Compact image vectorization by stochastic approaches

Jean-Dominique Favreau

UNIVERSITÉ CÔTE D'AZUR

Inria informatics mathematics

UNIVERSITY COTE D'AZUR

DOCTORAL SCHOOL STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

P H D T H E S I S

to obtain the title of

PhD of Science

of the University Côte d'Azur

Specialty: Automatic and image and signal processing

By Jean-Dominique FAVREAU

Compact image vectorization by stochastic approaches

Thesis Advisors: Florent LAFARGE and Adrien BOUSSEAU

prepared at INRIA Sophia Antipolis

defended on March 15th, 2018

Jury:

Reviewers:	Yotam GINGOLD	-	George Mason University
	Stefanie HAHMANN	-	Grenoble INP - Ensimag
	Joelle THOLLOT	-	Grenoble INP - Ensimag
Examinators:	Xavier DESCOMBES	-	Inria, University Cote d'Azur
	Holger WINNEMOELLER	-	Adobe Systems
Advisors:	Adrien BOUSSEAU	-	Inria, University Cote d'Azur
	Florent LAFARGE	-	Inria, University Cote d'Azur

Abstract

Artists appreciate vector graphics for their compactness and editability. However many artists express their creativity by sketching, painting or taking photographs. Digitizing these images produces raster graphics. The goal of this thesis is to convert raster graphics into vector graphics that are easy to edit.

We cast image vectorization as an energy minimization problem. Our energy is a combination of two terms. The first term measures the fidelity of the vector graphics to the input raster graphics. This term is a standard term for image reconstruction problems. The main novelty is the second term which measures the simplicity of the vector graphics. The simplicity term is global and involves discrete unknowns which makes its minimization challenging. We propose two stochastic optimizations for this formulation: one for the line drawing vectorization problem and another one for the color image vectorization problem. These optimizations start by extracting geometric primitives (skeleton for sketches and segmentation for color images) and then assembling these primitives together to form the vector graphics. In the last chapter we propose a generic optimization method for the problem of geometric shape extraction. This new algorithm does not require any preprocessing step. We show its efficiency in a variety of vision problems including line network extraction, object contouring and image compression.

Keywords: Sketch vectorization, multilayer image vectorization, Bézier curves, Delaunay Point Process, Monte Carlo Sampling

Résumé

Les artistes apprécient les images vectorielles car elles sont compactes et facilement manipulables. Cependant, beaucoup d'artistes expriment leur créativité en dessinant, en peignant ou encore en prenant des photographies. Digitaliser ces contenus produit des images rasterisées. L'objectif de cette thèse est de convertir des images rasterisées en images vectorielles qui sont facilement manipulables.

Nous avons formulé le problème de vectorisation comme un problème de minimisation d'énergie. Nous avons défini une énergie composée de deux termes. Le premier terme mesure la fidélité de l'image vectorielle générée avec l'image rasterisée d'origine. Ce terme est un terme classique en reconstruction d'image. La nouveauté principale est le second terme qui mesure la simplicité de l'image vectorielle générée. Le terme de simplicité est global et contient des variables discrètes, ce qui rend sa minimisation difficile. Nous avons proposé deux algorithmes de vectorisation: un pour la vectorisation de croquis et un autre pour la vectorisation multicouches d'images couleurs. Ces deux algorithmes commencent par extraire des primitives géométriques (un squelette pour les croquis et une segmentation pour les images couleurs) qu'ils assemblent ensuite pour former l'image vectorielle. Dans la dernière partie de la thèse, nous proposons un nouvel algorithme qui est capable de vectoriser des croquis sans étapes préliminaires: on extrait et assemble les primitives simultanément. Nous montrons le potentiel de ce nouvel algorithme pour une variété de problèmes de vision par ordinateur comme l'extraction de réseaux linéiques, l'extraction d'objets et la compression d'images.

Mots cléfs: Vectorisation de croquis, vectorisation multicouche d'images, courbes de Bézier, Processus Ponctuels de Delaunay, échantillonnage de Monte Carlo

Contents

Chapter 1

Introduction

1.1 Context

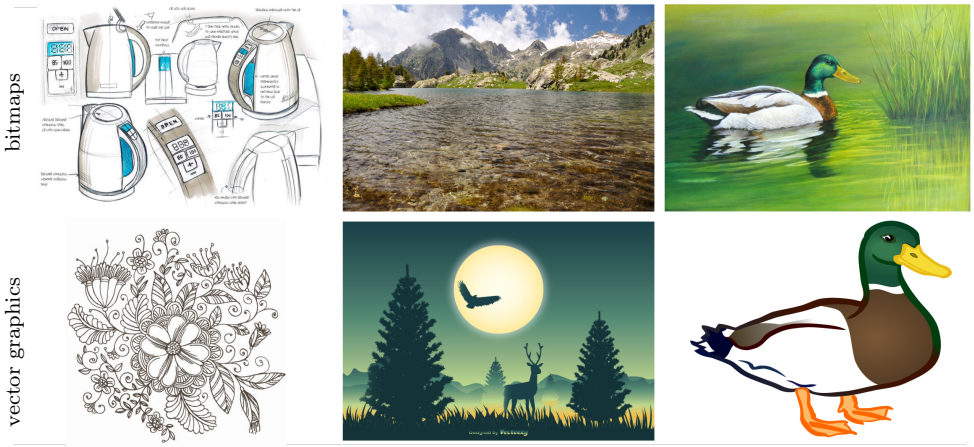


Figure 1.1: Raster graphics versus Vector graphics: two ways of representing images numerically. The first row illustrates several types of raster graphics (ie bitmaps): sketches, photographs, paintings. These bitmaps are composed of an array of pixels and were captured either by cameras or scanners. The second row shows three types of vector graphics: line drawings, wallpapers, cliparts. They are composed of parametric shapes. Vector graphics look cleaner and simpler than raster graphics. (images from clipart.me, vecteezy.com, openclipart.org)

In computer graphics, there are two ways to represent images: raster graphics, also called bitmaps, and vector graphics (Figure 1.1).

Bitmaps represent images as arrays of pixels. Bitmaps are easy to capture with cameras and easy to display on screens. Sensors of digital cameras capture images as a grid of color measurement converted into pixels and screens are manufactured as arrays of pixels. The main drawback of bitmap is editing: to modify a bitmap, individual pixels have to be modified. The second drawback is resolution: a bitmap has a fixed amount of pixels. If we

reduce the amount of pixels, there is no way to recover the initial bitmap perfectly.

A vector graphics is a mathematical representation of the content of an image using parametric curves or parametric shapes. Vector graphics are more difficult to create than bitmaps but they are easier to edit due to their compact representation. Vector graphics do not suffer from resolution limitation. We present here these advantages.

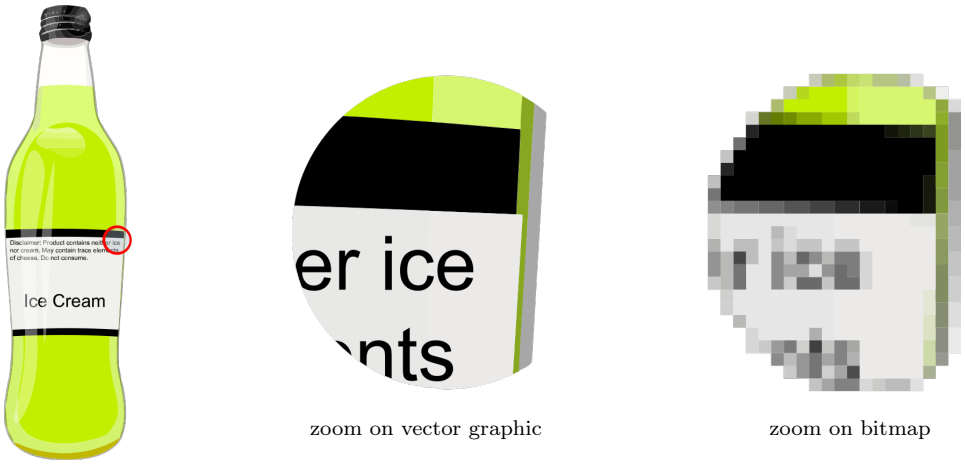
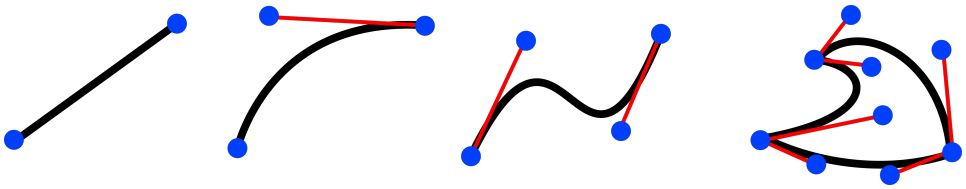


Figure 1.2: Zoom comparison between vector graphics and bitmaps. Vector graphics allows user to zoom-in indefinitely while keeping sharp contours of objects. With vector graphics, we can see clearly the text of the cropped area whereas with bitmap the text is blurred due to resolution limitation (image from wikipedia).

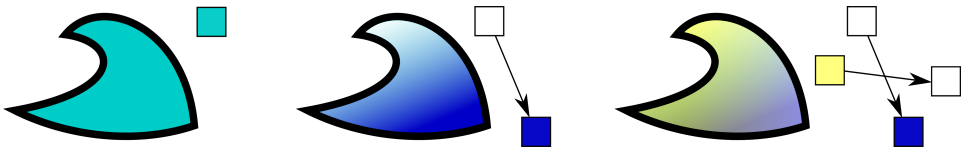
Resolution independent: A vector graphics is a continuous mathematical representation of the content of an image whereas a bitmap is a discretized version of the same image. Consequently, zooming on a bitmap produces a blurred image due to limited frequency content, while zooming on a vector graphics always produces sharp details (Figure 1.2).

Compactness: Vector graphics only store the coordinates of the control points defining the parametric curves and the color functions defining the colors of shapes. Parametric curves are defined by a convex combination of their control points. This combination is typically defined using polynomial basis. The Bernstein polynomial is the basis of Bézier curves, a parametric curve which was studied by the mathematician Paul de Casteljau in



(a) linear curve (b) quadratic curve (c) cubic curve (d) composite curve

Figure 1.3: Standard curves in vector graphics. A line (a) is defined by its two extremities. We obtain a quadratic Bézier curve (b) by adding a point. The red edge is tangent to the curve. Its length defines the curvature of the Bézier curve. The cubic Bézier curve (c) is defined by four control points. The composite Bézier curve in (d) is composed of three cubic Bézier curves. It forms a closed path that bounds a shape.



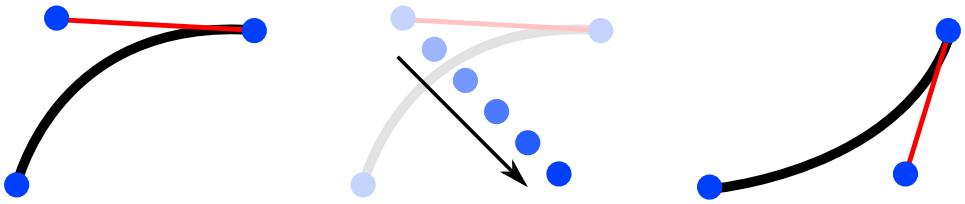
(a) constant color (b) linear gradient color (c) quadratic color

Figure 1.4: Color functions specified inside a shape. The color function in figure (a) is a constant color. The shape in figure (b) is filled with a linear color gradient. The two squares contain the colors defining the gradient. The arrow specifies the direction of the gradient of the color function. The quadratic color in figure (c) was obtained using two semi-transparent linear color functions.

1959 [DC63] and was promoted by Pierre Bézier in 1962 in the context of automotive design. In computer graphic, we generally use lines defined by two control points, quadratic Bézier curves defined by three control points, and cubic Bézier curves defined by four control points. A composite Bézier curve, also called path, is composed by a set of cubic Bézier curves sharing their extremities.

A shape is defined by a closed path. A color function can be defined inside shapes. The standard color functions are constant color, linear gradient and radial gradient. More complex color functions can be created by superposing several semi-transparent color functions. Figure 1.3 illustrates the standard vector graphics curves and Figure 1.4 illustrates several types of color function inside shapes. We discuss in chapter 2 more advanced color functions defined by gradient meshes [SLWS07] and diffusion curves [OBW+08].

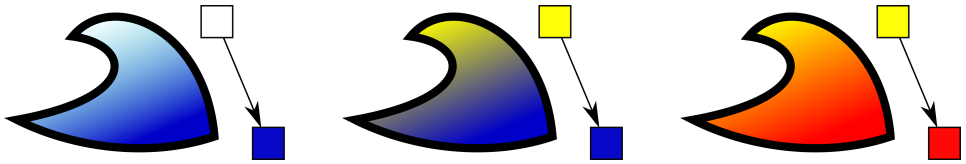
Editability: To modify a vector graphics, the user has to manipulate control points and color functions rather than individual pixels. In Figure 1.5,



(a) initial curve (b) moving a control point (c) final curve

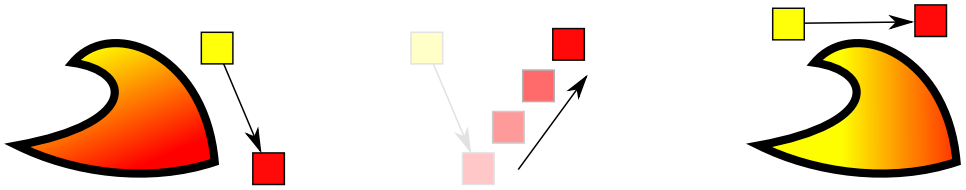
Figure 1.5: Curve editing. Modifying the curve requires only to move one control point instead of moving all the pixels of the curve.

the appearance of the curve is modified by moving only one control point. The position of the pixels of the curve are automatically recomputed.



(a) initial color gradient (b) editing the first stop (c) editing the second stop

Figure 1.6: Color editing. The color inside a shape is defined by a two stops gradient color. Modifying the color of the two stops will automatically modify the color of every pixels inside the shape.



(a) initial gradient direction (b) editing the direction (c) final gradient direction

Figure 1.7: Color gradient direction editing. To modify the direction of the color gradient, users only have to click on a stop of the gradient and move it.

Figures 1.6 and 1.7 illustrate editing of the color function. The colors of the two stop gradient are modified in the first figure and the direction of the gradient is modified in the second figure. All pixel colors are recomputed instantaneously. To make such editing on a bitmap, the user has to change the color of every pixel individually or resort to interactive image segmentation and color histogram manipulation to achieve the desired effect.

Vector artists often create compact line drawings by tracing few, long

curves rather than multiple short segments. Figure 1.8 shows a junction between several curves. Artist often create T-junctions as one curve touching a long curve rather than 3 curves meeting at a point. Similarly, artists design X-junctions as two curves intersecting each other rather than four curves meeting at a point.

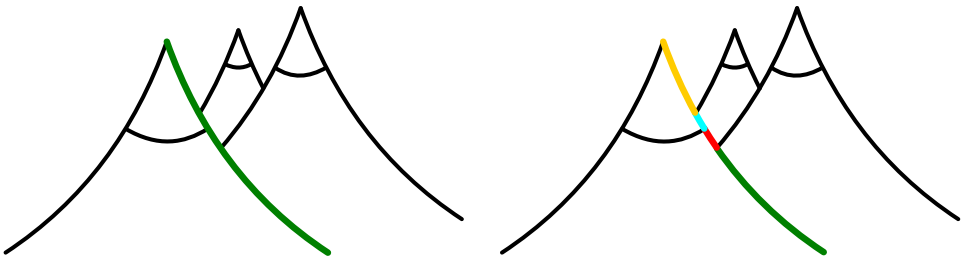
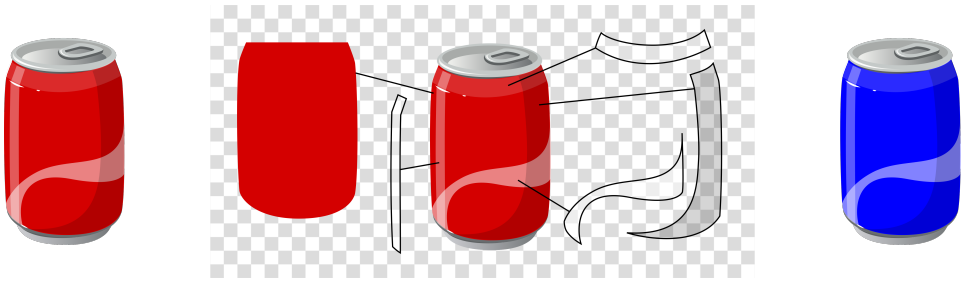


Figure 1.8: Compactness of vector drawings. To create the colored curve there are several possibilities: either creating one long curve (left figure) of four little curves (right image) stopping on each junction. Artists often prefer the simplest interpretation, which consists of one long curve, because this interpretation is easier to create and to edit: there are less control points and the $\mathcal{C}_2$ continuity of the green curve is difficult to create with many little curves.

Linear semi-transparent color functions are also used by artists to generate simple shadow and lighting effects as illustrated in figure 1.10. Semi-transparent color functions are stacked into layers: each layer contains a shape with a color function inside. Semi-transparent layers enable users to change easily the appearance of an object by modifying only the base color of the object (the color without any lighting effect). The layer compositing will automatically produce the right color for all the effects. Figure 1.9 shows a red soda can where the texture, shadows and highlights are all represented as semi-transparent layers filled by constant colors or color gradients. Changing the color of the bottom layer is sufficient to obtain a convincing blue can. Reproducing the same effect using bitmap requires to modify the color of every pixels and to redesign complex shadow and lighting effects of the can.



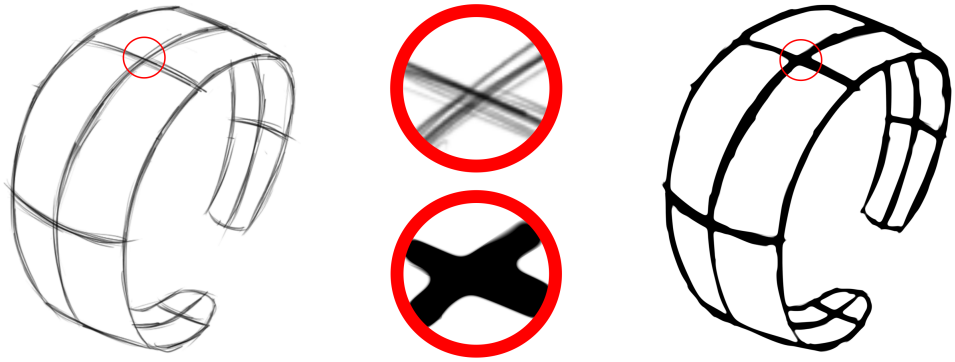
(a) red can (b) multi layered vector graphic representation (c) edited can

Figure 1.9: Multilayer vector graphics. The figure (a) shows a red soda can where the texture, shadows and highlights are all represented as semi-transparent layers filled by constant colors or color gradients (figure (b)). Changing the color of the bottom layer is sufficient to obtain a convincing blue can (figure (c)) because the artist has chosen to use semi-transparent layers to design shadows and highlights over a base opaque layer (Clipart by Altagracia Art on Shutterstock).

Figure 1.10: Four color variations of the same object. To create the blue, red or orange car from the green car, the user has to modify only one color. All variations of the base color are produced by semi-transparent grey layers. The image was taken from BSGStudio.

1.2 Challenges

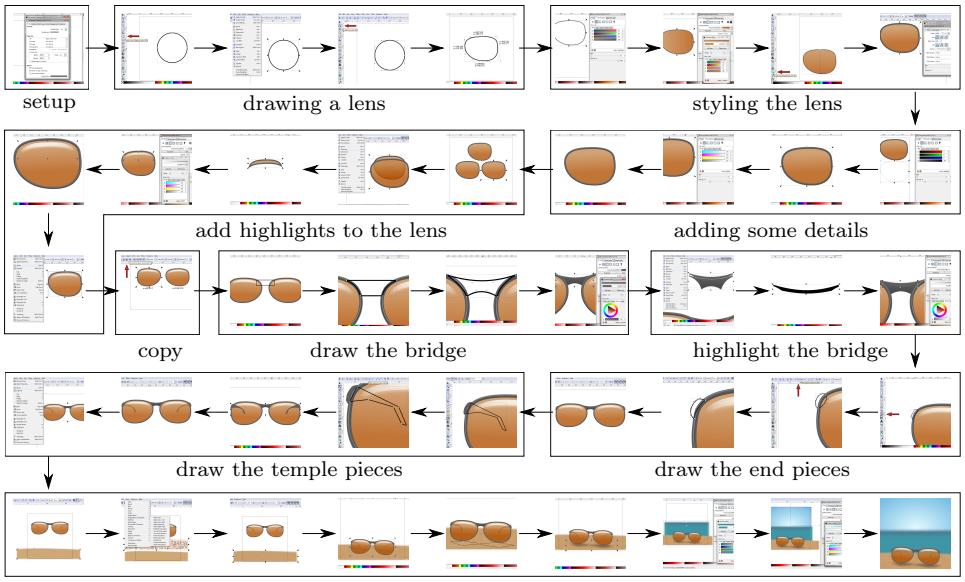
Drawing parametric curves and adjusting their control points requires more precision and user interaction than freehand sketching. Because of this, many artists still prefer drawing in a bitmap form, either with pen and paper or with digital painting tools like Adobe Photoshop and Autodesk SketchBook. However, commercial vectorization tools often convert thick pen strokes into parametric regions rather than clean Bézier curves (Figure 1.11).

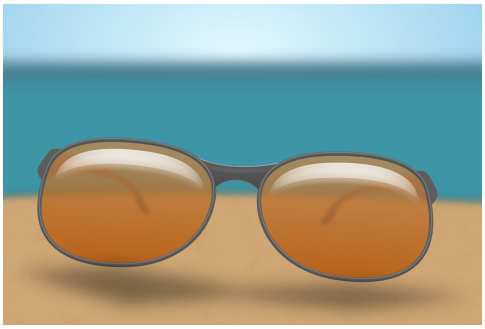


(a) input image (b) vectorization from Inkscape

Figure 1.11: Inkscape vectorization. The left image was vectorized into the right image using the vectorization tool and the morphological filters of Inkscape. The output is a parametric region.

Creating cliparts is even more complex than creating line drawings. The first step is to describe the contour of each shape. This step consists in positioning the control points of the curves describing contours. The second step is to define the color function of each shape. Figure 1.12 shows the main steps of a tutorial which explains how to draw a pair of sunglasses with Inkscape. A novice designer needs one hour to create a pair of sunglasses using this tutorial composed of 45 steps. Another way to create a clipart is to take a picture and convert it using Illustrator. But the transparency effects are not reconstructed neither complex color variations, as shown in figure 1.13.





generated image following the tutorial

Figure 1.12: Tutorial explaining how to create vector graphics (design.tutsplus). Creating simple sunglasses requires a lot of user interactions. A novice designer took one hour to reproduce the target image using Inkscape.

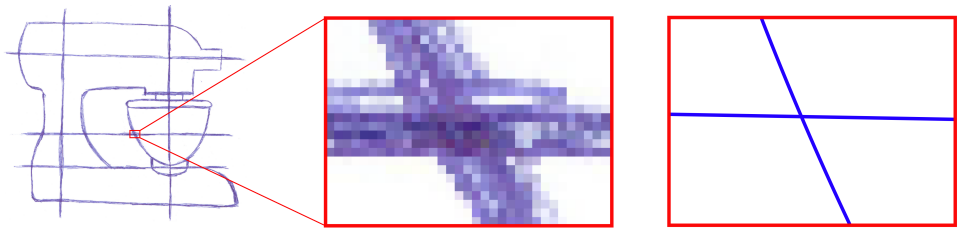
The goal of our thesis is to provide tools to help novice users to generate vector graphics. More precisely, we want to convert rough freehand bitmap sketches to clean vector line drawings and color images into multi-layered vector cliparts.

The main challenge is to understand how pixels interact together to form geometric structures. In standard vector graphics, classical geometric structures are curves and shapes. The main challenge in extracting such struc-

(a) input image (b) vectorization from adobe Illustrator

Figure 1.13: Illustrator CS6 vectorization. The left image was vectorized into the right image using the vectorization tool of Illustrator. The colors are constant within each region, giving a posterized look to the image. These pictures were taken from helpx.adobe.com/illustrator.

tures is that they often correspond to noisy pen strokes or regions in the input image. Figure 1.14 illustrates how curves are composed of several pen strokes, making the number of curves and their exact position ambiguous. To recover the right position of the curves, we need to associate each pixel to



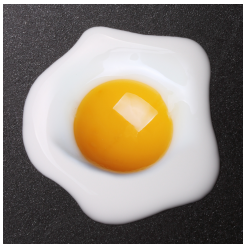
(a) a rough line drawing (b) zoom on stokes (c) curves reconstruction

Figure 1.14: Ambiguity on curves position. Thickness of curves due to several strokes generates ambiguities: we have to figure out how many curves there are and find their exact position in this set of pixels.

a curve which is a labelling problem that has k^n possible configurations if we consider that a pixel can only be associated to one curve. k is the unknown number of curves of the solution and n is the number of pixels of the input image. We can consider that n^n is an upper-bound of such problem. But if we want to associate the middle pixels of figure 1.14b to two overlapping curves, each pixel has to be assignable to several curves. The complexity explodes to 2^{kn} with an upper-bound of 2^{n^2}.

On color images, the main ambiguity comes from shadows and lighting

effects. Such effects are modelled by semi-transparent layers. For each pair of pixels of different colors, we have to decide if they belong to two separate opaque layers, if they share an opaque layer but one of the two pixels is also in a semi-transparent layer, or if they belong to the same opaque layer, the color difference between the two pixels being caused by noise.



For example, the inset shows an egg with an highlight on the egg yolks which is on the top of the egg white. A challenge is to detect that the highlight is a semi-transparent white layer on the top of the yellow layer representing the egg yolks and not an opaque layer. This can be formulated as a labelling problem, each label corresponding to an opaque layer or a semi-transparent layer. If we do not consider the semi-transparent layers, there is k^n possible configurations with k the unknown number of opaque layers on the image and n the number of pixels of the image. When we add semi-transparent layers, a pixel as to be assignable to several layers. The complexity becomes $(k2^s)^n$ with s the unknown number of semi-transparent layers. This complexity can be approximate by 2^{ln} with l the unknown number of layers (opaque and semi-transparent). In theory, the number of semi-transparent layers can by infinite, but in practice we assume that it is lower than the number of pixels. While the two problems of curve and layer assignment have a similar apparent complexity, the solution space is much smaller in the case of curve vectorization because most pixels of a drawing are white and as such are trivially assigned to no curve.

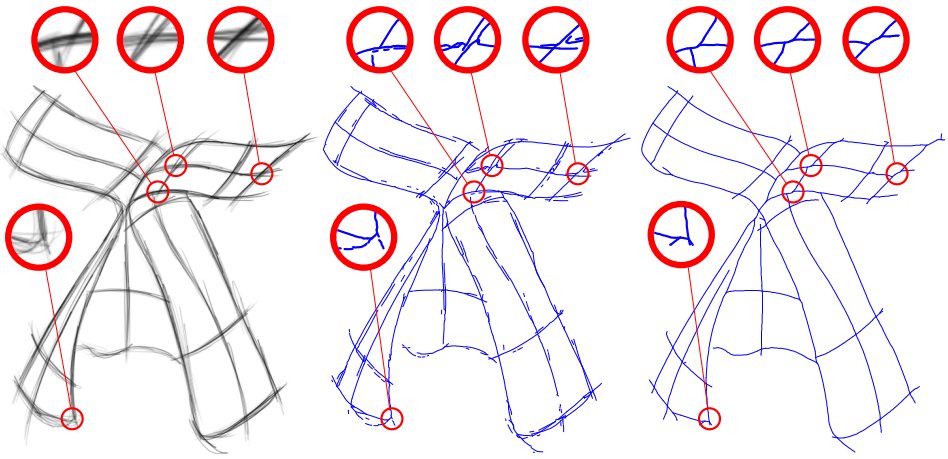
We will present in the next chapter different existing approaches which try to solve these problems.
In chapter 3 we will present our contributions, and in chapter 4 and 5 we will respectively present our method to vectorize line drawings and color images. In chapter 6 we will present a new stochastic algorithm called Delaunay Point Process which palliates the limitations of our vectorization algorithms and generalizes to other applications.

Chapter 2

Related work

2.1 Extracting parametric curves

The main challenge in sketch vectorization is to understand how pixels interact together to form curve networks. In figure 2.1a, all highlighted junctions look similar even if their interpretation differs. Figure 2.1b and c illustrate how existing methods struggle to extract the correct curve network, especially on junctions. In 2.1b, there are too many curves on junctions, and in 2.1c, junctions are reconstructed with too many intersection points.

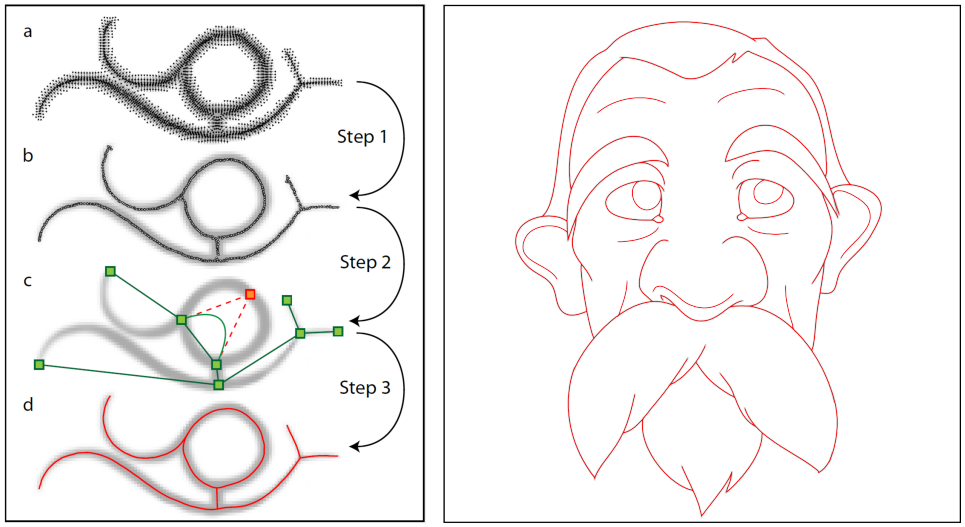


(a) Input rough sketch (b) Noris et al. [NHS+13] (c) Sketch filtered by [BCF+07] followed by [NHS+13]

Figure 2.1: Challenges on rough sketch vectorization. Rough sketches often contain overlapping strokes (a), which existing vectorization algorithms [NHS+13] represent as multiple curves (b). Pre-filtering the drawing with the method of Bartolo et al. [BCF+07] improves the vectorization, but produces spurious curve segments at junctions (c). Since existing algorithms analyze junctions locally, they cannot recover the proper topology of these seemingly similar line configurations.

2.1.1 Line drawing vectorization

A number of methods have been proposed to vectorize various types of drawings. Many algorithms target technical diagrams composed of straight lines and circular arcs [HT06], while freeform splines are more common in cartoon images [BF12, NHS+13, BLW15]. All these approaches follow a similar three-step procedure. First, a 1-pixel width *skeleton* of the drawing is extracted and junction points between multiple lines are identified. Second, a topological graph is extracted from the skeleton. Nodes of this graph are endpoints or junctions and each edge is associated to one primitive. The goal of this graph is to capture how pixels interact together to form the curve network. Finally, vectorial primitives (lines, arcs, curves) are fitted on each edge of the topological graph and primitives that meet at a junction are merged based on heuristics on tangent alignment or curvature agreement. Figure 2.2 shows the three steps of [NHS+13] and Figure 2.3 the ones of [BLW15].

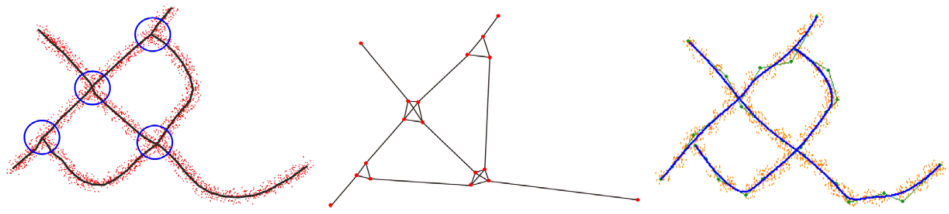


(a) Three steps algorithm

(b) Clean line drawing vectorization from [NHS+13]

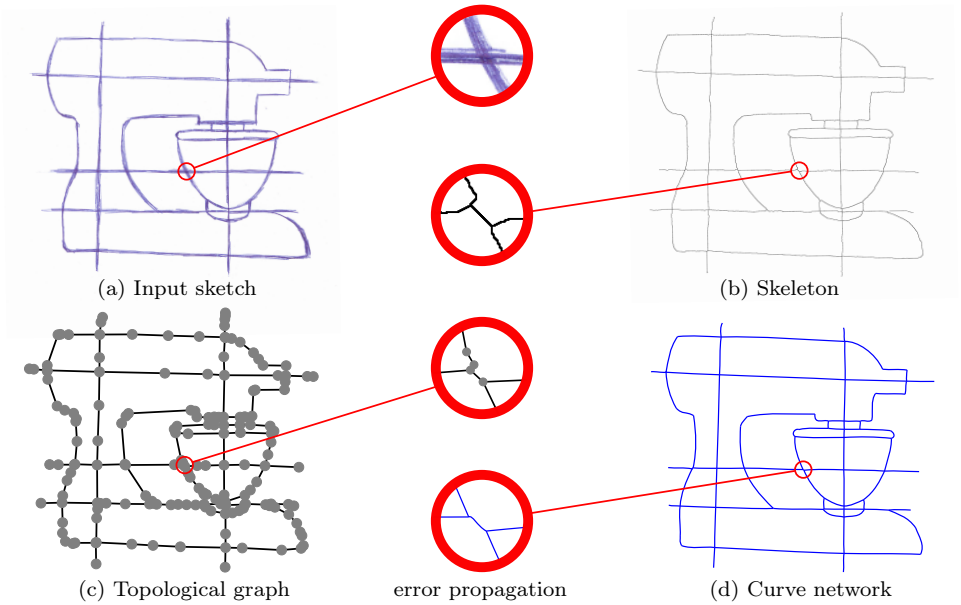
Figure 2.2: The three steps method of [NHS+13]. The first step is to extract a skeleton using a gradient-based clustering algorithm. The second step is to extract the topological graph from this skeleton to capture how pixels interact together to from the curve network. The nodes of this graph correspond to curve junctions and end-points and edges correspond to curves. From the information of this topological graph, the centerline position and the junctions between curves are recovered and vectorized in a third step.

However, because these three steps are applied in sequence, errors in one step are propagated to the subsequent steps as illustrated in figure 2.4. In



(a) Skeleton (b) Topological graph (c) Curve network

Figure 2.3: The three steps method of [BLW15]. The input is a 2D point cloud. The first step extracts a skeleton using alpha shape (a). The second step extracts a graph from this skeleton (b). In this graph, X-junctions are represented by four nodes fully connected. The multiple edges on junctions are due to the multiple interpretations. The curves are fitted in a third step (c).



(a) Input sketch (b) Skeleton

(c) Topological graph error propagation (d) Curve network

Figure 2.4: Error propagation in sequential approaches. The input bitmap (a) contains junctions between thick lines. The skeleton (b) contains multiple intersection points on the ambiguous junctions. These intersection points are transformed into nodes on the topological graph (c). Each edge of this graph is reconstructed as a parametric curve, yelding spurious short curves at junctions.

particular, the topology of the skeleton extracted in the first step remains fixed in the graph extraction and vectorization step, despite the fact that it is often erroneous at junctions, as illustrated in Figure 2.1b. In addition, existing methods refine each junction independently based on local informa-

tion whereas a global analysis of the network could help disambiguates its topology. For example, [NHS+13] tests all possible connectivities on junctions.

In chapter 4 we present a global optimization method that performs topological changes during curve fitting to find simple curve networks at junctions.

2.1.2 Line drawing simplification

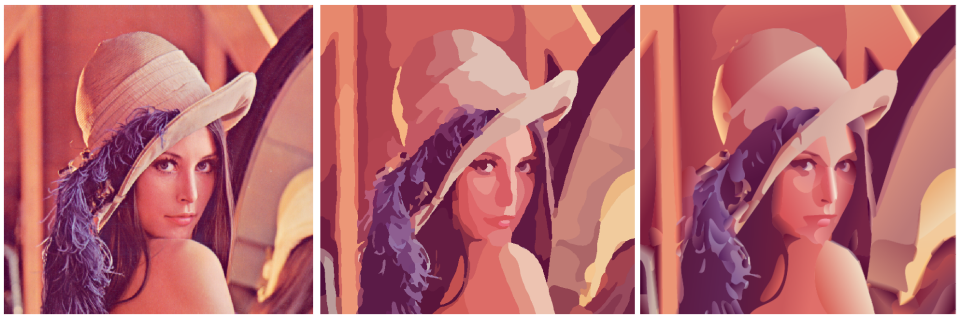
The input line drawing can be simplified using standard Gabor Filter [BCF+07] or more complicated filtering based on CNNs [SSISI16]. But, filtering the input sketch does not help much to recover the right topology at junctions, as shown in Figure 2.1c.

With the advent of digital drawing tools, several methods have been proposed to simplify drawings composed of vectorial pen strokes [BTS05, OK11, LWH15]. While such methods also face the challenge of merging strokes to form long curves, the additional knowledge provided by the shape and orientation of the input strokes greatly facilitates proper handling of junctions. Nevertheless, we will show in chapter 4 that our vectorization algorithm can produce results of comparable quality when applied on rasterized drawings, despite the fact that input bitmaps offer less information than digital strokes.

2.1.3 Line-network extraction

The problem of line drawing vectorization is also related to line-network extraction which has received significant attention in computer vision to identify roads in aerial images, blood vessels or neurons in medical images, or galaxy filament in astronomic images [PJPZ10, TBA+13, CFL13]. These methods build on strong shape priors to favor particular forms of line-networks, but these priors are often too specific for freehand drawings. For example a property of blood vessel is that the topological graph is a tree. So in blood vessel extraction, X-junctions should be penalized whereas in road network extraction of a city, X-junctions should be favored. In addition, these methods focus on localizing the lines and modeling the network topology rather than converting the lines to parametric curves. In chapter 6 we consider the use of an approach for such applications.

2.2 Extracting parametric color regions



(a) input image (b) constant color (c) higher order gradient

Figure 2.5: Simple gradient decomposition. [LL06] decomposes the input image into a single layer composed of several opaque shapes. Shapes can be filled by constant colors (b) or linear and quadratic gradient colors (c). This method do not capture semi-transparent effects.

Vectorization of color images is very challenging because each pixel has to be associated to parametric primitives and parameters of primitives have to be estimated. Most vectorization algorithms follow a general approach composed of two steps. The first step extracts shapes (segmentation, super-pixel, edge detection, ...). The second step fits vectorial model on these shapes and their colors. The output image can be defined either explicitly by the vector primitive (eg. mesh based vectorization) or implicitly (eg. diffusion curves). For explicit formulations, the image can be represented by one or several layers. A layer can be composed of regions filled by a constant color, a quadratic color, a mesh. The more the color model will contain parameters, the less the vector graphics will be editable.

2.2.1 Image vectorization

Region-based vectorization Most vectorization algorithms represent color images with a single layer. Commercial tools such as Adobe Illustrator's Image Trace [Ado13] only support constant color fills and as such require users to balance over-segmentation with quantization artefacts. Lecot and Lévy [LL06] were among the first to attempt vectorizing images with parametric gradients (linear and quadratic). Figure 2.5 shows results they can achieve.

[YCZ+16] focus their work on optimizing shapes of vector graphics. A segmentation into regions is first computed and the boundary of regions is vectorized into cubic Bézier curves.They express the problem of regularizing the vectorial shape of a region with an energy composed by a data term

measuring the distance between the input image and the reconstructed image and four prior terms to penalize self-intersections, false corners with small angle variations, short handles and twisted sections. Their energy is continuous and differentiable, they find a local minima using standard gradient methods. While we also express vectorization as an optimization, we focus on color simplification rather than shape simplification.

The main problem of fully automatic vectorization approaches is that if the segmentation loose details, they cannot be recovered in subsequent steps. [JHWS17] proposes an interactive method to vectorize color images with user guidances. Users can interactively select parts of the vectorization to be refined. We also obtained our best results using user-guided segmentation.

Follow-up work introduced more complex gradient representations, such as gradient meshes, subdivision surfaces and diffusion curves.

(a) input image (b) reconstructed image (c) gradient mesh

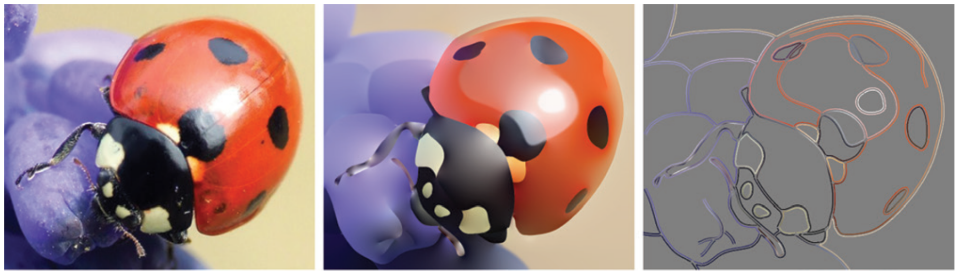
Figure 2.6: Gradient mesh. Each vertex of the mesh is associated to one color and each edge to a set of cubic Bézier splines. Images are taken from [SLWS07].

Mesh-based vectorization In region based vectorization, the input image is segmented into a set of regions. Each region contains a color function which can be constant, linear or quadratic. This generates images with discontinuity between each region. Mesh-based vectorization was introduced to generate $\mathcal{C}^0$ images. The image is represented by a mesh that interpolate colors stored at the vertices. Edges of the mesh are represented with Bézier curves. The classical meshes used are triangulations [LHM09] and quad-meshes [SLWS07, XLY09], called Gradient-Mesh. Vectorizing images with gradient meshes implies recovering the number of vertices of a mesh, the position of the vertices, the tangent of the curves on vertices and the color on vertices. [SLWS07] start from a manually designed gradient mesh. This mesh is optimized by minimizing an energy composed of a smoothness constraint and the distance between the gradient mesh and the input image.

The energy is optimized using Levenberg-Marquardt algorithm [Lev44].

The main problem of gradient mesh is that, because of its $\mathcal{C}^0$ property and grid topology, capturing fine details require very dense meshes, as shown on Figure 2.6 on the eye of the bird. To palliate this problem, [LHM09, LHFY12] introduce the concept of subdivision surfaces. The main idea is to add discontinuity curves on the mesh. A discontinuity curve is a set of connected edges of the mesh. The vertices of these edges contain two colors: one for each sides of the curves. Their method is composed of two steps: first, curvilinear features of the input image are extracted with method based on the Canny edge detector, second a gradient mesh is fitted in-between these curves.

Mesh-based vectorization produces very accurate vector graphics but these vector graphics do not reach the properties of compactness and editability. If we want to modify the shape of the eye of the bird of figure 2.6, we have to move many control points, either the node of the mesh or the ones defining the curvature of the Bézier splines. Changing the color of the bird requires also a lot of user interactions: the color of each node of the mesh has to be redefined and optimized manually to create some lighting and shadow effects.



(a) target image (b) output image (c) diffusion curve

Figure 2.7: Diffusion curve. Curves on image (c) are Bézier splines. On each side of each curve, colors are defined in arbitrary position. These colors are interpolated along curves. The output image (b) is obtained by interpolating curve colors inside shapes and applying blur effects on curves. Images are taken from [OBW+08].

Diffusion curves. To simplify the editing process, [OBW+08] proposed to only manipulate the discontinuity curves and to obtain the output image diffusing colors of the discontinuity curves. Figure 2.7 illustrates the principle

of diffusion curves: the vector graphic is composed by a set of Bézier splines with color functions to each sides of the Bézier splines. The output image is obtained diffusing colors of splines using PDE and optionally blurring the splines. Generating the input image from the diffusion curves is the forward problem.

The inverse problem (ie recovering diffusion curves from a raster image) is a difficult problem composed of two sub-problems. The first sub-problem is to recover the number of curves and their geometry. The second problem is, from a given set of curves, to recover the color functions of each sides of the curves. These sub-problems have been solved as two separate steps: extract edges to recover curves and then fit color models. [OBW+08, Jes16, XSTN14] focus mainly on recovering the colouring of the curves from a given set of Bézier curves manually segmented [Jes16], from curves obtained by Canny edge detector [OBW+08] of curves obtained in Laplacian domain [XSTN14]. More recent work focuses on recovering the curve geometry [ZDZ17]. They globally penalize the complexity of the set of curves by minimizing the total length of the curves.

Diffusion curves generate very accurate vector graphics similar to patch-based vectorization but more compact. However, the editability remains difficult: each modification requires to solve PDE. Moreover, commercial tools and standard format (SVG) do not support diffusion curves and their is no model for multilayered diffusion curves yet.

(a) input image (b) layer decomposition

Figure 2.8: Semi-transparent layer decomposition.[RLMB+14]

Multilayered vectorization The main drawback of foregoing cited methods is that they generate only one opaque layer. This results in the generation of vector graphics difficult to edit, mainly on shadow and lighting effects. [EWHS08, EPD09] produce high quality vector graphics with a multi layer decomposition from a 3D model. They reach the property of editability but they need the 3D model of the scene to generate the layered vector graphics. Little work focuses on producing multilayered vector graphics from images. Each pixel of the input image has to be associated to several layers and for a given layer, its color function has to be recovered. This is an inverse problem: from one raster 3-channels image, multiple images (ie a 4-channel color per layer) have to be reconstructed.

[ZCZ+09] vectorizes cartoon animation in two layers: the background layer and the foreground layer. This decomposition allows users to move objects on generated images. However their layers do not capture semi-transparent effects. Richardt et al. [RLMB+14] tackled this challenge by proposing an interactive method to convert bitmaps into opaque and semi-transparent linear vector layers (Figure 2.8). Their method iterates between two main steps: manual segmentation of a layer and then fitting of a color function on the region. However, their method requires extensive user intervention to iteratively select the semi-transparent regions in a front-to-back order.

(a) input image (b) reflectance (c) shading

Figure 2.9: Intrinsic image decomposition. The input image is decomposed by [BBS14] into two layers: one containing the reflectance and another one containing the shading.

2.2.2 Image decomposition

Figure 2.10: Image matting problem. The goal is to recover the transparency of an object (e). Two pictures (c) and (d) of this object are taken with two different known backgrounds (Figure (a) and (b)) [SB96].

Layered vectorization is also related to other ill-posed image decompositions such as image matting [SB96] illustrated in figure 2.10, reflection separation [LZW04] and intrinsic images [BBS14] illustrated in figure 2.9. However, a major difference between these methods and our work is that they aim at separating only two rather than multiple layers, being foreground and background or reflectance and shading. The second major difference is that the image is only decomposed in several layers. No fitting of parametric color functions is done. Nevertheless, several such algorithms make the decomposition well-posed by penalizing complexity via a prior on sparse image gradients [LZW04] and few refectances [BBS14], similar to our goal of producing simple vector graphics.

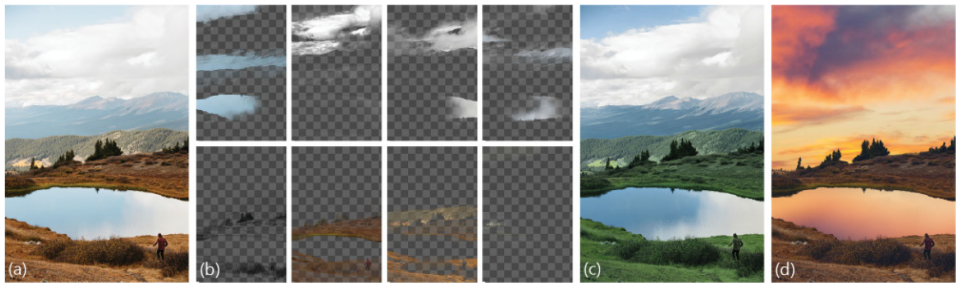
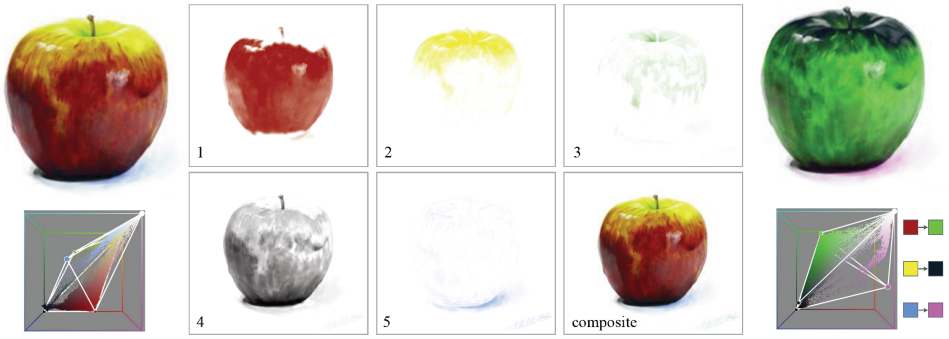


Figure 2.11: Layer decomposition. The method of [AASP17] decomposes an image (a) into several semi-transprent layers (b). Each layer is associated to a normal distribution embedded in the RGB color space. This decomposition allows color editing (c) and compositing (d).



(a) input image + convex-hull (b) layer decomposition (c) color editing

Figure 2.12: Layer decomposition. The method of [TLG16] decomposes an image in several semi-transparent layer. They first compute the convex-hull of the color space and then compute for each pixel of the input image a convex combination of the vertices of the convex-hull. The algorithm outputs constant color layers with per-pixel transparency.

A more recent work [TLG16] illustrated in figure 2.12 decomposes an image into more than two layers. They compute the convexhull of the input image in the RGB-color space. Then, they express each pixel of the image as a convex combination of the vertices of the convexhull. Each vertex of the convexhull is associated to a layer. The RGB color value of the layer is constant and equal to the color of the vertex and the transparency of the layer is the weight of the vertex on the convex combination. Their approach allows users to make simple color editing, such as changing the overall color of an object, but cannot handle complex color function editing. The number of layers and their order has to be provide by the user.

[AASP17] provides a more flexible layer decomposition. In their method, each layer is associated to a 3D normal distribution embedded in the RGB color space. Their method, illustrated in figure 2.11, provides a highly photorealistic decomposition. However, the layers are raster images. While existing single-layer vectorization algorithms could be used to vectorize each layer, the spatially-varying transparency would result in complex, hard-to-edit vector graphics. In chapter 5, we propose a method that seek to produce simple layers for easy editing.

2.2.3 Image simplification

Similarly to line drawing simplification and beautification, researchers in non-photorealistic rendering have proposed a variety of image filters to abstract and stylize photographs. Popular methods include the use of scale-space filtering [DS02] and edge-aware filtering [WOG06] to remove low-

contrast details. However, such image abstraction filters are usually applied independently of image vectorization. Our goal is to use vectorization as a mean of simplification by expressing the image with a small number of parametric color gradients. A similar idea of restricting the image formation model to achieve abstraction has been explored by Gerstner et al. [GDA+12], who produce pixel art by converting a photograph into a low-resolution image with a reduced color palette.

Chapter 3

Our Contributions

Limitations of previous work. Previous methods focused on local accuracy rather than global simplicity. While they produce high quality vectorizations, they are neither compact nor easy to edit. For color images, vectorization algorithms mainly focus on extracting only one opaque layer and representing the color as meshes or diffusion curves. Shadows and lighting effects of such vectorization can not be edited separately from the remaining of the image. Other algorithms decompose images in several raster layers. Transparent effects can be extracted but they remain difficult to edit due to their raster representation. For line drawings, previous work produces accurate vectorization of clean line drawings. Nevertheless, these vector line drawings are not easy to edit because junctions are too complex and curves are not well connected on junctions. This is due to the local optimization of each junction and the propagation of the error of the topological graph into the curve network.

To sum-up, the main drawback on previous work is that they do not reach the property of compactness and editability of hand-made vector graphics.

3.1 Simplicity

The main idea behind our work is to globally minimize the complexity of the output to favor the simplest interpretation of the input image in terms of number of curves, number of layers and transparency of layers. Favoring simplicity will generate vector graphics which are easy to edit.

Penalizing the complexity. While penalizing complexity is novel in the context of image vectorization, it has proven beneficial in other applications such as image segmentation [DOIB12], mesh decomposition [ZYH+15] and reflection separation [LZW04] among others. The rational behind these methods is that, when faced with an ill-posed inverse problem, humans often favor the *simplest* interpretation. This principle is known as the "law of prägnanz" in the gestalt psychology [Wer23], a branch of the cognitive psychology.

We already tested this principle of penalizing complexity in [FLB15]. The goal of this work was to transform a 2D vector line drawing into a 3D model.

In this approach, the 3D scene in which we wanted to add the new object is known. We optimized the 3D shape of the 2D line drawing minimizing the number of new surface orientations, ie the number of normals of the shape which are not aligned with the scene.

Fidelity versus Simplicity We now define an energy to measure the quality of a given vector graphics. On one hand a high quality vector graphics has to be similar to the input image, and on the other hand it has to be simple. We design the energy as a convex combination of two terms that express these two properties:

$$U(X) = (1 - \lambda)U_{fidelity}(X) + \lambda U_{simplicity}(X) \tag{3.1}$$

with X a vector graphics. For the case of the line drawing, the simplicity term can correspond to the number of curves, the number of control points, the number of junctions or how many curves meet at junctions. For color images, this term can be the number of opaque layers, the number of semi-transparent layers, the degree of the color function of layers, the number of curves of the contour of shapes. The parameter λ controls the amount of simplicity of the vector graphics X. A high λ produces compact, easy to edit vector graphics, while a low λ produces vector graphics very similar to the input.

3.2 Key role of junctions

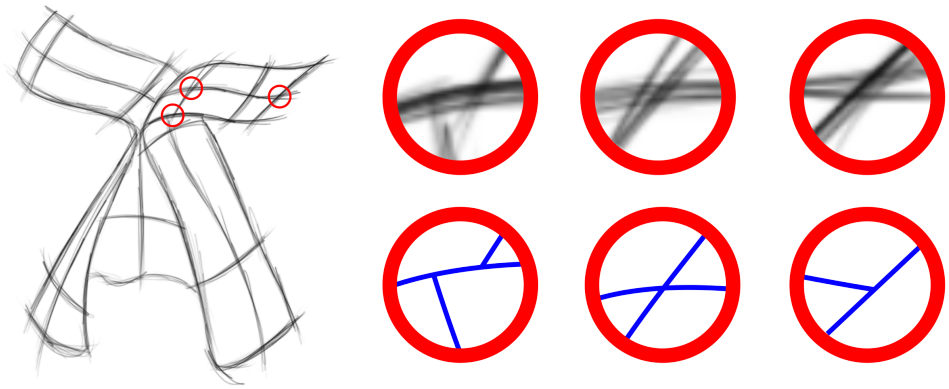
A junction is an intersection point between several curves or shapes' borders.

Ambiguities on junctions In the context of line drawings, junctions define the connectivity of the curve network. A X-junction gives many possible decompositions. If we assume that a X-junction is composed by two curves, there is three possible interpretations as illustrated in figure 3.1. Locally we cannot make the distinction between these interpretations. Another problem is that a part of a drawing can locally look like a X-junction without being one. Figure 3.2 shows several parts of a sketch looking like X-junctions but only one is a X-junction.

In the context of color images, a X-junction generates a lot of possible vector graphics. The junction can be composed by 1, 2, 3 or 4 opaque layers, and for each possibility several semi-transparent layers, especially if the number of opaque layers is low.

(a) input image (b) several interpretations composed by two curves

Figure 3.1: Ambiguity on X-junction . Three ways of interpreting an input image composed of five curves as a X-junction composed by two curves: the two curves can be tangeant or can intersect each-other.

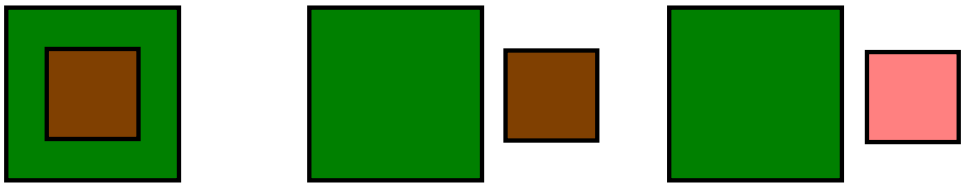


(a) a rough line drawing (b) zoom on areas which look like X-junctions

Figure 3.2: Ambiguity of X-junction interpretation. X-junctions on rough sketches are ambiguous: we have to make the distinction between one intersection point between four curves, two nearby intersection points, and pure noise.

Information on junctions When a color image does not contain junctions, the correct interpretation is relative to the way artists desire to edit it. Figure 3.3 shows an image containing a square above a background. If the color of the square is not orthogonal to the color of the background, there is an infinite number of possible decompositions. A critical information to make a choice between these configurations is missing.

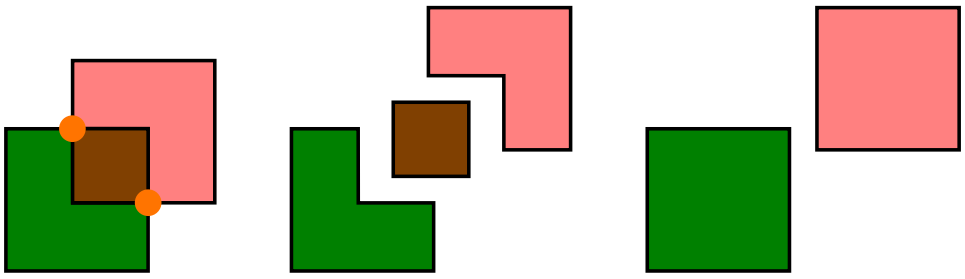
Perceptual studies emphasize the role of image junctions in the perception of occlusion and transparency [Met74, SC11]. In particular, T-junctions provide strong cues of local ordering between opaque layers [JGCC12, LMY+13] while luminance and chrominance patterns at X-junctions have been used for extracting transparent layers from images [DCKL97, SH03]. Figure 3.4 illustrates a X-junction on an image. The simplest decomposition in term of the number of layer is the decomposition which assumes that the image is composed by a semi-transparent red layer above the green opaque layer and the white background. The color functions are unique for the correct decom-



(a) input image (b) first interpretation (c) second interpretation

Figure 3.3: No junction ⇔ no transparency cue. The middle region of the input image (a) does not contain X-junctions. The image (b) and (c) are the two most probable decompositions in term of editability. The middle square is interpreted as an opaque brown region in image (b) and as a semi-transparent red square in image (c). The interpretation (b) minimizes the transparency of the middle square and the interpretation (c) maximizes its transparency. There is a continuous set of possible interpretations between these two extreme interpretations interpolating the transparency. There is no cue to decide which interpretation is the best because it depends on how an artist wants to edit the image.

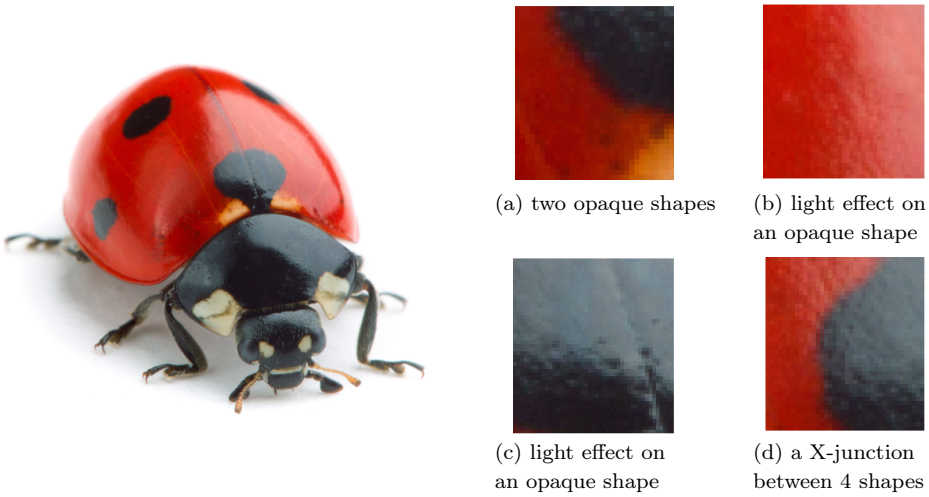
position in term of layers contrary to the color functions of the figure 3.3.



(a) input image (b) first interpretation (c) second interpretation

Figure 3.4: Simplicity of the transparency interpretation of X-junction. The input image on figure (a) can be interpreted in two valid ways in term of fidelity: a composition of three opaque layers (image (a)) or two layers whose one semi-transparent (image (c)). The second interpretation is better in term of simplicity: it contains less layers.

Figure 3.5 illustrates several crops of border between two or more shapes on a real photograph. We can see that making distinction between pixels at a border of two objects and pixel at the beginning of a light effect on an object is very challenging. But on the junction on the crop (d), we can see that there is probably a semi-transparent shape on the top of the red shape and the black shape which correspond to a light effect. The presence of the junction allows us to infer the right decomposition.



(a) two opaque shapes

(b) light effect on an opaque shape

(c) light effect on an opaque shape

(d) a X-junction between 4 shapes

Figure 3.5: Border between shapes. The first crop (a) corresponds to a separation between two different objects: the red one and the black one. The crop (b), as the crop (c), corresponds to one object with a light effect. Local information around border pixel is not enough to make the distinction between opaque shapes or semi-transparent shapes. In the last crop, even if it is very ambiguous, we can see that the light effect on the red region (b) and on the black region (c) is the same light effect thanks to the X-junction.

To sum up, junctions are the most ambiguous points on images but paradoxically the one which contain the most relevant information for layer decomposition.

Considering the entire image, and particularly junctions between regions or between curves and their interactions, gives some important cues which help recovering geometrical structure.

3.3 Optimization

The energy 3.1 is a complex non convex energy. The fidelity term is a local term and the simplicity term a global one. The unknowns are both discrete and continuous. Discrete unknowns are the number of geometric primitives and the association between pixels and geometric primitives. Continuous unknowns are the position of control points and the value of the color functions. Solving such a complex energy is a NP-hard problem because of the combinatorial. So we cannot use standard deterministic method to find a

good minima of this energy.

Instead of working on pixel level, we reduce the combinatorial by working on primitive level. We start by constructing an over-segmentation (piece of curves, regions) to have a solution which minimizes the fidelity term. Then we group primitives together into longer curves or extended regions to reduce the complexity of the solution. Grouping reduces the number of primitives while keeping a good fidelity. To group primitives together we adapt Monte Carlo methods proposing new models for the vectorization problem and new operators to explore the configuration space induced by these new models. We based our design of models and operators on X-junctions. In context of line drawing, the topology of the skeleton is generally not correct on X-junctions: it contains too many intersection points which are transformed in too many little curves. We allow these extra primitives to be part of several longer curves to correct the topology of junctions. For color images, we saw that X-junction is the signature of transparency. We base our exploration on this property: we propose a Monte Carlo tree search algorithm that efficiently explores the solution space by leveraging layering cues at image junctions. Local decompositions obtained from X-junction initialize a layer growing approach. Layers grows descending the tree to reach possible configurations on the leaves of the tree.

Line drawing vectorization In chapter 4 we will introduce our method on line drawing vectorization. We jointly optimize the topology and the geometry of the curve network using a Monte Carlo Markov Chain algorithm. Our method is well adapted for vectorizing rough line drawings. Such drawings make junctions very challenging because the thickness of strokes results in topological error on the skeleton and erroneous topological graph. We will demonstrate the robustness of our algorithm on a variety of drawings, sketchy cartoons and rough design sketches.

ClipArt vectorization We present in chapter 5 our method to create multilayered vector cliparts from photographs. We demonstrate the effectiveness of our method by reverse-engineering existing cliparts and by creating original cliparts from studio photographs.

3.4 Extraction of geometric structures by joint detection and assembling of primitives

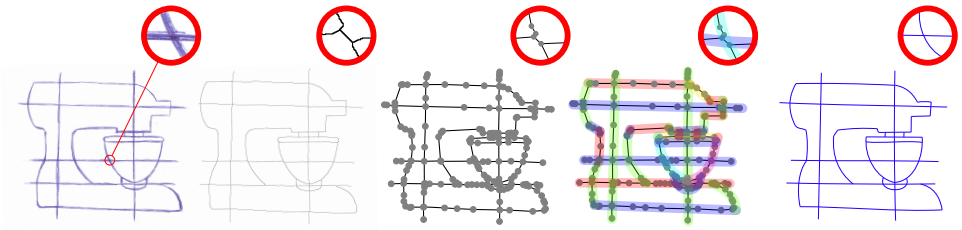
The main limitation of our approaches on line drawing vectorization and clipart vectorization is that they follow two independent steps. We first extract primitives from the input image (ie curves and regions) in order to construct an over-segmentation and then we run our optimizations on these primitives. Similar two steps approaches are often used on geometric shape extraction problems like line network extraction (road[PJPZ10], blood vessel[RP07], ...), object segmentation (windows on facade[HPFPL09], building from aerial images[WZS04], [BFL06]) and 3D reconstruction[SCD+06]. However, errors in the first step can hardly be recovered in the second step. On the last part of this thesis, we introduce Delaunay Point Processes, a framework for the extraction of geometric structures from images. Our approach simultaneously locates and groups geometric primitives (line segments, triangles) to form extended structures (line networks, polygons) for a variety of image analysis tasks. Similarly to traditional point processes, our approach uses Markov Chain Monte Carlo to minimize an energy that balances fidelity to the input image data with geometric priors on the output structures. However, while existing point processes struggle to model structures composed of inter-connected components, we propose to embed the point process into a Delaunay triangulation, which provides high-quality connectivity by construction. We further leverage key properties of the Delaunay triangulation to devise a fast Markov Chain Monte Carlo sampler. We demonstrate the flexibility of our approach on a variety of applications, including line network extraction, object contouring, mesh-based image compression and line drawing vectorization.

Chapter 4

Line drawing vectorization

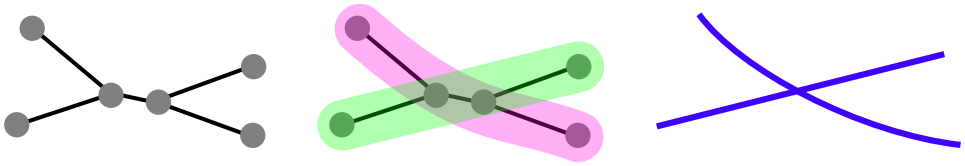
In this chapter, we present an approach to convert rough freehand bitmap sketches to clean vector drawings. Our algorithm takes as input bitmap line drawings, either scanned from pen-on-paper drawings or created with digital drawing tools like Adobe Photoshop or Autodesk SketchBook (Figure 4.1a). The output is a set of Bézier curves of order 1, 2 and 3. The main challenge of automatic vectorization is to extract the *topology* of the curve network, i.e. identify how the black pixels of the drawing should be grouped together to form different curves. Once this topology is extracted, the *geometry* of each curve is obtained by least-squares fitting, such that each curve best captures the black pixels it represents. Existing methods typically perform topology extraction and curve fitting as two sequential steps. Our key novelty is to perform these steps *jointly* to balance the compactness of the topology with the accuracy of the fitting. Figure 4.1 illustrates the main steps of our method. We start with an over-segmentation of the drawing where each segment between two consecutive junctions, sharp turns or endpoints is a curve. This initialization satisfies well our objective of accuracy, but often contains more curves than needed. Our optimization then consists in grouping these initial curves to reduce complexity without sacrificing accuracy. We introduce a new representation based on the concept of *hypergraph* [Bre13] to encode this grouping. In this representation, each group of edges forms a *hyperedge*, as illustrated in Figure 4.1d. A key advantage of this formulation is that two hyperedges can share one or more edges of the initial topological graph. This feature is critical to resolve extraneous branching at junctions, as it allows our optimization to simplify the overall curve network by assigning small spurious edges to multiple intersecting curves (see Figure 4.2 and close-ups in Figure 4.1).

While our algorithm produces high-quality vectorizations automatically, it achieves its full potential when guided by the user. In our interactive implementation, users can disambiguate junctions by imposing that two successive curves form a single curve, or that they form two separate curves. These local annotations are then propagated to the entire solution thanks to our global formulation. Users can also prevent local edits from having a global impact by fixing the parts of the solution that they want to preserve.



(a) Input sketch (b) Skeleton (c) Topological graph (d) Hypergraph (e) Curve network

Figure 4.1: Overview of the method. Our algorithm takes as input bitmap drawings (a). We first extract the 1-pixel width skeleton of the drawing to locate the curves and their junctions (b). We encode this information as a graph where edges correspond to curve segments and nodes to junctions, endpoints and sharp turns (c). The core of our algorithm consists in merging groups of successive edges to form *hyperedges* of a *hypergraph* (d). Note that several hyperedges can share the same edge of the original graph. Each hyperedge corresponds to a Bézier curve in the output (e). Edges that are shared by several hyperedges are implicitly collapsed by curve fitting, resulting in precise junctions despite extraneous branching of the skeleton.



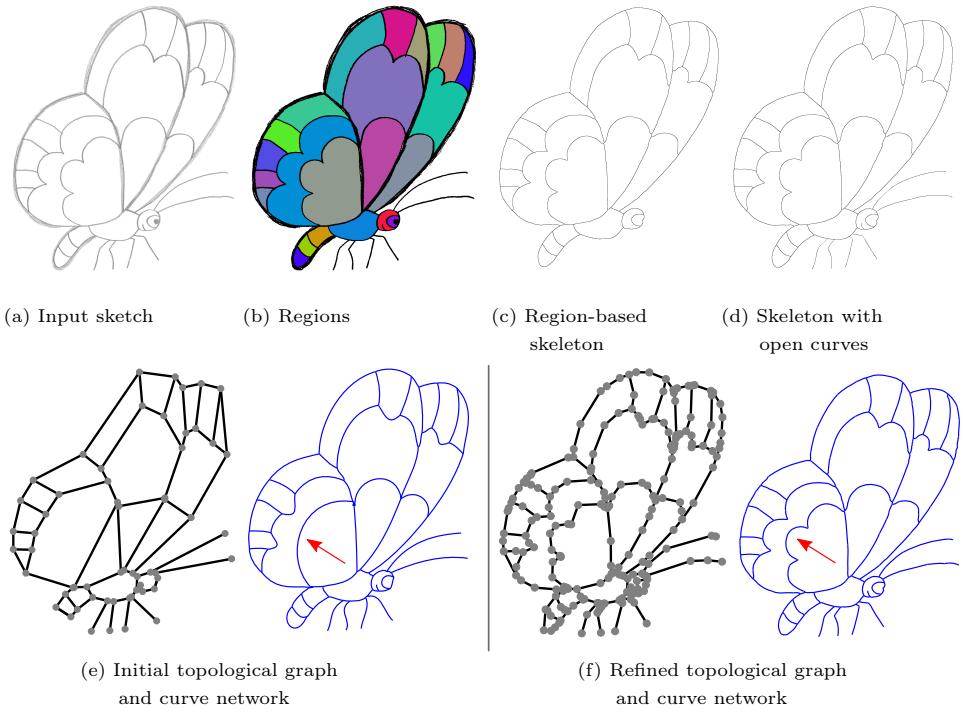
(a) Topological graph (b) Hypergraph (c) Reconstructed curves

Figure 4.2: Given the topological graph of the drawing (a), our algorithm groups successive edges to form *hyperedges*. We fit a Bézier curve on each hyperedge (c). Edges that are shared by several hyperedges, such as the central edge in this example collapse to a single point after fitting.

4.1 Problem formulation

4.1.1 Initialization by over-segmentation

Extracting the skeleton. Following standard practice, we initialize our curve network from the 1-pixel width skeleton of the drawing. Many solutions exist to compute such a skeleton. For clean line drawings, popular methods include morphological thinning [HT06] and iterative stroke pixel clustering [NHS+13]. However, these line-based methods tend to produce many extraneous branches on sketchy drawings. Inspired by [LWH15], we adopt a more robust region-based approach where we define the skeleton as the frontiers between adjacent regions of the drawing, as illustrated in Figure 4.3. We first detect the regions of the drawing by running the trapped-ball segmen-



(a) Input sketch (b) Regions (c) Region-based skeleton (d) Skeleton with open curves

(e) Initial topological graph and curve network (f) Refined topological graph and curve network

Figure 4.3: Extraction of the skeleton and topological graph. We adopt a region-based approach to be robust to sketchy lines (b,c), which we complement with a morphological approach for open curves (d). The initial graph only encodes junctions and endpoints (e). We refine it to include sharp turns (f).

tation algorithm [ZCZ+09], which is robust to small leakage between regions. We then iteratively dilate the regions until they meet and assign the pixels adjacent to two or more regions to the skeleton. The number of dilation iterations gives us an estimate of the local thickness of the lines. However, this region-based algorithm does not capture open curves. As a second step, we identify pixels of open curves as the ones that are at a distance greater than the local thickness of the closest skeleton point. We then compute the skeleton of these additional pixels using morphological thinning. Figure 4.3d shows the skeleton we obtain for a typical drawing.

Initializing the curve network. The drawing skeleton forms a network of 1-pixel width lines. The next step towards a vectorial representation is to identify which pixels should be grouped together to form curves. Following the terminology of Noris et al. [NHS+13], we call this grouping the *topology* of the drawing and we represent it as a graph $\mathbf{g} = (V, E)$ where nodes V correspond to the junctions and endpoints of the skeleton, and edges E

correspond to the skeleton branches. Each edge $e \in E$ is associated with a single Bézier curve segment B^e. We compute the geometry of each curve by chaining the corresponding pixels and minimizing the fitting error

$$\varepsilon(e) = \sum_{p \in S^e} (1 - \frac{w_p}{2}) \|B^e(t_p) - p\|_2^2 \tag{4.1}$$

where S^e is the chain of pixels associated with edge e, $t_p \in [0, 1]$ is the normalized position of pixel p along the pixel chain, and w_p is the thickness of the line at p normalized with respect to the maximal thickness over the entire drawing. We weight the fitting error by the line thickness to account for the fact that the skeleton is less precise along thick lines. We compute the initial curves by fitting Bézier curves of degree three, as illustrated in Figure 4.3e, although our optimization later considers curves of lower degree for higher compactness.

We further improve the accuracy of this initialization by recursively splitting the graph edges until the average fitting error of all Bézier curves is below 2 pixels. This operation ensures that we capture sharp turns along the skeleton branches, as shown in Figure 4.3f. We define the splitting point on an edge such that the fitting error of the two resulting curves is the lowest, as found by a binary search.

4.1.2 Simplification by hypergraph exploration

Given our initial, over-segmented vectorization, we now need to merge successive curve segments to reduce overall complexity and remove extraneous branching at junctions. Since each curve segment corresponds to an edge in the topological graph, merging multiple curve segments is equivalent to grouping edges of the topological graph. To model this operation, we rely on the concept of *hypergraph*, illustrated in Figure 4.2. In its most general definition, a hypergraph is a generalization of a graph in which an edge (also called a *hyperedge*) can connect any number of vertices. In this work, we adopt a more restrictive definition where each vertex is covered by at least one hyperedge, and each hyperedge connects at least two vertices. In addition, we impose that each hyperedge corresponds to a sequence of adjacent edges in the initial topological graph. These conditions are guaranteed by the perturbation operators of our stochastic optimization, described in Section 4.2.

Let $\mathbf{x} = (V, H_\mathbf{x})$ be a hypergraph of the topological graph $\mathbf{g}$, where V is the set of nodes and $H_\mathbf{x}$ the set of hyperedges. We associate each hyperedge $h \in H_\mathbf{x}$ with a chain of pixels $S^h_\mathbf{x}$ by concatenating the pixels of the skeleton associated with the edges grouped into h. Each pixel chain $S^h_\mathbf{x}$ yields a fitted Bézier curve $B^h_\mathbf{x}$ in the curve network. The degree of the Bézier curve is a

free parameter that allows the optimization to consider straight, quadratic and cubic curves, all being supported by the SVG format.

4.1.3 Energy formulation

Our goal is to explore the space $\mathcal{H}$ of hypergraphs generated from the initial graph $\mathbf{g}$ to find $\mathbf{x} \in \mathcal{H}$ that offers the best trade-off between the simplicity of the curve network and its fidelity to the input drawing. We measure the quality of this trade-off with an energy U composed of two terms:

$$U(\mathbf{x}) = (1-\lambda)U_{\text{fidelity}}(\mathbf{x}) + \lambda U_{\text{simplicity}}(\mathbf{x}) \tag{4.2}$$

where λ is a model parameter that balances the two terms.

Fidelity term. We measure the accuracy of a curve network as the sum of the fitting error of all its hyperedges

$$U_{\text{fidelity}}(\mathbf{x}) = \sum_{h \in H_{\mathbf{x}}} \varepsilon(h) \tag{4.3}$$

where $\varepsilon(h)$ is given by Equation 4.1.

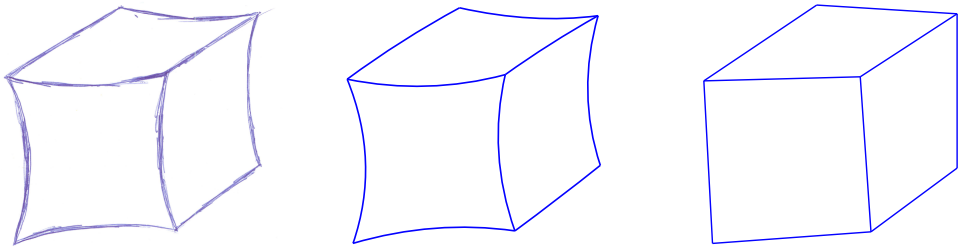
Simplicity term. The main novelty of our approach resides in explicitly optimizing for simple curve networks. We measure the simplicity of a network by the number of hyperedges, where lower is simpler. We also favor curve networks whose Bezier curves have low degrees since they are more compact and can be edited with fewer control points. The complexity term is defined as a sum of these two types of information weighted by the model parameter μ

$$U_{\text{simplicity}}(\mathbf{x}) = \sum_{h \in H_{\mathbf{x}}} 1 + \mu \text{Deg}(B^h_{\mathbf{x}}) \tag{4.4}$$

where $\text{Deg}(B^h_{\mathbf{x}})$ is the degree of the Bézier curve $B^h_{\mathbf{x}}$. As illustrated in Figure 4.4, a high μ value increases the presence of straight lines.

4.2 Exploration mechanism

Searching for the hypergraph that minimizes energy U is a non trivial optimization problem as U is non convex and contains global terms. Exhaustive exploration of the hypergraph space is only tractable for very simplistic input drawings as evaluating each configuration requires solving a least squares fitting problem (Equation 4.3). We adopt a more scalable strategy based on the Metropolis-Hastings algorithm [Has70]. In a nutshell, this algorithm makes a random exploration of the solution space by iteratively perturbing



(a) Input sketch (b) Low value of μ (c) High value of μ

Figure 4.4: The parameter μ controls the penalization of high degree curves. A high μ favors straight lines.

the current configuration $\mathbf{x} \in \mathcal{H}$ into $\mathbf{x}' \in \mathcal{H}$. The perturbed hypergraph $\mathbf{x}'$ becomes the current configuration with a certain probability depending on the energy variation between the two configurations, and a relaxation parameter T. In addition to scalability, such a Monte Carlo sampler easily supports user-provided constraints, as explained further in Section 4.2.4. We now detail perturbation operators and a relaxation schedule adapted to our problem. Algorithm 1 details the main steps of our optimization.

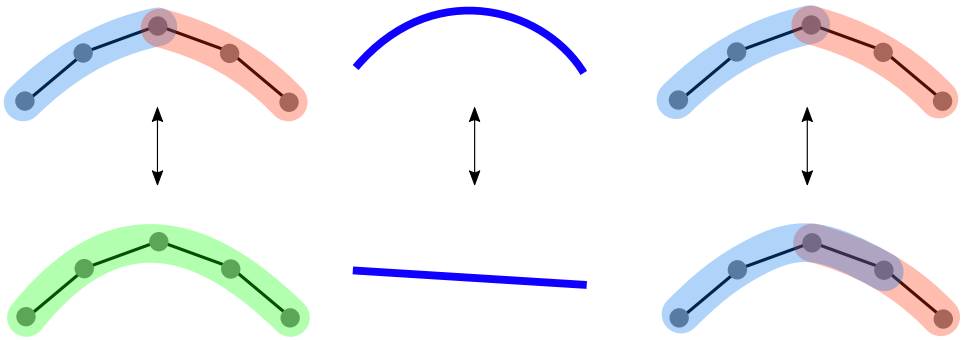
Algorithm 1 Exploration mechanism

Compute initial topological graph $\mathbf{g}$ (Sec. 4.1.1)
Initialize relaxation parameter $T = T_{\text{init}}$
Initialize $\mathbf{x} = \mathbf{g}$
repeat
  Generate $\mathbf{x}'$ from $\mathbf{x}$ with a random perturbation operator
  Fit Bézier curves $B^h_{\mathbf{x}'}$ on the perturbed hyperedges
  Draw a random value $p \in [0, 1]$
  if $p < \exp\left(\frac{U(\mathbf{x})-U(\mathbf{x}')}{T}\right)$ **then** update $\mathbf{x} \leftarrow \mathbf{x}'$
  else update $\mathbf{x} \leftarrow \mathbf{x}$
  Update $T \leftarrow C \times T$
until $T < T_{end}$
Finalize output representation (Sec. 4.2.3)

4.2.1 Perturbation operators

Our optimization seeks to simplify the curve network by merging Bézier curve segments and reducing their degree. We explore these objectives with three types of operators (Figure 4.5):

- Hyperedge merge and split. This operator splits a hyperedge into two adjacent hyperedges, and reversibly merges two adjacent hyperedges



(a) Merge/Split (b) Degree Switch (c) Overlap/Dissociation

Figure 4.5: Our optimization explores the solution space with three reversible perturbation operators: merging or splitting hyperedges (a), changing the degree of the curve associated with a hyperedge (b) and creating or removing overlap between hyperedges (c).

into one. These two operations are implemented by splitting or merging the sets of edges included in each hyperedge. As a result, a hyperedge containing only one edge of the initial graph cannot be split.

- Bézier degree switch. This operator modifies the Bézier degree of a hyperedge to take any value from degree one (straight line) to degree three (cubic Bézier).

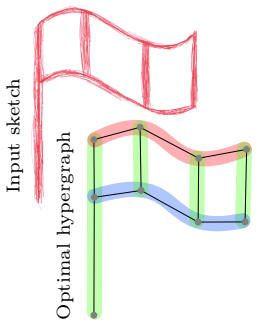
- Hyperedge overlap and dissociation. This operator integrates an edge of a hyperedge into a second hyperedge, and reversibly dissociates an edge associated to two hyperedges from one of them. This operator is particularly effective at simplifying topology at junctions.

Starting from the initial hypergraph $\mathbf{x}_0 = \mathbf{g}$, these three operators are sufficient to guarantee that (i) any hypergraph in $\mathcal{H}$ is reachable with a finite number of perturbations from any hypergraph of $\mathcal{H}$, (ii) the reverse pertubations exist, and (iii) perturbations only affect a hypergraph locally.

4.2.2 Relaxation schedule

The relaxation parameter T controls both the speed and the quality of the exploration.

Although the Metropolis-Hastings algorithm is guaranteed to converge to the global minimum of our energy when using a logarithmic decrease [SSF02], we prefer to use a geometric decrease of rate C to achieve reasonable running times.



While this approximation removes the guarantee of reaching the global minimum, it finds solutions close to this optimum in practice. To quantify this approximation, we performed 1000 runs of our algorithm on the simple sketch shown as inset, for which we computed the global minimum. The correct solution was found in 78% of the cases. The remaining 22% corresponded to local minima close to the global solution, with small visual differences on the resulting curves. In our experiments, we fix the initial temperature $T_{\text{init}} = 1$ and the decrease rate $C = 0.999^{\frac{1}{\text{Card}(V)}}$. Figure 4.6 shows the evolution of the configurations during the optimization.

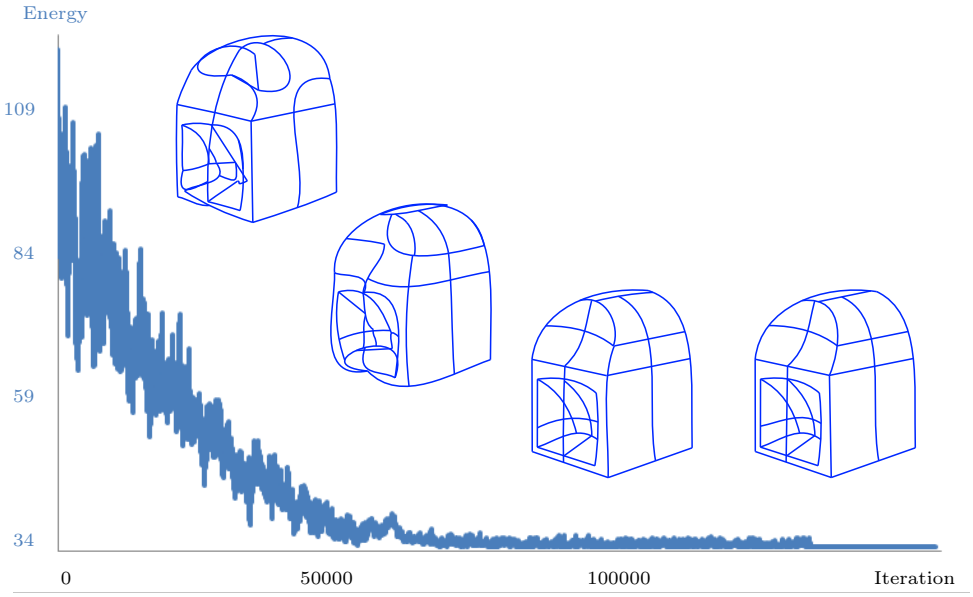


Figure 4.6: Evolution of energy U (Equation 4.2) during the Metropolis-Hastings optimization on a typical sketch. At the beginning of the optimization, perturbations are easily accepted (high energy). The process then becomes progressively selective until converging towards a configuration of interest. Although the two right configurations have both a low energy and are visually identical, their Bezier curves do not have exactly similar degrees.

4.2.3 finalization

We now describe two additional features to refine the curve network by imposing curve connectivity and continuity.

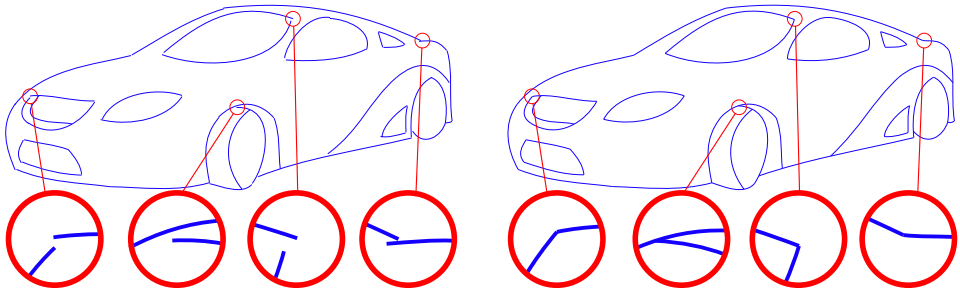


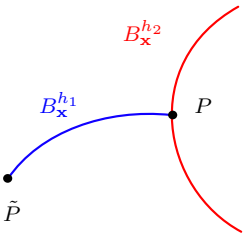
Figure 4.7: Fitting each curve independently does not preserve the connectivity of the drawing (left). Adding connectivity constraints ensures that the final result has the same connectivity as the initial skeleton (right).

Curve connectivity. Our iterative optimization fits a Bézier curve on each hyperedge independently. While this computation is fast, it does not ensure that curves connect at junctions, as shown in Figure 4.7 (left). To address this problem, we include two connectivity constraints to the fitting:

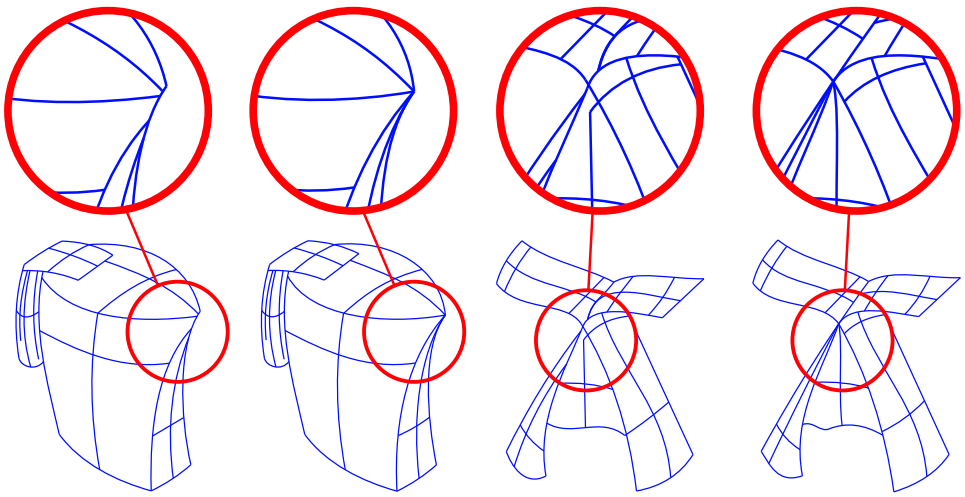
- If two hyperedges are connected at their extremities, the corresponding control points of the two Bézier cuves must be the same.
- If the extremity of hyperedge h_1 connects with a non extremity of hyperedge h_2, the corresponding control point P of h_1 must be on the curve $B_{\mathbf{x}}^{h_2}$ (i.e. $\exists t \in [0,1]$ s.t. $P = B_{\mathbf{x}}^{h_2}(t)$).

The first constraint potentially links all curves together. Satisfying it thus requires solving for the position of all control points at once, which is computationally much more expensive than independently fitting the curves impacted by a perturbation. In addition, the second term makes the optimization non-linear.

We linearize the problem by first minimizing the fitting error subject to the first constraint only. Then, for each hyperedge h_1 verifying the second constraint, we perform a binary search of $t \in [0,1]$ such that the control point $P = B_{\mathbf{x}}^{h_2}(t)$ of h_1 minimizes the fitting error $\varepsilon(h_1)$ (see inset for notations).



Since accounting for the connectivity constraints makes the fitting too costly to be performed at each iteration of the Metropolis-Hastings optimization, we only apply the constraints once the optimal hypergraph has been found. In practice, these constraints have a limited impact on the overall curve network and thus do not degrade accuracy significantly. Using similar conditions as the convergence stability



(a) Result obtained automatically (b) Result obtained with user interaction (c) Result obtained automatically (d) Result obtained with user interaction

Figure 4.8: User interaction. The user ensures that multiple curves intersect by imposing that they share the same segment of the initial vectorization.

experiment realized in Section 4.2, the global minimum was found in 80% of cases when applying these constraints at each iteration and in 78% of cases when applying them after the optimization, while the computation was 30 times slower in the former case.

Tangent continuity. Since our optimization considers Bézier curves of at most degree three, it decomposes curves with more than one inflection point into multiple segments. As an optional feature, we enforce tangent continuity of successive segments by aligning their tangents if they are almost co-linear. Similarly to the connectivity constraints, imposing curve continuity yields a non-linear optimization which we only perform once the optimal hypergraph has been found.

4.2.4 user interaction

One of the benefits of the Metropolis-Hastings algorithm is that it can easily incorporate user-provided constraints. Since the algorithm is iterative, users can stop the optimization process at any time to specify constraints, and let the optimization continue to see their effect. We support three types of constraints:

- **Merge.** The user can select two curves from the initialization and impose that they end up in the same curve after optimization. If the two curves are not consecutive, we select all other curves along the shortest path between the selected ones.

Figure 4.9: A selection of line drawings from different domains and our vectorization. Input drawings in the third column courtesy of [OK11].

- **Split.** The user can select two curves from the initialization and impose that they end up in different curves after optimization.
- **Freeze.** When the user is satisfied about part of a solution, she can freeze it by selecting the curves that should no longer be perturbed.

The optimization then only considers the perturbations that do not violate the constraints. Figure 4.8 illustrates the effect of user-provided constraints.

We also found that exposing the relaxation parameter T offers users useful control on the explorative behavior of the optimization. When T is high, the algorithm accepts drastic perturbations to escape local minima, while when T is low, the algorithm only retains small perturbations that improve the solution locally. With this control, users can force the algorithm to consider other alternatives when they are not satisfied with a solution, or in contrast can accelerate convergence by reducing T when they feel that the solution is close to optimal. Please see the accompanying video for a demonstration of this control.

4.3 Experiments

All results shown in this chapter were obtained with the automatic algorithm, except the two examples in Figure 4.8. We used a fixed $\lambda = 0.6$ for all results except the mechanical piece in Figure 4.9, where we used $\lambda = 0.3$ to capture the curve discontinuities on its side. We provide all our input

bitmaps and output curves as supplemental material. We have applied our algorithm on a variety of drawings from different domains, as illustrated in Figure 4.9 with a selection of cartoon, engineering, architectural and fashion design sketches. Note that since the drawing of the stool in Figure 4.8c,d and the shoe in Figure 4.9 are dominated by closed region, we did not activate the detection of open curves, which is why all dangling segments have been removed. We now compare our method to prior work on line drawing simplification and vectorization and evaluate robustness, impact of parameters and performance.

4.3.1 Comparisons with existing work

Figure 4.13 provides a visual comparison with a state-of-the-art vectorization algorithm [NHS+13] and with the *Image Trace* feature of Adobe Illustrator CC. We first performed the comparison on a sketchy drawing, and then also evaluated the impact of pre-filtering the sketch with the method of Bartolo et al. [BCF+07] to group the sketchy strokes into thick lines. Both algorithms produce multiple curves along sketchy lines and short spurious curves at junctions on the filtered sketches. In contrast, our method produces almost identical results on the two versions of each sketch, and recovers junctions with precision.

Figure 4.14 also compares our method with a recent line drawing simplification algorithm [LWH15]. We insist on the fact that [LWH15] takes as input digital drawings composed of vectorial strokes. Still, we obtain similar results even though we take bitmap drawings as input. Our results are even more accurate at junctions thanks to the connectivity constraints described in Section 4.2.3.

4.3.2 Impact of parameters

Our algorithm offers a trade-off between accuracy and simplicity, controlled by the parameter λ in Equation 4.2. Figure 4.11 illustrates the effect of this parameter. A low λ yields a very low fitting error but a high number of curves. In contrast, increasing λ greatly reduces the number of curves but the resulting network deviates more from the input. We again measured error with respect to a ground truth vector drawing that we rasterized to serve as input to our algorithm. Note that at low λ, the top-right part of the shape is best approximated by small linear segments, while a high λ produces a smoother, albeit less accurate output. We fixed the other parameters μ to 0.2 for all results in this chapter except Figure 4.4, and the ball radius of trapped-ball segmentation algorithm to 3 pixels.

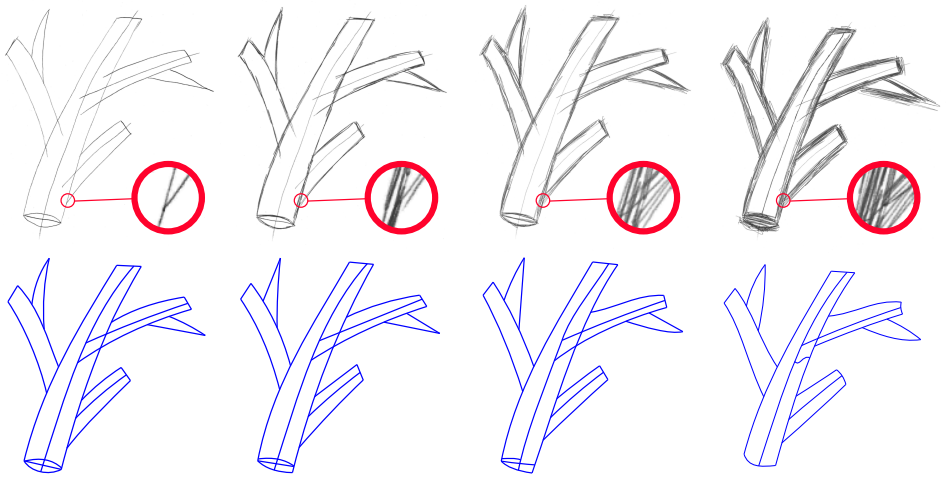


Figure 4.10: Robustness to sketchiness. Our algorithm generates very similar curve networks for various levels of sketchiness, even though some details are lost in very sketchy drawings.

4.3.3 Robustness

Sketchy and thick lines are very challenging for existing vectorization algorithms because they result in noisy skeletons with many extraneous branches, especially at junctions. Figure 4.10 shows that our algorithm produces consistent curve networks for increasing levels of sketchiness. Figure 4.12 provides a quantitative evaluation of the impact of line thickness. We performed this evaluation by rasterizing a vector drawing with increasing line thickness and measuring the distance between the recovered curve network and the ground truth, expressed in pixels. The average error remains below 0.6 pixels for a thickness of 24 pixels.

4.3.4 Performances

Depending on the complexity of the input bitmap, our algorithm takes a few seconds to a few minutes to produce output curves automatically. However, since our optimization is iterative, the user does not have to wait until completion to edit the result. Instead, she can stop the algorithm at any time to add constraints and appreciate their effect. The user can also speed-up the optimization by increasing the relaxation parameter T when the current configuration is satisfactory. As shown in Table 4.1, the results obtained with user interaction did not take more time than the ones obtained automatically.

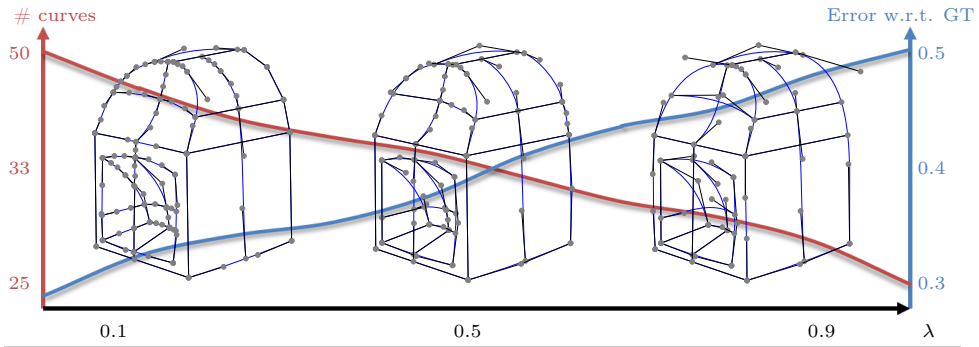


Figure 4.11: Parameter λ controls the balance between fidelity and simplicity. Increasing λ augments error to Ground Truth while reducing the number of Bézier curves, and by extension, the number of control points (shown as grey dots).

Table 4.1: Timing with and without user interactions.

	# hyperedge	# interaction	time
Figure 4.8a (automatic)	25	–	$45s$
Figure 4.8b (interactive)	25	4	$34s$
Figure 4.8c (automatic)	27	–	$25s$
Figure 4.8d (interactive)	26	6	$32s$
Figure 4.9 right (automatic)	673	–	$95s$

4.3.5 Limitations

Our algorithm is not designed to deal with missing data, such as broken strokes. Filling such holes would require extrapolating the curves, which adds significant complexity to the optimization. Note however that the trapped-ball segmentation algorithm [ZCZ+09] for skeleton extraction is robust to small holes. Another limitation of our current optimization is that we only consider Bézier curves, while other primitives such as circular arcs would be better adapted to regular structures in technical drawings. Our algorithm also does not consider high-order geometric regularities such as parallelism, orthogonality or symmetry. Detecting and enforcing such regularities at each iteration of the optimization would be costly if implemented naively.

Our current implementation seeks a uniform trade-off between fidelity and simplicity over the entire drawing. Nevertheless, our Metropolis-Hastings optimization could easily adapt this trade-off locally by taking as additional input a spatially-varying λ parameter, which could be painted by the user or estimated from local image statistics. Finally, while our region-based skeleton extraction effectively merges overlapping strokes in sketchy drawings, it

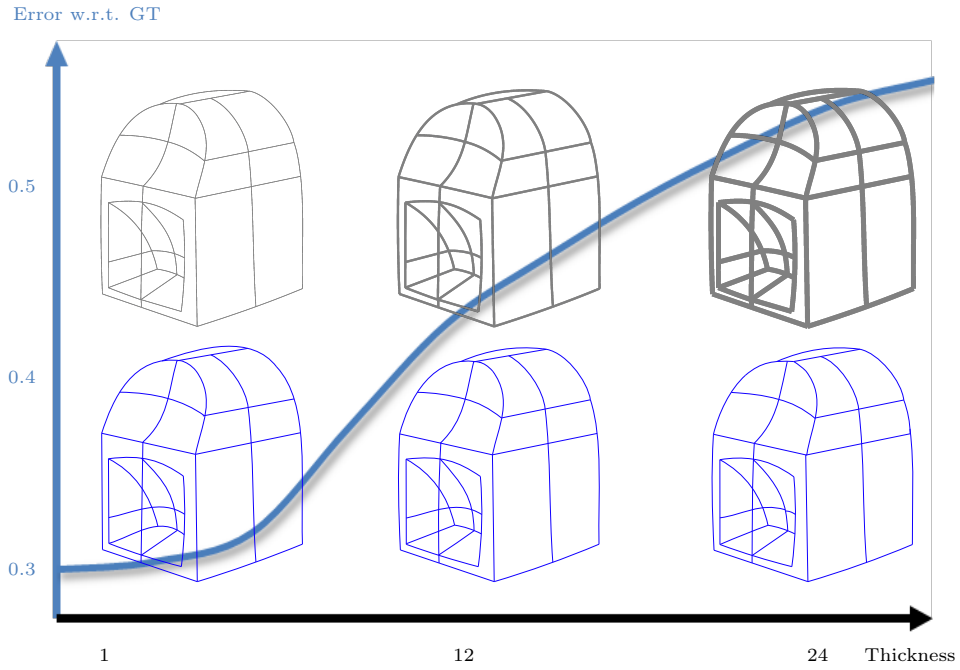


Figure 4.12: Our algorithm produces almost identical output curves for increasingly thick lines. Even at thickness 24, the junctions remain accurate and the average error is bellow 0.6 pixels.

can also remove intended lines since there is an inherent ambiguity between noisy strokes and fine details. When dealing with clean drawings, a smaller trapped-ball should be used.

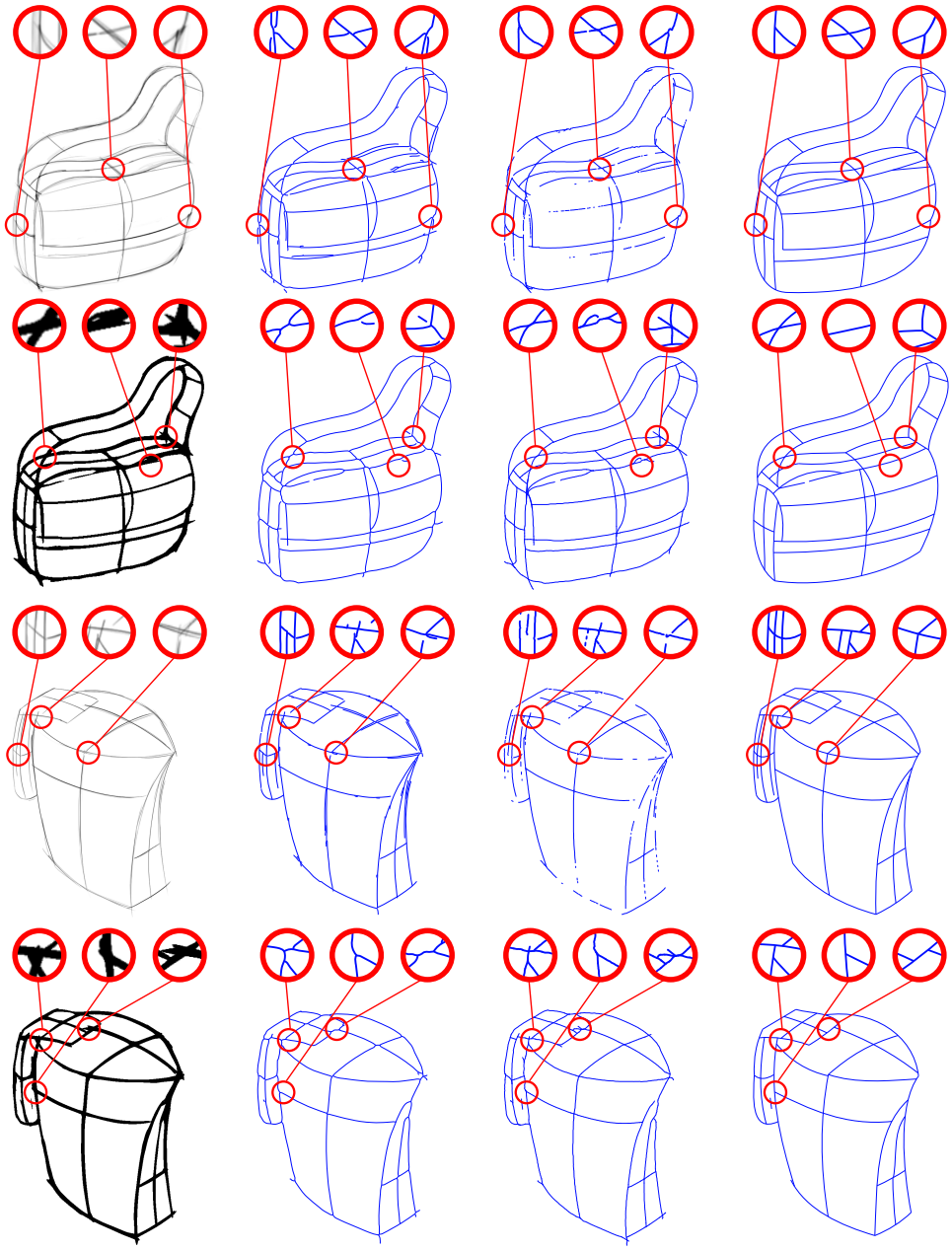
4.4 Conclusion

Skillful vector artists create drawings composed of few curves because they result in clean, compact and easily editable artworks. This observation motivated us to propose the first vectorization algorithm that explicitly attempts to minimize the number of curves and their degree. This new, global objective is also extremely effective in disambiguating line junctions, where prior methods tend to produce spurious short curves. While the resulting optimization involves non-convex and non-local terms, we describe an efficient exploration algorithm to support interactive user control.

Our algorithm takes as input bitmap drawings, which allows it to deal with both scanned drawings as well as rasterized digital drawings. Nevertheless, we hope that our energy formulation will inspire novel algorithms dedicated to the simplification of digital drawings composed of vector strokes.

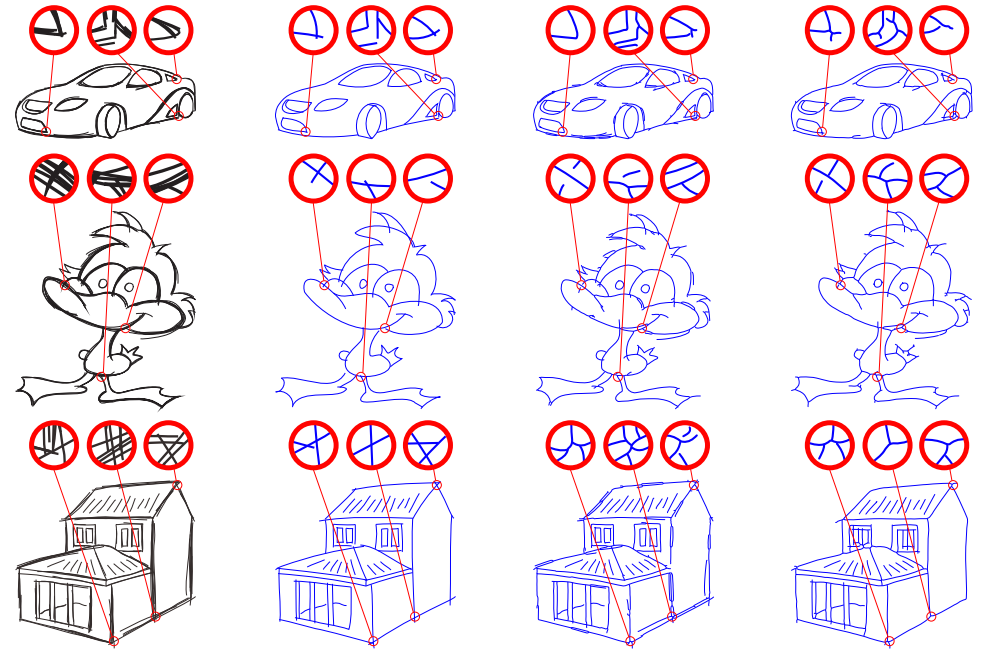
Our idea of minimizing the complexity of the output representation also

has great potential for the vectorization of color images, in particular to extract layers that compactly represent transparency and occlusion effects [RLMB+14]. However, this new domain raises specific challenges, since the optimization should evaluate many interpretations of the shape and color of image regions. In next chapter, we present an effective solution to this color vectorization problem.

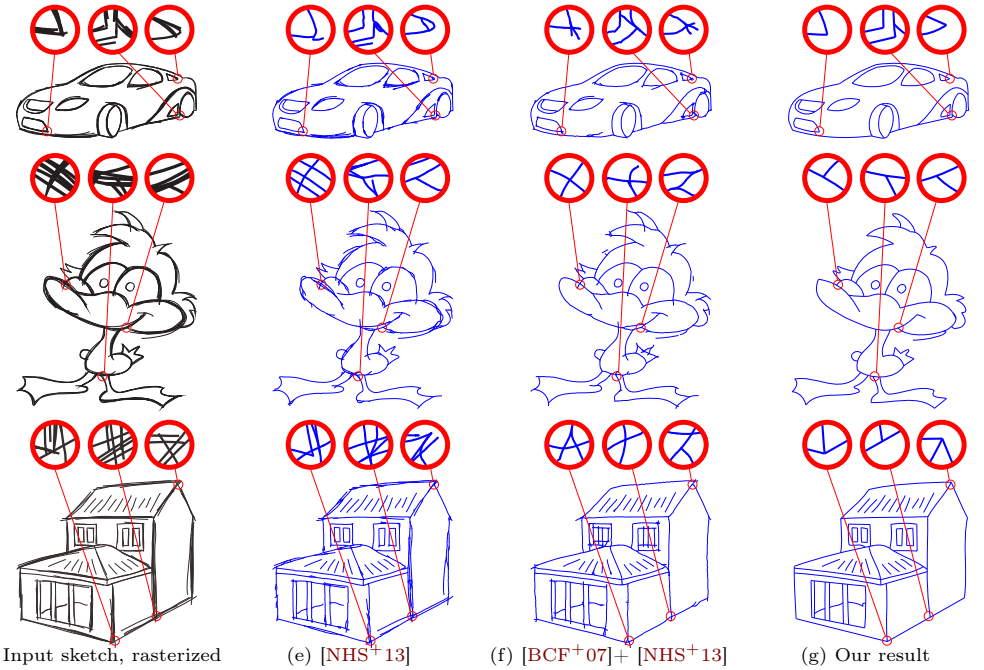


(a) Input sketch (b) Adobe Illustrator CC (c) [NHS+13] (d) Our result

Figure 4.13: Comparison to existing vectorization algorithms. In the 2nd and 4th row, the sketches were pre-processed with the Gabor filter bank from [BCF+07] to group neighboring strokes into thick lines. Existing methods produce multiple curves on sketchy lines and extraneous curves at junctions of thick lines. In contrast, our method recovers precise junctions by favoring the simplest interpretation.



(a) Input sketch, rasterized from ©[LWH15] (b) ©[LWH15] (c) Adobe Illustrator CC (d) [BCF+07] + Adobe Illustrator CC



Input sketch, rasterized (e) [NHS+13] (f) [BCF+07]+ [NHS+13] (g) Our result

Figure 4.14: Comparison to existing line drawing simplification and line drawing vectorization algorithms. In column (d) and (f), the sketches were pre-processed with the Gabor filter bank from [BCF+07] to group neighboring strokes into thick lines. **Note that [LWH15] takes vector strokes as input**, while all other methods deal with bitmaps. Our approach produces results on par with [LWH15] even though we process bitmaps rather than vector strokes. Compared to existing vectorization algorithms, our method produces more accurate curves and junctions despite the high ambiguity of the sketchy input.

Chapter 5

ClipArt vectorization

In the previous chapter, we proposed a new approach to vectorize rough line drawings by minimizing the complexity of the output. In this chapter, we extend this approach to vectorize color images.
More precisely, our approach seeks to decompose the input image into a small number of semi-transparent layers, each layer filled with a constant color or a two-color linear gradient.

(a) Input image and segmentation (b) Multi-layer abstraction and vectorization (c) Editing

Figure 5.1: Given a segmented bitmap image as input (a), our method generates an abstract, layered vector clipart, where each layer is filled with an opaque or semi-transparent linear color gradient (b). By expressing the image as a stack of linear color gradients, our vector graphics reproduce the visual style of traditional cliparts and are easy to edit (c). In this example, we turned the lady bug blue (top) and replaced its dots by little stars (bottom). The black dots in the segmentation indicate opaque regions selected by the user to initialize or constrain our algorithm. Ladybug by Alex Staroseltsev on Shutterstock.com

Our algorithm uses the local decompositions obtained from X-junctions to initialize a region-growing approach, which progressively assigns new pixels to their best-fitting existing layers and only creates a new layer when no good assignment is found. If no X-junction exists, we ask the user to indicate

an opaque region in the image to initialize the region-growing process. However, growing the layers in a greedy manner would quickly lead us to poor local minima of our energy formulation. Our algorithm explores multiple assignments concurrently by building a tree of candidate configurations. Each node of the tree corresponds to an intermediate decomposition where only a subset of the image pixels have been explored, while the leafs of the tree correspond to all the possible decompositions of the entire image. We present a Monte Carlo Tree Search algorithm [BPW+12] guided by image junctions to quickly identify the low-energy branches of this tree. We further accelerate the search by pre-segmenting the image into smooth color regions, turning our approach into a region labelling rather than pixel labelling problem.

While our algorithm can produce plausible vector cliparts automatically, we allow users to refine the result by indicating opaque regions. We demonstrate our method by reverse-engineering bitmap cliparts and by creating a range of new cliparts from photographs.

5.1 Problem Formulation

Our goal is to estimate a multilayer vectorial representation of an input image I, where each layer is composed of

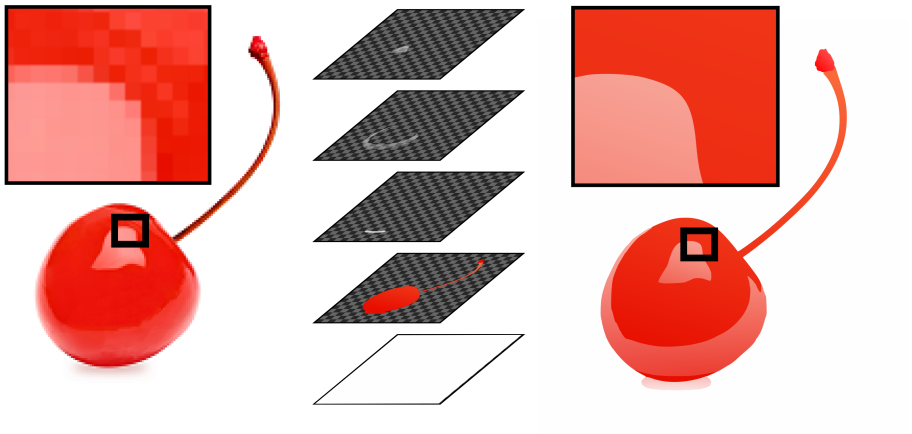
- A supporting domain D covering a subset of pixels from the input image,
- A color gradient function $C(p)$ that associates an RGB color to each pixel $p \in D$,
- An opacity gradient function $A(p)$ that associates an opacity to each pixel $p \in D$. We set $A(p) = 0$ when $p \notin D_n$.

We synthesize the output image I_n from a n-layer representation by recursively α-blending the ordered layers

$$I_n(p) = A_n(p)C_n(p) + (1 - A_n(p))I_{n-1}(p) \tag{5.1}$$

where C_n and A_n are the color gradient function and opacity gradient function of layer n. Figure 5.2 illustrates this representation.

Linear gradients. While the above formulation is general, in this work we restrict the color and opacity gradients to linear forms with the same orientation, which corresponds to the 2-stop linear gradient of the SVG standard. The color and opacity gradients functions are expressed as $C(p) = c_0 + c_1 O^t p$ and $A(p) = a_0 + a_1 O^t p$ where c_0 and c_1 (resp. a_0 and a_1) are color vectors



(a) Input (b) Layers (c) Output

Figure 5.2: Multilayer representation. Our goal is to represent the input image (a) as a stack of opaque and semi-transparent layers (b) such that α-blending the layers using Eq. 5.1 reproduces well the input (c). Cherry by M. Unal Ozmen on Shutterstock.com

(resp. opacity scalars) and O is the orientation vector. While linear gradients are less expressive than more complex primitives, such as gradient meshes [SLWS07] and diffusion curves [OBW+08], they are easier to manipulate thanks to their small number of parameters and are supported by most vector graphics software. In addition, many vector artists employ linear gradients as a means to simplify and stylize the image. Restricting our algorithm to linear gradients allows us to reproduce this characteristic style of vector cliparts.

Pre-segmentation. Recovering the color, opacity and supporting domain of each layer at each pixel is a formidable task. We reduce the complexity of this task by pre-segmenting the image into smooth regions and by imposing that all pixels of a region share the same layers. While our algorithm produces convincing results from automatic segmentation, we achieved our best results using user-assisted segmentation as detailed in Section 5.3. In addition, we apply a 3-pixel wide erosion on each region to exclude border pixels since those often contain a mixture of colors from neighboring regions. We process these pixels in a separate step once the multi-layer representation is computed, as detailed in Section 5.2.6.

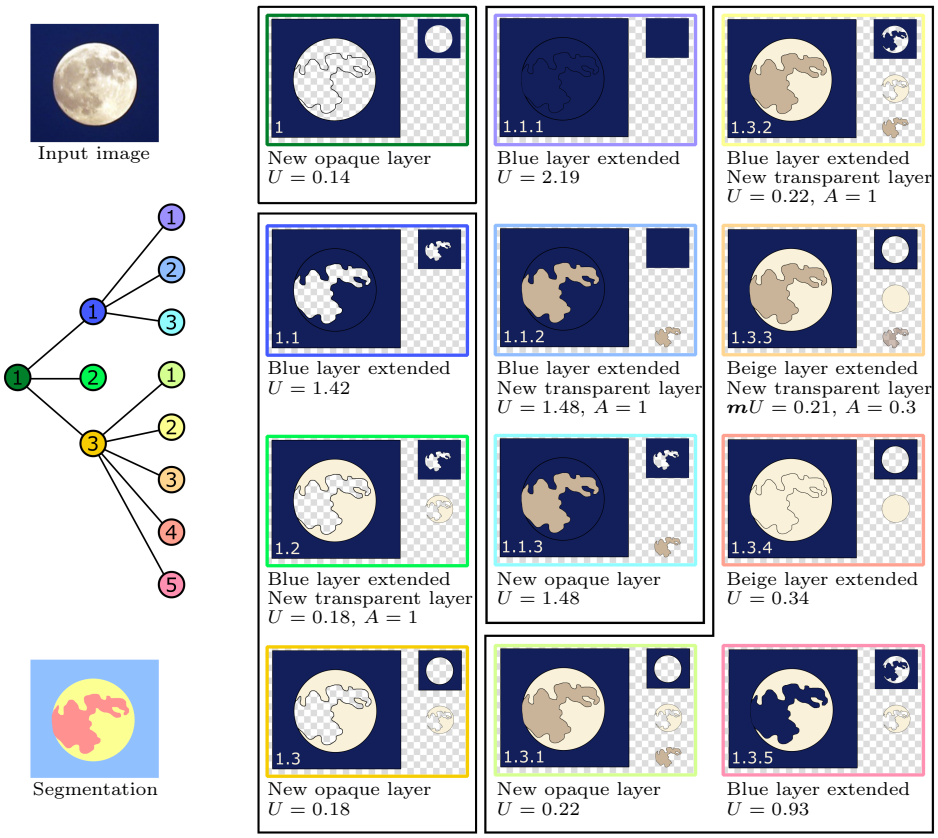


Figure 5.3: Overview of our exploration mechanism on a simple example. Given an input image and its region segmentation, we cast the exploration of the solution space as a tree search. Each node of the tree corresponds to an intermediate solution where only a subset of the regions has been decomposed. Each branch of the tree expands an intermediate solution by adding a region to the decomposition. The new region can be assigned to one or several existing layers, as well as to a new layer. Leafs of the tree correspond to complete decompositions. We use Monte Carlo Tree Search to quickly reach low-energy leaves. In this example, the best solution is reached at leaf 1.3.3, where the moon is represented as an opaque beige layer with a semi-transparent brown layer on top. Note that in some configurations, such as node 1.2, a region can be assigned to a new semi-transparent layer which receives an opacity value of 1 after optimization. We convert such layers to opaque when this situation occurs. Moon by stux on pixabay.com

5.1.1 Energy formulation

We denote by $\mathbf{x} = (D_1, C_1, A_1, ..., D_n, C_n, A_n)$, an output vectorial representation composed of n layers. We define three criteria to measure the quality

of a configuration $\mathbf{x}$:

- Faithfulness to input data - the reconstructed image I_n has to be similar to the input image I,
- Simplicity of the decomposition - the number of layers should be minimized to yield a compact and editable representation,
- Simplicity of opacity functions - semi-transparent layers should be favored over opaque ones to reduce occlusion.

Based on these criteria, we formulate an energy U of the form

$$U(\mathbf{x}) = (1 - \lambda)U_{fidelity}(\mathbf{x}) + \lambda U_{simplicity}(\mathbf{x}) \tag{5.2}$$

where $U_{fidelity}$ measures the faithfulness to input data and $U_{simplicity}$ accounts for the overall simplicity of the reconstruction. The parameter $\lambda \in [0, 1]$ weights the two terms.

We define $U_{fidelity}$ as the RGB error under L_2 norm between the input image I and the reconstructed image I_n:

$$U_{fidelity}(\mathbf{x}) = \frac{1}{|I|} \sum_{p \in I} \|I(p) - I_n(p)\|_2^2. \tag{5.3}$$

We express $U_{simplicity}$ as

$$U_{simplicity}(\mathbf{x}) = \frac{1}{N} \sum_{l=1}^{n} w_l \tag{5.4}$$

where N is the maximal number of layers and w_l penalizes opacity functions according to their simplicity. We set $w_l = 1$ for semi-transparent layers and $w_l = \beta > 1$ for opaque layers. Figure 5.2 illustrates the intuition behind this strategy: the highlight on the cherry could be either interpreted as an opaque pink region surrounded by a red region, or as a semi-transparent white region over a red region. Setting $\beta > 1$ favors the latter interpretation, which facilitates subsequent editing such as changing the color of the cherry. We used $\beta = 1.2$ in all our examples. In addition, we convert a semi-transparent layer to opaque if its estimated opacity is above 0.999.

5.2 Exploration mechanism

Searching for the configuration that minimizes energy U is a complex optimization problem which combines discrete variables (the number of layers and their supporting domains) and continuous variables (the parameters of the linear gradient functions). While the number of layers can be infinite, we

can reasonably assume that there is at most as many layers as input regions. Yet, since a region can appear in at most all N layers, finding the global minimum would require estimating the continuous variables for 2^{N^2} layer assignments, which is impractical despite this upper bound.

We address this combinatorial challenge by devising a scalable exploration strategy of the solution space, which we illustrate on a simple example in Figure 5.3. Our exploration starts from initial decompositions of a few regions, which we obtain by exploiting transparency assumptions on X-junctions, or by asking the user to indicate one or more opaque regions. We then expand these decompositions to the regions adjacent to the ones already decomposed. Each new region can either be assigned to one or more existing layers, or to a new layer, for which the color gradient is estimated. We thus obtain a set of possible assignments of the new regions, each forming a decomposition that can be further expanded to adjacent regions. We can represent the ensemble of decompositions generated by this approach as a tree, where each node corresponds to an intermediate decomposition, which branches to multiple decompositions after an expansion step[1]. The leafs of this tree correspond to all the 2^{N^2} possible decompositions, and our goal is to find a low-energy leaf without exploring the entire tree.

We propose a stochastic algorithm for fast exploration of the solution space, which can be seen as a form of Monte Carlo Tree Search (MCTS) [BPW+12]. The main idea of this method is to build the solution tree incrementally and asymmetrically using a balance between exploration of the space and exploitation of the already explored configurations. Figure 5.4 illustrates the four successive operations performed at each iteration of an MCTS method: (i) selection of a node according to a tree policy, (ii) expansion of the tree by adding child nodes, (iii) evaluation of the quality of the added child nodes (also called *reward*), and (iv) update of the tree policy by back-propagation to the root. Algorithm 2 summarizes this process.

Our main technical contribution resides in defining efficient expansion and reward operations that exploit characteristics of our problem. We denote by $\mathbf{x}_{/t}$ an intermediate decomposition restricted to the regions visited between the root and node t. We measure the energy $U(\mathbf{x}_{/t})$ by restricting I to the visited regions (Eq. 5.3), and by setting the maximal number of layers N to the number of visited regions (Eq. 5.4).

[1] In theory, the tree of solutions is actually a directed acyclic graph because multiple paths can yield the same configuration. However, in practice such cases are very rare because our acceleration heuristics trim many branches of that graph.

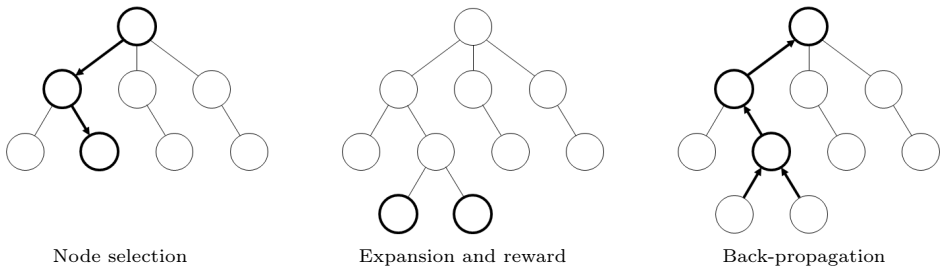


Figure 5.4: Exploration mechanism by Monte Carlo Tree Search (after [BPW+12]). A node is first selected using a tree policy (section 5.2.1). New nodes are then added (section 5.2.2) and their corresponding energy are computed (section 5.2.3). The energy of the new nodes is back-propagated to their parent node to discourage exploring bad configurations (section 5.2.4).

Algorithm 2 Exploration mechanism

Initialize child nodes of the root (sec 5.2.5)
repeat
  Node selection: select a leaf node using tree policy (sec 5.2.1)
  Expansion: generate child nodes (sec 5.2.2)
  Reward: compute energy of child nodes (sec 5.2.3)
  Back-propagation: update tree policy (sec 5.2.4)
until stopping criterion (sec 5.2.5)

5.2.1 Node selection

The tree policy seeks to favor the exploration of high-quality configurations. To do so, each child node t_i of a node t is associated to a parent-to-child probability that is proportional to its energy and the energy of its siblings

$$P_{t \to t_i} = \frac{\exp(-U(\mathbf{x}_{/t_i}))\mathbb{1}_{\{t_i \in t^\star\}}}{\sum_{t_j \in t^\star} \exp(-U(\mathbf{x}_{/t_j}))} \tag{5.5}$$

where $t^\star$ is the set of child nodes of t and $\mathbb{1}$ is the indicator function. The algorithm selects a terminal node for expansion according to the product of parent-to-child probabilities between the root and the node, i.e. the overall probability of reaching this configuration.

We further accelerate the search by restricting $t^\star$ to the k children nodes with the lowest energies. The parameter k offers a balance between spreed and accuracy of the algorithm. We set k to 2 in all our experiments.

5.2.2 Tree expansion

Once a candidate terminal node is selected, the tree expansion step generates its child nodes by adding one or more regions to the ones already visited. We first present a naive generator that simply adds one region with all its possible layer assignments. We then detail a more efficient generator that exploits a transparency assumption on X-junctions to add multiple regions at once. We detect X-junctions as cliques of order 4 in the adjacency graph of the regions.

Single-region expansion. This generator inserts an unvisited region to the decomposition associated with the selected node. This region is chosen randomly among the regions adjacent to the ones already present in the decomposition. We assume that the new region can be part of any layer of its adjacent regions in the decomposition, as well as part of a new layer. Note that we do not consider layers of non-adjacent regions, which drastically reduces the number of child nodes while ensuring that each layer contains a single connected component by construction. Given the M layers from the adjacent regions, a naive enumeration gives 2^{M+1} child nodes. However, since opaque layers occlude all layers below them, we can discard all but one of the configurations that have the same visible layers, which further reduces the number of child nodes.

X-junction expansion. This generator only applies on the nodes for which the decomposition contains part of an X-junction, and expands the decomposition to all four regions forming the X-junction. We build on the assumption that an X-junction results from the boundary of a semi-transparent layer crossing the boundary of two other layers [SC11]. This assumption yields a limited set of possible interpretations, which depends on how many of the four regions are already present in the decomposition of the selected nodes:

- None of the regions of the junction have been decomposed. This configuration occurs when we initialize the algorithm, i.e. when the X-junction is used to expand the root of the tree. In the absence of other information, we make the additional assumption that the junction is formed by one semi-transparent layer over two opaque layers, which yields 4 possible configurations illustrated in Figure 5.5.
- One or more regions of the junction are already in the decomposition. We can deduce that the boundary between an unknown region and a known one is either caused by the end of a known layer or the beginning of a new one. Denoting M the number of layers in an adjacent known

region, we obtain $M + 1$ configurations if the unknown region is in the 4-neighborhood of the known one (i.e. the two regions are separated by one boundary), and $M + 2$ configurations if the unknown region is in the 8-neighborhood of the known one (i.e. the two regions are separated by two boundaries).

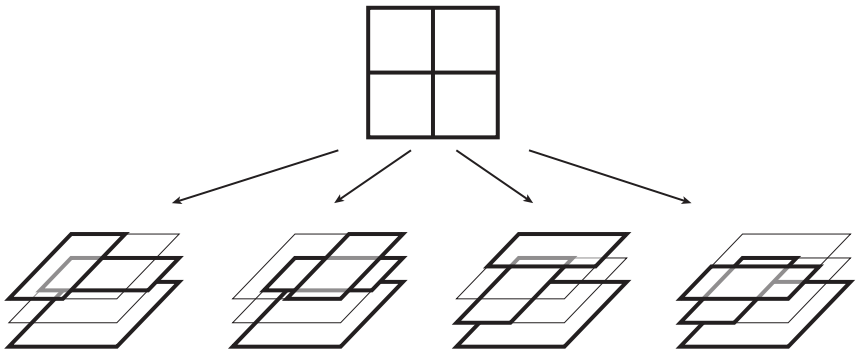


Figure 5.5: We assume that X-junctions are formed by a semi-transparent layer crossing two other layers. When none of the layers are known, we assume that the two bottom layers are opaque, yielding 4 possible configurations.

We call the X-junction generator in priority since it allows faster expansion of the tree than the single-region generator.

5.2.3 Reward

Given the layer assignment of a child node, we can estimate its energy along with the color and opacity gradient function of each layer by minimizing Equation 5.3. However, this optimization is highly non-linear, which requires the use of an iterative algorithm like Newton-Raphson with a good initialization of the solution. We propose three strategies to avoid computing such a costly optimization for most visited nodes of the tree.

Layer re-use. Our first strategy to speed up the evaluation of Equation 5.3 is to re-use the color and opacity gradients of a parent node when present in a child node. In other words, we extend the known gradient functions over the new regions without re-fitting them. While this approach is approximate, it yields a significant gain in performance with a marginal loss in visual quality, as evaluated in Section 5.3.

Exploitation of X-junctions. As detailed above, expanding the decomposition at an X-junction only yields configurations with at most one new

semi-transparent layer. In addition, our assumption on the nature of X-junctions tells us that this semi-transparent layer covers the boundary between two regions with known layers. Our strategy is to deduce from this setup a configuration where the semi-transparent layer is observed over two known colors, which makes its estimation well-posed [SB96, RLMB+14].

Denoting l the unknown layer, each of the two regions on which it appears gives us equations of the form $I_l(p) = A_l(p)C_l(p) + (1 - A_l(p))I_{l-1}(p)$ where I_l is the input image from which we have substracted all semi-transparent layers above l and I_{l-1} is the image formed by all layers below l, which are known for the two regions considered. The challenge is now to express these equations over the same pixels, so that we can remove the term $A_l(p)C_l(p)$ by substitution and leave $A_l(p)$ as the only unknown.

Figure 5.6 illustrates our algorithm on a toy example, where the new layer covers two regions I_l^1 and I_l^2, which share layers with their known neighboring regions I_{l-1}^1 and I_{l-1}^2 respectively. We first extend each region over all three other regions so that all regions share the same spatial domain. Regions that are already in the decomposition are easy to extend since all their layers are known and have a parametric form. However, regions covered by the new layer are still in a bitmap form. Our solution consists in approximating each such region with a polynomial function using a least square fit on all pixels. Since each layer is represented by a linear gradient, a new region can at best be represented by a polynomial of degree $M + 1$, where M is the number of layers shared with the adjacent regions. We then extend the resulting parametric functions over the other regions to know their values over all pixels. We now have the necessary ingredients to form two equations at each pixel

$$\tilde{I}_l^1(p) = A_l(p)C_l(p) + \big(1 - A_l(p)\big)\tilde{I}_{l-1}^1(p) \tag{5.6}$$

$$\tilde{I}_l^2(p) = A_l(p)C_l(p) + \big(1 - A_l(p)\big)\tilde{I}_{l-1}^2(p), \tag{5.7}$$

where $\tilde{I}_j^i$ denotes the extended version of region I_j^i. Subtracting the two equations gives us an expression where $A_l(p)$ is the only unknown

$$\tilde{I}_l^1(p) - \tilde{I}_l^2(p) = \big(1 - A_l(p)\big)\big(\tilde{I}_{l-1}^1(p) - \tilde{I}_{l-1}^2(p)\big). \tag{5.8}$$

Once $A_l(p)$ is known the minimization of Equation 5.3 becomes quadratic in $C_l(p)$ and has a unique solution.

Note that since each region of the image is visited multiple times during the tree search, we only compute its polynomial representation at a given degree the first time it is needed, and keep it in memory for later use.

Sub-sampling. Our last strategy to reduce computational burden is to use a sparse uniform random sampling of the input image pixels when fitting the color and opacity gradients of the layers. However, using very few

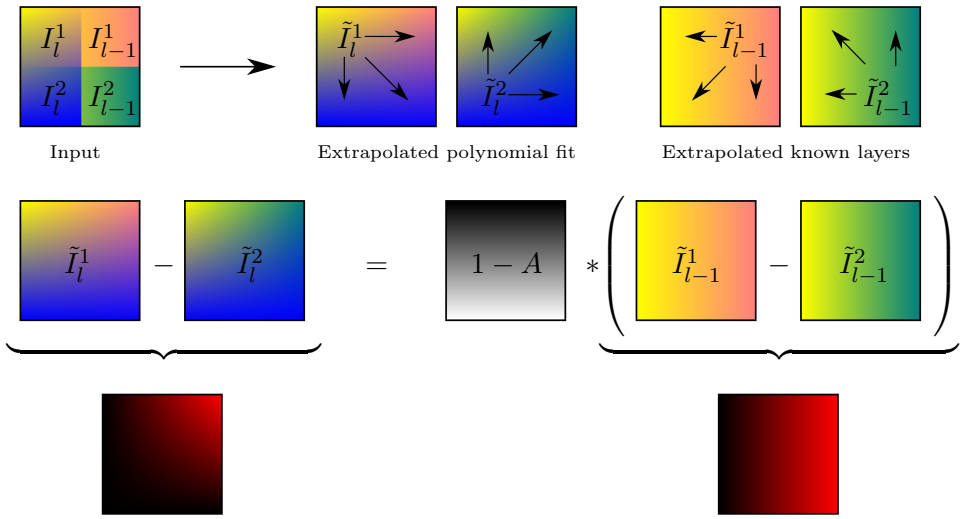


Figure 5.6: Derivation of the opacity of a new layer at an X-junction. Let us assume that the new layer is shared by the two regions I_l^1 and I_l^2. Furthermore, region I_l^1 shares all of its other layers with region I_{l-1}^1, while region I_l^2 shares all of its other layers with region I_{l-1}^2. We exploit this redundancy to deduce the opacity of layer l. We first extend each region over the other regions to know their values at each pixel (top). Regions I_{l-1}^1 and I_{l-1}^2 are trivially extended since their layers are known and have a linear form. We extend regions I_l^1 and I_l^2 by fitting a polynomial on their pixel colors. We can then compute a per-pixel opacity A by simple arithmetic (bottom).

samples raises the risk of making the method very sensitive to image noise. Fortunately, we can leverage the polynomial approximation of the image introduced above as a means to remove high frequency content that cannot be captured by the layers. While each polynomial needs to be computed from all pixels of its region, this is a one time computation which is quickly amortized over all the nodes where the layers of a given region need to be estimated.

To sum-up, we first generate for each node a number of configurations, which form its child nodes. When a child node does not contain any new layer, computing its energy U by Eq. 5.2 is trivial since the parameters of all layers are known. When new layers are involved, the parameters of their color and opacity gradient functions must be estimated first. If the new layer is opaque and below known semi-transparent layers, or if the new layer is part of an X-junction, we can estimate the gradient functions as a quadratic minimization problem that has a unique solution. Otherwise, we run the Newton-Raphson algorithm. Both the quadratic minimization and

the Newton-Raphson algorithm are computed with a sparse sampling of the input image.

To further reduce the complexity of the tree, we remove the child nodes whose energies are one order of magnitude higher than the best child node.

5.2.4 Back-Propagation

Once the energy of new child nodes is computed, we update the tree policy to favor the visit of low energy configurations. We back-propagate information on child node energy towards the root by giving to each node t the lowest energy of its child nodes

$$U(\mathbf{x}_{/t}) \leftarrow \min_{t_i \in t^\star} U(\mathbf{x}_{/t_i}). \tag{5.9}$$

5.2.5 Initialization and stopping criterion

We initialize the exploration by generating one root node for each X-junction, using as initial decompositions of these junctions the configurations that minimize $U_{fidelity}$. In the cases where no X-junction exists, we ask the user to seed the search by selecting an opaque region in the image. We display a black dot over the selected region of the segmentation for each result where such indication was provided. We stop the exploration once we have reached $4N$ leaves of the tree, with N the number of regions in the image, and keep the decomposition with the lowest energy as our final solution. We adjusted this stopping threshold experimentally by running our algorithm multiple times on the same image using a high threshold, keeping track of the number of leaves visited before reaching the best configuration.

5.2.6 Finalization

As a reminder, we excluded pixels on the boundary of each region of the segmentation, since those pixel are often corrupted by blur or anti-aliasing. The last step of our algorithm assigns each such pixel to one of its neighboring regions based on goodness of fit. In addition, when an opaque layer is surrounded by another opaque layer, we position the surrounding layer below the surrounded one and extend it to cover the hole, which is occluded by the surrounded layer. As an example, the red layer of the lady bug in Figure 5.1 extends below the black dots.

Finally, we vectorize the supporting domain of each layer using Potrace [Sel17].

Figure 5.7: Layered cliparts produced with our method from three studio photographs and a bitmap diagram. We used manual segmentation to properly delineate small and blurry regions (letters on the stop button, highlights on the tomato) and to ignore spurious details (arrows on the chart). Stop button and battery by Photo Melon, chart by Allies Interactive, tomato by bajinda on Shutterstock.com

5.2.7 User interaction

We offer users several means of controlling our method. First, we expose the global λ parameter, which balances fidelity to the input with simplicity of the output. We found that clean bitmap cliparts can be decomposed using a low λ value of 0.1, while photographs often require a higher value of 0.5 to abstract away small details, non-linear color variations, and image noise. We also allow users to increase β to favor the extraction of semi-transparent layers, although we used the default value for all results in this chapter.

Second, we allow users to impose that a region of the image be expressed as a single opaque layer. We implement this constraint during the tree search by only generating child nodes with opaque layers over the selected region. We display a black dot over the selected regions of the segmentation for each result where such additional constraint was provided.

Finally, users can also control the appearance of the decomposition via the input segmentation, for instance to approximate fine details or complex color variations with a unique region.

5.3 Experiments

Figure 5.1 and Figure 5.7 show results of our method on a variety of illustrations and studio photographs. Our approach is especially effective at expressing highlights and shadows with semi-transparent layers, which facilitates subsequent manipulation like color and shape editing, as shown in Figure 5.1(c).

We now compare our method to existing work in the field and evaluate the impact of parameters and performance.

5.3.1 Comparisons with existing work

Figure 5.8 compares our method with the work by Tan et al. [TLG16] and Richardt et al. [RLMB+14]. The method by Tan and colleagues targets the different application of bitmap color editing. As a result, while their method succeeds in separating the image into layers based on the dominant colors of the image, each layer is a bitmap with a constant color and spatially-varying transparency rather than a vector path filled with a parametric gradient.

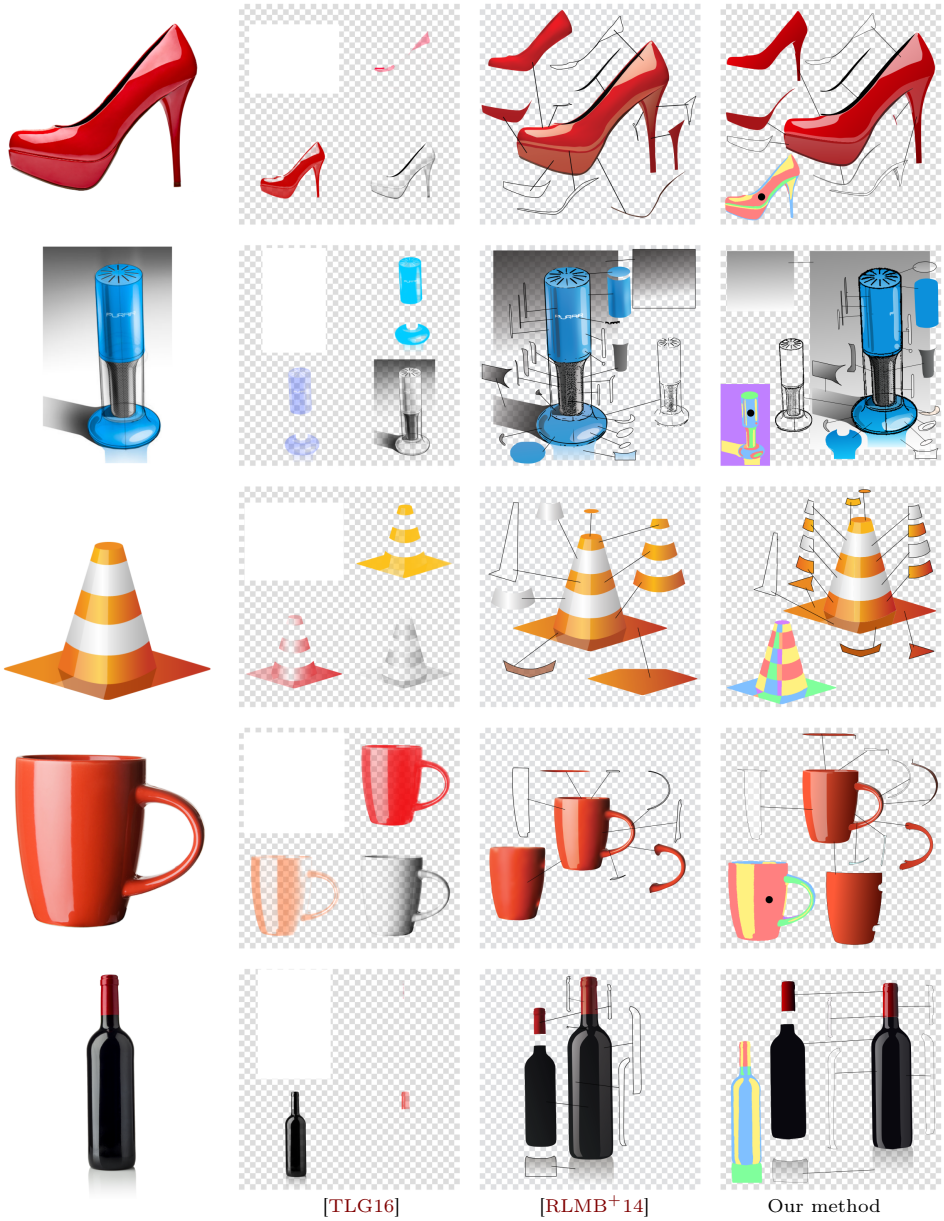


Figure 5.8: Comparison with existing decomposition methods. The layers generated by [TLG16] are bitmaps filled with a constant color and spatially varying transparency, which is suitable for color editing but not for other applications of vector graphics. Our results are similar to the ones by [RLMB+14], although our method requires less user intervention and solves for the layer parameters more efficiently. All our results were produced with a manual segmentation, except the cone (3rd row). Cup by George Dolgikh, shoe by Picsfive, wine bottle by Dmitri Gristsenko on Shutterstock.com, air purifier by Spencer Nugent on sketch-a-day.com

Our goal and approach is more similar to the work by Richardt et al., and our method produces similar results to theirs. The main difference between the two methods resides in the user workflow. The interactive workflow of Richardt et al. requires users to extract each layer of the decomposition one by one, in a front to back order. Since their layer extraction takes between a few seconds to a minute per layer, their results took between several minutes to an hour to create, as detailed in Section 5 of their paper. In contrast, users of our system simply have to provide a segmentation of the image, which takes a few minutes with an interactive tool, and optional indications of opaque regions before letting our algorithm produce the entire decomposition within seconds. Another difference resides in the family of gradients supported by the two methods. Our method extracts 2-stop gradients, while Richard et al. also support 3-stop gradients, but requires users to indicate the number of stops for each region. This is why their method extracts each color band of the cone as a single 3-stop gradient while we extract them as two 2-stop gradients (Figure 5.8, 3rd row). In addition, Richard et al. represent opaque layers with gradient meshes, while we only use linear gradients. Their gradient mesh better captures contrast on the opaque layer of the cylindrical mug (Figure 5.8, 4th row).

5.3.2 Impact of parameters

The main parameter of our algorithm is λ, which controls the amount of abstraction of our vectorization. Figure 5.9 details its impact on the vectorization of a bitmap clipart. A low value of λ reproduces the input as closely as possible with linear gradients. Increasing λ favors the use of semi-transparent layers at the price of more approximate rendition of the original colors. A high value of λ abstracts away details in an effort to reduce the number of layers.

5.3.3 Impact of pre-segmentation

Figure 5.10 compares the output of our method with different input segmentations generated with a Mean Shift algorithm [CM02] and a manual segmentation. Our method produces visually similar vectorizations with two different automatic segmentations. In particular, multiple regions are merged to form a few layers thanks to our simplicity term. However, we obtain more stylized cliparts using manual segmentations.

5.3.4 Performance

Table 5.1 provides timings for several of our results, measured with a 3rd generation core i7 processor and 16Gb DDR3 memory. Our algorithm took less

Table 5.1: Our method generates layered vectorizations within seconds. The cake image is one of the slowest to process because it has many regions and no X-junctions.

Image	Exploration	Finalization	Total	# regions	# clicks
Lady bug	28s	6s	35s	26	3
Cherry	5s	2s	7s	8	1
Stop	16s	12s	28s	23	2
Tomato	9s	6s	16s	19	1
Diagram	12s	1s	13s	11	3
Battery	44s	15s	60s	24	2
Cake	71s	148s	220s	30	1
Egg	7s	2s	9s	6	1
Red shoes	5s	11s	17s	11	1
Purair	16s	4s	21s	20	1
Cone	44s	3s	48s	27	0
Cup	17s	16s	33s	11	1
Wine	13s	13s	26s	11	0
House	337s	159s	496s	105	3

Table 5.2: Ablation study. We evaluated the performance of our algorithm on the smartphone image (Figure 5.10) after removing some of its key components. Removing the exploitation of X-junctions increases computation time by one order of magnitude, while re-fitting all layers for each child node further increases computation by two orders of magnitude without significant change in the energy. Similarly, augmenting the maximum number of child nodes k slows down the algorithm without much impact on the energy. We provide the resulting vector cliparts as supplemental material, they are all visually similar.

X-junctions	yes	yes	no	no
k	2	4	2	2
Layer re-use	yes	yes	yes	no
Energy U	1.00	1.04	0.870	0.869
Time (s)	14	28	307	29890
Number of visited nodes	318	332	54358	52357

Figure 5.9: Impact of parameter λ. A low value of λ favors accurate reproduction of the input, but produces a decomposition dominated by opaque layers (left). Increasing λ turns many of the opaque layers semi-transparent (middle). A high value reduces the number of layers, at the price of missing details (right). Cake by vectorsme on openclipart.org

than a minute to produce each decomposition, which allows an interactive workflow where users can refine the result by adding a new opacity constraint or modifying the segmentation if necessary. The last column of Table 5.1 details the number of opaque regions selected by the user to initialize the optimization.

Table 5.2 compares several versions of our algorithm where we removed important features. In particular, exploiting X-junctions greatly speeds up the exploration of the solution space, while increasing the maximum number of child nodes does not yield a significant increase in quality. We also implemented re-fitting of all layers for each child node, which increases timing by two orders of magnitude without much impact on the energy.

5.3.5 Limitations

Our choice of restricting the image representation to linear gradients is key to achieve joint abstraction and vectorization. However, our current implementation does not support variants of linear gradients, such as radial gradients, although such primitives could be included in the configurations

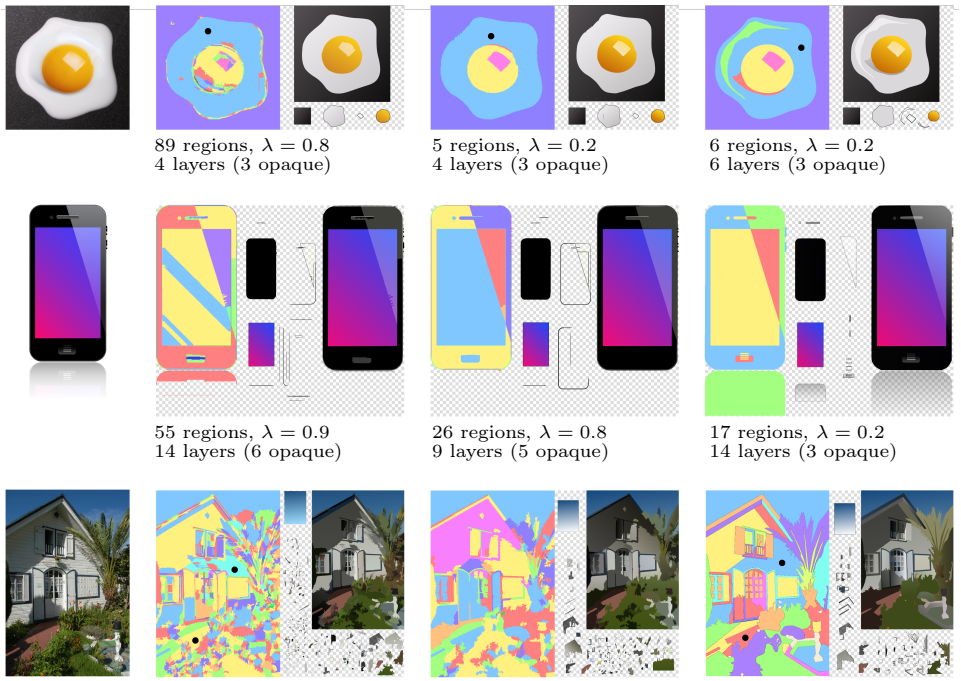


Figure 5.10: While our method produces consistent results with different automatic segmentations (b,c), we achieved our best results by refining the segmentation by hand (d). In particular, while our method can fuse small regions to reduce the number of layers, spurious high-contrast details remain, such as thin highlights along the boundary of the smartphone. In contrast, a manual segmentation allows users to remove unwanted details while preserving others, such as an extra highlight on the egg, the button of the smartphone, and the window tiles on the house. Note that the shadows on the house and ground are extracted as semi-transparent layers. Egg by Valentina Razumova, house by Stefano Ember, smart phone by Gor Grigoryan on Shutterstock.com

evaluated by the tree search. More advanced primitives like gradient meshes would be more difficult to integrate since they cannot be trivially expanded to adjacent regions, which is a key assumption made by our algorithm.

Our method targets piecewise-smooth images and as such reaches its limits in the presence of texture. A low-contrast texture is averaged-out if segmented as a single region, while high-contrast textures can be segmented in multiple small regions, which result in a cluttered clipart with many layers. The house in Figure 5.10 illustrates how our approach performs on a complex natural image with textures.

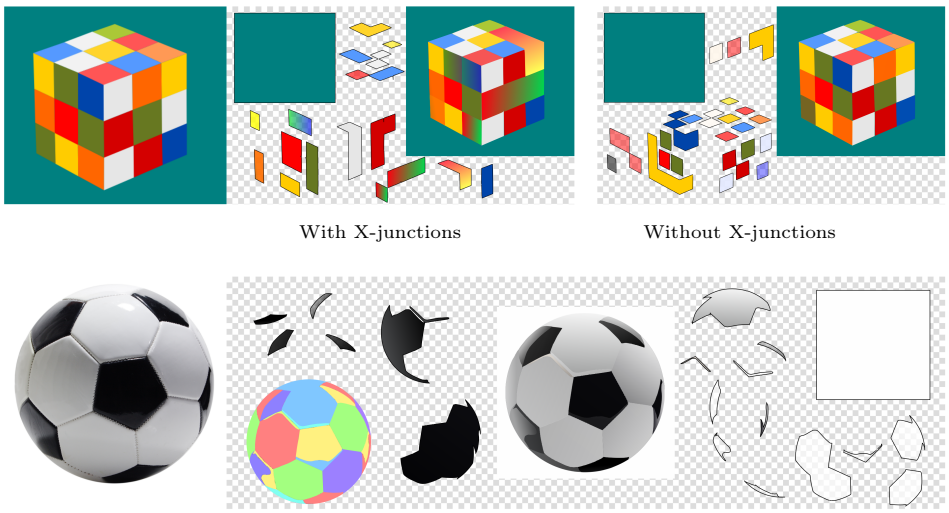


Figure 5.11: Limitations. Top: We accelerate our algorithm by assuming that each X-junction is produced by a semi-transparent layer crossing two other layers. This assumption is not valid on the Rubik's cube, where there are multiple X-junctions between tiles of different colors. Exploiting our assumption on this example forces the method to include transparent layers that do not reproduce well the input. Bottom: Decomposing an image into opaque and transparent layers is especially ambiguous when the layers share the same color. In this example, the algorithm cannot make the distinction between a dark shadow over a white panel and a white highlight over a dark panel. As a result, some black panels are interpreted as white and vice-versa. Soccer ball by Le Do on Shutterstock.com

We speed up our algorithm by assuming that X-junctions are caused by transparency. This assumption breaks on texture patterns such as a checker-board, where many X-junctions are not due to transparency. Figure 5.11(top) illustrates the behavior of our algorithm in such cases, with and without the exploitation of X-junctions. While our method produces a faithful vectorization when X-junctions are not exploited, exploiting them forces the algorithm to use transparency, which reduces reconstruction accuracy.

Inverting Equation 5.1 is an ill-posed problem, especially when the foreground and background layers contain the same colors. This ambiguity is illustrated in Figure 5.11 where the soccer ball is only composed of shades of gray. As a result, while the decomposition found by our optimization captures well the appearance of the input image, it does not properly separate the white and black panels of the ball because it cannot make the distinction between a dark shadow over a white panel and white highlight over a dark panel.

5.4 Conclusion

While image vectorization has received significant attention in the computer graphics community, very little work has been done on reproducing the style and layer structure of traditional vector cliparts. Motivated by the ubiquity of linear gradients in vector art, we have presented an algorithm based on Monte Carlo Tree Search to jointly decompose an image into layers and approximate these layers with linear gradients.

Our method takes a segmented image as input and does not attempt to modify the shape of the segmented regions, apart from fusing small segments to crate bigger ones. However, the regions produced by automatic segmentation algorithms often have more intricate shapes than the ones created by vector artists. An interesting direction for future research would be to jointly simplify shape and color during the vectorization process, potentially by including a shape simplicity term in our energy formulation.

In the next chapter, we introduce a generic framework to extract geometric structures in images without dependency to any preprocess. This method could be used to simultaneously sample the region partition and decompose it into a stack of semi-transparent layers.

Chapter 6

Towards a unified algorithm for extraction of geometric structures

In chapters 4 and 5 we proposed two approaches to vectorize images. The first algorithm vectorizes rough sketches into a clean curve network and the second one vectorizes color images into several semi-transparent layers. The main limitation of these two algorithms is the dependency to an over-segmentation, ie a noisy topological graph for sketch vectorization and an image partition into regions for multilayer image vectorization. We observed the same limitation for many computer vision problems including the extraction of networks of blood vessels from retinal images, the extraction of building footprints from urban satellite images, or the extraction of 3D surfaces from multiple photographs of a scene. All these algorithms first extract local primitives and then group them to construct the global structure.

For example, object contouring methods typically detect line segments along image discontinuities before assembling them to form polygons [ZFW+12, SCF14], and multiview stereo reconstruction algorithms extract 3D points by feature matching before interpolating them with a surface mesh [LPK07, MPFB17]. While this two-step approach reduces computational burden, the quality of the resulting structures depends heavily on the local decisions taken during primitive detection.

As an alternative to primitive detection, *generative models* seek to synthesize structures and measure their agreement with image data. In particular, *point processes* have shown their ability to generate configurations of geometric elements that align with image content [Lie94]. However, the synthesis of large-scale structures often requires strong interactions between geometric primitives, which are hard to model with existing formulations. For example, a point process that would generate independent line segments is very unlikely to produce coherent line networks where segments only join at their endpoints. The key idea of our work is to constrain point processes to only produce well-connected geometric structures.

This chapter focuses on the extraction of 2D structures and leaves the extraction of 3D entities to future work. Our main observation is that any 2D

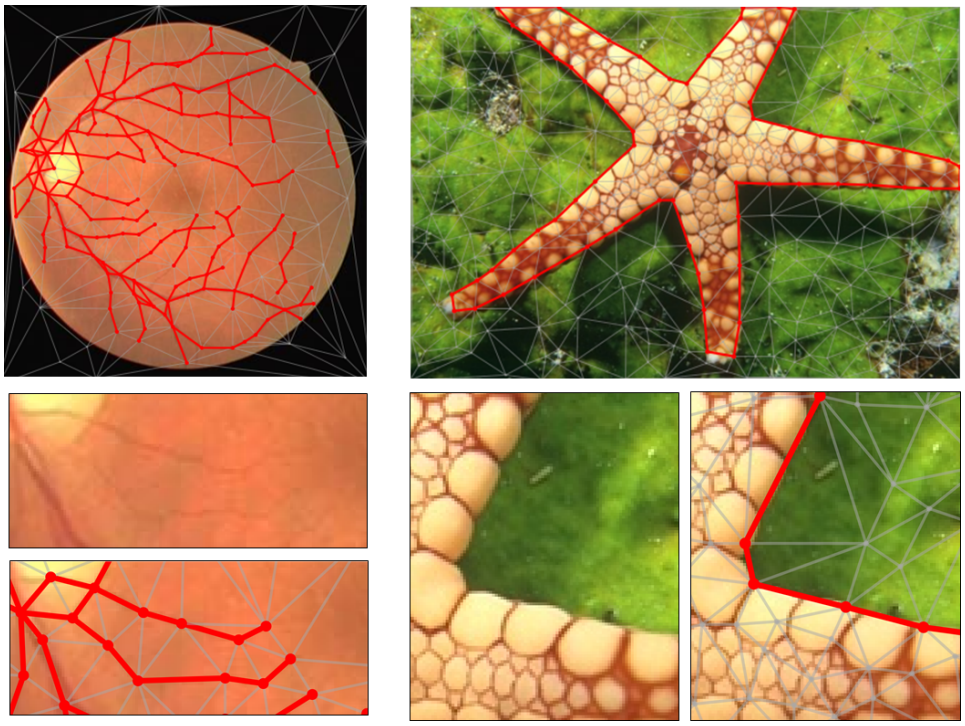


Figure 6.1: Example applications of Delaunay Point Processes to extract planar graphs representing blood vessels in retina images (left), and complex polygons representing object silhouettes (right). The point distribution creates a dynamic Delaunay triangulation while edge and facet labels specify the geometric structure (see red edges on close-ups).

structure composed of non-overlapping lines or polygons can be embedded in a triangulation of the image domain. Given this representation, generating linear or triangular primitives amounts to inserting new vertices in the triangulation, which is a standard operation for existing geometry libraries [The17]. Extracting global structures then amounts to grouping subsets of edges or facets of the triangulation. By construction, our representation offers strong geometric guaranties, such as the fact that line segments and polygons always meet vertex-to-vertex or edge-to-edge. We further build on properties of the Delaunay triangulation to propose an efficient sampler for fast stochastic optimization of the geometric structures we wish to extract. We demonstrate the versatility of this approach to extract 2D structures for a variety of Vision tasks.

In summary, our main contributions are (i) a general framework to extract geometric structures for a variety of Vision problems, (ii) an efficient stochastic optimization to find high-quality structures, and (iii) models for

line network extraction, object contouring, and image compression; demonstrating the potential of our approach on real-world tasks.

6.1 Background on Point Processes

We first provide the necessary background on point processes and their applications before describing our novel approach, which we call *Delaunay Point Processes*. We refer the reader to the book by Descombes [Des11] for a detailed presentation of the theory of point processes for image analysis.

Definition. Point processes are probabilistic models introduced in Vision by Baddeley and Moller to extend traditional Markov Random Fields (MRFs) with an object-based formalism [BL93]. A point process describes random configurations of points in a continuous bounded domain K, where the number of points of a configuration and their positions in the domain are random variables. We denote $\Omega = \bigcup_{n \in \mathbb{N}} \Omega_n$ the configuration space of a point process, where the sub-spaces Ω_n correspond to configurations of exactly n points distributed in K. We denote $\boldsymbol{p} \in \Omega$ a realization of this point process, and $p \in \boldsymbol{p}$ a point of the resulting configuration.

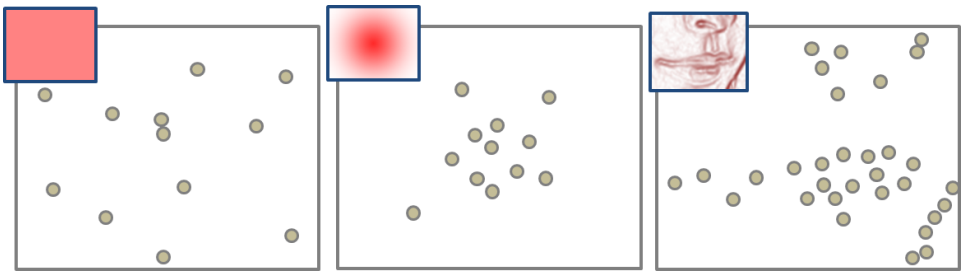


Figure 6.2: Point processes distribute points randomly in a bounded domain. While the left example illustrates a uniform distribution, the middle and right examples show point processes guided by a non-uniform density h (top left insets). In particular, the right example uses the image gradient magnitude as a density to distribute points along image contours.

The simplest point process is the homogeneous Poisson point process, for which the number of points follows a discrete Poisson distribution and the position of the points follows a uniform distribution. As illustrated on Figure 6.2, more complex configurations can be obtained by guiding the point process with a density $h(.)$ defined in Ω. Intuitively, $h(\boldsymbol{p})$ measures the probability of the realization $\boldsymbol{p}$ to occur. By carefully designing the density $h(.)$, practitioners can model processes where the number and position of

the points are consistent with input data and where neighboring points obey specific spatial interactions. In the remaining of this chapter we often express the density as a Gibbs energy $U(.)$ which we seek to minimize

$$h(\boldsymbol{p}) \propto \exp -U(\boldsymbol{p}). \tag{6.1}$$

Markovian property. Similarly to Markov Random Fields, the Markovian property of a point process provides a spatial dependency between neighboring points in a configuration. Formally, a point process of density h is *Markovian under the neighborhood relationship* $\sim$ if and only if $\forall \boldsymbol{p} \in \Omega$ such that $h(\boldsymbol{p}) > 0$, and $\forall q \in K$, $h(\boldsymbol{p} \cup \{q\})/h(\boldsymbol{p})$ only depends on q and its neighbors $\{p \in \boldsymbol{p} \ : \ q \sim p\}$. In other words, when adding a point to a configuration, the resulting variation of density only depends on the new point and its neighbors in the configuration. As discussed next, the Markovian property is essential to many efficient optimization algorithms because it guarantees that the variation of energy induced by a local perturbation of a configuration can be computed using a small number of points around that perturbation.

The symmetric relationship $\sim$ is usually defined via a maximal Euclidean distance ε between two points of K such that

$$p_i \sim p_j = \{(p_i, p_j) \in \boldsymbol{p}^2 : i > j, ||p_i - p_j||_2 < \varepsilon\} \tag{6.2}$$

Figure 6.3-left shows a realization of such a point process for $K \subset \mathbb{R}^2$.

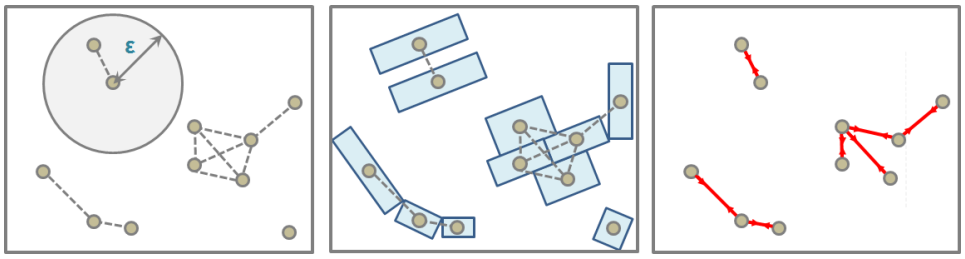


Figure 6.3: Markovian point processes. Traditional point processes exploit the Markovian property to define pairs of interacting points, typically a maximal Euclidean distance ε between two points (left). Such processes are used for detecting objects in images by associating a simple geometric shape, eg a rectangle [ODZ07], to each point (middle), and for extracting line-networks by selecting a subset of pairs of interacting points [CFL13] (right).

Inference. Reversible Jump Markov Chain Monte Carlo (RJMCMC) [Gre95] is a popular family of algorithms to search for configurations that

maximize the density h, or equivalently, that minimize the energy U. A RJMCMC sampler simulates a discrete Markov chain $(X_t)_{t\in\mathbb{N}}$ on the configuration space Ω, converging towards a target density specified by U.

Algorithm 3 provides the pseudo-code of a RJMCMC sampler for point processes. The algorithm starts with a random configuration $\boldsymbol{p}_0$. At each iteration, the current configuration $\boldsymbol{p}$ of the chain is perturbed to a configuration $\boldsymbol{p}'$ according to a proposition density $\mathcal{Q}(\boldsymbol{p} \to .)$, also called a *kernel*. The perturbations are local, which implies that the energy variation between configuration $\boldsymbol{p}$ and $\boldsymbol{p}'$ depends only on a few points thanks to the Markovian property of the point process. The configuration $\boldsymbol{p}'$ is then accepted as the new state of the chain with a probability that depends on the ratio of kernels $\mathcal{Q}(\boldsymbol{p} \to \boldsymbol{p}')$ and $\mathcal{Q}(\boldsymbol{p}' \to \boldsymbol{p})$, the energy variation between $\boldsymbol{p}$ and $\boldsymbol{p}'$, and a relaxation parameter T_t. We next detail the role of kernels and relaxation, followed by a discussion of existing work on object and structure extraction using point processes.

Algorithm 3 RJMCMC sampler for point processes

1- Initialize $X_0 = \boldsymbol{p}_0$ and T_0 at $t = 0$;
2- At iteration t, with $X_t = \boldsymbol{p}$,

- Choose a kernel $\mathcal{Q}_m$ according to probability q_m
- Perturb $\boldsymbol{p}$ to $\boldsymbol{p}'$ according to $Q_m(\boldsymbol{p} \to .)$
- Compute the Green rate

$$R = \frac{\mathcal{Q}_m(\boldsymbol{p}' \to \boldsymbol{p})}{\mathcal{Q}_m(\boldsymbol{p} \to \boldsymbol{p}')} \exp\left(\frac{U(\boldsymbol{p}) - U(\boldsymbol{p}')}{T_t}\right) \tag{6.3}$$

- Choose $X_{t+1} = \boldsymbol{p}'$ with probability $\min(1, R)$, and $X_{t+1} = \boldsymbol{p}$ otherwise

Kernels. For many applications, the kernel $\mathcal{Q}$ is formulated as a mixture of kernels $\mathcal{Q}_m$ associated with probabilities q_m

$$\mathcal{Q}(\boldsymbol{p} \to .) = \sum_m q_m \mathcal{Q}_m(\boldsymbol{p} \to .), \tag{6.4}$$

where each kernel $\mathcal{Q}_m$ is typically dedicated to a specific type of perturbation. The kernel mixture must satisfy two necessary conditions to guarantee the convergence of the Markov chain. First, to make the Markov chain *irreductible*, the kernel mixture must allow any configuration in Ω to be reached from any other configuration with a finite number of perturbations. Second, to make the Markov chain *reversible*, each kernel must be able to propose a

perturbation and its reverse with a non-zero probability. This second condition is necessary to compute the kernel ratio in Equation 6.3, which provides a detailed balance between a perturbation and its reverse [Pes73].

All point processes rely on a *birth and death* kernel that adds or removes a point from $\boldsymbol{p}$ [Des11]. This kernel is parameterized by a birth probability P_b, the death probability P_d being equal to $1 - P_b$. Denoting $|\boldsymbol{p}|$ the number of points in the current configuration, and λ the parameter of the discrete Poisson distribution that governs the number of points in $\boldsymbol{p}$, the kernel ratio for a birth event can be expressed as

$$\frac{\mathcal{Q}_m(\boldsymbol{p}' \to \boldsymbol{p})}{\mathcal{Q}_m(\boldsymbol{p} \to \boldsymbol{p}')} = \frac{P_d}{P_b} \frac{\lambda}{|\boldsymbol{p}| + 1}. \tag{6.5}$$

Similarly, the kernel ratio for a death event is expressed as $\frac{P_b}{P_d} \frac{|\boldsymbol{p}|}{\lambda}$. Intuitively, λ represents the expected number of points in the output configuration. Choosing a birth (respectively a death) when the number of points in the current configuration is higher (resp. lower) than λ will reduce the chance of accepting the proposed perturbation.

Relaxation. The relaxation parameter T_t, also called the *temperature*, controls the acceptation rate of the RJMCMC sampler. A high temperature allows the algorithm to explore very different configurations, including configurations that temporarily increase the energy. In contrast, a low temperature encourages the algorithm to only accept perturbations that decrease the energy. A common practice is to start with a high temperature for initial exploration of the solution space, before decreasing the temperature to converge to a local miminum. Although a logarithmic decrease of T_t is necessary to ensure convergence to the global minimum from any initial configuration, practitioners typically use a faster geometric decrease of the form

$$T_t = T_0.\alpha^t \tag{6.6}$$

where T_0 is the initial temperature and α controls the speed of decrease and is typically set to a value inferior yet close to 1. Such a geometric decrease gives an approximate solution close to the optimum [SSF02].

From points to objects. Many Vision tasks involve the extraction of extended objects rather than infinitesimal points. *Marked point processes* are a family of point processes that tackle such tasks by associating each point with a parametric object. For example buildings can be represented with rectangles [ODZ07], persons with cylinders [Ge09, UB11], or textures with sets of parametric shapes [LGD10]. Figure 6.3-middle illustrates a realization of a marked point process where each point is associated with

a rectangle defined by its orientation, width and length [ODZ07]. The domain of this marked point process is thus $K \times M$ with $K \subset \mathbb{R}^2$ and $M =]-\frac{\pi}{2}, \frac{\pi}{2}] \times [l_{min}, l_{max}] \times [L_{min}, L_{max}]$. Marked point processes are particularly effective for the extraction of groups of objects from images because they can model rich spatial interactions and because they do not require the number of objects to be known a-priori.

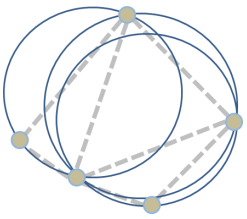
From points to structures. While marked point processes are well adapted to the extraction of groups of disconnected objects, their application to the extraction of connected structures is more challenging because local perturbations may affect the structure globally, breaking the Markovian property necessary for efficient RJMCMC sampling. Prior work attempted to extract geometric structures by designing the point process energy such that it encourages neighboring objects to connect. For example, Lacoste et al. [LDZ05] and Sun et al. [SSZ07] extract line networks by encouraging line segments to form a graph, while Drot et al. [DDLMZ02] segment images by encouraging triangles to form a tessellation. However, this strategy does not scale well because the probability of sampling objects that connect together decreases quickly with the number of objects. To the best of our knowledge, junction-point processes [CFL13] is the only solution designed to extract structures that are well-connected by construction. As illustrated in Figure 6.3-right, junction-point processes exploit the $\sim$neighborhood relationship of Equation 6.2 to define a graph over $\boldsymbol{p}$, where edges link pairs of neighboring points. A subset of edges is then selected to form a planar graph, which represents the output line network. However, a planar graph should not contain crossing edges. Enforcing this topological constraint makes RJMCMC sampling of junction-point processes very slow because a large majority of perturbations yield crossings.

6.2 Delaunay Point Processes

As discussed in the previous section, existing attempts to extract structures with point processes were strongly limited in their ability to enforce the connectivity of the structure elements. We address this challenge by embedding the point process into a Delaunay triangulation from which we extract structures as groups of edges or triangles. Since the triangulation forms a tesselation of the image plane, our structures are well-connected by construction. While a few studies also combined spatial point processes with Delaunay triangulations [BM89, BBD99], they only demonstrated the synthesis of point configurations. In contrast, we augment Delaunay Point Processes with point, edge and facet parameters to extract geometric structures from

images.

6.2.1 Delaunay-Based Neighborhoods



The Delaunay triangulation of a point configuration $\boldsymbol{p}$ subdivides the image domain K into triangles such that no point in $\boldsymbol{p}$ is inside the circumcircle of any triangle. Figure 6.4-left shows the Delaunay triangulation of the same point configuration as in Figure 6.3.

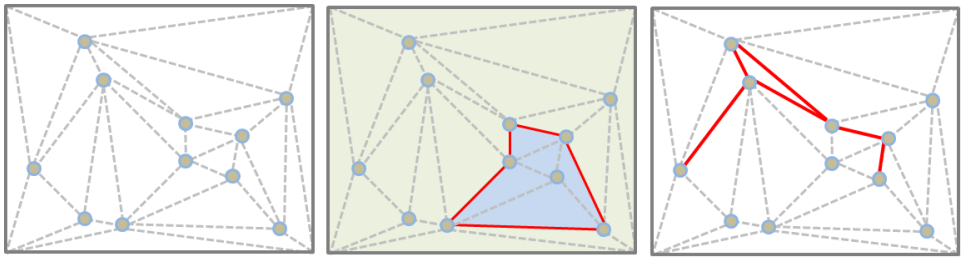


Figure 6.4: Delaunay Point Processes. Contrary to traditional point processes, pairs of interacting points are defined more naturally by a Delaunay triangulation instead of an arbitrary distance parameter ε (left). Exploiting such a geometric meta-structure allows us to partition the image domain into complex polygons by jointly labeling the triangles (middle) or to extract planar graphs by jointly labeling the edges (right). We add the four corner points of Domain K to $\boldsymbol{p}$ for computing the Delaunay triangulation so that K is entirely partition by triangles.

We denote the set of edges and facets of the Delaunay triangulation of $\boldsymbol{p}$ as $C_2(\boldsymbol{p})$ and $C_3(\boldsymbol{p})$ respectively. This triangulation offers a convenient neighborhood relationship $\sim_D$ for point processes: two points are neighbors if they are connected by an edge in the Delaunay triangulation

$$p_i \sim_D p_j = \{(p_i, p_j) \in \boldsymbol{p}^2 : (p_i, p_j) \in C_2(\boldsymbol{p})\}. \tag{6.7}$$

We define Delaunay Point Processes as Markovian point processes supported by the Delaunay neighborhood $\sim_D$. Delaunay Point Processes inherit from several interesting properties of the Delaunay triangulation:

- Parameter-free neighborhood. Traditional point processes require a parameter to specify the area of attraction of the neighborhood relationship. Tuning this parameter is often problem-dependent and strongly impacts result quality. Instead, Delaunay edges connect neighboring points without requiring any parameter.

- Geometric guarantees. Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation, and thus tends to avoid skinny triangles. This property is especially beneficial to reduce imprecision when measuring radiometric quantities over the pixels covered by a triangle, as is the case in our applications to object contouring (Section 6.3.3) and image compression (Section 6.3.4).

- Uniqueness. The Delaunay triangulation of a point set is unique, unless four or more points are inscribed on the same circle, which is very unlikely when the point coordinates are expressed in floating point precision. This property implies that sampling a point set is equivalent to sampling a triangulation.

- Efficient sampling. Perbubations during RJMCMC sampling, *e.g.* removing or adding a point in $\boldsymbol{p}$, only affects the Delaunay triangulation locally. Moreover, such perturbations correspond to the basic operators offered by existing computational geometry libraries to modify Delaunay triangulations [The17].

- Flexibility. The Delaunay triangulation is a flexible geometric representation to address numerous Vision problems. As illustrated on Figure 6.4, we can select edges of the triangulation to represent line-networks, we can group triangles to represent closed contours, or we can assign labels to the triangles to segment the image into parts.

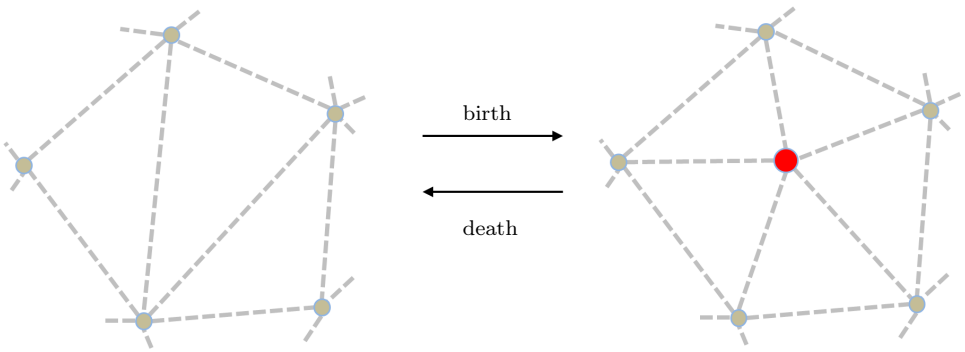


Figure 6.5: Birth and Death kernel. A birth inserts a new vertex in the triangulation and recomputes the edge connectivity around it by applying edge flips recursively until the circumcirle condition is valid everywhere. A death removes a vertex and its adjacent edges and reconnect its adjacent vertices so that the circumcirle condition is valid.

6.2.2 Marks and Energy Formulation

Similarly to marked point processes, we tackle the extraction of extended geometric structures by associating parameters – or marks – to the elements of the Delaunay point process. However, while traditional marked point processes only associate parameters to points, we also associate parameters to the edges and facets formed by the points. This novel feature of Delaunay point processes is key to extract well-connected line-networks and polygons, which are better expressed via edges and triangles than via punctual primitives.

We denote a geometric structure as $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{m})$ where $\boldsymbol{p}$ defines the geometric configuration of the triangulation, while $\boldsymbol{m}$ represents the set of additional parameters on the triangulation elements. For example, $\boldsymbol{m}$ can identify active edges for line-network extraction, or assign different labels to facets for polygonal object segmentation (Figure 6.4). Note that $\boldsymbol{m}$ can also take real values such as colors, as demonstrated in our application to image compression (Section 6.3.4).

Our study of various structure extraction tasks led us to formulate a generic energy U for Delaunay point processes, which we express as the sum of two terms balanced by a parameter $\alpha \in [0, 1]$

$$U(\boldsymbol{x}) = (1 - \alpha)U_{fidelity}(\boldsymbol{x}) + \alpha U_{prior}(\boldsymbol{x}). \tag{6.8}$$

The first term, $U_{fidelity}(\boldsymbol{x})$, measures the agreement of the configuration with image data. For instance, it can measure the alignment of the Delaunay edges with image contours for line network extraction, as detailed in Section 6.3.1. The second term, $U_{prior}(\boldsymbol{x})$, encodes shape priors on the structures we wish to extract. In the example of line network extraction, U_{prior} can penalize acute angles between successive edges to favor smooth polylines. These two terms can be expressed with local energies on points, edges and facets of the Delaunay triangulation. Note that U not only measures the quality of an output structure $\boldsymbol{x}$, but also accounts for the quality of the underlying triangulation $\boldsymbol{p}$.

6.2.3 Sampling procedure

We use the RJMCMC algorithm detailed in Algorithm 3 to search for a good approximation of the optimal configuration. In all our applications, the sampling operates on configurations of geometric structures $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{m})$, which live in a wider space than the point configurations $\boldsymbol{p}$. We now propose three kernels to explore this configuration space: birth or death of a point, relocation of a point, and alteration of a mark. Each of these operators only affect a configuration $\boldsymbol{x}$ locally, which is critical for efficient evaluation of the

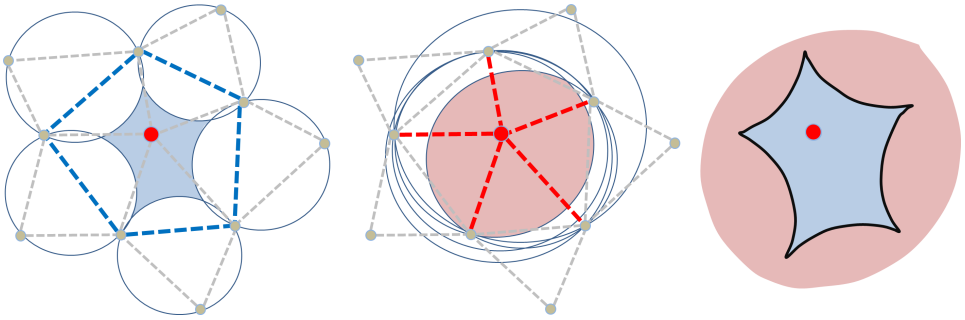


Figure 6.6: Safety domain for point relocation. The blue region (left) corresponds to the region where the red vertex can move without entering the circumcirle of another triangle, *i.e.* without flipping the blue edges. The red region (middle) corresponds to the region where the red vertex can move without leaving the circumcirle of any three successive adjacent vertices, *i.e.* without flipping the red edges. The safety domain, drawn with a black border (right), is the intersection of the blue and red regions. In this example, the blue region lies entirely inside the red region, although this is not true in the general case.

energy variation at each iteration (Equation 6.3).

Birth and death kernel adds or removes a point from $\boldsymbol{p}$, as detailed in Section 6.1. In computational geometry terms, it inserts or removes a vertex from the Delaunay triangulation as illustrated in Figure 6.5. In practice, we give birth and death the same probability ($P_b = P_d = 0.5$). In case of a death, we select one of the points from $\boldsymbol{p}$ randomly. In case of a birth, we create a new point in the image domain K. While we could draw the position of this point from a uniform distribution, this is often inefficient for Vision applications where the structures of interest lie along image contours. We achieve much faster sampling by following a distribution specified by image gradients. Once a vertex is added (respectively removed), we update the marks of its adjacent edges and facets by uniform sampling from the mark domain.

Point relocation modifies the position of a random point in $\boldsymbol{p}$. We make this operator efficient by constraining the point to remain in its *safety domain*, which corresponds to the domain in which a vertex can move without producing edge flips in the Delaunay triangulation (Figure 6.6). We draw the new position of a vertex p from a uniform distribution over the safety domain. As the safety domain for the reverse move is identical, the kernel

ratio in the Green rate (Equation 6.3) is equal to 1. Note that this kernel does not modify the marks of the configuration.

Mark alteration changes the value of a mark in $\boldsymbol{m}$. In practice, we randomly select a point, an edge or a facet of the Delaunay triangulation depending on which type of element the marks are associated with. We then draw a new random value for the mark following a uniform distribution. The kernel ratio is thus equal to 1.

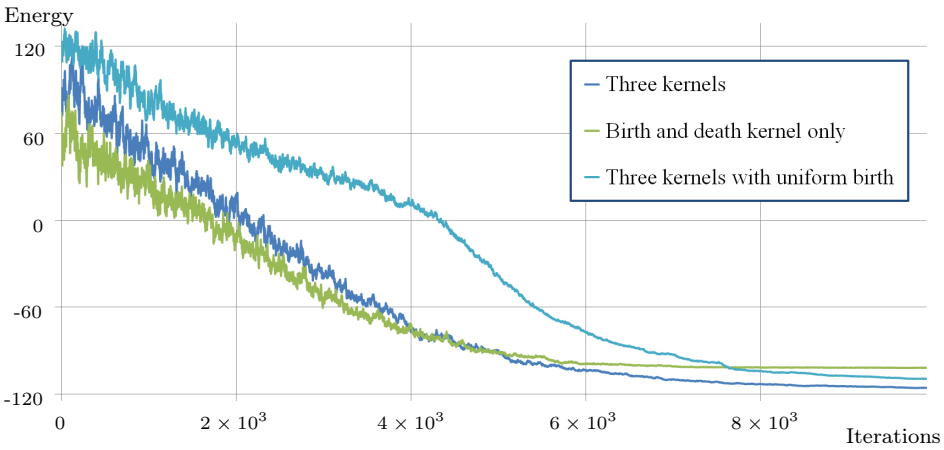


Figure 6.7: Energy decrease with different combinations of kernels. A better energy is reached with our combination of three kernels than with just a birth and death kernel or with the three kernels with uniform birth.

In practice, we give equal probability to the three kernels at each iteration of Algorithm 3, *i.e.* $q_m = \frac{1}{3}$. They play different roles during the sampling procedure. Birth and death is the core operator to simulate Delaunay triangulations with varying complexity, and to access any configuration of the solution space. Point relocation and mark alteration allow local adjustments that would take many iterations to obtain using solely birth and death. As illustrated in Figure 6.7, using only the birth and death kernel gives a fast energy decrease at the beginning of the optimization as the Delaunay vertices quickly align with the main image gradients. However, the energy reaches a high plateau later on, when the low temperature prevents the sampling to propose successive births and deaths that would be necessary to eventually displace a point to another position, or replace a mark by another one. Including the point relocation and mark alteration kernels allows the optimization to decrease the energy further. Figure 6.7 also shows that the optimization reaches high quality configurations faster when we guide

the birth kernel with the image gradients instead of distributing new points uniformly.

6.3 Applications

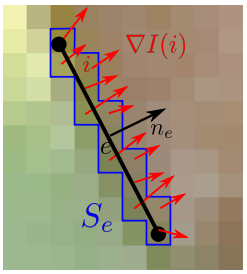
We now demonstrate the versatility of Delaunay point processes on three Vision tasks involving geometric structures: line-network extraction, object contouring, and mesh-based image compression. We provide for each application a brief discussion of related work.

6.3.1 Line-network extraction

Line networks form important structures in many application domains such as medical imaging (vessel networks), remote sensing (road networks), document analysis (line drawings). While many pixel-based algorithms have been proposed to detect such structures [WMZS13], pixel chains often need to be vectorized for further analysis of the resulting planar graph. Several methods rely on a two-step procedure to extract planar graphs by first generating an overcomplete graph that is later simplified using optimization [TBA+13, dGCSAD11, NHS+13, FLB16, MWFU15]. In contrast, our approach samples dynamic planar graphs over the image without resorting to a fixed, overcomplete intermediate representation.

Given the Delaunay triangulation of a point configuration $\boldsymbol{p}$, we model a line-network by associating each edge with a binary activation variable indicating if it belongs to the structure or not. Formally, we define the mark space as $\boldsymbol{m} = (m_e)_{e \in C_2(\boldsymbol{p})}$ with $m_e \in \{0, 1\}$ the activation mark of edge e. We denote $\widetilde{C_2}(\boldsymbol{p})$ the set of active edges in $C_2(\boldsymbol{p})$.

Energy. We design the data fidelity term to encourage active edges to align with strong image gradients. To do so, we define for each active edge $e \in \widetilde{C_2}$ an energy term that measures the strength of the image gradient and its alignment with edge e. Summing over all active edges gives



$$U_{fidelity}(\boldsymbol{x}) = \sum_{e \in \widetilde{C_2}(\boldsymbol{p})} \frac{1}{|S_e|} \sum_{i \in S_e} \exp(-\mu |\nabla I(i) \cdot \boldsymbol{n}_e|) - \gamma \tag{6.9}$$

with S_e the pixels covered by e, $\nabla I(i)$ the image gradient at pixel i, and $\boldsymbol{n}_e$ the unit vector orthogonal to edge e (see inset).The parameter μ controls the sensibility to image noise, and γ is an offset to make the unary term negative

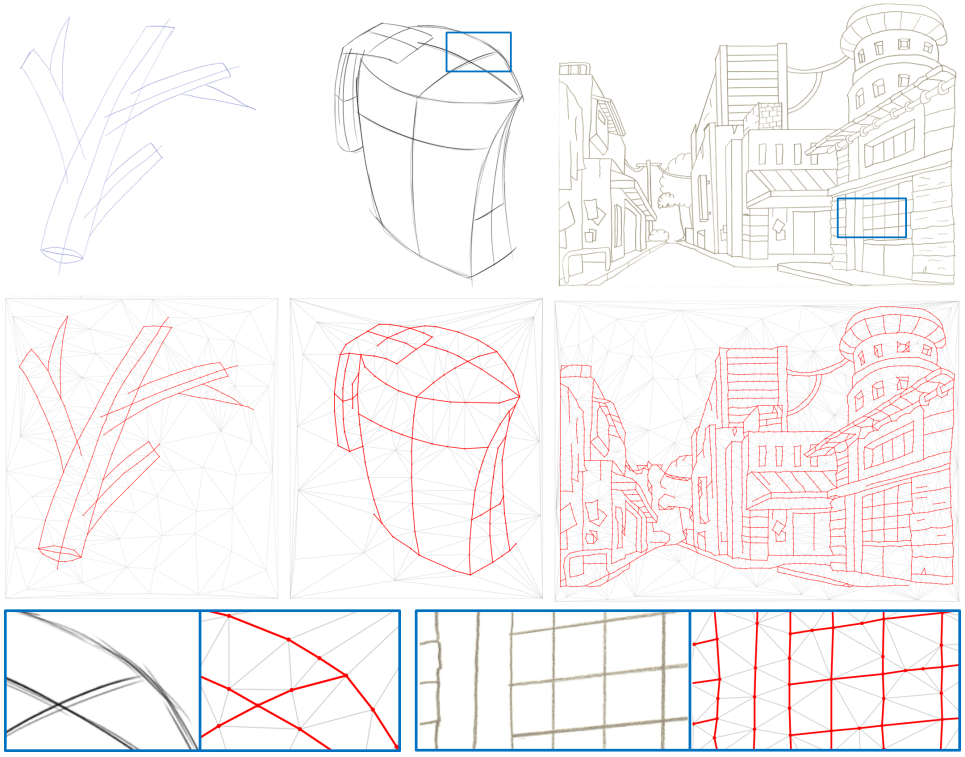


Figure 6.8: Vectorization of line-drawings. Our Delaunay point process recovers the center-line of sketchy pen strokes in bitmap line drawings (active edges shown in red). The insets shows that our model produces well-connected structures even in the presence of multiple overlapping strokes and complex regular patterns.

for strong well-aligned gradients, which encourages their capture by the end structure. We set $\mu = 8$ and $\gamma = 0.5$ in our experiments.

We design the shape prior to penalize isolated edges, short edges, and sharp angles between adjacent edges. The two last criteria prevent the line network to zigzag over image contours. We achieve this behavior with two terms, $w_l(e)$ measuring the length of edges and $w_c(p)$ evaluating the connectivity of active edges at vertices

$$U_{prior}(\boldsymbol{x}) = \sum_{e \in C_2(\boldsymbol{p})} w_l(e) + \beta \sum_{p \in \boldsymbol{p}} w_c(p). \tag{6.10}$$

Parameter β balances these two terms, we fixed it to 1 in our experiments.

We define the edge length penalty $w_l(e)$ to be close to 1 when the edge is shorter than a threshold, and close to 0 otherwise. While we could use a Heaviside function to model this penalty, such as discontinuous energy

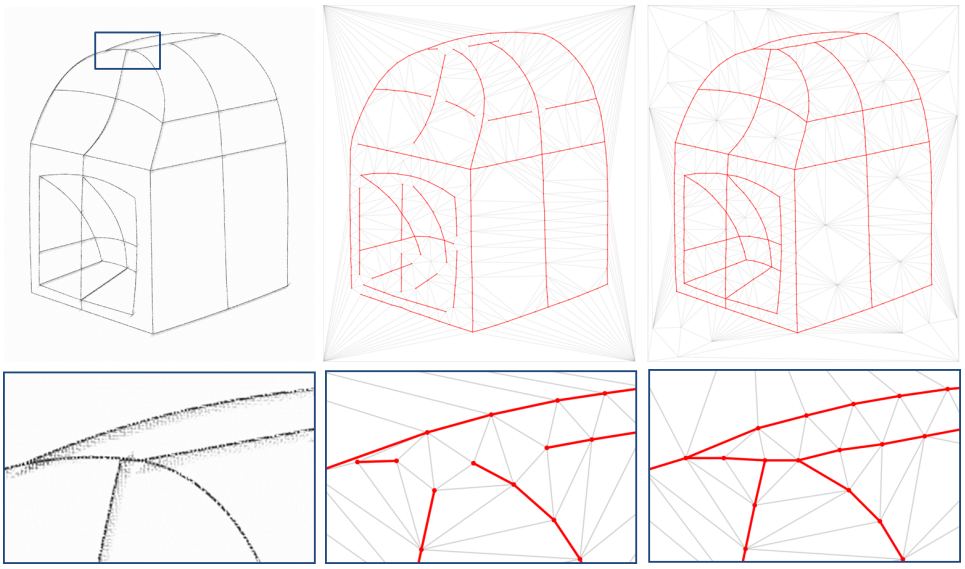


Figure 6.9: Comparison with a two-steps optimization for line drawing vectorization. When sampling points independently of the marks (middle), edges of the Delaunay triangulation often miss important line junctions, which cannot be recovered by a subsequent marking step. Our Delaunay point process samples points and marks jointly, which favors the emergence of a well-connected line network (right).

would hinder the stochastic optimization. We use instead a smooth sigmoid function of the form $\mathcal{S}_{a,b}(|e|) = (1 + \exp(a(\frac{2|e|}{b} - 1)))^{-1}$ with $|e|$ the edge length and a, b two real values. In our context, b corresponds to the desired minimal length of edges, which we typically fix to 5% of the image diagonal in our experiments. We set the positive constant a to 5 to shape the sigmoid like a smoothed Heaviside function.

The connectivity penalty $w_c(p)$ should penalize both isolated active edges and pairs of active edges forming sharp angles. We achieve this behavior by setting $w_c(p) = 0$ if vertex p has no adjacent active edge, $w_c(p) = 1$ if vertex p has one (*i.e.* isolated) adjacent active edge, and $w_c(p) = \mathcal{S}_{a,b}(\delta(p))$ if vertex p has more than one adjacent active edge, where δ_p is the maximal value of dot products between any pair of active edges around p. We set $a = 5$ and $b = cos(\frac{\pi}{12})$ in our experiments.

Experiments. We applied our model to extract line networks from different application domains, such as vessels in organic images (retina in Figure 6.1, leaf in Figure 6.10), pen strokes in line drawings (Figure 6.8), regular edge patterns in man-made textures (tiles in Figure 6.10). Our model

performs well on this diverse set of images thanks to its generic fidelity term, which only depends on image gradients (Equation 6.9). The insets in Figure 6.8 show that our model extracts clean line intersections from rough drawings. Recovering such topological information is a necessary step for many line drawing vectorization algorithms [NHS+13, FLB16].

A major strength of our approach over existing two-steps strategies is its ability to jointly recover the geometric configuration of the Delaunay triangulation and identify its active edges. Figure 6.9 illustrates the benefits of this joint procedure compared to a two-steps method that first estimates the position of the vertices, and then estimates the activation of the edges. We implemented the first step by sampling points according to an energy that encourages them to be on strong image gradients and to form long edges, *i.e.* $U_{fidelity}(\boldsymbol{x}) = \sum_{p \in \boldsymbol{p}}(1 - |\nabla I(p)| - \gamma)$ and $U_{prior}(\boldsymbol{x}) = \sum_{e \in C_2(\boldsymbol{p})} w_l(e)$. Note that we cannot encourage alignment of the edges to image gradients at this point, since we don't know yet which edges will form the end structure. The second step identifies these active edges by minimizing the complete energy (Equation 6.9 and 6.10) using only the mark alteration kernel of RJMCMC to keep the triangulation fixed. The comparison shows that the two-steps approach often misses line junctions in the final network because such junctions are not captured by the triangulation during the first step.

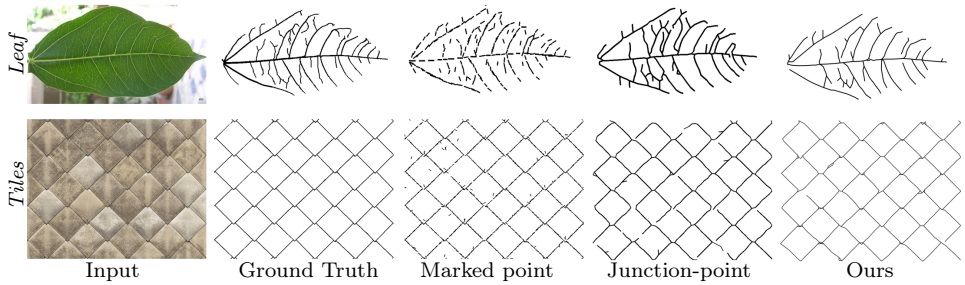


Figure 6.10: Visual comparisons with existing point processes. Marked point process [VL14] produces configurations of mostly disconnected line-segments. Junction-point process [CFL13] better preserves the connectivity of edges but recover badly complex junctions of at least four branches (see junctions in *tiles*). Our Delaunay point process exhibits better connectivity and accuracy for both cyclic (bottom) and acyclic (top) line-networks.

Figure 6.10 and Table 6.1 provide qualitative and quantitative comparisons of our model to existing line-network extraction methods based on point processes. Using a marked point process with line segments [VL14] results in many isolated segments as the algorithm struggles to enforce connectivity. Junction-point processes [CFL13] better model connectivity, but have diffi-

Table 6.1: Quantitative comparisons with existing point processes. Our Delaunay point process outperforms marked point process [VL14] and junction-point process [CFL13] in terms of precision and F-measure while being faster.

		Precision	F-measure	Time
Leaf	Junction-point process [CFL13]	0.59	0.64	73s
	Marked point process [VL14]	0.76	0.70	33s
	ours	**0.79**	**0.73**	20s
Tiles	Junction-point process [CFL13]	0.46	0.54	227s
	Marked point process [VL14]	0.67	0.72	103s
	ours	**0.70**	**0.74**	70s

culties extracting high degree junctions such as the crossings of the tiles. Our method extracts well-connected networks and gives higher precision scores.

6.3.2 Line drawing vectorization

In this section, we extend our model of line network extraction to extract curve networks. In chapter 4 we used a hypergraph to represent the curve network. Each hyperedge was associated to a curve and to a set of edges of a topological graph. We now describe how to implement a similar representation with Delaunay Point Process and we show on figure 6.11 a curve network and its associated labelled Delaunay triangulation.

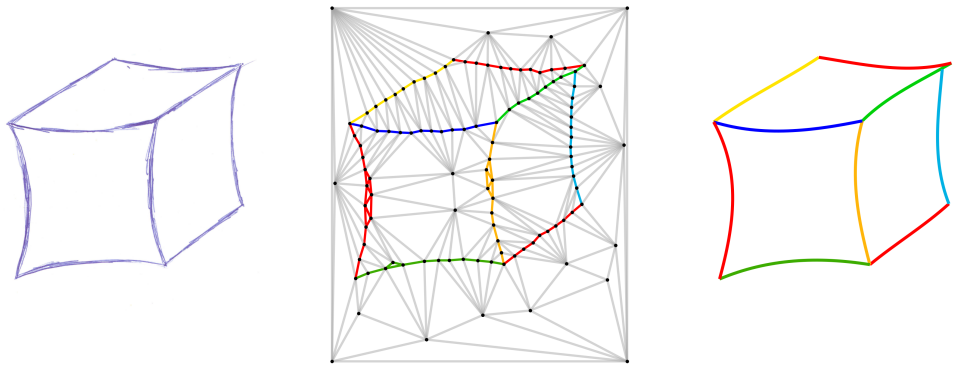


Figure 6.11: Curve network extraction by Delaunay point process. Each edge is associated to a set of curve indices. Then the curve network is fitted minimizing the distance between each curves and its associated edges.

Marks definition. Marks are used to regroup Delaunay edges to form an hypergraph representing the curve network. Formally we define the mark

space as $\boldsymbol{m} = (m_e)_{e \in C_2(\boldsymbol{p})}$ with $m_e \subset \mathbb{N}$ a set of curve indices. Each edge can be associated to several curves and a curve can be associated to several edges. Parameters of curves are computed by minimizing the L_2 distance between curves and their associated edges. Such marks require custom kernels.

Kernel. While we keep the birth and death and the point relocation kernels unchanged, we adapt the mark alteration kernel to support hyperedge labelling. To alter a mark of a given edge, a curve index can be added or removed from its set of curve. The index of an added curve can be one of the indices of the set of curves of the adjacent edges or an index of a new curve. We define a new kernel called *curve merge and split*. The curve merge operator fuses two curves. In practice, for each edge, if its set of curves contains the index of the second curve, we replace it by the index of the first curve. The curve split kernel is the reverse operator.

When a curve index is added or removed from the set of curves of an edge, or when a point of an edge associated to a curve is modified, the curve is refitted by minimizing the distance between the curve and its associated edges. The fitted curves are cubic Bézier curves.

Energy We use the energy of the model presented in section 6.3.1 and we add a simplified version of the energy of chapter 4 weighted by a parameter which controls the impact on the energy of the curve network.

$$U(\boldsymbol{x}) = U_{line\_network}(\boldsymbol{x}) + \varepsilon U_{curve\_network}(\boldsymbol{x}) \tag{6.11}$$

with $\varepsilon << 1$ and

$$U_{curve\_network}(\boldsymbol{x}) = (1-\lambda)U_{fidelity}(\boldsymbol{x}) + \lambda U_{simplicity}(\boldsymbol{x}) \tag{6.12}$$

$U_{fidelity}$ is the distance between curves and their associated edges and $U_{simplicity}$ is the number of curves. On figure 6.13 we show some results and we compare with the algorithm introduced in chapter 4 on figure 6.12.

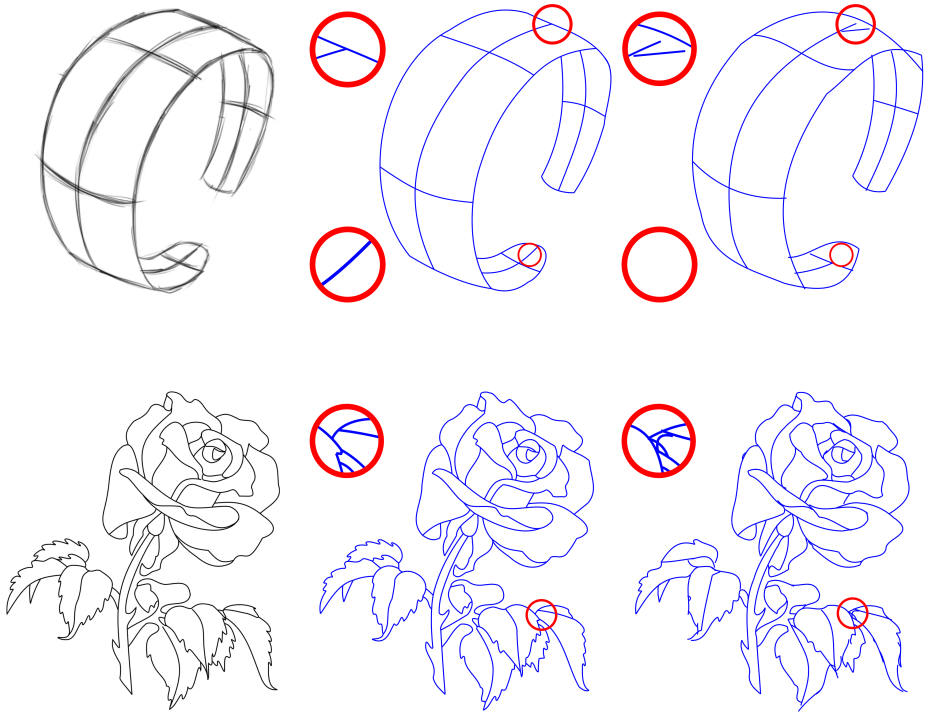


Figure 6.12: We compare our vectorization method by Delaunay Point process with our vectorization method by hypergraph optimization of chapter 4 on a rough sketch with few details and a clean sketch with fine details. Overall, the results are visually similar. However, Delaunay Point Process sometimes misses small curves or produces spurious curves on sketchy strokes. The region based segmentation used in chapter 4 is more robust in such configuration. Note that we did not implement snapping for Delaunay Point Process.

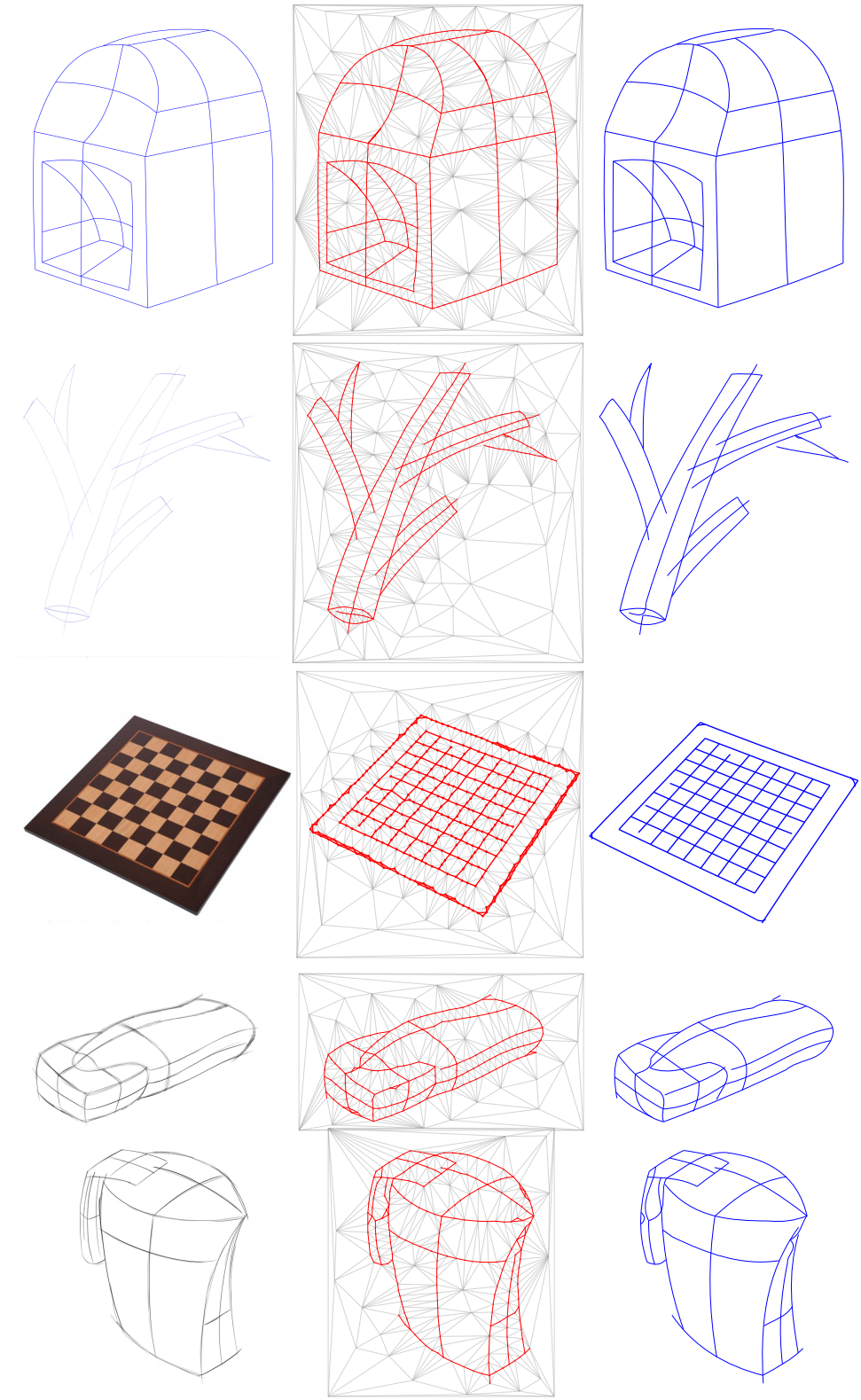


Figure 6.13: Curve network extraction by Delaunay point process. We run our algorithm on a variety of sketches and on a photograph.

6.3.3 Object contouring

Object contouring by polygonal shapes provides a compact and structure-aware representation of the object silhouette. Polygonal contours are particularly well suited to represent man-made objects like buildings, cars or furniture that are dominated by straight segments. Existing object polygonalization methods typically start by detecting line-segments, which are then assembled into polygons. This second step can be done by searching for cycles in a graph of line-segments [SCF14], or by connecting line-segments using gap filling [ZFW+12]. Another strategy for object contouring consists in over-segmenting the image before extracting objects as groups of superpixels [LSD10]. However, obtaining polygonal objects with this strategy either requires a preprocessing step to convert superpixels into small polygons [DL15] [AS17], or a post-processing step to vectorize chains of pixels into polygons [WM03], which often introduces inaccuracy. Recent work considered the use of recurrent neural networks to sequentially predict the vertices of a polygonal object contour [CKUF17], but such a black-box algorithm offers little control on the complexity of the outcome. Closer to our work are Polygonal Markov Fields [KvLS07], which are stochastic models designed to sample polygons in images. Based on local operators that add or remove vertices to a polygon, these models struggles to explore topological variations and remain very slow to converge on natural images. Our model also shares ideas with the work of Ren et al. [RFM05] who builds on a constrained Delaunay triangulation to fill gaps in object contours. However, they formulate contour completion as an edge labeling task on a fixed triangulation, while our method lets the triangulation evolve dynamically to best capture the object contour.

To achieve polygonal object contouring within our framework, we associate each facet of the Delaunay triangulation with a binary activation variable indicating if it belongs to the object or not. The output polygonal contours correspond to the set of edges separating active polygons from inactive ones, which ensures that the contours are closed by construction. Formally, we define the mark space as $\boldsymbol{m} = (m_f)_{f \in C_3(\boldsymbol{p})}$ with $m_f = \{0, 1\}$ the activation mark of triangle f. We guide the object segmentation with a pixelwise probability map H. The computation of this probability map depends on the application scenario, as detailed in our experiments.

Figure 6.14: Object contouring on a few example images. Our Delaunay point process samples polygons that capture the silhouettes of foreground objects. The user first draws a few scribbles that roughly characterize the objects of interest (blue lines) and the image background (red lines). Output polygons preserve details, such as the flower petals and the vase handles, while having low complexity.

Energy. We express our data fidelity energy as the sum of two terms, one measuring the agreement between the binary mark of each facet and the underlying probability map, the other one encouraging homogeneous colors within each facet to preserve image contours

$$U_{fidelity}(\boldsymbol{x}) = \frac{1}{|I|} \sum_{f \in C_3(\boldsymbol{p})} \sum_{i \in f} (1 - H(i|m_f)) + \beta_1 \, |I_{|f}| \sigma^2_{I_{|f}} \tag{6.13}$$

where $|I|$ is the number of pixels of image I, $|I_{|f}|$ is the number of pixels inside facet f, $\sigma^2_{I_{|f}} \in [0,1]$ is the normalized variance of pixel colors inside facet f, and $H(i|m_f)$ is the probability of assigning mark m_f to pixel i. The parameter β_1 balances the two terms, we fixed it to 1 in our experiments.

Our shape prior for object contouring uses the same term as for line network extraction to penalize short edges. In addition, we define a smoothness term based on Potts model to favor compact polygons. Summing the two terms gives

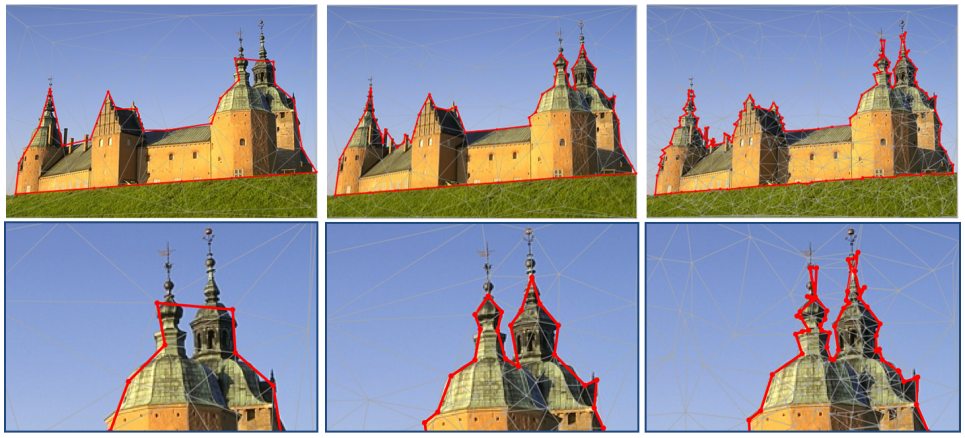
$$U_{prior}(\boldsymbol{x}) = \sum_{e \in C_2(\boldsymbol{p})} w_l(e) + \beta_2 \, w_s(e) \tag{6.14}$$

where the edge length penalty $w_l(e)$ is defined as in equation 6.10, and $w_s(e) = |e|$ if the two facets adjacent to edge e have different marks, and $w_s(e) = 0$ otherwise. The parameter β_2 balances the two terms, we fixed it to $\frac{1}{|I|}$ in our experiments.

Experiments. We tested our contouring model on the Berkeley segmentation dataset [MFTM01] as well as on images with regular man-made structures, such as facades and urban aerial photographs. For each input image, we compute the probability map H from a few user-provided scribbles, which roughly characterize the radiometric distribution of the foreground objects of interest and the image background. We express the probability $H(i|m_f)$ of a pixel i to belong to class m_f as its normalized RGB distance to the closest color in the set of scribbled pixels belonging to that class

$$H(i|m_f) = \frac{\min\limits_{j \in S_{m_f}} \|I(i) - \widehat{I}(j)\|_2^2}{\min\limits_{j \in S_0} \|I(i) - \widehat{I}(j)\|_2^2 + \min\limits_{j \in S_1} \|I(i) - \widehat{I}(j)\|_2^2} \tag{6.15}$$

where S_0 (respectively S_1) is the set of pixels scribbled as foreground (resp. background), and $\widehat{I}$ is the input image convolved by a 11×11 mean filter to remove noise. Note that more advanced methods could be used to predict foreground and background pixels, but this is beyond the scope of this chapter.



$\alpha = 0.5,\ |\boldsymbol{p}| = 51$ $\alpha = 0.005,\ |\boldsymbol{p}| = 221$ $\alpha = 0.0005,\ |\boldsymbol{p}| = 771$

Figure 6.15: Trade-off between fidelity and simplicity. Parameter α offers control over the complexity of the output polygon. A low α value gives more weight to the fidelity term of the energy, resulting in more complex polygons that tightly fit to the object silhouettes.

Figure 6.14 shows the results of this model on a variety of images with organic and man-made shapes. Our method extracts low-complexity polygons that accurately capture the object silhouettes, despite the simplicity of our color model H. Our method performs best on man-made objects composed of piecewise-linear contours, such as the facade elements in the street-level picture and the roofs in the aerial picture. Figure 6.15 illustrates the trade-off between fidelity and simplicity for different values of parameter α, keeping the Poisson parameter fixed. Although we cannot control the exact number of edges in the output polygons, tuning α has a direct impact on polygon complexity.

Our Delaunay point process is competitive with existing object contouring techniques. Figure 6.16 provides a comparison to several two-steps strategies, for an increasing number of user scribbles. We first compare to a pixel-based segmentation algorithm (GrabCut [RKB04]), which requires many scribbles to capture fine details accurately. In contrast, our approach achieves better segmentations with fewer scribbles by working at the scale of Delaunay triangles. Converting the GrabCut pixel segmentation to a polygon as a post-process [WM03] reduces accuracy further. We also compare to running GraphCut segmentation [BK04] on a graph of polygonal superpixels [DL15], using $H(i|m_f)$ for the unary term and a Potts model for the pairwise term. However, we only obtained satisfactory results when using small superpixels, which result in very complex output polygons. In contrast, our

Figure 6.16: Comparisons with two-steps object contouring methods given different sets of input scribbles. The GrabCut pixel-based segmentation [RKB04] requires many input scribbles to correctly capture horse silhouettes (see the bottom parts of the legs). Converting the output pixel chains to polygons using Douglas-Peucker algorithm [WM03] accentuates their defects, whereas pre-segmenting the input image into polygonal superpixels [DL15] only give satisfactory results when small superpixels are used, *i.e.* for high output complexity. In contrast, our Delaunay point process produces low complexity polygons that accurately capture the horses, even when only two scribbles are provided.

method achieves both high accuracy and low polygon complexity, even when very few scribbles are provided.

6.3.4 Image compression

Our third application consists in representing an image as a colored triangular mesh, as illustrated in Figure 6.17. While this geometric representation is not as flexible as wavelet-based compression schemes [Mal08], we show that it achieves competitive compression rates on images dominated by smooth color variations (Figure 6.19). Our approach is inspired by prior work on Delaunay-based image compression [DI06, DDI06, BPC09], image vectorization [LHFY12], and image stylization [GW16]. However, existing methods employ heuristic or greedy strategies to define the location of the Delaunay vertices. In contrast, Delaunay point processes allow us to jointly optimize the position and color of the vertices to best balance image reproduction with image compression.

We represent a color image as a Delaunay triangulation, where each vertex is associated to a color and each triangle interpolates the colors of its vertices bilinearly. Formally, we define the mark space as $\boldsymbol{m} = (c_p)_{p \in \boldsymbol{p}}$ where c_p is a RGB color. Since we assume that the Delaunay triangulation is uniquely defined by its vertices, we can store the image compactly as a list of colored points.

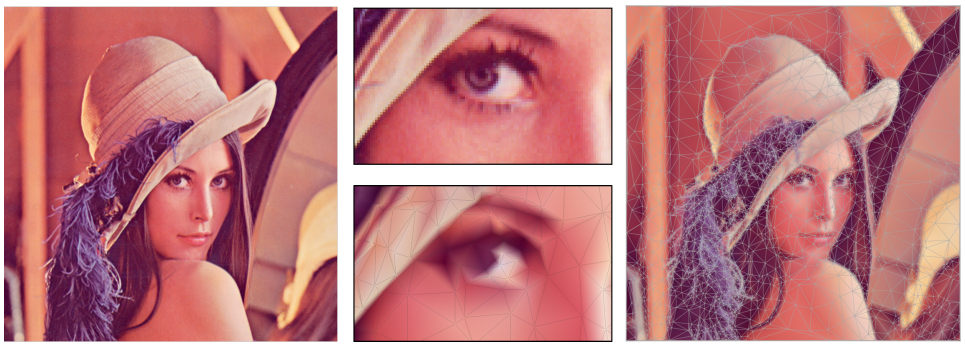


Figure 6.17: Image compression by Delaunay point process. A Delaunay triangulation with only 2.8K colored points (right) is sufficient to approximate a 262Kpixels image (left) with a structural similarity (SSIM) greater than 0.97. Each triangle is colored by bilinear interpolation of its three vertices. Here we display the Delaunay edges in grey for visualization.

Energy. We design the point process energy to offer a trade-off between fidelity to the input image and simplicity of the output mesh. The fidelity

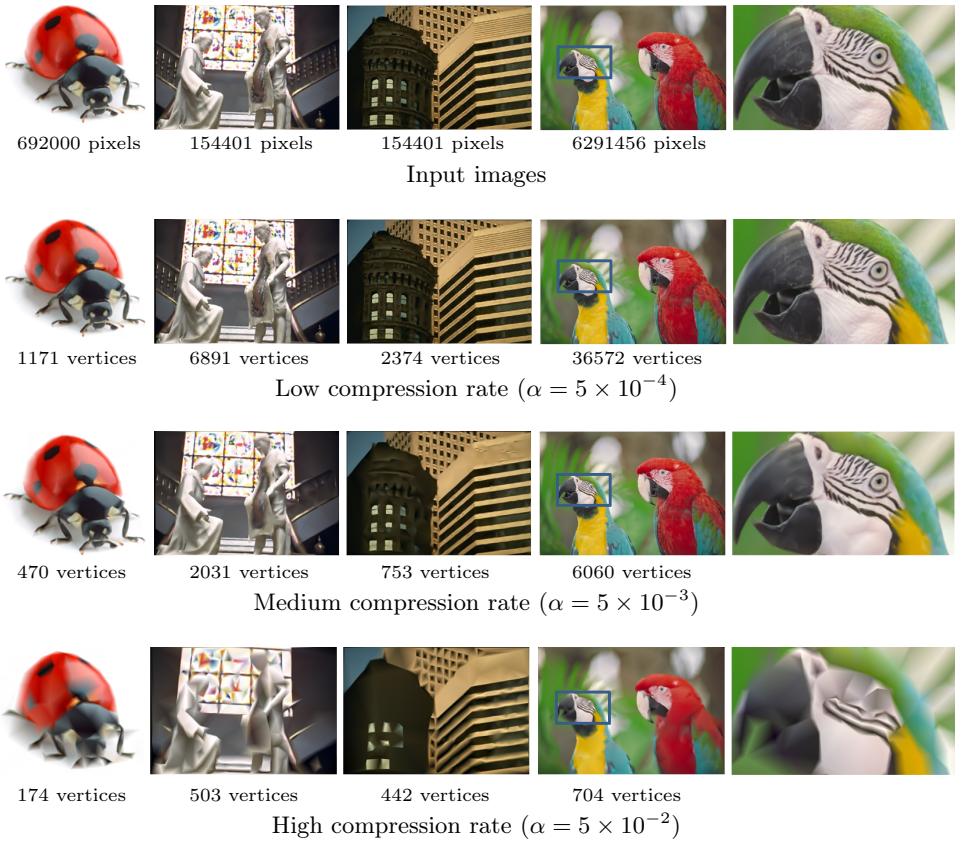


Figure 6.18: Parameter α offers a trade-off between visual quality and compression rate. A high α value preserves high frequency details and sharp discontinuities, but gives a low compression rate. Highly compressed results (bottom) expose the underlying triangulation, especially on fine details and regular patterns such as the building facades.

term measures the per-pixel error between input and output,

$$U_{fidelity}(\boldsymbol{x}) = \frac{1}{|S|} \sum_{i \in S} ||I(i) - I_{\boldsymbol{x}}(i)||_2^2, \tag{6.16}$$

where S is the set of pixels of input image I and $I_{\boldsymbol{x}}$ is the image reconstructed from configuration $\boldsymbol{x}$ using bilinear color interpolation over each triangle. $U_{fidelity}$ can be seen as a sum of unary data terms on each facet. The shape prior penalizes the number of points in the configuration,

$$U_{prior}(\boldsymbol{x}) = \frac{|\boldsymbol{p}|}{\lambda}, \tag{6.17}$$

where $|\boldsymbol{p}|$ is the number of points in configuration $\boldsymbol{x}$ and λ is the Poisson

parameter of the point process.

Experiments. We implemented the algorithm with the *birth and death* and *point relocation* kernels described in Section 6.2.3. However, we found that we can avoid using the *mark alteration* kernel by computing the optimal color of a mark every time the corresponding point is created or relocated. We compute this color by minimizing Equation 6.16 over the pixel domain covered by the facets adjacent to the point.

Figure 6.18 illustrates the effect of parameter α, which weights $U_{fidelity}$ and U_{prior} according to Equation 6.8. A low α preserves well the input image but generates complex configurations, with typically more points than the Poisson parameter. Increasing α yields simpler configurations where fine details are removed.

We compare the performance of our model to state-of-the-art image compression algorithms in Figure 6.19. We perform this comparison on images with varying levels of realism and noise (a clipart, a studio photograph, and a real-world photograph). Since our model represents an image as a piecewise-linear function, it performs best on cliparts that are often composed of linear color gradients. Our approach is also competitive on studio photographs that contain large, uniform highlights and soft shadows with little image noise. However, our approach tends to smooth-out the high-frequency grain of real-world photographs, achieving a low SSIM score on such images. Note also that, similarly to other vector graphics representations, our colored meshes can be rasterized at any resolution.

6.4 Discussion

We have introduced Delaunay Point Processes for the extraction of 2D geometric structures composed of line segments or polygons. By building on point processes, our approach simultaneously detects geometric primitives and group them into structures, which is more robust than performing these two tasks in sequence. By building on the Delaunay triangulation, our approach produces well-connected structures and allows more efficient stochastic optimization than point processes based on an Euclidean distance neighborhood. Our three applications demonstrate the flexibility of this framework. We now detail the performance of our method and give some design guidelines for practitioners who would like to apply it to other structure extraction tasks.

Performance. Because the RJMCMC sampler is memoryless, Delaunay point processes are very memory efficient with a constant memory allo-

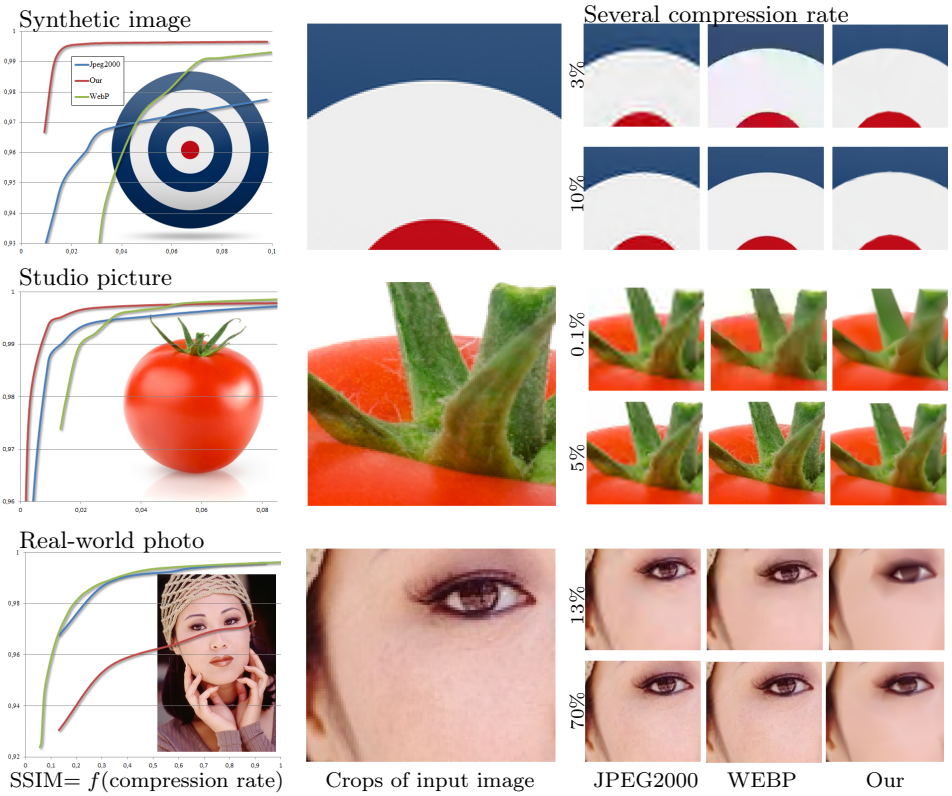


Figure 6.19: Comparisons with state-of-the-art image compression algorithms. Our method competes well with image compression standards on synthetic images (top) and, to a lesser extent, on studio photographs (middle). Our use of bilinear color interpolation inside triangles is well suited to the smooth color variations of synthetic images. However, this interpolation tends to remove the fine grain of real world photographs (bottom). As a result, our approach achieves low SSIM scores on the portrait, even though its tendency to smooth out noise may be appreciated in some applications.

cation during sampling. Running times range from a few seconds, *e.g.* for the extraction of horse silhouettes on Figure 6.16, to a few minutes, *e.g.* for the compression of the parrots on Figure 6.18. Timings depend on the input image size and, to a lesser extent, on the complexity of the energy formulation. In particular, the line-network extraction model requires more iterations to converge than our two other models. The design of kernels tailor-made for manipulating Delaunay triangulations allows us to reach attractive timings compared to traditional point processes, especially with the use of an efficient computational geometry library [The17]. Note that recent work proposed faster optimization strategies for point processes, such as par-

allel Monte Carlo samplers [VL14] and binary labeling of object proposals [PRRC16], although these approaches make restrictive assumptions on the energy. Sampling Delaunay point processes in parallel would require an efficient GPU implementation of Delaunay data-structures, which is currently not available in standard geometry libraries.

Design guidelines. We have identified several guidelines to follow to develop efficient models for Delaunay Point Processes.

- *Small mark space.* The mark space should only contain a few discrete values to allow efficient exploration by random mark alterations. While our model for image compression relied on the much larger space of 24bits colors, we resorted to a closed-form optimization to estimate the mark value instead of random sampling.

- *Simple energy.* Simple energy formulations improve convergence stability. In particular, we recommend using (i) no more than three energy terms to avoid unstable parameter tuning, and (ii) as-continuous-as-possible energy functions so that the Monte Carlo sampler better guides the current configuration into interesting energy valleys.

- *Application-specific kernels.* Although the birth and death kernel is theoretically sufficient to explore the entire configuration space, application-specific kernels often greatly speed-up the optimization. In particular, data-driven kernels can concentrate perturbations in the interesting areas of the input data, *e.g.* on the high image gradients when extracting line-networks.

- *Reasonably-sized images.* Because the Monte Carlo sampler operates sequentially, timings are strongly impacted by the input image resolution. We obtained competitive performances by running Delaunay point processes on images with a few million pixels. The parallelization of Delaunay point processes constitutes an important research challenge to scale to high-resolution images.

Perspectives. Besides the optimization challenges, an interesting direction for future research would be to develop efficient strategies for marking Delaunay point processes with parametric functions. This would allow the extraction of more complex geometric structures, such as non-linear color gradients for image compression. We also would like to investigate the extension of Delaunay point processes to 3D, opening the door to many vision problems that involve the extraction of surfaces and volumes.

Chapter 7

Conclusion and perspectives

7.1 Conclusion

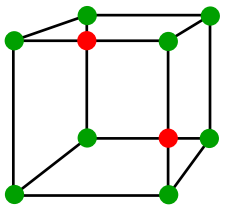
In this thesis, we studied the problem of 2D vectorization of clipart and line drawings. We focus on producing vector graphics compact that are easy to edit representing the input images in a clean and abstract way.
We give a particular attention to junctions which are the most ambiguous points in images and also the ones which contains the most relevant informations.
Our main contribution was to express the vectorization problem as the optimization of an energy which explicitly minimize the complexity of the reconstruction to favor the simplest interpretation. We design specific operators to be able to explore the configuration space using Monte Carlo algorithms. We also propose a generic stochastic algorithm to deal with the problem of shapes extraction and show results on a variety of vision problems like line network extraction, image contouring, image vectorization as a mean of compression and sketch vectorization.

7.2 Perspectives

We can extend our methodology in the following directions.

3D modelling from single rough sketches One direction is to create a full pipeline for 3D modelling from a single rough sketch. Such a pipeline combine several key ingredients:

1. extract a clean skeleton from the rough sketches using method based on deep learning [SSISI16]
2. vectorize the clean skeleton to obtain a vectorial line network using our contribution on line drawing vectorization (chapter 4)
3. apply our algorithm [FLB15] which reconstructs in 3D a 2D vectorial line drawing



The main challenge is to find the 3D position of endpoints of 2D curves. In [FLB15], we assume that the topology **and** the connectivity of the input vectorial line drawing is correct, so that each 2D point corresponds to a single 3D point. For the inset cube, the input has to be composed of the 12 lines having the green node as endpoints. The red nodes correspond to occlusions that should be ignored. A major challenge is to automatically deals with such occlusions. A possible solution, in the spirit of our treatment of X-junction for sketches, is to associate each node with several 3D positions. In other words, a junction between four curves has to be interpretable either as four lines meeting at a point or one line occluding another one. These additional degrees of freedom greatly expand the solution space.

In [FLB15], we limit the reconstruction to lines. Our approach has to be extended to be able to manage quadratic and cubic Bézier curves as input and Bézier surfaces as output. While other methods support Bézier curves [XCS+14], they assume that occlusions are labelled by the user.

Parametric surfaces from point cloud Another direction is to extend our work of 2D vectorization of line drawings to 3D to extract parametric surfaces from point clouds. Standard methods like [HDD+94] decompose this problem in three main steps: the first step is to extract a dense mesh from a point cloud, the second step is to simplify the mesh and the last step is to recover a piecewise smooth surface from the mesh. Following our approach on line drawing vectorization composed by two steps (extracting first a set of Bézier curves and then grouping them together to generate a clean and compact curve network), we can extract from a point cloud Bézier surfaces and then group them together in bigger surfaces to simplify the surface network (ie the 3D parametric model). The main challenge is about junctions. In 3D, there are two categories of junctions: curves which are junctions between two surfaces and points which are junctions between three or more surfaces. But intersections of noisy surfaces may not coincide on one vertex. So we may have to add a term to penalize each junction.

Delaunay point process with curved edges The main limitation of Delaunay point process is that we only extract piecewise linear geometric shapes. In section 6.3.2 of chapter 6, we propose a way to sample directly a curve network. Each edge was associated to several curves and the curves were fitted to minimize their distance to their associated edges. But we can go further by replacing edges of the Delaunay triangulation by Bézier curves

and triangles by Bézier surfaces. In such a case, we would have to not only sample the vertices of the triangulation, but to also sample the nodes of the control points of the curves. This algorithm could then be used for extracting gradient mesh and diffusion curves. For diffusion curves, curved edges would be labelled as diffusion curve or not. Vertices which are not connected to any labelled curve would be used to support the finite element method [BBG12] to compute the diffusion.

3D reconstruction using Delaunay point process For Delaunay point process, we only showed application on 2D shape extraction problems. However our framework can work in any dimension. A challenging application in higher dimensions is the 3D reconstruction problem. State of the art methods start by extracting SIFT points on images and then use a Delaunay triangulation to constrain the reconstructed 3D mesh to fill holes on area with no SIFT points. The main advantages of Delaunay point process for this problem is that there is no need to extract SIFT point in preprocess and then build all the algorithm on these SIFT points which are often erroneous on low contrast areas. The energy could use photo-consistency as a fidelity term and minimize the number of vertices for simplicity. Such algorithm could also be used for remeshing.

Finally, instead of sampling a triangulation, we could also directly sample a parametric surface on the scene. Care must be taken to define a model with few, stable energy terms. To go further, we could label parametric surfaces as objects belonging to classes, where two objects are in the same class if their are invariant by rigid transformation. Such a formulation would allow the automatic extraction of repetitive elements and their exploitation to consolidate the reconstruction. Our simplicity term could offer a control on the number of classes, and as such on the amount of repetition in the scene.

List of publications:

Line drawing interpretation in a multi-view context[FLB15]
Jean-Dominique Favreau, Florent Lafarge, Adrien Bousseau
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015

Fidelity vs. Simplicity: a Global Approach to Line Drawing Vectorization.[FLB16]
Jean-Dominique Favreau, Florent Lafarge, Adrien Bousseau
ACM Transaction on Graphics (Proc. of SIGGRAPH), 2016

Photo2ClipArt: Image Abstraction and vectorization Using Layered Linear Gradients.[FLB17]
Jean-Dominique Favreau, Florent Lafarge, Adrien Bousseau
ACM Transaction on Graphics (Proc. of SIGGRAPH Asia), 2017

Extracting Geometric Structures in Images with Delaunay Point Processes.
Jean-Dominique Favreau, Florent Lafarge, Adrien Bousseau
Submitted to IEEE Transaction on Pattern Analysis and Machine Intelligence

Bibliography

[AASP17] Yağiz Aksoy, Tunç Ozan Aydin, Aljoša Smolić, and Marc Pollefeys. Unmixing-based soft color segmentation for image manipulation. *ACM Transactions on Graphics (TOG)*, 36(2):19, 2017. (Cited on pages 20 and 21.)

[Ado13] Adobe. Adobe illustrator image trace. *http://blogs.adobe.com/adobeillustrator/2013/07/image-trace-in-illustrator-a-tutorial-and-guide.html*, 2013. (Cited on page 15.)

[AS17] R. Achanta and S. Susstrunk. Superpixels and polygons using simple non-iterative clustering. In *CVPR*, 2017. (Cited on page 91.)

[BBD99] E. Bertin, J.-M. Billiot, and R. Drouilhet. Spatial delaunay gibbs point processes. *Communications in Statistics. Stochastic Models*, 15(2), 1999. (Cited on page 77.)

[BBG12] Simon Boyé, Pascal Barla, and Gaël Guennebaud. A vectorial solver for free-form vector gradients. *ACM Transactions on Graphics (TOG)*, 31(6):173, 2012. (Cited on page 103.)

[BBS14] Sean Bell, Kavita Bala, and Noah Snavely. Intrinsic images in the wild. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 33(4), 2014. (Cited on pages 19 and 20.)

[BCF+07] Alexandra Bartolo, Kenneth P Camilleri, Simon G Fabri, Jonathan C Borg, and Philip J Farrugia. Scribbles to vectors: preparation of scribble drawings for cad interpretation. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, 2007. (Cited on pages 11, 14, 42, 47 and 48.)

[BF12] Bin Bao and Hongbo Fu. Vectorizing line drawings with near-constant line width. In *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pages 805–808. IEEE, 2012. (Cited on page 12.)

[BFL06] Yuri Boykov and Gareth Funka-Lea. Graph cuts and efficient nd image segmentation. *International journal of computer vision*, 70(2):109–131, 2006. (Cited on page 29.)

[BK04] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Analysis Machine Intelligence*, 26(9):1124–1137, September 2004. (Cited on pages 94 and 95.)

[BL93] A. J. Baddeley and M. Van Lieshout. Stochastic geometry models in high-level vision. *Journal of Applied Statistics*, 20(5-6), 1993. (Cited on page 73.)

[BLW15] Pengbo Bo, Gongning Luo, and Kuanquan Wang. A graph-based method for fitting planar b-spline curves with intersections. *Journal of Computational Design and Engineering*, 2015. (Cited on pages 12 and 13.)

[BM89] A. Baddeley and J. Moller. Nearest-neighbour markov point processes and random sets. *International Statistical Review*, 57(2), 1989. (Cited on page 77.)

[BPC09] S. Bougleux, G. Peyre, and L. Cohen. Image compression with anisotropic geodesic triangulations. In *ICCV*, 2009. (Cited on page 96.)

[BPW+12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012. (Cited on pages 50, 54 and 55.)

[Bre13] A. Bretto. *Hypergraph Theory: an introduction.* Springer, 2013. (Cited on page 31.)

[BTS05] Pascal Barla, Joëlle Thollot, and François Sillion. Geometric clustering for line drawing simplification. In *Proceedings of the Eurographics Symposium on Rendering*, 2005. (Cited on page 14.)

[CFL13] Dengfeng Chai, Wolfgang Forstner, and Florent Lafarge. Recovering line-networks in images by junction-point processes. In *Proc. of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2013. (Cited on pages 14, 74, 77, 86 and 87.)

[CKUF17] L. Castrejon, K. Kundu, R. Urtasun, and S. Fidler. Annotating object instances with a polygon-rnn. In *CVPR*, 2017. (Cited on page 91.)

[CM02] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, May 2002. (Cited on page 64.)

[DC63] Paul De Casteljau. Courbes et surfaces à pôles. *André Citroën, Automobiles SA, Paris*, 1963. (Cited on page 3.)

[DCKL97] Michael D'Zmura, Philippe Colantoni, Kenneth Knoblauch, and Bernard Laget. Color transparency. *Perception*, 26(4):471–492, 1997. (Cited on page 25.)

[DDI06] Laurent Demaret, Nira Dyn, and Armin Iske. Image compression by linear splines over adaptive triangulations. *Signal Process.*, 86(7), July 2006. (Cited on page 96.)

[DDLMZ02] S. Drot, X. Descombes, H. Le Men, and J. Zerubia. Object point processes for image segmentation. In *ICPR*, 2002. (Cited on page 77.)

[Des11] X. Descombes. *Stochastic geometry for image analysis*. Wiley-ISTE, 2011. (Cited on pages 73 and 76.)

[dGCSAD11] Fernando de Goes, David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. An optimal transport approach to robust reconstruction and simplification of 2d shapes. *Computer Graphics Forum*, 30(5), 2011. (Cited on page 83.)

[DI06] L. Demaret and A. Iske. Adaptive image approximation by linear splines over locally optimal delaunay triangulations. *Signal Processing Letters*, 13(5), 2006. (Cited on page 96.)

[DL15] L. Duan and F. Lafarge. Image partitioning into convex polygons. In *CVPR*, 2015. (Cited on pages 91, 94 and 95.)

[DOIB12] Andrew Delong, Anton Osokin, Hossam N. Isack, and Yuri Boykov. Fast approximate energy minimization with label costs. *International Journal Computer Vision (IJCV)*, 96(1), 2012. (Cited on page 23.)

[DS02] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 21(3):769–776, July 2002. (Cited on page 21.)

[EPD09] Elmar Eisemann, Sylvain Paris, and Fredo Durand. A visibility algorithm for converting 3d meshes into editable 2d vector graphics. *ACM Trans. Graph. (Proc. of SIGGRAPH)*, 28:83:1–83:8, July 2009. (Cited on page 19.)

[EWHS08] Elmar Eisemann, Holger Winnemoeller, John C. Hart, and David Salesin. Stylized vector art from 3d models with region support. *Computer Graphics Forum (Proc. of EGSR)*, 27(4), June 2008. (Cited on page 19.)

[FLB15] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. Line drawing interpretation in a multi-view context. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. (Cited on pages 23, 101, 102 and 104.)

[FLB16] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. Fidelity vs. Simplicity: a Global Approach to Line Drawing Vectorization. *ACM Trans. on Graphics (Proc. of SIGGRAPH)*, 35(4), 2016. (Cited on pages 83, 86 and 104.)

[FLB17] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. Photo2clipart: Image abstraction and vectorization using layered linear gradients. *ACM Transactions on Graphics (SIGGRAPH Asia Conference Proceedings)*, 36(6), November 2017. (Cited on page 104.)

[GDA+12] Timothy Gerstner, Doug DeCarlo, Marc Alexa, Adam Finkelstein, Yotam Gingold, and Andrew Nealen. Pixelated image abstraction. In *Proc. International Symposium on Non-photorealistic Animation and Rendering (NPAR)*, June 2012. (Cited on page 22.)

[Ge09] R. Ge, W. andCollins. Marked point processes for crowd counting. In *CVPR*, 2009. (Cited on page 76.)

[Gre95] Peter J. Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4), 1995. (Cited on page 74.)

[GW16] Meng Gai and Guoping Wang. Artistic low poly rendering for images. *The Visual Computer*, 32(4):491–500, Apr 2016. (Cited on page 96.)

[Has70] W.K. Hastings. Monte Carlo sampling using Markov chains and their applications. *Biometrika*, 57(1), 1970. (Cited on page 35.)

[HDD+94] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 295–302. ACM, 1994. (Cited on page 102.)

[HPFPL09] Jean-Emmanuel Haugeard, Sylvie Philipp-Foliguet, Frédéric Precioso, and Justine Lebrun. Extraction of windows in facade using kernel on graph of contours. In *Image Analysis*, pages 646–656. Springer, 2009. (Cited on page 29.)

[HT06] X. Hilaire and K. Tombre. Robust and accurate vectorization of line drawings. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 28(6), 2006. (Cited on pages 12 and 32.)

[Jes16] Stefan Jeschke. Generalized diffusion curves: An improved vector representation for smooth-shaded images. In *Computer Graphics Forum*, volume 35.2, pages 71–79. Wiley Online Library, 2016. (Cited on page 18.)

[JGCC12] Zhaoyin Jia, A. Gallagher, Yao-Jen Chang, and Tsuhan Chen. A learning-based framework for depth ordering. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, June 2012. (Cited on page 25.)

[JHWS17] Xie Jun, Winnemöller Holger, Li Wilmot, and Schiller Stephen. Interactive vectorization. In *Proceedings of SIGCHI 2017*. ACM, 2017. (Cited on page 16.)

[KvLS07] R. Kluszczyński, M. N. M. van Lieshout, and T. Schreiber. Image segmentation by polygonal markov fields. *Annals of the Institute of Statistical Mathematics*, 59(3), 2007. (Cited on page 91.)

[LDZ05] C. Lacoste, X. Descombes, and J. Zerubia. Point processes for unsupervised line network extraction in remote sensing. *PAMI*, 27(10), 2005. (Cited on page 77.)

[Lev44] Kenneth Levenberg. A method for the solution of certain problems in least squares. *Quarterly of applied mathematics*, 2:164–168, 1944. (Cited on page 17.)

[LGD10] F. Lafarge, G. Gimel'farb, and X. Descombes. Geometric feature extraction by a multi-marked point process. *PAMI*, 32(9), 2010. (Cited on page 76.)

[LHFY12] Zicheng Liao, Hugues Hoppe, David Forsyth, and Yizhou Yu. A subdivision-based representation for vector image editing. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 18(11), 2012. (Cited on pages 17 and 96.)

[LHM09] Yu-Kun Lai, Shi-Min Hu, and Ralph R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Trans. Graph.*, 28(3):85:1–85:8, July 2009. (Cited on pages 16 and 17.)

[Lie94] M. N. M. Van Lieshout. Stochastic annealing for nearest-neighbour point processes with application to object recognition. *Advances in Applied Probability*, 26(2), 1994. (Cited on page 71.)

[LL06] Gregory Lecot and Bruno Lévy. Ardeco: automatic region detection and conversion. In *Proceedings of the Eurographics Symposium on Rendering Techniques*, 2006. (Cited on page 15.)

[LMY+13] Xueting Liu, Xiangyu Mao, Xuan Yang, Linling Zhang, and Tien-Tsin Wong. Stereoscopizing cel animations. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 32(6):223:1–223:10, November 2013. (Cited on page 25.)

[LPK07] P. Labatut, J.-P. Pons, and R. Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cut. In *ICCV*, 2007. (Cited on page 71.)

[LSD10] A. Levinshtein, C. Sminchisescu, and S. Dickinson. Optimal contour closure by superpixel grouping. In *ECCV*, 2010. (Cited on page 91.)

[LWH15] Xueting Liu, Tien-Tsin Wong, and Pheng-Ann Heng. Closure-aware sketch simplification. *ACM Transactions on Graphics (TOG)*, 34(6), 2015. (Cited on pages 14, 32, 42 and 48.)

[LZW04] Anat Levin, Assaf Zomet, and Yair Weiss. Separating reflections from a single image using local features. In *Proc. of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2004. (Cited on pages 20 and 23.)

[Mal08] Stphane Mallat. *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, 3rd edition, 2008. (Cited on page 96.)

[Met74] Fabio Metelli. The perception of transparency. *Scientific American*, 1974. (Cited on page 25.)

[MFTM01] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, 2001. (Cited on page 93.)

[MPFB17] C. Mostegel, R. Prettenthaler, F. Fraundorfer, and H. Bischof. Scalable surface reconstruction from point clouds with extreme scale and density diversity. In *CVPR*, 2017. (Cited on page 71.)

[MWFU15] G. Mattyus, S. Wang, S. Fidler, and R. Urtasun. Enhancing road maps by parsing aerial images around the world. In *ICCV*, 2015. (Cited on page 83.)

[NHS+13] Gioacchino Noris, Alexander Hornung, Robert W Sumner, Maryann Simmons, and Markus Gross. Topology-driven vectorization of clean line drawings. *ACM Transactions on Graphics (TOG)*, 32(1):4, 2013. (Cited on pages 11, 12, 14, 32, 33, 42, 47, 48, 83 and 86.)

[OBW+08] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: a vector representation for smooth-shaded images. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 27(3), 2008. (Cited on pages 3, 17, 18 and 51.)

[ODZ07] M. Ortner, X. Descombes, and J. Zerubia. Building outline extraction from digital elevation models using marked point processes. *International Journal of Computer Vision*, 72(2), 2007. (Cited on pages 74, 76 and 77.)

[OK11] Günay Orbay and Levent Burak Kara. Beautification of design sketches using trainable stroke clustering and curve fitting. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 17(5), 2011. (Cited on pages 14 and 41.)

[Pes73] P. H. Peskun. Optimum monte carlo sampling using markov chains. *Biometrika*, 60, 1973. (Cited on page 76.)

[PJPZ10] T. Peng, I. Jermyn, V. Prinet, and J. Zerubia. Extended phase field higher-order active contour models for networks - its application to road network extraction from VHR satellite images. *International Journal Computer Vision (IJCV)*, 88(1), 2010. (Cited on pages 14 and 29.)

[PRRC16] Trung Pham, Seyed Rezatofighi, Ian Reid, and Tat-Jun Chin. Efficient point process inference for large-scale object detection. In *CVPR*, 2016. (Cited on page 100.)

[RFM05] Xiaofeng Ren, Charless Fowlkes, and Jitendra Malik. Scale-invariant contour completion using conditional random fields. In *ICCV*, 2005. (Cited on page 91.)

[RKB04] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut -interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (Proc. of Siggraph)*, 23(3), 2004. (Cited on pages 94 and 95.)

[RLMB+14] Christian Richardt, Jorge Lopez-Moreno, Adrien Bousseau, Maneesh Agrawala, and George Drettakis. Vectorising bitmaps into semi-transparent gradient layers. *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering)*, 33(4), 2014. (Cited on pages 18, 19, 46, 58, 62 and 63.)

[RP07] Elisa Ricci and Renzo Perfetti. Retinal blood vessel segmentation using line operators and support vector classification. *Medical Imaging, IEEE Transactions on*, 26(10):1357–1365, 2007. (Cited on page 29.)

[SB96] Alvy Ray Smith and James F. Blinn. Blue screen matting. *SIGGRAPH*, pages 259–268, 1996. (Cited on pages 20 and 58.)

[SC11] Bilge Sayim and Patrick Cavanagh. The art of transparency. *i-Perception*, 2, 2011. (Cited on pages 25 and 56.)

[SCD+06] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Computer vision and pattern recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 519–528. IEEE, 2006. (Cited on page 29.)

[SCF14] X. Sun, M. Christoudias, and P. Fua. Free-shape polygonal object localization. In *ECCV*, 2014. (Cited on pages 71 and 91.)

[Sel17] Peter Selinger. Potrace. *http://potrace.sourceforge.net*, 2017. (Cited on page 60.)

[SH03] Manish Singh and Xiaolei Huang. Computing layered surface representations: an algorithm for detecting and separating transparent overlays. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2003. (Cited on page 25.)

[SLWS07] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 26(3), 2007. (Cited on pages 3, 16 and 51.)

[SSF02] P. Salamon, P. Sibani, and R. Frost. *Facts, Conjectures, and Improvements for Simulated Annealing*. SIAM Monographs on Mathematical Modeling and Computation, Philadelphia, United States, 2002. (Cited on pages 37 and 76.)

[SSISI16] Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. Learning to simplify: fully convolutional networks for rough sketch cleanup. *ACM Trans. Graph.*, 35:121:1–121:11, 2016. (Cited on pages 14 and 101.)

[SSZ07] Kaiqiong Sun, Nong Sang, and Tianxu Zhang. *Marked Point Process for Vascular Tree Extraction on Angiogram*, pages 467–478. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. (Cited on page 77.)

[TBA+13] Engin Turetken, Fethallah Benmansour, Bjorn Andres, Hanspeter Pfister, and Pascal Fua. Reconstructing Loopy Curvilinear Structures Using Integer Programming. In *Proc. of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2013. (Cited on pages 14 and 83.)

[The17] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.11 edition, 2017. (Cited on pages 72, 79 and 99.)

[TLG16] Jianchao Tan, Jyh-Ming Lien, and Yotam Gingold. Decomposing images into layers via RGB-space geometry. *ACM Transactions on Graphics*, 36(1):7:1–7:14, November 2016. (Cited on pages 21, 62 and 63.)

[UB11] A. Utasi and C. Benedek. A 3-D marked point process model for multi-view people detection. In *CVPR*, 2011. (Cited on page 76.)

[VL14] Yannick Verdie and Florent Lafarge. Detecting parametric objects in large scenes by Monte Carlo sampling. *IJCV*, 106(1), 2014. (Cited on pages 86, 87 and 100.)

[Wer23] Max Wertheimer. A brief introduction to gestalt, identifying key theories and principles. *Psychol Forsch*, 4:301–350, 1923. (Cited on page 23.)

[WM03] S. T. Wu and M. R. G. Marquez. A non-self-intersection Douglas-Peucker algorithm. In *IEEE Symposium on Computer Graphics and Image Processing*, 2003. (Cited on pages 91, 94 and 95.)

[WMZS13] J. Wegner, J. Montoya-Zegarra, and K. Schindler. A higher-order crf model for road network extraction. In *CVPR*, 2013. (Cited on page 83.)

[WOG06] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-time video abstraction. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 25(3):1221–1226, July 2006. (Cited on page 21.)

[WZS04] Yanfeng Wei, Zhongming Zhao, and Jianghong Song. Urban building extraction from high-resolution satellite panchromatic image using clustering and edge detection. In *Geoscience and Remote Sensing Symposium, 2004. IGARSS'04. Proceedings. 2004 IEEE International*, volume 3, pages 2008–2010. IEEE, 2004. (Cited on page 29.)

[XCS+14] Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 33(4), 2014. (Cited on page 102.)

[XLY09] Tian Xia, Binbin Liao, and Yizhou Yu. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Trans. Graph.*, 28(5):115:1–115:10, December 2009. (Cited on page 16.)

[XSTN14] Guofu Xie, Xin Sun, Xin Tong, and Derek Nowrouzezahrai. Hierarchical diffusion curves for accurate automatic image vectorization. *ACM Trans. Graph.*, 33(6):230:1–230:11, November 2014. (Cited on page 18.)

[YCZ+16] Ming Yang, Hongyang Chao, Chi Zhang, Jun Guo, Lu Yuan, and Jian Sun. Effective clipart image vectorization through direct optimization of bezigons. *IEEE transactions on visualization and computer graphics*, 22(2):1063–1075, 2016. (Cited on page 15.)

[ZCZ+09] Song-Hai Zhang, Tao Chen, Yi-Fei Zhang, Shi-Min Hu, and Ralph R. Martin. Vectorizing cartoon animations. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 15(4), 2009. (Cited on pages 19, 33 and 44.)

[ZDZ17] Shuang Zhao, Fredo Durand, and Changxi Zheng. Inverse diffusion curves using shape optimization. *IEEE Transactions on Visualization and Computer Graphics*, 2017. (Cited on page 18.)

[ZFW+12] Z. Zhang, S. Fidler, J. Waggoner, Y. Cao, S. Dickinson, J. Siskind, and S. Wang. Superedge grouping for object localization by combining appearance and shape informations. In *CVPR*, 2012. (Cited on pages 71 and 91.)

[ZYH+15] Yang Zhou, Kangxue Yin, Hui Huang, Hao Zhang, Minglun Gong, and Daniel Cohen-Or. Generalized cylinder decomposition. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 34(6), 2015. (Cited on page 23.)