Image Reconstruction from Multiple Projections

ECE 6258
Class project

By: Anil Rohatgi
Gtg618b
Georgia Institute of Technology

Introduction:

The ability to reconstruct an object from multiple angular projections is a powerful tool. What this procedure gives people is the ability to capture spatial information about a multidimensional object and store it in an array of lesser dimensional signals. Not only does this make image storage easier, but it also allows the user to capture depth and substance information in a noninvasive manner.

Medical imaging uses these techniques to gain visual knowledge of a patient in areas that are not easily physically accessible. Techniques such as X-ray photos or computer tomography, and NMR imaging are just a few uses for this procedure. In fact, in 1971 the fathers of computer tomography Sir Godfrey Hounsfield and Dr. Alan Cormack, were awarded the first Nobel Prize in digital image processing for their achievements.

Projections:

The theory behind this technology is as follows. The object to be captured is centered on a surface. A devise is used to propagate multiple waves through the object. As these waves pass through the object, they store depth information. The waves are received on the opposite side of the object, and the value of each wave is recorded on an axis. The more material between the emission plane and the receptor plane, the higher the recorded value will be. The result of this operation is a depth projection of a multi-dimensional object on a lesser dimensional axis. These readings are taken at multiple angular orientations to the object. Figure 1 graphically shows this procedure.

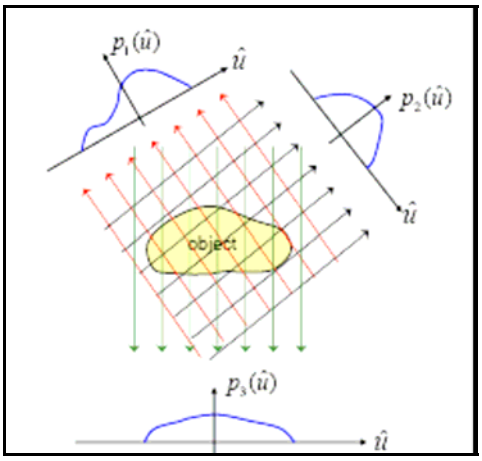


Figure 1. Projection capture geometry.

The mathematical equation for a projection along the vertical axis of the object is as follows.

$$projection_o(u) = \int Object(u,v)dv$$

Equation 1.

A more generalized equation for the projection of an object from any arbitrary angle on to the u axis is shown below.

$$P_{\Theta}(u) = \int Ob(u\cos\Theta - v\sin\Theta, -u\sin\Theta + v\cos\Theta)dv$$

Equation 2.

For this project, we were given 180 projections given at one degree increments to the object, and asked to blindly reconstruct the object based upon these projections. These projections are plotted below. The following paper describes multiple procedures to reconstruct the image, and identify its attributes.

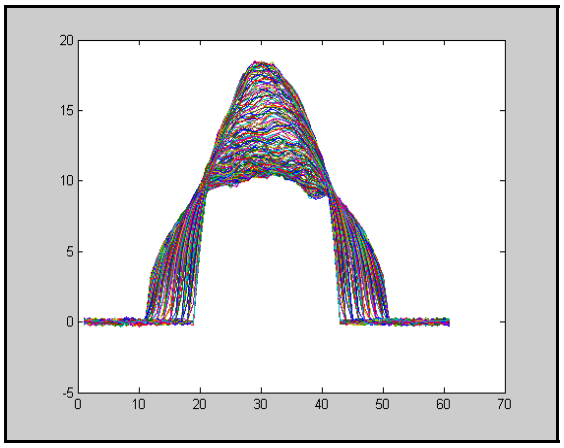


Figure 2. All of the projections plotted at once

Reconstruction:

To reconstruct an image based on one dimensional projection, there are three basic techniques.

1.) Filtered back projections
2.) Iterative approach
3.) Fourier reconstruction.

In this paper we will explore and analyze the first two approaches, and view the results of their completion.

Filtered Back Projections:

The quickest and easiest method of image reconstruction is the filtered back projection approach. What this approach basically does is begin with a blank zero valued image. Next, each projection is lined up along the image in its correct orientation. The values of these projections are then projected onto the correct corresponding pixels in the image and added to their existing value. The pixels that receive the value of each sample of the projection are those which are intersected by a ray extending from the projection sample oriented perpendicular to the axis of projection. This procedure is known as back projecting. Figure 3 below graphically shows this procedure.

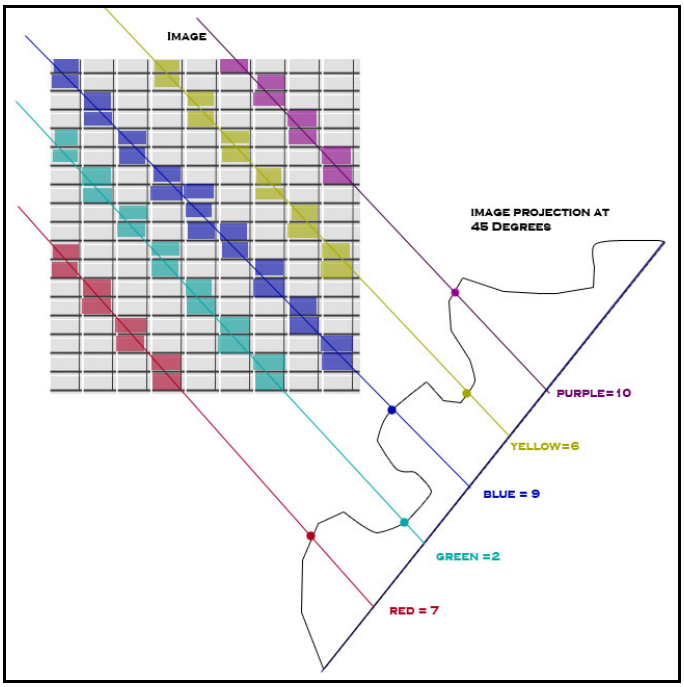


Figure 3. Graphical representation of back projection

Once this back projecting procedure has been run for all of the projections, the resulting image is an approximation of the original image. The figure below shows the results of a direct back projection.

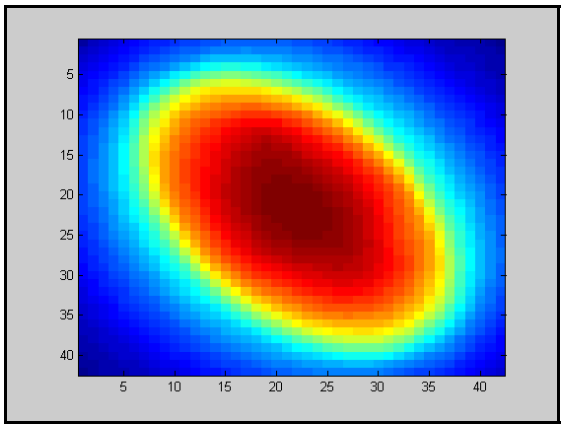


Figure 4. Results of direct back projection

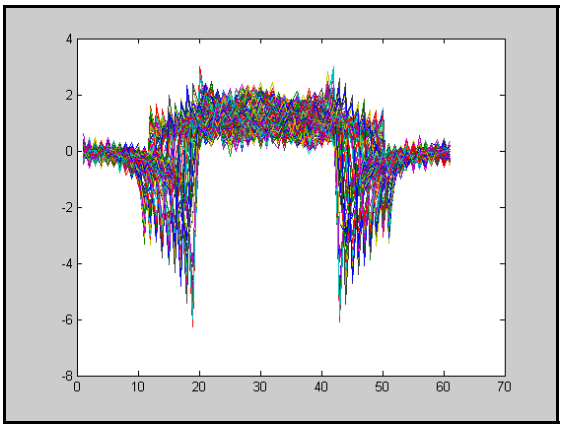


Figure 5. Filtered projections with amplified high frequency

As you can see, some shapes begin to emerge from the back projections, and a general trend in the image can be seen. However, it appears that there is no high frequency content remaining in the image. This is a result of the back projection algorithm adding such large values to the slowly varying parts of the image that the high frequency content gets washed out.

It is for this reason that a filter needed to be applied to the projections themselves before they are back projected on to the image. The filter needed to amplify the high frequency changes in the projection while keeping the low frequency changes the same. Essentially, it weighs the high frequency components so that they stand out in the reconstruction. This was accomplished by shifting the Fourier transform of each projection so that the high frequency contents of the image now dominate the middle of the spectrum. This resulted in their amplification. Below is a plot of the newly amplified projections to be compared with the original projections shown in figure 2.

After the projections were filtered, they were the run through the same algorithm to back project them into an image. The results of this operation are shown below.

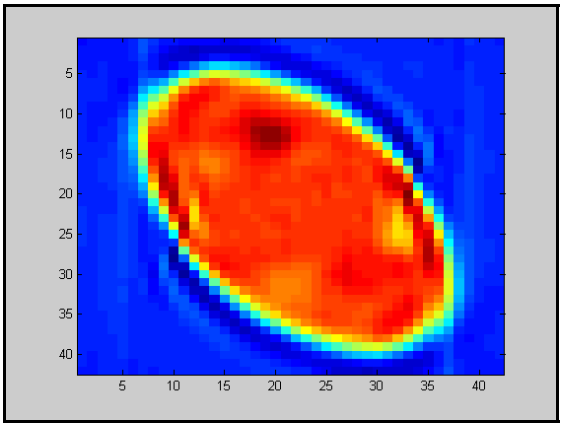


Figure 6. Results of filtered back projection

As you can see, after the filtering many of the small details in the image begin to emerge. However, there is still a resolution problem, and it is hard to distinguish exactly what certain shapes are. This is because the projection data only contained 61 samples per slice. This means that the overall image that is reconstructed could at most be 61/sqrt (2) = 42 pixels in each dimension. With such a small resolution, it is hard to recognize any important features.

To correct the resolution problem proposed above, a method of interpolation was added to fill in the gaps in resolution and approximate a larger version of the reconstructed image. To do this however, the interpolation was not run on the image itself, but rather on the projections. A factor was added into the code to control a re-sampling of the projections by interpolating intermediate points between data points. Once these newly sampled projections were filtered and back projected, the result was a much higher resolution image. The results of this final operation are shown below.

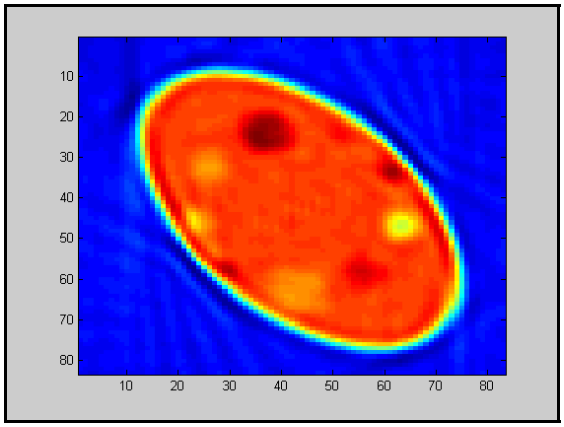


Figure 7. Filtered back projection with twice as many samples

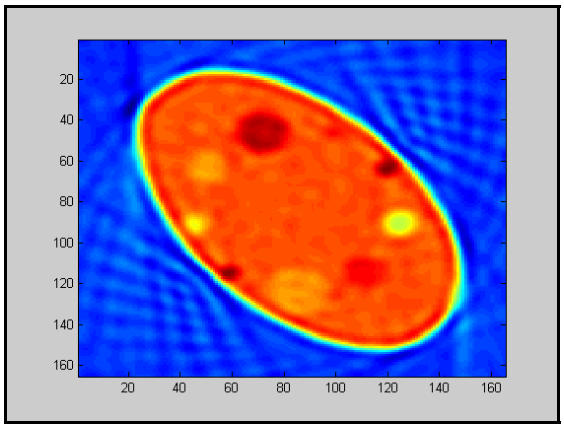


Figure 8. Filtered back projection with 4 times as many samples

An overall flow diagram of this algorithm is shown as figure 9.

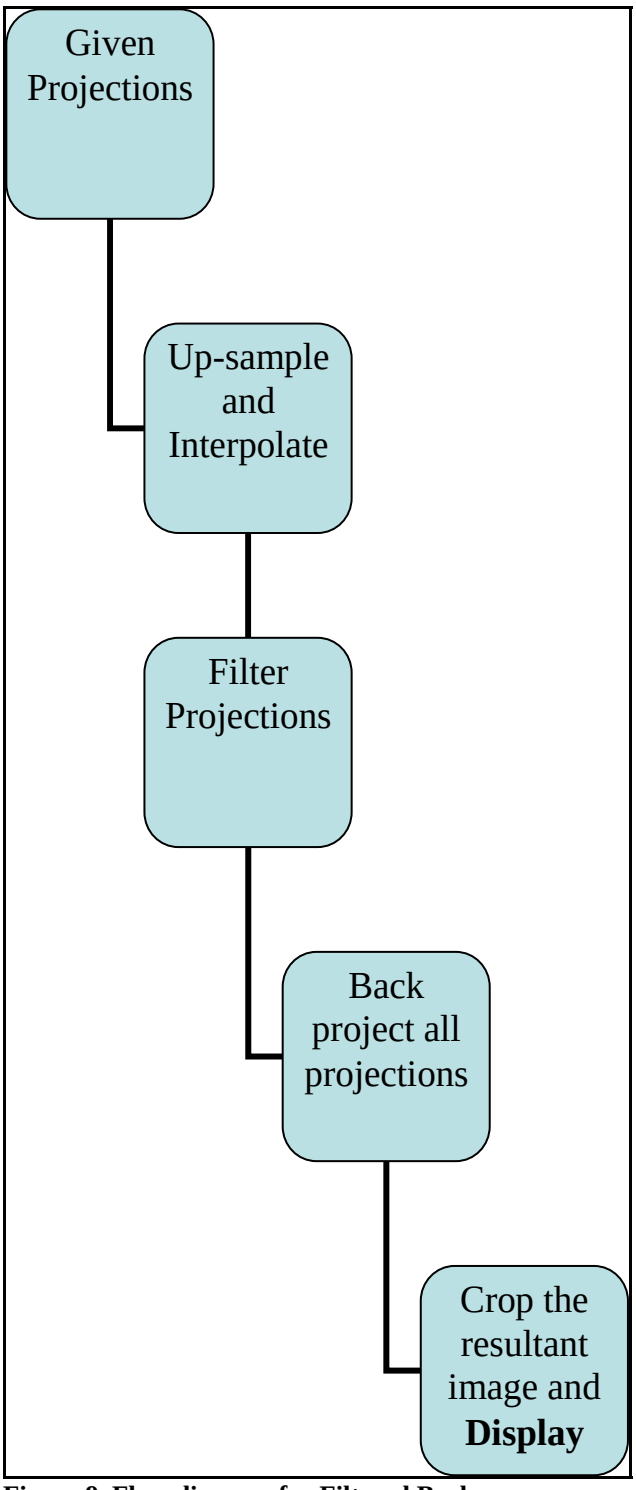


Figure 9. Flow diagram for Filtered Back Projection Algorithm

Iteration:

Another method used for image reconstruction is an iterative method. The implementation behind this method is very similar to the back projection method presented above. In fact, many of the filters used on the projections above are re-used in this method. The basic algorithm works as follows.

First, a temporary dummy image is created with a constant value. Next, an array of projections is calculated for this image. The projections generated for this image should have the same number of samples and the same orientation as the given set. Next, an array of errors is calculated from the difference between the new projections and the original projections. This error matrix should contain the same number of rows and columns as the projection matrices. This error matrix is then back projected onto the dummy image, and the process begins again with this new dummy image until the number of iterations is complete. The result of three iterations is shown below.

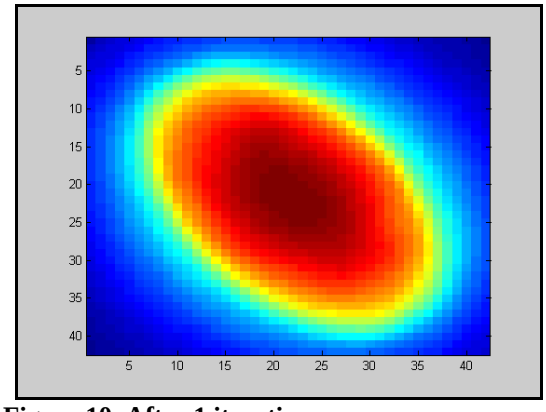


Figure 10. After 1 iteration

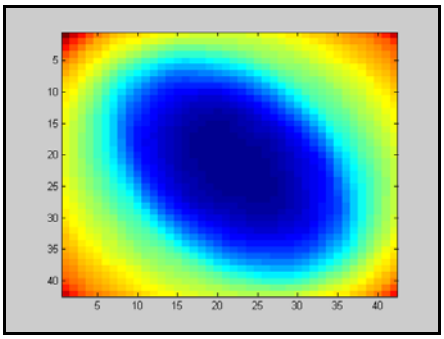


Figure 11. After 2 iterations

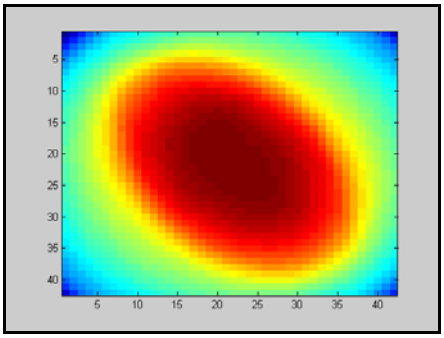


Figure 12. After 3 iterations

As you can see, the same problems that occurred with the last method are reoccurring again with this method. However, with successive iterations the effects get worse. Notice how the boundaries of the shape begin to bleed outward as more and more detail gets lost. This is the effect of the high frequency content being lost increasingly during the iteration cycle.

These errors in the image were corrected using the same frequency filter and up-sampling algorithm described above. Once they were introduced into the code, the results appeared much more satisfactory. Shown below is a succession of iterations using these filters. The up-scaling used for these images is a factor of 4.

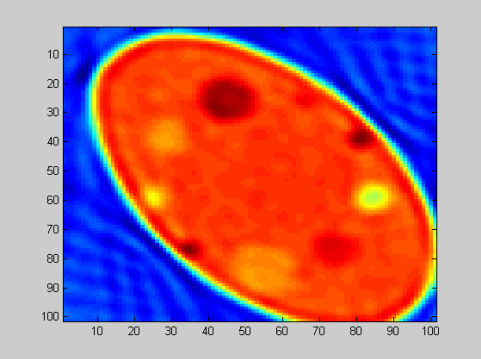


Figure 13. Filtered iteration after 1 projection

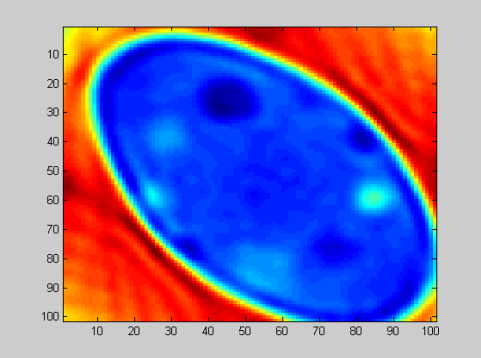


Figure 14. Filtered iteration after 2 projections

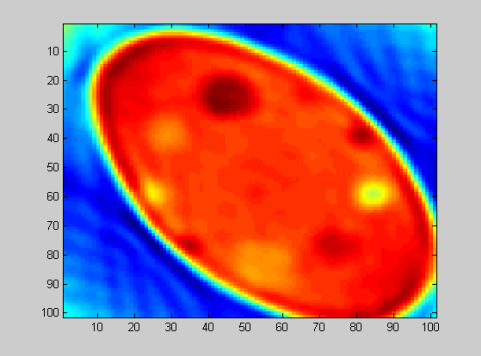


Figure 15. Filtered iteration after 3 projections

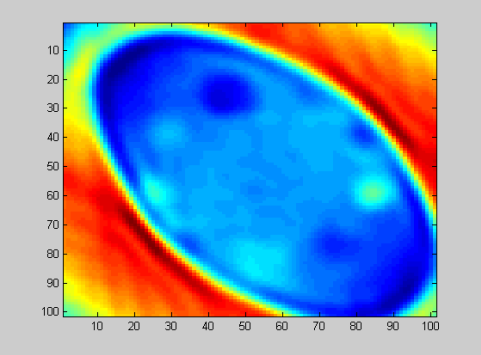


Figure 16. Filtered iteration after 4projections

Included below is a general high level flow diagram of the iterative procedure

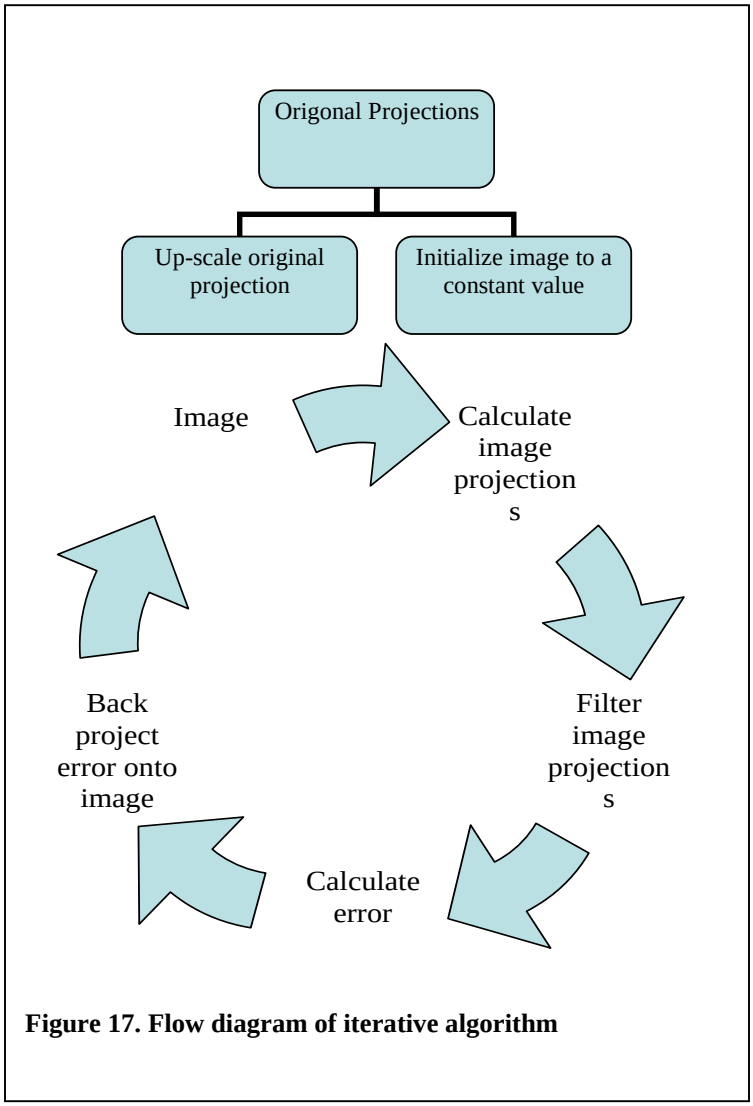


Figure 17. Flow diagram of iterative algorithm

Shape Recognition:

Now that the original image is recovered, the second task is to identify and characterize important features present in the image. Doing this directly from an image produced by non-altered projections is nearly impossible, because of the lack of resolution. In order to identify features, I used an up-sampling factor of 6. The resulting image is shown below.

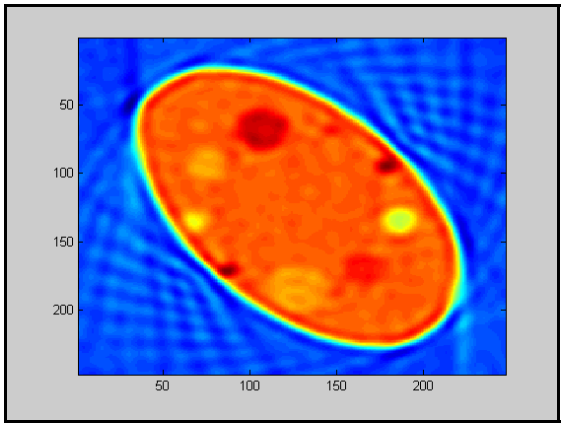


Figure 18. Reconstructed image at 6 times the resolution

However, this interpolation comes with a price. Because the interpolation scheme used in Matlab finds a smooth curve approximation between points, many jagged sharp edges became removed during the scaling operation. This made many shape identifying features go away.

In order to see the shapes better in the image, I used a contour approximation of the image to see the outline of equal valued sections. However, even this was tricky, because the sections that were outlined fluctuated with the number of contours specified. Shown below are a few examples of the different contours grids.

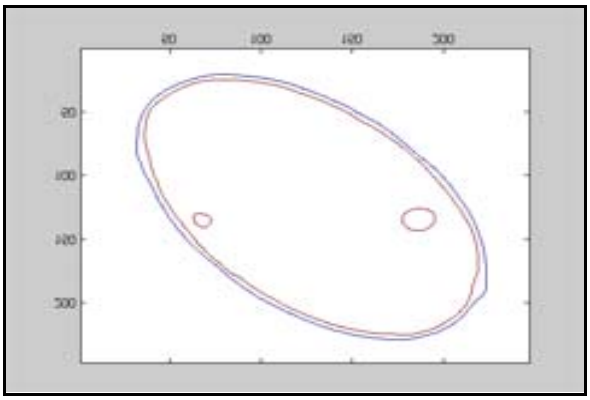


Figure 19. Contour map with two contours

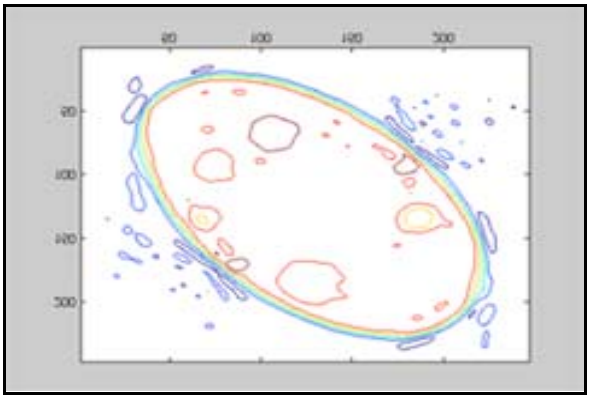


Figure 20. Contour map with seven contours

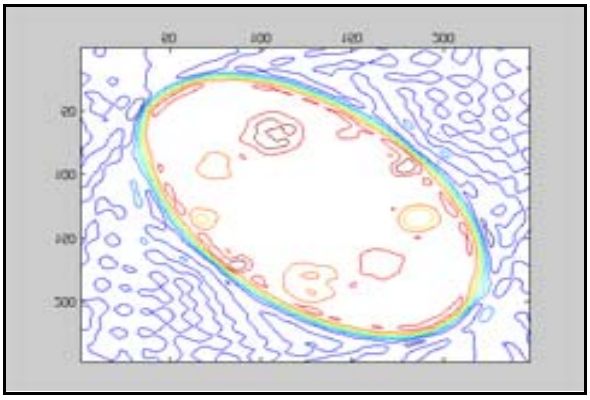


Figure 21. Contour map with ten contours

Some shapes however, continued to re-occur in most of the contour maps, and these are the ones I chose to be the salient important features. The best contour number to show these features turned out

to be four. An image of this contour map is shown below with the labeled shapes.

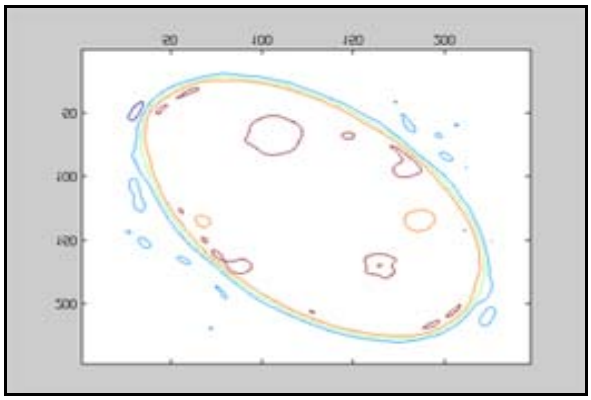


Figure 22. Contour map with four iterations

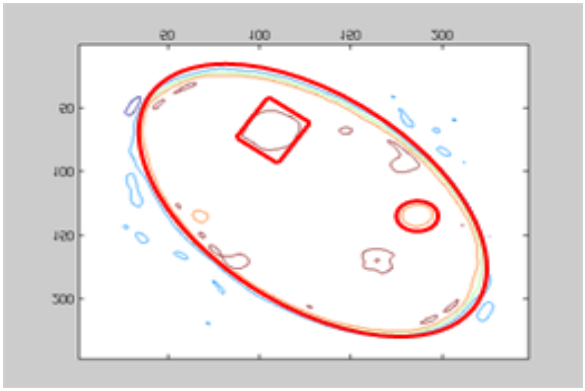


Figure 23. Contour map with labeled shapes

Rectangle

For the rectangle, I measured the three corners to be at grid coordinates

P1= (84, 74)
P2= (108, 90)
P3= (127, 61)

Using these three points, I calculated the lengths of the rectangle in respect to my image to be

L1= 31.38
L2= 34.668 samples.

However, the size of my image spans from 0-247 on each axis, where the original image spanned from -1 to 1. Therefore the overall scaling from samples to real space is 247/2 or 123.5. So in actuality the lengths of the rectangle are

L1' = L1/123.5 = 0.2541
L2' = L2/123.5 = 0.2807

Using the same logic, the center location of the rectangle was found on my image to be the point (107, 68). Scaling this point by 123.5 and subtracting 1 to center the value gives the real center to be at **(-0.1336, -0.4494)** The amplitude at this point is **.1658**.

Ellipse

I assumed by the ellipse, the assignment was referring to the circular shaped object outlined in the lower right hand portion of the image. Using this as my shape, the coordinates defining the major and minor axis are:

(186,143), (186-127)
(177,133), (194, 133)

However, the length of these axis is equal, so the ellipse equation becomes that of a circle with diameter = 16. After rescaling, this diameter becomes 16/123.5 = 0.1296. The center of the circle in the image is (135,186). The actual center of the circle is

Center = (.0931, .5061)

At this point, the equation of the circle is

(x-.0931)^2 +(y-.5061) ^2 <=0.1296

Gaussian Ellipsoid with level curves

Assuming the ellipsoid to be solved for is the outer ring defined by the cell, the procedure for identifying the coefficients is as follows.

First, simply by observation the center of the ellipsoid is located at

Center = (125,125)
This converts to (.0121, .0121)

The equation for an ellipsoid is as follows:

Equation 4. Equation for an ellipse

$$K = a(x - xo)^2 + b(y - yo)^2 + c(x - xo)(y - yo)$$

Where xo and yo are the center of the ellipse. This leaves four unknowns: a, b, c, and K. To solve for these four unknowns, four equations are needed. These equations can be generated by picking four independent points on the ellipse. The four points selected were:

(44, 34)
(211, 204)
(173, 71)
(90, 184)

Using these values four equations were created.

$$K = a(-81)^2 + b(-91)^2 + c(-81)(-91)$$

$$K = a(86)^2 + b(79)^2 + c(86)(79)$$

$$K = a(48)^2 + b(-54)^2 + c(48)(-54)$$

$$K = a(-35)^2 + b(59)^2 + c(-35)(59)$$

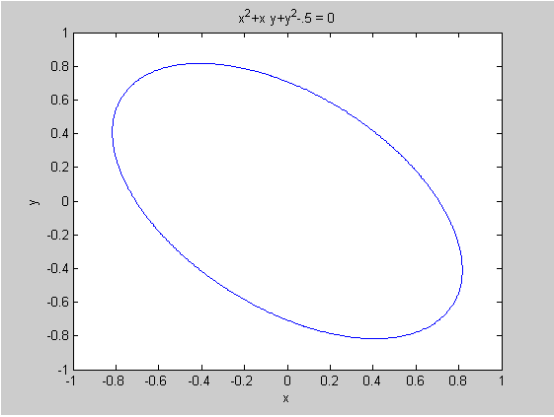
By simultaneously solving these equations the coefficients for the equation of the ellipsoid come out to be

a=1
b=1
c=1.

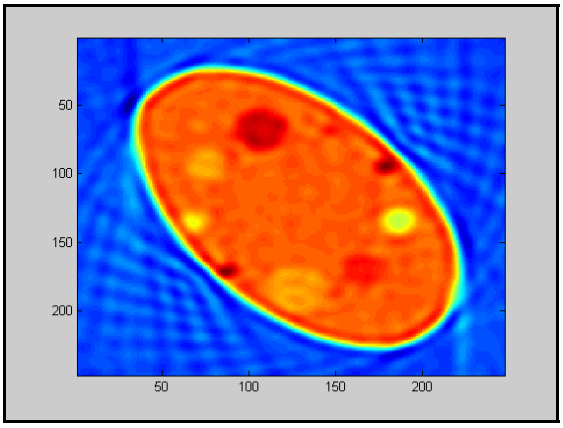
Therefore the overall equation for the ellipsoidal level curves becomes

$$.75 = (x - .0121)^2 + (y - .0121)^2 + (x - .0121)(y - .0121)$$

A plot of this ellipse is shown below:



When compared to the original image:



The equation of the ellipse emulates the image almost perfectly on the scale of the original image.

Conclusion:

The ability to recreate an image from multiple projections is a valuable method. Implementing this technique has saved countless lives over the years since its conception.

In this project I learned how to turn this theory into a functioning program. Had I possessed more time, I may have attempted the third type of reconstruction, and compared the results of all three methods. Although the reconstruction is still vulnerable to noise and blur, the results were satisfactory enough to show proof of concept.

References:

http://www.math.ucdavis.edu/~kouba/CalcOneDIRECTORY/implicitdiffdirectory/ImplicitDiff.html

http://thayer.dartmouth.edu/~bpogue/ENGG167/13%20Image%20reconstruction.pdf

http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0405/HAYDEN/Slice_Reconstruction.html

https://prod.webct.gatech.edu/SCRIPT/83203200508/scripts/serve_home

http://www.owlnet.rice.edu/~elec431/projects96/DSP/backproject3.html

http://www.owlnet.rice.edu/~elec431/projects96/DSP/bpanalysis.html

Appendix A: Filtered Back Projection Code

```
%backproj.m
%Given an array of projections reconstruct the image using
%filtered back projections.

load projdata.mat   %import the given data

factor=6;       %Scaling factor

for j=1:180
    PPupsamp(:,j)=interp(PP(:,j),factor);   %Supersample the given
projections
end                                         %by the factor

PPfilt=projfilter(PPupsamp);            %Filter the upsampled
projections



image=zeros(61*factor,61*factor);       %initialize the new image
                                        %start the image by rotating
the
                                        %new image to the starting
angle
updateimage=imrotate(image,theta(1)-1,'bicubic','crop');

for i=1:180                             %for every projection
    updateimage=imrotate(updateimage,1,'bicubic','crop'); %rotate image
1 degree
    for j=1:61*factor
        updateimage(j,:)=updateimage(j,:)+PPfilt(j,i);  %back project
projections
    end
end
updateimage=updateimage/180;        %normalize the output
image=imrotate(updateimage,-90,'bicubic','crop');   %rotate it from
last angle to
                                                    %zero degrees

                            %Crop image to origonal image size
icrop=image(factor*(61-42)/2:factor*(60-(61-42)/2), factor*(61-
42)/2:factor*(60-(61-42)/2));
```

Appendix B: Filter Function Code

```
%Filter the projections
%input is the projection matrix
%output is ramp filter modified projection matrix

function newproj=projfilter(proj);
%//----------------filters the image----------------------------

[L,C] = size(proj);         %set size bounds for input and output
matrix
w = -pi : (2*pi)/L : pi-(2*pi)/L;   %initialize freqency domain to fit
in bounds


Filt=fftshift(abs(w));      %shift high frequency to middle of spectrum

for i = 1:C,            % apply this change to every projection
    IMG = fft(proj(:,i));
    FiltIMG = IMG.*Filt';
    FIL(:,i) = ifft(FiltIMG);
end

newproj=real(FIL);  %eliminate imagninary componants
```

Appendix C: Iterative Method Driver Program

```
%Anil Rohatgi
%Iteration driver function
%project

load projdata.mat       %load given data

sampf=2;            %scaling factor

itnum=20;           %iteration number

Pold=projfilter(PP);    %Filter the origonal projections


for u=1:180         %upsample the origonal projections
  pup(:,u)=interp(Pold(:,u),sampf);
end

Pold=pup;
image=.25*ones(61*sampf,61*sampf);   %initialize starting image


for i=1:itnum               %till done every iteration

    P=project(image,theta); %Calculate the projections of the image
    Pnew=projfilter(P);     %filter these projections
    error=Pold-Pnew;        %calculate difference from given
projections
    image=backproj(error);  %backproject these errors

                             %Crop image to origonal image size
    icrop=image(sampf*(61-42)/2:sampf*(60-(61-42)/2), sampf*(61-
42)/2:sampf*(60-(61-42)/2));
    figure;imagesc(icrop);   %plot every iteration

end


figure;contour(icrop,3);    %plot contour map
```

Appendix D: Back Projection Function

```
%function to backproject a matrix on an image
%inputs a matrix and ouputs the updated image

function image=backproj(vals)

L=length(vals(:,1));    %Length of projection array
image=zeros(L,L);       %start return image
                        %initialize temp image by rotating it to the
                        %starting orientation
updateimage=imrotate(image,-90,'bicubic','crop');

for i=1:180 %for all projections
    updateimage=imrotate(updateimage,1,'bicubic','crop'); %rotate 1
degree
    for j=1:L           %for every row on image
        updateimage(j,:)=updateimage(j,:)+vals(j,i);    %back project
value
    end
end
updateimage=updateimage/180;        %normalize image
image=imrotate(updateimage,-90,'bicubic','crop'); %rotate it back to
zero
                                                  %degrees and return.
```

Appendix D: Projection Function

```
%Anil Rohatgi
%Projections code
%generate projections from an input image at the angles in theta
%inputs an image and an angle matrix
%outputs projection matrix

function projections = project(image,theta);
for t=1:180 %for all angles
    J = imrotate(image,theta(t),'bicubic','crop'); %rotate the image to
each orientation
    out=zeros(1,length(J(:,1)));  %initialize depth array
    for i=1:length(J(:,1))      %for every row
        for j=1:length(J(1,:))      %for every column
            out(i)=out(i)+J(i,j);   %update projection
        end
    end
    projections(:,t)=out;       %store projection array
end
```