Gillian Schmitz

1835892

Abstract

This document contains the description and comparison of 2 different image compression algorithms, comparing both in their strong and weak points.

Image compression

For AZERTY

Contents

[Assignment 2](#_Toc137935913)

[The Plan 2](#_Toc137935914)

[Algorithm 1 – simple 2](#_Toc137935915)

[Colour Compression 2](#_Toc137935916)

[Run Length Encoding 3](#_Toc137935917)

[Rebuilding the image 3](#_Toc137935918)

[Algorithm 2 – complex 3](#_Toc137935919)

[Colour Space Conversion 3](#_Toc137935920)

[Chrominance Down Sampling 3](#_Toc137935921)

[Discrete Cosine Transform 4](#_Toc137935922)

[Quantization 4](#_Toc137935923)

[Run Length and Huffman Encoding 4](#_Toc137935924)

[Comparison 4](#_Toc137935925)

[Sources 4](#_Toc137935926)

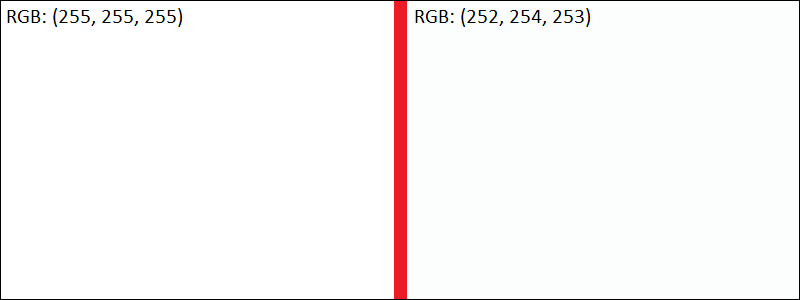
# Assignment

# The Plan

# Algorithm 1 – simple

The first algorithm uses a very simple technique, it mainly uses the Run Length Encoding (RLE) algorithm. But before RLE you can use colour compression to make RLE more effective. In short RLE uses a counter to count how many of the same colour is aligned after one another, for instance instead of saving “white, white, white, white”, you would save “white x 4”. Understanding the concept of this helps with why we use colour compression.

## Colour Compression

With colour compression we mean to generalize the colour values of the pixels, this will make our RLE algorithm more effective. This is because a lot of the time a pixel has just a slightly different value, while it is seemingly the same colour. The RGB value (0, 1, 0) is almost indistinguishable from the (0, 0, 0) RGB value. They are both so close to being black that you would really have to look for it to find it, and even then, it’s still very difficult to do. 

So, if we round each RGB value down we can generalize the colour value, the RLE algorithm is going to be better at picking up multiples of that rounded colour. This is especially effective for an object with the same colour, because each pixel will vary slightly because of the light/shading creating very small differences in the RGB values of the pixels.

The colour compression algorithm used in this algorithm is very simple, it takes the original value from 0 to 255 and floors it to the closest multiple of 4. It achieves this using the modulo operator.

(6, 3, 1) → (6 – 6 % 4, 3 – 3% 4, 1 – 1 % 4)

= (6 – 2, 3 – 3, 1 – 1)

= (4, 0, 0)

(4, 1, 3) → (4 – 4 % 4, 1 – 1% 4, 3 – 3 % 4)

= (4 – 0, 1 – 1, 3 – 3)

= (4, 0, 0)

Thus:  
(6, 3, 1) → (4, 0, 0)

(4, 1, 3) → (4, 0, 0)

## Run Length Encoding

## Rebuilding the image

# Algorithm 2 – complex

The second algorithm is based on the JPEG image compression algorithm, to reconstruct this method 5 main steps are used:

## Colour Space Conversion

The first step of the algorithm is Colour Space Conversion. A regular image is made from a matrix of pixels, with each pixel having 3 colour values: red, green, and blue (RGB). But RGB aren’t the only colours you van use to create an image. The JPEG algorithm uses the YCbCr colour space, this colour space consists of Luminance (Y), Blue Chrominance (Cb) and Red Chrominance (Cr). The Luminance layer has the black/white values from the image, whilst the Blue- and Red Chrominance layers consists of the colours, that altogether create the same image as when you would use the RGB colour space.

## Chrominance Down Sampling

With the YCbCr colour space, we can use a trick on our eyes. We can down sample the chrominance layers whilst keeping the sharpness by using the Luminance layer. This creates a – still good-looking – image when we deleted half of the data we started with. Both the Chrominance layers are 1/4th the original size.

Y + Cb + Cr → 1 + 1 + 1 = 3

Y + Cb + Cr → 1 + ¼ + ¼ = 1.5

The down sampling on the chrominance layer works by taking a 2×2 square of pixels and putting the average of those 4 values in the new pixel. The old layer is deleted and when reconstructing the image, the layers that are down sampled get upscaled back to the original resolution, creating the full image.

## Discrete Cosine Transform

## Quantization

## Run Length and Huffman Encoding

# Comparison

# Conclusion

# Sources