The Generalised Image Fusion Toolkit (GIFT)

Release 1.1

Daniel Mueller[1]

July 20, 2006

[1]Queensland University of Technology, Brisbane, Australia

Abstract

Image fusion provides a mechanism to combine multiple images into a single representation to aid human visual perception and image processing tasks. Such algorithms endeavour to create a *fused* image containing the salient information from each source image, without introducing artefacts or inconsistencies. Image fusion is applicable for numerous fields including: defence systems, remote sensing and geoscience, robotics and industrial engineering, and medical imaging. In the medical imaging domain, image fusion may aid diagnosis and surgical planning tasks requiring the segmentation, feature extraction, and/or visualisation of *multi-modal* datasets.

This paper discusses the implementation of an image fusion toolkit built upon the Insight Toolkit (ITK). Based on an existing architecture [15, 17], the proposed framework (GIFT) offers a 'plug-and-play' environment for the construction of n-D multi-scale image fusion methods. We give a brief overview of the toolkit design and demonstrate how to construct image fusion algorithms from low-level components (such as multi-scale methods and feature generators). A number of worked examples for medical applications are presented in Appendix A, including quadrature mirror filter discrete wavelet transform (QMF DWT) image fusion.

Keywords: *image fusion, multi-modal, wavelet, ITK*

1 Introduction

Image fusion algorithms seek to combine multiple input images into a single representation. Such algorithms have numerous applications including the segmentation, feature extraction, and/or visualisation of *multi-modal* medical images. This paper presents an open-source toolkit for multi-scale image fusion and aims to be the start of a central repository for fusion algorithm implementations. The Generalised Image Fusion Toolkit (GIFT) currently implements quadrature mirror filter discrete wavelet transform (QMF DWT) multi-scale fusion algorithms. GIFT is built upon the Insight Toolkit (ITK) - an open-source software system able to perform a range of registration and segmentation algorithms in two- or three-dimensions [24]. Continuing the concept of generic programming advocated by ITK, GIFT has a flexible design allowing for the easy addition of new components and the processing of both 2-D and 3-D images.

This paper is organised into a number of different sections. The first section discusses general image fusion concepts to provide a brief introduction for those not familiar with the field. The next section discusses the design and currently implemented components of GIFT. Following this we give a brief of list of instructions for getting started using the toolkit. Finally we discuss the areas of current and future development. Two examples are walked through in considerable detail in Appendix A.

2 Concepts

Multi-sensor fusion is the task of combining data from multiple input sensors, which can facilitate analysis not possible from a single sensor alone [4]. Image fusion is a specialisation of multi-sensor fusion in which the input and output signals are images. Multi-sensor fusion has been organised into a hierarchy of increasing abstract tasks [5]:

Signal-level is the lowest form of multi-sensor fusion. At this level multiple raw input signals are fused into a single output signal. For *image* fusion, the signal-level is known as 'pixel-level'.

Feature-level fusion operates on feature vectors extracted from either the input signals themselves or the output of the signal-level fusion stage.

Decision-level fusion generates probabilistic decision information obtained from processing the results from lower levels.

The general goal of image fusion is to combine a set of input images to generate a single output image which (a) preserves the salient information from each input image, (b) suppresses noise and irrelevant parts of inputs images, and (c) does not generate distortions, artefacts, or inconsistencies in the fused output [6]. A wide variety of algorithms have been proposed to implement these tenets. These algorithms can be grouped into the following categories:

Arithmetic image fusion is a simple and efficient scheme, which unfortunately suffers from loss of contrast due to the destructive nature of superposition. This type of fusion applies a weight (w_n) to each input image (i_n) and then arithmetically combines these to form the output image (f), as shown below:

$$f(\vec{x}) = w_1 i_1(\vec{x}) + w_2 i_2(\vec{x}) + \ldots + w_n i_n(\vec{x}) \quad (1)$$

Most commonly the weights are chosen to give an averaging effect (ie. $w_n = 1/n$). Another approach is to compute the weights via principal component analysis (PCA) - the weights are set as the eigenvector components corresponding to the largest eigenvalue computed after 'unfolding-backfolding' the image matrix [3].

Colour space fusion represents data using different colour channels, exploiting the ability of the human visual system distinguish three colour channels. The simplest approach maps each input to a single channel in the chosen colour space (eg. RGB, HSV, YUV, CIELAB, etc). As briefly discussed in [15], the obvious challenge with this approach is to create meaningful mappings from input to channel. Within the medical imaging domain this form of fusion is most commonly used for structural-functional images (eg. CT-PET fusion).

Multi-scale image fusion is a biologically-inspired method which fuses images at different spatial resolutions. Similar to the human visual system, this fusion approach operates by firstly decomposing the input images into a resolution pyramid of numerous *levels*. Each level contains one or more *bands* representing orientation or detail/approximation information. Following this decomposition, the fusion now takes place between the corresponding coefficients or *samples* in each band. The fused pyramid is then reconstructed to form the final fused output image. Figure 1 depicts this process.

Other methods also exist including: neural network, statistical *a priori* models, and optimization based approaches.

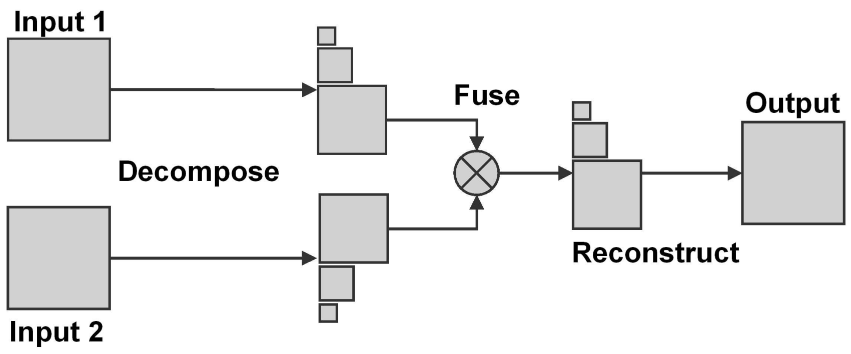


Figure 1: Multi-scale fusion flow diagram for two input images.

Currently GIFT is concerned with pixel-level multi-scale image fusion. However, the modular design allows for the future inclusion of new pixel-level and feature-level operations. For the interested reader, a number of other image fusion toolkits are available as detailed below:

MATIFUS: a MAtlab Toolbox for Image FUSion: by G. Piella [16, Appendix A].

Image Fusion Toolbox for Matlab: by O. Rockinger [22].

The Image Fusion Toolkit for Matlab: by E. Canga [2]

The main benefits of GIFT over these existing toolkits are it's open-source nature, extensibility, 'plug-and-play' architecture, and generic implementation (ie. the same code operates for both 2-D and 3-D images). Moreover, GIFT is easily integrated with ITK's rich set of registration methods - an essential first step for image fusion.

3 Design

3.1 Architecture

The architectural design of GIFT is inspired by existing work undertaken by G. Piella [15, 17]. The main difference is that GIFT has generalised the concepts of *activity measure*, *match measure*, and *region-based measures* into a single generic concept termed ***feature generators***. There are four (4) major components within a GIFT pipeline:

Multi-scale method: This is method for deconstructing and reconstructing into a scale-space image pyramid.

Feature generators: Any number of feature generators can be added to the fusion pipeline. These generators are responsible for processing the multi-level multi-band images to extract ‘features’ used to guide the fusion process. The output of a feature generator is termed a *feature-map*.

Weight generators: Weight generators are responsible for taking multiple feature-maps and creating *weight-maps*. The resultant weight-maps indicate which parts of each level-band image will contribute to the final fused output.

Weight combiner: The weight combiner is responsible for collapsing the input image pyramids into a single output pyramid.

The GIFT pipeline is depicted in Figure 2. Please note that while two multi-scale method boxes are shown, only one multi-scale method is set for the fusion pipeline. Multiple feature generators can be specified, and must be linked to the weight generator responsible for processing the feature. Only one weight combiner is specified for the pipeline. Theoretically multiple n-D input images can be fused using this pipeline, however testing has only focused on 2-D and 3-D images.

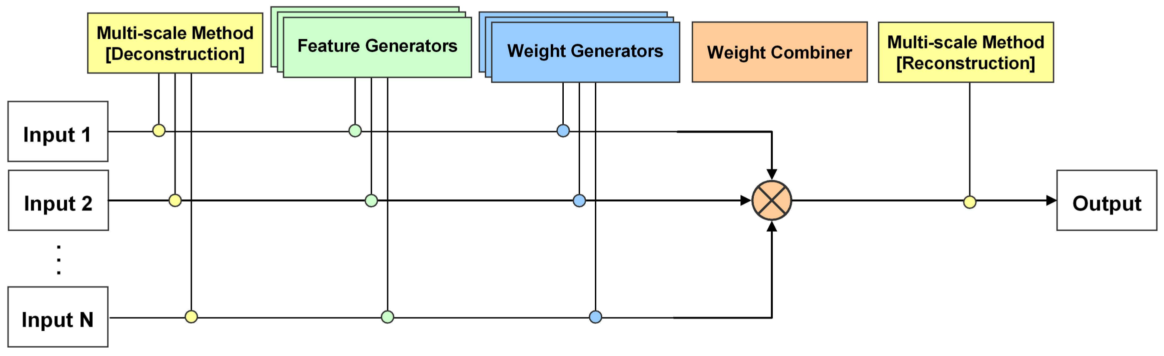


Figure 2: Generalised Image Fusion Toolkit (GIFT) architecture diagram.

3.2 Components

Multi-scale Multi-level Image Filter

At the heart of GIFT is the `gift::MultilevelMultibandImageFilter`. This subclass of `itk::ImageToImageFilter` encapsulates the concept of an input/output image pyramid (a set of images representing multiple levels and bands). All filters within GIFT (excluding `gift::ImageFusionFilter`) inherit from `gift::MultilevelMultibandImageFilter`. This common base adds functionality for easily accessing inputs and/or outputs consisting of multi-level, multi-band image pyramids. The inputs/outputs are indexed as shown in Figure 3, and can be accessed either via their index (as normal) or using an image-level-band 3-tuple (eg. image=2, level=0, band=1).

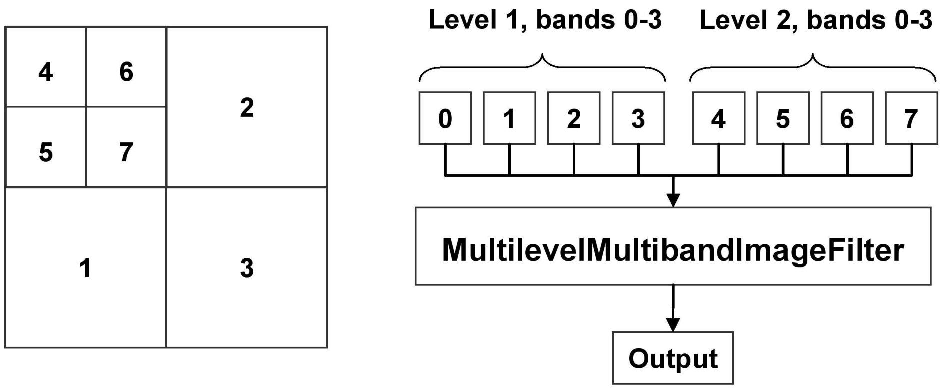


Figure 3: `gift::MultilevelMultibandImageFilter` allows access to inputs/outputs using a level-band tuple, as well using the index.

Multi-scale Methods

Obviously, multi-scale image filters derive from `gift::MultilevelMultibandImageFilter`. For deconstruction, these filters input 1 image and output an array of images accessible via a level-band tuple. For reconstruction the number of inputs and outputs are reversed. The number of images in a band is method-defined, and the number of levels is user-defined.

Image fusion techniques have used a number of different multi-scale methods including: the laplacian pyramid [1], QMF DWT [7, 11], shift-invariant DWT (SIWT) [21, 22], dual-tree continuous wavelet transform (DT-CWT) [12, 6], morphological pyramids [9, 10], and others.

Currently GIFT has only implemented one multi-scale method - quadrature mirror filter discrete wavelet transformation (QMF DWT). This method is encapsulated by the `gift::QmfWaveletImageFilter`. Further details regarding this filter can be found in Appendix A. More multi-scale methods are in development and once completed will easily drop into the GIFT architecture.

Feature Generators

Feature generators operate on the outputs of the multi-scale method and extract relevant 'features' which can guide the fusion process. The feature generators within GIFT can operate at the pixel-level or feature-level.

Currently only pixel-level generators have been implemented, however a number of feature-level generators are under development.

The most fundamental feature is the value of the image coefficient or sample itself. The `gift::SampleActivity` feature generator provides subsequent weight generators access to this feature. Another common feature is the absolute value of the sample, which is provided by the `gift::AbsSampleActivity`. To ensure consistency, it is common to compute a measure of activity for a small window centred at each pixel (eg. 3×3 or 5×5 windows are common for 2-D images). GIFT provides facilities for this form of activity computing through the `gift::MaximumWindowSampleActivity` and `gift::MedianWindowSampleActivity`, which find either the maximum or median within the window respectively.

Weight Generators

Weight generators combine multiple feature-maps into a single weight-map for each input level-band image. A weight-map indicates the proportion of contribution each level-band image makes to the final fused output. Weight generators typically apply some sort of logic, threshold, and/or arithmetic operation to the feature-maps.

For activity-based features (ie. sample, absolute value sample, and window-based features) the simplest approach is to choose the maximum feature from across the input images. For example, the weight generator looks at the feature value for level=1, band=2 in all input images and produces a weight-map with 1.0 for the level-band image with the maximum value and 0.0 for all other inputs. This functionality is commonly termed ‘select max’ and is implemented by `gift::SelectMaximumFeature`. Alternatively we may wish to favour one input over the others, so in this case we would always select the feature from the given input. This functionality is handled by the `gift::SelectFeature`. Finally, we can simply take the average of all features, which is implemented by the `gift::AverageFeature` filter.

The application of weight generators typically depends on the *type* of band image. For example, detail images are best handled with a ‘select max’ approach (recall that in the case of QMF DWT, detail images contain sparse high-frequency information). On the other hand, approximation images contain more compact information and an averaging approach is common practice. GIFT provides a mechanism to override the weight generator for specific level-band images. The default generator is specified via `gift::ImageFusionFilter::SetDefaultWeightGenerator(.)` and specific generators can be overridden using `gift::ImageFusionFilter::OverrideWeightGenerator(.)`.

Weight Combiners

The weight combiner has the ultimate responsiblity for collapsing the weight-maps into a single image pyramid. At the moment only one weight combiner exists within GIFT - `gift::LinearWeightCombiner`. This class generates the final level-band image by applying a *linear* weighted operation to each input level-band image using weights from each weight-map.

4 Using GIFT

To get starting using GIFT please follow the below directions:

1. Download ITK 2.4.1 or higher from http://www.itk.org.
2. Download GIFT from the Insight Journal.
3. Download CMake 2.2.3 or higher from http://www.cmake.org.
4. Configure and compile ITK using CMake (see the ITK Getting Started Manual for help).
5. Configure GIFT using CMake (follow the same procedure for ITK). Point CMake to the top-level folder and ensure all build variables are correct. Make sure the `ITK_DIR` variable is set - if ITK was not automatically found by the build system you will need to manually set this variable.
6. Open the generated project file in your favourite compiler and select `Build Solution` (or similar). Ensure that you compile GIFT with the same configuration (ie. debug or release) as you did ITK.
7. Experiment with the provided examples or create your own image fusion algorithm using `gift::ImageFusionFilter`. A number of test images can be found in the `Testing/Data/Input` directory, and further images are available from http://www.imagefusion.org.

Two worked examples are presented in Appendix A and some other examples are included with the toolkit source-code.

5 Future Work

GIFT currently only implements basic image fusion algorithms. However, it is designed for the easy addition of new components which can be 'plugged' into the `gift:ImageFusionFilter`. In particular, the following components are being considered for development:

Multi-scale methods: Laplacian and morphological image pyramid methods can be implemented in a similar fashion as QMF DWT.

Activity-based feature generators: a number of useful activity measures have been reported in literature, including a directive contrast activity measure [19] and a windowed-squared activity measure [15, pp.31]. These pixel-level methods could be easily added as feature generators.

Region-based feature generators: a number of interesting region-based fusion methods have been explored in current literature [15, 8, 6, 13]. These feature-level methods could be added as feature generators.

Another important area of development is image fusion *metrics*. A number of objective comparison metrics have been investigated in literature and it would be fitting to see these added to GIFT. Some of these include the root mean square error (RMSE) between a "ground-truth" image and the fused image [25], the RMSE between the input images and the fused image [10], image entropy [25], mutual information [20], spatial frequency [25, 26], edge strength and orientation [14], and the image quality index (IQI) [23, 18].

There are also a number of smaller areas needing attention. Currently `gift:ImageFusionFilter` does not support progress accumulation, which would be useful seeing some fusion algorithms can take significant computation time (especially for 3-D input images). Finally, there is a known bug with `gift::QmfWaveletImageFilter`: if `Update()` is not explicitly called, the reconstructed output image has the wrong dimensions (see `Examples/MultiscaleMethods/giftExampleMultiscaleQmfWavelet.cxx` for further details).

6 Conclusions

We have presented the design and initial implementation of GIFT - the Generalised Image Fusion Toolkit. Construction of simple n-D image fusion schemes using QMF DWT is possible using the toolkit. New components can be easily added to enable the implementation of other image fusion algorithms. A number of examples using the toolkit are discussed in Appendix A.

Appendix A Examples

A.1 QMF Wavelet Analysis

The source code for this example can be found in
`Examples/MultiscaleMethods/giftExampleMultiscaleQmfWavelet.cxx`.

The following example shows the usage of `gift::QmfWaveletImageFilter`. This particular filter performs QMF wavelet decomposition or reconstruction on the given input image(s) using a user-specified biorthogonal (bior) filter kernel. `gift::QmfWaveletImageFilter` is a subclass of `gift::MultilevelMultibandImageFilter` and has a user configurable number of levels. This filter has 2^{dims} bands per level - the first band is a decimated approximation image and the remaining bands are detail images for each direction. The number of inputs and outputs is dependent on the mode of operation: deconstruction or reconstruction. In deconstruction mode, the filter takes 1 input image and outputs $levels \times bands \times 2^{dims}$ output images. In reconstruction mode, the number of inputs and outputs are swapped.

To use the `gift::QmfWaveletImageFilter` the first step is to include the relevant headers:

```cpp
#include "giftBiorthogonalWaveletOperator.h"
#include "giftQmfWaveletImageFilter.h"
```

We now setup the internal image type, making sure we use a real pixel type (either `float` or `double`) and remembering `itk:ImageFileReader` will cast the input image to the specified internal type. The number of dimensions has been tested for 2-D and 3-D images, however this example is compiled for 2-D images.

```cpp
    const unsigned int Dimension = 2;
    typedef float InternalPixelType;
    typedef itk::Image<InternalPixelType, Dimension>
        InternalImageType;
```

The `gift::QmfWaveletImageFilter` takes the filter kernel coefficients as a template argument. A number of biorthogonal filter kernels are defined in `gift::BiorthogonalWaveletOperator`. We define the wavelet kernels and filter as such:

```cpp
    typedef gift::BiorthogonalWaveletOperator<InternalPixelType, Dimension>
        BiorWavelet;
    typedef gift::QmfWaveletImageFilter<InternalImageType, BiorWavelet>
        WaveletFilter;
```

We now create the operator by specifying the desired bior kernel:

```cpp
    //Create the wavelet operator
    BiorWavelet bior(BiorWavelet::Bior_1_3);
```

The available kernels are found in `Source/giftBiorthogonalWaveletOperator.h`:

```
        Bior_1_1 = 0x11,
        Bior_1_3 = 0x13,
        Bior_1_5 = 0x15,
        Bior_2_2 = 0x22,
        Bior_2_4 = 0x24,
        Bior_2_6 = 0x26, //Not yet implemented
        Bior_2_8 = 0x28, //Not yet implemented
        Bior_3_1 = 0x31,
        Bior_3_3 = 0x33,
        Bior_3_5 = 0x35,
        Bior_3_7 = 0x37, //Not yet implemented
        Bior_3_9 = 0x39, //Not yet implemented
        Bior_4_4 = 0x44, //Not yet implemented
        Bior_5_5 = 0x55, //Not yet implemented
        Bior_6_8 = 0x68  //Not yet implemented
```

We now create and setup a filter for deconstruction,

```
    WaveletFilter::Pointer filterWaveletDeconstruct = WaveletFilter::New();
    filterWaveletDeconstruct->SetNumberOfLevels(NumberOfLevels);
    filterWaveletDeconstruct->SetDeconstruction();
    filterWaveletDeconstruct->SetWavelet(bior);
```

and a filter for reconstruction (note that `NumberOfLevels` is a command line argument).

```
    WaveletFilter::Pointer filterWaveletReconstruct = WaveletFilter::New();
    filterWaveletReconstruct->SetNumberOfLevels(NumberOfLevels);
    filterWaveletReconstruct->SetReconstruction();
    filterWaveletReconstruct->SetWavelet(bior);
```

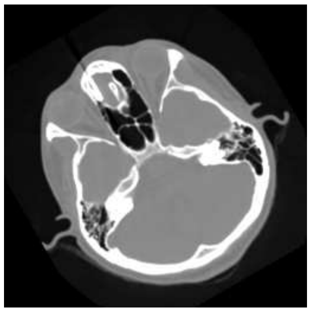
In this particular example we simply want to deconstruct the input image, save these deconstructed images, and reconstruct the output image (to confirm the input and output are the same). To achieve this we loop over `filterWaveletDeconstruct` outputs, construct the output filename using string replace functions, write the intermediate outputs, and attach them to the inputs of `filterWaveletReconstruct`.

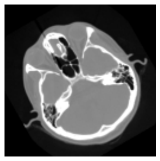
```
    for (unsigned int index=0; index < filterWaveletDeconstruct->GetNumberOfOutputs(); index++)
    {
        //Set input for reconstruction
        filterWaveletReconstruct->SetInput(index, filterWaveletDeconstruct->GetOutput(index));

        //Construct output filename
        char strIndex[5];
        sprintf(strIndex, "%02d", index);
        std::string outputFilename = OutputDeconstructionFilepattern;
        itksys::SystemTools::ReplaceString(outputFilename, "{0}", strIndex);

        //Setup Writer
        WriterType::Pointer writer = WriterType::New();
        writer->SetFileName(outputFilename.c_str());
        writer->SetInput(filterRescale->GetOutput());

        //Perform write
        try
        {
            writer->Update();
        }
    }
```

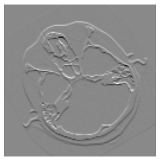
Finally we update `filterWaveletReconstruct` and write the final reconstructed image:

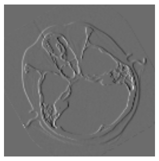
```
    //Force update on reconstruction filter
    filterWaveletReconstruct->Update();

    //Setup final writer
    WriterType::Pointer writer = WriterType::New();
    writer->SetFileName(OutputFilename);
    writer->SetInput(filterCast->GetOutput());

    //Perform write
    try
    {
        writer->Update();
    }
```

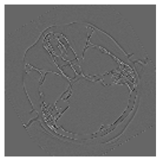
Some results from the example are shown in Figure 4.


(a) Input (A_0)


(b) A_1^0


(c) D_1^1

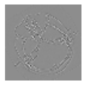

(d) D_1^2


(e) D_1^3

(f) A_2^0

(g) D_2^1

(h) D_2^2


(i) D_2^3

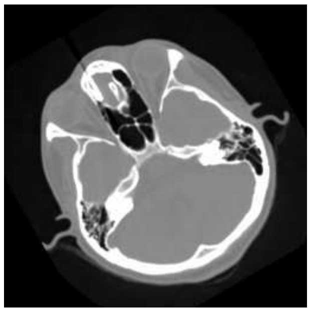

(j) Reconstructed output

Figure 4: Output from `Examples/MultiscaleMethods/giftExampleMultiscaleQmfWavelet.cxx` with input=`cthead1.png` and `NumberOfLevels`=2.

A.2 3-D Image Fusion for Multi-modal Visualisation

The source code for this example can be found in
`Examples/Fusion/giftExampleFusionQmfWavelet1.cxx`.

The following example shows the usage of `gift::ImageFusionFilter`. To setup the filter we must set the multi-scale method, any feature generators, any weight generators, and a weight combiner. In this example we use a 'select max' approach for detail level-band images, and an average approach for approximation level-band images. We use open-source data from the NLM Image Data Collection Project[1]. `giftExampleRegistration` is used to perform rigid registration of the two volumes, and invoked using:

```
giftExampleRegistration
    Testing/Data/Input/P2_MRA_RESCALE.mhd
    Testing/Data/Input/P2_MRI_T1_PRE_RESCALE.mhd
    Testing/Data/Input/P2_T1_PRE_REGISTERED.mhd
```

with final values as below:

```
Result =
 versor X      = -0.00731512
 versor Y      = -0.000214034
 versor Z      = -0.0250347
 Translation X = 23.5739
 Translation Y = 18.397
 Translation Z = 22.1149
 Iterations    = 51
 Metric value  = 86.6556
Matrix =
0.998746 0.0500554 -6.1659e-005
-0.0500491 0.99864 0.014636
0.000794185 -0.0146145 0.999893

Offset =
[18.9281, 22.3807, 23.4384]
```

Following the preparation of the input images, the first step in the fusion process is to create a multi-scale method, in this case `gift::QmfWaveletImageFilter`:

```cpp
      typedef gift::BiorthogonalWaveletOperator<InternalPixelType, Dimension>
          BiorWavelet;
      BiorWavelet bior(BiorWavelet::Bior_3_5);
      typedef gift::QmfWaveletImageFilter<InternalImageType, BiorWavelet>
          WaveletFilterType;
      WaveletFilterType::Pointer filterMultiscale = WaveletFilterType::New();
      filterMultiscale->SetNumberOfLevels(NumberOfLevels);
      filterMultiscale->SetWavelet(bior);
```

[1] http://nova.nlm.nih.gov/Mayo/NLMDATA/Brain/LargeAVM/

This example then creates two feature generators:

```
        typedef gift::SampleActivity<InternalImageType>
            SampleActivityType;
        typedef gift::AbsSampleActivity<InternalImageType>
            AbsSampleActivityType;
        SampleActivityType::Pointer featureSampleActivity =
            SampleActivityType::New();
        AbsSampleActivityType::Pointer featureAbsSampleActivity =
            AbsSampleActivityType::New();
```

and two weight generators:

```
        typedef gift::SelectMaximumFeature<InternalImageType>
            SelectMaximumWeightGeneratorType;
        typedef gift::AverageFeature<InternalImageType>
            AverageWeightGeneratorType;
        AverageWeightGeneratorType::Pointer averageWeightGenerator =
            AverageWeightGeneratorType::New();
        SelectMaximumWeightGeneratorType::Pointer selectMaximumWeightGenerator =
            SelectMaximumWeightGeneratorType::New();
```

We now link the weight generators to the appropriate feature generators. In this case each weight generator uses the output of only one feature generator, however a weight generator could use more than one.

```
        averageWeightGenerator->AddFeatureToUse(
            (AverageWeightGeneratorType::FeatureGeneratorPointer)featureSampleActivity);
        selectMaximumWeightGenerator->AddFeatureToUse(
            (SelectMaximumWeightGeneratorType::FeatureGeneratorPointer)featureAbsSampleActivity);
```

A `gift::LinearWeightCombiner` is created as follows:

```
        typedef gift::LinearWeightCombiner<InternalImageType>
            WeightCombinerType;
        WeightCombinerType::Pointer weightCombiner = WeightCombinerType::New();
```

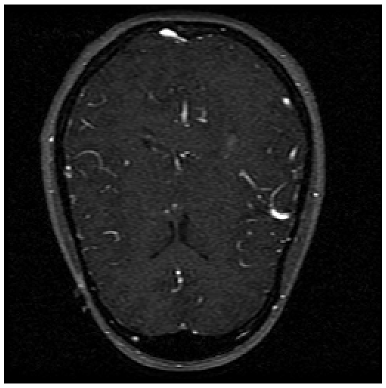
The only task remaining is to plug these components into the `gift::ImageFusionFilter`:

```
        typedef gift::ImageFusionFilter<InternalImageType, WriteImageType>
            ImageFusionFilterType;
        ImageFusionFilterType::Pointer filterFusion = ImageFusionFilterType::New();
        filterFusion->SetMultiscaleMethod(
            (ImageFusionFilterType::MultiscaleMethodPointer)filterMultiscale);
        filterFusion->AddFeatureGenerator(
            (ImageFusionFilterType::FeatureGeneratorPointer)featureSampleActivity);
        filterFusion->AddFeatureGenerator(
            (ImageFusionFilterType::FeatureGeneratorPointer)featureAbsSampleActivity);
        filterFusion->SetDefaultWeightGenerator(
            (ImageFusionFilterType::WeightGeneratorPointer)selectMaximumWeightGenerator);
        filterFusion->OverrideWeightGenerator(
            (ImageFusionFilterType::WeightGeneratorPointer)averageWeightGenerator, 0, 0);
        filterFusion->SetWeightCombiner(
            (ImageFusionFilterType::WeightCombinerPointer)weightCombiner);
        filterFusion->SetInput(0, reader1->GetOutput());
        filterFusion->SetInput(1, reader2->GetOutput());
        filterFusion->Update();
```

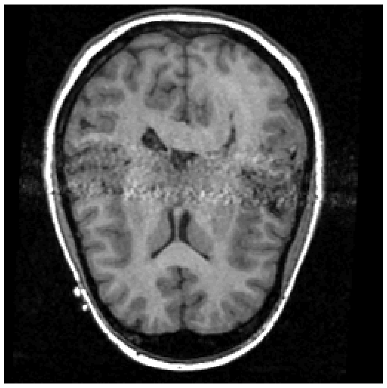
Note the following lines which allow us to override the weight generator for the approximation images in the deconstructed image pyramids:

```
        filterFusion->OverrideWeightGenerator(
            (ImageFusionFilterType::WeightGeneratorPointer)averageWeightGenerator, 0, 0);
```

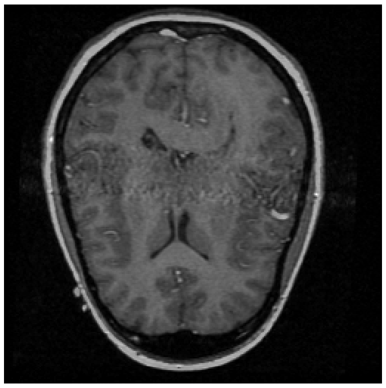
Some results from the example are shown in Figure 5.



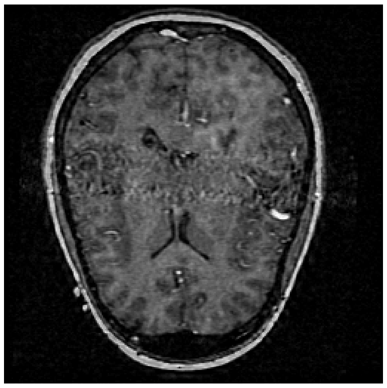
(a) MRA



(b) Pre-contrast T1



(c) Arithmetic Fusion



(d) QMF DWT Fusion

Figure 5: Slice 40 from the output of `Examples/Fusion/giftExampleQmfWaveletFusion1.cxx` with input P2_MRA_RESCALE_WL.mhd and P2_T1_PRE_REGISTERED_WL.mhd. Notice in the fused output that both the vessels (from MRA) and the grey-matter (from pre-contrast T1) are visible. Also note the loss of contrast in the arithmetic fusion output compared to the QMF DWT fusion output. For the approximation band images we used the weight generator `gift::AverageFeature`, and for the detail band images we used `gift::SelectMaximumFeature`. The `Bior_3_5` filter kernel was selected for QMF wavelet decomposition with `NumberOfLevels`=2.

References

[1] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. In *IEEE Transactions on Communications*, volume 31, pages 532–540, 1983. 3.2

[2] E. Canga. Image fusion toolkit for Matlab. Technical report, University of Bath, 2003. http://www.ablen.com/hosting/imagefusion/software/canga-image-fusion-code.zip. 2

[3] P. Geladi and H. Grahn. *Multivariate image analysis*. Wiley, New York, 1996. 2

[4] D. Hall. *Mathematical techniques in multi-sensor data fusion*. Artech House, Boston, 2nd edition, 2004. 2

[5] D. Hall and J. Llinas. Multisensor data fusion. In David L Hall and James Llinas, editors, *The Handbook of Multisensor Data Fusion*. CRC Press, Boca Raton, 2001. 2

[6] J. Lewis, R O'Callaghan, S. Nikolov, D. Bull, and C. Canagarajah. Region-based image fusion using complex wavelets. In *Seventh International Conference on Information Fusion (FUSION)*, volume 1, pages 555–562, 2004. 2, 3.2, 5

[7] H. Li, B. Manjunath, and S. Mitra. Multisensor image fusion using the wavelet transform. *Graphical Models and Image Processing*, 57(3):235–245, 1995. 3.2

[8] Z. Li, Z. Jing, G. Liu, S. Sun, and H. Leung. A region-based image fusion algorithm using multiresolution segmentation. In *IIntelligent Transportation Systems*, volume 1, pages 96–101. IEEE, 2003. 5

[9] G. Matsopoulos, S. Marshall, and J. Brunt. Multiresolution morphological fusion of MR and CT images of the human brain. *IEE Proceedings- Vision, Image and Signal Processing*, 141(3):137–142, 1994. 3.2

[10] S. Mukhopadhyay and B. Chanda. Fusion of 2D grayscale images using multiscale morphology. *Pattern Recognition*, 34(10):1939–1949, 2001. 3.2, 5

[11] S. Nikolov, D. Bull, C. Canagarajah, M. Halliwell, and P. Wells. Image fusion using a 3-D wavelet transform. In *Seventh International Conference on Image Processing And Its Applications*, volume 1, pages 235–239. IEEE, 1999. 3.2

[12] S. Nikolov, P. Hill, D. Bull, and C. Canagarajah. Wavelets for image fusion. In A. Petrosian and F. Meyer, editors, *Wavelets in signal and image analysis*, pages 213–244. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001. 3.2

[13] S. Nikolov, J. Lewis, R. O'Callaghan, D. Bull, and C. Canagarajah. Hybrid fused displays: Between pixel- and region-based image fusion. *Proceedings of the Seventh International Conference on Information Fusion (FUSION 2004)*, 2:1072–1079, 2004. 5

[14] V. Petrovic. Subjective tests for image fusion evaluation and objective metric validation. *Information Fusion*, In Press, Corrected Proof, 2006. 5

[15] G. Piella. A general framework for multiresolution image fusion: from pixels to regions. Technical Report PNA-R0211, Center for Mathematics and Computer Science, 2002. http://www.cwi.nl/ftp/CWIreports/PNA/PNA-R0211.pdf. (document), 2, 3.1, 5

[16] G. Piella. *Adaptive wavelets and their application to image fusion and compression*. PhD dissertation, CWI and University of Amsterdam, 2003. 2

[17] G. Piella. A general framework for multiresolution image fusion: from pixels to regions. *Information Fusion*, 4(4):259–280, 2003. (document), 3.1

[18] G. Piella and H. Heijmans. A new quality metric for image fusion. *IEEE International Conference on Image Processing*, 3:173–176, 2003. 5

[19] T. Pu and G. Ni. Contrast-based image fusion using the discrete wavelet transform. *Optical Engineering*, 39(8):2075–2082, 2000. 5

[20] G. Qu, D. Zhang, and P. Yan. Information measure for performance of image fusion. *Electronics Letters*, 38(7):313–315, 2002. 5

[21] O. Rockinger. Image sequence fusion using a shift-invariant wavelet transform. In *International Conference on Image Processing*, volume 3, pages 288–291, 1997. 3.2

[22] O. Rockinger. Image fusion toolbox for Matlab. Technical report, Metapix, 1999. http://www.metapix.de/toolbox.htm. 2, 3.2

[23] A. Toet and M. Hogervorst. Performance comparison of different graylevel image fusion schemes through a universal image quality index. In Ivan Kadar, editor, *Signal Processing, Sensor Fusion, and Target Recognition XII*, volume 5096, pages 552–561. SPIE, 2003. 5

[24] T. Yoo, M. Ackerman, W. Lorensen, W. Schroeder, V. Chalana, S. Aylward, D. Metaxes, and R. Whitaker. Engineering and algorithm design for an image processing API: A technical report on ITK - the Insight Toolkit. *Proc. of Medicine Meets Virtual Reality*, pages 586–592, 2002. 1

[25] Y. Zheng, E. Essock, and B. Hansen. An advanced image fusion algorithm based on wavelet transform: incorporation with PCA and morphological processing. In *Image Processing: Algorithms and Systems III*, volume 5298, pages 177–187. SPIE, 2004. 5

[26] Y. Zheng, E. Essock, B. Hansen, and A. Haun. A new metric based on extended spatial frequency and its application to DWT based fusion algorithms. *Information Fusion*, In Press, Corrected Proof, 2006. 5