

Image resizing using
Order Hold method
(keeping aspect ratio)

Student: *Stoean Andrei Cosmin*

Profesor: *Hossu Andrei*

Grupa: *332AB*

Data: *Ianuarie 2023*

Introducere

Metoda *Zero order hold* este efectuată prin repetarea valorilor pixelilor anteriori, creând astfel un efect blocant.

Exemplu: dacă avem o imagine de mărime $(n * n)$, putem să o mărim folosind metoda de ordin zero cu mărimea $(2n) * (2n)$

$$\begin{array}{c}
 \left[\begin{array}{cc} f(x, y) & f(x, y + 1) \\ f(x + 1, y) & f(x + 1, y + 1) \end{array} \right]_{2 \times 2} \\
 \downarrow \\
 \left[\begin{array}{cccc} f(x, y) & f(x, y) & f(x, y + 1) & f(x, y + 1) \\ f(x, y) & f(x, y) & f(x, y + 1) & f(x, y + 1) \\ f(x + 1, y) & f(x + 1, y) & f(x + 1, y + 1) & f(x + 1, y + 1) \\ f(x + 1, y) & f(x + 1, y) & f(x + 1, y + 1) & f(x + 1, y + 1) \end{array} \right]_{4 \times 4}
 \end{array}$$

Descrierea aplicatiei cerute

Proiectul are ca scop citirea unei imagini din calculator, prelucrarea acesteia pixel cu pixel astfel încât să se obțină imaginea marita sau micșorata și scrierea noii imagini în calculator. Dorim de asemenea să aflăm și timpii de execuție ai fiecărei etape. Aceasta metoda este cunoscut ca zoom de două ori. Deoarece se poate mări doar de două ori.

Ne folosim de extrapolatorul de ordinul 0, acesta se refera la faptul ca mentinem valoarea precedenta pana la urmatoarea valoare.

Punem cate un pixel intre fiecare doi pixeli cu valoarea celui precedent. Dupa care copiem fiecare rand inca o data, unul sub altul, ca sa pastram raportul dintre inaltime si latime.

Pentru micșorare trebuie sa stergem tot ce am adaugat imaginii marite.

Unul dintre avantajele acestei tehnici de zoom, este că nu creează o imagine neclara, comparativ cu alte metode de interpolare.. Are și un dezavantaj, acela că nu poate funcționa decât pe puterea lui 2.

Descrierea structurala, arhitecturala si functionala

Din punct de vedere structural, proiectul este alcatuit din 6 clase:

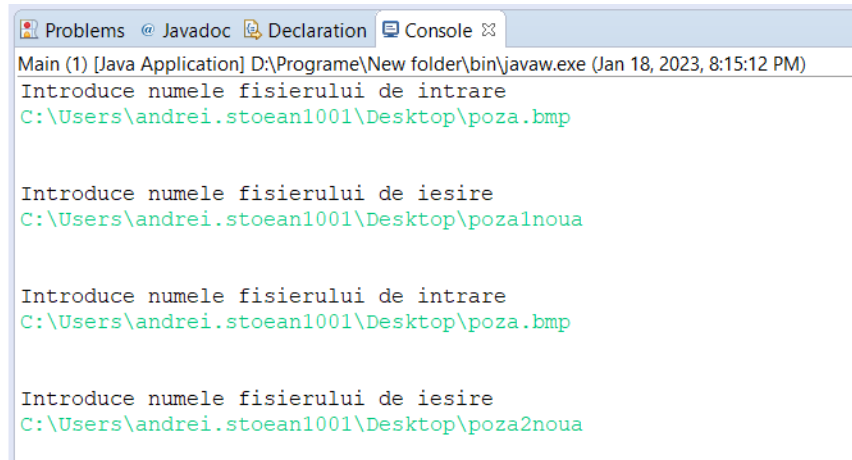
- ImageProcessor
- IOProcessor
- Main
- Resize
- ResizeTask
- Scalable

Ca mediu de lucru am folosit platforma Eclipse Mars 2, cu Java 8 pe Windows-x64.

Etapele de executie ale aplicatiei sunt:

- citire informatii de identificare fisier sursa si citire informatii de identificare fisier destinatie
- citire fisier sursa
- procesare imagine
- scriere fisier destinatie
- inregistrare timp de executie fiecare etapa
- afisare rezultate timp de procesare fiecare etapa

Pentru executie, am ales citirea de la tastatura, in consola, a path-ului catre imaginea respective pe care o dorim sa o modificam:



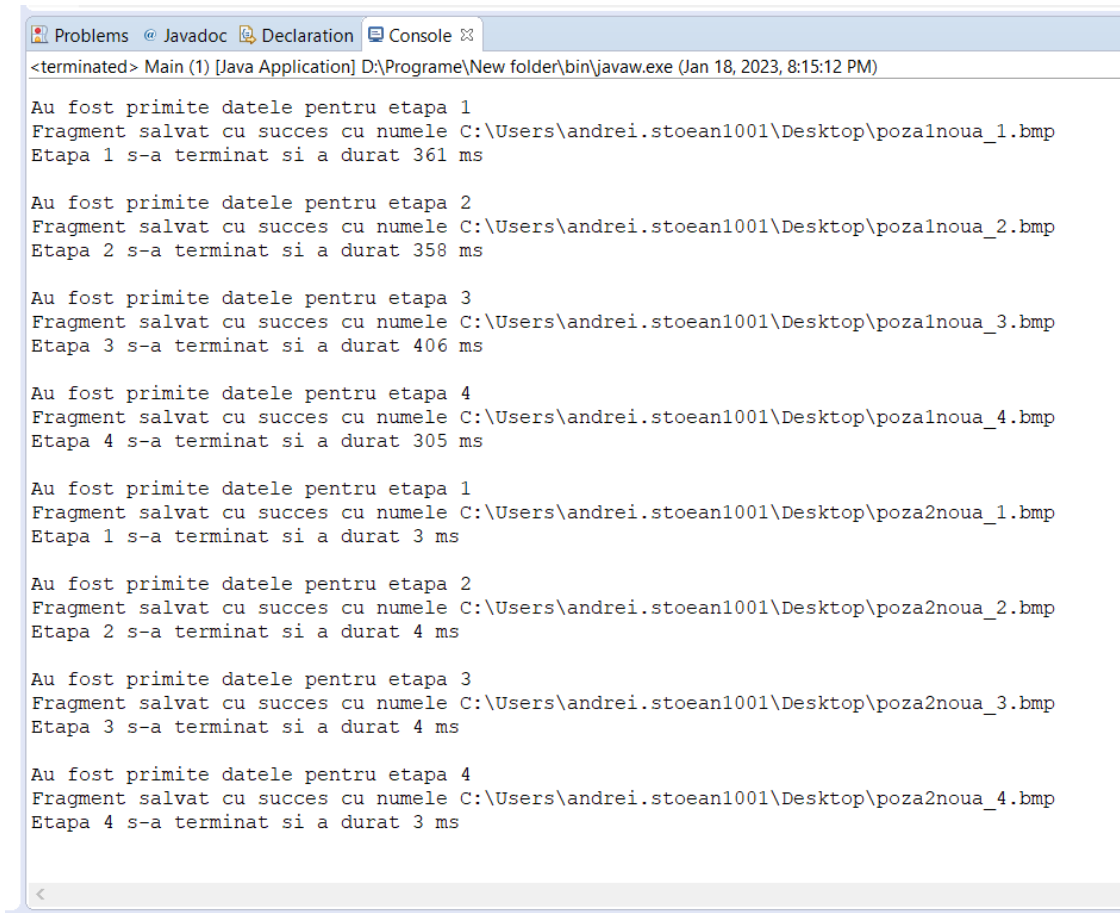
```
Problems @ Javadoc Declaration Console
Main (1) [Java Application] D:\Programe\New folder\bin\javaw.exe (Jan 18, 2023, 8:15:12 PM)
Introduce numele fisierului de intrare
C:\Users\andrei.stoean1001\Desktop\poza.bmp

Introduce numele fisierului de iesire
C:\Users\andrei.stoean1001\Desktop\poza1noua

Introduce numele fisierului de intrare
C:\Users\andrei.stoean1001\Desktop\poza.bmp

Introduce numele fisierului de iesire
C:\Users\andrei.stoean1001\Desktop\poza2noua
```

In urma executiei programului, in consola, acesta va afisa urmatoarele:



```
Problems @ Javadoc Declaration Console
<terminated> Main (1) [Java Application] D:\Programe\New folder\bin\javaw.exe (Jan 18, 2023, 8:15:12 PM)

Au fost primite datele pentru etapa 1
Fragment salvat cu succes cu numele C:\Users\andrei.stoean1001\Desktop\poza1noua_1.bmp
Etapa 1 s-a terminat si a durat 361 ms

Au fost primite datele pentru etapa 2
Fragment salvat cu succes cu numele C:\Users\andrei.stoean1001\Desktop\poza1noua_2.bmp
Etapa 2 s-a terminat si a durat 358 ms

Au fost primite datele pentru etapa 3
Fragment salvat cu succes cu numele C:\Users\andrei.stoean1001\Desktop\poza1noua_3.bmp
Etapa 3 s-a terminat si a durat 406 ms

Au fost primite datele pentru etapa 4
Fragment salvat cu succes cu numele C:\Users\andrei.stoean1001\Desktop\poza1noua_4.bmp
Etapa 4 s-a terminat si a durat 305 ms

Au fost primite datele pentru etapa 1
Fragment salvat cu succes cu numele C:\Users\andrei.stoean1001\Desktop\poza2noua_1.bmp
Etapa 1 s-a terminat si a durat 3 ms

Au fost primite datele pentru etapa 2
Fragment salvat cu succes cu numele C:\Users\andrei.stoean1001\Desktop\poza2noua_2.bmp
Etapa 2 s-a terminat si a durat 4 ms

Au fost primite datele pentru etapa 3
Fragment salvat cu succes cu numele C:\Users\andrei.stoean1001\Desktop\poza2noua_3.bmp
Etapa 3 s-a terminat si a durat 4 ms

Au fost primite datele pentru etapa 4
Fragment salvat cu succes cu numele C:\Users\andrei.stoean1001\Desktop\poza2noua_4.bmp
Etapa 4 s-a terminat si a durat 3 ms
```

Implementarea aplicației a fost realizată urmărind cerințele de proiectare și de îndeplinire a scopului aplicației.

Operații de intrare de la tastatura - `java.util.Scanner`; pentru extragerea parametrilor de execuție, implementată într-un bloc static de inițializare pentru a se executa înainte de începerea procesului.

```
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Scanner;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Resize extends ImageProcessor implements Scalable {
    private final Scanner scanner = new Scanner(System.in);
    public Resize() {
        super();
        readInputFileName();
        readOutputFileName();
        try {
            inputImage = ImageIO.read(new File(inputFileName));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void callThreads(int newX, int newY, double scale) {
        // Se creeaza bufferul de scriere pentru noua imagine
        outputImage = new BufferedImage(newX, newY,
        BufferedImage.TYPE_INT_RGB);

        // Folosim executor service cu un thread alocat pentru o implementare
        de pipeline
        ExecutorService executorService = Executors.newFixedThreadPool(1);
        // Element de sincronizare care ne va ajuta sa asteptam rezultatul
        final al prelucrarii
        CountDownLatch countDownLatch = new CountDownLatch(4);

        // Se trimite primul task de prelucrare cu primul sfert de imagine
        executorService.submit(
            new ResizeTask(
                newX,
                newY,
                Scalable.firstQuarter,
                executorService,
                countDownLatch,
                this,
                scale
            )
        );
    }
}
```

```
        )
    );

    // Se asteapta terminarea celor 4 etape din pipeline
    try {
        countdownLatch.await();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    // Odata ce toate cele 4 etape s-au incheiat, putem inchide executor
    service pentru ca programul sa se poata termina
    executorService.shutdownNow();
}

@Override
public void invoke(int size, Preserve property) {
    if (property == Preserve.WIDTH) {
        double aspectRatio = (double)inputImage.getWidth() /
inputImage.getHeight();
        int newY = (int) (size / aspectRatio);
        callThreads(size, newY, 1);
    }
    else if (property == Preserve.HEIGHT) {
        double aspectRatio = (double)inputImage.getWidth() /
inputImage.getHeight();
        int newX = (int) (size / aspectRatio);
        callThreads(newX, size, 1);
    }
}

@Override
public void invoke(int newX, int newY, double scale) {
    newX = (int) (newX * scale);
    newY = (int) (newY * scale);
    callThreads(newX, newY, scale);
}

@Override
public void invoke(int size, Preserve property, double scale) {
    if (property == Preserve.WIDTH) {
        double aspectRatio = (double)inputImage.getWidth() /
inputImage.getHeight();
        int newY = (int) (size * scale / aspectRatio);
        callThreads(size, newY, scale);
    }
    else if (property == Preserve.HEIGHT) {
        double aspectRatio = (double)inputImage.getWidth() /
inputImage.getHeight();
        int newX = (int) (size * scale / aspectRatio);
        callThreads(newX, size, scale);
    }
}

/**
 * Metoda ce va citi numele fisierului de intrare de la stdin
 */
```

```
@Override
protected void readInputFileName() {
    System.out.println("Introduce numele fisierului de intrare");
    inputFileName = scanner.nextLine();
    System.out.println("\n");
}

/**
 * Metoda ce va citi numele fisierului de iesire de la stdin
 */
@Override
protected void readOutputFileName() {
    System.out.println("Introduce numele fisierului de iesire");
    outputFileName = scanner.nextLine();
    System.out.println("\n");
}
}
```

Rezolvarea problemei de sincronizare Producer-Consumer în transmiterea unei imagini împărțită în sferturi folosind Thread-uri si transmiterea imaginii procesate prin pipes.

```
public void run() {
    BufferedImage inputImage = imageProcessor.inputImage;
    BufferedImage outputImage = imageProcessor.outputImage;

    System.out.printf("Au fost primite datele pentru etapa %s\n",
currentState);

    // Aflam portiunea de date din imagine care trebuie prelucrata in
task-ul din pipeline
    int start = length(newY, currentState - 1);
    int end = length(newY, currentState);

    // Incepem contorizarea timpului
    Long startTime = System.currentTimeMillis();
    for (int y = start; y < end; y++) {
        for (int x = 0; x < newX; x++) {
            int originalY = (int) (y * ((double)inputImage.getHeight() /
scale / newY));
            int originalX = (int) (x * ((double)inputImage.getWidth() /
scale / newX));
            try {
                int pixel = inputImage.getRGB(originalX, originalY);
                outputImage.setRGB(x, y, pixel);
            }
            // Chiar daca este aruncata atunci doar cand dam zoom out
            // Este necesara pentru a observa efectul de zoom out
            catch (ArrayIndexOutOfBoundsException e) {
                continue;
            }
        }
    }
}
```

```

        // Se scrie imaginea din etapa curenta intr-un fisier
        imageProcessor.save(String.format("%s_%s.bmp",
imageProcessor.getOutputFileName(), currentState));
        // Oprim contorizarea timpului
        Long endTime = System.currentTimeMillis();
        System.out.printf("Etapa %s s-a terminat si a durat %s ms\n\n",
currentState, endTime - startTime);

        // Asteptam o secunda pentru a trece la urmatoarea etapa
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        // Semnalăm elementul de sincronizare ca trecem la urmatoarea etapa
        countdownLatch.countDown();

        // Pana cand nu ma voi afla in stagiul 4 al pipeline-ului, voi lansa
un task nou pentru procesarea urmatorului sfert de imagine
        if (currentState != Scalable.fourthQuarter) {
            executorService.submit(
                new ResizeTask(
                    newX,
                    newY,
                    currentState + 1,
                    executorService,
                    countdownLatch,
                    imageProcessor,
                    scale
                )
            );
        }
    }
}

private int length(int newCoordinate, int epoch) {
    switch (epoch) {
        case Scalable.start: return 0;
        case Scalable.firstQuarter: return newCoordinate / 4;
        case Scalable.secondQuarter: return newCoordinate / 2;
        case Scalable.thirdQuarter: return 3 * newCoordinate / 4;
        case Scalable.fourthQuarter: return newCoordinate;
        default: throw new IllegalArgumentException("Wrong end");
    }
}

```


Construirea și folosirea obiectelor(Main.java)

```
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException {
        Scalable resize1 = new Resize();
        Scalable resize2 = new Resize();
        resize1.invoke(5000, Scalable.Preserve.WIDTH);
        resize2.invoke(300, Scalable.Preserve.WIDTH);
    }
}
```

Salvarea imaginii cu nume specificat in clasa ImageProcessing:

```
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public abstract class ImageProcessor extends IOProcessor {
    protected BufferedImage inputImage;
    protected BufferedImage outputImage;
    public ImageProcessor() { super(); }

    protected abstract void readInputFileName();
    protected abstract void readOutputFileName();

    /**
     * Salveaza imaginea cu numele specificat ca parametru
     * @param outputFileName
     */
    public void save(String outputFileName) {
        try {
            ImageIO.write(outputImage, Scalable.fileExtension, new
File(outputFileName));
            System.out.printf("Fragment salvat cu succes cu numele %s\n",
outputFileName);
        } catch (IOException e) {
            System.err.println("Nu s-a putut salva acest fragment");
        }
    }
}
```

Bibliografie

- Suportul de curs AWJ 2023
- Laboratorul de AWJ 2023
- <https://www.javatpoint.com/>
- <https://stackoverflow.com/>
- <https://docs.oracle.com/>