Fan Beam Tomography

Matthew Hall
Oregon State University
hallm@math.orst.edu

August 18th, 1995

1 Introduction

Computerized tomography is concerned with the reconstruction of a function from its line integrals. A thorough introduction to this material can be found in the work of Natterer [3]. The applications of CT in industry and medical imaging are growing, so results in this field are of more than academic interest.

One popular algorithm in tomography is the filtered back-projection algorithm. There are two variations of it which can be characterized by the geometry of the scanning method used. The parallel beam algorithm reconstructs a function from a set of parallel line integrals taken at different angles around the function. The fan beam algorithm reconstructs the function from a set of line integrals spreading out from a single point which is rotated about the object. The parallel case has been well studied, and the fan beam case studied in conjunction with it. The reconstruction algorithm for the fan beam geometry is in essence the same as that for the parallel beam geometry. As is documented in [2], there is a discrepancy in the quality of the images reconstructed with the different scanning geometries, with the fan beam algorithm being markedly poorer in some cases.

This paper details the background necessary to implement a reconstruction algorithm which hopefully avoids some of the errors present in the fan beam reconstruction. We will derive the formulas needed, discretize them

and present algorithms for parallel and fan beam scanning geometries, confirm that the reconstructions differ, point out the difference between the two algorithms, and then consider a method of testing whether this difference is responsible for the poorer quality of the images.

2 The Radon Transform

The Radon transform takes a function f to integrals over hyper-planes. The research described in this paper is confined to $f : \Re^2 \to \Re$, so the Radon transform will take f to its line integrals. If φ is an angle perpendicular to a line, s is defined as the signed distance from the origin to the line, and $\theta = \binom{\cos\varphi}{\sin\varphi}$, $\theta^\perp = \binom{-\sin\varphi}{\cos\varphi}$. The Radon transform R is

$$Rf(\theta, s) = \int_{\Re} f(s\theta + \theta^\perp t)dt$$

Sometimes it is desirable to suppress the dependence of Rf on θ by writing $R_\theta f(s)$.

The most basic result in tomography is the projection slice theorem.

Theorem 2.1 *$\widehat{R_\theta f}(\sigma) = (2\pi)^{\frac{1}{2}}\hat{f}(\sigma\theta)$ where $\hat{f}$ indicates the Fourier transform of f.*

Proof:

$$\widehat{R_\theta f}(\sigma) = (2\pi)^{-\frac{1}{2}} \int_{-\infty}^{\infty} R_\theta f(s) e^{-is\sigma} ds$$

$$= (2\pi)^{-\frac{1}{2}} \int_{\Re}\int_{\Re} f(s\theta + t\theta^\perp) e^{-is\sigma} dt ds.$$

The substitution $x = s\theta + t\theta^\perp$ can be made. Taking the dot product with θ yields $s = x \cdot \theta$. This gives

$$\widehat{R_\theta f}(\sigma) = (2\pi)^{-\frac{1}{2}} \int_{\Re^2} f(x) e^{-i\sigma x\cdot\theta} dx = (2\pi)^{\frac{1}{2}} \hat{f}(\sigma\theta). \square$$

3 The Parallel Beam Algorithm

The filtered back projection algorithm involves filtering the values calculated for the line integrals. The reconstruction is then of $W_b \star f$, where $\star$ denotes the convolution. The point-spread function, W_b is assumed to be of the form $\hat{W}_b(\xi) = (2\pi)^{-1}\hat{\Phi}(|\xi|/b)$ where $0 \leq \hat{\Phi}(\sigma) \leq 1$, $\hat{\Phi}(\sigma) = 0$ when $|\sigma| \geq 1$. The last requirement is such that the convolution b band-limits the function f which is being reconstructed, ie. frequencies in f greater than b are lost.

By the convolution theorem for the Fourier transform

$$W_b \star f(x) = (2\pi)^{-1} \int_{\Re^2} (\widehat{W_b \star f})(\xi) e^{ix\cdot\xi} d\xi$$

$$= (2\pi)^{-1} \int_{\Re^2} (2\pi) \hat{W}_b(\xi) \hat{f}(\xi) e^{ix\cdot\xi} d\xi.$$

Now switch to polar coordinates. $\theta = \binom{\cos(\varphi)}{\sin(\varphi)}$, and $\xi = \sigma\theta$.

$$W_b \star f(x) = \frac{1}{2} \int_0^{2\pi} \int_{\Re} |\sigma| \hat{W}_b(\sigma\theta) \hat{f}(\sigma\theta) e^{i\sigma x\cdot\theta} d\sigma d\varphi.$$

By the projection slice theorem

$$W_b \star f(x) = \int_0^{2\pi} \int_{\Re} \frac{1}{2} (2\pi)^{-\frac{1}{2}} |\sigma| \hat{W}_b(\sigma\theta) \widehat{R_\theta f}(\sigma) e^{i\sigma x\cdot\theta} d\sigma d\varphi.$$

The function $\hat{w}_b$ is defined as $\hat{w}_b(\sigma) = \frac{1}{2}|\sigma|(2\pi)^{-\frac{1}{2}}\hat{W}_b(\sigma\theta)$.

$$W_b \star f(x) = \int_0^{2\pi} \int_{\Re} \hat{w}_b(\sigma) \widehat{R_\theta f}(\sigma) e^{i\sigma x\cdot\theta} d\sigma d\varphi$$

Use again the properties of the convolution and the Fourier transform, this time in one dimension, to result in

$$W_b \star f(x) = \int_0^{2\pi} (2\pi)^{-\frac{1}{2}} \int_{\Re} (\widehat{w_b \star R_\theta f})(\sigma) e^{i\sigma x\cdot\theta} d\sigma d\varphi$$

$$= \int_0^{2\pi} (2\pi)^{-\frac{1}{2}} (2\pi)^{\frac{1}{2}} (w_b \star R_\theta f)(x \cdot \theta) d\varphi$$

$$= \int_0^{2\pi} \int_{\Re} w_b(x \cdot \theta - s) R_\theta f(s) ds d\varphi.$$

It is customary to assume that $f(x) = 0$ when $\|x\|_2 \geq 1$, ie. that $Rf(\theta, s) = 0$ when $|s| > 1$. Note that $Rf(\theta, s) = Rf(-\theta, -s)$, where $-\theta = \binom{-\cos\varphi}{-\sin\varphi}$, so the substitutions $\psi = \varphi + \pi$, $\omega = -\theta = \binom{\cos\psi}{\sin\psi}$ and $t = -s$ yield

$$\int_0^{\pi} \int_{-1}^{1} w_b(x \cdot \theta - s) Rf(\theta, s) ds d\varphi = \int_{\pi}^{2\pi} \int_{-1}^{1} w_b(x \cdot (-\omega) + t) Rf(-\omega, -t) dt d\psi$$

Note $(x \cdot (-\omega)) + t = -(x \cdot \omega - t)$, and $w_b(-\rho) = w_b(\rho)$. The latter property can easily be confirmed by considering the inverse Fourier transform of the original definition of $\hat{w}_b$.

$$\int_0^{\pi} \int_{-1}^{1} w_b(x \cdot \theta - s) Rf(\theta, s) ds d\varphi = \int_{\pi}^{2\pi} \int_{-1}^{1} w_b(x \cdot \omega - t) Rf(\omega, t) dt d\psi$$

$$= \int_0^{\pi} \int_{-1}^{1} w_b(x \cdot \omega - t) Rf(\omega, t) dt d\psi.$$

The above implies

$$W_b \star f(x) = 2 \int_0^{\pi} \int_{-1}^{1} w_b(x \cdot \theta - s) Rf(\theta, s) ds d\varphi$$

3.1 Discretization of the parallel beam formula

The reconstruction algorithm assumes $Rf(\theta, s)$ is known for

$$\varphi_j = \frac{\pi}{p} j, \quad j = 0, .., p-1 \qquad \theta_j = \begin{pmatrix} \cos\varphi_j \\ \sin\varphi_j \end{pmatrix}$$

$$s_l = \frac{1}{q} l, \quad l = -q, .., q.$$

Using the trapezoidal rule

$$W_b \star f(x) \approx 2\frac{\pi}{p} \sum_{j=0}^{p-1} \int_{-1}^{1} w_b(x \cdot \theta_j - s) Rf(\varphi_j, s) ds$$

$$\approx 2\frac{\pi}{p}\frac{1}{q} \sum_{j=0}^{p-1} \sum_{l=-q}^{q} w_b(x \cdot \theta_j - s_l) Rf(\varphi_j, s_l).$$

3.2 The parallel beam reconstruction algorithm

In practice the image is reconstructed on a grid $X = (\frac{n}{M}, \frac{m}{M})$ where $n, m = -M, .., M$. This means the reconstruction formula is evaluated M^2 times. Usually $p \approx q \approx M$, so reconstructing an entire picture takes $O(p^4)$ operations. This is too large to be practical in application. Towards remedying this, the following simplification is made: For a given j we will compute the convolution at equidistant points, ie. we compute

$$\sum_{l=-q}^{q} w_b(Hk - s_l) Rf(\varphi_j, s_l), \quad k = -\frac{1}{H}, .., \frac{1}{H}.$$

Usually $H = \frac{1}{q}$. Then

$$\sum_{l=-q}^{q} w_b(x \cdot \theta_j - s_l) Rf(\varphi_j, s_l)$$

can be obtainted by linear interpolation. The standard implementation of the parallel beam algorithm is then:

for $j = 0$ to $p - 1$
  for $k = -\frac{1}{H}$ to $\frac{1}{H}$
    compute the convolution $Q_k = \sum_{l=-q}^{q} w_b(Hk - s_l) Rf(\varphi_j, s_l)$

for $x \in X$

find k such that $k \leq \frac{x \cdot \theta_j}{H} < k+1$

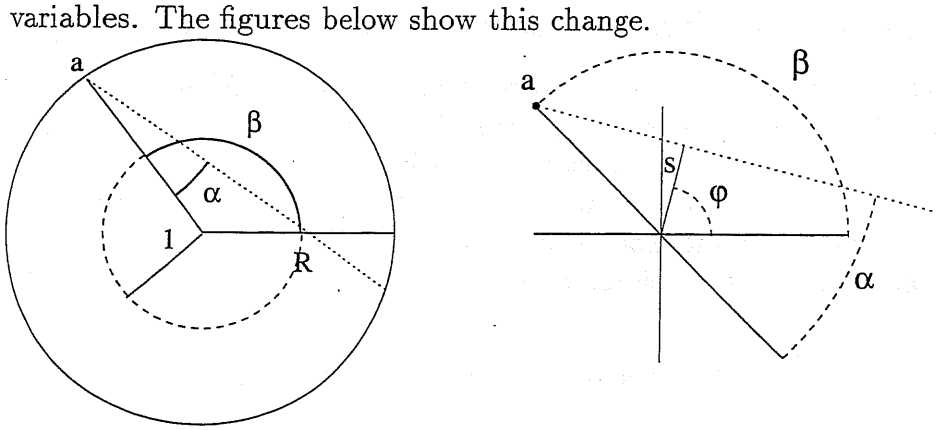
using $u = \frac{x \cdot \theta_j}{H} - k$ compute $Q_x = Q_k(1-u) + Q_{k+1}u$

add $\frac{2\pi}{pq} Q_x$ to the reconstruction of f at the point x.

The function $w_b(s)$ has remained undefined up to this point simply because it has not been necessary to examine any single choice of it. One choice used is $w_b(s) = \frac{b^2}{4\pi^2} u(bs)$ where $u(t) = \frac{\cos(t)-1}{t^2} + \frac{\sin(t)}{t}$ if $t \neq 0$ and $u(0) = \frac{1}{2}$. For more information on other choices for w_b see [3] chapter 5.

4 Derivation of the fan beam algorithm

The fan beam algorithm differs from the above mainly due to a change of variables. The figures below show this change.



The lines along which the Radon transform of f is known are specified by α and β. Compared to the parallel beam case $s = R\sin\alpha$ and $\varphi = \alpha + \beta - \frac{\pi}{2}$. $a = R\binom{\cos\beta}{\sin\beta}$ is the source position. D, the divergent beam transform is defined by

$$Df(\alpha, \beta) = Rf(\alpha + \beta - \frac{\pi}{2}, R\sin\alpha)$$

The formula

$$W_b \star f(x) = \int_0^{2\pi} \int_{-1}^{1} w_b(x \cdot \theta - s) Rf(\theta, s) ds d\varphi$$

is used for the reconstruction.

The change of variables requires the following Jacobian:

$$\frac{d(s,\varphi)}{d(\alpha,\beta)} = \begin{vmatrix} R\cos\alpha & 0 \\ 1 & 1 \end{vmatrix} = R\cos\alpha$$

Thus

$$W_b \star f(x) = R\int_0^{2\pi}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} w_b(x\cdot\theta - s)Df(\alpha,\beta)\cos\alpha d\alpha d\beta.$$

Since w_b is even $w_b(x\cdot\theta - s) = w_b(|x\cdot\theta - s|)$, so it is necessary to express $|x\cdot\theta - s|$ in terms of α and β. $x\cdot\theta = (x-a)\cdot\theta + a\cdot\theta$ where a is our source position. $a = R\binom{\cos\beta}{\sin\beta}$. $\theta = \binom{\cos(\alpha+\beta-\frac{\pi}{2})}{\sin(\alpha+\beta-\frac{\pi}{2})}$. $a\cdot\theta = R(\cos\beta\cos(\alpha+\beta-\frac{\pi}{2}) + \sin\beta\sin(\alpha+\beta-\frac{\pi}{2})) = R\cos(\alpha-\frac{\pi}{2}) = R\sin\alpha = s$. Let γ be the angle between the lines from a to the origin and a to x. Thus $(x-a) = -|x-a|\binom{\cos(\gamma+\beta)}{\sin(\gamma+\beta)}$. $(x-a)\cdot\theta = -|x-a|(\cos(\gamma+\beta)\cos(\alpha+\beta-\frac{\pi}{2}) + \sin(\gamma+\beta)\sin(\alpha+\beta-\frac{\pi}{2})) = -|x-a|\cos(\alpha-\gamma-\frac{\pi}{2}) = -|x-a|\sin(\alpha-\gamma) = |x-a|\sin(\gamma-\alpha)$. Only the magnitude of this is needed, so $|x-a||\sin(\gamma-\alpha)|$ is used.

$$W_b \star f(x) = R\int_0^{2\pi}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} w_b(|x-a||\sin(\gamma-\alpha)|)Df(\alpha,\beta)\cos\alpha d\alpha d\beta.$$

This equation could be discretized, however for the same efficiency reasons given in the parallel beam derivation it is useful to remove the dependence of the inner integral on x.

4.1 Notes on w_b

It is useful to develop some properties of w_b By definition

$$\hat{w}_b(\sigma) = \frac{1}{2}|\sigma|(2\pi)^{-\frac{1}{2}}\hat{W}_b(\sigma\theta) = \frac{1}{2}|\sigma|(2\pi)^{-\frac{3}{2}}\hat{\Phi}(\frac{|\sigma|}{b}).$$

Consider $w_b(as)$, $a \geq 0$.

$$w_b(as) = (2\pi)^{-\frac{1}{2}}\int_{\Re}\hat{w}_b(\sigma)e^{ias\sigma}d\sigma = (2\pi)^{-\frac{1}{2}}\int_{\Re}\frac{1}{2}(2\pi)^{-\frac{3}{2}}|\sigma|\hat{\Phi}(\frac{|\sigma|}{b})e^{ias\sigma}d\sigma.$$

The substitution $\eta = a\sigma$ yields

$$w_b(as) = (2\pi)^{-\frac{1}{2}} \int_{\Re} \frac{1}{2}(2\pi)^{-\frac{3}{2}} \frac{|\eta|}{a} \hat{\Phi}(\frac{|\eta|}{ab}) e^{is\eta} \frac{d\eta}{a} = a^{-2} w_{ab}(s)$$

Thus $w_b(|x-a||\sin(\gamma-\alpha)|) = |x-a|^{-2} w_{b|x-a|}(|\sin(\gamma-\alpha)|)$.

$$W_b \star f(x) = R \int_0^{2\pi} |x-a|^{-2} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} w_{b|x-a|}(|\sin(\gamma-\alpha)|) Df(\alpha,\beta) d\alpha d\beta$$

We have not removed the dependence of the inner integral on x because it is still in $w_{b|x-a|}$ and γ. The values for a given γ can be interpolated from regularly spaced data, which can be done in the discretization. The approximation $c = b|x-a|$ leads to

$$W_b \star f(x) = 2R \int_0^{2\pi} |x-a|^{-2} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} w_c(|\sin(\gamma-\alpha)|) Df(\alpha,\beta) d\alpha d\beta$$

This approximation is important and will be considered in more detail following the discretization of the reconstruction formula.

4.2 Discretization of the fan beam formula

The function Df has been sampled at regular intervals along the β and α axes. Values of α which lead to lines (α, β) that do not intersect the unit disk are irrelevant because f is supported in the unit disk.

$$\beta_j = \frac{2\pi}{p} j, \quad j = 0, .., p.$$

$$\alpha_l = \frac{\omega}{q} l, \quad l = -q, .., q.$$

$\omega = \arcsin(1/R)$. Using the trapezoidal rule we have the following algorithm for the reconstruction:

for $j = 1, ..p$

for $k = -q, ..., q$

compute $Q_k = \sum_{l=-q}^{q} w_c(\sin(\alpha_k - \alpha_l)) Df(\beta_j, \alpha_l) \cos \alpha_l$

for each $x \in X$

compute γ

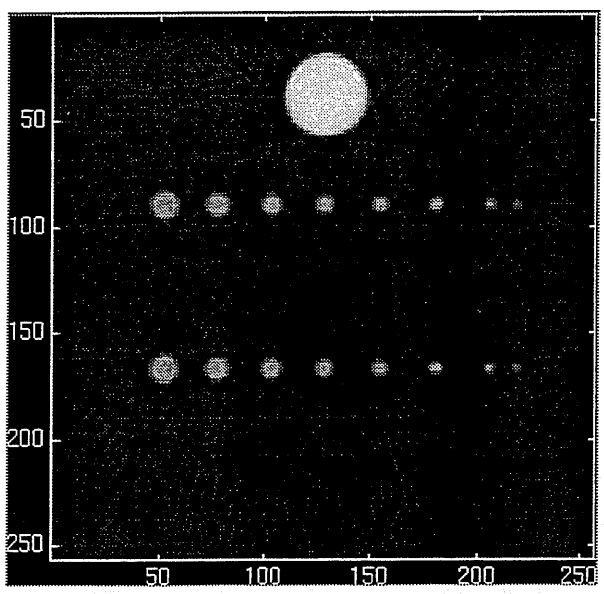
compute k such that $k \leq \frac{\gamma\omega}{q} < k + 1$

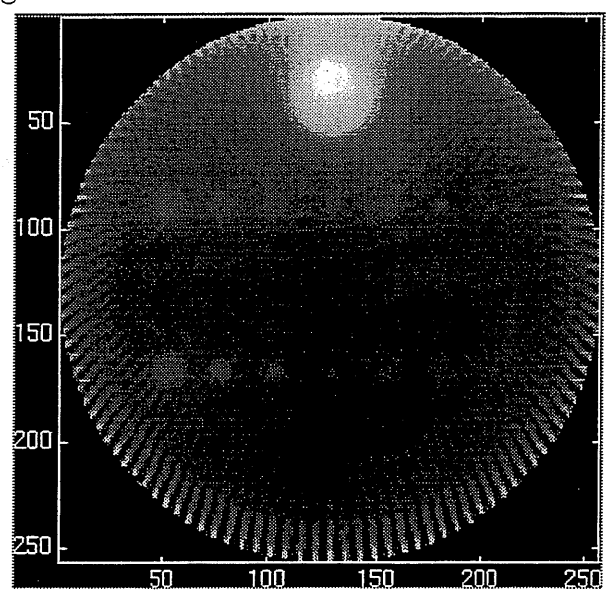
using $u = \frac{\gamma\omega}{q} - k$ compute $Q_x = (1-u)Q_k + uQ_{k+1}$

to the reconstruction of f at x add $|x - a_j|^{-2} \frac{\pi R\omega}{pq} Q_x$

5 Errors in the fan beam reconstruction

The focus of the research this summer was on investigating the discrepancy between the fan beam and parallel beam reconstructions when R is close to 1. This discrepancy was demonstrated in [2]. The figures below illustrate the errors found in the fan beam reconstruction algorithm.





On the left is a reconstruction with the parallel beam algorithm, computed using a program provided by A. Faridani. The figure on the right is a reconstruction using the author's implementation of the fan beam algorithm. The data was calculated with $R = 1.1$. The amount of data collected was the same in each case.

The approximation $c = b|x - a|$ is thought to be the primary cause of the

errors. If $R \approx 1$, and $c = (1+R)b$ as suggested by Natterer [3], chapter 5, the ratio $\frac{c}{|x-a|}$, for some a and x may be quite large. Thus the cut off frequency in w_c may be too large to ensure accurate computation of the convolution. Examination of the fan beam reconstruction above shows artifacts at $x \in X$ such that $\|x\|_2 \approx 1$, which puts these x near a source position.

One solution to this problem which has been tested is detailed in [2] is to reconstruct $W_b \star f$ on a polar grid. This approach suffers the drawback that the polar data must be interpolated to a grid for display. The solution suggested as a project this summer was to compute the convolution for a range of values c_i and interpolate in the calculations depending upon $b|x-a|$. More specifically, we let the $\frac{c_i}{b}$ be evenly spaced about $[(R-1),(R+1)]$, which is the interval of variation of $|x-a|$. Thus for $i = -N,..,N$, $c_i = b(R + \frac{i}{N})$.

The goal this summer was to write an implementation of the standard fan beam reconstruction algorithm and modify it to do this interpolation. The algorithm is

for $j = 1,..p$
  for $i = -N,..,N$
    $c_i = b(R + \frac{i}{N})$
    for $k = -q,..,q$
      compute $Q_{k,i} = \sum_{l=-q}^{q} w_{c_i}(\sin(\alpha_k - \alpha_l))Df(\beta_j, \alpha_l)\cos\alpha_l$
  for each $x \in X$
    compute γ
    compute k such that $k \leq \frac{\gamma\omega}{q} < k+1$
    compute i such that $i \leq N(|x-a| - R) < i+1$
    compute $u = \frac{\gamma\omega}{q} - k$
    compute $v = N(|x-a| - R) - i$
    compute $Q_x = ((1-u)Q_{k,i} + uQ_{k+1,i})(1-v) +$
      $((1-u)Q_{k,i+1} + uQ_{k+1,i+1})v$
    to the reconstruction of f at x add $|x-a_j|^{-2}\frac{\pi R\omega}{pq}Q_x$

This algorithm has not been implemented. The standard fan beam reconstruction program was written (see appendix for a copy of the source). The program was not optimized for speed due to the emphasis of the research this summer being on modifying the algorithm. The modifications were not

begun due to the time which understanding the background material and programming the algorithm required. This work will be continued now that the author is versed in the necessary background information.

Acknowledgments Thanks to A. Faridani for advising and instructing. Thanks to D. Garity for his support in the REU program.

References

[1] A. Faridani, *Reconstructing from efficiently sampled data in parallel-beam computed tomography*, in G.F. Roach (ed.), Inverse problems and imaging, Pitman Research Notes in Mathematics Series, Vol. 245, Longman, 1991, pp. 68-102.

[2] F. Natterer and A. Faridani, *Basic algorithms in tomography*, in: *Signal Processing. Part II: Control Theory and Applications*, F.A. Grünbaum, J.W. Helton, P. Khargonekar (eds.), The IMA Volumes in Mathematics and its Applications, Vol. 23, Springer-Verlag, New York, (1990), pp. 321-333.

[3] F. Natterer, *The Mathematics of Computerized Tomography*, Wiley, 1986.

[4] F. Natterer, *Recent developments in x-ray tomography*, in [5, pp. 177-198].

[5] E.T. Quinto, M. Cheney, and P. Kuchment (eds.), *Tomography, Impedance Imaging, and Integral Geometry*, Lectures in Applied Mathematics, Vol. 30, Amer. Math. Soc., 1994.

```
/*****
        NEW!    comand line specification of input file.
                using any file name with "TEST" in it should
                cause the program to use the values in the
                falseinit() function.  If no file is given it
                will read in the file 'fanconf.ini'
*/

/* compile with :

        gcc fanbeam.c -lm

*/
/*      fanbeam.c
        written by Matthew Hall

 algorithm:
        read input file
        for each source position
                read/generate data
          compute convolution
                compute picture matrix values
        output matrix

 read input file
        data needed:
        P:      number of source positions around the circle
                Q:      2Q-1 = number of detectors
                EPS:    filter to use
                CR:     Cut off frequency
                MX:     output matrix width
                MY:     output matrix height
                XMIN,XMAX:  region to reconstruct
                YMIN,YMAX:
                SIMDAT: boolean, wether to read in data (0) or
                                simulate using ellipses
                NE:     number of ellipses
                ISUP:   highest scaling for density
                rf:     projection data file
                bld:    ascii data file
                mat:    matlab picture file
                pic:    unformatted picture file
                Ellipses:
                        X,Y,HALFAXIS1,HALFAXIS2,ANGLE,DENSITY

 for each source position p <= P
         for each q in [-Q,Q-1]
        reconstruct data
                        compute (T,phi)
                        sum =0
                        for each ellipse
                                sum+= line integral of (T,Phi) through it
        compute convolution
    for each (x,y) in picture matrix
                compute gamma
                compute k to use
                compute u (interpolation factor
                pic(x,y)+= contribution of weight
*/

/******* include libraries ********/

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/****** definitions ********/

#define arccos  acos
#define arcsin  asin

#define MaxQ    256
#define MaxPic  256
#define PI      3.141592653589793
#define TWOPI   6.283185207179586
#define MAXELLIPSE      50
#define ERROR   0.00001

/****** structures ********/

typedef struct  {
        double  x,y,halfaxis1,halfaxis2,angle,density;
} Ellipse;

/****** functions ********/

double  line(Ellipse*   e, double beta, double alpha);
/*      Ellipse* points to the data for the ellipse
        beta is the source position angle
        alpha is the angle through the object being scanned */

void            convolute(double data[2*MaxQ-1],int eps);
void            readinit(char *name);
void            falseinit(void);

int             readdata(double data[2*MaxQ-1]);

void            writedata(void);


/****** global variables *******/

double  Pic[MaxPic][MaxPic];

int     P;      /*      number of source positions around the circle */
int     Q;      /*      2Q-1 = number of detectors */
double  R;      /* radius of outer circle */
int     EPS;    /*      filter to use */
double  CR;     /*      Cut off frequency */
int     MX;     /*      output matrix width  */
int     MY;     /*      output matrix height */
double  XMIN,XMAX;      /*  region to reconstruct */
double  YMIN,YMAX;
int     SIMDAT; /*      boolean, wether to read in data (0) or */
                /*      simulate using ellipses */
int     NE;     /*      number of ellipses */
char    rf[30]; /*      projection data file */
char    bld[30];        /*      ascii data file */
char    mat[30];        /*      matlab picture file */
char    pic[30];        /*      unformatted picture file */

double  projections[MaxQ*2];

Ellipse *ellipses[MAXELLIPSE];

double      lowalpha, highalpha, deltaAlpha;

/****** main *******/

int main(int    argc,   char*   argv[])
{
        /*      angles  */
        double  alpha;
        double  beta;

        /* indicies for loops: */
        int     i,j,k,l;

        double  a1,a2; /* source position components */

        double x,y;     /* picture reconstruction coordinates */
        double  deltax,deltay;
        double  temp;

        int X,Y;        /* picture matrix indicies */

        /* variables used in the reconstruction */
        double  normdiff, dotdiff, gamma, aperp1, aperp2, u;

        int     ERR=0;  /* error flag for the backprojection */

        /********* read in initialization file */
        if (argc>=2) {
                if (strstr(argv[1],"TEST")) {
                        falseinit();
                }
                else {
                        readinit(argv[1]);
                }
        }
        else {
                readinit("fanconf.ini");
        }

        /********* print out initialization info, just to check */
        printf("\nP=%d,  Q=%d,  R=%f\n",P,Q,R);
        printf("\nEPS=%d,  CR=%f\n",EPS,CR);
        printf("\nMX=%d,  MY=%d\n",MX,MY);
        printf("XMIN=%f,  XMAX=%f,  YMIN=%f,  YMAX=%f\n",XMIN,XMAX,YMIN,YMAX);
        printf("SIMDAT=%d,  NE=%d\n",SIMDAT,NE);
        printf("bld=%s\n",bld);
        printf("mat=%s\n",mat);
        printf("pic=%s\n",pic);
        for (i=0;i<NE;i++){
                printf("Ellipse data=%f,%f,%f,%f,%f,%f\n",ellipses[i]->x,ellipse
        }

        /********* compute bounds and increments on alpha */
        lowalpha= -arcsin(1/R);
        highalpha= arcsin(1/R);
        deltaAlpha=(highalpha-lowalpha)/(2*Q);

        printf("lowalpha=%f,  highalpha=%f,  deltaAlpha=%f\n",lowalpha,highalpha

        /********* compute deltax&y ***********/
        deltax = (( XMAX - XMIN ) / MX );
        deltay = (( YMAX - YMIN ) / MY );
        printf("deltax=%f, deltay=%f\n",deltax,deltay);

        /********* clear out picture array */
        memset(Pic,0,sizeof(double)*MaxPic^2);

        /******************  main loop  ************************/
        /******************* starts here ************************/
        for (j=0;j<P;j++) {

                /* tell the user where we are */
                if ( (j % 10) ==0) {
                        printf("angle number %d\n",j);
                }

                /* clear projection/convolution array */
                memset(projections,0,sizeof(projections));

                /* compute current source position */
                beta = ( TWOPI / P )* j;

                /* compute source positions */
                a1=R*cos(beta);
                a2=R*sin(beta);

                if (SIMDAT==1) { /* generate data */

                        /* compute lines */
                        for (k=0;k<2*Q-1;k++) {
                                /* compute alpha for this line */
                                alpha = lowalpha + k * deltaAlpha;
                                /* add up weights of ellipses */
                                for (l=0;l<NE;l++) {
                                        projections[k]+=line(ellipses[l],beta,al
                                }
                        } /* end of computing lines */
                }
                else {
                        printf("Help!  I don't know how to read in the data!");
                } /* end else statement */

                /******** convolute **********/
                convolute(projections,EPS);

                /******** compute weights to the picture matrix */

                for (X=0;X<MX;X++) {
                        x = XMIN + deltax*X;

/*                      printf("X=%d, x=%f\n",X,x);*/

                        for(Y=0;Y<MY;Y++){
                                y=YMIN+deltay*Y;
                           if (fabs(y) < sqrt ( 1 - x*x ) ) {
                                ERR=0;
                                normdiff=(a1-x)*(a1-x)+(a2-y)*(a2-y);
                                dotdiff=R*R-(a1*x+a2*y);

                                temp=dotdiff/(sqrt(normdiff)*R);
                                if(fabs(temp)>=1) {
                                        gamma=0.0;
                                }
                                else {
                                        gamma=arccos(temp);
                                }
                                /* need to determine sign of gamma */

                                aperp1=R*cos(beta+PI/2);
                                aperp2=R*sin(beta+PI/2);
                                if(!(aperp1*x+aperp2*y<=0)) {
                                        gamma*=-1;
                                }

                                if (fabs(gamma) >= highalpha) {
                                        printf("gamma=%f, normdiff=%f, dotdiff=%
```

```
                                return(0);
                        }

                        k=floor(gamma/deltaAlpha)+Q;
                        if (k<0) {
                                printf("!k = %d\n",k);
                                ERR=1;
                        }
                        else if(k>2*Q-1) {
                                printf("!k = %d\n",k);
                                ERR=1;
                        }
                        u=gamma/deltaAlpha+Q-k;

                        if (!(ERR==1)) {
                                Pic[X][Y]+=((1-u)*projections[k]+ u
                                        * projections[k+1])/normdiff;
                        }
                }
            } /* end of Y loop */
        } /* end of X loop */
    } /* end of main loop */

    /* scale the picture down */
    printf("scaling the picture matrix down\n");

    temp=/*deltaAlaph * */ 2 * R * PI / P;

    for (X=0;X<MX;X++) {
            for(Y=0;Y<MY;Y++){
                    Pic[X][Y]*=temp;
            }
    }

    /********* write data to a file *********/
    writedata();
    return(0);


void            convolute(double data[2*MaxQ-1],int eps)
{
/* returns the data convoluted with function W, indicated by eps,
        in the data array passed to it.
        the vectors dealt with are of physical length 2*MaxQ-1, but
        only the first 2Q-1 places are used.

        uses CR, Q, lowAlpha, deltaAlpha
*/

        double  tempdata[2*MaxQ-1];
        double  temp,w,alphak,alphal;
        double  s;
        int k,l;

        for (k=0;k<2*Q-1;k++) {
                temp =0;
                alphak=lowalpha+deltaAlpha*k;
                for (l=0;l<2*Q-1;l++) {
                        alphal=lowalpha+deltaAlpha*l;
                        switch (eps) {
                                case 0: s= (CR)*sin(alphak-alphal);
                                        w=(CR/TWOPI)*(CR/TWOPI);
                                        if (fabs(s)<=ERROR) {
                                                w*=.5;
                                        }
                                        else {
                                                w*=( (cos(s)-1)/(s*s) + sin(s)/s
                                        }

                                        break;
                                case 1: w=0;
                                        break;
                                case 2: w=0;
                                        break;
                        }
                        temp+=data[l] * w * cos(alphal);
                }
                /* copy and multiply by h*/
                tempdata[k]=deltaAlpha * temp;
                /*end of convolution for loop*/
        }
        /* copy convolution to data structure */
        memcpy(data,tempdata,sizeof(double)*(2*Q-1));
}

double  line(Ellipse*   e, double beta, double alpha){
/* computes line integral through an ellipse
        e is the ellipse
        beta and alpha define the line - beta is the source position angle
                alpha is the detector angle */

        double  s, phi;
        double  theta1,theta2;
        double  a,temp;

        s=R*sin(alpha);
        phi=beta+alpha-PI/2;

        theta1=cos(phi);
        theta2=sin(phi);

        a= (e->halfaxis1 * cos(phi-(e->angle))) *
                (e->halfaxis1 * cos(phi-(e->angle))) +
                (e->halfaxis2 * sin(phi-(e->angle))) *
                (e->halfaxis2 * sin(phi-(e->angle)));

        s-=theta1*e->x+theta2*e->y;
        s*=s;

        temp=a-s;

        if (temp<=0) {
                return(0);
        }
        else {
                temp=e->density * 2 * e->halfaxis1 * e->halfaxis2 *
                        sqrt(temp) / a;
                return(temp);
        }
}


/**** stuff to read configuration file *******/

#define MAX_CHARS 81            /* Maximum characters/line in INI file */
#define NEWLINE   "\n"

char*   ini_buffer;

int read_ini(char *ini_name);
char *find_header(char *header_name);
char *get_parameter(char *dest, char *param_name, char *offset);

Ellipse*        parseEllipse(char*      dest){
/* parses a string containing information for an ellipse */

#define NUMBERS "-.1234567890"
#define DELIMITERS      "\\t,"
Ellipse*        tempE;
char    tempc[80];
char* char1,char2;

int     dist;

        tempE=(Ellipse*)malloc(sizeof(Ellipse));

        /* get x coordinate */
        char1=strpbrk(dest,NUMBERS);/* find first number or . in dest */
        dist=strcspn(char1,DELIMITERS);
        /* find the first nonoccurence of and element of NUMBERS */
        memcpy(tempc,char1,dist); /* copy first part to tempc */
        tempc[dist]=0;
        tempE->x=atof(tempc);

        /* get y coordinate*/
        char1=strstr(char1,","); /*find next comma */
        char1=strpbrk(char1,NUMBERS); /* find a number of . in char1 */
        dist=strcspn(char1,DELIMITERS); /* find the first nonoccurence of and el
        memcpy(tempc,char1,dist); /* copy first part to tempc */
        tempc[dist]=0;
        tempE->y=atof(tempc);

        /* get halfaxis1 */
        char1=strstr(char1,","); /* find next comma */
        char1=strpbrk(char1,NUMBERS); /* find a number of . in char1 */
        dist=strcspn(char1,DELIMITERS); /* find the first nonoccurence of and el
        memcpy(tempc,char1,dist); /* copy first part to tempc */
        tempc[dist]=0;
        tempE->halfaxis1=atof(tempc);

      /* get halfaxis2 */
      char1=strstr(char1,","); /*find next comma */
      char1=strpbrk(char1,NUMBERS); /* find a number of . in char1 */
        dist=strcspn(char1,DELIMITERS); /* find the first nonoccurence of and el
      memcpy(tempc,char1,dist); /* copy first part to tempc */
      tempc[dist]=0;
      tempE->halfaxis2=atof(tempc);

      /* get angle */
      char1=strstr(char1,","); /* find next comma */
      char1=strpbrk(char1,NUMBERS); /* find a number of . in char1 */
        dist=strcspn(char1,DELIMITERS); /* find the first nonoccurence of and el
      memcpy(tempc,char1,dist); /* copy first part to tempc */
      tempc[dist]=0;
      tempE->angle=PI* atof(tempc) / 180;

      /* get density */
      char1=strstr(char1,","); /* find next comma */
      char1=strpbrk(char1,NUMBERS); /* find a number of . in char1 */
        dist=strcspn(char1,DELIMITERS); /* find the first nonoccurence of and el
      memcpy(tempc,char1,dist); /* copy first part to tempc */
      tempc[dist]=0;
      tempE->density=atof(tempc);

        return(tempE);
}

int read_ini(char *ini_name)
{
/* This function prepares an *.INI file for processing.  There's no error */
/* checking on the filename you pass to it.  Returns the number of bytes  */
/* read, or NULL on an error condition                                    */

#define SEEK_END        2 /* seek constant */
        long len;
        FILE *handle;
        char temp_buf[MAX_CHARS];

        if ((handle=fopen(ini_name, "rb"))==NULL){      /* Binary mode, no CR/LF
                                                        /* translation          */
                puts("Error opening INI file");
                return(NULL);
        }

        if ((fseek(handle, 0, SEEK_END))!=NULL){
                puts("Error Determining INI file size");
        }

        len=ftell(handle);
        rewind(handle);

        if (len >65534){
                puts("INI file too big");
                return(NULL);
        }

        if ((ini_buffer=(char *)malloc(len + 1))==NULL){
                puts("Out of memory");
                return(NULL);
        }

        fread(ini_buffer, len, 1, handle);
        ini_buffer[len]='\0';

        return (strlen(ini_buffer));
}

char *find_header(char *header_name) {

/* O.K. see my previous message for the SAMPLE.INI file to test this  */
/* program with.  You'll notice it has [Headers] enclosed in brackets. */
/* and always at the beginning of a line.  This routine will locate a  */
/* given header, and return a pointer to it, within the global string  */
/* ini_buffer.  If not found, returns a pointer to NULL                */

        char temp_buf[MAX_CHARS];

        temp_buf[0]=0;
/*      strcpy(temp_buf, NEWLINE);              /* A newline                  */
        strcat(temp_buf, "[");                  /*  and the opening bracket [*/
        strcat(temp_buf, header_name);
        strcat(temp_buf, "]");
        return(strstr(ini_buffer, temp_buf)+strlen(temp_buf));
}

char *get_parameter(char *dest, char *param_name, char *offset)
{
/* It's all downhill from here! (not)  Next, this routine scans through  */
/* the INI file, starting at offset, and looking for param_name followed */
/* immediately by an "=".  If found, a pointer to the string will be     */
/* returned (in the form Value=Parameter, or else a pointer to NULL.     */
```

```
/* I'm trying to change the return string - I don't want Value=Parameter
/*      but rather just Parameter */

        char temp_buf1[MAX_CHARS];
        char temp_buf2[MAX_CHARS];
        char *start_ptr;
        char *end_ptr;
        char *too_far_ptr;

        strcpy(temp_buf1, NEWLINE);
        strcat(temp_buf1, param_name);
        strcat(temp_buf1, "=");

        if((start_ptr=strstr(offset, temp_buf1))==NULL){
                *dest=NULL;
                printf("\nbailing out of finding parameter %s\n",temp_buf1);
                return(dest);
        }

        start_ptr+=sizeof(char)*strlen(temp_buf1);

      /* if we find another section in the file, bail out */
        if((too_far_ptr=strstr(offset+1, "["))==NULL)
                too_far_ptr=offset+strlen(offset);


        if((end_ptr=strstr(start_ptr, NEWLINE))==NULL)
                end_ptr=start_ptr+strlen(start_ptr);

        if (too_far_ptr<start_ptr){
                *dest=NULL;
                return(dest);
        }

        memcpy(temp_buf2, start_ptr, end_ptr-start_ptr);
        temp_buf2[end_ptr-start_ptr]='\0';

        strcpy(dest, temp_buf2);
        return(dest);
}


void    readinit(char*  name)
{

char    tempc1[MAX_CHARS];
int     i;
char *offset;
char dest[MAX_CHARS];
        read_ini(name);

        offset=find_header("Configure");

        get_parameter(dest, "P", offset);
        P=atoi(dest);

        get_parameter(dest, "Q", offset);
        Q=atoi(dest);

        get_parameter(dest, "R", offset);
        R=atof(dest);

        get_parameter(dest, "EPS", offset);
        EPS=atoi(dest);

        get_parameter(dest, "CR", offset);
```

```
        CR=atof(dest);

        get_parameter(dest, "MX", offset);
        MX=atoi(dest);

        get_parameter(dest, "MY", offset);
printf("\ndest=%s\n",dest);
        MY=atoi(dest);
printf("MY=%d",MY);

        get_parameter(dest, "XMIN", offset);
        XMIN=atof(dest);

        get_parameter(dest, "XMAX", offset);
        XMAX=atof(dest);

        get_parameter(dest, "YMIN", offset);
        YMIN=atof(dest);

        get_parameter(dest, "YMAX", offset);
        YMAX=atof(dest);

        get_parameter(dest, "SIMDAT",offset);
        SIMDAT=atoi(dest);

        get_parameter(dest, "NE", offset);
        NE=atoi(dest);

        offset=find_header("Files");

        get_parameter(dest, "rf", offset);
        strcpy(rf,dest);

        get_parameter(dest, "bld", offset);
        strcpy(bld,dest);

        get_parameter(dest, "mat", offset);
        strcpy(mat,dest);

        get_parameter(dest, "pic", offset);
        strcpy(pic,dest);

        offset=find_header("Ellipses");
        for (i=0;i<NE;i++){
        /* build the name Ellipse(i) */
                sprintf(tempc1,"Ellipse%d",i+1);
                get_parameter(dest, tempc1, offset);
                ellipses[i]=parseEllipse(dest);
      }
        free(ini_buffer);

}


/********** write data ***************/
void            writedata(void)
/* writes the global variables MX, MY, and Pic
/*      to the file specified in the global variable mat */

{

        int     i,j;
        FILE    *fp;

        /* open file */
        if(!(fp=fopen(mat,"wr"))==NULL){
```

```
                /* write X&Y */
                fprintf(fp,"%d\n",MX);
                fprintf(fp,"%d\n",MY);
                /* write data */
                for (i=0;i<MX;i++) {
                        for (j=0;j<MY;j++) {
                                fprintf(fp,"%f\n",Pic[i][j]);
                        }
                }
                fclose(fp);
        }
        else {
                /********** error in opening file */
                fprintf(stderr, "Cannot open output file %s",mat);
        }
}

void    falseinit(void)
{
        P=180;
        Q=64;
        R=2.88;
        MX=50;
        MY=50;
        EPS=0;
        CR=564.44;
        XMIN=0.0;
        XMAX=1.0;
        YMIN=0.0;
        YMAX=1.0;
        SIMDAT=1;
        NE=1;
        strcpy(bld,"noname.bld");
        strcpy(mat,"noname.mat");
        strcpy(pic,"noname.pic");
        ellipses[0]=(Ellipse *) malloc(sizeof(Ellipse));

        ellipses[0]->x=.10;
        ellipses[0]->y=.10;
        ellipses[0]->angle=.10;
        ellipses[0]->density=.5;
        ellipses[0]->halfaxis1=.5;
        ellipses[0]->halfaxis2=.5;

}
```