# Image Processing
# (Assessed Coursework 2)

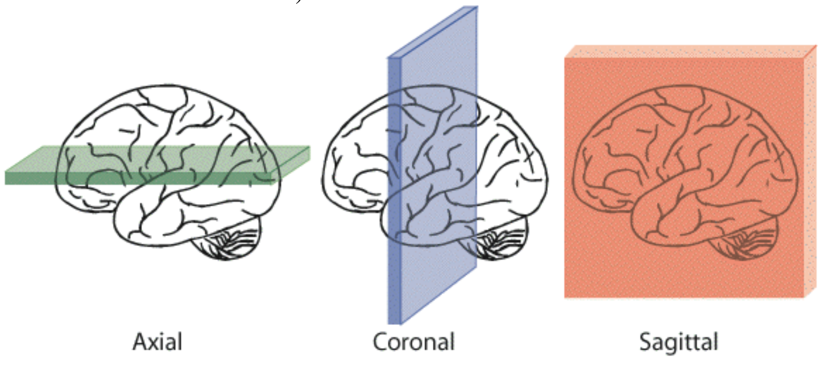
*It is expected that you will start this piece of work during a computer lab session, and that it should be written up as a short report (3-4 pages including any illustrations and images) to be handed in on Friday 22nd March 2013. The work will be assessed and counts 10% towards your overall mark for the course.*

*Your work should show how tomographic reconstruction from projections may be approached using the central slice theorem and how this may be implemented in real space using Filtered Back-projection (FBP). You should illustrate this with images based on the work below.*

Open the sample image 'T1 Head'. This is a stack of 129 "**sagittal"** slices [vertical front to back slices] through the human head. Each slice is a 256x256 2D image, and together they form a 3D image. Use the slider to scroll through the slices. We are going to use this dataset to demonstrate how tomographic images are reconstructed, so first we must do the equivalent of a CT scan on this head.

[For those of you who have taken "Medical Imaging" – note that this image was measured using MRI, and of course MRI images are usually measured and reconstructed directly in Fourier space, not using the backprojection techniques discussed below, which are mainly used for CT and tomographic nuclear medicine. We are only using this particular image because it is provided in the ImageJ samples.]

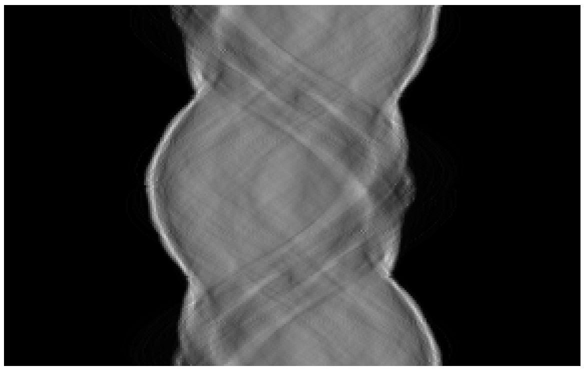
Use Image> Stacks> Reslice (default parameters) . You should now have a stack of 256 horizontal slices through the head, working down into the neck. These are **"axial"** slices. Basically a CT scanner attempts to measure axial slices, but the data is not acquired in this form but as a series of projections. Choose a particularly interesting slice (for example, one through the eyes, and remember which of the 256 slices it is).



Go back to the original "T1 Head" stack and use Image>Stacks>3D Project to calculate the effect of projecting this image at 180 angles from $0^o$ to $360^o$. ImageJ first has to convert the image to 8-bit. The "projection method" is "mean value", the "axis of rotation" is "y-axis", and the "rotation angle increment" should be set to 2 (degrees).

You should have a stack of 180 images each showing what the head looks like from a particular angle (averaged or integrated over its depth). This is what is measured by a tomographic scanner.

Now do Image>Stacks>Reslice on the projected dataset. Instead of getting nice axial slices through the head you should get a stack of 256 images like the one below.



Each image corresponds to a different axial slice and contains all the information needed to reconstruct this slice. Each row of the image is a projection of that slice at a different angle (the angle changes in the y-direction of the image – 2 degrees per row in this case.) Such images are called ‘Sinograms’ and the mathematical operation that maps the original axial slice image to its sinogram is called the ‘Radon Transform’ after the Austrian mathematician Johann Radon.

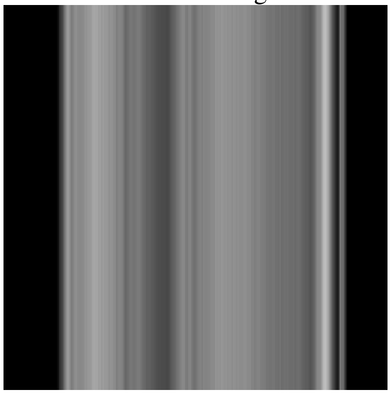
Select the <u>one</u> of these sinogram images that corresponds to the “interesting” slice you chose earlier. Use Edit>Copy followed by File> New > Internal Clipboard to create a single 2D image of this sinogram. It should be 288 pixels wide. Since later we shall need to perform an FFT on each row, crop the image to be exactly 256 pixels wide (for example using Image>Adjust>Canvas Size), then save it as a file (e.g. as sinogram.tif).

<u>Backprojection</u>

The first approach to reconstructing the axial slice from the sinogram uses backprojection. Take your selected sinogram. Use the following Macro to reproduce it 256 times, creating a new stack of 256 identical sinogram images:

```
run("Select All");
run("Copy");
for(i= 1;i<256;i++)
        {
        run("Add Slice");
        run("Paste");
        }
```

Now reslice the stack created (Image>Stacks>Reslice using default parameters). This should create a stack of 180 images, each of which should look something like the one shown below.



Each of these images is the view from a different angle of the selected slice, smeared out vertically. These are effectively backprojections, except that we need to rotate each to the appropriate angle. Convert this stack to 32 bit and save it, e.g. as backprojections.tif.

POINT A

We now need to rotate each image by the appropriate angle θ. To do this use the following Macro:

```
for(i= 1;i<181;i++)
        {
        setSlice(i);
        run("Arbitrarily...", "angle="+2*(i-1)+" interpolate slice");
        }
```

[Save this macro as you will need it again.]
Save the resulting stack of 180 rotated projections as 'Rotated_Back_Projections.tif'

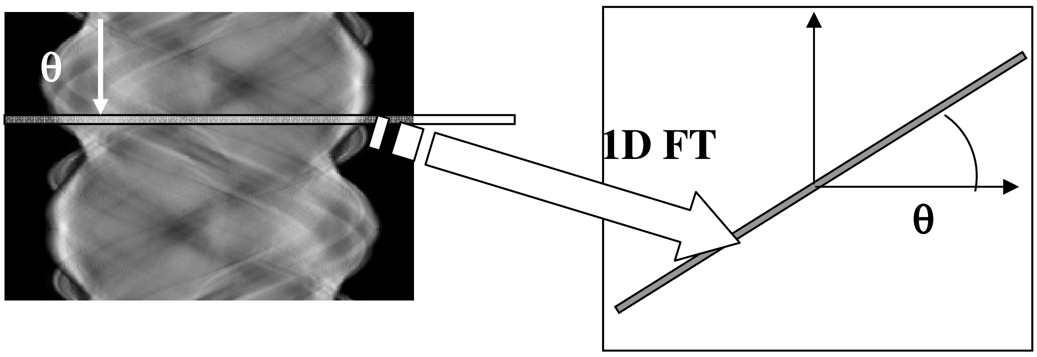
Now we simply add up all the projections in Rotated_Back_Projections.tif :
use Image> Stacks> Z Project… and select the Sum Slices option.

POINT B

Save the resulting image as "Backprojected image.tif"
This process we used to get this image is referred to as "Backprojection" (and consists of spreading each projection back along the appropriate angle, and then summing all the projections).
The result should be at least recognizable as something like one of the slices from the original MRI Stack, however our reconstruction process is clearly not 'quite right' – the image is very fuzzy!
The Backprojection process gives an image with the correct features but long tails.

## Central Slice Theorem

The central slice theorem says that we can use each row of the sinogram to build the 2D FT of the axial slice since the 1D FT of each row forms a radial (central) slice of the 2D FT.



Now there is a small problem. ImageJ is designed for image processing and has no facility for 1D Fourier Transforms. However, we can use a simple trick to overcome this – if take the 2D Fourier Transform of one of our rotated backprojections, since it does not vary along the direction of projection, but only perpendicular to the direction of projection, the 2D FT of this should be the 1D FT of the projection, rotated to the correct angle – so exactly what we want.

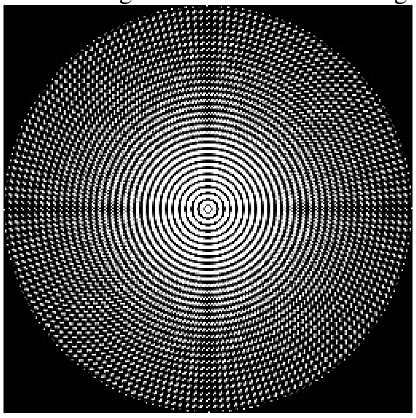
Select an arbitrary slice from this stack and use Process>FFT>FFT. Satisfy yourself that this gives a 1D Fourier transformed profile at the appropriate angle θ. [Actually it probably contains some other features due to the fact that we have truncated the smearing of the backprojection.] According to the Central Slice Theorem we can reconstruct the image by taking the FT of each projection in this way, adding them up, and then inverting the FT.

But since the FT is a linear operation, this is the same as adding the images first, then taking the FFT followed by the inverse FFT, which clearly gets us back to where we started from. In other words this would give us the backprojected image we had previously, with its tails.

## Filters

What was wrong with the argument from the Central Slice Theorem?

The answer is that our values are not uniformly sampled. Had we actually gone to the trouble of building the 2D FT from central slices it might have looked something like the following:



Notice how the central slices produce a very large number of samples of the 2D FT near $(k_x,k_y)= (0,0)$ and fewer samples (per unit area) at large $|\underline{k}|$. In fact there are the same number of samples at each radius – but they are spread out over an area that increases in proportion to $|\underline{k}|$.

If we were using the 2D FT like a 'recipe book' to build the slice we would visit each pixel and produce a basis image with the amplitude prescribed by the recipe (i.e. the pixel value.) This is fine if the samples are uniformly spread out, but in the above situation they are not – the number of samples per unit area is $\propto {}^{1}\!/_{|\underline{k}|}$. It is still OK to use each one, but first we need to multiply by a sampling density weighting factor i.e. by the function $f(\underline{k}) = |\underline{k}|$.

To correct this effect we need to apply a filter. There are two ways to do this:

1. In frequency space

Since the problem is caused by the variation in sampling in frequency space, we can simply apply a correction factor $|\underline{k}|$

Create a ramp filter using the following macro:

```
newImage("RampFilter", "32-bit Black", 256, 256, 1);

X= 128;
Y= 128;

for(x= 0;x<256;x++)
        {
        for(y= 0;y<256;y++)
                {
                ramp= sqrt((x-X)*(x-X)+(y-Y)*(y-Y))/100;
                if(ramp<1.28)setPixel(x,y,ramp);
                }
        }
```

This should create an image whose intensity varies linearly with radial distance from the centre (128,128) out to a radius of 128 pixels.

Now take the Fourier transform of your backprojected image. First ensure that Process> FFT>FFT Options has "Complex Fourier Transform" checked, and then use Process>FFT>FFT. In addition to the 8 bit FFT window (which shows the amplitude of the FT) you should get a stack, called Complex, consisting of two images, which are respectively the Real and Imaginary part of the Fourier transform of your backprojected image.

Multiply this complex FT by the Ramp Filter: use Process>Image Calculator with the operation set to Multiply and the names of your Complex image and your Ramp Filter. Create a new 32 bit image. ImageJ will ask you if you want to "Process all 2 images" – answer Yes.

This should create a new stack of two images which are the Real and Imaginary parts of the original FT multiplied by the Ramp Filter. Use Process>FFT>Inverse FFT to do the inverse FT on these.

The Real part (first slice) of the resulting image should be similar to the original image, but with some artefacts.

## 2. In real space.

Instead of doing the FFT and then inverse FFT, people often prefer to filter in real space. To see how to do this, note that multiplying each central slice by $|\underline{k}|$ is the same as convolving its inverse FT with the inverse FT of $|\underline{k}|$.

This means that the backprojection method will work provided we first convolve each backprojection with a suitable filter, whose kernel is given by $\mathrm{FT}^{-1}(|\underline{k}|)$. Unfortunately this inverse does not exist, but it can be found provided we limit $k$ to a finite range – and we are only interested in values of $k$ less than the Nyquist frequency.

To calculate $\mathrm{FT}^{-1}(|\underline{k}|)$ select your previous Ramp Filter image and do Process>FFT on this (ImageJ won't allow you to take an inverse transform of this image because it doesn't have an imaginary component, but since it is symmetrical a forward transform will give exactly the same answer anyway.) This should produce a Complex stack of two images, containing the real and imaginary parts of the FT. To get the filter function, select a horizontal profile across the exact centre of the real part (select the line and then use Analyze>Plot Profile) - you need to be very precise in locating the central profile – one pixel 'out' will really mess things up! Use List to print out the values along this profile. For our filter kernel we need only the few central values. You should have a central large positive value with negative values either side. In Process>Filter>Convolve construct a 5x5 filter kernel from the central 5 values of your profile to filter horizontally (you will need to repeat the same row 5 times). Then recall your saved Back_Projections.tif file, filter all the planes using your filter, and repeat the steps between POINTA and POINTB on the previous page to obtain the filtered backprojection image.

You may want to repeat this using 7x7 or 9x9 filters. Whatever values you end up with, they will only be an approximation to the ideal filter function and probably won't work spectacularly well (although the reconstructed image should be considerably better than the unfiltered backprojection.) As a tip, filters with elements that sum to zero work best, so you can tweak the numbers a little to achieve this.

## Extension:

A major problem in actual tomography is that the measured data are noisy. We can simulate this by adding noise to the Back_Projections.tif images (using Process>Noise>Add Specified Noise). Start by trying noise with a standard deviation of 1. Then try to reconstruct the image using either approach above. Increase the noise (by increasing the standard deviation) until the image reconstruction starts to break down. You may be able to restore the image by applying a low pass filter in frequency space.