IMAGE FLOWS AND ONE-LINER
GRAPHICAL IMAGE
REPRESENTATIONS

VADIM MAKHERVAKS
IMAGE FLOWS AND ONE-LINER

GRAPHICAL IMAGE REPRESENTATIONS

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS

FOR THE DEGREE OF MASTER OF SCIENCE

IN APPLIED MATHEMATICS

VADIM MAKHERVAKS

SUBMITTED TO THE SENATE OF THE TECHNION — ISRAEL INSTITUTE OF TECHNOLOGY

TEBETH, 5763

HAIFA

DECEMBER, 2002
ACKNOWLEDGMENT

I wish to express gratitude to my supervisors Prof. Alfred Bruckstein and Dr. Gill Barequet for their guidance, brilliant ideas and assistance. Without their help I could not have achieved these results.

Special thanks are to Vitaly Surazhsky for long and fruitful discussions. His assistance and help with setting up the development environment and providing various useful software libraries were invaluable.

Special thanks are due to the management team of IBM’s Haifa Research Lab, and in particular to Raanan Gewirtzman and Igor Chirashnya, for their support, understanding and encouragement of my M.Sc. studies.

THE GENEROUS FINANCIAL HELP OF TECHNION IS GRATEFULLY ACKNOWLEDGED
Contents

Abstract .................................................. 1

1 Introduction ........................................... 3
   1.1 Purpose of Work .................................. 3
   1.2 Edge Detection and Linking ....................... 6
       1.2.1 Previous Work ............................. 7
       1.2.2 Our Approach ............................. 9
   1.3 Edge Prioritarization Scheme ..................... 12
   1.4 Connecting Selected Edges into a One-Liner .... 14

2 Estimation of Derivatives ............................. 15
   2.1 Estimation of First Derivatives .................. 19
   2.2 Estimation of Second Derivatives ................. 21

3 Edge Exploration ......................................... 23
   3.1 Image Flows ...................................... 24
   3.2 The Algorithm .................................... 26
       3.2.1 Finding candidate edges .................. 28
       3.2.2 Stabilizing candidate edges............... 30
## CONTENTS

3.2.3 Subdividing candidate edges ........................................... 31
3.3 The Edge-Exploration Operator ............................................. 32
3.4 A Heuristic Operator ....................................................... 36
3.5 Edge Prioritization .......................................................... 37
3.6 Derivative Interpolation ..................................................... 40

4 Linking the Edges ............................................................ 42
4.1 Ordering the Edges ........................................................... 44
4.1.1 The one-liner problem is NP-Complete ............................... 46
4.1.2 Naive heuristic approaches .......................................... 48
4.1.3 A TSP-based approach ............................................... 50
4.1.4 A TSP solver .......................................................... 54
4.2 Determining the Connection Curves ..................................... 54

5 Experimental Results ....................................................... 57
5.1 The Development Environment .......................................... 57
5.2 Analysis of Experimental Results ...................................... 58
5.2.1 Overall execution time ............................................... 67
5.2.2 Improving the sorting algorithm ................................... 68

6 Conclusions ................................................................. 70
6.1 Results of the Thesis ...................................................... 70
6.2 Improvements and Future Enhancements ............................... 71

References ................................................................. 73

Hebrew Abstract ............................................................ i
List of Figures

1.1 Picasso’s one-liners ...................................................... 4
1.2 Single edge one-liners .................................................... 4
1.3 One-liner examples ....................................................... 5
1.4 A hill-climbing example .................................................. 10
1.5 Two components of the exploration operator ......................... 10
1.6 A bullfight image ......................................................... 12
1.7 Major edges in the bullfight image .................................... 13

2.1 A few neighborhood examples .......................................... 16
2.2 A few weighted neighborhoods ......................................... 16
2.3 Edge margins ............................................................ 19

3.1 Intensity levels in images ................................................ 25
3.2 Exploring one edge ....................................................... 27
3.3 Edge intersection and merging ........................................... 29
3.4 Edge stabilization ....................................................... 30
3.5 Candidate edge subdivision ............................................. 32
3.6 The components of the edge-exploration operator .................. 33
3.7 Selecting the next pixel .................................................. 37
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>Selected image edges</td>
<td>38</td>
</tr>
<tr>
<td>3.9</td>
<td>Varying ranking parameters</td>
<td>39</td>
</tr>
<tr>
<td>3.10</td>
<td>Varying selected edges</td>
<td>40</td>
</tr>
<tr>
<td>3.11</td>
<td>Subpixel location</td>
<td>40</td>
</tr>
<tr>
<td>3.12</td>
<td>Derivative interpolation</td>
<td>41</td>
</tr>
<tr>
<td>4.1</td>
<td>The main edges in various images</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>One-liner candidates</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>The one-liner problem vs. the segments-connection problem</td>
<td>45</td>
</tr>
<tr>
<td>4.4</td>
<td>The reduction</td>
<td>47</td>
</tr>
<tr>
<td>4.5</td>
<td>The one-liner problem vs. TSP</td>
<td>51</td>
</tr>
<tr>
<td>4.6</td>
<td>Reducing one-liner to TSP</td>
<td>52</td>
</tr>
<tr>
<td>4.7</td>
<td>Distances between one-liner vertices</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>Lamp</td>
<td>61</td>
</tr>
<tr>
<td>5.2</td>
<td>Paolina</td>
<td>62</td>
</tr>
<tr>
<td>5.3</td>
<td>Lena</td>
<td>63</td>
</tr>
<tr>
<td>5.4</td>
<td>Felix</td>
<td>64</td>
</tr>
<tr>
<td>5.5</td>
<td>Canova’s Images</td>
<td>65</td>
</tr>
<tr>
<td>5.6</td>
<td>Bullfights</td>
<td>66</td>
</tr>
</tbody>
</table>
Abstract

This work introduces a novel graphical image representation comprising a single curve—the one-liner. Building one-liner representation consists of several steps. The first step involves the detection and linking of the image edges. We use a new technique to simultaneously perform both tasks. We refer to this technique as edge exploration. This process is based on the concept of “image flows.” It uses the image-gradient vector field and a new edge-exploration operator to find and rank image edges. Estimating the derivatives of the image is performed by using local Taylor expansions in conjunction with a weighted least-squares estimation method. The edge-exploration process finds all the possible image edges without any pruning. During this process we also collect information that allows to prioritize the found detected edges; enabling us to select the most important edges of the image. The set of selected edges provides the frame for the subsequent one-liner representation.

The next step consists of connecting the image selected edges into a single continuous curve—“the one-liner.” This step involves the ordering of the selected edges and finding curves connecting them. We solve these two problems separately. We prove that an abstract graph setting of the first problem is NP-Complete. However, we reduce this problem to a variant of the Euclidean Traveling Salesman Problem (TSP) and compute an approximate solution to it by using a readily-available TSP
solver. We solve the problem of finding connection curves by using Dijkstra’s famous shortest-path algorithm. A full software implementation for the entire one-liner determination process enabled us to extensively experiment with various input images. The thesis provides several representative one-liner examples and statistics on running the software on various inputs.
Chapter 1

Introduction

1.1 Purpose of Work

This work was inspired by drawings of Pablo Picasso. In particular, it was motivated by drawings that generate complex scenes by continuous motions of the pen, thereby creating single-curve images called “one-liners”. See Figure 1.1 for several one-liner examples. We extend this notion to image processing, suggesting the representation of any given gray-scale image by one-liner drawing.

Let us first state the terminology used throughout this work. An edge is a continuous sequence of pixels whose image derivatives either reach a local extremum or cross a zero. Sometimes the term boundary is used with the same meaning. In a simple black and white image this representation consists of a single edge; see Figure 1.2.

A “one-liner representation” of an image is a continuous curve passing through the important image edges; see Figure 1.3. This curve is not necessarily composed from the image edges only. We may even allow self intersections of the one-liner and multiple passes through the same edge.
Figure 1.1: Picasso’s one-liners

Figure 1.2: Single edge one-liners
Figure 1.3: One-liner examples
Building an “artistic one-liner representation” of a given image is a complex process, composed of multiple phases. We take into consideration the following factors:

- Since the one-liner is based on capturing image edges, the success of its creation greatly depends on edge detection and linking. Edge detection is thus the first challenge of this work. We here use a novel approach to perform the detection and linking of image edges. We refer to this process as edge exploration and give an overview of this process in Section 1.2.

- A complex image may have many candidate edges, and not all of them should be included in the one-liner. Different edges may have a different importance in carrying image-sensitive information. Selecting of a high quality subset of representative edges is very important since different subsets of image edges lead to different one-liners. To achieve a proper selection we introduce an edge-prioritization approach and sort the image edges according to their importance is the second challenge of this work. An overview of this scheme is found in Section 1.3.

- The subset of selected image edges does not necessarily represent a connected curve. This sets the third challenge of this work: ordering the selected edges and connecting them into a one-liner by using “image-driven curves.” An overview of this method is found in Section 1.4.

1.2 Edge Detection and Linking

This work uses a novel approach to perform simultaneous edge detection and linking. This approach differs significantly from the conventional schemes (described in
Chapter 1. Introduction

Section 1.2.1) but an approach similar to ours was recently taken by Udpa and Eua-Anant [6]. For the benefit of the reader we first provide in the next section a brief description of their work, which was done in parallel and independently from our work.

1.2.1 Previous Work

Conventional edge detection schemes include three operations: smoothing, edge detection, and edge labeling. Image derivative estimation is necessarily involved in the process of detecting local grey-level changes, smoothing has the role of reducing the noise, and labeling is necessary in localizing the edges and suppressing the false edges.

Sobel [12] and Prewitt [18] provided what are today the best-known operators for edge detection. They focus on detecting pixels where the gradient magnitude is high by using the convolution windows:

\[
\Delta_x = \begin{bmatrix} -1 & 0 & 1 \\ -a & 0 & a \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \Delta_y = \begin{bmatrix} -1 & -a & -1 \\ 0 & 0 & 0 \\ 1 & a & 1 \end{bmatrix},
\]

where \( a \) is a positive number (1 in case of Prewitt’s and 2 in case of Sobel’s operator.)

The image Laplacian can be estimated using the mask

\[
\nabla = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.
\]

These operators are very sensitive to noise. To overcome this problem smoothing operators preceding edge detection were proposed by Marr and Hildreth [13] and by Torre and Poggio [20]. Marr and Hildreth introduced a theory of edge detection
involving the zero-crossings of the Laplacian operator applied to a Gaussian convolved image. Smoothing consists of replacing an image pixel by the average computed in some window centered in that pixel. Together with the positive effect of noise reduction, the smoothing operator has the negative effect of information loss. The most popular smoothing operator is a 2D Gaussian kernel:

\[ g(x, y) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}}. \]

The previously-mentioned Sobel and Prewitt edge detectors are based on thresholding the gradient modulus. This results in the need to “thin” the found edges and requires further thinning or skeletonization techniques. Canny [1], following Marr and Hildreth’s ideas, proposed an algorithm that labels local maxima along the direction of the gradient vector. This algorithm considers a given pixel as a local maximum if the gradient magnitude at this pixel is greater than the gradient magnitude of its two neighbors located at the same distance from the pixel along the gradient direction. Canny’s filter solves the localization problem without further processing.

Udpa and Eua-Anant [6] recently proposed a novel approach based on a vector field image model. A new edge operator was defined and was exploited in developing a boundary-extraction algorithm based on a “particle-motion in a force field” paradigm. An edge operator was defined using the notion of the Gaussian Weighted Image Moment Vector Operator, defined by

\[ M_x(i, j) = -G_y \ast I(x, y) \approx \frac{\partial I(x, y)}{\partial y}, \]

and

\[ M_y(i, j) = G_x \ast I(x, y) \approx -\frac{\partial I(x, y)}{\partial x}. \]
where the difference masks $G_x$ and $G_y$ are

$$G_x(x, y) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2 + y^2}{2\sigma^2}} \frac{x}{\sqrt{x^2 + y^2}}$$

and

$$G_y(x, y) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2 + y^2}{2\sigma^2}} \frac{y}{\sqrt{x^2 + y^2}}$$

and the normalized edge vector field using the normalization constant $k$ was defined by

$$\vec{e}(i, j) = \frac{1}{k} (M_x(i, j) \vec{i} + M_y(i, j) \vec{j}).$$

This edge-vector field forces a particle to move along the edges while an orthogonal “normalized Laplacian-gradient vector field” guarantees that the particle will not drift away from the path of particle trajectory, where a particle motion is defined by

$$\vec{P}_k = \vec{P}_{k-1} + \alpha \vec{e}_{k-1} + \beta \nabla |\vec{e}|_{k-1}.$$

### 1.2.2 Our Approach

Our algorithm is both simple and intuitive. An edge of the image is an extremum of the image gradient, that is, a sequence of pixels with locally-maximal gradient. This, in fact, is the traditional definition of an edge. Conventional edge detectors concentrate on looking for such points. Our approach is based on exploring an edge once we have found a “seed position” on it.

An association of the gradient field of the image with a hill gives us a good intuition of this approach. The height of every point of the hill is defined as the gradient magnitude in the corresponding image point. Then, an edge in the image corresponds to a ridge of this hill; see Figure 1.4. However, the goal is not only to find all points lying on the ridge of the hill by using the gradient magnitudes, but
rather to take advantage of the gradient orientation and to use it both to “climb” on the ridge of the hill and then to stay there. Thus, the edge-exploration process is combined of selecting the starting point, moving toward the ridge of the hill, reaching it, and then moving in a direction that keeps the gradient locally maximal.

The edge-exploration algorithm proposed in this work uses an operator based on locally-available information to select the next point that belongs to the edge. This operator has two components: The first component attempts to keep the image intensity unchanged, while the second component points toward a maximal change of the gradient; see Figure 1.5.
CHAPTER 1. INTRODUCTION

While climbing to the ridge of the hill, the second component has a major contribution to the climbing direction. Thereafter, when the ridge has been reached, the first component takes a lead and makes us follow the same image intensity level, while the second component stabilizes the tracking by pointing to a direction of growing of the gradient. The key idea of the algorithm is the use of different orders of derivatives to ensure the stability and convergence of the exploration process. For better precision all the calculations are done at a subpixel level.

The algorithm described above is applied by starting at each point of the image. This way one will explore all existing edges in the image, regardless of the gradient magnitude. No thresholding is done at this stage; all image information is preserved and can be used for further processing.

The edge-exploration operator works on the gradient vector field. The conventional way for estimating the gradient and other image derivatives is to use convolutions of the scalar image with various difference masks (filters, like Sobel’s, Prewitt’s, and Canny’s; see Section 1.2.1).

Here we use Taylor’s expansion around a point to estimate the image derivatives in that point, where the expanded function is applied on the original image. For each image point (origin) we write the Taylor expansion (up to the required derivative degree) around this point for each of its neighbors. This gives us \( N \) equations (where \( N \) is the number of points in the neighborhood) with \( m \) unknowns (where \( m \) is the number of partial derivative attributes, two for the first derivative and three for the second derivative) and \( N \) remainders for noise terms. We are not really interested in finding the \( N \) Taylor’s expansion remainders. We rather look for the \( m \) derivative attributes that optimize the Taylor fitting for the neighboring points in the sense of best approximating the function over the neighborhood. To solve the minimization
problem we use a weighted least-squares method which provides us with good, and analytically-well founded estimation of the image derivatives in the point.

An example of a gradient vector field is shown in Figure 1.6. Arrows are used to exhibit the gradient magnitude and an orthogonal direction at every field point. The length of an arrow corresponds to the relative gradient magnitude, while its direction is orthogonal to the gradient orientation. This is the direction in which the gradient magnitude will remain unchanged.

1.3 Edge Prioritarization Scheme

The algorithm described in the previous section allows us to explore all image edges regardless of the gradient magnitude there.

A very important feature of the edge-exploration algorithm is its edge-prioritization capability. The number of edges in any given image can be very large (up to hundreds of thousands). Thus, edge prioritization is necessary for further processing the image.
Figure 1.7: Major edges in the bullfight image

Such a prioritization scheme will allow us to select the most important edges of the image from all the edges that were found by the edge-exploration step. Figure 1.7 demonstrates the results of the edge-exploration algorithm and the prioritization of the explored edges.

The prioritization scheme used in this work allows us to associate with each edge its ranking and hence to order all detected edges by their “importance.” The edge ranking is defined by two attributes:

- **Average gradient magnitude along the edge.** This is a trivial attribute which selects strong-enough edges to be used for further processing.

- **Number of other edges connected, or flowing into the edge.** This is a less trivial and much more important prioritization parameter. Since the edge-exploration
algorithm is applied from every image point, multiple exploratory path will flow into any specific edge. The number of converging edges is related to the height of the hill: the higher the hill is, the more paths lead to its ridge.

1.4 Connecting Selected Edges into a One-Liner

After we explored all the image edges using the algorithm described in Section 1.2, and selected the most important edges using the prioritization scheme described in Section 1.3, we can proceed with constructing a one-liner.

At this stage we consider the one-liner as a Hamiltonian cycle containing the selected set of image edges. Since the selected edges do not necessarily form a continuous curve, they can be connected by numerous one-liners. Every one-liner is specified by the order in which we connect the selected edges and by the “additional” curves that connect between them.

It should be noted that these two problems of ordering the selected edges and of determining the connecting curves are distinct, and that one may choose various solutions to each one of them. Finding the “optimal” solution to first problem appears to be computationally hard; see Section 4.1.1. We propose to reduce the problem of optimally ordering edges to an instance of the classic Traveling Salesman Problem (TSP). We also use a readily-available TSP solver to find a good approximate solution.

To complete a full one-liner representation of the image we must connect the edges according to the order provided by the TSP solution. The proposal we have to connect the endpoints of consecutive edges is based on using again the image gradient vector field. We use the famous Dijkstra algorithm to find a connecting path going through points with locally maximal gradients.
Chapter 2

Estimation of Derivatives

The conventional way to estimate the gradient and other image derivatives is using a convolution of the scalar image with difference masks and filters like Sobel, Prewitt, and different derivatives of the Gaussian (DOG) (see Section 1.2.1).

This work takes a different approach. It uses the bivariate Taylor series expansion to perform this estimation. In the sequel \( f \) is a bivariate function that has at each pixel the image intensity or grey level. The bivariate Taylor expansion of a function \( f(x, y) \) around a pixel \((x_0, y_0)\) is

\[
\begin{align*}
  f(x_0 + dx, y_0 + dy) &= f(x_0, y_0) + \frac{\partial f}{\partial x}(x_0, y_0)dx + \frac{\partial f}{\partial y}(x_0, y_0)dy \\
  &\quad + \frac{\partial^2 f}{\partial x^2}(x_0, y_0)dx^2 + 2 \frac{\partial^2 f}{\partial x \partial y}(x_0, y_0)dxdy + \frac{\partial^2 f}{\partial y^2}(x_0, y_0)dy^2 + \varepsilon,
\end{align*}
\]

where \( \varepsilon = o\left(\sum_{n=0}^{\infty} \int_{\eta}^{p} \int_{\zeta}^{q} \frac{\partial^{m+1}}{\partial x^{m+1} \partial y} dxdy\right) \).

Let us introduce the notion of the neighborhood \( \mathcal{N}_n(x_0, y_0) \) of a pixel \((x_0, y_0)\). Such neighborhood may be defined to comprise all pixels whose distance from \((x_0, y_0)\) is less than or equal to some value \( n \). Denote the number of those pixels by \( |\mathcal{N}_n| \). Pixel neighborhoods may have various forms: “circular,” elliptic, or rectangular; for
examples see Figure 2.1.

To estimate the image derivatives of degree $k$ at the pixel $P = (x_0, y_0)$ one has to:

1. Pick a neighborhood $N_n$ around the pixel $P$. The “influence” of a neighboring pixel on the derivatives’ estimation at $P$ will depend on the distance between them. Thus every pixel in $N_n$ should be assigned a weight depending on its distance from $P$; see Figure 2.2.

There are various ways to set the weights for a given neighborhood $N_n$. The neighborhood weights can be defined in a matrix notation, where each entry in the matrix specifies a weight corresponding to the neighboring pixel. Here is an example of a $N_2$ neighborhood matrix:
\[
\begin{pmatrix}
0 & 1 & 2 & 1 & 0 \\
1 & 4 & 8 & 4 & 1 \\
2 & 8 & 0 & 8 & 2 \\
0 & 1 & 2 & 1 & 0
\end{pmatrix}
\]

In this example, the influence of the pixel depends on the distance between the neighboring pixel and the origin. However, other weighing schemes and neighborhood widths can be used.

Note that each neighborhood can be placed inside a rectangular bounding box, where some pixels have zero-weights. In other words \((x_i, y_i) = (x_0 + dx_i, y_0 + dy_i)\) is one of the neighboring pixels if \(-n \leq dx_i \leq n\) and \(-n \leq dy_i \leq n\).

2. Assuming that the derivative’s degree is \(k\), let us write for each neighboring pixel the Taylor expansion of the degree \(k\) around the pixel \(P = (x_0, y_0)\), see Equation (2.2). That gives us \(|N_n|\) equations with \(|N_n| + m\) unknowns, where \(m\) is the number of different partial derivatives of the degree up to \(k\) and \(|N_n|\) is the number of remainders \(\varepsilon_i\).

\[
f(x_0 + dx_i, y_0 + dy_i) = f(x_0, y_0) - \frac{\partial f}{\partial x}(x_0, y_0)dx_i - \frac{\partial f}{\partial y}(x_0, y_0)dy_i - \frac{\partial^k f}{\partial x^k}(x_0, y_0)dx_i^k - k \frac{\partial^{k+1} f}{\partial x^{k+1} \partial y}(x_0, y_0)dx_i^k dy_i - \frac{\partial^k f}{\partial y^k}(x_0, y_0)dy_i^k = \varepsilon_i
\]  \hspace{1cm} (2.2)

In the last equation \((dx_i, dy_i)\) is the distance from the original pixel \((x_0, y_0)\) to the neighboring pixel \((x_i, y_i)\), and \(\varepsilon_i\) is an approximation error.
CHAPTER 2. ESTIMATION OF DERIVATIVES

Denote the left-hand side of Equation (2.2) by \( F_i \), thus \( F_i = \varepsilon_i \), for \( 1 \leq i \leq N \).

3. To minimize the error in the estimation of image derivatives, one can find \( k + 1 \) partial derivatives of the Taylor expansion, which minimize the remainders \( \varepsilon_i \) in Equation (2.2).

One way to solve this problem is to minimize the sum of the squares of the remainders, for example by using the Weighted Least Square method, see Equation (2.3).

\[
L \equiv \sqrt{\sum_{i=1}^{N} w_i F_i^2} \tag{2.3}
\]

In this method each equation is multiplied by the weight \( w_i \) corresponding to the neighboring pixel.

\( L \) is a function of \( m \) variables (the partial derivatives sought). The partial derivatives that minimize the right hand side of Equation (2.3) are the extrema of the function \( L \). Thus in order to compute them one should find values for which the partial derivatives of the function \( L \) vanish. This gives us a system of \( m \) equations with \( m \) unknowns, which can be solved by using any one of several well-known methods.

In the next two sections we provide a detailed example of such systems for the first two derivatives and their solution.

We also observe that the method of derivatives estimation using the pixel’s neighbors “creates margins” around the original edge; see Figure 2.3. However, this side-effect doesn’t influence the results of this work. It is handled by the edge-exploration algorithm described in Section 3.
2.1 Estimation of First Derivatives

This section demonstrates the first derivatives estimation using the method described above.

In this section we use the notation

\[ p(x,y) \equiv \frac{\partial f}{\partial x}(x,y), \quad q(x,y) \equiv \frac{\partial f}{\partial y}(x,y), \]

and rewrite equation (2.1) using only the first derivatives:

\[ f(x_0 + dx, y_0 + dy) = f(x_0, y_0) + p(x_0, y_0)dx + q(x_0, y_0)dy + \varepsilon. \]

In this work \( f(x_0, y_0) \) denotes the intensity of the pixel \( (x_0, y_0) \). Thus a noisy pixel may have a negative influence on the estimation of \( f(x_0, y_0) \). To improve the estimation we can use the weighted average of the intensities of the pixels in some neighborhood of the original pixel (similar to those described in Section 2). However, since derivatives estimation uses a weighted average of the pixel intensities in some neighborhood, the use of the \( f(x_0, y_0) \) without smoothing it does not affects the estimation results.
CHAPTER 2. ESTIMATION OF DERIVATIVES

Using the $F_i$ notation and rewriting equation (2.2) for the first derivatives, we get:

$$F_i = f(x_0 + dx_i, y_0 + dy_i) - f(x_0, y_0) + p(x_0, y_0)dx_i + q(x_0, y_0)dy_i \quad (2.5)$$

To find the values of $p$ and $q$ that minimize Equation (2.3) we solve the equation

$L' = \left( \frac{\partial L}{\partial p}, \frac{\partial L}{\partial q} \right) = 0$:

$$\frac{\partial L}{\partial p} = -\sum_{i=1}^{N} \frac{w_i F_i dx_i}{L} = 0, \quad (2.6)$$

$$\frac{\partial L}{\partial q} = -\sum_{i=1}^{N} \frac{w_i F_i dy_i}{L} = 0. \quad (2.7)$$

That is,

$$\mathbf{A} \begin{pmatrix} p(x_0, y_0) \\ q(x_0, y_0) \end{pmatrix} = \mathbf{b}, \quad (2.8)$$

where

$$\mathbf{A} = \begin{pmatrix} \sum_{i=1}^{N} w_i dx_i^2 & \sum_{i=1}^{N} w_i dx_i dy_i \\ \sum_{i=1}^{N} w_i dx_i dy_i & \sum_{i=1}^{N} w_i dy_i^2 \end{pmatrix}$$

and

$$\mathbf{b} = \begin{pmatrix} \sum_{i=1}^{N} w_i (f(x_0 + dx_i, y_0 + dy_i) - f(x_0, y_0)) dx_i \\ \sum_{i=1}^{N} w_i (f(x_0 + dx_i, y_0 + dy_i) - f(x_0, y_0)) dy_i \end{pmatrix}.$$
the solution is:

\[ p = \frac{|A^1_b|}{|A|}, \quad q = \frac{|A^2_b|}{|A|} \]  

(2.9)

### 2.2 Estimation of Second Derivatives

This section demonstrates the second derivatives estimation using the method described above.

In this section we use the notation

\[
a(x, y) \equiv \frac{\partial^2 f}{\partial x^2}(x, y), \quad b(x, y) \equiv \frac{\partial^2 f}{\partial x \partial y}(x, y), \quad c(x, y) \equiv \frac{\partial^2 f}{\partial y^2}(x, y),
\]  

(2.10)

and rewrite Equation (2.1) including the second derivatives:

\[
f(x_0 + dx, y_0 + dy) = f(x_0, y_0) + p(x_0, y_0)dx + q(x_0, y_0)dy \\
+ a(x_0, y_0)dx^2 + 2b(x_0, y_0)dx dy + c(x_0, y_0)dy^2 + \varepsilon
\]  

(2.11)

To approximate the second derivatives we use the same Taylor expansion method. By Equation (2.11) each pixel in the \(N\) neighborhood satisfies the following equation:

\[
\tilde{F}_n \equiv f(x_0 + dx_n, y_0 + dy_n) - f(x_0, y_0) - p(x_0, y_0)dx_n - q(x_0, y_0)dy_n \\
- a(x_0, y_0)dx_n^2 - 2b(x_0, y_0)dx_n dy_n - c(x_0, y_0)dy_n^2 = \varepsilon_n
\]  

(2.12)

where \(p, q\) are the first derivatives calculated by Equation (2.9), and \(a, b, c, d\) are the partial second derivatives, defined in Equation (2.10). Then,

\[
\bar{L} \equiv \sqrt{\sum_{i=1}^{N} w_i \tilde{F}_i^2},
\]  

(2.13)

\[
\frac{\partial \bar{L}}{\partial a} = -\sum_{i=1}^{N} w_i \tilde{F}_i \frac{dx_n^2}{L} = 0,
\]  

(2.14)
\[ \frac{\partial L}{\partial b} = -2 \sum_{i=1}^{N} \frac{w_i \tilde{F}_i}{L} \, dx_i \, dy_i = 0, \]  
(2.15) 
\[ \frac{\partial \bar{L}}{\partial c} = - \frac{\sum_{i=1}^{N} w_i \tilde{F}_i \, dy_i^2}{L} = 0 \]  
(2.16) 
\[ \tilde{\mathbf{A}} \begin{pmatrix} a(x_0, y_0) \\ b(x_0, y_0) \\ c(x_0, y_0) \end{pmatrix} = \bar{\mathbf{b}} \]  
(2.17) 

where 
\[ \tilde{\mathbf{A}} = \begin{pmatrix} \sum_{i=1}^{N} w_i dx_i^4 & 2 \sum_{i=1}^{N} w_i dx_i^3 dy_i & \sum_{i=1}^{N} w_i dx_i^2 dy_i^2 \\ \sum_{i=1}^{N} w_i dx_i^3 dy_i & 4 \sum_{i=1}^{N} w_i dx_i^2 dy_i^2 & \sum_{i=1}^{N} w_i dx_i dy_i^3 \\ \sum_{i=1}^{N} w_i dx_i^2 dy_i^2 & 2 \sum_{i=1}^{N} w_i dx_i dy_i^3 & \sum_{i=1}^{N} w_i dy_i^4 \end{pmatrix} \]

and 
\[ \bar{\mathbf{b}} = \begin{pmatrix} \sum_{i=1}^{N} w_i \left( f(x_0 + dx_i, y_0 + dy_i) - f(x_0, y_0) - p(x_0, y_0) dx_i - q(x_0, y_0) dy_i \right) dx_i^2 \\ \sum_{i=1}^{N} w_i \left( f(x_0 + dx_i, y_0 + dy_i) - f(x_0, y_0) - p(x_0, y_0) dx_i - q(x_0, y_0) dy_i \right) dx_i dy_i \\ \sum_{i=1}^{N} w_i \left( f(x_0 + dx_i, y_0 + dy_i) - f(x_0, y_0) - p(x_0, y_0) dx_i - q(x_0, y_0) dy_i \right) dy_i^2 \end{pmatrix} \].

Using equation (2.9) and the solution of system (2.17), we obtain an analytical representation of the first and second derivatives of the image magnitude.

This system of equations can be solved by using the triangulation method or by using Kramer’s rule.
Chapter 3

Edge Exploration

Detecting image edges is an old and well-known problem. Image edges are widely considered as carrying a lot of information [20] and are used traditionally for representing images. This work uses image edges to build a different kind of graphical image representation: the one-liner.

In this section we regard image edges as continuous curves consisting of points with locally-maximal gradient. Our edge-exploration algorithm, described in Section 3.2, detects these curves without any additional processing, such as thinning, skeletonization, etc. The algorithm is based on the image-flows concept described in Section 3.1. It uses an edge-exploration operator applied on the gradient vector field for extracting image edges. Two versions of the operator are presented in Sections 3.3 and 3.4.

In a complex image almost every pixel has a positive gradient and there are many local maxima of the gradient. One possible approach for reducing the number of detected edges is by setting a threshold on the minimum accepted gradient. However, this method leads to a significant loss of information at a very early stage of the
processing. Therefore we use here another approach.

We start with finding all the image edges regardless of their importance or intensity. Then we prioritize all the edges using information collected during the edge-exploration algorithm. Finally we select the most important edges for our representation.

A typical image consists of a large number of edges (up to hundreds of thousands of edges). These edges differ by their characteristics. Some of them represent crucial details of the image and contribute to its representation. Other edges are the result of noise or belong to some patterned background. The classification of image edges into “more” or “less” important edges is a complex task. Image representations based on selected edges hence greatly depend on the quality of the selection process.

The approach presented in this section differs significantly from conventional methods for edge detection as described in Section 1.2.1. In parallel with our work, Udpa and Eua-Anant [6] recently presented a novel approach for edge detection and boundary extraction. Their ideas are very similar to ours, as we discussed in Section 1.2.1.

3.1 Image Flows

In this section we introduce the idea of image flows and its application to the edge-exploration process.

We associate a given image with its gradient vector field. We build this field using a method described in Section 2.1. This field in each its point consists a vector orthogonal to the image gradient and having a magnitude equal to the gradient magnitude. It is well known that this vector points in a direction preserving image
intensity.

This vector field defines “flows” in the image. At every point of the vector field there is a “force” operating on the object placed in this field. This force will have different strengths and directions depending on the location in the vector field.

A trivial force which can be applied in such a field is a motion in the field direction, i.e., at every point the object moves in the direction defined by the local vector field. In this example, the object placed at an arbitrary pixel follows the same image intensity level, regardless of the gradient magnitude.

Image edges are presented in this vector field by vectors with locally maximal length, see Figure 3.1, which correspond to the pixels with locally-maximal gradient magnitude. Each such pixel corresponds to some intensity level of the image. In simple images, all edge pixels lie at the same intensity level. Thus one of the possible edge exploration schemes can start from an arbitrary edge pixel and follow the equal intensity level by moving in the direction pointed by the gradient field vector.

This strategy would not work for complex images, where the image intensity varies along the edges, but it still gives us a good sense about an overall direction in which
we should proceed with our edge exploration. In complex images we need to introduce another force, which not just follows an intensity preserving direction, but also drives toward the direction of increase of gradient magnitude. Thus a particle placed in this field feels two forces: one leading it along equal intensity contours of the image, and another driving it toward an intensity level with higher gradient magnitude.

The new strategy allows to us not only to explore edges starting from the initial point, but also to discover strong edges from nearby points. This happens due to the force leading us toward higher gradient magnitudes, i.e., toward the nearby image edge. Once the edge is reached, the first force component makes us follow the equal intensity level of the image while the second keeps the particle on the edge.

To conclude, the edge-exploration operator defines forces applied to test particles in an image-defined vector field. Here we shall use an operator as described in Section 3.3.

3.2 The Algorithm

The purpose of the algorithm is to explore all image edges and to collect information necessary for edge prioritization. This is a scan-line algorithm that starts from the bottom-left pixel of the image and proceeds to the right, and bottom-up line-by-line. It maintains several data structures needed to store the already-explored edges, the current state of the algorithm, and some additional information about the image pixels.

For each image pixel the algorithm performs the following steps:

1. Find a candidate edge starting at the current pixel (origin);

2. Stabilize the candidate edge and;
Figure 3.2: Exploring one edge
3. Subdivide the candidate edge into edge segments.

Due to the nature of this algorithm, it can be executed more than once for the same point (see Section 3.2.2). Beside exploring edges, this algorithm collects information allowing to prioritize image edges.

We use two edge-prioritization parameters calculated for each edge:

- The number of starting points from which particles converge to the edge and;
- An average gradient magnitude along the edge path.

We prioritize edges using a two-key scheme. We sort all edges by the number of paths that converged to it, and if two edges have the same number we use an average gradient magnitude. We allow the user to select a “minimal average gradient” and a “minimal number of converging particles,” and keep only edges meeting these criteria sorted by the prioritization parameters, as mentioned above.

Beside thresholding the average gradient and the number of converging particles, we allow the user to select the number of the most important edges without any limits.

### 3.2.1 Finding candidate edges

We start from the initial point, and move to the next pixel along the edge by using the edge-exploration operator. The algorithm assumes that the resolution of the operator is better than the size of a single pixel. The new “explored” pixel is added to the edge, and the algorithm proceeds using this pixel to find the next one.

The algorithm has three termination conditions:

- Reaching a “white” pixel (a pixel with zero gradient).
CHAPTER 3. EDGE EXPLORATION

Figure 3.3: Edge intersection and merging

The exploration operator does not implement a notion of inertia, thus a point that does not "contribute" any force to the edge-exploration process is considered a terminal point and the exploration process stops there.

- Reaching a stream of pixels belonging to an already-explored edge (including the current edge).

In order to support edge intersections, the algorithm does not stop upon detecting a pixel belonging to another edge. It proceeds its course until it explores a stream of $k$ successive pixels belonging to another edge (see Figure 3.3). We found out empirically that a value of $k = 5$ was efficient in practically all cases. However, two edge sometimes share several pixels in the neighborhood of their intersection. In such a case we divide the second edge into two subsections and process them separately.

Since the edge-exploration operator operates at a subpixel resolution, two pixels are considered equal if their coordinates are identical up to the second digit after the decimal point.

When the algorithm identifies an intersection of two edges, it increases by one
the number of converging paths of both edges. The same rule applies for self-intersecting edges.

- Reaching a pixel that is located outside of the image boundary.

The output of this step of the edge-exploration algorithm is a set of candidate image edges.

3.2.2 Stabilizing candidate edges

The edge-exploration algorithm does not make any assumption about the location of the starting pixel. Thus the start point is not necessarily located on a “strong” image edge. Therefore, the first step of the algorithm can merge a “weak” (ghost)
path starting from an arbitrary point to a “strong” image edge (see Figure 3.4).

Such merges can result in the detection of unnatural image edges. To resolve this problem we use the following idea: Instead of halting the algorithm upon reaching the first termination point, we restart the process backward from that termination point, as shown in Figure 3.4(d).

The stabilization of a candidate edge is considered complete when the average gradient of the last $m$ edge points is less than or equal to the average gradient of the first $m$ edge points. This guaranties that the starting point is not located on a “weak” edge, and eliminates most cases of “ghost” edges. In practice we used the value $m = 10$.

A “stabilization candidate-edge” can be traveled several times due to different paths leading to it. However, on the same exploration path it may participate in at most two passes, which ensured the termination of the algorithm.

### 3.2.3 Subdividing candidate edges

The last step of the edge-exploration algorithm is subdividing the candidate edges into edge segments. The purpose of this step is to divide the candidates into edge curves with homogeneous average gradient (see Figure 3.2(d)). This allows us to detect and isolate “weak” edges within the candidate edges.

To perform the subdivision we define a pixel-window on a curve of width $w$ pixels. In this work we used the value $w = 5$. First we place the window at the beginning of the candidate edge and calculate the average gradient of all curve points covered by the window. This value is considered as the starting edge magnitude $m_0$. We associate with this value a range of allowed average magnitudes, defined by $[m_0 \ast (1 - d), m_0 \ast (1 + d)]$. In this work we set $d = 0.4$. Then we move the window along the
edge and calculate at every pixel the average gradient of all window pixels. We check whether the average falls in the allowed range. If it does, we move the window to the next pixel. Otherwise, we split this edge into two edge segments. At such a splitting point the calculated average is considered as the new edge average magnitude; we then proceed with the algorithm till we reach the other end of the original candidate edge (see Figure 3.5).

### 3.3 The Edge-Exploration Operator

This section discusses in detail the edge-exploration operator. This operator provides us with the location of the next edge pixel, when it is locally applied to the gradient vector field. It works at a subpixel accuracy level to improve the quality of the resulting edges.

Let us denote an input image by $I(x, y)$ and the corresponding gradient field as $\nabla I(x, y) = [p(x, y), q(x, y)]$, where $p$ and $q$ are defined by Equation (2.4) in Section 2.1. This field consists of vectors calculated for every image pixel using the method described in Section 2.1.
CHAPTER 3. EDGE EXPLORATION

(a) Climbing to the ridge  
(b) Walking along the ridge

Figure 3.6: The components of the edge-exploration operator

The explicit edge exploration process is defined by:

\[
\frac{\partial P}{\partial t} = \alpha \left( \frac{\nabla I(x, y)}{\|\nabla I(x, y)\|} \right) + (1 - \alpha) \left( \frac{\nabla \|\nabla I(x, y)\|}{\|\nabla I(x, y)\|} \right), \quad 0 \leq \alpha \leq 1
\]  

(3.1)

The left-hand side of Equation (3.1) represents the time shift in both coordinates, which is required to move to the next pixel. The right-hand side contains two terms. The first term represents a motion in the direction that preserves image intensity. This is achieved by pointing orthogonally to the gradient direction. The second term represents a motion in the direction of maximal change of the image gradient.

Each one of the two terms contributes its share in the edge-exploration process. The edge-exploration algorithm does not assume that it starts at a pixel on an edge (see Section 3.2). In fact usually a starting point does not lie on an image edge as we would normally define it. When approaching a real edge, the second term points to the “fastest” way to reach the edge and has the major influence on the selection of the next subpixel location. After the edge was reached, the first term takes the leading role in tracking the edge, whereas the second term “fine-tunes” the direction toward a growing gradient (see Figure 3.6).

The tuning parameter \( \alpha \) controls the proportion between the two components of the operator. A value of \( \alpha = 0.1 \) suited all grey-scale images with which we have
CHAPTER 3. EDGE EXPLORATION

experimented. For some black/white images this parameter can be reduced even further to \( \alpha = 0.05 \). We explain this by the fact that black/white images have a very sharp gradient change near the edges, making possible the reduction of the influence of the second component.

An attempt to use a smaller value of \( \alpha \) leads to increasing the influence of the second component on the edge-exploration process. This results in changing the general motion direction from along equal intensity levels to the direction of maximal change of gradient. A small change in the value of \( \alpha \) increases the “width” of the explored edges (see Section 5.2 for the image examples), and a sharp change in the value of \( \alpha \) will cause the edge-exploration operator to lose the edge.

Note that both components of the operator are normalized. This allows tight control over the total displacement between two consecutive points by keeping it at subpixel resolution. A large value of the gradient or a fast change in the gradient magnitude may lead to a large step spanning several pixels. This may not only hurt the smoothness of the explored edge, but it may even cause the loss of the edge.

The fine tuning of the granularity in this step can be achieved by multiplying Equation (3.1) by a scaling factor:

\[
\frac{\partial P}{\partial t} = \delta \left( \alpha \frac{\nabla \| \nabla I(x, y) \|}{\nabla \| \nabla I(x, y) \|} \right) + (1 - \alpha) \frac{\nabla \| \nabla I(x, y) \|}{\nabla \| \nabla I(x, y) \|}, \quad 0 \leq \alpha, \delta \leq 1.
\]

(3.2)

In this work we used a value of \( \delta = 0.25 \) in order to ensure a quarter-pixel granularity.

Let us simplify Equation (3.1) and demonstrate how it can be evaluated using terms introduced in Sections 2.1 and 2.2. Rewrite equation (3.1) using the coordinates
of the point \(P\):

\[
\frac{\partial x}{\partial t} = \alpha \left( \frac{\nabla I(x, y)}{\|\nabla I(x, y)\|} \right)_x + (1 - \alpha) \left( \frac{\nabla \|\nabla I(x, y)\|}{\|\nabla I(x, y)\|} \right)_x,
\]

\[
\frac{\partial y}{\partial t} = \alpha \left( \frac{\nabla I(x, y)}{\|\nabla I(x, y)\|} \right)_y + (1 - \alpha) \left( \frac{\nabla \|\nabla I(x, y)\|}{\|\nabla I(x, y)\|} \right)_y.
\]

Substitute now in Equations (3.3) terms introduced in Equation (2.4) from Section 2, and note that

\[
(\nabla I(x, y))_\perp = [-q, p].
\]

The gradient magnitude can be written as:

\[
\|\nabla I(x, y)\| = \sqrt{\left( \frac{\partial I(x, y)}{\partial x} \right)^2 + \left( \frac{\partial I(x, y)}{\partial y} \right)^2} = \sqrt{p^2 + q^2}.
\]

There are two methods for estimating the gradient of the gradient magnitude. One approach is to apply the method we used for estimating the gradient of the image gradient magnitude (see Section 2.1). The main disadvantage of this method is the increase in the estimation error. A second approach is to divide the gradient of the gradient magnitude \((\nabla \|\nabla I(x, y)\|)\) into its components, and then to estimate it using the first and second partial derivatives, as described in Sections 2.1 and 2.2, respectively.

In this work we used the second method to estimate the gradient of the gradient magnitude, as follows:

\[
\nabla \|\nabla I(x, y)\| = \left[ \frac{\partial}{\partial x}, \frac{\partial}{\partial y} \right] \sqrt{\left( \frac{\partial I(x, y)}{\partial x} \right)^2 + \left( \frac{\partial I(x, y)}{\partial y} \right)^2} - \frac{1}{\|\nabla I\|} \left[ \frac{\partial}{\partial x} \frac{\partial^2 I}{\partial x^2} + \frac{\partial}{\partial y} \frac{\partial^2 I}{\partial x \partial y} + \frac{\partial}{\partial x} \frac{\partial^2 I}{\partial y \partial x} + \frac{\partial}{\partial y} \frac{\partial^2 I}{\partial y^2} \right]
\]

\[
= \frac{1}{\sqrt{p^2 + q^2}} [pa + qb, pb + qc].
\]

(3.5)
CHAPTER 3. EDGE EXPLORATION

Thus, the magnitude gradient of the gradient magnitude can be estimated via

$$\left\| \nabla \| \nabla I(x, y) \| \right\| = \frac{1}{\sqrt{p^2 + q^2}} \sqrt{(pa + qb)^2 + (pb + qc)^2}.$$  (3.6)

where $a, b, c$ are defined by Equation (2.10).

By substituting Equations (3.4), (3.5), and (3.6) in Equations (3.3), we obtain:

$$\frac{\partial x}{\partial t} = \alpha \frac{-q}{\sqrt{p^2 + q^2}} + (1 - \alpha) \frac{pa + qb}{\sqrt{(pa + qb)^2 + (pb + qc)^2}}.$$  (3.7)

$$\frac{\partial y}{\partial t} = \alpha \frac{p}{\sqrt{p^2 + q^2}} + (1 - \alpha) \frac{pb + qc}{\sqrt{(pa + qb)^2 + (pb + qc)^2}}.$$

Equations (3.7) define the edge-exploration operator in terms of the first and second partial image derivatives.

The real coordinates space is used to minimize the estimated errors. Due to the nature of Equations (3.7), the displacement is not assumed to be discrete and it allows smooth transitions between pixels. However, the first and second derivatives of the image magnitude are calculated based on discrete pixel values. To approximate the derivatives on the subpixel level, we use an interpolation method, described below in Section 3.6.

3.4 A Heuristic Operator

This section introduces another edge-exploration operator simpler than the previous operator. It suits the algorithmic approach described in Section 3.2 and operates on the image gradient vector field in a discrete grid.

This operator uses the definition of an edge in the exploration process. It uses a
CHAPTER 3. EDGE EXPLORATION

heuristic approach for selecting the next pixel belonging to the edge. Unlike the edge-exploration operator presented in Section 3.3, this operator incorporates a notion of “inertia” in the motion of the “particle” performing edge exploration. It uses the past trace of the exploration path to identify the next pixel on the edge. More precisely, the next pixel to be explored is determined based on the direction from which we entered the current pixel. Given the direction, the exploration algorithm picks three neighboring pixels continuing the exploration in the same direction (see Figure 3.7). The pixel with the largest gradient magnitude is selected to be the next pixel along the edge.

Despite its simplicity, this operator too produces relatively good results. However, due to the discretized scale, the explored edges are more much less smooth than the edges detected using the previously described edge-exploration operator.

3.5 Edge Prioritization

This sections discusses the edge prioritization (or ranking) process. The algorithm presented above detects all image edges. Since the number of edges in any given
CHAPTER 3. EDGE EXPLORATION

![Figure 3.8: Selected image edges](image)

image is relatively large (up to hundreds of thousand), one needs to select a subset of the most important image edges.

Here we present an edge-prioritization method which allows to rank the image edges accordingly to their importance for the image representation. This method uses information collected during the edge-exploration process. Figure 3.8 demonstrates all the detected edges and a selected subset.

Each explored edge is characterized by two parameters:

- Average gradient magnitude along the edge;
- Number of other edges connected or flowing into the edge.

Given two “edge-importance” characteristics one can pick the thresholds for both parameters and thereby select the “interesting” subset of edge-curves. The next step can be ordering the selected curves using a two-key scheme: first order by number of connected edges, and then, if several edges got the same rate, order them by the gradient magnitude along the edge. This way one can obtain a set of selected edges ordered by their importance.
The first parameter ranks detected edges by their “strongness.” This is a trivial prioritization parameter which is widely used by traditional edge-detection schemes to detect the edge pixels among other pixels of the image. One of the stages of the edge-exploration algorithm (see Section 3.2.3) subdivides the detected edges into edge segments having a homogeneous average gradient. This isolates “weak” edges within the candidate edges and makes the edge-prioritization by their “strongness” cleaner.

The second parameter promotes the long edges having many neighboring edges flowing into them. Because of the nature of the edge-exploration algorithm, the fact that a secondary edge converges to a primary edge, means that the direction along the secondary edge in which the image gradient magnitude is increased leads to the primary edge, or that the primary edge has a superior gradient “strength.” Having many connecting edges makes an edge a local “convergence curve”, a place to which a particle will be drifted from many source points.

The use of both prioritization attributes described above allows us to select and rank the “most important” image edges and to use them for further processing. Figure 3.9 demonstrates the influence of each ranking parameter, while Figure 3.10 shows several sets of selected image edges.
CHAPTER 3. EDGE EXPLORATION

Figure 3.10: Varying selected edges

(a) Subpixel with integral coordinates  (b) Subpixel with nonintegral coordinates

Figure 3.11: Subpixel location

3.6 Derivative Interpolation

Our current goal is to approximate the first and second derivatives at the “subpixel location” $P = (x, y)$ using the derivatives calculated for discrete pixels.

If $P$ has integral coordinates, then we do not need to interpolate at all and we simply use the first and second derivatives as estimated for that pixel. For a subpixel location $P$ with nonintegral coordinates we proceed as follows.

We examine the four closest neighboring pixels of $P$; see Figure 3.11. Denote these pixels by $P_1, P_2, P_3,$ and $P_4$. Each of these pixels has a first and a second derivative
calculated as described in Section 2. Since four spatial points do not necessarily lie in the same plane, we cannot simply move on a line defined by any two of these points.

Consider the pixels shown in Figure 3.12. We interpolate linearly the derivatives in the point $P_{12}$ that lies between $P_1$ and $P_2$. Similarly, we interpolate the derivatives in the point $P_{34}$ that lies between $P_3$ and $P_4$. The points $P_{12}$ and $P_{34}$ are selected so that $P$ lies between these two subpixels. Finally, the derivatives at $P$ are calculated using the same linear-interpolation method.

Another way to estimate the image derivative value at the pixel $P$ is to use the formula:

$$\frac{\partial y}{\partial t}(P) \approx \sum_{i=1}^{n} \frac{1}{d(P_i, P)} \sum_{i=1}^{n} \frac{\partial y}{\partial t}(P_i) \frac{1}{d(P_i, P)}$$

This formula demonstrates estimation of the $y$-partial derivative at the point $P$ as a weighted average of the $y$-partial derivatives of some neighboring $P_i$. In our case $n = 4$, and $d(\cdot, \cdot)$ denotes the Euclidian distance between two points in the plane.
Chapter 4

Linking the Edges

This section discusses the last step of the one-liner generator process: linking the most important image edges into a one-liner.

As mentioned above, the edge-exploration algorithm presented in Section 3 detects all the edges regardless of their significance in the image. Some of them are background edges or the result of noise; see Figures 4.1(b,e,h).

However, the edge-exploration algorithm allows us to prioritize the detected edges and to select from them only the most important edges. See Figures 4.1(c,f,i) for an illustration.

We now present our method for connecting the selected image edges into a continuous curve—a one-liner. The problem of connecting $n$ given edges $e_1, \ldots, e_n$ into a single continuous curve $C$ can be divided into two separate tasks:

- Order the set of edges so that their connected version is “nice” and intuitive.
- Determine a set of curves that connect the original edges to form a one-liner.

We accomplish each task separately and combine the solutions together.
Figure 4.1: The main edges in various images
4.1 Ordering the Edges

From all the candidate edges we select $n$ curves, and each will connect via its endpoints to two other selected edges. A set of $n$ edges can be connected into a single continuous curve in numerous ways. There are $\Theta(n^2)$ possible curves connecting the original edges; see Figure 4.2(b). From all the possible connecting curves we have to choose $n$ to complete the one-liner.

Since there are many ways to obtain a one-liner from the set of original edges, we need to provide a criterion for evaluating the “goodness” of a one-liner. Since the one-liner must be a graphical image representation, we will seek a criterion that is based on the similarity between the image and its representation. This can be achieved by using “image-driven” connection curves that somehow “fit naturally” into a one-liner image representation. In this work we chose to limit the total length of the connecting
curves and make them “image-driven” by using the approach described in Section 4.2.

The edges chosen to participate in the one-liner are planar curves, as well as the expected connectors. However, since we allow a one-liner to intersect itself, our solution will not depend on the behavior of a curve between its endpoints. Thus, the original one-liner problem can be reduced to the problem of connecting the \( n \) given segments into a continuous polyline with an upper bound on the length of the connectors; see Figure 4.3.

This reduction allows us to state the one-liner problem in terms of a graph. First let us define the graph \( G \) corresponding to the one-liner problem. Let \( S \) be a set of explored edges in the image. We assume that no two edge-curves of \( S \) share an endpoint.\(^1\)

**Definition 1** Let \( G = (V, E) \) be a undirected graph defined by:

- \( V \), the set of the endpoints of all the edges of \( S \) and
- \( E \), the set of all possible segments connecting vertices in \( V \).

\(^1\)Otherwise, this endpoint can be replaced by two infinitesimally-close distinct points.
CHAPTER 4. LINKING THE EDGES

It follows from Definition 1 that \( G \) does not contain any self or parallel edges.

We replace the curves of \( S \) by line segments having the same endpoints. Let \( E' \) be the set of these segments. Definition 1 implies that \( E' \subset E \), and \( E' \) is a “spanning independent set of \( G \),” satisfying the following conditions:

- No two edges in \( E' \) share an endpoint and
- Every vertex in \( V \) is an endpoint of some edge in \( E' \).

Now we are ready to state the one-liner problem in graph terms:

**Problem 1 One-liner:** Let \( G = (V, E) \) be a weighted graph with a spanning independent set of edges \( E' \). Augment (if possible) \( E' \) with a minimum-weight set \( E^* \subseteq (E \setminus E') \) such that \( G^* = (V, E^* = E' \cup E^*) \) contains an Euler path or Euler cycle.

In Section 4.1.1 we prove that the one-liner problem (in its abstract graph notation, disregarding its image origin) is NP-Complete. In Section 4.1.2 we present a few naive attempts to solve this problem using greedy algorithms, while in Section 4.1.3 we introduce an efficient approximate solution to this problem by reducing it to a variant of the Euclidean TSP.

4.1.1 The one-liner problem is NP-Complete

**Theorem 1** Both variants of Problem One-liner (path and cycle) are NP-Complete.

**Proof:** We establish the claim by a reduction from the Hamiltonian Path/Cycle problems, which are known to be NP-Complete [7]. Given an instance \( \overline{G} = (V, \overline{E}) \) of the Hamiltonian path/cycle problem, we construct an instance \( G = (V, E) \) (with a subset \( E' \) of \( E \)) of the one-liner path/cycle problem, as follows.
For each vertex $v \in \overline{V}$ we construct two vertices $v_1, v_2 \in V$ and an edge $v_1v_2 \in E'$ with weight 0.

For each edge $uv \in \overline{V}$ we construct four edges $u_1v_1, u_1v_2, u_2v_1, u_2v_2 \in E$, all of which are not in $E'$ and with weight 1.

See Figure 4.4 for an illustration.

We show below that the graph $\overline{G}$ contains a Hamiltonian path (resp., cycle) if and only if the set of edges $E'$ in the graph $G$ can be augmented into an Euler path (resp., cycle) by additional edges in $E$ whose total weight is no more than $|V| - 1$ (resp., $|V|$).

It is clear that the weight of any solution to the one-liner path (resp., cycle) instance is at least $|V| - 1$ (resp., $|V|$), because there are $|V|$ independent edges of weight 0 in $G$, and in order to connect them into one path (resp., cycle) at least $|V| - 1$ (resp., $|V|$) additional edges, all with weight 1, are required. Thus we need to show that this total weight is attainable if and only if the graph $\overline{G}$ contains a Hamiltonian path (resp., cycle).

First, assume that $\overline{G}$ contains a Hamiltonian path (resp., cycle). Then, by construction, it induces an augmentation of $E$ into a path (resp., cycle) $E^*$ in $G$ whose weight is $|V| - 1$ (resp., $V$). This is done by collecting all the 0-weight edges in $G$
induced by vertices in $\overline{G}$ and by choosing consistently 1-weight edges in $G$ induced by
the order of vertices along the Hamiltonian path $P$ (resp., cycle) in $\overline{G}$. For example, if
$P = u_1v_2 ...$ in $\overline{G}$, then we collect in $G$ the edges $u_2v_1, v_1v_2, v_2w_1, w_1w_2, ...$
Altogether we obtain the path (cycle) $u_1u_2, u_2v_1, v_1v_2, v_2w_1, w_1w_2, ...$

Second, assume that there exists a path (resp., cycle) of weight $|V| - 1$ (resp. $|V|$) in $G$, that contains all the 0-weight edges in $E'$. Since the 0-weight edges are
independent (sharing no endpoints), there are no two consecutive 1-weight edges along
the path (cycle). (Otherwise the total weight would exceed $|V| - 1$ (resp., $|V|$).) Thus,
the 0- and 1-weight edges are alternating along the path (cycle) in $G$. This instantly
induces a Hamiltonian path (cycle) in $\overline{G}$.

Finally, it is trivial to verify in $O(|V|)$ time whether a candidate solution to the
one-liner problem is correct. The whole reduction is linear in the size of the graph $\overline{G}$.
Hence, both variants of the one-liner problem are NP-Complete.\footnote{More precisely, the decision versions of one-liner are NP-Complete, while its optimization versions are NP-Hard.}

\section{Naive heuristic approaches}

Let $S = \{e_1, e_2, ..., e_n\}$ be the edge-curves explored in the image. The edge-exploration
algorithm sorts the edges according to their priority, that is, $e_1$ has the highest pri-

ority and $e_n$ has the lowest. A trivial solution to the ordering problem is to connect
the edges according to this order: $e_1$ to $e_2$, $e_2$ to $e_3$, etc. However, the priorities of
dges have nothing to do with their order along a one-liner. Hence, any arbitrary
permutation of the edges may be equally good. The major disadvantage of this solu-
tion is that it does not have any influence on the total length of the added connecting
curves. In most cases it gives a very poorly-looking one-liner.

To obtain a better solution, the following algorithm can be used:

1. Set $L \leftarrow \emptyset$ ($L$ is the list of connecting curves);

2. (a) Let $e_1 \in S$ be the edge with the highest priority;
   
   (b) Set $L \leftarrow L \cup \{e_1\}$ and $S \leftarrow S \setminus \{e_1\}$;
   
   (c) Set $P$ to an arbitrary endpoint of $e_1$;

3. (a) Find the edge $e \in S$, one of whose endpoints is the closest to $P$ among all endpoints of edges in $S$;
   
   (b) Set $L \leftarrow L \cup \{e\}$ and $S \leftarrow S \setminus \{e\}$;
   
   (c) Set $P$ to the other endpoint of $e$ (the one that was not the closest to $P$ in Step 3a);

4. If $S$ is not empty, go to Step 3a.

This algorithm finds a better solution than the trivial algorithm, but still it does not provide the solution that minimizes the total length of the connecting curves. The above greedy algorithm uses a local optimization scheme, selecting the closest neighbor to the current edge, but local optimization does not always solve the global optimization problem. This algorithm connects the first edges with short connectors and the latter edges with much longer connectors.

The next heuristic slightly improves over the above greedy algorithm. It accepts as an input parameter the maximum distance $D$ between two neighbors which we may connect, and executes the following procedure:

1. Find the edge $e_1 \in S$ with the minimum number of neighbors within distance $D$ from its endpoints;
2. If there is no such edge, halt the algorithm;

3. (a) Find the edge $e_2$, the closest neighbor of $e_1$ and connect between them by a curve $c$;

(b) Let $e$ be the united edge $(e_1, e_2$, and their connector $c$);

(c) Set $S \leftarrow (S \setminus \{e_1, e_2\}) \cup \{e\}$;

4. Go to Step 1.

The output of this heuristic is a set of connected edges which still need to be connected into a one-liner. This algorithm improves over the previous algorithm because it uses a threshold $D$ to limit the maximum length of the connecting curve, and because it first connects the most problematic edges—those that have only a small number of neighbors. However, it is still a greedy algorithm, and it leaves open the problem of connecting the smaller number of resulting long edges into a one-liner.

4.1.3 A TSP-based approach

All the algorithms mentioned in Section 4.1.2 usually fail because they try to address a global optimization problem using local optimizations.

In Section 4.1.1 we proved that the one-liner problem is NP-Complete. In order to determine a good solution of the global optimization problem we shall first reduce it to the Traveling Salesman Problem and then use a publicly-available approximate TSP solver to obtain an “optimal” one-liner.

The TSP is a well-known optimization problem stated as follows:

**Problem 2 TSP:** Given a finite set of sites and the cost of traveling between each pair of sites, find the cheapest cycle visiting all of them.
The one-liner problem differs slightly from the classic TSP: in addition to the set of $2n$ vertices and weights of edges between them, it is also constrained by a set of $n$ edges that must be used in the solution; see Figure 4.5. That is, the one-liner problem adds two constraints to the classic TSP:

- A given subset of the edges must be used by the solution;
- The solution path should alternate between the given edges and the added edges.

To satisfy those constraints we reduce the one-liner problem to the TSP problem as follows:

Given $n$ one-liner edges $\{e_1, \ldots, e_n\}$, we define $3n$ TSP vertices: $\{A_1, \ldots, A_n\}$, $\{B_1, \ldots, B_n\}$, and $\{C_1, \ldots, C_n\}$, where the $A$'s and $B$'s are the endpoints of the $n$ edges, and the $C$'s consist of one new vertex per edge; see Figure 4.6.
The distances between the vertices are defined as follows:

- $d(A_i, C_i) = 0$, where $1 \leq i \leq n$.
- $d(C_i, B_i) = 0$, where $1 \leq i \leq n$.
- $d(A_i, B_i) = \infty$,\(^{3}\) where $1 \leq i \leq n$.
- $d(A_i, A_j) = d_E(A_i, A_j)$,\(^{4}\) where $1 \leq i, j \leq n$ and $i \neq j$.
- $d(A_i, B_j) = d_E(A_i, B_j)$, where $1 \leq i, j \leq n$ and $i \neq j$.
- $d(B_i, B_j) = d_E(B_i, B_j)$, where $1 \leq i, j \leq n$ and $i \neq j$.
- $d(A_i, C_j) = \infty$, where $1 \leq i, j \leq n$ and $i \neq j$.

\(^{3}\)Implementation note: Instead of $\infty$ we can use a value greater than the sum of the weights of all the graph edges.

\(^{4}\) $d_E(X, Y)$ denotes the Euclidean distance between the points $X$ and $Y$ in the plane.
Figure 4.7: Distances between one-liner vertices

- \( d(B_i, C_j) = \infty \), where \( 1 \leq i, j \leq n \) and \( i \neq j \).

- \( d(C_i, C_j) = \infty \), where \( 1 \leq i, j \leq n \) and \( i \neq j \).

See Figure 4.7 for an illustration. The dotted lines denote the Euclidian distance between vertices.

Note that a vertex \( C_i \) is connected only to \( A_i \) and \( B_i \). Thus, the TSP solution must use both \( A_i C_i \) and \( C_i B_i \) because it has to pass through \( C_i \). This means that all the original one-liner edges will be included in the TSP solution, and that once the TSP solution reaches one of the endpoints of a one-liner edge (\( A_i \) or \( B_i \)), it will go straight ahead to the other endpoint passing through \( C_i \).

It is easy to verify that such a definition of the TSP problem forces its solution to satisfy the one-liner constraints mentioned above.

Although the TSP problem is NP-Hard [7, 17], many special cases of it can be solved efficiently. In particular, we can solve rather efficiently the TSP variant required for our one-liner problem.
4.1.4 A TSP solver

The LK program (written by D. Neto as part of [16]) implements the Lin-Kernighan heuristic for the symmetric traveling salesman problem (TSP). It mostly follows the design outlined by Johnson and McGeoch [10].

Lin and Kernighan [11] first described and motivated this local-search algorithm. They take a “tour-improvement” approach, starting from a given tour and attempting to modify it in order to obtain an improved tour, in the sense of lesser cost. Martin, Otto, and Felten [14, 15] proposed an improvement to the original Lin-Kernighan algorithm. The proposed strategy is to slightly perturb the Lin-Kernighan tour and then to reapply the algorithm. If this results in a better tour, they discard the old tour and proceed with the new one. Otherwise, they keep the old tour and proceed. This algorithm is usually referred to as Chained Lin-Kernighan. It offers a great performance boost over the original algorithm.

4.2 Determining the Connection Curves

This section describes the generation of the “image-driven” connecting curves, given the ordered set of image edges. The order of the image edges \( \{A_1B_1, ..., A_nB_n\} \) defines the sequence in which these edges will be connected together: \( \{B_nA_1, B_1A_2, ..., B_{n-1}A_n\} \).

The problem of finding the image-driven connectors is defined as:

**Problem 3 (Image-Driven Connectors):** Given two pixels \( A \) and \( B \), find an image-driven curve connecting them.

We still need to clarify the term *image-driven* connecting curve. Since the connectors are supposed to connect between endpoints of image edges, we use the following
criterion for the definition of a connector:

**Definition 2** A connector is image-driven if it is a continuous curve leaving a locally maximal average gradient. In addition, it must be simple, i.e. not have loops and should be located in the neighborhood of its endpoints.

Various neighborhoods can be used to bound the connectors. In this work we use a rectangular neighborhood. We regard all the pixels in the neighborhood as vertices in a graph, and apply a well-known Dijkstra algorithm [5] to find the shortest path between the two given vertices. Given the neighborhood corresponding to the two connector’s endpoints, we build the following graph:

- For each pixel in the neighborhood we add a vertex to the graph.

- For each pixel in the neighborhood we add eight edges connecting the pixel with its neighboring pixels.

- Since Dijkstra’s algorithm finds shortest paths (and not longest paths) in a graph with positive edge weights, we set the weight of every edge connecting a pixel to its neighbors to the absolute value of the difference between the gradient at the neighbor pixel to the maximum gradient in some predefined rectangular neighborhood of it. Actually, we square this value in order to further favor high gradients. In other words, if the gradient at a pixel $p$ is $g$ and the maximum gradient at some neighborhood of $p$ is $G$, then the weight of outgoing edges from $p$ is set to $(G - g)^2$.

- One of the connector’s endpoints is used as the target vertex.

This graph serves as the input for Dijkstra’s algorithm. The output of the algorithm is a set of shortest paths from all the graph vertices to the target vertex. This
CHAPTER 4. LINKING THE EDGES

is naturally much more than we want, since we only need to find a shortest path
between two given points.

There is one special case that should be handled separately. When the region
between two points consists of pixels with equal gradient magnitude (e.g. “white”
pixels), the natural connection of the two end-points is a straight segment, which is
not necessarily the path selected by Dijkstra’s algorithm. This is because there are
many paths connecting two points, all having the same “Manhattan distance.” A
simple workaround for this case is manually select the straight segment connecting
the two points.

We used an implementation of Dijkstra’s algorithm which is part of SPLIB de-
veloped by B. Cherkassky, A. Goldberg, and T. Radzik [3]. This solver uses a bucket
data structure proposed by R. Dial [4]. A typical instance graph in our setting con-
sists of 2300 vertices. The typical running time of the solver we use ranges between
3.4 to 4.5 milliseconds.
Chapter 5

Experimental Results

5.1 The Development Environment

We have developed a program that allows us to demonstrate and test the results of our proposed solution of the one-liner representation problem. The program runs under the Windows 2000 operating system. It uses the MFC library for the Graphical User Interface (GUI) and the OpenGL library for displaying the drawings. The whole system was implemented in C++ using the Microsoft Visual Studio development environment.

In addition we used two external software packages:

- The LK program: a TSP solver implemented by D. Neto for his doctoral research [16] at the Department of Computer Science of the University of Toronto. We used this program as a stand-alone application, providing it with an input file and parsing its output file to access the results of the program.

- An implementation of Dijkstra’s algorithm which is part of SPLIB developed by B. Cherkassky, A. Goldberg, and T. Radzik. We used this program to find
connectors between pairs of endpoints of the selected edges in the image. To optimize its running time we integrated this code in our software.

Both the LK and SPLIB programs were developed under the Linux operating system. For integrating them with our system we also needed to install the Sygwin (Linux emulation for Windows) environment.

We ran our experiments on a PIII 600Mhz personal computer with 256MB of SRAM.

5.2 Analysis of Experimental Results

In this section we demonstrate the application of our algorithm on various grey-scale and black/white images.

Several parameters control the edge-exploration algorithm:

- A neighborhood weight matrix used to estimate image derivatives. The default neighborhood is a 5x5 $N_2$ neighborhood, defined by the weights

$$\begin{pmatrix}
0 & 1 & 2 & 1 & 0 \\
1 & 4 & 8 & 4 & 1 \\
2 & 8 & 0 & 8 & 2 \\
1 & 4 & 8 & 4 & 1 \\
0 & 1 & 2 & 1 & 0 \\
\end{pmatrix}.$$

- The edge-exploration tuning parameter, whose default value was $\alpha = 0.1$.

- Two edge ranking thresholds were used:
  
  - The number of converging edges. (The default value is 20 edges.)
CHAPTER 5. EXPERIMENTAL RESULTS

- The average gradient along an edge. (The default value is 5.)

- The number of selected edges. This parameter can be set by the user instead of the using prioritization parameters mentioned above.

When the user does not specify the value of a parameter explicitly, our system uses the default value.

We next present four sequences of images, each sequence exhibiting partial results in the processing of a particular image. All images are well-known test cases in image processing: Figure 5.1 is an image of a lamp, and Figure 5.2 is the image of Paolina, Figure 5.3 is the image of Lena, Figure 5.4 is an image of the cat Felix. The sequences show the following stages of our algorithm:

- (a) The original image.

- (b-f) The results of the edge-exploration step using various prioritization parameters:
  
  - (b) When default parameter is used;
  
  - (c) When the \( k \) most important edges in the image are selected and
  
  - (d-f) Various combinations of the prioritization parameters (all the image edges are found by setting both prioritization parameters to zero).

- (g-i) The dependence of the edge-exploration algorithm on the value of \( \alpha \). Note that the “width” of the explored edge is proportional to its importance. This could also be a prioritization parameter, which can be a future extension of this work. However, a sharp change (increase) of \( \alpha \) affects the stability of the edge-exploration operation (see Section 3.3), thus it is difficult to use for the edge prioritization.
• The last images show examples of one-liner image representations.

The last image sequence, Figure 5.6, demonstrates the application of the edge-exploration algorithm to the rather trivial case of black/white images. In practice a small number of edges is sufficient for a one-liner representation of such images.
Several configurations of converging edges and average gradient

The effect of the tuning parameter $\alpha$

Figure 5.1: Lamp
CHAPTER 5. EXPERIMENTAL RESULTS

Several configurations of converging edges and average gradient

The effect of the tuning parameter $\alpha$

Figure 5.2: Paolina
CHAPTER 5. EXPERIMENTAL RESULTS

(a) Image
(b) Default parameters
(c) 83 selected edges
(d) All edges
(e) Average gradient ≥ 0, converging edges ≥ 40
(f) Average gradient ≥ 10, converging edges ≥ 40

Several configurations of converging edges and average gradient

(g) α = 0.5
(h) α = 0.3
(i) α = 0.05

The effect of the tuning parameter α

(j) One-liners

Figure 5.3: Lena
Several configurations of converging edges and average gradient

Figure 5.4: Felix
(a) Image
(b) Selected edges
(c) One-Liners

Vinere Vincitirce

(d) Image
(e) Selected edges
(f) One-Liners

Ebe

(g) Image
(h) Selected edges
(i) One-Liners

Amore e Psiche

(j) Image
(k) Selected edges
(l) One-Liners

Venere Italica

Figure 5.5: Canova’s Images
CHAPTER 5. EXPERIMENTAL RESULTS

Figure 5.6: Bullfights
### Edge-exploration statistics

<table>
<thead>
<tr>
<th></th>
<th>Lamp</th>
<th>Paolina</th>
<th>Lena</th>
<th>Felix</th>
<th>Dragon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions</td>
<td>256,256</td>
<td>512,480</td>
<td>512,512</td>
<td>240,346</td>
<td>195,244</td>
</tr>
<tr>
<td>Derivatives</td>
<td>0.841</td>
<td>3.135</td>
<td>3.545</td>
<td>1.072</td>
<td>0.601</td>
</tr>
<tr>
<td>estimation time (sec)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path exploration time</td>
<td>7.190</td>
<td>21.531</td>
<td>39.797</td>
<td>9.954</td>
<td>4.286</td>
</tr>
<tr>
<td># of explored paths</td>
<td>7,703</td>
<td>31,865</td>
<td>30,304</td>
<td>10,515</td>
<td>5,820</td>
</tr>
<tr>
<td>Connecting selected edges (first run)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of selected edges</td>
<td>122</td>
<td>574</td>
<td>242</td>
<td>182</td>
<td>124</td>
</tr>
<tr>
<td># of TSP vertices</td>
<td>183</td>
<td>861</td>
<td>363</td>
<td>273</td>
<td>186</td>
</tr>
<tr>
<td>TSP execution time (sec)</td>
<td>0.140</td>
<td>1.422</td>
<td>0.841</td>
<td>0.250</td>
<td>0.150</td>
</tr>
<tr>
<td>Average number of graph vertices</td>
<td>2,301</td>
<td>2,248</td>
<td>2,103</td>
<td>2,428</td>
<td>2,272</td>
</tr>
<tr>
<td>Dijkstra execution time</td>
<td>0.531</td>
<td>2.213</td>
<td>1.532</td>
<td>0.801</td>
<td>0.551</td>
</tr>
<tr>
<td>Connecting selected edges (second run)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of selected edges</td>
<td>156</td>
<td>1,394</td>
<td>824</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># of TSP vertices</td>
<td>234</td>
<td>2,091</td>
<td>1,236</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TSP execution time</td>
<td>0.181</td>
<td>7.751</td>
<td>2.754</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Average number of graph vertices</td>
<td>2,421</td>
<td>2,372</td>
<td>2,004</td>
<td>2,578</td>
<td>2,114</td>
</tr>
<tr>
<td>Dijkstra execution time</td>
<td>0.691</td>
<td>4.426</td>
<td>2.794</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.1: Measurements of the execution time

#### 5.2.1 Overall execution time

Table 5.1 summarizes the overall running times for the program. We took these measurements on five grey-scale images differing in their dimensions and complexities. For all the images we used the default value of $\alpha = 0.1$ as the edge-exploration parameter. The first section of the table shows some statistics of the edge-exploration step. The next two sections show the performances of the TSP’s and Dijkstra’s implementations. They differ in the number of edges selected for the one-liner. All the running times are given in seconds.
5.2.2 Improving the sorting algorithm

The edge-prioritization algorithm involves sorting of the explored edges according to priority parameters; see Section 3.2. The number of explored edges can be very large; see Table 5.1. Thus, the performance of the sorting algorithm affects significantly the overall running time. We first attempted to use the implementation of the quicksort algorithm given in the standard C library. However, it performed very poorly because the number of converging edges was distributed nonuniformly: Most of the explored edges had a very small number of converging edges.

To overcome this problem we did some preprocessing before running quicksort. The input to the algorithm is an unsorted array of explored edges. We moved edges with less than \( k \) converging edges to the end of the array, for \( k = 10 \). First we moved the edges with 0 converging edges, then we moved edges with 1 converging edge, etc. After each step we were left with less edges to sort. Table 5.2 shows the reduction of the number of edges-to-sort with each step of our preprocessing together with time consumed by this operation. We applied the standard quicksort algorithm on the remaining set of edges. This modification improved dramatically the time required for sorting; see Table 5.2.
## Table 5.2: Performance of the modified quick sort

<table>
<thead>
<tr>
<th></th>
<th>Paolina</th>
<th>Lena</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of edges</td>
<td>Time (milliseconds)</td>
</tr>
<tr>
<td># of converging edges = 1</td>
<td>31,865 ≤ 1</td>
<td>30,303 ≤ 1</td>
</tr>
<tr>
<td># of converging edges = 2</td>
<td>27,124 ≤ 1</td>
<td>25,726 ≤ 1</td>
</tr>
<tr>
<td># of converging edges = 5</td>
<td>12,041 10</td>
<td>11,246 10</td>
</tr>
<tr>
<td># of converging edges = 8</td>
<td>5,262 ≤ 1</td>
<td>4,949 ≤ 1</td>
</tr>
<tr>
<td>qsort</td>
<td>3,277 20</td>
<td>3,121 10</td>
</tr>
</tbody>
</table>
Chapter 6

Conclusions

6.1 Results of the Thesis

This work raised several interesting questions and a lot of interesting challenges. We developed a novel approach for edge detection and linking using “image flows” and implemented it using an edge-exploration algorithm. In the process of searching for a one-liner image representation we introduced an edge-exploration operator, which turned out to be very similar to an operator discovered by researchers from a team at Iowa State university. Our works were done independently and came out with the same novel approach to edge detection. The results which we demonstrated in this work prove that this is a very promising direction and might have many future extensions and uses.

We used an analytic method to evaluate the image derivatives using Taylor expansion. This approach allows us to estimate quite accurately and in a unified framework various partial derivatives of the image.
CHAPTER 6. CONCLUSIONS

Then we faced the interesting issue of connecting the selected edges. Allowing self-intersecting edges we reduced this problem to the problem of connecting optimally line segments. We proved that this problem is NP-Complete and found a reduction of it to an instance of the TSP problem (for which we obtained an approximate solution).

The last step which completed the one-liner finder was the use of Dijkstra’s shortest path algorithm. We used it to find connecting curves with maximum gradient along the curve.

In summary, we presented in this work various techniques which can be used in different areas of image processing: partial derivatives estimation using Taylor extension, an edge-exploration algorithm for edge detection and linking, and a variant of the Euclidian TSP to solve a segment-connecting problem. We needed all this to define a novel one-liner graphical image representation, inspired by drawings of Picasso and Colder.

As expected our one-liners do not compete well with the ones produced by artists, but to some extend they capture graphically some of the meaningful aspects of the input image.

6.2 Improvements and Future Enhancements

Our work deals with a new problem in image representation and uses innovative approaches to solve it. We identify a few possible future enhancements to the various stages of our algorithm. Here is a sample of these descriptions:

- The current implementation does not put any constraint on connecting the selected edges into a one-liner, except for limiting the length of the connecting curves. One of the possible enhancements is to find one-liners with a minimal
number of self-intersections. In that case the one-liner problem cannot be readily reduced to a variant of TSP and its NP-Completeness should be reviewed again.

- Another interesting issue is to determine whether it is possible to connect the given set of curves or segments into a Hamiltonian path (or cycle) without self-intersections. This problem can be considered as a subtask of the previous problem.

- Extending the edge-exploration operator and “image flow-based” edge linking model to consider the notion of inertia. It is not obvious that such a change will improve the edge exploration, but it is definitely worth research.

- Finding another approach for computing the connection curves between the selected image edges, e.g., by using an algorithm similar to our basic edge-exploration algorithm.
References


REFERENCES


REFERENCES


דרימות ייצוג גורפי וד- מקומי של תמרות

רדימ ממשקים
דרימות ריצוף גראפ היד-קרי של תמונות

חיבור על מחקר

לשם מדליית חלקי של הדרימות לקבליות הנوان
מגיסטר למדעי
במודימייקה שימושית

דרימ מכרווקס

הוגש לטנט הטכניון — מרכז טכנולוגי לישראל
היפה
 CPF
 2002
etri

שנת הפר

הכרת הוגדה

ברצוני להרודות לפניי אלפרד וורקנשטיין ו"ר על ברכה על ההנהיה
המשלח, התמיינה, וה↗バラ הכל שלבי התחמק.

אני מחל פרקון על התמיינה הבכפרין הגדולה בברחוביות
חוכן עניינים

1 הקציר באנגלית

3 מבוא 1
3 모르ית העבודה 1.1
6 מציאת הקשות והיבחר 1.2
7 רקע 1.2.1
9 הימשה שלם 1.2.2
12 שיטוט מווי הקשותות 1.3
14 קשר הקשותות הנבחרות להתפוגה ו- קווית 1.4

2 חישוב גנרטור הלקחים
19 חישוב הגנרטור והארפונים 2.1
21 חישוב הגנרטור ושנייה 2.2

3 מציאת הקשותות
24 רימיות ב למעונות 3.1
26 האלגוריתמים 3.2
28 מציאת הקשותות למעונות 3.2.1
30 יзор הקשותות למעונות 3.2.2
ורכזת ענינם

31 .................................................. לתוך הקשתות המועדון 3.2.3
32 .................................................. אפרטיםライフ של הקשתות 3.3
36 .................................................. אפרטים הוריסוניים 3.4
37 .................................................. מבוא הקשתות 3.5
40 .................................................. אנטרפורציה של נגרות 3.6

42 איסור הקשתות 4
44 .................................................. מבוא וסוקר הקשתות 4.1
46 .................................................. בניה ועל חキ שלמה ב- NP 4.1.1
48 .................................................. יישום הוריסוניים 4.1.2
50 .................................................. TSP יישום 4.1.3
54 .................................................. TSP 4.1.4
54 .................................................. היכרות עם מודל היבר 4.2

57 תרשיםי הנסכים 5
57 .................................................. סכנת הפיות 5.1
58 .................................................. נוכחות התרצאות 5.2
67 .................................................. זמני ביצוע מומלץ 5.2.1
68 .................................................. שיפור אלגוריתמים המわかり 5.2.2

70 מסקנה 6
70 .................................................. סיכום 6.1
71 .................................................. הצעות לשיפורים והרחבת 6.2

73 רישום מקורות

טכני
تكون

עבודת ומאגר השיטות的变化. בין עבידה וכדי של התמונת היה תהליך מסובך
ורב שבים, המושפע מגורמים שונים. העבידה黄石ון בין התמונת של פיקסא וקאלדר,蕃
морצאת התמונת, על עכדים, וחוזר אחרון לשלושה דודי צלילות עיבוד התמונת.
למרות השיווק של המומחה הוא מסוגל להטרום ייצורים של שונים וחל, ואג שғננים
ככ עבודה ומאגר השיטות בכדי של התמונת喱 אפ意见反馈י.
השלב האפשרי הוא ממצא הקושותות עלולה מבכסים וフア כת.
שהזוב הקור
לבשת את יוטיע עבורה ופיזחת שירותなし התמונת לתמונת והמענה מונף.
לפי החיבור, שירות דומה לשיטות פותחה במבקע עיון.
מיצא התמשכותיה היה ב胯ית החסנית והenticator הפרטורת תחת עיבוד התמונת.
ה AssemblyDescription של מצאות הקושותות בתמונת שונתה ברורה ממקורות מיחסיות החסנית והדוערה.
שיטות ממוסfindById תהליך דמי תנועה בשיטות הקוסמי. השידה מוגדר עיィ אופטורים מסיים, הקובע לכל
نكודה בתמונת את הכפייה או יש להתקדם על Mattis להצלחת להצלחת פעילות, שירותו ואנ אל הכנסת כאשר מكدימים של התמונת, עצפ區域 למטרה אחר כל הקושותות האפ fout.
נקודה בתמונת אופטורים稣י התמונת עד כי הקושות. חצץ את מככנת למקורה את
כל הקושותות האפ shouting. אחר התמונת של השיטה היא שלולית לימי אחר הקושותות בפי
שיטות בתמונת.مون אופטלים על稃ים הגאฟา עד כי מיצאת הקושותות. הקול

[6] Eua-Anant-ו Udpa
למי לבלוח חול הקובעה של הקשתות עם מספר absorptive לקלל יוגים שינון שמל החמצה,

המכססימעל מסמר שניהם של הקשתות המחודשות כבלי ציון.

את האופנות המשפעים על דיבור או מספר הקשתות משתנות או מספר ההנחות והחלקונות

של החמצה. האופנות או מספר מספרים שפתולים היפוך הקשתות מחודשות בגרעין

least-squares מסדר ראשוניות. השורות הנוגעות מרוכבות על פינה דרוי משולב בשתי

משקלות. ישויות של האופנות מחודשות להנחות על הנחות. הקשתות המחודשות

מזורות את הבסיס ליגון זה כי של החמצה.

השלב הבא בנין ייזום זה של החמצה התיאור הקשתות המחודשות "ע" עקומים.

שלב זה פותח ליצוא האסיט של צד אחד ואת הקשתות, וشروש העיקים הקשורים

את קרטון הקשתות הבסיס זה. הקשתות אליך צירקוק של החמצה שלב קוריא עביה

אנחון הזירה המפרקים.

טבע ומסתתת לเธอ לזרת את הקשתות עם שיא과장 הכלל של העיקים המפרקים

יתא מאייט. והמציעים כי ביצוע מצאות הסדר האופטימלי של החמצה יאת שלהמה

לך נבר כי מתונה פלטורות שלה. על מגרול לפתח קודרים שהל פיתורים

ואנ רשפת רודינק של התאונה לאת החמצה של העיקים יא商业地产 והرَ והנייעת

매ונטורייד יגור התו,abra מימים שינון של פירוטותמקופים. בקדוכה

וזה לא מתמשכים באת הקישה המאלה,SHARE פותח ע"ע

הליך השני של בל הקישור של הקשתות הבגרות ע颉ק במעמסת העיקים המפרקים.

אם את הקשתות בסדר הקבוע. על מגרול ליצוא עקומים ממסים תומנה המקצועית ואת

הקשתות הבגרות. או מספרים הסלורהים יזהי של

Dijkstra ליצואת המסרקל הקור.

ביוחם ביא אימות קצוף להמצאות בגרה. הגר אלי מובר והאולטראים בהן את המקדחים

ב Antar השיקוט לסביא של שיח קודוד הקצף של הקשתות הבגרות שיש לברחר.

אלא שים במשלמות את תחילת בייחודי ייגטר הח קיס של החמצה, והזג גר את האולטראים

יוס על הגרמאות רבוד של תומנה בפרק אפור התמונות שחרורumbling. יבודד זא איא מאורה

מספר דוממה של מיינצט ופרואן הרזת של השלבים והזגו של הדינה.