The Salzburg NTN-Method for the Radon Transform: Introduction and Description of the Radon Software Tool

Erika Hausenblas
Research Institute for Softwaretechnology
University of Salzburg

December 1994

Abstract

The radon transform arise in natural way, if X-rays traverses a body, like in a computer tomograph. Thus, to find out the internal structur of a body in computer tomography corresponds to reconstructions a function from their radon transform. This report deals with a parallized numerical reconstruction algorithm, in particular the filtered backprojection. The filtered backprojection uses high dimensional integration. This integration is done by an equidistant grid and good lattice points. The results are discussed and presented. Further the corresponding program is described.
First we pointed out the mathematical background of the radon transform. The second part deals with the program package we have implemented in C++. The intention of the program was to give the user a software tool for investigate backprojection methods by himself. Thus the program package is written as a C++ library. First, there is an instruction how to use the library. Then we give a list of the objects and an example. Further the paper treats the parallelized version and the speed up of the main functions. The softwaretool is aviable on the World Wide Web, address: troi.coma.sbg.ac.at. The last chapter presents some numerical results.

Contents

1 Introduction

The computerized tomography is a technique to reconstruct internal structurs of a body from data collected outside the body. This means, computer tomography provides a very good method to investigate the internal structur of an object without destroying it. In particular this method is very important in medicine. Theoretically it is possible to reconstruct the internal structur without errors. There is only one problem. If the values of the radon transform have been obtained by physical measurement, one cannot assume, that the data are complete. Therefore it is very important to find algorithm which yields to good approximation of the original object in spite of discrete or incomplete datas.

This paper deals with a certain reconstruction algorithm, the filtered backprojection. To reconstruct a point, a filter function is convoluted by the radon transform. Thus, the result is determined by numerical integration. The common algorithm uses equidistant points for the integration. But, instead of equidistant points, one can use good lattice points explored by Hlawka and Koksma. In this paper we test a new filter function combined with numerical integration based on good lattice points and compare the result with other algorithms. Further, the executing time of the numerical implementation can be improved by parallelizing. This is done by using **PVM**.

The paper can be divided in the following sections: First we explore the radon transform and the algorithm we deal with. Since it is not possible to give a complete introduction, we only give a motivation. In the next section we present the software tool written for research and for testing the new algorithm. The program is written in C++ and is built up like a C++ library. The aim was to give the user a tool for research in the radon transform. With the library the user can outline a program by himself with only a few lines. First we describe the structure of such a program, then we give an example. Because of the executing time, some parts of the programm are written in **PVM**. The last section deals with the result. We present the picture, created by the program, compare the algorithm using different points for the numerical integration and give at least some conclusions.

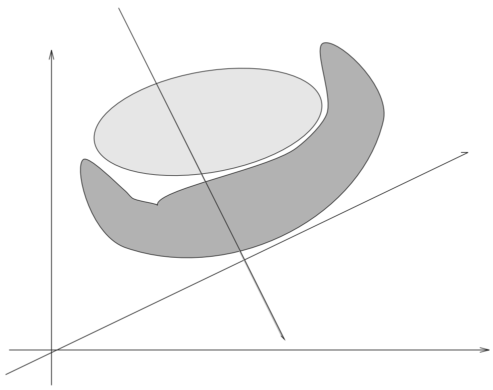
2 The Radon Transform

2.1 The Radon Transform

The radon transform of a three dimensional body is the parallel projection of the denseness function onto a plane. For simplicity in this paper the radon transform will be only treated for two dimensional objects. To generalize the radon transform, you have to take into account the fact that a two dimensional object is a crosscut of a three dimensional one. So in order to obtain a three dimensional image one scans the body layer by layer.

A two dimensional object can be described by a function $f(x, y) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$, where $f(x, y)$ is the denseness of the object in the point (x, y). The Radon transform $R_\theta(s)$ of a point (θ, s) is given by the parallel projection onto the corresponding line L_θ at an angle θ with the x-axis, evaluated at a distance s to the origin. The projection onto the line L_θ can be described by the following integral:

$$p_\theta(s) = \int_{(L_\theta(s))^\perp} f(x(u), y(u))\, du$$



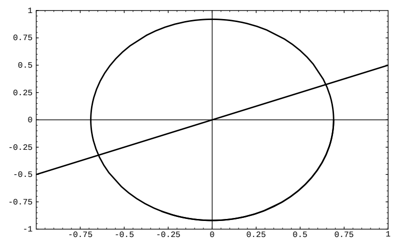
where $L_\theta^\perp$ denotes a line orthogonal to L_θ and through the point $L_\theta(s)$, s the distance to the origin. Therefore each point of the state space $[0, \pi] \times \mathbb{R}$ corresponds to a value $p_\theta(t)$. The Radon transform of the function $f(x, y)$ is the two dimensional function $(\mathcal{R}f)(\theta, t) : [0, \pi] \times \mathbb{R} \to \mathbb{R}^+$, with $(\mathcal{R}f)(\theta, t) = p_\theta(t)$

The problem is to reconstruct the object, respective its denseness function $f(x, y)$, given the radon transform. The solution is given by the inversion formula of the radon transform. Johann Radon developed an explicit inversion formula in 1917 [8]. If there are available all values of $R_\theta(t)$, the inversion formula works quite well.

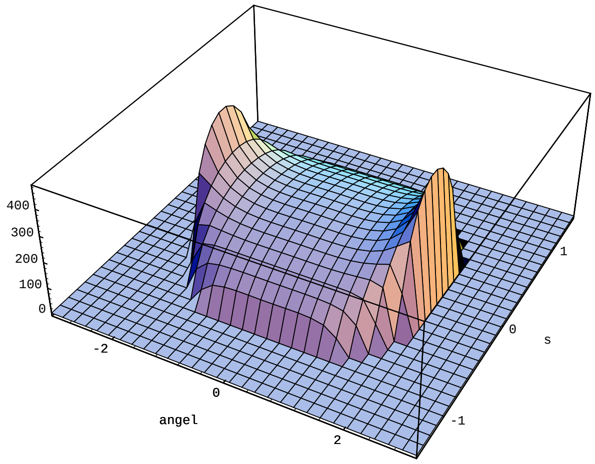
But this formula leads to numerical instabilities, if there are only a finite number of angles and values of s, where the radon transform can be evaluated. The problem now is, that in physics the condition of a continuous sample space is violated. That means that one can measure the Radon transform only on a finite number of points, for example on a lattice $M_n = \{(s_i, \theta_i) \,|\, i = 1 \cdots n\}$. Further the datas are often incomplete or it is expensive to get a great number of datas. Therefore it is of practical importance to find algorithms which yield a good approximation of the denseness function, utilizing only few number of points.

2.2 The Filtered Backprojection Algorithm

The radon transform $(\mathcal{R}f)(\theta, \cdot)$ is a parallel projection onto the line L_θ through the origin with the angle θ. Along the line L_θ the values of the denseness function are integrated. To test the algorithm, we first create a phantom picture of superposed ellipses and then we determine the radon transform. The next figure shows an example of the radon transform of an ellipse. The angle, drawn in the ellipse is $\frac{\pi}{6}$.



The ellipse



The corresponding radon transform

Note, that the value of the denseness function in the point (x, y) appears at the distance $s(\theta) = \sqrt{x^2 + y^2} \cos(\arctan(\frac{x}{y} - \theta))$ on the line L_θ to the origin. For a fixed angle, each point (x, y) influences the projection onto the line L_θ. To get the exact value of the denseness function, you wobble around the value $s(\phi)$ and weight the result by an appropiate factor. This means, first the path integral around a circle with middle point $s(\theta)$ will be determined and than the derivate to the radius of the path integral divided by the radius will be taken. This can only be done by continuous values. If we want a good approximation of the denseness function by discrete datas we have to add one stepp. The function value $f(x_0, y_0)$ at a point (x_0, y_0) can be represented by a convolution of f with a delta function[1]. If the function is available only on a finite set of points $\{x_i, i = 1 \cdots n\}$, one can approximate the function g by convolution of the sample function with an approximated delta function.

$$\tilde{g}_{appr}(x) = \delta_{appr} * \left(\frac{1}{n} \sum_{i}^{n} g(x_i) \right) (x)$$

[1]The delta function is roughly spoken zero everywhere but of one point x_0. In this point, the function is not defined resp. infinite. The most important property of a delta function δ_{x_0} is

$$g * \delta_{x_o} = \int_0^\infty g(x)\, \delta_{x_0}(x)\, dx = g(x_0)$$

where $g : \mathrm{IR} \rightarrow \mathrm{IR}$ is an arbitrary continuous function.

Most interpolation formulas can be written as a convolution with an approximation of the delta function. It can be compared with smearing the exactly value in the neighborhood of each point. Now, the idea behind the filtered backprojection is to convolute the reconstructed function with an appropriate filter. The trick is to convolute first the radon transform with the filter, and then by the second step to reconstruct the function by the exact formula of Johann Radon. Such an approximative delta function is called low pass filter, because the approximated δ function take into account only the fourier coefficients untill a certain frequency and cut off the higher by convolution.

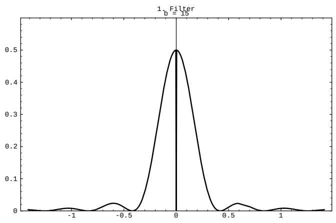
This trick can be applied, because both operations, calculating the radon transform and convolution, commute. This significates, that it doesn't matter to go in the left diagram first right and then down or first down and then right. Because the inversion radon transform is unique, the second diagram commutes also. The advantage is that the convolution of two radon transform is just a multiplication and much more quicker to determine than the convolution of the functions themselfs.

$$\begin{tikzcd} f \arrow[r] \arrow[d] & Rf \arrow[d] \\ f*d \arrow[r] & R(f*d)=(Rf)*(Rd) \end{tikzcd} \qquad \begin{tikzcd} Rf \arrow[r] \arrow[d] & f \arrow[d] \\ (Rf)*(Rd) \arrow[r] & f*d \end{tikzcd}$$

Thus, to reconstruct the value, first the radon transform is multiplicated by a filter and then the inversion formula of J. Radon is applied:

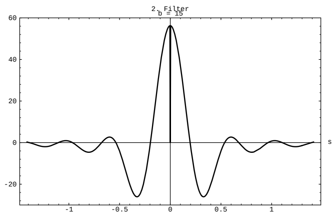
$$\tilde{f}(x,y) = \int_0^{\pi} \int_{\sqrt{2}}^{\sqrt{2}} g_{(x,y)}(\theta, s)\ (\mathcal{R}f)(\theta, s)\ ds\ d\theta \tag{1}$$

where $g_{(x,y)}$ is the kernel of the filter. The kind of filter function depends on the algorithm and vice versa. We have implemented two filter functions. The first is treated in the PhThesis [9], the second in the article [10].



$$g_1 = \frac{\cos(bz) - 1}{b^2\ z^2}$$

and



$$g_2 = b^2 \int_0^1 \cos(z\pi bu)\, u(1-u^2)\, du$$

where $z = z_{x,y}(\theta, s) = x\sin(\theta) + y\cos(\theta) - s$ is a function of (θ, s) and denotes the distance of $L_\theta(s)$ to the intersection point of P.

The first smears the value around the point much more than the second. Thus, the reconstruction picture by the first filter is much more blury than the of the second filter. Chapter 2.4 deals with the influence of the bandparameter.

2.3 Numbertheoretical Methods in Numerical Integration

The integration in 1 can only be determined by numerical methods, because it is impossible to measure the values of (θ, s) on all points in $\left[-\sqrt{2}, \sqrt{2}\right] \times [0, \pi]$. Usually, this numerical integration is done on a grid with equidistant points:

$$\theta_i = \frac{i}{n}\,\pi \quad i = 1 \cdots n_\theta,$$

$$s_i = (2\,\frac{i}{n} - 1)\,\sqrt{2} \quad i = 1 \cdots n_s,$$

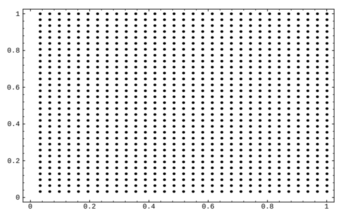
$$f_{appr}(x,y) = \frac{1}{n_\theta}\,\frac{1}{n_s}\,\sum_{i=1}^{n_\theta}\sum_{j=1}^{n_s}\,g_{(x,y)}(\theta_i, s_i)(\mathcal{R}f)(\theta_i, s_i)$$

In the theory of numerical integration the method of good lattice points is well known. Good lattice points are a sequence of points with small discrepancy. It exceeds the scope of this paper to point out numbertheoretical methods in numerical integration. For a good introduction, please see: [6, 7]. In this paper we used good lattice points created by Fibonacci numbers[2].

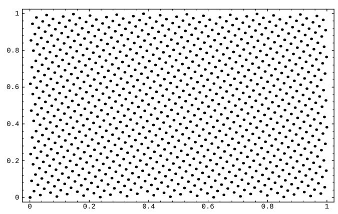
$$\theta_i = \left[\frac{i\,F_{n-1}}{F_n}\right]\,\pi \quad i = 1 \cdots F_n,$$

$$s_i = (2\,\frac{i}{F_n} - 1)\,\sqrt{2} \quad i = 1 \cdots F_n,$$

randomly choosen points and an equidistant grid. The following picture shows an equidistant grid and a fibonacci lattice on the intervall $[0,1] \times [0,1]$.



equidistant points



good lattice points

[2]The n'th Fibonacci number is given by the following recursion:

$$F_0 = 1,\ F_1 = 1,\ F_n = F_{n-2} + F_{n-1}$$

Now, the questions arise, where is the different between good lattice points and an equidistant grid? Why should we use good lattice points? Good lattice points have a smaller discrepancy than a grid with equidistant points. The importance of the discrepancy is given by the inequality of Koksma and Hlawka:

$$\left| \int_0^1 f(x)\,dx - \frac{1}{N}\sum_{n=1}^{N} f(\xi_n) \right| \quad \leq \quad D_N^*\, V(f)$$ [3]

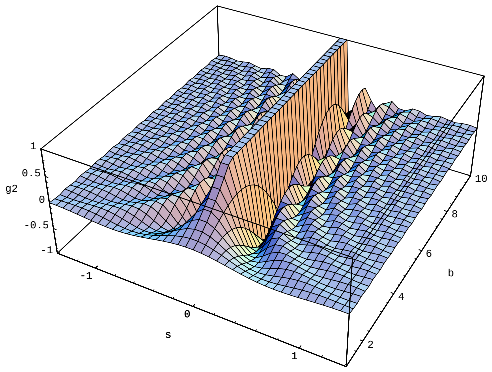
where D_N^* denotes the discrepancy of the sequence $(\xi)_{i=1}^N$ [4] and $V(f)$ the total Variation of the function f. You can estimate the error by the discrepancy and the total variation. Thus if we reduce the discrepancy, the error will decrease as well. In $\mathbb{R}$ the most function are of bounded variation. But if you deal with functions from $\mathbb{R}^2$ to $\mathbb{R}$ for example characteristic functions are of unbounded variation. Because of simplicity we have used in our example the characteristic function of an ellipse, which are of unbounded variation. In this case the discrepancy D_N^* must be replaced by the isotropic discrepancy D_N. Now, the isotropic discrepancy of good lattice points and equidistant points are of the same order. Therefore the error is also of the same order. In the subsequent paper we want to test the algorithm by differentiable functions.

2.4 The Influence of the Parameters

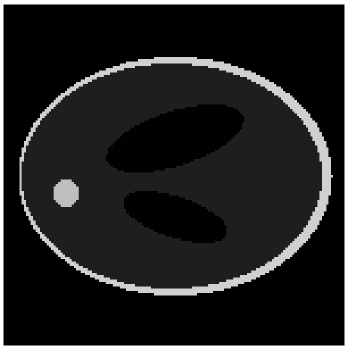
In order to illustrate the influence of the band parameter, we present a sequence of reconstructions, varying only the band parameter. Varying the band parameter corresponds to the approximation the delta function better or worse. The filter is stretched by increasing the band parameter and is contracted by decreasing the parameter. If the δ function is approximated worse, the fouriercoefficient of high order are cut off by convolution. Thus you can compare an approximated delta function by a low pass filter, which suppresses noise.

[3] This is the one dimensional version of the inequality

[4]
$$D_N^* = \sup_{I_1 \times I_2} \left| \frac{|\,(I_1, I_2) \cap \{\xi_i \mid i = 1 \cdots N\}}{N} - \lambda(I_1)\,\lambda(I_2) \right|$$



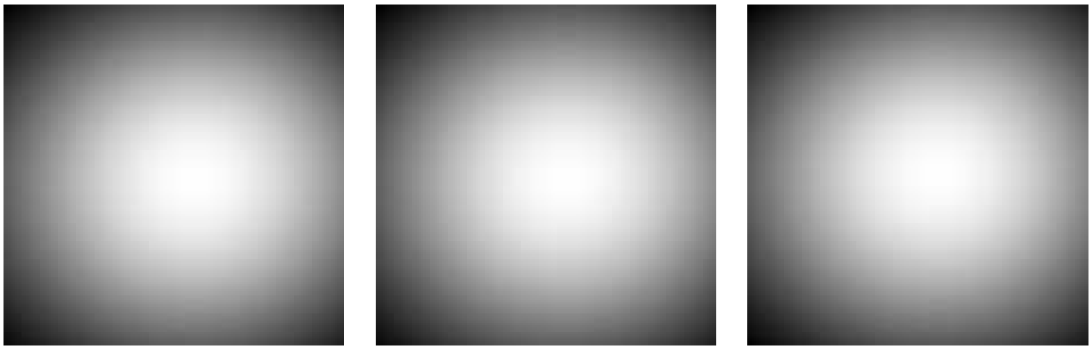
If there are only a few radon transforms available, the band parameter must be choosen low. In this case the reconstructed picture will be bleary. If it can be assumed, that there are many points of the radon transform, the band parameter can be choosen higher. If the band parameter increases, also the accurancy of the pictures increases. But the artefact caused by the numerical method increases as well. Further depends the band parameter on the filter kernel. For example, the first filter needs a higher band parameter than the second. The influence is easy seen in the following pictures. First you can see the original:



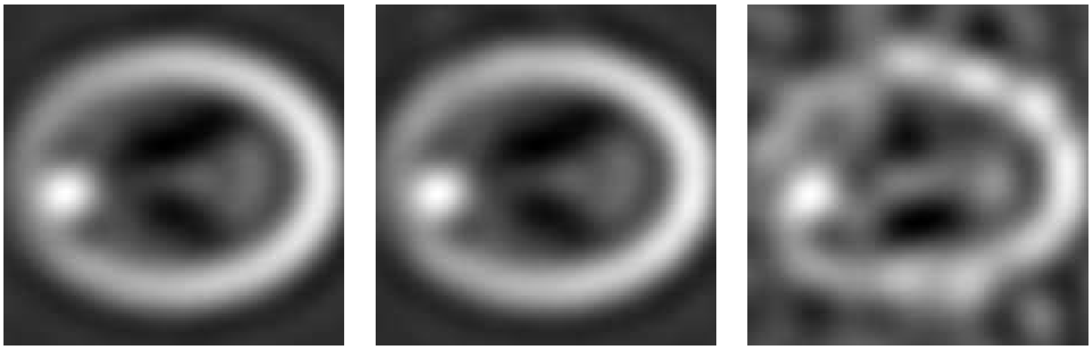
150×150 pixels, the intergration is done in (1) with an equidistant lattice of 82×82 points, in (2) with fibonacci numbers, 6765 points, and in (3) with 6765 randomly choosen points. The picture is drawn on $[0, 1]$. The point (0,0) corresponds to the point left above, (1,1) to right down.

We have reconstruct the picture by an equidistant grid, fibonacci numbers and randomly choosen points. If the band parameter is to low, the reconstructed pictures are blury and you cannot make out any object but of a circle. By bandwith 13 the objects are easy to recognize. If the bandwith decreases, artefacts arises.

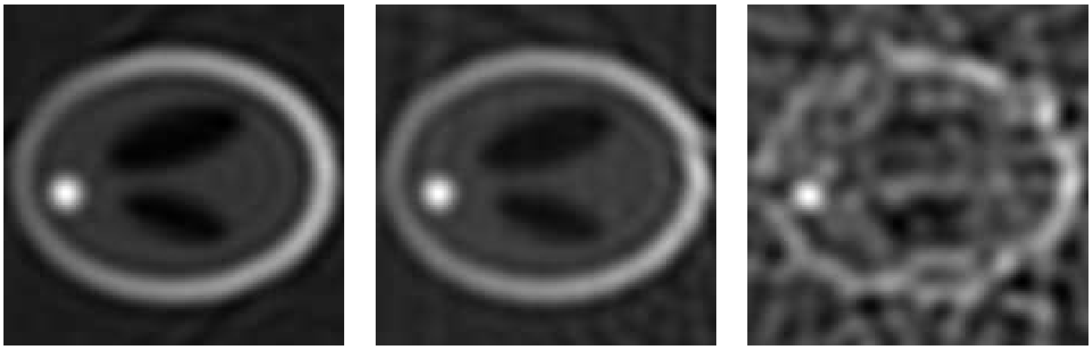
(1) **(2)** **(3)**



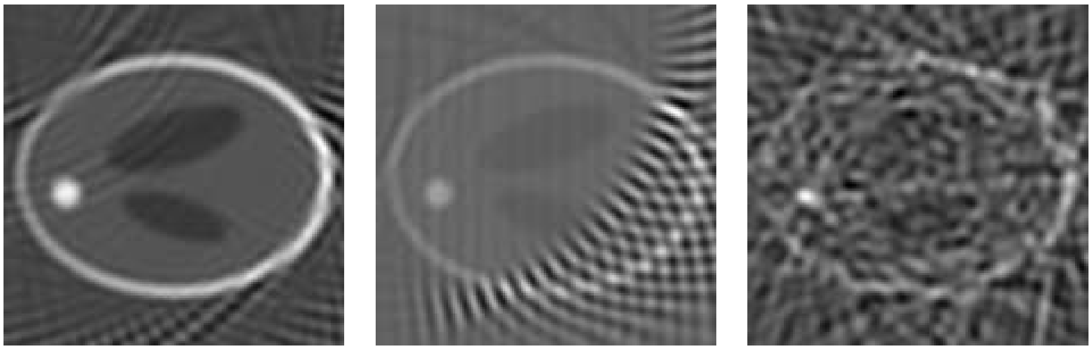
Bandwith: 1



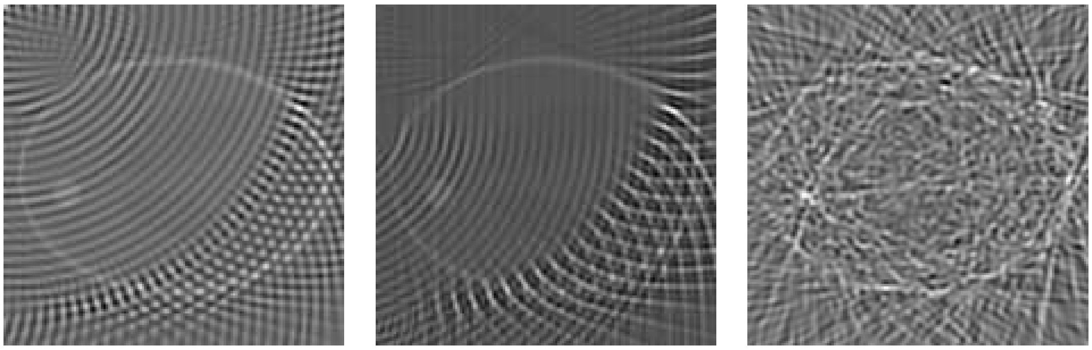
Bandwith: 7



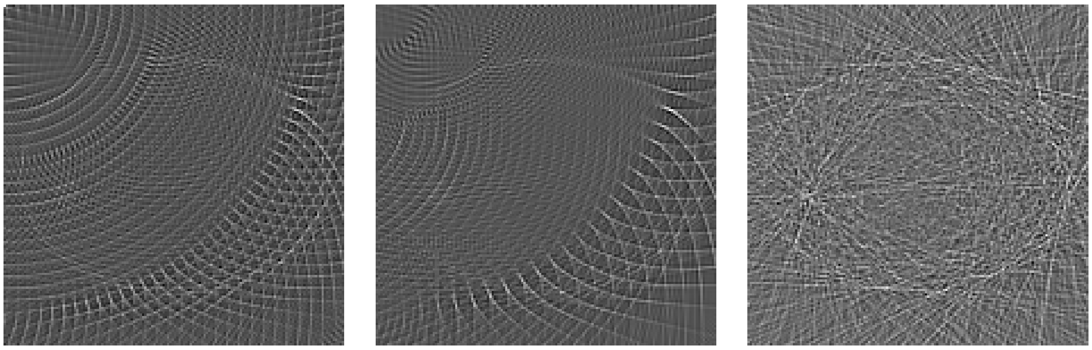
Bandwith: 13



Bandwith: 25



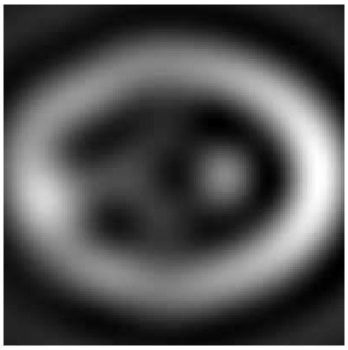
Bandwith: 50



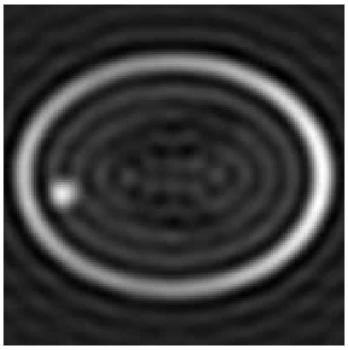
Bandwith: 100

To compare the two algorithm we present a sequence of pictures reconstructed by the algorithm of th PhThesis [9]. The bandwith must be choosen in another range. Further the artefacts never disappear like in the pictures above. The integration is done by an equidistant grid, the number of pixels are 200×200, the number of reconstrunction points 82×82. The picture is drawn on $[-1, 1]$. The point (-1,-1) corresponds to the point left above, (1,1) to right down.

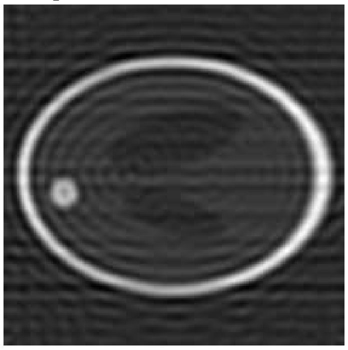
Original



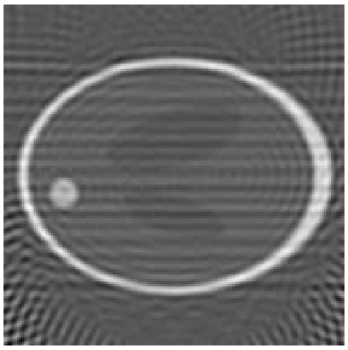
bandwith 15



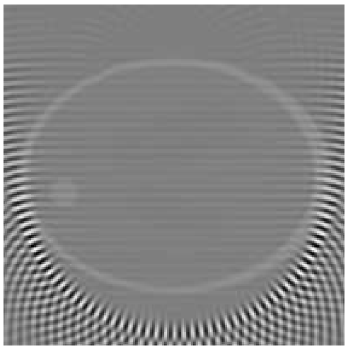
bandwith 45



bandwith 75



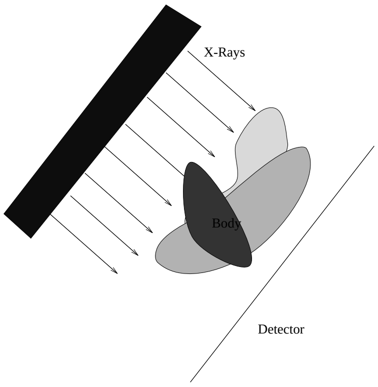
bandwith 100



bandwith 125

2.5 Applications of the Radon Transform

The most common application of the radon transform is computerized tomography. The radon transform arises in a natural way, if X-rays traverse a body. A cross section of the human body is scanned by a thin X-ray beam whose intensity loss is recorded by a detector. Let $f(s)$ be the denseness of the tissue along the line $L_\theta^\perp(s)$, $s \in \mathbb{R}$. The amount of absorbed intensity ΔI in a small distance Δs at the point $L_\theta^\perp(s)$ is proportional to $\Delta I / I = f(s)\, \Delta s$. Therefore the intensity of the beam after having passed the body along $L_\theta^\perp$ is



$$I_0 = \exp\left(-\int_{L_\theta^\perp} f(s)\, ds\right)$$

Thus the reconstruction problem of computerized tomography is just the inversion problem of the radon transform.

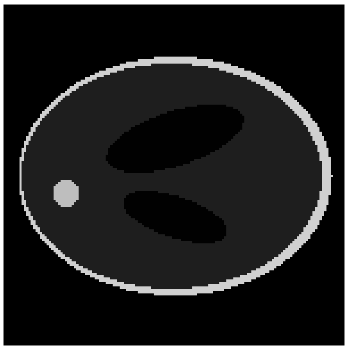
Further over the years the physical method of computerized tomography became to be used in a wider sense, namly for any technique of reconstruction internal structures in a body or tissue from data collected by detectors outside the body, for example in electron microscopy and electromagnetic scattering. Therefore computerized tomography is a method which is used in many fields like physics, engineers, bioscientists, medicin and many others.

3 The Software Tool: Radon

The program is written in C++ and is built up like a C++ library. The aim was to give the user a tool for research in the radon transform. With the library the user can outline a program by himself with only a few lines. First we describe the structure of such a program, then we give an example.

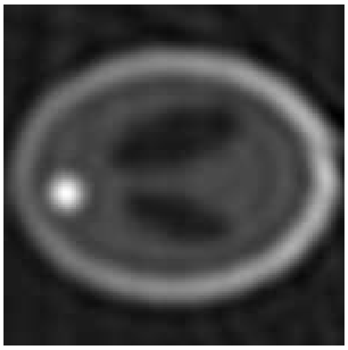
3.1 The Structure of a Test Program

To test the algorithm a phantom object of superimposed ellipses will be designed. Such a picture has three advantages: first the radon transform of a ellipses is known exactly and second it is calculated quickly. Third, the picture looks like a crosscut of a brain.



The phantom picture is drawn by 150×150 pixels and on the range $[0,1] \times [0,1]$. The point (0,0) corresponds to the point left above, (1,1) to right down.

The next step is to transform the phantom picture. Then the inversion formula will be applied to the radon transform. Now the reconstructed picture can be compared with the original.



This picture shows a reconstructed phantom picture. The numerical integration is done by 6765 Fibonacci points. The error is 51, this means the percentual average of the difference to the original is $20 \equiv \frac{51}{256}$.

The test algorithm is divided in the following steps:

1. **Create the Phantom Picture**

   First the user defines a variable of type *phantom*. Further he adds ellipses one by one to the object. This can be done by the following commands:

   phantom phanta;
   ellipse elli;
   elli.initialize(middlepoint, axis, angle, color);
   phanta.add_ellipse(elli);

2. **Draw the Phantom Picture**
   The user defines a variable of type *bild* with the appropiate size. Then the range of the picture must be set. Take into account that the function *radon-transform()* operates on the range $[-1,1]\times[-1,1]$ and the function with prefix *new1* like *new1_radontransform()* on $[0,1]\times[0,1]$. The last command draws the picture, this means the function proofes if a point of the range is inside of a ellipse and add the color if yes.

   bild picture(100,100);
   picture.set_intervall(0,1);
   phanta.pvm_draw(picture);

3. **Transform the Phantom Picture**
   Here the user has two classes available. One class to determine the radon transform on an equidistant grid, the second class to determine the radon transform on good lattice points. The following two kinds of good lattice points are implemented: Fibonacci points and points choosen randomly by the C++ random generator. The corresponding commands are given by:

   - defines a equidistant grid of size 80×80:

     radon radon_picture(80,80);

   - defines a grid of good lattice points:

     radon_rand radon_picture(6450);
     This two commands entry the Fiboncci points respective the randomly choosen points to the grid:

     radon_picture.fibonacci(18);
     radon_picture.randomize();

   Now, the radon transform can be determined by the command:

   phanta.new1_radontransform(radon_picture);

4. **Reconstruct the Object**
   First, the user defines a picture of the same size like the original picture. Then

he applies the inversion formula to the radon transform. The user can choose also another size, but in this case the error cannot be determined.

filter reconstructed_picture;
reconstructed_picture=picture;
reconstructed_picture.new1_reconstruction(radon_picture,band);

5. **Determine the Error**
   Now, if the user is interested in the mean square error, he can get the error by the function:

   double error = picture.error(reconstructed_picture);

The error of the reconstructed picture is the 2-norm of the difference between the reconstructed picture and the original. Note that after reconstructing the values of the denseness function, the picture will be prepared for visualization. This will be done by stretching the range of the function onto the intervall $[0,255]$. Therefore, first the range of the phantom picture will be prepared like the reconstructed picture and afterwards the error will be determined.

$$error = \sqrt{\frac{1}{n_x}\frac{1}{n_y}\sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\left(f(x_i,y_j) - f_{rec}(x_i,y_j)\right)^2} \tag{2}$$

Further, there exists a routine which transforms a picture in **xv**-format and stores it in a file. Thus the user can visualize the picture and compare the quality also by looking. The corresponding command is *reconstructed_picture.print(filename);*.

You see, the user can write a program with only a few simple commands and must not bother his head about handling the pixels of the pictures, the arrays of pointers and so on. Further, to add a new inversion formula, the user can do it by adding two procedures, one to the object *radon* and one to the object *radon_rand*. In the next section is presented a description of the objects and the corresponding functions.

3.2 An Example

The program library uses mathematical functions, input output functions and also **PVM**. Therefore the following files must be included: iostream.h, stdio.h, stdlib.h, math.h. Further the **PVM** library and the program tool itself must be included. After the header the main function begins. The structur is pointed out above. Here we present only a example.

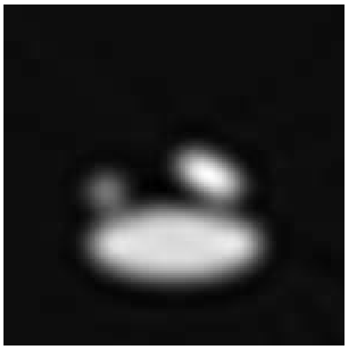
```
# include <iostream.h>
# include <time.h>
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include "/usr/local/pvm/pvm3/include/pvm3.h"
# include "radon.cc"

void main();

void main() {

phantom phanta;
ellipse elli;
double error = 0.0;
char* name;

    elli.initialize( 0.7, 0.5, 0.10, 0.25, 0, 175);
    phanta.add_ellipse(elli);

    elli.initialize( 0.5, 0.6, 0.1, 0.05, 60, 207);
    phanta.add_ellipse(elli);

    elli.initialize( 0.55,0.3, 0.05, 0.05, 0, 111);
    phanta.add_ellipse(elli);

// ==================================================
// the size of the picture will be initalized
// ==================================================

    bild picture(150,150);
        picture.set_intervall(0,1);
        phanta.draw(picture);
        name="original.pbm";
        picture.print(name,0.0,10);


// zum testen mit fibonacci
// ***********************

    radon_rand radon_fib_picture(2);
```

```
    radon_fib_picture.fibonacci(18);

// zum testen mit random
// ************************

    radon_rand radon_rand_picture(5465);
    radon_rand_picture.randomize();

// zum testen mit equidistante grid
// ***************************

    radon radon_picture(84,84);


// evaluate the radon transform of the cooresponding picture
//*********************************************************
    phanta.pvm_new1_radontransform(radon_fib_picture);
    phanta.pvm_new1_radontransform(radon_rand_picture);
    phanta.pvm_new1_radontransform(radon_picture);


// three pictures are defined
//*********************************************************
    filter reconstructed_fib_picture;
    reconstructed_picture= picture;

    filter reconstructed_rand_picture;
    reconstructed_picture= picture;

    filter reconstructed_picture;
    reconstructed_picture= picture;


// the pictures are reconstructed
//*********************************************************
    reconstructed_fib_picture.pvm_new1_reconstruction( radon_fib_picture,10);
    reconstructed_fib_picture.entry();
    error = picture.error( reconstructed_fib_picture);
    name="fib";
    print(name,10,error);
```

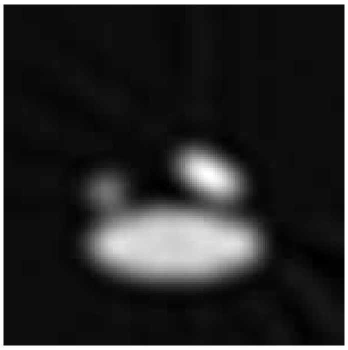
```
reconstructed_rand_picture.pvm_new1_reconstruction( radon_rand_picture,10);
reconstructed_rand_picture.entry();
error = picture.error( reconstructed_rand_picture);
name="rand";
print(name,10,error);

reconstructed_picture.pvm_new1_reconstruction( radon_picture,10);
reconstructed_picture.entry();
error = picture.error( reconstructed_picture);
name="equi";
print(name,10,error);

}
```

The program reconstructs a given phantom picture by an equidistant grid, Fibonacci points and randomly choosen points. Further it converts the original and reconstructed picture into a **xv**-format. After we have translate the pictures into a postscript file:

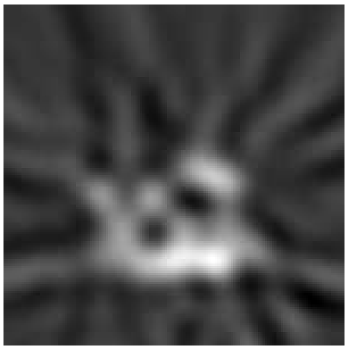
This picture shows the phantom pictures. It is drawn by the command *phanta.drawn(picture);* and converted in **xv**-format by *picture.print(name,0,0);*. By **xv** the picture can be converted in a postscript file.



equidistant grid



Fibonacci points



randomly choosen points

3.3 Description of the Classes

3.3.1 The Object Phantom

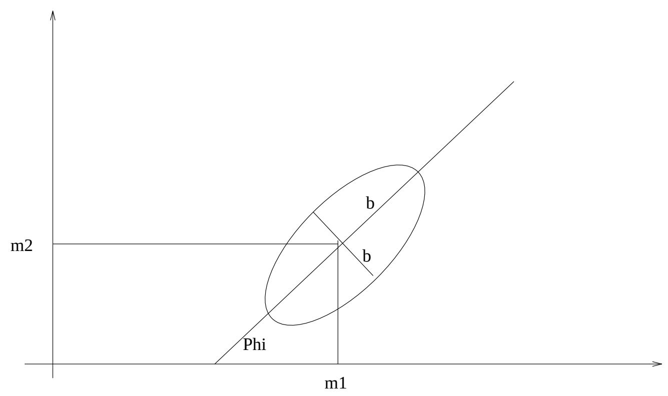
The object *Phantom* is roughly spoken an array of ellipses. Further, the object consists of a number of functions, e.g. with the function *add()* you can add one ellipse to the phantom. Thus you create your phantom picture by adding ellipses one by one. The next step is to draw the picture. This can be done by the function *draw(picture)*. There are also functions to determine the radon transform and to assign the radon transform to an appropiate object. A list of the function is given below.

- *void add_ellipse(middlepoint,axis,angle,color);*
  add a ellipse to the object.
- *void draw(picture):*
  draws the phantom in the given pictures.
- *void pvm_draw(picture):*
  the same procedure like above. the only difference is, it uses **PVM**.
- *void radontransform(radon picture):*
  determines the radon picture on the intervall $[-1,1] \times [-1,1]$.
- *void new1_radontransform(radon picture):*
  determines the radon picture on the intervall $[0,1] \times [0,1]$.
- *void pvm_new_radontransform(radon picture):*
  determines the radon picture using **PVM**.

3.3.2 The Object ellipse:

An ellipse is described by its middle point, axis and angle. Thus the object *ellipse* contains the corresponding variables. If the user wants to set the internal variables of the ellipse to certain values, he can do it by initializing:

- *void ellipse ell1(middlepoint_x, middlepoint_y,a,b,phi,color)*
- or if the ellipse exists already: *void ell1.initialize(m1,m2,a,b,phi,color).*



The other functions are used by the object *phantom*.

- *void radontransform(radon picture):*
  determines the radon picture. It is assumed that the ellipse is in the intervall $[-1,1] \times [-1,1]$.
- *void new1_radontransform(radon picture):*
  determines the radon picture of the ellipse. The ellipse must be contain in the intervall $[0,1] \times [0,1]$.
- *void pvm_new1_radontransform(radon picture):*
  determines the radon picture using **PVM**.
- *void draw(bild):*
  entries the color of the ellipse to each point inside the ellipse.
- *void draw_and _add(bild):* adds the color of the ellipse to each point inside the ellipse .
- *int inside(x, y):*
  proofs if the point (x, y) lies inside or outside the ellipse.

- *int transform_x(x, y), int transform_y(x, y):*
  evaluates the transformed x-coordinate respective y-coordinate of the point (x, y) relative to the ellipse.

3.3.3 The Object bild and filter

The object picture is just an array of integer values. Further, it consists of the following functions:

- *void resize(a,b):*
  allocates the matrix of the given size and fills it with zero.
- *void set_intervall(x,y):*
  sets the intervall of the range to $[x, y] \times [x, y]$.
- *double error():*
  determines the error described in equation 2
- *void print(char: name of the file):*
  creates a file, which can be visualized by the program tool **xv**.
- *void output():*
  write the values of the pixels on the screen.
- *int minimum(), int maximum():*
  determines the minimum respective maximum of all pixels.

More important is the *subclass filter*, which belongs to the object *bild*. The filter contains a second array of double, with the same size like the other array. The functions are:

- *void new1_reconstruction(radon_picture):*
  reconstruct the denseness function by the radon transform in the range defined by the picture. The algorithm described in [10] is applied.
- *void pvm_new1_reconstruction(radon_picture):*
  reconstruct the denseness function by the radon transform using **PVM**.
- *void reconstruction(radon_picture):*
  reconstruct the denseness function by the radon transform in the range of the picture by the algorithm in the PhThesis.

- *void entry():*
  carries over the array of double into the array of integer and prepares the values to visualize. This means, the range is stretched on the range $[0, 255]$. There are some possibilities implemented for visualization.

  attention: the error function is calculated by the visualized picture.

- *double minimum(), double maximum():*
  determines the minimum respective maximum of the array of double.

3.3.4 The Object radon:

The object *radon* stores the evaluated values of the radon transform. A subclass belongs also to the object: the object *radon_rand*. This subclass possesses the same functions as the superclass but of two initialization procedure. It treats the case of good lattice points. The most functions are overloaded. The functions are:

- *void resize(a,b), void resize(a)*
  allocates the matrix where the radon transform will be entered and fills it with zero.

- *void add(radon):*
  adds the radon transform to the own.

- *void output():*
  print the reconstructed values of the denseness function on the screen.

- *void fibonacci():*
  belongs to *radon_rand*. Determines the good lattice points based on Fibonacci number and stores it.

- *void randomize():*
  belongs to *radon_rand*. Throws random points.

- *void set_zero(int,int):*
  sets the given point in the array to zero. If the first entry is minus one, the procedure sets all the values of the given value by the second argument to zero and vice versa. By *random_rand* you can only choose the number of points which should be set to zero.

- *double new1_reconstruction(x,y):*
  reconstruct the denseness function in (x, y) by the radon transform. The function is called by *filter::new1_reconstruction(radon_picture)*.

- *double reconstruction(x,y):*
  reconstruct the denseness function in (x, y) by the radon transform using the algorithm of the PhThesis [9]. The fuction is called by *filter::reconstruction(radon_picture).*

4 Parallel Virtual Machine (PVM)

The Parallel Virtual Machine is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible parallel computational resource. It can be used especially for the development and execution of large concurrent or parallel applications that consist of many interacting, but relatively independent components [3]. **PVM** was developed initially at the Emory University and Oak Ridge National Laboratory.

The individual computers may be shared- or local-memory multiprocessors, vector supercomputers or scalar workstations that can be interconnected by a variety of networks (ethernet, FDDI, etc.). User programs written in C, C++ or Fortran are provided access to **PVM** through the use of calls to **PVM** library routines for functions such as process initiation, message transmission and reception. The **PVM** system handles message routing, data conversion for incompatible architectures and other tasks that are necessary for operation in a heterogeneous network environment. For more details see [2].

4.1 Implementation with PVM and Load Balancing

Since the radon transform or reconstruction for different points (s_i, θ_i) of the lattice for a point (x_i, y_i) of the image are evaluated independently, the outer loop may be distributed among the processors. On a dedicated network, where the user disposes 100% of the machines' capacity on each node, this can be done by partitioning the lattice into equal parts for each node. If there are other processes running or if a heterogeneous network is used, a processor with high workload or a slower one would cause a delay in the whole computation. With a finer partitioning of the lattice it is possible to react on changes in the system performance. Therefore the algorithm is implemented using the master–slave paradigm and the asychronous pool of tasks methodology.

4.2 PVM and Load Balancing applied to the Program Package

The program of determining the radon transform of a given phantom and reconstructing the image contains three time consuming procedures: *phantom::draw( )*, *phantom::transform( )* and *radon::reconstruct( )*. Therefore for each procedure there exists a version using **PVM** with load balancing. The rough structure of the functions are given below:

4.2.1 The Function phantom::pvm_draw()

First the master sends the size of the image to each slave. The next step is to distribute the ellipses among the slaves. The task of each slave is to draw the received ellipse. After he finished, he will send the image back to the master. If there are ellipses left, he gets the next one. The master collects all images and sums up all together.

The rough structure of the master program:

```
enroll in PVM
get startparameters (number of processors (n) and so on)
invoke the slaves
send the size of the image
send the first n ellipses to the slaves
loop (i=1) to number of the ellipses
        wait until a image arrives
        add the image to the own
        if there is a ellipse left
                send it to the slave
end loop
kill the slaves
```

The rough structure of the slave program:

```
enroll in PVM
receive the size of the image
infinite loop
        get ellipse
        compute the image
        send the image to the master
end loop
```

4.2.2 The Function phantom::pvm_radon_transform()

The task is divided in exactly the same manner like in *phantom::pvm_draw()*. Each slave calculates the radon transform of the given ellipse. The master adds one by one to his own radon picture and orders the slave to calculate another ellipse if it is necessary.

The rough structure of the master program:

```
enroll in PVM
get startparameters (number of processors (n) and so on)
invoke the slaves
send the size of the radon picture
if good lattice points are choosen
      send the angles and s-values
send the first n ellipses to the slaves
loop (i=1) to number of the ellipses
      wait until a radon picture arrives
      add the radon picture to the own
      if there is a ellipse left
            send it to the slave
end loop
kill the slaves
```

The rough structure of the slave program:

```
enroll in PVM
receive the size of the radon picture
if good lattice points are choosen
      get the angles and s-values
infinite loop
      get ellipse
      compute the radon picture
      send the radon picture to the master
end loop
```

4.2.3 The Function filter::pvm_reconstruct()

By reconstructing a image, the denseness function of each point in the unit square can be reconstructed independently. Therefore each slave can determine the value of the denseness function in one point. But the amount of communication time increases if the tasks are to small. Because of this for casting the task each slave gets the task to reconstruct a column of the image. First the master distributes the radon transform and the size of the image to reconstruct to all. After this each slave gets the number of the column he has to evaluate. The next step, after a slave finished his task, he will send back the result and get a new number of column if there are some left.

The rough structure of the master program:

```
enroll in PVM
get startparameters (number of processors (n) and so on)
invoke the slaves
send the radon picture
if good lattice points are choosen
      send the angles and s-values
cast the tasks to the slaves (send each slave the number of the column, he has to evaluate)
loop (i=1) to number of the x_pixels
      wait until a vector arrives
      entry the values to the own reconstruction image
      if there is a column left
            send the number of the column to the slave
end loop
cast to all slaves −1
```

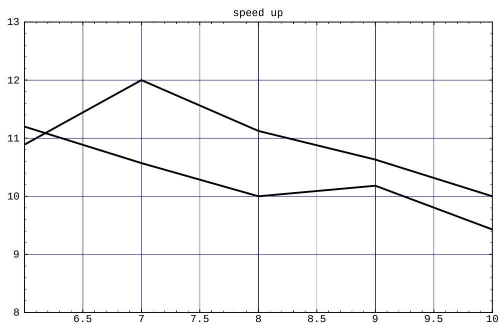
The rough structure of the slave program:

```
enroll in PVM
get the radon picture
if good lattice points are choosen
      get the angles and s-values
get the size of the image to reconstruct
get the number of column to reconstruct
while (number of column >-1)
      evaluate the corresponding column of the reconstructed image
      send the image to the master
end loop
pvm_exit()
```

4.3 The Speedup of Executing Time

Because all time consuming functions are also implemented in **PVM**, you can neglect the amount of time needed by the other procedures. Further, the executing time of a program using **PVM** depends first and foremost on the number of hosts. Thus, the factor of the execution time of a sequentiell version to the paraellized version of the program is equivalent to the number of hosts. In our example, the computations dedicated to eight DEC 5000/33 Workstations. In order to demonstrate the speed up, we reconstuct pictures of 150×150 and 200×200 pixels by the function *bild::pvm_new1_reconstruct( )*. We vary the number of points evaluating the radon transform between 60×60 and 100×100. The result is given in the table below.

Pixels	*Points*	*Time*		speed up
		parallelized version	sequentiell version	
150×150	60×60	5 min	56 min	11.200
	70×70	7 min	1h 14 min	10.571
	80×80	9 min	1h 30 min	10.000
	90×90	11 min	1h 52 min	10.182
	100×100	14 min	2h 12 min	9.429
200×200	60×60	9 min	1h 38 min	10.889
	70×70	11 min	2h 12 min	12.000
	80×80	16 min	2h 58 min	11.125
	90×90	19 min	3h 22 min	10.632
	100×100	27 min	4h 30 min	10.000



Observe, that the factor by sequentiell versus parallel is higher than eight. This fact arises by load balancing. A machine which is idle will evaluate more points than another, which is occupied with other tasks. If the program is started on only one machine, it will not be taken into account if the machine is busy by other tasks.

5 Results and Conclusions

5.1 Numerical Results

5.1.1 Reconstructed Images

In order to test the algorithm we reconstuct two sequences of imagees. The images of the first sequences are approximated by an algorithm described in the PhD-thesis of Michael Revers (algorithm 1). The images of the second sequences are evaluated by an new method which is based on an article of Michael Revers (algorithm 2). Further, one sequence can be divided into three kinds of reconstruction. At each reconstruction we used different grids for the numerical integration.

- Equidistance Points

$$\theta_i = \frac{i}{n}\,\pi \quad i = 1\cdots n,$$

$$s_i = (2\,\frac{i}{n} - 1)\,\sqrt{2} \quad i = 1\cdots n,$$

- Randomly Choosen Points

$$\theta_i = \frac{i}{n}\,\pi \quad i = [rand()/RAND\_MAX],$$

$$s_i = (2\,\frac{i}{n} - 1)\,\sqrt{2} \quad i = [rand()/RAND\_MAX],$$

- Good Lattice Points

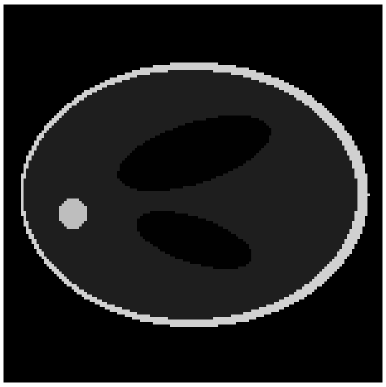
$$\theta_i = \left[\frac{i\,F_{n-1}}{F_n}\right]\,\pi \quad i = 1\cdots F_n,$$

$$s_i = (2\,\frac{i}{F_n} - 1)\,\sqrt{2} \quad i = 1\cdots F_n,$$

Note, that the angle is only running from 0 to π. The points of $[\pi, 2\pi]$ are given by the symmetric relation $(\mathcal{R}f)(\theta, s) = (\mathcal{R}f)(\theta+\pi, -s)$. Thus, in the program we identified the two points and reconstruct the pictures where θ runs over $[0, 2\pi]$.

For testing the algorithm we create two phantom iamges, phantom 1 and phantom 2. The denseness functions of the phantom iamges are given by superimposed ellipses of different colors.

phantom 1

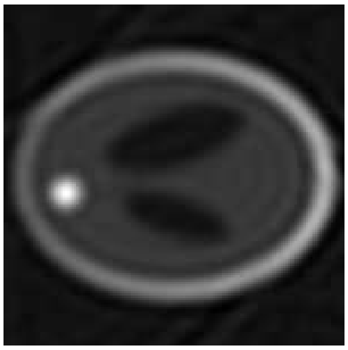


phantom 2

150 x 150 pixels, the intergration is done in (1) with an equidistant lattice of 82 x 82 points, in (2) with fibonacci numbers, 6765 points, and in (3) with 6765 randomly choosen points. The picture is drawn on $[0,1]$. The point (0,0) corresponds to the point left above, (1,1) to right down.

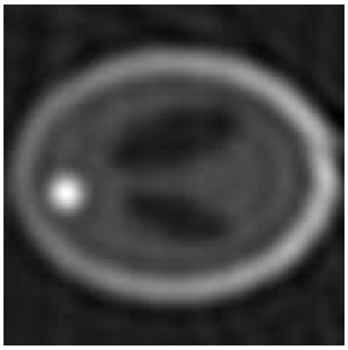
Further, there are presented pictures, reconstructed by equidistant points, random choosen points and Fibonacci points. We took the choice of the best band parameter. Thereas the band parameter was assumed to be eleven. Between 11 and 15, the error is minimized.

(1)



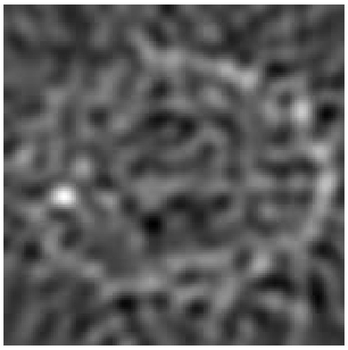
error: 39.2428
band: 15

(2)

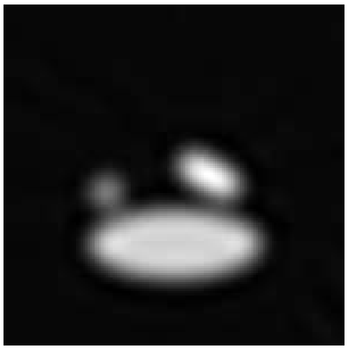


error: 51.2055
band: 12

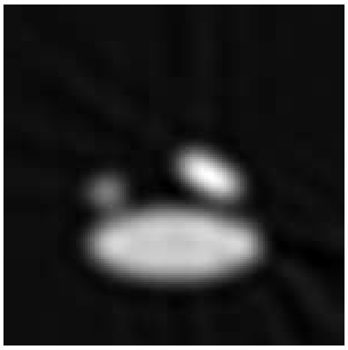
(3)



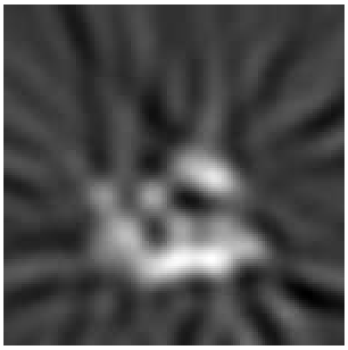
error: 74.7663
band: 16



error: 24.571465
band: 11



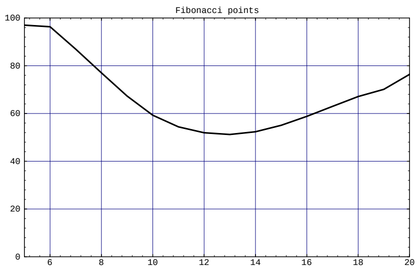
error: 23.92083
band: 11

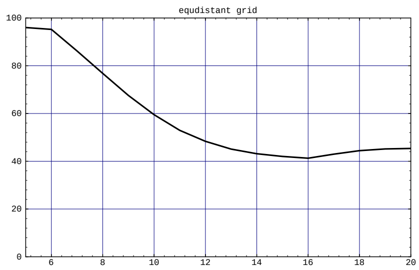


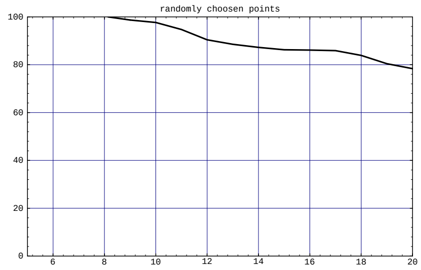
error: 61.19316
band: 11

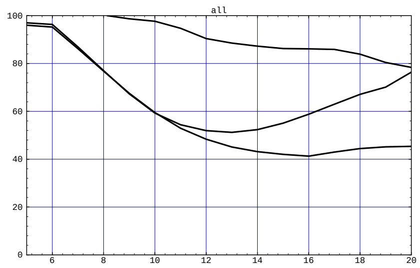
The error of the randomly choosen points is about the double of the others. If you look at the corresponding pictures you will notice, that to make out the object reconstructed by randomly choosen points is much more difficult than the others. Further, we have pointed out in section 2.3, that the error of Fibonacci points and equidistant points differs not much by pictures of superimposed ellipses. In the first example, the pictures reconstruct by equidistant points is better, in the second worse. The error function is drawn in the following pictures:

The error function of phantom 1:

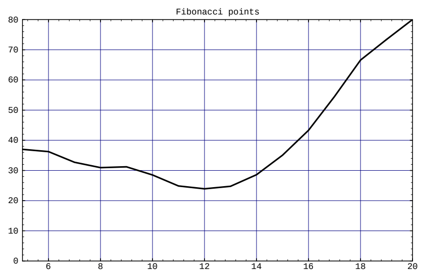


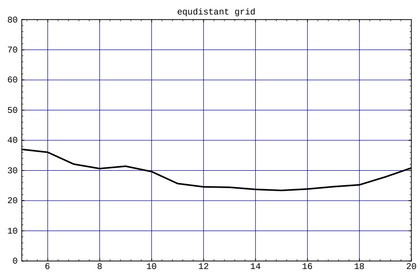


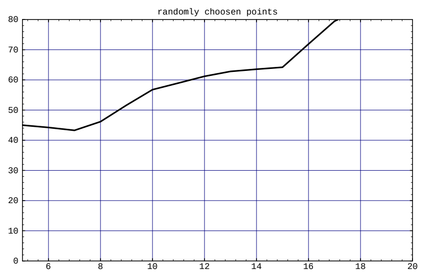


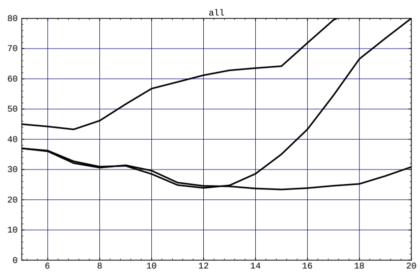


The error function of phantom 2:

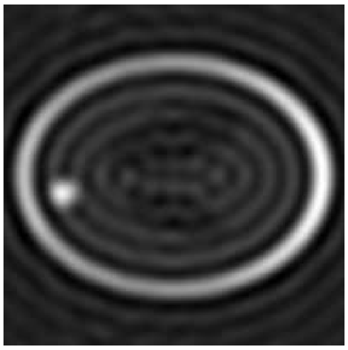


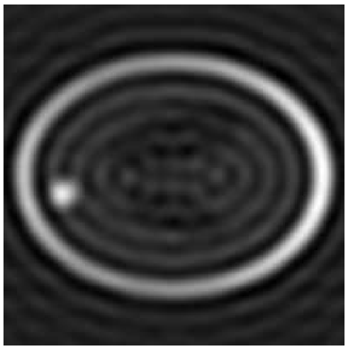


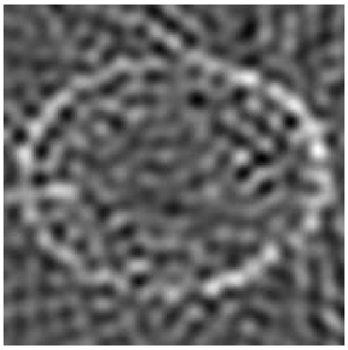




The next pictures are reconstructed by the algorithm of the PhThesis [9]. We have seen in chapter 2.2 the corresponding filter function. Because the filter function is not such a good appoximation of the δ function like the other, the pictures are much more blury. Observe, that the artefacts caused by the convolution not vanish. Thus to make out the original picture is much more difficult than at the first sequence.


band: 45


band: 45


band: 50

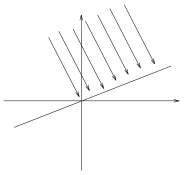
5.1.2 The Case of Failing some Points

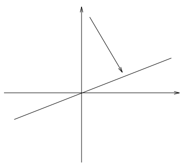
By physical damage, it can happen that some measurement points fail. We simulate this case by setting some points of the radon transform to zero.

1. First, we set all points of one angle to zero. This could be regarded as the failure of all beams at one fixed angle. We took two different angles: 10 and 30 degrees (pixels 150×150, 82×82 points and b=13). The picture is presented below. The same event cannot happen by Fibonacci numbers. Thus, to simulate an equivalent event, we erase 82 points of the radon transform (pixels 150×150, 8765 points and $b = 13$).

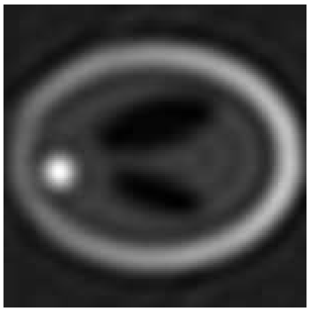
2. Here only one point is set to zero. You can see the picture also in appendix 3. We took the same choice of the number of pixels, points and band parameter. In order to compare the robustness of the two algorithms, we have done the same with a picture, reconstructed by Fibonacci numbers.

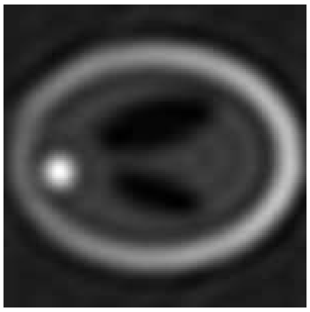
The first two picture shows, what happen, if we set the radon transform of one point, e.g. all points corresponding to a fixed angle, to zero:



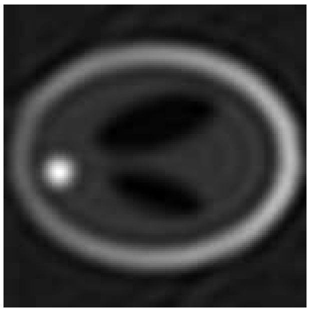


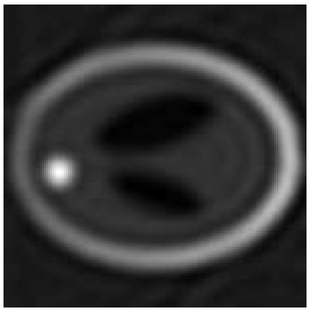
The following pictures are reconstructed by equidistant points:





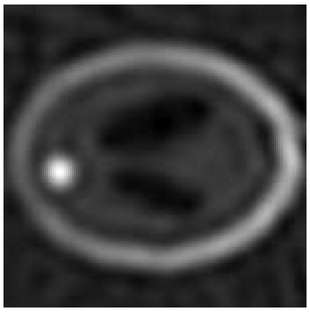
150×150 pixels, 82×82 points of radon transform, band: 11
(1): all values of angle 30 degrees fail
(2): all values of angle 0 degrees fail



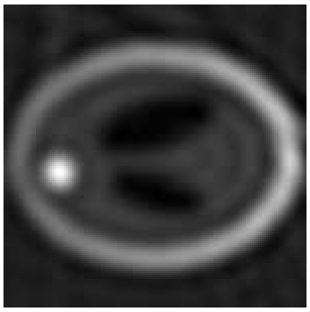


150×150 pixels, 82×82 points of radon transform, band: 13
(1): the point $(30^0, 0)$ fails
(2): the point $(00^0, 0)$ fails

The following pictures are reconstructed by fibbonacci points:



(1)



(2)

150×150 pixels, 6765 points of radon transform
(1): 84 values are set to zero, band:12
(2): only one point fails, band: 11

5.2 Conclusions

In a subsequent paper we want to investigate the algorithm by differentiable function. In the case of superimposed ellipses we can only outline the following two advantages:

1. It can happen, that it is necessary to add some points to improve the error rate. Using good lattice, you can add points one by one. This means, if you have already n measure points, you can add m other ones, where the m can be choosen arbitrary. If the points are equidistant and you want $n+m$ points, you have to choose between three possibilities: to cast the old points and to measure new $n+m$ points, to have a grid, which consists no more of equidistant points, or to go by steps of order n^2. To choose the first alternative is to waste a lot of time. If you choose the second, there will appear artefacts because the domain of integration is not covered uniformly. Thus the third alternative is the only left[5].

2. If some points fail because of a physical damage or something else, the algorithm using good lattice points yields to a lower error than the algorithm using equidistant points. Thus, the good lattice points lead to an algorithm which is much more robust than the other. The cause is the absence of a regular structure. Looking at a grid of equidistance points, you will note, that you can rotate it only in angles of $\frac{\pi}{4}$, $\frac{\pi}{2}$, $\frac{3\pi}{4}$, π. If you single out a circle of a lattice, generated by randomly choosen points, you will not observe a significant difference, rotationg the circle in any angle you want. The circles are all symmetric. Thus, one point fails, there is no regularity or high structure destroyed.

Acknowledgment: I am gratefull to Andreas Uhl for his assistance and advices in UNIX, printing images and **PVM**. Further, I have to thank Michael Revers for his help.

[5]If you use Fibonacci points, you have also to determine the points a second time, but you can raise up the number of points by smaller steps.

6 References

References

[1] Natterer, Frank: *The Mathematics of Computerized Tomography,* John Wiley and Sons (1985)

[2] Geist, A.: *PVM3 beyond network computation* In J. Volkert, editor Parallel Computing, Lectures Notes in Computer Sience 73, p. 194-203, Springer (1993)

[3] Geist, A. and V.S. Sunderam: *Network based on concurrent computing on the PVM system: Concurrency:* Practice and Experience 4(4), p.293-3115 (1992)

[4] Hausenblas, Erika: *The Salzburg NTN-Method fot the Radon Transform.* (1994) Deliverable D5Z-2.

[5] Helgason, S.:*The Radon Transform*(1980) Birkhaeuser Berlin

[6] Hlawka, E.: *Funktionen von beschraenkter Variation in der Theorie der Gleichverteilung*, Annali di Mathematica,(1961)

[7] Hlawka, E.: *Theorie der Gleichverteilung*, Annali di Mathematica,(1979)

[8] Radon, Johann: *Ueber die Bestimmung von Funktionen durch ihre Intergralwerte laengs gewisser Mannigfaltigkeiten* (1917)Berichte der Saechsische Akademie der Wissenschaften, Leipzig, Math.-Phys., p. 262-267,69

[9] Revers, Michael: *Anwendung zahlentheoretischer Methoden fuer die Numerik der Computertomographie*, PhThesis,(1990), Salzburg

[10] Revers, Michael: *Numeric Integration of the Radon Transform on Classes* E_s^α *in multiple (finite) dimensions* (1994), preprint

[11] Sanz, Jorge L.C. and Eric B. Hinkle and Anil K.Jain: *Radon and Projection Transform-Based Computer Vision*, Springer Series in Information Sciences (1988)