Watershed Segmentation Documentation

WatershedSegmentation2.cxx

This is application which can handle 3D input images (.vtk format in our case).

We initialize read and write types:
```
typedef itk::Image< InternalPixelType,  Dimension >  InternalImageType;
typedef itk::Image< RGBPixelType,       Dimension >  RGBImageType;
```

We instantiate reader and writer types
```
typedef  itk::ImageFileReader< InternalImageType   >  ReaderType;
typedef  itk::ImageFileWriter< RGBImageType  >        WriterType;
```

```
ReaderType::Pointer reader = ReaderType::New();
WriterType::Pointer writer = WriterType::New();
```

```
Setting the input and output file names:
reader->SetFileName( argv[1] );
writer->SetFileName( argv[2] );
```

Instantiate the GradientMagnitude image filter:

```
typedef   itk::GradientMagnitudeRecursiveGaussianImageFilter<
InternalImageType,
InternalImageType
> GradientMagnitudeFilterType;
```

```
GradientMagnitudeFilterType::Pointer gradienMagnitudeFilter =
GradientMagnitudeFilterType::New();
```

Set input image as input for gradient magnitude filter:
```
gradienMagnitudeFilter->SetInput( reader->GetOutput() );
gradienMagnitudeFilter->SetSigma( 1.0 );
```

GradientMagnitude filter computes the Magnitude of the Gradient of an image by convolution with the first derivative of a Gaussian.

```
typedef  itk::WatershedImageFilter<
    InternalImageType
    > WatershedFilterType;

WatershedFilterType::Pointer watershedFilter =
WatershedFilterType::New();
```

Result from gradient filtering is propagated to watershed filter:

```
watershedFilter->SetInput( gradienMagnitudeFilter->GetOutput() );
```

Set threshold and level based on argument values:

```
watershedFilter->SetThreshold( atof( argv[3] ) );
watershedFilter->SetLevel(     atof( argv[4] ) );
```

Instantiate the filter that will encode the label image into a color image (random color attribution):

```
typedef itk::Functor::ScalarToRGBPixelFunctor<
unsigned long
> ColorMapFunctorType;

typedef WatershedFilterType::OutputImageType  LabeledImageType;

typedef itk::UnaryFunctorImageFilter<
LabeledImageType,
RGBImageType,
ColorMapFunctorType
> ColorMapFilterType;

ColorMapFilterType::Pointer colorMapFilter =
ColorMapFilterType::New();
```

Filtering watershed result to colored output image:

```
colorMapFilter->SetInput(  watershedFilter->GetOutput() );
```

Writing to output file:

```
writer->SetInput( colorMapFilter->GetOutput() );
writer->Update();
```

General workflow

GetOutput() functions return the output of the filters that they are being called from. The output of the particular filter is not valid until an appropriate Update() method has been called, either explicitly or implicitly. Both the filter itself and the data object have Update() methods, and both methods update the data.

So, instead of calling Update method for each filter, we make a pipeline and then call Update() method on the last one, which will implicitly call the others (Update methods) and produce desired effect. In our case the constitution of pipeline is as follows:

```
gradienMagnitudeFilter->SetInput( reader->GetOutput() );
watershedFilter->SetInput( gradienMagnitudeFilter->GetOutput() );
colorMapFilter->SetInput(  watershedFilter->GetOutput() );
writer->SetInput( colorMapFilter->GetOutput() );
writer->Update();
```

WatershedImageFilter.cxx

In this case we have pretty much the same filters and data used. The main difference is that here the data is used as a 2D data. We used this application to test threshold and level values, so that we can narrow down the test cases for 3D .vtk image processing, because its so time consuming.

```
itk::TimeProbesCollectorBase timer;
    for (threshold=0.01; threshold<0.02; threshold+=0.001) {
      for (level=0.15; level<0.25; level+=0.01) {
            std::stringstream ss;
            ss << threshold << "_" << level;
            std::cout << "Segment input for threshold_level = " <<
ss.str() << std::endl;
            timer.Start(ss.str().c_str());
            PerformSegmentation(gradientMagnitudeImageFilter-
>GetOutput(), threshold, level);
            timer.Stop(ss.str().c_str());
      }
    }
timer.Report( std::cout );
```

This code snippet test cross product of values for threshold and level values and then prints out the times report. All of the 2D outputs are stored in separate folder:

```
ss << "times_thresh_0.01_0.02_level_0.15_0.25/output_" << threshold
<< "_" << level << ".png";
typedef itk::ImageFileWriter<RGBImageType> FileWriterType;
FileWriterType::Pointer writer = FileWriterType::New();
writer->SetFileName(ss.str());
writer->SetInput(colormapImageFilter->GetOutput());
writer->Update();
```

After that we would review the outputs and decide which values to use in our 3D application.