Luana Ferreira Marques da Silva
Relatório Atividade 3

Introdução

Para realizar esta atividade, a linguagem Java fora escolhida. A estrutura de dados para representar as imagens é a matriz inteira bidimensional: int [][]. Nesta atividade, 3 funções foram apresentadas: espelhamento horizontal, fatiamento das imagens e transformação gama.
Para utilizar a aplicação, basta inserir o caminho da imagem pgm que se deseja transformar e após o processamento as imagens resultantes estarão na pasta do projeto.
O arquivo .rar “testes” apresenta alguns dos resultados obtidos a partir de testes realizados na aplicação.

Funções

loadMatrix: esta função lê o arquivo PGM e gera sua matriz int[coluna][linha].

```java
public static int[][] loadMatrix(String img) throws FileNotFoundException {
        Scanner reader = new Scanner(new FileReader(img));

        String aux = reader.nextLine();
        aux = reader.nextLine();
        System.out.println(aux);

        if (aux.charAt(0) == '#') {
            int coluna = reader.nextInt();
            coluna_ = coluna;

            int linha = reader.nextInt();
            linha_ = linha;

            int[][] Matriz = new int[linha + 1][coluna + 1];

            maxValue = reader.nextInt();

            for (int x = 0; x < linha; x++) {
                for (int y = 0; y < coluna; y++) {
                    Matriz[x][y] = reader.nextInt();
                }

            }
            return Matriz;
        } else {
            return null;
        }

    }
```

```java
public static int[][][] loadMatrix_PPM(String img) throws FileNotFoundException {
        Scanner reader = new Scanner(new FileReader(img));

        String aux = reader.nextLine();
        aux = reader.nextLine();
        System.out.println(aux);

        if (aux.charAt(0) == '#') {
            int coluna = reader.nextInt();
            coluna_ = coluna;

            int linha = reader.nextInt();
            linha_ = linha;

            int[][][] Matriz = new int[linha + 1][coluna + 1][3];

            maxValue = reader.nextInt();

            for (int x = 0; x < linha; x++) {
                for (int y = 0; y < coluna; y++) {
                    Matriz[x][y][0] = reader.nextInt();
                    Matriz[x][y][1] = reader.nextInt();
                    Matriz[x][y][2] = reader.nextInt();

                }

            }
            return Matriz;
        } else {
            return null;
        }

    }
```

salvarMatrix: esta função gera os arquivos .pgm com as matrizes geradas nas transformações.

```java
public static void salvarImagem(int[][] imagem, String namepath) {
    try {
        File file = new File(namepath);
        FileWriter fw = new FileWriter(file);
        fw.write("P2");
        fw.write("\n");
        fw.write(Integer.toString(coluna_));
        fw.write(" ");
        fw.write(Integer.toString(linha_));
        fw.write("\n");
        fw.write(Integer.toString(maxValue));
        fw.write("\n");
        for (int i = 0; i < linha_; i++) {
            for (int j = 0; j < coluna_; j++) {
                fw.write(imagem[i][j] + " ");
            }
            fw.write("\n");
        }
        fw.flush();
    } catch (IOException e) {
    }
}
```

```java
public static void salvarImagem_PPM(int[][][] imagem, String namepath) {
        try {
            File file = new File(namepath);
            FileWriter fw = new FileWriter(file);
            fw.write("P3");
            fw.write("\n");
            fw.write(Integer.toString(coluna_));
            fw.write(" ");
            fw.write(Integer.toString(linha_));
            fw.write("\n");
            fw.write(Integer.toString(maxValue));
            fw.write("\n");
            for (int i = 0; i < linha_; i++) {
                for (int j = 0; j < coluna_; j++) {
                    fw.write(imagem[i][j][0] + " ");
                    fw.write(imagem[i][j][1] + " ");
                    fw.write(imagem[i][j][2] + " ");

                }
                fw.write("\n");
            }
            fw.flush();
        } catch (IOException e) {
        }
    }
```

binFat: realiza o fatiamento das imagens. O parâmetro op define qual fatiamento será realizado.
Os limites para opção de fatiamento 1 para teste: 125 e 220.
Os limites para opção de fatiamento 2 para teste: 150 e 250.
Na aplicação final, os limites são parâmetros dados pelo usuário.

```java
public static int[][] binFat(int[][] matrix, int op) {
        int coluna, linha;

        linha = matrix.length - 1;
        coluna = matrix[0].length - 1;

        int[][] img = new int[linha][coluna];

        for (int x = 0; x < linha; x++) {
            for (int y = 0; y < coluna; y++) {
                img[x][y] = matrix[x][y];
            }
        }
        if (op == 1) {
            for (int x = 0; x < linha; x++) {
                for (int y = 0; y < coluna; y++) {

                    if ((img[x][y] <= 125) || (img[x][y] > 220)) {
                        img[x][y] = 10;
                    } else {
                        img[x][y] = 250;
                    }

                }
            }
        } else {
            for (int x = 0; x < linha; x++) {
                for (int y = 0; y < coluna; y++) {
                    if (!((img[x][y] <= 150) || (img[x][y] > 250))) {
                        img[x][y] = 200;
                    }
                }
            }
        }

        return img;
    }
```

```java
public static int[][][] binFat_PPM(int[][][] matrix, int op, int sup, int inf) {
        int coluna, linha;

        linha = matrix.length - 1;
        coluna = matrix[0].length - 1;

        int[][][] img = new int[linha][coluna][3];

        for (int x = 0; x < linha; x++) {
            for (int y = 0; y < coluna; y++) {
                img[x][y][0] = matrix[x][y][0];
                img[x][y][1] = matrix[x][y][1];
                img[x][y][2] = matrix[x][y][2];

            }
        }
        if (op == 1) {
            for (int x = 0; x < linha; x++) {
                for (int y = 0; y < coluna; y++) {

                    if ((img[x][y][0] < inf) || (img[x][y][0] > sup)) {
                        img[x][y][0] = 10;
                    } else {
                        img[x][y][0] = 250;
                    }
                    if ((img[x][y][1] < inf) || (img[x][y][1] > sup)) {
                        img[x][y][1] = 10;
                    } else {
                        img[x][y][1] = 250;
                    }
                    if ((img[x][y][2] < inf) || (img[x][y][2] > sup)) {
                        img[x][y][2] = 10;
                    } else {
                        img[x][y][2] = 250;
                    }

                }
            }
        } else {
            for (int x = 0; x < linha; x++) {
                for (int y = 0; y < coluna; y++) {
                    if (!((img[x][y][0] <= inf) || (img[x][y][0] > sup))) {
                        img[x][y][0] = 200;
                    }
                    if (!((img[x][y][1] <= inf) || (img[x][y][1] > sup))) {
                        img[x][y][1] = 200;
                    }
                    if (!((img[x][y][2] <= inf) || (img[x][y][2] > sup))) {
                        img[x][y][2] = 200;
                    }
                }
            }
        }

        return img;
    }
```

Transformação em Potência/Gama: esta função realiza a transformação gama nas imagens, onde const_ é a constante para clarear a imagem, imagem[x][y] é o tom de cinza e gama é o valor escolhido.

```java
public static int[][] transfPotencia(int[][] imagem, double gama, int const_) {
        int coluna, linha;

        linha = imagem.length - 1;
        coluna = imagem[0].length - 1;

        int[][] img = new int[linha][coluna];

        for (int x = 0; x < linha; x++) {
            for (int y = 0; y < coluna; y++) {
                img[x][y] = imagem[x][y];
            }
        }

        for (int x = 0; x < linha; x++) {
            for (int y = 0; y < coluna; y++) {
                img[x][y] = const_ * (int) Math.pow((imagem[x][y]), gama);
                if (img[x][y] > 255) {
                    img[x][y] = 255;
                }
            }
        }
        return img;
    }
```

```java
public static int[][][] transfPotencia_PPM(int[][][] imagem, double gama, int const_) {
        int coluna, linha;

        linha = imagem.length - 1;
        coluna = imagem[0].length - 1;

        int[][][] img = new int[linha][coluna][3];

        for (int x = 0; x < linha; x++) {
            for (int y = 0; y < coluna; y++) {
                img[x][y][0] = imagem[x][y][0];
                img[x][y][1] = imagem[x][y][1];
                img[x][y][2] = imagem[x][y][2];

            }
        }

        for (int x = 0; x < linha; x++) {
            for (int y = 0; y < coluna; y++) {
                double convertido = img[x][y][0] / 255.0;
                double convertido1 = img[x][y][1] / 255.0;
                double convertido2 = img[x][y][2] / 255.0;

                int funcao = (int) Math.round((const_ * (Math.pow(convertido, gama))));
                int funcao1 = (int) Math.round((const_ * (Math.pow(convertido1, gama))));
                int funcao2 = (int) Math.round((const_ * (Math.pow(convertido2, gama))));

                //img[x][y] = const_ * (int) Math.pow((imagem[x][y]), gama);
                img[x][y][0] = (int) Math.round(funcao * 255.0);
                img[x][y][1] = (int) Math.round(funcao1 * 255.0);
                img[x][y][2] = (int) Math.round(funcao2 * 255.0);

            }
        }
        return img;
    }
```

Função de Espelhamento: esta função realiza o flip horizontal das imagens.

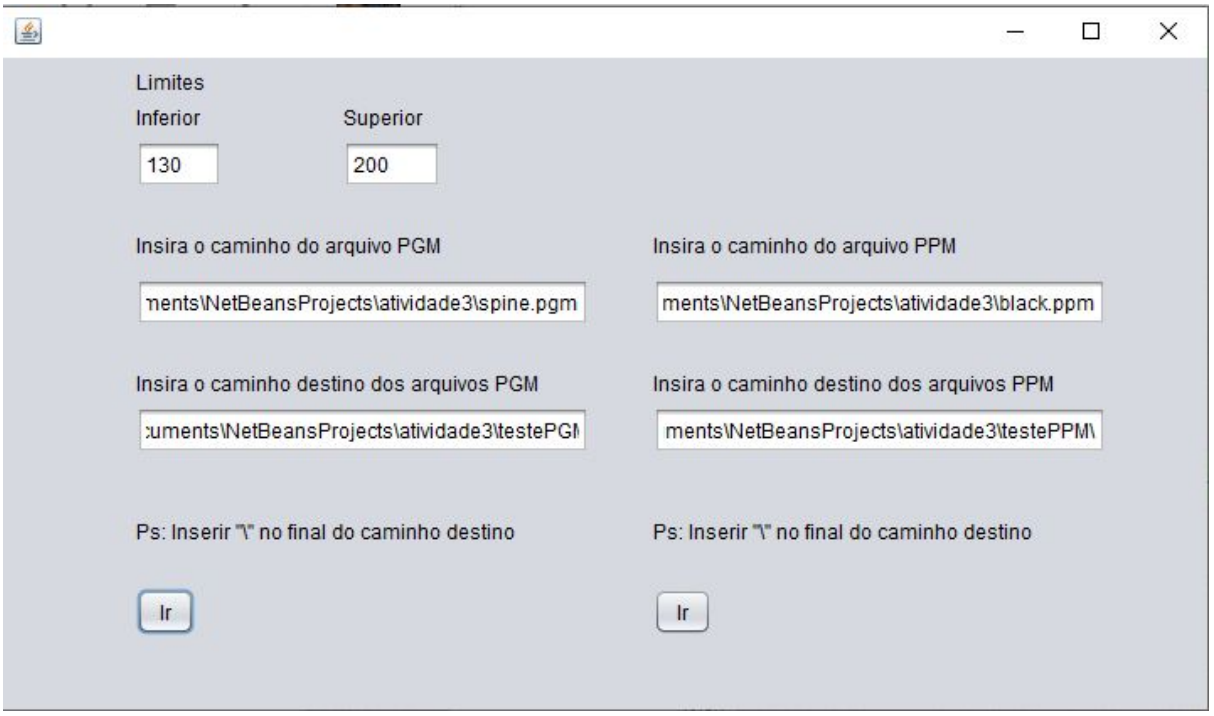
```java
public static int[][] espelhamento(int[][] img) {
        int coluna, linha;
        int auxLinha, auxColuna;
        int z;

        linha = img.length;
        coluna = img[0].length;

        auxLinha = 0;
        auxColuna = coluna - 1;

        int aux;

        int[][] espelhada = new int[linha][coluna];

        for (int x = 0; x < linha; x++) {
            for (int y = 0; y < coluna; y++) {
                espelhada[x][y] = img[x][y];
            }
        }

        for (int i = 0; i < linha; i++) {
            for (int j = 0; j < coluna; j++) {
                espelhada[i][j] = img[i][coluna - j - 1];
            }
        }

        return espelhada;
    }
```

```java
public static int[][][] espelhamento_PPM(int[][][] img) {
        int coluna, linha;
        int auxLinha, auxColuna;
        int z;

        linha = img.length;
        coluna = img[0].length;

        auxLinha = 0;
        auxColuna = coluna - 1;

        int aux;

        int[][][] espelhada = new int[linha][coluna][3];

        for (int x = 0; x < linha; x++) {
            for (int y = 0; y < coluna; y++) {
                espelhada[x][y][0] = img[x][y][0];
                espelhada[x][y][1] = img[x][y][1];
                espelhada[x][y][2] = img[x][y][2];

            }
        }

        for (int i = 0; i < linha; i++) {
            for (int j = 0; j < coluna; j++) {
                espelhada[i][j][0] = img[i][coluna - j - 1][0];
                espelhada[i][j][1] = img[i][coluna - j - 1][1];
                espelhada[i][j][2] = img[i][coluna - j - 1][2];

            }
        }

        return espelhada;
    }
```

Aplicação

Para utilizar a aplicação, basta inserir o caminho da imagem pgm a ser modificada, o caminho no qual as imagens serão salvas e os limites superior e inferior para os fatiamentos.

Exemplo:



Exemplos de caminhos fonte válidos:
C:\Users\Luana\Documents\NetBeansProjects\spine.pgm
C:\Users\Luana\Documents\NetBeansProjects\imagem.pgm
C:\Users\Luana\Documents\NetBeansProjects\atividade3\imagem.pgm
C:\Users\Luana\Documents\NetBeansProjects\black.pgm

Exemplos de caminhos destino válidos:
C:\Users\Luana\Documents\NetBeansProjects\
C:\Users\Luana\Desktop\

Os arquivos resultantes são:

- Para PGM

- Para PPM