# Numerical Q-Ball Image Reconstruction: an ITK Implementation

*Release 1.00*

Klaus H. Fritzsche[1] and Hans-Peter Meinzer[1]

May 13, 2009

[1]German Cancer Research Center, Division of Medical and Biological Informatics

**Abstract**

This document describes an implementation of David S. Tuch's numerical Q-Ball reconstruction algorithm [3] implemented using the Insight Toolkit ITK `www.itk.org`. The document is accompanied with the source code, input data, parameters and output data that were used for validating the implementation.

## Contents

## 1 Background

Diffusion imaging data sets provide information about the diffusion of water at each voxel in the image and thus reveal powerful information for mapping neural histoarchitecture in vivo. The interested reader may refer to [2] for an introduction to different diffusion imaging techniques.

In order to interpret the raw diffusion signal from the scanner, a reconstruction of diffusion profiles at each voxel position needs to be performed. The well known Diffusion Tensor model assumes gaussian diffusion and thus has its limitation especially in areas with multiple fiber directions within one voxel (e.g. crossing or fanning fiber tracts). To overcome these limitations, Q-Ball imaging was introduced allowing model free reconstruction of high angular resolution diffusion imaging data.

This paper presents an easy to use, multithreaded ITK filter for performing Q-Ball image reconstruction using the analytical algorithm described by David S. Tuch in 2004 [3], which derives a linear matrix formulation for the q-ball reconstruction based on spherical radial basis function interpolation.

## 2 Description

The diffusion orientation distribution function (ODF) $\Psi(\mathbf{u})$ is defined as the radial projection of the diffusion function,

$$\Psi(\mathbf{u}) = \frac{1}{Z}\int_0^{\inf} P(r\mathbf{u})dr, \tag{1}$$

where Z is a dimensionless normalization constant. The Q-Ball imaging (QBI) reconstruction is based on the Funk-Radeon Transform (FRT). Given a function on the sphere $f(\mathbf{w})$, the FRT is defined as the sum over the corresponding equator, i.e., the set of points perpendicular to $\mathbf{w}$. The FRT for a direction $\mathbf{u}$ can be written

$$\mathrm{FRT}\left[f(\mathbf{w})\right](\mathbf{u}) = \int_{w\in\mathbf{u}^{\perp}} f(\mathbf{w})d\mathbf{w} \tag{2}$$

The reconstruction algorithm assumes

$$\Psi(\mathbf{u}) \approx \frac{1}{Z}\mathrm{FRT}\left[E(\mathbf{q})\right], \tag{3}$$

where $E(\mathbf{q})$ is the measured MR diffusion signal for a given direction $\mathbf{q}$ (Here we omitted the definition of the extended FRT, for details refer to [3]).

Implementing the FRT in practice requires a numerical procedure for calculating the equator integral. Since the equator points will not coincide with the diffusion wavevector sampling points it is necessary to regrid the diffusion data onto the equatorial circles. The regridding is implemented using a form of spherical interpolation called spherical radial basis function (sRBF) interpolation using spherical Gaussian kernels.

## 3 Implementation

The interface of `itk::DiffusionQballReconstructionImageFilter` was designed to be similar with `itk::DiffusionTensor3DReconstructionImageFilter`. The ITK `itk::DiffusionTensor3DReconstructionImageFilter` is another example for a filter that takes as input multiple MR image volumes acquired with varying gradient directions and strengths.

The class is templated over the pixel type of the reference and gradient images (expected to be scalar data types), the internal pixel type of the ODFs, the number of resulting ODF directions and the number of basis function centers for the sRBF.

There are two ways to use this class. With one reference image and n gradient images, the class can be used as

- `filter->SetReferenceImage( image0 );`
- `filter->AddGradientImage( direction1, image1 );`
- `filter->AddGradientImage( direction2, image2 );`
- ...

With multiple gradient and reference images in a single multi-component image (VectorImage), the images can be specified simply as

- `filter->SetGradientImage( directionsContainer, vectorImage );`

where `directionsContainer` contains a gradient vector for each of the images. The gradient vector for reference images must be set to $[0\ 0\ 0]^T$.

A number of Get/Set methods are provided for controlling the output of the filter:

- At least 6 gradient images must be specified for the filter to be able to run. If the input gradient directions $g_i$ are majorly sampled on one half of the sqhere, then each input image $I_i$ will be assigned to two gradient directions $-g_i$ and $g_i$ in order to guarantee stability of the algorithm.
- `Threshold` - Threshold on the reference image data. The output ODF will be a null pdf for pixels in the reference image that have a value less than this.
- `BValue` - B-value, not yet in use, but potentially usefull for the implementation of different normalization methods.
- `NrEquatorSamplingPoints` - Number of sampling points on equator when performing Funk Radeon Transform (FRT). If not set, this defaults to $\sqrt{8*\pi*\#\text{GradientDirections}}$.

The reconstruction is performed over every voxel in the output image region across the entire image using the overridden `void ThreadedGenerateData()` function.

## 3.1 Handling of Half Shell Acquisition Schemes

Half shell acquisition schemes are detected by applying a threshold on the center of mass of the gradient directions. In order to guarantee stability of the algorithm, they are handled by assigning two gradient directions $-g_i$ and $g_i$ to each input image $I_i$. The member `DirectionsDuplicated` indicates, weather such a scheme was detected.

## 3.2 ODF Normalization

The protected member functions `PreNormalize()` and `Normalize()` are called right before and after multiplication of the reconstruction matrix. They allow implementation of different normalization methods that can be configured using the member `NormalizationMethod`. The current implementation realizes two normalization methods: `QBR_STANDARD` and `QBR_NONE`. `QBR_STANDARD` ensures unit mass for the resulting ODFs, `QBR_NONE` leaves the results untouched. An example for further normalization schemes would be the min-max normalization for visualization of the ODFs.

## 3.3 Modifications to Tuch's Reconstruction Algorithm

One addition to the original tuch algorithm was introduced in order to get more stable results especially in cases where the sphere is only sparsely populated by directions. Each row of the matrix $G * H_+$ is normalized to unit mass.

```
    // the following two loops resemble an addition to the original tuch
    // algorithm that was introduced in order to get more stable results
    // especially in case the sphere is only sparsely populated
    typename vnl_matrix<double>::iterator it3;
    for( it3 = (*GH_plus).begin(); it3 != (*GH_plus).end(); it3++)
    {
      if(*it3<0.0)
        *it3 = 0;
    }

    for(int i=0; i<NOdfDirections*m_NumberOfEquatorSamplingPoints; i++)
    {
      vnl_vector< double > r = GH_plus->get_row(i);
      r /= r.sum();
      GH_plus->set_row(i,r);
    }
```

A second addition to the original tuch algorithm was introduced in order to preserve magnitudes of measured values: Each row of the reconstruction matrix is normalized to unit mass. This only takes effect in cases where the `NormalizationMethod` is set to `QBR_NONE`.

```
// this is also an addition to the original tuch algorithm that was
// introduced in order to preserve magnitudes of measured values.
// In case of standard normalization this has no effect.
for(int i=0; i<NOdfDirections; i++)
```

```
{
  vnl_vector< TOdfPixelType > r = m_ReconstructionMatrix->get_row(i);
  r /= r.sum();
  m_ReconstructionMatrix->set_row(i,r);
}
```

### 3.4 Point Distribution on the Sphere

The class `itk::PointShell`, which is distributed with this publication, takes care of evenly distributing points on a sphere. For the special cases of 12, 42, 92, 162, 252, 362, 492, 642, 812 or 1002 points the distributions are given by hard coded coordinates originally calculated by n-fold subdivisions of an icosahedron.

In all other cases, the computation follows the method described by Rakhmanov *et al.* [1].

## 4 Performance

A Performance analysis was performed on a Intel Core2 Quad Processor (Q6600 at 2.40GHz each) with 4Gb of RAM and Visual Studio 9.0 compiler. We tested images with varying resolution and varying number of gradient directions. In all cases we reconstructed ODFs with 252 directions and averaged the computation times over 10 repetitions (refer to Figure 1). Single and multiple core reconstructions only differ in the actual time taken for reconstruction while *Choosing Sigma* and *Building Matrix* is performed in the `BeforeThreadedGenerateData()` method on a single core for all cases.

In order to perform a similar performance analysis under different settings, the provided example code can be run with corresponding options for number of threads and averaged repetitions. In order to enable timing,

```
#define __QBR_TIMING__
```

has to be commented in at line 57 of the file `DiffusionQballReconstructionImageFilter.cxx`.

## 5 Usage

The following example demonstrates how to use the filter described in the previous sections. Data and source for the example are included with this article.

```
typedef short int          ReferencePixelType;
typedef short int          GradientPixelType;
typedef float              OdfPrecisionType;

// The angular resolution of the resulting ODFs depends on the number of
// directions distributed over the sphere
const int nOdfDirections = 252;

// type definitions
typedef itk::DiffusionQballReconstructionImageFilter<
  ReferencePixelType, GradientPixelType, OdfPrecisionType,
  nOdfDirections, nOdfDirections >
```

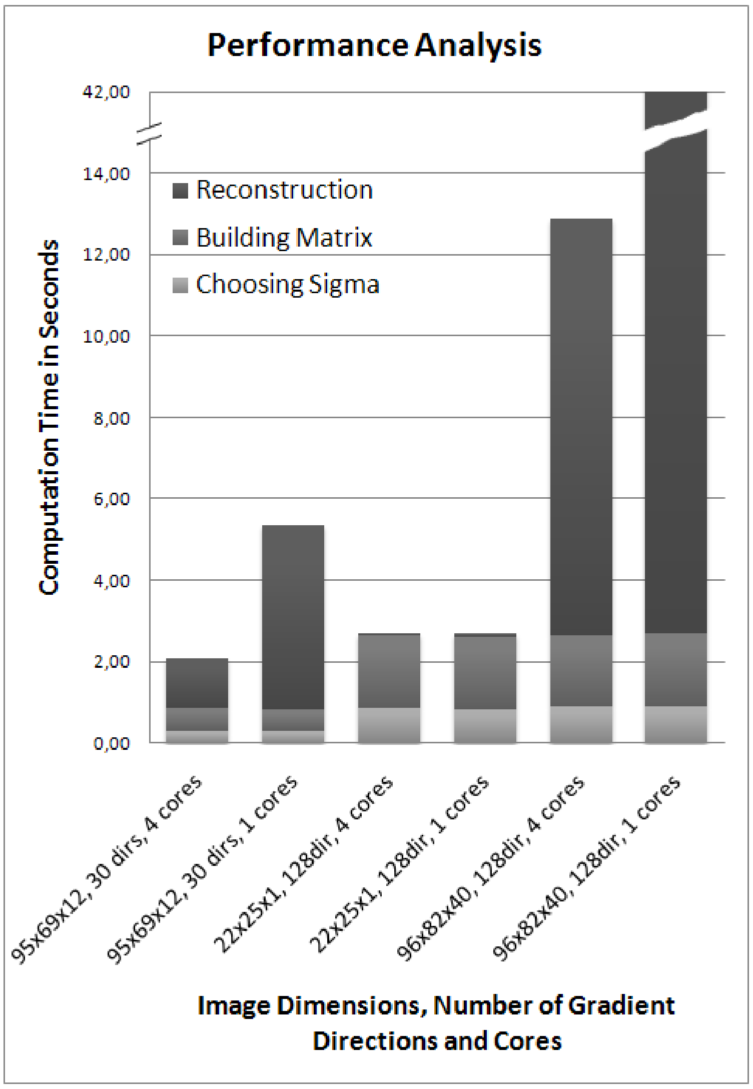

Figure 1: Computation times in seconds for reconstruction of images with different sizes and different number of gradient directions using single and multiple cores.

```
  QBallReconstructionImageFilterType;

typedef QBallReconstructionImageFilterType::GradientImagesType
  GradientImageType;
typedef QBallReconstructionImageFilterType::OutputImageType
  OutputImageType;
typedef QBallReconstructionImageFilterType::BZeroImageType
  BZeroImageType;

typedef vnl_vector_fixed< double, 3 >   GradientDirectionType;
typedef itk::VectorContainer< unsigned int,
  GradientDirectionType >               GradientDirectionContainerType;

// gradient directions retreived somehow, see example files for details
GradientDirectionContainerType::Pointer gradientDirections = ... ;

// input images retreived somehow, see example files for details
GradientImageType::Pointer images = ... ;

// init and run the reconstruction
QBallReconstructionImageFilterType::Pointer qBallReconstructionFilter =
  QBallReconstructionImageFilterType::New();

// set threshold on reference image (defaults to 0)
qBallReconstructionFilter->SetThreshold(70);

// set standard normalization (defaults to QBR_STANDARD)
qBallReconstructionFilter->SetNormalizationMethod(
  QBallReconstructionImageFilterType::QBR_STANDARD);

// set b-value, currently has no effect on anything (optional)
qBallReconstructionFilter->SetBValue(3500);

// set input images
qBallReconstructionFilter->SetGradientImage(
  gradientDirections, images );

// update
qBallReconstructionFilter->Update();

// get filter results
OutputImageType::Pointer odfs   = qBallReconstructionFilter->GetOutput();
BZeroImageType::Pointer b0Image = qBallReconstructionFilter->GetBZeroImage();
```

## 6 Conclusion

In this paper we present an ITK implementation of the numerical Q-Ball reconstructin algorithm present by David S. Tuch in 2004 [3]. All parameters default to meaningful values in order to ease use of the filter. The implementation allows multithreaded reconstruction reducing computation time on multi-processor systems. Please feel free to contact the author for suggestions or bugs (http://www.dkfz.de/de/mbi/people/Klaus_Fritzsche.html). An examplary visualization of min-

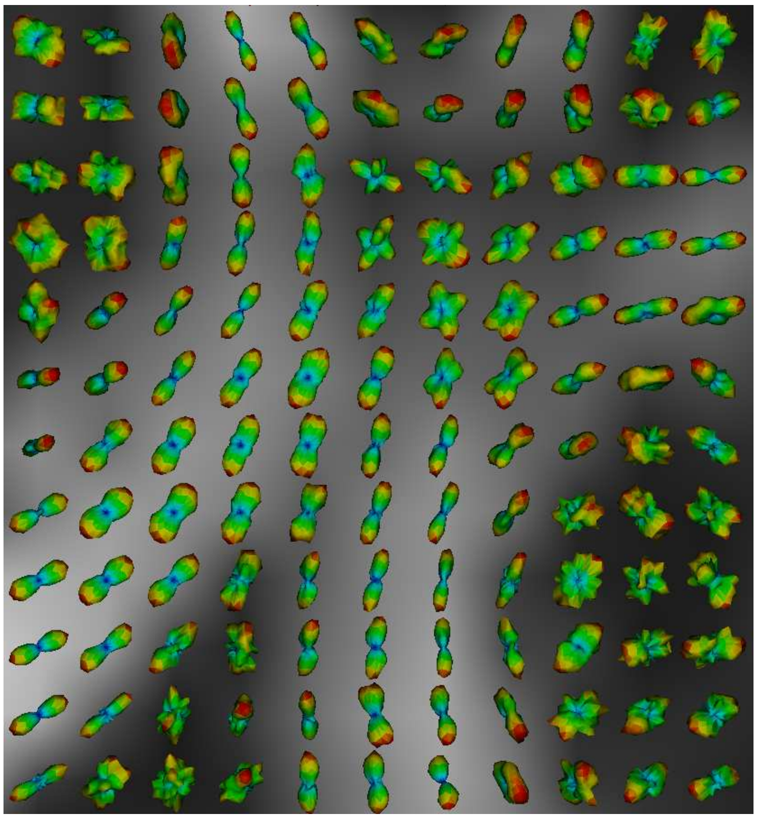

Figure 2: Exemplary visualization of ODFs reconstructed from a transversal slice of a human brain.

max normalized ODFs can be seen in Figure 2.

## References

[1] Rakhmanov E A, Saff E B, and Zhou Y M. Minimal discrete energy on the sphere. *Mathematical Research Letters*, 1:647–662, 1994. 3.4

[2] Patric Hagmann, Lisa Jonasson, Philippe Maeder, Jean-Philippe Thiran, Van J Wedeen, and Reto Meuli. Understanding diffusion mr imaging techniques: from scalar diffusion-weighted imaging to diffusion tensor imaging and beyond. *Radiographics*, 26 Suppl 1:S205–S223, Oct 2006. 1

[3] David S Tuch. Q-ball imaging. *Magn Reson Med*, 52(6):1358–1372, Dec 2004. (document), 1, 2, 6