

Trabalho 2

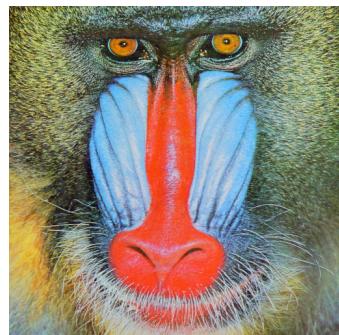
Tiago de Paula Alves (187679)
tiagodepalves@gmail.com

31 de março de 2025

1 Introdução

Este trabalho teve como objetivo a implementação de técnicas de pontilhado com difusão de erros. A ideia é reduzir a quantidade de cores tentando manter a imagem o mais próximo possível da original. Isso é feito reduzindo a intensidade para seu limite mais próximo, ao longo de um caminho pela imagem, mas aplicando uma distribuição de erros na vizinhança do pixel.

A figura 1 apresenta as imagens base deste trabalho, usadas para análise e discussão das formas diferentes de aplicar o pontilhado. As figuras 1a e 1b são 512×512 , enquanto a figura 1c é 256×256 e a figura 1d é 1024×768 . Ao todo serão aplicadas 6 distribuições de erro com 4 curvas diferentes em cada figura. As distribuições de erros estão apresentadas na seção 3.3, enquanto as curvas de varrimento podem ser encontradas na seção 3.2.



(a) `imagens/baboon.png`



(b) `imagens/peppers.png`



(c) `imagens/monalisa.png`



(d) `imagens/watch.png`

Figura 1: Imagens base da comparação dos filtros.

2 O Programa

Além das bibliotecas padrão de Python, foram utilizados os pacotes NumPy¹ e, de forma opcional, Numba.²

¹ NumPy. URL: <https://numpy.org/>.

² Numba. URL: <https://numba.pydata.org/>.

2.1 Código Fonte

O programa foi desenvolvido em Python 3.8, mas deveria funcionar com as versões 3.7 e 3.9 também. Além disso, o código fonte foi separado nos seguintes arquivos:

main.py É o corpo do programa, responsável por processar os comandos e as opções da linha de comando.

lib Pacote interno com as operações de pontilhado.

lib/_init_.py Funções gerais de aplicação do pontilhado em imagens coloridas e em escala de cinza, com as opções de varredura.

lib/nb.py Mock up do decorador `@numba.jit`,³ para que a ferramenta funcione com ou sem a biblioteca Numba.

lib/horizontal.py Implementação das varreduras horizontais: varredura unidirecional e a alternada.

lib/direcao.py Controle de direção para varreduras não horizontais (espiral e curvas de Hilbert).

lib/espiral.py Implementação da varredura em espiral.

lib/hilbert.py Varredura seguindo uma curva de Hilbert.

dists.py Definição das distribuições de erro ao longo da operação de meios-tonos.

inout.py Funções que tratam da entrada e saída do programa, como leitura e escrita de arquivos de imagem e também da apresentação da imagem em uma janela gráfica.

tipos.py Definição de alguns tipos para checagem estática com `mypy`.⁴

Todas as figuras base utilizadas neste relatório podem ser encontradas na pasta `imagens` do código fonte, como descrito nos rótulos da figura 1. Além disso, foi disponibilizado também um *script* em bash, `run.sh`, que realiza todos os processamentos requeridos em cada uma das imagens na pasta.

2.2 Execução

A execução deve ser feita através do interpretador de Python 3.7+. A única entrada obrigatória é o caminho para a imagem PNG que será processada. Ao final da execução, a imagem resultante será exibida na tela.

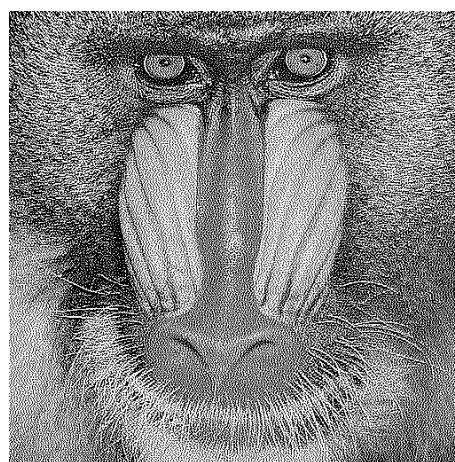


Figura 2: Aplicação de pontilhado com a `baboon.png`.

³ Compiling Python Code with `@jit`. URL: <https://numba.readthedocs.io/en/stable/user/jit.html>.

⁴ Mypy: Optional Static Typing for Python. URL: <http://mypy-lang.org/>.

Os argumentos opcionais podem ser vistos com `$ python3 main.py --help`. A mais importante das opções é `--output`, ou `-o`, que salva o resultado em um arquivo PNG em vez de exibir na tela. Se é desejável tanto a exibição da imagem quanto o salvamento no arquivo, o argumento `--force-show` ou `-f` pode ser usado. Também existe a opção `--grayscale` ou `-g` que faz o processamento em escala de cinza.

As outras opções são referentes à técnica de pontilhado. A flag `--varredura` ou `-v` controla a forma de varredura na imagem, como descrito na seção 3.2. A distribuição de erros aplicada pode ser modificada com `--distribuicao` ou `-d` e aceita o nome de qualquer um dos idealizadores da distribuição.

Por exemplo, o comando abaixo apresenta a figura 2 em uma nova janela gráfica.

```
$ python3 main.py imagens/baboon.png -g -d ninke
```

2.3 A biblioteca Numba

A ferramenta foi desenvolvida com requerimento mínimo sendo apenas o NumPy. No entanto, se presente, a biblioteca Numba causa um grande diferencial no tempo de execução das técnicas de pontilhado (cerca de trinta vezes mais rápido na máquina de desenvolvimento). Por isso, a biblioteca é fortemente recomendada na execução do código.

3 Implementação

3.1 Técnica de Meios-Tons com Difusão de Erro

Essa técnica de pontilhado se baseia em alterar cada pixel com escala de 8 bits para escala de 2 bits, levando para o seu nível de intensidade mais próximo. No entanto, ao longo do processo, o erro é calculado em relação ao novo pixel, que é então distribuído na sua vizinhança, influenciando as aplicações nos pixels seguintes.

Assim, quando um pixel resulta em valor muito distante do original, seus vizinhos são reduzidos ou incrementados, aumentando a chance de que eles sejam transformados para outro nível. Isso faz com que a vizinhança mantenha um pouco mais da distribuição local de intensidade, deixando a imagem resultante mais similar à original, considerando o nosso sistema visual.

A distribuição de erros pode ser feita de várias formas, como pode ser visto na seção 3.3.

3.2 Formas de Varredura

A técnica de pontilhado com distribuição de erros altera a imagem enquanto é aplicada, o que faz com que o caminhos diferentes levam à resultados distintos. Por isso, o pontilhado foi implementado seguindo quatro formas de varredura, apresentadas na figura 3. Apesar disso, as varreduras que serão mais discutidas aqui serão a unidirecional (3a) e a alternada (3b).

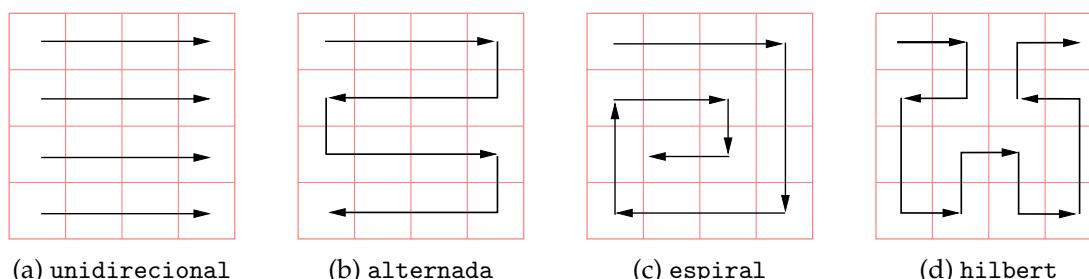


Figura 3: Argumentos válidos para `--varredura` ou `-v`.

Para a varredura alternada, a distribuição é aplicada em duas orientações, normal e invertida horizontalmente, enquanto para as curvas em espiral e de Hilbert a distribuição é considerada em quatro orientações, seguindo as rotações sucessivas de 90° na matriz.

3.3 Distribuições de Erro

A distribuições são escolhidas na linha de comando pelo nome de um de seus idealizadores ou de todos juntos, separados por "_". Assim, as seguintes opções são equivalentes:

```
$ python3 main.py imagens/peppers.png -d jarvis
# ou
$ python3 main.py imagens/peppers.png -d Judice
# ou
$ python3 main.py imagens/peppers.png -d JARVIS_judice_Ninke
```

As distribuições estão apresentadas na figura 4 abaixo.

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

(a) Floyd e Steinberg

		$f(x, y)$		$\frac{32}{200}$	
$\frac{12}{200}$		$\frac{26}{200}$		$\frac{30}{200}$	$\frac{16}{200}$
	$\frac{12}{200}$		$\frac{26}{200}$		$\frac{12}{200}$
$\frac{5}{200}$		$\frac{12}{200}$		$\frac{12}{200}$	$\frac{5}{200}$

(b) Stevenson e Arce

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

(c) Burkes

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

(d) 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

(e) 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

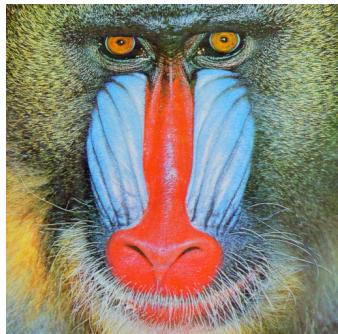
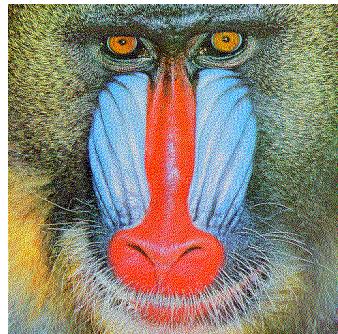
(f) Jarvis, Judice e Ninke

Figura 4: Distribuições de erro aplicadas neste trabalho.

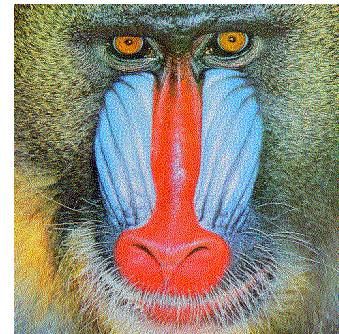
4 Resultados

Os resultados da implementação do pontilhado serão comparadas pelas métricas *Root Mean Square Error* (RMSE), *Peak Signal to Noise Ratio* (PSNR), *Signal to Noise Ratio* (SNR) e covariância.

4.1 Varredura Unidirecional e Alternada

(a) `figuras/baboon.png`

(b) Varredura unidirecional.



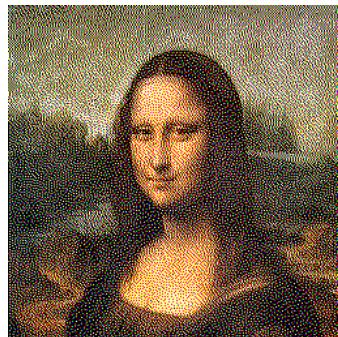
(c) Varredura alternada.

(d) `figuras/peppers.png`

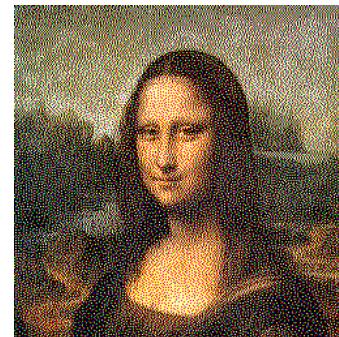
(e) Varredura unidirecional.



(f) Varredura alternada.

(g) `figuras/monalisa.png`

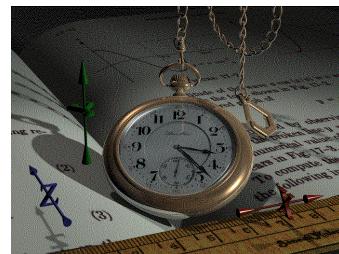
(h) Varredura unidirecional.



(i) Varredura alternada.

(j) `figuras/watch.png`

(k) Varredura unidirecional.



(l) Varredura alternada.

Figura 5: Aplicação da distribuição de Floyd e Steinberg com as duas varreduras.

Na figura 5, os resultados das duas varreduras são visualmente muito semelhantes e as métricas da tabela 1 corroboram com essa ideia, mostrando uma variação pequena de uma opção

para a outra. A maior diferença visual aparece na imagem `peppers.png`, onde a varredura unidirecional (5e) gera pequenos artefatos em formato de linhas horizontais, principalmente no pimentão vermelho.

Tabela 1: Comparativo entre os resultados da figura 5.

Figura	Varredura	RMSE	SNR (dB)	PSNR (dB)	Correlação
<code>baboon.png</code>	Unidirecional	10.367	-0.004	27.818	0.480
	Alternada	10.364	-0.001	27.820	0.477
<code>peppers.png</code>	Unidirecional	10.322	-0.017	27.856	0.531
	Alternada	10.325	-0.020	27.853	0.531
<code>monalisa.png</code>	Unidirecional	10.411	0.009	27.781	0.419
	Alternada	10.407	0.013	27.784	0.418
<code>watch.png</code>	Unidirecional	10.262	0.016	27.907	0.372
	Alternada	10.263	0.015	27.905	0.371

Nos resultados seguintes, as imagens serão varridas de forma alternada, como este é o padrão da ferramenta.

4.2 Distribuições de Erro

Os valores da tabela 2 indicam que a figura `monalisa.png` foi a que mais sofreu com a mudança de distribuição, enquanto a `watch.png` quase não teve diferença nas métricas. Isso se deve principalmente ao tamanho da imagem digital, que foram descritos na seção 1. Isso fica mais claro observando os resultados, nas figuras 8 e 9.

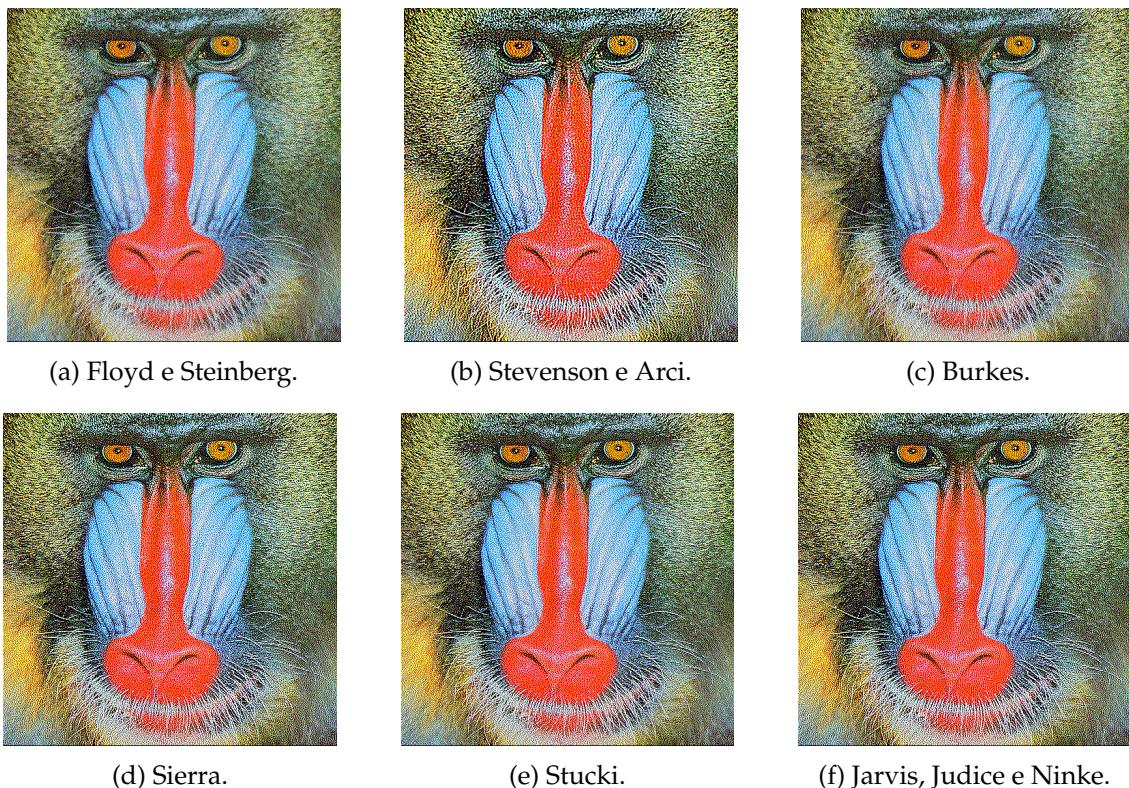


Figura 6: Aplicação das distribuições de erro na figura 5a.

Além disso, a diferença entre `baboon.png` e `peppers.png`, que têm mesma quantidade de

pixels, acontece principalmente por causa das regiões de alta frequência, mais presentes na borda da baboon.png. Essa é a região onde mais se percebe diferenças visuais na figura 6.



Figura 7: Aplicação das distribuições de erro na figura 5d.

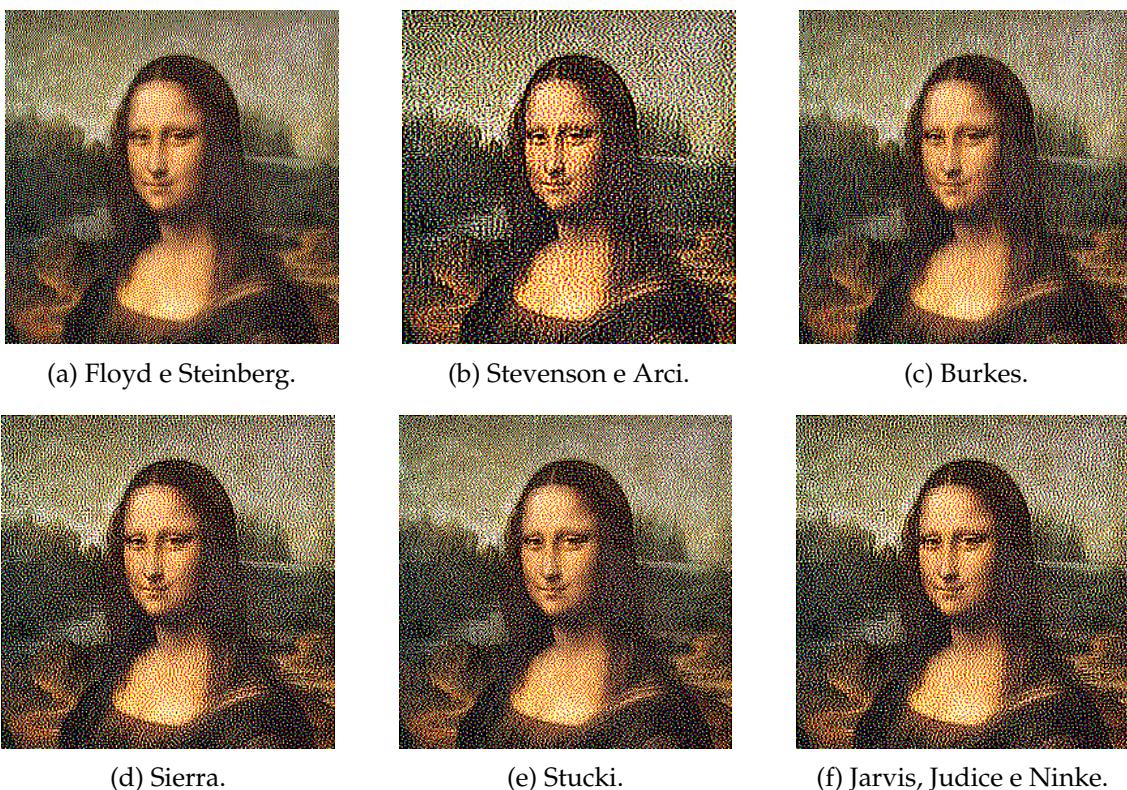


Figura 8: Aplicação das distribuições de erro na figura 5g.

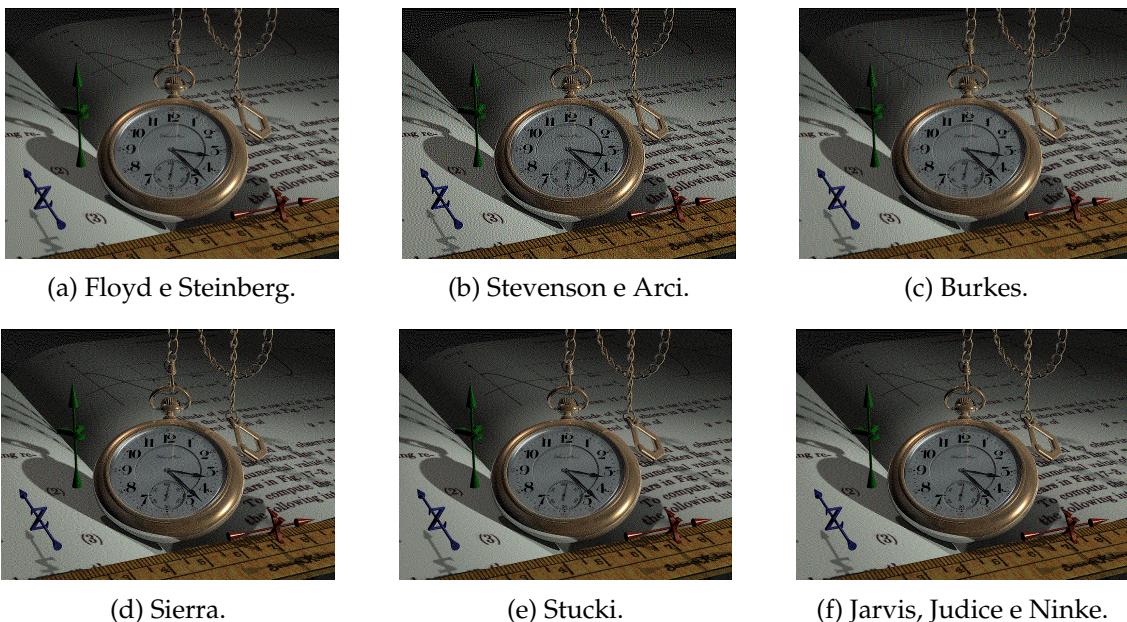


Figura 9: Aplicação das distribuições de erro na figura 5j.

Tabela 2: Comparativo entre as distribuições de erros.

Figura	Distribuição	RMSE	SNR (dB)	PSNR (dB)	Correlação
baboon.png	Floyd e Steinberg	10.364	-0.001	27.820	0.477
	Stevenson e Arci	10.367	-0.003	27.818	0.578
	Burkes	10.365	-0.002	27.819	0.512
	Sierra	10.364	-0.001	27.820	0.547
	Stucki	10.369	-0.005	27.816	0.540
	Jarvis, Judice e Ninke	10.367	-0.003	27.818	0.554
peppers.png	Floyd e Steinberg	10.325	-0.020	27.853	0.531
	Stevenson e Arci	10.328	-0.022	27.851	0.549
	Burkes	10.321	-0.016	27.857	0.537
	Sierra	10.326	-0.021	27.852	0.545
	Stucki	10.330	-0.024	27.849	0.543
	Jarvis, Judice e Ninke	10.326	-0.020	27.853	0.547
monalisa.png	Floyd e Steinberg	10.407	0.013	27.784	0.418
	Stevenson e Arci	10.411	0.009	27.781	0.443
	Burkes	10.412	0.008	27.780	0.422
	Sierra	10.415	0.006	27.777	0.433
	Stucki	10.413	0.008	27.780	0.428
	Jarvis, Judice e Ninke	10.406	0.013	27.785	0.435
watch.png	Floyd e Steinberg	10.263	0.015	27.905	0.371
	Stevenson e Arci	10.263	0.015	27.906	0.392
	Burkes	10.264	0.014	27.905	0.374
	Sierra	10.265	0.013	27.904	0.379
	Stucki	10.264	0.015	27.905	0.379
	Jarvis, Judice e Ninke	10.264	0.014	27.905	0.380

4.3 Varredura em Espiral e em Curvas de Hilbert

Nas figuras 10d e 11d, podemos ver que as curvas de Hilbert causam um contraste maior na imagem. Isso é um indicativo que os erros não estão sendo distribuídos corretamente, provavelmente devido à algum problema de implementação.

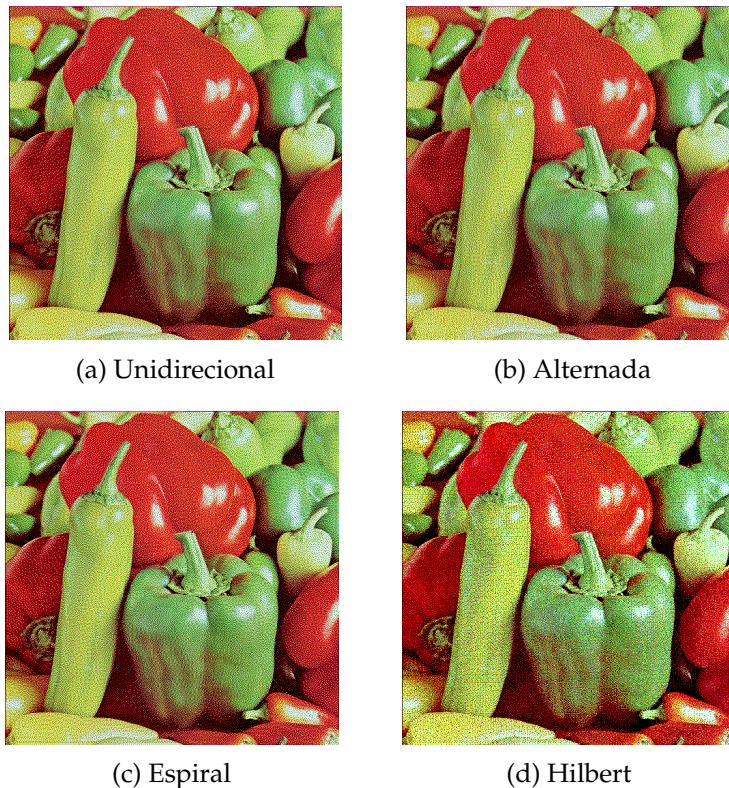


Figura 10: Aplicação das quatro varreduras na peppers.png.

Já as curvas em espiral funcionam como esperado, entretanto aparecem, em ambas as figuras 10c e 11c, artefatos nas diagonais da imagem, onde ocorre a rotação da matriz de distribuição de erros.

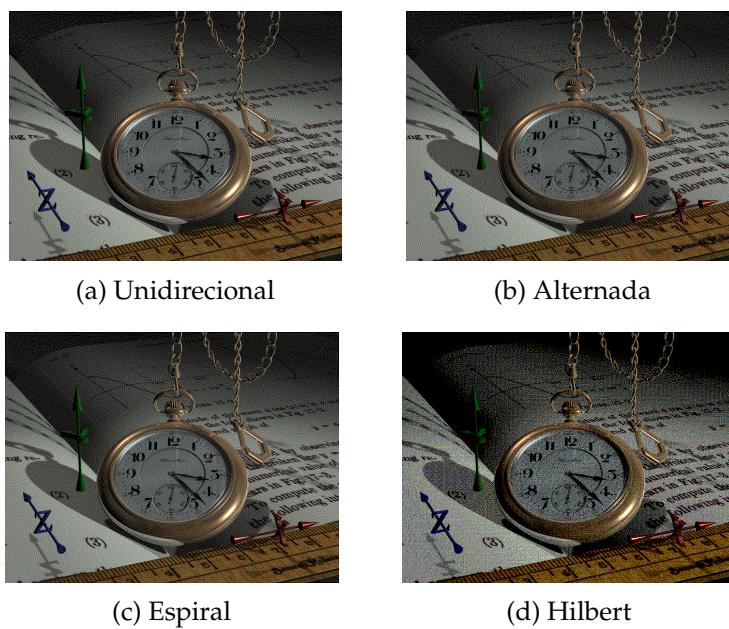


Figura 11: Aplicação das quatro varreduras na watch.png.