Trabalho 2 - MC920 - Técnicas de Pontilhado

Luiz Eduardo Cartolano - 183012

Maio de 2019

Resumo

As técnicas de pontilhado visam reduzir a quantidade de cores (quantização de cores) utilizadas para exibir uma imagem, procurando manter uma boa percepção por parte do usuário. Neste trabalho foram usadas duas técnicas diferentes de pontilhado, a ordenada e a com difusão de erro

A partir delas foi possível concluir que existe um *trade-off* óbvio entre qualidade (e fidelidade) da imagem de saída e custo computacional. Sendo que a técnica de pontilhado ordenada é menos custosa, enquanto que a técnica por difusão de erro apresenta uma maior qualidade de *outputs*.

1 Introdução

O objetivo deste trabalho é a implementação de técnicas de pontilhado que visam reduzir a quantidade de cores (quantização de cores) utilizadas para exibir uma imagem, procurando manter uma boa percepção por parte do usuário. Neste trabalho foram usadas duas técnicas diferentes de pontilhado, a ordenada e a com difusão de erro, como solicitado em 1.

As técnicas de pontilhado são muito usadas no nosso dia a dia, estando presente em muitas situações sem que tenhamos tal consentimento. Uma aplicação comum é exibir com mais precisão as imagens que contêm um intervalo maior de cores do que o *hardware* é capaz de exibir. Por exemplo, o pontilhamento pode ser usado para exibir uma imagem fotográfica contendo milhões de cores em um *hardware* de vídeo que só é capaz de exibir 256 cores por vez.

Este relatório encontra-se organizado da seguinte forma: a Seção 2 apresenta a maneira como o trabalho foi implementado a fim de solucionar o problema do enunciado. A Seção 3 apresenta os resultados obtidos pela implementação feita para determinadas imagens de entrada. As discussões a respeito dos resultados encontrados estão dispostas na Seção 4, e as conclusões na Seção 5. As imagens, tabelas e equações apresentadas encontram-se nos Anexos ao final do relatório.

2 Implementação

2.1 Uso Do Código

O arquivo *LuizCartolano_183012.zip* submetido no *Google Classroom*, possui a seguinte estrutura:

```
proj2-2019
├── basicImage
├── constants
├── inputs
├── logger_lib
├── main
└── outputs
```

Os arquivos *constants.py* e *logger_lib.py* são códigos auxiliares que ajudam no gerenciamento do projeto, o primeiro guarda os *paths* importantes para as pastas de *inputs* e *outputs*, enquanto o segundo armazena em um arquivo *.log* todas as ações executadas pelo programa (e eventuais erros).

Os arquivos *basicImage.py* e *main.py* são os principais e podem ser considerados o "coração"do projeto. O primeiro, foi montado com a intenção de ser uma biblioteca que implemente funções para manipulação de imagens, portanto, ele contém as funções solicitadas para esse projeto (aplicação das técnicas ordenadas

e com difusão de erro) e também funções solicitadas para os projetos antigos, além de manipulações básicas que podem ser feitas com imagens (como leituras ou armazenamentos). O segundo arquivo citado, por sua vez, contém o fluxo de execução, sendo inclusive o responsável por fazer o projeto funcionar, como será apresentado adiante.

Para executar o projeto basta executar a linha de comando *python*3.7 *main.py*, e então todos as imagens de saída serão salvas na pasta de *outputs*, como explicaremos na próxima subseção. Para garantir que o projeto execute como esperado é necessário possuir a versão 3.7 do *python* e também as seguintes bibliotecas: *datetime*, *cv2*, *numpy*, *scipy* e *matplotlib*, as demais bibliotecas usadas não necessitam de *download*.

2.2 Detalhes da Implementação

Halftone(meio-tom) é uma técnica de processamento de imagens que emprega padrões de pontos pretos ou brancos para reduzir o número de níveis de cinza de uma representação. Ele pode ser aplicado usando técnicas de pontilhado ordenado, por difusão de erro, entre outras. A seguir, serão comentadas as duas técnicas usadas neste trabalho.

2.2.1 Pontilhado Ordenado

Para a técnica de pontilhado ordenado usou-se duas matrizes diferentes a fim de gerar os padrões que seriam usados para aproximar os níveis de cinza. A primeira delas, foi a Matriz 1 que gera os padrões que podem ser observados na Figura 12. A segunda matriz usada, foi a matriz de Bayer, que é bastante conhecida no meio de processamento e é representada pela Matriz 2.

Cada conjunto de cinza é representado por um dos padrões de preto e branco. No modelo de implementação adotado são usados, para a primeira matriz nove limiares (e dezesseis para a segunda). Assim cada pixel da imagem original irá corresponder a um dos padrões. Uma implementação, em alto nível, para a matriz $3x3$ está descrito no Algoritmo 1.

Algoritmo 1: Algoritmo para aplicação da técnica de pontilhado ordenado a partir de padrões gerados por uma matriz $3x3$.

```
   Data: img: imagem original
   Result: new_img: imagem pós aplicação
 1 masks = get_masks(matriz) ;
 2 h,l = img.shape ;
 3 masked = ceil((10.0 / 255.0) * img) ;
 4 flag = 0 ;
 5 for i ←0 to h do
 6 |   if flag = 0 then
 7 |   |   for j ←0 to l do
 8 |   |   |   new_img[s_line:f_line,s_col:f_col] = masks[int(masked[i,j]) - 1][:,:] ;
 9 |   |   end
10 |   |   flag = 1 ;
11 |   end
12 |   else
13 |   |   for j ←l − 1 to 0 do
14 |   |   |   new_img[s_line:f_line,s_col:f_col] = masks[int(masked[i,j]) - 1][:,:] ;
15 |   |   end
16 |   |   flag = 0 ;
17 |   end
18 end
```

2.2.2 Pontilhado por Difusão de Erro

O pontilhado com difusão de erro é uma técnica de *halftone* que procura distribuir a diferença entre o valor exato de cada pixel e seu valor aproximado a um conjunto de pixels adjacentes. Diversas técnicas

foram desenvolvidas com esta proposta, sendo o algoritmo de *Floyd-Steinberg* o primeiro desenvolvido utilizando esta abordagem.

De modo simplificado, podemos descrever o funcionamento do algoritmo da seguinte maneira: percorra todos os pixels da imagem, para cada pixel, se seu valor for maior do que 128, troque-o para 255 (branco). Caso contrário, troque-o para 0 (preto). Armazene o erro e distribua os erros de acordo com a Matriz 3, é importante notar que, dependendo do modo que se percorre a matriz será necessário redistribuir o peso dos erros.

Uma implementação, em alto nível, está descrito no Algoritmo 2. A implementação apresentada, percorre a matriz sempre da esquerda para a direita, contudo, fazer tal caminho de maneira alternada traz resultados melhores e, inclusive, foi a técnica abordade nesse trabalho.

Algoritmo 2: Algoritmo para aplicação da técnica de *Floyd-Steinberg*.

```
Data: img: imagem original
Result: new_img: imagem pós aplicação
h,l = img.shape ;
for i ←0 to h do
    for j ←0 to l do
        if img[i,j] > 128 then
            new_img[i,j] = 0 ;
            err = img[i,j] - 255 ;
        end
        else
            new_img[i,j] = 255 ;
            err = img[i,j] ;
        end
        img[i,j+1] += err * (7/16) ;
        img[i+1,j+1] += err * (1/16) ;
        img[i+1,j] += err * (5/16) ;
        img[i+1,j-1] += err * (3/16) ;
    end
end
```

3 Resultados

Após execução do código são obtidos três novas imagens para cada imagem original de entrada. São elas, a imagem pós aplicação da técnica de pontilhado ordenada usando os padrões gerados por uma matriz $3x3$ (vista na Matriz 1), a imagem pós aplicação da técnica ordenada usando os padrões gerados por uma matriz de *Bayer* (vista na Matriz 2) e, por fim, a imagem obtida após a aplicação da técnica de difusão de erro de *Floyd-Steinberg*. Estes podem ser vistos, respectivamente, nas Figuras 2, 4, 6. Sendo a Figura 1 a imagem usada como *input* para as técnicas.

Todas as figuras citadas anteriormente foram obtidas percorrendo a matriz da imagem em dois sentidos, em um momento da esquerda para a direita e em outro da direita para a esquerda, já as Figuras 3, 5 e 7 foram obtidas percorrendo a matriz sempre da esquerda para a direita. É notável que elas são mais escuras e com mais ruídos.

A Figura 2 no mostra que a imagem traz uma percepção mais suave aos olhos, contudo, é visível a presença de vários pequenos quadrados na imagem. Já a Figura 4 nos traz uma imagem que da uma percepção mais escura (níveis de cinza mais homogêneos), contudo, assim como para a figura anterior, ainda são bem perceptíveis a presença de pequenos quadrados na imagem. Por fim, a Figura 6, é a mais clara entre as três, e se assemelha bastante a imagem original (vista na Figura 1). Ao contrário das anteriores ela não apresenta, com tanta nitidez, os pequenos quadrados, contudo, é notável a presença de alguns pontos pretos ao longo da imagem.

O tempo de processamento gasto por cada uma das técnicas pode ser observado na Tabela 1, da qual é possível concluir que a técnica de pontilhado ordenado foi a mais rápida. Já a técnica por difusão de erro de *Floyd-Steinberg* foi a mais lenta, logo, é que exige mais processamento.

4 Discussão

A técnica de pontilhado ordenado adotada faz com que seja gerada uma imagem de saída que possui uma resolução muito maior do que a imagem original, portanto, a tendência é que ocorra uma perda de nitidez. Tal situação é visível quando comparamos as imagens das Figuras 2 e 4 com a original vista na Figura 1. Outro detalhe interessante que é possível notar nas imagens obtidas a partir dessa técnica é que os *outputs* são consideravelmente diferentes do *inputs*, e mesmo entre si, sendo a Figura 2, obtida através dos padrões gerados pela Matriz 1 a mais nítida dentre elas. Entretando, a imagem gerada pela aplicação dos padrões de *Bayer* tem maior semelhança com a original. Um detalhe que pode ser observado em ambas é a presença de pequenos quadrados (como comentado na Seção 3), estes provavelmente são frutos dos padrões que substituem o pixel original.

A difusão de erro é uma comparação entre pixels, baseada em um limiar, gerando pontos pretos ou brancos conforme o resultado da comparação. Nesta técnica, diferente do método ordenado, distribui a defasagem dos valores aos pixels vizinhos, elevando, consideravelmente a qualidade do método. A melhora na imagem de saída é visível ao observarmos a Figura 6, contudo, como mostrado na Tabela 1, esse melhor desempenho vem acompanhado de um alto custo computacional. Também é notável que a figura obtida por esse algoritmo é mais nítida dentre as três, muito em questão dela preservar a resolução original.

Uma última comparação que pode ser feita está relacionada a maneira como a matriz é percorrida para aplicação da técnica, para verificar tal diferença, pode-se analisar, par a par, as Figuras 2 e 3, as Figuras 4 e 5 e as Figuras 6 e 7. Para o primeiro par, é possível perceber que a imagem percorrida em sentido único é mais escura, especialmente nos pontos de mudança mais significativa nos níveis de cinza de pixels adjacentes, como acontece no "nariz do babuino". Já o último par, que é obtido por uma técnica diferente do primeiro, é o que as mudanças são mais significativas. Além de possuir uma menor nitidez e fornecer uma percepção de ser mais escura, a imagem obtida no percorrimento unidirecional apresenta mais ruídos, o que diminui sua qualidade.

Ainda que as soluções apresentadas tenham apresentados resultados significativos e sejam executadas de maneira eficiente, o programa apresenta limitações. Uma delas é o fato de que o algoritmo não pode ser aplicado para imagens que não sejam as pré-definidas na pasta de *inputs*.

Outra limitação encontrada pode ser observada quando se trabalha com imagens muito claras, como a Figura 8. As técnicas adotas geram uma imagem com baixíssima nitidez sendo, praticamente impossível, observar os detalhes originais. Como pode ser visto ao analisar as Figuras 9, 10 e 11, a técnica de pontilhado ordenado a partir dos padrões de *Bayer* foi a que manteve o texto mais legível, enquanto que a técnica por difusão de erro foi a que representou mais fielmente os níveis de cinza da imagem original, o que era esperado.

Por fim, uma última limitação que pode ser citada, no âmbito da implementação, é com relação aos métodos de comparação a serem aplicados as imagens de saída do *half-toning* ordenado, uma vez que as estas são muito maiores do que as originais, as comparações tem que ser feitas majoritariamente em questões qualitativas.

Um segundo problema causado pelo tamanho das imagens, diz respeito a exibição delas neste relatório, para que fosse possível exibí-las, foi preciso realizar duas modificações com relação a imagem gerada pelo programa em *python*, sendo elas a conversão para a extensão *.png* e o *resize* delas. Uma consequência dessas aplicações foi que os resultados visuais aqui apresentados não representam fielmente os dados gerados na implementação. Outro problema da extensão *.png* é que ela pode realizar uma compressão com perda, alterando o valor original dos pixels.

5 Conclusão

Após a finalização do trabalho, pode-se concluir que o *half-toning* é um sistema de tratamento de imagem bastante interessante e de fácil implementação. Além disso, ele possui diversas técnicas (que vão além das abordadas por este relatório), de modo a fornecer uma abrangência interessante para os seus usuários.

É importante ressaltar, como visto na Seção 4, que cada técnica possui suas vantagens e desvantagens, desse modo, a escolha por qual abordagem utilizar depende de um conjunto de fatores, que vão desde a aplicação desejada até a máquina que será utilizada.

Das técnicas usadas neste trabalho foi possível concluir que existe um *trade-off* óbvio entre qualidade

(e fidelidade) da imagem de saída e custo computacional. Sendo que a técnica de pontilhado ordenada é menos custosa, enquanto que a técnica por difusão de erro apresenta uma maior qualidade de *outputs*.

Referências

[1] PEDRINI Helio. Projeto 2. Disponível em: http://www.ic.unicamp.br/~helio/disciplinas/MC920/trabalho2.pdf, Acesso em: 09-04-2019.

[2] GARCIA Pedro. Técnica de pontilhado floyd-steinberg. Disponível em: http://www.sawp.com.br/blog/?p=1041, Acesso em: 07-05-2019.

[3] GARCIA Pedro. Técnica de pontilhado ordenada. Disponível em: http://www.sawp.com.br/blog/?p=940, Acesso em: 07-05-2019.

ANEXOS

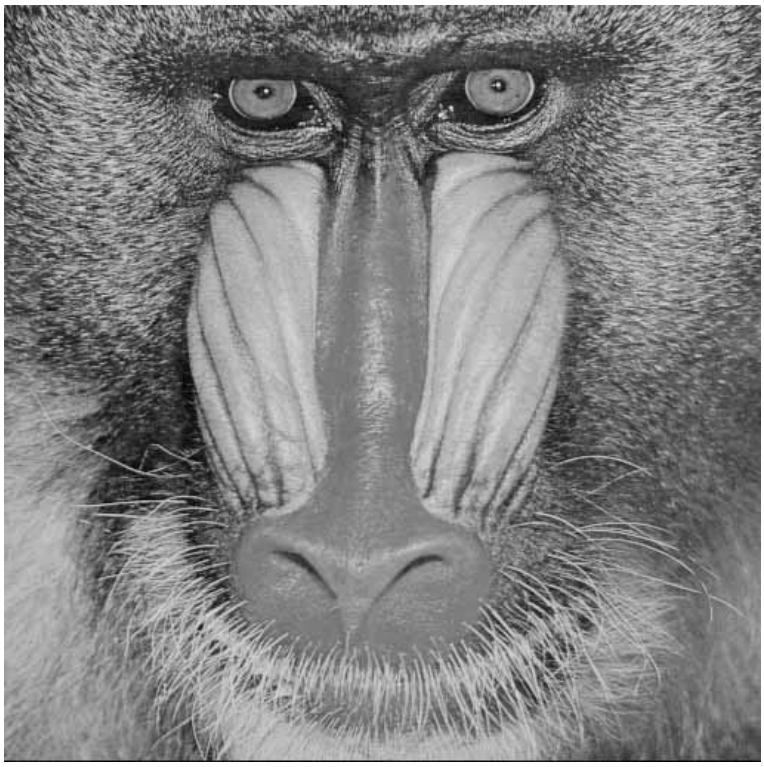

Figura (1) Imagem original.

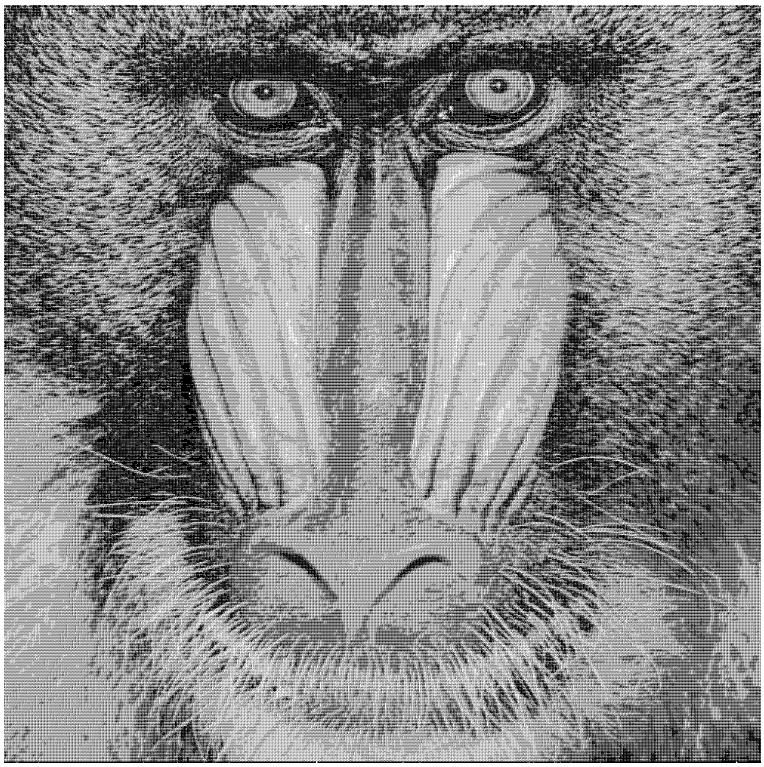

Figura (2) Imagem após técnica ordenada com matriz 3x3 e percorrendo matriz em dois sentidos.

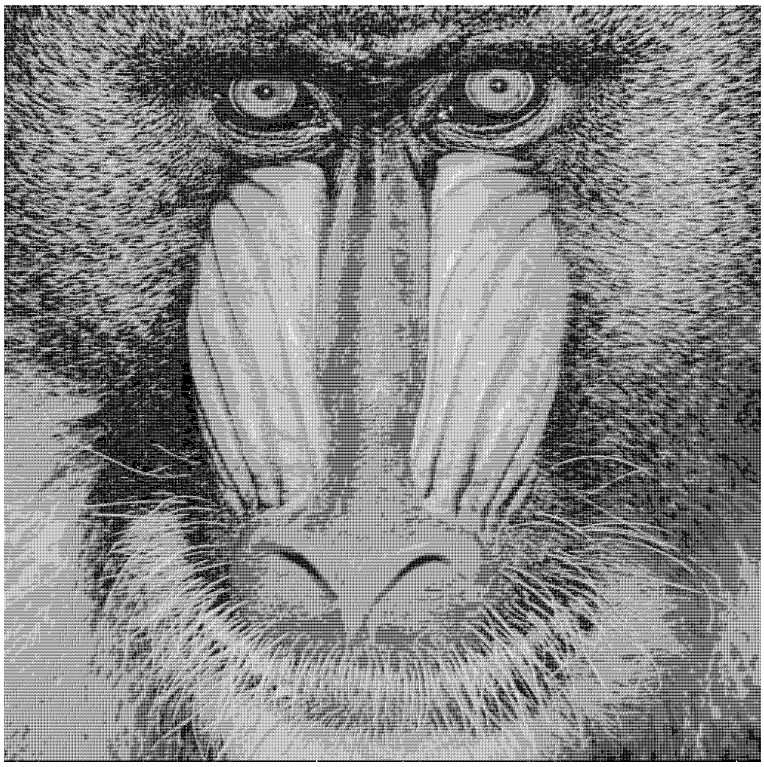

Figura (3) Imagem após técnica ordenada com matriz 3x3 e percorrendo matriz em sentido único.

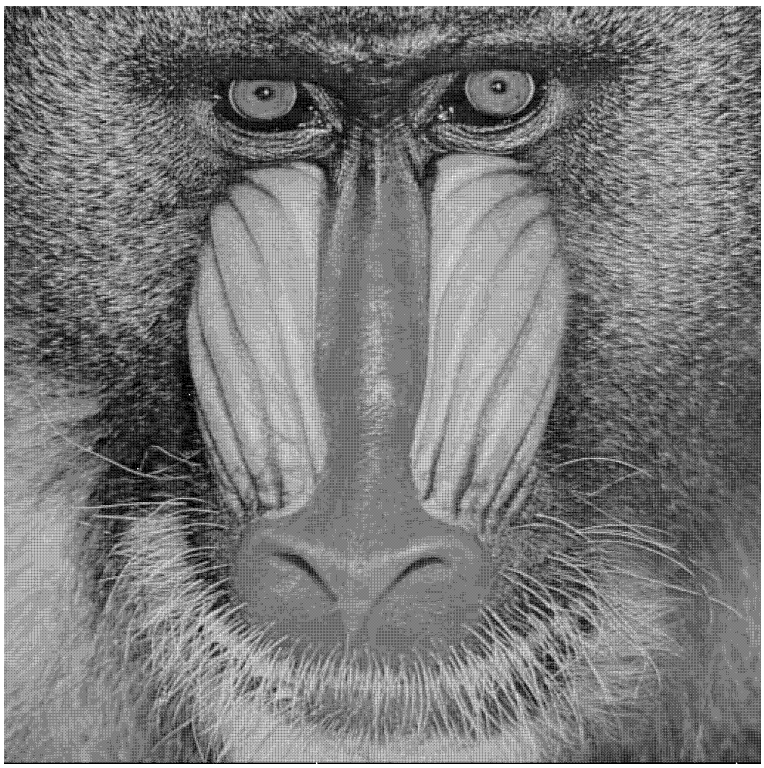

Figura (4) Imagem após técnica ordenada com matriz de Bayer e percorrendo matriz em dois sentidos.

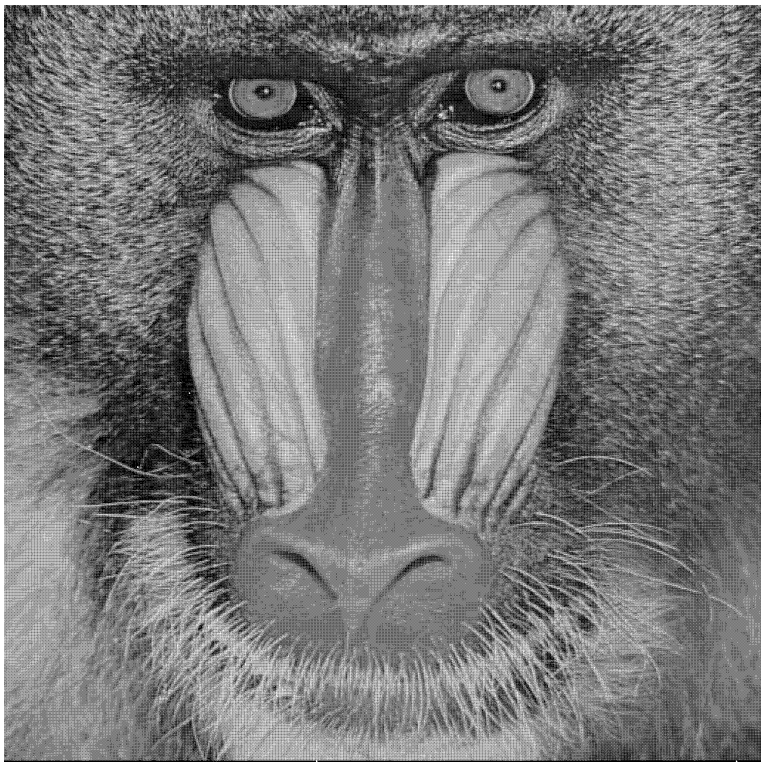

Figura (5) Imagem após técnica ordenada com matriz de Bayer e percorrendo matriz em sentido único.

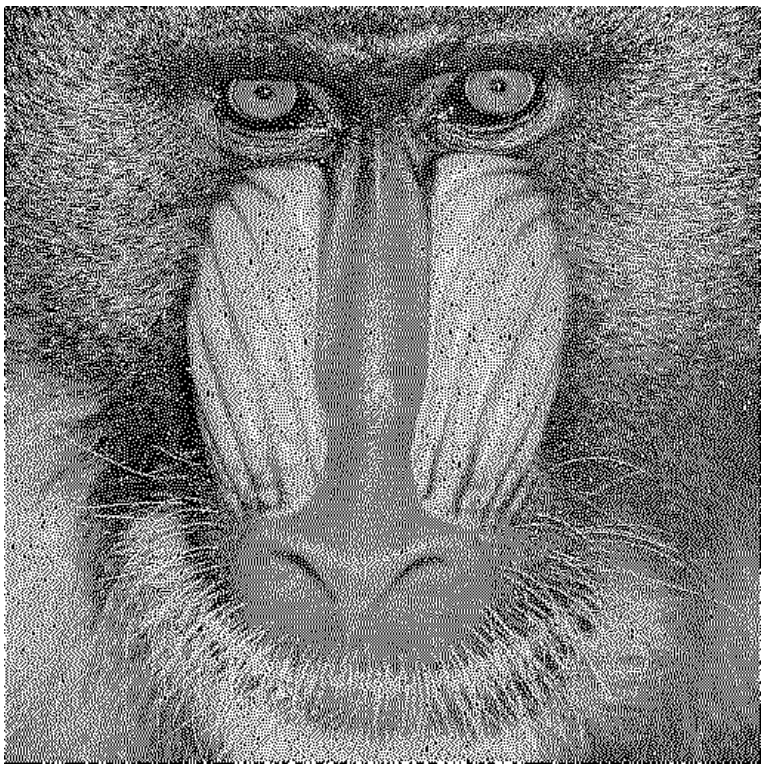

Figura (6) Imagem após técnica com difusão de erro e percorrendo matriz em dois sentidos.

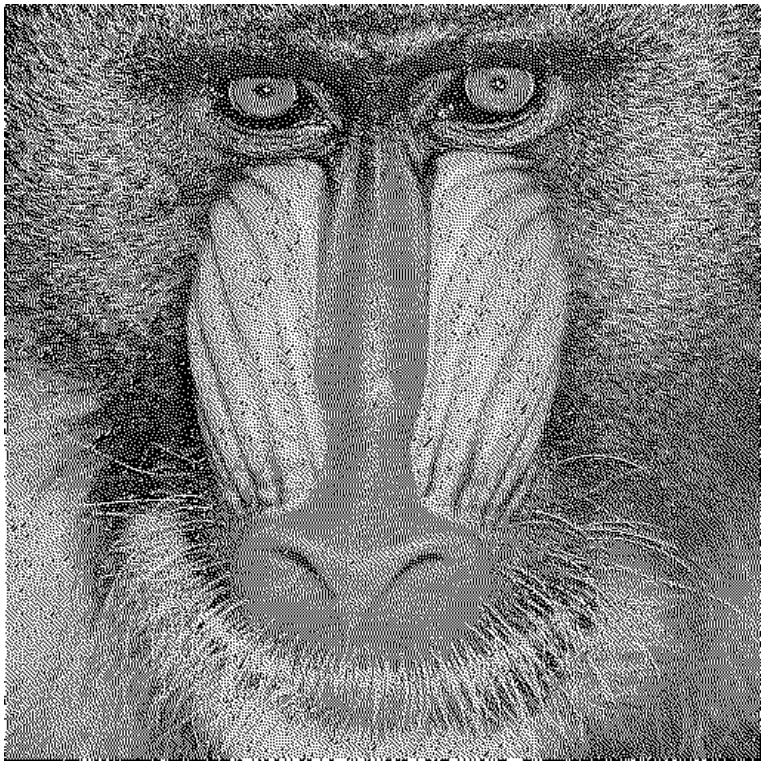

Figura (7) Imagem após técnica com difusão de erro e percorrendo matriz em sentido único.

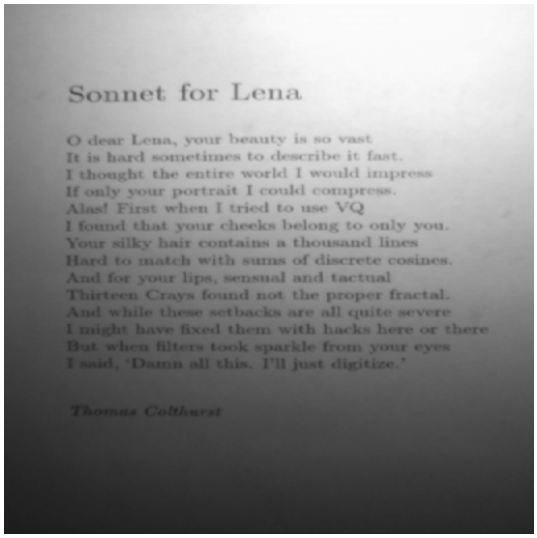

Figura (8) Imagem original.

$$\begin{bmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{bmatrix} \tag{1}$$

(1) Padrão para a máscara $3x3$.

Tabela (1) Tabela com o desempenho de cada uma das técnicas implementadas.

Funcao	**Média(μs)**	**Desvio (μs)**
Halftone 3x3	94.27	25.57
Halftone 4x4	67.74	16.70
Floyd-Steinberg	123.15	71.00

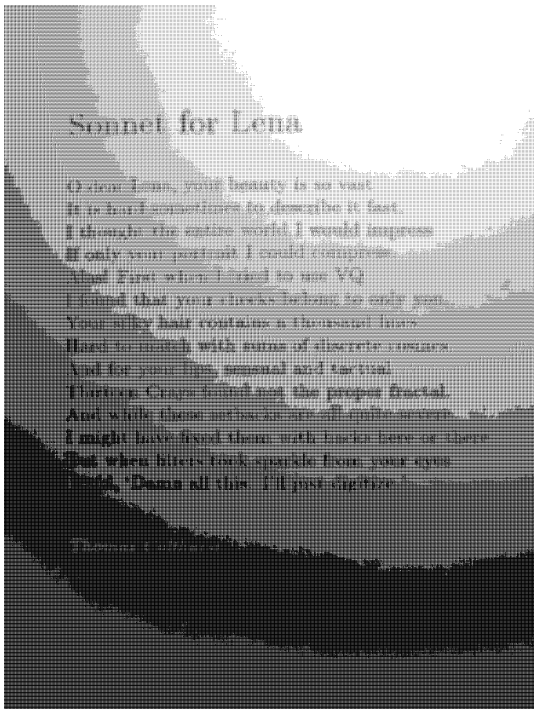

Figura (9) Imagem após técnica ordenada com matriz 3x3.

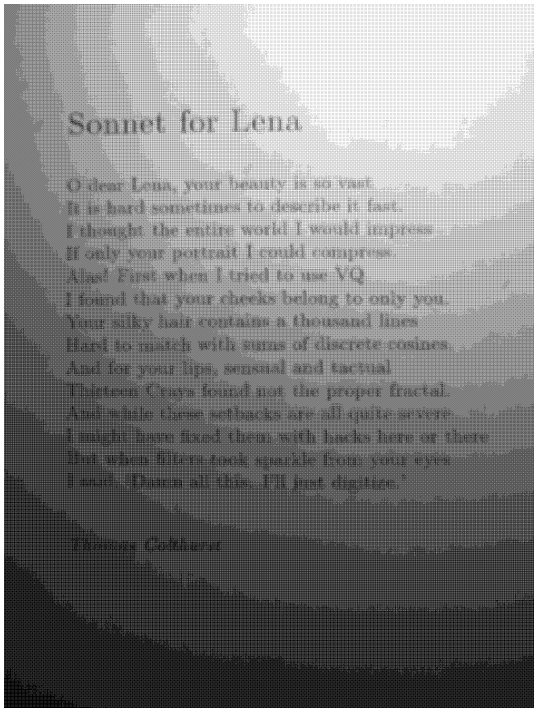

Figura (10) Imagem após técnica ordenada com matriz de Bayer.

$$\begin{bmatrix} 0 & 12 & 3 & 15 \\ 8 & 4 & 11 & 7 \\ 2 & 14 & 1 & 13 \\ 10 & 6 & 9 & 5 \end{bmatrix} \tag{2}$$

(2) Padrão para a máscara de Bayer.

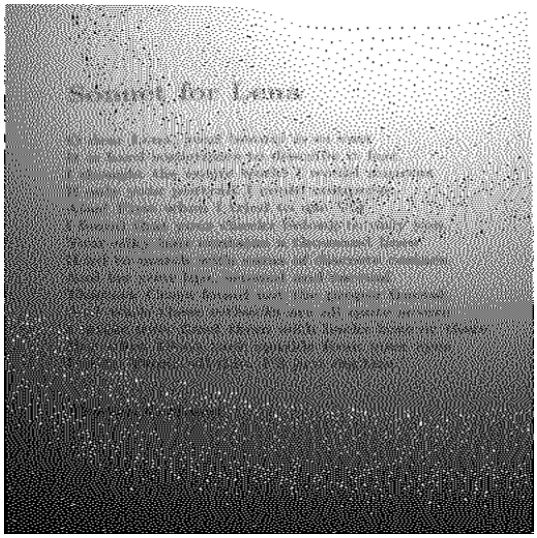

Figura (11) Imagem após técnica com difusão de erro.

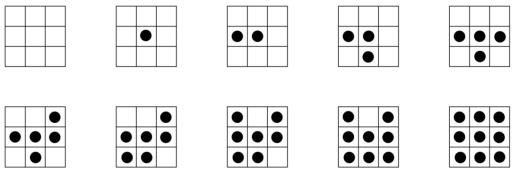

Figura (12) Dez padrões de 3x3 pixels.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & f(x,y) & 7/16 \\ 3/16 & 5/16 & 1/16 \end{bmatrix} \tag{3}$$

(3) Distribuição do erro no algoritmo de *Floyd-Steinberg*.