DENSE MEDIAL DESCRIPTORS FOR IMAGE COMPRESSION AND MANIPULATION

JIEYING WANG
Cover: The left half of the cover image comes from a brain CT slice. The right half shows one of its threshold sets and the corresponding Spline Medial Axis Transform structure, with each separate spline and its control points colored separately.

Dense Medial Descriptors for Image Compression and Manipulation

Jieying Wang
PhD Thesis

The research for this dissertation was conducted at the Scientific Visualization and Computer Graphics (SVCG) research group, part of the Bernoulli Institute (BI) and the Faculty of Science and Engineering (FSE) at the University of Groningen, the Netherlands.
Dense Medial Descriptors for Image Compression and Manipulation

PhD thesis

to obtain the degree of PhD at the University of Groningen on the authority of the Rector Magnificus Prof. C. Wijmenga and in accordance with the decision by the College of Deans.

This thesis will be defended in public on Tuesday 27 September 2022 at 14.30 hours

by

Jieying Wang
born on May 29th, 1992 in Shandong, China
Supervisors
Prof. A. C. Telea
Prof. J. Kosinka

Assessment committee
Prof. J. B. T. M. Roerdink
Prof. E. L. van den Broek
Prof. G. R. Renardel de Lavalette
ABSTRACT

Innumerable images are created, manipulated, transmitted, and stored every day. With the rapid development of the internet and multimedia, the resolution, diversity, and applications of image data have grown unprecedentedly. As such, the demand for efficient and effective image compression and manipulation is ever increasing.

Many image compression methods exist, ranging from early transform domain coding to deep neural network methods that have attracted interest in recent years. In this thesis, we explore a class of methods that encode images represented as threshold sets by using their dense medial descriptors (DMD), building on prior research that used DMD for image analysis and segmentation.

We start by making several improvements to the original method, which confirm that DMD can offer interesting and viable possibilities to compress grayscale and color images. Next, we combine DMD with the information provided by spatial saliency maps to perform non-uniform and optionally user-driven simplification of images. These two methods show that dense medial descriptors represented by raster models are effective tools for image compression. To further improve the efficiency of compression, we next apply a vector representation: piecewise B-splines encoding the medial descriptors (also called skeletons). Experiments and evaluations on binary, grayscale as well as color images show that DMD modeled by vector representations are also effective and efficient tools for image compression. We next incorporate the saliency maps into the vector-based dense medial descriptors for spatially-dependent image simplification and compression. Through these combined efforts, we achieve better trade-offs of image quality vs compression than JPEG on a wide variety of image types, and produce comparable results to JPEG 2000 and BPG on certain image types.

In addition to image compression, we also implement super-resolution image generation, multiscale medial axis transform representations, and image deformation. To perform these image manipulations more conveniently, we have developed an interactive tool which combines the icicle representation of morphological trees of images with spline-based dense medial descriptors. Additional applications, including watermark removal, image deformation, and lighting change simulation, show that spline-based dense skeletons, when combined with morphological trees, are also effective and efficient tools for manipulating grayscale or color images. This allows us to conclude that dense skeletons are a valuable addition to the toolkit of image processing researchers and practitioners.
SAMENVATTING

Veel beelden worden dagelijks gecreëerd, gemanipuleerd, verstuurd en opgeslagen. De resolutie, diversiteit en toepassingen van beelden zijn enorm gegroeid met de de snelle ontwikkeling van internet en multimedia. Daarom groeit de vraag gestaag naar efficiënte en effectieve beeldcompressie en -manipulatie methodes.

Veel beeldcomprissiemethodes bestaan – van vroegtijdige transformdomeincodering tot diepe neurale netwerken in recente jaren. In dit proefschrift analyseren we een klasse van methodes die beelden coderen via zogenaamde threshold sets met gebruik van hun dichte mediale descriptoren (DMD), bouwden op eerder onderzoek van DMD voor beeldanalyse en -segmentatie.

We presenteren eerst verbeteringen voor de oorspronkelijke methode die laten zien dat DMD interessante mogelijkheden biedt voor de compressie van grijswaarde en kleurbeelden. Vervolgens combineren wij DMD met zogenaamde ruimtelijke saliency maps om niet uniforme en/of gebruikergeredene beeldsimplificatie te realiseren. Dit werk laat zien dat DMD, gecodeerd als raster modellen, effectief zijn voor beeldcompressie. We verbeteren de compressie verder door een vector representatie met stuksgewijze B-splines die de mediale descriptoren (ook skeletten genoemd) coderen. Experimenten en evaluaties op binaire, grijswaarde en kleurbeelden laten zien dat DMD met vectormodellen zijn ook efficiënt en effectief voor beeldcompressie. Vervolgens voegen wij saliency maps toe aan onze vectorgebaseerde DMD voor ruimteafhankelijke beeldsimplificatie en -codering. Al deze technieken leiden tot een beter balans van beeldkwaliteit en compressie dan JPEG voor een groot bereik van beelden en leiden zelfs tot vergelijkbare resultaten met JPEG 2000 en BPG voor sommige beeldtypes.

Naast beeldcompressie implementeren wij ook superresolutie beeldgeneratie, multischaal medial axis transform representaties en beeldvervorming. We voeren uit deze beeldmanipulaties op een effectieve manier door het ontwikkelen van een interactief programma dat de icicle representatie van morfologische bomen combineert met splinegebaseerde DMD. Bijkomende toepassingen zoals watermerk verwijderen, beelddeformatie en belichtingsveranderingssimulaties laten zien dat splinegebaseerde DMD samen met morfologische bomen ook effectief en efficiënt zijn voor grijswaarde- of kleurbeeldmanipulatie. Onze conclusie is dat dichte skeletten een waardevolle toevoeging zijn aan het gereedschapskist van beeldbewerking voor onderzoekers en eindgebruikers.
This thesis is the result of the following publications:


This thesis is also based on the following papers which are currently under review:

- J. Wang, J. Kosinka, and A. Telea. Spline-Based Dense Medial Descriptors for Image Simplification Using Saliency Maps. Submitted to *Communication in Computer and Information*.

Images are everywhere. They are generated by a wide range of applications spanning Earth sciences, medical sciences, engineering, the consumer and media industries and, last but not least, everyone of us who uses smart phones and social media. This unprecedented growth of the size, diversity, and applications of image data has been accompanied by a parallel growth of technologies that support the creation, storage, analysis, and processing of image content. Well-known examples of such technologies that are present in virtually all fields that rely on image data are image compression, image search, and image editing. As the usage of image data becomes increasingly widespread, the demand for techniques and tools to support working with such data continuously increases.

At the foundations of every technology or application that consumes (or produces) image data one has a so-called representation or model of such data. Simply put, this is a way for the respective technology to handle the various types of information that are present in, or created from, the actual images. Image representations span a wide spectrum, depending on the type of information that they aim to capture. For example, so-called image metadata (Caldelli et al., 2003; Gangwar and Pathania, 2018), or image tags, describe where, at which moment, and/or by whom an image was taken. Image features are descriptors extracted from the actual raw image information to characterize the type of scenery that the image depicts, and are widely used in content-based image retrieval (Kato, 1992; Veltkamp et al., 2001; Davies, 2004), image indexing (Rasmussen, 1997; Chu, 2001), and reverse image search (Gaillard and Egyed-Zsigmond, 2017; Diyasa et al., 2020). Image measurements quantize various properties of the image itself and/or specific shapes present in the image, and are used in numerous engineering applications for robotic control (Tawiah, 2020), quality assurance (Abdulkadir, 2020), medical diagnosis and prognosis (Tariq et al., 2021), and automatic production lines (Nagato et al., 2017). Image data can be also further refined and structured to extract higher-level information such as segments or specific shapes present in it (Yang et al., 2008). At an even higher level, images can be processed to extract semantic information concerning the types of activities they describe, with applications in surveillance (Zhang et al., 2018) or remote healthcare and monitoring (Bao and Bao, 2021).

However diverse, all these types of refined, or derived, information extracted from images need a representation of the basic data present
at the lowest level to build upon. Two such main representations exist, as follows.

### 1.1 Image Representations

The *raster* representation describes the raw pixels an image consists of. More formally put, this representation is a (typically) uniform-grid, regular, sampling of the continuous grayscale or color field that an imaging device has acquired. Raster representations are ubiquitous in image applications. They are very simple to implement, store, exchange, and process, and lend themselves naturally to parallel processing, which can significantly accelerate the execution of image processing operations. Raster representations are familiar to most users and related to so-called image formats such as JPEG (Wallace, 1992), GIF (CompuServe, 1987), or PNG (Boutell, 1997). However, raster representations for high-resolution sampling can take significant amounts of memory to store. Separately, this representation is fundamentally a low (possibly, the lowest) level one. Manipulating higher-level concepts or entities present in raster representations, such as shapes present in the respective images, requires non-trivial ways to extract such entities from the raw image data.

At the other end of the spectrum, the *vector* representation describes actual geometric shapes present in an image. More formally put, vector representations capture a family of geometric primitives, such as curves, shading gradients, textures, by analytic representations parameterized by a number of so-called degrees of freedom. Vector representations are well known and widely used in the creative industry (Barla and Bousseau, 2013), computer-assisted geometric design (Paoluzzi et al., 2003), and engineering fields. Compared to raster representations, they offer a much more precise way to describe the underlying information (shapes) present in an image. Additionally, this representation is often considerably more compact than the uniform sampling proposed by raster images, leading to significant reductions of storage cost. Last but not least, vector representations allow users to directly manipulate the higher-level entities that they describe, or putting it simpler, make image editing a faster and easier process.

Given the above complementary characteristics of raster and vector representations, researchers have been interested to find ways to join their advantages in a single model. Such a model was actually proposed decades ago by the computer vision and shape analysis community for so-called binary images – that is, images that consist of only foreground (or object) and background pixels, for example, black-and-white images. In such images, the foreground pixels typically represent the shapes of interest, while the background pixels are used to embed such shapes in the uniform grid structure required by the raster model. The aforementioned model consists of so-called *skeletons* or medial axes. Informally
put, these are pixel-thin, connected, curve-like structures that are locally centered within the foreground shape. Skeletons, together with the distances of their pixels to the closest pixels of the boundary separating the foreground shape from the background – more formally put, the so-called Medial Axis Transform (MAT) of the shape (Blum, 1967) – are dual representations of shape. That is, all information present in the shape can be derived from its skeleton and vice versa. However, the curve-like structure of skeletons makes several operations on shape, such as characterization, part-whole structure analysis, matching, comparison, denoising, simplification, and segmentation, far easier to describe and implement than when using the raw representation. Importantly, the curve-like representation of skeletons is very similar to primitives used by the vector model such as splines. Skeletons have been used in numerous science fields such as computer vision (Tam and Heidrich, 2003; Reniers and Telea, 2008; Bai and Latecki, 2008), robotic navigation (Garrido et al., 2006b,a; Mirtich and Canny, 1992), medical imaging (Morse et al., 1993), and information visualization (Ersoy et al., 2011; van der Zwan et al., 2016). Many accurate and computationally efficient algorithms exist to extract skeletons from raster images, see Sec. 2.1.2 for details.

1.2 SKELETONS FOR IMAGE DATA

Despite their widespread use and understanding, skeletons have so far been applied mainly to binary images. This is natural since they require the existence of shapes, thus, a clear separation between foreground and background. Recently, however, researchers have proposed to extend skeletons to grayscale and color images (Van Der Zwan et al., 2013; Terpstra, 2017). This so-called dense skeleton image representation essentially extracts binary shapes defined by all luminance threshold-sets (see Fig. 1.1(b)) in an image and next constructs a stack of skeletons (Fig. 1.1(c)), one per luminance layer. The original image can be next reconstructed from the dense skeletons (Fig. 1.1(d) and (e)). Following the earlier introduced metaphor, we can say that dense skeletons are a dual representation of grayscale (or color) images. The results shown in Van Der Zwan et al. (2013); Terpstra (2017) support this claim from a practical viewpoint, showing how one can reconstruct high-accuracy color and grayscale images from their dense skeleton representations. Additionally, the respective works show some applications of dense skeletons for image segmentation, simplification, and nonphotorealistic rendering.

However, while the above works showed that dense skeletons are, technically, a dual representation of images, they did not explore in depth why such a representation would be useful and usable. Concerning usefulness, we believe that dense skeletons have many more potential applications. Concerning usability, such applications need to be
tested extensively to gain insights on the actual extent of the performance of dense skeleton based methods.

One such application regards image compression and simplification. The two are closely related techniques in the toolset of the imaging practitioner: Compression creates images of a smaller file size for archiving, transmission, and rendering purposes; lossy compression achieves this by removing certain image details, though typically not in an explicitly user-controlled manner. Simplification creates images which keep visual structures of interest to the use-case at hand, and remove the other, less important, structures, to ease further analysis and processing of such images; simplification also achieves image-size reduction, although as a by-product rather than a key goal. In this thesis, we mainly discuss image compression since it has been a well-studied field. While a few early results highlighting the potential of dense skeletons for compression were listed in Van Der Zwan et al. (2013); Terpstra (2017), it is clear that the method, as presented there, is not competitive with modern image compression methods such as JPEG (Wallace, 1992) or its newer variant JPEG2000 (Taubman and Marcellin, 2001). One topic that we explore in this thesis is how dense skeletons can be adapted to reach this competitiveness level, and demonstrate this extensively on a wide collection of images of different types.

Another application regards image manipulation. Skeletons are a simple, compact, and low-dimensional representation of shape. As such, they have been used in various contexts to perform manipulations and editing of binary shapes (Bloomenthal and Lim, 1999) e.g. in the design or animation domains (Vasilakis and Fudos, 2009). An open question that we aim to answer in this thesis is whether the dense skeleton extension of classical binary-image skeletons is also useful and usable for performing similar manipulations of color or grayscale images.

Finally, let us make the connection between the raster and vector representations of images. As stated earlier, binary-image skeletons consist of a set of pixel-based curves. As such, it seems natural to represent these curves with vector models. This would likely support both applications outlined above. Vector representations are more compact than pixel-chain ones, hence, they should support compression. Also, ma-
Manipulating vector representations of curves via control points is a well-proven method in computer aided editing (Orzan et al., 2013; Kosinka et al., 2015). Hence, vector representations of skeletons should support image manipulation. Moreover, vector representations are resolution-independent. Hence, they can be effective tools for creating so-called super-resolution representations of images at a variety of scales. In this thesis we explore such vector representations both for the classical binary-image skeletons as well as for the dense skeletons.

1.3 Research Question

Summarizing the above goals, we can at this point state our central research question:

*Are dense skeletons, represented suitably by raster or vector models, effective and efficient tools for image compression and manipulation?*

We refine and address this question as follows.

In Chapter 2, we introduce definitions and related work on skeletons, image compression, and quality metrics. In particular, we highlight efficient and effective skeletonization methods and image quality metrics which are frequently used in our research. This material serves as a foundation based on which our developments, covered by the subsequent chapters, are presented.

Chapter 3 explores the usage of dense medial descriptors for image compression. We present several enhancements of the original method proposed in Van Der Zwan et al. (2013) for simplifying and compactly storing dense skeletons with the aim of obtaining an optimal ratio between image quality and the amount of information reduction (compression) achieved. We evaluate our results on a wide set of images of different types. The work in Chapter 3 shows that dense skeletons do have significant potential for image compression. However, it also shows that the obtained results vary widely as a function of the kind of image under study. Separately, the chapter introduces the methodology we use for exploring and benchmarking the efficiency and effectiveness of dense skeletons. We reuse and extend this methodology in the subsequent chapters.

Chapter 4 explores the usage of dense skeletons for non-uniform, user-driven simplification of images. Specifically, we use saliency maps that encode the relative importance of different parts of an image to guide the image simplification (and thereby image compression). We argue that these so-called spatial saliency dense medial descriptors are useful in applications where one desires different simplification levels for different zones of an image. We show that this method is effective in preserving small-scale, but visually important, details of natural images.
without trading off the compression power. Rather, the method directs the compression effort to specific areas of interest in the image.

Chapters 3 and 4 use raster representations for the dense skeletons. While these showed good results of the trade-off between image quality and compression, we also found inherent limitations in how far raster representations can push compression. Chapter 5 aims to push this limit further, and for this introduces the use of vector representations for skeletons. To do this, we consider here the simplest case – that of skeletons of binary images. We show that these can be accurately and efficiently modeled with B-splines in a way that enables users to control the trade-off between precision and compression. We compare our proposal to a recent related method that uses vector representations for binary skeletons and show the advantages of our method in terms of simplicity, ease of use, versatility, computational speed, and accuracy. Also, we show how our vector-based skeletons can be used to create superresolution versions of binary images.

Chapter 6 extends the vector representation proposed in Chapter 5 from binary images to grayscale and color images or, in other words, proposes a vector representation for dense skeletons. We show how this representation performs on a wide variety of image types and demonstrate that it can lead to better trade-offs of image quality vs compression than JPEG and, for certain image types, also than JPEG 2000. Following the last use-case of Chapter 5, we also show how our method can generate superresolution results from color and grayscale images. Evaluation on a benchmark composed of 100 images shows that our SDMD method can generate comparable superresolution results with state-of-the-art deep learning-based approaches.

Chapter 7 combines the techniques proposed in Chapters 4 and 6 by presenting the addition of saliency maps to vector based dense skeletons. Since both the usage of saliency maps and that of vector based dense skeletons helps image compression, our combined method only increases this desirable property further. We demonstrate our proposal by comparing it with JPEG and JPEG 2000 and show improved results as compared to the method presented in Chapter 6.

Chapters 3–7 explore the first half of the research question extensively and exhaustively, and draw the conclusion that dense skeletons, represented suitably by raster or vector models, are effective and efficient tools for image compression. Chapter 8 aims to answer the second half of the research question, i.e., it explores the potential of dense skeletons represented by vector models for image manipulation and editing. To this end, we design an interactive tool which combines the icicle representation of morphological trees of images with spline-based dense medial descriptors. Various application examples show that spline-based dense skeletons, when combined with morphological trees, are useful and usable tools for manipulating color or grayscale images.
Finally, Chapter 9 summarizes our results and relates them back to the central research question of the thesis, and also outlines potential directions for future work on using vector based dense skeletons for additional application areas.
As outlined in Chapter 1, our main research question regards the potential of using medial descriptors to implement image simplification and compression. As such, we review related work on medial descriptors (Sec. 2.1), image compression methods (Sec. 2.2), and metrics to evaluate compression quality (Sec. 2.3) in this chapter.

2.1 MEDIAL DESCRIPTORS

Medial descriptors, also known as medial axes, or more generally the skeletons, are efficient and effective tools for a wide range of computer science applications, including shape recognition (Shen et al., 2016; Ayzenberg, 2019), shape segmentation (Shah, 2005; Reniers and Telea, 2007, 2008), matching and retrieval (Bai and Latecki, 2008; Goh, 2008; Sundar et al., 2003; Xie et al., 2008), and shape simplification (Tam and Heidrich, 2003; Hajdu et al., 2007) and denoising (Telea, 2012; Schubert et al., 2020). The process of computing skeletons from given binary shapes is called skeletonization. Conceptually, skeletonization generates m-dimensional descriptors from an n-dimensional shape $\Omega \subset \mathbb{R}^n$ (Reniers, 2009). In this thesis, we explore the potential of medial descriptors for 2D image compression. Thus, we focus on $S^{2,1}$, or, in other words, skeletons of two-dimensional binary shapes. For readers interested in 3D skeletonization $S^{3,2}$ (surface skeletons) and $S^{3,1}$ (curve skeletons) and their applications, we refer to Tagliasacchi et al. (2016); Cornea et al. (2007); Saha et al. (2016). In the remainder of this section, we overview the definitions of medial descriptors (Sec. 2.1.1), skeletonization methods (Sec. 2.1.2), skeleton regularization methods (Sec. 2.1.3), and methods for shape reconstruction from medial descriptors (Sec. 2.1.4).

2.1.1 Definitions

Medial descriptors, first introduced by Blum (Blum, 1967), were defined as the loci of centers of maximal discs contained in a shape $\Omega \subset \mathbb{R}^2$, as shown in Fig. 2.1 (a). A maximally inscribed disc is completely inside the shape $\Omega$ and is not fully included in another inscribed disc. While such a definition is simple and intuitive, it is difficult to derive an effective way for computing the skeleton in practice from it. Some methods have been able to directly use this definition to compute skeletons (Jalba et al., 2013). However, this typically entails the use of brute-force computation to search for the centers of maximally inscribed discs.
As an alternative, another more commonly used definition, the grassfire analogy, was proposed in Leymarie and Levine (1992). Imagine $\Omega \subset \mathbb{R}^2$ as a compact, uniformly growing grass field whose entire boundary $\partial \Omega$ catches fire at the exact same time $t_0$. The flame propagates from $\partial \Omega$ to the interior of $\Omega$ with an isotropic velocity (constant speed) along the internal normal of $\partial \Omega$. The locations where fire fronts coming from different parts of $\partial \Omega$ meet and quench define the skeleton $S_\Omega$ of $\Omega$. The arrival time $t > t_0$ of the front is positively related to the inscribed disc radius. The grassfire model definition is depicted in Fig. 2.1 (b). This definition leads to direct techniques, e.g., morphological thinning methods, for computing skeletons, as described next in Sec. 2.1.2.

In the grassfire definition, the medial (skeletal) point, i.e., the quenching point, is always generated by the meeting of at least two fire fronts. Since the grassfire propagates isotropically, a skeleton point always associates with at least two different closest points on the boundary $\partial \Omega$, which are called feature points (Meijster et al., 2002; Hesselink and Roerdink, 2008). This model is shown in Fig. 2.1 (c). These feature points, $f_1, f_2$, are exactly the tangent points between the maximally inscribed disc and $\partial \Omega$ in Blum’s definition (Fig. 2.1 (a)). Using these feature points, the skeleton $S_\Omega$ of $\Omega$ can be defined as

$$S_\Omega = \{ x \in \Omega \mid \exists f_1, f_2 \in \partial \Omega, f_1 \neq f_2: ||f_1 - x|| = ||f_2 - x|| = DT_\Omega(x) \}.$$ (2.1)

Here, $DT_\Omega$ is the so-called distance transform of $\Omega$, which is defined as

$$DT_\Omega(x \in \Omega) = \min_{y \in \partial \Omega} ||x - y||.$$ (2.2)

The notation $|| \cdot ||$ in Eqn. 2.1 and Eqn. 2.2 denotes the Euclidean distance in $\mathbb{R}^2$. Intuitively, $DT_\Omega(x)$ indicates the shortest distance from $x$ to $\partial \Omega$. Thus, $DT_\Omega(x)$ monotonically increase as $x$ goes from $\partial \Omega$ towards the interior of $\Omega$; see the example shown in Fig. 2.2 (b).
Figure 2.2: A rectangular shape with random noise added to the boundary (a) and its distance transform shown by color-coding and isolines (b), and its corresponding medial axes generated by the Fast Marching Method (FMM) (Sethian, 1996).

For completeness, we note that the concept of feature points defines a so-called feature point transform

$$ FT_Ω(x ∈ Ω) = \arg \min_{y ∈ ∂Ω} \|x − y\|, \quad (2.3) $$

which associates to each point $x$ inside $Ω$ the set of its feature points. By definition, a point $x ∈ Ω$ is on $S_Ω$ if $FT_Ω(x)$ yields at least two points; non-skeletal points $x ∈ Ω$ have a $FT_Ω(x)$ yielding a single point; and skeleton bifurcations and branch endpoints $x ∈ S_Ω$ have a $FT_Ω(x)$ that yields more than two points. Analyzing the feature transform has proven useful in many applications of skeletons such as shape segmentation (Kustra et al., 2015; Feng et al., 2016).

The tuple formed by the distance transform and skeleton of a shape, denoted

$$ MAT(Ω) = (S_Ω, DT_Ω), \quad (2.4) $$

is called the Medial Axis Transform (MAT) of the shape $Ω$. More exactly, the MAT records all the points $x ∈ S_Ω$ together with their distance transform values $DT_Ω(x)$. As we shall see later in Sec. 2.1.4, the shape $Ω$ can be reconstructed from its MAT. Since the MAT can be obviously computed from $Ω$, one says that the MAT is a dual representation of the shape.

2.1.2 Skeletonization techniques

In Sec. 2.1.1, we described three alternative definitions of medial descriptors. More alternative definitions which lead to the same descriptor are presented in Tagliasacchi et al. (2016); Siddiqi and Pizer
We next overview the most prevalent classes of skeletonization techniques which use various forms of these alternative definitions to compute $S_Ω$.

**Morphological thinning methods** simulate the grassfire evolution (Fig. 2.1 (b) and Fig. 2.2 (b)) by iteratively eroding $Ω$ inwards with constant speed until left with a one-pixel-thin connected structure representing $S_Ω$ (Beucher, 1994; Peter and Breuß, 2013). While relatively straightforward to implement and fast, thinning methods generate different results due to different pixel removal orders. Thus, such methods do not in general guarantee that $S_Ω$ is centered within $Ω$, i.e., $DT_Ω$ can be poorly approximated (Lam et al., 1992; Pudney, 1998). Another deficiency is that it is difficult to prune the skeleton (see next in Sec. 2.1.3) based on the feature points which are typically not stored during the thinning process.

**Geometric methods** (Ogniewicz and Kübler, 1995; Attali and Montanvert, 1997) find $S_Ω$ as a subset of the edges of the Voronoi diagram of a piecewise-linear (polyline) representation of $∂Ω$, as illustrated in Fig. 2.3. Voronoi diagrams decompose $Ω$ into cells based on sites, where each point in a cell is closer to that cell’s site than to any other site. Sites are a dense discrete set of sampling points on $∂Ω$ (Fig. 2.3 (a)), i.e., the endpoints of the segments forming the above mentioned polyline representation of the boundary. Sampling criteria play an important role in the quality of the obtained skeletons. The more sampling points on $∂Ω$, the more spurious branches the skeleton has, which need to be eliminated next. Conversely, the fewer sampling points are used on $∂Ω$, the worse can this sampling capture fine-scale details of the shape boundary. In practice, the sampling density is determined by the local feature size (Amenta and Bern, 1999; Zhu et al., 2014) to capture the boundary topology faithfully. The strongest point of geometric methods is that they fully work with a vector representation of both $∂Ω$ and $S_Ω$ which, as outlined in Chapter 1, is advantageous for many reasons. Although also very accurate, compact, and connected in representation, geometric approaches are rather complex to implement, require a robust boundary discretization, and are computationally expensive (Telea and van Wijk, 2002).

**Distance field methods** compute $DT_Ω$ (Eqn. 2.2) from $∂Ω$, and next find $S_Ω$ along singularities of $DT_Ω$ (Kimmel et al., 1995; Sethian, 1996; Telea and van Wijk, 2002; Falcão et al., 2004; Meijster et al., 2002; Hesselink and Roerdink, 2008). To start with, these methods need to compute $DT_Ω$. A well known technique for this is the Fast Marching Method (FMM) introduced by Sethian (Sethian, 1996) as an $O(n \log(n))$ algorithm (for $n$ pixels used to discretize $Ω$) to solve the Eikonal equation $|∇ DT_Ω| = 1$. Other, even faster, techniques compute the exact Euclidean
distance transform in time linear in the pixel count $n$ (Meijster et al., 2002).

However, the detection of the singularities of the $DT_{\Omega}$ field is non-trivial. Direct computation of singularities does not guarantee connected, one-pixel-thin skeletons (Niblack et al., 1990; Bouix and Siddiqi, 2000; Reinders et al., 2000), and is numerically unstable (Saha et al., 2016). As shown in Fig. 2.2 (c), perturbations along $\partial \Omega$ introduce many so-called spurious medial branches. To address this, Telea and van Wijk (2002) proposed the Augmented Fast Marching Method (AFMM), which is simple to implement, behaves robustly to boundary noise, and delivers connected skeletons. Key to this technique is tracking, for every point $x \in \Omega$ visited by the FMM during the computation of $DT_{\Omega}$, the ‘span’ of the boundary $\partial \Omega$ that is delimited by the feature points in $FT_{\Omega}(x)$. For non-skeletal points, this span is exactly one pixel. For skeletal points on branches that correspond to small-scale bumps on $\partial \Omega$, this span is small – equal to the arc-length of the boundary fragments corresponding to those bumps. Points located increasingly deeper along the skeleton have increasingly large spans. As such, the AFMM is a method for both skeleton detection and regularization – the latter is discussed in more detail in Sec. 2.1.3. The AFMM technique can be further accelerated on the GPU, yielding real-time skeletonization computation (Cao et al., 2010; Telea, 2014). Therefore, we next adopt this GPU-based distance-field-and-skeleton (thus MAT) computation approach for our work in this thesis.

2.1.3 Regularization methods

As stated in Sec. 2.1.2, medial axes $S_{\Omega}$ estimated directly from skeletonization approaches are notoriously unstable (Saha et al., 2016): Small perturbations along $\partial \Omega$, created e.g. by sampling inherent to both raster and vector representations, introduce spurious medial...
branches (Fig. 2.2 (c)), which contribute little (or not at all in practice) to the description of $\Omega$, but considerably complicate $S_\Omega$. Effort has been invested in regularizing or simplifying medial axes, by removing (parts of) the spurious branches, to make them stable. However, a simplified skeleton $\tilde{S}_\Omega$ cannot exactly represent, or encode, $\Omega$. Hence, accuracy (of representing a shape) and stability (of MAT computation) are related, but competing goals. We classify attempts to improve stability and accuracy into two groups, as follows.

**Reconstruction-based methods** approach the joint stability-accuracy problem by maximizing reconstruction accuracy, i.e., the difference between $\Omega$ and $\tilde{\Omega}$ (Attali et al., 2009). This can be estimated using the Hausdorff distance (Rote, 1991) between (sampled representations) of $\partial \Omega$ and $\partial \tilde{\Omega}$, defined as

$$H(\Omega, \tilde{\Omega}) = \max \{ h(\Omega, \tilde{\Omega}), h(\tilde{\Omega}, \Omega) \},$$

where $h(A, B)$ is the one-sided Hausdorff distance given by

$$h(A, B) = \max_{a \in A} \left\{ \min_{b \in B} \| a - b \| \right\}.$$

These methods (such as that of Zhu et al. (2014)) compute the simplified $\tilde{S}_\Omega$ by iteratively removing endpoints from $S_\Omega$, continuously checking their reconstruction error (Eqn. 2.5) and stopping when this reaches a user-allowed level. While yielding accurate medial axes due to their explicit goal of optimizing for $H$, reconstruction-based methods are computationally expensive.

**Medial-axis-based methods** aim mainly to compute a stable, or regularized, $S_\Omega$ by removing spurious branches from $S_\Omega$ following criteria that only use the information present in the MAT. Arguably the most successful class of such criteria computes a so-called importance $\rho(x)$ of every medial point $x \in S_\Omega$ as the boundary length between the feature points $f_1$ and $f_2$ of $x$. This is precisely the length of the so-called boundary ’span’ mentioned earlier in Sec. 2.1.2. Only medial points with $\rho(x)$ above a user-given threshold are taken over from $S_\Omega$ into $\tilde{S}_\Omega$. Importance thresholding is simple to implement for both raster (Falcão et al., 2004; Telea and van Wijk, 2002) and vector (Ogniewicz and Kübler, 1995; Attali and Montanvert, 1997) medial representations, delivers connected skeletons, and has an intuitive interpretation: The reconstructed shape $\tilde{\Omega}$ from $\tilde{S}_\Omega$ replaces all bumps along $\partial \Omega$ shorter than the threshold by circular arcs, effectively acting like a low-pass noise-boundary filter or multiscale representation of the shape and its skeleton.

However, the collapsed boundary length metric $\rho$ has a key limitation: It treats boundary details of a given size (scale) similarly. In practice, this can be undesirable. Consider e.g. the rightmost noisy rectangle in Fig. 2.4. Using $\rho$ to simplify this shape (and/or its skeleton, the
two being duals) will yield a shape where all small scale jaggies along the boundaries are eliminated, including rounding off the corners of the shape. Arguably, the latter is not desired. From a perceptual standpoint, the corners of the rectangle are more important, so we would like to have a way to obtain a shape with the jaggies along the edges removed but still having sharp corners, like the leftmost image in Fig. 2.4. For this goal, a modified version of the collapsed boundary length, called the salience metric (Telea, 2012) was proposed, as

\[ \sigma(x) = \rho(x)/\text{DT}_\Omega(x). \] (2.7)

Using this metric to regularize the skeleton, branches representing small-scale boundary bumps will be removed while branches that represent important (salient) corners are kept untouched. Figure 2.4 explores this insight by showing five rectangular shapes with randomly added noise of different scales on their boundary, and their simplified medial axes for \( \sigma > 1.5 \). We see that as the noise increases, the simplified medial axes change little, and are thus quite stable to noise.

Other skeleton-based regularization metrics include the angle between feature vectors (Attali and Montanvert, 1996; Foskey et al., 2003; Dey and Zhao, 2004; Hesselink and Roerdink, 2008), the divergence of the distance transform (Siddiqi et al., 2002), and higher order moments of the distance transform (Rumpf and Telea, 2002). Such metrics have proven very effective in producing simplified stable skeletons for 2D but also for 3D shapes (Tagliasacchi et al., 2016). However, they do not have the joint properties of multiscale representation/simplification, delivering a connected skeleton, and capturing directly an intuitive geometric property of the boundary that the collapsed boundary length \( \rho \) and its refinement, the saliency metric \( \sigma \) mentioned above, have. Moreover, the computation of \( \rho \) and \( \sigma \) do not add any overheads to their underlying skeletonization processes, as they directly use the distance and feature transform information that these processes need to compute. As such,
we next choose the salience metric $\sigma$ (Telea, 2012) to simplify skeletons in our work.

### 2.1.4 Shape reconstruction

As stated in Chapter 1 and earlier in Sec. 2.1.1, the pair $(S_\Omega, DT_\Omega)$, called the Medial Axis Transform (MAT), is a *dual* representation of shape: All information in a shape can be derived from its MAT and vice versa. Section 2.1.2 has illustrated three types of methods on how to extract the MAT from a shape. In turn, in this section, we discuss how to reconstruct a shape from its MAT.

**Reverse fast marching method** is exactly the reversed execution of the FMM depicted in Sec. 2.1.2. We evolve starting from each skeleton point $x \in S_\Omega$ outwards until reaching its distance transform value $DT_\Omega(x)$, which is equivalent to solving the Eikonal equation $|\nabla (-DT_\Omega)| = 1$, and obtain the reconstructed shape $\hat{\Omega}$ when $DT_\Omega = 0$. The advantages of this method are its computational efficiency ($O(n \log n)$ for a shape $\Omega$ of $n$ pixels) and the ability to stop the ‘inflation’ of the skeleton $S_\Omega$ towards $\Omega$ at any desired moment, based on local criteria. The latter can be important in applications where one wants to construct intermediate shapes between $S_\Omega$ and $\Omega$, see e.g. Rumpf and Telea (2002).

![Figure 2.5: Illustration of the medial discs envelope method. (a) Medial axis transform of a 2D shape $\Omega$. (b) The reconstructed shape $\hat{\Omega}$ is approximated as the envelope of medial discs. Image adapted from Zhu et al. (2014).](image)

**Medial discs envelope methods** reconstruct $\hat{\Omega}$ as the union $\bigcup_{x \in S_\Omega} B(x, DT_\Omega(x))$ of discs $B$ centered at pixels $x$ of the simplified skeleton $S_\Omega$ with skeletal-point radii given by the distance transform $DT_\Omega(x)$, as illustrated in Fig. 2.5. The disc $B$, called the *medial disc*, is the maximal inscribed disc in $\Omega$ as given by Blum’s definition (Fig. 2.1 (a)). The envelope, or boundary, of the union-of-discs yields thus the reconstructed shape boundary $\partial \hat{\Omega}$. Medial discs envelope methods can be efficiently
implemented on the GPU, see e.g. the detailed pseudocode in Meiburg (2011). As such, we next adopt this method to reconstruct a binary shape from its simplified skeleton.

2.2 IMAGE COMPRESSION METHODS

As our goal is to use medial descriptors to efficiently and effectively represent (encode) binary, grayscale, and color images; methods that aim for similar goals are of interest. We now briefly review the main classes of such methods.

Image compression is a well-studied field (Shum et al., 2003; Satone et al., 2017) which can be divided into two main classes: Lossless and lossy methods. Lossy compression has seen great interest due to its particularly high compression ratio (CR) while maintaining visual quality. In the past few decades, countless lossy compression approaches have been proposed. In the early days, transform domain coding dominated, such as the well-known discrete cosine transform (DCT) and related mechanisms used by JPEG (Wallace, 1992). DCT-based method divides the image into non-overlapping blocks for processing. When a high compression rate is desired, the results tend to show specific artifacts such as blocking or banding. To address this, the JPEG committee subsequently replaced the DCT’s block-based algorithm with wavelet transform and proposed the efficient JPEG 2000 (Taubman and Marcellin, 2001), which not only yields better compression performance than JPEG, but has significant flexibility in the codestream.

In recent years, Deep Neural Network (DNN) methods have attracted increasing interest due to their high compression rate and good quality. Important methods in this area use Recurrent Neural Networks (RNNs) (Toderici et al., 2016, 2017; Johnston et al., 2018) and autoencoders (Choi et al., 2019; Theis et al., 2017). Generative Adversarial Network (GAN) methods (Agustsson et al., 2019; Mentzer et al., 2020) have also been developed recently. However, all such approaches expose issues with the distortion metric that was used to train the networks (Mentzer et al., 2020). They can react in hard to predict ways to unseen data (images that are far from the types present during training). Besides, DNN methods require significant training data and training computational effort.

Let us position our research with respect to the above existing developments. Our dense skeleton-based methods, described in Chapters 3, 4, 6, and 7, do not aim to compete with the compression rates of the above DNN techniques. However, our explicit ‘feature engineering’ approaches offer more control over how images are simplified during compression, are fast, and do not require training data. Separately, technique-wise, we show, for the first time, that medial descriptors, represented suitably by raster or vector models, are useful and usable tools for image compression.
## 2.3 Image Compression Quality Metrics

Section 2.2 introduced a few main classes of methods for image compression. For all such methods, quality metrics are required to measure their performance. As stated earlier, lossy compression – our field of interest – has an inherent trade-off between how much we can compress a given image and how similar the compressed image will look to the raw, uncompressed, one. We thus distinguish two types of metrics to capture these two aspects, as follows.

First, we need to measure the amount of compression of an image. Let $I$ be a raw, uncompressed, image and $\tilde{I}$ be its compressed version, by whichever method we want to assess. The amount of compression can be computed by the so-called compression ratio metric $CR(I, \tilde{I}) \in \mathbb{R}^+$ which measures the size of the binary representation of $I$ divided by the size of the binary representation of $\tilde{I}$. In this model, the sizes of the images’ binary representations are simply the amount of bytes used to store the respective images, and depend on how the images are represented. For example, for a raw (uncompressed) image $I$ of $n \times m$ pixels, using $b$ bytes-per-pixel, the storage size will be $O(n \times m \times b)$. Storage sizes for the compressed representation $\tilde{I}$ strongly depend on what this representation actually is. Obviously, we want that $CR \gg 1$ for a good compression method.

Secondly, we need to measure how close the compressed image $\tilde{I}$ is to the original $I$, using so-called quality metrics $Q(I, \tilde{I}) \in \mathbb{R}^+$. Such metrics include the mean squared error (MSE) and peak signal-to-noise ratio (PSNR). While simple to compute and with clear physical meanings, these do not match well perceived visual quality (Wang and Bovik, 2009; Zhang et al., 2011, 2012). As visible in Fig. 2.6 (b), the JPEG compression of the original image (a), with a JPEG quality setting of 10%, is fuzzy, blocky, and has color quantization effects. Image (c) shows the result of one of our proposed compression methods, called SSDMD, discussed further in this thesis (Chapter 4). Although SSDMD’s compression result also looks possibly a bit fuzzy, it is, we argue, perceptually closer to the original image than the JPEG compression result. However, the PSNR metric values, listed in the figure, tells the opposite. Thus, PSNR may not correspond well with perceived quality.

The structural similarity (SSIM) index (Wang et al., 2004) alleviates the above issue by measuring, pixel-wise, how similar two images – an image $I$ and its reconstruction $\tilde{I}$, in our case – are by considering human perception, and is defined as

$$SSIM(I, \tilde{I}) = \frac{(2\mu_I \mu_{\tilde{I}} + C_1)(2\sigma_{I\tilde{I}} + C_2)}{\mu_I^2 + \mu_{\tilde{I}}^2 + \sigma_I^2 + \sigma_{\tilde{I}}^2 + 2\sigma_{I\tilde{I}} + C_2},$$

where $\mu_I$ and $\sigma_I$ are the mean intensity and the standard deviation of $I$, respectively, and similarly for $\tilde{I}$. In the above, $\sigma_{I\tilde{I}}$ is the covariance between $I$ and $\tilde{I}$. Also, $C_1 = (k_1L)^2$ and $C_2 = (k_2L)^2$, in which $L$ is
2.3 Image Compression Quality Metrics

Figure 2.6: Example of PSNR tending not to match perceived visual quality well.
(a) The original image. (b) JPEG compression with a quality of 10%.
(c) SSDMD compression (introduced in Chapter 4).

the dynamic range of the pixel values (255 for 8-bit grayscale images) and $k_1 = 0.01$ and $k_2 = 0.03$ are typical defaults to evaluate Eqn. 2.8. SSIM was extended to three-component SSIM (3-SSIM) by using non-uniform weights for the SSIM map over three region types: edges, texture, and smooth areas (Li and Bovik, 2010). Further on, multiscale SSIM (MS-SSIM) (Wang et al., 2003) is an advanced top-down interpretation of how the human visual system interprets images that considers variations of image resolution and viewing conditions, and is defined as

$$MS-SSIM(I, \tilde{I}) = \left[SSIM(I, \tilde{I})\right]^{\beta_m} \prod_{j=1}^{M-1} [c_j(I, \tilde{I})]^{\beta_j} \tag{2.9}$$

where $c_j$ is the contrast map $c(I, \tilde{I})$ iteratively downsampled by a factor of 2 on scale $1 \leq j \leq M$. $c(I, \tilde{I}) = (2\sigma_I\sigma_{\tilde{I}} + C_2)/(\sigma_I^2 + \sigma_{\tilde{I}}^2 + C_2)$. The exponent $\beta_j$ models the relative importance of different scales. Comprehensive evaluations (Sheikh et al., 2006; Ponomarenko et al., 2009) have demonstrated that SSIM and MS-SSIM can offer statistically much better performance in assessing image quality than other quality metrics. Therefore, we next consider either SSIM or MS-SSIM in this thesis to compare a raw image $I$ with its simplified, or compressed, representation $\tilde{I}$.

Having introduced related work on medial descriptors, image compression methods, and metrics to evaluate compression quality, we are now ready to present our first contribution in the area of dense medial descriptor compression.
From this chapter onwards, we start to elaborate on the dense medial descriptor-based image compression. We first propose several improvements to the original method, used mainly for image analysis, for simplifying and compactly storing dense medial descriptors with the aim of obtaining the best trade-off between image quality and compression. These enhancements include effective layer selection heuristics, a refined skeleton pixel-chain encoding, and a postprocessing compression scheme. We then propose a benchmark to assess the encoding power of dense skeletons for a wide set of natural and synthetic color and grayscale images. We use this benchmark to derive optimal parameters for dense skeletons. A comprehensive evaluation of a wide set of image types reveals that dense skeletons do have the compelling potential for image compression, achieving higher-compression ratios at similar quality to the well-known JPEG technique for certain types of images.

3.1 Introduction

As stated in Chapter 1 and Sec. 2.1, skeletons are well-known descriptors used for analysis and processing of 2D binary images. Recently, Dense Medial Descriptors (DMD) have been proposed as an extension of classical binary-image skeletons to allow the representation of grayscale and color images (Van Der Zwan et al., 2013). DMD extracts binary skeletons from all threshold sets (luminance, hue, and/or saturation layers) of an input image and allows the image to be reconstructed from these skeletons. By simplifying such skeletons and/or selecting a subset of layers, DMD effectively acts as a dual (lossy) image representation method. While DMD was applied for image segmentation, small-scale detail removal, and artistic modification (Van Der Zwan et al., 2013; Koehoorn et al., 2015; Sobiecki et al., 2015), it has not been used for image compression.

In this chapter, we exploit the simplification power of DMD for image compression, with two contributions. First, we propose Compressing Dense Medial Descriptors (CDMD), an adaptation of DMD for lossy image compression, by proposing better encoding and compression schemes for the skeletal information. Secondly, we develop a bench-
mark with both natural and synthetic images, and use it to evaluate our method to answer the following questions:

- What kinds of images does CDMD perform on best?
- What is CDMD’s trade-off between reconstructed quality and compression ratio?
- Which parameter values give best quality and/or compression for a given image type?
- How does CDMD compression compare with JPEG?

The joint answers to these questions, which we discuss in this chapter, show that CDMD is an effective tool for both color and grayscale image compression, thereby showing that medial descriptors are an interesting tool to consider, and next refine, for this task.

3.2 RELATED WORK

We start by outlining related work regarding the original dense medial descriptors (DMD) method and the related simplification parameters.

3.2.1 DMD method

The original DMD method is illustrated in Fig. 3.1. To ease presentation, we consider only grayscale images here. However, DMD can also handle color images by considering each of the three components of an YUV or RGB space in turn (see next Sec. 3.4). Let $I : \mathbb{R}^2 \rightarrow [0, 255]$ be an 8-bit grayscale image.

![Figure 3.1: Dense medial descriptor (DMD) computation pipeline.](image)

The key idea of DMD is to use 2D skeletons to efficiently encode isoluminant structures in an image. Skeletons can only be computed for
binary shapes, so \( I \) is first reduced to \( n \) (256 for 8-bit images) threshold sets (see Fig. 3.1, step 1) defined as

\[
T_i = \{ x \in \mathbb{R}^2 | I(x) \geq i \}, \quad 0 \leq i \leq n - 1.
\] (3.1)

Next, a binary skeleton \( S_{T_i} \) and its distance transform, \( \text{MAT} \left( S_{T_i}, DT_{T_i} \right) \) are extracted from each \( T_i \) (Fig. 3.1, step 2), using any skeletonization techniques listed in Sec. 2.1.2. Directly computing skeletons of binary images is notoriously unstable and complex, which causes many spurious branches introduced by small perturbations along \( \partial T_i \) (Siddiqi and Pizer, 2008; Saha et al., 2016). DMD then eliminates such spurious branches (Fig. 3.1, step 3) using the salience metric \( f \) introduced in Sec. 2.1.3. Note that such a salience metric is different from existing saliency metrics on the image, e.g., Timor and Michael (2001); Battiato et al. (2014). Skeleton points with \( f \) below a user-defined threshold \( f_0 \) are discarded, thereby disconnecting spurious skeletal branches from the skeleton rump. The final regularized \( \hat{S}_{T_i} \) is then the largest connected component in the thresholded skeleton.

The next step of DMD (Fig. 3.1, step 4) is to reconstruct a simplified version \( \hat{T}_i \) of each layer \( T_i \), applying the medial discs envelope method depicted in Sec. 2.1.4. Note that skeletonization, regularization, and reconstruction can be efficiently computed either on the CPU (Telea and van Wijk, 2002) or on the GPU (Telea, 2014). GPU methods can process images up to \( 1024^2 \) pixel resolution in a few milliseconds, allowing for high-throughput image processing applications (Ersoy et al., 2011; Koehoorn et al., 2015) and interactive applications (Zhai et al., 2020).

Finally, in step 5, a simplified version \( \hat{I} \) of the input image \( I \) is reconstructed by drawing the reconstructed layers \( \hat{T}_i \) atop each other, in increasing order of luminance \( i \). Since DMD doesn’t keep all threshold sets (see next in Sec. 3.2.2), an intensity-banding effect can occur. To solve this, DMD applies a smooth distance-based interpolation between two consecutive selected layers \( T_i \) and \( T_{i+1} \). In detail, for a pixel \( x \) located between the boundaries \( \partial T_i \) and \( \partial T_{i+1} \), DMD interpolates its value \( v(x) \) (in all three channels independently) from the corresponding intensity values \( v_i \) and \( v_{i+1} \) of \( T_i \) and \( T_{i+1} \) respectively as

\[
v(x) = \frac{1}{2} \left[ \min \left( DT_{T_i}(x), DT_{T_{i+1}}(x) \right), 1 \right] v_i + \max \left( 1 - \frac{DT_{T_{i+1}}(x)}{DT_{T_i}(x)}, 0 \right) v_{i+1} \right].
\] (3.2)

For further details, including implementation of DMD, we refer to Van Der Zwan et al. (2013).

### 3.2.2 Image simplification parameters

DMD parameterizes the threshold-set extraction and skeletonization steps (Sec. 3.2.1) to achieve several image simplification effects, such as segmentation, small-scale detail removal, and artistic image manipulation (Van Der Zwan et al., 2013; Koehoorn et al., 2015; Sobiecki
et al., 2015). We further discuss the roles of these parameters, as they crucially affect DMD’s suitability for image compression, which we analyze next in Sections 3.3–3.5.

**Island removal:** During threshold-set extraction, islands (connected components in the image foreground $T_i$ or background $\overline{T}_i$) smaller than a fraction $\epsilon$ of $|T_i|$, respectively $|\overline{T}_i|$, are filled in, respectively removed. Higher $\epsilon$ values yield layers $T_i$ having fewer small-scale holes and/or disconnected components. This creates simpler skeletons $S_{T_i}$ which lead to better image compression. However, too high $\epsilon$ values will lead to oversimplified images.

**Layer selection:** As noted in Van Der Zwan et al. (2013), one does not need all layers $T_i$ to obtain a perceptually good reconstruction $\tilde{I}$ of the input $I$. Selecting a small layer subset of $L < n$ layers from the $n$ available ones leads to less information needed to represent $\tilde{I}$, so better compression. Yet, too few layers and/or suboptimal selection of these degrades the quality of $\tilde{I}$. We study how many (and which) layers are needed for a good reconstruction quality in Sec. 3.3.1.

**Skeleton regularization:** The intuition behind saliency regularization (Eqn. (2.7)) follows a similar argument as for layer selection: One can obtain a perceptually good reconstruction $\tilde{I}$, using less information, by only keeping skeletal branches above a certain saliency $\sigma_0$. Yet, how the choice of $\sigma_0$ affects reconstruction quality has not been investigated, neither in the original paper proposing saliency regularization (Telea, 2012) nor by DMD. We study this relationship in Sec. 3.4.

## 3.3 Proposed Compression Method

Our proposed Compressing Dense Medial Skeletons (CDMD) adapt the original DMD pipeline (Fig. 3.1) to make it effective for image compression in two directions: layer selection (Sec. 3.3.1) and encoding the resulting MAT (Sec. 3.3.2), as follows.

### 3.3.1 Layer selection

DMD selects a subset of $L < n$ layers $T_i$ from the total set of $n$ layers based on a simple greedy heuristic: Let $\tilde{I}_i$ be the reconstruction of image $I$ using all layers, except $T_i$. The layer $T_i$ yielding the smallest reconstruction error $\min_{1 \leq i \leq n} SSIM(I, \tilde{I}_i)$ is deemed the least relevant and thus first removed. The procedure is repeated over the remaining layers, until only $L$ layers are left. This approach has two key downsides: Removing the least-relevant layer (for reconstruction) at a time does not guarantee that subsequent removals do not lead to poor quality. For an optimal result, one would have to maximize quality over all combinations of $L$ (kept) layers selected from $n$, which is prohibitively expensive.
Secondly, this procedure is very expensive, as it requires $O((n - L)^2)$ reconstructions and image comparisons to be computed.

We improve layer selection by testing three new strategies, as follows.

**Histogram thresholding:** We compute a histogram of how many pixels each layer $T_i$ individually encodes, i.e., $|T_i \setminus T_{i+1}|$. Next, we select all layers having values above a given threshold. To make this process easy, we do a layer-to-threshold conversion: given a number of layers $L$ to keep, we find the corresponding threshold based on binary search.

**Histogram local maxima:** Histogram thresholding can discard layers containing small but visually important features such as highlights. Furthermore, all layers below the threshold are kept, which does not lead to optimal compression. We refine this by finding histogram local maxima (shown in Fig. 3.2 (b) for the test image in Fig. 3.2 (a)). The intuition here is that the human eye cannot distinguish subtle differences between adjacent (similar-luminance) layers (Hecht, 2003), so, from all such layers, we can keep only the one contributing the most pixels to the reconstruction. As Fig. 3.2 (c) shows, 15 layers are enough for a good-quality reconstruction, also indicated by a high MS-SSIM score.

**Cumulative histogram:** We further improve layer selection by using a cumulative layer histogram (see Fig. 3.2 (d) for the image in Fig. 3.2 (a)). We scan this histogram left to right, comparing each layer $T_i$ with layer $T_{j=i+m}$, where $m$ is the minimally-perceivable luminance difference to a human eye (set empirically to 5 (Hecht, 2003) on a luminance range of $[0, 255]$). If the histogram difference between layers $T_i$ and $T_j$ is smaller than a given threshold $\lambda$, we increase $j$ until the difference is above $\lambda$. At that point, we select layer $T_j$ and repeat the process until we reach the last layer. However, setting a suitable $\lambda$ is not easy for inexperienced users. Therefore, we do a layer-to-threshold conversion by a binary search method, as follows. Let $[r_{\min}, r_{\max}]$ be the range of the cumulative histogram. At the beginning of the search, this range equals $[0, 1]$. We next set $\hat{\lambda} = (r_{\min} + r_{\max})/2$ and compare the number of layers $L'$ produced under this condition with the target, i.e. desired, user-given value $L$. If $L' = L$, then the search ends with the current value of $\hat{\lambda}$. If $L' < L$, we continue the search in the lower half $[r_{\min}, (r_{\min} + r_{\max})/2]$ of the current range. If $L' > L$, we continue the search in the upper half $[(r_{\min} + r_{\max})/2, r_{\max}]$ of the current range. Since $L$ is an integer value, the search may sometimes oscillate, yielding values $L'$ that swing around, but do not precisely equal, the target $L$. To make the search end in such situations, we monitor the computed $L'$ over subsequent iterations and, if oscillation, i.e., a non-monotonic evolution of the $L'$ values over subsequent iterations, is detected, we stop the search and return the current $\hat{\lambda}$. Through this conversion, what users need to set is only the desired number of layers, which makes it simple to use by any target group – much like setting the ‘quality’ parameter in typical JPEG compression. Compared to local maxima selection, the cumulative his-
Figure 3.2: Layer selection methods. (a) Original image. (b) Histogram of (a), with local maxima marked in red. (c) Reconstruction of (a) using 15 most relevant layers given by (b). (d) Cumulative histogram of (a), with selected layers marked red. (e) Reconstruction of (a) using the 15 most relevant layers given by (d).

togram method selects smoother transition layers, which yields a better visual effect. For example, in Fig. 3.2 (c), the local details around the shoulder show clear banding effects; the same region is much smoother when cumulative histogram selection is used (Fig. 3.2 (e)). Besides improved quality, cumulative histogram selection is simpler to implement and use, as it does not require complex and/or sensitive heuristics for detecting local maxima. Figure 3.3 compares the four layer selection methods discussed above. We test these on a 100-image database with 10 different image types, each having 10 images (see Table 1). The 10 types aim to capture general-purpose imagery (people, houses, scenery, animals, paintings) which are typically rich in details and textures; images having a clear structure, i.e., few textures, sharp contrasts, well-delineated shapes (ArtDeco, cartoon, text); and synthetic images being somewhere between the previous two types (scientific visualization).

Average MS-SSIM scores show that the cumulative histogram selection yields the best results for all image types, closely followed by local maxima selection and next by the original greedy method in DMD. The naive histogram thresholding yields the poorest MS-SSIM scores, which also strongly depend on image type. Besides better quality, the cumulative histogram method is also dramatically faster, 3000 times more than the greedy selection method in Van Der Zwan et al. (2013). Hence, cumulative histogram is our method of choice for layer selection for CDMD.
3.3 Proposed Compression Method

![Figure 3.3: Average MS-SSIM scores for four layer selection methods (30 layers selected) for images in ten different classes. The cumulative histogram method performs the best and is hence used in CDMD.](image)

### Table 1: The benchmark of 100 images (available at Wang et al. (2020b)) used throughout this work for testing CDMD.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>animal</td>
<td>Wild animals in their natural habitat</td>
</tr>
<tr>
<td>artDeco</td>
<td>Art deco artistic images</td>
</tr>
<tr>
<td>cartoon</td>
<td>Cartoons and comic strips</td>
</tr>
<tr>
<td>house</td>
<td>Residential homes surrounded by greenery</td>
</tr>
<tr>
<td>nature</td>
<td>Panorama landscapes and close-ins of plants</td>
</tr>
<tr>
<td>other</td>
<td>Miscellaneous (fruit, planets, natural scenery)</td>
</tr>
<tr>
<td>painting</td>
<td>Classical and modern paintings</td>
</tr>
<tr>
<td>people</td>
<td>Portrait photos of various people</td>
</tr>
<tr>
<td>SVdata</td>
<td>Scientific visualizations (scalar and vector fields)</td>
</tr>
<tr>
<td>text</td>
<td>Typography of various styles and scales</td>
</tr>
</tbody>
</table>

#### 3.3.2 MAT encoding

MAT computation (Sec. 3.2.1) delivers, for each selected layer $T_i$, pairs of skeletal pixels $x$ with corresponding inscribed circle radii $r = DT_T(x)$. Naively storing this data requires two 16-bit integer values for the two
components of $x$ and one 32-bit floating-point value for $r$, respectively. We propose next two strategies to compress this data losslessly.

**Intra-layer compression:** As two neighbor pixels in a skeleton are 8-connected, their differences in $x$ and $y$ coordinates are limited to $\Delta x, \Delta y \in \{-1, 0, 1\}$, and similarly $\Delta r \in \{-2, -1, 0, 1, 2\}$. Hence, we visit all pixels in a depth-first manner (Cormen et al., 2001) and encode, for each pixel, only the $\Delta x, \Delta y,$ and $\Delta r$ values. This leads to an entropy of $2 \log_2(3) + \log_2(5) \approx 5.89$ bits per skeleton pixel. In practice, however, we use one byte per MAT point in which $\Delta x$ and $\Delta y$ take up two bits each, and $\Delta r$ three bits, i.e., 0xxyyrrr. We further compress this direct delta-representation of each MAT point by testing nine other lossless encoding methods: Huffman (Geelnard, 2007), Canonical Huffman, Unitary (Roy and Scott, 2009), Exponential Golomb, Arithmetic (Langdon, 1984), Predictive, Compact, Raw, and Move-to-Front (MTF) (Bentley et al., 1986). To compare the effectiveness of these methods, we use the compression ratio of an image $I$ defined as

$$CR(I) = \frac{|I|}{|\text{MAT}(I)|},$$

where $|I|$ is the byte-size of the original image $I$ and $|\text{MAT}(\tilde{I})|$ is the byte-size of the MAT encoding for all selected layers of $\tilde{I}$. Table 2 (top row) compares the 10 tested encoding methods, showing average $CR(I)$ value for the 10 image types in Fig. 3.3, and 12 different combinations of parameters $\epsilon, L,$ and $\sigma_0$ per compression-run. The highest value in each row is marked in bold.

**Inter-layer compression:** The inter-layer compression leaves, likely, still significant redundancy in the MATs of different layers. To remove this, we compress the MAT of all layers (each encoded using all 10 lossless methods discussed above) with eight lossless-compression algorithms: Lempel–Ziv–Markov Chain (LZMA) (Pavlov, 

Table 2: Comparison of average compression ratios (Eqn. (3.3)) for 10 lossless MAT-encoding methods on 100 images using only intra-layer compression (top row) and inter-layer compression (bottom row).

<table>
<thead>
<tr>
<th>Method</th>
<th>Direct Intra-layer</th>
<th>Huffman</th>
<th>Canonical</th>
<th>Unitary</th>
<th>Exponential Golomb</th>
<th>Arithmetic</th>
<th>Predictive</th>
<th>Compact</th>
<th>Raw</th>
<th>MTF</th>
<th>40-Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intra-layer</td>
<td>1.67</td>
<td>2.46</td>
<td>2.46</td>
<td>2.07</td>
<td>1.8</td>
<td>2.67</td>
<td>1.87</td>
<td>2.12</td>
<td>2.42</td>
<td>1.87</td>
<td>1.67</td>
</tr>
<tr>
<td>Inter-layer</td>
<td>4.08</td>
<td>2.73</td>
<td>2.75</td>
<td>2.91</td>
<td>2.9</td>
<td>1.69</td>
<td>2.87</td>
<td>3.16</td>
<td>2.82</td>
<td>2.46</td>
<td>4.36</td>
</tr>
</tbody>
</table>
3.3 proposed compression method

2007), LZHAM (Geldreich, 2011), Brotli (Alakuijala and Szabadka, 2016), ZPAQ (Mahoney, 2015), BZip2 (Seward, 1998), LZMA2 (Pavlov, 2007), BSC (Grebnov, 2011), and ZLib (Deutsch and Gailly, 1996), all available in the Squash library (Nemerson, 2015). Figure 3.4 shows CR boxplots (Eqn. (3.3)) for all our 100 test images. Blue boxes show the 25–75% quantile; red lines are medians; black whiskers show extreme data points not considered outliers; outliers are shown by red ‘+’ marks. Overall, ZPAQ is the best compression method, 20.15% better than LZMA, which was used in the original DMD method (Van Der Zwan et al., 2013). Hence, we select ZPAQ for CDMD.

![Figure 3.4: Compression ratio boxplots for eight compression methods run on 100 images.](image)

Table 2 (second row) shows the average CR values after applying inter-layer compression. Interestingly, direct encoding turns to be better than the nine other considered lossless encoding methods. This is because the pattern matching of the inter-layer compressor is rendered ineffective when the signal encoding already approaches the entropy of the original direct encoding (5.89 bits). Given this finding, we further improve direct encoding by considering all combinations among possible values of $\Delta x$, $\Delta y$ and $\Delta r$. Among the $3 \times 3 \times 5 = 45$ combinations, only 40 are possible as the five cases with $\Delta x = \Delta y = 0$ cannot exist in practice. This leads to an information content of $\log_2(40) \approx 5.32$ bits per skeleton pixel. Table 2 (rightmost column) shows the average CR values with the 40-case encoding, which is 6.86% better than the best in the tested methods after all-layer compression. Hence, we keep this encoding method for CDMD.
3.4 Evaluation and Optimization

Our CDMD method described in Sec. 3.3 introduced three improvements with respect to DMD: the cumulative histogram layer selection, the intra-layer compression, and the inter-layer compression. On our 100-image benchmark, these jointly deliver the following improvements:

- Layer selection: 3000 times faster and 3.28% higher quality;
- MAT encoding: 20.15% better compression ratio.

CDMD depends, however, on three parameters: the number of selected layers \( L \), the size of removed islands \( \varepsilon \), and the saliency threshold \( \sigma_0 \). Moreover, a compressed image \( \tilde{I} \) is characterized by two factors: the visual quality that captures how well \( \tilde{I} \) depicts the original image \( I \), e.g., measured by the MS-SSIM metric, and the compression ratio \( CR \) (Eqn. 3.3). Hence, the overall quality of CDMD can be modeled as

\[
(\text{MS-SSIM}, CR) = CDMD(L, \varepsilon, \sigma_0). \tag{3.4}
\]

Optimizing this two-variate function of three variables is not easy. Several commercial solutions exist, e.g., TinyJPG (Tinify, 2019) but their algorithms are neither public nor transparent. To address this, we first merge the two dependent variables, \( \text{MS-SSIM} \) and \( CR \), into a single one (Sec. 3.4.1). Next, we describe how we optimize for this single variable over all three free parameters (Sec. 3.4.2).

3.4.1 Joint compression quality

We need to optimize for both image quality \( \text{MS-SSIM} \) and compression ratio \( CR \) (Eqn. (3.4)). These two variables are, in general, inversely correlated: strong compression (high \( CR \)) means poor image quality (low \( \text{MS-SSIM} \)), and vice versa. To handle this, we combine \( \text{MS-SSIM} \) and \( CR \) into a single joint quality metric

\[
Q' = \frac{f_{\text{MS-SSIM}}(\text{MS-SSIM}) + f_{CR}(\overline{CR})}{2}, \tag{3.5}
\]

where \( \overline{CR} \) is the \( CR \) of a given image \( I \) normalized (divided) by the maximal \( CR \) value over all images in our benchmark. After extensive experimentation with images from our benchmark, we found that \( \text{MS-SSIM} \) perceptually weighs more than \( CR \), which requires the former to have a higher power weighting factor than the latter. In practice, the transfer functions \( f_{\text{MS-SSIM}}(x) = x^2 \) and \( f_{CR}(x) = x \) are used to combine (weigh) the two criteria we want to optimize for, namely quality \( \text{MS-SSIM} \) and compression ratio \( CR \). Note that, if desired, \( f_{\text{MS-SSIM}} \) and \( f_{CR} \) can be set to the identity function, which would imply a joint quality \( Q' \) defined as the mean of the two.
3.4.2 Optimizing the joint compression quality

To find parameter values that maximize $Q'$ (Eqn. 3.5), we fix, in turn, two of the three free parameters $L$, $\epsilon$, and $\sigma_0$ to empirically-determined average values, and vary the third parameter over its allowable range via uniform sampling. The maximum $Q'$ value found this way determines the value of the varied parameter. This is simpler, and faster, than the usual hyper-parameter grid-search used, e.g., in machine learning (Bergstra and Bengio, 2012), and is motivated by the fact that our parameter space is quite large (three-dimensional) and thus costly to search exhaustively by dense grid sampling. This process leads to the following results.

Number of layers: To study how $L$ affects the joint quality $Q'$, we plot $Q'$ as a function of $L$ for our benchmark images. We sample $L$ from 10 to 90 with a step of 10, following observations in Van Der Zwan et al. (2013) stating that 50–60 layers typically achieve good SSIM quality. The two other free variables are set to $\epsilon = 0.02$ and $\sigma_0 = 1$. Figure 3.5 (a) shows the results. We see that CDMD works particularly well for images of art deco and scientific visualization types. We also see that $Q'$ hardly changes for $L > 40$. Figure 3.5 (b) summarizes these insights, showing that values $L \in \{20, 30, 40\}$ give an overall high $Q'$ for all image types.

Figure 3.5: Quality $Q'$ as a function of number of layers $L$. (a) $Q'$ plots per image type. (b) Average $Q'$ for all image types. Black dots indicate good $L$ values (20, 30, and 40).

Island size and saliency: We repeat the same evaluation for the other two free parameters, i.e., minimal island size $\epsilon$ and skeleton saliency $\sigma_0$, fixing each time the other two parameters to average values. Figure 3.6 shows how $Q'$ varies when changing $\epsilon$ and $\sigma_0$ over their respective ranges of $\epsilon \in [0, 0.04]$ and $\sigma_0 \in [0, 6]$, similar to Fig. 3.5. These ranges are determined by considerations outlined earlier in related work (Telea, 2012; Van Der Zwan et al., 2013; Koehoorn et al., 2015; Sobiecki et al., 2015). Optimal values for $\epsilon$ and $\sigma_0$ are indicated in Fig. 3.6 by black dots.
3.4.3 Trade-off between MS-SSIM and CR

As already mentioned, our method, and actually any lossy image compression method, has a trade-off between compression (which we measure by CR) and quality (which we measure by MS-SSIM). Figure 3.7 shows the negative, almost-linear, correlation between CR and MS-SSIM for the 10 house images in our benchmark, with each image represented by a different color. Same-color dots show $3 \times 4 \times 4 = 48$ different settings of $L$, $\varepsilon$, and $\sigma_0$ parameters, computed as explained in Sec. 3.4.2. This negative correlation is present for both the color version of the test image (Fig. 3.7 (b)) and its grayscale variant (Fig. 3.7 (a)). However, if we compare a set of same-color dots in Fig. 3.7 (a), i.e., compressions of a given grayscale image for the 48 parameter combinations, with the similar set in Fig. 3.7 (b), i.e., compressions of the same image, color variant for the same parameter combinations, we see that the first set is roughly lower and more to the left than the second set. That is, CDMD handles color images compressed better than grayscale ones, i.e., yields higher CR and/or higher MS-SSIM values. Very similar patterns occur for all other nine image types in our benchmark. For full results, we refer to Wang et al. (2020b).
Besides parameter values, the trade-off between MS-SSIM and CR depends on the image type. Figure 3.8 shows this by plotting the average MS-SSIM vs CR for all 10 image types in our benchmark. Here, one dot represents the average values of the two metrics for a given parameter-setting over all images in the respective class. We see the same inverse correlation as in Fig. 3.7. We also see that CDMD works best for art decoration (artDeco) and scientific visualization (SVdata) image types.

Figure 3.8: Average MS-SSIM vs. CR for 10 image types for CDMD (filled dots) and JPEG (hollow dots). Left shows results for the grayscale variants of the color images (shown right).

3.4.4 Comparison with JPEG

Figure 3.8 also compares the MS-SSIM and CR values of CDMD (full dots) with JPEG (hollow dots) for all our benchmark images, for their grayscale versions (a) and color versions (b), respectively. Overall, JPEG yields higher MS-SSIM values, but CDMD yields better CR values for most of its parameter settings. We also see that CDMD performs relatively better for the color images. Figure 3.9 further explores this insight by showing ten images, one of each type, from our benchmark, compressed by CDMD and JPEG, and their corresponding CR and MS-SSIM values. Results for the entire 100-image database are available in the supplementary material. We see that, if one prefers a higher CR for a given image quality, CDMD is a better choice than JPEG. Furthermore, there are two image types for which we get both a higher CR than JPEG and a similar quality: Art Deco and Scientific Visualization. Figure 3.10 explores these classes in further detail, by showing four additional examples, compressed with CDMD and JPEG. We see that CDMD and JPEG yield results which are visually almost identical (and have basically identical MS-SSIM values). However, CDMD yields compression values 1.8 up to 3.5 times higher than JPEG. Figure 3.10 (a3–d3) shows the per-pixel difference maps between the compressed images with CDMD and JPEG (differences coded in luminance). These difference images are almost everywhere black, indicating no differences between the two com-
Figure 3.9: Comparison of JPEG (a1–j1) with our method (a2–j2) for 10 image types. For each image, we show the MS-SSIM quality and compression ratio CR.

...pressions. Minimal differences can be seen, upon careful examination of these difference images, along a few luminance contours, as indicated by the few bright pixels in the images. These small differences are due to the salience-based skeleton simplification in CDMD.

For a more detailed comparison with JPEG, we next consider JPEG’s quality setting $q$. This value, set typically between 10% and 100%, controls JPEG’s trade-off between quality and compression, with higher values favoring quality. Figure 3.11 compares CDMD for the Scientific Visualization and ArtDeco image types (filled dots) with 10 different set-
Figure 3.10: Our method (a2–d2) yields higher compression than, and visually identical quality with, JPEG (a1–d1) for two image classes: Scientific Visualization (a,b) and Art Deco (c,d)).

Settings of JPEG’s \( q \) parameter, uniformly spread in the \([10, 100]\) interval (hollow dots). Each dot represents the average of MS-SSIM and CR for a given method and image type for a given parameter combination. Figure 3.11 shows an almost-linear negative correlation between MS-SSIM and CR for both CDMD and JPEG. Yet, the results of CDMD are almost all in the top right of JPEG, i.e., CDMD yields a higher compression for a given MS-SSIM score (smaller than 0.97) and almost always gets a better quality score for a given compression ratio.

3.4.5 Handling noisy images

As explained in Sec. 3.2.2, the island removal parameter \( \varepsilon \) and the saliency threshold \( \sigma_0 \) jointly ‘simplify’ the compressed image by re-
moving, respectively, small-scale islands and small-scale indentations along the threshold-set boundaries. Hence, it is insightful to study how these parameters affect the compression of images which have high-frequency, small-scale details and/or noise. Figure 3.12 shows an experiment that illustrates this. An original image was selected which contains high amounts of small-scale high-frequency detail, e.g., the mandrill’s whiskers and fur patterns.

The left column shows the CDMD results for four combinations of $\varepsilon$ and $\sigma_0$. In all cases, we used $L = 30$. As visible, and in line with expectations, increasing $\varepsilon$ and/or $\sigma_0$ has the effect of smoothing out small-scale details, thereby decreasing MS-SSIM and increasing the compression ratio $CR$. However, note that contours that separate large image elements, such as the red nose from the blue cheeks, or the pupils from the eyes, are kept sharp. Furthermore, thin-but-long details such as the whiskers have a high saliency, and are thus kept quite well.

The middle column in Fig. 3.12 shows the CDMD results for the same image, this time corrupted by salt-and-pepper noise of density 0.1, compressed with the same parameter settings. We see that the noise is removed very well for all parameter values, the compression results being visually nearly identical to those generated from the uncorrupted image. The MS-SSIM and $CR$ values are now slightly lower, since, although visually difficult to spot, the added noise does affect the threshold sets in the image. Finally, the right column in Fig. 3.12 shows the CDMD results for the same image, this time corrupted by zero-mean Gaussian white noise with variance 0.01. Unlike salt-and-pepper noise, which is distributed randomly over different locations and has similar
Figure 3.12: Results of CDMD on an image with fine-grained detail (left column) additionally corrupted by small-scale noise (middle and right columns), for different values of the $\varepsilon$ and $\sigma_0$ parameters.
amplitudes, the Gaussian noise has a normal amplitude distribution and affects all locations in an image uniformly. Hence, CDMD does not remove Gaussian noise as well as the salt-and-pepper one, as we can see both from the actual images and the corresponding MS-SSIM and CR values. Yet, even for this noise type, we argue that CDMD does not produce disturbing artifacts in the compressed images, and still succeeds in preserving the main image structures and also a significant amount of the small-scale details.

3.5 Discussion

We next discuss several aspects of our CDMD image compression method.

Genericity, ease of use: CDMD is a general-purpose compression method for any types of grayscale and color images. It relies on simple operations such as histogram computation and thresholding, as well as on well-tested, robust, algorithms, such as the skeletonization method in Telea and van Wijk (2002); Falcão et al. (2004), and ZPAQ. CDMD has three user parameters – the number of selected layers $L$, island thresholding $\epsilon$, and skeleton saliency threshold $\sigma_0$. These three parameters affect the trade-off between compression ratio and image quality (see Sec. 3.4.2). End users can easily understand these parameters as follows: $L$ controls how smooth the gradients (colors or shades) are captured in the compressed image (higher values yield smoother gradients); $\epsilon$ controls the scale of details that are kept in the image (higher values remove larger details); and $\sigma_0$ controls the scale of corners that are kept in the image (larger values round-off larger corners). Good default ranges of these parameters are given in Sec. 3.4.2.

Speed: The most complex operation of the CDMD pipeline, the computation of the regularized skeletons $\hat{S}$, is efficiently done on the GPU (see Sec. 3.2.1). Formally, CDMD’s computational complexity is $O(n)$ for an image of $n$ pixels, since the underlying skeletonization is linear in image size, being based on a linear-time distance transform (Cao et al., 2010). This is the best that one can achieve complexity-wise. Given this, the CDMD method is quite fast: For images of up to $1024^2$ pixels, on a Linux PC with an Nvidia RTX 2060 GPU, layer selection takes under 1 millisecond; skeletonization takes about 1 second per color channel; and reconstruction takes a few hundred milliseconds. Obviously, state-of-the-art image compression methods have highly engineered implementations which are faster. We argue that the linear complexity of CDMD also allows speed-ups to be gained by subsequent engineering and optimization.

Image size: We are unaware of studies showing the relationship between quality and compression performance vs. image size for, e.g., JPEG. Still, analyzing JPEG, we see that its size complexity linearly de-
3.5 Discussion

Depends on the image size. That is, the compression ratio CR is overall linear in the input image size n for a given, fixed, quality, since JPEG encodes an image by separate 8 × 8 blocks. In contrast, CDMD’s skeletons are of √n complexity, since they are 1D structures. While a formal evaluation pending, this suggests CDMD may scale better for large image sizes.

Color spaces: As explained in Sec. 3.2.1, for color images, (C)DMD is applied to the individual channels of these, following representations in various color spaces. We currently tested the RGB and HSV color spaces, following the original DMD method proposal. For these, we obtained very similar compression vs. quality results. We also tested YUV (more precisely, YCbCr), and obtained compression ratios about twice as high as those reported earlier in this paper (for the RGB space). However, layer selection in the YCbCr space is more delicate than in RGB space: While the U and V channels can be described well with just a few layers (which is good for compression), a slightly too aggressive compression (setting a slightly too low L value) can yield strong visual differences between the original and compressed images. Hence, the method becomes more difficult to control, parameter-wise, by the user. Exploring how to make this control simpler for the end user, while retaining the higher compression rate of the YUV space, is an interesting point for future work.

Best image types: Layer removal is a key factor to CDMD. Images that have large and salient threshold-sets, such as Art Deco and Scientific Visualization, can be summarized by just a few such layers (low L). For instance, the Art Deco image in Fig. 3.10 (c1) has only a few distinct gray levels, and large, salient, shapes in each layer. Its CDMD compression (Fig. 3.10 (c2)) is of high quality, and is more than 60 times smaller than the original. The JPEG compression of the same image is just 17 times smaller than the original. At the other extreme, we see that CDMD is somewhat less suitable for images with many fine details, such as animal furs and greenery (Fig. 3.9 (e2)). This suggests that CDMD could be very well suited (and superior to JPEG) for compressing data-visualization imagery, e.g., in the context of remote/online viewing of medical image databases.

Preprocessing for JPEG: Given the above observation, CDMD and JPEG seem to work best for different types of images. Hence, a valid idea is to combine the two methods rather than let them compete against each other, following earlier work that preprocesses images to aid JPEG’s compression (Tushabe and Wilkinson, 2007). We consider the same idea, i.e., use CDMD as a preprocessor for JPEG. Figure 3.13 shows three examples of this combination. When using only JPEG, the original images (a1–c1), at 20% quality (JPEG setting q), yield blocking artifacts (a2–c2). When using JPEG with CDMD preprocessing, these artifacts are decreased (a3–c3). This can be explained by the rounding-off
of small-scale noise dents and bumps that the saliency-based skeleton simplification performs (Telea, 2012). Such details correspond to high frequencies in the image spectrum which next adversely impact JPEG. Preprocessing by CDMD has the effect of an adaptive low-pass filter that keeps sharp and large-scale details in the image while removing sharp and small-scale ones. As Fig. 3.13 shows, using CDMD as preprocessor for JPEG yields a 10% to 20% compression ratio increase as compared to plain JPEG, with a limited loss of visible quality.

**Limitations:** Besides the limited evaluation (on only 100 color images and their grayscale equivalents), CDMD is here only evaluated against a single generic image compression method, i.e., JPEG. As outlined in Sec. 2.2, tens of other image compression methods exist. We did not perform an evaluation against these since, as already noted, our main research question was to show that skeletons can be used for image compression with good results—something that has not been done so far. We confirmed this by comparing CDMD against JPEG. Given our current positive results, we next aim to improve CDMD, at which point com-
3.6 conclusion

In this chapter, we have presented Compressing Dense Medial Descriptors (CDMD), an end-to-end method for compressing color and grayscale images using a dense medial descriptor approach. CDMD adapts the existing DMD method, proposed for image segmentation and simplification, for the task of image compression. For this, we proposed an improved layer-selection algorithm, a lossless MAT-encoding scheme, and an all-layer lossless compression scheme. To study the effectiveness of our method, we considered a benchmark of 100 images of 10 different types, and did an exhaustive search of the free-parameters of our method, in order to measure and optimize the compression-ratio, perceptual quality, and combination of these two metrics. On a practical side, our evaluation showed that CDMD delivers superior compression to JPEG at a small quality loss; that it delivers both superior compression and quality for specific image types. On a more theoretical (algorithmic) side, CDMD shows, for the first time, that medial descriptors offer interesting and viable possibilities to compress grayscale and color images, thereby extending their applicability beyond the processing of binary shapes.

In the next chapter, we explore how dense medial descriptors can be combined with so-called spatial saliency maps, which encode the relative importance of different parts of an image, to selectively simplify and compress an image while preserving its salient regions.
In the previous chapter, we have demonstrated that dense medial descriptors offer a promising way for simplifying and compressing images. However, to date, these have been applied in a global manner that is oblivious to salient features. In this chapter, we adapt medial descriptors to use the information provided by saliency maps to selectively simplify and encode an image while preserving its salient regions. This allows us to improve the trade-off between compression ratio and image quality as compared to the standard dense-skeleton method while keeping perceptually salient features, in a focus-and-context manner. We show how our method can be combined with JPEG to increase overall compression rates at the cost of a slightly lower image quality. We demonstrate our method on a benchmark composed of a broad set of images.

4.1 Introduction

In Chapter 3, we have presented Compressing Dense Medial Descriptor (CDMD), which adapts the original DMD method to make it effective for image compression and simplification. CDMD models an image as \( L \) luminance threshold-sets \( T_i \), or layers, each layer being encoded by its medial axis transform \( (S_{T_i}, DT_{T_i}) \). \( S_{T_i} \) contains spurious branches caused by small perturbations along the boundary of \( T_i \), which can be eliminated using the salience regularization metric (Sec. 2.1.3). A simplified version of the image \( f \) is finally reconstructed from the regularized skeleton \( \tilde{S}_{T_i} \) and its distance transforms, i.e., \( (\tilde{S}_{T_i}, \tilde{D}T_{T_i}) \) of \( L \) selected layers \( T_i \). Qualitative and quantitative evaluation has shown that CDMD delivers good compression ratios while preserving image quality. However, CDMD can only simplify an image globally. High simplification will easily remove small, but visually important, details (poor quality). Conversely, low simplification will allocate storage to unimportant image areas (poor compression).

In this chapter, we extend the CDMD method with the so-called spatial saliency map that models the importance of various areas in an image. Formally, a saliency map \( \mu(x) : \mathbb{R}^2 \rightarrow [0, 1] \) gives, for each image pixel \( x \), its importance or saliency, between totally irrelevant (\( \mu = 0 \)) and maximal importance (\( \mu = 1 \)). Such maps have been used for image quality assessment (Liu and Heynderickx, 2011), content-based image retrieval (Chen et al., 2009), context-aware image resizing (Goferman

This chapter is based on the paper "Focus-and-Context Skeleton-Based Image Simplification Using Saliency Maps" (Wang et al., 2021a)
et al., 2011), and saliency-based image compression (Zünd et al., 2013; Andrushia and Thangarajan, 2018). Saliency maps can be created either in supervised mode—by users via manual annotation—or in unsupervised mode, automatically computed from images.

**Supervised** methods use ground-truth images to learn discriminant features of salient objects (de Melo Joao et al., 2020). The most accurate supervised methods use deep-learning (Borji et al., 2015; Wang et al., 2021g) and typically outperform unsupervised methods. Yet, they need large amounts of human-annotated training data, and the generalization of training models across image domains usually requires adaptation and retraining (de Melo Joao et al., 2020).

**Unsupervised** methods use prior knowledge about salient objects and local image characteristics. Most methods start by finding image regions (e.g. superpixels) with high color contrast relative to neighbors (Jiang et al., 2013; Li et al., 2013; Zhang et al., 2018). Besides contrast, objects in focus (Jiang et al., 2013), near the image center (Cheng et al., 2014), or having red and yellow tones, important for human vision (Peng et al., 2017), are all considered as salient factors. Conversely, regions similar to the boundary will have low saliency as most image boundaries are background in natural images (Cheng et al., 2014; Zhang et al., 2018; Li et al., 2013; Jiang et al., 2013). In this chapter, we use the DSR (Li et al., 2013) unsupervised bottom-up saliency estimation method which provides reliable saliency maps without requiring parameter tuning and is fast. Any other saliency estimators can be directly used instead as long as users find the produced maps suitable for their tasks at hand.

Next, we illustrate how to introduce such saliency maps into the dense skeleton-based image compression pipeline (in Fig. 4.1) to implement finer-grained spatial control of the simplification. We propose several metrics to gauge the effectiveness of our method and the trade-off between image size and perceptual similarity. We evaluate these metrics on a collection of real-world images to illustrate the advantages of our extended method.

### 4.2 Proposed Method

As stated in Sec. 4.1, an important limitation of CDMD is that it simplifies an image globally. Therefore, we improve CDMD by considering spatially-dependent simplification of image foreground and background. We call our method Spatial Saliency DMD (SSDMD for short). Fig. 4.1 (red) shows the steps that SSDMD adds to CDMD. These steps are described next.
4.2 PROPOSED METHOD

4.2.1 Salient islands detection

As explained in Sec. 3.2.1, CDMD removes islands smaller than a global value of ε area units. This removes not only noise but also small important features, e.g. the animal eyes in Fig. 4.3 (a1)–(c1). To address this, we compute a saliency-aware metric $C_i^\mu = \sum_{x \in C_i} \mu(x)$, where $C_i$ is the $i$th connected component, and next remove only islands where $C_i^\mu$ is below a newly added user-given threshold $\varepsilon_0$, as shown in step 2 in Fig. 4.1. This keeps small-size, but salient, details, in the compressed image.

4.2.2 Saliency-based skeletons

We further simplify the regularized skeletons $\tilde{S}_i$ by removing pixels whose saliency $\mu$ is below a user-given threshold $\mu_0$, resulting in saliency-aware skeletons $\tilde{S}_i^\mu = \{x \in S_i | \mu(x) > \mu_0\}$. The threshold $\mu_0$ controls the amount of the non-salient areas. To avoid low-saliency areas (with saliency $\mu$ that are below the global threshold $\mu_0$) being completely removed, resulting in poor image quality, we reserve one layer every $<\text{layers}$ for these areas. The skeletons $\tilde{S}_i$ to be reconstructed are then computed using the piecewise formulation

$$\tilde{S}_i = \begin{cases} \tilde{S}_i, & \text{if } i \text{ mod } m = 0, \\ \tilde{S}_i^\mu, & \text{otherwise.} \end{cases}$$

The parameter $m$ controls how smooth color or brightness gradients will be in the non-salient areas; smaller $m$ values yield smoother gradients. Since only several layers are reserved in non-salient areas, an intensity-banding effect can occur. To solve this, we apply the smooth distance-based interpolation operation described in Eqn. 3.2 between two consecutive selected layers.

Figure 4.1: Compressing Dense Medial Descriptor (CDMD) pipeline with free parameters in green. Red: Elements added by our SSDMD method.
4.2.3 Saliency-aware quality metric

To measure the visual accuracy of the proposed technique, a reliable full reference quality metric is required. While MS-SSIM models human perception well (Sec. 2.3), it treats focus (high $\mu(x)$) and context (low $\mu(x)$) areas identically. Figure 4.2 shows this: Image (a) shows the CDMD compression of a car image. Image (b) shows the SSIM map, i.e., the per-pixel structural similarity between the original image and its CDMD compression, in which darker pixels indicate lower similarity. Image (a) shows some artifacts on the car roof, also visible as dark regions in the SSIM map (b). Image (c) shows the SSDMD compression of the same image, with strong background simplification and high detail retention in the focus (car) area. The car-roof compression artifacts are removed, so (c) is a better representation than (a) of the original image. However, the MS-SSIM score of (c) is much lower than for CDMD compression (0.9088 vs 0.9527). The large dark areas in the background of the SSIM map (d) explain this: While our saliency map $\mu$ clearly says that background is unimportant, MS-SSIM considers it equally important as foreground, which is counterintuitive.

Figure 4.2: CDMD compression has artifacts (a) found as low-SSIM regions (b). SSDMD (c) removes these but finds subtle background differences as important for quality (d).

Given the above, saliency data should be (visually) considered in the quality metric so that the latter is more consistent with the human visual system. This is also reflected by saliency-based objective metrics reported in literatures Le Callet and Niebur (2013); Engelke and Le Callet (2015); Liu and Heynderickx (2011); Liu et al. (2013); Alaei et al. (2017).
ric as a weighting map, which improves image quality prediction performance. We follow the same idea, by integrating the spatial saliency map into the MS-SSIM (Wang et al., 2003) pooling function, as follows.

Take the MS-SSIM metric for the original image $I$ and the reconstructed image $\tilde{I}$

$$Q(I, \tilde{I}) = [\text{SSIM}(I, \tilde{I})]^{\beta_M} \prod_{j=1}^{M-1} [c_j(I, \tilde{I})]^\beta_j,$$

which is exactly Eqn. 2.9 in Sec. 2.3. We relisted it for readability and comprehension. We weigh $Q$ by the saliency map $\mu$, yielding the saliency-aware quality metric

$$Q^\mu = \left[ \frac{\sum_{x \in I} \mu(x) \text{SSIM}(x)}{\sum_{x \in I} \mu(x)} \right]^{\beta_M} \prod_{j=1}^{M-1} \left[ \frac{\sum_{x \in I_j} \mu_j(x)c_j(x)}{\sum_{x \in I_j} \mu_j(x)} \right]^{\beta_j},$$

where $\mu_j$ and $I_j$ are the saliency map and the original image $I$, respectively, at scale $j$. For notation brevity, we omitted $(I, \tilde{I})$ in $Q^\mu(I, \tilde{I})$, SSIM($I, \tilde{I}$), and $c_j(I, \tilde{I})$ in Eqn. 4.2. Using $Q^\mu$ instead of $Q$ allows in-focus values (high $\mu(x)$) to contribute more to similarity than context values (low $\mu(x)$), in line with our goal of spatially-controlled simplification.

### 4.3 Results

The proposed SSDMD method described in Sec. 4.2 adapts the original CDMD pipeline by using the spatial saliency information. We next demonstrate SSDMD, and discuss its properties, on several images. Same as Eqn. 3.3 in Chapter 3, in the following, the compression ratio of an image $I$ for SSDMD is defined as $CR = |I|/|\text{MAT}(\tilde{I})|$, i.e., the size (in bytes) of the original $I$ divided by the size (in bytes) of the MATs of the $L$ selected layers used to encode $\tilde{I}$. The latter includes the size of the encoded file that needs to be stored to reconstruct the original image using the SSDMD method.

**Increasing compression while retaining highlights** Figure 4.3 shows the simplification of three bird images by CDMD (a1–c1) and SSDMD (a2–c2). To test our new saliency-aware metric $Q^\mu$ (Sec. 4.2.1), we keep all parameter settings of CDMD and SSDMD the same, and only vary the island detection parameters $\epsilon$ and $\epsilon_0$ for CDMD and SSDMD separately. The identical parameters are set to empirically-determined values, which has been explored in Sec. 3.4, i.e., $L = 30$ and $\sigma_0 = 0.1$. Compared to CDMD, SSDMD preserves the birds’ eyes while simplifying the background more, which allows it to achieve higher compression ratios while keeping perceptually salient features.

**$Q^\mu$ achieves higher correlation with human perception** Figure 4.4 shows CDMD (a1–c1) and SSDMD (a2–c2) applied to three focus-and-context images. For each image, we indicate the standard MS-SSIM quality $Q$, spatial-saliency-aware quality $Q^\mu$, and compression ratio $CR$. The
Figure 4.3: Comparison of CDMD (a1–c1) with SSDMD (a2–c2). The compression ratio CR is indicated for each image.

\[ Q \] values for SSDMD are lower than those for CDMD, which suggests that SSDMD has a poorer quality than CDMD. Yet, we see that SSDMD produces images that are visually almost identical to CDMD, in line with the almost identical \( Q^\mu \) values for SSDMD and CDMD. Thus, we argue that \( Q^\mu \) is a better quality measure for focus-and-context simplification than \( Q \). Also, we see that, while \( Q^\mu \) stays almost identical, SSDMD compresses better than CDMD (\( CR \) values on average 24.6% higher).

**Increasing compression and/or quality** Figure 4.5 extends this insight to 150 images, selected randomly from the MSRA10K (Cheng, 2016), SOD (Movahedi and Elder, 2010), and ECSSD (Shi et al., 2016a) benchmarks. Hollow dots in Fig. 4.5 are CDMD compression results, and filled dots are SSDMD results. One dot represents the average \( Q^\mu \) and \( CR \) for a specific parameter-setting over all images in the benchmark. Same-kind dots show \( 2 \cdot 3 \cdot 3 = 18 \) different settings of the parameters \( L \), \( \varepsilon \), and \( \sigma_0 \) (actual values shown in Fig. 4.5). To find these, we first evaluated \( Q^\mu \) and \( CR \) by grid search over the full allowable ranges of \( L \), \( \varepsilon \), and \( \sigma_0 \), and then found subranges where both \( Q^\mu \) and \( CR \) yielded high val-
4.3 results

Figure 4.4: Comparison of CDMD (a1–c1) with SSDMD (a2–c2) for three focus-and-context images. For each image, we show the standard MS-SSIM quality $Q$, spatial-saliency-aware MS-SSIM $Q^s$, and compression ratio $CR$.

ues. Next, we took a few samples within these subranges, leading to the values shown in the figure. Finally, we set threshold $\mu_0 = 0.01$, i.e., keeping all but the least salient parts of the image; recall that $\mu(x) \in [0, 1]$.

As explained in Sec. 3.2.1, for color images, CDMD is applied to the individual channels of these, following representations in various color spaces. In contrast to CDMD depicted in Chapter 3, which uses the RGB color space, SSDMD choose to use YUV (more precisely, YCbCr) in all the experiments, for two reasons. First, YUV was shown to give better subjective quality than RGB due to its perceptual similarities to human vision (Podpora et al., 2014; Podpora, 2009). Secondly, since the human eye is less sensitive to the chrominance components Cb (blue projection) and Cr (red projection), strongly compressing these components achieves a higher compression ratio while keeping quality high (Nobuhara and Hirota, 2004). We see this also in Fig. 4.5: The SSDMD compression ratio (CR) of color images (red dots) is more than twice that of the grayscale images (black dots) on average, and nearly al-
ways higher than the CR of the same images computed by CDMD (red circles), while having the same quality. We also observe that for both color and grayscale images, the best CR values we obtain with SSDMD (points A, C) is about 14% higher than the best CR produced by CDMD (points B, D). Hence, SSDMD improves compression as compared with CDMD, with, as visible in Fig. 4.5, only a very slight decrease in quality $Q^\mu$. In particular, point E shows a run of SSDMD that improves on both compression (CR) and quality ($Q^\mu$) as compared to the highest-compression run of CDMD (point B). Figure 4.6 further explores this insight for six real-world images (plant, animal, natural scene, people, and man-made structure) from the MSRA10K, SOD, and ECSSD benchmarks. We show both color versions and their grayscale counterparts compressed by CDMD and SSDMD, and their corresponding CR and $Q^\mu$ values. We also show their saliency maps $\mu$ on top to illustrate what is considered focus and context. Both images and values in Fig. 4.6 show that the SSDMD method increases the compression ratio while maintaining perceived quality.

**Progressively simplification effect of $\mu_0$** As already discussed, Fig. 4.5 compares 18 different settings of the parameters $L$, $\epsilon$, and $\sigma_0$ for both CDMD and SSDMD, for a fixed value $\mu_0 = 0.01$. This was done to ease the interpretation of the respective scatterplots, as using multiple $\mu_0$ values in the same figure would have been hard to read. However, the parameter $\mu_0$ does affect the CR vs $Q^\mu$ trade-off, effectively allowing the user to specify how strongly s/he wants to simplify the image (increase CR) by trading off a certain quality amount (decrease $Q^\mu$). Figure 4.7
Figure 4.6: Comparison of the CDMD with the SSDMD method for color and grayscale versions of six input images. The top row shows the spatial saliency maps of each input image. For each image, we show the compression ratio CR and quality score $Q^\mu$.

gives insight into this, showing three images (flower in the saliency focus in all cases) for three settings $\mu_0 \in \{0.04, 0.08, 0.12\}$. The setting $\mu_0 = 0$ corresponds to CDMD. All other parameters are fixed to default values $L = 50$, $\varepsilon = 0.01$, $\sigma_0 = 0.5$, and $m = 8$. Compared with CDMD, the background areas of the SSDMD images are gradually simplified as $\mu_0$ increases; however, the flower is not changed, as it is in a high-saliency area. The $CR$ and $Q^\mu$ values shown below the images show that increasing $\mu_0$ greatly improves the compression ratio of SSDMD while quality is only slightly reduced.

**JPEG preprocessor** A final interesting use-case is to combine SSDMD’s simplification ability with a generic image compressor. For this, we ran SSDMD as a ‘preprocessor’ and subsequently compressed its re-
Figure 4.7: Progressive simplification control with the user-given threshold $\mu_0$.

Sutt with standard JPEG. Figure 4.8 shows the results of plain JPEG compression at 20% quality setting and SSDMD+JPEG for the same quality setting for three images. Values in green are the CR of SSDMD+JPEG divided by plain JPEG’s CR, i.e., the compression gain when using SSDMD as preprocessor for JPEG. This gain is 15%, 12% and 21% for the church, car, and spectacles image, respectively. For these images, the results using SSDMD+JPEG are visually almost identical in the focus areas (church building, car shape, and spectacles shape). Of course, in the context area (sky around church, scenery around car, book around spectacles) some differences are visible. This is expected — and intended —

Table 3: Performance of plain JPEG and SSDMD + JPEG under different quality settings for the images in Fig. 4.8.

<table>
<thead>
<tr>
<th>Images</th>
<th>Quality Settings (%)</th>
<th>Plain JPEG ($CR/Q^\mu$)</th>
<th>SSDMD + JPEG ($CR$ (gain)/$Q^\mu$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Church</td>
<td>40</td>
<td>61.9/0.991</td>
<td>72.1 (1.16)/0.955</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>47.1/0.994</td>
<td>55.6 (1.18)/0.958</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>30.7/0.997</td>
<td>37.5 (1.22)/0.960</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>4.3/1.0</td>
<td>6.7 (1.55)/0.961</td>
</tr>
<tr>
<td>Car</td>
<td>40</td>
<td>37.8/0.994</td>
<td>44.7 (1.18)/0.942</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>28.5/0.996</td>
<td>34.0 (1.19)/0.943</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>19.2/0.998</td>
<td>22.9 (1.19)/0.944</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>3.5/1.0</td>
<td>4.3 (1.23)/0.944</td>
</tr>
<tr>
<td>Spectacles</td>
<td>40</td>
<td>38.0/0.995</td>
<td>46.7 (1.23)/0.957</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>29.4/0.997</td>
<td>36.4 (1.24)/0.958</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>20.0/0.999</td>
<td>25.2 (1.26)/0.959</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>5.2/1.0</td>
<td>6.0 (1.15)/0.960</td>
</tr>
</tbody>
</table>
since, as explained, SSDMD aims to keep details in the focus area while simplifying them away in the context. Table 3 extends these insights by listing results under additional JPEG quality setting values for these three images. We see that the higher the quality setting, the higher the compression gain (green value) obtained by SSDMD as a preprocessor, except for the last row. In other words, for the same quality setting, SSDMD can help JPEG to increase compression rates for a minimal quality loss. This is explained by the fact that SSDMD removes small-scale sharp corners (which correspond to high frequencies in the image) in non-salient, background, image areas, thus making JPEG’s job overall easier.

Figure 4.8: Comparison of JPEG (a) with the SSDMD method applied as preprocessor to JPEG (b) for three images. Green values show CR gains as compared to JPEG.

4.4 DISCUSSION

We next discuss a few aspects of the proposed SSDMD method. **Genericity of the saliency map μ:** In general, any saliency map that encodes which image areas are more important (salient) and which not for the application at hand can be used. In contrast to segmentation tasks, we do not require precise saliency maps. Figure 4.9 (a–c) shows the SSDMD compression with three different saliency maps applied: DRFI (Jiang et al., 2013), SMD (Peng et al., 2017) and the very recent IT-
Figure 4.9: SSDMD performance with different spatial saliency maps applied.

Iterative Saliency Estimator fLexible Framework (ITSELF) (de Melo Joao et al., 2020). ITSELF’s flexibility allows significant changes in the resulting saliency maps by performing small adjustments to its parameters. Figure 4.9 (c1) is a more nuanced version created by a relaxed threshold. Just like the result obtained by the DSR saliency map in Fig. 4.6, all these three results get a higher compression ratio compared to the CDMD method in Fig. 4.6 while maintaining similar quality. Besides, users can even customize the saliency maps themselves if the available saliency map does not meet their preferences. For this example, all these saliency maps say the stop traffic sign in the image is very important. Yet, if the user does not care about the sign, but rather wants to focus on the human face in the foreground, s/he could manually tune this area to be less than \( \mu_0 \), as shown in Fig. 4.9 (d1), which is a user customization based on the DSR saliency map. This way, one can obtain a higher CR on the premise of meeting one’s quality requirements,
as shown in Fig. 4.9 (d2). We should stress again that what is a good saliency map is entirely at the user’s discretion and not a concern of SSDMD: Given a saliency map one is happy with, SSDMD compresses in low-saliency regions and preserves detail in high saliency regions.

**Ease of use:** SSDMD can be used on any image, and adds two simple-to-control parameters: the saliency-aware island threshold $\varepsilon_0$ and the spatially-regularized skeleton threshold $\mu_0$. These parameters have an intuitive meaning: $\varepsilon_0$ determines the scale of details that are kept in the image (higher values remove larger details); $\mu_0$ controls how much the background/unimportant areas are simplified (higher values simplify background more).

**Scalability:** We inherit the speed of CDMD (processing images up to $1000^2$ pixels in a few milliseconds) given by the GPU-based MAT computation. Applying the saliency map involves only two simple additional thresholding operations.

**Replicability:** We provide the full source code of SSDMD, implemented in C++ and NVidia CUDA for replication purposes (Wang et al., 2020a).

**Limitations:** SSDMD cannot yet produce higher quality and better compression ratios than JPEG. Yet, as shown in Sec. 4.3, combining it with JPEG generically increases the latter’s compression while maintaining quality. Separately, the focus-and-context compression is only as good as the quality of the used saliency maps. When such maps incorrectly mark focus details as context, these will be simplified away; conversely, when context is marked as focus, the compression ratio will be suboptimal.

### 4.5 Conclusion

In this chapter, we have presented SSDMD, a method for saliency-aware image simplification and compression. SSDMD uses dense medial skeletons and a saliency map specifying which image areas can be simplified without compromising overall image perception. Additionally, we have proposed a saliency-dependent version of the MS-SSIM metric to evaluate SSDMD on images having a focus-and-context structure. Our results show that compared with the CDMD method, SSDMD increases compression while keeping image quality high. SSDMD can also be used to improve the compression of standard JPEG though it yields slightly lower quality. Currently, SSDMD is far from competing, standalone, with JPEG2000, HEVCIntra (Nguyen and Marpe, 2012), not to mention recent image compression methods that use deep learning (Toderici et al., 2016).

However, as stated before, this was not the goal of this thesis. Rather, our purpose was to explore the potential of dense skeletons as an alternative tool to image representation. Evaluation on a benchmark composed of a broad set of images have shown that dense skeletons, when
combined with saliency maps, offer a promising tool for lossy image encoding.

So far, we have only used the raster representations of the dense skeletons, which has an inherent limitation that the compression ratio can be improved only marginally. Next, we explore a more effective way to encode skeletons, i.e., using piecewise-spline representations. In the next chapter, we start by exploring the potential of encoding binary images with medial descriptors represented by piece-wise splines.
In Chapters 3 and 4, medial axis transforms that encode color or grayscale images are modeled by a raster representation. Although comprehensive evaluations have shown a good trade-off between image compression and visual quality, there is an inherent limitation in how far raster representations can push compression. In this chapter, we use a more compact and accurate way to represent medial axis transforms for binary image representation by means of piece-wise B-splines. Evaluations on a benchmark show that our method, called Spline-based Medial Axis Transform (SMAT), achieves very high compression ratios (CRs) while keeping quality high. Compared with the regular MAT raster representation, SMAT yields a much higher CR at the expense of a slightly lower image quality. We illustrate our approach on a multi-scale SMAT representation, generating super-resolution images, as well as free-form binary image deformation.

5.1 INTRODUCTION

Binary image encoding plays a key role in applications such as image analysis, matching, and retrieval. It is also important for the compression of images and videos (Brady, 2000). The technologies for binary image encoding can be divided into three classes, namely contour-, bitmap-, and intrinsic methods.

Contour-based encoding represents binary images by a closed contour around their boundary via chain coding (Freeman, 1961) or geometrical approximations (Kim et al., 2000), as detailed further in Sánchez-Cruz et al. (2007); Žalik et al. (2018). Approximation methods reconstruct the contour in a lossy manner from a set of representative vertices via polygons or splines, and enable manipulation and deformation (Figueiredo et al., 1997; Gerken, 1994; Xiao et al., 2001; Aguilera-Aguilera et al., 2016, 2014). Bitmap techniques encode the pixels of an image as belonging to the shape (foreground) or outside it (background). These include the modified READ method (Yamaguchi et al., 1997), based on run-length encoding, and context-based arithmetic encoding (Brady et al., 1997), an efficient entropy coding scheme, which was adopted by the MPEG-4 standards (Brady, 2000).

Intrinsic image encoding considers the interior of a shape rather than its boundary. A key representative of this class uses the shape’s medial...
axis transform (MAT) to encode the shape interior (Kresch and Malah, 1998; Wang et al., 2003). This allows a flexible trade-off between approximation error and required storage (Wang et al., 2003). Recently, Zhu et al. (Zhu et al., 2014) followed earlier work that models MATs with multiple cubic B-splines (Yushkevich et al., 2003) to automatically compute a compact spline representation of the MAT of a 2D binary shape. Representing MATs with splines requires fewer (control) points to store than contour-based encoding, which can help image compression. However, this method does not offer a separate control of the MAT simplification and spline approximation.

In this chapter, we propose an alternative approach to Zhu et al. (2014) that extracts pixel-based MATs directly from binary images and represents them with splines. Our Spline-based MAT (SMAT for short) has the following features:

- Generality: SMAT can directly treat any pixel (binary) image, without requiring the extraction of a densely-sampled vector-representation of the shape contour.

- Algorithm advancement: We propose a more refined spline-fitting scheme, which includes adaptive B-spline fitting and a merge-split algorithm.

- Computational scalability: We inherit the real-time performance of the underlying GPU-based MAT extraction, allowing for high-throughput image processing applications.

- Evaluation: We show that SMAT effectively represents binary images with high accuracy and high compression ratio by measuring five quality metrics.

- Applications: We demonstrate the potential of SMAT by applications in super-resolution image generation, multiscale MAT representation, and free-form image deformation.

5.2 RELATED WORK

We structure related work into the computation of MATs (Sec. 5.2.1), medial axis accuracy metrics (Sec. 5.2.2), and spline representation models (Sec. 5.2.3).

5.2.1 Medial axis computation

As explained in Sec. 2.1.1, skeletons $S_\Omega$ of a shape $\Omega$ can be modeled as

\[ S_\Omega = \{ x \in \Omega \mid \exists f_1, f_2 \in \partial \Omega, f_1 \neq f_2: \| f_1 - x \| = \| f_2 - x \| = DT_\Omega(x) \} \]  \hspace{1cm} (5.1)
in which $DT_\Omega$ is defined as

$$DT_\Omega(x \in \Omega) = \min_{y \in \partial \Omega} \| x - y \|.$$  \hspace{1cm} (5.2)

The pair $(S_\Omega, DT_\Omega)$, called the Medial Axis Transform (MAT), allows an exact reconstruction of $\Omega$ as the union of disks centered at $x \in S_\Omega$ having radii $DT_\Omega(x)$, which makes the MAT a dual representation of a shape.

Analytic solutions of Eqn. 5.1 are very hard to compute, and require analytic descriptions of $\partial \Omega$, which are in general not available. Hence, one approximates both the input boundary $\partial \Omega$, but also the MAT itself. Two such main approximations exist: Raster methods represent $\partial \Omega$ and/or $S_\Omega$ on a fixed pixel grid; vector methods represent $\partial \Omega$ and/or $S_\Omega$ by a piecewise-continuous description in $\mathbb{R}^2$, e.g., using polylines or higher-order curves. With this model, existing MAT computation methods listed in Sec. 2.1.2 can be further classified as in Tab. 4.

**Morphological thinning** methods (Lam et al., 1992) represent both the image and MAT on pixel grids. Geometric methods (Ogniewicz and Kübler, 1995; Attali and Montanvert, 1997) find $S_\Omega$ as a subset of the edges of the Voronoi diagram of a polyline representation of $\partial \Omega$, thus representing both MAT and the shape boundary in vector form. While very accurate and compact in representation, finding a vector representation of $\partial \Omega$ for shapes provided in raster (image) form is not evident. They require specific sampling conditions for the points describing $\partial \Omega$ to be met (Amenta and Bern, 1999). The method of Zhu et al. (Zhu et al., 2014) falls in this class. Finally, **distance field** methods (Kimmel et al., 1995; Sethian, 1996; Telea and van Wijk, 2002; Falcão et al., 2004; Meijster et al., 2002; Hesselink and Roerdink, 2008) compute a raster representation of $DT_\Omega$ from either raster or vector representations of $\partial \Omega$, and next find $S_\Omega$ along singularities of $DT_\Omega$. Most current MAT methods fall in this class, given that $DT_\Omega$ can be estimated exactly and in linear time (Falcão et al., 2004; Meijster et al., 2002; Hesselink and Roerdink, 2008). Such methods can be further accelerated on the GPU, yielding real-time MAT computation (Van Der Zwan et al., 2013; Cao et al., 2010).

**Table 4:** MAT computation methods as a function of the representation of the input $\Omega$ and its MAT $S_\Omega$, with our method (SMAT) indicated.

<table>
<thead>
<tr>
<th>Rep. of $\partial \Omega$</th>
<th>raster</th>
<th>vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rep. of $S_\Omega$</td>
<td>raster</td>
<td>distance field, thinning</td>
</tr>
<tr>
<td></td>
<td>vector</td>
<td>SMAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Voronoi methods</td>
</tr>
</tbody>
</table>
Our method (SMAT) combines the advantages of distance field methods (it accepts any binary image as input, has real-time performance, and computes $D_T \Omega$ accurately) with those of a vector representation of $S_\Omega$ (built-in smoothness, compact storage). SMAT is, to our knowledge, the first method producing vector representations of the MAT from raster representations of input shapes (Tab. 4).

5.2.2 Medial axis accuracy metrics

To measure the medial axis accuracy, i.e., how close the represented shape $\hat{\Omega}$ is to the original shape $\Omega$, reliable metrics are required. A commonly used method is computing their Hausdorff distance ($H$) by considering the shape contour as point sets (Zhu et al., 2014). As also described in Sec. 2.1.3, the Hausdorff distance between $\Omega$ and $\hat{\Omega}$ is defined as

$$H(\Omega, \hat{\Omega}) = \max \{ h(\Omega, \hat{\Omega}), h(\hat{\Omega}, \Omega) \},$$

where $h(A, B)$ is the one-sided Hausdorff distance given by

$$h(A, B) = \max_{a \in \partial A} \min_{b \in \partial B} \|a - b\|.$$

While $H$ is a recognized metric for comparing contours, it is sensitive to noise or outliers (Zhang and Lu, 2004). Figure 5.1 explores this insight by showing a root shape with a resolution of 580 x 570 (a) and its SMAT reconstruction (b). We see there are tiny fragments (shown by red dotted circle) missing in the reconstruction, which is actually a small error overall, but produces a large $H$ value – 72, as indicated in the color-coded contour by the Hausdorff distance (c). Therefore, using only the $H$ metric is not comprehensive in judging the similarity of two shapes.

The above can be improved by replacing Eqn. 5.3 by a variant using the average operator, i.e.,

$$\bar{h}(A, B) = \avg_{a \in \partial A} \min_{b \in \partial B} \|a - b\|,$$

leading to the average Hausdorff distance

$$\bar{H}(\Omega, \hat{\Omega}) = \max \{ \bar{h}(\Omega, \hat{\Omega}), \bar{h}(\hat{\Omega}, \Omega) \}.$$

Besides considering the similarity of boundaries, one can take the overall shapes into account. The Jaccard similarity coefficient (Jaccard, 1912) achieves this by considering the size of the intersection, normalized by the size of the union, of two shapes

$$\text{Jaccard}(\Omega, \hat{\Omega}) = \frac{|\Omega \cap \hat{\Omega}|}{|\Omega \cup \hat{\Omega}|}.$$
5.2 related work

The Multi-Scale Structural SIMilarity (MS-SSIM) index (Wang et al., 2003; Wang et al., 2004) described in Sec. 2.3 provides an advanced top-down model of how the human visual system interprets images. Although designed to measure the similarity of grayscale images, it can also be used for binary images. Both Jaccard and MS-SSIM range in $[0, 1]$, where 1 indicates the two input shapes are exactly the same, while 0 means the two are completely different.

To comprehensively evaluate the reconstruction accuracy of SMAT, all four metrics mentioned above are adopted in this chapter.

5.2.3 B-spline representation

Storing and manipulating simplified medial axes $\tilde{S}_{\Omega}$ and their corresponding MATs can benefit from the observation that such structures correspond to piecewise smooth curves (branches) (Siddiqi and Pizer, 2008). Following this, Yushkevich et al. (2003) first proposed to model the MAT with cubic B-splines. However, their method needs to build a template continuous medial representation model manually, which is then manipulated to fit a target shape. Zhu et al. (2014) improve this by proposing a fully automatic way to represent MATs with B-splines. Yet, their method handles only vector representations (of both the shape and its medial axis). In contrast, our method uses raster representations for both $\Omega$ and $S_{\Omega}$ (Sec. 5.3), and converts the latter into a vector representation using splines. This (1) makes our method directly applicable to any binary image, without the need to extract a piecewise-continuous contour $\partial \Omega$ with sampling guarantees; and (2) provides vector-based medial representations for any raster-based MAT computation method, in contrast to Zhu et al. (2014), which only works with the Voronoi-based MAT method of Attali and Montanvert (1997).
B-splines are a common and preferred way of specifying very smooth curves ($C^{d-1}$ continuity for degree $d$) in computer graphics and geometric design (Marschner and Shirley, 2016). Given $n+1$ control points (CPs) $p_0,\ldots,p_n$ and a knot vector $U = [u_0,\ldots,u_m]$, the B-spline curve of degree $d$ is defined as (Piegl and Tiller, 1997)

$$c(u) = \sum_{i=0}^{n} N_{i,d}(u)p_i. \quad (5.8)$$

The functions $N_{i,d}(u)$ are the B-spline basis functions, defined recursively via

$$N_{i,0}(u) = \begin{cases} 1, & u_i \leq u \leq u_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$N_{i,d}(u) = \frac{u - u_i}{u_{i+d} - u_i} N_{i,d-1}(u) + \frac{u_{i+d+1} - u}{u_{i+d+1} - u_{i+1}} N_{i+1,d-1}(u). \quad (5.9)$$

A B-spline curve given by $n+1$ control points, $m+1$ knots, and degree $d$ must satisfy $m = n + d + 1$. The knot vector is either open or periodic. In this work, we use open-uniform knot vectors, given by

$$u_i = \begin{cases} a, & 0 \leq i \leq d \\ a + \frac{i-d}{n+1-d}(b-a), & d < i \leq n \\ b, & n < i \leq n + d + 1 \end{cases}, \quad (5.10)$$

where $a$ and $b$ are usually set to 0 and 1 respectively. This allows generating a B-spline curve based only on a set of $n+1$ control points and degree $d$ with $0 < d < n+1$.

5.3 Proposed SMAT Representation

We compute our SMAT representation as follows (see also Fig. 5.2). We start with a binary shape $\Omega$ as input. For simplicity of exposition, we next consider $\Omega$ has a single foreground connected component (black pixels in Fig. 5.2a); in practice, our implementation handles multiple such components, one at a time. We next compute the full MAT ($S_{\Omega}, DT_{\Omega}$) using the GPU-based AFMM method (Telea, 2014), as stated in Sec. 2.1.2. However, any other raster-based MAT computation can be used as well. Next, we compute the simplified medial axis

$$\tilde{S}_\Omega = \{x \in S_\Omega \mid \sigma(x) \geq \sigma_0\}, \quad (5.11)$$

by upper-thresholding the salience metric $\sigma$ (Sec. 2.1.3) by a user-specified value $\sigma_0$. The simplified MAT is given by $(\tilde{S}_\Omega, DT_{\Omega})$, where $DT_{\Omega}$ is the restriction of $DT_{\Omega}$ to the pixels of $\tilde{S}_\Omega$. We use the simplified MAT as input for our SMAT construction. For this, we segment $\tilde{S}_\Omega$
5.3 PROPOSED SMAT REPRESENTATION

Figure 5.2: Pipeline of the proposed SMAT representation.

into separate branches (Sec. 5.3.1) and fit them using splines (Sec. 5.3.2). Finally, we encode the entire set of spline control points efficiently (Sec. 5.3.3). The resulting encoding can be then used to reconstruct an approximation $\tilde{\Omega}$ of the input shape $\Omega$ (Sec. 5.3.4).

5.3.1 Medial axis segmentation

To carry out the piecewise B-spline fitting, the simplified medial axis should be segmented into branches (Fig. 5.2, Step 3). Algorithm 1 outlines the segmentation procedure. First, we clean up $\tilde{S}_\Omega$ so it is 8-connected. We next characterize medial points by the number of neighbors in $\tilde{S}_\Omega$ they have, as follows: Branch endpoints have a single neighbor; regular points have two neighbors; and branch junctions have three or more neighbors. We then find an endpoint (or an arbitrary point if no endpoints exist) $x$ of $\tilde{S}_\Omega$ and start tracing along the medial axis from there, adding the discovered MAT points $(y, DT_\Omega(y))$ to the current branch $B$. When arriving at a junction or endpoint, we add $B$ to the branch-set $B$ and, if at a junction, start tracing new branches from all medial neighbors $n$ of the current point $y$.

5.3.2 B-spline fitting

To each branch $B_i = \{(x \in \tilde{S}_\Omega, DT_\Omega(x))\}$ found in the branch-set $B = \{B_i\}$, we fit a B-spline curve $C_i$ using a least-squares algorithm (Fig. 5.2, step 4). For details of the least-squares fit, we refer to Eberly (2014). Given a user-provided fitting error $\gamma_0$ between $B_i$ and $C_i$, we compute the minimal number of needed control points $N$ and spline de-
Algorithm 1: Skeleton segmentation algorithm

Input: Simplified MAT \( \hat{S}_\Omega, \hat{D}T_\Omega \)
Output: Set \( B \) of medial branches
1. Make \( \hat{S}_\Omega \) 8-connected
2. Find \( x \) = a point in \( \hat{S}_\Omega \)
3. \( B = \emptyset \)
4. Trace(\( x, B \))
5. return \( B \)

Function Trace(\( y, B \)):
6. \( B.\text{push}_\text{back}(y, \hat{D}T_\Omega(y)) \)
7. erase \( y \) from \( \hat{S}_\Omega \)
8. if size(\( y.\text{neighbors} \)) = 1 then
9. \( \text{Trace}(y.\text{neighbors}[0], B) \)
10. else
11. \( \text{add} B \text{ to } B \)
12. for \( n \) in \( y.\text{neighbors} \) do
13. \( \text{Trace}(n, \emptyset) \)

5.3.2.1 Adaptive-degree fitting

Although it is common to use quadratic \((d = 2)\) or cubic \((d = 3)\) B-splines to approximate a set of points, we compute \( d \) so as to get the lowest number of control points \( N \) needed to reach an error below the user-given threshold \( \gamma_0 \). Computing \( d \) follows the constraints \( 1 \leq d < N \) (see Sec. 5.2.3) and \( N \geq 2 \), where \( N = 2 \) implies a line segment fit to the branch \( B \). The procedure (Algorithm 2) is a global minimization of \( N \) for all possible \( d \) values within a maximum number of iterations MaxIter (set in practice to 1000) to speed up computations for large branches. The fitting error \( \gamma \) of the spline \( C \) to branch \( B \) is given by the Hausdorff distance \( H(B, C) \) (Eqn. 5.3) computed over all the pixels \( x \in B \). It is known that the boundary approximation error \( H(\Omega', \hat{\Omega}) \) between \( \Omega' \) (the reconstruction from the simplified MAT \( (\hat{S}_\Omega, \hat{D}T_\Omega) \)) and \( \hat{\Omega} \) is upper bounded by \( \sqrt{2}y_{\text{max}} \) (Kosinka and Jüttler, 2006), where \( y_{\text{max}} \) is the maximal fitting error over all splines \( C_i \). Thus, users can set \( \gamma_0 \) either directly — to control the medial axis approximation error — or based on the desired boundary approximation error \( H(\Omega', \hat{\Omega}) \).

Figure 5.3 compares our adaptive B-spline fitting with quadratic and cubic B-spline fitting for a simple shape. Figure 5.3 (a) shows the image \( \Omega \) (black) and its medial axis \( \hat{S}_\Omega \) (white). We set here \( \gamma_0 = 0.5\% \) of the image diagonal. As visible, our adaptive fitting (Fig. 5.3 (d)) requires only \( N = 30 \) control points, whereas quadratic and cubic B-splines require \( N = 41 \) and \( N = 37 \) control points, respectively. Table 5 lists the number of control points and degree for the four medial branches of the shape, for each of the above three spline-fitting schemes. For long and
Algorithm 2: Adaptive B-spline fitting algorithm

\begin{verbatim}
Input: MAT branch \( B \) and user-given maximal error \( \gamma_0 \)
Output: B-spline curve \( C \) fitted to \( B \) under error \( \gamma_0 \)
1 Initialization: \( N = 2; \gamma_{\text{min}} = \text{INFINITY} \)
2 for \( i \) from 0 to MaxIter do
3     for \( d \) from 1 to \( N - 1 \) do
4         \( C = \text{LeastSquaresFit}(d, N, B) \)
5         \( \gamma = H(B, C) \)
6         if \( \gamma < \gamma_{\text{min}} \) then
7             \( \gamma_{\text{min}} = \gamma; d_{\text{min}} = d \)
8         if \( \gamma_{\text{min}} < \gamma_0 \) then
9             break
10        else
11            \( N++ \)
12     \end{verbatim}

Figure 5.3: Comparison of the adaptive B-spline fitting (d) with quadratic (b) and cubic (c) B-spline fitting for a given image (a) and error \( \gamma_0 = 0.5\% \) of the image diagonal. For (b–d), different colors show different branches. Control points are colored like their branches.

curved branches, like \( B_4 \) (black), our scheme uses a high degree \( d = 5 \) to reduce the required \( N \); for short and straight branches, like \( B_3 \) (green),

Table 5: Number of control points \( N \) needed to fit each of the four medial branches for the quadratic, cubic, and adaptive schemes in Fig. 5.3. Values in brackets give the degree \( d \) of each B-spline.

<table>
<thead>
<tr>
<th>Branch</th>
<th>Quadratic</th>
<th>Cubic</th>
<th>Adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_1 ) (blue)</td>
<td>10 (2)</td>
<td>8 (3)</td>
<td>8 (3)</td>
</tr>
<tr>
<td>( B_2 ) (red)</td>
<td>9 (2)</td>
<td>11 (3)</td>
<td>9 (2)</td>
</tr>
<tr>
<td>( B_3 ) (green)</td>
<td>3 (2)</td>
<td>4 (3)</td>
<td>2 (1)</td>
</tr>
<tr>
<td>( B_4 ) (black)</td>
<td>19 (2)</td>
<td>14 (3)</td>
<td>11 (5)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>41</strong></td>
<td><strong>37</strong></td>
<td><strong>30</strong></td>
</tr>
</tbody>
</table>
our method reduces $N$ by using a lower degree $d = 1$ than the quadratic and cubic schemes. Overall, our adaptive-degree B-spline fitting saves about 20% control points.

### 5.3.2.2 Merge-split algorithm

Algorithm 2 describes the adaptive B-spline fitting for a single branch. When the medial axis $S_\tilde{\Omega}$ is only slightly simplified, $\tilde{\Omega}$ contains many short branches corresponding to small-scale details along $\partial \Omega$. Encoding these requires many control points, so we propose to merge these short branches to alleviate this. For each branch-fragment $A$, we find all fragments $B_j$ connected to it, and select the most suitable one $B_j$ to merge with $A$. The criterion for the merge is

$$j = \arg \min_i N_{A+B_i} | N_{A+B_j} < N_A + N_{B_j},$$

(5.12)

where $N_{A+B_i}$ is the number of control points needed to fit the merged branch $A \cup B_i$, and $N_A$ and $N_{B_j}$ are the number of control points required for branch-fragments $A$ and $B_j$, respectively.

A separate issue is that there may be long and curved branches which are difficult to fit with a B-spline, requiring more than $N_{max}$ (set to 15 in practice) control points. We address this by splitting such branches before fitting. Let $B_i = \{(x_k \in \tilde{S}_\Omega, \tilde{DT}_\Omega(x_k))\}_{k=0}^n$ be the branch to be split; then $B_{i1} = \{(x_k \in \tilde{S}_\Omega, \tilde{DT}_\Omega(x_k))\}_{k=0}^{n/2}$ and $B_{i2} = \{(x_k \in \tilde{S}_\Omega, \tilde{DT}_\Omega(x_k))\}_{k=0}^{n/2+1}$ are the two branches obtained by splitting $B_i$ in half. We consecutively split long and curved branches in half until all resulting branch-segments can be fitted by Algorithm 2 with splines with fewer than $N_{max}$ control points, and the fitting error is under the user-specified $\gamma_0$.

To illustrate the effect of the merge-split (MS) algorithm, Fig. 5.4 compares SMAT results for a lizard shape (taken from Zhu et al. (2014)) when applying the MS algorithm (b) and not applying it (a). We set $\gamma_0 = 0.35\%$ of the diagonal of the image. MS creates a SMAT using only 57 control points instead of the 65 required if one B-spline per MAT branch is used. The negative numbers in Fig. 5.4 show the drop in control points, per branch, due to MS. The long and curved branch along the tail is split into two segments (black and green). Fewer control points are needed after splitting since the two segments can be separately fit with splines using different degrees. For short and straight branches, like the ones corresponding to a finger of the fore leg, merging them into one branch also decreases the control point count. Additionally, using MS greatly reduces the total number of splines from 27 to 20 used since several connected MAT branches can be approximated by a single spline. More information on the advantages of MS for reducing the information needed to store the SMAT is given in the supplementary material (Wang et al., 2021f).
5.3.3 SMAT encoding

Our SMAT representation (Fig. 5.2, Step 5) consists of a tuple $SM = (w, h, \{b_i\})$. Here, $w$ and $h$ represent the width and height, in pixels, of the input image $\Omega$; and $b_i = (d_i, \{c^j_i\})$ represents a B-spline output by the merge-split process (Sec. 5.3.2.2), i.e., which fits a merged branch under the user-given error $\gamma_0$. Each such B-spline has a degree $d_i$ and control points $c^j_i$ (both computed by the adaptive procedure in Sec. 5.3.2.1). Each control point $c^j_i = (p^j_i, DT(\Omega(p^j_i))) \in \mathbb{R}^3$ consists of a 2D position $p^j_i$ and its corresponding DT value.

5.3.4 Reconstruction

We reconstruct the approximation $\tilde{\Omega}$ of $\Omega$ from the SMAT representation (Sec. 5.3.3) as follows. For each spline $b_i$, the open-uniform knot vector can be determined following Eqn. 5.10. The basis functions $N_{l,q}(u)$ are next computed using Eqn. 5.9, followed by generating the B-spline (Eqn. 5.8). Each such spline is next rasterized on the desired pixel grid, using either the original image resolution $(w, h)$ or, if desired, a higher resolution (see next Sec. 5.5.1). For rasterization, we first split each branch $b_i$ into Bézier (polynomial) segments. This uses knot insertion to ensure that each internal knot has multiplicity equal to $d_i$, which is done efficiently using the Oslo algorithm (Cohen et al., 1980). Each Bézier segment is then rasterized using adaptive binary subdivision based on de Casteljau’s algorithm (Piegl and Tiller, 1997). The adaptive subdivision proceeds until the maximum distance of all inner Bernstein-Bézier control points from the line-segment given by the endpoints of the current (sub-)segment is below pixel precision, at which
point the (sub-)segment is drawn using Bresenham’s line-drawing algorithm (Bresenham, 1965) based on the mentioned end-points.

Once the medial pixels $x_i$ with DT values $DT_i$ are evaluated this way, we reconstruct $\hat{\Omega}$ as the union of disks with centers $x_i$ and radii $DT_i$, as described in Sec. 2.1.4.

5.3.5 Implementation

We implemented the MAT simplification (Eqn. 5.11), Algorithms 1 and 2, the SMAT encoding (Sec. 5.3.3), and spline rasterization (Sec. 5.3.4) in C++. We compute exact Euclidean MATs and also reconstruct the initial shape from a rasterized SMAT using the public CUDA implementation provided at Telea (2019). Our entire method, including source code, datasets, and evaluation scripts, is publicly available (Wang et al., 2021f).

5.4 Results

5.4.1 Evaluation methodology

SMAT depends on two parameters (Fig. 5.2): the salience threshold $\sigma_0$, which gives the simplification of the medial axis $\hat{S}_\Omega$, and the tolerance $\gamma_0$ that tells how accurately B-splines fit medial branches. We evaluate SMAT based on two factors:

**Similarity $Q$** of the reconstruction $\hat{\Omega}$ provided by SMAT (Sec. 5.3.4) to the original shape $\Omega$. Here, $Q$ stands for any of the metrics $H$, $H_{\overline{\Omega}}$, MS-SSIM, and Jaccard (Sec. 5.2.2).

**Compression ratio $CR$** that measures how more compact $SM(\hat{\Omega})$ is as compared to $\Omega$. We define $CR = |\Omega|/|SM(\hat{\Omega})|$. Here, $SM(\hat{\Omega})$ is the size (in bytes) of the SMAT storage scheme outlined in Sec. 5.3.3. In contrast, $|\Omega|$, i.e., the storage needed for shape $\Omega$, can be defined in many ways, depending on how $\Omega$ is represented, e.g. by chain coding (Freeman, 1961), geometrical approximation (Kim et al., 2000), or bitmap-based encoding (Yamaguchi et al., 1997; Brady et al., 1997). We next model $|\Omega|$ as the raw size (in bytes) of the image $\Omega$.

Given the above, the total quality of SMAT can be modeled as

$$(Q, CR) = \text{SMAT}(\sigma_0, \gamma_0).$$

To find a good trade-off between $Q$ and $CR$, we do a grid-search over $\sigma_0$ and $\gamma_0$. We use $\sigma_0 \in \{0.1, 0.5, 1.0, 1.5\}$ as this range was indicated in the original salience paper (Telea, 2012) as producing MAT simplifications that remove small-scale noise but keep salient details. We use $\gamma_0 \in \{0.002, 0.004, 0.006, 0.008\}$ (percentages of the bounding-box diagonal of $\Omega$) as these represent tight fits of the SMAT B-splines with the branches of $\hat{S}_\Omega$. Table 6 shows the resulting values for all four similarities $Q$, the compression ratio $CR$, and also the total number of control
points \( N \) in the SMAT, for all values of \( \sigma_0 \) and \( \gamma_0 \), for a binary image also used in Siddiqi et al. (2002); Telea and van Wijk (2002). Several other examples are available in the supplementary material (Wang et al., 2021f).

<table>
<thead>
<tr>
<th>Shape</th>
<th>( \sigma_0 ) (%)</th>
<th>( \gamma_0 ) (%)</th>
<th>( \tilde{H}(%))</th>
<th>( H(%))</th>
<th>Jaccard</th>
<th>MS-SSIM</th>
<th>CR</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Animal</td>
<td>0.1</td>
<td>0.2</td>
<td>0.47</td>
<td>0.11</td>
<td>0.981</td>
<td>0.986</td>
<td>2.1</td>
<td>540</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.47</td>
<td>0.12</td>
<td>0.979</td>
<td>0.984</td>
<td>3.0</td>
<td>358</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>0.47</td>
<td>0.13</td>
<td>0.978</td>
<td>0.984</td>
<td>3.3</td>
<td>327</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>0.60</td>
<td>0.13</td>
<td>0.978</td>
<td>0.983</td>
<td>3.3</td>
<td>329</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0.2</td>
<td>0.78</td>
<td>0.23</td>
<td>0.960</td>
<td>0.958</td>
<td>10.2</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.69</td>
<td>0.18</td>
<td>0.970</td>
<td>0.970</td>
<td>10.6</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>0.69</td>
<td>0.22</td>
<td>0.960</td>
<td>0.959</td>
<td>11.3</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>0.88</td>
<td>0.20</td>
<td>0.966</td>
<td>0.966</td>
<td>12.6</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.2</td>
<td>0.88</td>
<td>0.27</td>
<td>0.952</td>
<td>0.946</td>
<td>24.4</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1.2</td>
<td>0.24</td>
<td>0.957</td>
<td>0.953</td>
<td>28.0</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>1.1</td>
<td>0.26</td>
<td>0.955</td>
<td>0.951</td>
<td>33.9</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>1.2</td>
<td>0.29</td>
<td>0.949</td>
<td>0.942</td>
<td>35.2</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>0.2</td>
<td>1.2</td>
<td>0.28</td>
<td>0.950</td>
<td>0.944</td>
<td>24.9</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1.1</td>
<td>0.24</td>
<td>0.958</td>
<td>0.954</td>
<td>28.1</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>1.3</td>
<td>0.26</td>
<td>0.953</td>
<td>0.947</td>
<td>33.6</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>1.1</td>
<td>0.27</td>
<td>0.952</td>
<td>0.946</td>
<td>36.4</td>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>

To understand the trends in Tab. 6, we use next two scatterplots (Fig. 5.5) showing \( H \) vs \( CR \) and MS-SSIM vs \( CR \), respectively. The plots of \( \tilde{H} \) vs \( CR \) and Jaccard vs \( CR \) are similar to Figs. 5.5 (a) and Figs. 5.5 (b), respectively, and are not included for space constraints. In each plot, the sixteen points correspond to combinations of \((\sigma_0, \gamma_0)\) values. We encode \( \sigma_0 \) in four base colors (hues), and \( \gamma_0 \) in the size of the bullets. Figure 5.5 (a) shows a roughly direct correlation of \( H \) with \( CR \), while Fig. 5.5 (b) shows an inverse correlation of MS-SSIM with \( CR \), as expected. In general, the larger \( \sigma_0 \), the more simplified \( \tilde{S}_\Omega \), so the lower the MS-SSIM and the higher the \( H \) and \( CR \). Overall, the quality has not decreased much, but the \( CR \) has been greatly improved. Under a certain \( \sigma_0, \gamma_0 \) determines how close the B-spline is to the current skeleton. Although there is no very strict trend, in general, with the increase of \( \gamma_0 \), \( Q \) tends to decrease while \( CR \) increases.

### 5.4.2 Joint compression-quality evaluation

We next evaluate SMAT’s ability to compress images and retain similarity by a benchmark containing 30 images of five different types (Tab. 7 and Fig. 5.6), mainly selected from the MPEG-7 benchmark (Ralph, 2019). To optimize both reconstruction similarity \( Q \) and \( CR \) for each shape, we next set \( Q = \text{MS-SSIM} \), as we argue that, from the set of
four considered similarity metrics, MS-SSIM best represents how humans perceive two shapes $\Omega$ and $\tilde{\Omega}$ as being similar. Also, we combine MS-SSIM and CR into a simple joint quality metric

$$Q' = \frac{\text{MS-SSIM} + \text{CR}}{2},$$ \hspace{1cm} (5.14)

where $\text{CR}$ for a given shape is its $\text{CR}$ value normalized by the maximal $\text{CR}$ over all shapes in the benchmark. This way, both $\text{CR}$ and MS-SSIM range in $[0, 1]$, so can be combined in Eqn. 5.14. The optimization is the result that maximizes $Q'$ over all 16 studied $(y_0, \sigma_0)$ values.

Figure 5.6 shows the obtained results. Rows indicate shapes in the five benchmark classes (see Tab. 7). Cyan shows the reconstruction $\tilde{\Omega}$ with optimal $Q'$. Black outlines show the boundaries $\partial\Omega$ of the input shapes. In all cases, the reconstruction is visually almost identical to the original shape, also confirmed by the high MS-SSIM values. Compression values $\text{CR}$ also are in general quite high, less so for shapes having many small-scale details such as (d6) and (e6). Images (a6), (e1), and (e6) show that SMAT can handle shapes with holes with no problem. Relative boundary length, defined as boundary length $|\partial\Omega|$ divided by the diagonal of the image, is color-coded in the legend in Fig. 5.6. Figure 5.7 summarizes the SMAT performance on the 30 images in Fig. 5.6 by a scatterplot of MS-SSIM vs $\text{CR}$. Colors indicate the relative boundary length just as in Fig. 5.6. We see a slight inverse correlation of high $\text{CR}$ and MS-SSIM with the relative boundary length, which is expected:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Simple object shapes</td>
</tr>
<tr>
<td>b</td>
<td>Regular geometric structures</td>
</tr>
<tr>
<td>c</td>
<td>Animal contours</td>
</tr>
<tr>
<td>d</td>
<td>Geometric shapes with jagged or irregular edges</td>
</tr>
<tr>
<td>e</td>
<td>Shapes with complex contours</td>
</tr>
</tbody>
</table>
shorter boundaries have, on average, fewer details, so are easier to encode by SMAT. This is also visible in the fact that complex shapes (types d and e) reach lower CR and MS-SSIM, while simpler shapes (types a and b) reach higher CR and MS-SSIM. We also see the relatively strong effect that even tiny boundary details have on CR: Shapes a1, b1, b4, and b2 are arguably of very similar visual complexity and all have short boundaries (dark blue in Fig. 5.7). All compress with a very high MS-SSIM $> 0.985$. However, their CR’s vary between 37.2 (a1) and 133 (b2). This is caused by tiny, pixel-size, noise, present e.g. along a1’s boundary, but largely absent for the other three shapes. To keep such tiny details, we need to keep many branches in the MAT.
5.4.3 Comparison with compressed MAT representation

Besides comparing SMAT with the ground-truth $\Omega$, it is interesting to compare it with other methods that provide compressed MAT representations. Our method described in Chapter 3 essentially uses the same MAT extraction (Telea and van Wijk, 2002) and simplification (Telea, 2012) as SMAT, but next compresses the pixel-chains in $\tilde{\Omega}$ using delta encoding rather than B-splines as this chapter does. Figure 5.8 shows the average MS-SSIM vs CR for the five shape types in our benchmark for SMAT and the delta method in Chapter 3. For that method, we define $\gamma = |\Omega|/|\text{MAT}(\tilde{\Omega})|$, where $\text{MAT}$ is the size (in bytes) needed to store $\tilde{\Omega}$ with delta encoding for a binary image, which is an efficient way to store 8-connected pixel-paths. As in our case, (Sec. 5.4.1), $|\Omega|$ denotes the raw size (in bytes) needed to store $\Omega$.

In the figure, the larger the $\gamma_0$ (the larger the filled dots), means the larger the approximation error of the spline, so the lower the quality, the higher the $\gamma$. When $\gamma_0$ equals 0.002, the MS-SSIM score of SMAT is only 0.002 lower than the one of the delta encoding, but SMAT yields CR values 2 up to 6 times higher.

5.4.4 Comparison with Zhu et al.

Result quality: Figure 5.10 compares SMAT with Zhu et al. (2014), which, as mentioned earlier, is the most similar method (in aims) to ours. This comparison must however be done carefully. As mentioned (Sec. 5.2.3), their input shape boundaries $\partial \Omega$ must be carefully (densely) sampled to capture the boundary topology faithfully (Amenta and Bern,
Figure 5.8: Average MS-SSIM vs CR for the five shape types in our benchmark for delta-encoding MAT in Chapter 3 (asterisks) and SMAT for three different $\gamma_0$ values (filled dots in three different sizes).

1999). How this is done is not further detailed. Also, they use for reconstruction error $Q$ the one-sided Hausdorff distance (Eqn. 5.4) from these sample points $P \in \mathcal{P}$ of $\partial \Omega$ to the boundary $\partial \tilde{\Omega}$ of the reconstructed shape. This leads to a smaller distance value than if all points of $\partial \tilde{\Omega}$ were considered. Figure 5.9 illustrates this. If the (one-sided) Hausdorff distance is defined as $h(\Omega, \tilde{\Omega}) = \max_{x_i \in P} \left\{ \min_{x_j \in \partial \tilde{\Omega}} \|x_i - x_j\| \right\}$ as in Zhu et al. (2014), then some of the distance values (like $l$ in the figure) will be less than (maximally equal to) values obtained when considering all points on $\partial \tilde{\Omega}$ (like $l'$ in the figure). In addition, they only considered the one-sided Hausdorff distance $h(\Omega, \tilde{\Omega})$, for reasons deemed as simplicity. The other one-sided Hausdorff distance, $h(\tilde{\Omega}, \Omega) = \max_{x_j \in \partial \tilde{\Omega}} \left\{ \min_{x_i \in P} \|x_j - x_i\| \right\}$, would generate a higher value than $h(\Omega, \tilde{\Omega})$ (denoted $m$ in the figure).

In contrast to the above, Fig. 5.10 (a2–c2) shows the two-sided $H$ (Eqn. 5.3), and considers every pixel on both boundaries $\partial \Omega$ and $\partial \tilde{\Omega}$. We used here images extracted from Zhu et al. (2014), since no data or code for that method was available. In this image, $e$ is the one-sided Hausdorff distance used by Zhu et al. (2014) explained above. The other metrics are ours, explained earlier. Overall, SMAT produces results that are very similar to Zhu et al., but requires 15% fewer control points $N$. Importantly, $H$, the double-sided Hausdorff distance we use for SMAT, is always larger than the one-sided Hausdorff distance $e$ used by Zhu et al., by definition – meaning that our test for accurate reconstruction
Figure 5.9: Analysis of Hausdorff distance used in Zhu et al. (2014). $P$ is one of the sample points of the original shape boundary $\partial \Omega$. $\partial \tilde{\Omega}$ is the boundary of the reconstructed shape.

Figure 5.10: Comparison of SMAT (a2–f2) with Zhu et al. (2014) (a1–f1) for six shapes. Blue shows the reconstructed shapes $\tilde{\Omega}$. Black shows boundaries of the input shapes $\Omega$. White curves and black dots in (a1–f1) show the B-spline curves and their corresponding control points. In (a2–f2), different colors show different branches and their control points. The errors $\varepsilon$, $H$ and $\overline{H}$ are in percentages of the diagonal of the image. Timings $t$ for (a1–f1) cover only the spline extraction from simplified MATs in Zhu et al. (2014). Our timings $t$ (a2–f2) are the end-to-end costs.

is more stringent than the one in Zhu et al.

Parameters: Zhu et al. offer a single parameter $\hat{e}$ which controls both the MAT simplification and the spline fitting. Small $\hat{e}$ values, thus, will
keep most of the raw MAT branches and also tightly fit B-splines to them. Conversely, large \( \varepsilon \) values will simplify the MAT significantly and fit B-splines looser. In contrast, we separate the two concerns: Our parameter \( \sigma_0 \) controls the MAT simplification, thereby allowing us to specify what (of the MAT) we next want to approximate. Next, \( \gamma_0 \) controls the spline fitting, i.e., how well we want to approximate the simplified MAT. This separation has several advantages: (1) We can e.g. decide we want to keep the nearly-complete MAT (low \( \sigma_0 \)) but approximate it more loosely (high \( \gamma_0 \)), or simplify the MAT significantly (high \( \sigma_0 \)) but fit it tightly with splines (low \( \gamma_0 \)). (2) We can use any other MAT simplification besides the saliency, simply by replacing the definition of \( \tilde{S}_\Omega \) with any other MAT simplification deemed suitable; the rest of the pipeline stays unchanged. (3) We can compute multiscale SMATs by computing the full MAT only once and next simplify it using any desired combination of \( \sigma_0 \) and \( \gamma_0 \), at interactive rates (see Sec. 5.5.2 for details).

**Speed:** Zhu et al. only report the cost of computing the final B-splines from the simplified MATs. Yet, going from the raw MATs (produced by the Voronoi method (Attali and Montanvert, 1997)) to the simplified MATs takes tens of seconds for a shape of \( 512^2 \) pixels. In contrast, our total time, from receiving \( \Omega \) up to computing \( SM(\tilde{\Omega}) \), is only tens of milliseconds (see Fig. 5.10 (a2–f2)).

## 5.5 Applications

In addition to the shape or MAT compression (Sec. 5.4), SMAT also provides other useful results. We outline here super-resolution medial axes (Sec. 5.5.1), multiscale SMAT representations (Sec. 5.5.2), and shape manipulation by MAT control points (Sec. 5.5.3).

### 5.5.1 Super-resolution

As B-splines are smooth, one can rasterize them at any resolution in the reconstruction step (Sec. 5.3.4). This does not incur any extra storage: the same SMAT representation (Sec. 5.3.3) can be used. Figure 5.11 shows the effect of increasing the resolution by 10 times on a jagged shape. The top images show the SMAT reconstruction of a shape at its original resolution (200×200 pixels). Bottom images show the SMAT reconstruction, from the same \( SM \) representation, at a 10 times higher resolution than the input image. As visible, the super-resolution reconstruction removes the discretization artifacts of the original reconstruction while keeping the reconstructed boundary \( \partial \tilde{\Omega} \) (line separating black from white in the image) smooth. To our knowledge, this is the only method providing super-resolution raster MATs apart from Strzodka and Telea (2004). In comparison to Strzodka and Telea (2004), SMAT is
fully generic and simpler, i.e., does not need to use special (image-based) interpolation tricks to create the super-resolution MATs.

### 5.5.2 Multiscale SMAT representation

SMAT uses as input the simplified MATs \( S_\Omega \) of its input shapes \( \Omega \) (Sec. 5.3). These simplified MATs depend on the user-provided salience parameter \( \sigma_0 \). In practice, this means that if a user simplified \( S_\Omega \) too much (by setting \( \sigma_0 \) too high), one would have to re-run the entire SMAT pipeline, which is tedious and time consuming. We improve this by proposing a multiscale SMAT representation. Instead of simplifying the MAT (Fig. 5.2, Step 2), we encode the full MAT together with the importance (\( \rho \)) value at each MAT point as a 4D curve-set \( (S_\Omega, x, S_\Omega, y, DT_\Omega, \rho) \), using B-splines. As we encode both \( DT_\Omega \) and \( \rho \), we can next compute the salience \( \sigma \) using Eqn. 2.7.

A single multiscale SMAT allows generating an entire family of progressively simplified MATs, and their corresponding reconstructions, by running only the \( \sigma_0 \) thresholding on the decoded \( S_\Omega \) produced by Step 6 in Fig. 5.2, followed by reconstruction (Step 7). Figure 5.12 (b) shows the 4D data \( (S_\Omega, x, S_\Omega, y, DT_\Omega, \rho) \) in red, fitted by B-splines (green) for the jagged shape in Fig. 5.11. Videos showing additional results on this experiment are in the supplementary material (Wang et al., 2021f). Figures 5.12 (c1–c6) show the gradually-simplified MATs \( S_\Omega \) and corresponding reconstructions \( \tilde{\Omega} \) for increasing thresholds \( \sigma_0 \). Multiscale SMAT is cost-effective: obtaining the six simplifications in Fig. 5.12 (c1–c6) is about five times faster than running six single-scale SMATs. Note
that producing such multiscale SMATs is not possible with Zhu et al. (2014): Indeed, for any change of the error user parameter $\epsilon$ proposed there, the entire pipeline of MAT simplification and spline-fitting has to be re-run, which is expensive, as outlined in Sec. 5.4.4.

### 5.5.3 Shape manipulation

Manipulating 2D shapes is important in applications like character animation (Sýkora et al., 2009; Weng et al., 2006) and image editing (Mota et al., 2011). Several methods such as $m$-reps (Pizer et al., 2003), subdivision skeletons (Angelidis and Cani, 2002), and medial surface deformation (Yoshizawa et al., 2007) have used MATs to this end, by changing the shape via changing its medial axis. Still, picking suitable control points to manipulate the MAT is not easy. In contrast, SMAT allows deforming a shape simply by manipulating the SMAT descriptor. Figure 5.13 shows five deformations where we keep the human body and horse rump fixed and change the horse legs by adding (+c), deleting (−c), moving (m) control points and/or increasing the B-spline degrees (+d, −d). A few of the manipulations are outlined next. To overlap the two hind legs and vary the curvature of their joint silhouette (d, e) we increased the number of control points for the leg-to-body connection branch (dark blue) and also reduced the number of control points for these legs. A similar edit was done to the purple branch in (f) to overlap the front legs. To get a curled up right front leg (dark red, image (e)), we reduced the degree of its B-spline to 1 so as to better control it to clearly show each leg joint, and next moved its control points as desired.

Figure 5.12: Multiscale SMAT fitting (b) and the progressively simplified skeletons and the corresponding reconstruction (c1–c6) on a jagged shape (a).
5.6 Discussion

We next discuss several aspects of our SMAT representation.

Ease of use: SMAT has two parameters that affect the trade-off between compression ratio and reconstruction accuracy: $\sigma_0$ and $\gamma_0$. Intuitively, $\sigma_0$ controls how many small-scale details (bumps) on the shape boundary are removed (MAT simplification); while $\gamma_0$ controls how much the MA branches are ‘smoothed out’ into splines (MAT approximation).

Speed: Since our MAT is implemented on the GPU (Sec. 5.2.1), the SMAT pipeline is very fast. For a shape of $512^2$ pixels, the entire pipeline takes only a few tens of milliseconds on a Linux PC with an Nvidia RTX 2060 GPU. Following the CDMD described in Chapter 3, the complexity of the SMAT computation is linear in the number of pixels in the input image.

Comparison: It is useful to summarize the differences between SMAT and Zhu et al. (2014). A key point is that Zhu et al. simplify MATs based on the reconstruction error, which naturally yields small errors. SMAT simplifies MATs based on how salient their points are for shape perception. Hence, our reconstructions can have a possibly larger Hausdorff distance $H$ to the initial shape, as we are not explicitly optimizing for $H$. Both approaches are valid but for different goals: If one wants simplified MATs that reconstruct a shape as closely as possible with respect to $H$, and is fine with computation times of minutes, Zhu et al. is to be used. If one accepts small reconstruction errors in non-salient shape parts, desires a multiscale MAT for simplification with different thresholds, and needs a real-time response, then our method is to be used.

Figure 5.13: Deformations of a running horse shape by manipulating the control points and the degree of B-splines. Control points and manipulations are colored just as the B-spline they affect.
Limitations: SMAT cannot (yet) be lossless, i.e., yield an exact, zero Hausdorff-distance-to-original, reconstruction. To fully reconstruct the input shape, the full, unsimplified, MAT should be used. Figure 5.14 (a) shows the full medial axis of a simple rectangular shape. As Fig. 5.14 (b) shows, the full-MA branches are very close. Algorithm 1 will segment this MA into tens of thousands of very short branches (shown in Fig. 5.14 (c) with one color per such branch). The spline fitting (Secs. 5.3.2.1, 5.3.2.2) will approximate-and-merge such branches, resulting in a SMAT that cannot perfectly reconstruct the input shape. However, SMAT can achieve 0.3% approximation error and an MS-SSIM score of up to 0.99 (Sec. 5.4). We argue this is sufficient for most applications such as shape matching, retrieval, and deformation, which do not require perfectly lossless encoding.

![Figure 5.14: The full medial axis of a rectangular shape (a) with zoomed-in detail (b) and color-coded segmented branches (c).](image)

5.7 Conclusion

In this chapter, we have presented SMAT, a method for encoding the medial axis transform (MAT) of raster (binary image) shapes with B-splines. For this, we simplify raster MATs using a salience metric, segment them into branches and branch-segments, and optimize the fit of a set of B-splines over the resulting segments to minimize the number of required control points while maximizing the spline-to-MAT fit. We evaluated SMAT on a collection of raster shapes of different types and complexities, using several quality metrics for image and shape comparison. Our results show that SMAT can achieve visually indistinguishable results to the original images, while reducing the space needed to store the MAT by one to two orders of magnitude. SMAT has only two simple-to-set parameters: the degree of MAT simplification and desired MAT approximation error. We showed how SMAT enables generating super-resolution images, can capture multiscale MATs, and allows shape ma-
nipation. SMAT is implemented on the GPU making its application real-time for images up to $100^2$ pixels.

In the next chapter, we extend SMAT beyond binary shapes, to encode grayscale and color images. We explore the potential of the vector representation of dense skeletons used for image compression by comparing it with newer variants of JPEG, i.e., JPEG 2000 and BPG.
In the previous chapter, we have presented a vector-based medial representation for raster binary images. This not only has the benefits of vector representation – built-in smoothness, compact storage – but also makes it directly applicable to any binary images, without the need to extract a piecewise-continuous contour with sampling guarantees. In this chapter, we extend this idea beyond binary shapes, to encode grayscale and color images. A comprehensive evaluation shows that our Spline-based Dense Medial Descriptors (SDMD) method achieves much higher compression ratios at similar or even better quality to the well-known JPEG technique. Evaluations also show that SDMD produces comparable results to JPEG 2000 and BPG. We illustrate our approach with applications in generating super-resolution images and salient feature preserving image compression.

6.1 Introduction

In Chapter 3, we presented Compressing Dense Medial Descriptors (CDMD), which encodes images represented as threshold sets in luminance space (Van Der Zwan et al., 2013) by using their medial axis transforms (MATs). CDMD’s main value was in showing that a grayscale or color image can be faithfully encoded by a set of per-layer MATs. However, CDMD’s storage costs are prohibitive, as one has to store \( L \) MATs, each represented as a set of pixels with 2D locations and \( DT \) values.

The issue of compactly encoding MATs has received attention in areas outside image representation, most notably for encoding MATs for binary shapes. In particular, representing MATs with splines was found to be good for data compression as storing spline control points is less costly than storing all MAT points. Yushkevich et al. (2003) first proposed to fit the MAT with cubic B-splines for statistical shape analysis. Zhu et al. improved this by automatically computing a compact spline representation of the MAT of a 2D binary shape (Zhu et al., 2014). However, this approach handles only vector shape representations, i.e., only works with the Voronoi-based MAT method of Attali and Montanvert (1997). In contrast, our SMAT method demonstrated in Chapter 5 used raster representations for \( T_i, S_T \), and \( DT_T \), fitting \( S_T \) and \( DT_T \) with B-splines. SMAT is directly applicable to any binary image \( T_i \) and also can use the computationally efficient methods for ex-
tracting the MAT (Cao et al., 2010). SMAT applies a least-squares algorithm (Eberly, 2014) to fit every MAT branch in the 3D space \( S_{T_i} \times DT_{T_i} \) with a B-spline. For a user-provided approximation error \( \gamma_0 \) between the MAT and the B-splines, SMAT finds the fitting scheme with the minimal number of B-spline control points required. Each control point \( c_j = (p_j, DT_{T_i}(p_j)) \in \mathbb{R}^3 \) consists of a 2D position \( p_j \) and its corresponding \( DT \) value. Hence, instead of storing all MAT pixels (as CDMD does), SMAT stores only a smaller set of control points. From these, SMAT rasterizes the B-splines using de Casteljau’s algorithm (Piegl and Tiller, 1997). Thus, the rasterized B-splines give a pixel-based representation of the MAT. From this representation, a layer \( \tilde{T}_i \) is reconstructed by the disc-union method described earlier in Sec. 2.1.4.

Summarizing the above: The CDMD method faithfully (but not compactly) represents a grayscale or color image using pixel-based MATs for several layers. The SMAT method compactly and faithfully encodes a MAT for a single layer using B-splines. In this chapter, we combine the two to faithfully and compactly represent a grayscale or color image. The contributions of our work are thus as follows:

- **Novelty**: Our method is, to our knowledge, the first approach to encode color images with B-spline-based MATs;
- **Generality**: SDMD can directly handle any raster image of any resolution;
- **Scalability**: End-to-end, our method can encode (and decode) megapixel images in a few seconds on a commodity PC featuring a modern graphics processing unit (GPU);
- **Evaluation**: We show that SDMD has good performance (compression ratio and quality) on a wide set of natural and synthetic color images of different sizes;
- **Applications**: We show that SDMD enables additional applications besides compression, such as generating super-resolution images and compression that preserves salient features.

### 6.2 SDMD Method

The proposed SDMD method (Fig. 6.1) combines the advantages of CDMD (Fig. 6.1, steps 1, 2, 5), which encodes color images with those of SMAT (Fig. 6.1, steps 3 and 4), which compactly encodes MATs for binary images using B-splines. Moreover, besides simply integrating CDMD and SMAT, we propose three improvements that increase compression and quality: adaptively encoding upper vs. lower threshold-sets (Sec. 6.2.1); separately treating chrominance and luminance (Sec. 6.2.2); and removing Y-structures from the skeletons (Sec. 6.2.3).
6.2.1 Adaptive layer encoding

By definition, MATs require as input a binary image (Equations (2.1) and (2.2)). Hence, to encode a grayscale image \( I \) this way, we need first to decompose it in a set of threshold sets \( T_i \). For this, CDMD proposed upper thresholding \( T_i = \{ x \in I \mid I(x) \geq \theta \} \), \( 0 \leq i < n \). Applying this thresholding for all selected layers \( L \) is however not good for compression. Figure 6.2 (a) explores this by showing a cushion treemap image—a well-known visualization for hierarchical information (Van Wijk and Van de Wetering, 1999). Figure 6.2 (b) shows one of its upper threshold sets \( T_i \) (for \( i = 83 \)). Here, the black area is the region \( T_i \) to be skeletonized. If we do so, the obtained skeleton \( S_{T_i} \) is quite complicated (Fig. 6.2 (c)), meaning, it requires many B-spline control points to store via SMAT. However, if we chose instead to encode the white areas (regions in \( I \) outside \( T_i \)) from Fig. 6.2 (b), i.e., if we use a lower thresholding for this layer \( T_i = \{ x \in I \mid I(x) \leq \theta \} \), the resulting skeletons \( S_{T_i} \) will be significantly simpler, see Fig. 6.2 (d), leading to fewer B-spline control points needed to encode them. Hence, instead of using upper thresholding for all selected layers, we adaptively encode upper or lower threshold-sets as follows. Let \( T^\uparrow_i = \{ x \in \mathbb{R}^2 \mid I(x) \geq \theta \} \), \( 0 \leq i < n \) and \( T^\downarrow_i = \{ x \in \mathbb{R}^2 \mid I(x) \leq \theta \} \), \( 0 \leq i < n \) be the upper and lower thresholding operation (Silva et al., 2016), respectively. We choose between the two to compute \( T_i \) by greedy optimization, i.e.,

\[
T_i = \begin{cases} 
T^\downarrow_i, & \text{if } N_{T^\downarrow_i} < N_{T^\uparrow_i} \\
T^\uparrow_i, & \text{otherwise} 
\end{cases}
\]  

(6.1)
where \( N_{T_i^\downarrow} \) is the number of B-spline control points needed to encode \( T_i^\downarrow \), and similarly, \( N_{T_i^\uparrow} \) for \( T_i^\uparrow \). Simply put: We evaluate how expensive it is to encode \( T_i^\downarrow \) vs. \( T_i^\uparrow \) for each selected layer and choose the cheaper encoding of the two.

![Figure 6.2: Adaptively encoding upper vs. lower threshold-sets. (a) Input image. (b) One threshold-set of the luminance channel in (a) for intensity \( i = 83 \). (c) The generated medial axis of \( T_{83}^\downarrow \). (d) The generated medial axis of \( T_{83}^\uparrow \), which is far less complex than, and thus preferred to, \( T_{83}^\uparrow \).](image)

Adaptively encoding upper vs. lower threshold-sets is a simple idea, but it can greatly improve the compression rate. Furthermore, it can even get better quality. Figure 6.3 shows an example. For a heart anatomy image (400×460 pixels, Fig. 6.3 (a)), images (b) and (c) are reconstructions by the CDMD method (using only upper threshold sets) and our new adaptive scheme, respectively. Compared with (c), (b) misses several thin curve structures in the image, which are marked with red arrows in (a). This can be explained on one of the selected layers \( T_{90} \) (d). As said, CDMD only uses upper threshold-sets \( (T_{90}^\uparrow) \) in which the shape to be encoded corresponds to the white areas in (d). Since the curves marked by the red arrows are one or two pixels thick, CDMD fails to generate skeletons for these curves (see image (e)), resulting in these curves missing in the reconstruction (b). In contrast, our adaptive method considers both \( T_{90}^\uparrow \) and \( T_{90}^\downarrow \). Since \( N_{T_{90}^\downarrow} \) obtained by (f) is less than \( N_{T_{90}^\uparrow} \), SDMD encodes \( T_{90}^\downarrow \), i.e., the black areas in (d). Since those
black areas are thick enough to generate accurate skeletons, the white curves are preserved.

Figure 6.3: A comparison between reconstructions using the CDMD method (b) and our adaptive scheme (c) for a heart anatomy image (a). (d) The threshold-set (layer) 90 of (a). (e) The skeleton generated for $T_{90}^\uparrow$. (f) The skeleton generated for $T_{90}^\downarrow$.

6.2.2 Per-channel encoding

Figure 6.1 shows the SDMD method only for a grayscale image. For color images, the CDMD method uses the RGB color space, handling each of the channels independently, as in Fig. 6.1. This has a high redundancy, preventing high compression. Later, saliency-based DMD (Chapter 4) improved this by using the YCbCr color space given that YCbCr can give better subjective quality than RGB due to its perceptual similarities to human vision (Podpora, 2009; Podpora et al., 2014). However, this method used the same compression parameters for the three channels. We further take advantage of the human visual system’s lower acuity for chromatic differences (Cb and Cr components) than for achromatic difference (Y component) (Nobuhara and Hirota, 2004; Lambrecht, 2001) to treat the three channels separately. We select fewer layers $L$ for the two chrominance components (Cb and Cr) than for the luminance one (Y), and also compress Cb and Cr more than Y, using larger $\epsilon, \sigma_0$, and/or $\gamma_0$. Concretely, given a user-selected parameter set $(L, \epsilon, \sigma_0, \gamma_0)$ for the Y-channel, we use the set $(n_1L, n_2\epsilon, n_3\sigma_0, n_4\gamma_0)$ for the Cb and Cr channels. To find good values for $n_1, \ldots, n_4$, we fix three
of the four coefficients to the value of one, in turn, and vary the fourth coefficient over its allowable range and evaluate the result. This led to $n_1 = 0.5, n_2 = 5, n_3 = 2, n_4 = 1$ as good values for producing high-quality results (see Fig. 6.4).

Figure 6.4: SDMD computation framework for color images. The red parameters indicate SDMD treats chrominance (Cb and Cr components) and luminance (Y component) separately.

Treating the three channels separately to compress chromatic components more than the luminance one allows greater overall compression without a significant effect on image quality. Figure 6.5 shows this by a sample image, in which (b) and (c) are the results when setting the same and different parameters for different channels, respectively. In this figure, SSIM is short for the Structural SIMilarity index (Wang et al., 2004), which measures how perceptually close $\tilde{I}$ is to $I$, where 1 indicates the two input images are identical, while 0 means the two are completely different, see Sec. 2.3. We see that using different parameters for different channels (Fig. 6.5 (c)) takes up about 30% less storage, while yielding results that are almost identical to (b) both visually and SSIM-wise.

6.2.3 Boundary Y-Structure elimination

In contrast to Figures 6.1 and 6.3, real-world images do not always have a background that fully surrounds the foreground image structures. Hence, their threshold-sets to be skeletonized will yield Y-like skeleton branches when the foreground structures reach the image boundary. Figure 6.6 shows this where the black (foreground) spirals reach the image boundary. The image is on purpose simple, for illustration aims. Encoding these Y-branches costs additional B-spline control points, thus lowering the compression rate. To get more compact skeletons, we
present a Y-branch removal scheme, detailed next in Fig. 6.7. For any layer $T_i$, let $A_jB_j$ be the boundary segments corresponding to $T_i$ (black in Figures 6.6 and 6.7) that touch the image boundary; see Fig. 6.7 (b). Each such segment causes a Y-structure in the skeleton. To remove these Y-structures, we extend the size of the binary image $I_i$ to be skeletonized by adding a semi-disc to each boundary segment, centered at $(A_j + B_j)/2$ and of radius $\|A_j - B_j\|/2$ (see Algorithm 3).

Due to this extension (Fig. 6.7 (b)), the computed skeletons (and their corresponding MATs) will reach out beyond the borders of the input image (Fig. 6.7 (c)). We next clip these MATs by the input image and fit the remaining structure with B-splines using SMAT. This gets rid of the unwanted Y-structures (see Fig. 6.7 (d)). When reconstructing these B-spline representations, we need to prolong the MATs where they touch the image border. If not, sharp corners, i.e., acute angles between the image boundary and the semi-disc diameter will be rounded-off, as shown by the red dashed curves in Fig. 6.7 (f). This can be explained by the medial circle of an endpoint in Fig. 6.7 (d). A bit larger medial circle of an extended point in Fig. 6.7 (e), however, can totally cover the sharp corner. We extend the MATs by linear interpolation and then stop when
Figure 6.7: The pipeline of Y-structures elimination. (a) Layer $T_1^{128}$ of the image in Fig. 6.6. (b) Semi-disc extension of (a) with diameters in red. (c) The contour of (b) (in red) and the generated MATs (in black). (d) The cropped MATs of (c). A medial circle (in red) of an endpoint of the skeletons is also drawn in (d). (e) Extending MATs that touch the image boundary. A medial circle (in red) of an extended point is also shown in (e). (f) Reconstructed layer $\tilde{T}_1^{128}$. Red dashed curves show the sharp corner results when directly using the cropped MATs in (d). (g) A schematic diagram of the extension (black line segment) of one border point of the 3D MAT curves. The 45-degree outer plane of the spiral image and the generated medial circle (black dotted line) of the extended endpoint are also shown in (g).

the 3D MAT curves just reach the 45-degree outer planes of the image as this is where the generated medial circle is tangent to one of the image boundary edges, i.e., the medial circle is just about to leave the image, as shown in Fig. 6.7 (g).

Algorithm 3: Semi-disc extension algorithm

Input: Threshold-set $T_i$
Output: Extended $T_i$ to be skeletonized
1. Scan the pixel border of $T_i$ to detect the boundary segments $A_jB_j$.
2. Enlarge $T_i$ by a band of thickness $\max_j \|A_j - B_j\|/2$ in all four directions.
3. Draw a semi-disc atop each segment $A_jB_j$ with diameter $\|A_j - B_j\|$ and centered at $(A_j + B_j)/2$. 

88
Figure 6.8 compares SDMD with (a2–e2) and without (a1–e1) Y-structure removal on five images that all have objects touching the image border. Additional examples also using more parameter settings are available in the supplementary material (Wang et al., 2021e). For each image, we also list its SSIM value (similarity to the uncompressed original, see Sec. 6.2.2) and its compression ratio CR, defined as the size of the original image divided by the size of the SDMD encoding (for details, see Sec. 6.3.1). Green and red numbers in Fig. 6.8 indicate better and worse, respectively, SSIM and CR values for the Y-removal scheme as compared to using plain SDMD. From these values, we see that the Y-removal scheme increases CR by a percentage ranging between 3.5% and 31.4% for the 5 images considered, with negligible quality loss (around 0.002 SSIM decrease). The CR gain depends on how many objects in the image touch its borders. For instance, image (d) has only one object—the right green sphere—touching a small part of the border, so the Y-removal scheme boosts CR by only 3.5%. In contrast, image (b) has 8 color bands touching the image border along its entire extent, so the Y-removal scheme boosts CR by 31.4%.

Figure 6.8: A comparison of the SDMD method for five images without Y-structures removal (a1–e1) and with this scheme used (a2–e2). For each image, we show the SSIM quality and compression ratio CR. We also indicate what we lost (in red) and what we gained (in green) in CR and SSIM when using the Y-removal design.
6.3 Results

We comprehensively evaluate the proposed SDMD method from various angles, as follows.

- First, we build an evaluation benchmark (Sec. 6.3.1);
- We study how SDMD depends on its free parameters (Sec. 6.3.2);
- We quantitatively assess the adaptive layer and per-channel encoding extensions proposed earlier (Sec. 6.3.3);
- We compare our method with the original CDMD method, the well-known JPEG technique, and its newer variants JPEG 2000 and BPG (Sec. 6.3.4);
- Finally, we show how SDMD performs on images of different resolutions (Sec. 6.3.5).

6.3.1 Benchmark

The SDMD encoding consists of a tuple \((w, h, \{l_i\})\), i.e., the width \(w\) and height \(h\), in pixels, of the input image \(I\), and the \(L\) selected layers \(I_i\). Each layer \(I_i = (i, f, \{b^k_i\})\) encodes the layer number or intensity value \(i\), a flag bit \(f\) that tells whether this layer needs to be flipped or not (see Sec. 6.2.1) and a set of B-splines \(\{b^k_i\}\) encoding the layer’s MAT. Each B-spline \(b^k_i = (d^k_i, \{c_j\})\) consists of a degree \(d^k_i \in \mathbb{N}\) and a set of control points \(c_j \in \mathbb{R}^3\) (see Sec. 6.1).

SDMD is evaluated based on two factors: Quality \(Q\) of the reconstruction \(\tilde{I}\) of the input image \(I\), which is measured by the SSIM difference of the two images (see Sec. 2.3) and the Compression ratio \(CR\), defined as \(CR = |I|/|SDMD(\tilde{I})|\), i.e., the byte-size of the original image \(I\) divided by the byte-size of the SDMD encoding of \(\tilde{I}\), which has been described above.

To evaluate the SDMD method comprehensively, we need to create a benchmark involving multiple image types. Indeed, as earlier work using CDMD to represent images has shown (Chapter 3, 4), MAT-based image representations work best for images consisting of relatively large shapes overlaid on a smooth background. This is not surprising given that MATs were also originally found to be most effective for the analysis (and representation) of shapes (Siddiqi and Pizer, 2008). As such, we also target our method to represent color imagery of a similar type. We found several classes of imagery that fall within this typology, namely scientific visualizations of continuous data (scalar and vector fields), medical images (from, e.g., CT, X-ray, and MRI scans), synthesized images using graphics rendering and vectorization methods (Orzan et al., 2013), graphics art (logos, graphics design), and cartoon images. High-quality, low-size representations of these image types are needed for
many applications such as remote browsing of specialized content (SciVis, medical) or general-purpose content (webpages) when, e.g., using low-speed connections. For our evaluation, we consider a benchmark with the above-mentioned five image types, each type having at least 10 images. Table 8 shows a summary of the benchmark.

Table 8: The benchmark of five image types (available at Wang et al. (2021e)) used throughout this work for evaluating SDMD.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SciVis data</td>
<td>Scientific visualizations (scalar and vector fields)</td>
<td>15</td>
</tr>
<tr>
<td>Medical images</td>
<td>Images generated by CT, X-ray and MRI scans</td>
<td>10</td>
</tr>
<tr>
<td>Computer graphics</td>
<td>Images generated by rendering and vectorization</td>
<td>10</td>
</tr>
<tr>
<td>Graphics art images</td>
<td>Simple shapes such as clip art, logos, and graphics design</td>
<td>20</td>
</tr>
<tr>
<td>Cartoon images</td>
<td>Animated cartoons and comic strips</td>
<td>10</td>
</tr>
</tbody>
</table>

6.3.2 Parameter effect

As Fig. 6.1 shows, SDMD depends on four parameters: the number of selected layers \( L \), the size of removed islands \( \varepsilon \), the saliency threshold \( \sigma_0 \), and the spline fitting tolerance \( \gamma_0 \). To find a good trade-off between \( Q \) and \( CR \), we fix, in turn, three of the four free parameters \( L, \varepsilon, \sigma_0, \) and \( \gamma_0 \) to empirically-determined values and vary the fourth parameter over its allowable range via uniform sampling. This method is also applied in Chapter 3 and Chapter 4 and is much simpler and faster than the usual hyper-parameter grid-search used, e.g., in machine learning (Bergstra and Bengio, 2012).

Figure 6.9 shows the results of this parameter search for five images, one of each type in the benchmark. The actual images are shown on the top. The subsequent four plots (b1–b4) show how \( Q \) and \( CR \) are related when varying each of the \( L, \varepsilon, \sigma_0, \) and \( \gamma_0 \) parameters, while keeping the other three fixed to their default values. The colored line plots indicate the \( Q \) vs. \( CR \) graphs for each image, with dot sizes along these lines indicating the varying parameter’s values (see the legends). Overall, the plots in Fig. 6.9 (b1–b4) show a negative correlation between \( CR \) and \( Q \) for all images and parameter variation experiments, which is expected. Indeed, higher quality leads to a lower compression ratio.

Figure 6.9 (b1) shows the trade-off between \( Q \) and \( CR \) as a function of the number of layers \( L \). We sample \( L \) from 10 to 45 with a step of
Figure 6.9: Trade-off between \( Q \) (SSIM) and CR for five image types (a1–a5) as a function of parameters \( L \) (b1), \( \epsilon \) (b2), \( \sigma_0 \) (b3), and \( y_0 \) (b4), respectively. The box colors in (a1–a5) corresponds to the five colors of scatterplots in (b1–b4).

5, following observations in Chapter 3 stating that \( Q \) and CR hardly change for \( L > 40 \). This is also visible from (b1): when the number of layers \( L \) increases to around 40, the points along a line almost overlap. This is most salient for the blue curve (graphics art images), where \( Q \) and CR do not change at all when \( L > 10 \). In addition to \( L \), the other three parameters are set to \( \epsilon = 0.01 \), \( \sigma_0 = 1.0 \), and \( y_0 = 0.002 \). Chart (b1) also shows that except graphics art images (a1), the other four curves have a ‘tail’ pointing downward, indicating a notable drop of SSIM for low \( L \) values for a minimal increase in CR. As such, we deem that a value \( L \) of 10 to 15 for graphics art images and 15 to 20 for the other four types are good preset values. Fig. 6.9 (b2–b4) show quite similar trends for \( \sigma_0 \), \( y_0 \), and \( \epsilon \), as discussed above for \( L \). Lower parameter values yield lower CR and higher SSIM, and conversely.

Given all the above, we settle to the preset values (or ranges) \( L \in [10, 20] \), \( \epsilon = 0.01 \), \( \sigma_0 \in [0.6, 1.4] \) and \( y_0 = 0.002 \) that give a good SSIM vs. CR tradeoff. We next use SDMD with parameters in these ranges to
evaluate the method on more images and also compare its results with other compression methods.

6.3.3 Quantitative evaluation of adaptive layer and per-channel encoding

Section 6.2 details three improvements to the original CDMD method: adaptively encoding upper or lower level-sets, separately treating chrominance and achrominance channels, and eliminating Y-terminations in the MAT. We have discussed the added value of Y-termination removal already in Sec. 6.2.3, showing that it produces a significant CR boost for basically no SSIM decrease. As such, we next focus on the evaluation of the adaptive layer and per-channel encoding schemes.

Figure 6.10 shows the average SSIM vs. CR for our five image types for three SDMD schemes, i.e., the basic SDMD method (blue dots), SDMD with adaptive layer encoding on (red dots), and SDMD with both adaptive layer encoding and per-channel encoding on (green dots). Each of the five charts corresponds to one image type. Each polyline in a chart corresponds to a different parameter setting, as indicated in the legend, following the parameter-setting discussion in Sec. 6.3.2. Finally, each colored dot in a polyline corresponds to one of the three SDMD schemes mentioned above. For graphics art images, as Fig. 6.9 showed, fewer layers \( L \) and slightly larger island thresholds \( \varepsilon \) can produce good results, so we chose for these the parameter combination \( L \in \{10, 15\}, \varepsilon = 0.03, \sigma_0 \in \{0.6, 1.4\}, \gamma_0 = 0.002 \). For all the other image types, more layers \( L \) and a slightly smaller \( \varepsilon \) are used, as indicated in the legend in Fig. 6.10.

For ease of reading, Table 9 aggregates the results detailed in Fig. 6.10, showing the loss (\( \downarrow \)) and gain (\( \uparrow \)) in SSIM and CR, respectively, when using the adaptive layer encoding (ALE) and ALE plus the per-channel encoding (PCE). From this table and Fig. 6.10, we see that the quality

<table>
<thead>
<tr>
<th>Type</th>
<th>SDMD + ALE</th>
<th>SDMD + ALE + PCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a1) Graphics art images</td>
<td>0.0002(\downarrow) / 29%(\uparrow)</td>
<td>0.0010(\downarrow) / 74%(\uparrow)</td>
</tr>
<tr>
<td>(a2) Cartoon images</td>
<td>0.0006(\downarrow) / 38%(\uparrow)</td>
<td>0.0007(\downarrow) / 128%(\uparrow)</td>
</tr>
<tr>
<td>(a3) Computer graphics</td>
<td>0.0015(\downarrow) / 7%(\uparrow)</td>
<td>0.0019(\downarrow) / 45%(\uparrow)</td>
</tr>
<tr>
<td>(a4) Medical images</td>
<td>0.0025(\downarrow) / 7%(\uparrow)</td>
<td>0.0032(\downarrow) / 18%(\uparrow)</td>
</tr>
<tr>
<td>(a5) SciVis data</td>
<td>0.0024(\downarrow) / 9%(\uparrow)</td>
<td>0.0032(\downarrow) / 79%(\uparrow)</td>
</tr>
</tbody>
</table>
Figure 6.10: The average SSIM vs. CR for basic SDMD (blue dots), SDMD with adaptive layer encoding (ALE, red dots) and SDMD with both an adaptive layer and per-channel encoding (ALE + PCE, green dots) for graphics art images (a1), cartoon images (a2), computer graphics (a3), medical images (a4), and SciVis images (a5). (b) Summarization of the first five plots, with colors indicating image types.

loss is very little (from 0.0002 to 0.0032) for all image types, regardless of whether we use only ALE or both ALE and PCE. We also see that, for medical imagery, the gain in CR of both ALE and PCE is the smallest, 18% on average. This is mainly because most such images are grayscale. Hence, the effect of per-channel encoding (PCE) is almost zero. In contrast, for cartoon images, ALE + PCE yields an increase of CR of 128%, that is, the two enhancements more than double the compression ratio as compared to plain SDMD. This can be explained by the fact that most cartoon characters have a thin black outline. When lower thresholding such images, we obtain threshold sets that have very thin components, similar to the one shown in Fig. 6.2 (b). Hence, as in that example, ALE
will greatly simplify the MATs to be encoded for cartoon images, yielding higher CR values.

Finally, Fig. 6.10 (b) shows all the results from the previous five charts in the same figure but in a single image. Note that the ranges of both the CR and SSIM axes of all charts are different, chosen so that we ‘zoom in’ in each case on the range in which the actual data varies. Fig. 6.10 (b) lets us compare how SDMD (with the ALE and PCE adaptations) performs across different image types. We see here, in more detail than Table 9, that SDMD works particularly well for graphics art images (red dots). We also see that the ALE and PCE adaptations yield only small CR gains for the medical images (black dots are distributed along almost vertical lines). For the cartoon images, scientific visualizations, and computer graphics images, the two adaptations perform in-between medical images and graphics arts images, that is, increase CR for a limited SSIM decrease. Given these results, we conclude that both adaptations are of added value, as they create a negligible SSIM loss for a significant increase in CR for all image types and all parameter combinations.

6.3.4 **Comparison with CDMD and JPEG**

In this section, we compare the improved SDMD method—using the adaptive layer and per-channel encoding which showed added-value in the evaluation in Sec. 6.3.3—with the CDMD method and JPEG for all our benchmark images.

6.3.4.1 **Comparison with the original CDMD method**

Figure 6.11 compares the results of SDMD (red dots) and the original CDMD method (blue dots) under the same parameter settings. Each plot in the figure represents images of a different type. Similar to SDMD, we define compression ratio (CR) as $CR = \frac{|I|}{|MAT(\tilde{I})|}$, where $MAT$ is the size (in bytes) needed to store $S_{T_i}$ with the delta-encoding scheme proposed by CDMD, rather than the B-spline scheme used by SDMD. The large dots in the plot show the CR and SSIM averages over all the benchmark images for one parameter setting. Hence, different large dots correspond to different parameter settings. To show more details, we also display a star plot for one of the parameter settings, i.e., connect the large dot (average over all images) with small dots that indicate the CR and SSIM values for every individual image. Hence, small stars indicate little deviation in CR and SSIM from the average over the image benchmark; large stars indicate more variability of these metrics as a function of the actual image.

Figure 6.11 shows that the star plot shapes of CDMD and SDMD are quite similar. In other words, CDMD and SDMD exhibit a similar dependency on the image type. This is due to the fact that SDMD inherits the thresholding and skeletonization method of CDMD. More importantly,
the points plotted for SDMD (red) are always at the bottom right of those of CDMD. That is, SDMD always gets a significantly higher CR for only a small decrease in quality. Quantitatively, on average, compared with CDMD, SDMD reduces SSIM by 0.003 (art graphics images (a)), 0.008 (cartoon images (b)), 0.008 (computer graphics images (c)), 0.01 (medical images (d)), and 0.008 (SciVis images (e)). On average, compared to CDMD, SDMD increases compression by a factor of 3.4 (a), 3.7 (b), 3.2 (c), 2.5 (d), and 3.8 (e), which we deem to be a very substantial improvement.

Figure 6.11: Comparison of CDMD (blue dots), JPEG (green dots), and SDMD (red dots) for graphics art images (a), cartoon images (b), computer graphics (c), medical images (d), and SciVis data (e). The actual image data (smaller dots) are connected to the corresponding average value (larger dots) for one parameter setting of each method.
6.3.4.2 Comparison with JPEG

Figure 6.11 also allows comparing SDMD with JPEG, the latter run under five quality settings, i.e., 10%, 30%, 50%, 70%, and 90%. For each such quality setting, we plot the average CR and SSIM of JPEG as a single green dot in each chart in Fig. 6.11. Green dots are sorted right-to-left by increasing quality setting values—that is, the higher the JPEG quality setting, the lower the obtained CR. If we compare SDMD with JPEG, we see that SDMD cannot reach the same SSIM values as when JPEG uses its 90% quality setting—the topmost green dots in each plot are above the topmost red dots. However, the difference in quality (SSIM) is quite small, if we look at the vertical spread of the green vs. red dots—about 2% on average. Separately, we see that SDMD always gets higher compression rates than JPEG for all situations—red dots are always (significantly) to the right of the green dots. We also see that the green dots are spread far more along the vertical (SSIM) axis than the horizontal one, indicating that JPEG’s quality setting can influence SSIM far more than CR. In contrast, the red dots are spread far more along the horizontal (CR) axis than the vertical ones, indicating that SDMD’s settings can influence compression significantly for only a small drop of quality. In particular, if we are after strong compression, SDMD performs better than JPEG: Compared to JPEG with a quality of 10% (the rightmost green dot in each plot), SDMD always gets both higher CR and better quality, except for the cartoon images. When fewer layers and larger saliency thresholds are used (rightmost red point in each plot), SDMD not only gets better quality but yields a compression that is 12 (a), 3.3 (c), 2.5 (d), and 2.9 (e) times higher than that of JPEG with a quality setting of 10%.

Figures 6.12 and 6.13 further refine the above insights by showing 20 images, spanning the five types of our benchmark (Table 8), compressed by JPEG (with a quality of 10%) and with SDMD. The results for the entire benchmark are available in the supplementary material (Wang et al., 2021e). From the zoomed-in areas of specific blocks on the right, we observe that JPEG with a low-quality setting generates obvious artifacts such as checkerboarding (Fig. 6.12 (b1),(b7)), banding (Fig. 6.13 (b2–b4), (b8) and (b13)), background color changing (Fig. 6.13 (b2), (b9)), and object details missing (Fig. 6.13 (b9)). In contrast, SDMD yields better quality (SSIM) and does not exhibit the aforementioned artifacts, leading to images which, we argue, are almost indistinguishable from the originals. Separately, SDMD also achieves much higher compression rates than JPEG, especially for the scientific visualization (Fig. 6.13 (c1),(c2)), vector graphics (Fig. 6.13 (c4)), abstract shapes (Fig. 6.13 (c7)), and illustration (Fig. 6.13 (c9)) image types. The good performance of SDMD on medical images (Fig. 6.12 (c1–c5)) suggests that SDMD could be very well suited in the context of remote/online viewing of medical image databases.
Figure 6.12: Left pane: Comparison of JPEG for quality set to 10% (b1–b7) with SDMD (c1–c7) for 7 input images (a1–a7) of medical type. For each result, we show the SSIM quality Q and the compression ratio CR.

Right pane: Zoomed-in areas, marked in green in the images in the left pane, show subtle differences between the original, JPEG, and SDMD.

6.3.4.3 Additional comparisons

As stated in Sec. 2.2, tens of image compression methods exist. We did not perform an evaluation against these since, as already outlined in Sec. 6.1, our main research question was to explore the potential of spline-based MATs as an alternative tool to image representation, which includes image compression applications (Sec. 6.3) but also other applications such as super-resolution images generation (Sec. 6.4.1) and salient feature-preserving simplification (Sec. 6.4.2). Therefore, for image compression, we only evaluated SDMD in Sec. 6.3 against the arguably most widely used compression method, i.e., JPEG. Given the positive results outlined by the comparison with plain JPEG, next,
we explored how SDMD compares with newer variants proposed in the literature as replacements for JPEG that increase compression ratios while preserving image quality, i.e., BPG (Bellard, 2014) and JPEG 2000 (Taubman and Marcellin, 2001).

Figure 6.14 compares SDMD, BPG, and JPEG 2000 for five images, one of each type in our benchmark. The actual images are shown in Fig. 6.15. For each image, we run SDMD (solid line) under four parameter settings, JPEG 2000 (dotted line) under five compression settings, and BPG (star markers) under its default setting. As visible, for the graphics art image (blue dots), SDMD produces both higher quality and compression than JPEG 2000. For the other four types, JPEG 2000 generates better quality and/or higher CR than ours. For all these five types, BPG generates
higher SSIM than SDMD when the CRs of the two are similar. However, for all above cases, the differences, both in Q and CR, are quite small.

![Graph showing comparison of SDMD, JPEG 2000, and BPG for different image types.](image)

Figure 6.14: A comparison of SDMD (dots), JPEG 2000 (squares), and BPG (stars) for a graphics art image (blue), a cartoon image (red), a computer graphic (yellow), a medical image (cyan), and a SciVis image (green).

Figure 6.15 refines this insight by showing the reconstruction results under the quality setting indicated in Fig. 6.14 with the dashed box. Overall, the three methods perform visually very similarly, as already indicated by the similar SSIM values in Fig. 6.14. The zoomed-in areas show a few subtle differences: For strong-contrast images, such as the first two in Fig. 6.15, JPEG 2000 tends to create some small-scale blur artifacts. This is also seen in the fact that, for the first image in Fig. 6.15, SDMD yields both higher SSIM and CR than JPEG 2000. Compared to BPG, SDMD’s results are very similar. For the third image, which exhibits a smooth luminance gradient in the shadow area, SDMD captures this gradient quite well. In contrast, JPEG 2000 and BPG cause a slight amount of blocking artifacts. For the fourth image, JPEG 2000 and BPG create a small amount of blocking and false colors (purple) in the near-constant-luminance, dark blue, area. In contrast, SDMD does not have such problems but suffers from loss of small-scale, faint, details—due to its selection of threshold-sets to be encoded (Sec. 6.3.2). Finally, for the fifth image, all methods produce basically visually identical results.

Summarizing the above observations, we conclude that SDMD can create images that are visually very similar to those produced by modern variants of JPEG, with a slight loss in quality and compression ratio.
6.3.5 SDMD performance on images of different resolutions

All images in our benchmark have quite high resolutions (2000^2 to 3000^2 pixels). We next test how SDMD performs on images of different resolutions. For this, we start with a high-resolution image and generate m downscaled images from it using ImageMagick (Thyssen, 2012). Next, we run SDMD on the total m + 1 images and study how SSIM and CR vary as a function of the image size.

Figure 6.16 shows this analysis for two graphics art images of m = 8 different resolutions each, from 320 × 200 to 2560 × 1600 pixels. For additional insights, we also compare SDMD with CDMD and JPEG on these images. The charts show the CR vs. SSIM plots as we vary the image resolution. That is, for a given method, we plot a polyline of m = 8 points, indicating the respective CR and SSIM values for all the resolutions. We also show the actual images for the lowest and highest, respectively, resolutions for both SDMD and JPEG.

Several insights can be obtained as follows. First, we see that SDMD dominates CDMD in CR values, with no quality loss whatsoever (green background image) and a minimal quality loss of about 2% (spiral shape...
Figure 6.16: A comparison of SSIM vs. CR for the SDMD (red), CDMD (blue), and JPEG (green) methods for two graphics art images of 8 different resolutions each, from 320 × 200 to 2560 × 1600 pixels. Note that the image sizes shown in the figure are not proportional to their actual sizes, for space reasons.

image). We also see that quality increases with input image size. For example, for the spiral shape image at the lowest resolution (320 × 200 pixels), both JPEG and SDMD yield a quite low quality, with SDMD being about 5% better than JPEG. The loss of quality is also visible in the actual image snapshots (shown on the left of the chart) that exhibit fuzzy
effects. However, the reasons for fuzziness are different: For JPEG, this is caused by blocking artifacts; for SDMD, the fuzziness is caused by the inaccurate reconstruction of threshold-sets due to the spline fitting error. Still, the SDMD reconstruction looks overall smoother and sharper, as also reflected by its higher SSIM score. For the largest resolution image (2500 × 1600 pixels), both JPEG and SDMD produce visually good reconstructions and have similar (high) SSIM scores. However, SDMD compresses about 16 times more than JPEG. Interestingly, for the second image example (green background image), SDMD produces a quite smooth reconstruction both at the minimal and maximal resolution. In contrast, JPEG shows a pixelated reconstruction for the lowest resolution and strong banding artifacts for the highest resolution. Here, again, SDMD compresses better than JPEG: about 4 times more for the highest resolution.

From Fig. 6.16, we also observe that $S_{SDMD} > S_{CDMD} > S_{JPEG}$, where $S_{SDMD}$ indicates the slope of the curve of SDMD, and similar for CDMD and JPEG. This means that as the input image becomes larger, the compression rate of the SDMD method increases the fastest, followed by CDMD, and finally by JPEG. The reason for this is determined by the compression principle of the three methods. JPEG compresses images by splitting them into $8 \times 8$ blocks; CDMD captures shapes in the image using MATs; and SDMD further encodes skeletons with B-splines. Intuitively, we can say that the ‘compression unit’ is two-dimensional (block) for JPEG, one-dimensional (skeleton branch pixel-chain) for CDMD, and zero-dimensional (B-spline control point) for SDMD, respectively. Figure 6.17 further illustrates this by showing (a) one of the threshold-sets for the spiral image in Fig. 6.16 and (b) its corresponding spline representation. As already explained, SDMD only stores the locations of the control points shown in Fig. 6.17 (b). Hence, if we uniformly scale the input image by any arbitrary factor, the control points will stay the same in terms of relative positions and number, or only change very little due to small-scale sampling issues related to the fixed pixel-grid. Hence, SDMD will compress a larger version of the spiral image as efficiently as a smaller version.

![Figure 6.17: One of the level-sets for the geometric image in Fig. 6.16 (a) and its corresponding B-spline MAT representation (b).](image)
6.4 APPLICATIONS

Besides image compression, SDMD provides ways to create super-resolution images and selective encoding of salient features, as discussed next in Sections 6.4.1 and 6.4.2, respectively.

6.4.1 Super-resolution images

Image super-resolution (SR) is a popular technique for constructing higher-resolution images from low-resolution ones. In Sec. 5.5.1, we have illustrated how our vector-based skeletons can be used to create super-resolution versions of binary images. In this section, we extend the SR task for color and grayscale images. Following the same methodology in Sec. 5.5.1, to perform SR, SDMD directly rasterizes the reconstructed splines at the desired target resolution during step 4 in Fig. 6.1. As this occurs during reconstruction, generating the SR result does not incur any extra storage. Figure 6.18 (a2, b2) show a text image and a graphic generated by gradient meshes, both at a relatively low 500×500 pixels resolution. Images (a4) and (b4) show the SDMD reconstructions of these two images at a six-times higher resolution, i.e., 3000 × 3000 pixels. Any other target resolution can be used directly given a computed SDMD encoding of an image. As seen from the enlarged areas in Fig. 6.18, the SR reconstruction improves the discretization artifacts of the original images while keeping the reconstructed boundary clear and smooth.

![Figure 6.18: The super-resolution effect on a text image (a) and a graphic generated by gradient meshes (b). (a1, b1) Enlarged areas of the input images (500^2 pixels) (a2, b2). (a3, b3) Enlarged areas of SDMD reconstructions (3000^2 pixels) (a4, b4).](image-url)
We next intend to test our method with more images of various resolutions and types, and compare with other related state-of-the-art methods, as follows. Recently, AI researchers have used powerful deep learning algorithms for SR tasks and achieved high quality (Dong et al., 2014; Wang et al., 2019; Shi et al., 2016b; Ledig et al., 2017; Wang et al., 2021h). Among them, the recent Real-ESRGAN (Wang et al., 2021h) used a high-order degradation process and employed sinc filters to model common ringing and overshoot artifacts, achieving better visual performance than previous works in the same area, e.g., Dong et al. (2014); Wang et al. (2019); Shi et al. (2016b); Ledig et al. (2017). Given this, we next compare our method with Real-ESRGAN (Wang et al., 2021h). We test the two methods on a 100-image database, which spans 8 categories, from the Tinify dataset (Tinify, 2019) while aiming to enlarge them 8 times. The results are shown in Fig. 6.19. Every larger dot in the plot show the PSNR and MS-SSIM (see Sec. 2.3) average over all images of a given type. Hence, different large dots correspond to different types, indicated by different colors. Each large dot is connected with small dots, in a star plot metaphor, to indicate the PSNR and MS-SSIM values for every individual image in that type.

Figure 6.19 shows that, overall, Real-ESRGAN performs slightly better than the SDMD with an average PSNR of 20.79 vs. 18.04, and an average MS-SSIM of 0.83 compared to 0.73. Apart from that, we observe that the distribution of star plots of Real-ESRGAN and SDMD is quite similar. They both yield higher quality scores (whether PSNR or MS-SSIM) on the clothing and car parts while struggling to deal with the patterns of stickers and interior images. Yet, the internal reason is related to the size of each type as each category has the same size and size varies greatly between types. Figure 6.20 illustrates this insight by showing the results of three images taken from three categories: football cards, clothing, and car parts. As visible, when the input size is very small

![Figure 6.19: Comparison of the Real-ESRGAN method (a) and our SDMD (b) for 8 types of images. The actual image data (smaller dots) are connected to the corresponding average value (larger dots) for each type of each method.](image-url)
(a1), both Real-ESRGAN and SDMD generate poor visual effects. When the input image is large enough (b1, c1), the two methods yield perceptually good results. We also observe that when the input image has a very small size, the Real-ESRGAN method produces an abnormal and weird result (Fig. 6.20, region A3). Although our SDMD result is blurry and vague (A2), the basic facial features are in the right position. Another drawback of Real-ESRGAN, as well as other deep learning-based methods (Dong et al., 2014; Wang et al., 2019; Shi et al., 2016b; Ledig et al., 2017), is that they can perform super-resolution on only *one* discrete upscaling factor, e.g., increase the resolution by a factor of two or four, as demonstrated by the respective papers. When one requires another upscaling factor, a totally different model needs to be trained. In contrast, our method can easily generate results with different magnifications, even when dealing with floating-point factors, with no added complexity for this.

![Image](image.png)

Figure 6.20: The super-resolution performance of SDMD and the Real-ESRGAN for three images with different sizes. The upscaling factor is 8.

Summarizing the above, our SDMD method can generate comparable SR results with state-of-the-art deep learning-based approaches. Although yielding poorer quality results, our method will not produce weird results as the DL method does for small images. Besides, SDMD is fully generic and does not require considerable training data or special interpolation tricks (Maalouf and Larabi, 2012). More importantly, images can be enlarged by any factor with our method, which cannot be easily achieved by the deep learning-based methods.
6.4.2 **Salient detail encoding**

Similar to CDMD method, SDMD simplifies an image *globally*, e.g., removing islands smaller than a *global* threshold $\epsilon$ or pruning skeletal branches with a saliency below a *global* threshold $\sigma_0$. This is not desirable in practice for certain images that contain different levels of detail. Figure 6.21 gives an example. For the input image (a), (b) shows the SDMD reconstruction using the default global island threshold $\epsilon = 0.001$. As visible from the enlarged area on the right, SDMD loses some small but important details of the cat’s face. Further reducing $\epsilon$ can alleviate this, but this also allocates more information to encode the (less important) background, thereby reducing compression. To address this, we allow users to define salient areas based on manually drawn maps, as shown in Fig. 6.21 (c). Based on these maps, we use a low threshold for salient areas ($\epsilon = 0.0005$ in this example) and a larger threshold for regions outside the important areas ($\epsilon = 0.0015$ in this example). This way, we obtain an identical CR as when using the global $\epsilon$ setting. However, the quality slightly increases since we now preserve more details in the salient area (d). Apart from manually designed maps, automatically computed saliency-maps generated by supervised methods (Borji et al., 2015) or unsupervised methods (Zhang et al., 2018; Peng et al., 2017; Li et al., 2013) can also be used out-of-the-box with SDMD.

![Image](image.png)

Figure 6.21: The benefits of handling salient information. (a) Input image. (512×337) (b) The SDMD reconstruction with the enlarged area of the face on the right. (c) The manually set salient area. (d) The SDMD reconstruction considering salient information.

6.5 **Discussion**

We next discuss several aspects of our SDMD image compression method.
**Speed:** SDMD is linear in the number of pixels of the input image. To gain more insights, we measured the time SDMD needed for a color image at eight different resolutions on a Linux PC with an Nvidia RTX 2060 GPU. Table 10 lists the timings of the four key steps of SDMD (skeletonization, spline fitting, reconstruction, and interpolation). Each step shows the time needed to process all three channels (YCbCr). Skeletonization and reconstruction are relatively less expensive operations as they are very efficiently implemented on the GPU. Interpolation is a bit more expensive since it needs to compute distance transforms for all the selected layers and use them to perform a per-pixel interpolation (Sec. 3.2.1). Spline fitting executes the least-squares optimization and the adaptive-degree fit-and-split algorithm (Chapter 5), which dominates the running time.

Table 10: Running times of four SDMD steps on images of different resolutions, in milliseconds.

<table>
<thead>
<tr>
<th>Operation</th>
<th>320×200</th>
<th>640×400</th>
<th>960×600</th>
<th>1280×800</th>
<th>1600×1000</th>
<th>1920×1200</th>
<th>2240×1400</th>
<th>2560×2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skeletonization</td>
<td>48</td>
<td>182</td>
<td>294</td>
<td>729</td>
<td>1119</td>
<td>1559</td>
<td>3501</td>
<td>4168</td>
</tr>
<tr>
<td>Spline fitting</td>
<td>1648</td>
<td>1236</td>
<td>2136</td>
<td>2098</td>
<td>2812</td>
<td>3849</td>
<td>4650</td>
<td>5592</td>
</tr>
<tr>
<td>Reconstruction</td>
<td>62</td>
<td>118</td>
<td>292</td>
<td>561</td>
<td>951</td>
<td>1583</td>
<td>2502</td>
<td>3618</td>
</tr>
<tr>
<td>Interpolation</td>
<td>28</td>
<td>260</td>
<td>345</td>
<td>1004</td>
<td>1479</td>
<td>2105</td>
<td>3904</td>
<td>4960</td>
</tr>
</tbody>
</table>

**Ease of use:** SDMD has four free parameters that affect the trade-off between the compression ratio and image quality, as discussed in detail in Sec. 6.3.2. The meaning of these parameters is quite straightforward: \(L\) determines how many layers (image intensities for a grayscale image) are used for the reconstruction; \(\varepsilon\) controls the maximum size of small-scale details that are removed; \(\sigma_0\) controls how much to smooth isophote or isochrome contours in an image; and \(\gamma_0\) tells how accurately B-splines fit the MAT, i.e., how precisely we want to encode the position and shape of objects in an image. More importantly, Sec. 6.3.2 provides good defaults for all these parameters and also shows that the method is predictable and robust when these are varied away from their presets.

**Replicability:** We implemented the entire SDMD method in C++. We compute MAT and reconstruct the threshold-sets from a rasterized spline using the public CUDA code provided at Telea (2019). We provide the full source code of SDMD, as well as the image benchmark used in this chapter, for replication purposes (Wang et al., 2021c).

**Limitations:** While SDMD can handle any image type and resolution, it exhibits limited performance for small images (see Fig. 6.16). Figure 6.3 shows an additional result in this sense for an image of 400×2 pixels. Furthermore, SDMD cannot get better compression and quality than JPEG.
for all image types even for large resolution images: Like CDMD, SDMD is not good at handling images with many fine details (high spatial frequencies). Figure 6.22 illustrates this by showing three such images. Their SSIM scores are quite low due to the fact that SDMD cannot encode and reconstruct very thin image details. Additional insights shown in Sec. 6.3.4.3 show a similar positioning of SDMD vs. more modern variants of JPEG, specifically BPG and JPEG 2000. However, we argue that, for any practical purpose, the SDMD representations actually look visually very similar to the input images and are largely free of obvious artifacts, such as color banding, checkerboarding, or false hues. This may suggest that the SSIM metric used to compare images is too strongly penalizing such small details and opens the broader question on which metrics should be further considered to compare lossy-compressed images in practice. This is an important question that, albeit out of our current scope, deserves further research.

Figure 6.22: Poor performance (both in SSIM and CR) for SDMD when dealing with images with many fine details, such as animal furs (a), trivial objects (b), and greenery (c). The sizes of the three input images are all $2560 \times 1600$. 
6.6 CONCLUSION

We have presented SDMD, a method for compressing color and grayscale images by encoding dense medial descriptors obtained from the images’ threshold sets with accurate B-splines. SDMD adapts the existing CDMD method—proposed for encoding images with medial descriptors—in four directions, namely (a) replacing the expensive pixel-chain coding of medial descriptors by B-splines, (b) adaptively encoding upper or lower threshold-sets to minimize the amount of storage space, (c) separately treating chrominance and achrominance, and (d) eliminating medial Y-structures that touch the image boundary. To study the effectiveness of our method, we considered a benchmark of five different image types, each type having at least 10 images. The quantitative evaluation showed that our adaptations of CDMD greatly improve compression at only a small quality loss. Furthermore, the proposed SDMD delivers superior compression to the well-known JPEG method at similar or even better quality, especially for large images. Finally, we show how SDMD can be used out-of-the-box to generate super-resolution images and also can be adapted to perform local salience-based compression. SDMD is implemented on the GPU, making its application take only a few seconds on a modern PC for images up to 2000\(^2\) pixels.

In the next chapter, we explore the addition of saliency maps applied in Chapter 4 to vector-based dense skeletons. We illustrate the new combined proposal by comparing it with JPEG and JPEG 2000 and show improved results as compared to the method presented in this chapter.
In the previous chapter, we have explored the potential of spline-based dense medial descriptors (SDMD method) for image compression. While achieving much higher compression ratios at similar or even better quality to the well-known JPEG technique, SDMD can only simplify an image globally. In this chapter, we improve this by incorporating saliency maps used in Chapter 4 into the pipeline and present Spatial Saliency Spline Dense Medial Descriptors (3S-DMD) for saliency-aware image simplification-and-compression. Our method significantly improves the trade-off between compression and image quality of earlier medial-based methods while keeping perceptually salient features. We also demonstrate the added-value of user-designed, as compared to automatically-computed, saliency maps. We show that our method achieves both higher compression and better quality than JPEG for a broad range of images and, for specific image types, yields higher compression and similar quality than JPEG 2000.

7.1 Introduction

In Chapter 3, we have proposed Compressing Dense Medial Descriptors (CDMD), a particular class of simplification-and-compression methods that models images as a set of luminance threshold-sets and encodes these by their Medial Axis Transforms (MATs). While CDMD showed promising quantitative and qualitative results, it cannot yet compare in both visual quality and compression ratio (CR) with state-of-the-art compression methods like JPEG or similar. Two lines of research tried to address this issue.

Improving quality: CDMD simplifies an image globally, making it hard to preserve fine details in some areas while strongly simplifying the image in other areas. The SSDMD method demonstrated in Chapter 4 addressed this by adding a saliency map to CDMD, allowing users to specify different spatial simplification levels over an image. SSDMD delivers higher local quality than CDMD (as specified by the saliency map) but has two key limitations. First, it only marginally improves CR when compared to CDMD, since highly-salient image areas actually increase the MAT information needed to be stored. Secondly, SSDMD uses automatically computed saliency maps to control simplifica-

This chapter is based on the submitted paper “Spline-Based Dense Medial Descriptors for Image Simplification Using Saliency Maps”
tion. Such maps can significantly fail to capture what users perceive as salient (thus, to be preserved) vs non-salient (thus, to be simplified). Figure 7.1 outlines this problem for four images (a1–d1) with saliency maps (a2–d2; bright=salient; dark=non-salient) automatically computed by the DSR method (Li et al., 2013). SSDMD compression results (a3–d3) arguably lose details that humans would find salient, such as blurred faces (a3, b3, c3) and nearly complete loss of the leopard skin texture (d3).

Figure 7.1: Examples of the DSR saliency estimator failing to detect salient objects: (a3–d3) show the SSDMD results using the saliency maps (a2–d2) for images (a1–d1).

**Improving compression:** CDMD stores the MATs of an image’s threshold sets as pixel chains, which is exact, but inefficient storage-wise. The Spline Dense Medial Descriptors (SDMD) method illustrated in Chapter 6 improved CR by representing MATs with accurate and compact-storage B-spline descriptors for each threshold set. Yet, just as CDMD, SDMD simplifies images only globally, thus increasing CR but achieving limited visual quality.

**Our contributions:** We jointly address the visual quality and CR goals of all above earlier MAT-based image compression methods by a single method:

- We combine the strengths of SSDMD in Chapter 4 (spatial control of image simplification) with SDMD in Chapter 6 (compact encoding of MATs with B-splines);
- We allow users to interactively tune parameters of the joint method, including full control over the saliency map design;
- We evaluate our proposal on additional image types and compare it, with favorable results, with state-of-the-art methods (JPEG and JPEG 2000).
7.2 PROPOSED 3S-DMD METHOD

As stated above, an important limitation of SDMD is that it simplifies an image globally. Our earlier work, Spatial Saliency DMD (SS-DMD) in Chapter 4, addressed this by simplifying the CDMD’s MATs using a spatial saliency map. We next present both SSDMD and our new method, 3S-DMD, as depicted in Fig. 7.2, which improves SSDMD in several respects. Section 7.2.1 shows how 3S-DMD benefits from manually-designed saliency maps via an interactive application. Section 7.2.2 presents SSDMD’s saliency-map-based simplification of the MAT and how we improved this by saliency-based spline fitting.

![Figure 7.2: Spline-based dense medial descriptors pipeline with free parameters in green. Elements added by 3S-DMD method proposed in this chapter are marked in red.](image)

### 7.2.1 User-driven saliency map generation

Section 4.1 reviewed a variety of techniques to automatically compute saliency maps from an image. As mentioned in Sec. 7.1, such automatically-computed maps may not fully meet user needs (cf. Fig. 7.1 (a2, b2)) or even fail to detect salient objects (cf. Fig. 7.1 (c2, d2)). Even when such maps fit with what users expect, the simplification they induce can lead to unwanted results due to the hard-to-predict shapes that skeletons have. To handle all such issues, we developed an interactive application that allows users to create their custom saliency maps or adjust maps created by automatic methods. Figure 7.3 (a) shows the user interface, in which one can draw the saliency map using tools listed in the toolbar, tune all the method’s parameters, run the end-to-end pipeline, and check the obtained results. A video of our tool is provided in the supplementary material (Wang et al., 2021b).

We provide three ways for users to manually design saliency maps, as follows.
Figure 7.3: Direct saliency drawing. (a) Interface with a loaded image. (b) User drawing to specify the saliency. (c) Computed saliency map. (d–f) Generated saliency maps.

### 7.2.1.1 Direct drawing

Users can directly paint a saliency map with various brushes, whose brightness gives the desired saliency (black=0, white=1). Figure 7.3 (b) shows the drawing of a map for the car image in Fig. 7.3 (a). The user marked the car area as highly salient (white ellipse, region J) and background areas farther from the car as zero salient (black scribbled bands, region H). Figure 7.3 (c) shows the computed saliency map $\mu$. Regions where the user painted saliency are taken over from the drawing (H’ and J’ are copies of H and J, respectively). Unpainted areas carry no hints that the user found them important or not (Fig. 7.3 (b), region I). We set here the saliency to the average value $\mu = 0.5$ (Fig. 7.3 (b), region I’).

### 7.2.1.2 Adjust precomputed saliency map

Fully painting a custom saliency map can be cumbersome, especially when one wants to use multiple saliency values. We support this use-case by allowing users to draw to modify a precomputed saliency map. Figures 7.3 (d–f) show three such precomputed maps obtained with the DSR method (Li et al., 2013), structured matrix decomposition (SMD) method (Peng et al., 2017), and the recent ITerative Saliency Estimator fLExible Framework (ITSELF) (de Melo Joao et al., 2020) method.

### 7.2.1.3 SSIM-guided user-specified saliency

Users may be unfamiliar with, or unable to run, existing saliency estimation methods. Also, they may not know how to tweak saliency to get the best quality-compression balance. We address these issues by
computing the saliency map in a corrective way, i.e., by comparing the compression method’s output with its input. Figure 7.4 shows how this works. Given an input image (a), we first run SDMD without a saliency map. Next, we evaluate the quality of the output (b) by the Structural SIMilarity (SSIM) metric (Wang et al., 2004), as described in Sec. 2.3. The generated SSIM map (c) shows the per-pixel structural similarity between the original (a) and the output (b), with darker pixels indicating less similar regions. Figure 7.4 (c) shows that SDMD yields poorer quality over several car details, especially its two wheels. Having this insight, we scribble bright colors on the two wheels to tell their importance (Fig. 7.4 (d)). We now use this quite simple saliency map (e) to run 3S-DMD to generate a new result (f). As visible in the last image, the quality of the left front wheel has improved.

Figure 7.4: SSIM-guided user-specified saliency map generation.

7.2.2 Saliency-based parameter control

We next show how to use the saliency maps created by the various methods in the denoising (step 2), regularization (step 4), and spline fitting (step 5) of our end-to-end pipeline (Fig. 7.2).

7.2.2.1 Salient islands detection

As explained in Sec. 3.2.2, CDMD only keeps islands, or connected components $C_i$, which meet the condition $|C_i| \geq \varepsilon |T_i|$, and similarly for SDMD. Such an operation can remove small but salient features (see Fig. 7.5): For $\varepsilon = 0.04$, the pigeon’s eyes, visible in the original image (a1), are removed (a2). We confirm this by verifying that the small islands in region A in the threshold-set $T_{127}$ (b1) get lost in $T'_{127}$ (b2). Lowering $\varepsilon$ can alleviate this, but this allocates more information to encode the less important background, thereby increasing image size. To address this, we use the saliency map $\mu$ to compute a saliency-aware metric

$$C_i^\mu = \sum_{x \in C_i} k_1^{2\mu(x) - 1},$$

(7.1)
and next remove only islands for which \( C_i^\mu < \epsilon \). The factor \( k_1 \) in Eqn. 7.1 controls how much \( \mu \) affects island removal. For \( k_1 = 1 \), \( C_i^\mu = |C_i| \), so our method behaves like the original SDMD. In practice, we set \( k_1 = 5 \), which means that the most salient pixels \( (\mu(x) = 1) \) are given five times their original unit weight; the least important pixels \( (\mu(x) = 0) \), in contrast, get one-fifth of their original unit weight. This keeps small-size, but salient, details in the compressed image. Figure 7.5 (b3) shows this for a saliency map computed with the DSR method (Li et al., 2013). Islands in region A, while small, have a high \( \mu \), so they are retained. In contrast, although large, the island in region B has a low saliency, so it is removed. This ends up with a smaller size, but perceptually better, result (a3).

Figure 7.5: Pigeon image (a1) encoded with SDMD (a2) and 3S-DMD (a3) with saliency map (c) computed by the DSR (Li et al., 2013) method. Images (b1–b3) show details in layer 127.

7.2.2.2 Saliency-aware skeleton simplification

As Sec. 3.2.2 outlined, C/SDMD regularizes skeletons \( S_l^\sigma \) by keeping only pixels \( x \in S_l^\sigma \) where \( \sigma(x) \) exceeds a user-set threshold \( \sigma_0 \) (Fig. 7.2, step 4). The SSDMD method (Chapter 4) further simplifies \( S_l^\sigma \) by removing points \( x \in S_l^\sigma \) whose saliency \( \mu(x) \) is smaller than a new threshold. This not only increases the number of thresholds users have to deal with but also yields poor quality as low-saliency areas get completely removed. SSDMD alleviates this by using various heuristics such as selective layer keeping and interpolation tricks. However, this makes the end-to-end method quite complex.

In contrast to SSDMD, we blend \( \sigma \) with the saliency map \( \mu \) by computing

\[
\sigma'(x) = \sigma(x) \cdot k_2^{\mu(x) - 1}
\]
and then obtain $S'_{T_l}$ by upper-thresholding $\sigma'(x)$ with the user-set value $\sigma_0$, i.e.,

$$S'_{T_l} = \{ x \in S_{T_l} | \sigma'(x) > \sigma_0 \}.$$  

(7.3)

The value $k_2$ in Eqn. 7.2 controls how much $\mu$ affects the skeleton simplification. For $k_2 = 1$, our new metric $\sigma'$ equals the original $\sigma$ from SCDMD. In practice, we set $k_2 = 2$. Hence, the salient-skeleton values $\sigma'(x)$ of the least important pixels ($\mu(x) = 0$) become half of their original $\sigma(x)$ values; in contrast, the $\sigma$ values of the most important pixels ($\mu(x) = 1$) stay unchanged. Figure 7.6 shows the improvement given by our new metric $\sigma'$. Images (e, f) show the regularized skeletons $S'_{T_{43}}$ of one layer, $T_{43}$, computed with SDMD’s $\sigma$ metric and 3S-DMD’s $\sigma'$ metric, for the same user-set $\delta = 0.6$, and a simple manually-designed saliency map, for illustration purposes (image d). We see how 3S-DMD (image f) simplifies skeletons in the image background more, since the saliency is low there, than SDMD (image e), which has no notion of a low-importance background. In contrast, in the foreground image areas (white areas in the saliency map $\mu$), the 3S-DMD and SDMD skeletons are identical. As a result, 3S-DMD yields the same image quality as SDMD, but with about 10% extra compression.

Figure 7.6: Salient skeleton detection on a squirrel image (a) with SDMD (b, e) and 3S-DMD (c, f) using a manually-designed saliency map (d).

7.2.2.3 Saliency-based spline fitting

Section 6.1 stated that SDMD finds the minimal number of B-spline control points needed to reach a user-given fitting error $\gamma_0$ between a skeleton branch $B_i$ and the B-spline $C_i$. This error is given by the Hausdorff distance $H(B_i, C_i)$ computed over all pixels $x \in B_i$. We modify the fixed user-set threshold $\gamma_0$ to involve the saliency map $\mu$ by

$$\gamma' = \gamma \frac{\sum_{x \in B_i} k_3^{1-\mu(x)}}{|B_i|},$$  

(7.4)
where $k_3$ controls how much $\mu$ influences the spline fitting. For $k_3 = 1$, $\gamma'$ equals the original $\gamma_0$. We set $k_3 = 2$ in practice. Hence, when a branch is fully within a zero-saliency region ($\mu(x) = 0$), $\gamma' = 2\gamma$, i.e., we allow a double fitting error as compared to the original SDMD. For branches located in a maximum saliency regions ($\mu(x) = 1$), the fitting error stays the same, i.e., $\gamma' = \gamma$.

### 7.3 Results

Section 7.2 proposed 3S-DMD, a method that incorporates three schemes for users to create a spatial saliency map, three ways for adjusting the original SDMD with these maps, and a saliency-aware quality metric $Q^\mu$ to measure how well the reconstructed image $\tilde{I}$ captures the input image $I$. We next evaluate 3S-DMD’s results in detail, as follows.

- First, we describe our evaluation methodology (Sec. 7.3.1).
- We show how 3S-DMD depends on its free parameters (Sec. 7.3.2).
- We compare 3S-DMD with CDMD in Chapter 3 and SDMD in Chapter 6 (Sec. 7.3.3).
- Atop Chapter 4, we also compare with the JPEG and JPEG 2000 methods (Sec. 7.3.4).

#### 7.3.1 Evaluation methodology

The 3S-DMD encoding consists of a tuple $(w, h, \{I_i\})$, i.e., the pixel width $w$ and height $h$ of the input image $I$, and the $L$ selected layers $I_i$. A layer $I_i = (i, f, \{b_i^k\})$ has an intensity value $i$, a flag $f$ that tells if it uses upper- or lower-thresholding (for details, see Chapter 6), and a B-spline set $\{b_i^k\}$ encoding its MAT. Each B-spline $b_i^k = (d_i^k, \{c_j\})$ has a degree $d_i^k \in \mathbb{N}$ and control points $c_j \in \mathbb{R}^3$. Each control point $c_j = (p_j, DT_{\text{II}}(p_j)) \in \mathbb{R}^3$ consists of a 2D position $p_j$ and its corresponding $DT$ value.

Sizes of the images $\tilde{I}$ and $I$ are typically measured by bits per pixel (bpp), i.e., the number of bits used to encode a pixel’s grayscale or color value (Daintith and Wright, 2008). Yet, in an encoding context, we want to compare the sizes of $\tilde{I}$ and $I$, rather than measure their absolute sizes. For this, we define $CR = |\tilde{I}|/|3SDMD(\tilde{I})|$. Here, $|3SDMD(\tilde{I})|$ is the bytesize of the 3S-DMD storage scheme outlined above, while $|\tilde{I}|$ is the size (in bytes) of the original image $I$.

To measure how perceptually close the 3S-DMD result is to the original image, we apply the saliency-aware MS-SSIM metric $Q^\mu$, see details in Sec. 4.2.3. The quality $Q^\mu$ and compression ratio $CR$ of 3S-DMD depend on four parameters (Fig. 7.2): the number of selected thresholds $L$, the size of removed islands $\epsilon$, the skeletal saliency threshold
7.3 results

... and the spline fitting tolerance $\gamma_0$. We establish ranges for these parameters based on results of previous work (Chapter 3, 4, 6), as follows: $L \in [1, 60]$, $\varepsilon \in [0.001, 0.1]$, $\delta \in [0.01, 3]$, and $\gamma \in [0.001, 0.005]$. We further sample these ranges by the following representative values: $L \in \{15, 25\}$, $\varepsilon = 0.02$, $\delta \in [0.3, 0.8]$, and $\gamma = 0.0015$. We use these values to compare CDMD, SDMD, and 3S-DMD (Sec. 7.3.3) and 3S-DMD with JPEG and JPEG 2000 (Sec. 7.3.4). We test all these methods on a 50-image database, which is selected randomly from the MSRA10K (Cheng, 2016), SOD (Movahedi and Elder, 2010), and ECSSD (Shi et al., 2016a) benchmarks. In addition to these real-world pictures, we also tested 3S-DMD on several artificially-designed images (Sec. 7.3.4). All test images have a resolution between $1000^2$ to $2000^2$ pixels.

7.3.2 Effect of parameters

To intuitively illustrate how 3S-DMD performs for different parameter values, we first group these into weighting factors and user thresholds, and show the effect of these for a specific image.

7.3.2.1 Weighting factors effect

As explained in Sec. 7.2.2, the $k_1$, $k_2$, and $k_3$ factors control how much the saliency map $\mu$ affects the island detection, skeleton simplification, and spline fitting, respectively. We call these weighting factors—in contrast to the user parameters discussed next—since they are more technical parameters, which do not arguably need to be exposed to end users. Secondly, their effect is strongly related to the way 3S-DMD treats image areas of different saliency. Let ‘foreground’ and ‘background’ describe areas of high, respectively, low saliency map $\mu$ values. Simply put, increasing all (or any) of these three weighting factors progressively simplifies the image background, similarly to a (soft) blurring effect, but keeps the image foreground relatively untouched. Figure 7.7 shows this for a flower image under different values for $k_1$, $k_2$, and $k_3$. The user parameters are all fixed to the default values $L = 25$, $\varepsilon = 0.02$, $\delta = 0.3$, $\gamma = 0.0015$.

The setting $k_1 = k_2 = k_3 = 1$, shown in Fig 7.7 (c1), corresponds to the original SDMD method since, for this setting, $\mu$ has no effect on island detection, skeleton regularization, and spline fitting (see Eqns. 7.1, 7.2, and 7.4). As we increase $k_1$, $k_2$, and $k_3$, the image background gets progressively more simplified; see Figs. 7.7 (c2–c4). However, the flower in the foreground stays roughly the same in all images. The $CR$ and $Q^\mu$ values shown below the images match the above observations: as the weights increase, $Q^\mu$ drops only slightly, but $CR$ increases strongly. In practice, as stated in Sec. 7.2.2, we found $k_1 = 5$, $k_2 = 2$, and $k_3 = 2$ to be a good default for balancing $CR$ and $Q^\mu$. 

119
Figure 7.7: Progressive simplification of a flower image (a) using a saliency map (b) for different weight values $k_1$, $k_2$, and $k_3$ (c1–c4).

7.3.2.2 User thresholds effect

3S-DMD depends on four thresholds, as follows:

- $L$ controls how smoothly the simplified image captures color gradients; larger values yield smoother gradients;

- $\varepsilon$ gives the scale of details that are kept in the image; larger values remove larger details;

- $\sigma_0$ controls the scale of corners that are kept in the image; larger values round off larger corners;

- $\gamma_0$ tells how accurately B-splines fit skeleton branches; larger values yield more distorted results.

In contrast to the weighting factors discussed earlier, these four thresholds significantly influence the ‘style’ of the simplified image. Hence, we believe they are best left under the direct control of the end users.

Figure 7.8 shows the effect of the thresholds $L$, $\varepsilon$, $\sigma_0$, and $\gamma_0$ by showing the 3S-DMD results on the same flower image, using the same saliency map, as in Fig. 7.7. Image (a2) shows the results of 3S-DMD when setting user thresholds. The remaining images (b1–b4) are each the effect of a single user threshold change (red in the legend). If we decrease $L$ (image (b1)), even if we select only $L = 15$ layers, we still get a visually convincing result. Yet, the stamens in region B and the flowers in regions A and C look duller than in image (a). Image (b2) uses a higher $\varepsilon$ value, which removes many large islands in the image background, e.g., the one corresponding to the yellow flower in region A. Image (b3) uses a higher $\sigma_0$, which rounds off corners of background shapes, e.g. the flowers in regions A and C. Finally, image (b4) uses a
higher $y_0$, which distorts the boundaries of the flower in region A and creates subtly false colors in region D.

Figure 7.8: 3S-DMD results for a flower image (a) using the saliency map of Fig. 7.7 (b) for different combinations of parameters $L$, $\varepsilon$, $\sigma_0$, and $y_0$.

7.3.3 Comparison with CDMD and SDMD

Figure 7.9 compares the $Q^\mu$ and CR values of CDMD (blue markers), SDMD (red markers), and our proposed 3S-DMD (green markers), for the four user-parameter settings listed in Sec. 7.3.1, using a star plot. Small dots indicate metric values for a run involving a method-and-parameter-setting on a single image. Markers at the ‘star centers’ show average values for all runs over the 50 images in the benchmark for one parameter setting and one method. For each method (color), there are four stars, one for each of the four parameter-settings used, as indicated by the four glyph types in the figure’s legend. The star center triples depicted using the same glyph show runs that use the same parameter settings. We fixed $\varepsilon = 0.02$ and $y = 0.0015$ so these user parameters are not listed in the figure’s legend.

Figure 7.9 offers several insights. Small stars show little variance in $Q^\mu$ and CR from the average for a given method-and-parameter-set. Large stars indicate more variance as a function of the actual images. The sizes and shapes of the stars in the figure are quite similar. Hence, CDMD, SDMD, and 3S-DMD show a similar dependency of CR and $Q^\mu$ on the real-world image type. This is due to the fact that SDMD and 3S-DMD inherit the thresholding and skeletonization used by CDMD. Yet, the green stars are slightly larger and more spread horizontally, indicating that 3S-DMD can produce greater changes in CR for similar $Q^\mu$. 
Figure 7.9: Star plots of CDMD (blue markers), SDMD (red markers), and 3S-DMD (green markers) for 50 real-world images. The actual image data (smaller dots) are connected to the corresponding average value (star center markers) for each method. Marker shapes indicate the four parameter settings being used.

For each color (method), its four stars show an inverse correlation of CR with $Q^\mu$. Indeed, more layers and smaller $\sigma_0$ yield higher quality but less compression; conversely, fewer layers and larger $\sigma_0$ slightly reduce quality, but strongly increase compression. The axes ranges show this too: CR varies roughly from 50 to 700, while quality varies between 0.91 and 0.98. The three large dots of the same glyph types let us compare the CDMD, SDMD, and 3S-DMD methods under the same parameter setting. We see a clear inverse correlation pattern going from high $Q^\mu$ and low CR (CDMD, blue dots) to average $Q^\mu$ and CR (SDMD, red dots) and then to lower $Q^\mu$ and highest CR (3S-DMD, green dots). Hence, 3S-DMD always gets higher CR than CDMD and SDMD for only a small quality loss. On average, 3S-DMD increases CR by 234.2% relative to CDMD, while $Q^\mu$ drops by only 0.014. Compared with SDMD, 3S-DMD increases CR on average by 53.8%, while $Q^\mu$ drops by a tiny 0.009. More importantly, when we compare CR and $Q^\mu$ for different parameter settings, e.g., comparing the large round green marker with the star-shaped blue marker and square red marker, 3S-DMD not only yields a higher CR but also better quality.

Figure 7.10 further compares the three methods for six focus-and-context images of insects, birds, animals, and plants from the MSRA10K benchmarks (Cheng, 2016). More results are given in the supplementary material (Wang et al., 2021b). The zoomed-in areas show that, compared with CDMD (a1–f1) and SDMD (a2–f2), 3S-DMD (a3–f3) preserves well important features marked as such by the saliency maps, like highlights (a, d), animals’ eyes (b, c, f), and the flower stamen (e). For background areas, all three methods perform visually roughly the same. The quality
Figure 7.10: Comparison of 3S-DMD (a3–f3) with CDMD (a1–f1) and SDMD (a2–f2) for six focus-and-context images. For each result, we show the saliency-aware MS-SSIM $\mu$ and CR. The rightmost column shows the manually-modified DSR saliency map.

values $\mu$ are also similar for the three methods, with 3S-DMD scoring twice as best, three times as second-best, and once in the third place. CR values show that 3S-DMD achieves (significantly) higher compression than CDMD and SDMD, except for image (b), where it scores slightly below SDMD. On the other hand, 3S-DMD retains for this example more details than SDMD for the foreground area, such as the bird’s eye, as also reflected by its higher quality score.

7.3.4 Comparison with JPEG and JPEG 2000

Tens of image compression exist, see e.g. Toderici et al. (2016); Choi et al. (2019); Agustsson et al. (2019) and the methods cited therein. Comparing 3S-DMD with all of them is not feasible in the scope of this work.
However, we provide a comparison with JPEG (Wallace, 1992) and JPEG 2000 (J2K) (Taubman and Marcellin, 2001) which are arguably among the most well-known, frequently-used, and generic, image compressors.

### 7.3.4.1 Comparison with JPEG

Figure 7.11 compares 3S-DMD with JPEG on our image benchmark. The parameter setting of 3S-DMD (green dots) follows Sec. 7.3.3. JPEG (blue dots) is run under five quality settings: 10%, 30%, 50%, 70%, and 90%. As in Fig. 7.9, we use star plots for both 3S-DMD and JPEG: small dots are individual runs and large dots are averages. We see that 3S-DMD cannot reach the same $Q^\mu$ values as when JPEG uses its 90% quality setting: the topmost blue dot is above the topmost green dot. However, the vertical spread of the blue vs green dots shows that the difference in quality ($Q^\mu$) is small, about 0.05 on average. If we accept this small quality loss, 3S-DMD always gets higher compression rates than JPEG. In the limit, compared to JPEG with a quality setting of 10% (point A), 3S-DMD (point B) gets both higher CR and better quality.

![Figure 7.11: Comparison of JPEG (blue dots), J2K (red dots), and 3S-DMD (green dots) for 50 images. The actual image data (smaller dots) are connected to the corresponding average value (larger dots) for each parameter setting of the three methods.](image)

Figure 7.12 refines the above insights by showing six real-world images (building, plant, animal, natural scene, man-made structure, and people), compressed by 3S-DMD (a1–f1), JPEG (a2–f2), and J2K (a3–f3). We see that JPEG with a 10% quality creates obvious artifacts: checkerboarding (b2, c2, e2, f2), banding (a2, c2), and color faking (d2). 3S-DMD yields better quality ($Q^\mu$) and does not exhibit such artifacts. Yet, 3S-DMD loses small-scale, faint, details in the background, like the gravel in the sea (c1) and the red color of the traffic sign (f1). We argue that these are acceptable losses since these details are located in low-saliency areas. Separately, 3S-DMD always achieves higher CR than JPEG.
Figure 7.12: Comparison of 3S-DMD (a1–f1) with JPEG -10% (a2–f2) and J2K (a3–f3) for six real-world images. For each image, we show the saliency-aware metric \(Q^H\) and CR.

7.3.4.2 Comparison with J2K

Figure 7.11 shows J2K (red dots) run under five fixed compression ratios: 100, 200, 300, 400, and 500. As CR increases, J2K has only a slightly quality loss and performs practically always better than JPEG. Figure 7.12 also verifies this: J2K’s quality \(Q^H\) is always higher than 0.99 and the compressed results are indistinguishable from the originals. 3S-DMD cannot (yet) achieve such quality. However, 3S-DMD can obtain comparable, and sometimes higher, CR values. We further refine the comparison with J2K by considering a narrower class of artificially made images, such as graphics art (logos, graphics design), scientific visualization images, synthesized images using graphics rendering and vec-
torization methods (Orzan et al., 2013), and cartoon images. For such images, 3S-DMD produces higher CR and comparable quality than J2K and JPEG. Figure 7.13 shows four representative images, one from each of the above four categories, compressed with 3S-DMD (a1–d1), JPEG (a2–d2), and J2K (a3–d3). As in earlier cases, JPEG with a quality setting of 10% generates obvious artifacts such as blocking (a2, b2, c2, d2), banding (c2), and color faking (c2), and has a CR well below the other two methods. When compared with J2K, our method yields similar $Q^\mu$ values. We show some zoomed-in areas to expose a few subtle differences: For the graphics design example (a), 3S-DMD achieves visually much better results, without the checkerboarding and blur artifacts of J2K. This is also seen in the first image in Fig. 7.13 where 3S-DMD got a higher $Q^\mu$ than J2K. For the second image (b) in Fig. 7.13, 3S-DMD captures the smooth luminance gradient in the shadow area quite well. In contrast, J2K causes a slight amount of false color artifacts. For the strong-contrast images (c) and (d), J2K creates some small-scale blur artifacts. 3S-SDMD does not have such problems but suffers from a slight color change issue due to its selection of threshold-sets to be encoded. Most importantly, with a similar or better quality, 3S-DMD always yields higher compression than J2K for such synthetic images.

![Figure 7.13: Comparison of 3S-DMD (a1–d1) with JPEG-10% (a2–d2) and J2K (a3–d3) for four synthetic images. For each image, we show the saliency-aware $Q^\mu$ and CR. The leftmost column shows the saliency map obtained by directly scribbling on the input image. The rightmost three columns show zoomed-in areas for 3S-DMD, JPEG, and J2K for detailed comparison.](image-url)
We conclude that, for real-world images, 3S-DMD gets both higher CR and quality than JPEG but cannot match J2K’s quality at the same CR. Yet, for synthetic images, 3S-DMD gets both much higher CR and quality than JPEG, and also achieves higher CR at similar quality but with fewer artifacts than J2K.

7.4 Discussion

We now discuss several aspects of our 3S-DMD compression method. **Genericity and ease of use:** 3S-DMD is a general-purpose compression method for generic grayscale and color images. It relies on well-tested and robust algorithms such as the skeletonization method in Falcão et al. (2004); Telea and van Wijk (2002) and the least-squares B-spline fitting algorithm (Eberly, 2014). In contrast to segmentation tasks (Falcão and Bragantini, 2019), 3S-DMD does not require precise saliency maps. Any saliency map that encodes which image areas are more important and which less for an application at hand can be used. 3S-DMD has four user parameters: the number of selected layers $L$, island size $\varepsilon$, skeleton saliency threshold $\sigma_0$, and spline fitting error $y_0$. These parameters have intuitive effects and default values, as detailed in Sec. 7.3.2.

**Speed:** We compute the most complex step in 3S-DMD, skeletonization, on the GPU (Cao et al., 2010; Telea, 2019). On a Linux PC with an Nvidia RTX 2060, this takes a few hundred milliseconds for images up to $1024^2$ pixels. Spline fitting uses about 1 second per color channel, yielding a total of about 3 to 4 seconds for the compression.

**Replicability:** We provide our full C++ source code and data for replication purposes, as well as a demo video and additional comparisons with CDMD and SDMD, in the supplementary material (Wang et al., 2021b).

**Limitations:** Image layer components that are one or two pixels thin cannot be encoded by MATs, so 3S-DMD cannot deal optimally with images with many thin-and-long details, such as animal fur, fine textures, and greenery. Figure 7.14 shows this for two such images. For smooth regions in the background (red boxes), 3S-DMD yields results that are indistinguishable from the originals. However, 3S-DMD cannot capture all the fine-grained details present in the foreground (green boxes). One way to handle such cases is to artificially upscale the images, leading to fine details thicker than a few pixels, which next can be skeletonized with no problems. Studying how to perform this efficiently and with good CR values is an interesting topic for future work.

7.5 Conclusion

In this chapter, we have presented 3S-DMD, a method for saliency-aware image simplification and compression. 3S-DMD combines the
strengths of two of its precursors: SSDMD (Chapter 4) that allows spatial control of image simplification, and SDMD (Chapter 6) that compactly encodes MATs with B-splines. We have developed an interactive application for users to set parameters and customize saliency maps in three ways. We have illustrated how saliency maps involved in the SDMD pipeline offer spatially-dependent simplification. We have shown graphically and intuitively how 3S-DMD performs under different parameter combinations. To study the effectiveness of 3S-DMD, we have considered a database of 50 real-world images. Quantitative evaluation showed that 3S-DMD greatly improves the compression of SS-DMD and SDMD at only a small quality loss. Our method delivers both higher CR and quality than JPEG. While we cannot reach the same high quality at the same CR values as J2K, our method yields similar quality, higher CR, and fewer artifacts for a wide class of synthetic images.

So far, we have explored several possibilities of dense skeletons, represented suitably by raster or vector models, using saliency maps or not, for lossy image compression. Quantitative evaluation showed that our methods can finally deliver both higher CR and quality than the well-known JPEG. They can also produce comparable results to the JPEG 2000 and BPG for certain types of images. In addition to image compression, we also demonstrated various applications in previous chapters, such as super-resolution image generation (Sec. 5.5.1 and Sec. 6.4.1), multiscale MAT representations (Sec. 5.5.2), and manipulation for binary images (Sec. 5.5.3). In the next chapter, we further explore whether the dense skeletons represented by B-splines are also useful and usable for performing manipulations of color or grayscale images.
In the previous technical chapters, we have presented several methods for image compression and simplification. Apart from that, we have also explored the potential of the spline-based medial axis transform (SMAT) used for binary image manipulation (Sec. 5.5.3). While generating very interesting deformation results, it is limited to binary images and only to the most basic operations. In Chapter 6, we have extended SMAT to the spline-based dense medial descriptors (SDMD), which generates superb results for color image compression. In this chapter, we further explore the possibilities of SDMD for grayscale or color image manipulation by designing an interactive tool. We replace the upper threshold sets with morphological tree representations of an image, enabling finer-grained spatial control of each level set. We improve the basic operation presented in Sec. 5.5.3 to more diverse and powerful functions, yielding more convenient operations for users. We also demonstrate various interesting applications to verify the effectiveness of our tool.

8.1 INTRODUCTION

Image representation and compression lie at the core of image processing, and we have explored these extensively in this thesis. Nevertheless, image manipulation is just as important due to the popularity and commercial importance of the task. Many image modification methods exist (Milliron et al., 2002; Pérez et al., 2003; Igarashi et al., 2005; Wang et al., 2013; Choi et al., 2018), most of which are based on raster techniques. Arguably, when an image is represented in vector form (Barla and Bousseau, 2013), it is easier and more natural for humans to edit, mainly because vector images are represented using higher-level primitives, often controlled by arrangements of control points with an intuitive and predictable influence on the image.

Equipped with the vector image representation developed in Chapter 6, SDMD, we now explore its suitability for image manipulation. To this end, we develop an experimental tool for users to interactively manipulate grayscale and color images. It exploits SDMD to reach its full potential by providing both local and global control to the user over the elements of the method. At the same time, we leverage the icicle representation of morphological trees of an image, and combine it with SDMD. The contributions of our work are as follows:

This chapter is based on the submitted paper “Interactive image manipulation using morphological trees and spline-based skeletons”
- **Novelty**: Our method is, to our knowledge, the first approach to combine morphological trees and vector representations for image manipulation;

- **Generality**: Our tool can directly handle any raster image of any resolution;

- **Interactivity**: Except for the initial encoding process, which can be pre-computed and is calculated only once, all subsequent manipulations are in real-time, which brings users instant interactivity;

- **Applications**: We demonstrate the good performance of our tool in a variety of applications, including watermark removal, image deformation, lighting change simulation, image rearrangement, and clothing design.

We start by reviewing related work (Sec. 8.2), which is followed by a detailed description of our image manipulation tool (Sec. 8.3). Then we show concrete applications of our tool (Sec. 8.4) and discuss its merits (Sec. 8.5), before concluding this chapter (Sec. 8.6).

### 8.2 Related Work

We structure related work into two groups: image manipulation methods (Sec. 8.2.1) and morphological tree representations (Sec. 8.2.2).

#### 8.2.1 Image manipulation methods

Image manipulation has attracted a lot of research over the years due to its popularity and commercial importance. One such application that attracts a lot of attention is image or shape deformation, which can be roughly classified as follows.

**Free-form deformation (FFD)** is a popular approach for image (and shape) manipulation (Sederberg and Parry, 1986; MacCracken and Joy, 1996; Milliron et al., 2002). This method explicitly divides the (image) space into many domains, e.g., lattices (Milliron et al., 2002) and cages (Lipman et al., 2008; Gain and Bechmann, 2008), and manipulates each domain by moving control points defined in it, as illustrated in Fig. 8.1 (a). While allowing precise and flexible control (Wang et al., 2013; Reis and Kosinka, 2018), setting FFD domains is tedious, requiring the user to laboriously manipulate many control vertices (Igarashi et al., 2005). In addition, FFD methods do not take into account the natural way in which objects move in the real world (Weng et al., 2006; Mota et al., 2011).
Figure 8.1: Illustrative images of three shape deformation techniques. (a) A free-form deformation example taken from Milliron et al. (2002). (b) Skeleton-based image manipulation in Wang et al. (2013). (c) A physically-based approach example taken from Igarashi et al. (2005).

**Skeleton-based approaches** are also widely used for shape deformation, using a pre-defined skeleton to manipulate the input shape (Wang et al., 2013; Tagliasacchi et al., 2016). Note that this skeleton is not exactly the medial axis (Sec. 2.1.1) we used in the previous chapters. Rather, it is similar to the bones of a character, see Fig. 8.1 (b). The key differences are in the properties of the aforementioned skeleton – most notably, the local centrality property, enforced for medial axes by their very definition (Eqn. 2.1), does not necessarily hold for the above-mentioned skeleton structures.

The typical workflow of skeleton-based methods is to bind the components of the character to be edited to a pre-defined skeleton such that each component follows the motions of its associated bones. Skeleton-driven approaches are also commonly used in the deformation of 3D shapes (Lewis et al., 2000; Yan et al., 2008; Jacobson and Sorkine, 2011). While offering intuitive control of 2D or 3D shapes, binding a shape to a skeleton, either manually or automatically, is not a trivial task (Lewis et al., 2000), especially for shapes lacking an obvious bone-and-joint structure, e.g., jellies (Igarashi et al., 2005), to mention just one salient example.

**Physics-based methods** (Igarashi et al., 2005; Sýkora et al., 2009; Weng et al., 2006; Yang et al., 2012) can be regarded as variants of detail-preserving differential mesh deformation techniques (Yu et al., 2004), which deform shapes by modeling their rigidity. These methods allow the user to directly manipulate a shape through a click-and-drag interface, as shown in Fig. 8.1 (c), and generate physically natural results by minimizing local shape distortion. However, such methods are compu-
tationally expensive, resulting in slow convergence, and require careful tuning of several parameters (Weng et al., 2006).

Image deformation techniques, as described above, are most suitable for images with sharp delineated and simple shapes. For more complex images, additional image manipulation applications have been investigated. Pérez et al. proposed Poisson Image Editing (Pérez et al., 2003), a gradient-based image manipulation method, which is a simple and efficient way for many operations, such as seamless cloning, contrast enhancement, texture flattening, and local illumination/color changes. Since then, numerous applications have exploited the benefits of working in the gradient domain. Raskar et al. (2004) presented a class of image fusion techniques to automatically combine images of a scene captured under different illuminations. Levin et al. (2004) proposed a technique for image stitching which combines several individual images that have some overlapped regions. Sun et al. (2004) formulated the problem of natural image matting as one of solving Poisson equations with the matte gradient field. Finally, Arias et al. (2011) proposed a general variational framework for non-local image inpainting. More related work can be found in Di Martino et al. (2016).

In recent years, deep learning-based methods have significantly boosted the performance of image manipulation due to the availability of large amounts of data that one can train on (Isola et al., 2017; Kim et al., 2017; Zhu et al., 2017; Choi et al., 2018). These methods mainly focus on a task called image-to-image translation, which aims to convert a specific aspect of a given image into another, ranging from changing the facial expression (Choi et al., 2018) or hair color (Kim et al., 2017) of a person to modifying the seasons of scenery images (Zhu et al., 2017). While yielding amazing image manipulation results, these methods require a significant number of labeled image pairs. To avoid this, Vinker et al. (2020) introduced a novel method for training deep conditional generative models from a single image. After training, this method is able to perform challenging image manipulation tasks by modifying the primitive representation. However, this approach requires training a separate network for every image, which can be expensive on large datasets. Furthermore, deep learning-based methods generally do not have a convenient interface for user-interactive operation.

In this chapter, we propose an interactive image manipulation method that differs from all the previously described techniques. We integrate two novel works: the icicle representation for morphological trees (described next) and the interactive spline-based medial axis for shape manipulation (Sec. 5.5.3).

8.2.2 Morphological tree representation

As described in Chapters 3–7, in order to be able to represent a grayscale image $I$ with medial descriptors, we decompose $I$ in
a set of binary images (called level sets) by upper thresholding \( T_i^\uparrow = \{ x \in I \mid I(x) \geq i \}, 0 \leq i < n \) or lower thresholding \( T_i^\downarrow = \{ x \in I \mid I(x) \leq i \}, 0 \leq i < n \) (Sec. 6.2.1). This works efficiently for image compression tasks. Yet, when it comes to image manipulation, finer-grained spatial control of each level set is required. Figure 8.2 shows an example. The synthetic image (a) considered in the figure contains nested triangles and nested discs; (b) then shows its four upper-level sets. When one wants to remove, rotate, scale, or move those triangles, it is inconvenient to manipulate them individually. To conquer this, we propose to use the morphological tree representation (Salembier et al., 1998; Berger et al., 2007; Souza et al., 2016), which represents hierarchically all connected components of an image. Thus, a morphological tree is a complete representation of an image.

The most common morphological tree representations contain trees of shapes (Ballester et al., 2003) and component trees (Najman and Coupprie, 2006). The latter are usually represented by compact and non-redundant data structures called max-trees \( (U(I), \subseteq) \) and min-trees \( (L(I), \subseteq) \). The sets \( U(I) \) and \( L(I) \) are composed of the connected components (CCs) of \( T_i^\uparrow \) and \( T_i^\downarrow \), respectively, i.e.,

\[
U(I) = \{ C \in CC(T_i^\uparrow) : i \in [0, n) \}
\]

and

\[
L(I) = \{ C \in CC(T_i^\downarrow) : i \in [0, n) \},
\]

where \( CC(T_i) \) denotes the sets of either 4- or 8-connected components of the threshold sets \( T_i \). The max-tree representation of Fig. 8.2 (a) is shown in (c). From that, one can either process each triangle shape individually, or, alternatively, all triangles collectively by selecting all descendant nodes of node \( E \). Component trees can be computed and processed efficiently (Wilkinson et al., 2008; Carlinet and Géraud, 2014; Souza et al., 2015), which is widely used in object recognition (Souza et al., 2015), 3D segmentation (Donoser and Bischof, 2006), and remote sensing (Benediktsson et al., 2011).

Figure 8.2: A synthetic image (a), its 4 upper-level sets (b), and the corresponding max-tree with 7 nodes, i.e., connected components (c).
To interactively manipulate a grayscale image with component trees, many visualization tools have been proposed (Westenberg et al., 2007; Passat et al., 2011). In such tools, the user either sets parameters for the manipulation task or selects regions in the input image. Next, the tool shows interactively the filtering or segmentation results. However, since max-trees of natural images have tens of thousands of nodes (see the example in Fig. 8.3), the user only interacts with the image and parameters, and not directly with the max-tree.

Figure 8.3: Lena image (512 × 512 pixels) and its complete max-tree, which contains approximately 41000 nodes. Image taken from Tavares et al. (2015).

To simplify the structure of component trees, Tavares et al. (2015) proposed a simplification procedure based on two attributes: extinction value (Vachier, 1995) and the area of nodes. They further improved the simplification by applying an area difference filter, yielding a more meaningful graphical representation of component trees (Tavares et al., 2016), as shown in Fig. 8.4. However, the simplified tree is no longer a complete representation of the original image. To alleviate this, we next propose to apply a new representation of the component trees: icicle plots (Kruskal and Landwehr, 1983) in Sec. 8.3.1. Icicle plots not only contain all the information of the original image, but they are also more compact and more organized.

8.3 Proposed Method

As stated in Sec. 8.2.1, our proposed method combines two novel works: an icicle representation for component trees and an interactive spline manipulation. Figure 8.5 demonstrates the pipeline of our proposed method. Given a grayscale image, we first compute its max-tree or min-tree, which is next represented in an icicle plot (Sec. 8.3.1). All the nodes in the icicle plot are associated with their corresponding spline control points (step 1). Next, we allow users to select single or multiple nodes for
subsequent manipulation (step 2). Section 8.3.2 describes several methods for node selection. The associated connected components and control points of the selected nodes are displayed for interactive spline deformation (step 3), which is described in detail in Sec. 8.3.3. Finally, the manipulated image is reconstructed (step 4).

8.3.1 *Icicle plot representation*

Icicle plots (Kruskal and Landwehr, 1983; Fekete, 2004; Bostock and Heer, 2009), also called icicle trees, represent hierarchical data in the form of stacked rectangles, usually ordered from top to bottom, following the order of nodes in a tree from its root to its leaves. Compared with other representations, such as node-link visualizations, icicle plots allow an easier reading of the nesting relationships, the areas of the nodes, and various attributes of the nodes, such as, in our case, the grayscale of the encoded objects, their perimeter, circularity, complexity, or the number of skeleton points or spline control points. As such, we choose
the icicle plot metaphor to represent the hierarchical component trees of grayscale images.

Figure 8.6 shows the icicle plot of the max-tree for the synthetic image in Fig. 8.2. The selected image is on purpose simple, for illustration purposes. Each icicle, or node, corresponds to a connected component (CC) in the upper-level sets of the input image. As visible, the brightest disc (filled with red) in the original image corresponds to the node marked by the red box in the lower right corner of the icicle tree. The slender red rectangle at the top is the root node, which takes up the entire width. Each child node is placed under its parent with the width proportional to the area of the component. The grayscale that each node reaches down to is exactly the gray value of the selected CC. The height spanned by each node on the grayscale bar is the gray level difference between the level of the selected CC and the previous level. The fill color of each node can be coded by various attributes, e.g., the number of skeleton points, as shown in Fig. 8.6. Other attributes, including area, perimeter, circularity, and complexity, are also implemented in the tool and available via its user interface.

Figure 8.6: The synthetic image in Fig. 8.2 and its max-tree in icicle representation. One component/node has been selected (red).

To sum up, icicle representations clearly show the nesting relationship, the size, the gray level, and other custom attributes of components of an input image. Having this compact and well-organized representation, we are now ready to perform image manipulation. The proposed interactive image editing tool is the combination of a high-level global manipulation and a more detailed deformation of local components.

8.3.2 Global manipulation

Global manipulation, also considered to be inter-node manipulation, generally includes removing or adding single or multiple nodes or CCs,
which is useful for applications such as image segmentation, local luminance changes, and watermark removal (Sec. 8.4.1). Node removal can be implemented manually or algorithmically, as shown in Fig. 8.7 and described next.

Manual removal refers to the user directly selecting the node or component that one wants to cut out in the interface, and then clicking the node removal icon (B). Since the image components in the left window and the nodes in the right window are associated, one can select the part one wants to remove by either directly clicking the node in the icicle tree or the component on the image. In both ways, one can select multiple nodes by holding down the shift key. The selection-and-deletion operation is straightforward and convenient. Yet, this operation can be cumbersome when there are plenty of nodes to be deleted. To address this, we added a function to select all descendant nodes of the currently selected node by clicking icon A. Figure 8.7 (a) illustrates this by manipulating a simple art deco image. Deleted nodes are set to translucent. As visible, by discarding all descendant nodes of node D, their corresponding components (the rightmost petals in multiple levels) on the image are also eliminated, which indirectly achieves the effect of local

Figure 8.7: Removal of nodes manually (a) and algorithmically (b).
brightness changes. The operation of restoring a node is similar to deleting one. By selecting a deleted (translucent) node and then clicking the node inclusion icon (C), one can restore the node. However, one cannot add nodes that did not exist in the original tree.

**Algorithm-based removal** aims to set the number of layers $L$ in the parameters setting area, and then run the SDMD method, so that the method will select and retain the most representative $L$ layers using the *cumulative histogram* layer selection scheme (see details in Sec. 3.3.1). Figure 8.7 (b) shows an example. The original image contains tens of level sets or layers. Although not easily visible, there are various grayscale values at the edges of the petals. By setting $L$ to 3, the method selects the three most informative layers. As can be seen from the results in the left window, almost no important information is removed. The algorithmic removal method can only preserve or remove all nodes of a certain layer. Still, in combination with the manual operations described above, more refined global manipulation can be achieved.

Global manipulation is useful for image segmentation (Pal and Pal, 1993; Pham et al., 2000; Zaitoun and Aqel, 2015). Figure 8.8 illustrates a skull-stripping segmentation (Hahn and Peitgen, 2000; Doshi et al., 2013) by showing several components (A, A1, A2, B, C) of a magnetic resonance (MR) image (Fig. 8.8 (a)) reconstructed from several nodes and their corresponding descendants. As visible, the whole brain (A), including the brain stem (A1) and cerebellum (A2), as well as the parotid tissue (B) and nasal tissue (C), are successfully segmented. Moreover, our proposed method is fast and does not require any preprocessing such as intensity normalization or denoising.

**Figure 8.8:** Skull-stripping segmentation. An MR image (a), its icicle tree (b), and several components (A, A1, A2, B, C) of the image.
8.3.3 Local manipulation

Local manipulation is also seen as per-node deformation and mainly refers to deforming a single connected component by manipulating its spline control points (CPs). Section 5.5.3 has illustrated several preliminary operations, including moving, adding, and removing CPs and increasing or decreasing the degree of the spline representing the medial axis transform. In this section, we further expand this idea by presenting more functions. We use the node D in Fig. 8.7 (a) as an example to introduce our user interface. By selecting node D and then clicking the icon to the right of icon C, we open the user manipulation interface, as shown in Fig. 8.9. We next introduce one by one all the tools we propose for this task.

![User interface for detailed spline manipulation.](image)

**Displaying all CPs:** By clicking icon A, all CPs of the current component are shown in the manipulation interface. Each component has one or several skeleton branches, thus resulting in one or more splines. Control points (CPs) on the same spline are connected by lines of the same color, which indicates the degree of the spline, as shown at the bottom of the interface. In contrast to icon A, the function of icon B is to make all CPs invisible.

**Changing the radius/degree:** When the mouse hovers over a CP, its radius size is displayed, and the radius value is also updated in the K area. When a point is clicked, it is highlighted in blue. Then, when holding down the `shift` key and scrolling the mouse wheel, the radius (both the graphical representation and the actual value) changes accordingly. The operation of modifying the degree is similar, except that the `shift` key has to be replaced with the `D` key.
Adding a CP to the spline: Icon C allows users to add a CP to the spline. Note that the clicked point needs to fall in the (invisible) rectangle formed by any two consecutive CPs of the spline. Otherwise, a new spline (with two CPs) will be created upon such a click.

Removing CPs in a spline: The user is allowed to remove one or more CPs in a spline by pressing icons D or F. One can also delete the entire spline via icon E.

Rotating/scaling CPs: Icon G is used for rotating all selected CPs. After clicking this icon, one first needs to select a rotation center, then select the CPs to be processed by dragging the displayed rubber band marker with the mouse. Next, one can hold down the \( R \) key and scroll the mouse wheel to specify the desired rotation angle. The scaling function is similar, except that icon H and the \( R \) key have to be replaced with icon I and the \( S \) key.

Copying/cutting CPs: These two functions are similar. First, the user can select one or more CPs, then press the \( C/X \) key, then click somewhere for the CP(s) to be pasted and press the \( V \) key to effectuate the actual CP pasting.

Reconstruction: Icons I and J are used to reconstruct the manipulated component and the whole image, respectively. The changed splines are first rasterized on the desired pixel grid to generate the manipulated skeletons using the method described in Sec. 5.3.4. Then, we reconstruct the component with the medial discs envelope method, as described in Sec. 2.1.4.

Figure 8.10 shows the manipulation of a shadow puppet character, in which most of the above operations are covered, including deleting CPs (−c), decreasing radius values (−r), and moving (M), rotating (R), scaling (S), and copying (C) control points. We start by making the figure head smaller by pressing icon H, next selecting all the CPs that represent the head, and then scrolling the mouse wheel down to adjust them to the appropriate size. Then we move all CPs down slightly to make the result more realistic. For the left arm, we intend to separate the hand from the body. For this, we first delete the selected spline A (with 5 CPs) in Fig. 8.10 (a), then rotate the arm clockwise by about 30 degrees, and next copy the right hand into the left side and rotate it by the suitable angle. We also decrease the radius of both CPs of the spline at the elbow by 8 pixels. In addition, the left leg and right arm of the character are also rotated by about 20 degrees clockwise and counterclockwise, respectively.

8.4 Applications

In the previous section, we introduced our proposed interactive image editing tool. Section 8.3.2 introduced several schemes for selecting multiple icicle nodes. Section 8.3.3 demonstrated the internal manipulation
of a single node. Combining the two, i.e. to move, scale, rotate, remove, and paste multiple nodes at once, we enable more interesting and powerful applications of our method, as detailed next.

8.4.1 Visible watermark removal

Visible watermarks are widely used in images and videos to protect copyright ownership. Analyzing watermark removal helps to strengthen the anti-attack techniques in an adversarial way, which attracted increasing attention and became a hot research topic (Huang and Wu, 2004; Pei and Zeng, 2007; Cheng et al., 2018; Qin et al., 2018). Due to the uncertainty of the size, shape, color, transparency, and location of watermarks, developing an automatic visible watermark removal method remains a difficult task. Some techniques even require user-guidance (Huang and Wu, 2004; Pei and Zeng, 2007) or assume that test images have the same watermark region (Xu et al., 2017). Our image manipulation tool provides a way to remove watermarks, but in an interactive way, rather than automatically like the methods mentioned above. Our proposed method is very simple. We first select watermark-related nodes through the several schemes introduced in Sec. 8.3.2. Then we enter local manipulation (Sec. 8.3.3), and press icon E in Fig. 8.9 to delete all control points associated to the watermark. Two manipulation demonstrations are available in the supplementary material (Wang et al., 2022).

Figure 8.11 shows the results of our method on six watermarked images. As can be seen from the three grayscale images, our method works perfectly not only for images where the embedded watermark is brighter than the surrounding area (Fig. 8.11 (b1, f1)), which can be easily manipulated with the selection-and-deletion scheme described in Sec. 8.3.2, but also for images where the embedded watermark has a
similar or lower intensity to the surrounding area (Fig. 8.11 (d1)). Our proposed tool also yields good results for color images (a2, c2, e2) by manipulating their three components, e.g. YUV, independently. However, since the manipulated image is reconstructed from the skeletons, this watermark removal method also has the common drawbacks of the previously described compression methods (Chapter 3–7), i.e. it cannot deal optimally with images with many thin and small-scale details, such as plants on the mountains (b2, e2) and animal fur (c2, d2). Apart from that, our method cleanly removes watermarks without introducing artifacts to the input images.

Figure 8.11: Example results on watermark removal. (a1)–(f1) Watermarked images. (a2)–(f2) The results of removing the watermark with our method.

8.4.2 Image deformation

The previous section has shown how to achieve good watermark removal performance by combining the multiple node selection in the global manipulation (Sec. 8.3.2) with the CPs deletion in the local manipulation (Sec. 8.3.3). In this section, we combine the node selection with deletion and addition of CPs in local manipulation to implement image deformation.

Figure 8.12 illustrates an example by showing several steps to remove glasses from a cartoon avatar. We first remove the glasses’ lenses (step 1) by deleting nodes A, B, and C in Fig. 8.12 (b2) by pressing icon B in Fig. 8.7 (a). Then we eliminate the glasses’ frame (step 2) by selecting node D and all its descendants in (b2), entering the local manipulation interface (c2), and deleting all CPs related to the frame shape (region E in c2). Step 2 produces a very light eye contour (see (c1)), so in the next
step, we aim to darken the eye outline (step 3). We select only node D in (b2) and enter the user interface (d2). Then we use the CP adding function (icon C in Fig. 8.7 (a)) to put in several splines and manipulate their CPs to form the eye contour, as shown in region F in (d2). Now we successfully remove the glasses from the original image (a1) and generate a reasonable result (d1).

Figure 8.12: Key steps for removing glasses from a cartoon avatar (a1). Images (a2) and (b2) show icicle trees of (a1) and (b1), respectively. Image (c2) shows the corresponding components and control points of node D and its descendants, while (d2) shows those of node D only.

Figure 8.13: Five different facial changes (b–f) generated from the manipulation of the original image (a).

Following the same idea, we further generate four other facial changes, as shown in Fig. 8.13. To generate (c), we first remove all CPs
related to the glasses and the eyes. Then we add four splines to represent the smiling eyes. The remaining ones use similar operations. We first delete control points that represent the smiling mouth in (a), then we add the new mouth (d), mustache (e), and beard (f) in turn by adding new splines.

8.4.3 Other applications

In this section, we further combine the node selection in the global manipulation (Sec. 8.3.2) with more features in local manipulation (Sec. 8.3.3), including scaling, moving, rotating, cutting and copying CPs, to implement additional applications.

**Simulation of illumination changes** can be achieved with our method. Figure. 8.14 shows two examples to illustrate the potential of our tool to simulate light changes. The simulations of light sources approaching (b2), moving away (b2), offsetting (a3,b3), and removing (a4,b4) can be implemented by scaling up, scaling down, moving or rotating, and removing, respectively, all control points in all nodes representing the white highlight on the three components (YUV) of the tomato (a1) or the copper ball image (b1). The manipulation demonstration is available in the supplementary material (Wang et al., 2022). While the proposed image editing tool handles these simple objects with ease, we do not claim that our tool does already have a well-established relighting mechanism for dealing with very complex objects.

![Images of tomato and copper ball with lighting changes](image_url)

Figure 8.14: Examples of simulating lighting changes. Original images (a1,b1). Simulations of light sources approaching (a2), moving away (b2), offsetting (a3,b3), and removing (a4,b4).

**Image rearrangement** is also easily implemented using our method. Figure 8.15 shows an example by rearranging the birds’ positions. We
start by cutting the CPs that encode the two birds on the far right (D, E) and pasting them into an empty space S. Then we move the CPs representing birds B and C to the far right. Next, we move the CPs in S to the second and third positions. We end up using the rotate function (icon G in Fig. 8.9) to rotate bird D by about 30 degrees clockwise; see the demonstration in the supplementary material (Wang et al., 2022).

![Image](image.png)

Figure 8.15: Image rearrangement example. (a) The original image and (b) the manipulated result. We swap the position of two birds on the far right (D, E) and the two birds in the middle (B, C) and rotate bird D.

**Clothing design** can also be executed with our tool. Figure 8.16 gives two examples. For Fig. 8.16 (a), we intend to change the long-tube shoe to a short-tube one, and the thick sole to be thinner. For this, we first eliminate CPs that encode the long tube, and then reduce the radius of all CPs of the splines representing the sole of the shoe by about 1/3. For Fig. 8.16 (b), we simply add two stripes to the T-shirt by adding several splines encoding the stripes. Note that the added stripes should be put in the same position for the three components, i.e., YUV, of the color image, otherwise false colors are produced; see Sec. 8.5. As visible, our approach generates images of good quality and fidelity.

![Image](image.png)

Figure 8.16: Clothing design examples. (a) Making a long tube, thick-soled, shoe into a short tube thin-soled shoe. (b) Adding two stripes to a T-shirt.
8.5 DISCUSSION

In this section, we discuss several aspects of our interactive image manipulation tool.

Ease of use: Our interface is straightforward and arguably user-friendly. There are only two windows: The left one is the image display interface while the right one shows the icicle tree; see Fig. 8.6 and Fig. 8.7, respectively. The two windows are linked, e.g., when one clicks an icicle node on the right, its corresponding component is highlighted on the left, as shown in Fig. 8.6, and vice versa. When the spline manipulation icon is clicked, one enters the local manipulation interface (Fig. 8.9), which is also shown in the left window. The spline manipulation operations, which are easy to handle, are detailed in Sec. 8.3.3. Yet, we admit that not all the operations are intuitive, e.g., adding splines to form the eye shape in Fig. 8.12 (d2) is not yet fully straightforward. Besides, for some more complex shape manipulations, such as the horse manipulation example in the supplementary material (Wang et al., 2022), some practice and experience with our tool is required to achieve that manipulation easily. Apart from that, removing, moving, rotating, scaling, cutting, and copying CPs are all easy to conduct for inexperienced users.

Running time: The longest (slowest) step in our end-to-end pipeline is the encoding process at the beginning. This is so since all the image information needs to be encoded, including all components on all layers. Take the original image in Fig. 8.8 as an example, whose intensities span from 0 to 255. Although its resolution is only $320 \times 320$ pixels, encoding all the information, i.e., computing skeletons, and running spline fitting for all the components, takes 118 seconds on a commodity PC. Fortunately, the coding operation only needs to be executed one time. Once the encoding process is complete, the subsequent series of operations, whether deleting, adding icicle nodes, all the manipulation operations on CPs, or reconstructing the components or images, occur in real-time. In practical applications, for inexperienced users, it takes approximately 2 to 4 minutes to successfully remove the watermark in an image or achieve a reasonable image deformation result. As for image rearrangement, simulation of lighting changes, and clothing design, it takes around 1 to 2 minutes to perform.

Replicability: We provide our full source code and data as well as all demonstration videos in the supplementary material (Wang et al., 2022) for replication purposes.

Limitations: Our interactive tool is not yet able to handle the three components of color images simultaneously, but only facilitates their manipulation separately. This leads to a problem where false colors, ghosting, and artifacts can occur when changes to the three components do not coincide. Figure 8.17 shows two examples. For (a), when
the added stripes for the Y component and U component of the right image in Fig. 8.16 are in different positions, ghosting is introduced, as indicated by the arrow. Similarly, in (b), when the added eye shapes in the three components of the image in Fig. 8.13 (c) do not coincide, artifacts and false colors are produced; see where the arrow points. However, we argue that these ghosting and false colors have perceptually little impact, and they can be avoided with careful manipulation.

![Figure 8.17: Ghosting, false colors, and artifacts are introduced when changes to the three color components do not coincide.](image)

8.6 Conclusion

In this chapter, we have presented a novel interactive image manipulation tool, which combines our spline-based medial axis for shape manipulation (Sec. 5.5.3) with an icicle representation of component trees. Dealing with component trees instead of threshold sets allows finer-grained spatial control of each level set. We have demonstrated how to operate our tool in detail in Sec. 8.3. To verify the effectiveness of our tool, we have illustrated it by several applications of editing real-world images. Only manipulating icicle nodes globally (Sec. 8.3.2), such as removing multiple nodes in the icicle tree, achieves simple watermark removal tasks (Fig. 8.11 (a, e)). When adding local spline manipulations, more interesting applications are achieved. When the global manipulation is combined with the control points deletion function, more complex watermark removal tasks are achieved (Fig. 8.11 (b, c, d, f)). When combined with removing and adding CPs, interesting image deformation is achieved (Sec. 8.4.2). We have also combined the global manipulation with more features in local manipulation, including scaling, moving, rotating, cutting and copying CPs, to implement the simula-
tion of illumination changes, image rearrangement, and clothing design (Sec. 8.4.3).

Several future work directions are possible. First, more functions can be added to our tool. One possibility is to allow to open two images simultaneously and stitch their content, such as seamlessly stitching objects from one image into the background of another image. Some additional functions, such as allowing users to directly move the node up or down in the icicle plot interface to achieve the intensity change of that node, can also be considered. Secondly, improving our tool to process color images more perfectly is also important to study. Finally, we aim to explore the potential of our tool for more applications in the future, such as image smoothing and image abstraction.
CONCLUSION

In this chapter, we conclude this thesis by revisiting the central research question proposed in Chapter 1, which we repeat here to assist the reader:

*Are dense skeletons, represented suitably by raster or vector models, effective and efficient tools for image compression and manipulation?*

We believe that we can answer this question positively. To support this assertion, we summarize our research results and contributions in Sec. 9.1. Subsequently, we give recommendations for future work in Sec. 9.2.

9.1 SUMMARY OF RESEARCH RESULTS

The research question shown above comprises two parts. The first part concerns image compression while the second part is about image manipulation. Chapters 3–7 have addressed the first part of the question extensively and exhaustively, and drawn the conclusion that dense skeletons, modeled by raster or vector representations, are effective and efficient tools for image compression. We elaborate this as follows.

Chapter 3 presented Compressing Dense Medial Descriptors (CDMD) and showed, for the first time, that medial descriptors offer interesting and viable possibilities to compress grayscale and color images. Yet, this chapter also shows that the generated results vary greatly depending on the type of images studied.

In Chapter 4, we have adapted CDMD to use the information provided by spatial saliency maps to selectively simplify and encode an image while preserving its salient regions. We have shown that the spatial saliency DMD (SSDMD) improved the trade-off between compression ratio (CR) and image quality as compared to CDMD while preserving small-scale, but visually important, details of natural images. SSDMD has also been shown to be useful in improving the compression of standard JPEG though yielding slightly lower quality. As such, Chapter 4 is a small step forward towards proving our claim of effectiveness of dense skeletons as a tool for image compression.

Chapters 3 and 4 have shown that dense skeletons represented by raster models are effective tools for image compression. However, these raster representations of dense skeletons have an inherent limitation: They have a built-in redundancy that makes them able to improve the compression ratio of the images they represent only up to a given limit.
Chapter 5 largely removes this limit by dropping the raster representation of dense skeletons in favor of a vector representation. We used the compact and accurate piecewise B-splines to encode skeletons and used these skeletons for binary image compression. The comparison of our Spline-based Medial Axis Transform (SMAT) to a recent related approach (Zhu et al., 2014) has shown the advantages of our proposal in terms of simplicity, ease of use, versatility, and computational speed. Compared with the regular MAT raster representation (as provided, for example, by CDMD), SMAT yields a much higher CR at the expense of a slightly lower image quality. We also leveraged the continuity of the vector representation to create super-resolution versions of binary images.

In Chapter 6, we have extended the vector representation proposed in Chapter 5 beyond binary shapes, to encode grayscale and color images. We have shown how our Spline-based DMD (SDMD) performs on a wide variety of image types and demonstrated that it leads to better trade-offs of image quality vs CR than JPEG and, for certain image types, also than JPEG 2000. We also show how SDMD can generate super-resolution results from color and grayscale images and that our super-resolution results are comparable with those created by state-of-the-art deep learning-based approaches.

Chapter 7 incorporated the saliency maps used in Chapter 4 into the SDMD pipeline introduced in Chapter 6 and presented Spatial Saliency Spline Dense Medial Descriptors (3S-DMD) for saliency-aware image simplification and compression. A quantitative evaluation has shown that 3S-DMD greatly improves the compression of SDMD at only a small quality loss. We also demonstrated that our method achieves both higher CR and better quality than JPEG for a broad range of images. While 3S-DMD cannot reach the same high quality at the same CR values as JPEG 2000, it yields similar quality, higher CR, and fewer artifacts for a wide class of synthetic images.

The chapters discussed above have explored the possibilities of dense skeletons, represented suitably by raster or vector models, using saliency maps or not, for lossy image compression. Chapter 8 turned to the second half of the main research question, i.e., using dense skeletons represented by B-splines for image manipulation. Various application examples, including watermark removal, image deformation, lighting change simulation, image rearrangement, and clothing design showed that spline-based dense skeletons, when combined with morphological trees, are effective and efficient tools for manipulating grayscale or color images.

9.2 Future Work

We next illustrate several potential directions for future work from the perspective of several applications.
**Image compression:** Chapters 3–7 have extensively explored image compression using dense skeletons represented by raster or vector models. Yet, there is still room for improvement. From the point of view of improving quality, one can further optimize the coordinates of B-spline control points. Related work Zhu et al. (2014) has implemented this optimization by applying the L-BFGS method (Liu and Nocedal, 1989), an iterative quasi-Newton method, to minimize the distance between the envelope of the union-of-discs $\partial \tilde{\Omega}$ (Sec. 2.1.4) and the original boundary $\partial \Omega$ of shape $\Omega$. In terms of improving compression, it is also possible to extend the way of storing skeletons to a video format. This provides another source of information for skeleton data encoding, which can take into account information between temporally close frames in addition to intra- and inter-layer encoding. Separately, in terms of evaluation, one possible direction is to consider more extensive comparisons with additional compression techniques, e.g., deep neural network methods.

**Image manipulation:** As mentioned in Sec. 8.6, several extensions of our image manipulation tool can be considered, including a better handling of color images; exploring more interesting manipulation applications, e.g., image smoothing and local luminance changes; and investigating more functions in the tool, e.g., stitching two images. A different but equally important direction is, as with any technique that addresses end users, to perform an actual evaluation involving controlled experiments with specified tasks and measure the precision, time-to-completion, and user satisfaction offered by our method.

**Image abstraction** has attracted a lot of research over the years (DeCarlo and Santella, 2002; Kyprianidis and Döllner, 2008; Henry Kang, 2009). Such applications involve, among others, the use of Non-Photorealistic Rendering (NPR) techniques. These methods give prominence to semantic edges while filtering out image details, presenting abstract visual information of the original image. NPR techniques aim to serve various artistic purposes, i.e., to elicit an aesthetic response from the public, which include painting (Hegde et al., 2013), mosaics (Hausner, 2002), and cubist rendering (Collomosse and Hall, 2003). By using a small layer threshold $L$, and a large island threshold $\varepsilon$ and saliency threshold $\sigma_0$ (Sec. 3.2.2), we believe that our dense-skeleton image-coding method can achieve good image abstraction. We also believe that the effect can be further enhanced by overlaying the simplified images with line drawing extracted by edge detection, e.g., using the Canny edge detector (Canny, 1986) or the difference-of-Gaussians (DoG) filter (Gooch et al., 2004). This opens a range of avenues for dense skeletons to be used for NPR manipulation of images.

**Corner detection** is an interesting task used for many applications, including image matching (Rosten et al., 2010), camera calibration, and motion estimation (Mokhtarian and Mohanna, 2006). Over the years, many handcrafted corner detectors have been presented, including SU-
SAN (Smith and Brady, 1997), FAST (Rosten and Drummond, 2005), Harris (Harris and Stephens, 1988), and some recent neural network approaches, e.g., TILDE (Verdie et al., 2015) and DNet (Lenc and Vedaldi, 2016). Apart from these, skeletons can also be a useful tool for corner detection. Dinesh and Guru (Ramegowda and Guru, 2006) proposed a corner detector for binary shapes $\Omega$ by searching for the intersections of the skeleton $S_\Omega$ and the boundary $\partial \Omega$. Renders (Renders, 2021) detected corners in grayscale images by finding the endpoints of dense skeletons, following several ideas based on our own research. This work can be further improved by assigning each corner point a *cornerness* value (Renders, 2021), which can be obtained from the saliency value $\sigma$, importance metric $\rho$ (Sec. 2.1.3), or by using image derivatives and gradients. Additionally, using the cumulative histogram scheme explained in Sec. 3.3.1 to select relevant layers instead of naively choosing the layers uniformly as in Renders (2021) can further improve the performance and reduce running time. Exploring all these directions could likely lead to perceptually better, more noise-robust, and thus practically useful, corner detection methods.

In addition to the applications described above, dense skeletons are also worth exploring in terms of inverse halftoning (Kim and de Queiroz, 2003; Li et al., 2020), image denoising (Tian et al., 2020), and salient region detection (Goferman et al., 2011; Li et al., 2013). One step further, our dense skeleton methods could be extended to simplify 2D and 3D scalar fields in scientific visualization applications. To finally conclude this thesis, we see a wide range of applications of dense skeletons, thereby supporting our overall claim that these are a valuable addition to the toolkit of image processing researchers.
ACKNOWLEDGMENTS

It is time for a happy end to my four-year journey. I would like to thank everyone I have met on this journey. First and foremost, I would like to express my deepest appreciation to my supervisors, Prof. Dr. Alexandru C. Telea and Prof. Dr. Jiří Kosinka. It is hard for me to imagine how I would have completed my PhD project without your consistent help and encouragement.

Professor Telea, I have always been grateful that you have given me the opportunity to pursue my PhD under your supervision even though I have no background in computer science. Your boundless enthusiasm for scientific research and the magic of making complex issues easy to understand have deeply infected me. I am also very impressed with your extensive knowledge of a broad range of subjects and languages. Whenever I come across an obscure task and do not know where to start first, you always help me lay out step by step how to execute it, which really helps me a lot. I would like to say that you are an outstanding supervisor and a very good leader, I do not know how lucky I am to find such a good mentor as you.

I also would like to express my special thanks to professor Kosinka. Thank you very much for your good ideas, careful revisions, and very useful comments on my work. Thank you for always making sure that all my processes go according to the plan and for holding our regular discussion every two weeks. In addition, I am very impressed with your professional knowledge in computer graphics and your meticulous attitude toward scientific research. At last, I would like to show my gratitude to both of you for considering my feelings and allowing me to return to China to write the thesis in the last few months of my PhD during the COVID time, which gives me great spiritual comfort.

Thanks to my colleagues from the SVCG group, Lingyun, Zizhao, Xiaokun, Chengtao, Xingyu, Gerben, Lorenzo, Youngjoo, Jun, Mateus, Caio, Hamid, Steffen, Heejun, Dennis and Kuanhao. I also thank colleagues from the VIG group in Utrecht, Samuel, Eduardo, Xiaorui, Zonglin, Diego, Jiao, Yuncong, and Boyu. I enjoyed my time with you.

Finally, thanks to all my family and relatives who have been so kind to me. I would like to mention, especially, my dear partner Jiamin Xie. Thank you for always being there for me and for coming to the Netherlands to accompany me at all costs, making my daily life happy during the hard COVID time.

The research conducted in this thesis was financed by China Scholarship Council and by the University of Groningen. My sincere thanks to these organizations for their support.
BIBLIOGRAPHY


V. Ayzenberg. Skeletal descriptions of shape provide unique perceptual information for object recognition. Scientific Reports, 9, 06 2019.


162


A. Telea. CUDASkel: real-time computation of exact euclidean multiscale skeletons on CUDA, 2019. [webspace.science.uu.nl/~telea001/Shapes/CUDASkel](webspace.science.uu.nl/~telea001/Shapes/CUDASkel).


COLOPHON

This document was typeset using the typographical look-and-feel classicthesis developed by André Miede. The style was inspired by Robert Bringhurst’s seminal book on typography "The Elements of Typographic Style". classicthesis is available for both \LaTeX and \LyX:

http://code.google.com/p/classicthesis/

Final Version as of July 13, 2022 (classicthesis).