UNIVERSIDADE FEDERAL DE PERNAMBUCO - UFPE
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS - CTG
CENTRO DE INFORMÁTICA - CIN
GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

Reconstrução de Imagens Tomográficas com uso de GPU

TRABALHO DE GRADUAÇÃO
2012.2

Luiz Gustavo da Rocha Charamba
lgrc@cin.ufpe.br

Recife, 10 de Maio de 2013.

UNIVERSIDADE FEDERAL DE PERNAMBUCO - UFPE
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS - CTG
CENTRO DE INFORMÁTICA - CIN
GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

Luiz Gustavo da Rocha Charamba

Reconstrução de Imagens Tomográficas com uso de GPU

Trabalho apresentado ao Programa de GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO do CENTRO DE INFORMÁTICA da UNIVERSIDADE FEDERAL DE PERNAMBUCO *como requisito parcial para obtenção do grau de Bacharel em* ENGENHARIA DA COMPUTAÇÃO.

Orientador: *Prof. Dr. Silvio de Barros Melo*

Recife, 10 de Maio de 2013.

“Compreender que há outros pontos de vista é o início da sabedoria.”
Thomas Campbell

Resumo

Este trabalho mostra algumas técnicas de reconstrução de imagens tomográficas e seus fundamentos teóricos, além do algoritmo da Transformada de Radon. Os algoritmos foram implementados para serem executados tanto em CPU quanto em GPU através da biblioteca CUDA na linguagem C. O intuito é mostrar os ganhos de performances conquistados com a GPU e o modelo de programação que explora o paralelismo computacional.

Abstract

This work shows some techniques for tomographic image reconstruction and its theoretical foundations, and the algorithm of the Radon Transform. The algorithms have been implemented to run on both CPU as the GPU through the CUDA library in C. The aim is to show the performance gains achieved with the GPU and the programming model that exploits the parallel computing.

Agradecimentos

À minha Mãe, Maria das Graças, que sem ela não teria chegado tão longe, e foi minha grande incentivadora no caminho do estudo e do conhecimento com seriedade e empenho.

Ao meu orientador, Prof. Silvio de Barros Melo, que me conduziu nesse trabalho, e também por ser um Professor raro na forma de ensinar e possuidor de grandes conhecimentos.

Ao Prof. Carlos Costa Dantas do Depto. de Energia Nuclear (DEN) da UFPE com o suporte que me foi dado no grupo de Tomografia e Radioquímica, para que esse trabalho fosse realizado.

Aos amigos do DEN que me ajudaram compartilhando conhecimentos preciosos. Eric Ferreira em processamento de imagens, e Alex Moura no funcionamento do tomógrafo implementado no departamento.

Aos colegas de curso pelo o apoio mútuo e formação de grandes amizades. Aos amigos e familiares que torcem pelo meu sucesso, e mesmo assistindo de longe ainda assim compreendem o drama da graduação e suas renuncias.

À minha namorada, Gabriela Albuquerque, pela sua doce companhia, paciência e apoio, e que sempre me incentivou nos momentos mais difíceis dessa jornada.

Sumário

1. Introdução

A tomografia é uma técnica de grande sucesso e bastante utilizada em áreas bem distintas como na medicina, astronomia, geologia e até na indústria. O surgimento da tomografia remonta à descoberta dos raios-x pelo físico alemão Wilhelm Röntgen em 1895, o que mais tarde lhe rendeu o prêmio Nobel de física em 1901 [4]. A emissão de raios-x foi um sucesso na medicina, era então possível fazer análises dos órgãos internos, de fraturas ósseas e descoberta de tumores através da técnica conhecida no meio médico como radiografia. Numa radiografia os raios-x atravessam o corpo ou um membro de um paciente e incidem com mais ou menos intensidade num anteparo que pode ser um filme sensível à radiação ionizante, de acordo com a distribuição de densidade, partes como pele e carne são pouco densas e os raios sofrem menos atenuações e chegam com maiores intensidades no anteparo, já as partes mais densas como os ossos, os raios sofrem mais atenuações e chegam com menores intensidades no anteparo. O resultado de uma radiografia é uma sombra projetiva devido à incidência de raios-x, onde cada ponto da imagem é uma densidade média da trajetória do raio, isso revela a limitação da técnica para alguns diagnósticos, pois objetos mais densos, como ossos, podem ocultar objetos pouco densos como eventuais tumores. A limitação da radiografia se resume a ser uma técnica que faz uma representação bidimensional de um objeto tridimensional.

Mais tarde na década de 1930, o radiologista Italiano Allessandro Vallebona desenvolve uma técnica que foi chamada por ele de estratigrafia [5], conhecida hoje como tomografia axial. A estratigrafia consistia em realizar radiografias de finas fatias do objeto em diferentes ângulos [6]. A técnica contornava os problemas da seção transversal obtida pela radiografia convencional e por muito tempo foi utilizada, porém a estratigrafia ou tomografia analógica não era de fácil interpretação e era usada em casos bem específicos. A técnica de Vallebona oferecia o problema de reconstrução da imagem para melhor entendimento, tal problema só seria superado anos mais tarde com o advento da eletrônica digital e a computação. A palavra tomografia vem do grego *tomo*, que significa parte e *grafia* que significa descrição ou registro.

Com o avanço da microeletrônica e computação na década de 70, o engenheiro Godfrey Hounsfield e o biólogo Allan M. Cormack propuseram um modelo matemático que se baseava numa transformada pouco conhecida, a Transformada de Radon, que foi primeiramente inserida pelo matemático austríaco Johann Radon em 1917. Hounsfield e Cormack deram então o início no que ficou conhecido como Tomografia Computadorizada (TC), seus esforços foram recompensados com o prêmio Nobel de medicina em 1979 [7]. A tomografia causou grande impacto na medicina, e hoje a técnica é bastante utilizada em diversos outros segmentos.

O processo de tomografia emula a Transformada de Radon, e o resultado gera um conjunto de dados correspondentes aos valores das atenuações dos raios formando as projeções em torno do objeto. Esse conjunto de dados é então processado, a quantidade massiva de dados exige dispositivos de processamento melhores para ter um tempo de resposta da reconstrução da imagem satisfatório. Nesse trabalho será exibida a solução do problema de reconstrução de imagens tomográficas por meio das unidades gráficas de processamento (G*raphic Processing Units, GPU*). As GPUs já são bastante populares, nasceram da necessidade de se ter unidades de processamento mais rápidas para os cálculos de geometria e renderização dos polígonos nos jogos de PC, é o "coração" das modernas placas aceleradoras gráficas. O grande poder das GPUs se dá pela grande quantidade de núcleos de processamento operando concorrentemente, essa arquitetura dá suporte ao paralelismo computacional real. Hoje as GPUs são usadas para diversos outros propósitos, a Nvidia, uma

das principais fabricantes de GPUs, percebendo isso criou CUDA, que é uma plataforma de programação em paralelo com o uso das GPUs, possui suporte para as linguagens C e Fortran. A GPU vem se consolidando como o dispositivo de processamento em paralelo mais usado na atualidade, a GPU já deixou de ser apenas um dispositivo para pipeline gráfico para jogos 3D, e hoje atua também como dispositivo de propósito geral para a solução de problemas com quantidade massiva de dados paralelizáveis [2]. O problema de reconstrução de imagens tomográficas parte de um conjunto de dados massivos que podem ser processados de forma paralela, justificando assim o adequado uso da GPU.

Nesse trabalho foram implementados três algoritmos de reconstrução de imagens tomográficas e a Transformada de Radon, para simular o processo tomográfico.

Todos esses algoritmos terão versões em CPU e GPU (com a exceção do ART que só foi feito para a CPU), no intuito de se comparar os desempenhos dos métodos nos dois dispositivos e também de mostrar as diferentes abordagens e problemas detectados na implementação em GPU, que será feito usando CUDA na linguagem C. As três técnicas de reconstrução abordadas são: o método da retroprojeção (*backprojection, BP*), o método algébrico (*algebraic reconstruction technique, ART*) e o método da retroprojeção filtrada (*filtered backprojection, FBP*), o ART só foi desenvolvido na versão que roda em CPU. Também será discutida a proposta do uso de cada método. Os hardwares utilizados foram a CPU Intel Core i5 M450 2.40 GHz, e a GPU Nvidia GeForce 310M 606/625 MHz.

Palavras Chave: Tomografia, Reconstrução de imagens, Retroprojeção, *Backprojection*, CUDA, GPU.

2. Fundamentos da Tomografia e Transformada de Radon

O processo de tomografia pode ser realizado através de amostragens paralelas de feixes de raios x ou raios gama, em diferentes ângulos em um determinado objeto de análise. Tais feixes de raios sofrem atenuações de acordo com a distribuição de densidade do objeto, todo esse conjunto de perfis é o resultado da transformada de Radon do objeto em análise [1]. Os dados resultantes da transformada de Radon são chamados de Sinograma.

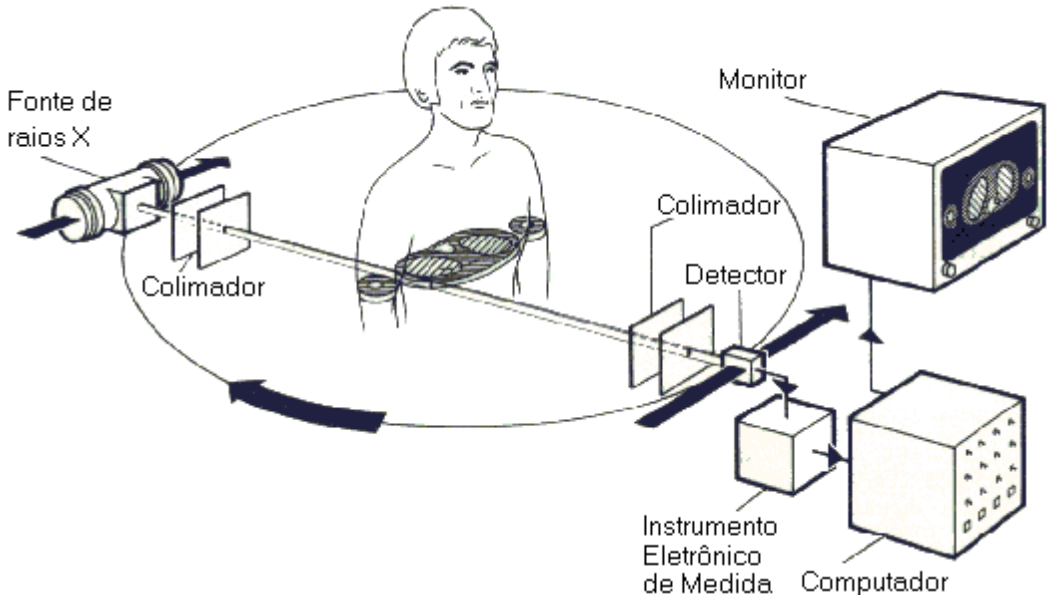


Figura 2.1 - Ilustração do processo de tomografia, par emissor-detector rotaciona e translada realizando cortes em diferentes ângulos e trajetórias paralelas em torno do objeto.

2.1. PRINCÍPIOS FÍSICOS

A trajetória do raio ionizante que atravessa o objeto pode ser aproximada por uma reta, e a intensidade desse raio é atenuada ao longo de uma linha com coeficiente de atenuação μ. Abaixo temos um exemplo de objeto com distribuição de atenuação não homogênea vista ao longo de uma trajetória, os intervalos no eixo u indicam o tamanho dos segmentos.

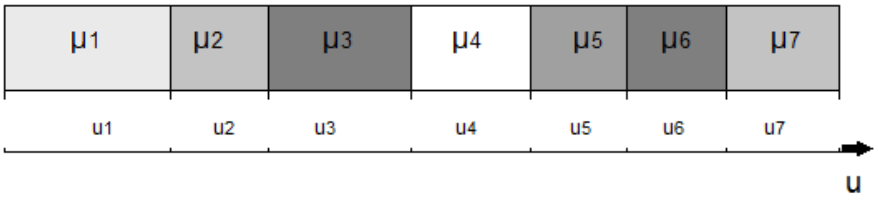


Figura 2.2 – Distribuição da atenuação ao longo de uma trajetória do raio em um objeto.

A atenuação da intensidade obedece a lei de decaimento exponencial de Beer-Lambert. Para um objeto com distribuição da atenuação uniforme μ_1, e tamanho do segmento percorrido u_1, tem-se:

$$\frac{I}{I_0} = e^{-u_1\mu_1}$$

(2.1)

Aonde I_0 é a intensidade máxima do raio ionizante, e I é a intensidade final do raio após atravessar o objeto. Para um objeto com distribuição não uniforme, similar ao objeto da figura 2.2, tem-se:

$$\frac{I}{I_0} = e^{-\sum_{i=1}^{n} u_i\mu_i}$$

(2.2)

Para se obter o análogo contínuo da equação 2.2, basta substituir o somatório dos elementos discretos pela integral, e tornar u como elemento de linha du, assim tem-se:

$$\frac{I}{I_0} = e^{-\int_a^b \mu(x,y)\, du}$$

(2.3)

Foi visto até então, como se obter os valores de intensidade para as trajetórias, agora há a necessidade de uma modelagem geométrica do problema e parâmetros que possam indicar a disposição desses raios.

2.2. MODELAGEM MATEMÁTICA

Em um tomógrafo de primeira geração, seu arranjo geométrico possui configuração que faz uso das projeções paralelas, cada trajetória de raio ionizante é parametrizado através da distância normal s à origem do sistema de coordenadas cartesiano e o ângulo θ da linha relativa ao primeiro eixo. Abaixo, na figura 2.2, uma ilustração de uma seção de um objeto sendo atravessado por uma trajetória do raio ionizante do tomógrafo.

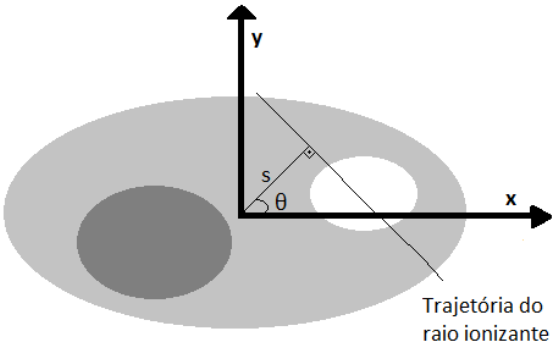


Figura 2.3 – A trajetória pode ser descrita pelos parâmetros s e θ.

Um conjunto de dos valores das intensidades das trajetórias com mesmo ângulo formam o que é chamado de projeção. A equação 2.3 pode ser reescrita da seguinte forma:

$$\frac{I(s,\theta)}{I_0} = e^{-\int_a^b \mu(x,y)\,du}$$

(2.4)

Dessa forma a intensidade está relacionada aos parâmetros que identificam a trajetória.

A projeção $g(s,\theta)$ é definida como:

$$g(s,\theta) = ln\frac{I(s,\theta)}{I_0}$$

(2.5)

Assim, da equação 2.4, tem-se que:

$$g(s,\theta) = \int_{-\infty}^{\infty} \mu(x,y)\,du = \int_{-\infty}^{\infty} \mu(s\cos\theta - u\,sen\theta\ , s\,sen\theta + u\,cos\theta)\,du$$

(2.6)

A função de $g(s,\theta)$ é uma integral de linha em du de $\mu(x,y)$, e também é a Transformada de Radon de $\mu(x,y)$.

2.2.1. Transformada de Radon

A transformada de Radon converte uma função de um sistema de coordenadas espaciais (x,y) para outro sistema de coordenadas espaciais de Radon. O domínio de Radon descreve os valores das integrais de linha como da equação 2.6, através dos parâmetros de (s,θ). A equação de linha é expressa por $xcos\theta + y\,sen\theta - s = 0$. A expressão da linha pode também ser escrita na forma de reta paramétrica, onde é usado o parâmetro u. Abaixo, um exemplo de projeção, com uso desse parâmetro:

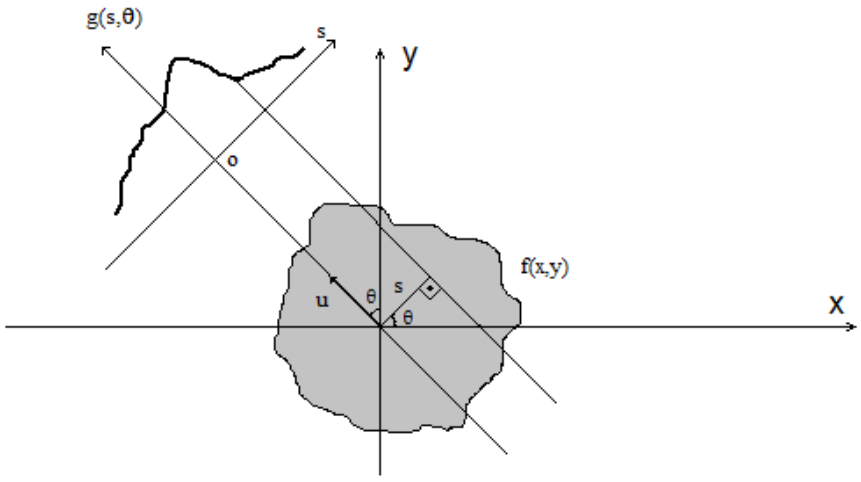


Figura 2.4 – A trajetória pode ser parametrizada pelo vetor unitário u de mesma direção.

Dessa forma a expressão que define a linha fica:

$$x = s\,cos\theta - u\,sen\,\theta$$
$$y = s\,sen\theta + u\,cos\,\theta \tag{2.7}$$

Também é possível deixar em função de x e y:

$$s = x\,cos\theta + y\,sen\,\theta$$
$$u = -x\,sen\theta + y\,cos\,\theta \tag{2.8}$$

Assim, da equação 2.6 e 2.7, tem-se a Transformada de Radon para a função $f(x,y)$ como:

$$g(s,\theta) = \int_{-\infty}^{\infty} f(s\cos\theta - u\,sen\theta\ , s\,sen\theta + u\,cos\theta)\,du \tag{2.9}$$

A projeção $g(s,\theta)$ é obtida por um conjunto de integrais de linha com mesmo ângulo θ. Abaixo, projeções nos ângulos θ_1 e θ_2:

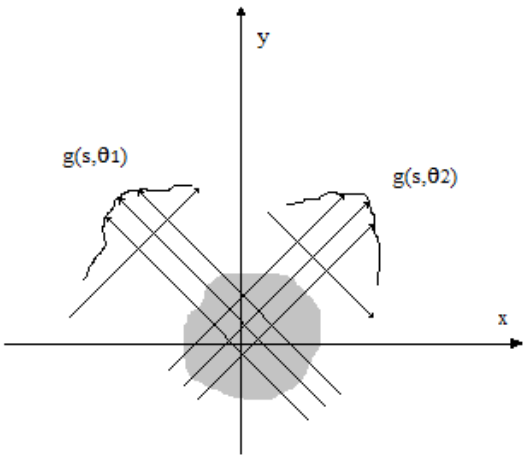


Figura 2.5 – Projeções em dois diferentes ângulos.

A composição de diversas projeções forma o que é chamado de Sinograma, que é obtido no processo de tomografia, e esse conjunto de dados servem de ponto de partida para a reconstrução da imagem. A Transformada de Radon tem fundamental importância nesse processo, a tomografia nada mais é que a transformada de Radon aplicada.

2.3. ALGORITMO DA TRANSFORMADA DE RADON

Após o entendimento da transformada de Radon, pode-se então elaborar um algoritmo que executa o cálculo das projeções de uma imagem de teste, essas imagens são conhecidas na literatura como Fantomas. É necessário se fazer algumas considerações relacionadas as amostragens. O número de projeções pode ser variável, dependendo do número de projeções desejado, os ângulos deverão fica espaçados em intervalos iguais, o ângulo θ varia de 0 a 180 graus. O mesmo vale para s, porém s varia da máxima distância negativa da imagem até a máxima distância da imagem de interesse, no caso seria metade da diagonal negativa até a metade da diagonal positiva. A imagem abaixo pode ajudar na compreensão:

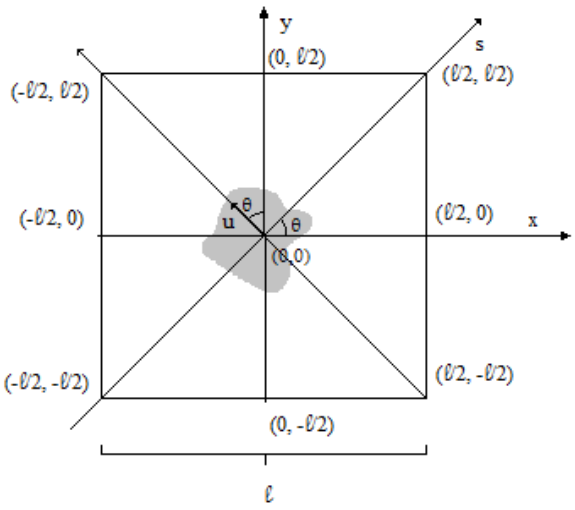


Figura 2.7 – Imagem quadrada de tamanho de lado l.

O método deve ser capaz de gerar todos os valores de pixel (s, θ) do Sinograma. Para isso serão usadas as equações de 2.7 que estão em função de s, θ e u. Abaixo o pseudocódigo da Transformada de Radon, o resultado desse processo gera o Sinograma da imagem de entrada.

Para todo (s, θ) faça:

Para todo u faça:

$x = s\ cos\theta - u\ sen\theta$

$y = s\ sen\theta + u\ cos\theta$

$g(s, \theta) = g(s, \theta) + f(x, y)$

fim

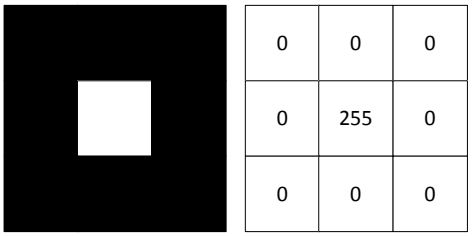
fim

O parâmetro u pode ter intervalos menores, no intuito de aumentar a contribuição naqueles pixels em que o segmento de linha da trajetória é maior. A função $f(x, y)$ é o Fantoma e $g(s, \theta)$ representa o Sinograma.

3. Método da Retroprojeção

O método da retroprojeção (*Backprojection, BP*) é o método mais simples de reconstrução, esse método consiste em redistribuir em cada ponto pertencente a linha da trajetória os valores dos somatórios das densidades das projeções. Se tratando de imagens, a retroprojeção irá distribuir valores de pixels do Sinograma aos pixels que contribuíram na trajetória da imagem em reconstrução.

Exemplo: Dado uma imagem 3x3 com pixel central branco e demais pixels pretos, como na figura abaixo:



0	0	0
0	255	0
0	0	0

Figura 3.1 – A matriz da direita é a representação numérica dos valores de pixel do Fantoma da esquerda.

Realizando a Transformada de Radon na imagem da figura 3.1 com configuração de amostragem de três trajetórias por projeção, e dois diferentes ângulos de projeção, um de 0 e outro de 90 graus. Obtém–se então um Sinograma de duas projeções com os seguintes valores abaixo:

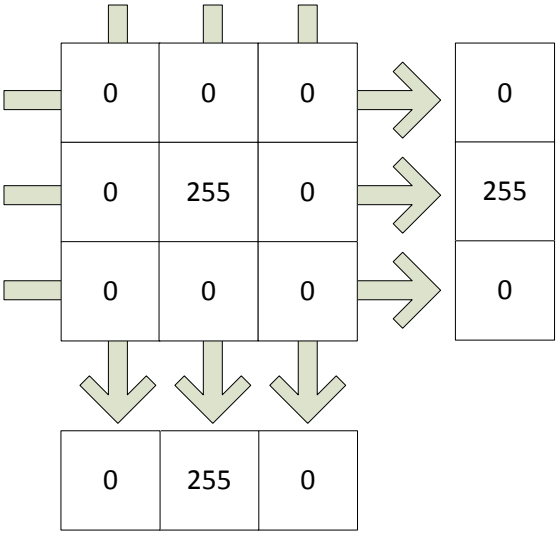


Figura 3.2 – Projeções obtidas em ângulos de 0 e 90 graus do Fantoma do exemplo.

A retroprojeção irá fazer o caminho inverso, irá redistribuir os valores das projeções aos seus respectivos pixels contribuintes, da seguinte maneira.

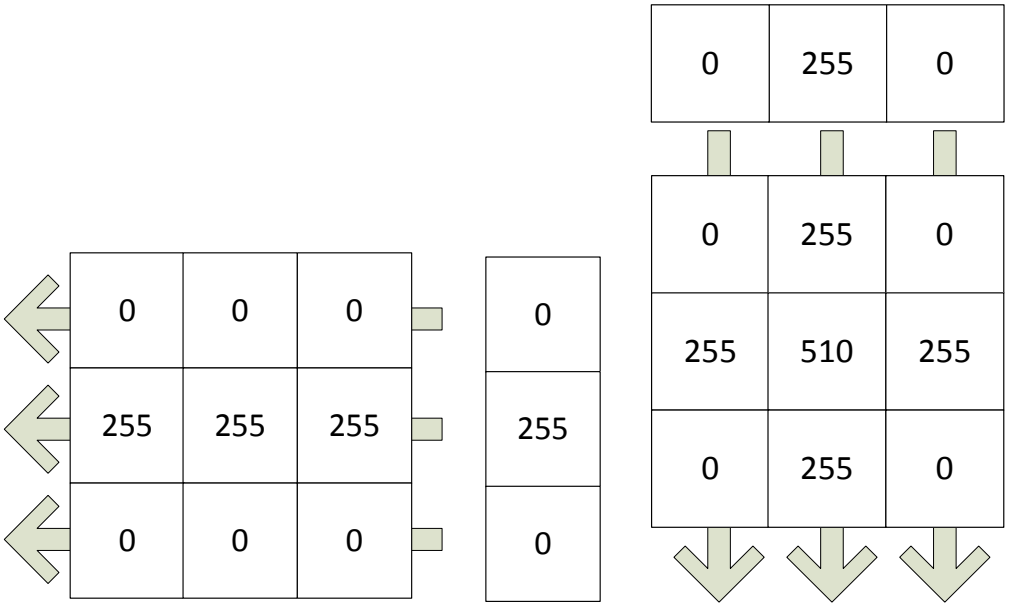


Figura 3.3 – Primeira e segunda retroprojeção.

Os valores dos pixels irão sendo acrescidos com os valores das projeções, a cada retroprojeção há um incremento. Finalizando o exemplo, a imagem reconstruída a partir de apenas duas projeções gera uma imagem com valor máximo de 510. Normalizando esses valores para um range de 0 a 255, obtemos a imagem abaixo.

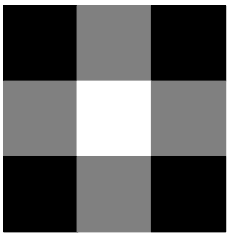


Figura 3.3 – Imagem reconstruída a partir de Sinograma de duas projeções.

3.1. ALGORITMO

Partindo da equação 2.8 onde se estima o valor de s, uma das coordenadas das linhas das trajetórias através das variáveis x, y e θ, é possível varrer todos os pixels (x, y) da imagem a ser reconstruída e contemplá-los.

Para todo (x,y) faça:

Para todo θ *faça:*

$s = x\cos\theta + y\,sen\theta$

$f\_rec(x,y) = f\_rec(x,y) + g(s, \theta)$

fim

fim

4. Método da Retroprojeção Filtrada

O método de reconstrução por Retroprojeção Filtrada (*Filtered Backprojection, FBP*) é o método de reconstrução atualmente usado em equipamentos de tomografia na área médica. O método é indicado em casos onde existe um grande número de amostragens durante o processo tomográfico. Para entender melhor esse método é necessário compreender o de teorema de *Fourier Slice*, ou Fatia de Fourier.

4.1. TEOREMA DE FOURIER SLICE

Curvas podem ser analisadas através da Transformada de Fourier, pela decomposição dela em funções senoidais e cossenoidais com diferentes frequências e amplitudes específicas. Uma imagem é uma função bidimensional e que pode ser analisada no domínio de Fourier. O Teorema de Fourier Slice faz uma relação com a Transformada de Radon e com a Transformada de Fourier, foi visto anteriormente que o processo de tomografia, nada mais é que a Transformada de Radon. O Teorema de Fourier Slice afirma que calcular a Transformada bidimensional de Fourier é o mesmo que calcular a transformada de Radon e em seguida calcular a transformada de Fourier da projeção em relação a variável s, os diagramas abaixo ilustram essa sequência:

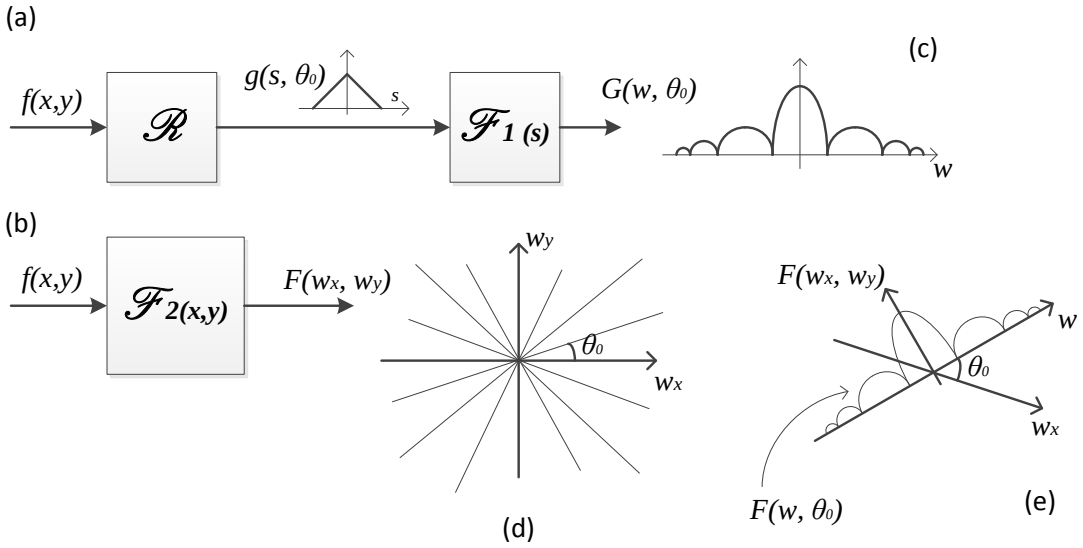


Figura 4.1 – Teorema de Fourier Slice, $G(w, \theta_0) = F(w, \theta_0)$.

No sistema em (a) da figura 4.1, o primeiro bloco realiza a Transformada de Radon, e o segundo realiza a Transformada de Fourier em s. O gráfico em (c) é o resultado da projeção no domínio de Fourier da projeção em um ângulo θ_0. O sistema em (b) realiza a Transformada de Fourier em x e y. O domínio ilustrado em (d) é o domínio de Fourier em duas dimensões, as linhas radiais são onde estão dispostas curvas iguais as projeções obtidas em (c). O gráfico em (e) mostra uma dessas fatias em um ângulo θ_0.

Escrevendo as sequências de operações dos dois sistemas equivalentes em termos de operadores, tem-se:

$$\mathscr{F}_2 = \mathscr{R}\,\mathscr{F}_1 \tag{4.1}$$

É possível expressar a função $f(x,y)$, em termos dos operadores de Transformada de Radon e Fourier direto e inverso, dessa forma:

$$f(x,y) = \mathscr{F}_2^{-1}\mathscr{R}\,\mathscr{F}_1\, f(x,y) \tag{4.2}$$

Lembrando que $g(s,\theta)$ é o resultado da Transformada de Radon de $f(x,y)$, tem-se:

$$g(s,\theta) = \mathscr{R}\, f(x,y) \tag{4.3}$$

A partir da equação 4.2 e 4.3 pode-se encontrar uma equação que expresse a Transformada inversa de Radon de $g(s,\theta)$, e obter uma reconstrução:

$$\mathscr{R}^{-1} g(s,\theta) = \mathscr{F}_2^{-1}\mathscr{F}_1\, g(s,\theta) \tag{4.4}$$

A partir da equação 4.4, em [9] o teorema de Fourier Slice foi enunciado como:

"A transformada de Fourier de uma projeção paralela de uma imagem $f(x,y)$ calculada em um dado ângulo θ resulta em um "slice" da transformada de Fourier 2D, $F(w_x, w_y)$, subentendido um ângulo θ com o eixo w_x. Em outras palavras, a transformada de Fourier de $g_\theta(s)$ produz os valores de $F(w_x, w_y)$ ao longo da linha BB, como ilustrado na Figura 4.2".

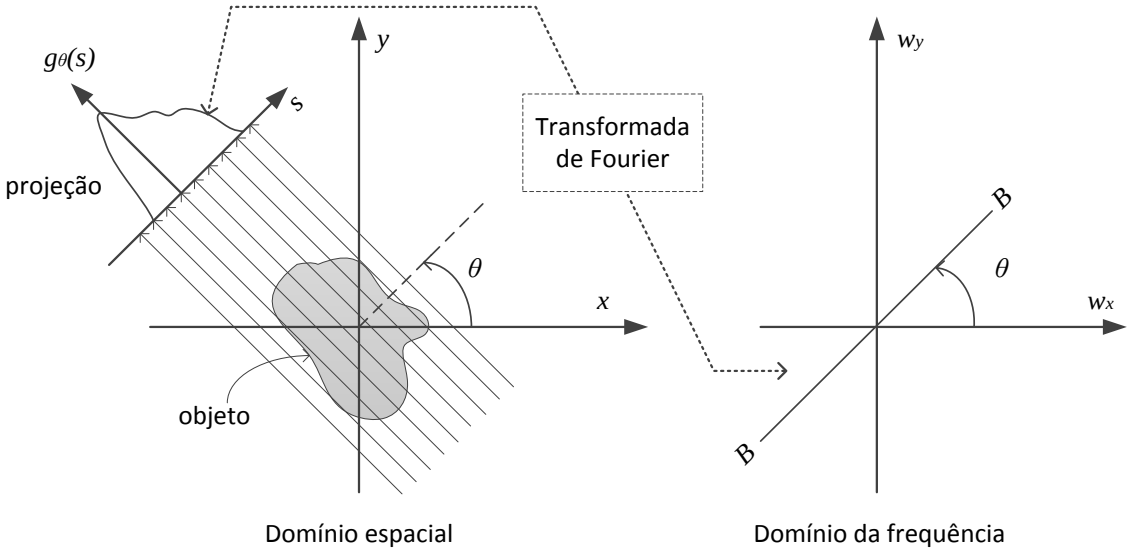


Figura 4.2 – Teorema de Fourier Slice.

A equação 4.2 fornece a reconstrução de uma imagem através da Transformada de Fourier das projeções seguida da Transformada de Fourier inversa bidimensional. Essa abordagem permite que haja uma filtragem nas projeções. Essa técnica é conhecida como método da Retroprojeção Filtrada. O objetivo da filtragem seria eliminar o problema de clareamento de fundo que é obtida na retroprojeção.

Existem três variações da Retroprojeção filtrada discutidas em [8], dependendo da forma de filtragem utilizada, estas variações são: filtragem de Fourier, filtragem no domínio de Radon e filtragem utilizando a convolução. A seguir descrevem-se as abordagens de filtragem no domínio de Fourier e por convolução.

4.2. FILTRAGEM NO DOMÍNIO DE FOURIER

A transformada de Fourier de uma função de duas variáveis *f(x,y)*, pode ser calculada a partir da Integral dupla abaixo:

$$F(w_x, w_y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-2\pi j(w_x x + w_y y)}dx\,dy \tag{4.5}$$

É possível também fazer o mesmo com a projeção $g_\theta(s)$, com um ângulo θ já conhecido através da Transformada de Fourier em uma dimensão.

$$G_\theta(w) = \int_{-\infty}^{\infty} g_\theta(s)\, e^{-2\pi i w s} ds \tag{4.6}$$

Foi visto no Teorema de Fourier Slice que a projeção $g(s, \theta)$ no domínio de Fourier é igual a as linhas radiais contidas no domínio de Fourier, ou seja, $G(w, \theta_0) = F(w, \theta_0)$ para um ângulo θ_0.

A operação de filtragem no domínio da frequência consiste em multiplicação de um filtro $H(w)$ com uma função $G(w)$, é possível então filtrar as projeções através dessa simples operação. O esquemático mostra o fluxo de operações para a filtragem e retroprojeção no domínio de Fourier.

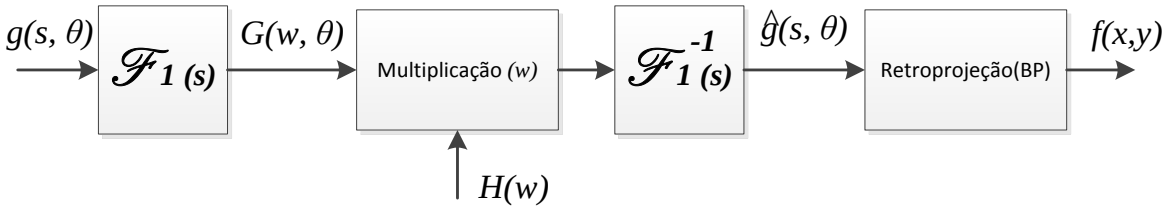


Figura 4.3 – Filtragem no domínio de Fourier.

Existe uma infinidade de filtros que podem ser usados no processo. Lembrando que a filtragem deseja eliminar as baixas frequências que são responsáveis pelo o efeito de névoa esbranquiçada causado na retroprojeção. Alguns filtro comumente utilizados para isso são os filtros de Ram-Lak e Shepp-Logan. Nesse trabalho foi utilizado o filtro de Shepp-Logan.

4.2.1. Filtro de Shepp-Logan no domínio da frequência

O filtro de Shepp-Logan no domínio da frequência elimina a componente de frequência nula e atenua as baixas.

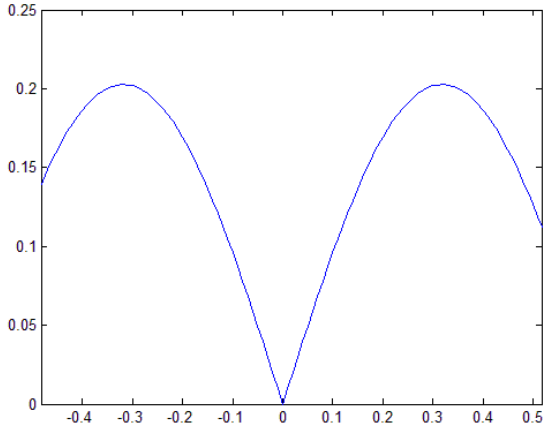


Figura 4.4 – Filtro de Shepp-Logan no domínio da frequência.

O filtro de Shepp-Logan no domínio da frequência é definido pela equação abaixo:

$$H(w) = |w| sinc\left(w\frac{\pi}{2}\right), \qquad 0 < w \leq 1 \tag{4.7}$$

4.3. FILTRAGEM POR CONVOLUÇÃO

O método da filtragem por convolução foi derivado por Bracewell e Riddle em 1967, e foi largamente usado em radiografia. Leddley, Di Chiro, Luessenhop e Twigg usaram o método pela primeira vez em um equipamento de tomografia em 1974 [1]. A operação de filtragem neste método é realizada no domínio de Radon, convoluindo as projeções em *s*. Foi visto que no domínio da frequência as projeções são filtradas com uma multiplicação, de forma equivalente no domínio espacial, as projeções podem ser filtradas convoluindo com o mesmo filtro agora expresso no domínio espacial. Abaixo um esquema que mostra a filtragem por convolução.

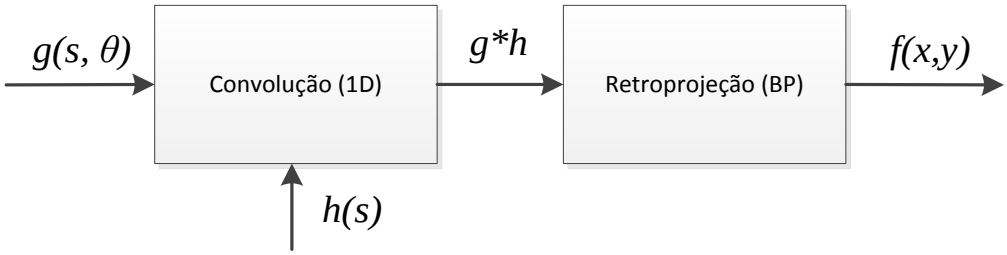


Figura 4.5 – Filtragem por convolução.

4.3.1. Filtro de Shepp-Logan no domínio espacial

O filtro de Shepp-Logan no domínio espacial possui a seguinte forma.

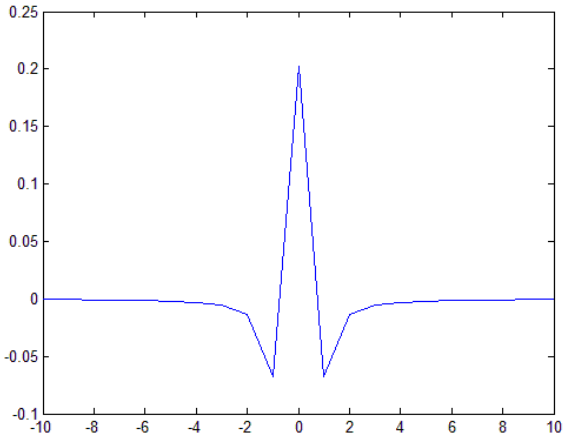


Figura 4.6 – Filtro de Shepp-Logan no domínio espacial.

O filtro de Shepp-Logan no domínio espacial é definido pela equação abaixo:

$$h(s) = \frac{2}{\pi^2(1-4s^2)} \tag{4.8}$$

4.4. ALGORITMO

O método da Retroprojeção filtrada nesse trabalho foi feito pela abordagem de filtragem por convolução. Dado duas funções contínuas $g(s)$ e $h(s)$, a convolução dessas duas funções é dada por:

$$g * h(s) = \int_{-\infty}^{\infty} g(S)h(s - S)dS \tag{4.9}$$

O equivalente discreto da convolução é dado por:

$$g * h(k) = \sum_{i=0}^{n} g(i)h(k - i) \tag{4.10}$$

O processo de reconstrução utilizando a convolução pode ser descrito como.

Passo 1 – Filtrar a projeção através da convolução usando a equação 4.10.

Passo 2 – Realizar a retroprojeção.

Passo 3 – Repetir os passos 1 e 2 para todas as projeções.

Devido ao aproveitamento e reuso de código, uma pequena modificação foi realizada, para fazer uso dos recursos já disponíveis.

Passo 1 – Filtrar através da convolução usando a equação 4.10, para todas as projeções.

Passo 2 – Realizar a retroprojeção para todas as projeções.

4.5. COMPLEXIDADE

A etapa de filtragem por convolução requer N operações aritméticas por valor de saída que dá N^2 operações aritméticas por N valores de saída, a complexidade então é $O(N^2)$. Enquanto a etapa de filtragem no domínio da frequência pode ser feita usando o algoritmo de FFT (*Fast Fourier Transform*) direta e inversa, a complexidade da filtragem por essa abordagem é O(*N log N)*. A técnica de filtragem escolhida para a implementação será usando a convolução, justamente por ter uma complexidade de algoritmo maior, o objetivo é calcular os tempos em circunstâncias onde há muitos cálculos e dados para serem processados.

5. Desenvolvimento

Nesse capítulo será mostrada a tecnologia que foi usada para programação em GPU, o desenvolvimento da Transformada de Radon e dos métodos de reconstrução até então discutidos, tanto na abordagem de execução tradicional em CPU e também na abordagem de execução em GPU usando CUDA.

5.1. PROGRAMAÇÃO EM CUDA

Compute *Unified Device Architecture* (CUDA) é uma plataforma de computação paralela de propósito geral que tira proveito das unidades de processamento gráfico (GPUs) NVIDIA, possuiu uma baixa curva de aprendizagem para programadores familiarizados com a linguagem C [10]. O modelo de programação em CUDA oferece três principais abstrações, tais como uma hierarquia de grupos de threads, memórias compartilhadas e sincronização de barreiras. Estas abstrações fornecem paralelismo de dados refinado e paralelismo de threads, aninhada dentro de paralelismo de dados de granulação grossa e paralelismo de tarefas. Eles guiam o programador para particionar o problema em grossos subproblemas que podem ser resolvidos de forma independente, em paralelo por blocos de threads, e cada subproblema em partes mais finas que podem ser resolvidos de forma cooperativa em paralelo por todas as threads dentro do bloco. Esta decomposição preserva a expressividade da linguagem por threads permitindo a cooperar para a resolução de cada subproblema, e ao mesmo tempo permite a escalabilidade automática. Com efeito, cada bloco de threads pode ser programado em qualquer um dos multiprocessadores disponíveis dentro de uma GPU, em qualquer ordem, simultaneamente ou sequencialmente, de modo que um programa compilado CUDA pode executar em qualquer número de multiprocessadores, tal como ilustrado na figura 5.1.

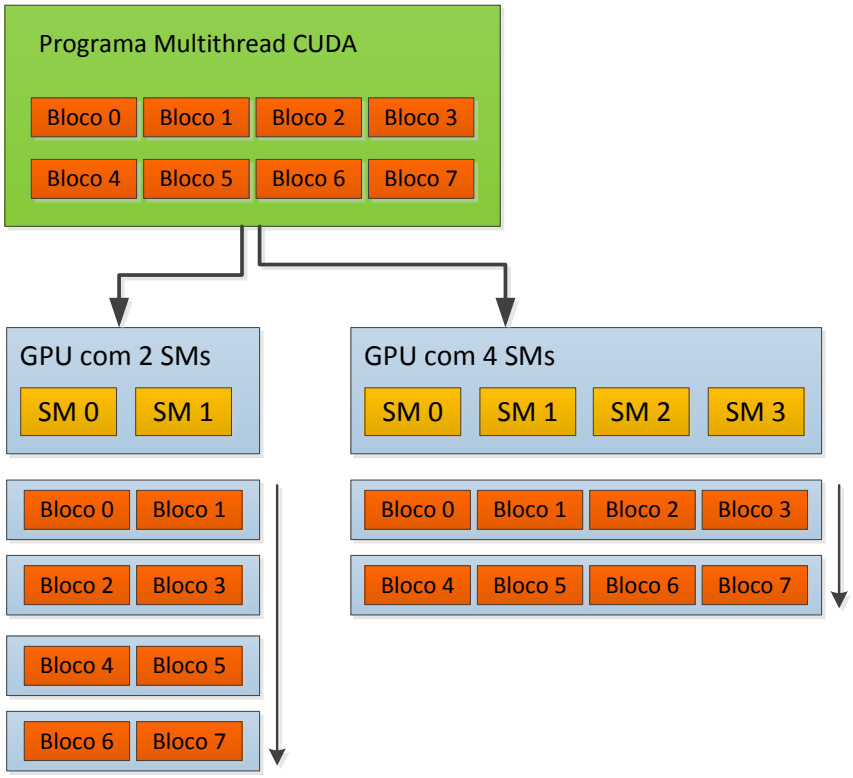


Figura 5.1 – Escalabilidade automática.

Na figura 5.1 [10], tem-se um programa CUDA multithread sendo executado em uma GPU com dois multiprocessadores de fluxo (*Streaming Multiprocessors, SMs*) e também sendo executado em outra GPU com quatro SMs. Um programa multithread é particionado em blocos de threads que executam independentemente, então a GPU que executará mais rapidamente será a que possui mais SMs.

5.1.1. Modelo de Programação

CUDA C Library permite ao programador definir funções em C chamadas de *kernels*, que quando chamadas são executadas N vezes em paralelo por N diferentes threads, ao invés de uma única vez, como as demais funções em C.

I. Kernels

Um *kernel* é definido utilizando o especificador `__global__` na declaração do *kernel* e o número de threads que executam no *kernel* para uma determinada chamada do *kernel* é especificado usando `<<< ... >>>` para configuração de execução durante as chamadas. Cada thread que executa o *kernel* é dado um thread ID único que pode ser acessado dentro do *kernel* através da variável `threadIdx`.

II. Hierarquia de Threads

Por conveniência, a variável `threadIdx` é um vector de três componentes, de modo que as threads podem ser identificadas utilizando um unidimensional, bidimensional ou tridimensional índice, formando um bloco de threads unidimensional, bidimensional ou tridimensional. Isso fornece uma maneira natural para realizar cálculos através dos elementos em um domínio como um vetor, matriz ou volume.

O índice de uma thread e sua thread ID (identificador de thread) se relacionam de uma forma simples: Para um bloco unidimensional, eles são o mesmo, mas em um bloco bidimensional de tamanho (Dx, Dy), o ID de uma thread de índice (x, y) é (x + y Dx), para um bloco tridimensional de tamanho (Dx, Dy, Dz), o ID de uma thread de índice (x, y, z) é (x + Dx y + z Dx Dy). O limite de threads por bloco é atualmente de 1024 threads. Os blocos são organizados numa rede unidimensional, bidimensional ou tridimensional de blocos de threads, como ilustrado pela figura 5.2. O número de blocos de threads em um grid é normalmente ditada pelo tamanho dos dados que estão sendo processados ou o número de processadores no sistema, que pode exceder em muito. O número de threads por bloco e o número de blocos por grid são especificados no `<<< ... >>>` e pode ser do tipo `int` ou `DIM3`. Blocos bidimensionais ou grids podem ser especificados. Cada bloco de dentro da grade pode ser identificado por um unidimensional, bidimensional ou tridimensional índice acessíveis no *kernel* através da variável `blockIdx`. A dimensão do bloco de thread é acessível dentro do *kernel* através da variável `blockDim` .

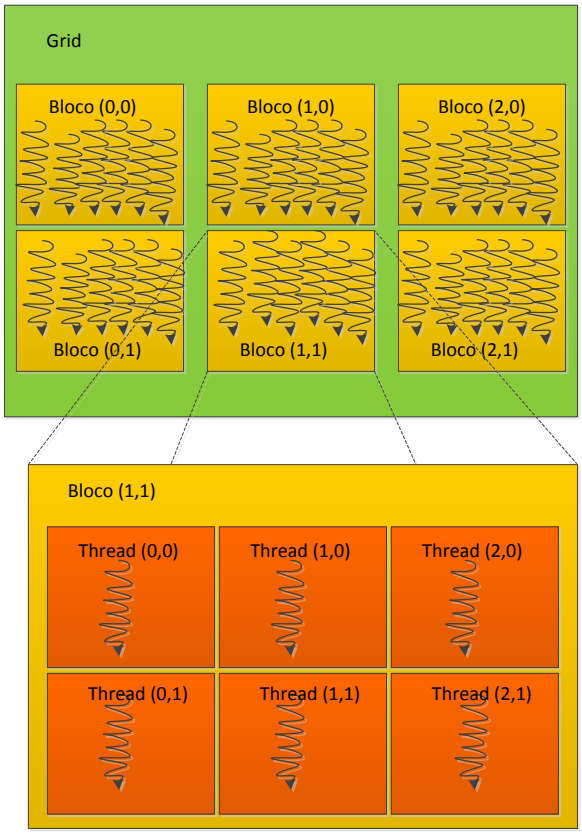


Figura 5.2 – Grid de blocos de threads.

Barreiras de sincronização de threads podem ser necessárias para coordenar os acessos a memória. É possível chamar explicitamente pontos de sincronização dentro do *kernel* através da função `__syncthreads()`. Essa função faz com que todas as threads dentro de um bloco aguardem até estiverem autorizadas a prosseguir.

III. Hierarquia de memória

Threads podem acessar dados de vários espaços de memória durante a sua execução. Cada thread tem memória privada local. Cada bloco de thread possui memória compartilhada visível a todas as threads. Todas as threads têm acesso à mesma memória global.

IV. Programação heterogênea

O modelo de programação CUDA assume que as threads executem em um dispositivo (*device*) fisicamente separado que funciona como um coprocessador para o hospedeiro (*host*) que executa o programa em C. Este é o caso, por exemplo, quando os *kernels* são chamados e executam na GPU e o resto do programa em C executa em CPU. Esse modelo também assume que o *host* e o *device* mantenham os seus próprios espaços de memória. Isso inclui a alocação de memória do dispositivo e desalocação, bem como a transferência de dados entre o hospedeiro e a memória do dispositivo.

5.2. *IMPLEMENTAÇÃO*

Serão mostradas nessa seção, as funções que executam os algoritmos citados nesse trabalho, e suas chamadas baseadas no modelo de programação em CUDA. Serão mostrados também os métodos que são processados no hospedeiro como base.

5.2.1. Transformada de Radon

Baseado no pseudocódigo da seção 2.3, é possível implementar um método que execute a Transformada de Radon. Abaixo a função que roda no hospedeiro (*host*).

```
// Transformada de Radon para o hospedeiro (processamento em CPU)
void TRadon(unsigned int * imagem, float * sinograma, int width, int nProj,
int nTraj, float dU)
{
	int x, y, s, t, u;
	float theta, dTheta = 3.14159265/ nProj;
	int diagonal = (int) width*1.4142;
	int halfWidth = (int) width/2;
	int halfD = (int) diagonal/2; // meia diagonal da imagem
	float cosT, sinT;
	double densidade;
	unsigned int sIdx, tIdx, yIdx, xIdx;

	// Varrendo Sinograma  a ser gerado
	for(sIdx = 0; sIdx < nTraj; sIdx++)
	{
		for(tIdx = 0; tIdx < nProj; tIdx++)
		{
			// s, t coordenadas para o calculo
			s = (int) sIdx;
			t = (int) tIdx;

			s = s - halfWidth;

			theta = (float) t*dTheta;
			densidade = sinograma[width*tIdx + sIdx];

			theta = t*dTheta;
			cosT = cos(theta);
			sinT = sin(theta);


			for(u = -halfD; u < halfD; u+= dU)
			{
				x = (int) s*cosT - u*sinT;
				y = (int) s*sinT + u*cosT;

				xIdx = (unsigned int) x + halfWidth;
				yIdx = (unsigned int) y + halfWidth;
```

```
                        // Verificando se pixel(x,y) está dentro da imagem
                        if( x + halfWidth >= 0 && x + halfWidth < width
                        && y + halfWidth >= 0 && y + halfWidth < width )
                        {

                        densidade = densidade + imagem[width*yIdx + xIdx];
                        }
                }

                sinograma[width*tIdx + sIdx] = densidade;
        }
    }
}
```

A chamada da função é simples, o parâmetro `imagem` é o Fantoma, o segundo é o `sinograma`, percebe-se que os dois primeiros parâmetros são ponteiros, que são alocados de forma bidimensional pela função `cudaMallocHost`, isso foi feito porque a biblioteca de CUDA trabalha dessa forma, com vetores unidimensionais, mas que podem ser alocados para duas e três dimensões. O terceiro parâmetro `width` representa a largura da imagem, essas são imagens quadradas, daí só um parâmetro de dimensão é necessário. Os dois últimos `nProj` e `nTraj` são os números de projeções e trajetórias respectivamente que se deseja obter. Há uma mudança nas coordenadas dos índices e as coordenadas para os cálculos no método, um exemplo é a variável `s` é igual ao índice `sIdx` menos a meia largura da imagem, isso é uma retificação, pois os cálculos consideram o centro da imagem como a origem, a variável `s` percorre de `-halfWidth` até `halfWidth` com `nTraj` trajetórias, o mesmo vale de maneira similar para `x` e `y`.

Abaixo o *kernel* da transformada de Radon:

```
__global__ void TRadonGPU (unsigned int *imagem, float *sinograma, int nProj,
int width, float dU){

        int x, y, s, t;
        float u, theta, dTheta = 3.14159265/ nProj;

        int diagonal = (int) width*1.4142;
        int halfWidth = (int) width/2;
        int halfD = (int) diagonal/2; /// meia diagonal
        float cosT, sinT;
        double densidade, temp;
        unsigned int sIdx, tIdx, yIdx, xIdx;

        /// Índices do Sinograma
        sIdx = blockDim.x * blockIdx.x + threadIdx.x;
        tIdx = blockDim.y * blockIdx.y + threadIdx.y;

        /// s, t coordenadas para o calculo
        s = (int) sIdx;
        t = (int) tIdx;

        s = s - halfWidth;

        theta = (float) t*dTheta;
        densidade = sinograma[width*tIdx + sIdx];

        theta = t*dTheta;
        cosT = cos(theta);
        sinT = sin(theta);
```

```
        if(t < nProj)
        {

                for(u = -halfD; u < halfD; u+= dU)
                {
                        x = (int) s*cosT - u*sinT;
                        y = (int) s*sinT + u*cosT;

                        xIdx = (unsigned int) x + halfWidth;
                        yIdx = (unsigned int) y + halfWidth;

                        /// Verificando se pixel(x,y) está dentro da imagem
                        if( x + halfWidth >= 0 && x + halfWidth < width && y +
                        halfWidth >= 0 && y + halfWidth < width )
                        {
                                densidade += (float) imagem[width*yIdx + xIdx];
                        }

                }
        }

        sinograma[width*tIdx + sIdx] = densidade;
}
```

Obedecendo ao modelo de programação em CUDA já discutido, o *kernel* da Transformada de Radon é chamado da seguinte maneira no corpo do programa.

```
TRadonGPU<<<gridDim, blockDim>>>(d_image, sinogramaGPU, width, U_sizePath);
```

A variável `gridDim` e `blockDim` especificam as dimensões do grid e do bloco respectivamente, são do tipo `dim3` e é incializada da seguinte maneira:

```
dim3 gridDim(width/BLOCK_X, height/BLOCK_Y);
dim3 blockDim(BLOCK_X,BLOCK_Y);
```

Multiplicando as dimensões do grid com as dimensões do bloco tem-se a quantidade de threads nesse caso é igual ao número de pixels da imagem. Dentro do *kernel* tem-se um diferença bastante notável, os laços que varrem o sinograma com os índices `sIdx` e `tIdx` não existem, pois ao chamar o kernel já está havendo uma paralelização das threads, e o índice das threads é calculado no trecho:

```
sIdx = blockDim.x * blockIdx.x + threadIdx.x;
tIdx = blockDim.y * blockIdx.y + threadIdx.y;
```

Isso se deve ao modelo de hierarquia de threads em CUDA, a figura 5.2 ilustra perfeitamente isso. Após a chamada do *kernel*, a função `cudaThreadSynchronize()` é chamada.

5.2.2. Retroprojeção

Baseado no pseudocódigo da seção 3.1, é possível implementar um método que execute a Retroprojeção conhecida também como *backprojection*, *BP*. Abaixo a função desse método de reconstrução que roda no hospedeiro (*host*).

```
void backprojection(float * backProjection, float *sinograma, int nProj,
                    int width)
{
       int x, y, s, t;
       float xFloat, yFloat, sFloat;
       float theta, dTheta = 3.14159265/nProj; // Pi/nProj
       float halfWidth = (float) width/2;
       float cosT, sinT;
       double projecao, temp;

       for(x = 0; x < width; x++)
       {
              xFloat = x - halfWidth + 0.1;
              for(y = 0; y < width; y++)
              {

                     yFloat = y -halfWidth + 0.1;
                     projecao = 0;
                     temp = 0;

                     for(t = 0; t < nProj; t++)
                     {
                            theta = t*dTheta;
                            cosT = cos(theta);
                            sinT = sin(theta);

                            sFloat = xFloat*cosT + yFloat*sinT;

                            s = sFloat + halfWidth;
                            if(s >= 0 && s < width)
                            {
                                   temp = sinograma[width*t + s];
                                   projecao = projecao + temp;
                            }
                     }

                     if(backProjection [width*y + x] + projecao <= 0)
                            backProjection [width*y + x] = 0;
                     else
                            backProjection [width*y + x] += projecao;

              }
       }
}
```

De forma semelhante tem-se o *kernel* para o mesmo algoritmo, seguindo o modelo de programação em CUDA, tem-se:

```
__global__ void backprojectionGPU (float *sinograma, float *backProjection,
                                   int nProj, int width)
{

	int x, y, s, t, u;
	float theta, dTheta = 3.14159265/ nProj; // Pi/nProj
	int halfWidth = width/2;
	float cosT, sinT;
	float projecao, temp;

	/// Índices do backProjection
	unsigned int xIdx = blockDim.x * blockIdx.x + threadIdx.x;
	unsigned int yIdx = blockDim.y * blockIdx.y + threadIdx.y;

	// x, y
	x = (int)xIdx;
	y = (int)yIdx;
	x = x - halfWidth;
	y = y - halfWidth;

	projecao = 0;
	temp = 0;
	for(t = 0; t < nProj; t++)
	{
		theta = t*dTheta;
		cosT = cos(theta);
		sinT = sin(theta);

		s = (int) x*cosT + y*sinT;

		if(s < halfWidth && s >= -halfWidth)
		{
			temp = sinograma[width*t + (s + halfWidth)];
			projecao += temp;
		}
	}

	backProjection[width*(yIdx) + (xIdx)] += projecao;
}
```

5.2.3. Retroprojeção filtrada

A retroprojeção filtrada possui uma etapa de filtragem que antecede a retroprojeção. Sendo assim, será mostrada aqui apenas a etapa de filtragem, já que a retroprojeção já foi exibida anteriormente. Abaixo a função desse método de reconstrução que roda no hospedeiro (*host*), baseado na fórmula da convolução (equação 4.10) e no filtro de Shepp-Logan no domínio espacial (equação 4.8).

```
void filtrateProjections (float * sinograma, float * filteredSinograma,
                          int filterWidth, int nProj, int width)
{
	int s, t;
	int i, j;
	float pi = 3.14159265;
	float pi2 = pi*pi;
	float filterValue;
```

```
	for(t =0; t < nProj; t++)
	{
		for(s = 0; s < width; s++)
		{
			i = s + t*width;

			j = 0;
			filterValue = (2.0)/(pi2*(1 - 4*j*j));
			filteredSinograma[i] = filterValue*sinograma[i];

			for(j=1; j < filterWidth;j++)
			{

				filterValue = (2.0)/(pi2*(1 - 4*j*j));

				if(s+j < width)
				filteredSinograma[i] += filterValue*sinograma[i+j];

				if(s-j >= 0)
				filteredSinograma[i] += filterValue*sinograma[i-j];
			}
		}
	}
}
```

A chamada da função de filtragem no *FBP (filtered backprojection)*, como já discutida deve anteceder a etapa de retroprojeção. Sendo assim as chamadas das funções ficam nessa ordem no corpo do programa:

```
filtrateProjections(sinogramaCPU, filteredSinogramaCPU, FILTER_SIZE, N_PROJ,
width);

backprojection(reconstruction, filteredSinogramaCPU, width, N_PROJ, N_TRAJ);
```

De forma semelhante tem-se o *kernel* para o mesmo algoritmo, seguindo o modelo de programação em CUDA, tem-se:

```
__global__ void filtrateProjectionsGPU (float * sinograma,
		float * filteredSinograma, int filterWidth, int nProj, int width)
{

	int s, t;
	int i, j;
	float pi = 3.14159265;
	float pi2 = pi*pi;
	float filterValue;
	float sum = 0;

	// Índices do Sinograma
	s = blockDim.x * blockIdx.x + threadIdx.x;
	t = blockDim.y * blockIdx.y + threadIdx.y;

	i = s + t*width;
```

```
	if(t < nProj)
	{

		j = 0;
		filterValue = (2.0)/(pi2*(1 - 4*j*j));
		filteredSinograma[i] = filterValue*sinograma[i];


		for(j = 1; j < filterWidth; j++)
		{
			filterValue = (2.0)/(pi2*(1 - 4*j*j));

			if(s+j < width)
				filteredSinograma[i] += filterValue*sinograma[i + j];


			if(s-j >= 0)
				filteredSinograma[i] += filterValue*sinograma[i - j];
		}
	}
}
```

As chamadas dos *kernels* ficam dispostas na seguinte ordem no corpo do programa:

```
filtrateProjectionsGPU<<<gridDim, blockDim>>>(sinogramaGPU, filteredSinogramaGPU,
FILTER_SIZE, N_PROJ, width);
cudaThreadSynchronize();


backprojectionGPU<<<gridDim, blockDim>>>(filteredSinogramaGPU, reconstructionGPU,
N_PROJ, width);
cudaThreadSynchronize();
```

Nas versões mais novas de cuda, é usado a função `cudaDeviceSynchronize()` no lugar de `cudaThreadSynchronize()`. O modelo de programação em CUDA oferece muito mais do que discutido aqui, uso de outros tipos de memória que podem ajudar a obter um melhor desempenho é um belo exemplo.

6. Resultados

Nesse capítulo serão mostrados os resultados obtidos com os algoritmos da Transformada de Radon, e dos métodos de reconstrução pela Retroprjeção e a Retroprojeção Filtrada por Convolução. Os resultados são as imagens geradas como os tempos de processamento em CPU e GPU.

6.1. TRANSFORMADA DE RADON

O intuito é poder simular o processo tomográfico em imagens e obter os Sinogramas para depois ser possível testar os métodos de reconstrução. As implementações foram feitas em CPU e GPU. Abaixo os Fantomas usados como imagens de entrada:

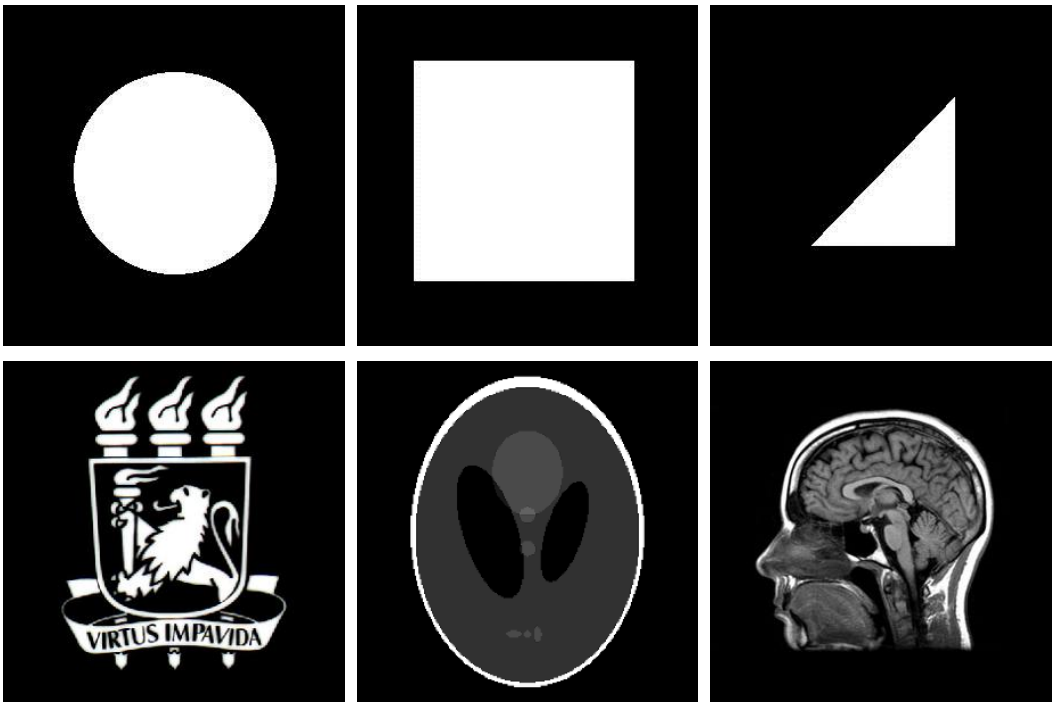


Figura 6.1 – Fantomas de dimensões 300 x 300 pixels. A imagem do córtex cerebral foi retirada de [11].

6.1.1. Imagens Obtidas

As imagens abaixo são conhecidas como Sinogramas, estão na configuração de 300 trajetórias por 300 projeções.

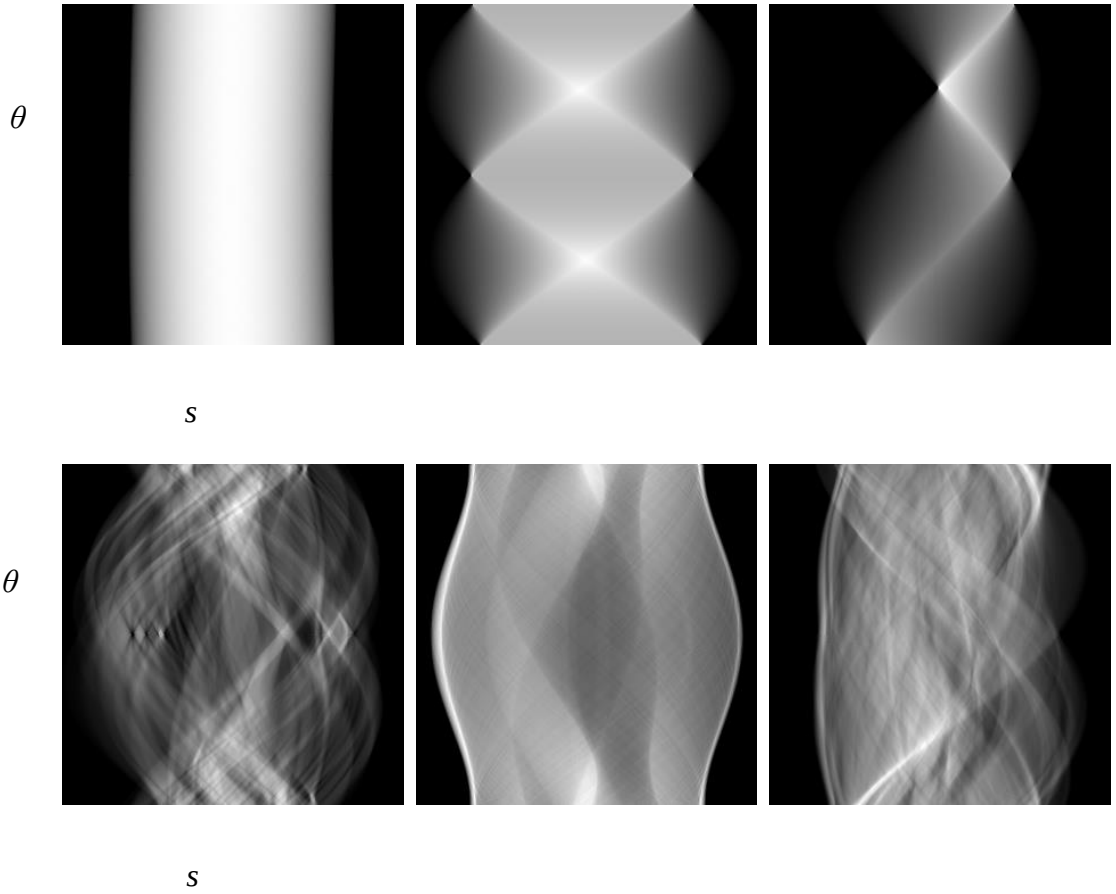
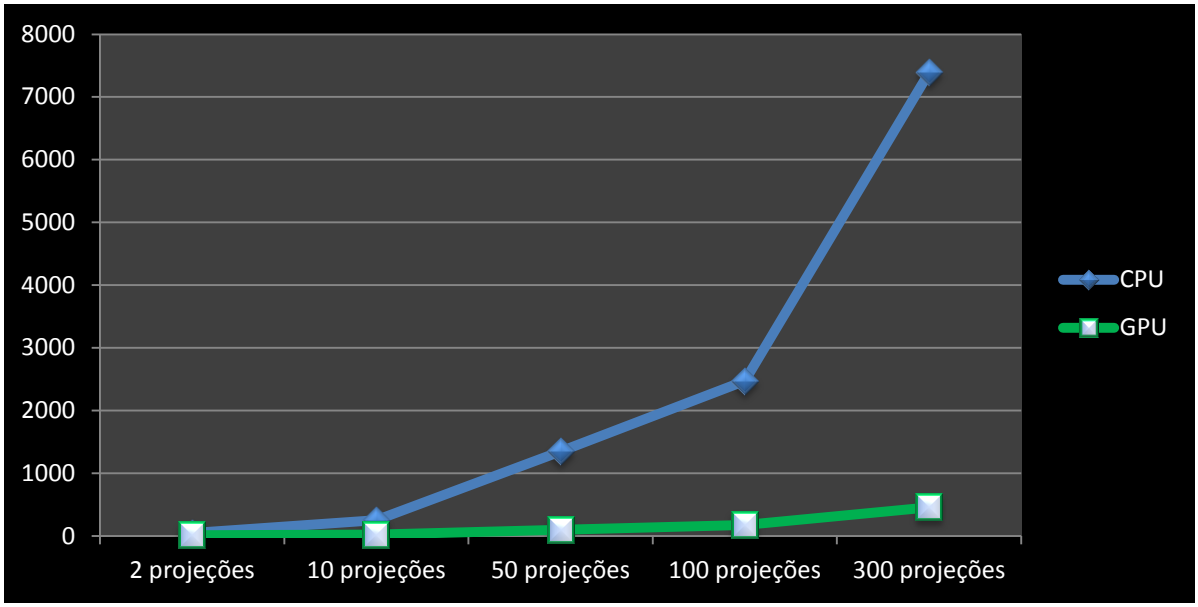


Figura 6.2 – Sinogramas do círculo, quadrado, triângulo, brasão da UFPE, *head phatom* de Shepp-Logan e córtex cerebral respectivamente.

O Sinograma é essencialmente uma representação no domínio de Radon, e é obtido a partir dos dados de atenuação durante o processo de tomografia, que nesse caso foi feito de forma simulada no computador pelo o algoritmo da Transformada de Radon.

6.1.2. Tempos Obtidos

Os processamentos realizados em CPU e GPU foram realizados a partir de Sinograma com diferentes configurações de amostragem, de 300 trajetórias e com 2, 10, 50, 100 e 300 diferentes ângulos de projeção, entre 0 a 180 graus, o tamanho do passo do parâmetro de linha u foi de 0,1. Abaixo gráfico e tabela dos tempos de processamento em ambos os dispositivos, em milissegundos. O menor ganho obtido foi de 4,53 vezes e o maior foi de 16,24 vezes.



	2 proj.	10 proj.	50 proj.	100 proj.	300 proj.
CPU(ms)	49,58	253,3	1351,53	2472,22	7393,52
GPU(ms)	10,93	22,31	99,3	177,11	455,2

Gráfico 6.1 – Tempos de processamento da Transformada de Radon para diferentes projeções

6.2. *RETROPROJEÇÃO*

Nessa seção serão mostradas as imagens resultantes e os tempos de execução da retroprojeção em cada um dos dispositivos.

6.2.1. Imagens Obtidas

As retroprojeções foram feitas a partir de Sinogramas com diferentes números de projeções, abaixo as imagens obtidas pela retroprojeção. A retroprojeção com 100 projeções foi omitida por haver pouca diferença de qualidade gráfica com a retroprojeção com 300 projeções, mas seu tempo de processamento foi computado e será exibido na outra seção.

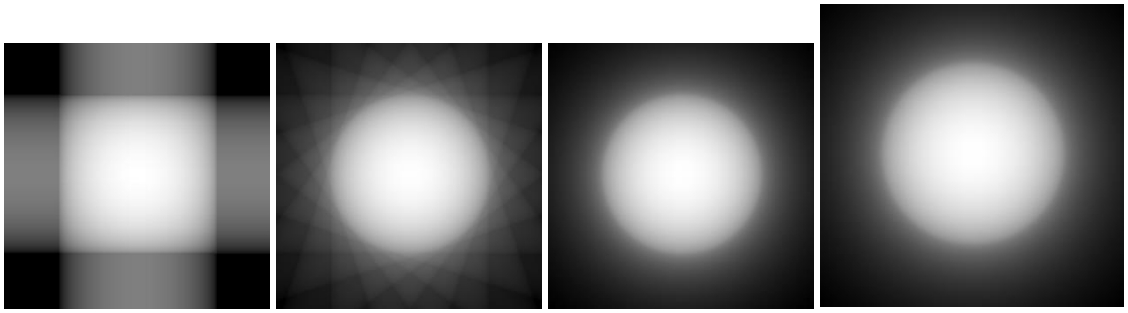


Figura 6.3 – Retroprojeção do circulo a partir de 2, 10, 50 e 300 projeções.

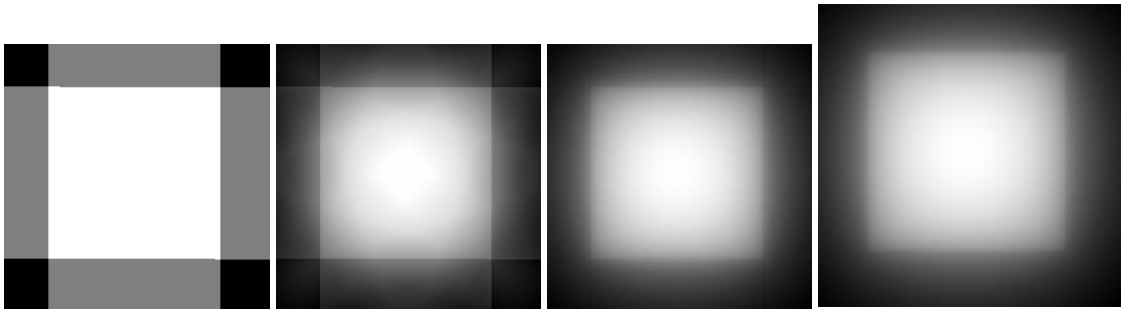


Figura 6.4 – Retroprojeção do quadrado a partir de 2, 10, 50 e 300 projeções.

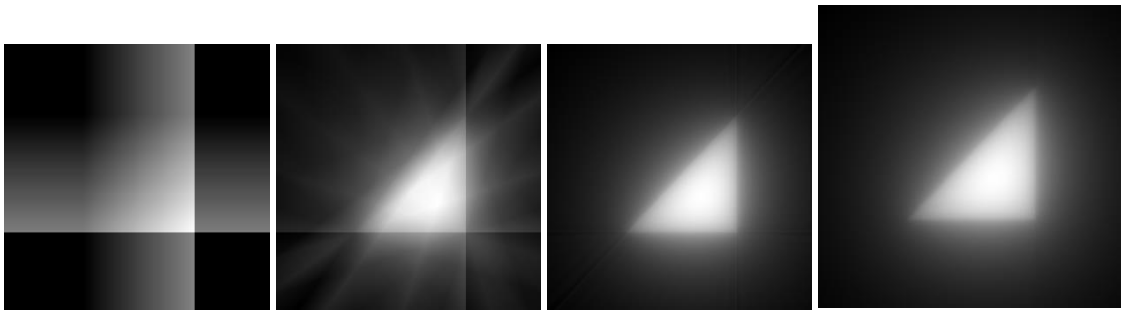


Figura 6.5 – Retroprojeção do triangulo a partir de 2, 10, 50 e 300 projeções.

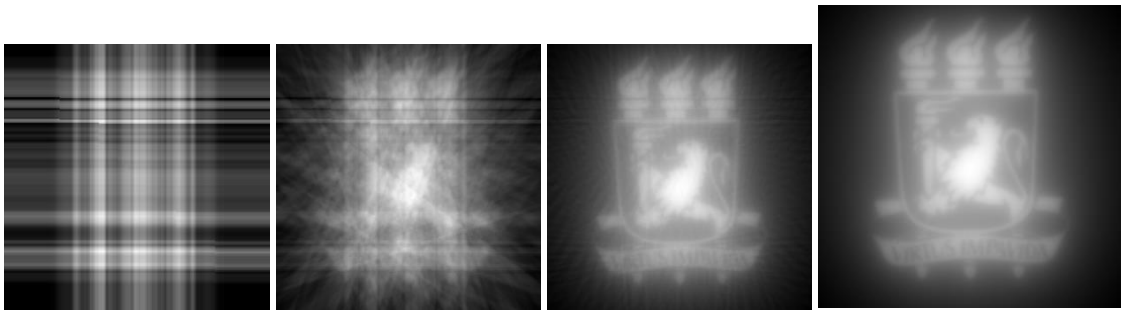


Figura 6.6 – Retroprojeção do brasão da UFPE a partir de 2, 10, 50 e 300 projeções.

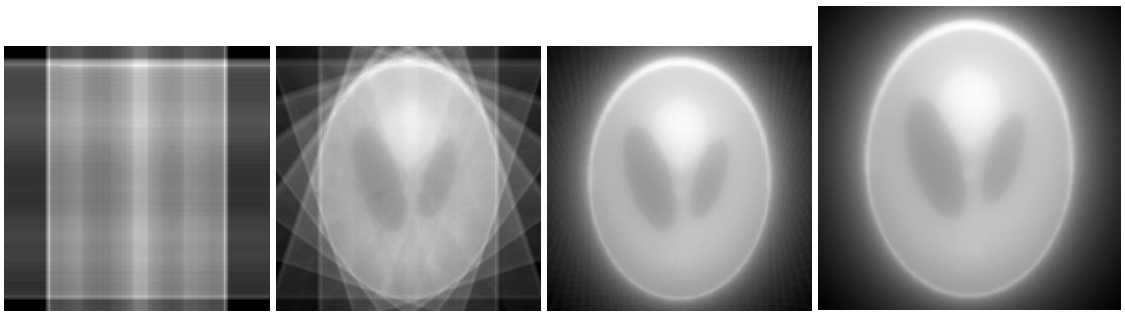


Figura 6.7 – Retroprojeção do *head phantom* de Shepp-Logan a partir de 2, 10, 50 e 300 projeções.

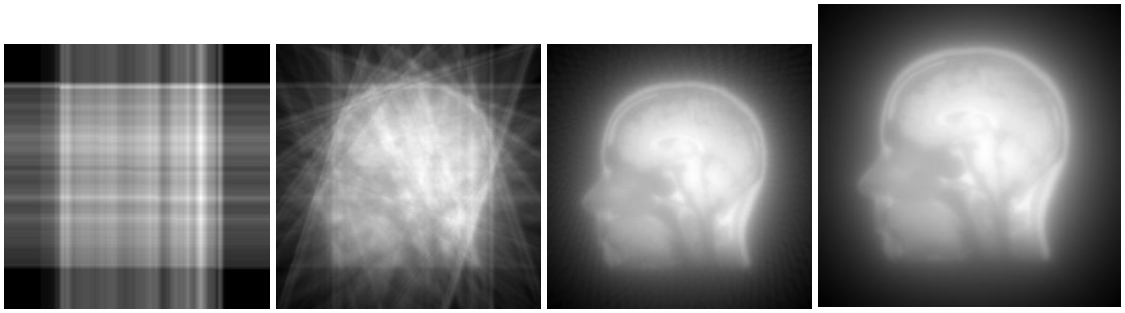
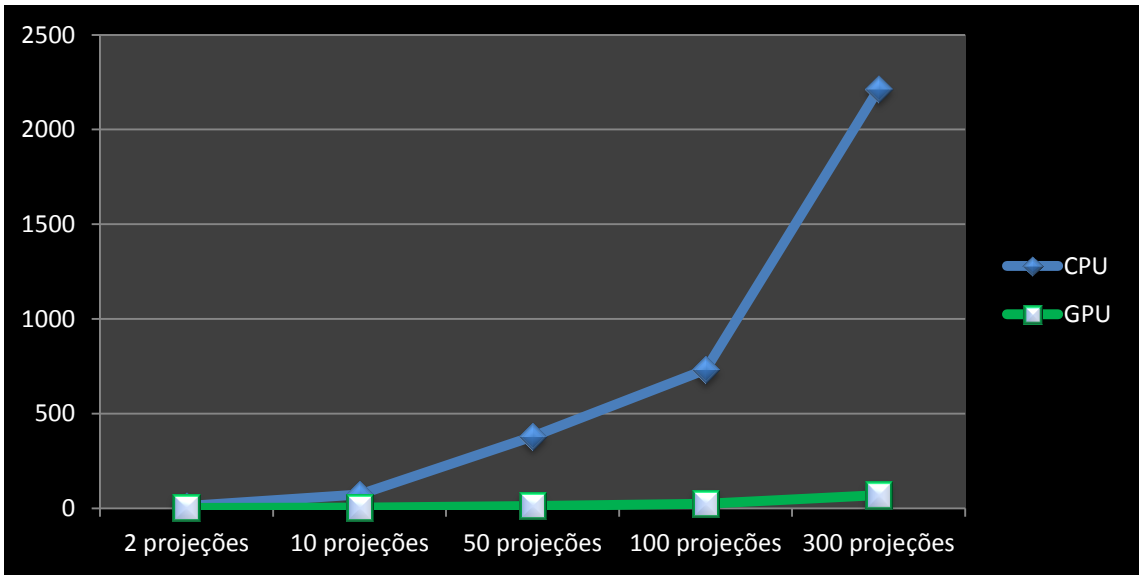


Figura 6.8 – Retroprojeção do córtex cerebral a partir de 2, 10, 50 e 300 projeções.

As imagens obtidas com o BP geram um problema de clareamento de fundo, a qualidade da imagem gerada não é tão boa, mesmo com um grande número de amostragens, e em alguns casos é muito ruim, quando a imagem possui uma distribuição de densidade bem variada, como no Fantoma de Shepp-Logan e no córtex cerebral, percebe-se também que o número de projeções com apenas duas vistas resulta em uma reconstrução muito precária. O próximo método discutido será o da Retroprojeção Filtrada (FBP) que promete ser um método que fornecerá um melhor resultado em termos de qualidade de imagem. Mas antes a próxima seção mostrará os tempos de processamento obtidos.

6.2.2. Tempos Obtidos

Os processamentos realizados em CPU e GPU foram realizados a partir de Sinograma com diferentes configurações de amostragem, de 300 trajetórias e com 2, 10, 50, 100 e 300 diferentes ângulos de projeção para realizar a retroprojeção. O ganho de desempenho no pior caso foi de 5,79 vezes e o ganho de desempenho no melhor caso foi de 31,58 vezes. Abaixo gráfico e tabela com tempos de processamento em milissegundos.



	2 proj.	10 proj.	50 proj.	100 proj.	300 proj.
CPU(ms)	12,56	73,27	378,06	732,35	2214,41
GPU(ms)	2,167	3,57	12,67	23,5	70,1

Gráfico 6.2 - Tempos de processamento da Retroprojeção.

6.3. RETROPROJEÇÃO FILTRADA POR CONVOLUÇÃO

Nessa seção serão mostradas as imagens resultantes e os tempos de execução da retroprojeção filtrada por convolução em cada um dos dispositivos, a partir de Sinograma com diferentes configurações de amostragem.

6.3.1. Imagens Obtidas

As reconstruções foram feitas a partir de Sinograma com diferentes números de projeções, abaixo as imagens obtidas pela retroprojeção filtrada. A reconstrução com 100 projeções foi omitida por haver pouca diferença perceptível da qualidade gráfica comparada a reconstrução com 300 projeções, mas seu tempo de processamento foi computado e será exibido na outra seção.

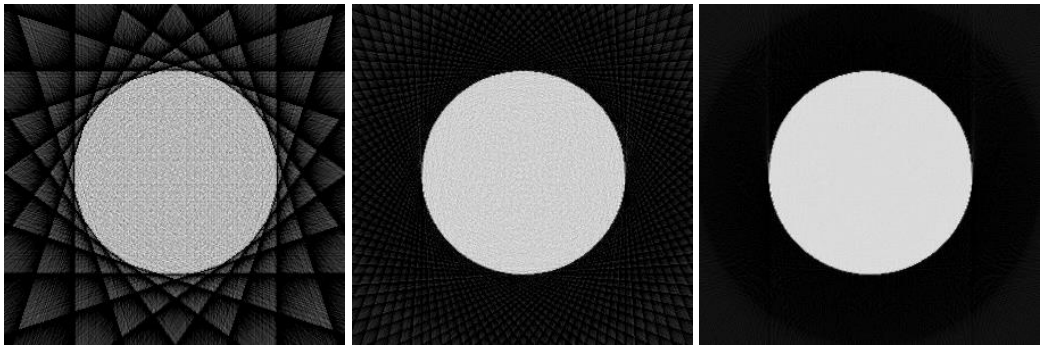


Figura 6.9 – Retroprojeção filtrada do círculo a partir de 10, 50 e 300 projeções.

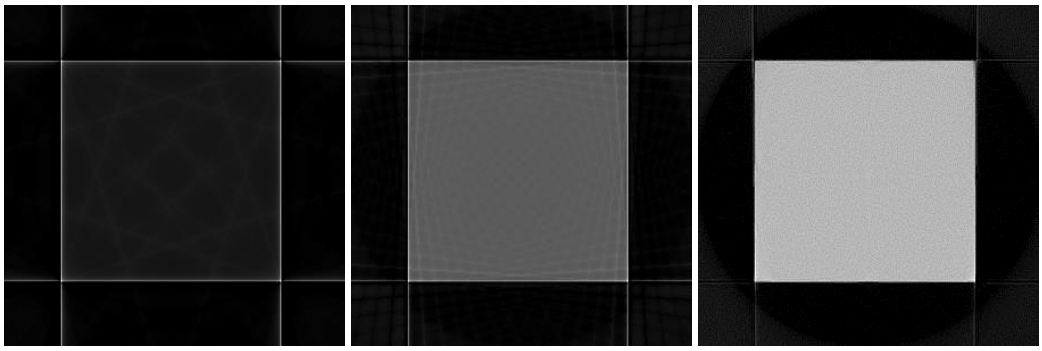


Figura 6.8 – Retroprojeção filtrada do quadrado a partir de 10, 50 e 300 projeções.

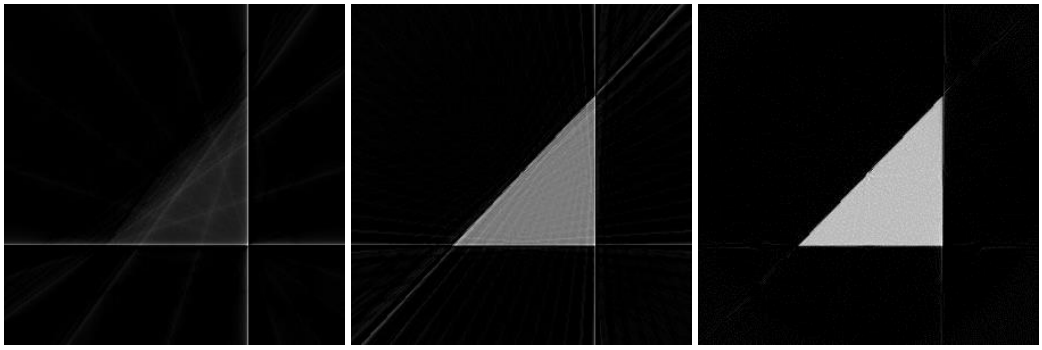


Figura 6.9 – Retroprojeção filtrada do triangulo a partir de 10, 50 e 300 projeções.

Figura 6.10 – Retroprojeção filtrada do brasão da UFPE a partir de 10, 50 e 300 projeções.

Figura 6.11 – Retroprojeção filtrada do *head phantom* de Shepp-Logan a partir de 10, 50 e 300 projeções.

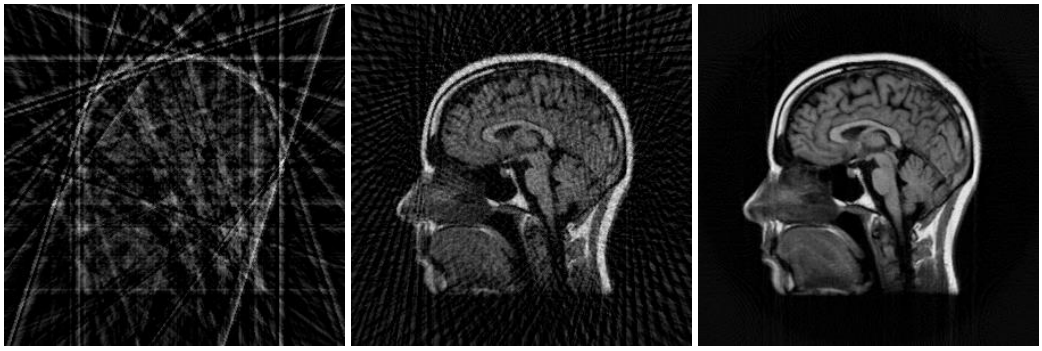
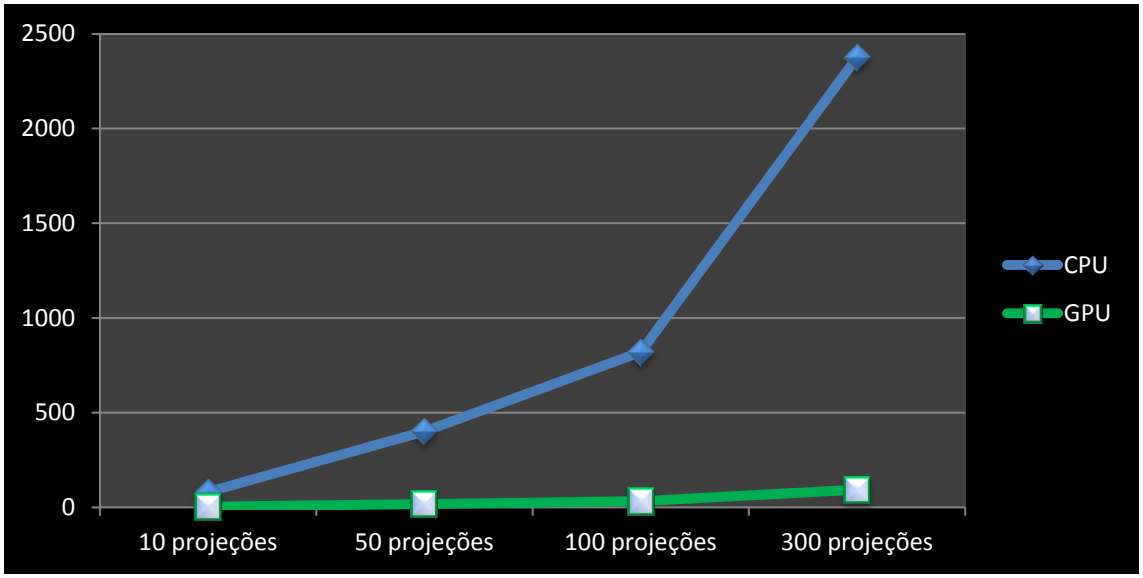


Figura 6.12 – Retroprojeção filtrada do córtex cerebral a partir de 10, 50 e 300 projeções.

Os resultados das imagens com o método da retroprojeção filtrada são muito gratificantes, o método se saiu muito bem para imagens com pouca e muita complexidade, a filtragem eliminou com eficiência a nebulosidade esbranquiçada causada pela retroprojeção. O tamanho do filtro usado para se obter esses resultados foi igual ao tamanho da largura da imagem, e à medida que aumenta o número de projeções se obtêm uma imagem reconstruída mais fiel.

6.3.2. Tempos Obtidos

O ganho de desempenho no pior caso foi de 19,37 vezes e o ganho de desempenho no melhor caso foi de 25,68 vezes, abaixo o gráfico e tabela com os tempos obtidos em milissegundos com a Retroprojeção filtrada em ambos os dispositivos.



	10 proj.	50 proj.	100 proj.	300 proj.
CPU(ms)	81,34	400,72	819,4	2376,56
GPU(ms)	4,2	16,75	33,21	92,5

Gráfico 6.3 - Tempos de processamento da Retroprojeção filtrada a partir de Sinograma com diferentes configurações de amostragens.

Foi visto em todos os resultados que à medida que a quantidade dos dados a serem processados crescia os valores dos ganhos da GPU em relação a CPU também crescia. Na Transformada de Radon os valores de ganho para poucas projeções começavam de apenas duas projeções e iam até 300 projeções, e os ganhos de tempo também crescia, o mesmo vale para a Retroprojeção e Retroprojeção Filtrada por convolução. Isso mostra que quando se está trabalhando com uma quantidade massiva de dados o uso de um dispositivo que faz processamento paralelo é indispensável.

7. ART em CUDA, como Trabalho Futuro.

O ART (*Algebraic Reconstruction Technique*) é uma técnica de reconstrução algébrica desenvolvida em 1970 por Gordon, Bender e Herman [3], para reconstruir a imagem, o ART se baseia na solução de um sistema linear pelo método de Kaczmarz. Os métodos algébricos são também conhecidos como métodos iterativos. A solução é obtida a partir de uma aproximação inicial e o cálculo é realizado iterativamente até ser atingida a precisão desejada. Embora estes métodos sejam conceitualmente mais simples essa abordagem é normalmente menos eficientes do ponto de vista computacional.

No ART os valores das atenuações ao longo da trajetória são os resultados das equações, e os tamanhos de segmentos que são calculados e armazenados na matriz de pesos são valores já conhecidos [9], o que ainda não se conhece são os coeficientes que representam os pixels da imagem a ser reconstruída. O sistema de equações abaixo representa essa forma algébrica de encarar o problema:

$$\begin{gathered} w11v1 + w12v2 + w13v3 + \ldots + w1NvN = p1 \\ w21v1 + w22v2 + w23v3 + \ldots + w2NvN = p2 \\ w31v1 + w32v2 + w33v3 + \ldots + w3NvN = p3 \\ \ldots \\ wM1v1 + wM2v2 + wM3v3 + \ldots + wMNvN = pM \end{gathered} \tag{7.1}$$

As variáveis w são os segmentos da matriz de pesos. As variáveis p a direita das equações são os valores de $g(s, \theta)$, valores contidos no sinograma, e por fim os coeficientes v são os valores procurados, que representam os pixels da imagem a ser reconstruída. Essa resolução de equações é necessária para ter uma aproximação de valores desses coeficientes. O método de Kaczmarz é iterativo, e ao passo de cada iteração, vai se obtendo um resultado melhor, até alcançar um limite de convergência satisfatório. A equação abaixo é o calculo realizado a cada iteração:

$$f_j(i) = \ f_j(i-1) \ + \frac{\lambda(p_t \ - q_t)}{\sum_{k=1}^{N}(w_{tk})^{\,2}} \, w_{tj} \tag{7.2}$$

Onde $f_j(i)$ é o valor do pixel j da imagem f a ser reconstruída na atual iteração i, p_t é o valor de uma trajetória t do Sinograma original, q é o sinograma calculado entre uma iteração e outra, esse Sinograma pode ter um valor inicial com um chute inicial, ou simplesmente valores nulos, w_{tj} é o tamanho do segmento do raio que passou em determinado elemento do objeto na trajetória t em um elemento (pixel) j, e λ é um fator de relaxação. O ART faz bastante uso da matriz de pesos, isso alivia o custo computacional apesar de ser iterativo. Abaixo resultados obtidos com o ART na versão feita para processamento em CPU.

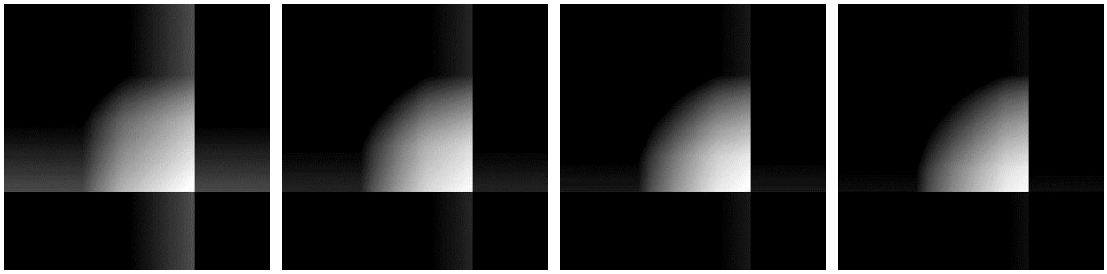


Figura 7.1 - Reconstruções algébricas do triangulo retângulo com 100, 300, 500 e 900 iterações respectivamente, a partir de Sinograma com apenas duas projeções.

É possível perceber que é um algoritmo muito bom em tomografia com poucas amostragens, e a medida que cresce o número de iterações há uma convergência para um melhor resultado.

Nesse trabalho o ART foi desenvolvido apenas na versão que é executada em CPU. Porém uma implementação em ART que funciona em GPU é bem-vinda, tanto na questão de um melhor desempenho como também na questão de estudo do desempenho da GPU em algoritmos iterativos, onde seria possível se extrair alguma relação com número de iterações e tempo de execução em GPU.

8. Conclusões

A importância da tomografia vai desde o uso de diagnósticos por imagens na medicina até a inspeção do interior de dispositivos manufaturados na indústria. Os algoritmos implementados para executarem em GPU mostraram ter em todos os casos um desempenho superior aos mesmos algoritmos implementados para CPU. Isso se deve ao grande poder de paralelismo fornecido pelas GPUs e também o fato de o problema de reconstrução de imagens ser um problema de natureza paralelizável. A Transformada de Radon foi implementada com objetivo de simular os dados obtidos por um tomógrafo real, na qual se tem como resultado o Sinograma, que é uma representação no domínio de Radon. Foi visto que o método da retroprojeção filtrada é um ótimo algoritmo para situações onde houve um grande número de trajetórias e projeções durante o processo tomográfico, tendo uma reconstrução muito satisfatória graças a filtragem, e a notável diferença das imagens tidas com a retroprojeção simples. Os ganhos de desempenho obtidos com a GPU teve comportamento sempre crescente na proporção que a quantidade de dados crescia, revelando o seu poder. Os tempos obtidos com a GPU tinham um tímido crescimento enquanto a CPU teve um grande crescimento, o que revela que a necessidade de dispositivos paralelizáveis para aplicações desse tipo é de vital importância. O uso de GPUs para aplicações de alto desempenho é bastante motivador, devido ao seu baixo custo em relação a outros hardwares que oferecem paralelismo computacional e suporte com bibliotecas de linguagens. Diversas áreas técnicas e científicas vão sofrer e já estão sofrendo com o impacto positivo das GPUs. As técnicas de reconstrução tiveram um ótimo desempenho com esse dispositivo.

9. Referências

[1] Gazzani, M. H. **Reconstrução de imagens a partir de projeções paralelas.** *Dissertação à Universidade Federal de Uberlândia para obtenção do título de Mestre em Engenharia Elétrica,* 1999.

[2] Luebke, D. & Humphreys, G. **How GPUs Work.** *IEEE Computer*, pp. 126-130, 2007.

[3] Gordon, R., Bender, R., and Herman, G.T. **"Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and x-ray photography."** *J. Theoret. Biol. 29,* pp. 471–481, 1970.

[4] Nobel Prize Institute. **Wilhelm Conrad Röntgen - Biography**. Disponível em: http://www.nobelprize.org/nobel_prizes/physics/laureates/1901/rontgen-bio.html. Acessado em: 12/02/2013.

[5] SIEMENS. **Computed Tomography - Its History and Technology.** Disponível em: http://www.medical.siemens.com/siemens/zh_CN/gg_ct_FBAs/files/brochures/CT_History_and_Technology.pdf. Acessado em: 14/02/2013.

[6] Carlos Alberto dos Santos. **A computação chega à tomografia.** Disponível em: http://cienciahoje.uol.com.br/colunas/do-laboratorio-para-a-fabrica/a-computacao-chega-a-tomografia. Acessado em: 14/02/2013.

[7] Nobel Prize Institute. **The Nobel Prize in Physiology or Medicine 1979 - Allan M. Cormack, Godfrey N. Hounsfield.** Disponível em: http://www.nobelprize.org/nobel_prizes/medicine/laureates/1979/#. Acessado em: 14/02/2013

[8] R. A. Brooks & G. Di Chiro. **Principles of Computer Assisted Tomography (CAT) in Radiographic and Radioisotopic Imaging.** *Phys. Med. Biol. 21, 689-739,* 1976.

[9] A.C. Kak & M. Slaney. **Principles of Computerized Tomographic Imaging.** 1996.

[10] NVIDIA CUDA Developer Zone **. CUDA C Programming Guide.** Disponível em: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#introduction. Acessado em: 29/04/2013

[11] Brain Cortex**, Nolan Law Group**. Disponível em: http://www.nolan-law.com/practice-areas/traumatic-brain-injury/tbi-case-construction/. Acessado em: 01/05/2013

Assinaturas

Prof. Dr. Carlos Costa Dantas (Avaliador)

Prof. Dr. Silvio de Barros Melo (Orientador)

Luiz Gustavo da Rocha Charamba (Proponente)

Recife, 10 de Maio de 2013.