Orthogonal Bisection of an Image

Release 0.01

Robert Tamburo

October 19, 2006

University of Pittsburgh
Department of Psychiatry

Abstract

This document describes a filter called `itkOrthogonalBisectImageFilter`, which bisects an input image and copies each divided region into two separate output images. Pixel values are copied while maintaining their original pixel location. The output images are of the same size, dimension, and pixel type as the input image. Pixels that have not been copied from the input image are set to a user-specified value. This filter is restricted to bisection along a cardinal axis, i.e., in a direction orthogonal to a bisecting slice (line for 2D images or plane for 3D images). Submitted with this document is the source code for the filter and source code for demonstrating filter usage via an image input as an argument. Also included is source code for testing the functionality of the filter.

Contents

1 Filter Description and Implementation

A basic filter called the `itkOrthogonalBisectImageFilter` has been developed for the Insight Toolkit. The operation of this filter on a 3D image is illustrated below in Fig. 1. This filter bisects the input image and copies each divided region (R_1 and R_2) into two separate output images. Pixel values are copied while maintaining their original pixel location. The output images are of the same size, dimension, and pixel type as the input image. Pixels in the output images that have not been copied from the input image can be initialized to a user-specified value (the default value is 0). This filter is restricted to bisection along a cardinal axis of the input image, i.e., in a direction orthogonal to a bisecting slice. The bisecting slice is determined

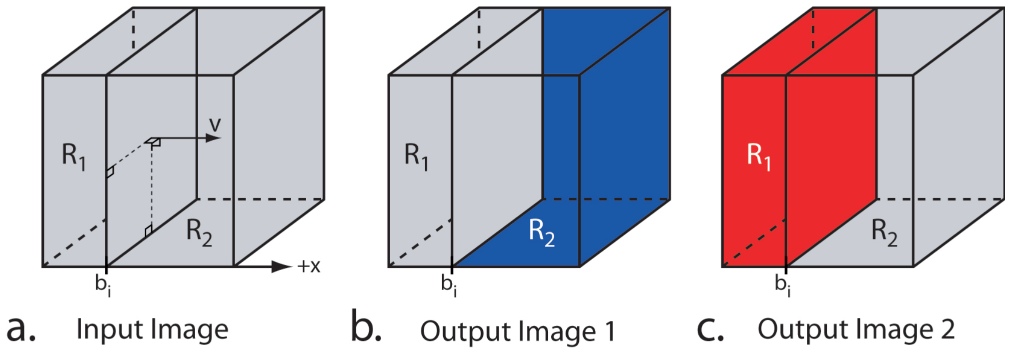

Figure 1: Operation of the filter on a 3D input image. In this example, a slice in the *yz*-plane bisects the image in the direction of vector **v** (*x*-axis). The region denoted by R_1 in the input image is copied to output image 1 as denoted as the gray region, which includes those pixels in the slice at b_i. The blue region in output image 1 is the pixel values specified by the user. Similarly, the gray valued pixels in R_2 are copied to output image 2 and the red region denotes the user-defined pixel values.

by `itkImageSliceIteratorWithIndex` and defined as a line for 2D images and a plane for 3D images. Bisection occurs in a direction orthogonal to the bisecting slice. The location of the bisecting slice is determined by specifying the bisect index (b_i in Fig. 1) of the slice iterator. Note that the pixels at the bisection plane are included in output image 1. If the index is not specified, it defaults to the middle of the image along the direction of bisection. Image regions can also be extracted with the `itkExtractImageFilter` and `itkCropImageFilter`, but the output of these filters take on the size of the new region.

2 Filter Usage

Usage of this filter is now described. The reader is referred `OrthogonalBisectImageFilterTest.cxx` or `OrthogonalBisectImageFilterExample.cxx` for a working demonstration. The filter is instantiated much like other filters found in the Insight Toolkit, i.e.,

```
Typedef OrthogonalBisectImageFilter<ImageType> FilterType;

FilterType::Pointer filter = FilterType::New();
```

The input image is set by passing an `itk::Image` smart pointer to the filter via `SetInput()`. The bisect direction is set by passing an `std::string` of the direction orthogonal to the bisect line or plane with `SetOrthogonalDirection()`, e.g., `SetOrthogonalDirection(std::string("x"))`. If a string other than *x*, *y*, or *z* is used an exception is thrown. The index of the bisect line or plane is set with the function `SetBisectIndex()`. The pixel values for the first and second output images can be set with the functions `SetOutput1FillIntensity()` and `SetOutput2FillIntensity()`, respectively. The default pixel values are 0. Once these filter settings have been determined, the filter can be executed with the standard `Update()` call. The first output image can be retrieved with `GetOutput(0)`, and the second image can be retrieved with `GetOutput(1)`.

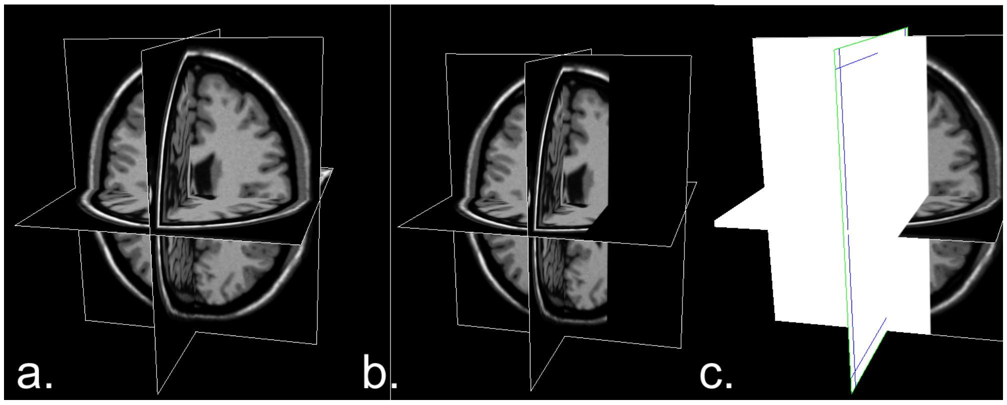

Figure 2: A). 3D image obtained from the BrainWeb database. This image is bisected by the *yz*-plane in the direction of the *x*-axis. B). The first output image and the C). second output image.

3 Example and Validation Test

A working example of this filter is included in `OrthogonalBisectImageFilterExample.cxx`. This is a user-interactive mode, which requires the user to specify parameters as arguments. The arguments are as follows: 1). The path and name of the input image, 2). The desired name of the first output image, 3). The desired name of the second output image, 4). The pixel value for filling in the first output image, 5). The pixel value for filling in the second output image, 6). The direction of bisection, and 7) optionally, the bisect index. If a bisect index is not specified, it will be set to half the size of the image in the direction of bisection. The file extension must be included when specifying the input and output images. Usage of this example with the included brain.img is shown below:

```
OrthogonalBisectImageFilterExample brain.img output1.img output2.img 0 255 x 123
```

This bisects `brain.img` at slice (*yz*-plane) index 123 along the *x*-axis and fill `output1.img` with 0 valued pixels and fill output2.img with 255 valued pixels. The output of the filter using these parameters is shown in Fig. 2 using an image obtained from the BrainWeb database (http://www.bic.mni.mcgill.ca/brainweb/). Visualization is done with an independent VTK/FLTK application.

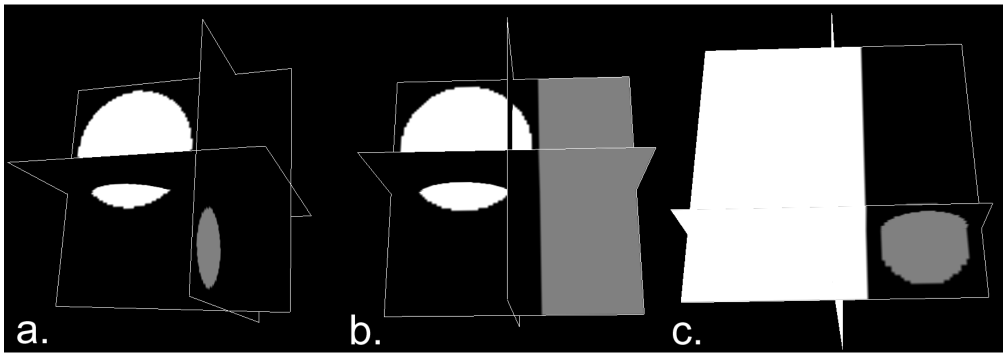

Figure 3: A). The 3D test image, which contains two spheres. This image is bisected by the *yz*-plane in the direction of the *x*-axis. B). The first output image and the C). second output image.

Also included is OrthogonalBisectImageFilterTest.cxx, which is meant to validate filter operation. A parametric image of size $100 \times 100 \times 100$ is created containing 2 spheres. One sphere is centered at $(30,30,45)$ with a radius of 28 and an intensity of 255. The second sphere is centered at $(80,80,80)$ with a radius of 15 and an intensity of 128. The image is bisected with a slice at index 60 along the *x*-axis. The filter is validated by iterating over the output images and verifying that the pixel values are correct as compared to the input image. The images are shown in Fig. 3. The generated input image and output images are saved to disk as Analyze images; the input image is named `synthTestImage.img` and the output images are named `synthOutput1.img` and `synthOutput2.img`.

4 Software Requirements

This filter was written with the following software installed:

1. Insight Toolkit 2.8.1.
2. CMake 2.4 patch 3.

This filter was tested with the included main.cxx on a MacBook pro with gcc 4.0.1 with 0 errors and 0 warnings.

Orthogonal Bisection of an Image

Release 0.01

Robert Tamburo

October 26, 2006

University of Pittsburgh
Department of Psychiatry

Abstract

This document describes a filter called `itkOrthogonalBisectImageFilter`, which bisects an input image and copies each divided region into two separate output images. Pixel values are copied while maintaining their original pixel location. The output images are of the same size, dimension, and pixel type as the input image. Pixels that have not been copied from the input image are set to a user-specified value. This filter is restricted to bisection along a cardinal axis, i.e., in a direction orthogonal to a bisecting slice (line for 2D images or plane for 3D images). Submitted with this document is the source code for the filter and source code for demonstrating filter usage via an image input as an argument. Also included is source code for testing the functionality of the filter.

Contents

1 Filter Description and Implementation

A basic filter called the `itkOrthogonalBisectImageFilter` has been developed for the Insight Toolkit. The operation of this filter on a 3D image is illustrated below in Fig. 1. This filter bisects the input image and copies each divided region (R_1 and R_2) into two separate output images. Pixel values are copied while maintaining their original pixel location. The output images are of the same size, dimension, and pixel type as the input image. Pixels in the output images that have not been copied from the input image can be initialized to a user-specified value (the default value is 0). This filter is restricted to bisection along a cardinal axis of the input image, i.e., in a direction orthogonal to a bisecting slice. The bisecting slice is determined

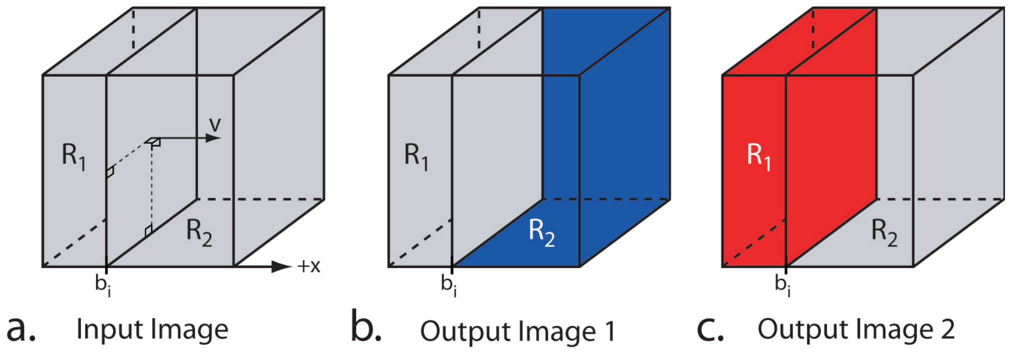

Figure 1: Operation of the filter on a 3D input image. In this example, a slice in the *yz*-plane bisects the image in the direction of vector **v** (*x*-axis). The region denoted by R_1 in the input image is copied to output image 1 as denoted as the gray region, which includes those pixels in the slice at b_i. The blue region in output image 1 is the pixel values specified by the user. Similarly, the gray valued pixels in R_2 are copied to output image 2 and the red region denotes the user-defined pixel values.

by `itkImageSliceIteratorWithIndex` and defined as a line for 2D images and a plane for 3D images. Bisection occurs in a direction orthogonal to the bisecting slice. The location of the bisecting slice is determined by specifying the bisect index (b_i in Fig. 1) of the slice iterator. Note that the pixels at the bisection plane are included in output image 1. If the index is not specified, it defaults to the middle of the image along the direction of bisection. Image regions can also be extracted with the `itkExtractImageFilter` and `itkCropImageFilter`, but the output of these filters take on the size of the new region.

2 Filter Usage

Usage of this filter is now described. The reader is referred `OrthogonalBisectImageFilterTest.cxx` or `OrthogonalBisectImageFilterExample.cxx` for a working demonstration. The filter is instantiated much like other filters found in the Insight Toolkit, i.e.,

```
Typedef OrthogonalBisectImageFilter<ImageType> FilterType;

FilterType::Pointer filter = FilterType::New();
```

The input image is set by passing an `itk::Image` smart pointer to the filter via `SetInput()`. The bisect direction is set by passing an `std::string` of the direction orthogonal to the bisect line or plane with `SetOrthogonalDirection()`, e.g., `SetOrthogonalDirection(std::string("x"))`. If a string other than *x*, *y*, or *z* is used an exception is thrown. The index of the bisect line or plane is set with the function `SetBisectIndex()`. The pixel values for the first and second output images can be set with the functions `SetOutput1FillIntensity()` and `SetOutput2FillIntensity()`, respectively. The default pixel values are 0. Once these filter settings have been determined, the filter can be executed with the standard `Update()` call. The first output image can be retrieved with `GetOutput(0)`, and the second image can be retrieved with `GetOutput(1)`.

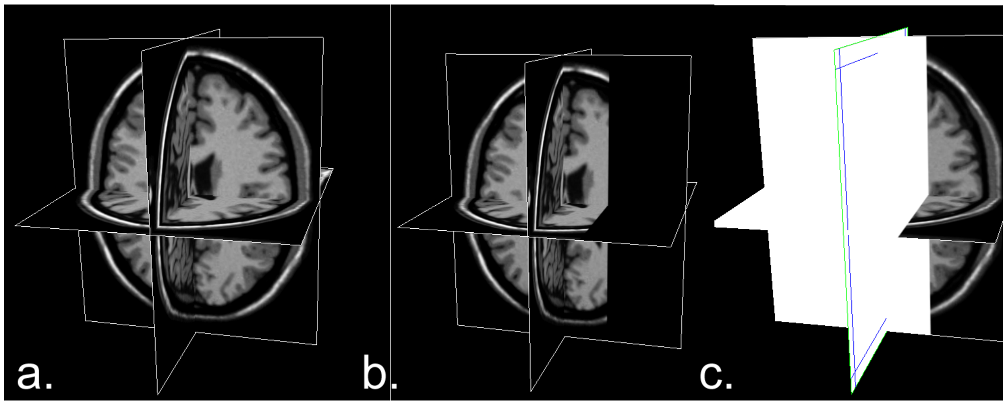

Figure 2: A). 3D image obtained from the BrainWeb database. This image is bisected by the *yz*-plane in the direction of the *x*-axis. B). The first output image and the C). second output image.

3 Example and Validation Test

A working example of this filter is included in `OrthogonalBisectImageFilterExample.cxx`. This is a user-interactive mode, which requires the user to specify parameters as arguments. The arguments are as follows: 1). The path and name of the input image, 2). The desired name of the first output image, 3). The desired name of the second output image, 4). The pixel value for filling in the first output image, 5). The pixel value for filling in the second output image, 6). The direction of bisection, and 7) optionally, the bisect index. If a bisect index is not specified, it will be set to half the size of the image in the direction of bisection. The file extension must be included when specifying the input and output images. Usage of this example with the included brain.nii is shown below:

```
OrthogonalBisectImageFilterExample brain.nii output1.nii output2.nii 0 255 x 123
```

This bisects `brain.nii` at slice (*yz*-plane) index 123 along the *x*-axis and fill `output1.nii` with 0 valued pixels and fill output2.nii with 255 valued pixels. The output of the filter using these parameters is shown in Fig. 2 using an image obtained from the BrainWeb database (http://www.bic.mni.mcgill.ca/brainweb/). Visualization is done with an independent VTK/FLTK application.

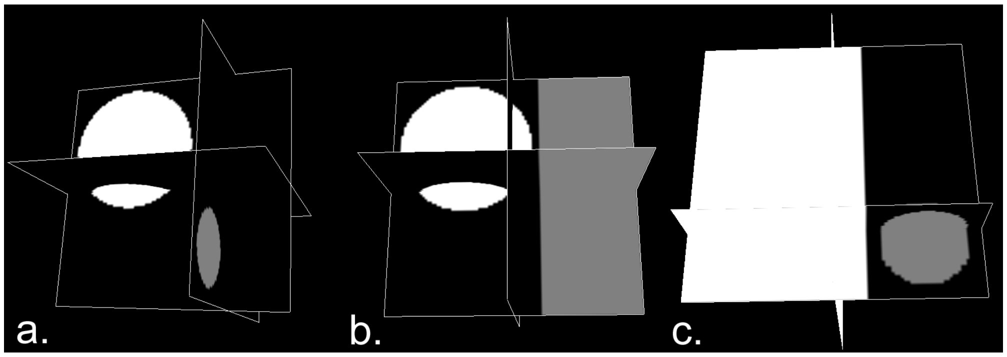

Figure 3: A). The 3D test image, which contains two spheres. This image is bisected by the *yz*-plane in the direction of the *x*-axis. B). The first output image and the C). second output image.

Also included is OrthogonalBisectImageFilterTest.cxx, which is meant to validate filter operation. A parametric image of size $100 \times 100 \times 100$ is created containing 2 spheres. One sphere is centered at $(30,30,45)$ with a radius of 28 and an intensity of 255. The second sphere is centered at $(80,80,80)$ with a radius of 15 and an intensity of 128. The image is bisected with a slice at index 60 along the x-axis. The filter is validated by iterating over the output images and verifying that the pixel values are correct as compared to the input image. The images are shown in Fig. 3. The generated input image and output images are saved to disk as Analyze images; the input image is named `synthTestImage.nii` and the output images are named `synthOutput1.nii` and `synthOutput2.nii`.

4 Software Requirements

This filter was written with the following software installed:

1. Insight Toolkit 2.8.1.
2. CMake 2.4 patch 3.

This filter was tested with the included main.cxx on a MacBook pro with gcc 4.0.1 with 0 errors and 0 warnings.