Дисципліна "Комп'ютерний синтез та обробка зображень"

тема "Оптимізація палітри і псевдо тонування зображення"

**Мета роботи:** вивчення методів оптимізації палітри зображення і методів псевдо тонування.

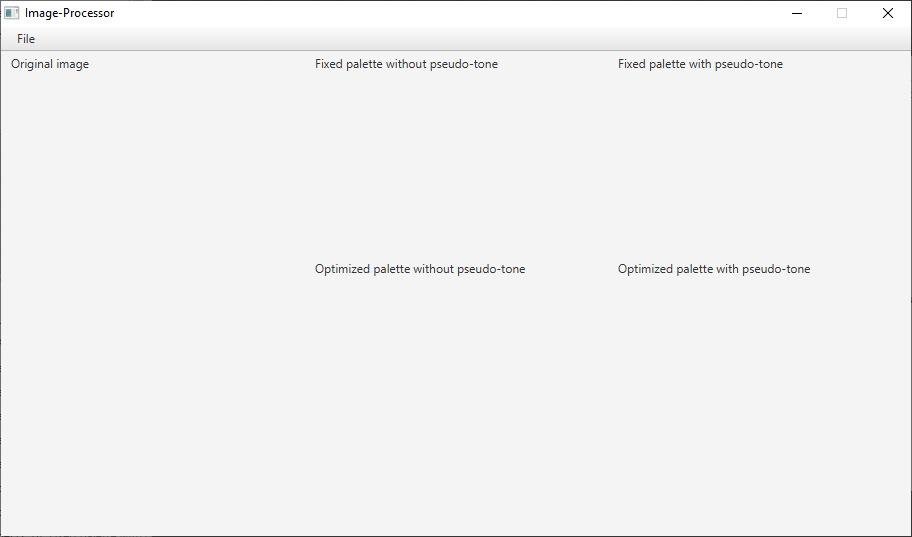
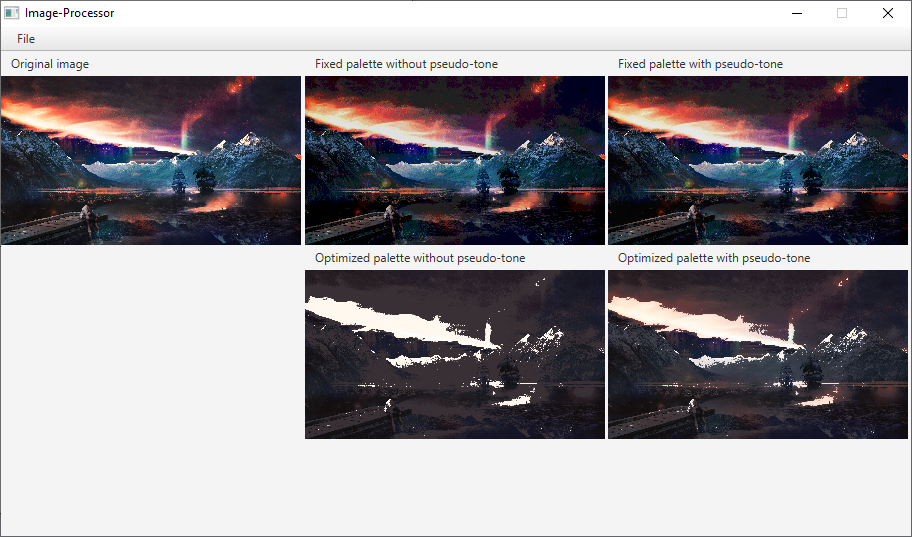
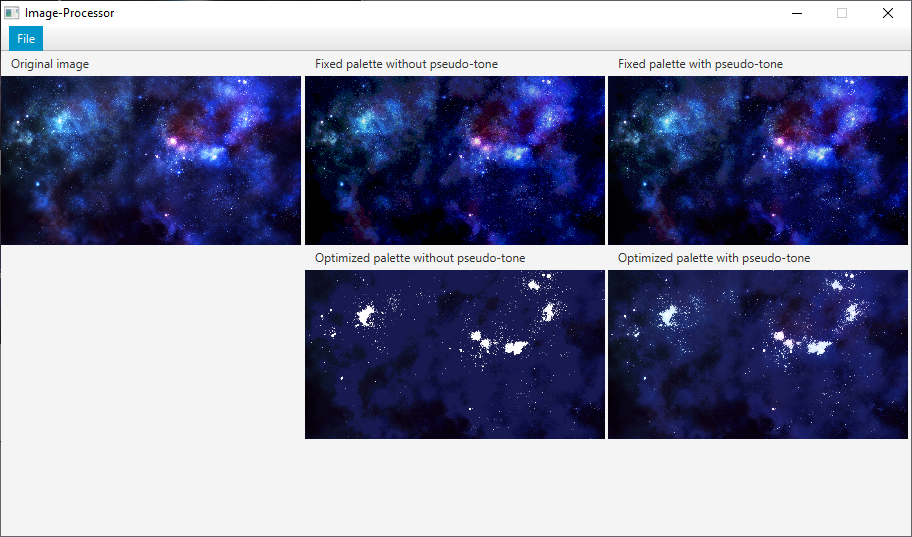
**Постановка задачі лабораторної роботи**

Реалізувати програмно перетворення початкового повно кольорового зображення (24 біта/піксель) в зображення з фіксованою і оптимізованою палітрою і псевдо тонуванням. При отриманні фіксованої палітри використовувати лінійне квантування, при отриманні оптимізованої палітри – метод популярності, при псевдо тонуванні – фільтр Флойда –Штейнбеpга (прохід по всіх рядках зліва направо).

Результати слід представити так, щоб була можливість візуально оцінити якість оптимізації палітри і псевдо тонування і порівняти реалізовані методи. У вікні програми одночасно повинні бути представлені початкове зображення і результати його перетворення: зображення з фіксованою палітрою з псевдо тонуванням і без нього, зображення з оптимізованою палітрою з псевдо тонуванням і без нього. Для формальної оцінки оптимізації палітри слід обчислити помилку в отриманому зображенні в порівнянні з початковим зображенням, використовуючи формулу квадратичного відхилення.

Кількість кольорів в палітрі після перетворення повинна дорівнювати 64.

**Інтерфейс розробленої програми**

**Вихідний код розробленої програми**

Main.java

package lab5;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

public class Main extends Application {

@Override

public void start(Stage primaryStage) throws Exception{

Parent root = FXMLLoader.load(getClass().getResource("Main.fxml"));

primaryStage.setTitle("CG lab5");

primaryStage.setScene(new Scene(root, 900, 500));

primaryStage.setResizable(false);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

Main.fxml

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>

<?import javafx.scene.control.Label?>

<?import javafx.scene.control.Menu?>

<?import javafx.scene.control.MenuBar?>

<?import javafx.scene.control.MenuItem?>

<?import javafx.scene.image.ImageView?>

<?import javafx.scene.layout.ColumnConstraints?>

<?import javafx.scene.layout.GridPane?>

<?import javafx.scene.layout.RowConstraints?>

<?import javafx.scene.layout.VBox?>

<VBox xmlns="http://javafx.com/javafx/8.0.121" xmlns:fx="http://javafx.com/fxml/1" fx:controller="lab5.Controller">

<MenuBar VBox.vgrow="NEVER">

<Menu mnemonicParsing="false" text="File">

<MenuItem mnemonicParsing="false" onAction="#OpenImage" text="Open…"/>

<MenuItem mnemonicParsing="false" onAction="#SaveImage" text="Save..."/>

<MenuItem mnemonicParsing="false" onAction="#QuitProgram" text="Quit"/>

</Menu>

</MenuBar>

<GridPane>

<columnConstraints>

<ColumnConstraints hgrow="SOMETIMES" minWidth="300.0" prefWidth="300.0"/>

<ColumnConstraints hgrow="SOMETIMES" minWidth="300.0" prefWidth="300.0"/>

<ColumnConstraints hgrow="SOMETIMES" minWidth="300.0" prefWidth="300.0"/>

</columnConstraints>

<rowConstraints>

<RowConstraints maxHeight="25.0" minHeight="25.0" prefHeight="25.0" vgrow="SOMETIMES"/>

<RowConstraints vgrow="SOMETIMES"/>

<RowConstraints maxHeight="25.0" minHeight="25.0" prefHeight="25.0" vgrow="SOMETIMES"/>

<RowConstraints vgrow="SOMETIMES"/>

</rowConstraints>

<ImageView fx:id="BaseImage" fitHeight="180.0" fitWidth="300.0" pickOnBounds="true" preserveRatio="true"

GridPane.columnIndex="0" GridPane.hgrow="SOMETIMES" GridPane.rowIndex="1"

GridPane.vgrow="SOMETIMES"/>

<ImageView fx:id="FPTNImage" fitHeight="180.0" fitWidth="300.0" pickOnBounds="true" preserveRatio="true"

GridPane.columnIndex="1" GridPane.rowIndex="1"/>

<ImageView fx:id="OPTNImage" fitHeight="180.0" fitWidth="300.0" pickOnBounds="true" preserveRatio="true"

GridPane.columnIndex="1" GridPane.rowIndex="3"/>

<ImageView fx:id="FPTYImage" fitHeight="180.0" fitWidth="300.0" pickOnBounds="true" preserveRatio="true"

GridPane.columnIndex="2" GridPane.rowIndex="1"/>

<ImageView fx:id="OPTYImage" blendMode="OVERLAY" fitHeight="180.0" fitWidth="300.0"

nodeOrientation="INHERIT" pickOnBounds="true" preserveRatio="true" GridPane.columnIndex="2"

GridPane.rowIndex="3"/>

<Label prefHeight="17.0" prefWidth="305.0" text="Оригінальне зображення" textAlignment="CENTER">

<opaqueInsets>

<Insets/>

</opaqueInsets>

<GridPane.margin>

<Insets bottom="5.0" left="10.0" right="10.0" top="5.0"/>

</GridPane.margin>

</Label>

<Label text="Фіксована палітра без псевдотонування" GridPane.columnIndex="1">

<GridPane.margin>

<Insets bottom="5.0" left="10.0" right="10.0" top="5.0"/>

</GridPane.margin>

</Label>

<Label text="Фіксована палітра з псевдотонуванням" GridPane.columnIndex="2">

<GridPane.margin>

<Insets bottom="5.0" left="10.0" right="10.0" top="5.0"/>

</GridPane.margin>

</Label>

<Label text="Оптимізована палітра без псевдотонування" GridPane.columnIndex="1" GridPane.rowIndex="2">

<GridPane.margin>

<Insets bottom="5.0" left="10.0" right="10.0" top="5.0"/>

</GridPane.margin>

</Label>

<Label text="Оптимізована палітра з псевдотонуванням" GridPane.columnIndex="2" GridPane.rowIndex="2">

<GridPane.margin>

<Insets bottom="5.0" left="10.0" right="10.0" top="5.0"/>

</GridPane.margin>

</Label>

</GridPane>

</VBox>

Controller.java

package lab5;

import javafx.application.Platform;

import javafx.embed.swing.SwingFXUtils;

import javafx.event.ActionEvent;

import javafx.scene.image.\*;

import javafx.scene.image.Image;

import javafx.stage.FileChooser;

import javafx.stage.FileChooser.ExtensionFilter;

import javax.imageio.ImageIO;

import java.io.File;

import java.io.IOException;

public class Controller {

public ImageView BaseImage;

public ImageView FPTNImage;

public ImageView FPTYImage;

public ImageView OPTNImage;

public ImageView OPTYImage;

private Image OpenImageFile() {

FileChooser fileChooser = new FileChooser();

fileChooser.setTitle("Open Image File");

fileChooser.getExtensionFilters().addAll(

new ExtensionFilter("Image Files", "\*.tif", "\*.tiff", "\*.png", "\*.jpg", ".jpeg", "\*.gif"),

new ExtensionFilter("All Files", "\*.\*"));

File file = fileChooser.showOpenDialog(null); ¶ return new Image(file.toURI().toString()); ¶ }

private void SaveImageFile(Image image) throws IOException {

FileChooser fileChooser = new FileChooser();

fileChooser.setTitle("Save Image");

File file = fileChooser.showSaveDialog(null);

if (file != null) {

ImageIO.write(SwingFXUtils.fromFXImage(image,null), "png", file);

}

}

//private Color GetQuntum

private Image FP(Image image) throws IOException {

Image base = image;

Image fptn = QImage.LinearQuantization(base, 256);

Image optn = QImage.PopularityMethod(base,256);

Image fpty = QImage.DoToning(base, fptn);

Image opty = QImage.DoToning(base, optn);

BaseImage.setImage(base);

FPTNImage.setImage(fptn);

OPTNImage.setImage(optn);

FPTYImage.setImage(fpty);

OPTYImage.setImage(opty);

return image;

}

public void QuitProgram(ActionEvent actionEvent) {

Platform.exit();

}

public void OpenImage(ActionEvent actionEvent) throws IOException {

Image i = OpenImageFile();

FP(i);

//BaseImage.setImage(FP(i));

}

public void SaveImage(ActionEvent actionEvent) {

}

}

QImage.java

package lab5;

import javafx.scene.image.Image;

import javafx.scene.image.PixelReader;

import javafx.scene.image.PixelWriter;

import javafx.scene.image.WritableImage;

import javafx.scene.paint.Color;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

public abstract class QImage {

/\*\*

\*

\* @param cl Color for compare

\* @param cr Color for compare

\* @return Evaluation of the discrepancy

\*/

public static double ColorEST(Color cl, Color cr) {

return Math.pow((cl.getRed() - cr.getRed()) \* 255, 2) + Math.pow((cl.getGreen() - cr.getGreen()) \* 255, 2) + Math.pow((cl.getBlue() - cr.getBlue()) \* 255, 2) + Math.pow((cl.getOpacity() - cr.getOpacity()) \* 255, 2);

}

/\*\*

\*

\* @param il Color for compare

\* @param ir Color for compare

\* @return Evaluation of the discrepancy

\*/

public static double ImageEST(Image il, Image ir) {

if (il.getWidth() != ir.getWidth() || il.getHeight() != ir.getHeight()) throw new IllegalArgumentException("Images must be the same size!");

double error = 0;

PixelReader rl = il.getPixelReader();

PixelReader rr = ir.getPixelReader();

for (int x = 0; x < il.getWidth(); x++) {

for (int y = 0; y < il.getHeight(); y++) {

error += ColorEST(rl.getColor(x, y), rr.getColor(x, y));

}

}

return error;

}

/\*\*

\*

\* @param image image for palette application

\* @param palette palette for application

\* @return image with applied palette

\*/

public static Image ApplyPalette(Image image, ArrayList<Color> palette) {

PixelReader reader = image.getPixelReader();

WritableImage edit = new WritableImage((int) image.getWidth(), (int) image.getHeight());

PixelWriter writer = edit.getPixelWriter();

for (int x = 0; x < image.getWidth(); x++) {

for (int y = 0; y < image.getHeight(); y++) {

Color nowColor = reader.getColor(x, y);

Color newColor = palette.get(0);

double est = ColorEST(newColor, nowColor);

for (Color tryColor: palette) {

double tryEST = ColorEST(nowColor,tryColor);

if (tryEST < est) {

newColor = tryColor;

est = tryEST;

}

}

writer.setColor(x,y,newColor);

}

}

return edit;

}

/\*\*

\*

\* @param image

\* @param numberOfColors

\* @return

\*/

public static Image LinearQuantization(Image image, int numberOfColors) {

PixelReader reader = image.getPixelReader();

WritableImage edit = new WritableImage((int) image.getWidth(), (int) image.getHeight());

PixelWriter writer = edit.getPixelWriter();

byte trimDegree = (byte)Math.pow(2, ((24 - (byte)(Math.log(numberOfColors) / Math.log(2))) / 3));

for (int x = 0; x < edit.getWidth(); x++) {

for (int y = 0; y < edit.getHeight(); y++) {

Color color = reader.getColor(x, y);

writer.setColor(x, y, Color.color(color.getRed() - color.getRed() \* 255 % trimDegree / 255, color.getGreen() - color.getGreen() \* 255 % trimDegree / 255, color.getBlue() - color.getBlue() \* 255 % trimDegree / 255, color.getOpacity()));

}

}

return edit;

}

/\*\*

\*

\* @param image

\* @param numberOfColors

\* @return

\*/

public static ArrayList<Color> GetPopularColors(Image image, long numberOfColors) {

PixelReader reader = image.getPixelReader();

HashMap<Color,Long> colorPallet = new HashMap<>();

for (int x = 0; x < image.getWidth(); x++) {

for (int y = 0; y < image.getHeight(); y++) {

Color color = reader.getColor(x, y);

colorPallet.put(color, colorPallet.getOrDefault(color, 0L) + 1L);

}

}

ArrayList<Color> popularColors = new ArrayList<>();

colorPallet.entrySet().stream().sorted(Map.Entry.<Color,Long>comparingByValue().reversed()).limit(numberOfColors).forEach(colorLongEntry -> popularColors.add(colorLongEntry.getKey()));

return popularColors;

}

/\*\*

\*

\* @param baseImage

\* @param editImage

\* @return

\*/

public static Image DoToning(Image baseImage, Image editImage) {

if (baseImage.getWidth() != editImage.getWidth() || baseImage.getHeight() != editImage.getHeight()) throw new IllegalArgumentException("Images must be the same size!");

PixelReader baseReader = baseImage.getPixelReader();

PixelReader editReader = editImage.getPixelReader();

WritableImage writeImage = new WritableImage((int) baseImage.getWidth(), (int) baseImage.getHeight());

PixelWriter writer = writeImage.getPixelWriter();

for (int x = 0; x < baseImage.getWidth(); x++)

{

writer.setColor(x,0,editReader.getColor(x,0));

}

for (int y = 0; y < baseImage.getHeight(); y++) {

for (int x = 0; x < baseImage.getWidth(); x++) {

Color baseColor = baseReader.getColor(x, y);

Color editColor = editReader.getColor(x, y);

double errorR = baseColor.getRed() - editColor.getRed();

double errorG = baseColor.getGreen() - editColor.getGreen();

double errorB = baseColor.getBlue() - editColor.getBlue();

if (y != baseImage.getHeight() - 1) {

if (x != 0) {

writer.setColor(x - 1,y + 1, Color.color(editColor.getRed() + 0.25 \* errorR, editColor.getGreen() + 0.25 \* errorG, editColor.getBlue() + 0.25 \* errorB, editColor.getOpacity()));

}

writer.setColor(x, y + 1, Color.color(editColor.getRed() + 0.25 \* errorR, editColor.getGreen() + 0.25 \* errorG, editColor.getBlue() + 0.25 \* errorB, editColor.getOpacity()));

}

if (x != baseImage.getWidth() - 1) {

if (y != baseImage.getHeight() - 1) {

writer.setColor(x + 1, y + 1, Color.color(editColor.getRed() + 0.25 \* errorR, editColor.getGreen() + 0.25 \* errorG, editColor.getBlue() + 0.25 \* errorB, editColor.getOpacity()));

}

writer.setColor(x + 1, y, Color.color(editColor.getRed() + 0.25 \* errorR, editColor.getGreen() + 0.25 \* errorG, editColor.getBlue() + 0.25 \* errorB, editColor.getOpacity()));

}

/\*if (y != baseImage.getHeight() - 1) {

if (x != 0) {

writer.setColor(x - 1,y + 1, Color.color(editColor.getRed() + 3/16 \* errorR, editColor.getGreen() + 3/16 \* errorG, editColor.getBlue() + 3/16 \* errorB, editColor.getOpacity()));

}

writer.setColor(x, y + 1, Color.color(editColor.getRed() + 5/16 \* errorR, editColor.getGreen() + 5/16 \* errorG, editColor.getBlue() + 5/16 \* errorB, editColor.getOpacity()));

}

if (x != baseImage.getWidth() - 1) {

if (y != baseImage.getHeight() - 1) {

writer.setColor(x + 1, y + 1, Color.color(editColor.getRed() + 1/16 \* errorR, editColor.getGreen() + 1/16 \* errorG, editColor.getBlue() + 1/16 \* errorB, editColor.getOpacity()));

}

writer.setColor(x + 1, y, Color.color(editColor.getRed() + 7/16 \* errorR, editColor.getGreen() + 7/16 \* errorG, editColor.getBlue() + 7/16 \* errorB, editColor.getOpacity()));

}\*/

}

}

return writeImage;

}

/\*\*

\*

\* @param image

\* @param numberOfColors

\* @return

\*/

public static Image PopularityMethod(Image image, int numberOfColors) {

return ApplyPalette(image, GetPopularColors(image, numberOfColors));

}

}