Projeto Final MC970 - Paralelização da Técnicas de Meios-Tons

Vinicius P. M. Miguel - RA260731
Henrique Pareda de Souza - RA260497
Raphael Salles Vitor de Souza - RA223641

23 de junho de 2025

Sumário

1 Introdução

A técnica de meios-tons (halftone) consiste na criação de padrões formados por pontos pretos e brancos para reduzir a quantidade de níveis de cinza de uma imagem monocromática. Este método é amplamente empregado por veículos de comunicação impressos, como o jornal (Figura 1), onde podemos representar diversas tonalidades de cinza utilizando apenas tinta preta disposta em forma de círculos de raio variado.

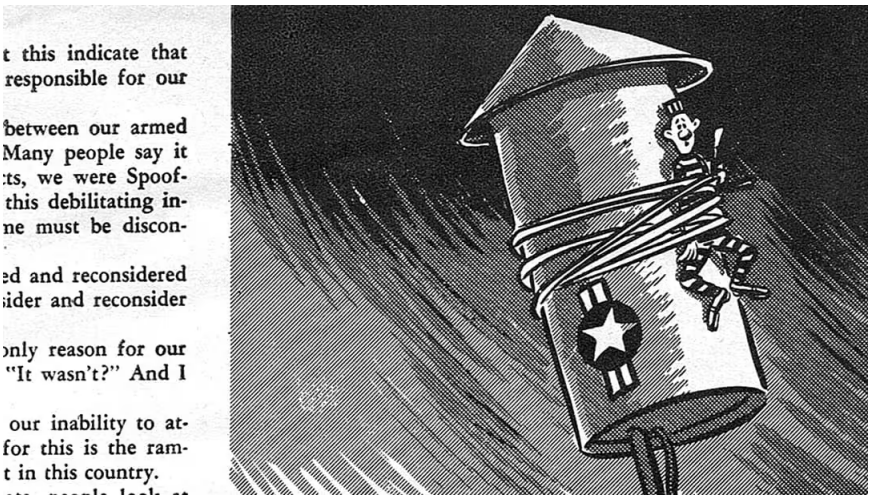

Figura 1: Aplicação de Halftone em materiais jornalísticos [1]

Em meios digitais, o halftone pode ser adaptado para transformar uma imagem monocromática de 256 níveis de cinza para uma com apenas pixels pretos e brancos, sendo muito utilizado para visualização de imagens e impressões.

Além dessas aplicações clássicas, esta técnica pode ser empregada, por exemplo, na criação de mensagens criptografadas [2] e de marcas d'água ocultas em conteúdos autorais [3].

Neste contexto, este exercício visa aplicar algumas técnicas de meios-tons por difusão de erro muito difundidas na literatura, explorando e discutindo as principais nuances de cada abordagem.

2 Formulação

2.1 Formulação

Dada uma imagem de entrada A com $[a_{min}, ..., a_{max}]$ níveis de cinza, desejamos aplicar um algoritmo de halftone que construa uma nova imagem B preta e branca, de forma que a imagem transformada fique visualmente parecida com a entrada. Este processo pode ser aplicado tanto para imagens em preto e branco quanto para imagens coloridas, sendo que, no caso das coloridas, o algoritmo é executado separadamente para cada canal de cor.

Este conteúdo está em desenvolvimento. Planejamos alterar a forma de percorrer a matriz para implementar um pipeline que permita que este código seja mais eficiente em GPUs.

A matriz de difusão armazena o peso que um determinado pixel possui na distribuição do erro. Para cada valor na matriz de difusão, atualizamos A de acordo com a Equação 1, onde x' e y' representam cada coordenada da matriz de difusão. Os métodos que serão apresentados diferem apenas quanto à escolha da matriz de difusão.

$$A(x',\ y') \ = \ A(x',\ y') + (x',\ y') \ \cdot \ erro \tag{1}$$

No escopo deste projeto, serão abordados 6 métodos de difusão de erros distintos: Floyd e Steinberg; Stevenson e Arce; Burkes; Sierra; Stucki; e Jarvis, Judice e Ninke. As suas respectivas matrizes de difusão de erro estão apresentadas nas Tabelas 1 a 6, onde $f(x, y)$ corresponde a $A(x, y, canal)$.

	$f(x,y)$	7 / 16
3 / 16	5 / 16	1 / 16

Tabela 1: Floyd e Steinberg

			$f(x,y)$		32 / 200	
12 / 200		26 / 200		30 / 200		16 / 200
	12 / 200		26 / 200		12 / 200	
5 / 200		12 / 200		12 / 200		5 / 200

Tabela 2: Stevenson e Arce

		$f(x,y)$	8 / 32	4 / 32
2 / 32	4 / 32	8 / 32	4 / 32	2 / 32

Tabela 3: Burkes

		$f(x,y)$	5 / 32	3 / 32
2 / 32	4 / 32	5 / 32	4 / 32	2 / 32
	2 / 32	3 / 32	2 / 32	

Tabela 4: Sierra

		$f(x, y)$	8 / 42	4 / 42
2 / 42	4 / 42	8 / 42	4 / 42	2 / 42
1 / 42	2 / 42	4 / 42	2 / 42	1 / 42

Tabela 5: Stucki

		$f(x, y)$	7 / 48	5 / 48
3 / 48	5 / 48	7 / 48	5 / 48	3 / 48
1 / 48	3 / 48	5 / 48	3 / 48	1 / 48

Tabela 6: Jarvis, Judice e Ninke

2.2 Implementação

2.2.1 Execução

O código referente à implementação do halftone pode ser encontrado em *1_ halftoning.py*, recebendo como parâmetro no terminal o caminho até a entrada. O programa aceita mais de uma imagem de entrada, basta passar os respectivos caminhos em sequência.

3 Implementação

Neste capítulo, apresentamos as três versões do código desenvolvidas para fins comparativos: uma versão serial, uma versão paralelizada utilizando OpenMP e uma versão implementada em CUDA. Cada uma dessas versões será explicada em detalhes nas subseções a seguir.

3.1 Versão Serial

A versão serial do código, localizada no arquivo `serial.cpp`, foi implementada como ponto de partida para o desenvolvimento das versões paralelas. Nesta implementação, o algoritmo é executado de forma sequencial, processando os dados de entrada um elemento por vez. Essa abordagem é simples e direta, mas não aproveita os recursos de paralelismo disponíveis em arquiteturas modernas.

O código foi estruturado para ser claro e modular, facilitando a compreensão e a posterior paralelização. Ele realiza as seguintes etapas principais:

- Leitura dos dados de entrada.
- Processamento sequencial dos dados, aplicando a técnica de meios-tons utilizando difusão de erro.
- Escrita dos resultados no arquivo de saída.

O algoritmo de difusão de erro utilizado na versão serial é configurado para suportar diferentes métodos de dithering, como Floyd-Steinberg, Burkes, Sierra, entre outros. A escolha do método é feita com base em um parâmetro de entrada, permitindo flexibilidade na execução.

Essa versão serve como base para medir o desempenho inicial e comparar os ganhos obtidos com as versões paralelas. O código completo da implementação pode ser encontrado abaixo:

3.2 Versão Paralela com OpenMP

A versão paralela utilizando OpenMP foi desenvolvida a partir da versão serial, com o objetivo de explorar o paralelismo em CPUs multicore. O código-fonte desta versão é essencialmente o mesmo da versão serial, diferindo apenas pela inclusão de diretivas de paralelização do OpenMP, que são ativadas em tempo de compilação. Os resultados e o impacto dessas alterações serão discutidos na Seção 4.

3.3 Versão em CUDA

A versão em CUDA, que será implementada no futuro, tem como objetivo explorar o paralelismo massivo oferecido por GPUs. Esta subseção será preenchida com os detalhes da implementação assim que o código estiver concluído.

4 Speedup

Nesta seção, apresentamos as métricas de Speedup obtidas durante os experimentos. O Speedup é calculado como a razão entre o tempo de execução da versão serial e o tempo de execução das versões paralelas (OpenMP e CUDA). Foram consideradas duas abordagens principais:

- **Speedup por Método:** Para cada método de dithering (*FloydSteinberg*, *StevensonArce*, *Burkes*, *Sierra*, *Stucki*, *JarvisJudiceNinke*), calculamos o Speedup médio considerando todas as imagens processadas.
- **Speedup por Tamanho de Imagem:** Para cada tamanho de imagem (*e.g.*, 1920x1080, 1280x720), calculamos o Speedup médio considerando todos os métodos de dithering aplicados.

Os resultados obtidos estão destacados nas Tabelas 7 e 8. Os campos em vermelho indicam os valores que ainda precisam ser preenchidos com os dados experimentais.

Tabela 7: Resultados de Speedup por Método

Método	Serial (s)	OpenMP (s)	CUDA (s)	Speedup OpenMP	Speedup CUDA
FloydSteinberg	x.xx	x.xx	x.xx	x.xx	x.xx
StevensonArce	x.xx	x.xx	x.xx	x.xx	x.xx
Burkes	x.xx	x.xx	x.xx	x.xx	x.xx
Sierra	x.xx	x.xx	x.xx	x.xx	x.xx
Stucki	x.xx	x.xx	x.xx	x.xx	x.xx
JarvisJudiceNinke	x.xx	x.xx	x.xx	x.xx	x.xx

Tabela 8: Resultados de Speedup por Tamanho de Imagem

Tamanho da Imagem	Serial (s)	OpenMP (s)	CUDA (s)	Speedup OpenMP	Speedup CUDA
800x600	x.xx	x.xx	x.xx	x.xx	x.xx
1280x720	x.xx	x.xx	x.xx	x.xx	x.xx
1920x1080	x.xx	x.xx	x.xx	x.xx	x.xx

Tabela 9: Resultados de Eficiência por Método

Método	Eficiência OpenMP	Eficiência CUDA
FloydSteinberg	x.xx	x.xx
StevensonArce	x.xx	x.xx
Burkes	x.xx	x.xx
Sierra	x.xx	x.xx
Stucki	x.xx	x.xx
JarvisJudiceNinke	x.xx	x.xx

Tabela 10: Resultados de Eficiência por Tamanho de Imagem

Tamanho da Imagem	Eficiência OpenMP	Eficiência CUDA
800x600	x.xx	x.xx
1280x720	x.xx	x.xx
1920x1080	x.xx	x.xx

5 Conclusão

Referências

[1] C. Sperandio, "Setting the right tone." `https://www.retrosupply.co/blogs/tutorials/setting-the-right-tone`.

[2] Z. Wang, G. R. Arce, and G. Di Crescenzo, "Halftone visual cryptography via error diffusion," *IEEE Transactions on Information Forensics and Security*, vol. 4, no. 3, pp. 383–396, 2009.

[3] M. S. Fu and O. Au, "Data hiding watermarking for halftone images," *IEEE Transactions on Image Processing*, vol. 11, no. 4, pp. 477–484, 2002.