Chapter – 1

Introduction

1.1 Introduction

Image Compression – A Brief Look

Image compression is a type of data compression which is applied to digital images.

It is used to reduce the cost for storage and of transmission of the image.

Algorithms usually take advantage of visual perception and the statistical properties of image data to provide superior results compared with generic data compression methods which are used for other digital data.

1.2 Our Project: Image Compressing using a Quad Tree Format

The Main Objective of our project is to build an image compression algorithm which can be used to compress images in any format and to any scale of compression.

Our Code is fully functional and can achieve compression without much loss in detail.

1.3 Our Reasons for choosing this Topic

We wished to implement this to

i) Understand the working and the concepts behind Image Compression which we use on a regular basis.

ii)To develop an easier to implement algorithm for Image Compression.

iii) we found that this topic will enable us to illustrate our understanding in a varied number of concepts we have learnt and build on them.

1.4 Conclusion

Photo Compressor is the technical process of reducing the file size of an image document [without compromising on its quality]. This makes the digital photo to retain its looks and physical characteristics, but with a much smaller size so that it fills less space and becomes acceptable when uploading on relevant sites.

The reduction in file size allows you to store the image more economically and efficiently, as the memory space it'll occupy will now be significantly less while the time and bandwidth required for uploading and downloading the image will also be at the barest minimum.

Chapter – 2

Literature Survey

2.1 Extensive Surveys

- Quadtree-based processing of digital images

  Ramin Naderi

  Portland State University

Region growing is the first problem considered. It is assumed that the image is available only in quadtree form.

2.2 Conclusion

The Survey gave a brief idea of the current methods and techniques used to check for electricity theft and also gave a clear idea on the approach and direction of our Project and that it can be implemented in a simpler form.

Chapter – 3

Methods Used

- **Entropy();**

Give the detail and the intensity of R,G and B values of the sub – image.

- **ChosseRandomPoint();**

Choses a random pixel in the sub – Image.

- **FindClosesetPoint();**

Finds the closest Point in the array list of the pixel co-ordinates to the random point choosen.

- **Colour();**

Finds the RGB value of each Pixel.

- **Fill();**

Fills a specified rectangle in the output image.

- **ColourAvg();**

Finds the average RGB value of the sub – Image.

- **Quadrant();**

i) Fills the specified using fill.

ii) Splits image/sub – image into 4 quadrants.

Chapter – 4

Libraries Used

Java.awt.Color;

Creates Colour by using RGBA values.

Java.awt.graphics;

Allows the algorithm to draw onto components.

Java.awt.Image.BufferedImage;

Used to handle and manipulate image data

Java.IO.file;

Allows the algorithm to use the system directory to access the file.

Java.IO.IOExecption();

Handles Exceptions thrown.

Java.imageio.ImageIO;

Allows access to various file formats for images.

Java.util.*;

Java util package contains collection framework, collection classes, classes related to date and time, event model, internationalization, and miscellaneous utility classes.

Java.awt.geom.Point2D;

The Point2D class defines a point representing a location in (x,y) coordinate space.

Chapter – 5

Code Flow Chart

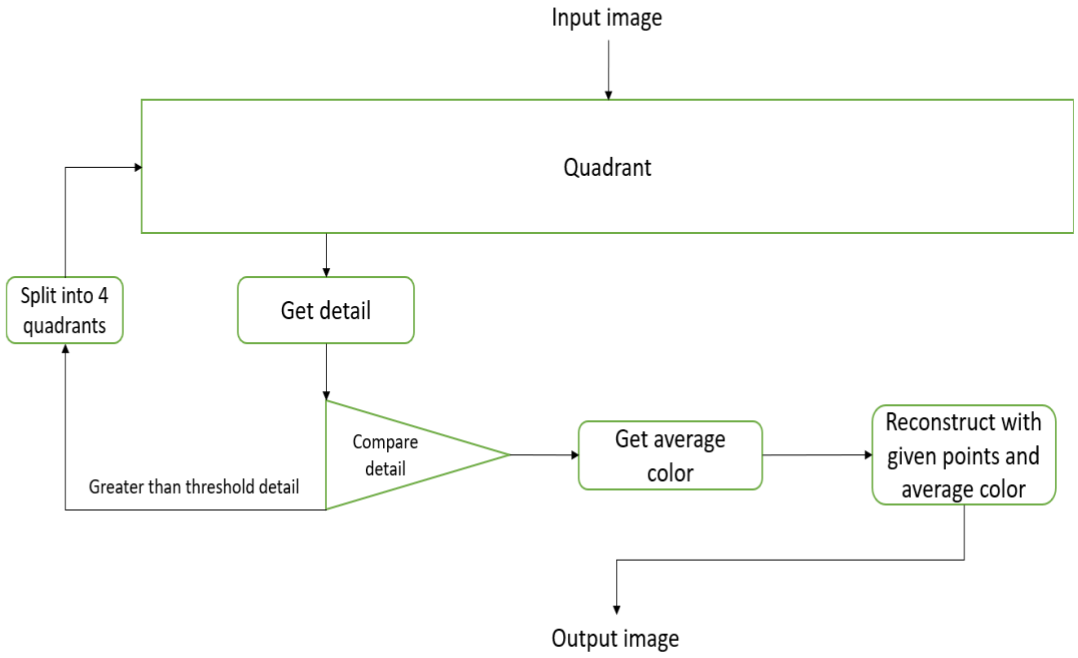

Fig 5.1 Code Flow Chart

Chapter – 6

Code

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import java.util.*;
import java.awt.geom.Point2D;

public class QuadTree {

	static int threshold = 5;
	static List<Point2D>centers = new ArrayList<>();
	static int MIN_SZ = 4;

	static double entropy(BufferedImage img, int x, int y, int szX, int szY) {
		double s = 0;
		Color avg = avg(img, x, y, szX, szY);
		for (int i = x; i < x + szX; i++) {
			for (int j = y; j < y + szY; j++) {
				Color c = new Color(img.getRGB(i, j));
				s += Math.abs(avg.getRed() - c.getRed());
				s += Math.abs(avg.getGreen() - c.getGreen());
				s += Math.abs(avg.getBlue() - c.getBlue());
			}
		}
		return (double) s / (szX * szY);
	}
	static void chooseRandomPoint(BufferedImage img, int x, int y, int szX, int szY){
		Random rng = new Random();
		int rngX = rng.nextInt(szX) + x;
		int rngY = rng.nextInt(szY) + y;
		Point2D rnd = new Point2D.Double(rngX, rngY);
		centers.add(rnd);
	}


	static Point2D findClosestPoint(int x, int y){
		Point2D closest = null;
		double minDist = Double.POSITIVE_INFINITY;
```

```
        Point2D p = new Point2D.Double(x, y);

        for(Point2D point: centers){
                double dist = point.distance(p);
                if(dist < minDist){
                        minDist = dist;
                        closest = point;
                }
        }
        return closest;
}

static void color(BufferedImage img, BufferedImage out) {
        for(int i = 0; i < img.getWidth(); i++){
                for(int j = 0; j < img.getHeight(); j++){
                        Point2D c = findClosestPoint(i, j);
                        out.setRGB(i, j, img.getRGB((int)c.getX(), (int)c.getY()));
                }
        }
}

static void fill(BufferedImage img, int x, int y, int szX, int szY, BufferedImage out) {
        Color avg = avg(img, x, y, szX, szY);
        Graphics g = out.getGraphics();
        g.setColor(avg);

                g.fillRect(x, y, szX, szY);
        g.dispose();
}

static Color avg(BufferedImage img, int x, int y, int szX, int szY) {
        int sumRed = 0;
        int sumGreen = 0;
        int sumBlue = 0;
        int count = 0;
        for (int i = x; i < x + szX; i++) {
                for (int j = y; j < y + szY; j++) {
                        Color c = new Color(img.getRGB(i, j));
                        sumRed += c.getRed();
                        sumGreen += c.getGreen();
                        sumBlue += c.getBlue();
                        count++;
                }
        }
```

```
            return new Color(sumRed / count, sumGreen / count, sumBlue / count);
        }

        static void rec(BufferedImage img, int x, int y, int szX, int szY, BufferedImage out) {
            if (szX <= MIN_SZ || szY <= MIN_SZ) {

                        fill(img, x, y, szX, szY, out);
                return;
            }

            if (entropy(img, x, y, szX, szY) > threshold) {
                rec(img, x + szX / 2, y, szX / 2, szY / 2, out);
                rec(img, x + szX / 2, y + szY / 2, szX / 2, szY / 2, out);
                rec(img, x, y, szX / 2, szY / 2, out);
                rec(img, x, y + szY / 2, szX / 2, szY / 2, out);
            } else {
                        fill(img, x, y, szX, szY, out);
                }
            }

        public static void main(String[] args) throws IOException {

            if (args.length >= 2) {
                MIN_SZ = Integer.parseInt(args[1]);
            }

            if (args.length >= 3) {
                threshold = Integer.parseInt(args[2]);
            }

            BufferedImage image = ImageIO.read(new File("Input pathway"));
            BufferedImage out = ImageIO.read(new File("Input pathway "));

            Graphics g = out.getGraphics();
            g.setColor(Color.BLACK);
            g.fillRect(0, 0, out.getWidth(), out.getHeight());
            g.dispose();
            int w = image.getWidth();
            int h = image.getHeight();
            rec(image, 0, 0, w, h, out);
            ImageIO.write(out, "png", new File("Input pathway "));
        }

}
```

Chapter – 7
Output

INPUT(1.81 mb) **OUTPUT(299 kb)**

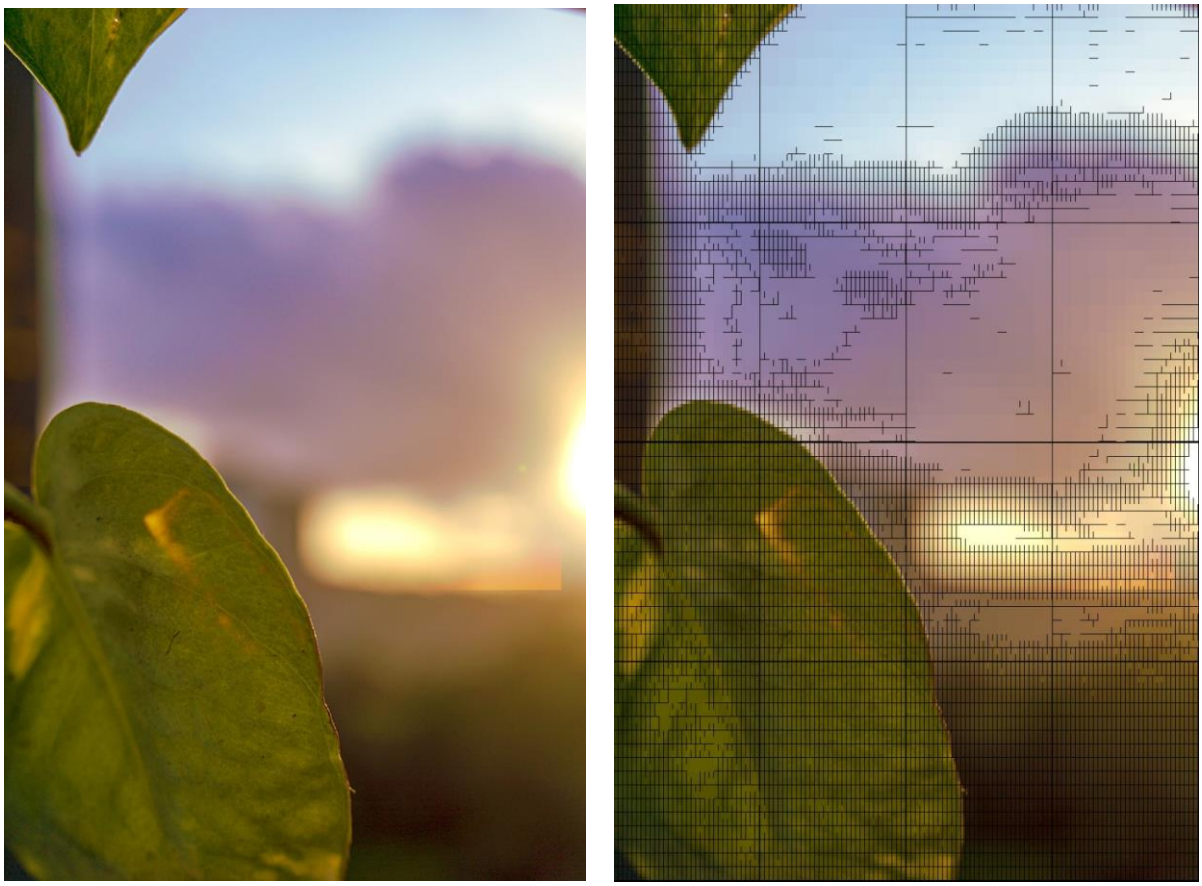

Fig 7.1 Shows input and output images

Chapter – 8
Conclusion

This project allows the user to compress any image of any format. Depending on the image complexity, the compression rate changes. Our project allows effective image compression by implementing quadtree structure. Note that the compression can take upto O(log n) time complexity depending on the complexity of the image.

Future scope:

- Image decompression algorithm can be written.
- Videos can be compressed by splitting them into frames.
- GUI can be implemented to improve user accessibility.

References

- Quadtree-based processing of digital images, Ramin Naderi ,Portland State University.
- Geeks for Geeks
- TutorialsPoint
- Stack Overflow