University of California, Merced

Final Project Report

Discrete Image Reconstruction using Parallel Beam Geometry

Authors:
Dan Hu (cs294-bu)
Lasith Adhikari (cs294-bz)
Garnet Vaz (cs294-bv)

Supervisor:
Dr. Phillip Colella

A report submitted in fulfilment of the requirements
for the course of Software Engineering for Scientific Computing
(COMPSCI 294-73)

in the

Electrical Engineering & Computer Sciences,
UC Berkely.

December 2013

Abstract

In the context of the computerized axial tomography (CAT or CT) scan, the discrete image reconstruction using parallel beam geometry is a one of the algorithms to reconstruct the cross-sectional images of the given object. Evenly spaced set of X-ray beams are passed through an object and the intensity of the beams at input and output are measured. The aim is to reconstruct an image of the internal structure of an object from the intensity variations arising from beams passing through heterogeneous material. This will be achieved by solving the filtered back projection formula for different readings from different angles.

Acknowledgement

We would like to express our special thanks of gratitude to Dr. Phillip Colella and his teaching assistance Christopher Chaplin who gave us the golden opportunity to do this wonderful project on scientific computing, which also helped us in doing a lot of research on C++ and much more.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Computerized Axial Tomography

A computerized axial tomography (CAT or CT) scan is one of the most important non-invasive medical imaging techniques, which was developed in the early 1970's by Godfrey Hounsfield and Allen Cormack (Fig: 1.1)[1]. X-ray CT reconstructs a cross-sectional image by computing the attenuation coefficient distribution of an object from projection data, which records the relative number of photons passing through the object.

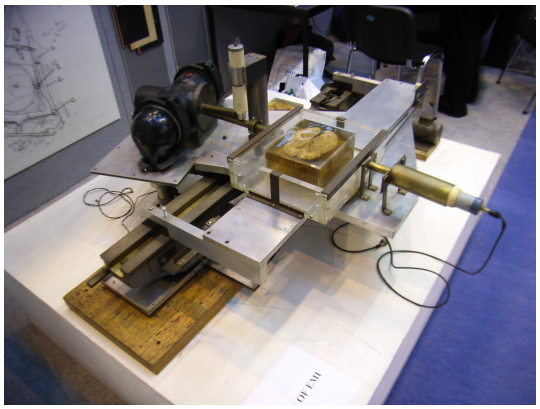

FIGURE 1.1: Hounsfield's prototype CT scanner.

1.2 Objective

For the final project our team has implemented the algorithm for computerized axial tomography (CAT or CT) based on parallel beam geometry. The problem aims to solve an ill-posed inverse problem in medical imaging where X-ray beams are passed through an object and the intensity of the beams at the input and output are measured. The

aim is to reconstruct an image of the internal structure of an object from the intensity variations arising from beams passing through heterogeneous material.

Due to lack of X-ray intensity data, Shepp-Logan phantom (Fig: 1.2) image of the brain is used to test the reconstruction algorithm in this project. When an algorithm is applied to produce a reconstructed image of the phantom, all inaccuracies are due to the algorithm. This makes it possible to compare different algorithms in a meaningful way.

FIGURE 1.2: Image of a Shepp-Logan Phantom used primarily in Tomography reconstruction.

Chapter 2

Theory

2.1 Mathematical Description

In our study of CT-scan, we will consider a 2D slice of the sample and assume which lies and centered on the XY-plane.

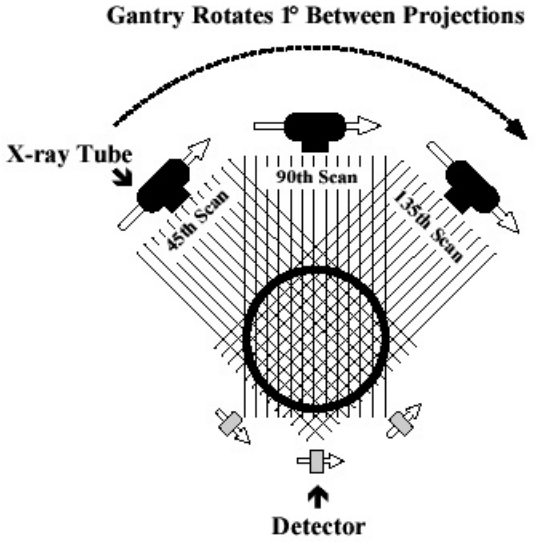

FIGURE 2.1: CT Scanner using parallel beam geometry.

The relationship between measured intensity data (I_0 :Intensity at source end, I_1 :Intensity at detector end) and attenuation-coefficients $f(x, y)$ of the material can be found by the Beer's law [2]. That is given by

$$ln\left(\frac{I_0}{I_1}\right) = \int_{x_0}^{x_1} f(x, y)ds. \tag{2.1}$$

Once we parametrize the above integral along some line $l_{t,\theta}$ is called the *Radon transform* and which is given by

$$\mathscr{R}f(t,\theta) := \int_{l_{t,\theta}} f ds = \int_{s=-\infty}^{\infty} f(tcos(\theta) - ssin(\theta), tsin(\theta) + scos(\theta)) ds. \tag{2.2}$$

Now our fundamental question is to find the function $f(x, y)$ if we know the total accumulation of the attenuation-coefficient function f along every line that passes through the region. Suppose if we select some point in the plane, call it (x_0, y_0), then which lies on many different lines in the plane. The first step in recovering $f(x_0, y_0)$ is to compute the average value of these line integrals averaged over all lines that pass through (x_0, y_0). This integral provides the motivation for the *back-projection transform*. The back-projection of the Radon transform at (x_0, y_0) is given by

$$\mathscr{B}\mathscr{R}f(x_0, y_0) := \frac{1}{\pi} \int_{\theta=0}^{\pi} \mathscr{R}f(x_0 \cos(\theta) + y_0 \sin(\theta), \theta) d\theta \tag{2.3}$$

Hence, the value of the Radon transform along a given line would be the same if all of the matter there were replaced with a homogeneous sample whose attenuation coefficient was the average of the actual sample. Then the integral in equation (2.3) is computing the average value of those averages. Thus, we can see this back-projection does not necessarily produce the exact value of $f(x_0, y_0)$. Instead it gives us an "smoothed-out" version of the attenuation coefficient. In order to solve this problem, it is good to know the interaction between the Fourier transform and the Radon transform that is known as *central slice theorem*.

Theorem 2.1. Central slice theorem. For any suitable f defined in the plane and all real numbers S and θ,

$$\mathscr{F}_2 f(S\cos(\theta), S\sin(\theta)) = \mathscr{F}(\mathscr{R}f)(S, \theta) \tag{2.4}$$

The theorem (2.1), can be used derive new formula called the "*filtered back-projection formula*" to correct the smoothing effect and recover the original function f.

For any suitable function f and any point (x, y) in the plane, by Fourier inversion theorem we can get

$$f(x, y) = \mathscr{F}_2^{-1} \mathscr{F}_2 f(x, y).$$

By applying the definition of 2D inverse Fourier transform,

$$f(x, y) = \frac{1}{4\pi^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathscr{F}_2 f(X, Y) e^{i(xX+yY)} dX dY. \tag{2.5}$$

Now change the variables such that $X = S\cos(\theta)$ and $Y = S\sin(\theta)$ where $0 \leq \theta \leq \pi$ and $S \in \mathbb{R}$. With these changes, equation (2.5) becomes,

$$f(x,y) = \frac{1}{4\pi^2}\int_0^{\pi}\int_{-\infty}^{\infty} \mathscr{F}_2 f(S\cos(\theta), S\sin(\theta)) e^{iS(x\cos(\theta)+y\sin(\theta))}|S| dS d\theta. \tag{2.6}$$

Now with use of the central slice theorem (2.4), we can derive the "*filtered back-projection formula*",

$$f(x,y) = \frac{1}{2}\mathscr{B}\left\{\mathscr{F}^{-1}\left[|S|\mathscr{F}(\mathscr{R}f)(S,\theta)\right]\right\}(x,y). \tag{2.7}$$

The filtered back-projection formula (2.7) is the fundamental basis for image reconstruction. Without the factor of $|S|$ in the formula, the Fourier transform and its inverse would cancel out and the result would be simply the back projection of the Radon transform of f (see equation 2.3), which we know does not lead to recovery of f. Thus, the essential component in the formula is the $|S|$. Therefore we say that the Fourier transform of $\mathscr{R}f$ is filtered by multiplication by $|S|$.

In practice, the above recipe for reconstructing f has a problem. If the $\mathscr{R}f$ has a component at high frequency, then that component is magnified by the factor $|S|$. This leads to exaggerate the noise by the same factor $|S|$ and which corrupts the reconstructed image. Thus, in practice, we replace $|S|$ by low-pass filter A. Then after some algebraic simplifications, function f can be approximated by

$$f(x,y) \approx \frac{1}{2}\mathscr{B}(\mathscr{F}^{-1}A \ * \ \mathscr{R}f)(x,y), \tag{2.8}$$

where, $(*)$ denotes the convolution of $\mathscr{F}^{-1}A$ and $\mathscr{R}f$.

Here are some of the low-pass filters most commonly used in medical imaging.

1. The Ram-Lak filter:

$$A_1(\omega) = |\omega| \cdot \sqcap_L(\omega) = \begin{cases} |\omega| & \text{if } |\omega| \leq L, \\ \\ 0 & \text{if } |\omega| > L. \end{cases}$$

2. The Shepp-Logan filter:

$$\begin{aligned} A_2(\omega) &= |\omega| \cdot \left(\frac{\sin(\pi\omega/(2L))}{\pi\omega/(2L)} \right) \cdot \sqcap_L(\omega) \\ &= \begin{cases} \frac{2L}{\pi} |\sin(\pi\omega/(2L))| & \text{if } |\omega| \leq L, \\ \\ 0 & \text{if } |\omega| > L. \end{cases} \end{aligned}$$

3. The Low-pass Cosine filter:

$$\begin{aligned} A_3(\omega) &= |\omega| \cdot \cos(\pi\omega/(2L)) \cdot \sqcap_L(\omega) \\ &= \begin{cases} |\omega| \cos(\pi\omega/(2L)) & \text{if } |\omega| \leq L, \\ \\ 0 & \text{if } |\omega| > L. \end{cases} \end{aligned}$$

2.2 Discrete Mathematical Model

In the discrete setting, let's implement the filtered back-projection formula (2.8) as

$$f(x,y) \approx \frac{1}{2} \mathscr{B}_D \left(\mathscr{I} \left(\mathscr{F}_D^{-1} A \circledast \mathscr{R}_D f \right) \right)(x,y),$$

where each term is described below.

2.2.1 $\mathscr{R}_D f$ is the discrete Radon transform.

Discrete Radon transform is given by

$$\mathscr{R}_D f_{j,k} = \mathscr{R} f(jd, k\pi/N),$$

for $-M \leq j \leq M$ and $0 \leq k \leq (N-1)$. Suppose that there are $2 \cdot M + 1$ parallel X-ray beams at each angle, N different scans and beam spacing is d .

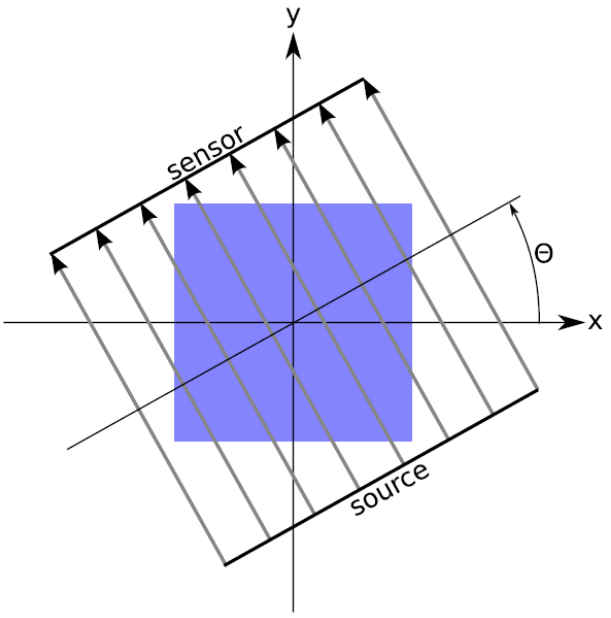

FIGURE 2.2: The Radon transform is a mapping from the Cartesian rectangular coordinates (x, y) to a distance and an angel (t, θ), also known as polar coordinates.

Finding $\mathscr{R}_D f_{j,k}$ can be thought of as computing the projection of the image along the kth angle and jth beam. The resulting projection is the sum of the intensities of the pixels along that line, i.e. a line integral. This is depicted in Figure 2.2 [3].

In order to reduce the number of calculations necessary, the maximum and minimum coordinates x or y are determined for each line (for more details see Appendix A).

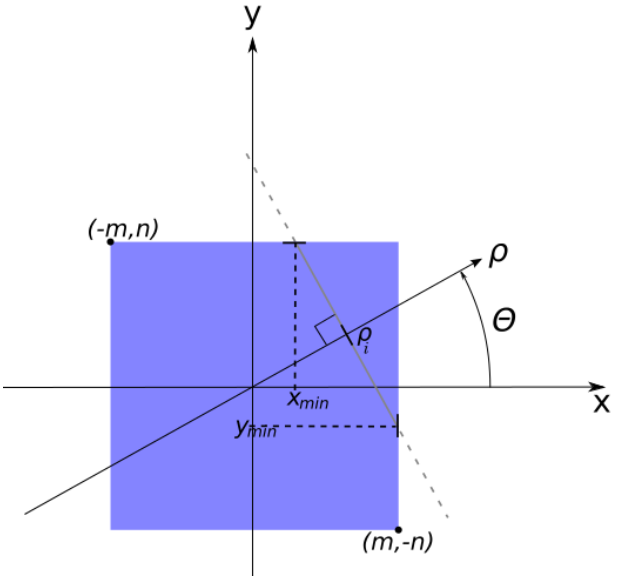

FIGURE 2.3: Determining the maximum and minimum coordinates.

2.2.2 $\mathscr{F}_D^{-1}$ is the discrete inverse Fourier transform.

Samples for $\mathscr{F}_D^{-1}$ can be generated for the Shepp-Logan, Ram-Lak and low-pass cosine filters using Nyquist distance π/L [4].
Samples for discrete inverse Fourier transform of Shepp-Logan filter(A) is given by

$$(\mathscr{F}_D^{-1}A)_n = \frac{4L^2}{\pi^3(1-4n^2)}$$

Samples for discrete inverse Fourier transform of Ram-Lak filter(A) is given by

$$\left(\mathscr{F}^{-1}A\right)_n = \frac{L^2}{2\pi}\left[\frac{2\sin(\pi n)}{\pi n} - \left(\frac{\sin(\frac{\pi n}{2})}{\frac{\pi n}{2}}\right)^2\right].$$

If $n = 0$: $(\mathscr{F}^{-1}A)(0) = L^2/2\pi$
If n is even : $(\mathscr{F}^{-1}A)(\frac{\pi n}{L}) = 0$
If n is odd : $(\mathscr{F}^{-1}A)(\frac{\pi n}{L}) = -2L^2/(\pi^3 n^2)$
Samples for discrete inverse Fourier transform of low-pass cosine filter(A) is given by

$$\left(\mathscr{F}^{-1}A\right)\left(\frac{\pi n}{L}\right) = \frac{2L^2}{\pi^3}\left\{\frac{\pi\cos(\pi n)}{(1-4n^2)} - \frac{2(4n^2+1)}{(1-4n^2)^2}\right\}$$

The band limit L in each above formulas can be found by

$$L = \frac{1}{2d}, \qquad \text{where } d \text{ is the beam spacing.}$$

Fast Fourier transform also can be used to obtain the samples for each filter.

2.2.3 $(\bar{*})$ denotes the discrete convolution.

For two N-periodic discrete functions g and h, the discrete convolution is defined by

$$(g \mathbin{\bar{*}} h)_m = \sum_{j=0}^{N-1} g_j \cdot h_{(m-j)} \text{ for each integer } m.$$

2.2.4 $\mathscr{I}$ is the W-interpolation function.

It is defined for some discrete function g as

$$\mathscr{I}_W(g)(x) = \sum_m g(m)\cdot W\left(\frac{x}{d} - m\right) \qquad \text{for } -\infty < x < \infty.$$

If W weighting function is square wave function,
i.e.

$$\sqcap_{1/2}(x) = \begin{cases} 1 & \text{if } |x| < 1/2, \\ 0 & \text{if } |x| > 1/2. \end{cases}$$

then it is called *nearest neighbor* interpolation.
If W is tent function,
i.e.

$$\bigwedge(x) = \begin{cases} 1 & \text{if } |x| \leq 1, \\ 0 & \text{if } |x| > 1. \end{cases}$$

then it is called *linear* interpolation.

2.2.5 $\mathscr{B}_D$ is the discrete back projection.

For some function h, discrete back projection is given by

$$\mathscr{B}_D h(x,y) = \left(\frac{1}{N}\right) \sum_{k=0}^{N-1} h(x\cos(k\pi/N) + y\sin(k\pi/N), k\pi/N).$$

2.3 Discrete Image Reconstruction Algorithm

By combining the all above discrete models together, we can derive the discrete image reconstruction algorithm in abstract manner [2].
In the reconstruction grid, we approximate the gray scale intensity at each lattice point (x_m, y_n) by

$$\begin{aligned} f(x_m,y_n) &\approx \left(\frac{1}{2}\right) \mathscr{B}_D \left(\underbrace{\mathscr{I}\left(\mathscr{F}_D^{-1} A \bar{*} \mathscr{R}_D f\right)}_{\mathscr{I}_0}(t, k\pi/N) \right)(x_m, y_n) \qquad (2.9) \\ &= \left(\frac{1}{2N}\right) \sum_{k=0}^{N-1} \mathscr{I}_0 \left(x_m \cos\left(\frac{k\pi}{N}\right) + y_n \sin\left(\frac{k\pi}{N}\right), \frac{k\pi}{N} \right). \end{aligned}$$

Chapter 3

Design and Implementation

3.1 Architectural Design

Class diagram of this software design is given in the figure 3.1.

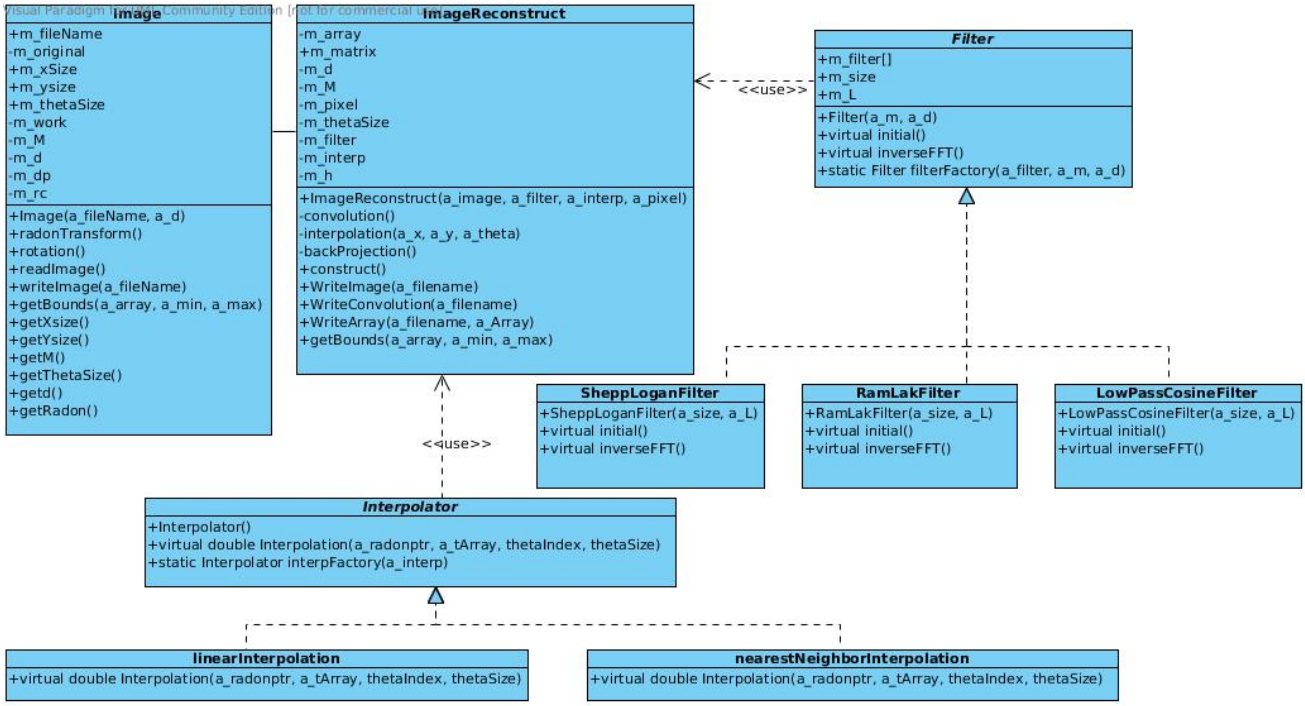

FIGURE 3.1: Class diagram.

3.2 Implementation Approach

C++ programming language in UBUNTU 12.04 platform with g++ 4.6 compiler was used to accomplish this software product. Factory design pattern was used to implement

different interpolators and filters that derived from the base class. The idea of the factory pattern is to have the classes ability to create other classes without them knowing about all the possible classes that exists.

3.3 A Top Level Design of the Software

Image reconstruction algorithm (2.9) is implemented using two main classes, i.e. *Image* and *ImageReconstruct* and another few minor classes for filters and interpolations. A top level design of the software used to solve this problem in the form of header files is given below.

3.3.1 Image Class

Main task of this class is to read the phantom image and compute the corresponding Radon transform. The constructor takes the file name of the image in order to read it into a RectMDArray m_original. The double number a_d is stored in the member data m_d which gives the beam spacing for radon transformation. The results of the radon transformation will be updated into the member data m_work with the type of RectMDArray which will be used in the ImageReconstruct class. This class also provides a method to write out the result of Radon transformation for testing purpose.

```
class Image
{
public:
  Image(std::string a_fileName, double a_d);
  void radonTransform();
  void WriteImage(std::string a_fileName);
  void getBounds(RectMDArray<double>& a_array,double & a_min, double & a_max);
  int getXsize() {return m_xsize;}
  int getYsize() {return m_ysize;}
  int getM() {return m_M;}
  int getThetaSize() {return m_thetaSize;}
  double getd() {return m_d;}
  double gethalfLen() {return m_rc;}
  RectMDArray<double> getRadon() {return m_work;}
  std::string m_fileName; // Record the name of phantom image file.
 private:
```

```
  RectMDArray<double> m_original;
  RectMDArray<double> m_work;
  int m_xsize; // Record the size in the x direction of the matrix data.
  int m_ysize; // Record the sizx in the y direction of the matrix data.
  int m_thetaSize;// Record the number of rotations for the image.
  int m_M; // Record the half number of beams except zero position beam.
  double m_d; // Record the beam spacing in units
  double m_dp; // Record the beam spacing in pixels
  double m_rc; // Record the center of the image along diagonal
};
```

3.3.2 ImageReconstruct Class

This class is the core of the design, which provides the major functionalities such as convolution and back projection. The member data m_array is used to store the result of radon transformation from the Image class and after the call of convolution it will be updated to store the results of convolution. The constructor will take Image object as an input and also two integer which indicates the choices of the specific filter and interpolator. Based on the choice given by the user, the constructor will call the two static factory function of the Filter class and Interpolator class to generate the pointer to the specific derived class. The destructor of ImageReconstruct class is used to delete the filter and Interpolator pointer since these two are generated by new. Inverse FFT of the filter class are invoked inside the convolution function while interpolation functions are called inside the back projection function.

```
class ImageReconstruct
{
public:
  RectMDArray<double> m_matrix;
  ImageReconstruct(Image & a_image, int a_filter, int a_interp, int a_pixel);
  ~ImageReconstruct();
  void construct();
  void WriteImage(std::string a_fileName);
  void WriteConvolution(std::string a_fileName);
  void WriteArray(std::string a_fileName, RectMDArray<double> a_Array);
  void getBounds(RectMDArray<double>& a_array,double & a_min, double & a_max);

private:
  void convolution(); // find the filtered radon transform
```

```
  void backProjection();
  RectMDArray<double> m_array;
  double m_d; // separation between neighbor beams
  int m_M;
  int m_thetaSize;
  int m_pixel;
  double m_h; // grid spacing in final grid
  Filter * m_filterptr;
  Interpolator * m_interppptr;
};
```

3.3.3 Interpolator Class

This class is a base class which has two virtual functions that are implemented inside the derived classes shown below. In order to let user implement different Interpolation in the image reconstrcution class a factory function method is used for this base class. The factory function takes one integer indicating a specific Interpolator to create a pointer to the derived class and it returns a pointer to base class.

```
class Interpolator
{
public:
  Interpolator(){};
  virtual ~Interpolator(){};
  virtual double Interpolation(double * a_radonptr, vector<double> & a_tArray,
      double a_t, int thetaIndex, int thetaSize) const =0;
  static Interpolator * interpFactory(int a_interp);
};
```

LinearInterpolator Class. This is a one of the derived classes of the Interpolator class, which provides the linear interpolation functionality by implementing the inherited virtual Interpolation function.

```
class linearInterpolator:public Interpolator
{
public:
  double Interpolation(double * a_radonptr, vector<double> & a_tArray,
  double a_t, int thetaIndex, int thetaSize) const;
};
```

NearestNeighborInterpolator Class. This is the other derived class of the Interpolator class, which provides the nearest neighbor interpolation functionality by implementing the inherited virtual Interpolation function.

```
class nearestNeighborInterpolator: public Interpolator
{
public:
  double Interpolation(double * a_radonptr, vector<double> & a_tArray, double a_t, in
};
```

3.3.4 Filter Class

This class is a base class which has three virtual functions that are implemented inside the derived classes shown below. The factory function filterFactory is also used here.

```
class Filter
{
public:
  Filter(int a_m, double a_d){ m_size=2*a_m; m_L = 1.0/(2.*a_d); m_filter.resize(2*m_
  virtual ~Filter(){};
  virtual void inverseFFT()=0;
  virtual void initial() = 0;
  static Filter* filterFactory(int a_filter, int a_m, double a_d);
  vector<double> m_filter;
  int m_size; // the size of filter = 2*m_size +1 = 4*a_size +1
  int m_L;
};
```

SheppLoganFilter Class. This is a one of the derived classes of the Filter class, which provides the Shepp-Logan filter functionality by implementing the inherited virtual inversFFT function.

```
class SheppLoganFilter: public Filter
{
public:
  SheppLoganFilter(int a_size, double a_L): Filter(a_size, a_L) { };
  void inverseFFT(){};
  void initial(){};
};
```

RamLakFilter Class. This is a another derived class of the Filter class, which provides the Ram-Lak filter functionality by implementing the inherited virtual inversFFT function.

```
class RamLakFilter: public Filter
{
public:
  RamLakFilter(int a_size, double a_L): Filter(a_size, a_L) { };
 void inverseFFT(){};
 void initial(){};
};
```

LowPassCosineFilter Class. This is also a derived classes of the Filter class, which provides the low-pass cosine filter functionality by implementing the inherited virtual inversFFT function.

```
class LowPassCosineFilter: public Filter
{
public:
  LowPassCosineFilter(int a_size, double a_d): Filter(a_size, a_d) { };
  void inverseFFT(){};
  void initial(){};
};
```

Chapter 4

Results

4.1 System Inputs

Using the Figure 4.1 as the input phantom image, all the resulted reconstructed images were obtained with the image reconstruction algorithm as described in the chapter 2.

FIGURE 4.1: Shepp-Logan Phantom Image.

This software allows user to select the size of the image to be constructed in the run time. For a instance,

```
Please input the image file number to be constructed:
 1 - Shepp_Logan image with size 256 x 256
 2 - Shepp_Logan image with size 200 x 200
 3 - Shepp_Logan image with size 100 x 100
Your choice: 1
```

As a system constrained, we always use square, even size gray-scale PGM ASCII image format.

Then the user has to input the required X-ray beam spacing that will be used to perform the radon transform. As a user guidance, this software system will provide minimum

beam spacing if the user really wants to use the maximum number of beams for the reconstruction process.

As the next inputs, user has to select the type of the filter and the interpolation method that he/she wants to apply on the image. For a instance,

```
Please input the type of the Filter:
 1 : Shepp-Logan filter
 2 : Ram-Lak filter
 3 : Low-pass cosine filter
Your choice: 1

Please input the type of the Interporlator:
 1 : Linear Interpolator
 2 : Nearest neighbor Interpolator
Your choice: 1
```

After that the system will generate and store the corresponding reconstructed image along with the Radon transform image and the convolution image inside the main folder. User can compare the output image with the original image stored inside the Phantom folder.

4.2 System Outputs

According to the above input parameters, we can see that there are different possible combinations that user can use while executing the system. But here we are going to consider only few important test cases.

4.2.1 Test case 1:

- Image size: 256×256
- Beam spacing: 0.005 (i.e 363 beams)
- Filter: Shepp-Logan
- Interpolator: Linear

Radon Transformation The output of the radon transformation for the test case 1 is shown in the Figure 4.2.

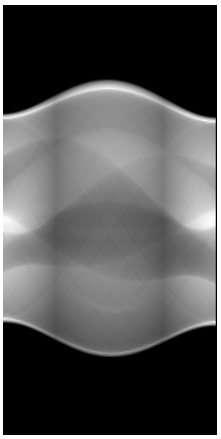

FIGURE 4.2: Radon transform of test case 1.

Convolution The output of the convolution (i.e. filtered Radon transformation) for the test case 1 is shown in the Figure 4.3.

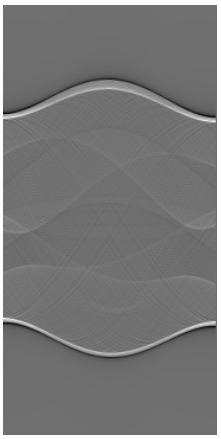

FIGURE 4.3: Convolution of test case 1.

Back projection Transformation Corresponding reconstructed image is given by the Figure 4.4.

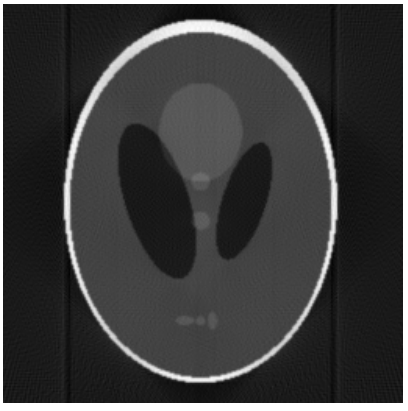

FIGURE 4.4: Test case1: Reconstructed phantom.

The error related to the above reconstruction is given by the Figure 4.5.

FIGURE 4.5: Error of test case 1.

According to the error image 4.5, we can conclude that the most of intensity errors occurred only near the edges of the phantom image.

4.2.2 Test case 2:

- Image size: 256×256
- Beam spacing: 0.02 (i.e 121 beams)
- Filter: Shepp-Logan
- Interpolator: Linear

Corresponding reconstructed image is given by the Figure 4.6.

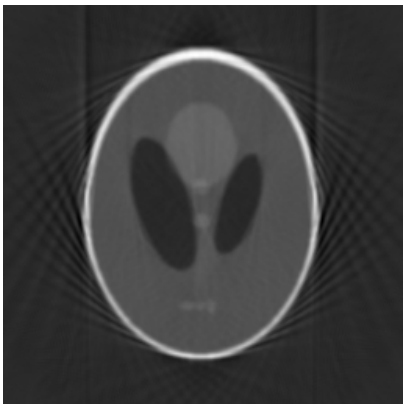

FIGURE 4.6: Test case2: Reconstructed phantom.

According to the output image 4.6, we can see that the image become little blurred and distorted with respect to the Figure 4.4. This is because, for test case 2, we only used

one third of beams compared to the test case 1. But this process is much faster than test case 1 process.

Similar to this test cases, user can reconstruct image using different interpolators and filters provide by this system.

4.3 Performance Analysis

From our experience the linear interpolator with the Shepp-logan filter or Low pass cosine filter will give the best result of the image reconstruction. Therefore in the following we will analyze the result using the linear interpolator and Shepp-logan filter setting for the code.

[1] main.cpp		time = 14.1911	
subroutine	Time Percentage	total time	loops
[6] Radon Transformation	0.9%	0.1225	1
[2] Image Reconstruction	70.1%	9.9471	1
[2] Image Reconstuction		time = 9.9471	
subroutine	Time Percentage	total time	loops
[3] Back Projection	99.8%	9.9236	1
[7] Convolution	0.2%	0.0235	1
[3] Back Projection		time = 14.0130	
subroutine	Time Percentage	total time	loops
[4] Summation over θ	96.2%	9.5455	65536
[8] Linear Index	0.0%	0.001	65536
[4] Summation over θ		time = 9.5455	
subroutine	Time Percentage	total time	loops
[5] Linear Interpolation	92.1%	8.7874	11796480
[8] Calculation of t	6.571%	0.6270	11796480
[5] Linear Interpolation		time = 8.7874	
subroutine	Time Percentage	total time*	loops
[5] tent	30.4%	2.671	4282122240
[8] Linear Index	45.3%	3.984	4282122240

TABLE 4.1: Time table for main.exe for image of 256 x 256 and the beam spacing is chosen as d=0.007. Shepp-logan filter and linear interpolator are used in this run. Notice that in the subroutine of Linear Interpolator only the precentage is measured by Timer. The time inside this subroutine is calculated from the precentage of total time.

The results from timer are in shown in the time Table.4.1. From this figure it is easy to notice that most of the time is cost in the back projection part. Because there are three loops inside the back projection and the interpolation is called 11796480 times in order to do the back projection for output figure with 256x256 pixels. Most of the time are cost inside these loops. In order to optimize the code we already used the linear index to access the element of all the RectMDArray objects. It is shown in the table that linear index has the same order of time consume with other flop operation. All function asr actually made in line since we used -o3 option in the compiler and we inline all possible member functions inside interpolator class. Therefore the only way to optimize the code is to parallelize the back projection because the back projection are independent with each other and the loops inside it is most time consuming.

The timer class has a bug in it which it could allow use the timer inside another timer and by including it inside the inner loops of interpolator class it really slow down the code into 100 seconds. Because there are too many loops here timer in the most inside loop will definitely slow down the code. Therefore in the most inner subroutine we use the timer to calculate the percentage of each function but calculate the real time by multiply the percentage with the total time of the parent subroutine.

Appendix A

Calculating Coordinates for Line Integrals of Radon Transform

These minimum and maximum coordinates are calculated depends on in which of the four areas (see Figure A.1) θ resides. Because when the summation line has an absolute inclination of more than 1 will cause some pixels to be skipped.

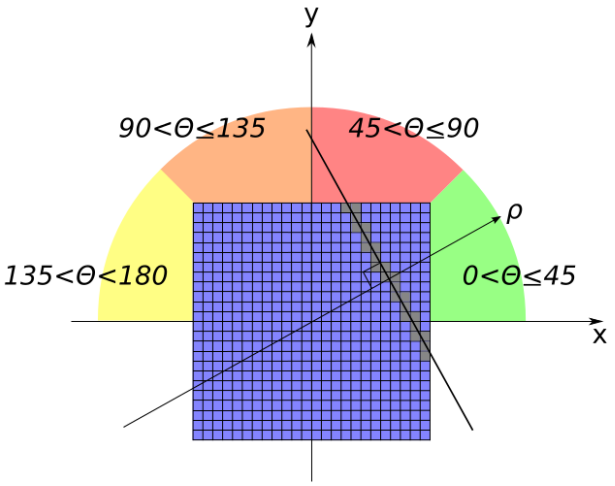

FIGURE A.1: Determining the maximum and minimum coordinates based on θ region.

The formulas used to calculate the minimum and maximum of the variable depending on the angle θ are given below. Where m is half the width of the image and n is half the height of the image, a is inclination, b is the intersection with the y-axis and the t_{max} is the size of the diagonal of the image , i.e. $t_{max}\lceil\sqrt{(2m)^2+(2n)^2}\rceil$.

$$
\begin{aligned}
0 < \theta \leq 45: \qquad x &= \frac{y-b}{a} \\
y_{min} &= max(-n, am+b) \\
y_{max} &= min(n, -am+b)
\end{aligned}
$$

$$
\begin{aligned}
45 < \theta \leq 90: \qquad y &= ax + b \\
y_{min} &= max(-m, \frac{n-b}{a}) \\
y_{max} &= min(m, \frac{-n-b}{a})
\end{aligned}
$$

$$
\begin{aligned}
90 < \theta \leq 135: \qquad y &= ax + b \\
y_{min} &= max(-m, \frac{-n-b}{a}) \\
y_{max} &= min(m, \frac{n-b}{a})
\end{aligned}
$$

$$
\begin{aligned}
135 < \theta \leq 180: \qquad x &= \frac{y-b}{a} \\
y_{min} &= max(-n, -am + b) \\
y_{max} &= min(n, am + b)
\end{aligned}
$$

$$
\begin{aligned}
\theta = 180: \qquad t &= x + \lceil \frac{t_{max} - 2m}{2} \rceil \\
y &= [-m, m]
\end{aligned}
$$

Bibliography

[1] Prototype model of the ct scanner, 1998-2009. URL http://www.impactscan.org/CThistory.htm.

[2] T.G. Feeman. *The Mathematics of Medical Imaging: A Beginner's Guide*. Springer Undergraduate Texts in Mathematics and Technology. Springer, 2009. ISBN 9780387927114. URL http://books.google.com/books?id=8uQwQv2wSBMC.

[3] C. Høilund. *The Radon Transform*. Aalborg University, VGIS, 07gr721. 2007.

[4] C.L. Epstein. *Introduction to the Mathematics of Medical Imaging*. Pearson Education, Inc, 2003. ISBN 0130675482. URL books.google.com/books?isbn=089871642X.