加上必要的头文件：

```
//PluginsSDK 头文件
#include "medVolume.h"

//对话框头文件
#include "DialogParameter.h"
```

8.3.3 将生成的 Plugin 加入 3DMed

经过了上面的步骤以后，现在可以编译整个工程，得到 SegPlugin.dll 文件，将其拷贝到 3DMed 安装目录的 Plugins 子目录下，这时运行 3DMed 主程序，加载完所有的插件以后，点击“分割算法”菜单时，就会发现我们的“My Segmentation Plugin”也在其中，如图 8.6 所示。

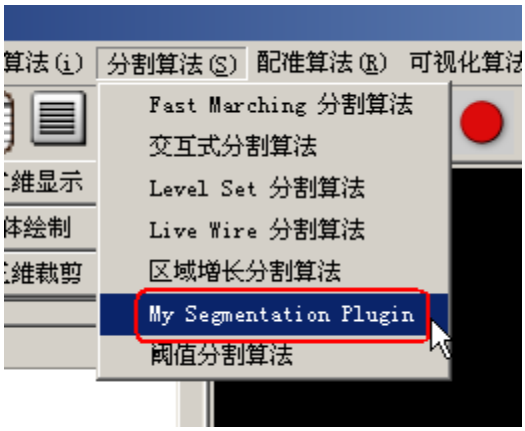


图 8.15　成功加载的 SegPlugin

当加载完体数据，点击“My Segmentation Plugin”后，就会弹出“设置阈值”对话框，让用户设置参数，如图 8.16 所示，设置完并点“确定”按钮以后， 3DMed 将利用分割后的数据进行三维重建并显示，其结果如图 8.17 所示。

图 8.16 SegPlugin 的运行界面

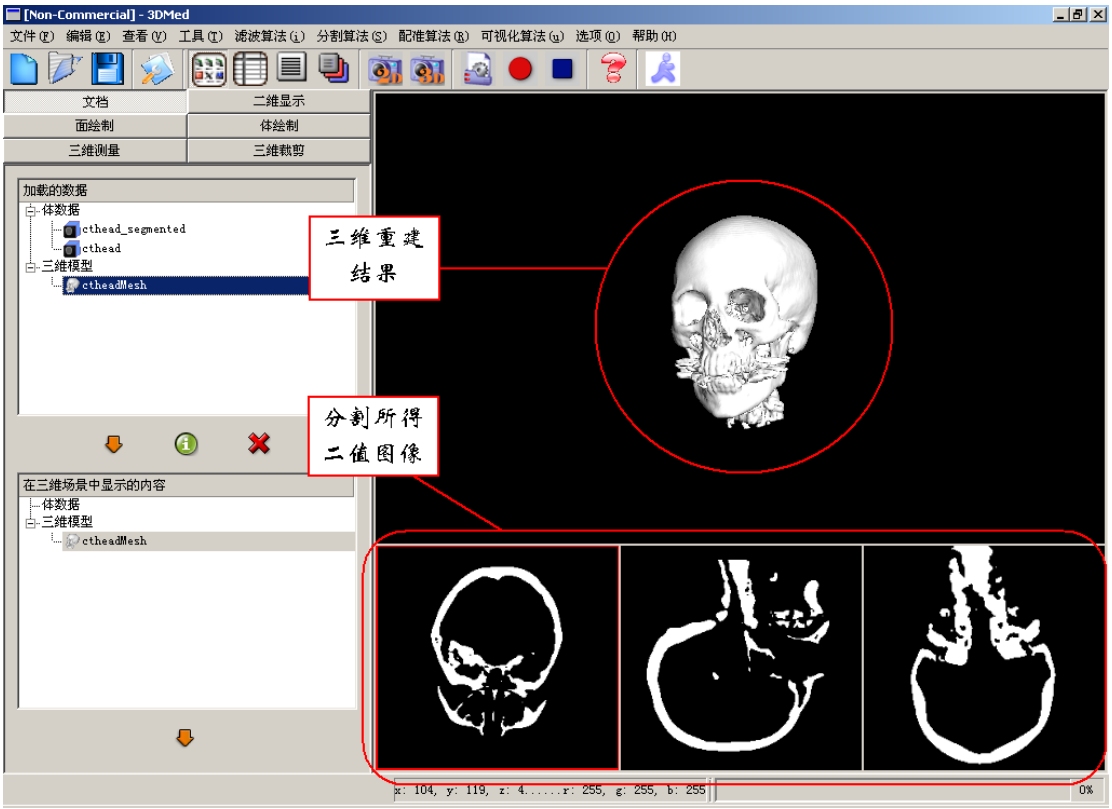


图 8.17 SegPlugin 的运行结果