# Reconstruction Demo

I have created a demo reconstruction (‘demo\_gridding.m’) to illustrate the basic use of the gridding code. The demo data I from a radial ventilation image.

## 1.1 Compiling MEX Code

In order to run the demo, you will need to be able to compile MEX code. To setup for MEX compiling:

1. Download and install [Visual C++ 2010 Express](http://www.microsoft.com/visualstudio/eng/downloads) (we need the c-compiler from it)
2. Download and install Wicrosoft Windows SDK 7.1
3. Run "mex -setup" from MATLAB and choose the Visual C++ version you just installed
4. Compile my MEX code by running "mex -g grid\_conv.c" in MATLAB

The above instructions have only been tested on Windows 7 with MATLAB version 2011a. If you have trouble installing another version of MATLAB you may need another compiler (<http://www.mathworks.com/support/compilers/R2012a/win64.html>).

If you have trouble getting MEX code to compile, here is MATLAB's MEX help (<http://www.mathworks.com/help/matlab/ref/mex.html>).

Once you are able to compile MEX code, you should be able to run the demo.

# Adapting Reconstruction for your needs

To modify the code for your specific application, you will need to at a minimum change the “data” and “coords” variables.

The “data” variable holds the complex data of each acquired k-space point. The format is very important. The “data variable is a 2xn matrix where n is the total number of k-space samples. Each column of the “data” variable is one sample point. The first row is the real part and the second row is the imaginary part of the complex number. Thus, the “data” variable could be illustrated by:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Pt 1** | **Pt 2** | **Pt 3** | **…** | **Pt n** |
| **Real** | real(dat\_1) | real(dat\_2) | real(dat\_3) | … | real(dat\_n) |
| **Imag** | imag(dat\_1) | imag(dat\_2) | imag(dat\_3) | … | imag(dat\_n) |

The “coords” variable holds the normalized coordinates of each data point along the k-space trajectory. The coordinates are normalized to lie between -0.5 and 0.5. The “coords” variable is a 3xn matrix where n is the total number of k-space samples. Each column of the “coords” variable is one sample point. The first row is the normalized x position, the second row is the normalized y position, and the third row is the normalized z dimension. Thus, the “coords” variable could be illustrated by:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Pt 1** | **Pt 2** | **Pt 3** | **…** | **Pt n** |
| **x-position** | dat\_1\_x | dat\_2\_x | dat\_3\_x | … | dat\_n\_x |
| **y-position** | dat\_1\_y | dat\_2\_y | dat\_3\_y | … | dat\_n\_y |
| **z-position** | dat\_1\_z | dat\_2\_z | dat\_3\_z |  | dat\_n\_z |

There are other reconstruction parameters that are variable for reconstruction optimization. A brief summary of each variable is offered below. For a more complete discussion, please read the recommended reading at the end of this README.

kernelwidth – describes the width of the kernel in the same units as the output dimensions. For example if kernelwidth is 1, then the kernel will only convolve data into pixels that are within one pixel. The kernel shape is also affected by kernel width. We typically leave kernel width at 1.

kernel\_lut\_size – Determines the number of values in the kernel lookup table. Because computing the exact kernel value for each point would take a very long time, we precalculate a finely sampled version of the kernel. Each computation of the kernel during gridding is thus just a linear interpolation of the presampled kernel. The higher kernel\_lut\_size is, the more accurate the estimation of the exact kernel value.

overgridfactor – Describes the amount of oversampling. Oversampling is a technique where you grid the data at a higher resolution in frequency space (same bandwidth, but more samples). The image domain effect is a larger FOV. The benefit to oversampling is that aliases are pushed further apart due to the larger FOV, so aliasing effects can be reduced through overgridding. Note, while overgridding is useful, it is very memory intensive (memory usage will go up proportional to overgridfactor3). We typically use an overgridfactor of 3, though many papers suggest that 2.1 is sufficient for most applications.

output\_dims – Defines the output image matrix size. You should consider Nyquist sampling when chosing this matrix (the matrix is [xdims ydims zdims]).

num\_dcf\_itter – Defines the number of iterations of the density compensation estimation algorithm. Density compensation is a very important part of gridding, thus an accurate density compensation estimation is necessary. This iterative DCF estimation is quite accurate and penalizes high frequencies based on aliasing; however it is also extremely slow. You can precompute the DCF values once, write them to file, then just use the precomputed values for future gridding (as long as the trajectories are the same). I should note that higher DCF iterations are not always better. You need to iterate a few times to get a decent DCF estimate, but sometimes the algorithm seems to diverge with too many iterations. 10-20 iterations seems to be reasonable for most of our gridding applications.

image\_size\_dcf – Describes the size of a temporary matrix for computing the DCF estimates. Using a smaller temporary matrix size will result in faster iterations (and quicker initial convergence), but using a large temporary matrix size will ultimately give more accurate DCF values. It is possible to run a few iterations with a small matrix size for a quick estimate, then switch to a larger matrix size for accuracy.

# Alternatives to this code

The following is a list of other open source gridding code:

1. Jim Pipe

<http://cds.ismrm.org/protected/MRI_Unbound/grid3_dct_11aug.zip>

1. Stanford:

<http://www-mrsrl.stanford.edu/~brian/gridding/>

<https://www.stanford.edu/group/bmr/cgi-bin/mediawiki/index.php/Software>  
<http://rsl.stanford.edu/research/software.html>  
<http://www-mrsrl.stanford.edu/links.html>

1. Jeff Fessler, University of Michigan  
   <http://www.eecs.umich.edu/~fessler/code/mri.htm>
2. Miki Lustig, University of California Berkeley  
   <http://www.eecs.berkeley.edu/~mlustig/Software.html>
3. Graz University of Technology (AGILE, open source GPU code for reconstruction)  
   <http://www.imt.tugraz.at/index.php/research/agile-gpu-image-reconstruction-library>
4. Berkin Bilgic, MIT  
   <http://web.mit.edu/berkin/www/software.html>  
   Forschungszentrum Juelich GmbH (open source MRI simulator and recon/pulse design code)  
   [www.jemris.org](http://www.jemris.org/)   
   [www.codeare.org](http://www.codeare.org/)
5. Leslie Ying, SUNY-Buffalo  
   <http://www.acsu.buffalo.edu/~jlv27/index_files/software.htm>

# Background Reading

The following is a list of background reading recommended for understanding the details of this gridding algorithm:

1. Jackson, J. I., Meyer, C. H., Nishimura, D. G., & Macovski, A. (1991). Selection of a Convolution Function for Fourier Inversion Using Gridding, I(3).
2. Pipe, J. G., & Menon, P. (1999). Sampling density compensation in MRI: rationale and an iterative numerical solution. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine, 41(1), 179–86. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/10025627
3. Beatty, P. J., Nishimura, D. G., & Pauly, J. M. (2005). Rapid Gridding Reconstruction With a Minimal Oversampling Ratio, 24(6), 799–808.
4. O’Sullivan, J. D. (1985). A fast sinc function gridding algorithm for fourier inversion in computer tomography. IEEE transactions on medical imaging, 4(4), 200–7. doi:10.1109/TMI.1985.4307723
5. Johnson, K. O., & Pipe, J. G. (2009). Convolution Kernel Design and Efficient Algorithm for Sampling Density Correction, 447, 439–447. doi:10.1002/mrm.21840
6. Zwart, N. R., Johnson, K. O., & Pipe, J. G. (2012). Efficient Sample Density Estimation by Combining Gridding and an Optimized Kernel, 710, 701–710. doi:10.1002/mrm.23041