TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

Adaptive Image Processing

ausgeführt am

Institut für
Analysis und Scientific Computing
TU Wien

zur Erlangung des Titels Master of Science (MSc) unter der Anleitung von

Prof. Dr. Michael Feischl

eingereicht an der Technischen Universität Wien
Fakultät für Mathematik und Geoinformation
von

Hubert Hackl, BSc
[redacted]

Wien, am September 12, 2022

Hubert Hackl

Michael Feischl

Kurzfassung

Seit Jahrzehnten ist der Standard zur Speicherung von Bildern das .jpg Format. Der JPEG Algorithmus zerlegt dabei das Bild in ein Gitter bestehend aus 8×8 Pixel. Auf jedem dieser Blöcke wird das Bild, dann effizient gespeichert (siehe [Wik21] für Details). Ziel dieser Arbeit ist es ein gröberes Gitter adaptiv zu generieren um noch mehr Speicherplatz zu sparen bei nahezu unveränderter Bildqualität. Neben der Formulierung des Algorithmus wird auch ein Quasi-Optimalitätsresultat von Binev P. und DeVore R. für den Algorithmus formuliert (siehe [BD04]). Abschließend wird der Algorithmus anhand einiger Beispielbilder getestet, was zeigt, dass der Algorithmus bei einigen Bildern deutlich bessere Kompressionsraten aufweist als JPEG.

Abstract

Since decades the standard for storing images is the .jpg format. The JPEG algorithm decomposes the image into a mesh consisting of 8×8 pixel blocks. On each such block, the image is stored efficiently (see [Wik21] for details). The goal of this work is to obtain a coarses mesh adaptively to save even more storage space while also maintaining the image quality. Besides the formulation of such an algorithm, there is also a near-optimality result given by Binev P. and DeVore R. for the algorithm (siehe [BD04]). Finally, the algorithm is tested on some test images, which shows that the algorithm has a significantly better compression rate than the JPEG algorithm for many images.

Acknowledgments

Many people were involved in making this work possible.
First, I would like to thank my supervisor Michael Feischl for coming up with various interesting problems to choose from for the topic of this work, his patience and extraordinarily good guidance. I could not have thought of a better supervisor.
Next, I would like to thank my family for both their mental and financial support. Thanks to them, I did not need to work some boring student job in contrast to many less fortunate study colleagues.
I also thank all my study colleagues for all their help in study related and unrelated problems and the fun after work.
Finally, I would like to give my thanks to all friends that do not happen to also be study colleagues. Especially after the 4th semester they helped greatly in reinstalling some work life balance by spending a lot of recreational time together.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am September 12, 2022

Hubert Hackl

Contents

1 Introduction **1**

2 Setting **2**
- 2.1 Images . . . 2
- 2.2 Meshes and approximate Images . . . 3
- 2.3 Refining rectangles . . . 3
- 2.4 Estimators and near-optimality . . . 5

3 I_{app} with mean and ∞-norm **7**
- 3.1 Algorithm 1 . . . 7

4 I_{app} with JPEG and L^2-norm **9**
- 4.1 Basics of the JPEG algorithm . . . 9
- 4.2 L^2 estimation for adaptive JPEG . . . 10
  - 4.2.1 L^2 estimators for adaptive JPEG . . . 10
  - 4.2.2 The refinement property . . . 11
  - 4.2.3 A modified estimator . . . 14
  - 4.2.4 Algorithm 2 . . . 15
  - 4.2.5 Basics of adaptive tree refinement . . . 15
  - 4.2.6 Near-optimality of Algorithm 2 . . . 17
- 4.3 An algorithm with prescribed global error threshold . . . 22
  - 4.3.1 Another modification of the error functional . . . 22
  - 4.3.2 Algorithm 3 . . . 23
  - 4.3.3 Algorithm 4 . . . 26

5 I_{app} with JPEG and BV-norm **28**
- 5.1 Algorithm 5 . . . 28

6 Efficient storage of I_{app} **30**
- 6.1 Data structures . . . 30
- 6.2 Compression of I_{app} . . . 30
- 6.3 Decompression of I_{app} . . . 32

7 The performance of the algorithms **35**
- 7.1 Computation time with respect to the threshold parameter . . . 36
- 7.2 Computation time with respect to the image size . . . 37
- 7.3 Comparison of storage space needed with JPEG . . . 38

Bibliography **41**

1 Introduction

The JPEG algorithm has been the standard image compressing algorithm for decades. It decomposes any given image into a fixed mesh of 8×8 pixel blocks. On these blocks the image is transformed via 2D cosine transform. By means of a quantization matrix Q and a rounding step, most higher frequencies are then omitted resulting in a decent compression rate, i.e. significant less storage space needed when compared to saving each frequency (see [Wik21]). The goal of this work is to generate a mesh adaptively instead of using a fixed block size of 8×8 in order to get an even better compression rate.
In the next chapter, the basic notation is introduced and the notion of near-optimality is defined.
In chapter 3 the first algorithm is formulated with the supremumsnorm as driving functional. This means, if a pixel of the approximate image differs more than a prescribed tolerance from the original image, the element of the current mesh, in which that pixel is part of, gets refined.
In chapter 4, the basics of the JPEG algorithm are presented and an algorithm based upon the L_2-Norm is formulated. The algorithm driving functional is furthermore modified, such that there can a near-optimality result be shown (see [BD04]).
In chapter 5, the norm is switched once again. This time for the norm of bounded variation. This norm should conserve sharp edges in images better, but the resulting output does not differ noticeably from the last algorithm, if the tolerances are chosen accordingly.
In chapter 6, the details of the implementation of the last two algorithms are given.
In chapter 7, the last two algorithms are tested with eight test images, which shows that the algorithms can save a lot storage space compared to JPEG, while preserving the quality. The improved compression rate varies for different images. The difference is most noticeable for monotone or blurry images. Since many images have this property (backgrounds like the sky, a wall, simple furniture,...) the algorithms can often improve the compression rate of the JPEG algorithm substantially.

2 Setting

2.1 Images

Let an image with h (height) times w (width) pixels be given by a function:

$$I : \{1, ..., h\} \times \{1, ..., w\} \to \mathbb{R}^3,$$

where $I(i, j)$ corresponds to the normalized RGB-values of the pixel (i, j), i.e. $I(i, j) \in [0, 1]^3$.
The coordinates of the pixels are chosen such that the pixel $(1, 1)$ is the top left pixel and (h, w) is the bottom right pixel of the image, hence we can think of the image as a Matrix of size $h \times w$ with values in $\mathbb{R}^3$ (see Figure 2.1).

Our goal is to come up with two algorithms with some properties: The first one (encoding algorithm) takes an image I as input and returns a vector S. The second one (decoding algorithm) takes S as input and returns some approximate image I_{app}.
They should satisfy the following properties:

Property 1: I_{app} is an approximation of I, i.e. there holds

$$\|I - I_{app}\| < \tau$$

for a certain norm and a preset tolerance τ.

Property 2: The necessary storage space for S is (near-)minimal.

Property 3: The runtime for both the encoding and decoding algorithm is (near-)optimal.

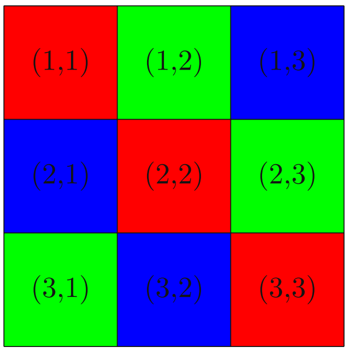


Figure 2.1: Similarity between coordinates of the pixels and the entries of a matrix.

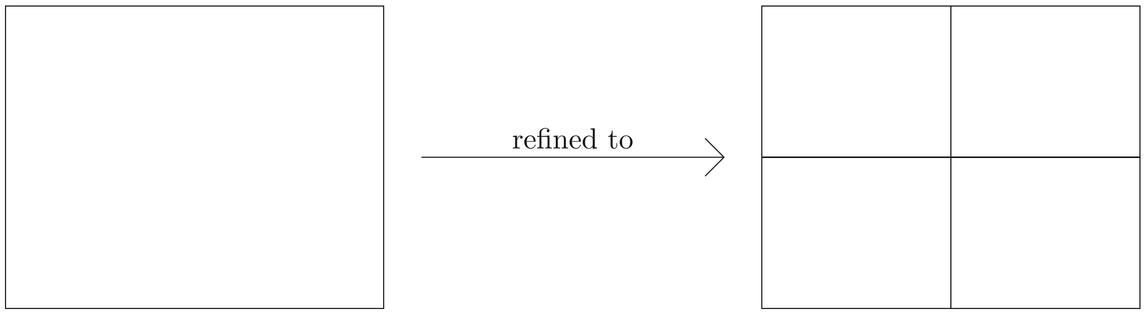


Figure 2.2: How rectangles are refined

Property 1 has to be satisfied in any case since the quality loss (w.r.t. the chosen norm) of the initial image being too large (i.e., greater than the tolerance τ) is unacceptable.

2.2 Meshes and approximate Images

The principal idea for obtaining an approximate image I_{app} is to divide the image into smaller rectangles and approximate the image on those.

Definition 2.2.1. For a given image I we define:

- A *rectangle* R is a subset of pixels in I which is given by a tripel $(fpy, fpx, width) \in \{1, ..., h\} \times \{1, ..., w\} \times \mathbb{N}$, such that (fpy, fpx) is the position in I of the top left pixel of R and $width$ is the width of R expressed in number of pixels. We also assume that the shape of R is similar to the shape of I, i.e. the ratio $height/width$ is the same.
- A *mesh* $\mathcal{T}$ is a set of rectangles, such that every pixel of the image is contained in exactly one rectangle.
- An *approximate image* I_{app} is an image that can be computed given only the image I and a mesh $\mathcal{T}$. While we hide the dependency upon I, we are often interested in which mesh was used to compute an approximate image and therefore usually denote it by $I_{app,\mathcal{T}}$. We also assume that for $R \in \mathcal{T}$ the restricted approximate image $I_{app,\mathcal{T}}|_R$ only depends on $I|_R$ and R.

Remark 1. The similarity assumption for rectangles implies, that if the size of I is $h \times w$ then the height of a rectangle R is given by $(h/w)width$.

2.3 Refining rectangles

Definition 2.3.1. Let I be an image. We define the initial mesh

$$\mathcal{T}_0 := \{R_0\}$$

with $R_0 := (1, 1, width(I))$, i.e., R_0 is the rectangle which covers the entire image.

We want to generate a new mesh $\mathcal{T}'$ from a given mesh $\mathcal{T}$ by refining a rectangle in $\mathcal{T}$. When we refine a rectangle $R \in \mathcal{T}$ we apply a *refinement rule* to substitute R in $\mathcal{T}$ with some new rectangles R_i, $i = 1, ..., m(R)$ such that this new set of rectangles is a mesh again, i.e., the set

$$\mathcal{T}' := \mathcal{T} \backslash \{R\} \cup \{R_i | i = 1, ..., m(R)\}$$

is a mesh. The refinement rule we are using creates new rectangles by bisecting both sides of the refined rectangle (see Figure 2.2). Hence $m(R) = 4$ for any rectangle R and the new rectangles are similar to the old one. Therefore they are also similar to R_0 if we started with the mesh $\mathcal{T}_0$.
We use the following terms:

- *Refinement* denotes the application of the refinement rule to one single rectangle.
- *Refinement step* denotes the refinement of each rectangle of a certain subset $M \subset \mathcal{T}$ of a mesh $\mathcal{T}$.

Definition 2.3.2. A mesh $\mathcal{T}$ is called *coarser* than a mesh $\mathcal{T}'$ if the mesh $\mathcal{T}'$ can be obtained by applying refinement steps on $\mathcal{T}$. In this case we also say $\mathcal{T}'$ is *finer* than $\mathcal{T}$ and write $\mathcal{T} < \mathcal{T}'$ or $\mathcal{T}' > \mathcal{T}$.
With this notion we further define the sets

$$\text{refine}(\mathcal{T}_0) := \{\mathcal{T} | \mathcal{T}_0 < \mathcal{T}\}$$

and

$$\mathbb{T}_n := \{\mathcal{T} \in \text{refine}(\mathcal{T}_0) | \text{ exactly } n \text{ refinements were done starting from } \mathcal{T}_0 \text{ to obtain } \mathcal{T}\}.$$

The relation "$<$" defined in Definition 2.3.2 is a semi-order on the set of meshes corresponding to image size $h \times w$ for any $h, w \in \mathbb{N}$. While two meshes might not be comparable, the refinement rule of our choice still gives *local nestedness*:

Lemma 2.3.3. *Let I be an image and $\mathcal{S}, \mathcal{T} \in \text{refine}(\mathcal{T}_0)$ two meshes. Then for any $R \in \mathcal{T}$ there holds*

$$\{R\} \subset \mathcal{S}|_R := \{T \in \mathcal{S} | T \textit{ is a subset of } R\} \tag{1.a}$$

or

$$\exists T \in \mathcal{S} : \{T\} \subset \mathcal{T}|_T. \tag{1.b}$$

Proof. Let $R \in \mathcal{T}$ be a rectangle. The assumption that $\mathcal{S}, \mathcal{T} \in \text{refine}(\mathcal{T}_0)$ implies that if R has to be refined for obtaining $\mathcal{S}$, then (1.a) holds. On the other hand if R doesn't have to be refined to obtain $\mathcal{S}$, then either $R \in \mathcal{S}$ (both, (1.a) and (1.b) are true) or we had to refine some $T \in \mathcal{S}$ to obtain $\mathcal{R}$, which means (1.b) holds. □

Note that (1.a) in Lemma 2.3.3 just means that if we restrict the meshes on $R \in \mathcal{T}$, the obtained mesh coming from $\mathcal{T}$ is coarser than the one obtained from $\mathcal{S}$. For (1.b) it is the other way around.

We have to decide before each refinement step for each rectangle in the mesh whether to refine it or not. For this we estimate the local error on each rectangle.

2.4 Estimators and near-optimality

Definition 2.4.1. Let $\|.\|$ be a norm of our choice on the space of images of size $h \times w$, I an image of size $h \times w$ and $\mathcal{T}$ a mesh. We define:

(i) The local error estimator:

$$\eta(R) := \|(I - I_{app,\mathcal{T}})|_R\| = \|(I - I_{app,\mathcal{T}})\mathbb{1}_R\| \qquad \forall R \in \mathcal{T}$$

(ii) The global error estimator

$$E(\mathcal{T}) := \|(I - I_{app,\mathcal{T}})\|$$

(iii) The best approximation error after refining n rectangles:

$$E_n := \min_{\mathcal{T} \in \mathbb{T}_n} E(\mathcal{T}).$$

Let a norm $\|.\|$ on the space of images of arbitrary size be fixed. The algorithms below are all based on the following procedure to generate adaptive meshes:

```
Input: Image $I$, tolerance $\tau > 0$
(1) Set $\mathcal{T} = \mathcal{T}_0$
(2) Compute $I_{app,\mathcal{T}}$ and $\eta(R)$ for all $R \in \mathcal{T}$
(3) Mark all $R \in \mathcal{T}$ which satisfy $\eta(R) > \tau$
(4) Obtain the mesh $\mathcal{T}'$ by doing a refinement step on the marked rectangles in $\mathcal{T}$
(5) Set $\mathcal{T} = \mathcal{T}'$ and go to (2)
Output: Sequence of meshes (which are obtained in step (4) in each iteration)
```

We will later add an additional step between the steps (4) and (5) to check if the global error $E(\mathcal{T})$ is smaller than some prescribed threshold parameter $\mu > 0$ to make an algorithm based on this procedure terminate at some point. Instead of the local error η in step (3) one can also use another algorithm driving functional $\widetilde{\eta}$ which might include other factors than just the local error. An algorithm based on this procedure has the important property, that after each iteration, the newly obtained mesh is finer than the old one.

Lemma 2.4.2. *Let I be an image, $\mathcal{T} \in \mathrm{refine}(\mathcal{T}_0)$ be a mesh. Then steps* (ii) *to* (v) *of the procedure described above produce a mesh $\mathcal{T}'$ which satisfies*

$$\mathcal{T} < \mathcal{T}'.$$

Proof. Let $M \subset \mathcal{T}$ denote the set of marked rectangles in step (iii) of the procedure. Then the mesh $\mathcal{T}'$ is obtained by refining the elements of M. This already implies that $\mathcal{T} < \mathcal{T}'$. $\square$

We now introduce the notion of near-optimality:

Definition 2.4.3. Let an Algorithm compute an approximation of a given image I on some (adaptively refined) mesh $\mathcal{T}_I$. We call the algorithm near-optimal (w.r.t. the number of refinements) if there exist positive constants C_1, C_2 and $n \in \mathbb{N}$ such that $\mathcal{T}_I \in \mathbb{T}_k$ for a $k \in \mathbb{N}$ with $0 \leq k \leq C_1 n$ and

$$E(\mathcal{T}_I) \leq C_2 E_n.$$

The constants are universal, i.e., they do not depend on the image I.

3 I_{app} with mean and ∞-norm

For our first approach, we set $S = I_{app}$ which makes the decoding algorithm redundant. This means for the moment, we focus on generating I_{app} and postpone the efficient storage of I_{app} to a later point.
In order to obtain the approximate picture I_{app}, we compute a sequence of meshes consisting of rectangles which are similar to the original picture I.
For a given mesh $\mathcal{T}$ we compute the approximate image $I_{app,\mathcal{T}}$ by taking the mean value on rectangles:

$$I_{app,\mathcal{T}}\Big|_R(i,j) := \frac{1}{|R|}\sum_{(l,m)\in R} I(l,m) \quad \forall (i,j) \in R \ \ \forall R \in \mathcal{T}.$$

We choose the L^∞ norm for adaptive refinement. The local estimator then can be written as

$$\eta(R) = \|(I - I_{app,\mathcal{T}})\Big|_R\|_\infty = \max_{(i,j)\in R}|I(i,j) - I_{app,\mathcal{T}}(i,j)|.$$

The global error estimator is then given by

$$\eta(\mathcal{T}) = \max_{R\in\mathcal{T}}\eta(R).$$

Based on the procedure in the last section this leads to:

3.1 Algorithm 1

Algorithm 1 L^∞-threshold with mean value approximation

```
Input: Original picture $I$, tolerance $\tau > 0$
(1) Set $\mathcal{T} = \mathcal{T}_0$
(2) Compute $I_{app,\mathcal{T}}$
(3) Compute local error $\eta(R)$ for all $R \in \mathcal{T}$
(4) Mark $R$ if $\eta(R) > \tau$ for all $R \in \mathcal{T}$
(5) If at least one rectangle is marked $\rightarrow$ refine marked elements, set $\mathcal{T}$ to the obtained mesh and go to (2)
(6) Else $\rightarrow$ return $I_{app} := I_{app,\mathcal{T}}$
```

The following theorem explains why it is convenient to choose the maximum norm in both the estimator and in **Property 1**.

Theorem 3.1.1. *Consider the last mesh $\mathcal{T}$ before termination of* **Algorithm 1**. *Let $\mathcal{S} \in \text{refine}(\mathcal{T}_0)$ and assume $\mathcal{S}$ has* **Property 1** *(i.e., $I_{app,\mathcal{S}}$ has* **Property 1***). Then there holds:*

$$\#\mathcal{T} \leq \#\mathcal{S},$$

i.e., the generated mesh by **Algorithm 1** *is optimal with respect to the number of elements.*

Proof. We prove the statement by contradiction. Assume that

$$|\mathcal{T}| > |\mathcal{S}|. \tag{3.1}$$

Lemma 2.3.3 and inequality (3.1) imply that there exists a rectangle $R' \in \mathcal{S}$ with $R' \notin \mathcal{T}$ such that R' has been refined while generating the mesh $\mathcal{T}$. The condition for marking and therefore refining an element R in **Algorithm 1** is:

$$\text{mark if } \eta(R) > \tau.$$

This implies

$$\max_{R \in \mathcal{S}} \eta(R) > \tau.$$

Therefore $\mathcal{S}$ doesn't have **Property 1** which contradicts the assumptions. □

We see experimentally that a constant mean value approximation on the rectangles is not well suited for practical purposes since the structure of the mesh always remains clearly visible unless for „very small“ tolerances τ which usually result in very fine meshes (many rectangles which only consist of one pixel!) leading to a very weak **Property 2** and a very weak **Property 3**.

4 I_{app} with JPEG and L^2-norm

This chapter is based on [BD04] and [Wik21].
We begin by trading the approximation by mean-value on a rectangle from the previous chapter for an approximation using the discrete cosine transform of the image. We will do this again locally on each rectangle of the mesh separately and omit higher frequencies if their coefficient is small enough.
Additionally we convert the function values of the image, which are given in the RGB color basis, to the YCbCr color basis. The motivation for this is the observation that the human eye is more sensitive towards brightness (the Y component) than towards chromatic components (Cb/Cr components) according to [Wik21]. We sample the C_b and C_r components down to half the image height and width by taking the mean values of four neighbouring pixels. Then we use the algorithm we come up with for the brightness component Y on these smaller components of the image and after the compression we double the height and width again by splitting each pixel into 4 like in the refinement rule to obtain the original size again. This leads to chromatic components being constant on certain blocks of 2×2 pixels which does not lead to a noticeable quality loss (see [HC92] for a study on noticeable quality losses using JPEG).

4.1 Basics of the JPEG algorithm

The JPEG algorithm realizes the above ideas but without generating an adaptive mesh, i.e., the domain of the picturefunction is split into 8×8 pixelblocks on which the discrete cosine transform is applied for each component of the YCbCr basis. Then the resulting three 8×8 matrices are divided entrywise by a so called quantization matrix Q of the same size. The entries of the obtained matrix are then rounded to the nearest integer, which usually gives a matrix with many zeros towards the bottom right. This can be stored efficiently and concludes the compression algorithm for JPEG files (see [HLN$^+$18] for further details about the advantages of JPEG).
The decoding algorithm multiplies entrywise the stored coefficientmatrix of a component on an 8×8 rectangle by the quantization matrix Q, performs the inverse discrete cosine transform and converts the resulting YCbCr values back to the RGB colorbasis to generate the picture.
The only loss comes from the rounding step in the encoding algorithm, which by choice of a suitable quantization matrix Q, mainly affects higher frequencies, which the human eye is less sensitive to anyway.
In the next sections we want to come up with a suitable error estimator for generating an adaptive mesh on which we proceed similar to the JPEG algorithm.

4.2 L^2 estimation for adaptive JPEG

In this section we change the estimators from **Algorithm 1**, where we used the maximum norm, to estimators based on the euclidean norm. As in the JPEG algorithm we limit the minimal size of rectangles obtained to be at least 8 pixels in each dimension. It remains to come up with a good marking strategy compared to the one we had in **Algorithm 1**. This is actually not trivial, since there is no obvious optimality result like for the marking strategy in **Algorithm 1** (Theorem 3.1.1). However, before we come to that, we clarify what estimators we are using:

4.2.1 L^2 estimators for adaptive JPEG

The estimators we are using are now based on the euclidean norm: The approximate image on a given mesh is given similarly to the JPEG algorithm, i.e., we use the discrete cosine transform (DCT) and a quantization Matrix Q. The quantization matrix we are using is the same as in [Wik21]:

$$Q := \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}$$

We now define some operators:

Definition 4.2.1. Let R be a rectangle of size $h \times w$.

$$\begin{aligned} \mathrm{TL}_R : & \mathbb{R}^{h\times w} \to \mathbb{R}^{8\times 8} \\ & (a_{ij})_{1\le i\le h, 1\le j\le w} \mapsto (a_{ij})_{1\le i,j\le 8} \end{aligned}$$

$$\begin{aligned} \mathrm{EMB}_R : & \mathbb{R}^{8\times 8} \to \mathbb{R}^{h\times w} \\ & (a_{ij})_{1\le i,j\le 8} \mapsto (b_{ij})_{1\le i\le h, 1\le j\le w}, \end{aligned}$$

where $b_{ij} = a_{ij}$ if $i \le 8 \wedge j \le 8$ and $b_{ij} = 0$ if $i > 8 \vee j > 8$. Let further DCT_R denote the 2D discrete cosine transform on $\mathbb{R}^{h\times w}$ and IDCT_R its inverse.

Remark 2. Note that these operators solely depend on the size of R. The operator TL_R projects matrices of the size of R onto their top left 8×8 submatrix, while the operator EMB_R embeds an 8×8 matrices into $\mathbb{R}^{h\times w}$ by adding zeros to the right and bottom. We will later ensure, that no rectangles R with $h < 8$ or $w < 8$ will occur. Consequently, the operators above are well defined.

Let the symbols $\odot$ and $./$ denote pointwise multiplication and division.

Definition 4.2.2. For a given image I and a mesh $\mathcal{T}$ we compute an approximate image $I_{app,\mathcal{T}}$ via

$$I_{app,\mathcal{T}}|_R = \mathrm{IDCT}_R(\mathrm{EMB}_R(Q \odot \mathrm{round}(\mathrm{TL}_R(\mathrm{DCT}_R(I|_R))./Q))) \quad \forall R \in \mathcal{T}. \tag{4.1}$$

However, for the adaptive procedure we will ignore the quantization and therefore only compute the approximate image for the final mesh this way. For a mesh $\mathcal{T}' \subsetneq \mathcal{T}$, which occurs during the algorithm, we compute an approximate image $I_{app,\mathcal{T}'}$ by

$$I_{app,\mathcal{T}'}|_R = \mathrm{IDCT}_R(\mathrm{EMB}_R(\mathrm{TL}_R(\mathrm{DCT}(I|_R)))) \quad \forall R \in \mathcal{T}'. \tag{4.2}$$

In the next sections we will only need (4.2). Only later, when our goal is to efficiently store and reconstruct the approximate image, we will go back to using (4.1) for the final mesh and approximate image.

Definition 4.2.3. The estimators with respect to the euclidean norm for a given image I of size $h \times w$, a mesh $\mathcal{T}$ and a given rectangle $R \in \mathcal{T}$ are given by:

(i) The local error estimator:

$$\eta(R)^2 := \frac{1}{hw} \sum_{(i,j)\in R} (I_{app,\mathcal{T}}(i,j) - I(i,j))^2$$

(ii) The global error estimator

$$E(\mathcal{T})^2 := \sum_{R'\in\mathcal{T}} \eta(R')^2 = \frac{1}{hw} \sum_{(i,j)\in I} (I_{app,\mathcal{T}}(i,j) - I(i,j))^2$$

(iii) The best approximation error after n refinements:

$$E_n := \min_{\mathcal{T}\in T_n} E(\mathcal{T}),$$

Note that this global error estimator is also called the mean squared error (MSE). The quantity E_n usually is not explicitly known or computed and neither is a mesh $\mathcal{T}$ on which this minimum is attained.

4.2.2 The refinement property

The principal idea is again to calculate local errors using a local estimator and refine those rectangles which exceed the preset tolerance τ. However, unlike in **Algorithm 1**, we have no guarantee that refining a rectangle actually decreases the global error with respect to the L^2 norm. Therefore we will need the *refinement property*:

$$\sum_{i=1}^{4} \eta(R_i)^2 \leq C_0 \eta(R)^2 \qquad \text{with } R_1, .., R_4 \text{ children of } R \quad \forall R \in \mathcal{T} \quad \forall \mathcal{T} \in \mathrm{refine}(\mathcal{T}_0), \tag{4.3}$$

for some constant $C_0 > 0$. While we do not know if refining leads to a smaller global error, we do know that it at least does not lead to a worse error if $C_0 \leq 1$. Unfortunately (4.3) does not hold for every image. However, we can still come up with a result that will prove to be sufficient for the results to follow. To this end, let I be an image and R the rectangle covering it, i.e. size(I)=size(R)=(h, w). Recall that the operators defined in **Definition** 4.2.1 only depend on the size of R. Therefore, we set

$$\mathrm{TL}_{R/2} := \mathrm{TL}_{\widetilde{R}}, \text{ with } \widetilde{R} \text{ being any rectangle of size } h/2 \times w/2 \tag{4.4}$$

and likewise define $\mathrm{EMB}_{R/2}$, $\mathrm{DCT}_{R/2}$ and $\mathrm{IDCT}_{R/2}$. For $i \in \{1, .., 4\}$, we consider the operators

$$\begin{aligned}
&A_i : \mathbb{R}^{h\times w} \to \mathbb{R}^{h/2\times w/2} \\
&A_i := (\mathrm{Id} - \mathrm{EMB}_{R/2}\mathrm{TL}_{R/2})(\mathrm{DCT}_{R/2}(\mathrm{P}_{\frac{h}{2}\times\frac{w}{2},i}(\mathrm{IDCT}_R(\mathrm{EMB}_R(\mathrm{TL}_R(\mathrm{DCT}_R)))))),
\end{aligned}$$

where $\mathrm{P}_{\frac{h}{2}\times\frac{w}{2},i}$ denotes the restriction to the i-th subimage of size $h/2 \times w/2$. The subimages are: 1.Top-Left, 2.Top-Right, 3.Bottom-Left, 4.Bottom-Right. Hence, the operator A_i is a composition of first mapping an image I to the approximate image on the size of I, then to the i-th subimage and finally subtracting the approximate image on the size of the subimage. These operators are continuous, because they are a composition of orthogonal projections, embeddings and (inverse) discrete cosine transforms. For further use, we compute the continuity constant exactly by rearranging the input image into a vector of size hw and representing the (inverse) discrete cosine transforms and projections/embeddings as matrices accordingly. The norm of the resulting operator, represented by a matrix, can be computed for instance in Matlab with the command 'norm'. This leads to

$$\|A_i\| = 0.08 =: \delta \quad \forall i \in \{1, .., 4\}$$

for $h = w = 8$ and even smaller for bigger h and w. Since we are aiming for an upper bound and we will not refine rectangles which are smaller than eight pixels in one dimension, we stick to this value. We aim to argue that the refinement property (4.3) is satisfied for most images. To that end we assume a small noise on the frequencies given by a term $\varepsilon\xi_i$, where the amplitude $\varepsilon > 0$ is small and ξ_i is a normally distributed random variable for each pixel i. Let $\widetilde{I}$ be an image we get from an image I of size $h \times w$ by adding such a noise, i.e., given as a vector $I = (\alpha_i)_{i=1}^{hw}$ and hence $\widetilde{I} = (\alpha_i + \varepsilon\xi_i)_{i=1}^{hw}$. With the additional noise, there might be pixels which have a value which is not within the permissible interval of the color component. However, for the calculations to follow, this is not a problem. $\widetilde{I}$ is a random variable and its L^2 norm is a non-central chi-squared distributed random variable. The local error $\eta(R_i)$ can be described by restricting the image to the subimage on R_i, subtracting the approximate image on R_i and then taking the norm. Therefore, we compute for $i \in \{1, ..4\}$:

$$\begin{aligned}
\eta(R_i) =&\|(\mathrm{Id} - \mathrm{EMB}_{R/2}\mathrm{TL}_{R/2})\mathrm{DCT}_{R/2}\mathrm{P}_{\frac{h}{2}\times\frac{w}{2},i}(\widetilde{I})\|_2 \\
\leq&\|(\mathrm{Id} - \mathrm{EMB}_{R/2}\mathrm{TL}_{R/2})\mathrm{DCT}_{R/2}\mathrm{P}_{\frac{h}{2}\times\frac{w}{2},i}\mathrm{IDCT}_R(\mathrm{Id} - \mathrm{EMB}_R\mathrm{TL}_R)\mathrm{DCT}_R(\widetilde{I})\|_2 \\
&+ \|A_i(\widetilde{I})\|_2 \\
\leq&\|(\mathrm{Id} - \mathrm{EMB}_R\mathrm{TL}_R)\mathrm{DCT}_R(\widetilde{I})\|_2 + \delta\|\widetilde{I}\|_2.
\end{aligned}$$

For the first inequality we used the triangular inequality for the identity represented as

$$\mathrm{Id} = \mathrm{IDCT}_R(\mathrm{Id} - \mathrm{EMB}_R\mathrm{TL}_R)\mathrm{DCT}_R + \mathrm{IDCT}_R(\mathrm{EMB}_R\mathrm{TL}_R(\mathrm{DCT}_R)).$$

Next, we show that

$$\delta\|\widetilde{I}\|_2 \leq \|(\mathrm{Id} - \mathrm{EMB}_R\mathrm{TL}_R)\mathrm{DCT}_R(\widetilde{I})\|_2 \tag{4.5}$$

holds with high probability for a suitably chosen amplitude ε. To show this, we first note, that we can w.l.o.g. assume that $\|\widetilde{I}\| = 1$ by linearity of the operators. The upper bound is a random variable, hence we start by computing a lower bound for the amplitude ε such that the inequality (4.5) is true for the expected value

$$\delta^2 \leq \mathbb{E}(\|(\mathrm{Id} - \mathrm{EMB}_R\mathrm{TL}_R)\mathrm{DCT}_R(\widetilde{I})\|_2^2) = \mathbb{E}(\varepsilon^2 \sum_{i=65}^{256} (\frac{\alpha_i}{\epsilon} + \xi_i)^2) = \varepsilon^2(192 + \sum_{i=65}^{256} \frac{\alpha_i^2}{\varepsilon^2}).$$

By estimating the last sum from below with 0, this leads to

$$\varepsilon \geq \sqrt{(\delta^2/192)} = 0.0058.$$

In this computation we used, that there is a noise in at least 192 frequencies, which occurs when we are refining a rectangle of size 16×16 or larger. The computation for the expected value shows, that for larger rectangles we get milder conditions on ε, i.e., a smaller amplitude ε is sufficient. However, we not only want the equality to be true for the expected value but to be true with high probability. For this, we use the distribution function for the chi-squared distribution for an even amount of degrees of freedom

$$F_{192}(x) = 1 - \mathrm{e}^{-\frac{x}{2}} \sum_{k=0}^{n/2-1} \frac{1}{\Gamma(k+1)} \left(\frac{x}{2}\right)^k,$$

with which we compute

$$\mathbb{P}(\sum_{i=65}^{256} \xi_i^2 \geq \frac{\delta^2}{\varepsilon^2}) = 1 - F_{192}(\frac{\delta^2}{\varepsilon^2}) = \begin{cases} 0.5221 & \varepsilon = 0.0058, \\ 0.9999 & \varepsilon = 0.0071. \end{cases}$$

We omitted the terms α_i/ϵ in the sum, since non-central chi-squared random variables have a distribution function which lies above the one for the central chi-squared distribution. A probability of 52.21% is a bit low since the inequality should hold in most cases. Changing our amplitude ε to 0.0071 we obtain a probability of 99.99% to satisfy the inequality, which we shall consider good enough in view of the discussion below. Finally, we can show a (stochastic) refinement property for our estimators:

$$\begin{aligned}\sum_{i=1}^{4} \eta(R_i)^2 =& \sum_{i=1}^{4} \|(\mathrm{Id} - \mathrm{EMB}_{R/2}\mathrm{TL}_{R/2})\mathrm{DCT}_{R/2}\mathrm{P}_{\frac{h}{2}\times\frac{w}{2},i}(\widetilde{I})\|_2^2 \\ \leq& \sum_{i=1}^{4} (2\|(\mathrm{Id} - \mathrm{EMB}_R\mathrm{TL}_R)\mathrm{DCT}_R(\widetilde{I})\|_2)^2 \\ \leq& 16\|(\mathrm{Id} - \mathrm{EMB}_R\mathrm{TL}_R)\mathrm{DCT}_R(\widetilde{I})\|_2^2 = 16\eta(R)^2,\end{aligned}$$

where the inequalites hold with a probability of 99.99%. We also want to note that a noise with an amplitude of 0.0071 on high frequencies is not noticeable to the human eye as demonstrated in Figure 4.1 even for an amplitude of 0.015. This means that images which do not satisfy (4.3) are the rare exception.

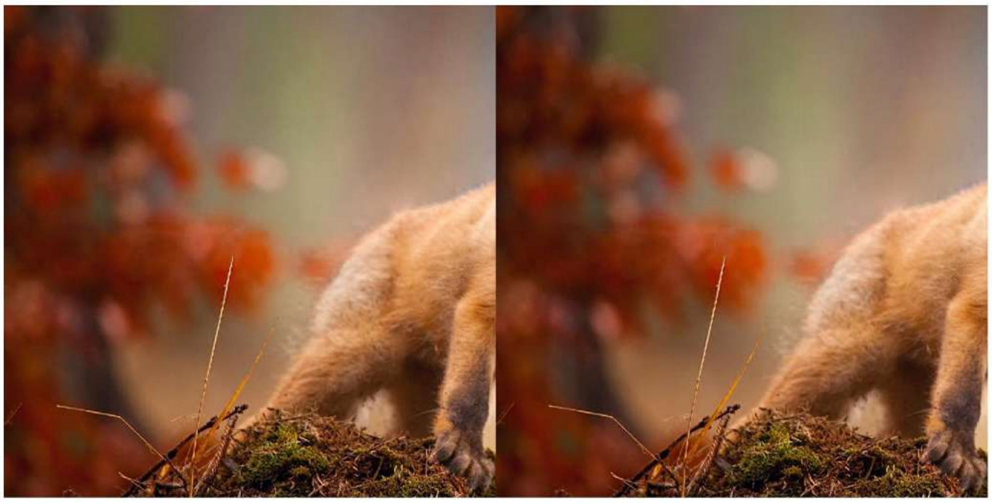


Figure 4.1: Left: The original image. Right: The original image with an additional noise of amplitude 0.015.

4.2.3 A modified estimator

In order to optimally decide which rectangles to refine, we modify the estimator by introducing some parameters:

Let $R \in \mathcal{T}$ be given and let $S(R)$ denote the set of its siblings (i.e., all $R' \in \mathcal{T}$ which are created from the same parent R). Our refinement rule leads to $\#(S(R)) = 4$ for any rectangle R. We define the quantity

$$\lambda(R) := \frac{\eta(R)^2}{\sum_{R' \in S(R)} \eta(R')^2} \tag{4.6}$$

in case the denominator is positive. It is the portion of the error on $S(R)$ that is caused by the approximation on R.

Let $\tau > 0$ be a preset threshold parameter, $R \in \mathcal{T}$, $\mathcal{T} \in \text{refine}(T_0)$ and $R_1, .., R_4$ the children of R. We define

$$\sigma(R) := \sum_{i=1}^{4} \eta(R_i)^2$$

$$d(R) := \eta(R)^2 - \sigma(R) \tag{4.7}$$

$$\delta(R) := \max\{\tau - d(R), 0\}. \tag{4.8}$$

We further define $\alpha(R_0) = 0$ for R_0 being the rectangle which corresponds to the entire picture (i.e., R_0 is the single element of the initial mesh $\mathcal{T}_0$). For any other rectangle R we define

$$\alpha(R) \coloneqq \lambda(R)\,(\alpha(R_p) + \delta(R_p)) \tag{4.9}$$

with R_p being the parent rectangle of R, i.e., R is obtained by applying the refinement rule to R_p. The first term in (4.9) corresponds to the error reduction on the way to creating R and the second term corresponds to the new error reduction caused by refining R.

We now define the modified estimator

$$\widetilde{\eta}(R)^2 \coloneqq \eta(R)^2 - \alpha(R). \tag{4.10}$$

In case the denominator in the definition of $\lambda(R)$ is zero, we simply set $\widetilde{\eta}(R)$ to zero.

We can rewrite the modified estimator:

$$\widetilde{\eta}(R)^2 = \eta(R)^2 - \alpha(R) = \eta(R)\left(1 - \frac{\alpha(R_p) + \delta(R_p)}{\sigma(R_p)}\right).$$

Remark 3. Since the terms in the big bracket only depend on the parent R_p of R and $\eta(R) \geq 0$ for any rectangle R, we can deduce that if $\widetilde{\eta}(R) > 0$ then all the siblings R' of R will satisfy $\widetilde{\eta}(R') \geq 0$.

4.2.4 Algorithm 2

Before we state the algorithm, we split the approximate image I_{app} into its components $I^Y_{(app)} \coloneqq \pi_1(I_{(app)})$, $I^{Cb}_{(app)} \coloneqq \pi_2(I_{(app)})$ and $I^{Cr}_{(app)} \coloneqq \pi_3(I_{(app)})$. This will allow the underlying meshes for each of these functions to vary. Together with the modified estimator this leads to the following encoding algorithm:

Algorithm 2 Modified L^2-threshold with DCT approximation

```
Let $X \in \{Y, C_b, C_r\}$
Input: image component $I^X$, tolerance $\tau > 0$
(1) Set initial mesh $\mathcal{T} = \mathcal{T}_0$
(2) Compute $I^X_{app,\mathcal{T}}$
(3) Compute modified error $\widetilde{\eta}(R)^2$ for all $R \in \mathcal{T}$
(4) Mark $R$ if $\widetilde{\eta}(R)^2 > \tau$ for all $R \in \mathcal{T}$
(5) If at least one rectangle is marked $\rightarrow$ refine marked elements, set $\mathcal{T}$ to the obtained mesh and go back to (2)
(6) Else $\rightarrow$ set $I^X_{app} = I^X_{app,\mathcal{T}}$
(7) Return $I^X_{app}$
```

4.2.5 Basics of adaptive tree refinement

Our next goal is to prove a near optimality result in the spirit of Definition 2.4.3 for **Algorithm 2**. However, before we can analyse **Algorithm 2** we have to introduce the concept of adaptive tree refinement:

Definition 4.2.4. We call a tupel $(Nodes, Edges)$, with

$$Edges \subset (Nodes \times Nodes) \setminus \{(e,e) | e \in Nodes\}$$

a *tree* if

(i) There is a "connection" between any two nodes, i.e.,

$\forall e, g \in Nodes \; e \neq g :$

(1) $(e,g) \in Nodes \vee$

(2) $(g,e) \in Nodes \vee$

(3) $\exists n \in \mathbb{N} : \exists f_1, ..., f_n \in Nodes : \{(e,f_1),(f_1,f_2),...,(f_{n-1},f_n),(f_n,g)\} \subset Edges \vee$

(4) $\exists n \in \mathbb{N} : \exists f_1, ..., f_n \in Nodes : \{(g,f_1),(f_1,f_2),...,(f_{n-1},f_n),(f_n,e)\} \subset Edges \vee$

(5) $\exists h \in Nodes : \exists m, n \in \mathbb{N} : \exists f_1, ..., f_m, k_1, ..., k_n \in Nodes :$

$(\{(h,f_1),(f_1,f_2),...,(f_{m-1},f_m),(f_m,g)\} \subset Edges \; \vee (h,g) \in Edges) \; \wedge$

$(\{(h,k_1),(k_1,k_2),...,(k_{n-1},k_n),(k_n,e)\} \subset Edges \; \vee (h,e) \in Edges)\,.$

If (1) or (3) is true, we say that g is a *descendant* of e and that e is a *predecessor* of g. Therefore, if (2) or (4) is true, e is a descendant of g and g is a predecessor of e.

(ii) These connections are unique, i.e.,

There is no node which is its own descendant. $\wedge$

$\forall e, g \in Nodes \; e \neq g,\;$ e is not a descendant of g, g is not a descendant of e :

$\nexists f \in Nodes : f$ is a descendant of both e and g.

Note that property (ii) of trees guarantees that no node can be both predecessor and descendant of another node. Case (5) of property (i) of trees just means that the two nodes have a common predecessor. It can be shown, that a tree always has a unique node, which is a predecessor of every other node. This node is called the root of the tree. We do not prove that here, because the trees we will consider all have such a node by construction.

Let an algorithm generate a mesh $T \in \text{refine}(\mathcal{T}_0)$, where $\mathcal{T}_0$ is our usual single element initial mesh. We consider an infinite master tree B^* with nodes corresponding to all rectangles that can be obtained by a finite amount of refinement steps on the initial mesh $\mathcal{T}_0$. The edges connect parents with their children. Since there are no coarsening steps in **Algorithm 2** we obtain for any possible output a corresponding tree $B \subset B^*$, which represents all refinements of **Algorithm 2** in order to generate $\mathcal{T}$. We introduce some notation:

Definition 4.2.5. Let B be the corresponding tree to an algorithm which generates a mesh. Then, we consider the sets

$$N(B) := \{R \in \text{Nodes}(B^*) | R \text{ and all its children are in } B\}$$
$$L(B) := \text{Nodes}(B) \setminus N(B).$$

We will also consider the cardinality of a tree which corresponds to the number of nodes in the tree. Therefore

$$\#B := |\text{Nodes}(B)|.$$

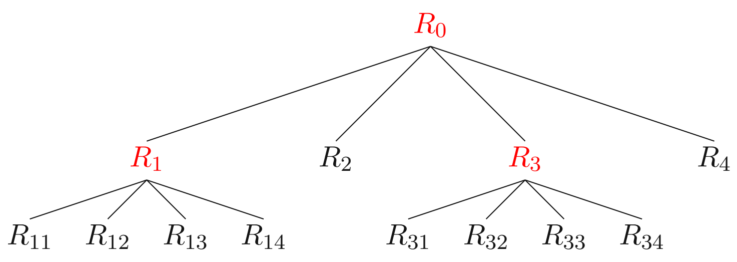


Figure 4.2: A tree with interior nodes R_0, R_1 and R_3. The other nodes are leaves.

We note that

$$\text{Nodes}(B) = N(B) \cup L(B) \text{ and}$$
$$L(B) \text{ corresponds to } \mathcal{T},$$

where the union is disjoint and $\mathcal{T}$ is the mesh generated by the underlying algorithm. The set $N(B)$ is called the set of interior nodes of B, which also corresponds to the number of refinements done. The set $L(B)$ is called the set of leaves of B. A mesh $\mathcal{T}$ defines a corresponding tree $B(\mathcal{T})$ by adding all predecessors of nodes in $\mathcal{T}$ within B^* to the tree in addition to the nodes corresponding to the rectangles in $\mathcal{T}$.

4.2.6 Near-optimality of Algorithm 2

The goal in this section is to show the following (near-)optimality result for **Algorithm 2**:

Theorem 4.2.6. *Let $\tau > 0$ be any threshold parameter for which* **Algorithm 2** *gives a final mesh $\mathcal{T}$ and corresponding tree $B := B(\mathcal{T})$, i.e., Step (6) of the algorithm is reached. Then*

$$E(\mathcal{T}) \leq 10 E_m \tag{4.11}$$

where $m = |N(B)|/2$. The algorithm uses $\#B = 1 + 4|N(B)|$ computations of η in generating $\mathcal{T}$.

For the proof we will need three lemmas. Unfortunately the given proof for one of these lemmas and consequently also for the proof of the theorem requires that the constant in the refinement property is (less or) equal to 1. This leads to $d \geq 0$ and hence $\delta \leq t$, which will be used towards the end of the respective lemmas proof.

Lemma 4.2.7. *Let $\tau > 0$ and let the tree B correspond to the output of* **Algorithm 2**. *If R is a node in $N(B)$ and B_R the subtree consisting of R and all $R' \in B$, such that R' is a descendant of R then*

$$\tau \frac{\#B_R}{5} \leq \tau |N(B_R)| \leq \widetilde{\eta}(R)^2 \leq \eta(R)^2. \tag{4.12}$$

Proof. We begin with the first inequality in (4.12). Since $\tau > 0$ and $\#B_R = |N(B_R)| + |L(B_R)|$, this inequality is equivalent to

$$L(B_R) \leq 4|N(B_R)|.$$

This is clearly true as the leaves are obtained by refining a subset of $N(B_R)$ which leads to a number of children bounded by 4 times the cardinality of the subset.
The last inequality in (4.12) is trivial by definition of $\widetilde{\eta}$ and the fact that $\alpha \geq 0$ for any rectangle.

It remains to show the second inequality of (4.12). For a rectangle respectively a node $R' \in B^*$ with children $R_1, .., R_4 \in B^*$, we obtain by (4.9) and (4.10)

$$\widetilde{\eta}(R_i)^2 = \eta(R_i)^2 - \lambda(R_i)\left(\alpha(R') + \delta(R')\right) \qquad \forall i \in \{1, ..4\}.$$

Summing up over all children of R' gives

$$\sum_{i=1}^{4} \widetilde{\eta}(R_i)^2 = \sum_{i=1}^{4} \eta(R_i)^2 - \left(\alpha(R') + \delta(R')\right) \sum_{i=1}^{4} \lambda(R_i).$$

Considering the definition of λ in (4.6), we immediately see that the last sum is equal to 1. This leads to

$$\sum_{i=1}^{4} \widetilde{\eta}(R_i)^2 = \sum_{i=1}^{4} \eta(R_i)^2 - \alpha(R') - \delta(R').$$

The definition of d in (4.7) and the last identity lead to

$$\begin{aligned}\widetilde{\eta}(R')^2 - \sum_{i=1}^{4} \widetilde{\eta}(R_i)^2 &= \eta(R')^2 - \alpha(R') - \sum_{i=1}^{4} \eta(R_i)^2 + \alpha(R') + \delta(R') \\ &= \eta(R')^2 - \sum_{i=1}^{4} \eta(R_i)^2 + \delta(R') \\ &= d(R') + \delta(R').\end{aligned}$$

The definition of δ in (4.8) implies that $d(R') + \delta(R') \geq \tau$ from which we obtain

$$\tau \leq \widetilde{\eta}(R')^2 - \sum_{i=1}^{4} \widetilde{\eta}(R_i)^2$$

and therefore

$$\sum_{i=1}^{4} \widetilde{\eta}(R_i)^2 \leq \widetilde{\eta}(R')^2 - \tau. \tag{4.13}$$

This equation shows that each subdivision reduces the error $\widetilde{\eta}^2$ by at least τ.
We define $\mathcal{B}_0$ to be the subtree of B_R obtained by removing all leaves $R' \in L(B_R)$ from B_R for which all its siblings are also elements of $L(B_R)$ (see Figure 4.3).
Below, we apply the estimate (4.13) to the bottom (i.e., nodes with the largest distance to the root which is a subset of $L(\mathcal{B}_0)$) of $\mathcal{B}_0$ and repeat inductively until we have an estimate for the root R:

$$\sum_{R' \in L(\mathcal{B}_0)} \widetilde{\eta}(R')^2 + \tau|N(\mathcal{B}_0)| \leq \widetilde{\eta}(R)^2. \tag{4.14}$$

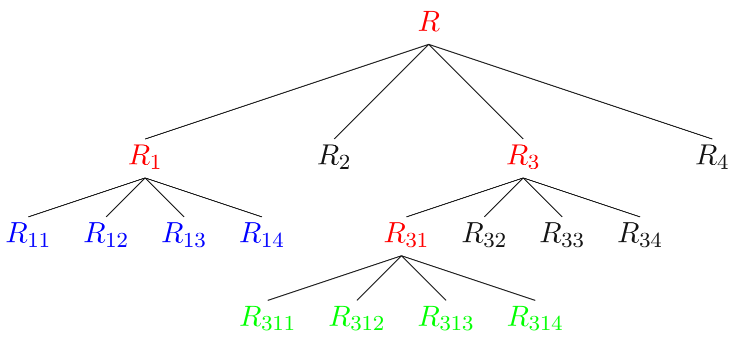


Figure 4.3: Here the subtree $\mathcal{B}_0$ is the tree without the blue and green nodes. Note that R_2, R_4, R_{32}, R_{33} and R_{34} are also leaves but they have a sibling which is not a leaf (red nodes are the interior nodes). The green nodes are the „bottom“ of the tree.

We prove (4.14) by induction over the distance *dist* from the root node to the bottom. If $dist = 1$ the statement is simply already given by (4.13). For the induction step we assume (4.14) holds for $dist = k$ and show it for $dist = k+1$. Define $\mathcal{B}_{00}$ as the subtree obtained from $\mathcal{B}_0$ by removing all nodes which have the largest distance to the root R which by our assumption is $k+1$. This means the tree $\mathcal{B}_{00}$ has only nodes which have at most distance k to the root. Therefore (4.14) holds true for $\mathcal{B}_{00}$. Applying (4.13) to all nodes in $L(\mathcal{B}_{00}) \cap N(\mathcal{B}_0)$ leads to

$$\begin{aligned}
\widetilde{\eta}(R)^2 &\geq \sum_{R' \in L(\mathcal{B}_{00})} \widetilde{\eta}(R')^2 + \tau|N(\mathcal{B}_{00})| \\
&= \sum_{R' \in L(\mathcal{B}_{00}) \cap N(\mathcal{B}_0)} \widetilde{\eta}(R')^2 + \sum_{R' \in L(\mathcal{B}_{00}) \setminus N(\mathcal{B}_0)} \widetilde{\eta}(R')^2 + \tau|N(\mathcal{B}_{00})| \\
&\geq \sum_{R' \in L(\mathcal{B}_0) \setminus L(\mathcal{B}_{00})} \widetilde{\eta}(R')^2 + \sum_{R' \in L(\mathcal{B}_{00}) \setminus N(\mathcal{B}_0)} \widetilde{\eta}(R')^2 + \tau|N(\mathcal{B}_{00})| + \tau|L(\mathcal{B}_{00}) \cap N(\mathcal{B}_0)| \\
&= \sum_{R' \in L(\mathcal{B}_0)} \widetilde{\eta}(R')^2 + \tau|N(\mathcal{B}_0)|.
\end{aligned}$$

In the last equality we used $L(\mathcal{B}_0) = (L(\mathcal{B}_0) \setminus L(\mathcal{B}_{00})) \cup (L(\mathcal{B}_{00}) \setminus N(\mathcal{B}_0))$ and $|N(\mathcal{B}_0)| = |N(\mathcal{B}_{00})| + |L(\mathcal{B}_{00}) \cap N(\mathcal{B}_0)|$. This concludes the induction for showing inequality (4.14).

We aim for a lower bound on the sum on the left-hand side of (4.14). Note that there are two types of nodes in the index set of the sum. The first type are those nodes which are interior nodes in B_R (R_1 and R_3 in Figure 4.3). A node R' of this type is further refined by the algorithm and hence satisfies $\widetilde{\eta}(R')^2 \geq \tau$. The tree $\mathcal{B}_0$ is obtained from the tree B_R by removing leaves, which implies $N(B_R) \subset Nodes(\mathcal{B}_0)$. Since interior nodes of $\mathcal{B}_0$ are also interior nodes in B_R there are exactly $|N(B_R)| - |N(\mathcal{B}_0)|$ many nodes of this type. A leaf R' of the second type satisfies $\widetilde{\eta}(R')^2 \geq 0$ because of Remark 3. Note that all nodes of the second type have a sibling of the first type as they would have otherwise been removed in the construction of $\mathcal{B}_0$.

This leads to

$$\tau\left(|N(B_R)| - |N(\mathcal{B}_0)|\right) \leq \sum_{R' \in L(\mathcal{B}_0)} \widetilde{\eta}(R')^2.$$

Finally, we get a lower estimate for the left-hand side in (4.14),

$$\tau|N(B_R)| = \tau\left(|N(B_R)| - |N(\mathcal{B}_0)|\right) + \tau|N(\mathcal{B}_0)| \leq \sum_{R' \in L(\mathcal{B}_0)} \widetilde{\eta}(R')^2 + \tau|N(\mathcal{B}_0)| \leq \widetilde{\eta}(R)^2,$$

which is the inequality we wanted to prove. □

Lemma 4.2.8. *Let $\tau > 0$ and let the tree B correspond to the output of* **Algorithm 2** *with root node R_0. If B' is a subtree of B with root node R_0 then*

$$\sum_{R' \in L(B')} \eta(R')^2 \leq \sum_{R' \in L(B')} \widetilde{\eta}(R')^2 + \tau|N(B')|. \tag{4.15}$$

Proof. If $|N(B')| = 0$, then $\text{Nodes}(B') = \{R_0\}$ and (4.15) is clear because $\alpha = 0$ for the root and hence $\eta(R_0)^2 = \widetilde{\eta}(R_0)^2$. Therefore we only need to consider the case with $|N(B')| \geq 1$. The definition of $\widetilde{\eta}$ in (4.10) leads to

$$\sum_{R' \in L(B')} \eta(R')^2 = \sum_{R' \in L(B')} \widetilde{\eta}(R')^2 + \sum_{R' \in L(B')} \alpha(R').$$

We complete the proof by showing that

$$\sum_{R' \in L(B')} \alpha(R') \leq \tau|N(B')| \tag{4.16}$$

via induction on $|N(B')|$. For $|N(B')| = 1$, the tree B' only consists of R_0 and its children, with each child being a leaf of the tree. By definition of δ in (4.8), we get $\delta(R) \leq \tau$ for any node R. Together with $\alpha(R_0) = 0$ and the definition of α in (4.9), we obtain for the children R'

$$\alpha(R') = \lambda(R')(\alpha(R_0) + \delta(R_0)) = \lambda(R')\delta(R_0) \leq \tau\lambda(R'). \tag{4.17}$$

Summing up over all children and recalling that the sum of the λ over all children of a node is equal to 1 gives (4.16). Let us assume that (4.16) holds for any tree B' with $|N(B')| = k$ and root node R_0. We consider a tree B' with $|N(B')| = k+1$ and root node R_0. Let $R \in \text{Nodes}(B')$ with children $R_1, .., R_4$ being leaves of B'. The tree B'' obtained by removing the children of R in B', and hence turning the interior node R into a leaf, satisfies $|N(B'_{R_0})| = k$ and therefore (4.16) by assumption. We compare (4.16) for B'' and for B'

$$\begin{aligned}
\sum_{R' \in L(B')} \alpha(R') &= \sum_{R' \in L(B'')} \alpha(R') - \alpha(R) + \sum_{i=1}^{4} \alpha(R_i) \\
&\leq \tau|N(B'')| - \alpha(R) + \sum_{i=1}^{4} \alpha(R_i) \\
&= \tau(|N(B')| - 1) - \alpha(R) + \sum_{i=1}^{4} \alpha(R_i).
\end{aligned}$$

Therefore it suffices to show that

$$\sum_{i=1}^{4} \alpha(R_i) \leq \alpha(R) + \tau. \tag{4.18}$$

Again, simply by using the definition of α and $\delta(R) \leq \tau$, we get

$$\begin{aligned} \alpha(R_i) &= \lambda(R_i)(\alpha(R) + \delta(R)) \\ &\leq \lambda(R_i)(\alpha(R) + \tau) \end{aligned}$$

for all $i \in \{1, .., 4\}$. Summing up over the children and using once again that $\sum_{i=1}^{4} \lambda(R_i) = 1$ shows (4.18) which concludes the proof. □

Lemma 4.2.9. *Let $B,\ B' \in \text{refine}(\mathcal{T}_0)$ be two trees and for $R \in L(B')$ let B_R be the subtree of B which consists of R and all its descendants. Then there holds*

$$\sum_{R \in L(B')} |N(B_R)| \geq |N(B)| - |N(B')|. \tag{4.19}$$

Proof. We consider 3 cases that can occur for an interior node of the two trees:

Case 1: $R \in N(B) \wedge R \notin N(B')$. In this case there has to be a leaf R' of B' such that either $R' = R$ or R is a descendant of R' in B. Either way, there holds $R \in B_{R'}$.

Case 2: $R \in N(B) \wedge R \in N(B')$. In this case R is a predecessor of a leaf in B', i.e. there exists a leaf $R^* \in B'$, such that R^* is a descendant of R. That implies that for any $R' \in B'$ the node R is not in $B_{R'}$.

Case 3: $R \notin N(B)$. Without loss of generality we can assume that this case doesn't occur, because it would only lower the right-hand side of estimate (4.19).

Altogether we see that any node which increases the right-hand side of (4.19) by 1 also increases the left-hand side by 1 which concludes the proof. □

Now we can prove Theorem 4.2.6.

Proof of Theorem 4.2.6. The statement about the number of calculations for η is clear as it has to be calculated for each node of B which is equal to one (the root node) plus $4|N(B)|$ as each interior node gets refined into 4 children, leading to 4 new nodes to compute η for.

Since the final mesh $\mathcal{T} = \mathcal{T}(B)$ corresponds to the leaves of B the global error can be written as

$$E(\mathcal{T}) = \sum_{R \in L(B)} \eta(R)^2. \tag{4.20}$$

We use Lemma 4.2.8 to see

$$E(\mathcal{T}) \leq \sum_{R \in L(B)} \widetilde{\eta}(R)^2 + \tau|N(B)| \leq 5\tau|N(B)|. \tag{4.21}$$

The second estimate above follows from $\widetilde{\eta}(R)^2 \leq \tau$ for each rectangle $R \in \mathcal{T}$ in the final mesh $\mathcal{T}$ of **Algorithm 2** and $|L(B)| \leq 4|N(B)|$ because every leaf is the child of an interior node.

Let B' be a tree obtained by at most m refinements and $\mathcal{T}'$ be the mesh corresponding to the leaves of B' such that $E(\mathcal{T}') = E_m$ is satisfied. Denote by B_R the subtree of B consisting of R and all its descendants. (B_R is empty if $R \notin B$.) We apply Lemma 4.2.7 to each such B_R and obtain

$$E_m = \sum_{R \in L(B')} \eta(R)^2 \geq \tau \sum_{R \in L(B')} |N(B_R)|. \tag{4.22}$$

Lemma 4.2.9 shows that the sum on the right is at least $|N(B)| - |N(B')|$. Since the number of refinements is equal to the number of interior nodes, the definition of m implies $|N(B')| \leq |N(B)|/2 = m$ leading to $-|N(B')| \geq -|N(B)|/2$. Using this to further estimate the right-hand side in (4.22) we get

$$E_m \geq \tau(|N(B)| - |N(B')|) \geq \tau|N(B)|/2.$$

Multiplying this equation with 10 and combining it with (4.21) leads to

$$E(B) \leq 5\tau|N(B)| \leq 10E_m,$$

which concludes the proof. $\square$

Note that with Theorem 4.2.6 we have obtained a near-optimality result in terms of number of refinements done in **Algorithm 2** but unfortunately we cannot guarantee that at least a certain amount of refinements are done. Inequality (4.13) shows that refining a rectangle leads to a reduction of the modified error $\widetilde{\eta}$ by at least τ. However, this strategy leads to little control over the global error E. Since we are interested in producing a mesh on which the approximate image has a smaller global error E than some prescribed tolerance we need to come up with another algorithm. Furthermore, we also want an algorithm for which we can prove a near-optimality result without the constraint that the constant of the refinement property (4.3) is (less or equal to) 1, because we have established (4.3) only for the constant equal to 16.

4.3 An algorithm with prescribed global error threshold

In this section we propose an algorithm which approximates a given image up to a given tolerance with near-optimal complexity in the sense of Definition 2.4.3. For our new algorithm we start again by modifying the error functional which drives the algorithm.

4.3.1 Another modification of the error functional

We once again denote the modified error functional with $\widetilde{\eta}$ and set $\widetilde{\eta}(R_0) = \eta(R_0)$ for the initial rectangle R_0. With $\widetilde{\eta}(R)$ already defined for some $R \in B^*$ we set for the the children $R_i,\ i = 1, .., 4$ of R

$$\widetilde{\eta}(R_i)^2 := q(R),$$

where

$$q(R) := \frac{\sum_{i=1}^{4} \eta(R_i)^2}{\eta(R)^2 + \widetilde{\eta}(R)^2} \widetilde{\eta}(R)^2.$$

This means $\widetilde{\eta}$ is constant among siblings. We define penalty terms

$$p(R_i) := \frac{\eta(R_i)^2}{\widetilde{\eta}(R_i)^2} = \frac{\eta(R_i)^2}{q(R)}$$

which describe how $\widetilde{\eta}^2$ differs from η^2. We calculate

$$\sum_{i=1}^{4} p(R_i) = \frac{\sum_{i=1}^4 \eta(R_i)^2}{q(R)} = \sum_{i=1}^{4} \eta(R_i)^2 \frac{\eta(R)^2 + \widetilde{\eta}(R)^2}{(\sum_{i=1}^4 \eta(R_i)^2)\widetilde{\eta}(R)^2} = p(R) + 1, \tag{4.23}$$

which is the main property we will need from $\widetilde{\eta}$.

Lemma 4.3.1. *Let $B_{R'} \subset B^*$ be a tree with root node R' and $\mathcal{T} = \mathcal{T}(B)$ be the mesh corresponding to B. Then there holds*

$$\sum_{R \in L(B_{R'})} p(R) = p(R') + |N(B_{R'})|. \tag{4.24}$$

In the special case $R' = R_0$ there holds

$$\sum_{R \in L(B_{R_0})} p(R) = 1 + |N(B_{R_0})|. \tag{4.25}$$

Proof. We show (4.24) by induction on the number of interior nodes. If $|N(B_{R'})| = 1$, then the tree $B_{R'}$ only consists of R' and its children. Therefore, the statement is just the identity (4.23). We now assume that the statement holds for $|N(B_{R'})| = k$ and deduce that it also holds for $|N(B_{R'})| = k + 1$.

Let a tree $B_{R'} \subset B^*$ be given with $|N(B_{R'})| = k + 1$. Let R^* be an interior node, such that all its children $R_1, .., R_4$ are leaves. We now consider the subtree $B_0 \subset B_{R'}$ obtained by removing the children of R^*, which makes R^* a leaf of B_0. Therefore the number of interior nodes in B_0 is k. With (4.23), we obtain

$$\sum_{R \in L(B_0)} p(R) = \sum_{R \in L(B_{R'})} p(R) + p(R^*) - \sum_{i=1}^{4} p(R_i) = \sum_{R \in L(B_{R'})} p(R) - 1.$$

By assumption B_0 satisfies (4.24) leading to

$$\begin{aligned}\sum_{R \in L(B_{R'})} p(R) = \sum_{R \in L(B_0)} p(R) + 1 &= p(R') + |N(B_0)| + 1 \\ &= p(R') + |N(B_{R'})|,\end{aligned}$$

which concludes the induction. The identity (4.25) follows immediately from (4.24), because for the root node R_0 of the master tree, there holds $\widetilde{\eta}(R_0) = \eta(R_0)$ and hence $p(R_0) = 1$. □

4.3.2 Algorithm 3

The new error functional $\widetilde{\eta}$ induces an algorithm which creates a sequence of trees $B_j,\ j = 1,\ 2, ..$ (more precisely: the algorithm creates meshes corresponding to a sequence of trees) as follows:

Algorithm 3 Another modified L^2 error functional with DCT approximation

```
Input: image component $I^Y$
(1) Set initial mesh $\mathcal{T} = \mathcal{T}_0$
For i=1,2,..
(2) Compute $I^Y_{app,\mathcal{T}}$
(3) Compute modified error $\widetilde{\eta}(R)^2$ for all $R \in \mathcal{T}$
(4) Set $\rho := \max_{R' \in \mathcal{T}} \widetilde{\eta}(R')^2$
(5) Mark $R$ if $\widetilde{\eta}(R)^2 = \rho$ for all $R \in \mathcal{T}$
(6) Refine marked elements, set $\mathcal{T}_i$ and $\mathcal{T}$ to the obtained mesh
End (of for-loop)
(7) return sequence $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, ..$
```

Note that in order for the algorithm to terminate at some point, we will need some threshold parameter. However, before we do that, we analyze the performance of **Algorithm 3** by showing the following theorem, which is similar to Theorem 4.2.6.

Theorem 4.3.2. *There is a universal constant $C > 0$ such that for each $i = 0,\ 1, ..$ the mesh $\mathcal{T} = \mathcal{T}_i$ given by* **Algorithm 3** *satisfies*

$$E(\mathcal{T}) \leq CE_m \tag{4.26}$$

whenever $m \leq n/10$ with $n := |N(B(\mathcal{T}))|$. To create $\mathcal{T}$ the algorithm uses at most $C(n+1)$ arithmetic operations and computations of η.

Proof. We split the proof into five steps:

Step 1. We consider any $j = 0,\ 1, ...$ such that $\mathcal{T} = \mathcal{T}_j$ satisfies $|N(B(\mathcal{T}))| =: n \geq 10$ and the best mesh $\mathcal{T}^*$ with $|N(B(\mathcal{T}^*))| \leq n/10$, i.e.,

$$E(\mathcal{T}^*) = \min_{\substack{B \subset B^* \\ |N(B)| \leq n/10}} E(\mathcal{T}(B)).$$

We set $t := \min_{R \in N(B(\mathcal{T}))} \widetilde{\eta}(R)^2$. Let $\Lambda := \left\{R \in N(B(\mathcal{T}))\colon \widetilde{\eta}(R)^2 = t\right\}$ be the collection of interior nodes where $\widetilde{\eta}^2$ attains the minimal value t. We will derive an upper bound for $E(\mathcal{T})$ by splitting $B(\mathcal{T})$ into $B(\mathcal{T}) = \Gamma_0 \cup \Gamma_1$ where $\Gamma_1 := \cup_{R \in \Lambda} B_R$ with $B_R \subset B(\mathcal{T})$ being the subtree consisting of R and all its descendants. The tree $\Gamma_0 \subset B(\mathcal{T})$ is obtained by deleting all the proper descendants of all $R \in \Lambda$. For a leaf R of $B(\mathcal{T})$, there are two possibilities: 1)$R \in \Lambda$ or R has a predecessor $R' \in \Lambda$ and 2)Neither R nor any of its predecessors are an element of Λ. In the first case there holds $R \in L(\Gamma_1) := \cup_{R \in \Lambda} L(B_R)$, in the second there holds $R \in L(\Gamma_0) \backslash \Lambda$. Therefore, we obtain that $L(\mathcal{T}) = L_0 \cup L_1$ where $L_1 = L(\Gamma_1)$ and $L_0 = L(\Gamma_0) \backslash \Lambda$. We obtain

$$E(\mathcal{T}) = \sum_{R \in L_0} \eta(R)^2 + \sum_{R \in L_1} \eta(R)^2 =: \Sigma_0 + \Sigma_1.$$

We bound each of the sums on the right-hand side separately. First we note that from the refinement property (4.3), it follows that for every $R \in \Lambda$, we have

$$\sum_{R' \in L(B_R)} \eta(R')^2 \leq C_0 \eta(R)^2 = C_0 \widetilde{\eta}(R)^2 p(R).$$

This gives

$$\Sigma_1 = \sum_{R\in\Lambda}\sum_{R'\in L(B_R)} \eta(R')^2 \le C_0 \sum_{R\in\Lambda} \widetilde{\eta}(R)^2 p(R).$$

Together with $\Sigma_0 = \sum_{R\in L_0} \widetilde{\eta}(R)^2 p(R)$ this leads to

$$\Sigma_0 + \Sigma_1 \le C_0 \sum_{R\in L(\Gamma_0)} \widetilde{\eta}(R)^2 p(R). \tag{4.27}$$

Step 2. Next we show that

$$\widetilde{\eta}(R)^2 \le t, \ \forall R \in L(\Gamma_0). \tag{4.28}$$

For $R \in \Lambda$, there holds equality by the definition of Λ. For $R \in L(\Gamma_0)\backslash\Lambda$ we consider the parent R_p of R. The parent of any node in a tree is obviously an interior node. Note that $R_p \notin \Lambda$, since there would hold $R \in L(\Gamma_1)$ otherwise. By definition of t, we obtain

$$\widetilde{\eta}(R_p)^2 > \widetilde{\eta}(R')^2 = t$$

for any $R' \in \Lambda$. This means R_p gets refined before the elements of Λ do. Consequently, when the elements of Λ get refined (which eventually happens since Λ is a subset of interior nodes) R is already part of the tree. The marking strategy in the algorithm then implies that

$$\widetilde{\eta}(R')^2 = t > \widetilde{\eta}(R)^2$$

for any $R' \in \Lambda$, which shows (4.28). Using (4.28) in (4.27) together with (4.25) we obtain

$$E(T) = \Sigma_0+\Sigma_1 \le C_0 \sum_{R\in L(\Gamma_0)} \widetilde{\eta}(R)^2 p(R) \le C_0 t(1+|N(\Gamma_0)|) \le C_0 t(1+|N(B(\mathcal{T}))|). \tag{4.29}$$

Step 3. We now show that $t(1+N(B(\mathcal{T}))) \le CE_m$. To that end, recall the best approximating mesh $\mathcal{T}^*$ from the beginning of the proof. Let L^* be the collection of all leaves $R^* \in L(\mathcal{T}^*)$ which are interior nodes of $\mathcal{T}$ and for each of these R^* let B_{R^*} be the tree consisting of all $R \in B(\mathcal{T})\backslash L(B(\mathcal{T}))$ such that R is a descendant of R^*. For every leaf $R \in L(B_{R^*})$, we have that R is in the interior of $B(\mathcal{T})$ and consequently $\widetilde{\eta}(R)^2 \ge t$ by definition of t. We apply (4.3) to obtain

$$t \sum_{R\in L(B_{R^*}))} p(R) \le \sum_{R\in L(B_{R^*}))} p(R)\widetilde{\eta}(R)^2 = \sum_{R\in L(B_{R^*}))} \eta(R)^2 \le C_0 \eta(R^*)^2. \tag{4.30}$$

Summing up (4.30) over all $R^* \in L^*$ and using (4.24) leads to

$$tM \le t \sum_{R^*\in L^*}\sum_{R\in L(B_{R^*})} p(R) \le C_0 \sum_{R^*\in L^*} \eta(R^*)^2 \le C_0 E_m, \tag{4.31}$$

where

$$M := \sum_{R^*\in L^*} |N(B_{R^*})|. \tag{4.32}$$

Step 4. To conclude the proof, we shall show that

$$1 + |N(B(\mathcal{T}))| \leq 12M. \tag{4.33}$$

We let B' be the tree obtained from $B(\mathcal{T})$ by removing all of the leaves of $B(\mathcal{T})$. Consequently B' contains B_{R^*} for all $R^* \in L^*$. Since each subdivision results in four new rectangles and hence $\#B' \leq 1 + 4|N(B')|$ we get

$$1 + |N(B(\mathcal{T}))| \leq 1 + \#B' \leq 2 + 4|N(B')| \leq 6|N(B')|. \tag{4.34}$$

It remains to show that

$$|N(B')| \leq 2M. \tag{4.35}$$

First we note that the only interior nodes in B' that are not interior nodes in one of the B_{R^*} are those which are interior nodes in $B(\mathcal{T}^*)$. Therefore, there are at most $|N(B(\mathcal{T}^*))| \leq m$ of them. With our refinement rule and $|N(B(\mathcal{T}))| = n \geq 10m$ we obtain

$$|N(B')| \leq M + m \leq M + \frac{|N(B(\mathcal{T}))|}{10} \leq M + \frac{1 + 4|N(B')|}{10} \leq M + |N(B')|/2.$$

Subtracting the right term and multiplying the inequality by 2 shows (4.35) which concludes the proof of (4.33) and hence (4.26) is shown. The statement regarding the amount of computations of η follows immediately. $\square$

4.3.3 Algorithm 4

Based on **Algorithm 3** we can formulate an Algorithm which terminates after the global error is smaller than some prescribed tolerance μ. For this we simply do a check after each iteration.

Algorithm 4 Algorithm 3 with thresholding parameter

```
Input: image component $I^Y$, Tolerance $\mu > 0$
(1) Compute $\eta(R_0)^2 = \widetilde{\eta}(R_0)^2$, $R_0 \in \mathcal{T}_0$. Set $\mathcal{T} = \mathcal{T}_0$
(2) If $E(\mathcal{T}) \leq \mu$, then set $\mathcal{T}_\mu := \mathcal{T}$ and return $\mathcal{T}_\mu$.
(3) Compute $\rho := \max_{R' \in \mathcal{T}} \widetilde{\eta}(R')^2$ and refine all rectangles $R$ with $\widetilde{\eta}(R)^2 = \rho$. This generates a new mesh $\mathcal{T}'$. Set $\mathcal{T} = \mathcal{T}'$ and go to step (2).
Output: Mesh $\mathcal{T}_\mu$ with $E(\mathcal{T}_\mu) \leq \mu$.
```

Note that in step (3) of **Algorithm 4** the computing of ρ implies that an approximate image I^Y_{app} has to be computed on the current mesh and the local errors η and modified estimators $\widetilde{\eta}$ on each rectangle afterwards.

Corollary 4.3.3. *There are absolute constants $C_1, c_1 > 0$ such that for any error tolerance $\mu > 0$, the mesh $\mathcal{T}_\mu$ produced by* **Algorithm 4** *has the properties:*

(i) If $\widetilde{B} \subset B_*$ is any tree satisfying $E(\widetilde{B}) \leq c_1\mu$ then

$$|N(B(\mathcal{T}_\mu))| \leq C_1|N(\widetilde{B})|. \tag{4.36}$$

(ii) *The number of evaluations of η and the number of arithmetic operations in producing $\mathcal{T}_\mu$ is lower than $C_1(1 + |N(B(\mathcal{T}_\mu))|)$.*

Proof. Statement (ii) is clear, because we only have to compute η once for each occurring rectangle (i.e. every node in $B(\mathcal{T}_\mu)$ once, which takes a uniformly bounded amount of arithmetic operations on the various rectangles and the amount of rectangles is equal to $1 + 4|N(B(\mathcal{T}_\mu))|$.
To prove (i), we set $c_1 = \lambda(C + 1)^{-1}$ where C is the constant from Theorem 4.3.2 and $\lambda \in (0,\ 1/4)$ is a constant which will be specified later. We further define $\mathcal{T}'$ to be the mesh before the last refinement step in **Algorithm 3**, i.e. immediately before $\mathcal{T} := \mathcal{T}_\mu$ is obtained. Consequently, there holds $E(\mathcal{T}') > \mu$. Finally, we set $\mathcal{T}^*$ to be a mesh with minimal global error after $m := |N(B(\widetilde{\mathcal{T}}))| = |N(B(\mathcal{T}^*))|$ refinements. We distinguish between three cases:

Case 1 $E(\mathcal{T}) \geq \lambda\mu$. We use Theorem 4.3.2 for $\mathcal{T}$. We have

$$E_m = E(\mathcal{T}^*) \leq E(\widetilde{\mathcal{T}}) \leq c_1\mu \leq (C + 1)^{-1}E(\mathcal{T}).$$

Therefore the theorem gives $m > |N(B(\mathcal{T}))|/10$ which is (i) in this case with $C_1 = 10$.

Case 2 $|N(B(\mathcal{T}'))| \geq \lambda|N(B(\mathcal{T}))|$. Now we start with the inequality

$$E(\mathcal{T}^*) \leq E(\widetilde{\mathcal{T}}) \leq c_1\mu \leq c_1E(\mathcal{T}') \leq (C + 1)^{-1}E(\mathcal{T}').$$

Applying Theorem 4.3.2 for $\mathcal{T}'$ gives $m > |N(B(\mathcal{T}'))|/10 \geq \lambda|N(B(\mathcal{T}))|/10$ which proves (i) in this case with $C_1 = 10\lambda^{-1}$.

Case 3 $(E(\mathcal{T}) < \lambda\mu)$ and $(|N(B(\mathcal{T}'))| < \lambda|N(B(\mathcal{T}))|)$. For always refining into 4 new rectangles and $\lambda < 1/4$ the second inequality of this case is never true, because the number of interior nodes of a tree increases at most by the number of leaves which is equal to $1+4|N(B(\mathcal{T}'))|-|N(B(\mathcal{T}'))| = 1+3|N(B(\mathcal{T}'))|$. Therefore we can choose for a given m a parameter $\lambda > 0$ small enough such that for all $m' \geq m$ this case never occurs with $\lambda \to 1/4$ as $m \to \infty$. We get finitely many smaller thresholds for λ when we consider $m' < m$ where we simply choose the smallest to guarantee that for every m this case doesn't occur. That the smallest threshold for λ is strictly greater than 0, we have to assume that the mesh $\mathcal{T}'$ doesn't only consist of the root node. But in this case the statement of the Corollary is trivial anyway. □

Remark 4. In the proof of Corollary 4.3.3 we essentially showed that Case 2 always occurs, which makes distinguishing between the cases redundant. The reason the proof is presented this way is that a more general version, i.e., instead of always refining into 4 new elements the number of new elements is merely bounded by some arbitrary $K \in \mathbb{N}$ and there may be more than one root node, can be proven this way with a more complicated Case 3 (which can actually occur then). See [BD04] Chapter 5 for details.

5 I_{app} with JPEG and BV-norm

5.1 Algorithm 5

In this chapter we substitute the L^2-norm with the norm of bounded variations or short BV-norm. The idea for doing this is to refine areas of the image where the error oscillates, e.g. a boundary between two different colors in the image, where it is likely to approximate at least one color rather badly if the mesh is still coarse around this boundary. The BV-norm for a function $u \in BV \subset L^1((0,h] \times (0,w])$, i.e., $u \in L^1((0,h] \times (0,w])$ and u has finite total variation, is defined by

$$\|u\|_{BV} := \|u\|_1 + V(u), \tag{5.1}$$

where

$$V(u) := \sup\left\{ \int_{(0,h]\times(0,w]} u(x)\mathrm{div}(\phi(x))dx : \phi \in C_c^1((0,h] \times (0,w], \mathbb{R}^n),\ \|\phi\|_\infty \leq 1 \right\}$$

denotes the total variation of u. For weakly differentiable functions $w \in L^1$, i.e., $w \in W^{2,1}$ the BV norm of w is given by

$$\|w\|_{BV} = \|w\|_1 + \|\nabla w\|_1.$$

However, the functions I, I_{app} we consider are only piecewise constant when we consider their extensions on the domain $(0,h] \times (0,w]$, which for simplicity we denote the same way. The extensions are given by

$$I_{(app),ext.}(x,y) = I_{(app)}(\mathrm{ceil}(x), \mathrm{ceil}(y)).$$

For these functions the second term on the right-hand side of (5.1) turns into a sum over all jumps of $I_{(app)}$. Therefore this sum can be indexed by the interior edges E° of the pixels in I and thus the BV norm of the approximation error $(I - I_{app})$ can be written as

$$\|I - I_{app}\|_{BV} = \|I - I_{app}\|_1 + \frac{1}{2}\sum_{e \in E^\circ} |(I - I_{app})(p_{e,1}) - (I - I_{app})(p_{e,2})|,$$

where $p_{e,1}$ and $p_{e,2}$ denote the two neighbouring pixels of the interior edge $e \in E^\circ$. The symmetry of the expression in the sum implies that it does not matter which neighbouring pixel is $p_{e,1}$ and which is the other. The factor $1/2$ arises from weighting a jump with the area of the picture it corresponds to, which is half the size of a pixel.

The estimators we use now are

$$\eta^2(R) = \|(I - I_{app})\|_{BV} \tag{5.2}$$

$$E(\mathcal{T}) = \sum_{R\in\mathcal{T}} \eta^2(R). \tag{5.3}$$

These estimators can be used to drive **Algorithm 4**:

Definition 5.1.1. **Algorithm 5** denotes the algorithm obtained by considering **Algorithm 4** but with the estimators defined in (4.2.3) being replaced by the estimators defined in (5.2) and in (5.3).

6 Efficient storage of I_{app}

In this chapter, we try to optimize the necessary storage space for I_{app} or in other words our goal is that the algorithms we come up with satisfy **Property 2**. Furthermore, we do not only want to find a good way to store I_{app} but also want to be able to reconstruct I_{app} fast from the stored data.

6.1 Data structures

We summarize which data structures are used to implement the objects occuring in the algorithms:

- An image I...
  ... is stored by a tensor of size $h \times w \times 3$, with h the height and w the width of the image. The matrix $(h, w, 1)$ corresponds to the Y component, the matrix $(h, w, 2)$ to the C_b component and the matrix $(h, w, 3)$ to the C_r component of the image.

- A rectangle R...
  ... is stored as a vector (fpy, fpx, w) with (fpy, fpx) being the coordinates of the top left pixel within an image and w the width of the rectangle.

- A mesh $\mathcal{T}$...
  ... is stored as a matrix of size $|\mathcal{T}| \times 3$, where each row corresponds to a rectangle of the mesh.

- The local parameters $\alpha, \delta, \eta^2, \widetilde{\eta}^2, d, \lambda$ on rectangles...
  ... are stored in a matrix M of size $|\mathcal{T}| \times 6$, where each row corresponds to the values of a rectangle.

- The parent relations of rectangles and the position (i.e., row index) of rectangles within M...
  ... are stored in a matrix of size $|\mathcal{T}| \times 2$, where $(i, 1)$ is the position (i.e., row index) within M of the rectangle corresponding to the $i-$th row of $\mathcal{T}$ and $(i, 2)$ is the position of its parent within M.

6.2 Compression of I_{app}

After computing an approximate image I_{app} for some image I of size $h \times w$ by any of the algorithms in the previous chapters, I_{app} is given by three matrices of size $h \times w$ (one for each of the three color components of the YC_bC_r basis). If we store I_{app} naively then we would have to store at least $3hw$ integers. Recall that there is a mesh $\mathcal{T}^X$ which corresponds

1	3	6	10	15	21	28	36
2	5	9	14	20	27	35	43
4	8	13	19	26	34	42	49
7	12	18	25	33	41	48	54
11	17	24	32	40	47	53	58
16	23	31	39	46	52	57	61
22	30	38	45	51	56	60	63
29	37	44	50	55	59	62	64

Figure 6.1: Visualizing the enumeration N of the matrix entries

to I_{app}^X for each component $X \in \{Y, C_b, C_r\}$. We enumerate the entries of a matrix of size 8×8 by a bijection N onto $\{1, .., 64\}$ by considering the diagonals from the bottom left to the top right. We set $N(1,1) = 1$ and continue inductively:

Let $N(k,l) = m$ be already defined. Then we set

$$\begin{aligned} N(k-1, l+1) &= m+1 && \text{if } k \neq 1 \wedge l \neq 8 \\ N(l+1, 1) &= m+1 && \text{if } k = 1 \wedge l \neq 8 \\ N(8, k+1) &= m+1 && \text{if } l = 8. \end{aligned}$$

The enumeration N of the matrix entries is visualized by Figure 6.2.

Compressing the Y component I_{app}^Y of I_{app} into some data that can be stored more efficiently is done by the procedure

- Compute $F^Y(R) := \text{round}(\text{TL}_R(\text{DCT}_R(I^Y|_R))./Q)$ for all $R \in \mathcal{T}$
- Store entries from $F^Y(R)$ in order of N in a vector S_I until the last non-zero entry $\forall R \in \mathcal{T}$.

Note that the discrete cosine transform returns a matrix of coefficients of basis functions, where the coefficients towards bottom and right correspond to basis functions of higher frequency. Typical images usually have small coefficients corresponding to basis functions of high frequency, which means that after rounding they become zero. This leads to a significant rate of compression of the image, i.e., storing F^Y requires significantly less storage space than just storing I_{app}^Y. Compressing the other components is done analogue.

We realize that in order to reconstruct I_{app} from S, we also need to store the width w of I_{app}, the ratio h/w of I_{app} with h being the height and the widths of the rectangles $R \in \mathcal{T}^X$, $X \in \{Y, C_b, C_r\}$. Furthermore, we need some entries a, b to know when to proceed to the next rectangle or color component and the location of the rectangles within the image. In order to solve the last issue, we introduce a total order on the rectangles R within a mesh $\mathcal{T}$. Recall that a rectangle R is given by the position of its top-left pixel and its width. Therefore we can simply order the top-left pixels of the rectangles.

Let $\mathcal{T}$ be a mesh and R, R' rectangles in $\mathcal{T}$. Denote by $(\text{fpy}_R, \text{fpx}_R)$ and $(\text{fpy}_{R'}, \text{fpx}_{R'})$ the position of the first pixels of R and R'. We define a total-order $\leq$ by

$$R < R' \quad :\Longleftrightarrow \quad (\text{fpy}_R < \text{fpy}_{R'}) \vee ((\text{fpy}_R = \text{fpy}_{R'}) \wedge (\text{fpx}_R < \text{fpx}_{R'}))\,.$$

We then order the rectangles in the mesh according to this total-order from lowest to highest, i.e. $R_i < R_j$ if and only if $i < j$, with $i, j \in \{1, .., |\mathcal{T}|\}$.

We consider the vector S_I for an approximate image I_{app} with corresponding meshes $\mathcal{T}^X$ consisting of rectangles R_i^X. $i = 1, .., n_X$ with $X \in \{Y, C_b, C_r\}$. Let us denote the number of coefficients to store for R_i^X by $k_{i,X}$. Then the vector S_I has the form

$$S_I = \begin{pmatrix} h/w \\ w \\ \text{width}(R_1^Y) \\ F^Y(R_1^Y)(1) \\ \vdots \\ F^Y(R_1^Y)(k_{1,Y}) \\ a \\ \text{width}(R_2^Y) \\ \vdots \\ \vdots \\ F^Y(R_{n_Y}^Y)(k_{n_Y,Y}) \\ b \\ \text{width}(R_1^{C_b}) \\ F^{C_b}(R_1^{C_b})(1) \\ \vdots \\ \vdots \\ \vdots \\ F^{C_r}(R_{n_{C_r}}^{C_r})(k_{n_{C_r},C_r}) \end{pmatrix}$$

The entries a are used to indicate that a new rectangle starts and the entries b are used to indicate that a new color component starts. Any integers that cannot occur for coefficients or widths of rectangles is suitable for either a or b. The absolute values of the coefficients are bound by $\|Q\|$ and widths are always positive, hence $a = -4096$ and $b = -8192$ will do.

6.3 Decompression of I_{app}

In this section we formulate a procedure to reconstruct the approximate image I_{app} from a corresponding vector S_I. For this we reconstruct each color component individually. For a rectangle $R \in \mathcal{T}^X$, $X \in \{Y, C_b, C_r\}$ we can compute $I_{app}^X|_R$ simply by

$$I_{app}^X|_R = \text{IDCT}_R(\text{EMB}_R(Q \odot F^X(R))),$$

where $\odot$ denotes pointwise multiplication. Note that $width(R)$ is stored in S_I and $height(R) = width(R)h/w = width(R)S_I(1)$ can be computed by entries of S_I.

The order of the rectangles introduced in the last section implies that the first pixel of the next rectangle to reconstruct is the top left pixel of the subimage consisting of the pixels which are not within already reconstructed rectangles. In order to not have to search the entire image for this pixel, we introduce a list L of candidates for this pixel which gets updated after reconstructing a rectangle. After the rectangle $R_i \in \mathcal{T}^X$ is reconstructed, we add the pixel p_i below the bottom left pixel of the rectangle and the pixel to the right of the top right pixel of the rectangle to L if they are not already part of a reconstructed rectangle and within the image. We then search L for the next first pixel, reconstruct R_{i+1} and delete the corresponding element from L. We can repeat this process until we have reconstructed the entire image. The described procedure to reconstruct I_{app} from S then looks like this:

```
• set $ratio = S(1)$ and width $w = S(2)$
• reconstruct $I_{app}^Y$:
    – set $L = \{(1,1)\}$, $k = 3$ and $F = zero(8,8)$
    – while $S(k) \neq b$ do
          set $c = 0$, $width = S(k)$, $height = width\ ratio$
          while $S(k+c+1) \neq a \wedge S(k+c+1) \neq b$
              set $F(N^{-1}(c+1)) = S(k+c+1)$
              set $c = c+1$
          end while
          (denote with $R$ a rectangle of size $height \times width$)
          set $M = \mathrm{IDCT}_R(\mathrm{EMB}_R(Q \odot F))$
          find the first pixel $(fpy, fpx)$ of the current rectangle within $L$
          set $I_{app}^Y(fpy : fpy + height, fpx : fpx + width) = M$
          delete $(fpy, fpx)$ from $L$ and add up to two new candidates to $L$
          set $k = k + c + 1$
          set $F = zero(8,8)$
      end while
• reconstruct $I_{app}^{C_b}$, $I_{app}^{C_r}$ the same way.
```

The function N^{-1} denotes the inverse of the enumeration of the previous section.

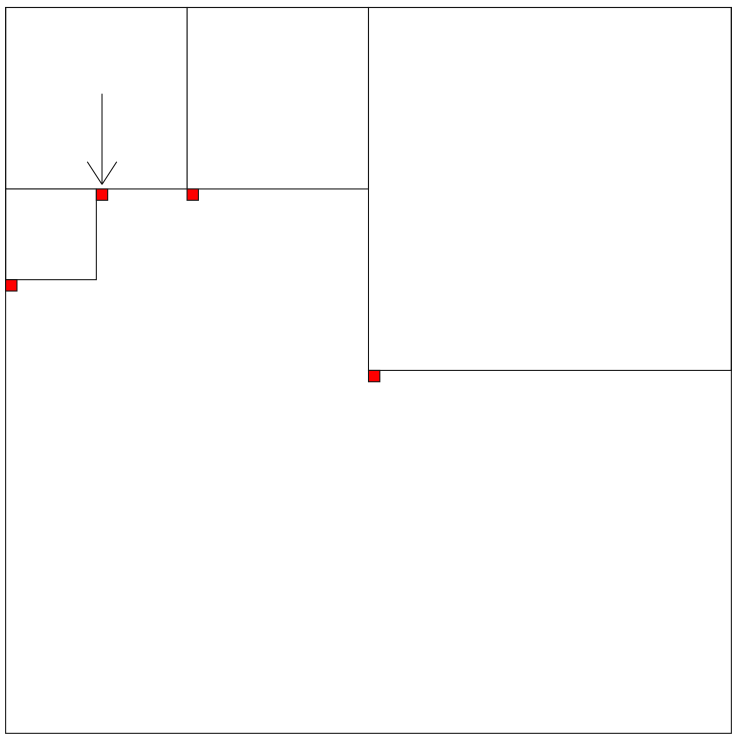


Figure 6.2: Visualizing the list L of candidates of first pixels for the next rectangle to reconstruct. The arrow points towards the next first pixel.

7 The performance of the algorithms

In this chapter we test the algorithms we came up with. **Algorithm 1** is not well suited for our purpose for reasons explained already earlier, **Algorithm 2** does not allow us to control the global error and **Algorithm 3** is just **Algorithm 4** without the threshold parameter introduced yet. Therefore, we only consider **Algorithm 4** and **Algorithm 5** in this chapter. We test these algorithms with the following test images:

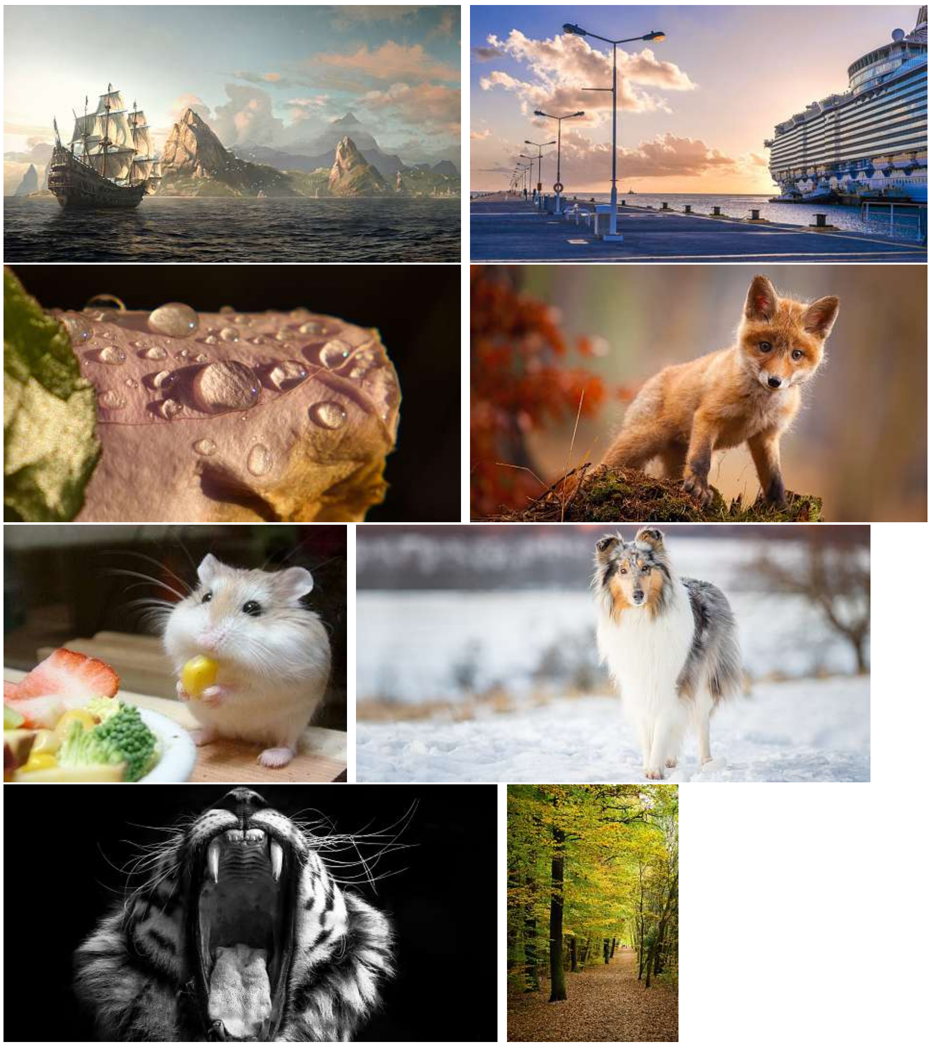
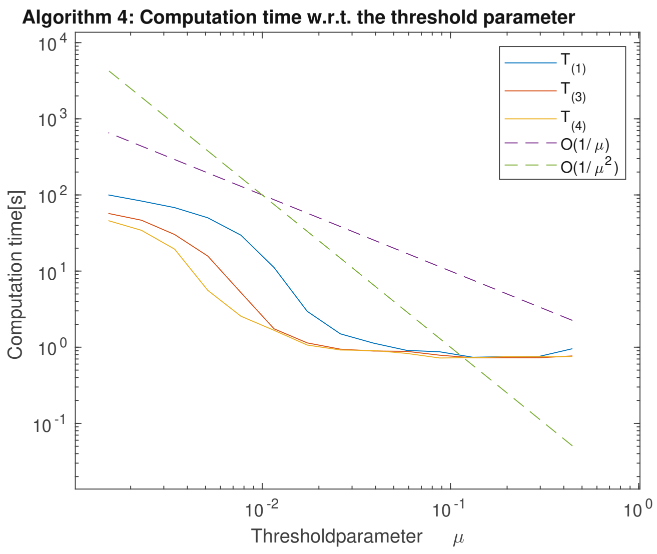
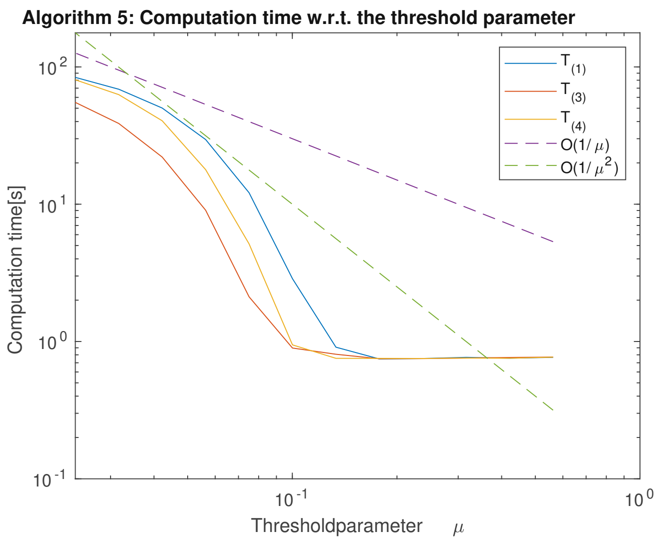


Figure 7.1: (1)Ship, (2)Harbor, (3)Leaf, (4)Fox, (5)Hamster, (6)Dog, (7)Tiger, (8)Forest (left to right, top to bottom)

7.1 Computation time with respect to the threshold parameter

In this section we track the computation time needed for certain values of the threshold parameter μ. For this we use images (or more precisely part of images) which are of size 1024×1024. Since the other test images are too small, we can only test the algorithms this way for test image (1), (3) and (4). (see Fig. 7.1).



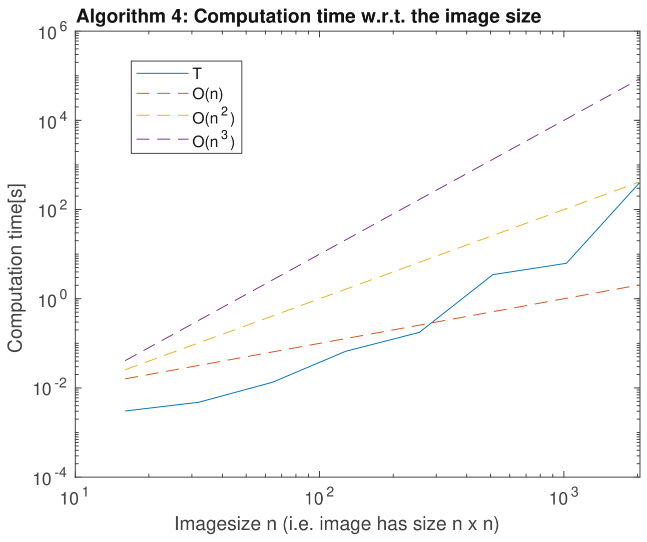


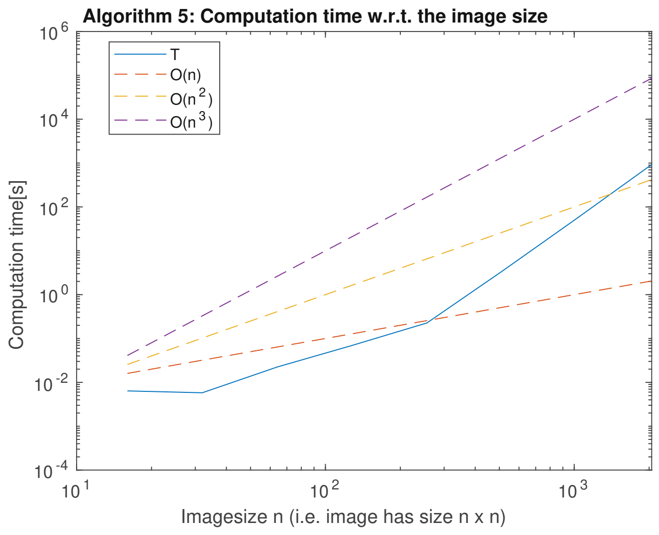
While the computation time tends to grow fast once the threshold parameter μ is small enough such that refining the initial rectangle becomes necessary, there seems to be a flattening later on. This can be explained by the restriction, that we do not refine rectangles which are already of size 8×8, leaving less eligible rectangles to be refined for decreasing μ.

7.2 Computation time with respect to the image size

Before we test the algorithms for different image sizes, we want to find a value for the threshold parameter μ such that the human eye cannot distinguish between the original image and the approximate image given by an algorithm. On the other hand we do not want to choose μ smaller than necessary for this, since we want to optimize computation time and storage space needed. Our choice of μ does not need to be the same for both algorithms.

We obtain a good choice for μ empirically by trying some values out and comparing the resulting approximate images to the original ones. For **Algorithm 4** a suitable choice of μ is given by $\mu = 0.005$. For **Algorithm 5** a suitable choice of μ is given by $\mu = 0.035$.

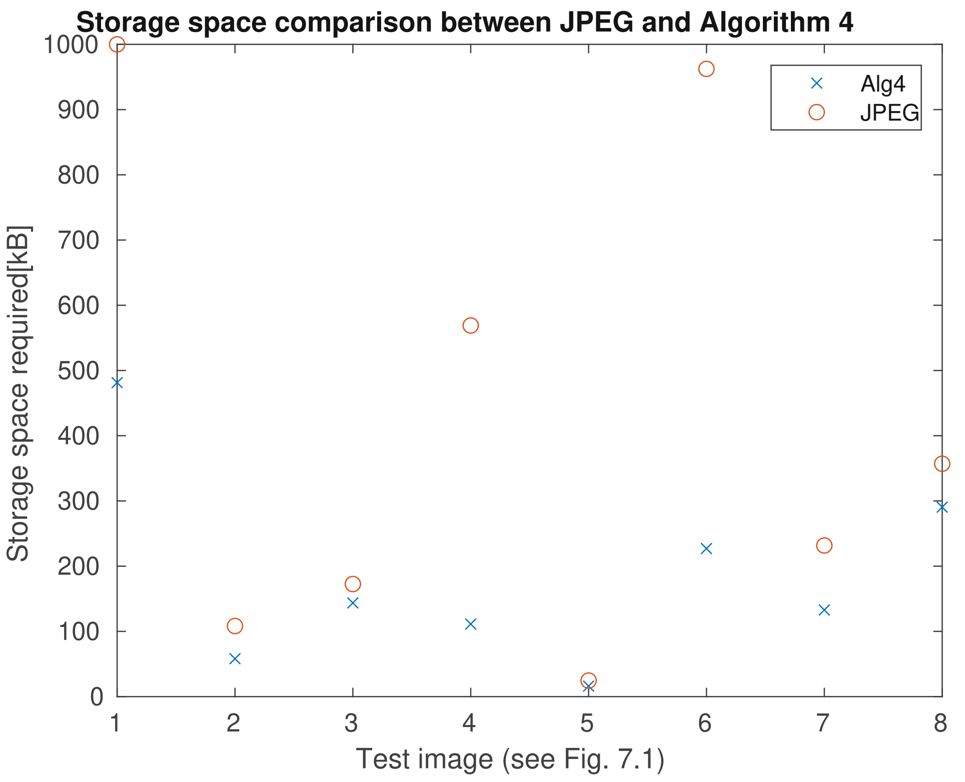


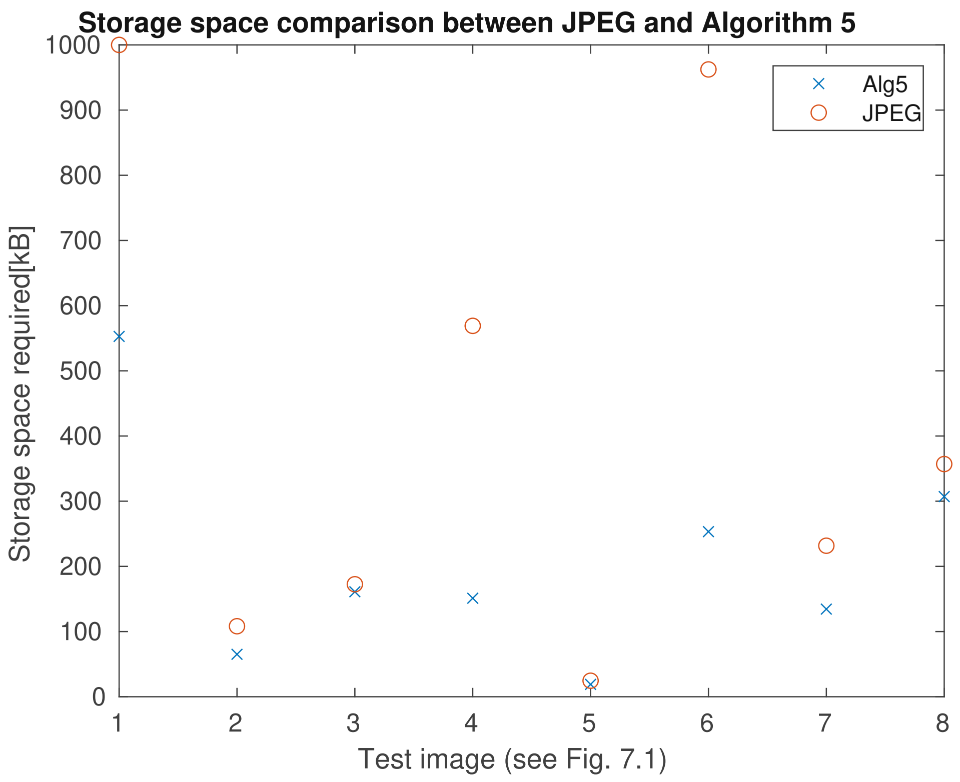


7.3 Comparison of storage space needed with JPEG

Before we make a comparison with the JPEG algorithm, we decrease the storage space our algorithms need further by compromising the vector S from Section 6.2. To this end, we use the compression method in [Hop22] to directly compress the resulting Matlab workspace variables we obtain as output from **Algorithm 4** and **Algorithm 5**. This method uses java gzip functions, which compresses the image size further by about 18-20%.

We test our algorithms with the same values of μ as in the last section.





We see that usually we can improve better upon larger JPEG Files than for smaller. Obviously, saving on storage space does not help much if the quality of the image is not (at least almost) preserved. Fig. 7.3 compares the outputs of the algorithms with the original test image 4 and shows that the quality loss is indeed very small (i.e., unnoticeable by the human eye).

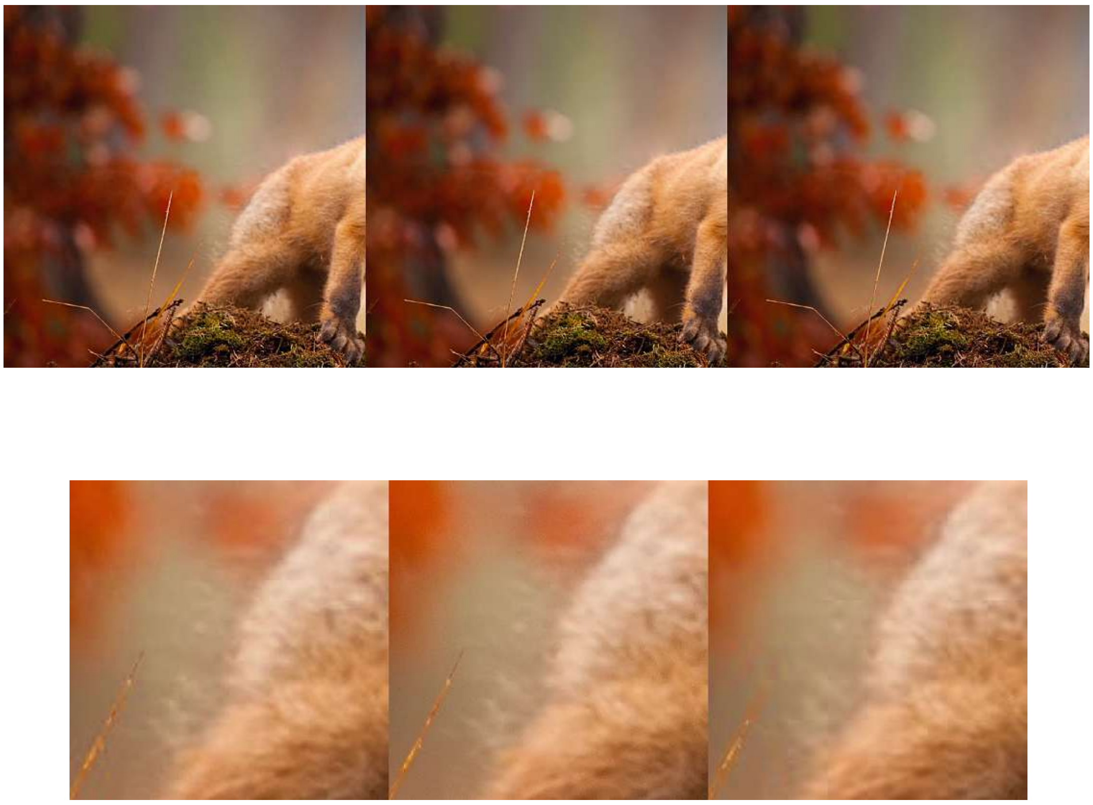


Figure 7.2: Comparison of the original image with the approximate images: Output of Algorithm 5(left), Original image (middle), Output of Algorithm 4(right). Even in close detail, there is barely any difference.

Bibliography

[BD04] Peter Binev and Ronald DeVore. Fast computation in adaptive tree approximation. *Numer. Math.*, 97(2):193–217, 2004.

[HC92] Richard F. Haines and Sherry L. Chuang. The effects of video compression on acceptability of images for monitoring life sciences experiments, 1992.

[HLN+18] Graham Hudson, Alain Léger, Birger Niss, István Sebestyén, and Jørgen Vaaben. JPEG-1 standard 25 years: past, present, and future reasons for a success. *Journal of Electronic Imaging*, 27(4):1 – 19, 2018.

[Hop22] Hopkins, Jesse. Compression routines. https://www.mathworks.com/matlabcentral/fileexchange/25656-compression-routines, 2022. [MATLAB Central File Exchange; Retrieved May 31, 2022].

[Wik21] Wikipedia contributors. Jpeg — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=JPEG&oldid=1040651992, 2021. [Online; accessed 8-September-2021].