SCC0251 — Profa. Leo Sampaio Ferraz Ribeiro

Trabalho 01 : amostragem e quantização

Desenvolva o trabalho sem olhar o de colegas. Plágio não é tolerado.
Se precisar de ajuda, estamos aqui para isso.

1 Gerador de Imagens

1.1 Objetivo

Familiarizar discentes com as ferramentas que terão que usar durante o resto do curso; Reforçar os conceitos de sampling/amostragem de imagens.

1.2 Tarefa

Nesse trabalho vocês devem implementar um **Gerador de Imagens** usando funções matemáticas, um **Amostrador de Imagens** [simulado]. Leia as instruções para cada passo. Use `python` com as bibliotecas `numpy` e `imageio`.

Seu programa vai gerar uma imagem sintética e depois simular o processo de amostragem sobre a mesma:

1. **Gere a Imagem Sintética**, f, de acordo com a função selecionada F e os parâmetros C e Q,
2. **Faça uma Amostragem e Quantização de** f para criar g, com os parâmetros de amostragem e quantização N e B,
3. **Compare** g com a imagem de referência r, usando a raiz do erro quadrático (Root Squared Error, RSE),
4. **Imprima na tela** o RSE computado entre g and r.

1.3 Parâmetros

Os seguintes parâmetros vão ser apresentados ao seu programa na seguinte ordem, através da `stdin`, como é o usual para o run.codes

1. nome do arquivo contendo a imagem de referência r
2. tamanho lateral da imagem sintetizada C (assuma que a imagem é quadrada e que seu tamanho é $C \times C$)
3. a função F que será usada para geração (1, 2, 3, 4 or 5)
4. o parâmetro Q, usado para geração de imagens
5. o tamanho lateral N da imagem amostrada, onde $N \leq C$
6. número de bits por pixel B, com $1 \leq B \leq 8$
7. ângulo de rotação R em graus ($0 \leq R \leq 360$)
8. o seed S para ser usado com as funções aleatórias

Mesmo se um parâmetro não for ser usado pela função de geração selecionada, ele vai estar na entrada para conveniência na leitura.

2 Gerando uma Imagem Sintética

Cinco funções de geração de imagem precisam ser implementadas e os casos de teste vão seleciona-las usando o parâmetro F:

1. $f(i, j) = (ij + 2j)$;
2. $f(i, j) = |\cos(i/Q) + 2\sin(j/Q)|$;
3. $f(i, j) = |3(i/Q) - \sqrt[3]{j/Q}|$;
4. $f(i, j) = \text{rand}(0, 1, \text{S})$:

   A função aleatória (rand) é uniforme entre 0 e 1. Use o seed S para inicializar o pacote `random` no python antes do primeiro uso. Use `random.random()` para gerar os números aleatórios. Preencha a imagem considerando a ordem $f(0, 0), f(0, 1), f(0, 2), ...$
5. $f(i, j) = \text{randomwalk(S)}$, A função random walk deve se comportar como o nome implica:
   a) Comece com uma imagem "zerada" (veja `numpy.zeros`),
   b) Atribua o valor da primeira posição ($i = 0, j = 0$) para 1, ou seja, $f(0, 0) = 1$
   c) Cada passo seguinte consiste em passos aleatórios em ambas direções i e j. Use a função `random.randint()` para obter dois números inteiros aleatórios entre $[-1, 1]$, um para cada direção, chamados de di e dj,

d) Use di e dj para computar as próximas posições $i = [(i + di) \bmod C]$, $j = [(j + dj) \bmod C]$[1] e depois atribua o valor 1 nessa posição: $f(i, j) = 1$,

e) Continue "caminhando" aleatoriamente $1 + C^2$ por passos

Note que, assim como na função anterior, você deve usar o seed S para inicializar o pacote `random` uma vez antes de gerar o primeiro inteiro

Imagens sintetizadas f precisam ser computadas usando valores do tipo `float`. **Depois que f for computada, normalize os valores de forma que o mínimo seja 0 e o máximo seja $2^{16} - 1 = 65535$.**

3 Amostrando e Quantizando a Imagem

Esse passo consiste em simular o processo de "digitalização" de uma imagem (amostra+quantização). Nós vamos imaginar que nossa imagem sintetizada f é uma cena do mundo real e aplicar uma função de amostragem seguida de quantização para obter a imagem g de tamanho $N \times N$, com seus pixels usando B bits no máximo (com B entre 1 e 8).

3.1 Amostragem

Como g terá uma resolução menor ($N \leq C$) que f, um operador de *downsampling* precisa ser implementado. O operador mais simples (e o que vocês devem usar) seleciona o primeiro elemento de cada região de tamanho $(C/N)^2$ e joga o resto fora.

Por exemplo, considere a matriz f com $C = 4$.

$$\begin{bmatrix} 5 & 15 & 36 & 0 \\ 18 & 0 & 0 & 1 \\ 0 & 100 & 154 & 0 \\ 0 & 99 & 159 & 100 \end{bmatrix}$$

Com $N = 2$, essa imagem tem $(C/N)^2 = 4$ regiões:

$$\begin{bmatrix} 5 & 15 & 36 & 0 \\ 18 & 0 & 0 & 1 \\ 0 & 100 & 154 & 0 \\ 0 & 99 & 159 & 100 \end{bmatrix}$$

Esse operador de *downsampling* pega apenas o primeiro pixel de cada região, gerando a imagem:

$$\begin{bmatrix} 5 & 36 \\ 0 & 154 \end{bmatrix}$$

Note que começamos de $g(0, 0) = f(0, 0)$; depois $g(0, 1)$ é computado pulando $\lfloor C/N \rfloor$ pixels na direção j, enquanto $g(1, 0)$ é computado pulando $\lfloor C/N \rfloor$ pixels na direção i, e assim por diante.

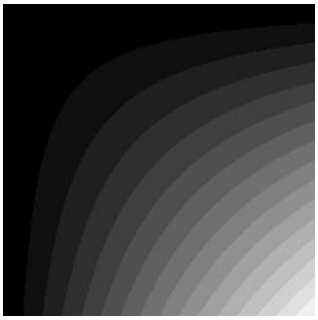
[1] Note que nós usamos o mod para evitar que i e j referenciem uma posição além da borda da imagem

3.2 Quantização

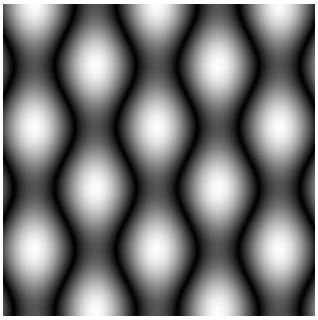
Finalmente, a imagem precisa ser quantizada para usar apenas B bits por pixel. Isso pode ser feito da seguinte forma:

1. Normalização dos valores da imagem para $[0, 255]$ para evitar *overflow* no próximo passo,
2. Conversão de float para inteiros sem sinal de 8 bits (veja no numpy a função `.astype` e o tipo `numpy.uint8`)
3. Use *bitwise shift* para deixar apenas os B bits mais significativos e zerar os outros (Veja os operadores `«` and `»` em `python`). Para ter esse efeito de quantização, um shift para direita seguido de um para esquerda vai zerar os bits menos significativos e manter os mais significativos intactos.

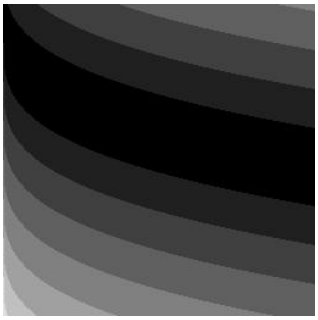
Exemplos de figuras geradas pelas 5 diferentes funções e amostradas e quantizadas usando diferentes parâmetros:



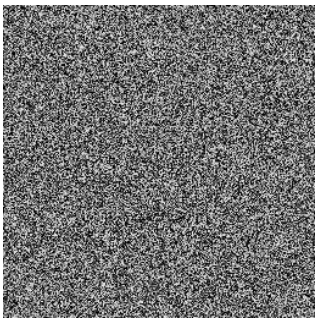
1 ($B = 4$)



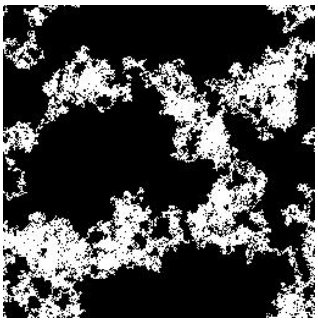
2 ($Q = 32$, $B = 6$)



3 ($Q = 1001$, $B = 3$)



4 ($S = 13$, $B = 3$)



5 ($S = 6666$, $B = 8$)

4 Comparando com a Referência

Seu programa precisa comparar a imagem gerada com a imagem de referência r. Essa comparação deve usar a raiz do erro quadrático (RSE). Imprima esse erro na tela, arredondando para 4 pontos decimais.

$$RSE = \sqrt{\sum_i \sum_j (g(i,j) - R(i,j))^2}$$

Note que esse fórmula não divide o erro pelo número de pixels. É uma modificação da mais comum Raiz da Média do Erro Quadrático (RMSE). **Lembre-se de que neste momento sua imagem é do tipo `numpy.uint8`, para evitar overflow na comparação faça a conversão de volta para `float`.**

A imagem de referência é guardada na forma de uma matriz `numpy`. Você pode carrega-la no código da seguinte forma:

```
import numpy as np

filename = input().rstrip()
R = np.load(filename)
```

Nesse código `input()` lê da `stdin` e `rstrip` remove qualquer carácter branco no fim da string, que é bem útil já que `input()` sempre inclui um `\n` no final de cada linha.

5 Exemplos de Input/output

Input: imagem de referência r `ex1.npy`, $C = 1024$, função $F = 1$, e parâmetros: $Q = 2$, $N = 720$, $B = 6$, $S = 1$

```
ex1.npy
1024
1
2
720
6
1
```

Output: apenas o valor de RSE com 4 casas decimais.

Exemplo 1 (alto RSE, indicando que a imagem gerada é muito diferente da referência):

```
7468.7864
```

Exemplo 2 (baixo RSE, indicando que as duas imagens são similares e o resultado está provavelmente correto):

```
4.1000
```

6 Submissão

Envie seu código fonte para o run.codes (apenas o arquivo `.py`).

1. **Comente seu código**. Além de comentários de explicação, use um header com os nomes, números USP, código do curso, ano/semestre e o título do trabalho. Uma penalidade na nota será aplicada se seu código estiver faltando o header e outros comentários.

2. **Organize seu código em funções**. Use uma função por tipo de imagem que será gerada (1,2,3,4,5).