MITK

MEDICAL IMAGING TOOLKIT

MITK 使用教程

MITK 使用教程

中科院自动化所医学影像研究室
联系人：田捷
E-mail：tian@doctor.com

第 1 章

使用 MITK 进行三维重建

本章将用一个例子来演示使用 MITK 进行三维断层图像的表面重建，例子使用 Microsoft 的 Visual Studio 6.0 作为开发环境，为了清晰起见，这里使用了比较多的截图，来演示 MITK 的一些基本概念和功能。

在讲述具体例子之前，首先理解 MITK 的一些基本概念和一些重要的类是非常必要的，下面将先介绍这些知识。

MITK 基础

MITK 的设计采用的是数据流的模型，以数据处理为中心，在概念上算法与数据是分开表达的，下面的框图表示了 MITK 对数据的整个处理流程。

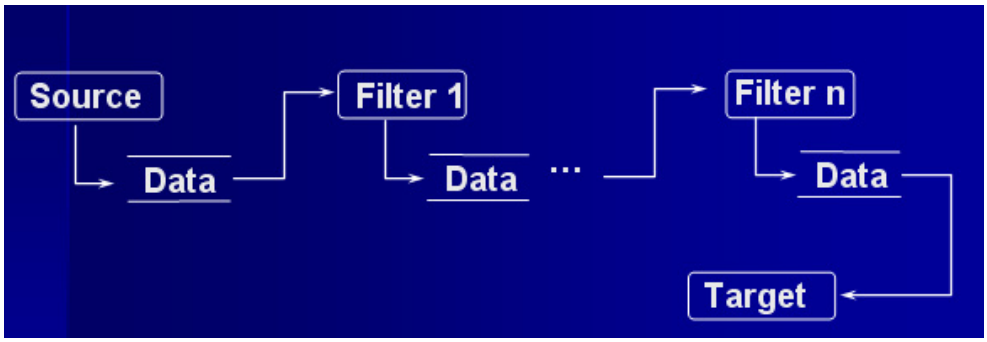


图 1 MITK 的整体框图

其中 Source、Filter、Data、Target 等都是一些高层抽象的类，封装的是一些概念，具体工作通过继承由它们的子类来完成。

Source: 负责生成数据，具体的例子包括从磁盘上读文件进来（在 MITK 中叫做 Reader），或者用某些算法生成数据，比如随机数产生器（在 MITK 中叫做 ProcedualSource）。

Filter: 对数据进行处理，也就是各种各样的算法。

Target：顾名思义，就是数据的终点。具体的例子包括：将得到的结果数据保存至磁盘（在 MITK 中叫做 Writer），或者将得到的结果在屏幕上显示出来（在 MITK 中叫做 View）。

Data: 上面的三种对象实际上都是封装的算法的行为，而 Data 就是对算法要处理的数据所进行的抽象。当然一个系统要能处理多种不同的数据，在 MITK 中是通过 Data 的各个具体的子类来描述不同的数据对象的。

清楚了这些高层的概念以后，剩下的就是如何将这些东西联系在一块，组成一个实用的系统。正如上面的图 1 所示的，系统通过这四种对象组成了一条流水线（Pipeline），数据起于 Source，终于 Target，中间经过多个 Filter 的处理。这种抽象关系足以描述我们大部分的以数据处理为中心的应用程序的需求，并且在概念上非常简单。

那么在具体写程序的时候，又如何联系起这个 Pipeline 呢？答案是通过 SetInput 和 GetOutput。这里举个例子，比如有两个连续的 Filter，一个叫 Filter1，一个叫 Filter2，可以参看图 1，那么可以通过 Filter2->SetInput(Filter1->GetOutput()) 来将两个 Filter 联系起来。那么对于 Filter2 来说，现在拿到了自己的输入数据，就可以调用 Filter2->Run() 来让 Filter2 干自己的活，干完以后，当然就可以通过 Filter2->GetOutput（）来将数据传给下一个 Filter 了。

通过这种所谓的 SetInput / GetOutput 方式，使用 MITK 的客户端程序员的工作将非常轻松，可以通过短短的几行代码来完成本来非常复杂的工作，这些将在下面的例子程序中得到体现。

三维重建需要使用的几个类

mitkVolume：首先，mitkVolume 是一个 Data，也就是说它是代表一个数据对象的。它是整个 MITK 中最重要的数据对象，代表一个三维断层图像数据，所有要处理的断层数据都必须先表达成 mitkVolume 的形式才能得到后续的处理。要使用它必须先包含 mitkVolume.h 头文件，关于它的接口函数的详细说明请参看 MITK 的帮助文件。

mitkMesh: mitkMesh 也是一个 Data，代表一个三维的几何数据对象，具体说就是三维重建以后物体表面的表达。要使用它必须先包含 mitkMesh.h 头文件，关于它的接口函数的详细说明请参看 MITK 的帮助文件。

mitkSurfaceModel: mitkSurfaceModel 是一个 Data Model，表示 Data 对象在三维场景中的显示模型。mitkSurfaceModel 即是对应于 mitkMesh 对象的显示模型。

mitkMarchingCubes: mitkMarchingCubes 是一个 Filter，也就是说它是代表一个算法对象的。它封装了三维重建算法，是我们这里要使用的主要的算法。要使用它必须先包含 mitkMarchingCubes.h 头文件，关于它的接口函数的详细说明请参看 MITK 的帮助文件。

mitkView: mitkView 是一个 View，封装的是屏幕上的一个窗口，它具有能显示多个 Model 对象的能力，并且支持鼠标的交互式操作。要使用它必须先包含 mitkView.h 头文件，关于它的接口函数的详细说明请参看 MITK 的帮助文件。

三维重建实例

首先，我们先看一下这个例子所完成的功能，下面这张图是例子运行时的主界面。

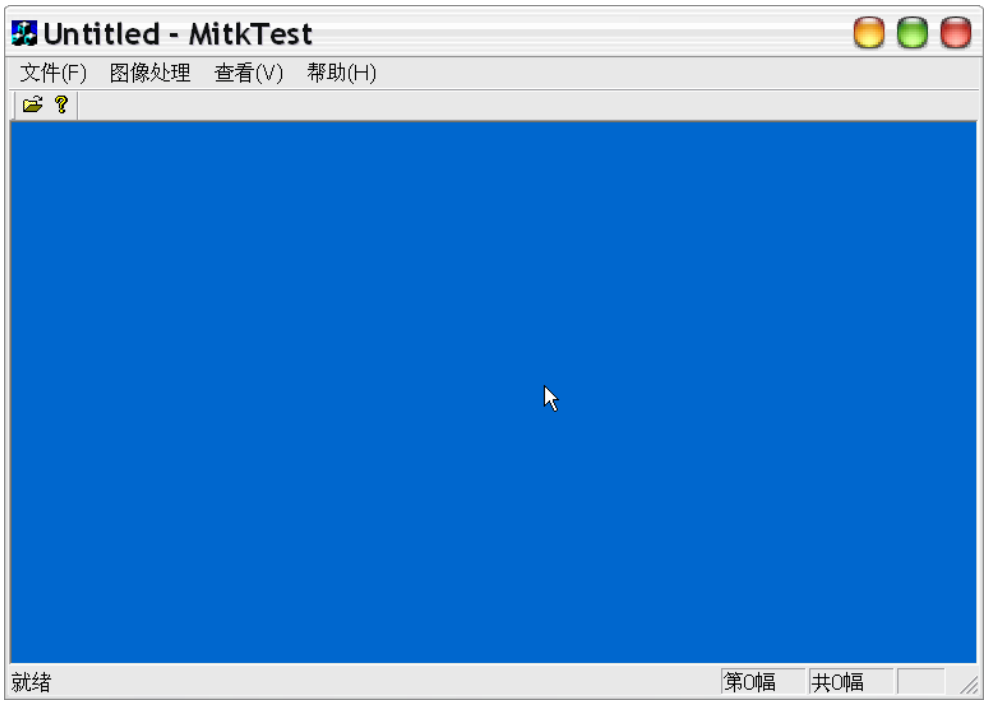


图 2 主界面

然后选“文件”菜单下的“打开体数据”。

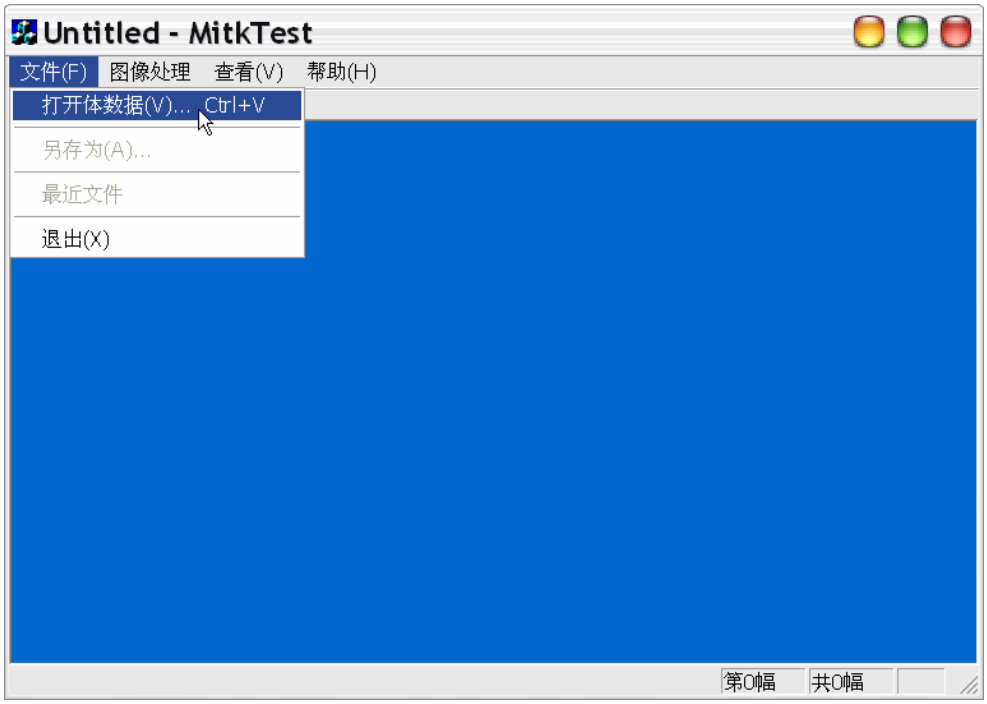


图 3 打开体数据

这时弹出一个打开文件的对话框。

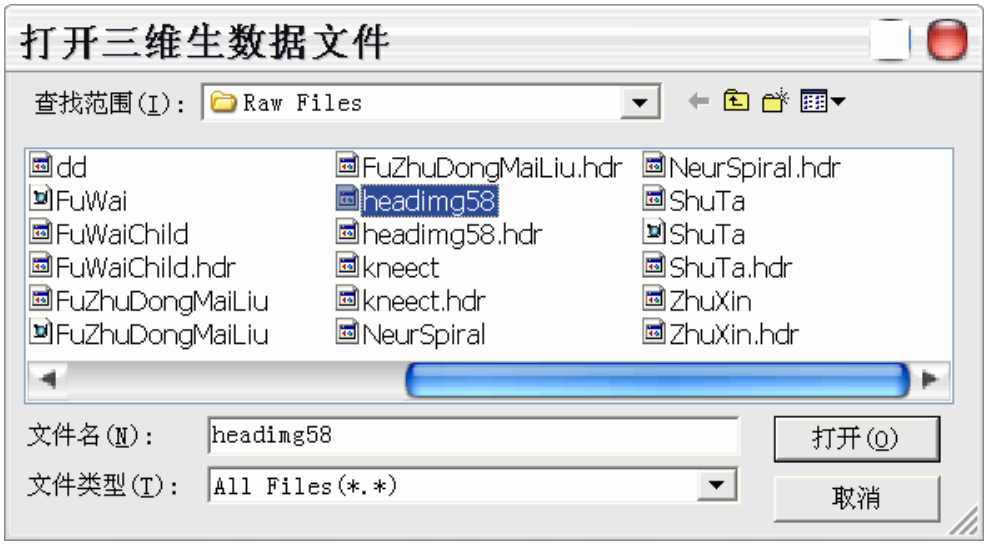


图 4 打开文件对话框

因为要打开的是生数据文件，没有任何头信息，所以当你选择了一个数据文件并打开的时候，下面的对话框出现：

图 5 设置生数据文件的一些必要信息

当你填完这些信息以后，例子程序已经得到了足够的信息，可以把这个文件打开，如果没有意外的话，在等待一会儿以后这个文件将会被成功地加载到内存中。这时再选“图像处理”下面的“三维重建”。

图 6 三维重建

这时会弹出一个对话框，要求你输入一个双阈值。

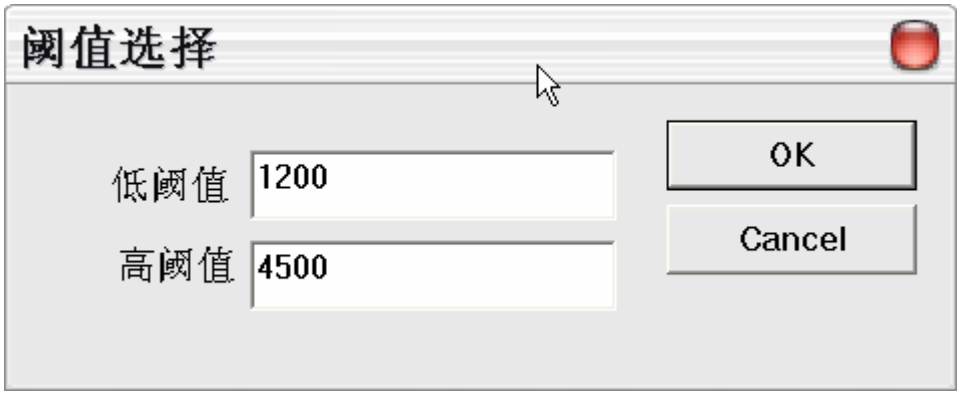


图 7 双阈值选择

阈值的选择取决于要抽取的物质的性质，需要一些先验知识。当输入合适的阈值以后，点“OK”，过一会儿，三维重建的结果就会显示出来，你可以用鼠标进行操作：左键旋转、中键平移、右键缩放。

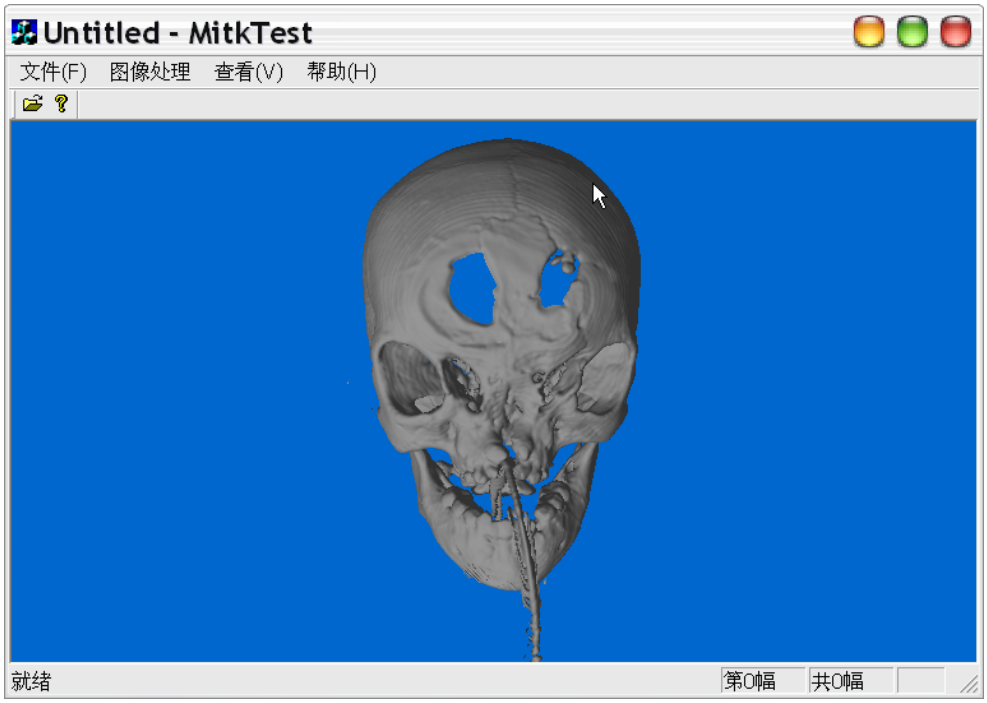
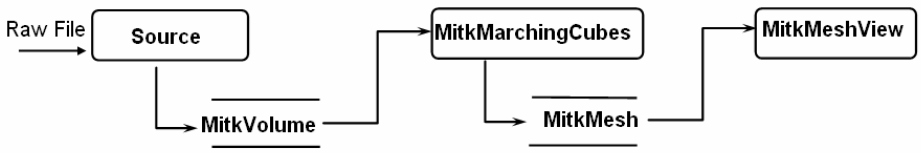


图 8 三维重建结果

好了，了解了例子程序的功能以后，现在可以考虑如何用上面介绍的 Pipeline 的概念来描述这个例子程序，见下图：



要注意的是，在这里为了演示一下 mitkVolume 的接口函数的使用（因为它是一个非常重要的数据类），在 Source 这一块并没有使用 MITK 中的 mitkRawReader 类，而是直接从文件里面读数据，然后填充至 mitkVolume 中。

经过了前面漫长的准备过程，终于到了该讲实际例子的时候，我相信这时你应该已经对 MITK 有了一个比较粗浅的认识了，如果你感觉前面非常抽象的话，那么现在就到了实践的时候了。

第一步，是在 Visual C++ 6.0 的 IDE 开发环境中新建一个 MFC 工程，单文档界面的，工程名字叫做 MitkTest，这一步的操作过程我想不用再多说了。

第二步，建立界面资源，如果你不想从头再来的话，可以直接使用例子源码中的现成的东西。下面将直接转入跟 MITK 相关的部分。

第三步，在 MitkTestDoc.h 里面添加两个成员变量，用来记录数据对象。当然别忘了包含头文件。

```
// MitkTestDoc.h : interface of the CMitkTestDoc class

#include "mitkVolume.h"
#include "mitkMesh.h"

class CMitkTestDoc : public CDocument
{
        … … …
        mitkVolume *m_Volume;
        mitkMesh   *m_Mesh;
        … … …
};
```

第四步，在 MitkTestView.h 里面添加两个成员变量，其中 m_Surface 表示 Mesh 数据的显示模型，m_View 则用于显示这个模型。不要忘了加上包含文件。

```
// MitkTestView.h : interface of the CMitkTestView class

#include "mitkSurfaceModel.h"
#include "mitkMesh.h"
#include "mitkView.h"

class CMitkTestView : public CView
{
        … … …
        void SetInput(mitkMesh* mesh);
```

```
        mitkSurfaceModel *m_Surface;
        mitkView *m_View;
        …… ……
};
```

这里的SetInput函数仅仅是为了把m_ Surface给封装起来而提供的接口函数，实际上它只是调用了m_Surface的SetData函数，见下：

```
void CMitkTestView::SetInput(mitkMesh* mesh)
{
        m_View->SetData (mesh);
}
```

第五步，在 MitkTestDoc 里面添加消息处理函数来响应“打开体数据”和“三维重建”菜单命令。

```
afx_msg void OnFileOpenVolume();
afx_msg void OnImageReconstruction();
```

下面是“打开体数据”的菜单命令处理代码：

```
void CMitkTestDoc::OnFileOpenVolume()
{
        // 弹出打开文件对话框
        COpenFileDialog fileDialog;
        fileDialog.SetTitle("打开三维生数据文件");
        fileDialog.SetFilter("Raw Data(*.raw)\0*.raw\0All
        Files(*.*)\0*.*\0\0");

        //如果用户选择了一个文件并按“OK”
        if(fileDialog.Run())
        {
                int nFilterIndex = fileDialog.GetFilterIndex();
                CString szFileName = fileDialog.GetPathName();

                clearVolume();  // 清除上次打开的Volume
                // 从文件里面读数据并生成一个Volume
                m_Volume = readRawFile(szFileName);
        }
}

// 从文件里面读数据并生成一个Volume
// 这里演示了如何通过mitkVolume提供的接口函数来手工地生成一个
// Volume
mitkVolume* CMitkTestDoc::readRawFile(const char* FileName)
{
        CRawSetDlg dlg;
```

```
        // 弹出如图5所示的对话框来获得生数据的信息
        if (dlg.DoModal() == IDOK)
        {
                m_Volume = new mitkVolume;
                unsigned char *buf = NULL;
                FILE *fp = NULL;

                // 从对话框取得长、宽、高并赋给Volume
                m_Volume->SetWidth(dlg.m_Width);
                m_Volume->SetHeight(dlg.m_Height);
                m_Volume->SetImageNum(dlg.m_ImageNum);

                // 从对话框取得像素物理尺寸并赋给Volume
                m_Volume->SetSpacingX(dlg.m_SpacingX);
                m_Volume->SetSpacingY(dlg.m_SpacingY);
                m_Volume->SetSpacingZ(dlg.m_SpacingZ);

                // 从对话框取得数据通道数并赋给Volume
                m_Volume->SetNumberOfChannel(dlg.m_ChannelNum);

                // 从对话框取得数据类型并赋给Volume
                m_Volume->SetDataType(dlg.m_DataType);

                // Volume获得足够多的信息后，就可以分配内存来容纳
                // 数据
                buf = (unsigned char*) m_Volume->Allocate();


                // 打开磁盘上的文件
                fp = fopen(FileName, "rb");
                if(fp == NULL)
                {
                        delete m_Volume;
                        return NULL;
                }
                // 读进Volume的内存里面
                fread(buf, 1, m_Volume->GetActualMemorySize(), fp);
                fclose(fp);

                return m_Volume; // 一切搞定，正常返回
        }

        return NULL;
}
```

下面是“三维重建”的菜单命令处理代码：

```
void CMitkTestDoc::OnImageReconstruction()
{
	// 如果Volume为空，直接返回
	if(!m_Volume) return;

	CThresholdSelectionDialog dlg;
	// 弹出阈值选择对话框
	if(dlg.DoModal() == IDOK)
	{
		// 使用mitkMarchingCubes来进行三维重建
		mitkMarchingCubes mc;
		// 首先连接Pipeline，设置算法的输入
		mc.SetInput(m_Volume);
		// 设置算法必须的参数，高低阈值
		mc.SetThreshold(dlg.m_LowThre, dlg.m_HighThre);
		// 运行算法
		mc.Run();

		// 清除上次生成的Mesh
		clearMesh();
		// 记录本次生成的Mesh
		m_Mesh = mc.GetOutput();

		// 得到MitkTestView的指针
		POSITION pos = this->GetFirstViewPosition();
		CMitkTestView *view = (CMitkTestView*) this-
		>GetNextView(pos);
		// 连接Pipeline，将Mesh送给MeshView显示
		view->SetInput(m_Mesh);
		// 更新View的显示
		view->Invalidate();
	}
}
```

第六步，在 MitkTestView 里面添加消息处理函数来处理 OnCreate 和 OnSize 事件，以便 mitkMeshView 能填满整个 mitkTestView 的客户区。代码如下：

```
int CMitkTestView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CView::OnCreate(lpCreateStruct) == -1)
		return -1;


	// 得到客户区的宽和高
```

```
	RECT clientRect;
	this->GetClientRect(&clientRect);
	int wWidth = clientRect.right - clientRect.left;
	int wHeight = clientRect.bottom - clientRect.top;


	// 生成mitkView
	m_View = new mitkView;
	// 设置其父窗口为mitkTestView
	m_View->SetParent(GetSafeHwnd());
	// 设置其位置与大小，以填满整个客户区
	m_View->SetLeft(0);
	m_View->SetTop(0);
	m_View->SetWidth(wWidth);
	m_View->SetHeight(wHeight);
	// 显示出来
	m_View->Show();

	// 生成可容纳Mesh对象的mitkSurfaceModel
	m_Surface = new mitkSurfaceModel;

	// 设置用于显示Surface Model的一些材质属性
	m_Surface->GetProperty()->SetAmbientColor(0.9f, 0.9f, 0.9f, 1.0f);
	m_Surface->GetProperty()->SetDiffuseColor(0.9f, 0.9f, 0.9f, 1.0f);
	m_Surface->GetProperty()->SetSpecularColor(0.0f, 0.0f, 0.0f, 1.0f);
	m_Surface->GetProperty()->SetSpecularPower(100.0f);
	m_Surface->GetProperty()->SetEmissionColor(0.0f, 0.0f, 0.0f, 0.0f);

	//将这个Surface Model加入View中
	m_View->AddModel(m_Surface);

	return 0;
}

void CMitkTestView::OnSize(UINT nType, int cx, int cy)
{
	CView::OnSize(nType, cx, cy);

	// 设置其位置与大小，以填满整个客户区
	m_View->SetLeft(0);
	m_View->SetTop(0);
	m_View->SetWidth(cx);
	m_View->SetHeight(cy);
	// 更新显示
	m_View->Update();
}
```

第七步，经过上面的六步，主要工作已经完成，其它剩下的就是析构函数里面的内存释放工作，需要注意的是从 mitkObject 继承下来的对象不能用 delete 直接删除，必须调用成员函数 Delete()进行删除，具体请参看源代码，这里就不多讲了。