university of groningen
faculty of science and engineering
bernoulli institute

Image Bit-Depth Manipulation using Distance Fields

Master Thesis CS

July 2023

Student: Niels Bügel

Primary supervisor: Jiří Kosinka

Secondary supervisor: Dênnis José da Silva

ABSTRACT

With the increase in popularity of high dynamic range (HDR) images, there is also an increasing demand for methods that can increase the bit-depth of existing low dynamic range images. Additionally, HDR images come with larger storage requirements, which is often undesirable. We present a method that can be utilised for both bit-depth expansion and bit-depth compression. To enhance the bit depth, we interpret the input image as a 3D surface. Our method makes use of signed distance field blending to smooth this surface, allowing us to remove visible contours while maintaining edge data. We make use of this technique to construct the decompression step in a new lossy bit-depth compression scheme. During the compression step, we remove the grey levels that are not expected to contribute significantly to the reconstruction process. The evaluation of the scheme is done using NVIDIA's ꟻLIP algorithm; a state-of-the-art metric for assessing perceived differences between images. For the implementation and evaluation of our method, we developed a new tool called NITRO. This is a powerful node editor that allows for the construction of complex image-processing routines via a visual graph representation. Using this tool, we showcase and evaluate the performance of the compression scheme on various images.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The number of colours that can be represented by a single pixel of an image is determined by its bit-depth. Most images are 8-bit, meaning that a single pixel in a single channel can represent 256 unique values, often 0–255. However, there are also numerous cases where higher or lower bit-depths are desirable. High-bit-depth images, also referred to as high dynamic range (HDR) images, are common in medical imaging, where monochromatic images typically have bit-depths ranging from 12 to 16 bits [56, 65]. Remote sensing techniques such as LiDAR also tend to use higher bit-depth images [24, 7]. On the other hand, low-bit-depth images are desirable when storage size, processing speed and bandwidth usage are important considerations [39, 17]. Given these various applications, methods for manipulating bit-depth are becoming increasingly important.

The reduction of bit-depth is called quantisation [58, 31, 51]. This process results in a reduction of the number of colours or greyscale values that can be presented in an image. Quantisation is often noticeable if the bit-depth is reduced to a low enough value and can result in the appearance of banding as seen in Figure 1.1. While a human observer is unlikely to notice the difference between higher bit-depth images (e.g. 8 and 16-bit), the differences tend to show once the images are being processed. Lower bit-depth images have reduced flexibility during processing, as there is less information to work with. As a result, significant manipulation of the colours tends to show the limitations of lower bit-depth.

The opposite of quantisation is the process of increasing bit-depth, frequently referred to as bit-depth expansion or bit-depth enhancement. This is the primary focus of this thesis, as we describe a method for bit-depth expansion using signed distance fields. While there exist methods that use deep learning to enhance bit-depth [83, 84, 13, 44, 43, 81], these can produce unpredictable results when being applied to images not included in the training data set. As such, we opted for a more traditional approach where the result is entirely determined by the information present in the input image. We do not claim that our method can outperform these deep learning methods. Instead,

(a) Input

(b) Quantised (3-bits)

Figure 1.1: Quantisation of a gradient resulting in visible banding.

we want to show that consistent good results can still be obtained efficiently and effectively using just the input image data.

One of the main applications of quantisation is reducing the image file sizes. Consequently, compression plays an important role in this. Compression can broadly be divided into lossless and lossy compression. In lossless compression schemes, data can be decoded from its compressed form without loss of information. A popular example of a lossless image compression scheme is PNG compression [10]. On the other hand, in lossy compression schemes, there is some degree of data lost during the compression process. While this tends to give a decrease in visual quality, it has the advantage that the compression ratios are often far better than those of lossless compression schemes. Lossy compression algorithms typically provide a way to control the trade-off between compression ratio and loss of visual quality, as is the case with JPEG compression [73, 64, 3].

1.1 PROPOSED METHODS & OVERVIEW

In this thesis, we describe two methods: a bit-depth expansion method and a lossy compression scheme. The idea behind the bit-depth expansion method is to interpret the image as a voxel height map. Using this interpretation, each grey level represents a unique horizontal slice or layer of this 3D surface. A low-bit-depth image, therefore, has fewer layers than a high-bit-depth image. The goal of the expansion method is to introduce new layers in such a way that the resulting 3D surface becomes increasingly smooth. By introducing these new layers, we effectively introduce gradients in areas that were previously flat colours. This reduces the appearance of banding and contours, which gives the appearance of increased bit-depth.

We leverage our newly developed bit-depth expansion technique to construct a lossy compression scheme. To effectively compress the image data, we use quantisation in combination with an efficient encoding algorithm. We compare our compression scheme to JPEG compression and analyse how it performs in terms of compression ratios and visual quality loss. For the implementation of the scheme, we built a tool called NITRO. NITRO is a visual, non-destructive node editor that can be used to construct image processing routines. It supports numerous filters and image operations that enable the user to construct a custom image-processing pipeline. This tool was invaluable in the development of our method as it allowed for rapid testing of various approaches.

There are three important remarks about our proposed bit-depth expansion method. First, it does not introduce artificial details in areas of the image where there originally were none; it only smooths existing contours. Introducing new details in the dark or bright regions could be achievable by using a data-driven approach, but this is not what our method uses. Second, while we use a 3D interpretation to reason about our method, we do not explicitly work in 3D. In that sense, this is similar to watershed segmentation [71], which also does not make use of explicit 3D operations, despite its intuition. Finally, the methods we discuss primarily focus on greyscale images. However, we briefly discuss how it can be extended to work for colour images.

The remainder of this thesis is organised as follows. We first discuss existing methods for quantisation, bit-depth expansion, and image compression in Chapter 2. Next, we introduce our method for enhancing bit-depth in Chapter 3 and show how this technique can be used for a compression scheme in Chapter 4. We briefly discuss the newly developed tool NITRO and the implementation details of the method in Chapter 5. Prior to presenting the results, we outline the methodology used to assess the effectiveness of the scheme in Chapter 6. The results and evaluation of the scheme can be found in Chapter 7. Finally, we provide concluding remarks in Chapter 8 and a number of future possible improvements in Chapter 9.

2 BACKGROUND & RELATED WORK

An image I can be defined as a rectangular grid of size $m \times n$ for which each pixel $I(\mathbf{p})$, $0 \leq \mathbf{p}_x < n$, $0 \leq \mathbf{p}_y < m$ maps to a particular greyscale value or colour. For a quantised image L, the pixel values $L(\mathbf{p})$ generally do not map to a greyscale or colour value, but rather to an index in a colour palette CP_L. In this thesis, we use the notation $CP_L(i)$ to refer to the greyscale value/colour at index i for the image L. For a greyscale image, the colour palette is ordered, which means that for an image consisting of k colours, we have:

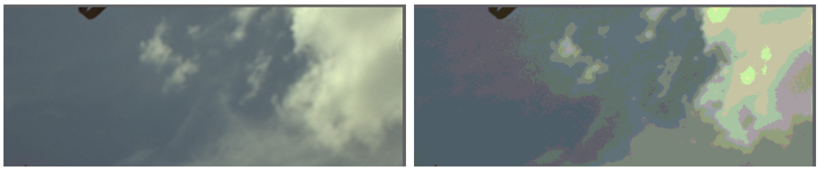
$$CP_L(i) < CP_L(i+1) \text{ for all } 0 \leq i < k.$$

Now that we have established some basic notations, we first look at existing methods for bit-depth manipulation. This involves quantisation (Section 2.1) and bit-depth expansion (Section 2.2). In Section 2.3 we discuss various image compression approaches.

2.1 QUANTISATION

The quantisation process consists of two main components: the generation of a colour palette CP_L and the mapping of each pixel $L(x, y)$ to a colour $CP_L(i)$ in the colour palette. The lower bit-depth that is used, the smaller the size of the colour palette. The difference between the input image and the quantised image is commonly referred to as the quantisation error. Numerous techniques exist for reducing the bit-depth of images, the most straightforward approach being uniform quantisation [31]. To reduce an image to d bits, this method creates a colour palette by splitting the colour range into 2^d equally spaced intervals. The pixels are then simply mapped according to which interval they fall in. Due to its simplicity, the method is fast but fails to produce good results when the intervals for images with an unbalanced histogram. To combat this, effective quantisation methods take the image data into consideration. An example of this is the median-cut algorithm [38], which continually splits the colour palette of the input image until 2^d regions remain. Other examples of these so-called splitting algorithms are the RWM-cut algorithm [79] and octree-based algorithms [28].

Alternatively, one can treat quantisation as a clustering problem. Various algorithms exist that use this approach. For example, the Lloyd-Max algorithm performs quantisation by minimising the least-squares error between the original data and the quantised data [45]. Another popular approach is using the (accelerated) k-means algorithm [15, 70,



(a) Input (b) Quantised (3-bits)

Figure 2.1: False contours appearing due to the RGB channels being misaligned as a result of the colour quantisation process.

36] or c-means algorithm [16] to find the most representable colour palette of the image. These methods produce good results and do not suffer from the same issues as uniform quantisation [54]. However, they can be expensive to compute.

A popular technique to improve the visual quality of quantised images is called dithering [11, 51]. This technique introduces controlled noise before or during the quantisation process to distribute the quantisation error. Consequently, the appearance of issues such as banding and contouring is reduced, while still maintaining the same bit-depth as quantisation without banding. However, due to the noise patterns dithering introduces, our proposed reconstruction method does not work well with it.

2.1.1 *Colour Quantisation*

Colour images are typically represented by multiple monochromatic channels that each describe a separate component. In sRGB images, these channels are the red, green and blue channels. To apply quantisation to a colour image, there are generally two main strategies. The first is to treat all channels at once, creating a colour palette consisting of RGB colours and a single channel containing the corresponding mappings, as done by e.g. [17]. However, a significant number of quantisation methods only work on greyscale images. This is where the second main strategy comes in, which treats each channel separately. Due to its simplicity, several algorithms do this in (a variant of) RGB colour space [39]. However, as the RGB channels each encode both colour and luminance information, a misalignment of the channels can result in false contours, as seen in Figure 2.1. A better approach is to process the channels individually in a colour space that splits luminance and chrominance. The YC_bC_r colour space is well suited for this, as it splits the luminance channel (Y) from the blue (C_b) and red (C_r) chrominance channels. Using such a colour space also enables quantisation methods to exploit the fact that humans are less perceptible to changes in chrominance than in luminance [68]. As such, it often pays to use more bits for the luminance channel, while using fewer for the chrominance channels.

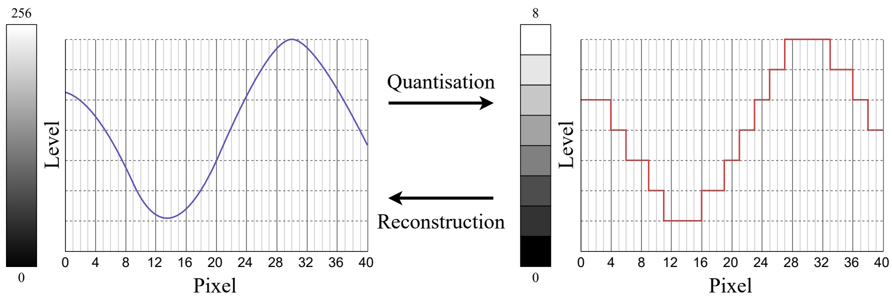


Figure 2.2: Signal quantisation and reconstruction for a simple 1D greyscale image.

2.2 BIT-DEPTH EXPANSION

With the growing availability of HDR displays, finding effective and efficient methods of how to display low-bit-depth images has become an increasingly important problem [83]. In addition to this, being able to reconstruct quantised images to some degree is also beneficial as quantisation often plays a role in compression methods. Quantisation and reconstruction are tightly related, as seen in Figure 2.2. The simplest way to convert an l-bit image to an h-bit image is to append $(h - l)$ zeros to each pixel value [69], a method referred to as zero-padding. Note that this method is intended for uniformly quantised images that do not rely on colour palettes, but store the grey values directly (although it can be adapted accordingly). The method clearly increases the bit depth in a fast and simple manner, but it does not improve the visual quality of the image. To improve the quality of the reconstructed image, most bit-depth expansion techniques focus on banding and contour removal. With the rise in popularity of the learning-based approach, we differentiate between two main categories: traditional methods and methods that involve some degree of learning. The traditional methods typically involve filtering or signal smoothing.

An example of a filter-based method is the approach presented by Chun Hung et al. [42]. Their approach consists of three stages: first the image is zero-padded, next smooth regions are identified, and these are smoothed using a low-pass filter in the final stage. Similarly, the method proposed by Teguchi et al. [66] uses an edge-preserving low-pass filter with an adaptive window size to achieve contour smoothing. The issue with filter-based approaches is that the filters are typically not fully accurate, which can result in blurred edges [13].

On the other hand, Cheng et al. [19] proposed a method based on signal smoothing that fills in the gaps between contour edges with smooth gradients. While this method produces good results, it has issues with local extrema as it fails to construct a smooth transition in these regions. Cheng et al. [18] use this method in a later work that combines this bit-depth expansion and dithering to remove the appearance of false contours. They do this by using the enhanced bit-depth image to apply a specific dithering pattern. The result is an image that maintains the same bit-depth as the input but has improved visual quality. Other methods using dithering exist [8], but these do not increase bit-depth without explicit zero padding.

An example of a method involving deep learning, Byun et al. introduced BitNet [13], a CNN-based approach for enhancing bit-depth. In contrast to other similar methods, their approach treats all the RGB channels at once. This has the advantage that colour restoration in the case of false contours is greatly improved. Other CNN-based approaches exist [43], but their execution times are typically high [13]. One common issue with learning-based methods is that they require extensive training data. Additionally, the results are highly dependent on the quality and extent of the training data, which can cause it to produce unpredictable results for certain images.

2.3 IMAGE COMPRESSION

Image compression is a popular topic and new methods are still constantly in development. This section does not serve as a complete and extensive summary of all of them, but we try to capture some of the most prominent ones and the different approaches and techniques that they use.

In its simplest form, the pixels of an image can be represented as one or more byte sequences. As such, one does not necessarily require a specialised method to compress these but can use any compression method that works on byte sequences. The most well-known of these methods is arguably the lossless compression method called run-length encoding [60]. The idea is that contiguous sequences of the same value can be represented much more efficiently by simply denoting the value and how often it appears. Due to its simplicity, run-length encoding is still a popular technique and is commonly used as part of existing compression schemes.

However, compression algorithms typically look for more ways to reduce data redundancy. A popular choice is the DEFLATE algorithm proposed by Oswal et al. [52]. The DEFLATE algorithm provides a lossless compression scheme that consists of two phases. First LZ77 [85] compression is used for a dictionary-based compression phase. In the second phase, Huffman coding [33] is used to compress this further. DEFLATE is a popular choice due to its speed. As such, it is used in various formats such as zip, gzip, and PNG compression [59, 40, 10]. There are, however, algorithms out there with better compression ratios, such as Bzip2, PPM, and LZMA, but these tend to be slower in terms of speed [29].

PNG compression [10] makes use of the DEFLATE algorithm as its main tool for encoding. In addition to this, it also comprises a preprocessing step and allows for different filters to be used to predict the values of the next colour value. The philosophy behind this is that pixel values in most images tend to have some spatial relation to their neighbouring pixels. As such, PNG tends to perform better on images with fewer fine details, such as line art or images with smooth gradients [62]. PNG compression has a variable compression level parameter that can be set by the user. It ranges from 0 to 9 and affects both the compression speed and resulting file size.

Arguably the most well-known lossy compression method is JPEG compression [73]. It divides an image into small blocks and transforms each of these blocks using the discrete cosine transform (DCT) [2]. Using the DCT as one of the building blocks for a compression scheme is not unique to JPEG as other image formats such as webp [80] and HEIF [30] also use this approach. In JPEG compression, the coefficients of the DCT are then quantised so that they can be efficiently encoded. The coefficients affected most by the quantisation tend to be the ones representing higher frequencies, which are the parts of the image that are typically the least noticeable. For colour images, JPEG does not use the sRGB colour space, but rather the YC_bC_r colour space. This allows for extra compression by lowering the resolution of the C_b and C_r components, as the human eye is less sensitive to changes in chrominance compared to luminance [68]. The successor of the JPEG format is JPEG 2000 [64], which uses a wavelet [63] based approach instead of using the DCT. The authors state that the JPEG 2000 format outperforms the original JPEG format by 10-20% for high-quality images. The next generation of the JPEG format, called JPEG XL [3], aims to improve this even further. For similar quality images, file sizes are only 30-40% of those produced by the JPEG format.

Other approaches to image compression include methods that use image skeletons [74, 75] or function-fitting [24, 53]. These methods use a different image representation to reduce data redundancy. All in all, there are various techniques that are used when creating a compression scheme. In the case of lossy compression schemes, the compression effort is typically focused on the higher frequencies, as these differences tend to be less noticeable in the decompressed images. Naturally, bit-depth and quantisation play an important role in this process.

3 BIT-DEPTH EXPANSION

Our proposed method for bit-depth expansion could be categorised as a signal smoothing method that makes use of interpolation. A quantised signal consists of a limited number of levels, so the idea behind our method is to smooth a low-bit-depth signal by inserting new levels in such a way that the reconstructed signal shows gradual changes in intensity (Figure 2.2). In the case of images, we can interpret these as 3D surfaces where each grey level corresponds to a given height. Consequently, low-bit depth images have a small number of layers and therefore contain large flat areas with abrupt transitions to the layers above them. In higher bit-depth images, these flat areas are generally very rare. The goal is therefore to make the transition between grey levels gradual, effectively creating gradients instead of flat areas. This can be seen for a simple 1D signal in Figure 3.1.

While the idea is simple, the solution is not necessarily straightforward. Applying techniques suited for 1D signal-smoothing are generally not easily extended to 2D signals. One can apply 1D signal processing methods to 2D signals by applying them row-wise or column-wise (or both). However, this requires the method in question to be separable, which is not the case for bit-depth expansion. An alternative idea would be to use explicit 3D geometry to construct the smooth surface, but this presents numerous challenges such as how to handle the topology. While subdivision methods could be applied here, the resulting algorithm would be complex and unlikely to be efficient due to the large amounts of geometry introduced. Alternatively, an implicit surface could be fitted, but this is also unlikely to be efficient and tends to be rather complex. As such, we decided to approach it differently.

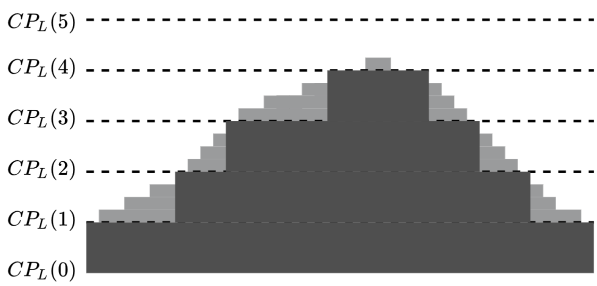


Figure 3.1: A quantised 1D signal (dark grey) consisting of 6 colours and its smoothed signal as a result of signal smoothing (light grey).

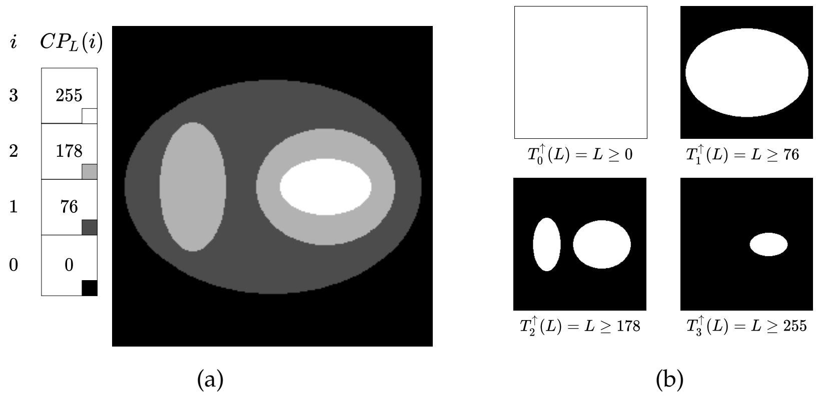


Figure 3.2: The upper-level sets of a given image L. (a) Image L of bit-depth $d = 2$ consisting of 4 unique colours as defined by the colour palette CP_L. (b) The upper-level sets of L, obtained by upper thresholding at each of the colour palette values.

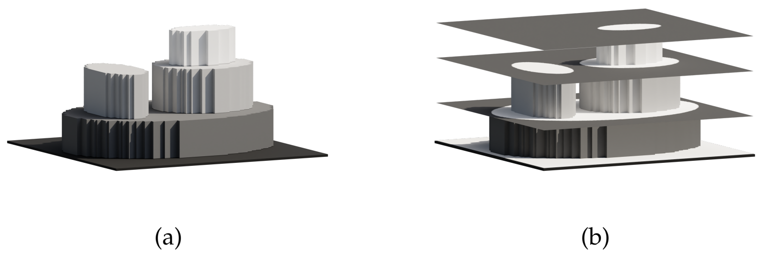


Figure 3.3: A 3D representation of images. (a) The 3D voxel height map of L. (b) Upper-level sets of L and their relation to the voxel height map. Each level set corresponds to a specific layer/slice in the 3D representation.

Given an indexed greyscale image L with a bit-depth d and a colour palette CP_L, we can construct its upper-level sets $T_i^\uparrow(L)$, $0 \leq i < 2^d$ by upper thresholding on each of its grey values:

$$T_i^\uparrow(L) = \{(\mathbf{p}) \in L \mid CP_L(L(\mathbf{p})) \geq CP_L(i)\}. \tag{3.1}$$

The upper-level sets of a simple 2-bit image can be seen in Figure 3.2. Each of these level sets corresponds to a particular horizontal slice or layer of the 3D image representation as seen in Figure 3.3. By increasing the bit-depth, we effectively increase the number of layers. As such, the problem we need to solve is how to insert new layers such that the 3D surface becomes increasingly smooth.

An important observation we can make here is that the upper-level sets can only shrink as the grey value increases:

$$T_j^\uparrow(L) \subseteq T_i^\uparrow(L) \quad \text{for all } i < j. \tag{3.2}$$

It is important to preserve this property when reconstructing the image, as undesirable shapes can be introduced if this is not the case.

As such, for a reconstructed image $\bar{I}$, any layer l we introduce between two layers i and $i+1$ must therefore satisfy:

$$T^{\uparrow}_{i+1}(\bar{I}) \subseteq T^{\uparrow}_{l}(\bar{I}) \subseteq T^{\uparrow}_{i}(\bar{I}) \quad \text{for all} \quad i < l < i+1. \tag{3.3}$$

To construct such a new layer l, we choose to smoothly blend the two consecutive level sets $T^{\uparrow}_{i}(L)$ and $T^{\uparrow}_{i+1}(L)$ based on an interpolation parameter t:

$$T^{\uparrow}_{l}(\bar{I}) = \text{blend}(T^{\uparrow}_{i}(L), T^{\uparrow}_{i+1}(L), t), \quad t \in [0,1], \quad i \leq l \leq i+1. \tag{3.4}$$

Here t can then be used to determine at which height between the two levels sets to insert the new layer and how much the surrounding layers contribute to the final shape at said layer.

It should be noted that a portion of the existing bit-depth expansion methods suffers from excessive blurring and failure to maintain edge data [48]. The advantage of only blending between two consecutive layers is that contours and sharp edges are preserved, provided that sufficient information is available in the input image. Sharp edges in an image can be seen as steep cliffs consisting of multiple layers in 3D. These steep cliffs are preserved due to Equation 3.3 and Equation 3.4. Consequently, textured regions and hard edges are maintained, while regions with only minor differences are smoothed.

3.1 SIGNED DISTANCE FIELDS

To smoothly and efficiently blend between layers, we use signed distance fields. A signed distance field is an alternative representation of binary shapes. Let B be a binary image where its foreground pixels make up the set Ω and SDF_B be the signed distance field of B. We denote the boundary of Ω by $\partial\Omega$ and the shortest distance of a pixel $\mathbf{p} \in B$ to $\partial\Omega$ by $sd(\mathbf{p}, \partial\Omega)$. Then SDF_B is defined as follows:

$$SDF_B = \begin{cases} sd(\mathbf{p}, \partial\Omega), & \text{if } \mathbf{p} \in \Omega \\ -sd(\mathbf{p}, \partial\Omega), & \text{otherwise.} \end{cases} \tag{3.5}$$

Using this, we can draw the following conclusions from a distance field value:

- $SDF_B(\mathbf{p}) < 0 \rightarrow B(\mathbf{p})$ is a foreground (interior) pixel.
- $SDF_B(\mathbf{p}) > 0 \rightarrow B(\mathbf{p})$ is a background (exterior) pixel.
- $SDF_B(\mathbf{p}) = 0 \rightarrow B(\mathbf{p})$ lies exactly on the boundary of Ω.

As such, given SDF_B, we can reconstruct B exactly by thresholding SDF_B with a value of zero:

$$B = \begin{cases} 1, & \text{if } SDF_B(\mathbf{p}) \leq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{3.6}$$

It should be noted that the case $sd(\mathbf{p}) = 0$ does not occur when working in raster graphics. This is due to the fact that in practice the distance is evaluated between foreground and background pixels, instead of using $\partial\Omega$. As such, the smallest possible distance is 1 in this case. However, we will still use this property later to efficiently reconstruct an image from a collection of upper-level sets.

3.1.1 *Distance Metrics*

There are numerous distance metrics that can be used for the calculation of a (signed) distance field. A distance metric specifies how to calculate the distance between two pixel locations. The most well-known ones are the Manhattan (MDT), Chessboard (CDT), and Euclidean (EDT) distance metrics. Let $\mathbf{p}$ and $\mathbf{q}$ be two pixel locations. These metrics are then defined as follows:

- $MDT(\mathbf{p}, \mathbf{q}) = |\mathbf{p}_x - \mathbf{q}_x| + |\mathbf{p}_y - \mathbf{q}_y|$
- $CBD(\mathbf{p}, \mathbf{q}) = \max(|\mathbf{p}_x - \mathbf{q}_x|, |\mathbf{p}_y - \mathbf{q}_y|)$
- $EDT(\mathbf{p}, \mathbf{q}) = \sqrt{(\mathbf{p}_x - \mathbf{q}_x)^2 + (\mathbf{p}_y - \mathbf{q}_y)^2}$

The most obvious choice for our method is the Euclidean distance as this gives the smoothest distance field. This is also the distance metric most often used in signed distance fields [50]. In addition to this, the Euclidean distance transform has the nice property that it is radially symmetric, which means that it is rotation invariant [25]. As such, it should produce the most natural and consistent blend shapes.

3.1.2 *Shape Interpolation*

A convenient property of signed distance fields is that we can perform pixel-wise interpolation to blend between two binary images. This ability to use signed distance fields for shape blending is common throughout the literature [21, 37, 32, 20]. One can linearly interpolate between two binary images B_0 and B_1 using t, $0 \leq t \leq 1$ by first interpolating their signed distance fields:

$$SDF_{B_t}(\mathbf{p}) = (1 - t) \cdot SDF_{B_0}(\mathbf{p}) + t \cdot SDF_{B_1}(\mathbf{p}). \tag{3.7}$$

This can then be used to construct B_t:

$$B_t = \begin{cases} 1, & \text{if } SDF_{B_t}(\mathbf{p}) \leq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{3.8}$$

Cohen-Or et al. [20] remark that simple linear interpolation between signed distance fields can present numerous issues. One example they provide is that in a linear blend between a thin rod and a rotated version of said rod, the object may shrink significantly or even disappear for particular values of t. However, given the constraint Equation 3.2, these or similar situations cannot occur in our setting. While their warp-based blending method might show some very minor improvements, it is unlikely to improve the reconstruction result significantly.

Apart from linear interpolation, it is also possible to use other interpolation methods. A number of these were tested, including monotone cubic interpolation [26]. However, we found that for the purposes of bit-depth expansion, these did not show a meaningful improvement in visual quality over linear interpolation. Additionally, defining proper boundary conditions for cubic splines is a challenging task in this context. This, on top of the fact that linear interpolation allows us to evaluate the final image pixel in constant time, is why we decided to stick with linear interpolation. In the next section, we show how we can use Equation 3.8 to construct a higher bit-depth image.

3.2 BIT-DEPTH EXPANSION

Given Equation 3.8, we can now insert arbitrarily many layers and using these layers, the image can be reconstructed. Instead of inserting a specific number of layers, we assume an infinite number of layers are inserted, resulting in a smooth 3D surface. We first make the observation that the greyscale value at a given pixel $\mathbf{p}$ corresponds to $CP_L(i)$ such that $\mathbf{p} \in T_i^{\uparrow}(L)$ and $\mathbf{p} \notin T_j^{\uparrow}(L)$, $i < j < 2^d$:

$$L(\mathbf{p}) = \max\{i : \mathbf{p} \in T_i^{\uparrow}(L)\}. \tag{3.9}$$

That is, the grey level of a given pixel corresponds to the height of the 3D surface at said pixel. Since we assume an infinite number of layers, a pixel $\mathbf{p}$ lies exactly on the surface when $SDF_{\bar{I}}(\mathbf{p}) = 0$. As such, we need to find $l \in \mathbb{R}$ such that $SDF_{T_l^{\uparrow}(\bar{I})}(\mathbf{p}) = 0$.

Due to the newly inserted layers satisfying Equation 3.2, the grey scale value at a given pixel $\mathbf{p}$ must lie between $CP_L(i)$ and $CP_L(i+1)$, where $i = L(\mathbf{p})$, which can also be seen in Figure 3.1. Let SDF_i be the signed distance field of layer i, i.e. $SDF_i = SDF_{T_i^{\uparrow}(L)}$. As we defined $i = L(\mathbf{p})$, we know that $\mathbf{p} \in T_i^{\uparrow}(L)$ and $\mathbf{p} \notin T_{i+1}^{\uparrow}(L)$, which allows us to conclude:

$$SDF_i(\mathbf{p}) \leq 0 \quad \text{and} \quad SDF_{i+1}(\mathbf{p}) \geq 0. \tag{3.10}$$

Given that we linearly blend between the two consecutive layers at i and $i+1$, we need to find t such that:

$$(1-t) \cdot SDF_i(\mathbf{p}) + t \cdot SDF_{i+1}(\mathbf{p}) = 0. \tag{3.11}$$

Given Equation 3.10, we can find $t \in [0,1]$ by rewriting Equation 3.11:

$$t = \frac{SDF_i(\mathbf{p})}{SDF_i(\mathbf{p}) - SDF_{i+1}(\mathbf{p})}. \tag{3.12}$$

Using t, the grey level of the reconstructed image $\bar{I}$ at $\mathbf{p}$ can be calculated using:

$$\bar{I}(\mathbf{p}) = t \cdot (CP_L(i+1) - CP_L(i)) + CP_L(i). \tag{3.13}$$

This allows us to evaluate the grey level at each pixel in $\mathcal{O}(1)$. As such, the performance of this stage is independent of the desired bit-depth and depends only on the image size. The final result $\bar{I}$ is not an indexed image. In practice, we assume $\bar{I}$ to be a floating-point image instead of integer-valued grey values to make this process easier. As such, its pixel values are in the range $[0, 1]$ where 0 equates to black and 1 to white. Consequently, we also assume that the colour palette of L consists of floating point values: $CP_L(i) \in [0, 1],\ 0 \leq i < 2^d$. Using floating-point images has the advantage that Equation 3.13 can be evaluated directly without any intermediate rounding. It can then be easily converted to an integer image I_{int} of the desired bit-depth k using:

$$I_{int} = \lfloor \bar{I}(\mathbf{p}) \cdot 2^k - \frac{1}{2} \rfloor. \tag{3.14}$$

The pseudocode for the reconstruction process can be seen in Algorithm 1.

Algorithm 1: reconstruct(L, CP, SDF)

Input : A d-bit indexed image L of size $m \times n$, its colour palette CP_L with $CP_L(i) \in [0, 1],\ 0 \leq i < 2^d$, and an array of signed distance fields $SDFs$ defined as $[SDF_{T_0^\uparrow}, SDF_{T_1^\uparrow}, \ldots, SDF_{T_{2^d}^\uparrow}]$.

Output: The reconstructed output floating-point image $\bar{I}$.

for y = 0 to m **do**
    for x = 0 to n **do**
        $i_0 \leftarrow L(x, y)$;
        $i_1 \leftarrow L(x, y) + 1$;
        $SDF_i \leftarrow SDF[i_0](x, y)$; // $SDF[i_0] = SDF_{T_{i_0}^\uparrow(L)}$
        $SDF_{i+1} \leftarrow SDF[i_1](x, y)$; // $SDF[i_1] = SDF_{T_{i_1}^\uparrow(L)}$
        $t \leftarrow \frac{SDF_i}{SDF_i - SDF_{i+1}}$; // Equation 3.12
        $\triangle \leftarrow CP(i_1) - CP(i_0)$;
        $\bar{I}(x, y) \leftarrow t \cdot \triangle + CP(i_0)$; // Equation 3.13
    end
end

3.3 DISTANCE FIELD EVALUATION

Evaluating Equation 3.12 requires the calculation of a signed distance field for every grey level of the input image. However, calculating for every foreground pixel the distance to the nearest background pixel (or vice versa) is an expensive task, with the naive implementation taking $\mathcal{O}(m^2 \cdot n^2)$ operations for an image of size $m \times n$. As such, numerous algorithms have been proposed to do this efficiently. For the Chessboard and Manhattan distance, a simple two-pass approach can be used, where the image is first scanned row-wise, and then column-wise (or vice versa) [61]. For the Euclidean distance, this is more challenging to do efficiently as the problem is not easily separable. This gave rise to a number of algorithms that approximate the Euclidean distance transform [22]. However, there also exist methods that can evaluate the exact Euclidean distance field in linear time. Due to its good performance and relatively simple implementation, we use the distance transform as proposed by Meijster et al. [47, 25].

It should be noted that the above distance transform algorithms do not evaluate a signed distance field; they only evaluate the distance from each foreground pixel to each background pixel. As such, given a binary image B, the unsigned distance field DF_B is defined as:

$$DF_B = \begin{cases} sd(\mathbf{p}, \partial\Omega), & \text{if } \mathbf{p} \in \Omega \\ 0, & \text{otherwise.} \end{cases} \tag{3.15}$$

A signed distance field also requires the calculation of the distance from each background pixel to each foreground pixel. To do this, we can simply evaluate DF_{B^c}, where B^c is the complement of B. In order to obtain a signed distance field, we then simply subtract the distance field B^c from the distance field of B:

$$SDF_B = DF_B - DF_{B^c}. \tag{3.16}$$

3.4 SHADOWS & HIGHLIGHTS

While the blending approach works well, it is not immediately clear how to deal with the bottom and top-most layers. The bottom-most layer representing the darkest regions (shadows) is empty by construction (i.e. only contains foreground pixels) as seen in Figure 3.2. As such, the reconstruction result is what can be seen in Figure 3.4a, since the distance values for layer $i = 0$ are all constant. Instead, we aim to achieve the result in Figure 3.4b. To do this, the distance values cannot be constant. However, it is still important to be able to reconstruct $T_0^{\uparrow}(\bar{I})$ from the distance field using Equation 3.6. Given that a signed distance field stores the closest distance between different pixels, we can move the contours of the shape(s) it represents outwards by subtracting a positive constant value from each pixel. The idea is to

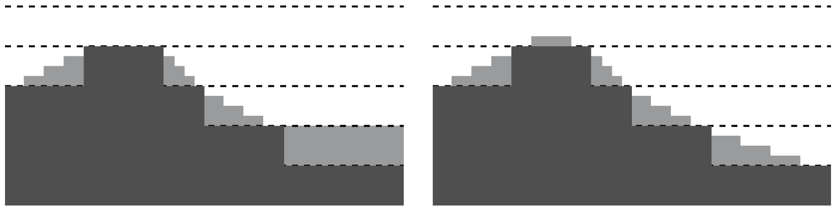


Figure 3.4: The effect of the blending approach in 1D signal reconstruction on the bottom and top layers. (a) Basic blending approach, where the top and bottom layers remain flat. (b) Improved blending strategy that gradually blends the bottom layer and introduces a smooth fall-off for the top layer.

construct SDF_0 by growing SDF_1 such that $\max(SDF_0) = 0$. This can be done by subtracting the maximum value of SDF_1. Therefore, the bottom-most layer can be evaluated as follows:

$$SDF_0 = SDF_1 - \max(SDF_1). \tag{3.17}$$

In the case of the top-most layer representing the brightest regions (highlights), there is no layer to blend to. This would result in a completely flat surface for these brightest pixels. To prevent this, we introduce an additional top layer and use a similar strategy as before. Given that the input image has a bit-depth d, we introduce a new layer SDF_{2^d}. However, this time we shrink the contours by *adding* a positive constant value such that $\min(SDF_{2^d}) = 0$. As such, this new layer can be calculated using:

$$SDF_{2^d} = SDF_{2^d-1} + |\min(SDF_{2^d-1})|. \tag{3.18}$$

In contrast to the bottom-most layer which already had the colour value $CP_L(0)$ associated with it, the introduction of an *extra* layer also requires an extra colour. An initial guess for this value might be to always let this be the brightest value a pixel can be. However, this causes issues for dark images whose maximum value is significantly lower than this. As such, we ideally use the maximum value of the input image I for this. However, this information is typically not available, since one often has access only to the quantised image L. Therefore, we estimate this value by simply extrapolating the colour palette based on the last two values:

$$CP_L(2^d) = 2 \cdot CP_L(2^d - 1) - CP_L(2^d - 2). \tag{3.19}$$

In the case of uniformly quantised images, this should reproduce exactly what we are after. Otherwise, it provides a good enough estimate. Overall, applying these techniques to the shadows and highlights is vital for the performance of our bit-depth expansion method.

3.5 BRIGHTNESS CORRECTION

Due to the nature of this approach, a given pixel value can only ever increase in brightness. This would not be an issue when the quantisation process is done by truncation. However, this is often not the case as this does not produce optimal quantisation results. As such, the shadows are lifted and the mean intensity of the reconstructed image is greater than that of the original.

To solve this issue, we introduce a simple brightness correction step. Let L be the quantised version of I with mean intensity $\mu(L)$ and let $\bar{I}$ be the reconstructed image with mean intensity $\mu(\bar{I})$. First, we note that quantisation preserves primarily the low-frequencies of an image. As these carry the majority of intensity information, we assume $\mu(L) \approx \mu(I)$. As such, the simplest way to do brightness correction is by subtracting the difference in mean image values:

$$\bar{I}_c = \bar{I} - (\mu(\bar{I}) - \mu(L)), \tag{3.20}$$

where $\bar{I}_c$ is the brightness corrected version of $\bar{I}$. The problem with this approach is that it reduces the brightness values uniformly, which tends to reduce the highlights too much. An improved approach to this is to take into consideration local illumination. Once again, we rely on the fact that quantisation removes primarily the high frequencies, so the low frequencies of a quantised image should be nearly identical to that of the input image. As such, we can take the difference in the low-frequency brightness instead of the mean brightness. This can be done by a low-pass filter such as a Gaussian blur:

$$\bar{I}_c = \bar{I} - (\bar{I} * G - L * G), \tag{3.21}$$

where $L * G$ denotes the convolution of L with the kernel G. Here both $\bar{I}$ and L are convolved with the same Gaussian kernel G defined as:

$$G(\mathbf{p}) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{p_x^2 + p_y^2}{2\sigma^2}}. \tag{3.22}$$

The standard deviation σ used to construct G affects the final output. Using an infinitely large value for σ will make this method equivalent to subtracting the mean difference as seen in Equation 3.20. However, using a value for σ that is too small (i.e. no blur) will result in $\bar{I}_c = L$. As such, σ should be large enough to mask out the contours of the quantised image, while being small enough to capture its lower frequencies. Since we are working in the discrete domain, the Gaussian kernel G has a maximum size s_G. Additionally, we already have information on the resolution of the input image, so we can first define s_G and use this to calculate σ. To do this, we let s_G depend on the size of L. Therefore, we define:

$$s_G = p_s \cdot \max(m, n), \tag{3.23}$$

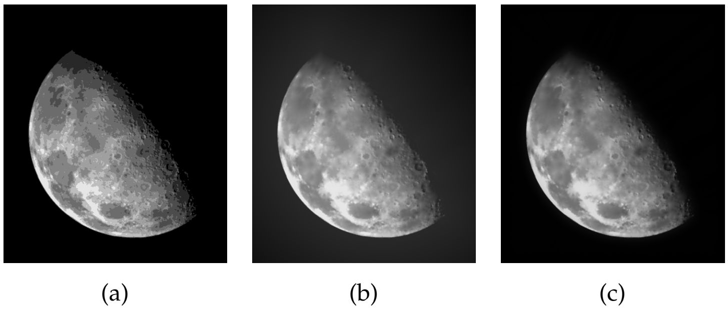


Figure 3.5: Bit-depth expansion with the brightness correction step. By construction our method only *increases* pixel intensities, requiring a correction to prevent overly bright output images. (a) Quantised input image with mean intensity 0.167. (b) Reconstructed image with a mean intensity 0.262. (c) Reconstructed image after brightness correction with mean intensity 0.167.

where $p_s \in [0, 1]$ is a scaling factor that can be defined by the user. Using the three-sigma rule [57], we infer σ from the kernel size using:

$$\sigma = \frac{s_G - 1}{6}. \tag{3.24}$$

This ensures that at least 99% of the values fit in the kernel size. The optimal value for p_s depends on the image, but we provide a reasonable default value for this in Chapter 6. The effect of the brightness correction step can be seen in Figure 3.5. The issue with the filter approach is that small shadow and highlight regions are still not fully accurate, as these higher frequency details are not captured due to the Gaussian filter. As such, for these regions, shadows tend to be lifted, while highlights can be diminished. Additional improvements could be made to this, but that is left to future work. The pseudocode for the entire bit-depth expansion process can be seen in Algorithm 2.

3.6 COLOUR IMAGES

While we primarily focus on greyscale images in this thesis, we also want to provide a simple and brief outline of how the method can be applied to colour images for the sake of completeness. Colour images are generally represented in (a variant of) the RGB colour space. However, similar to how quantisation methods can suffer from the false contours seen in Figure 2.1, applying the bit-depth expansion scheme on each RGB channel separately will result in similar issues. As such, a better approach is to use the YC_bC_r colour space instead. It should be noted, however, that if the quantisation process introduces false contours, the bit-depth expansion process will not eliminate these. Despite this, the method can still produce visually pleasing results on RGB images provided that the image was quantised properly, as we show in Section 7.1.

Algorithm 2: expandBitDepth(L, CP)

Input : A d-bit indexed image L of size $m \times n$, its colour palette CP_L with $CP_L(i) \in [0, 1]$, $0 \leq i < 2^d$ and a value p_s controlling the filter size in the brightness correction step.

Output: The reconstructed floating-point image $\bar{I}_c$.

$SDFs \leftarrow$ array of size $2^d + 1$; `// Evaluate signed distance field for each layer`

for $i = 1$ to 2^d **do**

  `// Due to` L `being indexed, we can threshold directly using` i

  $B \leftarrow$ threshold(L, i);

  $B^c \leftarrow$ 1 - B;

  $SDFs[i] \leftarrow DF_B$ - DF_{B^c}; `// Equation 3.16`

end

`// Bottom layer`

$SDFs[0] \leftarrow SDFs[1] - \max(SDFs[1])$; `// Equation 3.17`

`// Top layer`

$SDFs[2^d] \leftarrow SDFs[2^d - 1] + |\min(SDFs[2^d - 1])|$; `// Equation 3.18`

$CP[2^d] \leftarrow 2 * CP[2^d - 1] - CP[2^d - 2]$; `// Equation 3.19`

`// Reconstruction`

$\bar{I} \leftarrow$ reconstruct(L, CP, $SDFs$); `// Algorithm 1`

`// Brightness Correction`

$s_G \leftarrow p_s \cdot \max(m, n)$;

$\sigma \leftarrow (s_G - 1)/6$; `// Equation 3.24`

$G \leftarrow$ gaussianKernel(σ, s_G);

$\bar{I}_c \leftarrow \bar{I} - ((\bar{I} * G) - (L * G))$; `// Equation 3.21`

4 COMPRESSION SCHEME

Having introduced a means of enhancing bit-depth, the natural next step is to leverage this for a compression scheme. Quantisation provides an effective way of reducing the amount of data needed to be encoded. Clearly, using fewer bits to represent an image reduces the amount of data to encode. In addition to this, the reduction in bit-depth also means that fewer unique values can be represented. This increase in data redundancy allows the data to be encoded more efficiently. During the quantisation process, data is lost, meaning that the scheme we present here is a lossy compression scheme.

4.1 COMPRESSION

The initial idea of our compression scheme is relatively simple. The first step is to quantise the input image I to d bits. The quantisation process can be seen as the removal of layers. Ideally, the quantisation algorithm should progressively remove those layers that cause the least amount of visual quality loss. However, this is not a trivial problem to solve. Progressively removing layers can cause the algorithm to get stuck in local minima, which would ultimately mean that all possible combinations of layers would need to be evaluated, which would result in an extremely expensive algorithm. Even if one were to progressively remove layers, finding out which layer contributes least to the reconstruction is still extremely expensive. This was also noted by Wang et al. [76]. As an alternative, they proposed to use heuristics based on the histogram of the input image to select which layers to remove. While this sounds promising, we found that k-means outperformed this method for our purposes. As such, we opted to use k-means instead. Despite modern hardware getting faster, k-means can still be very slow for large images. Finding a global optimum to the k-means objective function is NP-hard [4], so in practice, the algorithm runs for a maximum number of iterations max_{iter}. As a result, the time complexity of k-means for an $m \times n$ image is $\mathcal{O}(m \cdot n \cdot k \cdot max_{iter})$. Clearly, its performance is therefore heavily dependent on the image size.

One can make the observation that k-means is only used for finding an optimal colour palette. This means that the spatial pixel information is not relevant. Using this information, we can gain a significant performance improvement by applying k-means on the histogram of the image instead. This requires a small modification of the algorithm, where the cluster centres are represented by and based on the indices of the histogram, while the means are updated by taking into consideration the number of pixels at each index. The result of this is surprisingly powerful, as the time complexity of k-means is no longer dependent on the image size, but on the bit-depth d_L of the input image: $\mathcal{O}(2^{d_L} \cdot k \cdot max_{iter})$. Even for 16-bit images (which would require a large histogram), this approach is already beneficial for image sizes starting at 256×256.

After quantisation of I, we obtain an image L with a colour palette consisting of $k = 2^d$ colours. Without any further modification, our image byte sequence still consists of $m \cdot n$ bytes (assuming the input image was 8-bit). Therefore, the $8 - d$ most significant bits are always zero. While most encoding algorithms can deal with this data repetition, packing the pixel data first tends to improve the compression ratios. By packing the pixel data, we can obtain a pixel sequence of $m \cdot n \cdot \frac{d}{8}$ bytes. The resulting byte sequence can then be efficiently encoded using any of the existing encoding algorithms such as DEFLATE [52]. It should be noted, however, that the pixel data does not necessarily need to be treated as a plain byte sequence, so existing image compression algorithms such as PNG could also be used. For the purposes of our compression scheme, we use the DEFLATE algorithm as implemented by zlib [23], providing an effective and easy way to compress the pixel byte sequence. We found that this was more effective in compressing the pixel data compared to PNG compression.

It is important to use a lossless compression algorithm when compressing the quantised image. Using a lossy algorithm might provide a better compression ratio, but it can introduce artefacts in the quantised image that affect the reconstruction. Recall that the bit-depth expansion process blends consecutive layers. As a result, any lossy algorithm that introduces new layers in between the existing layers, or shifts the layers in any way can severely affect the visual quality of the reconstructed image.

4.1.1 *Multi-scale Compression*

A compression scheme solely consisting of a quantisation step and bit-depth expansion does not necessarily provide good reconstruction results. To achieve performance similar to existing lossless schemes, the quantisation step should quantise the input image to 3 or 4 bits at most. However, despite the ability of our expansion scheme to improve the visual quality of low-bit-depth images, it does not introduce new details and in order for the compression scheme to be usable, the reconstructed results should be as close as possible to the input image. The quantisation process captures primarily the low frequencies, as these convey the most information regarding the overall structure and shape of the image. However, this means that a lot of the higher-frequency details are lost during this quantisation step. If we could efficiently find, extract and store the low-frequencies, then the quantisation process could focus primarily on the higher-frequency details, resulting in improved reconstructions.

As such, the idea behind this part is to first remove the low frequencies of the image and store them in an efficient manner. With the removal of the low frequencies, the quantisation step then has to encode a smaller range of colours and can capture the higher frequency details of the input image. For the extraction of the low frequencies, there are numerous approaches possible. Methods often use the frequency domain for this by using DCT or wavelet transforms. While these methods would likely produce comparable results, we propose a simpler approach that extracts the low frequencies of an image by applying a low-pass filter, using e.g. the Gaussian kernel described in Equation 3.22. We can extract the low-pass-filtered image from the input image to obtain a residual that only contains the higher frequencies. The issue with this approach is we also need to be able to efficiently store the low frequencies. This is where we can use an idea that is relatively common in multi-scale image processing.

Instead of filtering the input image directly, we first create a smaller resolution version S. This version can then be filtered and upscaled again to calculate the residual. The advantage of this is that the storage of the low-resolution image is extremely cheap. To obtain this small-scale image S, we resize the input image to a portion $p \in \mathbb{R},\ 0 < p < 1$ of its original size while maintaining the original aspect ratio:

$$S = \texttt{downscale}(I,\ p \cdot m,\ p \cdot n). \tag{4.1}$$

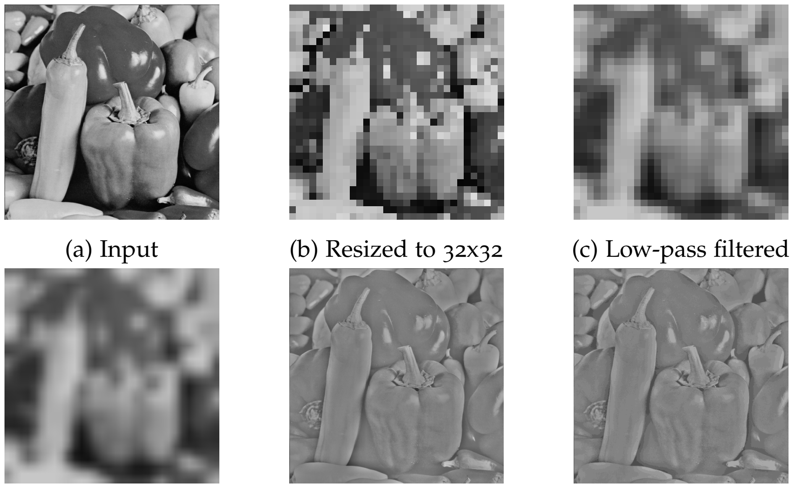
The exact value of p depends on the input image. We found that for the majority of the images, $p = \frac{1}{8}$ provided a good trade-off between compression size and visual quality. This means that for an input image of 256×256, the resized image is 32×32. We still need some degree of low-pass filtering as upscaling S results in a blocky image. There is the possibility of applying this filter after the upscale step, but we can also apply it beforehand:

$$S_b = S * G. \tag{4.2}$$

This is much faster performance-wise and still reduces the appearance of blockiness significantly. We can now extract the high frequencies from I by calculating the residual:

$$I_r = I - \texttt{upscale}(S_b,\ m,\ n). \tag{4.3}$$

The resulting residual I_r is then ready to be quantised to L using k-means. The outputs of the compression step are L and S and the entire process can be seen for the `peppers` image in Figure 4.1. We also observed that the method appears to perform best when `downscale` and `upscale` use some form of interpolation instead of using a nearest-neighbours approach. The choice between cubic and linear interpolation does not make a significant impact on the reconstruction, so



(a) Input (b) Resized to 32x32 (c) Low-pass filtered

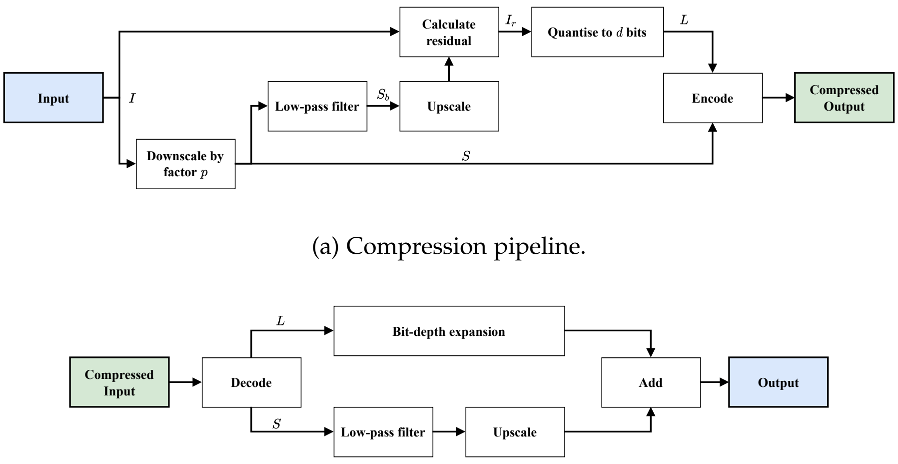
(d) Resized to input size (e) Residual (f) Residual (3-bits)

Figure 4.1: Compression steps for the `peppers` image. The final output of the compression algorithm consists of (a) and (f).

we opt to use linear interpolation for the sake of performance and simplicity.

Evidently, this approach comes with the added cost of storing the low-resolution version of the input image. However, the benefit outweighs the cost here, as the extra storage space required is generally only around 10% or less for $p = \frac{1}{8}$. This could be reduced even further by also quantising S. However, whether this is worth doing depends on the size of S. When S is small, the increase in compression effectiveness by quantising S is minimal. For larger images, this could make a bigger difference, but it is still only a small portion of the total file size that is affected by this.

It should also be noted that in addition to storing this low-resolution image, the final compression result would need additional metadata. The compression scheme essentially produces two separate images, so if these are to be stored in a single file, then extra information has to be added to denote where and how L and S are stored. While this adds to the complexity of the file format, it does not contribute to the final file size in a considerable way.



(a) Compression pipeline.

(b) Decompression pipeline.

Figure 4.2: Proposed compression scheme.

4.2 PIPELINE

The full pipeline for both compression and decompression can be seen in Figure 4.2. We should note that this compression approach is similar to existing ideas from multi-scale image processing [1], specifically, the image encoding scheme proposed by Burt et al. [12]. There are a few key differences, however. First, we only use a single image to store the lower frequencies. Second, we apply the low-pass filter before upscaling. Lastly, we use resizing with interpolation, whereas the method of Burt et al. uses simple downsample/upsample functions.

Experiments were also done involving perceptually uniform colour space. Greyscale images are typically linear in nature, but not perceptually uniform. The CIELAB colour space [14] provides such a colour space where the L-component is perceptually uniform. Additionally, it ranges from 0-100 instead of 0-255 for traditional greyscale images. As such, an idea was to remove the first 155 layers by simply converting to this colour space. However, this did not show any considerable improvements. This, in addition to the added performance penalty, made us decide not to include it. Various other modifications could be made, such as using different encoding/decoding methods, pre- or post-processing steps or alternative ways of extracting the low-frequency components. We leave this for future work, however.

5 NITRO

Many methods in the image processing domain consist of multiple stages and the construction of these stages can require a significant amount of trial and error. This process can be very time-consuming as the developer needs to modify the source code, recompile, and run the program countless times. Additionally, it is often desirable to compare various options or versions, which tends to result in a very complex UI and potential code duplication. Our newly developed open-source application NITRO[1] attempts to solve all of these issues. NITRO (Figure 5.1) is a powerful tool for building complex image processing routines in a non-destructive manner. At its core, NITRO is a visual node editor, meaning that the user can construct a custom graph of nodes to create their own pipeline. When building a new image processing pipeline, one often uses existing building blocks. This is where NITRO shines: it comes with a large collection of these building blocks in the form of nodes, ranging from simple transform and blend operations to more complex filters or frequency domain transforms.

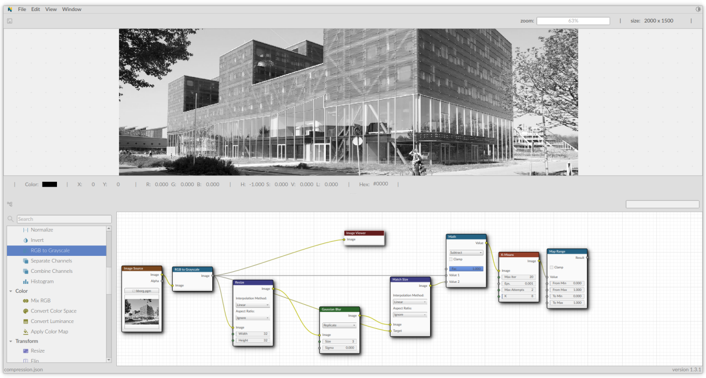


Figure 5.1: NITRO, a visual node editor for creating custom image processing pipelines.

NITRO was built using C++, Qt, CMake, and OpenGL. It uses OpenCV as the backbone for its image representation and the QtNodes library[2] to do the heavy lifting of the visual graph representation. However, a significant number of changes to this library were required in order to get the desired qualities. To this end, we created a fork containing these numerous modifications. The GUI of NITRO was inspired by Blender's Shader Editor given its user-friendliness and

1 https://github.com/BugelNiels/nitro
2 https://github.com/paceholder/nodeeditor

flexibility. The project was developed with extensibility in mind, which means that it is easy to define custom nodes, data types, or even custom GUI widgets. Below we outline a few of the implementation and design details of NITRO to highlight its usability for future image processing research.

5.1 NODES

Fundamentally, a node consists of a number of input ports, output ports, and an operation. This operation is what is executed when the node graph is evaluated. The input ports are used to retrieve the input data for the operation, while the results are propagated to the output ports. One of the primary goals of NITRO is to make it easy to add new nodes and it should streamline new research by removing as many obstacles as possible when designing new image processing operations. To this end, the way nodes are constructed was carefully designed to use a combination of the builder pattern and the command pattern to achieve this [27]. The combination of these patterns makes it straightforward to add new nodes while providing optimal flexibility. A simple example of a node implementation can be found in Appendix A.1.

5.2 DATA TYPES

A data type in NITRO is a wrapper for the data it represents. These data types are the things passed between the nodes when the graph is evaluated. At the time of writing, NITRO supports 4 different data types:

- Integers
- Doubles
- Greyscale Images
- Colour Images

We made the explicit distinction between greyscale and colour images, seeing that a number of image processing routines only work on (or only produce) greyscale images. However, this presented a challenge, as user experience is vital for NITRO. By using two different data types for images, it would mean that the user would always have to explicitly convert between greyscale and colour images using a dedicated node when using a node solely intended for greyscale/colour images. This is where the type conversion system comes in. Any data type can register a dedicated conversion function that specifies how it can be converted from any other data type. For example, a colour image can be converted to a greyscale image by extracting its luminance channel. Similarly, greyscale images can be converted to colour images by simply duplicating the greyscale channel into the RGB channels. Another example would be the conversion between integers and doubles. These conversions can all be specified within the data type itself, making it very flexible. A simple data type implementation example can be found in Appendix A.1.

There are some use cases where — even though a type conversion is possible — a node should not accept said type as input. For example, any double input port always accepts integers (as defined in the double data type). However, in certain cases, it might also be desirable to accept a greyscale image instead of a plain double. This prevents the user from having to spawn separate nodes when working with operations that are applicable to both images and numbers (such as basic math operations). This is why it is also possible to specify additional data types that a given port should accept.

While the main purpose of NITRO is to construct image processing pipelines, its core functionality is not dependent on images. As a matter of fact, NITRO can work with any data type the developer defines. This means that it is possible to define custom data types to extend the functionality further. Some possible future extensions to NITRO could be support for e.g. strings, meshes, curves, etc.

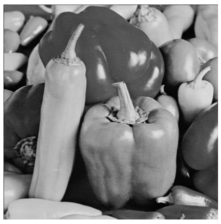
5.3 MODULES

To facilitate the addition of new data types and nodes, NITRO uses a module system. Not only does this promote decoupling, but it also allows developers to work on their own modules independently without affecting the source code of NITRO itself. The module system allows the developer to separate certain functionality into a module, which can then be enabled/disabled at compile time. This means that e.g. support for meshes and geometry manipulation can be implemented as a separate module. Additionally, the user can then decide which modules to include and/or exclude depending on their needs. For example, a user aiming to design an image processing pipeline is likely not interested in any functionality related to 3D meshes or curves. To prevent clutter in the UI and potential hogging of resources, they can decide not to include these modules. Examples of subjects that could have their own module in the future could be image segmentation, geometry, curves, machine learning, or videos.

6 METHODOLOGY

To gain an understanding of how well the bit-depth expansion (Chapter 3) and compression scheme (Chapter 4) perform, we use a number of different images of varying sizes (Figure 6.1[1]). The image contents range from simple gradients to highly textured regions to ensure we can test how the methods perform in different situations.

(a) `Gradient` (256 × 256)



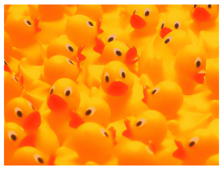
(b) `Peppers` (512 × 512)

(c) `Cat` (1024 × 1024)

(d) `Lighthouse` (768 × 512)

(e) `Plane` (768 × 512)

(f) `Bernoulliborg` (2000 × 1500)



(g) `Ducks` (960 × 720)

Figure 6.1: Images used to assess the performance of the bit-depth expansion method and compression scheme.

The bit-depth expansion and compression schemes are evaluated separately. For the bit-depth expansion method, we use images quantised using *k*-means to determine how well it can reconstruct the input image. It should be noted that the reconstruction performance is in part also influenced by the quality of the quantisation process. Given that *k*-means produces good results, however, this effect is limited.

We evaluate the bit-depth expansion method by testing it on images quantised to 3 and 4 bits respectively. While our method also works for high bit-depths, the differences in visual quality are difficult to assess there as they are typically not perceptible. However, showing that the method can perform well on these low-bit depth images allows us to draw conclusions on how well it performs on higher bit-depth images.

For the analysis of the compression scheme, we analyse two things. First, we assess the loss of visual quality resulting from the lossy

1 Note that the `ducks` image is used only for the colour bit-depth expansion.

compression scheme using a similarity metric. Next, we compute the compression ratios based on the compressed data:

$$CR = \frac{n_u}{n_c}, \tag{6.1}$$

where CR is the compression ratio, n_u is the number of bits of the uncompressed image and n_c is the number of bits for the compressed image. These results are then compared to JPEG compression to give a frame of reference. To do this, we compress the images using JPEG in such a way that the final file sizes (and therefore compression ratios) are as close together as possible. This is done using the `quality` parameter that JPEG compression provides. The visual quality of the decompressed image can then be compared between our method and JPEG compression. It is worth mentioning that the `quality` parameter is specified by an integer and ranges from 0 to 100. As such, the final file sizes cannot be matched perfectly.

6.1 SIMILARITY METRICS

There are numerous methods for measuring image similarity. The most popular ones include the mean-square error (MSE), peak-signal-to-noise ratio (PSNR) and structural similarity index (SSIM). The three metrics are all referred to as full-reference metrics, i.e. metrics requiring both an input image I and a reference image I_{ref}.

The mean-squared error is calculated as follows:

$$MSE(I, I_{ref}) = \frac{1}{m \cdot n} \sum_{\mathbf{p}} (I(\mathbf{p}) - I_{ref}(\mathbf{p}))^2. \tag{6.2}$$

It represents the average squared distance between two images. As the name suggests, MSE is an error metric, so the lower the value, the higher the similarity. However, its value can be difficult to interpret. This is where PSNR comes in: it uses the MSE value and gives it context by using a logarithmic transform and combining it with the maximum possible value a pixel can have:

$$PSNR(I, I_{ref}) = 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE(I, I_{ref})), \tag{6.3}$$

where MAX_I is the maximum possible value a pixel can have. For an image of k bits, $MAX_I = 2^k - 1$[2]. PSNR is expressed in decibels (dB), where higher values indicate a greater similarity.

Neither MSE nor PSNR takes into consideration aspects of the human visual system. Various methods have been proposed to improve on this including SMAPE [72], SSIM [77], S-CIELAB [35, 82], and HDR-VDP-2 [46]. Arguably the most used of these is SSIM, which is based on three components: luminance, contrast, and structure. Using these

2 In the case of floating-point images, $MAX_I = 1.0$.

three components, it attempts to take into account how the human visual system reacts to these changes. Despite SSIM aligning more with human perception compared to MSE and PSNR, there are still numerous issues with it [55]. To give an example, a slight rotation or translation of the image can produce extreme errors, despite the images being visually nearly equivalent. There are a number of variations on SSIM that attempt to improve its accuracy, such as MSSIM [41] and MS-SSIM [78]. However, these methods are still not without issues, so we opt to use a different metric instead.

6.1.1 *NVIDIA ꟻLIP*

ꟻLIP is a new visual similarity metric for images developed by NVIDIA [5]. Their method uses various aspects of the human visual system to build an error map that is representative of the differences perceived by humans. The primary purpose of ꟻLIP was to compare rendered images by flipping between them in place (hence the name). One fundamental aspect that sets ꟻLIP apart is its usage of viewing distance as part of the parameters. For example, a black and white chessboard viewed from some distance will appear grey to the observer. This is not captured by similarity metrics such as PSNR or SSIM, while ꟻLIP takes this into consideration by letting the user define the PPD (pixels per degree).

The ꟻLIP pipeline consists of two important parts: a colour pipeline and a feature pipeline. In the colour pipeline, they detect differences in colour. They first perform a spatial filtering step using filters based on the human contrast sensitivity functions. The colours are then converted to a perceptually uniform colour space before they are compared. This colour space conversion allows ꟻLIP to deal with phenomena such as the Hunt effect [34]. In the feature pipeline, feature differences between the two images are evaluated using edge and point detection. The resulting errors from the colour and feature pipelines are then combined into a single value.

As ꟻLIP outperforms all of the aforementioned metrics, this is also the metric that is used in this thesis. Fundamentally, ꟻLIP produces a per-pixel difference map instead of a single value such as MSE or PSNR. For a number of images, we provide the error map, but it is also important to have a quantitative value associated with this. In their paper, the authors of ꟻLIP specify that it is suitable for pooling. As such, we also provide the mean ꟻLIP error for the images. For the results, we used a PPD value of 33.5. This corresponds to viewing a 70 cm wide screen with a horizontal resolution of 1920 pixels from a distance of 70 cm. Despite the aforementioned issues with MSE and PSNR, we recognise that they are still widely used. As such, we will not only provide the mean ꟻLIP error but also the MSE and PSNR for each image comparison.

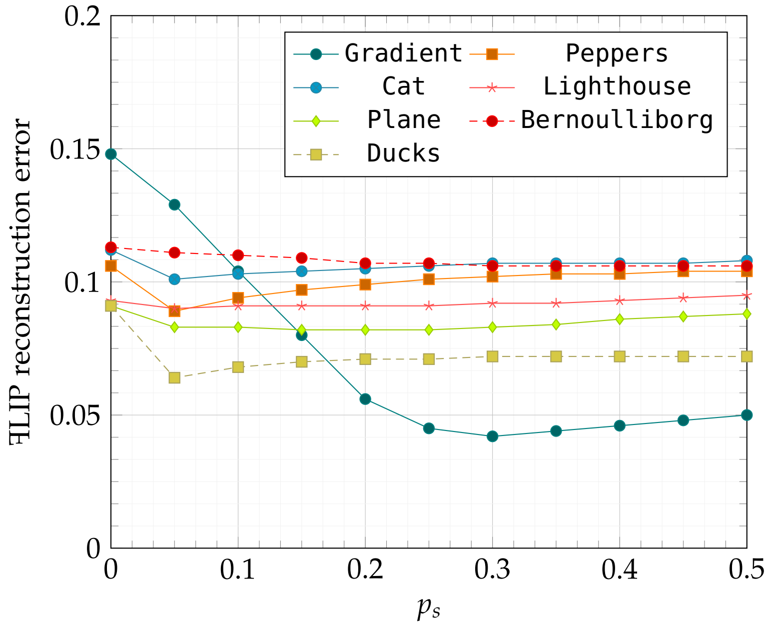


Figure 6.2: Impact of the filter size $s_G = p_s \cdot \max(m, n)$ as determined by p_s in the brightness correction step on the ꟻLIP reconstruction error. Images with large smooth regions tend to favour a larger p_s, whereas heavily textured images are reconstructed more effectively using a smaller p_s.

6.2 BRIGHTNESS CORRECTION

The brightness correction step has a user-controllable parameter p_s. However, it is not always desirable or feasible to give the user control over this, so we also provide a sensible default value for p_s. The impact of p_s on the reconstruction of various images can be seen in Figure 6.2. Here we see that heavily textured images such as the `cat` image favour lower values of p_s, while images containing larger regions or gradients favour higher values of p_s. It is clear that there is no single best value, and while it might be possible to estimate a good value for p_s based on image content, this would require significant extra computations. Histogram-based approaches are unlikely to provide a good estimate, as spatial pixel information is important. A better approach could be to look at the average or maximum component size of the component tree, but computing this component tree is an expensive operation [49]. As such, the results shown in Chapter 7 were obtained the default value $p_s = \frac{1}{8}$, which seems to provide a reasonable trade-off between contour removal and maintaining details in the shadows and highlights. However, if contour removal is the primary objective, then it is worth increasing p_s with a slight sacrifice to the reconstruction error in textured regions.

6.3 PERFORMANCE EVALUATION

While performance was an important consideration when designing the methods, it was not the main focus of this project. Nevertheless, we want to give some indication of how fast our methods run. As such, we also provide some basic performance measurements of both the bit-depth expansion algorithm and the compression scheme.

As the bit-depth expansion is well-suited for parallelization, we created a simple parallel implementation using OpenMP. For this, we follow a very simple strategy where two primary parts are parallelised: first, each thread is responsible for calculating the signed distance field of a single layer of the input image. Second, the reconstruction is parallelised by assigning each thread a portion of the pixels. It should be noted that a fully optimised version can be made to run significantly faster. In this case, the signed distance field calculation itself should be split up, since the distance transform algorithm allows for this. This would make better use of spatial locality. Additionally, a number of the functionalities that are currently performed by OpenCV, such as handling the matrix expressions and finding minima/maxima, can then be parallelised much more effectively. Finally, by doing this, all threads only need to be spawned once, instead of for each operation, which would reduce a significant amount of overhead. As such, this performance test should only be used to give an intuition on how it can perform; a well-optimised version would be significantly faster.

In that same vein, the brightness correction step has been disabled for the purposes of this parallelisation performance test. The current implementation relies on OpenCV implementations for Gaussian-blur and there are a significant number of improvements to be made here when it comes to (parallel) performance. To give a realistic idea of the parallel performance it can achieve, we omit this from the performance results (although they can still be found in the Appendix).

To this end, we also provide a brief evaluation of the parallel performance and the speed-up it achieves. The speed-up s is calculated as:

$$s(p) = \frac{T_{seq}}{T_{par}(p)}, \tag{6.4}$$

where T_{seq} is the execution time and $T_{par}(p)$ the execution time using p processors. The efficiency $e(p)$ is then determined using:

$$e(p) = \frac{T_{par}(p)}{p}. \tag{6.5}$$

All execution times were obtained by taking the median runtime over 10 runs using a Ryzen 5900x.

7 RESULTS

Below we provide the results and corresponding discussion for the bit-depth expansion method and compression scheme. In addition to this, we also evaluate its performance. This chapter is structured as follows. First, we show the results of the bit-depth expansion algorithm in Section 7.1. Next, we demonstrate the performance of the compression scheme and how it compares to JPEG compression in Section 7.2. Finally, we analyse the runtime performance of both the bit-depth expansion method and compression scheme in Section 7.3.

7.1 BIT-DEPTH EXPANSION

The results for our bit-depth expansion scheme on the `peppers` image can be seen in Figure 7.1. There are a few things to note here. First, a number of highlights are lost in the quantisation process. It can be seen in Figure 7.1c that these highlights are not fully recovered by the expansion process, which is also evident when inspecting the ꟻLIP error map. This shows, as explained in Section 3.4, that the majority of the errors lie in the shadows and the highlights. Part of this can be attributed to the quantisation process, as quantisation tends not to capture small but important colour regions in the shadows and highlights of the image. Given that our approach is not data-driven, it is unrealistic to expect it to reproduce these areas perfectly at such a low bit depth. However, for the shadows, the errors can be primarily attributed to the fact that our bit-depth expansion method at its core only increases brightness — something that the brightness correction step is unable to fully compensate for in these regions.

As we increase the bit-depth from 3-bit to 4-bit, we see that these issues are reduced. Where the initial quantised image clearly shows noise and contours, this is significantly reduced in the reconstructed image. It can be observed though that the method is relatively sensitive to noise in the quantised input, which is particularly visible in the dark pepper at the centre of the image. Whether this is for better or for worse is difficult to say. On one hand, it can produce these slightly spotty patches, but on the other hand, it does not cause any completely smooth/flat areas and gives the illusion of texture.

Looking at the `Lighthouse` image in Figure 7.2, we can see that even with very little colour information, our bit-depth expansion method can still reproduce a relatively convincing output image. In particular, the sky and clouds appear of higher quality compared to their quantised counterpart. Granted, as one zooms in further, it

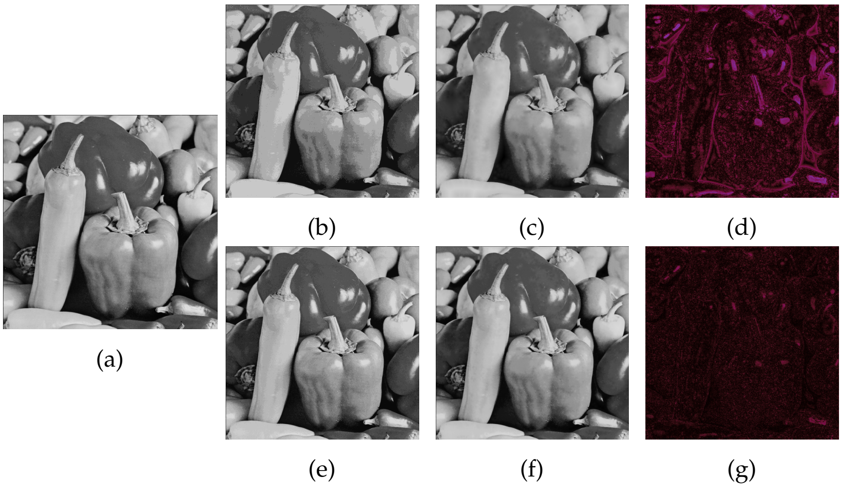


Figure 7.1: Bit-depth expansion results of the `Peppers` image. The majority of the reconstruction errors lie in the shadows and highlights. (a) Input image. (b) Quantised 3-bit image. (c) Reconstructed 3-bit to 8-bit. (d) ꟻLIP 3 to 8-bit reconstruction error. (e) Quantised 4-bit image. (f) Reconstructed 4-bit to 8-bit. (g) ꟻLIP 4 to 8-bit reconstruction error.

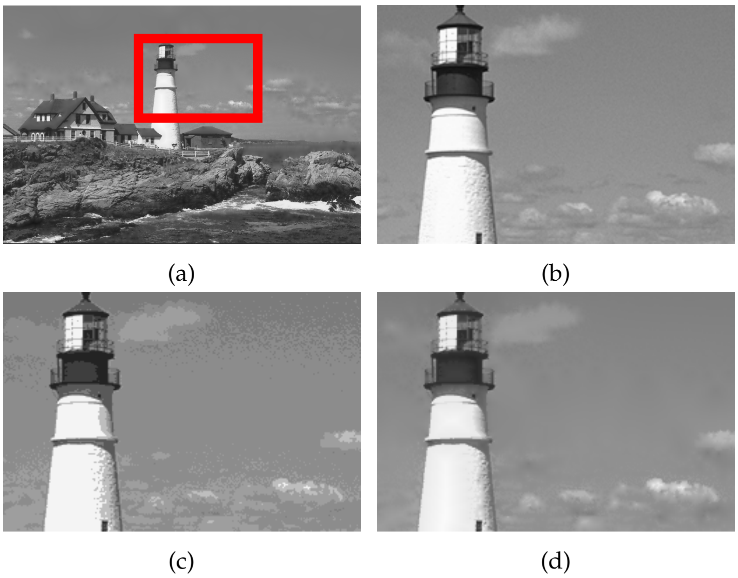


Figure 7.2: Reconstruction of the `Lighthouse` image. The visible contours and noise in the sky are significantly reduced, while the sharp edges are maintained. (a) Original 8-bit image. (b) Cropped 8-bit image. (c) Quantised 3-bit image. (d) Reconstructed 8-bit image.

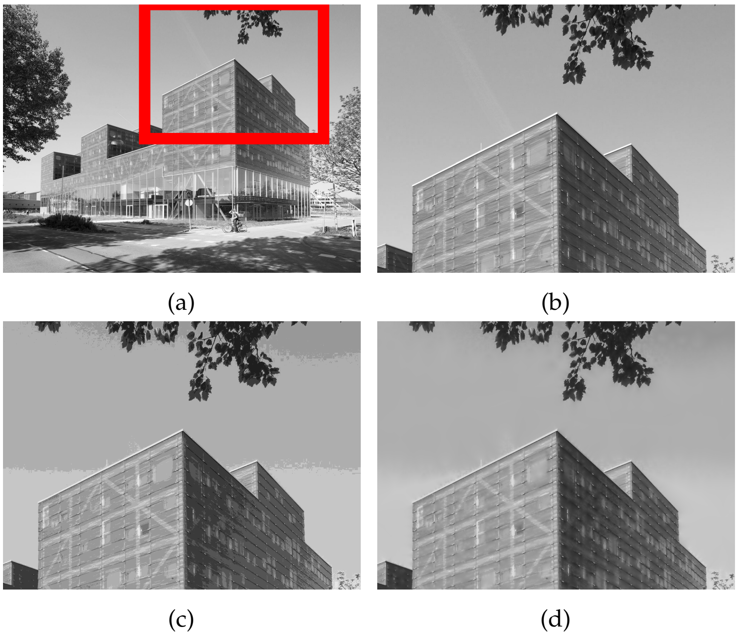


Figure 7.3: Reconstruction of the `Bernoulliborg` image reducing the visibility of contours. (a) Original 8-bit image. (b) Cropped 8-bit image. (c) Quantised 3-bit image. (d) Reconstructed 8-bit image.

is clear that details are missing, but this is as expected given the small amount of information in the quantised image. Generally, the reconstructed image is significantly more visually pleasing that the quantised image.

The effects of quantisation such as contouring are particularly visible in areas that contained some sort of gradient. One common example of these areas are skies in photos. This could already be seen in the `Lighthouse` image, but it is even more apparent in the `Bernoulliborg` image in Figure 7.3. The quantised image shows very clear contours in the sky, whereas this is significantly reduced in the reconstructed image. However, it does not fully eliminate the gradient, which is due to the brightness correction step. In the brightness correction step, a trade-off has to be made when choosing the filter size s_G (Figure 6.2). As the sky in this image is relatively large, the filter is too small for this particular region of the image. As such, contours are still present in the low-frequencies of the image, causing the corrected reconstructed image to introduce some of these contours as well. A comparison of this can be seen for the `Bernoulliborg` image in Figure 7.4 where the reconstructed image prior to brightness correction shows a significantly smoother gradient. While it is possible to opt for a larger filter size, this means that other regions are affected negatively. In more natural images this is undesirable, so despite it introducing some degree of contours, the current filter size offers a reasonable trade-off between smoothing larger regions and preserving intensities in detailed regions. However, we recognise that this can be improved

(a) (b)

Figure 7.4: Brightness correction step introducing contours as a result of the filter size as determined by $p_s = \frac{1}{8}$ being too small for the large contour regions in this image. (a) Reconstruction before brightness correction. (b) Reconstruction after brightness correction.

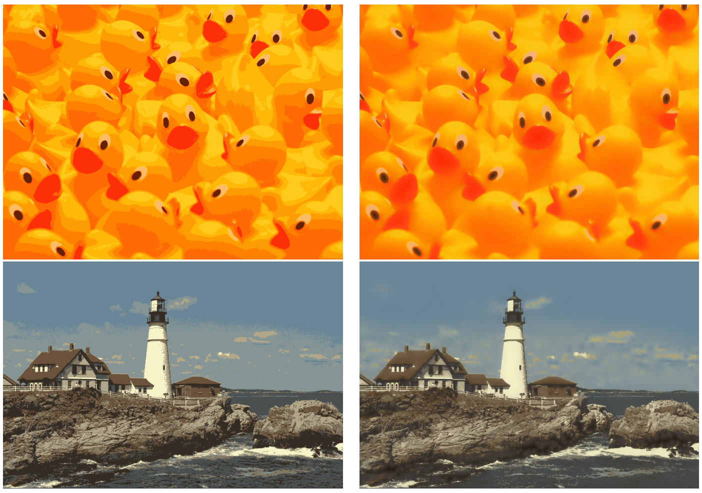
further, so we have outlined a short approach on how to do this in Chapter 9.

The visual quality differences are summarised in Table 7.1. From the above images, it was already clear that the bit-depth expansion process improves visual quality over the quantised image, and the results here confirm this as it improves on all three metrics. The only exception to this is that for some images the PSNR is reduced slightly in the reconstructed image. However, even in these cases, the values are nearly equal.

Table 7.1: Differences in visual quality between the 3-bit quantised input image and the reconstructed 8-bit image. In nearly all cases, the three similarity metrics show improvements in visual quality when the image is reconstructed using our bit-depth expansion method.

	Quantised			Reconstructed		
Image	**MSE**	**PSNR**	**ꟻLIP**	**MSE**	**PSNR**	**ꟻLIP**
`Gradient`	71.518	29.521	0.148	22.095	34.617	0.092
`Peppers`	56.262	30.529	0.106	42.254	31.089	0.095
`Cat`	61.901	29.961	0.112	49.330	29.745	0.104
`Lighthouse`	45.380	31.234	0.093	40.017	31.266	0.091
`Plane`	47.687	31.118	0.091	32.888	32.276	0.082
`Bernoulliborg`	60.255	30.125	0.113	48.035	30.060	0.109

Overall, the expansion scheme produces good results, even for images with low bit-depth. It does not overly blur sharp edges, except in cases where the quantised image contains too little information and the sharp edges are exactly one grey-level step apart. In these cases, it is impossible to distinguish between what needs to be a sharp edge and a smooth edge regardless, so this is expected. There are some issues with the shadows and the highlights, in addition to the brightness correction step not being perfect, which still leaves room for further improvements.



(a) (b)

Figure 7.5: Our bit-depth expansion method applied on a quantised colour image consisting of 8 colours on the `Duck` and `Lighthouse` images respectively. Our method does not introduce false contours and maintains the majority of the sharp edges, despite limited information being available in the input image. (a) Quantised colour image consisting of 8 colours. (b) Reconstructed 8-bit colour image.

7.1.1 *Colour Images*

For the sake of completeness, we provide two examples to show our method can also be applied to colour images. This was done by applying the bit-depth expansion method individually on each channel in the YC_bC_r colour space. The results for this can be seen in Figure 7.5. Generally, the results are good, as there seems to be no appearance of false contours. While the image appears slightly blurred, this can be attributed to the lack of information in the input image. This lack of information is greater in colour images, because a colour image quantised to 8 colours loses significantly more information compared to a greyscale image quantised to 8 greyscale values. It should also be noted that, unlike BitNet [13] or the method proposed by Cheng et al. [19], our method does not remove any false contouring.

However, when using more colours in the input image, it does not smooth all the desired contours. The reason for this is that the method only blends between consecutive layers. As such, as soon as a gradient area misses one of the intermediate colours, a contour will appear. It should be noted that the colours were quantised directly, instead of on a per-channel basis. A per-channel quantisation produces better results and tends not to have these issues, but that also increases storage costs. In general, a well-quantised image should not have this issue, but this is not always controllable. This particular issue, along with a possible solution, is further explained in the Chapter 9.

7.2 COMPRESSION

Our compression uses quantisation as the core mechanism to achieve good compression ratios. Given that the degree of quantisation is configurable by the user, we provide the results for both 3-bit and 4-bit quantisation here. In Table 7.2, we can see the performance for the 3-bit compression scheme. The algorithm performs relatively well, generally achieving compression ratios of around 5. The exception to this is the gradient image, which compresses extremely well. On the other hand, heavily textured images, such as the `Cat` image, have lower compression ratios. This is as expected since a higher variance in pixel intensity typically reduces the spatial correlations that many compression algorithms depend on. Image size does not seem to have a significant impact on the compression ratio, given that the `Bernoulliborg` achieves a comparable compression ratio compared to smaller images.

Table 7.2: Compression performance of our method using 3-bit quantisation. The compression scheme achieves good compression ratios, but the similarity metrics show a notable loss in visual quality.

Image		**Quality**			**Compression**	
Name	**Size (KB)**	**MSE**	**PSNR**	**ꟻLIP**	**Size (KB)**	**CR**
`Gradient`	65.536	0.8	48.5	0.025	0.878	74.64
`Peppers`	262.144	30.4	31.1	0.082	52.085	5.03
`Cat`	1048.576	32.9	31.5	0.082	244.253	4.29
`Lighthouse`	393.216	25.8	32.1	0.072	84.923	4.63
`Plane`	393.216	22.0	32.2	0.060	69.709	5.64
`Bernoulliborg`	3000	29.9	31.7	0.079	554.338	5.41

For the JPEG comparison, we tried to match the final file size of 3-bit compression as best as possible. As such, the results for JPEG compression seen in Table 7.3 have a separate quality parameter. First to note is that for similar file sizes, JPEG produces higher-quality images. The exception to this is the gradient image, which the JPEG compression algorithm could not come close to. Overall, JPEG performs much better, since it is always able to use a *quality* value of 80+ when trying to achieve similar compression ratios. As such, the corresponding ꟻLIP error map is typically much less visible than that of our 3-bit compression scheme (Figure 7.6).

Despite JPEG performing considerably better, our compression scheme still achieves PSNR values over 30 and the average ꟻLIP error never goes above 0.1. As is evident from the ꟻLIP error map, however, the compression scheme still struggles with the shadows and highlights due to the bit-depth expansion process. Additionally, we can also clearly see the loss of detail during the quantisation process.

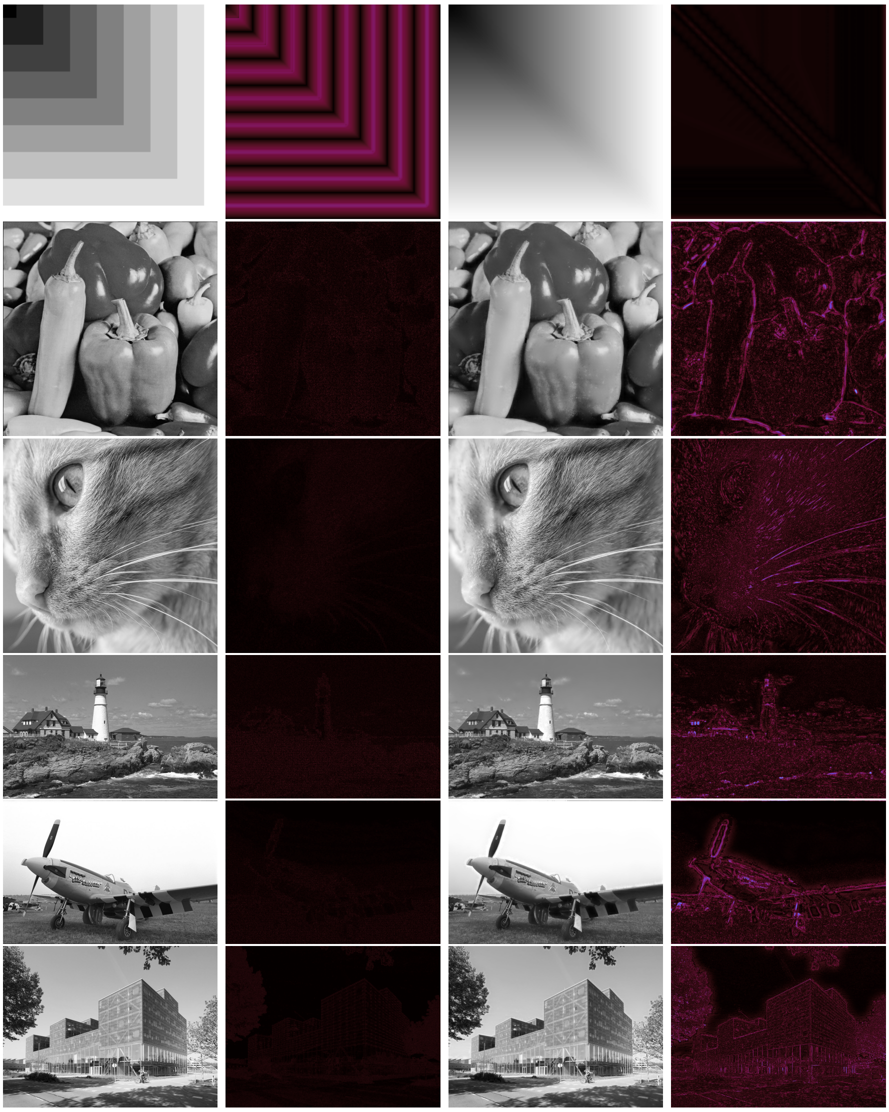
Table 7.3: Compression performance of JPEG. The quality parameter was chosen in such a way that the file size is closest to that produced by our 3-bit compression scheme. Its compression ratios are similar to that of 3-bit compression (by construction), but JPEG compression typically scores better on the similarity metrics compared to our 3-bit compression.

Image		Quality			Compression	
Name	**Quality**	**MSE**	**PSNR**	**ꟻLIP**	**Size (KB)**	**CR**
`Gradient`	0	83.8	28.9	0.156	1.144	57.29
`Peppers`	86	11.6	37.5	0.038	53.192	4.93
`Cat`	82	3.3	42.9	0.024	243.971	4.30
`Lighthouse`	85	11.7	37.4	0.038	82.753	4.75
`Plane`	90	4.4	41.7	0.024	70.275	5.60
`Bernoulliborg`	82	5.8	40.5	0.025	547.923	5.48

If the quantisation process is increased to 4 bits, the visual quality of the decompressed image is significantly better. All three metrics improve, while the final image size does not increase substantially. Surprising is that going from 3-bit to 4-bit has no effect on the decompressed image for the gradient image. This most likely has to do with the fact that this image can already be compressed extremely efficiently using 3 bits. Overall, the slight increase in file size is worth the trade-off considering the increase in visual quality, since the loss of details due to the quantisation process is notably reduced. A comparison between 3-bit and 4-bit compression can be found in Appendix A.2.

Table 7.4: Compression performance of our method using 4-bit quantisation. There is a slight reduction in the compression ratios compared to 3-bit quantisation, but the visual quality of the reconstructed images is improved significantly.

Image		Quality			Compression	
Name	**Size (KB)**	**MSE**	**PSNR**	**ꟻLIP**	**Size (KB)**	**CR**
`Gradient`	65.536	0.8	48.5	0.025	0.878	74.64
`Peppers`	262.144	17.5	34.7	0.062	62.198	4.21
`Cat`	1048.576	18.7	35.1	0.064	285.572	3.67
`Lighthouse`	393.216	16.3	35.5	0.057	94.850	4.15
`Plane`	393.216	11.0	37.0	0.043	80.576	4.88
`Bernoulliborg`	3000	13.3	36.7	0.054	679.378	4.42



(a) (b) (c) (d)

Figure 7.6: Compression comparison between JPEG and our method (3-bits) as per the results in Table 7.3 and Table 7.2 respectively. Except for the `Gradient` image, the decompressed images produced by JPEG compression are of better visual quality than those of our 3-bit compression scheme for similar compression ratios. (a) JPEG compression result. (b) JPEG ꟻLIP error. (c) 3-bit compression result. (d) 3-bit ꟻLIP error.

7.3 PERFORMANCE

Due to its excellent suitability for parallelization, we also provide a brief comparison between the serial bit-depth expansion scheme and the parallel implementation. The main results can be seen in Table 7.5. It is clear that, despite the issues mentioned in Section 6.3, the parallelisation was still beneficial. Depending on the image size, the speed-up ranges from 3.4 to 5.9. The results also show that the speed-up and efficiency are reduced for images containing more high-frequency details. This can most likely be attributed to the fact that the performance of the distance transform algorithm is to a certain degree dependent on the image contents [25], resulting in worse load-balancing compared to images with fewer details. However, this would be less noticeable in a well-optimised parallelization. Note that this is the version without the brightness correction step; the results for the algorithm with the (unoptimised) brightness correction step enabled can be found in Appendix A.2. Overall, the parallelisation was, despite its simplicity, effective.

Table 7.5: Parallel execution times for bit-depth expansion in 3-bit input images. The parallelization strategy achieves speed-up and efficiency for $p = 8$ processors, despite its simplicity and room for improvement.

Image	$p = 1$ **(msec)**	$p = 8$ **(msec)**	**Speedup**	**Efficiency (%)**
`Gradient`	10.69	1.82	5.87	73.42
`Peppers`	56.48	13.24	4.27	53.32
`Cat`	142.37	41.86	3.40	42.51
`Lighthouse`	76.07	14.34	5.30	66.31
`Plane`	73.06	13.63	5.36	67.00
`Bernoulliborg`	345.72	80.41	4.30	53.74

The performance results for the compression process of the various images can be seen in Table 7.6. It should be noted that these results were obtained by using 8 threads and that the brightness step *was* enabled for these results. From these timing results, we can see that the method provides excellent results, despite the fact that a signed distance field has to be calculated for each layer. In the compression step, the primary bottleneck is the encoding step as performed by the *zlib* library. The optimisation of the k-means quantisation really shows its benefits here, as an unoptimised version would be significantly slower for larger images. The performance of the compression step could be improved even further, considering that this part is not parallelised. However, looking at where most of the time is spent, this would yield diminishing returns.

Table 7.6: Execution time in seconds of the compression scheme (3-bits) on various images for $p = 1$. Compression runs extremely fast, whereas the decompression step is slower. For the decompression step, most time is spent on the bit-depth expansion part. The overall execution time is primarily determined by the image size

	Compression (sec)			Decompression (msec)		
Image	**Reduction**	**Encoding**	**Total**	**Decoding**	**Expansion**	**Total**
Gradient	0.16	0.31	0.47	0.10	1.92	2.02
Peppers	0.60	16.43	17.03	0.98	21.60	22.58
Cat	3.35	33.51	36.86	2.61	113.60	116.21
Lighthouse	1.37	14.32	15.69	0.94	32.27	33.21
Plane	0.78	14.75	15.53	0.88	30.28	31.16
Bernoulliborg	10.86	83.47	94.33	6.84	473.58	480.42

The decompression step is slower in comparison to the compression step but still provides good results. Images up to 2000×1500 can be decompressed in less than half a second. Here, the bottleneck is clearly the bit-depth expansion step. This can mostly be attributed to two things. First, a signed distance field has to be calculated for each layer of the input image. Despite that, the method does not scale extremely poorly as the bit-depth increases. To give an indication, even for 256 grey levels in the input image, the bit-depth expansion part only takes 2.1 seconds. Second, the brightness correction step is relatively slow, as it fully depends on the OpenCV implementation. The fact that the method still performs well as the number of grey levels in the input image increase seems to suggest that this step is nearly as detrimental to performance as the signed distance field calculations. Despite all this, the bit-depth expansion algorithm is still relatively fast. For images of size 768×512, the decompression takes around 33 milliseconds, which is equivalent to roughly 30 fps. Given its potential to be optimised further, we can conclude that its performance is very promising.

8 CONCLUSIONS

In this thesis, we proposed a method for bit-depth expansion and used it to construct a lossy compression scheme. The bit-depth expansion method makes use of signed distance field blending to create a smoother version of the input image. Overall, it performed well as it is able to improve the visual quality of quantised images by a considerable amount. However, there are still various improvements to be made to the scheme, as it tends to produce the majority of the errors in the shadow and highlight regions of the image. The method is relatively expensive, given that it requires a signed distance field to be calculated for each grey level of the input image. Fortunately, the algorithm is trivially parallelised, which we demonstrated provided a good speedup. Using this parallel approach, we showed that the majority of images can still be reconstructed in a short amount of time.

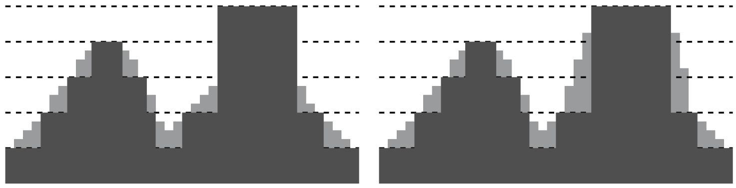
The compression scheme makes use of quantisation in the compression step. To improve the visual quality of the decompressed image while maintaining good compression ratios, we extract the low-frequencies into a low-resolution image. The residual and low-resolution are then encoded using the DEFLATE algorithm. In the decompression step, these images are decoded and bit-depth expansion is applied to the residual image. This is then combined with the low-frequencies to obtain the decompressed image. The compression scheme can effectively compress images, achieving compression ratios of around 5. Still, it still lags behind existing lossy compression methods such as JPEG compression.

Ultimately, both the bit-depth expansion method and the compression scheme seem to be promising. However, before these methods are viable to use, the suggestions outlined in Chapter 9 should ideally be implemented and experimented with. This might improve its reconstruction results and runtime performance to the degree that this method can then be viably used in the real world.

9 FUTURE WORK

There are various improvements that can still be made to both the expansion method and the compression scheme. A number of these were already briefly commented on before, but we outline them below in further detail.

- As mentioned in Chapter 3, the bit-depth method only increases the pixel intensity by construction. To combat this, we introduced a brightness correction step using a low-pass filter. However, this correction step suffers from two main issues: it requires a trade-off for the filter size and small regions that require this correction are typically not captured by this filter approach. There are two approaches that could improve this. The first approach focuses on the brightness correction step. Here an adaptive filter [67] could be used to change the degree to which particular regions are filtered. This could possibly prevent the issues shown in Figure 7.4, where the brightness correction step re-introduced some contouring effects. The second, more complicated approach would be to change the blending method such that existing layers are not necessarily interpolated. The result of this would be a method that does not always increase pixel intensity but can also lower it. This would however require a different approach for interpolating the distance layers.
- In certain cases, the bit-depth expansion method does not smooth all contours it is expected to smooth. This is typically the result of a suboptimal quantisation process, but this is not always controllable by the user. The reason that this occurs is that it only blends two consecutive layers, which means that if there is another layer in between (even though it is not present in that particular region of the image), the contour will remain. On a global image scale, this is desirable for maintaining edges. However, locally it might be desirable to still blend between layers even if they are consecutive. The issue is showcased in Figure 9.1. A fix for this would be to do the blending based on the Max-Tree [49]. In these cases, each component is blended with its parent component (or vice versa). As the Max-Tree is flattened during its construction, this means that locally two layers can still blend even if they are not consecutive. However, whether this will actually improve the visual quality or blur the image and edges too much is difficult to say.



(a) (b)

Figure 9.1: Different approaches to layer blending, (**a**) Current blending method which only blends between consecutive layers, (**b**) Blending method that blends between any two components.

- The current bit-depth expansion requires a signed distance field to be calculated for every grey level of the input image. While we already use an algorithm that does this in linear time, this is still an expensive operation to do. Additionally, its performance is highly dependent on the bit-depth of the input image, as a higher bit-depth requires more signed distance fields to be calculated. Improving the efficiency of this is not a trivial matter, but we try to provide some pointers here.

  First, the algorithm we use proposed by Meijster et al. [47] consists of two phases. The vertical phase is relatively independent in the sense that it does not require any neighbour information, besides its top and bottom neighbours. As such, this phase can be altered slightly by calculating for each pixel **p** the distance to the nearest pixel **q** such that $I(\mathbf{p}) < I(\mathbf{q})$. For a given column, this can be done efficiently by constructing two 1D max trees: one constructed by traversing the column top-down and one constructed by traversing the column bottom-up. The distances of the pixels are simply the y-differences between the pixel and its parent component. The final vertical scan result can then be obtained by taking the minimum of these distances from the two Max-Trees. It should be noted that one still requires additional index information for the second phase, as just the distances are not enough in this case. Currently, we have not found a way to alter the second phase in a similar fashion.

  An alternative approach could be to use Max-Trees. As we can calculate the greyscale value directly using linear interpolation, we require only two distance values per pixel. As such, it should not be necessary to calculate the distance for every layer. One possible way to do this using max trees is by storing the contours of each component and then calculating the distance for a given pixel by finding the closest contour pixel of its parent component. This would most likely be reasonably fast, provided that the Max-Tree can be constructed efficiently using e.g. [49] and that the contours can also be extracted efficiently. However, the efficiency would still be very much dependent on the image contents. It has the advantage that it becomes more efficient as the bit-depth of the input image increases. This is because the average component size will be smaller in these cases, so each pixel has to search a smaller set of pixels to find the closest distance. This method would also work well with the point mentioned before regarding Max-Trees.

- The bit-depth expansion method is extremely suitable for parallelisation. Essentially, every pixel can be evaluated independently from other pixels (depending on the distance transform algorithm used). This means that it would not only be possible to efficiently parallelise this on the CPU (as implemented already) but can also be parallelised on the GPU. There are existing methods for evaluating distance fields on the GPU [6] and there are also methods available for Max-Trees [9] in case the aforementioned approach is used.

- There is room for improvement in the quantisation process of the compression scheme. While *k*-means generally produces good results, it does not take into consideration how are method is used for reconstruction. Ideally, the quantisation method removes those layers in such a way that the bit-depth expansion algorithm can reconstruct them as closely as possible. Ideally, this method should then also consider the fact that the expansion method only increases pixel intensity, which might then negate the need for the brightness-correction step.

- NITRO still has room for numerous features and improvements. Two features that would be nice to have are code generation and node procedures. With code generation, it would be possible to generate the C++ of the current node graph. This allows programmers to include their custom-built pipeline into other programs. As such, NITRO can then be used for generating a quick prototype/skeleton of the pipeline, which can then be optimised and extended manually using the generated code. On the other hand, node procedures would allow the user to group a number of nodes into a single procedure node. The advantage of this is that custom routines can then be easily re-used without cluttering the node graph view. Other improvements include support for GPU processing, additional image-processing nodes, extra modules with functionalities for e.g. image segmentation, an automatic build pipeline, improved dock widgets and the inclusion of unit tests.

- A large number of existing lossy compression algorithms rely on quantisation. It would be interesting to see how the integration of our bit-depth expansion method would improve the visual quality of the decompressed image.

APPENDIX

A

A.1 NITRO

A.1.1 *Nodes*

Source Code A.1: Command pattern implementation for a node that calculates the discrete cosine transform of a greyscale image.

```
void nitro::DCTOperator::execute(NodePorts &ports) {
    if(!ports.allInputsPresent()) {
        return;
    }
    // Get the input data
    cv::Mat inputImg = *ports.inGetAs<GrayImageData>("Image");
    int inverse = ports.getOption("Inverse");
    // Evaluate
    cv::Mat result;
    cv::dct(inputImg, result, inverse);
    // Store the result
    ports.output<GrayImageData>("Image", result);
}
```

Source Code A.2: Construction of a node using the builder pattern.

```
NitroNodeBuilder builder("DCT", "dct", category);
return builder.
       withOperator(std::make_unique<DCTOperator>())->
       withIcon("frequency.png")->
       withNodeColor({255,0,0})->
       withInputPort<GrayImageData>("Image")->
       withCheckBox("Inverse", false)->
       withOutputPort<GrayImageData>("Image")->
       build();
```

A.1.2 *Data Types*

Source Code A.3: Header file for an integer data type.

```
#pragma once

#include <utility>

#include "QtNodes/NodeData"
#include "flexibledata.hpp"

namespace nitro {
    class IntegerData : public FlexibleData<int, IntegerData> {
    public:
        IntegerData();

        explicit IntegerData(int value);

        static QString id() {
            return id_;
        }

        static void registerConversions();

        [[nodiscard]] QString getDescription() const override;

    private:
        inline static const QString name_ = "Integer";
        inline static const QString id_ = "Integer";
        inline static const QColor baseColor_ = {89, 140, 92};
    };
} // nitro
```

Source Code A.4: Source file for an integer data type.

```
#include "nodes/datatypes/integerdata.hpp"
#include "nodes/datatypes/decimaldata.hpp"

namespace nitro {
    IntegerData::IntegerData()
    : FlexibleData<int, IntegerData>(0, id_, name_, baseColor_) {
        // By default, always allow conversions from doubles
        allowConversionFrom(DecimalData::id());
    }

    IntegerData::IntegerData(int value)
    : FlexibleData<int, IntegerData>(value, id_, name_, baseColor_) {
        // By default, always allow conversions from doubles
        allowConversionFrom(DecimalData::id());
    }

    QString IntegerData::getDescription() const {
        return QString::number(data());
    }

    void IntegerData::registerConversions() {

        // Every type needs a "conversion" to itself
        IntegerData::registerConversionFrom<IntegerData>(
                [](const std::shared_ptr<QtNodes::NodeData> &nd) {
                    auto d = std::static_pointer_cast<IntegerData>(nd);
                    return d->data();
                });

        IntegerData::registerConversionFrom<DecimalData>(
                [](const std::shared_ptr<QtNodes::NodeData> &nd) {
                    auto d = std::static_pointer_cast<DecimalData>(nd);
                    return int(std::round(d->data()));
                });
    }
} // nitro
```

A.2 RESULTS

A.2.1 *4-bit Compression*

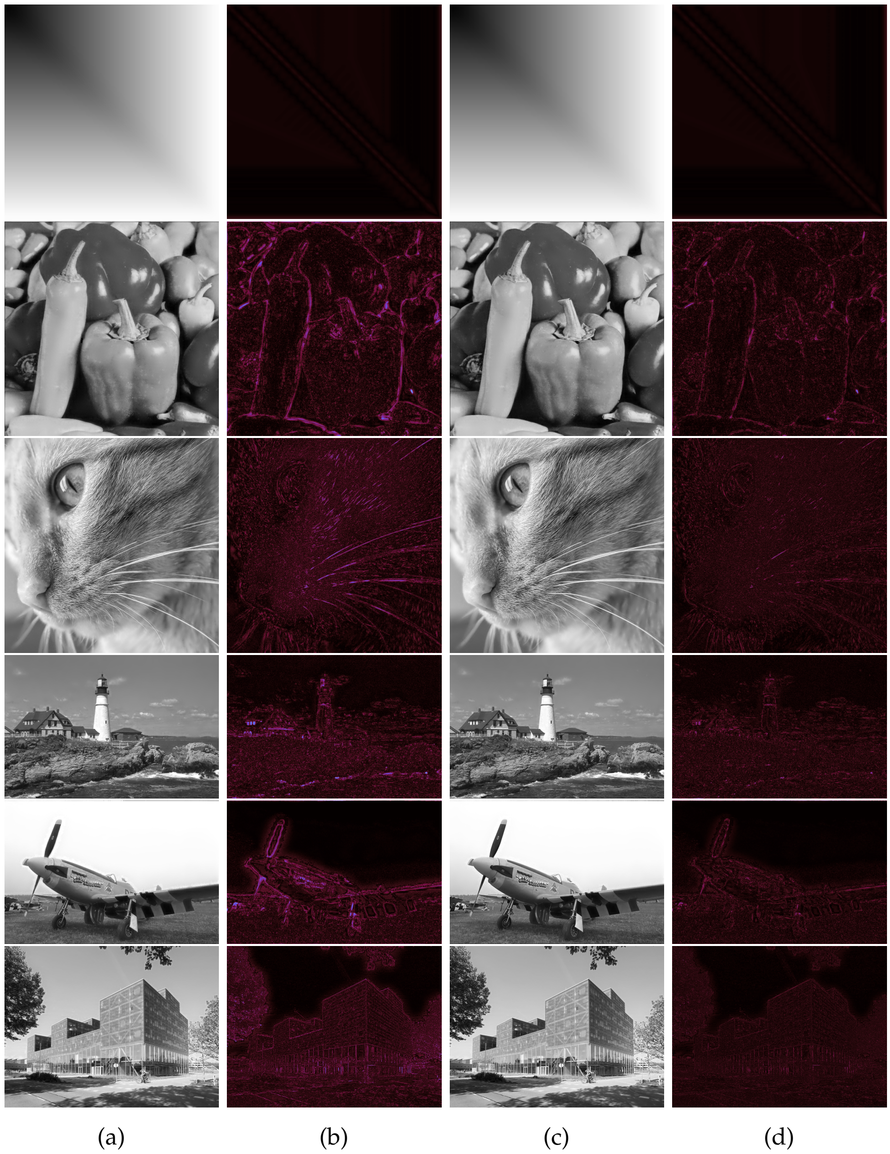


Figure A.1: Compression comparison between 3-bit and 4-bit compression of our method as per the results in Table 7.2 and Table 7.4 respectively. While there is a slight decrease in the compression ratios, the visual quality of the decompressed images is improved considerably. (a) 3-bit compression result. (b) 3-bit ꟻLIP error. (c) 4-bit compression result. (d) 4-bit ꟻLIP error.

A.2.2 *Performance*

Table A.1: Parallel execution times for bit-depth expansion in 3-bit input images (with brightness correction enabled). Compared to Table 7.5, the speed-up and efficiency values are lower due to the (inefficient) Gaussian filter taking up a large portion of the execution time.

Image	$p = 1$ **(sec)**	$p = 8$ **(sec)**	**Speedup**	**Efficiency (%)**
`Gradient`	11.82	2.84	4.16	52.02
`Peppers`	66.20	21.23	3.12	38.98
`Cat`	201.05	128.52	1.56	19.55
`Lighthouse`	87.81	37.08	2.37	29.60
`Plane`	86.38	28.36	3.05	38.07
`Bernoulliborg`	690.10	431.68	1.60	19.98

[1] Edward Adelson, Charles Anderson, James Bergen, Peter Burt and Joan Ogden. 'Pyramid Methods in Image Processing'. In: *RCA Eng.* 29 (Nov. 1983).

[2] Nasir Ahmed, T_ Natarajan and Kamisetty R Rao. 'Discrete Cosine Transform'. In: *IEEE Transactions on Computers* C-23.1 (1974), pp. 90–93.

[3] Jyrki Alakuijala, Ruud Van Asseldonk, Sami Boukortt, Martin Bruse, Iulia-Maria Comșa, Moritz Firsching, Thomas Fischbacher, Evgenii Kliuchnikov, Sebastian Gomez, Robert Obryk et al. 'JPEG XL next-generation image compression architecture and coding tools'. In: *Applications of Digital Image Processing XLII*. Vol. 11137. SPIE. 2019, pp. 112–124.

[4] Daniel Aloise, Amit Deshpande, Pierre Hansen and Preyas Popat. 'NP-hardness of Euclidean sum-of-squares clustering'. In: *Machine learning* 75 (2009), pp. 245–248.

[5] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström and Mark D. Fairchild. 'FLIP: A Difference Evaluator for Alternating Images'. In: *Proc. ACM Comput. Graph. Interact. Tech.* 3.2 (2020).

[6] Francisco de Assis Zampirolli and Leonardo Filipe. 'A Fast CUDA-Based Implementation for the Euclidean Distance Transform'. In: *2017 International Conference on High Performance Computing & Simulation (HPCS)*. 2017, pp. 815–818.

[7] Emre Başeski. '16-Bit to 8-Bit Conversion in Remote Sensing Images by Using Image Content'. In: *2019 9th International Conference on Recent Advances in Space Technologies (RAST)*. 2019, pp. 413–417.

[8] Sitaram Bhagavathy, Joan Llach and Jiefu Zhai. 'Multiscale Probabilistic Dithering for Suppressing Contour Artifacts in Digital Images'. In: *IEEE Transactions on Image Processing* 18.9 (2009), pp. 1936–1945.

[9] Nicolas Blin, Edwin Carlinet, Florian Lemaitre, Lionel Lacassagne and Thierry Géraud. 'Max-Tree Computation on GPUs'. In: *IEEE Transactions on Parallel and Distributed Systems* 33.12 (2022), pp. 3520–3531.

[10] Thomas Boutell. *PNG (Portable Network Graphics) Specification Version 1.0*. RFC 2083. Mar. 1997.

[11] Joachim M. Buhmann, Dieter W. Fellner, Marcus Held, Jens Ketterer and Jan Puzicha. 'Dithered Color Quantization'. In: *Computer Graphics Forum*. Vol. 17. 3. Wiley Online Library. 1998, pp. 219–231.

[12] Peter J. Burt and Edward H. Adelson. 'The Laplacian Pyramid as a Compact Image Code'. In: *Readings in Computer Vision*. Ed. by Martin A. Fischler and Oscar Firschein. San Francisco (CA): Morgan Kaufmann, 1987, pp. 671–679.

[13] Junyoung Byun, Kyujin Shim and Changick Kim. 'BitNet: Learning-based bit-depth expansion'. In: *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part II 14*. Springer. 2019, pp. 67–82.

[14] 'CIE Recommendations on Uniform Color Spaces, Color-Difference Equations, and Metric Color Terms'. In: *Color Research & Application* 2.1 (1977), pp. 5–6.

[15] M. Emre Celebi. 'Improving the performance of k-means for color quantization'. In: *Image and Vision Computing* 29.4 (2011), pp. 260–271.

[16] Mehmet Celenk. 'A color clustering technique for image segmentation'. In: *Computer Vision, Graphics, and Image Processing* 52.2 (1990), pp. 145–170.

[17] Jyh-Shan Chang, J.-H.J. Lin and Tzi-Dar Chiueh. 'Color image vector quantization using binary tree structured self-organizing feature maps'. In: *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*. Vol. 2. 1998, 1428–1432 vol.2.

[18] Cheuk-Hong Cheng, Oscar C. Au, Ngai-Man Cheung, Chun-Hung Liu and Ka-Yue Yip. 'Low color bit-depth image enhancement by contour-region dithering'. In: *2009 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing* (2009), pp. 666–670.

[19] Cheuk-Hong Cheng, Oscar C. Au, Chun-Hung Liu and Ka-Yue Yip. 'Bit-depth expansion by contour region reconstruction'. In: *2009 IEEE International Symposium on Circuits and Systems*. 2009, pp. 944–947.

[20] Daniel Cohen-Or, David Levin and Amira Solomovici. 'Contour blending using warp-guided distance field interpolation'. In: *Proceedings of Seventh Annual IEEE Visualization'96*. IEEE. 1996, pp. 165–172.

[21] Daniel Cohen-Or, Amira Solomovic and David Levin. 'Three-Dimensional Distance Field Metamorphosis'. In: *ACM Trans. Graph.* 17.2 (1998), 116–141.

[22] Per-Erik Danielsson. 'Euclidean distance mapping'. In: *Computer Graphics and image processing* 14.3 (1980), pp. 227–248.

[23] Peter Deutsch and Jean-Loup Gailly. *Zlib compressed data format specification version 3.3*. Tech. rep. 1996.

[24] Juan P. D'Amato. 'FitDepth: fast and lite 16-bit depth image compression algorithm'. In: *EURASIP Journal on Image and Video Processing* 2023.1 (2023), p. 5.

[25] Ricardo Fabbri, Luciano Da F. Costa, Julio C. Torelli and Odemir M. Bruno. '2D Euclidean Distance Transform Algorithms: A Comparative Survey'. In: *ACM Comput. Surv.* 40.1 (2008).

[26] Frederick N Fritsch and Ralph E Carlson. 'Monotone Piecewise Cubic Interpolation'. In: *SIAM Journal on Numerical Analysis* 17.2 (1980), pp. 238–246.

[27] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley professional computing series. Addison-Wesley, 1995.

[28] Michael Gervautz and Werner Purgathofer. 'A Simple Method for Color Quantization: Octree Quantization'. In: *New Trends in Computer Graphics*. Ed. by Nadia Magnenat-Thalmann and Daniel Thalmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 219–231.

[29] Apoorv Gupta, Aman Bansal and Vidhi Khanduja. 'Modern lossless compression techniques: Review, comparison and analysis'. In: *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. IEEE. 2017, pp. 1–8.

[30] Miska Hannuksela, Jani Lainema and Vinod Malamal Vadakital. 'The High Efficiency Image File Format Standard [Standards in a Nutshell]'. In: *Signal Processing Magazine, IEEE* 32 (July 2015), pp. 150–156.

[31] Paul Heckbert. 'Color Image Quantization for Frame Buffer Display'. In: *Proceedings of the 9th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '82. Boston, Massachusetts, USA: Association for Computing Machinery, 1982, 297–307.

[32] Gabor T Herman, Jingsheng Zheng and Carolyn A Bucholtz. 'Shape-based interpolation'. In: *IEEE Computer Graphics and Applications* 12.3 (1992), pp. 69–79.

[33] David A. Huffman. 'A Method for the Construction of Minimum-Redundancy Codes'. In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101.

[34] Robert William Gainer Hunt. 'Light and Dark Adaptation and the Perception of Color'. In: *J. Opt. Soc. Am.* 42.3 (1952), pp. 190–199.

[35] Garrett M. Johnson and Mark D. Fairchild. 'A top down description of S-CIELAB and CIEDE2000'. In: *Color Research & Application* 28.6 (2003), pp. 425–435.

[36] Hideo Kasuga, Hiroaki Yamamoto and Masayuki Okamoto. 'Color quantization using the fast K-means algorithm'. In: *Systems and Computers in Japan* 31.8 (2000), pp. 33–40.

[37] James R. Kent, Wayne E. Carlson and Richard E. Parent. 'Shape transformation for polyhedral objects'. In: *Proceedings of the 19th annual conference on Computer graphics and interactive techniques* (1992).

[38] Anton Kruger. 'Median-cut color quantization'. In: *Dr Dobb's Journal-Software Tools for the Professional Programmer* 19.10 (1994), pp. 46–55.

[39] Chih-Hung Lee, Hao-ying Lu and Ji-Hwei Horng. 'Color quantization by hierarchical octa-partition in RGB color space'. In: *2018 IEEE International Conference on Applied System Invention (ICASI)*. 2018, pp. 147–150.

[40] John R. Levine. *The 'application/zlib' and 'application/gzip' Media Types*. RFC 6713. Aug. 2012.

[41] Chaofeng Li and Alan Bovik. 'Content-Weighted Video Quality Assessment Using a Three-Component Image Model'. In: *Journal of Electronic Imaging* 29 (Jan. 2010).

[42] Chun Hung Liu, Oscar C. Au, Peter H. W. Wong, M. C. Kung and Shen Chang Chao. 'Bit-depth expansion by adaptive filter'. In: *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2008, pp. 496–499.

[43] Jing Liu, Wanning Sun and Yutao Liu. 'Bit-depth enhancement via convolutional neural network'. In: *International Forum on Digital TV and Wireless Multimedia Communications*. Springer. 2017, pp. 255–264.

[44] Yuqing Liu, Qi Jia, Jian Zhang, Xin Fan, Shanshe Wang, Siwei Ma and Wen Gao. *Learning Weighting Map for Bit-Depth Expansion within a Rational Range*. 2022.

[45] Stuart Lloyd. 'Least squares quantization in PCM'. In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[46] Rafał Mantiuk, Kil Joong Kim, Allan G. Rempel and Wolfgang Heidrich. 'HDR-VDP-2: A Calibrated Visual Metric for Visibility and Quality Predictions in All Luminance Conditions'. In: *ACM Trans. Graph.* 30.4 (2011).

[47] Arnold Meijster, Jos BTM Roerdink and Wim H Hesselink. 'A general algorithm for computing distance transforms in linear time'. In: *Mathematical Morphology and its applications to image and signal processing* (2000), pp. 331–340.

[48] Gaurav Mittal, Vinit Jakhetiya, Sunil Prasad Jaiswal, Oscar C Au, Anil Kumar Tiwari and Dai Wei. 'Bit-depth expansion using minimum risk based classification'. In: *2012 Visual Communications and Image Processing*. IEEE. 2012, pp. 1–5.

[49] Laurent Najman and Michel Couprie. 'Building the Component Tree in Quasi-Linear Time'. In: *IEEE Transactions on Image Processing* 15.11 (2006), pp. 3531–3539.

[50] Helen Oleynikova, Zachary Taylor, Marius Fehr, Juan I. Nieto and Roland Siegwart. 'Voxblox: Building 3D Signed Distance Fields for Planning'. In: *CoRR* abs/1611.03631 (2016).

[51] Michael T Orchard, Charles A Bouman et al. 'Color quantization of images'. In: *IEEE Transactions on Signal Processing* 39.12 (1991), pp. 2677–2690.

[52] Savan Oswal, Anjali Singh and Kirthi Kumari. 'Deflate compression algorithm'. In: *International Journal of Engineering Research and General Science* 4.1 (2016), pp. 430–436.

[53] Shaimaa M. Othman, Amr E. Mohamed, Zaki Nossair and M. I. El-Adawy. 'Image Compression Using Polynomial Fitting'. In: *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*. 2019, pp. 344–349.

[54] Celal Ozturk, Emrah Hancer and Dervis Karaboga. 'Color Image Quantization: A Short Review and an Application with Artificial Bee Colony Algorithm'. In: *INFORMATICA,* 25 (Oct. 2014), pp. 485–503.

[55] Jean-François Pambrun and Rita Noumeir. 'Limitations of the SSIM quality metric in the context of diagnostic imaging'. In: *2015 IEEE International Conference on Image Processing (ICIP)*. 2015, pp. 2960–2963.

[56] Saurin S. Parikh, Damian Ruiz, Hari Kalva, Gerardo Fernández-Escribano and Velibor Adzic. 'High Bit-Depth Medical Image Compression With HEVC'. In: *IEEE Journal of Biomedical and Health Informatics* 22.2 (2018), pp. 552–560.

[57] Friedrich Pukelsheim. 'The three sigma rule'. In: *The American Statistician* 48.2 (1994), pp. 88–91.

[58] Jan Puzicha, Marcus Held, Jens Ketterer, Joachim Buhmann and Dieter Fellner. 'On spatial quantization of color images'. In: *IEEE Transactions on Image Processing* 9 (Apr. 2000), pp. 666–682.

[59] *Registration of a new MIME Content-Type/Subtype - application/zip*. IANA. Accessed: 2023-02-07. 1991.

[60] A. Harry Robinson and Colin Cherry. 'Results of a prototype television bandwidth compression scheme'. In: *Proceedings of the IEEE* 55.3 (1967), pp. 356–364.

[61] Azriel Rosenfeld and John L Pfaltz. 'Distance functions on digital pictures'. In: *Pattern Recognition* 1.1 (1968), pp. 33–61.

[62] Diego Santa-Cruz, Touradj Ebrahimi, Joel Askelof, Mathias Larsson and Charilaos Christopoulos. 'JPEG 2000 still image coding versus other standards'. In: *Proc SPIE* 4115 (Dec. 2000).

[63] Mark J Shensa et al. 'The discrete wavelet transform: wedding the a trous and Mallat algorithms'. In: *IEEE Transactions on Signal Processing* 40.10 (1992), pp. 2464–2482.

[64] Athanassios Skodras, Charilaos Christopoulos and Touradj Ebrahimi. 'The JPEG 2000 still image compression standard'. In: *IEEE Signal Processing Magazine* 18.5 (2001), pp. 36–58.

[65] Wen Sun, Yan Lu, Feng Wu and Shipeng Li. 'Level embedded medical image compression based on value of interest'. In: *2009 16th IEEE International Conference on Image Processing (ICIP)*. 2009, pp. 1769–1772.

[66] Akira Taguchi and Johji Nishiyama. 'Bit-length expansion by inverse quantization process'. In: *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*. 2012, pp. 1543–1547.

[67] Victor T. Tom. 'Adaptive Filter Techniques For Digital Image Enhancement'. In: *Digital Image Processing*. Ed. by Andrew G. Tescher. Vol. 0528. International Society for Optics and Photonics. SPIE, 1985, pp. 29 –42.

[68] Shiaw-Tsyr Uang, Yi-Lun Kao and Cheng-Li Liu. 'The Effects of Luminance Levels and Colors on Chromatic Perception'. In: *Proceedings of the Fifth Asia Pacific Industrial Engineering and Management Systems Conference* (Jan. 2004).

[69] Robert Ulichney and Shiufun Cheung. 'Pixel Bit-Depth Increase by Bit Replication'. In: *Proceedings of SPIE - The International Society for Optical Engineering* 3300 (May 1998).

[70] Oleg Verevka and John W. Buchanan. 'Local K-means Algorithm for Colour Image Quantization'. In: *Proceedings of Graphics Interface '95*. GI '95. Quebec, Quebec, Canada: Canadian Human-Computer Communications Society, 1995, pp. 128–135.

[71] Luc Vincent and Pierre Soille. 'Watersheds in digital spaces: an efficient algorithm based on immersion simulations'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.6 (1991), pp. 583–598.

[72] Thijs Vogels, Fabrice Rousselle, Brian Mcwilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer and Jan Novák. 'Denoising with Kernel Prediction and Asymmetric Loss Functions'. In: *ACM Trans. Graph.* 37.4 (2018).

[73] Gregory K Wallace. 'The JPEG still picture compression standard'. In: *IEEE Transactions on Consumer Electronics* 38.1 (1992), pp. xviii–xxxiv.

[74] Haohong Wang, G.M. Schuster, A.K. Katsaggelos and T.N. Pappas. 'An efficient rate-distortion optimal shape coding approach utilizing a skeleton-based decomposition'. In: *IEEE Transactions on Image Processing* 12.10 (2003), pp. 1181–1193.

[75] Jieying Wang, Jiří Kosinka and Alexandru Telea. 'Spline-based medial axis transform representation of binary images'. In: *Computers & Graphics* 98 (2021), pp. 165–176.

[76] Jieying Wang, Maarten Terpstra, Jiří Kosinka and Alexandru Telea. 'Quantitative Evaluation of Dense Skeletons for Image Compression'. In: *Information* 11.5 (2020).

[77] Zhou Wang, A.C. Bovik, H.R. Sheikh and E.P. Simoncelli. 'Image quality assessment: from error visibility to structural similarity'. In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612.

[78] Zhou Wang, Eero P. Simoncelli and Alan C. Bovik. 'Multiscale structural similarity for image quality assessment'. In: *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*. Vol. 2. 2003, 1398–1402 Vol.2.

[79] Ching-Yung Yang and Ja-Chen Lin. 'RWM-cut for color image quantization'. In: *Computers & Graphics* 20.4 (1996). Hardware Supported Texturing, pp. 577–588.

[80] James Zern, Pascal Massimino and Jyrki Alakuijala. *WebP Image Format*. Internet-Draft draft-zern-webp-12. Work in Progress. Internet Engineering Task Force, Dec. 2022. 51 pp.

[81] Jing Zhang, Qianqian Dou, Jing Liu, Yuting Su and Wanning Sun. 'BE-ACGAN: Photo-realistic residual bit-depth enhancement by advanced conditional GAN'. In: *Displays* 69 (2021), p. 102040.

[82] Xuemei Zhang, Brian A Wandell et al. 'A spatial extension of CIELAB for digital color image reproduction'. In: *SID international symposium digest of technical papers*. Vol. 27. Citeseer. 1996, pp. 731–734.

[83] Yang Zhao, Ronggang Wang, Yuan Chen, Wei Jia, Xiaoping Liu and Wen Gao. 'Lighter but Efficient Bit-Depth Expansion Network'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.5 (2021), pp. 2063–2069.

[84] Yang Zhao, Ronggang Wang, Wei Jia, Wangmeng Zuo, Xiaoping Liu and Wen Gao. 'Deep Reconstruction of Least Significant Bits for Bit-Depth Expansion'. In: *IEEE Transactions on Image Processing* 28.6 (2019), pp. 2847–2859.

[85] Jacob Ziv and Abraham Lempel. 'A universal algorithm for sequential data compression'. In: *IEEE Transactions on Information Theory* 23.3 (1977), pp. 337–343.