Image mask adaptor for pixel order independent filters

Release 0.00

Richard Beare

August 1, 2007

Department of Medicine, Monash University, Australia.

Abstract

It is sometimes useful to apply a filter to a region defined by a mask. This article introduces a simple "mask adaptor" filter that allows this type of operation to be tested easily. It is applicable to filters that do not need any pixel location information, such as histogram operations, which can't necessarily be achieved by simple masking approaches.

Contents

1 Introduction

The application that inspired this filter was Otsu thresholding applied to a specific region of the image, with the rest of the image being ignored.Simply masking the image and applying an Otsu threshold does not give the same result because the mask operation introduces a large number of zeros that influences histogram based computations. This filter, which is loosely termed a "mask adaptor" simply copies all pixels defined by a mask to a temporary one dimensional image, applies a user defined pipeline and copies the results back to the correct image format. This procedure probably breaks streaming rules and may only be useful for histogram operations.

The most recent versions of this packages can be obtained from http://voxel.jouy.inra.fr/darcsweb/ [1]

[1] The most recent versions can be obtained using darcs [1] with the command *darcs get http://voxel.jouy.inra.fr/darcs/contrib-itk/maskAdaptor*

2 Usage

The new filter is called the *itkMaskAdaptorImageFilter* and can be used as follows[2]. This example demonstrates separating the background into original and padded regions, which is difficult to do using histogram operations applied to the entire image.

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkCommand.h"
#include <itkBinaryThresholdImageFilter.h>

#include "itkOtsuMultipleThresholdsImageFilter.h"
#include "itkMaskAdaptorImageFilter.h"

int main(int, char * argv[])
{
  const int dim = 2;

  typedef unsigned char PType;
  typedef itk::Image< PType, dim > IType;

  typedef itk::ImageFileReader< IType > ReaderType;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName( argv[1] );

  typedef itk::BinaryThresholdImageFilter<IType, IType> ThreshType;
  ThreshType::Pointer thresh = ThreshType::New();
  thresh->SetInput(reader->GetOutput());
  thresh->SetUpperThreshold(50);
  thresh->SetInsideValue(1);
  thresh->SetOutsideValue(0);

  typedef itk::OtsuMultipleThresholdsImageFilter<IType, IType> OtsuType;
  OtsuType::Pointer filter = OtsuType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetNumberOfThresholds(2);

  typedef itk::ImageFileWriter< IType > WriterType;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput( filter->GetOutput() );
  writer->SetFileName( argv[2] );
  writer->Update();

  // now for the masked version
  typedef itk::MaskAdaptorImageFilter<IType, IType, IType> MaskAdaptorType;
  typedef itk::OtsuMultipleThresholdsImageFilter<MaskAdaptorType::InternalInputImageType, MaskAdaptorTy

  MaskAdaptorType::Pointer masker = MaskAdaptorType::New();
  MaskOtsuType::Pointer maskotsu = MaskOtsuType::New();
  maskotsu->SetNumberOfThresholds(1);
  masker->SetFilter(maskotsu);
  masker->SetInput(reader->GetOutput());
```

[2]This code is in check.cxx, included in the distribution

```
  masker->SetMaskImage(thresh->GetOutput());
  masker->SetDefaultValue(0);

  writer->SetInput( masker->GetOutput() );
  writer->SetFileName( argv[3] );
  writer->Update();

  return 0;
}
```

The first part of the code uses a standard thresholding operation to generate a mask of all the background region. We then apply standard Otsu thresholding to the entire image and the region defined by the mask. The latter requires declaring the Otsu threshold using the internal image types defined by the MaskAdaptorType.

Results of the various steps are shown in Figures 2 and 3.

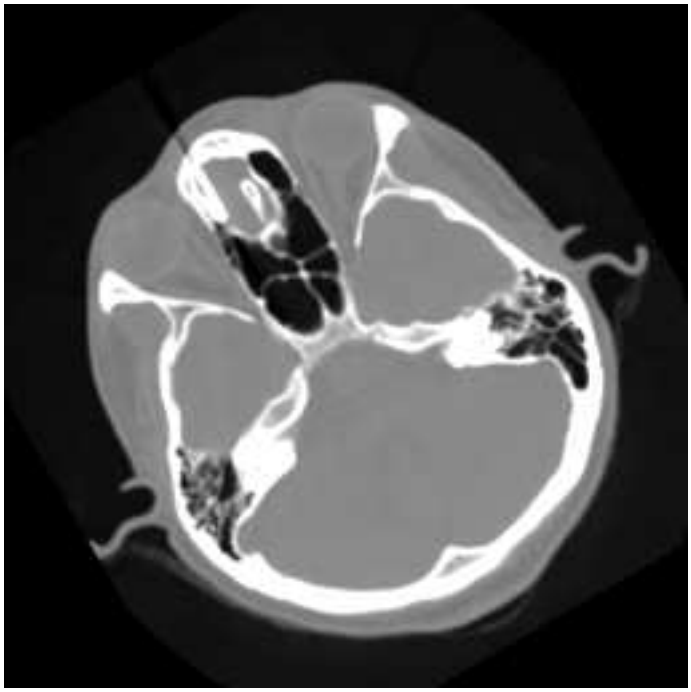

Figure 1: The input image.

3 Methods

- **Set/GetFilter** : Set/Get the user defined filter applied to the masked region.
- **Set/GetMaskImage** : Set/Get the mask image.
- **Set/GetInputFilter** : Set/Get the start of a processing pipeline.
- **Set/GetOutputFilter** : Set/Get the end of a processing pipeline.
- **Set/GetDefaultValue** : Set/Get the output value where the mask is zero.
- **Set/GetPassOutsideMask** : Set/Get (boolean) whether the output is copied from the input where the mask is zero. Overrides DefaultValue.

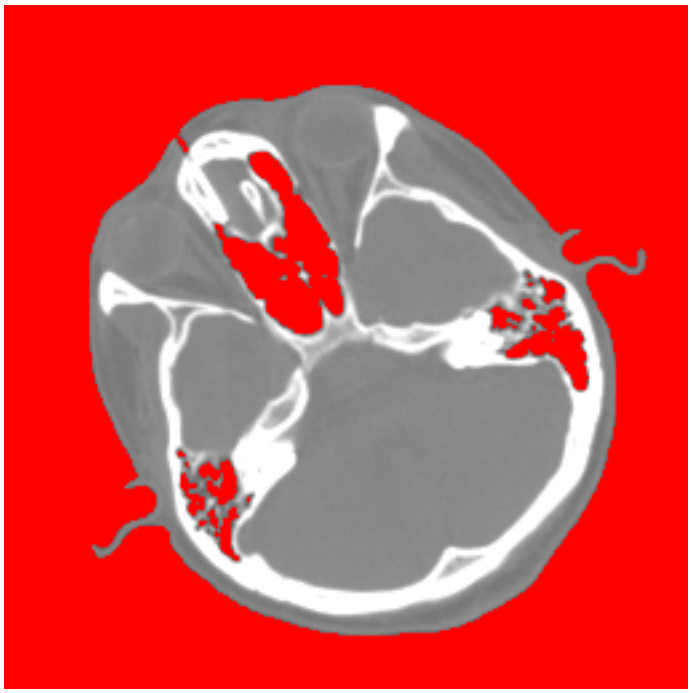

Figure 2: The mask defining the region to which Otsu thresholding is applied.

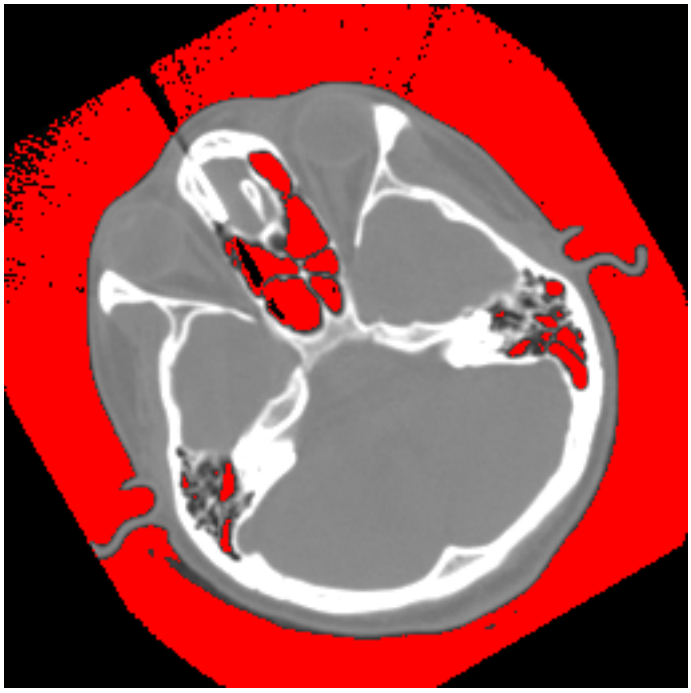

Figure 3: The threshold calculated within the mask region. The padded region is separated from the original background.

4 Comments

The best strategy for implementing this filter isn't clear to me. The necessity for allocating different buffers and copying the results of the user defined pipeline to the output buffer means that the usual pipeline structure isn't followed. I'm not sure how best to deal with this, and any advice on improvements are welcome.

References

[1] http://www.darcs.net. 1

[2] L. Ibanez and W. Schroeder. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, 2003.