YURI MEIBURG

DENSE-FIELD SKELETON-BASED IMAGE REPRESENTATIONS

rijksuniversiteit groningen

faculteit wiskunde en natuurwetenschappen

DENSE-FIELD SKELETON-BASED IMAGE REPRESENTATIONS

YURI MEIBURG

A Lossy Compression Technique For Monochrome Images

MSc. Computing Science and Visualization
Mathematics and Natural Sciences
Rijksuniversiteit Groningen

September 2011

Yuri Meiburg: *Dense-Field Skeleton-Based Image Representations,* A Lossy Compression Technique For Monochrome Images, MSc. Computing Science and Visualization, © September 2011

SUPERVISORS:
Prof. dr. A.C. Telea
dr. M.H.F. Wilkinson

LOCATION:
Groningen

September 2011

ABSTRACT

Skeletons are good 2D or 3D shape descriptors. However, so far they have only been used to encode simple, compact, closed boundaries such as isolines or isosurfaces. In this project, we study an extension of classical 2D multiscale skeletons to a new notion: dense field skeletons. Dense skeletons will be used to encode an entire 2D field, such as a monochrome image, into a scale-space of skeletons. By this method, operations such as image compression, progressive image encoding and/or transmission will be approached using the robust, well-proven, descriptive powers of skeleton features. Efficient storage is achieved by skeleton simplification and a state history based neighbour-coding scheme to encode skeleton-trees. The result is then further compressed using the Lempel-Ziv-Markov Chain Algorithm (LZMA). Reconstruction is done by inflating skeletons per layer, and smoothly interpolating the edges to reduce sharp transitions on high compression.

ACKNOWLEDGMENTS

This research project would not have been possible without the support of many people. I wish to express my deepest gratitude to my supervisor, Prof. Dr. A.C. Telea who was abundantly helpful and offered invaluable assistance, support and guidance, and my second supervisor Dr. M.H.F. Wilkinson for the endless (and fruitful) discussion halfway throughout the research. I would also like to thank all the fellow student-assistants, for the ample times I have asked for advice or verification.

I would like to thank Samantha, my father-in-law, and my parents; for their understanding and endless patience throughout the duration of my studies, and for providing the equipment to carry out this research.

Lastly I wishes to express my gratitude to all whom have helped during the countless times I asked for advice.

CONTENTS

LIST OF FIGURES

ACRONYMS

AFMM Augmented Fast Marching Method

BFS Breadth First Search

CCA Connected Component Analysis

DCT Discrete Cosine Transform

DFS Depth First Search

DT Distance Transform

FMM Fast Marching Method

JFIF JPEG File Interchange Format

JPEG Joint Photographic Experts Group

LZMA Lempel-Ziv-Markov Chain Algorithm

MAT Medial Axis Transform

MOS Mean Opinion Score

MSSIM Mean Structural Similarity Index

PGM Portable Grayscale Map

RIFF Resource Interchange File Format

SIR Skeletonal Image Representation

SPIFF Still Picture Interchange File Format

SSIM Structural Similarity Index

Part I

THESIS

1 INTRODUCTION

Image compression is essential in many application fields, and serves many purposes. For example it can be used to reduce file size, in order to make it fit on low capacity chips - such as present in modern biometric passports - or to store images in large quantities for databases. It can also be used to store only relevant features of an image (i.e. switching to a different representation), in order to optimize pattern matching techniques.

Classical image compression uses relatively low-level representations of the input data. Typically, images are subdivided into small blocks, and each block is compressed using signal-processing methods. For example the current camera standard (JPEG) compresses 8×8 blocks with a Discrete Cosine Transform (DCT), or Google's alternative "WebP", which encodes blocks of similar size using a prediction scheme.

Apart from the above, image analysis methods have looked into the extraction of relevant 'features' from images. Such features are meant to capture the most salient items present in an image. Different feature extraction techniques and methods exist. In shape processing skeletons, or medial axes, are an important class of features. They capture the symmetry, geometry, and topology of a binary shape.

However, skeletons cannot be directly used on continuous images, since they are designed to work on binary shapes. Yet, it is interesting to consider their usage for image simplification and/or image compression. For this, suitable methods must be found to (a) reduce a continuous, grayscale, image to a set of binary images; (b) efficiently and effectively extract skeletons from these images in order to capture the essential structure and shape of these images; and (c) use the extracted skeletons to reconstruct a simplified grayscale version of the original image.

In this thesis, we study the usage of skeletons for the task of image simplification and image compression. For this, we proceed as follows.

First, we reduce a grayscale image to a set of binary shapes. For this, we threshold an image for all possible intensities (assuming 8 bits), and the result of each threshold becomes a layer in a *threshold set*. We use a threshold set as this creates larger shapes, which are better to describe with skeletons. All layers which contribute very little to the image are discarded to keep the file size to a minimum.

Secondly, we encode each threshold set using the Medial Axis Transform (MAT). For this, we use an efficient and effective skeletonization method which can treat any 2D shapes, regardless of their

geometric complexity or genus [30], and also allows skeleton simplification in order to retain only the most salient aspects of the shapes to be encoded. The skeletons are first simplified using the saliency metric defined by Telea in [29]. Then small and unimportant objects are removed, as they are likely to be (a) perceived as noise; or (b) take up a lot of space whilst contributing very little to the reconstruction.

Thirdly, we encode the skeleton (and its distance transform) of each shape using an efficient method which attempts to minimize the amount of data stored. For this we make use of an efficient neighbour-chain encoding scheme, exploiting the sparse and elongated structure of skeletons, and the *continuous* variation in maximally inscribed disc radius.

The output of the three steps above is a compact representation of the initial image, which trades off the amount of image detail retained in the encoding against the amount of data used for the encoding. In the final step, we use this representation to reconstruct a simplified version of the initial image using a combination of distance transforms and blending on the encoded skeletons to achieve gradual changes in intensities. The result is a simplified rendering of the initial image.

The overall method exhibits some interesting similarities and differences with classical image compression methods such as JPEG. First, our global aim is similar: we want to encode a grayscale image in a compact representation whose (binary) size is smaller than the original image. In this sense, our encoding is also lossy. However, the types of artifacts our lossy encoding generates are different from JPEG: while, at high compression ratios, JPEG will create high-frequency ringing artifacts which follow a grid pattern, our encoding creates smooth, round, shapes, since simplified skeletons ignore sharp details on their shape boundaries. Secondly, we encode the image one grayscale level at a time (thus, in grayscale space), rather than one block at a time (thus, in geometric XY space).

Our method is a first attempt to explore the usage of multiscale skeletons for image simplification and compression. So far, the results of our method cannot compete with methods such as JPEG in terms of compression ratio and perceptual quality of the compressed image. However, the results obtained so far are promising and show that skeletons can be used for representing simplified images in a compact way, with different trade-offs and with a fundamentally different approach than classical image compression methods. Our approach, if extended, could be used for different image simplification and compression tasks, in cases where shapes in the image are central, e.g. shape-based image manipulation and editing, or nonphotorealistic rendering, as the occuring compression artefacts somewhat resemble paint-strokes.

The structure of this thesis is as follows. It is split in two parts: Part I is the actual thesis, and Part II is an explanation on how the

accompanying software functions. Part I consists of the following chapters: Chapter 2 provides some insight in the JPEG and WebP image compression techniques, necessary to understand the fundamental difference in approach, and provides some theoretical background in skeletons. Chapter 3 describes how to create segments from a grayscale image. Chapter 4 describes the full encoding process of these skeletons, up to and including the file format used. Chapter 5 describes how to reconstruct an image, and demonstrates a blending technique to reduce boundary artefacts. Chapter 6 contains examples on the various parameters used in our program, and their effects. It also contains a few example images, which show our current status. Sections 7, 8 conclude this thesis and discuss the obtained results.

2 RELATED WORK

This chapter aims to provide relevant background data and some insight in the methods used in this thesis. As we aim to create a lossy image compression, it is relevant to understand – at a coarse scale – the way the current leading image compression technique (JPEG) works, and how it fundamentally differs from our approach. We will also glance over a recent alternative: "*WebP*" in section 2.1.3. Section 2.2 will cover the Structural Similarity Index (SSIM), which we use to measure the quality of the output. Finally in section 2.3 we provide some background theory into skeletons.

2.1 LOSSY IMAGE COMPRESSION

In 1992 the Joint Photographic Experts Group (JPEG) introduced the *ISO 10918-1* standard, which describes a lossy compression method aimed at storing photographic images, aptly named: "JPEG" [8]. This standard defines how to convert an image into a stream of bytes, and a stream of bytes back to an image, and is based on the DCT. The original specification spans 186 pages, yet did not define a file standard [18]. Section 2.1.1 will coarsely describe how JPEG's most common encoding technique works. It is not possible to cover the full JPEG specification, as it covers four encoding techniques, two different entropy encodings, supports multiple numbers of bits per pixel, etc. Finally section 2.1.2 will briefly discuss the file formats.

2.1.1 *JPEG Encoding*

The JPEG standard defines more than one method to encode an image in a stream of bytes. The most straightforward method is the *Sequential* encoding method. This method consists of encoding an image single pass from top to bottom. A singe pass through an image (for one or more components) is called a *scan*, and is stored in a distinct data block [8].

The second method is *Progressive* encoding. This consists of multiple passes through the image, where each pass enhances the detail of the image. This is useful for sending large images over a slow data-connection, as an image can be shown in multiple stages, each stage increasing the resolution. One of the downsides of this method is that it is harder to implement, and thus not as widely supported as the sequential method.

The third method is called *Hierarchical* encoding. This method aims at getting smaller image files than other JPEG modes. Each pass consists of a few stages:

1. Down-sample image in both dimensions by a factor of 2 (e.g. 640×480 becomes 320×240)
2. Encode the smaller image using on of the other JPEG encoding techniques.
3. Decode and upsample the encoded image.
4. Compute the difference between the original and the upsampled image.
5. Encode this result using one of the other JPEG encoding techniques.

Even though the details of the aforementioned methods differ, they are all based on the same technique: Discrete Cosine Transform encoding. [1]

An image is first separated in the components Y, C_b and C_r, which respectively denote a luma component (brightness) and the blue-difference and red-difference components from the chroma color space. Since the human eye is noticeably better at perceiving differences in the brightness of an image than in the hue and color saturation of an image, the C_b and C_r are usually reduced in spatial resolution. This process is called "*Chroma subsampling*". The most common subsampling rate for JPEG is $4:2:0$, which means that the C_b and C_r are reduced to half the spatial resolution of the Y component. Other possible subsampling rates are $4:4:4$ (which is no downsampling), and $4:2:0$ (which is a reduction of a factor two in the horizontal direction) [33].

Each channel is then split into 8×8 blocks, and transformed using the DCT (as shown in eq. (2.1)).

$$G_{u,v} = \sum_{x=0}^{7} \sum_{x=0}^{7} \alpha(u)\alpha(v) g_{x,y} \cos\left[\frac{\pi}{8}(x+\frac{1}{2})u\right] \cos\left[\frac{\pi}{8}(y+\frac{1}{2})v\right] \tag{2.1}$$

where

- u is the horizontal spatial frequency, for the integers $0 \leqslant u \leqslant 8$.
- v is the vertical spatial frequency, for the integers $0 \leqslant v \leqslant 8$.
- $\alpha(u) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } u = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$ is a normalizing scale factor.
- $g_{x,y}$ is the pixel value at coordinates (x, y).
- $G_{u,v}$ is the DCT coefficient at coordinates (u, v).

The result of this transformation is an 8×8 matrix, with the DCT coefficients. After this transformation most information of the signal will be concentrated in the upper left corner of the matrix. The high

1 There is also a method for losless storage in the JPEG specification, but this is based on a predictive process in contrast to DCT. It is therefore not mentioned further in this thesis.

frequencies (towards bottom-right) are harder to perceive for the human eye. Therefore their precision is deemed less important. The resulting coefficients are usually multiplied by a "*Quantization Matrix*", and then rounded to the nearest integer. This is done such that the higher frequencies are rounded to zero, while the lower frequencies remain, and is called "*quantization*".

Given that the DCT is performed with enough precision, this rounding process is the *only* lossy step in the DCT compression. As the top-left corner now contains mostly non-zero values, and the bottom-right mostly zeroes, the entropy coding is performed in a "zig-zag" ordering (as shown in fig. 1), rather than left to right, top to bottom.

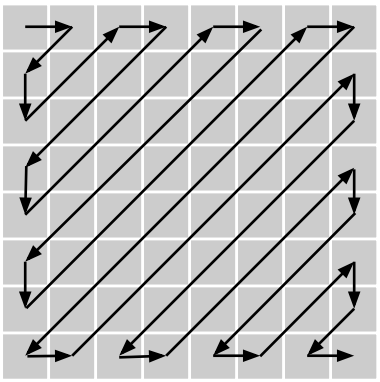


Figure 1: The zig-zag like pattern the entropy encoding follows for the DCT-coefficients (note: lower frequencies are stored top-left, and high frequencies are stored bottom-right)

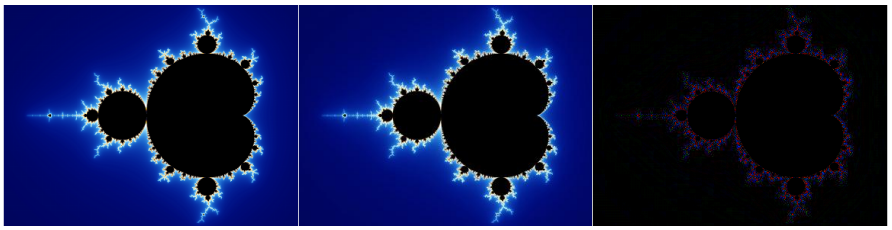
This result is then stored in one of the JPEG file formats. Further explained in section 2.1.2.

As high frequencies are rounded and stored, artefacts appear mostly at borders in the image. This effect can be seen in fig. 2.

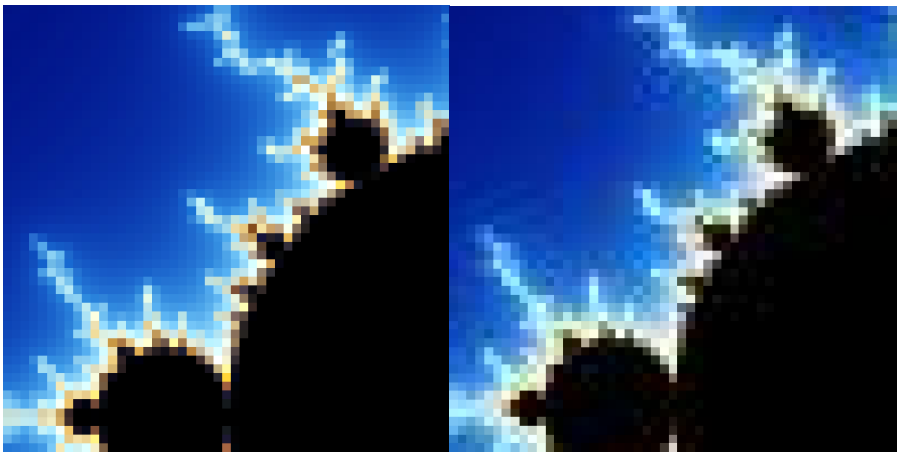
2.1.2 *The "JPG" File Format*

The current standard file format is the JPEG File Interchange Format (JFIF), which was developed by Hamilton [10] in 1992. In 1996 JPEG tried to fill the lack of a file format in their standard by releasing theStill Picture Interchange File Format (SPIFF) [9]. Despite SPIFF being the official standard file format, virtually all JPEG files are stored as JFIF. This is probably due to the fact that JFIF was released four years earlier, combined with the fact that the SPIFF standard is considered "too inclusive" [14]. As an example: The format is defined for 11 different color spaces. [2] This makes it hard for decoders to fully support the file format.

2 [14] incorrectly claims that SPIFF supports 13 color spaces. There are 15 accepted values, of which the value 2 denotes an unsupported value, and 5-7 are merely reserved values.



(a) Mandelbrot Original (b) Mandelbrot JPG (c) Difference (multiplied by 3)



(d) Mandelbrot Original Zoom (e) JPG Compression Zoom

Figure 2: Artefacts caused by JPEG compression.

2.1.3 *WebP*

Recently Google Inc. released a new lossy image compression format "*WebP*". In a large scale study of 900,000 web images, WebP images were 39.8% smaller than jpeg images of similar quality (using the SSIM [37]). The WebP technique is open-source, and is in essence an intra-frame encoded with the VP8 video compression format [3], stored in a Resource Interchange File Format (RIFF) container [15].

Similar to JPEG an image is divided in to blocks, and each block is encoded separately. VP8 uses a predictive encoding scheme. This means that given some datapoints, it predicts neighbours, and encodes the difference between the actual value and the prediction. These difference-values are smaller than the original values, and can thus be encoded more efficiently. Figure 3 shows the classification of an encoding block (VP8 supports 4×4 and 16×16).

C, A_i, L_i are either stored values, or values computed from previous frames (for video, WebP uses only intra-frame encoding), and X_{ij} are values which are approximated. Approximation of X occurs following eq. (2.2).

$$X_{i,j} = L_i + A_j - C \qquad (i, j \in \{0, 1, 2, 3\}) \tag{2.2}$$

C	A_0	A_1	A_2	A_3
L_0	X_{00}	X_{01}	X_{02}	X_{03}
L_1	X_{10}	X_{11}	X_{12}	X_{13}
L_2	X_{20}	X_{21}	X_{22}	X_{23}
L_3	X_{30}	X_{31}	X_{32}	X_{33}

Figure 3: VP8 Encoding Block

Although on average outperforming JPEG, there are a few disadvantages to WebP. It does not support a lossless mode, and only has support for $4:2:0$ chroma subsampling, while JPEG can also handle $4:2:2$ and $4:4:4$.

2.2 STRUCTURAL SIMILARITY INDEX

In order to measure the visual accuracy of our compression algorithm we need a *full-reference quality metric*; A metric which denotes the quality as one of the images being compared, provided the other image is regarded as of perfect quality. The simplest and most widely used full-reference quality metric is the mean squared error (MSE), computed by averaging the squared intensity differences of distorted and reference image pixels, along with the related quantity of peak signal-to-noise ratio (PSNR). These are appealing because they are simple to calculate, have clear physical meanings, and are mathematically convenient in the context of optimization. But they are not very well matched to perceived visual quality [36]. In 2004 Zhou and Bovik proposed the Structural Similarity Index (SSIM) [37]. SSIM is an objective method for measuring the similarity between two images and is specifically designed to match the quality as perceived by the human eye. It is computed over multiple $N \times N$ windows of an image and is defined as shown in eq. (2.3).

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.3}$$

with:

$\mu_{x,y}$	the average of x and y respectively
$\sigma^2_{x,y}$	the variance of x and y respectively
$c1 = (\kappa_1 L)^2$, $c2 = (\kappa_2 L)^2$)	two variables to stabilize the divison with weak denominator
L	the dynamic range of the pixel-values
$k_1 = 0.01, k_2 = 0.03$	by default.

A SSIM score lies in the interval $[-1..1]$, and a value of $\text{SSIM}(x, y) = 1$ is *only* possible for $x = y$. A mean SSIM (or MSSIM) is used to denote the quality of an image, and is computed according to eq. (2.4).

$$\text{MSSIM}(\mathbf{X}, \mathbf{Y}) = \frac{1}{M} \sum_{j=1}^{M} \text{SSIM}(\mathbf{x}_j, \mathbf{y}_j) \tag{2.4}$$

Where $\mathbf{X}, \mathbf{Y}$ denote the full images, and $\mathbf{x}_j, \mathbf{y}_j$ the content of the j-th window. Interpretation of this metric is best described using an actual human reference. It is therefore matched in a case study against a Mean Opinion Score (MOS). A MOS-score is the average score given by a group of human participants. Figure 4 shows the correlation between MOS and MSSIM in a case study of 29 high-resolution images (with over 300 distorted images), and between $13 - 25$ human participants per image. The MOS was graded on a continous linear scale with adjectives "Bad", "Poor", "Fair", "Good", "Excellent", and then filtered and rescaled to $[0..100]$ (0 denoting "Bad").

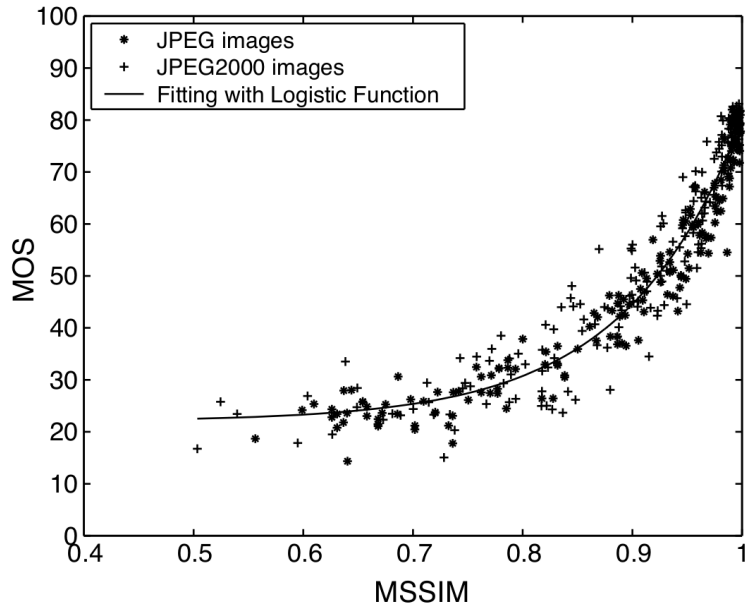


Figure 4: MSSIM versus MOS

2.3 SKELETONS

Skeletonization is a transformation of a component of a digital image Ω into a subset S of the original component, so that S is locally centered within Ω. The resulting reduced shapes appear to have some interesting properties, and thus have been utilized in a very diverse set of problems, such as: shape recognition [31, 39, 41], shape representation [19, 21, 22, 38], flow visualization [20], animation of computer models [24, 34, 35] and data compression [5, 12].

There are different methods to calculate a skeleton, and each method produces slightly different skeletons (according to slightly differing definitions). It is important to note that in this thesis the words *skeleton* and *medial axis* are used interchangeably to denote the *same* thing. Our definition of a skeleton follows Blum's [4], and is included below for clarity's sake:

SKELETON / MEDIAL AXIS

Let A be the object to be skeletonized, ∂A its boundary and $d(x, \partial A)$ the distance from x to A's boundary. The skeleton S is then defined as:

$$S = \{x \in A \mid \exists y, z \in \partial A, y \neq z, d(x, \partial A) = \|x - y\| = \|x - z\|\} \tag{2.5}$$

Or in words: $S(A)$ is the set of all centers of maximum discs inscribed in A.

MEDIAL AXIS TRANSFORM

The MAT is a full descriptor of an object, and can be used for reconstruction. The MAT consists of the locus of disks in the skeleton S, along with their radius. A set definition:

$$\mathrm{MAT}(A) = \{(x, d(x, \partial A)) : x \in S\} \tag{2.6}$$

RECONSTRUCTED OBJECT

An object Ω can be reconstructed from the union of all the discs in the skeleton S. Let $D(x, r)$ be the disk characterized by position x, of radius r. The reconstruction is defined by:

$$A = \bigcup \{D(x) : x \in \mathrm{MAT}(A)\} \tag{2.7}$$

2.3.1 *Computation of Skeletons*

There are various methods for computing a skeleton. *Morphological computation* is done by repeatedly morphologically thinning an object [13, 32, 40]. Each iteration boundary points which do not affect an objects topology are identified and removed. This process is repeated until no further points can be removed. This method is conceptually

rather straightforward, however implementations need intricate heuristics to ensure skeletal connectivity. Moreover, several thinning methods do not produce a true skeleton according to our definition [30].

Geometric methods compute the Voronoi diagram of a discrete polyline-like sampling of the boundary. The Voronoi diagram is the boundary's medial axis. Although these methods produce accurate connected skeletons, they are complex to implement, computationally expensive and require a robust boundary discretization [30]. The resulting skeletons are usually referred to as *straight skeletons*, and are first introduced in [2] for simple polygons. Later they have been refined for general polygonal figures [1].

The method we have used, and will describe in more detail is from a third family of methods, and is based on the Distance Transform (DT) of the objects boundary. The DT provides a description of the minimal distance to the boundary for each point in an object. Recent approaches of this family use the robust and simple to implement Fast Marching Method (FMM), first introduced in [26] as an $\mathcal{O}(n \log(n))$ algorithm to solve the Eikonal equation. The drawback of the DT is the difficulty in detecting singularities. Direct computation of singularities is numerically unstable and not trivial. In 2002 Telea and Van Wijk proposed the Augmented Fast Marching Method (AFMM), which overcomes this problem, and can produce skeletons of large 2D datasets in real-time [30]. This method is described in detail in section 4.1.

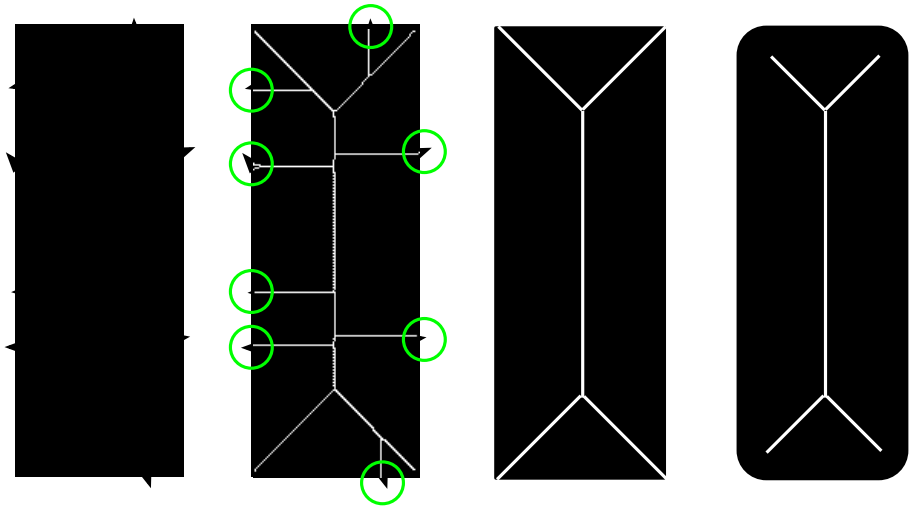
2.3.2 *Salience Skeletons*

To explain the essence of salience skeletons it is necessary to first provide a clear definition of *saliency*:

SALIENCE The salience (also called saliency) of an item – be it an object, a person, a pixel, etc. – is its state or quality of standing out relative to neighboring items.

As small pertubations are extremely common in measured data, there is a variety of algorithms to try and simplify objects, such that salient features are preserved [6, 7, 11]. Each with a different definition of features and noise. In 2011 Telea proposed a feature-preserving smoothing method based on *saliency skeletons* [29]. Saliency skeletons are defined as: skeletons which try to simplify / smooth objects, while preserving the salient features of this object. Thus removing branches in a skeleton, while trying to keep the object perceptually as equal as possible. An example is given in fig. 5.

In order to simplify a skeleton without removing perceptually important features, Telea defines a saliency-metric σ for each skeleton-point $x \in S(\Omega)$ of object Ω as in eq. (2.8).



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

Figure 5: Removal of non-salient points. a) is the original object, b) is the skeletonized version, with non-salient points marked in green, c) is the simplified object. Notice the perceptually similar object, whilst being reconstructed by a hugely simplified skeleton, d) Some salient points are removed as well. Notice the perceptual difference.

$$\sigma(x) = \frac{\rho(x)}{D(x)}, \quad \forall x \in \Omega \tag{2.8}$$

where $\rho(x)$ denotes the length of the border between the two boundary points of x, and $D(x)$ is the distance to the border, as given by the Distance Transform (DT).The saliency metric $\sigma(x)$ only takes border irregularities and corners into account, and does not look at other features of shapes. This is justified by the fact that, ultimately, border irregularities and corners determine the essence of a shape. It is based on the following two observations:

I. Saliency is proportional with *size*, which can be measured by boundary length. Longer features are more salient than shorter ones. [16]

II. Saliency is inversely proportional with the local object *thickness*. A feature located on a thick object is less salient than the same feature located on a thin object. [28]

An example of this saliency measurement applied to the jagged rectangle shown in fig. 6 is shown in fig. 7.

The main drawback of this method is that it does not monotonically increase over a skeleton, i.e. after thresholding we will get disconnected skeleton branches, shown in fig. 7d. However, this is easily remedied by using a connected-component filter to removes all but the

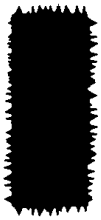
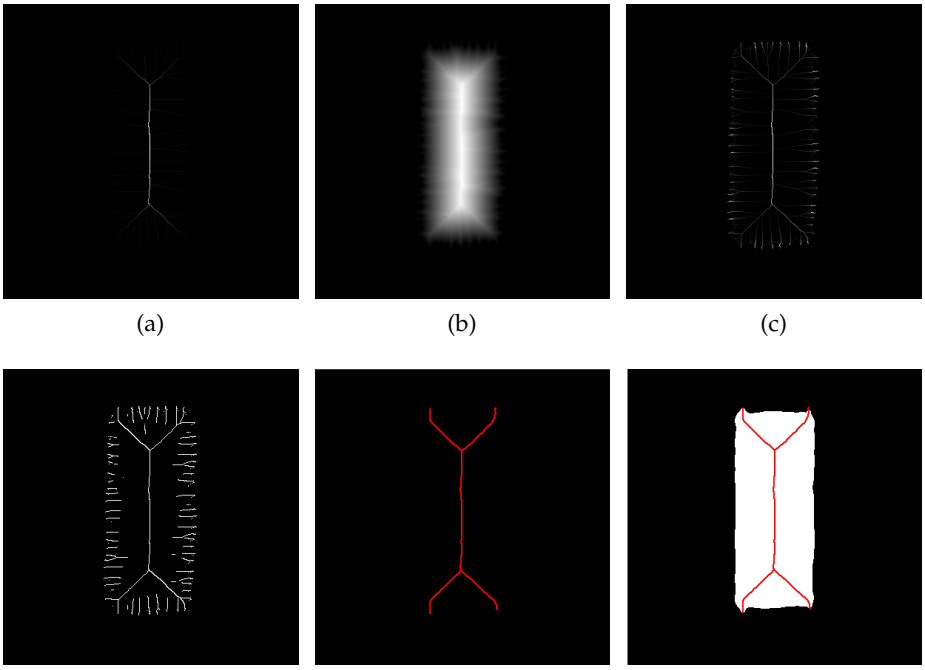


Figure 6: Jagged Rectangle



(a) (b) (c)

(d) (e) (f)

Figure 7: a) Boundary length $\rho(x)$, b) Distance $D(x)$, c) Saliency metric $\sigma(x)$, d) Thresholded, e)Removed small components, f) Reconstruction

largest skeleton in the image. To demonstrate its simplicity, a possible matlab implementation is shown in lst. 1. The resulting skeleton is shown in fig. 7e, and its corresponding reconstruction in fig. 7f.

Listing 1: Remove all but largest component

```
% Let IM contain our bitmap
CC = bwconncomp(IM);
numPixels = cellfun(@numel,CC.PixelIdxList);
[biggest,idx] = max(numPixels);
out = zeros(CC.ImageSize);
out(CC.PixelIdxList{idx}) = 1;
```

This saliency filtering provides a feature-preserving smoothing algorithm, while still greatly reducing skeleton complexity. And does so at the cost of just 1 extra parameter (threshold for σ).

3

SHAPE EXTRACTION FROM IMAGES

The strength of skeletons is that they can be a very compact representation of large objects. However, skeletons are only defined for binary images (0/1 denotes *inside / outside* object). For this thesis we are looking at grayscale images. One way to use the powerful properties of skeletons for encoding would be, here, to reduce such grayscale images to sets of binary images (by segmentation), and next apply our skeleton-based encoding on these binary images. After segmenting the grayscale image we refine our data, such that unimportant parts are removed.

3.1 SEGMENTATION

For our segmentation algorithm we propose to use a "*Threshold-set*". We define a threshold-set T on an image I as shown in eq. (3.1).

$$T_i(x,y) = \begin{cases} 1, & \text{if } I(x,y) \geqslant i \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

Using this segmentation, we try to introduce more coherency within each layer i (i.e. we try to make larger objects). Figure 8 tries to visualize why a normal set is not suitable for extracting objects, by showing the low coherency in a normal layer.

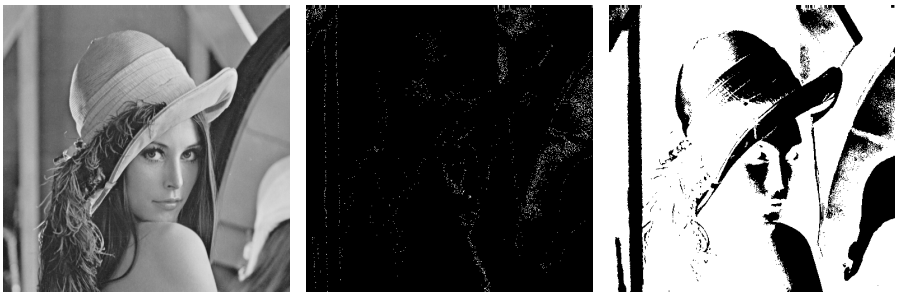
An important observation for this segmentation is that $\forall i,j : i \leqslant j \implies T_j \subseteq T_i$, or in words: Each layer in the thresholdset is a subset of (or equal to) its previous layer. A proof is given below.

Proof: *A threshold-set consists of subsets.*
if $i \leqslant j \ \wedge \ (\forall x \in T_j : x \geqslant j)$, then:
$\forall x \in T_j : i \leqslant j \leqslant x$, therefore (by definition):
$\forall x : x \in T_j \implies x \in T_i$ □

This observation is important, because this guarantees that if we delete a point in a layer T_i, the point is still in T_{i-1} (i.e. by removing a point (x,y) from the highest layer it is in, we reduce the intensity of that pixel by exactly 1).

3.2 REMOVING SMALL OBJECTS

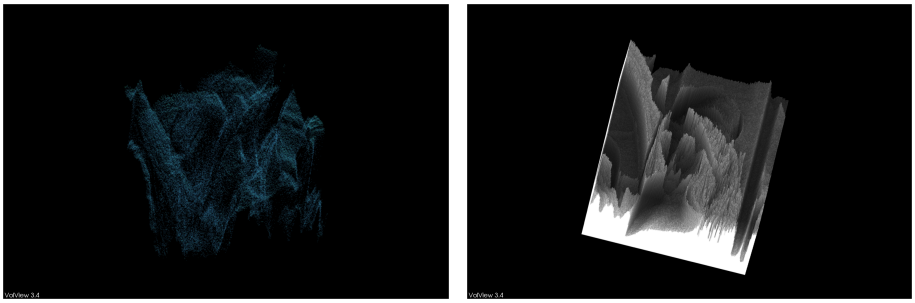
Looking at fig. 8c, we still see a lot of noisy edges. The problem with these noisy edges is that they introduce a lot of small objects



(a) Original

(b) Thresholded at the intensity with the most pixels ($i = 155$)

(c) Thresholdset level $i = 155$



(d) 3D View Intensities

(e) 3D View Threshold Set

Figure 8: Different segmentation methods for shape extraction. Notice how (b) and (d) show that there are no large objects.

(hard to compress using skeletons), and a lot of small holes in objects (increases complexity of the skeleton), while these small segments are fairly unimportant in the image (see fig. 9). This claim is in accordance with the saliency metric σ mentioned in section 2.3.2: we only want to retain features which are visible on a coarse scale.

In order to get a good compression rate it is therefore vital to remove such small segments. We therefore introduce a new parameter ω to our algorithm, which denotes the minimum size an object must be in order to be retained.

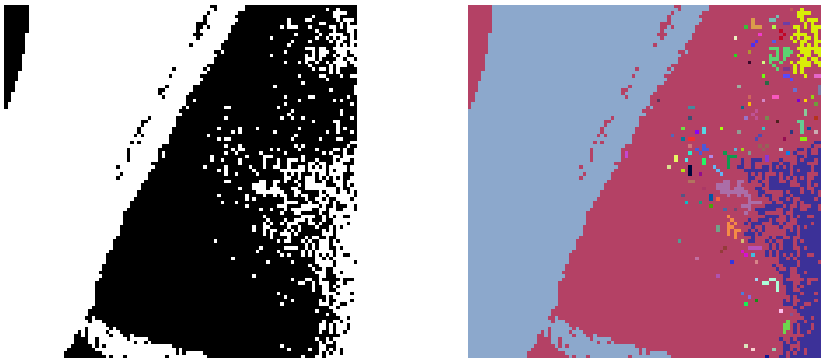
Because both foreground and background pixels need to be filtered, we perform our Connected Component Analysis (CCA)-algorithm twice. Filtering happens as shown in alg. 1 (for brevity's sake we have ommitted the second pass).

3.3 REMOVING LAYERS

The fact that a layer T_i contains a complex and / or large skeleton does not guarantee that it is important for the reconstruction. It is therefore important to define a Γ_i which provides an intuitive importance metric, such that we can remove layers by thresholding Γ. We propose the definition given in eq. (3.2), for three reasons: 1) The metric has an intuitive scaling factor, where 0.0 represents a layer which is not used

Algorithm 1 Removing Small Objects

```
labels ← CCA(im)
hist ← histogram(labels)
for all pixels (x, y) do
  if (x, y) ∈ background ∧ (hist(labels(x,y)) < ω then
    Remove (x,y) from background
    Add (x,y) to foreground
  end if
end for
```



(a) Crop

(b) Connected Component Analysis (random colors, 118 objects)

Figure 9: A small crop of T_{155} of *lena* showing the need for small object removal.

for the reconstruction at all, and 1.0 is the layer which contributes the most to the reconstruction. 2) To compute this metric you only need to look at the next layer, because $x \notin T_i \implies x \notin T_{i+n} (n \in \mathbb{N}^0)$. 3) This metric has an intuitive interpretation: The set Γ is equal to the normalized histogram of the image. Thus thresholding effectively means that all layers are removed for which the pixel ratio $\Gamma_i : \max \Gamma$ is smaller than $T_\Gamma : 1$.

$$\Gamma_i = \frac{\gamma_i}{\max \gamma} \quad \text{, with:} \tag{3.2}$$
$$\gamma_i = \#\{\, x | x \in T_i \wedge x \notin T_{i+1} \,\}$$

4 SHAPE ENCODING WITH SKELETONS

After we have segmented our image, we have increased our data by a factor 32 (a monochrome $m \times n$ image, with 256 intensities takes $8mn$ bits. Our thresholdset contains 256 binary $m \times n$ layers, thus takes $256mn$ bits). This section describes the transformation from segments to skeletons, the filtering of data in skeleton-space, and the storage format.

4.1 SKELETONISATION

For the skeletonisation of each layer T_i we use the FMM based algorithm, called: Augmented Fast Marching Method (AFMM). The layers are processed one at a time, due to the high amount of memory involved otherwise.[1]

4.1.1 *FMM*

The FMM [26, 27] is a scheme to solve the Eikonal equation (see eq. (4.1)). [2] FMM propagates from T *upwards* from the smallest known values for T. This is done by considering a thin zone around the existing front – also referred to as *narrow band* [30] – and marching this thin zone forward, freezing the values of existing points and bringing new ones into the narrow band structure.

The full implementation is given in [30].

$$|\nabla T|F = 1, \text{ with } F = 1 \tag{4.1}$$

In order to maintain such a narrow band, each pixel with coordinate (i, j) gets a flag $f_{i,j}$, which can be either of these 3 values:

BAND The point belongs to the current position of the moving front. Its T value is undergoing update.

INSIDE The point is inside the moving front. Its T value is not yet known.

KNOWN The point is behind the moving front. Its T value is already known.

The initialization of the values T and the flags f is shown in alg. 2. The algorithm itself is described in high-level pseudo-code in alg. 3

Efficient computation of T_n is not trivial. See [30] for a C++ implementation of an efficient upwinding scheme.

4.1.2 *AFMM*

The main difference between FMM and AFMM lies in a single extra value $U_{i,j}$ for each pixel (i, j). This value $U_{i,j}$ is set to 0 on an arbitrarily chosen boundary point, and is then increased monotonically along

1 For example an image of 1024×1024 would take $1024^2 \cdot 256 \cdot 2 \cdot \text{sizeof(float)} = 2\text{GB}$.

2 The Eikonal equation is a non-linear differential equation used to describe the travel-time propagation in an isotropic medium.

Algorithm 2 Initialize T and f for FMM

for all (i,j) **do**
  if (i,j) is on boundary **then**
    $f_{i,j} \leftarrow$ BAND ; $T_{i,j} \leftarrow 0$
    add (i,j) to NarrowBand
  else if (i,j) is inside boundary **then**
    $f_{i,j} \leftarrow$ INSIDE ; $T_{i,j} \leftarrow \infty$
  else {(i,j) is outside boundary}
    $f_{i,j} \leftarrow$ KNOWN ; $T_{i,j} \leftarrow 0$
  end if
end for

Algorithm 3 FMM Band propagation

while NarrowBand $\neq \emptyset$ **do**
  $A \leftarrow (i,j)$ with lowest value T in NarrowBand
  $f_{i,j} \leftarrow$ KNOWN
  Remove A from NarrowBand
  for all $n \leftarrow$ Neighbours$((i,j)$ **do**
    if $f_n =$ INSIDE **then**
      $f_n \leftarrow$ BAND
      Add n to NarrowBand
    end if
  end for
  for all $n \leftarrow$ Neighbours$((i,j)$ **do**
    if $f_n =$ BAND **then**
      Compute new T_n
    end if
  end for
end while

the boundary, starting from the $U = 0$ pixel. U is thus a boundary parameterization with the property that the distance between any two boundary points measured along the boundary is equal to the difference in the corresponding U values (see fig. 10). U is then propagated along with T. They are interpolated, via averaging, on concave boundary segments (as the segments increase length when marching inwards). On convex segments when a point has neighbours with a difference in U greater than $\sqrt{2}$, one of the U's is simply propagated further. The result is not averaged, since a difference greater than $\sqrt{2}$ means it is a skeleton point, as the difference in U for two neighbours can never exceed $\sqrt{2}$ (see fig. 11).

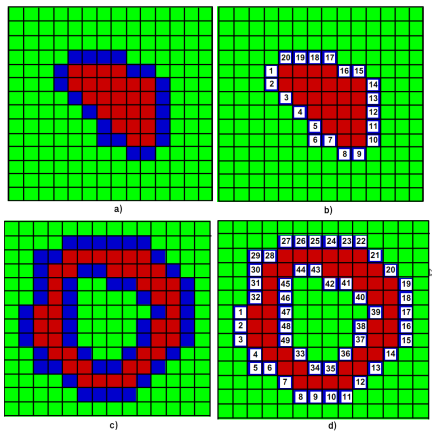


Figure 10: Objects(a,c) and the order in which U is assigned to their boundaries (b,d) by AFMM

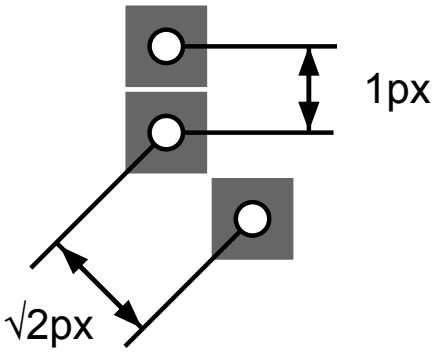


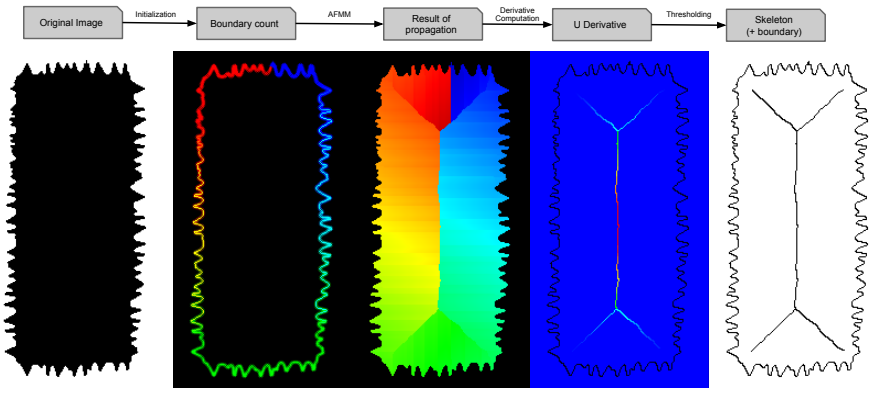
Figure 11: $\Delta U > \sqrt{2} \equiv$ skeleton point

After U is computed, the skeleton points can be detected by finding *sharp discontinuities*. The discontinuities are "strong enough" in the sense that a simple differentation scheme is sufficient to find skeleton points. Due to the order of visitation of the FMM algorithm the algorithm generates connected skeletons. All points that have a difference in U with their neighbours higher than some threshold t are retained, while the others are discarded. This is the *only* parameter of the algorithm, and has a well-defined and intuitive meaning, such that even non-expert users can set appropriate values. Figure 12 is a graphical representation of the aforementioned stages of the AFMM process.

The original AFMM implementation has been further refined to better numerically handle several border cases. For details, we refer to [23].

4.1.3 *Image space skeleton simplification*

The boundary of the segments of which we compute the skeletons are virtually always noisy, which means that the skeletons have a lot



(a) Original (b) Boundary detection (c) Boundary propagation (d) U Derivative (e) Skeleton

Figure 12: The steps of- and data generated by the AFMM

of unimportant branches. We filter these branches by computing the saliency metric σ, as defined in eq. (2.8). The problem of the saliency metric is that thresholding results in disconnected skeletons, of which we want to filter all but the largest. Our segmentation however, does not guarantee that one layer will consist of one object. Because we want to retain the largest remaining skeleton from each object, we need to perform some additional object analysis, such that we retain the largest skeleton *per object*. The algorithm is shown in alg. 4.

Algorithm 4 Saliency Thresholding Multiple Objects

```
obj ← CCA(I)
for all pixels (x, y) do
  if saliency(x, y) < κ then
    I(x, y) ← 0
  end if
end for
post ← CCA(I)
for all o ∈ obj do
  m ← largest(o, post) {Get largest segment in post from o}
  keepSegments.add(m)
end for
for all pixels (x, y) do
  if post(x, y) ∉ keepSegments then
    I(x, y) ← 0
  end if
end for
```

After thresholding we perform a morphological closing I_K $((I \oplus K) \ominus K)$ on I, with a block-kernel K of 1's [25]. This connects skeletons which are close to each other, by inserting skeleton points with a DT value of 0. This helps optimize our encoding process described in section 4.4.

Finally we remove all points which are not critical for preserving the shape. Even though AFMM guarantees that skeletons are one pixel thick, this holds only for 4-connectedness. Our encoding method supports 8-connectedness, thus we can remove additional skeleton points. For each point $(x, y) \in S$ we take a 3×3 window, and check if it is not an endpoint. If not: the point is removed, and we verify if the skeleton is splitted. Points which do not split the skeleton are removed. Note that this could also have been implemented using morphological thinning / erosion. A graphical explanation is shown in fig. 13. A proof demonstrating that these "redundant" points do not contribute very much to the reconstruction of the object is shown in appendix A.

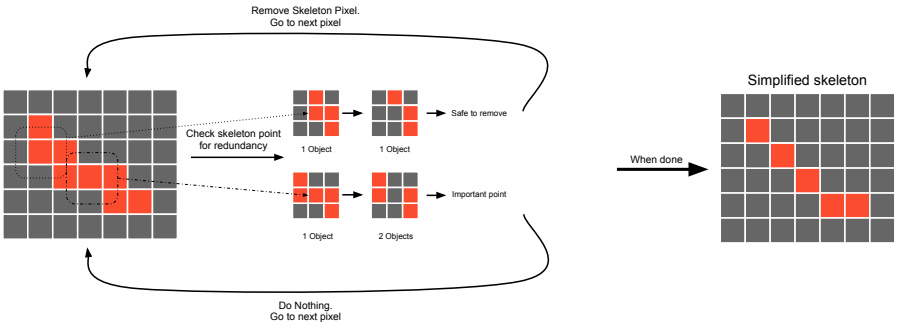


Figure 13: Graphical explanation of the removal of "redundant" skeleton points

4.2 TREE REPRESENTATION OF SKELETONS

After having filtered the skeletons on image-space, it becomes more convenient to switch to a different representation: *Trees*. Trees are favourable over image-space because (1) they take up far less space (it does not represent non-skeletonal points); (2) they combine the skeleton map with the DT map; and mostimportantly (3) *They have a well defined beginning and end*. In image-space it is possible for a skeleton to have neither (e.g. the skeleton of a donut). A well defined beginning and end drastically eases the encoding process, and is therefore favourable.

Each skeleton in a layer is represented by a tree. This is done by scanning a layer in image-space, until we reach a skeleton point. That point is "promoted" to *root* of the tree, and the image is recursively traveled in a Depth First Search (DFS) manner along all neighbours, until the entire skeleton is represented in the tree. This is then repeated for all skeletons in the image. To avoid cycles we keep track of the skeleton points we have visited. A few examples are shown in fig. 14. The algorithm is shown in pseudocode in alg. 5.

Algorithm 5 Convert image skeleton to tree

Function: traceLayer

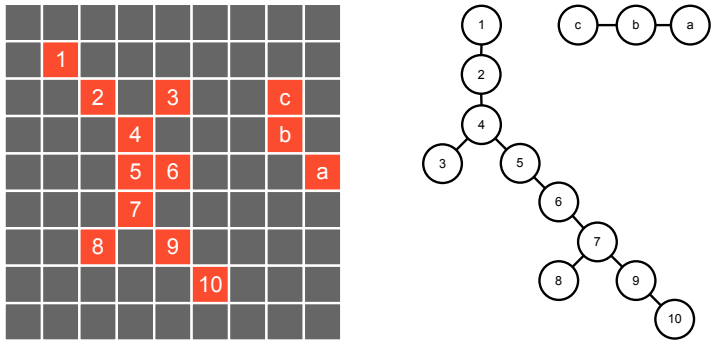
Require: DT Distance Transform Map
SKEL Skeleton Map

```
l ← ∅
for y = 0 to height do
  for x = 0 to width do
    if SKEL(x,y) > 0 then {Skeleton point}
      p ← tracePath((x,y), SKEL, DT)
      add p to l
    end if
  end for
end for
return l
```

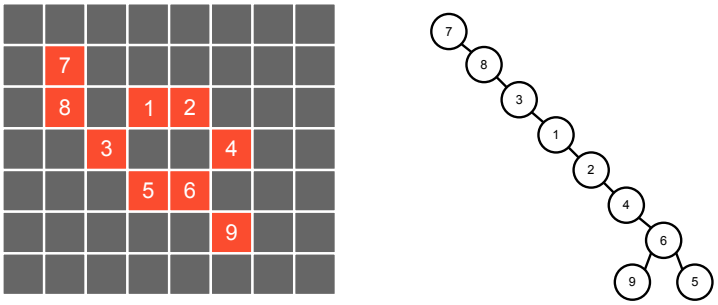
Function: tracePath

Require: (x,y) Location of current skeleton point
DT Distance Transform Map
SKEL Skeleton Map

```
Node n ← {x, y, DT(x, y)}
SKEL.remove(x,y)
while (x, y) has neighbouring skeleton-points do
  ne ← first neighbour of (x, y)
  n.addChild(tracePath((x,y), SKEL, DT)
end while
return n
```



(a)



(b)

Figure 14: Examples of skeletons in image space and represented in a tree. a) Shows multiple objects, b) Shows how cycles are handled.

4.3 FILTERING TREE SKELETONS

The goal is to further reduce the presence of points which do not contribute much to the reconstruction. We do so by removing "*small paths*" from the tree. For each node which has more than one child, we check the depth of those children. Branches whose depth is smaller than a threshold is removed. This is done using Breadth First Search (BFS) to avoid removing longer paths (which would happen with DFS, as we would then first remove a child and then check the length of its parent).

Furthermore we remove unimportant objects o, where we define importance as $\varphi(O) = \sum_{p \in O} r_p$, where r_p denotes the radius of p. If φ is below the threshold, then the object has neither large discs, nor a lot of discs. It is therefore deemed unimportant, and removed.

4.4 ENCODING TREES

Each node in a a tree has a different (x, y) coordinate, and a radius r. A naive method would be to store each (x, y, r) triple as three "shorts". But due to our segmentation this would result in a file larger than the raw format. This is because our threshold set most likely contains

a short is 16 bits in C, thus limiting the dimensions to 65536×65536.

more than $m \times n$ discs for an $m \times n$ image. We therefore need a more sophisticated encoding scheme. A key observation is that for each node we *roughly* know where the children will be. *Children are always adjacent (8-connectedness) to the parent*. This observation enables us to encode a skeleton "path" using the scheme shown in fig. 15. As this limits the possible values to a mere 8, we only need 3 bits, rather than 2×16. A similar trick is performed for storage of the radii. We know that the radius of two adjacent skeleton points cannot differ more than $\sqrt{2}$ (see Appendix A). As we do not need sub-pixel accuracy, all radii are rounded to their nearest integer. Due to some rounding errors, the difference in two adjacent radii is in $[-2, -1, 0, 1, 2]$. This means that storing the radii differences, rather than radii values we only need 3 bits, in contrast to the 16 raw storage takes.

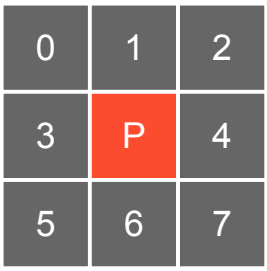


Figure 15: Neighbour encoding

As we want to convert a tree with all its branches into a single difference-stream, we encode using a *state-history*. When the encoder reaches the end of a branch (but not the end of the object), a *GO-BACK*-tag is inserted, which contains the length l of the branch it encoded. If the state is then restored to l states earlier, it contains the (x, y, r) values at the start of the branch. It can then continue storing differences for another branch, without wasting bits by storing new start points. This process is demonstrated in fig. 16. Note that for brevity's sake we have omitted the radii corresponding to the skeleton points.

4.5 SKELETONAL IMAGE REPRESENTATION FILE FORMAT

In order to measure the compression rate, we have developed a preliminary file-format. The file format has the extension *.sir*, which stands for *Skeletonal Image Representation*. To store the data as compact as possible, we encode the stream using the Lempel-Ziv-Markov Chain Algorithm (LZMA)[17]. A Skeletonal Image Representation (SIR) file consists of the following data :

The superscript denotes the number of bits reserved

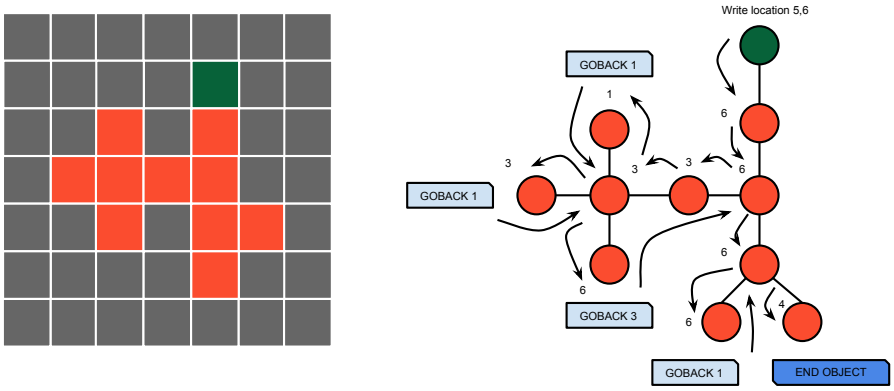


Figure 16: Encoding process. a) Shows a skeleton, b) shows a possible tree representation. The arrows denote the order in which they are processed. At the bottom is the encoded sequence.

VERSION^{16}	The version number of the file. (Currently: 9) Useful for backwards compatability.
WIDTH^{16}	The width of the image
HEIGHT^{16}	The height of the image
LZMA-PROPERTIES^{40}	LZMA requires 5 bytes of properties to be supplied for decoding.
LZMA-ENCODED-DATA	The actual image is stored here. Size varies.

The decoded LZMA data is stored as a list of:

ALL values are unsigned, and stored little-endian

INTENSITY^{8}	The intensity of the objects that will now follow.
NUMPATHS^{16}	The number of paths for this intensity
PATH DATA	The path of skeletons

Where PATH-DATA is stored as:

X^{16}	The initial X coordinate.
Y^{16}	The initial X coordinate.
R^{16}	The initial X coordinate.
{{ CHAIN PATH }}^{8n}	A list of bytes, where the high nibble represents the next position of the skeleton, according to the values in fig. 15. The lower nibble represents the difference in radius. To avoid the signed bit all values are shifted to positive values by adding 8 to the real value.

- A neighbour value of 10 is a `GOBACK`-tag, the radius value of a `GOBACK`-tag is undefined. A `GOBACK`-tag is followed by 16 bits, which is the number of states to go back.
- A neighbour value of 9 is an `END`-tag. The path ends after this tag. The radius value of a `END`-tag is undefined.

5 SIMPLIFIED IMAGE RECONSTRUCTION

This section will describe how we can reconstruct an image from a set of points (x, y, r, i), where (x, y) is the position of the center of the disc, r the radius, and i the intensity. Reconstruction happens on a per-layer basis, and is done from the lowest intensity to the highest intensity (due to the threshold-set definition). After all layers are reconstructed, they are visualized using a smooth transition function, in order to reduce boundary artefacts. This chapter assumes the reconstructed images are in grayscale, although the reconstruction technique can be applied to all monochrome images.

5.1 LAYER RECONSTRUCTION

Reconstruction of a layer l_i is an inflation of the layer's skeleton with equal speed until the inflated shape locally reaches a distance from the skeleton equal to the radii values stored on the skeleton. The reconstruction provided here has the main advantage of simplicity and a relatively efficient implementation in graphics hardware. However, the same result can be obtained using the Fast Marching Method (FMM), starting from the skeleton outwards with the local stop criterion given by the skeleton radii, as described in e.g. [29].

Our reconstruction method for a layer l_i with intensity i iteratively draws all discs with the intensity i on a 0/1-map, denoting outside/inside object respectively (this corresponds to the alpha-map as used in *OpenGL*). Due to the fact that the actual algorithm contains a few subtleties (such as texture coordinates), it is possibly best explained by providing a hybrid between pseudocode and *OpenGL*-calls, as shown in alg. 6. This shows C style calls to OpenGL, and iteratively draws all points as quads on the screen. The quads are then textured, using four channels *RGBA*. For each pixel it is computed if it is inside our outside the corresponding disc. If it is inside, the pixel is drawn full white, and placed "in front". Otherwise it is drawn as black, and "far-away". As we have enabled depth-testing, the result is a 2D-texture where all pixels that have been "on" (white) at least for one disc are drawn white, and all other pixels remain black.

5.2 TRANSITION FUNCTION

One of the key-points of our compression algorithm is the ability to remove entire layers of information. In order to compensate for the "border-artefacts" that may occur due to this compression (see fig. 18

Algorithm 6 Reconstruction of a layer

Function: Reconstruct()
Require: P list of points
  glEnable(GL_DEPTH_TEST);
  glBegin(GL_QUADS);
  for all $p \in P$ **do**
    glTexCoord2f(-1.0, -1.0) ; glVertex2f($x_p - r_p$, $y_p - r_p$);
    glTexCoord2f(-1.0, 1.0) ; glVertex2f($x_p - r_p$, $y_p + r_p$);
    glTexCoord2f(1.0, 1.0) ; glVertex2f($x_p + r_p$, $y_p + r_p$);
    glTexCoord2f(1.0, -1.0) ; glVertex2f($x_p + r_p$, $y_p - r_p$);
  end for
  glEnd(GL_QUADS);

Function: Fragment Shader
  float alpha = TexCoord.x^2 + TexCoord.y^2 $\leqslant$ 1.0 ? 1.0 : 0.0;
  gl_FragColor = vec4(alpha,alpha,alpha,alpha);
  gl_FragDepth = 1.0-alpha;

for an example), we have looked at a transition function to create more gradually changing intensities. As a reference fig. 19 shows the border of a single layer without interpolation. To generate a smooth border we set a parameter b, denoting the maximal distance from the border for which the opacity will be lowered (thus every point which is farther from the border is left untouched).

This is done by calculating the DT, using AFMM. The DT is then transformed using the function $t(x, y) = \min(\frac{1.0}{b} DT(x, y), 1.0)$ (E.g. for $b = 5$ this would lead to an alpha map of $[0.0, 0.2, 0.4, 0.6, 0.8]$ for distances: $[0, 1, 2, 3, 4]$ respectively, and 1.0 otherwise). The result of this transformation is shown in fig. 20, and a full reconstruction using this transition function in fig. 21.

The main issue with this transition function is that it modifies the shape of the object (thicker border results in a smaller object). We tried to overcome this by - rather than changing the opacity from $100\% - 0\%$ inside the object - expanding the object by half the border size, and creating a transition function such that the transition from $100\% - 50\%$ opacity happens inside the object, and the transition of $50\% - 0\%$ opacity happens outside of the object (see fig. 17). The transition is rotated $180°$ (i.e. area A = area B), such that we have the theoretical advantage that we modify equally much inside as outside the object. Unfortunately the visual results are far from optimal. Even for small border ranges - e.g 2px - the image becomes gravely deformed, as shown in fig. 22. This is most likely due to the asymmetry in the threshold set. A light spot is a segment, while a dark spot is a hole in one or more segments. Thus this effectively expands all the white areas, while shrinking dark areas.

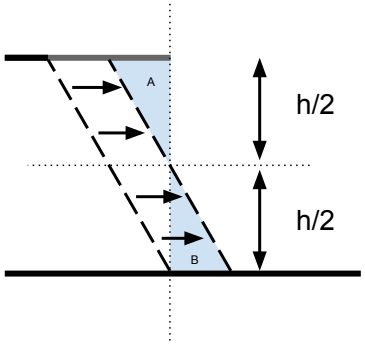


Figure 17: Transition function which keeps objects the same size

5.3 VISUALIZATION

To reconstruct the image, we draw each layer iteratively, starting from the lowest layer. The output of the transition function as explained in Section 5.2 can be used as an alpha map for our visualization. This is done by setting the OpenGL colour state (*glColor3f*) to the intensity of the layer, and drawing a quad of that color over the entire window, using the alpha map as a stencil.

OpenGL note: Depth testing should be disabled, as we always want to overwrite previous layers

5.3.1 *Base color*

Let i be the lowest intensity in an image, then our algorithm does not store layers $0 - i$, as we can easily see that using a background intensity of i is equal to using the (perfect) reconstruction of the skeleton of i. In other words: the reconstruction of the first layer contains "holes". This means that if i is much larger than 0 (e.g. $25 - 50$), the reconstruction will show prominent gaps. To avoid this problem we use an estimate of the background color, such that the gaps of the lowest layer L_j have an intensity of $j - 1$. Although it is not guaranteed that $i = j$, this estimate suffices for most purposes.

Figure 18: Border effects due to heavy layer compression (removed 154 layers)

Figure 19: Single layer of lena, reconstructed without interpolation

Figure 20: Interpolation function for values $b = 5, 10, 15, 25$

(a) $b = 2$ (b) $b = 25$

Figure 21: Transition function which alters the object's form by making the boundary transition happen solely inside the object.

(a) $b = 2$ (b) $b = 25$

Figure 22: Transition function which enlarges objects, such that the boundary transition happens equally much inside as outside the object.

6 EXAMPLES

In order to provide a good impression of what our method can and cannot do, we use two well known images *mandril* and *peppers*, as shown in fig. 23. We have chosen these images in particular, as they represent an example of an image which can be compressed very well with our method (*peppers*), and an image which compresses not very well (*mandril*). The file sizes of the raw images are 256KB.

(a) Mandril (b) Peppers

Figure 23: Reference images for parameter testing

The subsections below will consider one parameter per section, to provide a feeling of the meaning of each parameter, as well as its effectiveness. While looking at the arising artefacts, it is crucial to keep the segmentation algorithm in mind. Due to the fact that a threshold set is used, some of these filtermethods are not symmetric (i.e. dark spots are actually holes in lighter segments).

6.1 LAYER THRESHOLD

The layer threshold parameter T_Γ filters layers as described in section 3.3. Intuitively speaking: setting $T_\Gamma = 0.5$ means that all layers which have less than half the pixels compared to the most important layer will be removed. Setting this parameter really low (e.g. 0.000001) removes layers which are hardly visible, while still reducing file size. It is therefore recommended to always choose $T_\Gamma > 0$. Figure 24 shows the mandril and peppers with various thresholding levels. It can be seen that removing layers can have a lot of impact on the reconstruction. When removing too much layers (which is best shown in fig. 24d) the image loses a lot of detail and contrast. Highlights become a flat color, rather than a gradient, or are removed entirely.

6.2 SMALL COMPONENT REMOVAL

The Small Component Removal tries to eliminate smaller objects on a per layer basis, in order to remove small skeletons in the image. This also has the side effect that it "smoothes" noisy images. fig. 25 shows the effects of this parameter. The effect is best seen on the detail of the moustache of the mandril, where the thin hairs are gradually replaced by "blobs". The effect isn't very clearly noticeable in the peppers image, due to the fact that it consists of relatively large shapes. This also shows in the relatively poor compression with respect to the mandril (40% : 20%).

6.3 SALIENCY THRESHOLDING

Saliency thresholding tries to remove all the features from a segment which are perceived as noise. It can be seen that shapes in general become "simpler", best seen in the *pepper* image in fig. 26. The gradient of the pepper in the back has more straight lines when increasing the saliency threshold.

6.4 SKELETON DISTANCE TRANSFORM THRESHOLDING

This method removes points in a rather naive way. Generally this value is set to $\sim 2-3$, and the filtering is done via saliency thresholding. Although it is somewhat easy to visualize the effects for grayscale images, it might not be as straightforward to visualize the effect for monochromatic images. The effect is best seen in the peduncle of the red pepper in fig. 27, which becomes disconnected due to its short and long appearance.

6.5 SHORT PATH REMOVAL

Short Path Removal removes all branches in a skeleton which are below a certain length. This method is based on the encoding technique used, where a branch is more expensive to represent as a normal path. Thus short paths (which should contribute little to the image) are best removed to achieve a better compression. The effects can be seen in both images from fig. 28, where highlights are removed, and small branches. The mandril has larger artefacts at the bottom of the image, probably due to boundary artefacts.

6.6 STRUCTURAL SIMILARITY INDEX

6.7 GENERAL EXAMPLES

To demonstrate the strengths and weaknesses of our compression, we have compressed a few well known images in the field of Image Processing. This section provides more details of these images, to provide more insight in what the method can and cannot do. We also use the Structural Similarity Index (SSIM) to provide a metric for the image quality.

Cameraman

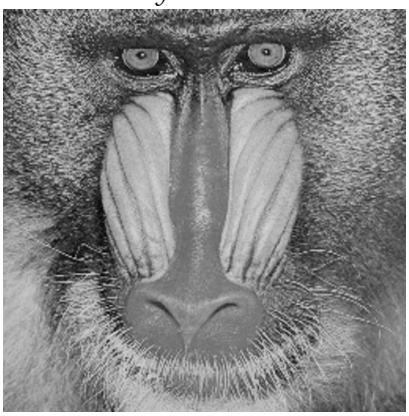
Original file-size: 256.0KB.

Compressed file-size: 102.6KB.

MSSIM: 0.835

Computation time: 148 seconds

Mandril Gray



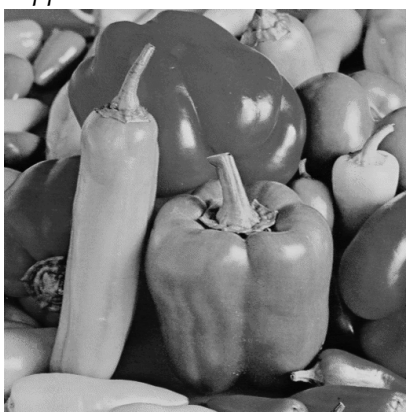
Original file-size: 256.0KB.

Compressed file-size: 195.9KB.

MSSIM: 0.547

Computation time: 126 seconds

Peppers



Original file-size: 256.0KB.

Compressed file-size: 57.4KB.

MSSIM: 0.728

Computation time: 91 seconds

Lake

Original file-size: 256.0KB.

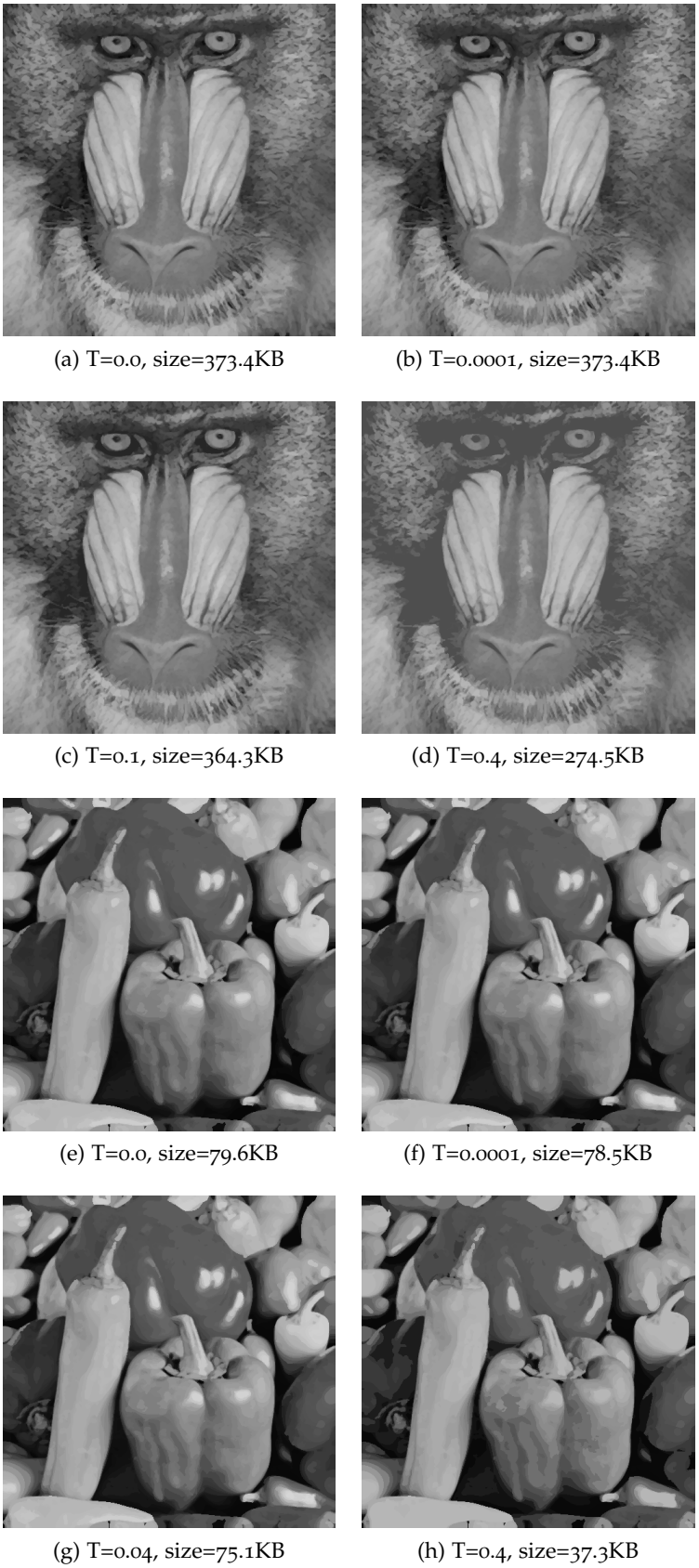
Compressed file-size: 160.1KB.

MSSIM: 0.688

Computation time: 145 seconds

The above examples demonstrate that even though the images are compressed and still provide a visually attractive image, more work needs to be done in order to compete with other compression techniques (such as the JPEG family). As a comparison: the cameraman saved as JPEG with an MSSIM-score of 97% only takes 29KB.

It also tells us that the weakness of our method lies in areas with high detail and a lot of different intensities (e.g. the hairs on the mandril). They take up far more space than larger objects, such as the peppers image.



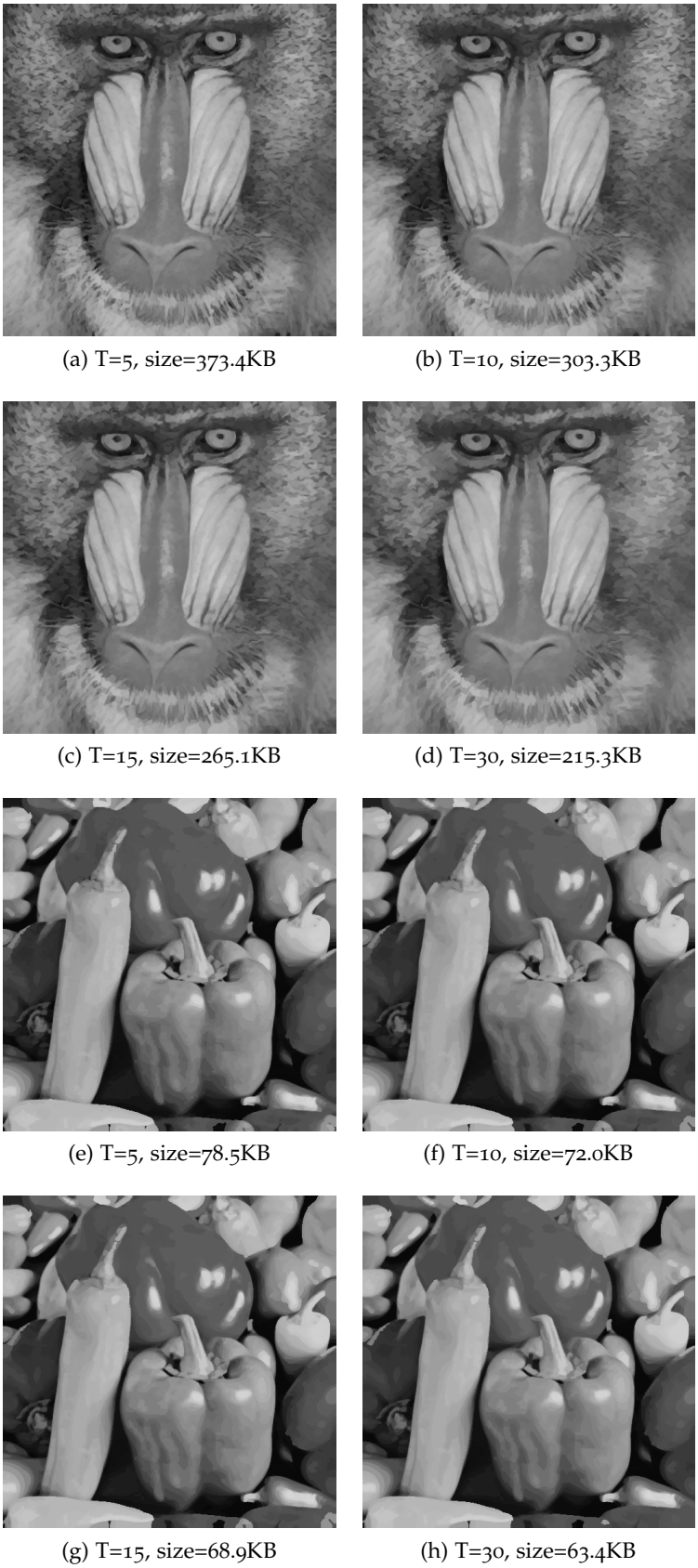
(a) T=0.0, size=373.4KB (b) T=0.0001, size=373.4KB

(c) T=0.1, size=364.3KB (d) T=0.4, size=274.5KB

(e) T=0.0, size=79.6KB (f) T=0.0001, size=78.5KB

(g) T=0.04, size=75.1KB (h) T=0.4, size=37.3KB

Figure 24: Demonstrating the effect of Layer Thresholding



(a) T=5, size=373.4KB (b) T=10, size=303.3KB

(c) T=15, size=265.1KB (d) T=30, size=215.3KB

(e) T=5, size=78.5KB (f) T=10, size=72.0KB

(g) T=15, size=68.9KB (h) T=30, size=63.4KB

Figure 25: Demonstrating the effect of Small Component Removal

(a) T=2.0, size=373.4KB (b) T=3.0, size=332.9KB

(c) T=4.0, size=303.5KB (d) T=5.0, size=282.8KB

(e) T=2.0, size=78.5KB (f) T=3.0, size=64.5KB

(g) T=4.0, size=57.2KB (h) T=5.0, size=51.7KB

Figure 26: Demonstrating the effect of Saliency Thresholding

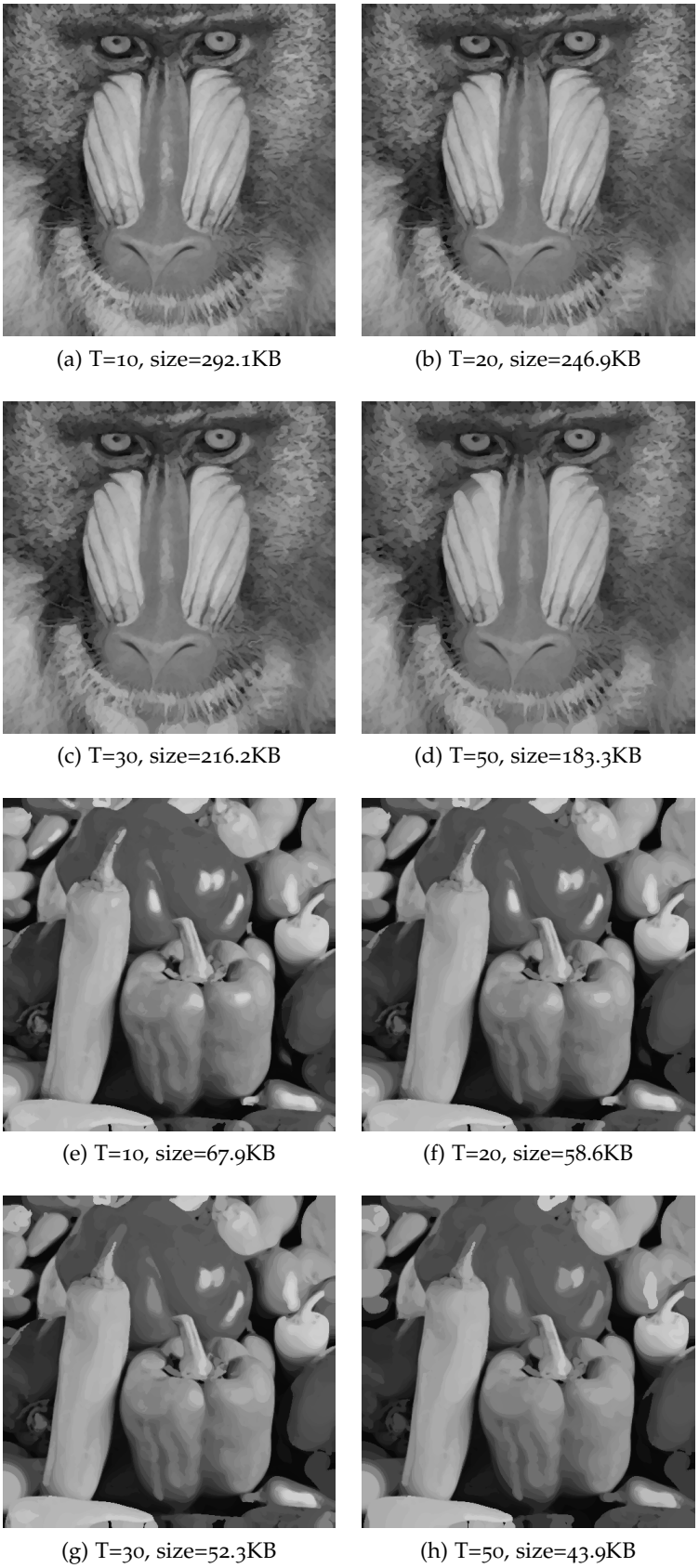
(a) T=3, size=450.8KB (b) T=6, size=401.8KB

(c) T=9, size=368.2KB (d) T=12, size=343.5KB

(e) T=3, size=114.0KB (f) T=6, size=101.4KB

(g) T=9, size=92.0KB (h) T=12, size=84.0KB

Figure 27: Demonstrating the effect of Distance Transform Thresholding



(a) T=10, size=292.1KB (b) T=20, size=246.9KB

(c) T=30, size=216.2KB (d) T=50, size=183.3KB

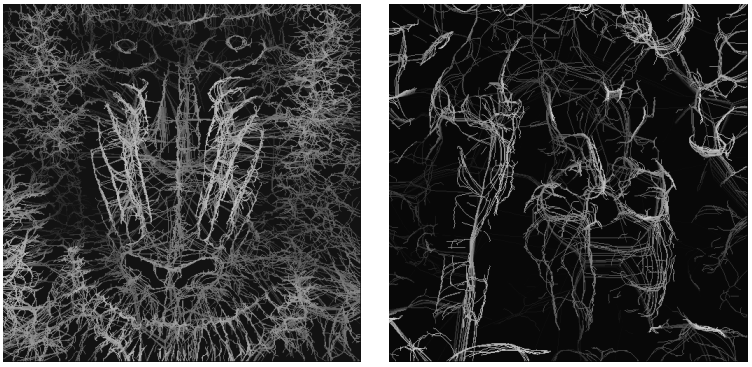
(e) T=10, size=67.9KB (f) T=20, size=58.6KB

(g) T=30, size=52.3KB (h) T=50, size=43.9KB

Figure 28: Demonstrating the effect of Distance Transform Thresholding

7 DISCUSSION

Although we have a skeleton based image representation technique, which can restore a monochromatic image using a lossy compression technique, it is not yet ready to compete with JPEG. Chapter 6 supports this with various examples. For all of these examples holds that - although they closely resemble the original - their file size is too large and their SSIM value is too low. In order to decrease the file size further, it is probably best to look at inter-level coherence. A visualization of the skeletons of the *mandril*, and the *peppers* shown in fig. 29 supports the fact that levels have a high coherence.



(a) Mandril skeletons (b) Peppers skeletons

Figure 29: Skeletons of the images show the need to further explore inter-level coherence.

An interesting feature of our technique is that the artefacts very much resemble a non-photorealistic painting effect. In contrast to for instance JPEG, where the aim is to minimize the artefacts, the artefacts of our method might have desired properties in e.g. image filters. An example of an image for which the painting effect is striking is shown in appendix B.

At this stage there are quite a few parameters, namely:

LAYER IMPORTANCE T_Γ THRESHOLD This threshold uses layer importance as defined in section 3.3 to remove layers which do not contribute much the reconstruction.

SMALL COMPONENT REMOVAL Remove small spots in the image, to remove small skeletons, which take up a lot of space, but are hardly visible.

SKELETON SALIENCY σ THRESHOLD This threshold uses the saliency of a skeleton point as defined in section 4.1.3, to simplify the skeleton by removing non salient features.

SKELETON DISTANCE TRANSFORM THRESHOLD Used to define which points on the distance transform are used as skeleton point. Used in combination with the saliency.

SKELETON PATH THRESHOLD Remove small paths in skeleton trees, as they take up a lot of space and are not very salient.

Although all of these parameters have an intuitive explanation, properly configuring all parameters such that a high compression is achieved while retaining detail is cumbersome. For further research it would be sensible to look at either a reduction in parameters, or a better indication / full computation of sensible values.

GPU skeletonization available here: http://www.cs.rug.nl/svcg/Shapes/CUDASkel

The AFMM implementation we have used has been superseded by a much faster variant, running on the GPU instead of the CPU. The new version is between 20..80 times faster than the CPU implementation, which effectively means that switching to the new implementation would reduce our computation times from ~ 2 minutes to a few seconds. The GPU version is favored over the CPU implementation, but at the time of writing a few important bugs were still present in the CUDA code which made it too unreliable to use for research purposes.

One of the major limitations of our algorithm in its current state is the fact that it only runs on monochromatic images. In our opinion an extension to colors is not trivial, as simply skeletonizing each color channel separately will produce border artefacts in the reconstruction. Due to the removal of skeleton points it is possible that objects are dismissed in some layers, while retained in others. This would produce large perceptual differences. To overcome this further research is necessary.

8 CONCLUSION

With this thesis we have presented a first exploration of the usage of skeletons for image simplification and compression purposes. We provide a segmentation algorithm for monochromatic images, and a multitude of filter algorithms aimed at reducing size while retaining salient features. Furthermore we have provided an efficient method to store skeletons and their distance transform, using neighbour encoding. Although our implementation in its current form takes in excess of two minutes to compress an image, incoorporating a (already existing) more efficient AFMM would make our method run in (near) real-time, and with that proving viable to be used in every day applications.

The current compression results cannot compete with the JPEG-family nor WebP, but a lot of research is to be done in inter-layer encoding, as there seems to be a high coherency between layers.

Its fundamentally different way of looking images opens up several new potential applications, such as *non-photorealistic rendering*, as the characteristic artefacts of our compression resemble the outcome of current non-photorealistic painterly rendering methods. It might also be useful in *shape-based image manipulation*. Due to the fact that we work with segments - rather than pixels - a new editor might enable users to alter images on a shape-based level, rather than pixel level. A third possible application is in *shape-based image encoding*. If we have additional knowledge of an image, such that we know which shapes / segments are most important to store, we can compress the interesting shapes with higher detail than the less interesting shapes.

Our entire algorithm is very easily extendable to 3D. Skeletons and Distance Transform (DT)'s are well defined for 3D and there are readily implementations available. The chain encoding would become a bit more sophisticated in choosing a path order, but the neighbour encoding principle remains the same. This is particularly interesting in medical applications (CT/MRI), where some shapes need to be stored very accurately, while other features can be removed in their entirety.

Part II

SOFTWARE INSTRUCTIONS

This thesis is accompanied by two programs, namely *imConvert* and *imShow*. This section will describe how to compile the software (it should work for Linux, Windows and MacOS), what each program does, and how to create and view SIR files.

9 THE SOFTWARE

9.1 INTRODUCTION AND INSTALLATION

The software accompanying this thesis is split into two parts: *imConvert* and *imShow*. imConvert is the program responsible for the conversion to our Skeletonal Image Representation (SIR) format, and imShow is responsible for the reconstruction. Both programs are written in C++, under Ubuntu. In order to compile them the system needs to contain OpenGL development headers, OpenGL Utility Library headers (GLU) and the toolkit (GLUT), and the OpenGL Extension Wrangler Library (GLEW). Under a Debian based linux they can be installed using the command shown in lst. 2.

Listing 2: Install necessary dependencies

```
sudo apt-get install libgl1-mesa-dev freeglut3 freeglut3-dev
    libglew1.5-dev libglew1.5
```

The package consists of four folders:

imConvert	The folder containing the imConvert program
imShow	The folder containing the imShow program
examples	The folder which has a few Portable Grayscale Map (PGM)'s, and accompanying configuration files.
shared	This folder contains libraries which both programs depend on. They are compiled automatically along with the other programs.

To compile either imConvert or imShow, go into the corresponding folder and type `make`. This will also compile the shared libraries.

9.2 IMCONVERT - GENERATING SIR IMAGES.

Using the imConvert program should be fairly straightforward. Simply start the program using `./main <CONFIG_FILE>`, where `<CONFIG_FILE>` is a configuration file setting a few thresholds, the input image and the output file. A configuration file can contain the following options:

Note: The configuration file parser is a bit sensitive, always use `varname␣=␣value`

```
# Input file (make sure it is a binary PGM):
filename = path_to_input_image
# Output level: {q,e,n,v} quiet, errors only, normal, verbose
outputLevel = n
# Layer threshold - Float between [0..1]
lThreshold = 0.0001
# Small object Threshold, integer value denoting minimal object size
sThreshold = 20
# Skeleton Distance Transform Threshold.
sdtThreshold = 3
# Skeleton Saliency Threshold
ssThreshold = 2.0
# Skeleton Small Object Removal Threshold (OBSOLETE, SET TO 0)
siThreshold = 0
# Minimal Object Size -> New Method
```

```
minObjSize = 5
# Unimportant Object Threshold
minSumRadius = 25
# Minimal Path Length Threshold
minPathLength = 10
# Output File
outputFile = cameraman_best.sir
```

There are various examples in the `example` folder, for demonstration purposes.

9.3 CONVERTING IMAGES TO PGM

The current implementation works best with binary PGM's. This due to their easy decoding process. Converting images to PGM under linux is done using: `convert <IMAGE> pgm:<NEWNAME>.pgm`. Converting a batch of images to PGM can be done with a simple bash script, as shown in lst. 3. This script converts all images in the folder `input/` to PGM and places them in `img/`.

Listing 3: Batch conversion to PGM

```
#!/bin/bash
IMDIR="img";
mkdir -p $IMDIR
for file in input/*; do
  if [ -f $file ] ; then
    # name without extension
    name=${file%\.*}
    name=‘basename ${name}‘
    convert ${file} pgm:${IMDIR}/${name}.pgm
  fi ;

done
```

9.4 IMSHOW - VIEWING SIR IMAGES

To view an image, type `./main <SIR_FILE>` while in the corresponding folder, where `<SIR_FILE>` is the location to a file generated with imConvert as shown in the previous section. This opens up an OpenGL window, and will use the simple reconstruction shader. To use smooth interpolation between layers, use: `./main <SIR_FILE> <BORDER_SIZE>`, where `<BORDER_SIZE>` is an integer denoting the width of the border in pixels.

To make a screenshot (saved as PNG), press 's'. A file named "output.png" will be generated in the same folder, which contains the reconstructed image.

Part III

APPENDIX

A

MAXIMUM DIFFERENCE IN RADIUS FOR TWO NEIGHBOURING SKELETON-POINTS

Following is a topological proof that the maximum difference in radius r for two skeleton points can not exceed $\sqrt{2}$.

Given two neighbouring skeleton-points A, B. If they are 4-connected, their distance is $d(A,B) = 1$. Otherwise their distance is $d(A,B) = \sqrt{2}$. Let $r_A = r_B$, then fig. 30a, 30c show the two possible objects. The distance between the points is measured in pixels, thus it can be seen that the distance in fig. 30a is $d(A,B) = 1$, and the distance in fig. 30c is $d(A,B) = \sqrt{1^2 + 1^2} = \sqrt{2}$.

Now let the radius $r_B \leqslant r_A - d(A,B)$, it can easily be seen that the area of B is completely in A. Since $A \neq B$, and $a(B) \subset a(A)$ it touches *at most* 1 point on the boundary of A, thus *at most* 1 point of the boundary of the object. The definition of a skeleton-point is the center of the maximal disc in an object, touching *at least* 2 points on the boundary ↯

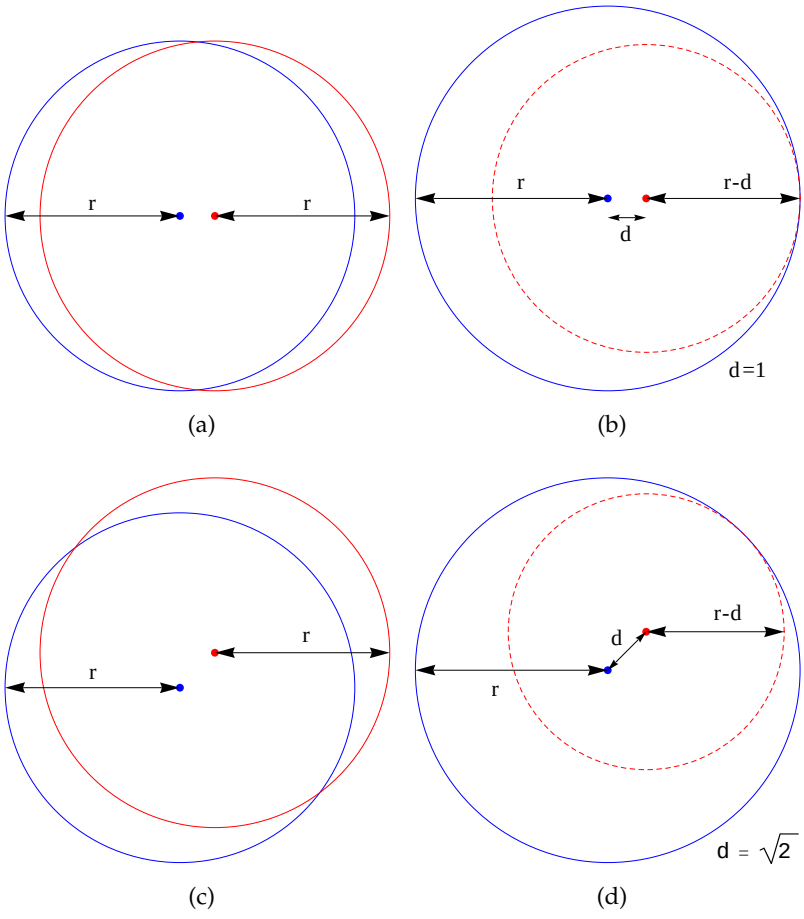


Figure 30: Different configurations for skeleton sizes

B

PAINTING EFFECT

This appendix provides an example of an image which has a striking painterly rendering effect when compressed using our SIR-algorithm.

Original:

SIR-format:

BIBLIOGRAPHY

[1] Oswin Aichholzer and Franz Aurenhammer. Straight skeletons for general polygonal figures in the plane. In Jin-Yi Cai and Chak Wong, editors, *Computing and Combinatorics*, volume 1090 of *Lecture Notes in Computer Science*, pages 117–126. Springer Berlin / Heidelberg, 1996. ISBN 978-3-540-61332-9. 10.1007/3-540-61332-3_144.

[2] Oswin Aichholzer, Franz Aurenhammer, David Alberts, and Bernd Gärtner. A novel type of skeleton for polygons. *Journal of Universal Computer Science*, 1(12):752–761, dec 1995. http://www.jucs.org/jucs_1_12/a_novel_type_of.

[3] J. Bankoski, J. Koleszar, L. Quillio, J. Salonen, P. Wilkins, and Y. Xu. Vp8 data format and decoding guide, 2011. URL http://tools.ietf.org/html/draft-bankoski-vp8-bitstream-06.

[4] H. Blum and Roger N. Nagel. Shape description using weighted symmetric axis features. *Pattern Recognition*, 10(3):167–180, 1978. ISSN 00313203. doi: 10.1016/0031-3203(78)90025-0. URL http://dx.doi.org/10.1016/0031-3203(78)90025-0.

[5] Jonathan W. Brandt, Anil K. Jain, and V. Ralph Algazi. Medial axis representation and encoding of scanned documents. *Journal of Visual Communication and Image Representation*, 2(2):151 – 165, 1991. ISSN 1047-3203. doi: 10.1016/1047-3203(91)90005-Z. URL http://www.sciencedirect.com/science/article/pii/104732039190005Z.

[6] Mathieu Desbrun, Mark Meyer, Peter SchrÃ¶der, and Alan H. Barr. Anisotropic feature-preserving denoising of height fields and bivariate data. In *In Graphics Interface*, pages 145–152, 2000.

[7] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. Bilateral mesh denoising. *ACM Trans. Graph.*, 22:950–953, July 2003. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/882262.882368. URL http://doi.acm.org/10.1145/882262.882368.

[8] Joint Photographic Experts Group. The "JPEG" Standard (ISO/IEC 10918-1, 1992. URL http://www.jpeg.org/jpeg/index.html.

[9] Joint Photographic Experts Group. Still picture interchange file format (spiff), 1992. URL http://www.jpeg.org/public/spiff.pdf.

[10] Eric Hamilton. Jpeg file interchange format, version 1.02. *Interchange*, 1992. URL http://www.dspace.cam.ac.uk/handle/1810/54.

[11] Thouis R. Jones, Frédo Durand, and Mathieu Desbrun. Non-iterative, feature-preserving mesh smoothing. *ACM Trans. Graph.*, 22:943–949, July 2003. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/882262.882367. URL http://doi.acm.org/10.1145/882262.882367.

[12] Renato Kresch and David Malah. Skeleton-based morphological coding of binary images. *IEEE Transactions on Image Processing*, pages 1387–1399, 1998.

[13] Ta-Chih Lee, Rangasami L. Kashyap, and Chong-Nam Chu. Building skeleton models via 3-d medial surface/axis thinning algorithms. *CVGIP: Graph. Models Image Process.*, 56:462–478, November 1994. ISSN 1049-9652. doi: 10.1006/cgip.1994.1042. URL http://dl.acm.org/citation.cfm?id=202862.202867.

[14] John Miano. *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*. SIGGRAPH series. Addison Wesley Longman, Inc, 1999. ISBN 0-201-60443-4. URL http://books.google.com/books?id=_nJLvY757dQC.

[15] Microsoft Corporation and IBM Corporation. Multimedia programming interface and data specifications 1.0, August 1991. URL http://www.digitalpreservation.gov/formats/fdd/fdd000025.shtml.

[16] R. L. Ogniewicz and O. KÃŒbler. Hierarchic voronoi skeletons. *Pattern Recognition*, 28(3):343 – 359, 1995. ISSN 0031-3203. doi: 10.1016/0031-3203(94)00105-U. URL http://www.sciencedirect.com/science/article/pii/003132039400105U.

[17] Igor Pavlov. LZMA SDK (Software Development Kit), July 2007. URL http://www.7-zip.org/sdk.html.

[18] William B. Pennebaker and Joan L. Mitchell. *JPEG Still Image Data Compression Standard*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 1992. ISBN 0442012721.

[19] I. Pitas and A.N. Venetsanopoulos. Morphological shape representation. *Pattern Recognition*, 25(6):555 – 565, 1992. ISSN 0031-3203. doi: 10.1016/0031-3203(92)90073-R. URL http://www.sciencedirect.com/science/article/pii/003132039290073R.

[20] Freek Reinders, Melvin E. D. Jacobson, and Frits H. Post. Skeleton graph generation for feature shape description. In *IN JOINT EUROGRAPHICS–IEEE TCVG SYMPOSIUM ON VISUALIZATION*, pages 73–82. Springer Verlag, 2000.

[21] Joseph M. Reinhardt and William E. Higgins. Toward efficient morphological shape representation. In *Proceedings of the 1993 IEEE international conference on Acoustics, speech, and signal processing: image and multidimensional signal processing - Volume V*, ICASSP'93, pages 125–128, Washington, DC, USA, 1993. IEEE Computer Society. ISBN 0-7803-0946-4. URL http://dl.acm.org/citation.cfm?id=1947148.1947184.

[22] Joseph M. Reinhardt and William E. Higgins. Efficient morphological shape representation. *Transactions on Image Processing, IEEE*, 5:89 – 101, January 1996. ISSN 1057-7149. doi: 10.1109/83.481673.

[23] Dennie Reniers and Alexandru Telea. Tolerance-based feature transforms. In *Advances in Computer Graphics and Computer Vision (Intl. Confs. VISAPP and GRAPP, Setubal, Portugal, Feb. 25-28, 2006, Revised Selected Papers)*, volume 4, pages 107–114. Springer LNCS, 2008. URL http://www.win.tue.nl/~alext/ALEX/PAPERS/CGVTA07/chapter.pdf.

[24] Lionel Reveret, Laurent Favreau, Christine Depraz, and Marie-Paule Cani. Morphable model of quadrupeds skeletons for animating 3d animals. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 135–142, New York, NY, USA, 2005. ACM. ISBN 1-59593-198-8. doi: http://doi.acm.org/10.1145/1073368.1073386. URL http://doi.acm.org/10.1145/1073368.1073386.

[25] Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, 1983. ISBN 0126372403.

[26] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proc. Nat. Acad. Sci*, pages 1591–1595, 1995.

[27] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science ... on Applied and Computational Mathematics)*. Cambridge University Press, 2 edition, June 1999. ISBN 0521645573. URL http://math.berkeley.edu/~sethian/2006/Publications/Book/2006/OnLine/book_sethian.html.

[28] Hüseyin Tek and Benjamin B. Kimia. Boundary smoothing via symmetry transforms. *J. Math. Imaging Vis.*, 14:211–223, May 2001. ISSN 0924-9907. doi: 10.1023/A:1011229911541. URL http://dl.acm.org/citation.cfm?id=570162.570164.

[29] Alexandru Telea. Feature preserving smoothing of shapes using saliency skeletons. In *Visualization and Mathematics (Proc. VMLS'09)*, 2011. Accepted for publication.

[30] Alexandru C. Telea and Jarke J. Van Wijk. An augmented fast marching method for computing skeletons and centerlines. In *in Proc. of the Symposium on Data Visualisation (VisSym'02)*, pages 251–259, 2002.

[31] P.E. Trahanias. Binary shape recognition using the morphological skeleton transform. *Pattern Recognition*, 25(11): 1277 – 1288, 1992. ISSN 0031-3203. doi: 10.1016/0031-3203(92)90141-5. URL http://www.sciencedirect.com/science/article/pii/0031320392901415.

[32] Y.F. Tsao and K.S. Fu. A 3d parallel skeletonwise thinning algorithm pictures. In *Pattern Recognition and Image Processing*, pages 678–683, 1982.

[33] Christian J. van den Branden Lambrecht. *Vision models and applications to image and video processing*. Springer; 1st edition, 2001. ISBN 0792374223, 978-0792374220. URL http://books.google.com/books?id=_s2BlAeR66YC&pg=PA209#v=onepage&q&f=false.

[34] Lawson Wade and Richard E. Parent. Fast, fully-automated generation of control skeletons for use in animation. In *Proceedings of the Computer Animation*, CA '00, pages 164–169, Washington, DC, USA, 2000. IEEE Computer Society. URL http://dl.acm.org/citation.cfm?id=872743.872901.

[35] Lawson Wade and Richard E. Parent. Automated generation of control skeletons for use in animation. *The Visual Computer*, 18:97–110, 2002. ISSN 0178-2789. URL http://dx.doi.org/10.1007/s003710100139. 10.1007/s003710100139.

[36] Zhou Wang and A. C. Bovik. Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures. *IEEE Signal Processing Magazine*, 26(1):98–117, January 2009. ISSN 1053-5888. doi: 10.1109/MSP.2008.930649. URL http://dx.doi.org/10.1109/MSP.2008.930649.

[37] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 13 (4):600–612, 2004.

[38] Li Yu and Runsheng Wang. Shape representation based on mathematical morphology. *Pattern Recogn. Lett.*, 26:1354–1362, July 2005. ISSN 0167-8655. doi: http://dx.doi.org/10.1016/j.patrec.2004.11.013. URL http://dx.doi.org/10.1016/j.patrec.2004.11.013.

[39] Hamidreza Zaboli, Mohammad Rahmati, and Abdolreza Mirzaei. Shape recognition by clustering and matching of skeletons. *JCP*, 3(5):24–33, 2008.

[40] T. Y. Zhang and C. Y. Suen. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, 27:236–239, March 1984. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/357994.358023. URL http://doi.acm.org/10.1145/357994.358023.

[41] Z. Zhou and A.N. Venetsanopoulos. Morphological skeleton representation and shape recognition. *Acoustics, Speech, and Signal Processing*, 2:948 – 951, 1988. ISSN 1520-6149. doi: 10.1109/ICASSP.1988.196747.