Projeto 1: Transformada Rápida de Radon

Marcos Teixeira

21 de abril de 2018

1 Descrição

Na tomografia computadorizada, o problema de reconstrução tomográfica é obter uma imagem tomográfica de um conjunto de projeções [1]. Uma projeção é formada desenhando um conjunto de raios paralelos através do objeto 2D de interesse, atribuindo a integral do contraste do objeto ao longo de cada raio a um único pixel na projeção. Uma única projeção de um objeto 2D é unidimensional. Para permitir a reconstrução da tomografia computadorizada do objeto, várias projeções devem ser adquiridas, cada uma delas correspondendo a um ângulo diferente entre os raios em relação ao objeto.

Nesse contexto, a Transformada de Radon trata o problema de reconstrução a partir de projeções. Portanto, se uma função $f(x, y)$ representa uma função de densidade desconhecida, a Transformação de Radon representa os dados de projeção obtidos como a saída de uma varredura tomográfica. Uma característica forte é que a transformada de Radon de um número de pequenos objetos aparece graficamente como um número de ondas senoidais borradas com diferentes amplitudes e fases [2]. A transformada inversa de Radon é usada na tomografia computadorizada para reconstruir uma imagem 2D ou 3D a partir das projeções medidas. Uma implementação prática e exata da transformação inversa do Radon não existe, mas existem vários bons algoritmos aproximados disponíveis. Como a transformação inversa de Radon reconstrói o objeto a partir de um conjunto de projeções, a transformada do Radon (forward) pode ser usada para simular um experimento de tomografia.

Para este projeto a tarefa é implementar a Transformada Rápida de Radon, que baseia-se no Teorema Central de Fourier para efetuar de uma maneira mais eficiente essa tranformada. Nas seções posteriores apresentamos os detalhes deste método, bem como os resultados em algumas imagens 2D.

2 Transformada de Radon

A matemática por trás da tomografia aplica algoritmos para a reconstrução de imagens. Teoricamente procura-se encontrar a função f, uma vez que sejam conhecidas as integrais de linha ao longo de infinitas direções no plano[3]. Na prática, é claro, tem-se uma amostragem dessas integrais de linha apenas para um número finito de direções.

A integral de linha que define a projeção pode ser escrita na forma de uma transformada, definida por Johann Radon, conhecida como Transformada de Radon. Nessa transformada, considerando o caso 2D, para cada rotação $\theta \in [0, 180)$ de um sistema fonte-sensor em torno do centro da imagem, com atenuaçãoes $f(x, y)$, esta transformada gera uma linha da imagem da projeção $R(\rho, \theta)$, cujos valores somam as atenuaçãoes $f(x, y)$ da imagem sobre o segmento de reta que liga a fonte ao sensor. Portanto, a imagem resultante pode ser expressa pela seguinte equação:

$$R(\rho, \theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x', y')\delta(y' - y)dx'dy',$$

onde y = ax + b é a equção do segmento de reta ortogonal à direção do eixo γ e a função σ considera apenas as atenuações $f(x', y')$ da imagem, tais que y' satisfaz a equação do segmento de reta.

O pipeline de execução da Transformada de Radon pode ser expresso da seguinte maneira:

i Criar a imagem de saída R(180, D)

ii Para θ=0 até 180 faça

(a) Criar a matriz de Radon M

(b) Aplicar M em cada ponto p da imagem

(c) Adicionar em $R(\theta, \rho)$ o valor acumulado das intensidades

iii Retornar R

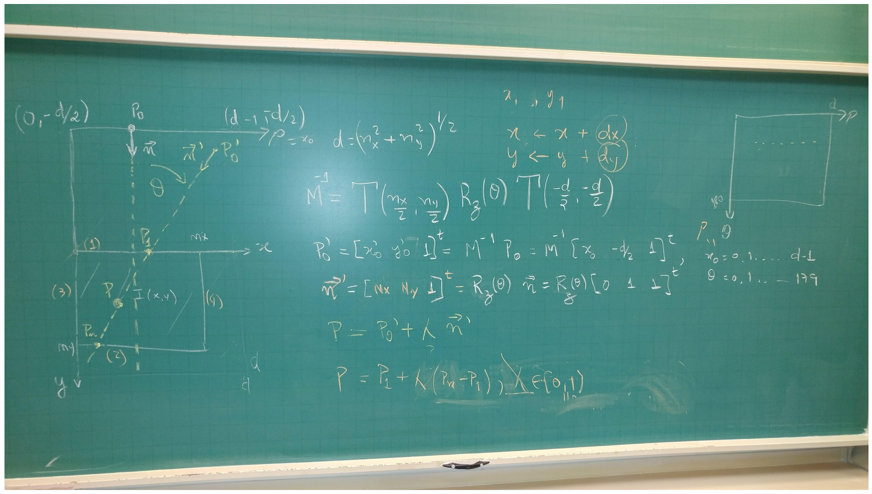
, onde a matriz M é igual a

$$M = \begin{bmatrix} 1 & 0 & 0 & d/2 \\ 0 & 1 & 0 & d/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & n_x/2 \\ 0 & 1 & 0 & n_y/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

com $D = \sqrt{nx^2 + ny^2}$ e n_x igual a largura e n_y a altura da imagem de entrada.

3 Transformada Rápida de Radon

Para melhorar a performance da formulação original da Transformada de Radon, vários outros algoritmos já foram propostos. Neste projeto nós implementamos uma forma mais intuitiva e que oferece um bom ganho de performance. A ideia principal dessa abordagem está na observação de que é muito custoso aplicar a matriz de Radon na imagem em si. Portanto, a proposta é realizar a rotacao na linha ρ e deixar a imagem parada. Esta



sutil modificação já produz uma melhoria significativa na performance da Transformada de Randon.

Uma formulação matemática desta abordagem é apresentada na Figura 3. A reta ρ tem tamanho $D = \sqrt{nx^2 + ny^2}$ (nx=altura e ny=largura) e lança raios em direção da imagem de entrada. Como iremos girar a reta ao invés do plano, a matriz de Radon nesse caso será a inversa da matriz utilizada no algoritmo original. Esta matriz está apresentada na figura como M^{-1}, composta então de uma translação de $(nx/2, ny/2)$, uma rotação no eixo z de θ graus e uma translação de $(-D/2, -D/2)$. Esta matriz será aplicada em cada ponto $P0$ da reta, gerando o novo ponto $P0^{'}$. O vetor normal, diferente do que é sugerido na equação da figura, foi feito também utilizando a matriz M^{-1} produto com a o vetor normal a $P0^{'}$.

$$P = (x_0^{'}, y_0^{'}) + \lambda(N_x, N_y) = (x_0^{'} + \lambda Nx, y_0^{'} + \lambda Ny) \tag{1}$$

Dado isso, nós temos a equação da reta (Equação 1) necessária para calcular nosso DDA. No entanto, precisamos definir qual o ponto $P1$ e Pn que representam onde o raio entrou e onde ele saiu na imagem, respectivamente. Analiticamente, só existirão 4 pontos possíveis de interesse nesse caso:

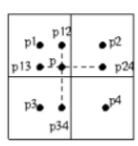
1. $y = 0$, resultando em $\lambda = -y_0^{'}/Ny$
2. $y = n_y - 1$, resultando em $\lambda = (n_y - 1 - y_0^{'})/Ny$
3. $x = 0$, resultando em $\lambda = -x_0^{'}/Nx$
4. $x = n_x - 1$, resultando em $\lambda = (n_x - 1 - x_0^{'})/Nx$

Destes 4 valores de lambdas possíveis, nós encontraremos 2 válidos que irão caracterizar o pontos $P1$ e Pn. O valor de menor lambda será atribuído a equação da reta para encontrar $P1$ e o maior lamba para encontrar Pn. Note que é necessário garantir que Nx e Ny não sejam 0 para que os testes das equações acima funcionem.

Algorithm 1: DDA

Data: Imagem *img*; Ponto $P1$; plano de bits Pn
Result: Acumulado do brilho dos pontos de $P1$ a Pn
if *p1* = *pn* **then**
  n = 1
else
  Dx = x_n - x_1
  Dy = y_n - y_1
  if $\mid Dx \mid > \mid Dy \mid$ **then**
    $n = \mid Dx \mid +1$
    $dx = sign(Dx)$
    $dy = (dxDy)/Dx$
  else
    $n = \mid Dy \mid +1$
    $dy = sign(Dy)$
    $dx = (dyDx)/Dy$
  $(x_p, y_p) = (x_1, y_1)$
  for $k \in \{1, \ldots, n\}$ **do**
    $J += I[x_p, y_p]$
    $(x_p, y_p) = (x_p + dx, y_p + dy)$
return J

Posteriormente, o algoritmo DDA será utilizado para encontrar todos os pontos da linha entre $P1$ e Pn. O pseudocódigo deste algoritmo é apresentado no Algorithm 1. Além de encontrar esses pontos, o DDA acumula os brilhos dos respectivos pontos que for passando. Como as coordenadas x e y são reais, duas estratégias foram adotadas para a resposta do DDA: aplicar um *ROUND* das coordenadas e aplicar interpolação linear. Esperamos que o resultado utilizando interpolação linear possua menos buracos que na abordagem que só realiza o *ROUND*, dado que de um lado estaremos fazendo uma estimativa linear do brilho do pixel e do outro lado estaremos apenas truncando o valor das coordenadas. Esta interpolação foi implementada seguindo as equações da Figura 3.



$$
\begin{aligned}
I_p &= (y_p - y_{p_{12}})I_{p_{34}} + (y_{p_{34}} - y_p)I_{p_{12}} \\
I_{p_{12}} &= (x_p - x_{p_1})I(p_2) + (x_{p_2} - x_p)I(p_1) \\
I_{p_{34}} &= (x_p - x_{p_3})I(p_4) + (x_{p_4} - x_p)I(p_3)
\end{aligned}
$$

O trecho de código implementado para fazer a interpolação está apresentado abaixo:

```
int LinearInterpolationValue(iftImage *img, float x, float y)
{
```

```
    iftVoxel u[4];
    float dx = 1.0;
    float dy = 1.0;
    float  P12, P34;
    int Pi;

    if ((int) (x + 1.0) == img->xsize)
        dx = 0.0;
    if ((int) (y + 1.0) == img->ysize)
        dy = 0.0;

    //closest neighbour in each direction
    u[0].x = (int)x;       u[0].y = (int)y;
    u[1].x = (int)(x + dx); u[1].y = (int)y;
    u[2].x = (int)x;       u[2].y = (int)(y + dy);
    u[3].x = (int)(x + dx); u[3].y = (int)(y + dy);


    P12 = (float)iftImgVal2D(img,u[1].x,u[1].y) * (x - u[0].x) + (float)iftImgVal2D(img,u[0].x,u[0].y) * (u[1].x - x);
    P34 = (float)iftImgVal2D(img,u[3].x,u[3].y) * (x - u[2].x) + (float)iftImgVal2D(img,u[2].x,u[2].y) * (u[3].x - x);
    Pi  = (int)P34 * (y - u[0].y) + P12 * (u[2].y - y);


    return Pi;
}
```

Listing 1: Implementação da Interpolação

“newlabel3528

Sumarizando, o pipeline de funcionamento da Transformada Rápida de Radon pode ser desfinida da seguinte forma:

i Criar a imagem R(180, D)

ii Criar o vetor normal $[0, 1, 0, 0]$

iii Para θ=0 até 180 faça

  (a) Criar a matriz de Radon M^{-1}
  (b) Criar o vetor $P_0 = [p, -D/2, 0, 1]$
  (c) Calcular o vetor $P_0' = M^{-1} * P_0$
  (d) Testar se existe interseção, se tiver:
    i. chamar o DDA para obter a acumulação das intensidades J na linha
    ii. $R(\theta, p) = J$
  (e) caso contrário, $R(\theta, p) = 0$

iv Retornar R

Figura 1: Imagens de entrada

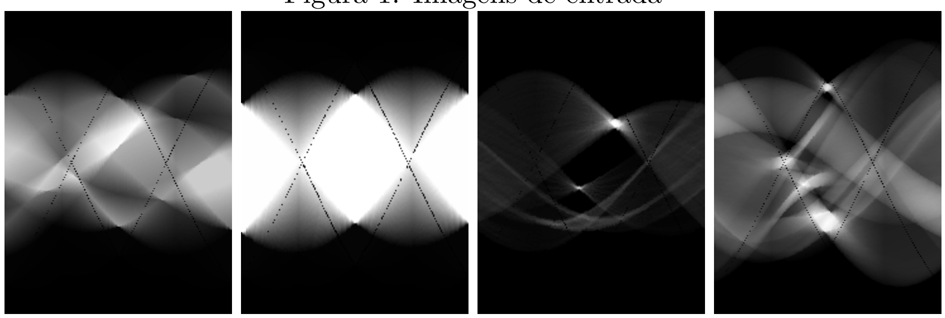


Figura 2: Transformada Rápida de Radon sem interpolação

4 Resultados

Nesta seção apresentamos os resultados obtidos em algumas imagens de teste. As imagens de saída são da forma $(180, D)$, onde D é igual a $\sqrt{nx^2 + ny^2}$, representando a diagonal da imagem. Para padronizar as figuras nesse relatório, dado que esse D depende de cada imagem, nós limitamos a altura e largura para 3cm e altura para 4cm para todas as imagens.

Na Figura 4 mostramos os resultados da Transformada Rápida de Radon em 4 imagens de teste. Nesse primeiro experimento não empregamos a interpolação, ou seja, tomando apenas o valor arredondado das coordenadas ao executar o algoritmo DDA. Como esperado, vários buracos apareceram nas imagens da transformação.

Aplicando a interpolação, nós obtivemos os resultados apresentados na Figura 4. Intrigantemente, o comportamento esperado com a aplicação do processo de interpolação (redução de buracos na imagem) não foi satisfeito para esses exemplos de teste, pelo menos visualmente. As imagens geradas aparentam ser o mesmo resultado da abordagem que não utiliza interpolação.

5 Conclusão

Neste trabalho foi apresentado uma implementação para melhoria da Transformada de Radon original. Essa variante consiste na rotação da linha ρ que projeta vários raios em

Figura 3: Imagens de entrada

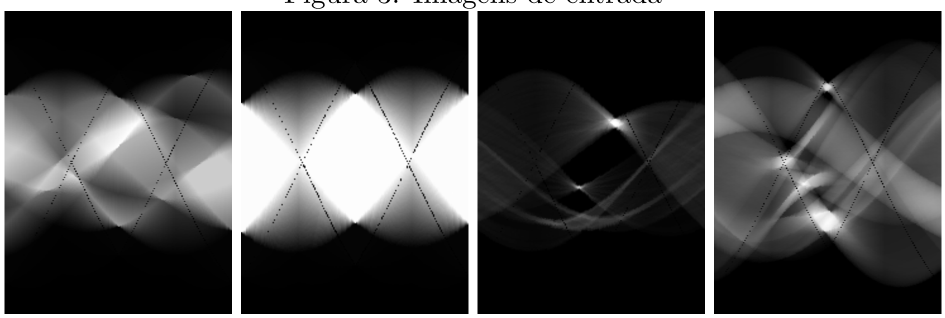


Figura 4: Transformada Rápida de Radon com interpolação

direção a imagem, ao invês de rotacionar a imagem em si, que encarece muito o processo da Transformada original. Através desta modificação, conseguimos obter melhorias de tempo significativas, como apresentadas na Tabela .... Também analisamos o uso de interpolação linear no processo do DDA acumular as intensidades de $P1$ até Pn. No entanto, os resultados visualmente não mostraram melhoria com relação ao método usando *ROUND*. Uma hipótese para os buracos que ficam tanto no resultado com interpolação e sem interpolação, é que existam descontinuidades claras nas imagens de entrada.

Apesar das dificuldades iniciais com a implementação, o trabalho fluiu de forma satisfatória. Do ponto de vista conceitual, esse trabalho é muito importante para o resto da disciplina, pois envolve mecanismos que são empregados em várias outras técnicas de Rendering (DDA, calcular intersecção, matrizes de transformação, etc.). Portanto, o aprendizado obtido nessa tarefa será muito bem utilizado para os demais problemas que iremos nos deparar.

Algumas observações técnicas para a implementação. Foi necessário utilizar a biblioteca compilada para Mac OS disponibilizada pelo monitor para o código. Além disso, mudamos também a versão do gcc no Makefile para o gcc-7, para poder funcionar.

Referências

[1] A. C. Kak and M. Slaney, *Principles of computerized tomographic imaging.* IEEE press, 1988.

[2] Radon transform, "Radon transform — Wikipedia, the free encyclopedia," 2016. [Online; accessed 21-April-2018].

[3] P. Toft, "The radon transform. theory and implementation," 1996.