**第 十 章 命令行模块的扩展与开发**

**10.1 命令行模块的界面开发**

**10.2 命令行模块的功能开发**

ITK库使用的简单介绍

在介绍如何编写自己的SLICER CLI模块之前，你需要先了解ITK库的使用和编写方法，这里简单地对ITK做出讲解

ITK 是一个开放源码、面向对象的软件系统，提供一个医学图像处理、图像分割与配准的算法平台。ITK是用C++实现的，结构庞大、复杂，但是有C++基础的读者可以很快了解它的面向对象和执行基本方法，并且灵活应用。限于篇幅，本篇将不会对ITK进行非常详细系统地介绍，而只是选取其中利于作者读懂和编写CLI的部分进行大致地讲述，详细内容请作者自行阅读ITK相关书籍。

在阅读和编写CLI代码之前，你最需要了解的是ITK的数据对象、处理对象和管道功能，熟练掌握这几个方面的内容，有助于你较快的上手

ITK是基于C++的面向对象编程，并且大量使用了泛型编程技术，因此在ITK代码中，有大量自定义的类和容器用于表示和容纳数据对象，但是从类型上来分主要是两大类，其中一大类为数据对象类，例如itk::Image就是表示图像数据的基本类，itk::PointSet是表示点集数据的基本类。而另外一个大类则是处理对象类，它也是构成管道的元素之一。所谓的处理对象类，包括了各种对图像进行处理的类，例如文件读取类itk::ImageFileReader和滤镜处理类itk::BinaryThresholdImageFilter等等。

数据处理管道则表示利用处理对象处理数据对象的过程，并且多个处理对象可以组织成一条处理管道。管道可以记录并维护数据对象和处理对象的状态，只有需要的时候才执行响应的处理对象。ITK的管道支持线程处理和流式处理。

下图演示一个简单的管道示意图，从这个示意图中，我们可以看到数据是如何在各个处理对象中之间传递并进行处理的

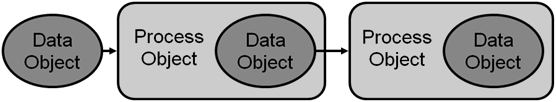


图1 管道简单示意图

ITK中的管道一般按照如下步骤创建

1, 确定处理对象的类类型，例如重采样滤镜的类类型为itk::ResampleImageFilter，二值门限处理滤镜的类类型为itk::BinaryThresholdImageFilter。注意一旦决定了要用的处理对象类型，就要及时添加包含相关头文件，例如如果决定要使用itk::ResampleImageFilter对象，则必须包含头文件"itkResampleImageFilter.h"；

2, 使用处理对象的静态方法New创建对象的实例，例如typename ReaderType::Pointer reader = ReaderType::New();就创建了一个reader对象的实例；

3, 使用处理对象的方法SetInput(upperstreamfilter->GetOutput)将多个处理对象连在一起，同样地利用resample->SetInput(reader->GetOutput() )就使得图像读取对象和重采样滤镜对象前后连接在了一起；

4, 设置处理对象所需的算法参数。不同的处理对象具有不同的算法流程，因此设置参数的方法和函数各不相同，要详细了解所用处理对象的参数设置方法，请详阅ITK的函数手册；

5, 调用处理对象的Update()方法，触发管道的执行操作

下图举例演示一个完整的管道处理执行顺序。其中输入图像的获取由itk::ImageReader这个处理对象处理，输出图像的保存由itk::ImageWriter处理。而管道中的多个Filter与itk::ImageReader一样都是处理对象例如我们后面会提到的itk::ResampleImageFilter等，可以根据图1的管道创建方法将多个处理对象，并将它们连接起来组成一个完整的管道。在图2所示的这个管道中，具有2个分支，最后调用update()完整的管道才得到了执行

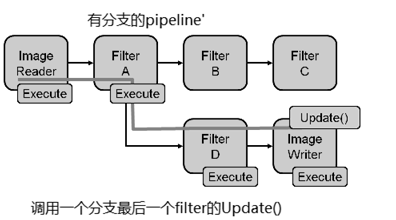


图2 举例一个完整的管道流程

作为图像处理库，ITK提供一些常用的图像处理类，供用户调用来高效快速地完成图像处理任务。常用的简单的滤镜处理类有如下：

1. 门限处理：门限处理(Thresholding)操作是基于指定一个或多个值（称为阈值，Threshold）来改变或确定像素值的。在ITK中，提供了一个二值门限处理类，用户需要指定上下门限，从而将一个图像转化成一个二值图像
2. 边缘检测：ITK提供了最为常用的边缘检测方法，即canny边缘检测
3. 投射和亮度映射：投射用于对像素数据类型进行强制转换，等同于C风格代码中的类型强制转换。而亮度映射则包含了线性映射和非线性映射。例如利用变换函数，将图像的像素值转换到适合于图像显示的一个特定范围内
4. 梯度：梯度主要用来帮助检测对象轮廓和分离均匀区域。ITK提供了类有三种：简单的梯度强度，带高斯模糊的梯度强度，不带高斯模糊的偏导数
5. 二阶微分：包括二阶高斯递归滤波器，拉普拉斯滤波器
6. 邻域滤波器：邻域滤波通过对当前像素附近的小块区域范围内的像素进行处理，从而得到输出值，包括均值滤波、中值滤波和形态学（包括腐蚀、膨胀等）
7. 平滑滤波器：又称模糊滤波，包括各种常见的模糊（包括高斯模糊、二项式模糊等）、局部模糊和保留边缘的模糊（包括在向量图像中的保留边缘模糊和在彩色图像中的保留边缘模糊）
8. 距离映射：即距离变换，它是通过计算空间点对目标点的距离，把图像变换为灰度图像的方法，其中每个栅格的灰度值等于它到最近目标点的距离。
9. 几何变换：几何变换即对图像进行翻转、旋转、重采样等变换操作。
10. 频域变换：频域变换将图像从图像空间转换到频域空间，包括快速傅立叶变换(FFT)以及频域平滑滤波

当然和其他的图像处理库一样，除了提供上述较为简单常用的处理类之外，ITK也提供了一些较为复杂处理类，用于配准、分割和统计等工作。例如在配准中，提供了多种尺度变换和插值方法；在分割中，提供了区域生长和分水岭分割等经典方法；在统计中则提供了直方图分析、贝叶斯分类等高级方法。读者可以自行查阅ITK手册了解其原理和具体用法

**深入分析CLI模块编写：**

**比较简单的例子**

经过上一节对ITK的介绍，本节我们来看一个比较简单的CLI模块：Gaussian Blur Image Filter模块，详细讲解如何利用ITK库来编写CLI模块。其代码请查阅\Modules\CLI\AddScalarVolumes\GaussianBlurImageFilter.cxx。

该模块的用途就是对一副图像使用高斯模糊处理，为了更加形象地阐述这个模块的处理过程，我们给出了其管道流程图3。由于其中只采用了一个高斯滤波器，因此它的参数非常简单，如图4所示的SLICER GUI界面中，仅仅只需要设定模糊参数Sigma、输入图像Input Volume和输出图像Output Volume三个参数而已。



图3 Gaussian Blur Image Filter 管道图

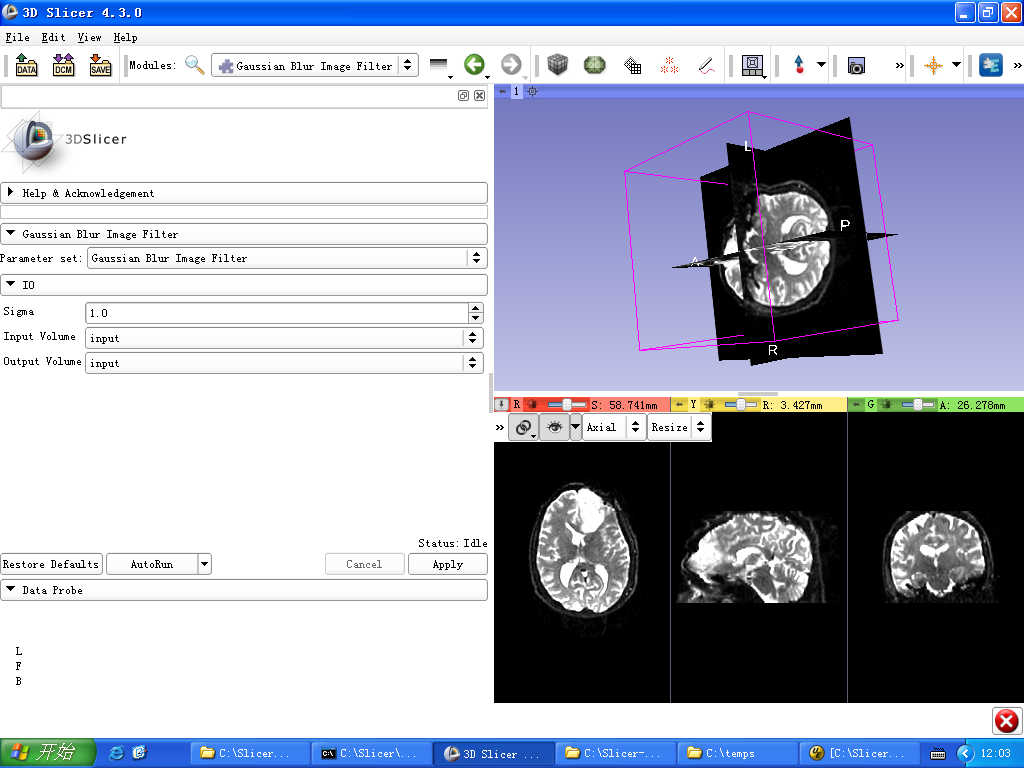


图4 SLICER用户界面中的相关参数设定

其中模糊参数Sigma()的定义由如下高斯函数给出：

该高斯函数用于计算距离目标像素为(x,y)像素点上的权重值，该权重值用于对领域像素值进行加权平均，从而得到该目标像素的结果值

接下来我们结合上述图对该模块源代码进行详细的讲解：

首先要包含程序中所需要的相关头文件，因为这里我们用到了多个处理对象（滤镜），因此这些对象相关的头文件都应该被包含进去（源代码确实没有包含itkImageReader.h，可能在其他头文件中已经被包含了）

1. #include "itkImageFileWriter.h" //文件读写对象
2. #include " itkSmoothingRecursiveGaussianImageFilter.h" //平滑高斯滤波器
3. #include "itkPluginUtilities.h"
4. #include " GaussianBlurImageFilterCLP.h" //CLP模板生成的头文件

在main函数中，PARSE\_ARGS宏用于获得用户在3D SLICER GUI界面选择的参数，即图4中的sigma, input volume和output volume。PARSE\_ARGS宏定义可以参考CLP头文件AddScalarVolumesCLP.h，它将获取inputVolume等用于指定的输入参数值供程序处理使用。

1. int main( int argc, char \* argv[] )
2. {
3. PARSE\_ARGS;

……….

………

使用GetImageType来获得当前输入图像的像素类型，其中inputVolume就是利用宏PARSE\_ARGS得到的输入图像的路径

1. itk::GetImageType(inputVolume, pixelType, componentType);

根据传出的componentType的值的不同，由switch条件分支分别处理

1. switch( componentType )
2. {

根据componentType的数据类型，可以设置DoIt函数的第三个传入参数，以便让同一个函数DoIt来处理不同类型的图像。例如普通的NRRD图像，像素类型为UINT类型，因此DoIt的第三个参数为static\_cast<unsigned int>(0)

1. case itk::ImageIOBase::UINT:
2. return DoIt( argc, argv, static\_cast<unsigned int>(0) );
3. break;

………

}

由上面的分析可见，处理的核心函数为DoIt，下面我们就详细解析一下DoIt函数。首先同样利用了PARSE\_ARGS获得用户在3D SLICER GUI界面设置的参数

1. int DoIt( int argc, char \* argv[], T )
2. {
3. PARSE\_ARGS;
4. typedef T InputPixelType; //根据传入T的类型来定义输入像素类型
5. typedef T OutputPixelType; //根据传入T的类型来定义输出像素类型

利用上述像素类型来定义输入和输出的图像类型，维度为3维

1. typedef itk::Image<InputPixelType, 3> InputImageType;
2. typedef itk::Image<OutputPixelType, 3> OutputImageType;

接下来定义了itk::ImageFileReader 类来从一个文件中读取图像

1. typedef itk::ImageFileReader<InputImageType> ReaderType;

定义itk::imageFileWriter类用于保存输出图像

1. typedef itk::ImageFileWriter<OutputImageType> WriterType;

接下来的代码定义了一个平滑的高斯滤波器，用于将图像和高斯内核做卷积实现模糊操作。该滤镜具体的算法请自行查阅ITK官方手册。

1. typedef itk::SmoothingRecursiveGaussianImageFilter<
2. InputImageType, OutputImageType> FilterType;

调用对象的New()方式来创建对象实例

1. typename ReaderType::Pointer reader = ReaderType::New();

实例化reader对象以后，调用其SetFileName方法设置输入文件路径

1. reader->SetFileName( inputVolume.c\_str() );

接下来创建并设置高斯滤镜，同样是利用其new方法创建其实例。而reader->GetOutput()方法我们之前也讲到了是用来将多个处理对象连接起来，这里是把reader对象和高斯滤波对象连接起来

1. typename FilterType::Pointer filter = FilterType::New();
2. filter->SetInput( reader->GetOutput() );
3. filter->SetSigma( sigma );

创建输出图像对象实例并设置其路径，传递滤镜输出结果并显示。其中outputVolume就是在GUI界面设置的同名的参数

1. typename WriterType::Pointer writer = WriterType::New();
2. writer->SetFileName( outputVolume.c\_str() );
3. writer->SetInput( filter->GetOutput() );
4. writer->SetUseCompression(1); //对输出图像启用压缩
5. writer->Update();
6. return EXIT\_SUCCESS;
7. }

下面利用SLICER对该模块进行简单的演示，图5显示的是第一个输入图像即input volume为脑部的CT图像，尺寸大小为256\*256\*51。在图4的用户设置中我们把sigma设置为20，然后设置好输入和输出之后点击”apply”，即得到了图6。可以看到高斯滤镜对所有方向上的切片都进行了模糊处理。

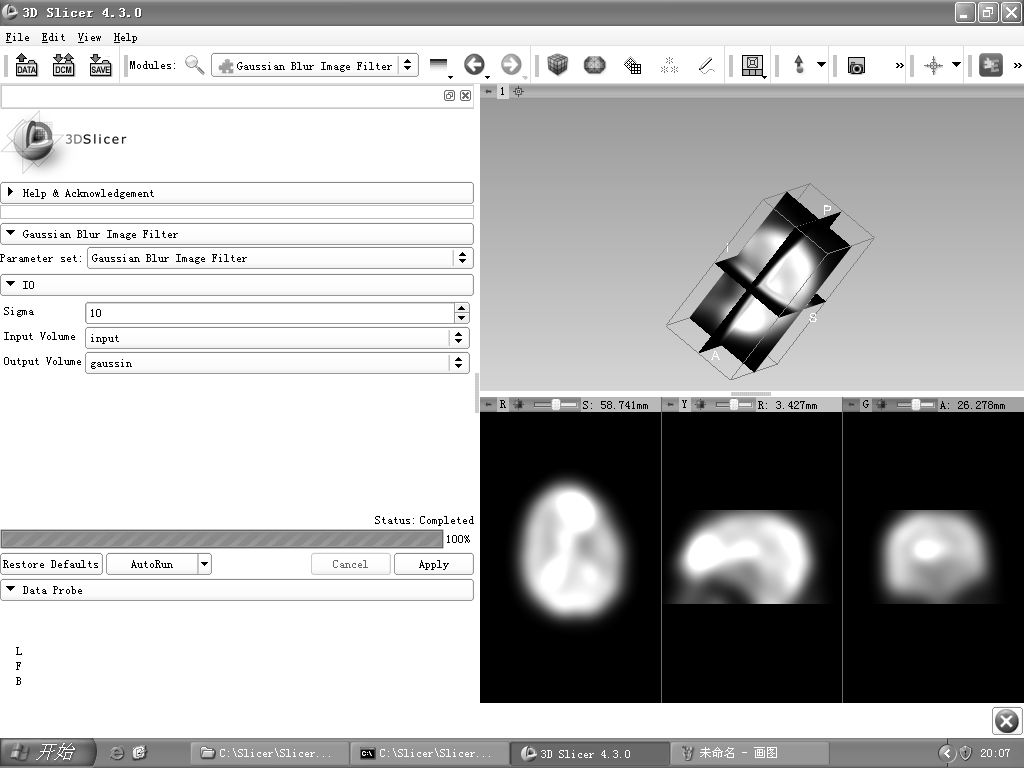
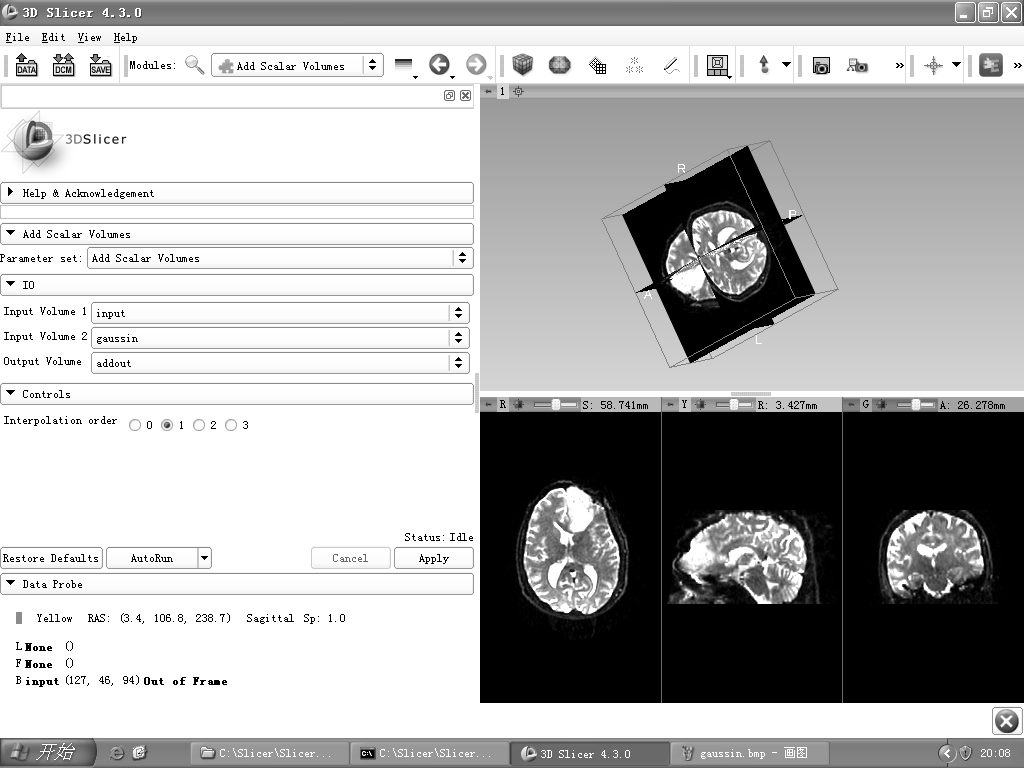


图5 输入图像 图6 输出图像

**稍微复杂一点的例子**

经过对上面简单例子的讲解，相信读者对ITK的管道操作已经有了大致的了解，下面我们再举一个稍微复杂一点的例子，示例的AddScalarVolume模块，其代码请查阅\Modules\CLI\AddScalarVolumes\AddScalarVolumes.cxx。

在这个例子中，我们将两个不同的医学图像进行像素值相加求和操作。由于医学图像是由二维图像切片构成的，因此这两个输入源在不同坐标轴方向的切片数目和尺寸可能不相同，这就要求我们必须要考虑在相加之前做配准操作。此例中包含了更多的滤波处理对象，管道相对与上一个例子自然要稍稍复杂一点。为了更加形象的阐述这个模块的处理过程，我们和之前一样，将其流程和管道具体化为图3。可见其中包含了重采样滤波器(resample filter)和插值滤波器(interpolate filter)，图6是其对应的SLICER GUI用户设置参数的界面，参数包括两个输入图像的路径、一个输出图像的路径和插值参数。



图7 AddScalarVolume的管道和流程图

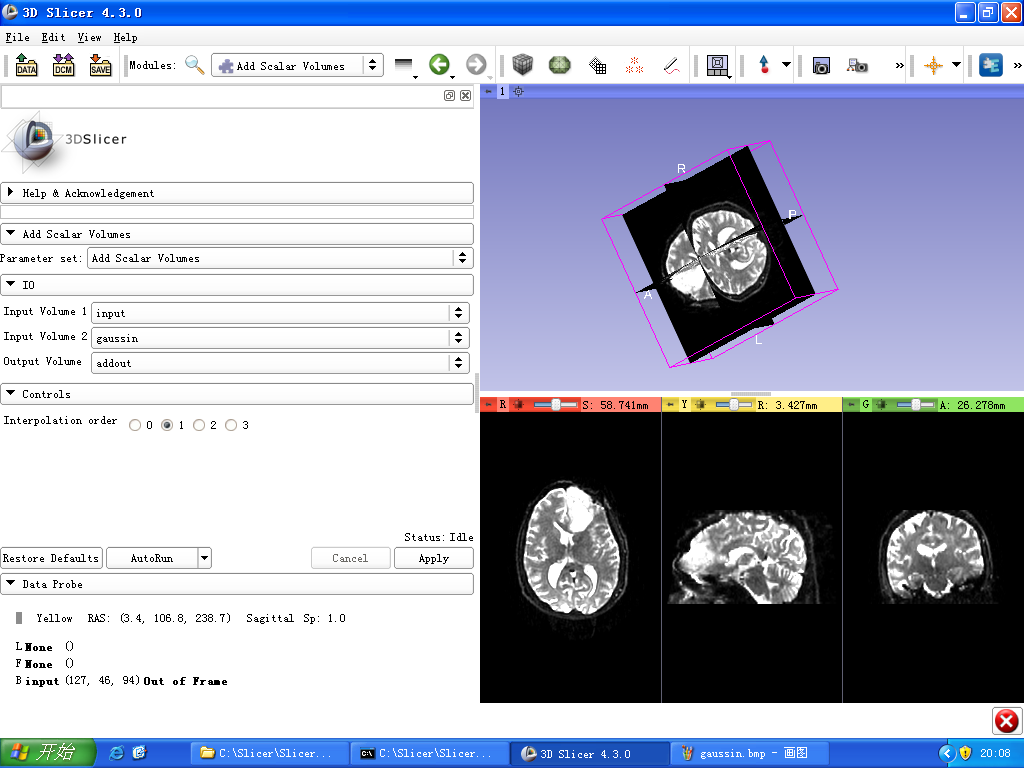


图8 SLICER用户界面中的相关参数设定

图7中，黄色部分为调用的处理对象，黑色实线的部分代表了管道的方向，而虚线则是调用滤镜对象所需要参数设置的流程图，我们将结合图7对整个源代码进行详细的讲述

首先要包含程序中所需要的相关头文件，因为这里我们用到了多个处理对象（滤镜），因此这些对象相关的头文件都应该被包含进去

1. #include "itkImageFileWriter.h" //文件读写对象
2. #include "itkResampleImageFilter.h" //重采样滤镜对象
3. #include "itkConstrainedValueAdditionImageFilter.h" //相加滤镜对象
4. #include "itkBSplineInterpolateImageFunction.h" //B样条插值对象
5. #include "itkPluginUtilities.h"
6. #include "AddScalarVolumesCLP.h" //CLP模板生成的头文件

在main函数中，PARSE\_ARGS宏用于获得用户在3D SLICER GUI界面选择的参数，例如选择输入图像输出图像、插值参数等。PARSE\_ARGS宏定义可以参考CLP头文件AddScalarVolumesCLP.h，它将获取inputVolume1等用于指定的输入参数值供程序处理使用

1. int main( int argc, char \* argv[] )
2. {
3. PARSE\_ARGS;

……….

………

使用GetImageType来获得当前输入图像的像素类型，其中inputVolume1就是利用宏PARSE\_ARGS得到的输入图像的路径

1. itk::GetImageType(inputVolume1, pixelType, componentType);

根据传出的componentType的值的不同，由switch条件分支分别处理

1. switch( componentType )
2. {

根据componentType的数据类型，可以设置DoIt函数的第三个传入参数，以便让同一个函数DoIt来处理不同类型的图像。例如普通的NRRD图像，像素类型为UINT类型，因此DoIt的第三个参数为static\_cast<unsigned int>(0)

1. case itk::ImageIOBase::UINT:
2. return DoIt( argc, argv, static\_cast<unsigned int>(0) );
3. break;

………

}

由上面的分析可见，处理的核心函数为DoIt，下面我们就详细解析一下DoIt函数。首先同样利用了PARSE\_ARGS获得用户在3D SLICER GUI界面选择的参数

1. int DoIt( int argc, char \* argv[], T )
2. {
3. PARSE\_ARGS;
4. typedef T InputPixelType; //根据传入T的类型来定义像素类型
5. typedef T OutputPixelType;

利用上述像素类型来定义输入和输出的图像类型，维度为3维

1. typedef itk::Image<InputPixelType, 3> InputImageType;
2. typedef itk::Image<OutputPixelType, 3> OutputImageType;

接下来定义了itk::ImageFileReader 类来从一个文件中读取图像

1. typedef itk::ImageFileReader<InputImageType> ReaderType;

定义itk::imageFileWriter类用于保存输出图像

1. typedef itk::ImageFileWriter<OutputImageType> WriterType;

请注意，由于执行的两个不同图像的像素相加操作，需要将其中一个图像的坐标映射到另外一个图像对应的坐标上，所以必须对其中一副图像进行重采样。而重采样是在空间坐标中执行，而不是像素/网格坐标。因此要获得在一个非像素/网格点上的值，必须进行插值计算。这里我们采用了精度较好的B样条插值运算

定义B样条插值类对象用于对输入图像2进行插值操作

1. typedef itk::BSplineInterpolateImageFunction<InputImageType> Interpolator;

定义重采样函数滤镜对象用于对输入图像2个进行重采样操作。

1. typedef itk::ResampleImageFilter<InputImageType, OutputImageType> ResampleType;

定义像素求和滤镜，用于将两个图像对应像素点的值进行相加操作

1. typedef itk::ConstrainedValueAdditionImageFilter<InputImageType, OutputImageType, OutputImageType> FilterType;

调用对象的New()方式来创建对象实例，由于是两幅图像相加，因此需要实例化两个reader对象，创建方法完全一致

1. typename ReaderType::Pointer reader1 = ReaderType::New();

这里itk::PluginFilterWatcher作用是监视对象或者是滤镜在管道处理中的事件，例如滤镜开始执行、执行进度、执行时间、执行结束，并将事件通过std::cout输出为XML格式

1. itk::PluginFilterWatcher watchReader1(reader1, "Read Volume 1",
2. CLPProcessInformation);

实例化reader对象以后，调用其SetFileName方法设置输入文件路径

1. reader1->SetFileName( inputVolume1.c\_str() );
2. reader2->SetFileName( inputVolume2.c\_str() );

调用ReleaseDataFlagOn方法用于标记将reader2中的数据清空，以便重置输入图像数据

1. reader2->ReleaseDataFlagOn();

我们已经讲到利用Update方法来触发管道执行，这里是将图像读入reader内存位置

1. reader1->Update();
2. reader2->Update();

接下来创建并设置插值滤镜，同样是利用其new方法创建其实例。而reader2->GetOutput()方法我们之前也讲到了是用来将多个处理对象连接起来，这里是把reader对象和插值对象连接起来

1. typename Interpolator::Pointer interp = Interpolator::New();
2. interp->SetInputImage(reader2->GetOutput() );

设置插值参数，其中order是宏PARG\_ARG从3D SLICER GUI界面获得的用户输入参数之一

1. interp->SetSplineOrder(order);

以下代码用于创建并设置重采样滤镜

1. typename ResampleType::Pointer resample = ResampleType::New();
2. resample->SetInput(reader2->GetOutput() );

利用SetOutputParametersFromImage从reader1指向的图像中获得并设置输出图像参数，例如图像的尺寸等，以便让reader2中的图像2能获取到和图像1一样的参数来做采样校准

1. resample->SetOutputParametersFromImage(reader1->GetOutput() );
2. resample->SetInterpolator( interp );

由于两个图像输入源的尺寸不同，因此可能发生在采样某个像素的时候，其坐标落在图像之外的情况，此时就有必要决定给这些像素以默认值。设置这个默认值可以通过调用SetDefaultPixelValue( )方式来实现。这里我们把默认值设为0

1. resample->SetDefaultPixelValue( 0 );

注意要使用ReleaseDataFlagOn函数来保证清空了上一次处理过的数据

1. resample->ReleaseDataFlagOn();

同样监视该滤镜的处理事件，上面已经介绍过了不多赘述

1. itk::PluginFilterWatcher watchResample(resample, "Resampling",
2. CLPProcessInformation);

创建像素相加滤镜并设置滤镜的2个输入，并监视滤镜变化输出信息

1. typename FilterType::Pointer filter = FilterType::New();
2. filter->SetInput1( reader1->GetOutput() );
3. filter->SetInput2( resample->GetOutput() );
4. itk::PluginFilterWatcher watchFilter(filter, "Adding",
5. CLPProcessInformation);

创建输出图像类型对象，并监视其输出

1. typename WriterType::Pointer writer = WriterType::New();
2. itk::PluginFilterWatcher watchWriter(writer,
3. "Write Volume",
4. CLPProcessInformation);

设置输出图像路径，得到滤镜输出结果并显示。其中outputVolume就是在GUI界面设置的同名的参数

1. writer->SetFileName( outputVolume.c\_str() );
2. writer->SetInput( filter->GetOutput() );
3. writer->SetUseCompression(1); //对输出图像启用压缩
4. writer->Update();
5. return EXIT\_SUCCESS;
6. }

下面利用SLICER对该模块进行简单的演示，图9显示的是input volume1，实际上它就是我们之前的高斯滤波中的输入图像图5，而图10的input volume2则是我们之前做过高斯滤波的输出图像图6，它们的尺寸均为256\*256\*51

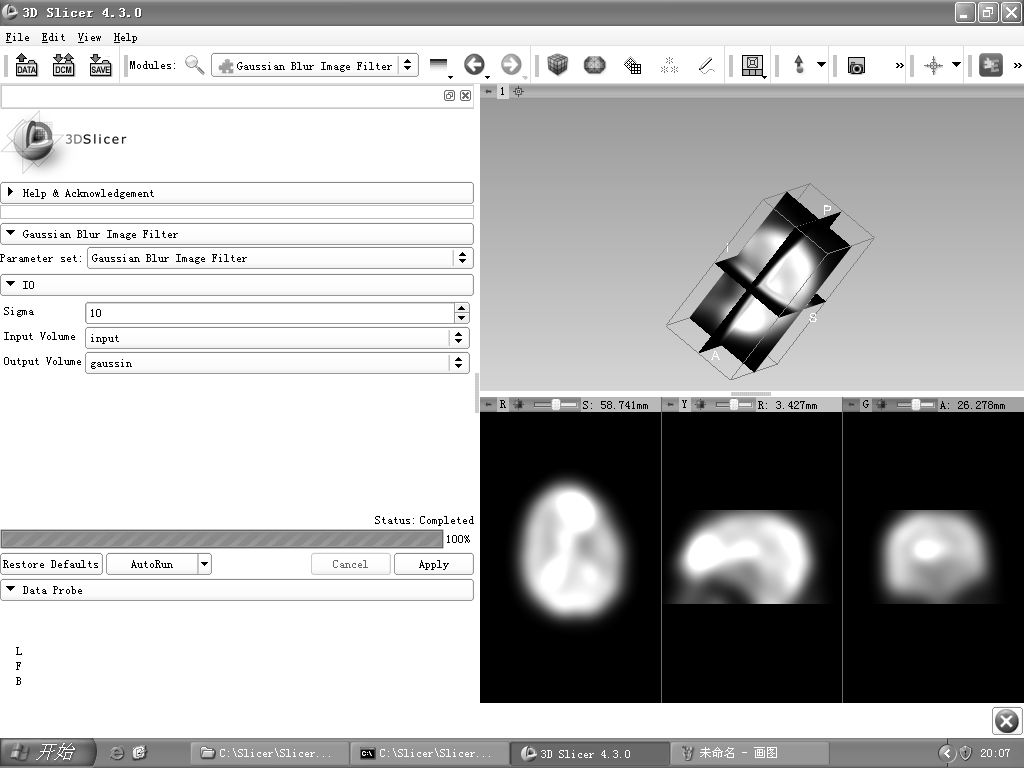
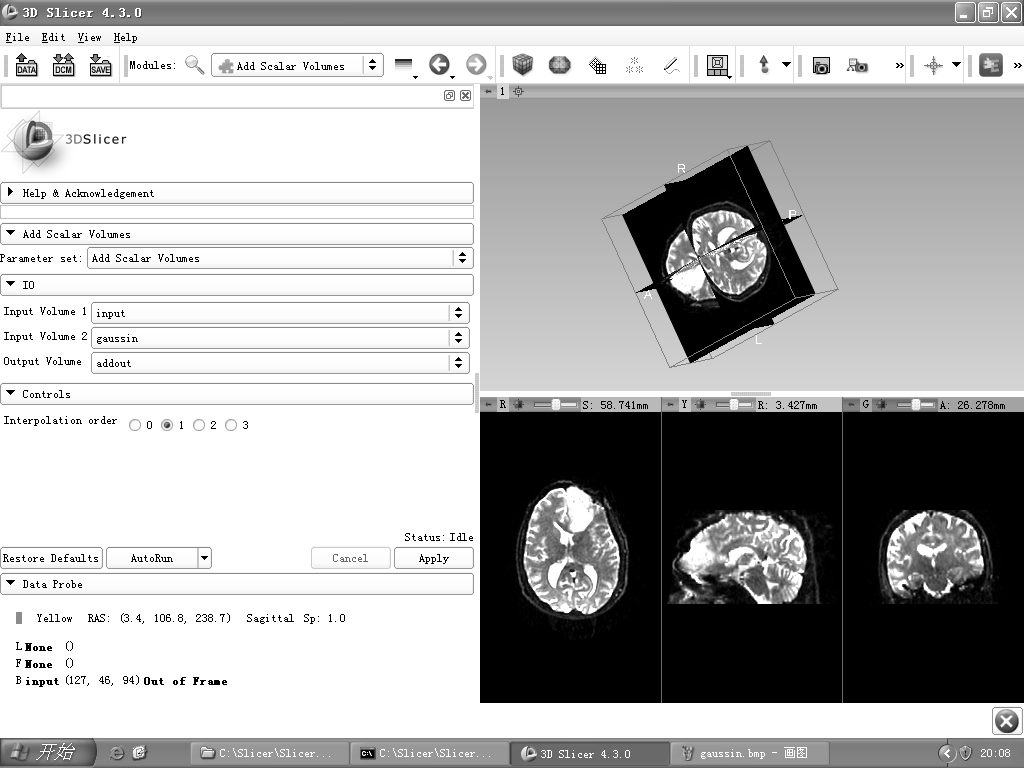


图9 输入图像input volume1 图10 输入图像input volume2

我们设点好输出图像output volume以后，将插值参数interoplation order设定为默认的1，然后点apply，得到了图11的结果，从结果可以看到输出图像为两个输入图像的叠加。

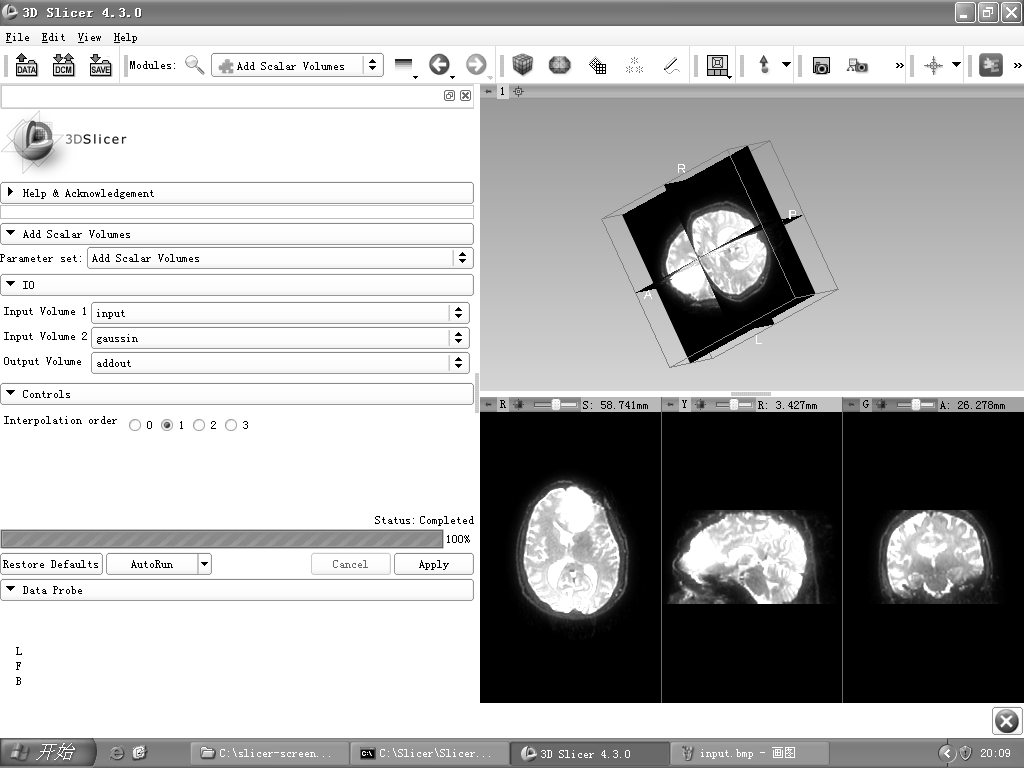


图11 输出图像

事实上该AddScalarVolume模块因为采用了重采样和插值滤镜，可以支持不同尺寸的输入图像的相加，读者自己设定输入图像尝试

**10.3 3DSLICER扩展命令行模块**

当我们完成了CLI的编写以后，就可以打开3D SLICER来调用这个CLI了。步骤是点击菜单栏的Edit，选中Application Setting，在弹出的对话框中选中Modules如下图12所示。在Paths中点击Add，查找选中编译好的CLI模块的文件夹。在本例中，我们编译了一个名叫MyModule的CLI模块，并将编译好的目录添加进来。

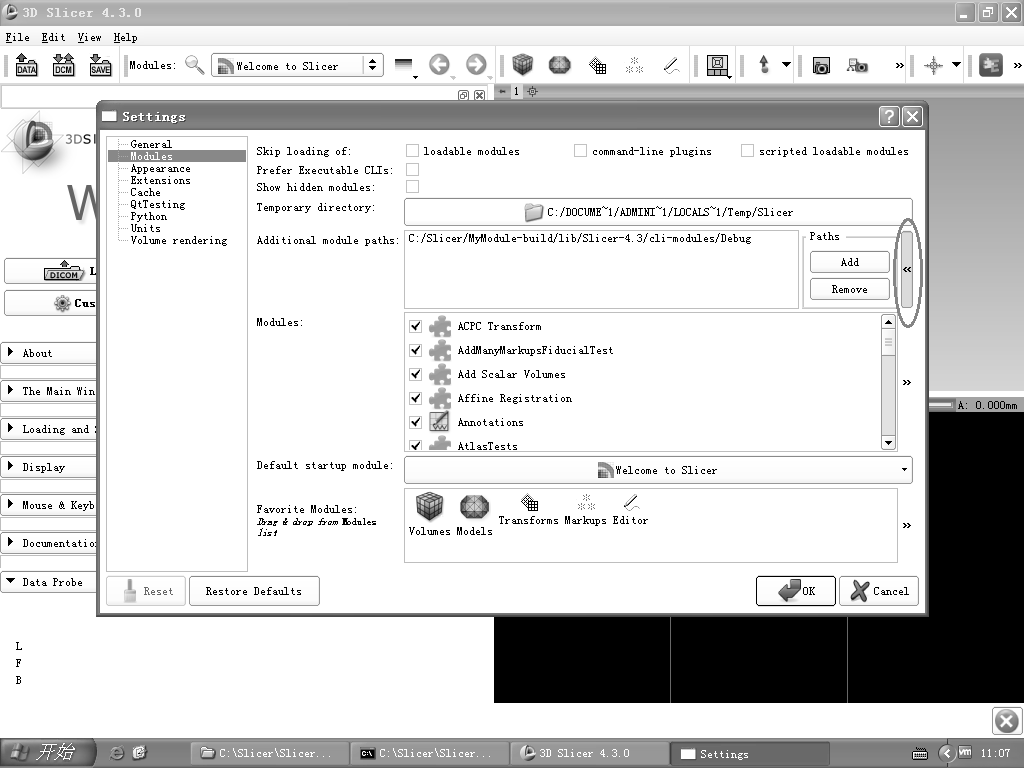


图12 添加编译好的模块文件夹目录

在完成了这一系列的添加步骤之后，我们自己编译的模块，就可以显示在模块的下拉列表中并且进行使用了。

