Lappeenrannan teknillinen yliopisto
Lappeenranta University of Technology

Leena Ikonen

DISTANCE TRANSFORMS ON GRAY-LEVEL SURFACES

Thesis for the degree of Doctor of Science (Technology) to be presented with due permission for public examination and criticism in the Auditorium 1382 at Lappeenranta University of Technology, Lappeenranta, Finland on the 15th of June, 2006, at noon.

Acta Universitatis
Lappeenrantaensis
240

Supervisor Professor Pekka Toivanen
Laboratory of Information Processing
Department of Information Technology
Lappeenranta University of Technology
Finland

Reviewers Docent Ingela Nyström
Centre for Image Analysis
Uppsala University
Sweden

Professor Annick Montanvert
Signal and Image Laboratory
University of Grenoble
France

Opponent Professor Gunilla Borgefors
Centre for Image Analysis
Swedish University of Agricultural Sciences
Sweden

ISBN 952-214-206-9
ISBN 952-214-207-7 (PDF)
ISSN 1456-4491

Lappeenrannan teknillinen yliopisto
Digipaino 2006

Abstract

Leena Ikonen
DISTANCE TRANSFORMS ON GRAY-LEVEL SURFACES
Lappeenranta, 2006
132 p.

Acta Universitatis Lappeenrantaensis 240
Diss. Lappeenranta University of Technology

ISBN 952-214-206-9
ISBN 952-214-207-7 (PDF)
ISSN 1456-4491

This thesis studies gray-level distance transforms, particularly the Distance Transform on Curved Space (DTOCS). The transform is produced by calculating distances on a gray-level surface. The DTOCS is improved by defining more accurate local distances, and developing a faster transformation algorithm. The Optimal DTOCS enhances the locally Euclidean Weighted DTOCS (WDTOCS) with local distance coefficients, which minimize the maximum error from the Euclidean distance in the image plane, and produce more accurate global distance values. Convergence properties of the traditional mask operation, or sequential local transformation, and the ordered propagation approach are analyzed, and compared to the new efficient priority pixel queue algorithm.

The Route DTOCS algorithm developed in this work can be used to find and visualize shortest routes between two points, or two point sets, along a varying height surface. In a digital image, there can be several paths sharing the same minimal length, and the Route DTOCS visualizes them all. A single optimal path can be extracted from the route set using a simple backtracking algorithm.

A new extension of the priority pixel queue algorithm produces the nearest neighbor transform, or Voronoi or Dirichlet tessellation, simultaneously with the distance map. The transformation divides the image into regions so that each pixel belongs to the region surrounding the reference point, which is nearest according to the distance definition used.

Applications and application ideas for the DTOCS and its extensions are presented, including obstacle avoidance, image compression and surface roughness evaluation.

Keywords: distance transforms, gray-level distance transforms, nearest neighbor transforms, shortest paths, minimal geodesics, image analysis

UDC 004.932.2 : 004.921

$\mathcal{G}(p)$	Gray-level value of pixel p
X and X^C	Calculation area and its complement
$N_4(p)$	Set of 4-connected edge neighbors of pixel p (also called face or square or horizontal/vertical neighbors)
$N_8(p)$	Set of all 8 neighbors of pixel p (4 edge neighbors, 4 vertex neighbors)
$N_8(p) \setminus N_4(p)$	Set of vertex neighbors of pixel p (also called diagonal neighbors)
$d(p_i, p_{i-1})$	Local distance between subsequent pixels p_i and p_{i-1} on a digital path
$\Delta(p)$	Local distance between mask pixel p and mask center pixel
$\mathcal{F}(x)$	Distance value of pixel x before distance transformation
$\mathcal{F}_1^*(x)$	Distance value of x after one pass of sequential local transformation
$\mathcal{F}^*(x)$	Final distance value of x
$\mathcal{F}_a^*(x)$	Distance value of x calculated from reference pixel a
$\mathcal{F}_A^*(x)$	Distance value of x calculated from reference pixel set A
$\mathcal{D}_R(x)$	Route distance image $\mathcal{F}_a^*(x) + \mathcal{F}_b^*(x)$ or $\mathcal{F}_A^*(x) + \mathcal{F}_B^*(x)$
$\mathcal{R}(a, b)$	Shortest route between pixels a and b
$\mathcal{R}(A, B)$	Shortest route between pixel sets A and B

1D	One dimensional
2D	Two dimensional
3D	Three dimensional
DT	Distance Transform
DTOCS	Distance Transform on Curved Space
FDT	Fuzzy Distance Transform
GRAYMAT	Gray-weighted Medial Axis Transform
MM	Mathematical Morphology
NNT	Nearest Neighbor Transform
PDT	Propagated Distance Transform
SKIZ	Skeleton by Influence Zones
SLT	Sequential Local Transformation
WDTOCS	Weighted Distance Transform on Curved Space

List of Publications

I. Ikonen, L., Kyrki, V., Toivanen, P., Kälviäinen, H., “Image Compression Using Derivative Information with Distance Transforms”, *Proceedings of the European Signal Processing Conference (EUSIPCO)*, Tampere, Finland, September 5-8, 2000, pages 957-960.

II. Ikonen, L., Toivanen, P., “Shortest Route on Height Map Using Gray-Level Distance Transforms”, *Proceedings of Discrete Geometry for Computer Imagery (DGCI), 11th International Conference*, Naples, Italy, November 19–21, 2003, pages 308–316.

III. Ikonen, L., Toivanen, P., “Shortest Routes Between Sets on Gray-Level Surfaces”, *Pattern Recognition and Image Analysis*, Vol. 15, No. 1, 2005, pages 195–198.

IV. Ikonen, L., Toivanen, P., “Shortest Routes on Varying Height Surfaces using Gray-Level Distance Transforms”, *Image and Vision Computing*, Vol. 23, No. 2, 2005, pages 133–141.

V. Ikonen, L., “Pixel Queue Algorithm for Geodesic Distance Transforms”, *Proceedings of Discrete Geometry for Computer Imagery (DGCI), 12th International Conference*, Poitiers, France, April 13-15, 2005, pages 228–239.

VI. Ikonen, L., Toivanen, P., “Distance and Nearest Neighbor Transforms of Gray-Level Surfaces using Priority Pixel Queue Algorithm” *Proceedings of Advanced Concepts for Intelligent Vision Systems (ACIVS), 7th International Conference*, Antwerp, Belgium, September 20-23, 2005, pages 308–315.

In this thesis, these publications are referred to as *Publication I*, *Publication II*, *Publication III*, *Publication IV*, *Publication V* and *Publication VI.*

1 Introduction **11**

2 Distance Transforms **15**
2.1 Basic Distance Measures . . . 15
2.2 Distance Propagation . . . 18
2.3 DTOCS and WDTOCS . . . 19
2.4 Properties of DTOCS Paths . . . 21
2.5 Geodesic Distance Transforms . . . 22
2.6 Gray-Level Distance Transforms . . . 23
2.7 Salience Distance Transforms . . . 24

3 DTOCS Modifications **27**
3.1 Chamfer Metrics . . . 27
3.2 Optimal DTOCS . . . 30
3.3 Chamfer Variations of the DTOCS . . . 33
3.4 DTOCS for Anisotropic Grids . . . 34

4 Transformation Algorithms **37**
4.1 Sequential and Parallel Local Transformations . . . 38
4.2 Recursive and Ordered Propagation . . . 40
4.3 Priority Pixel Queue Distance Transformation . . . 42
4.4 Nearest Neighbor Transformation . . . 47

5 Shortest Routes on Surfaces **49**
5.1 Route DTOCS . . . 49
5.2 Path Planning and Obstacle Avoidance . . . 52

6 Applications **55**
6.1 Image Compression . . . 55
6.2 Measuring Surface Roughness . . . 56

7 Discussion **59**

8 Errata **63**

Bibliography **65**

Publications **71**

Chapter I

Introduction

Distance is a fundamental concept in image analysis. The size and shape of an object can be used to detect or classify the object in a machine vision application, and distance information can be utilized in measuring both features. A distance transformation, as introduced already in 1966 by Rosenfeld and Pfaltz [47], is an operation, which determines the distance from every picture element, or pixel, to a given subset of pixels. The result, which is called a distance transform (DT), or a distance map, is an image, where the value of a pixel indicates its distance to the nearest feature or background pixel, depending on how the calculation area is defined. If distances are calculated from the background into the object, the maximum distance value provides an approximation of the size, or the smaller dimension of the object, like the width of an elongated object.

DTs belong to the fields of digital geometry and mathematical morphology. Digital geometry, or the geometry of the computer screen, is an application of discrete geometry. Digital images are inherently discrete, as they are represented using an evenly spaced grid of pixels. The resolution of the computer screen or the printer determines how fine shapes can be drawn, but regardless of how high the resolution is, Euclidean geometry dealing with continuous lines and shapes is not directly applicable.

Mathematical morphology (MM) is a theory for the analysis of spatial structures [62, p. 1]. The term morphology refers to the shape or the structure of an object, and is used in biology for the branch that deals with the form of living organisms and their parts. MM aims at analyzing the shape of objects using mathematical tools, like set theory, integral geometry, and lattice algebra. Originally, MM was defined for continuous Euclidean spaces, but image analysis techniques rely on the extension of MM to discrete spaces. The basic morphological operations, the erosion, which effectively peels off pixel layers from an object, and the dilation, which expands the object, are closely linked to DTs. In the case of binary images, the simplest DT is a sequence of erosions, where the number of erosions needed to remove a pixel from the object set, defines its distance value.

The aim of this thesis is to study DTs and transformation algorithms, with an emphasis on gray-level DTs. Particularly, the Distance Transform on Curved Space (DTOCS) is

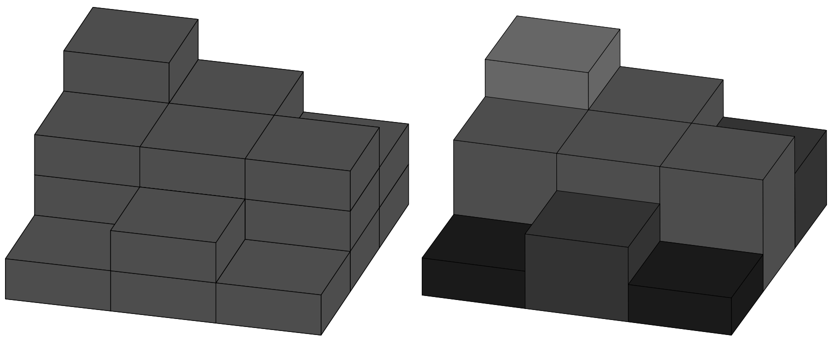


Figure 1.1: A 3D object (left) and a 3D visualization of the gray-level surface representation of the same shape (right).

studied, and developed further. The DTOCS, developed by Toivanen [68], [69], approximates distances along a gray-level surface, when gray-levels are understood as height values. The images are treated as height maps, where low gray-values (black and dark pixels) indicate low areas, and high gray-values (white and light pixels) indicate high areas. The distance values resemble geodesic distances defined in geography, that is, the distance between two points on a surface is the length of the shortest path along the varying height terrain between the points. The length of a digital path is approximated by summing the lengths of local steps between pixels along the path, and the step lengths are defined using two terms, one for the horizontal displacement and one for the height difference between the two pixels.

Gray-level surfaces, for which the DTOCS is defined, are slightly simpler than digital surfaces in general, as there can be only one height value for each coordinate in the image plane. Digital surfaces of 3D objects consist of all the object voxel faces, which are connected to the background in the 3D space, and these voxel faces are called surface pixels [46]. An example of a 3D object consisting of 22 voxels is shown to the left in Figure 1.1. Representing the object requires a volume of size $3 \times 3 \times 4$, and each voxel face visible from any direction belongs to the 3D surface of the object. For example, the single voxel on top has 5 visible faces, contributing to 5 surface pixels in the 3D surface of the object. The gray-level surface of the same 3D shape is visualized to the right in Figure 1.1. The gray-level image representing the surface consists of only 3×3 pixels, whose values correspond to the heights of the vertical boxes shown in the image. A gray-level surface can always be transformed into a 3D voxel image, but a 3D object can be represented as a gray-level surface only if it is a so called umbra, a solid set which extends unbroken indefinitely in the negative z-direction [63]. In practise, the object must extend unbroken until its flat base, so that the third dimension can be described with a single parameter, the height z of the object at coordinates (x, y). If any part of an object is not visible when viewing the object in the direction of the negative z-axis, the object is not an umbra, and the DTOCS approach can not be applied.

The DTOCS can be applied to any gray-level images, but true distances are approximated only if the image is a height map representing a 3D surface. In practise, range images, where each gray-value corresponds to the height z of the surface at location (x, y) in the image plane, must be used. Range images, which are also referred to as depth images, surface profiles, or 2 1/2 dimensional images, fulfill the conditions of an umbra, and can be acquired by special sensors, like sonars or laser scanners [71, pp. 16, 41]. Similar surface images can be constructed, for example, by interpolating altitude contours in a terrain map. The height information, or the distance from the sensor to the measured surface, is scaled and represented using gray-levels so that the difference between consecutive gray-levels corresponds to the horizontal distance between pixels. Alternatively, the input image can be scaled to make height differences and horizontal distances comparable.

Other gray-level DTs use the gray-value as the cost of traversing the pixel, so if the gray-level image is viewed as a height map, low lying paths are preferred, rather than shortest paths along the surface. The paths are not stored by the basic transformation algorithms. Only the distance values, that is, the lengths of the shortest paths are calculated. The Route DTOCS algorithm presented in this thesis finds and visualizes the shortest paths. Obvious applications are in terrain navigation, for example, in orienteering, or in planning a route for a new railroad or highway. Obstacle avoidance, with obstacles that can be crossed with a higher cost in addition to completely restricted areas, can also be implemented using the DTOCS.

In distance transformations, distance values are typically calculated by conveying local distances inside a mask to obtain the sum of the local distances between pixels along a digital path. In DTs of binary images, the local distances are defined based on distances between pixels in the flat image plane. The transforms are here often called binary DTs, even though the resulting distance maps are gray-level images. In the DTOCS and other gray-level DTs, the pixel values in the gray-level input image are used in the local distance definitions. In this work, the DTOCS was improved by redefining the local distance values to obtain more accurate approximations for global distances. Also, a new priority pixel queue algorithm for calculating the DTOCS was developed, and demonstrated to be more efficient than the traditional chamfering, or sequential local transformation (SLT). The main focus of this basic research is on analyzing and improving the DT itself, but practical image processing applications were also explored. Image compression was the first, and before this research the only application of the DTOCS. It is based on the idea that the DTOCS measures the amount of variation on the image surface, so that more data can be stored from locations with abrupt changes [68], [70]. Similar ideas can be utilized in evaluating surface roughness, and a DT based method for measuring surface roughness is outlined in this thesis.

The thesis is divided into eight chapters. Chapter 2 explains the basic concepts behind DTs, including the definition of the DTOCS, and an overview of other gray-level DTs. Chapter 3 describes chamfer distances used in binary DTs, and the new more accurate modifications of the DTOCS defined in this work. Chapter 4 reviews distance transformation algorithms, and introduces the new priority pixel queue algorithm for calculating the DTOCS, and its simple extension, the nearest neighbor transformation. In Chapter 5, the Route DTOCS algorithm for finding shortest routes along gray-level surfaces is described, and compared to similar approaches found in the literature. Applications utilizing the DTOCS, image compression and surface roughness evaluation, are described in Chapter 6. Chapter 7 contains conclusions and discussion, and Chapter 8 corrections to minor errors in the publications.

Summary of publications

In *Publication I*, the DTOCS is used in image compression. Compression results are improved by selecting control points from the derivative image instead of the original image. The author of this thesis was involved in developing the method based on Ville Kyrki's idea, performed the experiments, and took part in writing the article.

Publication II presents the Route DTOCS algorithm for finding shortest routes along a varying height surface. The routes found on the gray-level surface are like minimal geodesics in geography, that is, shortest routes along a terrain of varying height. The WDTOCS is demonstrated to provide more accurate global distances than the chessboard DTOCS, and the $\sqrt{2}$-DTOCS is presented as a hybrid DT combining locally Euclidean distances in the flat image plane with gray-level height differences. The author developed the algorithm and wrote the article.

In *Publication III*, the Route DTOCS algorithm is applied to the problem of finding routes between sets of points rather than single end-points. The article was first published in the Pattern Recognition and Image Analysis (PRIA) conference held in St. Petersburg in October 2004. An identical version included in the PRIA journal was selected for this thesis, as information about it can be found on the publisher's web page, whereas the conference is little known outside of Russia. The author carried out the research based on an idea provided by Professor Longin Jan Latecki, and wrote the article.

Publication IV is an extended journal version of *Publication II* and most of the material, except the $\sqrt{2}$-DTOCS, is repeated. In addition, the Route DTOCS algorithm is redefined with new more accurate versions of the DT, the 3-4-DTOCS, and the Optimal DTOCS. An algorithm for extracting a single path from the set of pixels on the route is also presented. Benefits of calculating distances along the gray-level surface rather than on the 3D representation of it are discussed. The method is also demonstrated to be useful in obstacle avoidance applications. The author developed and implemented the algorithms, and wrote the article.

Publication V presents a new efficient priority pixel queue algorithm for calculating the DTOCS. The pixel queue algorithm is compared to the SLT and the ordered propagation approach. At the same time, convergence properties of the SLT are analyzed more thoroughly than before. The individual work of the author was inspired by a reviewer of *Publication IV*, who suggested to abandon the chamfering.

In *Publication VI*, the priority pixel queue algorithm is extended to calculate the nearest neighbor transform (NNT) simultaneously with the distance map. The NNT according to DTOCS distances results in each pixel being assigned to the region surrounding the reference pixel, to which the distance along the varying height surface is the shortest. An application idea, where DTs are combined with the nearest neighbor transform to evaluate surface roughness, is briefly introduced. The author implemented the method and wrote the article.

Chapter II

Distance Transforms

A distance transformation is an operation, which transforms an image to a distance image, where the value of a pixel indicates its distance to the nearest reference pixel. The set of reference pixels is typically defined as the set of feature pixels, that is, the distances are calculated from the nearest feature pixel. Alternatively, distances can be calculated from the background into the object, and in that case the background pixels form the reference set, and the object pixels form the calculation area. DTs of binary images approximate straight line distances, but in a discrete grid, the definition of a straight line is not trivial, unless the line is exactly horizontal, vertical or diagonal. In practice, distances are approximated by summing local pixel steps, which form a digital path between the source and the destination point. The steps can be defined between immediate pixel neighbors, or using longer steps in a larger neighborhood. The distance value of a pixel is by definition the distance calculated along the shortest path, or one of several equally short paths, to the nearest reference pixel.

This chapter defines basic distances and neighborhoods used in DTs, including the DTOCS. Chamfer metrics, which produce improved distance approximations, are described in Chapter 3. The properties of metrics are reviewed to distinguish DTs based on distance functions, which are metrics, from pseudometric DTs. An overview of gray-level DTs is included, even though the basic and constrained DTs of binary images provide more of a foundation for this research than the gray-level DTs known before the DTOCS. The DTOCS approximates actual distances, whereas most gray-level distance transformations calculate minimal cost paths using gray-levels as cost values. This chapter also describes the mechanism of propagating local distances to produce the DT. Detailed descriptions of the distance transformation algorithms are provided in Chapter 4.

2.1 Basic Distance Measures

The basic distance measures, or distance functions, used in image processing are listed, for example, in [45, p. 209]. The Euclidean distance between points $P = (x, y)$ and

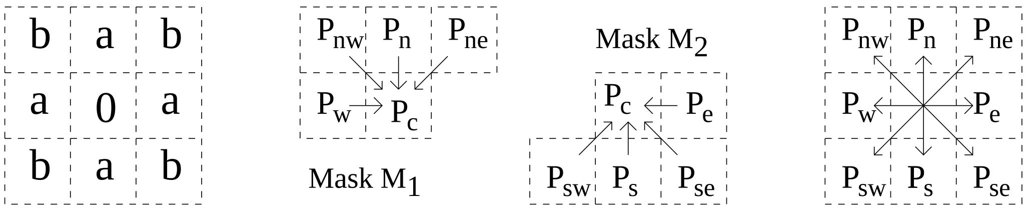


Figure 2.1: Pixel neighborhoods and masks used in DTs.

$Q = (u, v)$, where x and y (resp. u and v) are the coordinates of pixel P (resp. Q), is:

$$d_e(P, Q) = \sqrt{(x-u)^2 + (y-v)^2} \tag{2.1}$$

DTs are typically based on distance functions, which are easier to calculate than the Euclidean distance. The city block, or Manhattan distance, is defined as:

$$d_4(P, Q) = |x-u| + |y-v| \tag{2.2}$$

and the chessboard distance as:

$$d_8(P, Q) = max(|x-u|, |y-v|) \tag{2.3}$$

The distances d_e, d_8 and d_4 are globally defined in the sense that distances from one feature pixel can be calculated in a straightforward manner just from the coordinate difference of the feature pixel and all other pixels in the image. However, as the nearest feature pixel is not known beforehand, a brute force method would have to calculate the distance from each pixel in the calculation area to each feature pixel, and then select the minimum. The idea behind basic DTs is to propagate local distances defined within the small neighborhood of a pixel. The masks for distance propagation in Figure 2.1 and the propagation mechanism are described in Section 2.2. The most common 3×3 neighborhood, with local distances a and b from the center pixel, which has distance value 0 to itself, can be seen to the left in Figure 2.1. Here, the pixels with value a are called edge neighbors of the mask center pixel, and the pixels with value b are called vertex neighbors. In the publications, the edge neighbors are also called face or square neighbors, and the vertex neighbors are generally referred to as diagonal neighbors. The chessboard distance d_8, which is also called the 8-neighbor distance, is calculated by propagating local distances $a = b = 1$, and the city block distance function using $a = 1$ and $b = \infty$. The city block distance can also be called the 4-neighbor distance, as a pixel, in practice, has four neighbors. These n-neighbor distances are the fastest and the simplest to implement, but produce the worst approximations of the Euclidean distance [5]. The city block distance is an overestimate, and the chessboard distance an underestimate of the true Euclidean distance, unless the two points between which the distance is calculated differ only by one coordinate.

The basic distance definitions are visualized in Figure 2.2. There are several equally long digital paths of 7 steps according to the city block definition, some of which are

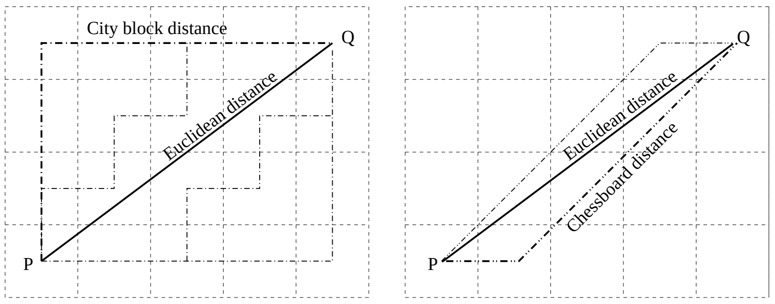
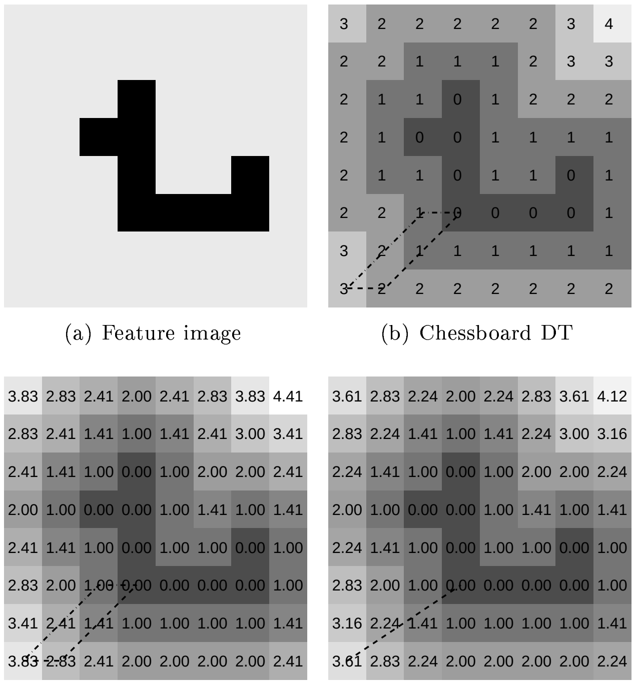


Figure 2.2: Example paths using the city block (left) and the chessboard (right) distance definition.

shown to the left of the figure. In this example, the chessboard distance of 4 steps is the same along two different paths, as shown to the right of the figure. The Euclidean distance can not be measured as the sum of local pixel steps like the other basic distance functions, as it corresponds to the length of a straight line, which generally does not coincide with a digital path. Using local distances $a = 1$ and $b = \sqrt{2}$ results in the so called locally Euclidean distance, which corresponds to the actual length of a digital path, as the exact length of the diagonal displacement from a pixel to its vertex neighbor is used. The shortest paths according to the locally Euclidean distance coincide with the paths of the chessboard distance in the example in Figure 2.2. Their lengths, however, are $1 + 3\sqrt{2} \approx 5.24$ instead of the corresponding chessboard distance 4. The locally Euclidean distance is one of the so called chamfer distances discussed in Section 3.1.

The distance functions define the distance values between single pixel pairs. In distance transforms, the distance value is determined for all image pixels. Feature or reference pixels, that is, pixels from which distances are calculated, have a distance value zero. The distance value of a pixel corresponds to its distance to the nearest feature pixel, or its distance to the nearest pixel in the background, depending on how the reference set is defined. The pixels, which do not belong to the reference set, that is, the pixels which obtain a distance value larger than zero, are said to belong to the calculation area X, and the reference pixels belong to the complement, X^C, of the calculation area. Figure 2.3 shows a binary feature image, and three of its different DTs. Each distance value corresponds to the distance from the nearest feature pixel according to the distance definition used. Chessboard distance values in Figure 2.3 (b) indicate the number of pixel steps required to reach the feature set in the 8-connected square grid. The two shortest paths from the bottom left corner point to the nearest feature pixel are shown in Figures 2.3 (b) and (c), and the corresponding Euclidean distance is visualized in Figure 2.3 (d). The chessboard and the piecewise Euclidean distances are approximations of the Euclidean distance shown with rounded values in Figure 2.3 (d). None of the approximations are very good, but nevertheless, the DTOCS and its locally Euclidean modification, the WDTOCS, are based on them. More accurate approximations are achieved by using so called chamfer or quasi-Euclidean distances described in Section 3.1.



(a) Feature image (b) Chessboard DT

(c) Piecewise Euclidean DT (d) Euclidean DT

Figure 2.3: Distance transforms of a binary feature image.

2.2 Distance Propagation

The best known approach to propagating local distances to produce a DT is the SLT algorithm presented by Rosenfeld and Pfaltz [47], which calculates the distance value of a pixel as the number of steps from the pixel to the nearest feature pixel in a binary image. The algorithm is described for the city block DT, where a pixel effectively has only four neighbors, the edge neighbors, but the idea is presented also for the chessboard DT, where the steps to the vertex neighbors are included. In the following, a modification of the SLT, which can handle any local distances a and b is presented. In the SLT, the pixel neighborhood is divided into two masks as shown in the middle of Figure 2.1, and the image is scanned sliding one mask at a time row-wise across the image. The first computation pass proceeds using the mask $M_1 = \{p_{nw}, p_n, p_{ne}, p_w\}$ from the top left corner of the image, substituting the value of center pixel p_c, $\mathcal{F}(p_c)$, with the distance value:

$$\mathcal{F}_1^*(p_c) = \min[\mathcal{F}(p_c), \min_{p \in M_1} (\Delta(p) + \mathcal{F}_1^*(p))] \tag{2.4}$$

The mask configuration ensures that all the pixels in mask M_1 are already processed, when calculating the distance value for the center pixel p_c. The distance value propagates to pixel p_c from one of the mask pixels. The local distance $\Delta(p)$ between a mask pixel p and the center pixel p_c is determined according to the distance definition used. For binary DTs the local distance is a if p and p_c are edge neighbors, and b if they are vertex neighbors. The reverse pass, which proceeds row-wise from the bottom right corner of the image with mask $M_2 = \{p_e, p_{sw}, p_s, p_{se}\}$, replaces the distance value $\mathcal{F}_1^*(p_c)$ calculated by the forward pass with:

$$\mathcal{F}^*(p_c) = \min[\mathcal{F}_1^*(p_c), \min_{p \in M_2}(\Delta(p) + \mathcal{F}^*(p))] \tag{2.5}$$

The original SLT algorithm in [47] transforms a binary image to a distance image, where each pixel with value 1 in the original image gets a value corresponding to its distance from the nearest pixel with value 0 in the original image. The pixels with value 0 remain unchanged, as their distance to themselves is 0. The first pass assigns a value larger than any distance value that can appear in the image, for example, $\max(m, n)$ for the chessboard distance in an image of size mn, to the pixel in the top left corner, where the scanning starts. The large value propagates until the first reference pixel with distance value 0 is reached, and correct distance values start propagating. In the reverse pass, the large values are replaced by correct distance values. The modification presented above, which is used for the DTOCS, needs an input $\mathcal{F}$, where the reference pixels are set to 0 and the pixels in the calculation area to a value, which is larger than any distance value that can appear.

The original SLT algorithm, in which distances are calculated only based on the mask pixels, not including the pixel p_c currently being processed, is based on the assumption that the two passes produce the final DT. The modification presented above is applicable for more complex DTs, where the two passes need to be iterated several times, using the result $\mathcal{F}^*$ of the reverse pass as the input $\mathcal{F}$ for the next forward pass. The minimum operation in Equations 2.4 or 2.5 selects the distance value of the current pixel, if it is smaller than the smallest distance calculated from the neighbors, for example, if the current pixel p_c is a reference pixel having value 0. The distance values remain unchanged, until the mask reaches the first reference pixel.

The SLT is only one of several distance transformation algorithms, but the basic idea of propagating local distances within a small neighborhood is applied also in the other approaches discussed in Chapter 4. In Equations 2.4 and 2.5, the distance value of the mask center pixel is determined by adding local distances to distance values of the other mask pixels, that is, the distance propagates from one of the mask pixels to the center pixel. Alternatively, the distance values can be propagated forward from the center pixel, as visualized to the right in Figure 2.1. These alternative approaches can be categorized as acquiring and deriving DTs [56]. In SLTs, distances are typically propagated in the acquiring fashion, whereas the deriving approach is used in the priority pixel queue algorithm described in Section 4.3.

2.3 DTOCS and WDTOCS

The DTOCS and the WDTOCS are calculated using the distance propagation mechanism presented in Section 2.2. First, the image $\mathcal{F}$ used as input to Equation 2.4, is initialized so that reference pixels get value 0 and pixels in the calculation area get value *max* (the maximal representative number of memory). In the DTOCS, any pixels can be selected as reference pixels, that is, distances can be calculated from single points on

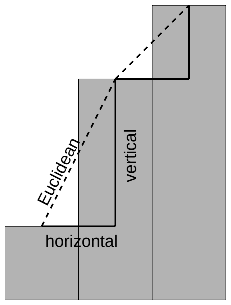


Figure 2.4: Local distance definition for the DTOCS (solid line) and the WDTOCS (dashed line).

the surface, or from areas, and the calculation area can be connected or disconnected. The propagated local distance values contain a horizontal component for the shift in the image plane, and a vertical component defined by the gray-level difference between the pixels. The difference corresponds to the change in altitude between the pixels. Figure 2.4 shows an example of how local distances are calculated on a surface segment containing three pixels. The height of each bar indicates the gray-value of the corresponding pixel. The DTOCS local distance is defined as the sum of the horizontal and the vertical component of the displacement along the digital surface. The WDTOCS, originally called the Euclidean DTOCS (EDTOCS) [67], uses the Euclidean distance between the centers of the neighbor pixels calculated with Pythagoras' theorem from the horizontal and the vertical component. The horizontal component is always one in the DTOCS based on the chessboard distance, whereas the locally Euclidean horizontal distances, $a = 1$ and $b = \sqrt{2}$, are used in the WDTOCS. The definitions for the local distance values are:

$$\text{DTOCS:} \quad d(p_i, p_{i-1}) = \quad |\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})| + 1 \tag{2.6}$$

$$\text{WDTOCS:} \quad d(p_i, p_{i-1}) = \quad \begin{cases} \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + 1}\,, & p_{i-1} \in N_4(p_i) \\ \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + 2}\,, & p_{i-1} \in N_8(p_i) \setminus N_4(p_i) \end{cases} \tag{2.7}$$

where $\mathcal{G}(p)$ is the gray-value of pixel p, and p_i and p_{i-1} are subsequent pixels on a digital path. The DTOCS is a metric, as it fulfills the following three criteria [45, p. 209]:

1. The distance between two points P and Q is positive definite, that is, $d(P,Q) \geq 0$ and the equality holds only if $P = Q$.
2. The distance is symmetric, that is, $d(P,Q) = d(Q,P)$.
3. The distance fulfills the triangle inequality $d(P,R) \leq d(P,Q) + d(Q,R)$.

Positive definiteness is guaranteed by the fact that the local distance between two pixels is strictly positive, as it contains a strictly positive term for the horizontal displacement, and a non-negative term for the height difference between the pixels (see Equations 2.6

and 2.7). Symmetricity also applies, as each local distance is symmetric, and the sum of the local distances along a path is symmetric. The proof of the triangle inequality $d(P, R) \leq d(P, Q) + d(Q, R)$ follows directly from the definition of a DT. If $d(P, Q) + d(Q, R) < d(P, R)$, there must exist a path from pixel P via pixel Q to pixel R, and the length of this path is less than $d(P, R)$. This contradicts the definition of the DTOCS, where the distance $d(P, R)$ is the length of the shortest path from P to R.

The chessboard and city block distances, and the DTs based on them, are also metrics. The Euclidean DT, the locally Euclidean DT, and the WDTOCS are by definition metrics, but the limited number of bits available for representing the floating point distance values may lead to violations of the metrics criteria. Adding several square root terms in calculating the distance from pixel P to pixel Q and the distance from Q to P may result in values $d(P, Q)$ and $d(Q, P)$, which differ in their least significant bits, leading to a violation of the symmetricity criterion. In an unlucky case, the difference may result in two values, which do not round up to the same integer (for example, 4.999... and 5.000...). The rounded Euclidean distance, which is calculated using only one square root operation from integer coordinates, is a metric according to Soille [62, p. 46].

2.4 Properties of DTOCS Paths

In a discrete image, two points can be connected with numerous digital paths, and there can be several paths sharing the same minimal length. The distance transformation calculates the length of one of the shortest paths according to the distance definition used. DTOCS paths are typically curved, as the straight line between two points can be blocked with an obstacle consisting of higher or lower gray-values than the surrounding area. Consequently, the concept of DT regularity is not applicable in the DTOCS setting, as DT regularity is defined by Borgefors as follows:

"Consider two pixels that can be connected by a straight line, i.e., by using only one type of step. If that line defines the distance between the pixels, i.e., is a minimal path, then the resulting DT is semi-regular. If there are no other minimal paths, then the DT is regular." [9]

The unconstrained DT of a binary image is regular if the local distance to edge neighbors, a, and the local distance to vertex neighbors, b, fulfill the inequality $a < b < 2a$, and semi-regular if at least one of the inequalities is replaced by an equality [9]. The WDTOCS weights $a = 1$ and $b = \sqrt{2}$ fulfill the regularity criterion in the horizontal plane, but due to the height component, the WDTOCS is not regular. The DTOCS with chessboard distances is only semi-regular even in the flat image plane without height differences. As the length of steps between vertex neighbors is underestimated, the resulting paths may not seem intuitively optimal. This is demonstrated in Figure 2.5, extracted from *Publication II*. The straight line between points A and B is a minimal path but not the only one according to the chessboard distance definition. The path via point 'x' is equally short as the straight path, that is, both paths consist of 10 pixel steps. Defining the local distance values so that $b < a$ would violate the semi-regularity criterion, as the path via 'x' would become shorter than the straight path. Regularity properties, which are an important factor in developing chamfer distances producing more accurate DTs, are discussed in mathematical detail in [29].

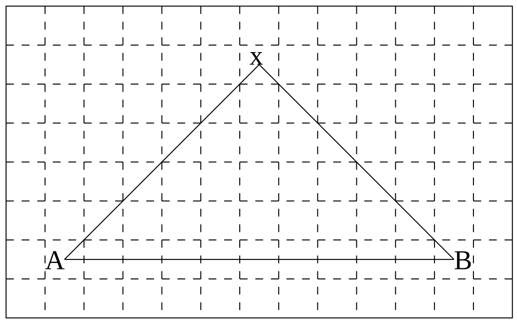


Figure 2.5: Two of several shortest paths from pixel A to pixel B on a flat image surface according to the chessboard distance definition.

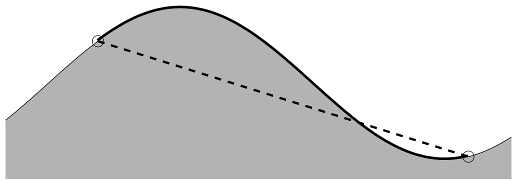


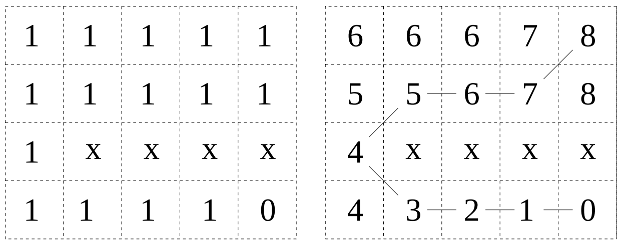
Figure 2.6: The Euclidean distance (dotted line) and the geodesic distance (thick curve) between two points on a surface of varying height.

Piper and Granum [40] state that a DT can be produced in two or at most four passes of the SLT, if the transformed domain is convex. Any curvature in the gray-level surface makes the surface non-convex, as a domain is convex if and only if any two points in the domain can be joined by a digital straight line included in the domain. Figure 2.6 demonstrates how a straight line joining two surface points only crosses the surface domain. The straight dotted line corresponds to the Euclidean distance between the two points, whereas the thick curve represents the actual distance along the surface, which the DTOCS tries to approximate.

2.5 Geodesic Distance Transforms

The basic DTs of binary images calculate the distance to the nearest feature pixel for all pixels in the image, including the feature pixels themselves, as they maintain their distance value 0 in the transformation. In constrained or geodesic DTs, some pixels are restricted. Distances to constraint pixels are not calculated, and paths to other pixels can not cross the restricted image areas. In other words, the paths linking pixels are constrained to remain within a subset of the image plane [62, p. 219]. For example, distances can be calculated inside non-convex objects by defining the background as the constrained area [40]. The DTOCS fits well into the definition of a geodesic DT. The binary image, which is used to define the calculation area, is transformed into the distance image, and the gray-level image only affects the result without being modified. Using two input images, transforming one image, and restricting the result with another image, the geodesic mask, is the approach taken in geodesic transformations. For example, in planning the path of a robot, the geodesic mask corresponds to the regions where the robot can move [62, p. 219]. In the DTOCS, the original image can be thought of as the geodesic mask restricting paths to follow points at certain heights in the 3D scene represented as a height map, and the binary image $\mathcal{F}$ used for selecting the reference pixels is the one being transformed. Geodesic versions of morphological operations, like dilation, erosion and skeletonization, can be found in [32], but this work is restricted to geodesic distances.

The calculation area in geodesic or constrained DTs is non-convex, and paths become curved. Figure 2.7 shows a small example of a constrained DT. To the left, the constraint pixels are marked with 'x', the only feature pixel has value 0, and pixels in the calculation area have value 1. The constrained DT based on the chessboard metric can be seen to the right. The path, along which distances propagate from the feature pixel to other pixels in the image, is forced to go around the constraint pixels. An example of a curved shortest path is shown on the transform image to the right. Constrained DTs are useful in obstacle avoidance and path planning problems. The DTOCS can easily be used as a constrained DT for flexible obstacle avoidance, as explained in Section 5.2.



1	1	1	1	1
1	1	1	1	1
1	x	x	x	x
1	1	1	1	0

6	6	6	7	8
5	5	6	7	8
4	x	x	x	x
4	3	2	1	0

Figure 2.7: A constrained domain (left), and its DT (right)

2.6 Gray-Level Distance Transforms

Gray-level DTs typically use the gray-value as the cost of traversing the pixel. The path lengths in such transforms are sums, or weighted sums, of gray-values along the path. Rutovitz proposed a transformation, where each pixel gets a distance value corresponding to the sum of gray-values on the path to the nearest zero-valued pixel [50]. The original image is thought of as a surface, where the gray-level corresponds to height, but low-lying paths are preferred, as opposed to shortest paths along the surface found by the DTOCS. The same applies to the geodesic time DT by Soille [61]. The Gray-weighted Medial Axis Transform (GRAYMAT) by Levi and Montanari [34] uses weighting factors, which make diagonal steps more costly than straight steps, but still paths with low gray-level sums are found.

The Fuzzy DT (FDT) is very similar to gray-level DTs calculating minimal cost paths, even though it is defined for so called fuzzy images. In a fuzzy image, pixels have values from zero to one, describing the membership value of each pixel. A pixel with value one belongs to the object, and a pixel with value zero belongs to the background. Intermediate values result from uncertainties, like data inaccuracies or limited image resolution [52]. For example, the perimeter or the area of an object can be estimated based on fuzzy borders introduced in segmentation of low resolution images [60]. In the FDT, the membership values are used as local distance weights, so in practice, scaling the fuzzy image into a gray-level image, and using a gray-level DT, like the GRAYMAT or the geodesic time, would produce the same results apart from quantization errors introduced by the scaling.

The topographical distance based on the concept of connection cost, and the differential distance derived from the concept of deviation cost, consider the gray-level image as a height map similarly as the DTOCS. The connection cost corresponds to the altitude of the lowest neck of the mountain linking two valleys, and the deviation cost corresponds to the cost to be paid in order to take a path deviating from the path of greatest slope [41]. The distance functions can be useful in analyzing the topography of gray-level images, but do not approximate true distances.

Gray-level distances are generally not metrics, only pseudometrics, as the condition of positive definiteness is violated. The GRAYMAT and the geodesic time distance can be zero between points P and Q even if $P \neq Q$, if the points are connected by a path containing only pixels with value zero. For example, the distance value along the path shown to the left in Figure 2.8 is 0 according to most gray-level DTs. Positive definite distances can be achieved by requiring strictly positive values in the original image [61]. Zero valued pixels can be eliminated by adding value one to all image pixels, but the resulting shortest paths are generally not the same. In Figure 2.8, the shortest path consisting of several pixels with value zero is replaced with a path containing fewer pixels, when all gray-values are increased by one. Gray-level DTs defined using the sums of gray-values are linear with respect to scaling, and find the same shortest paths regardless of the scaling of the input. However, as demonstrated here, they are not shift invariant in the sense that the shortest paths would be the same after adding a constant to the gray-values of the original image. The increase in length of each digital path is proportional to the number of pixels on the path, so shortest paths, which contain many pixels with low values, can be replaced with paths containing fewer pixels with higher gray-values. The DTOCS, however, is invariant with respect to the addition of a constant, but not linear with respect to scaling. The DTOCS can be scaled to produce the same results for images with different gray-level scaling, provided that the scale is known (see Section 3.4). The DTOCS also fulfills all three criteria for metrics unlike most gray-level DTs, as shown in Section 2.3.

2.7 Salience Distance Transforms

Many DT applications are based on calculating distances from edges extracted from a gray-level image. In template matching, the template or model can be superimposed on a distance transformed edge image. The lower the average of the distance values lying underneath the model is, the closer is the match. The hierarchical chamfer matching

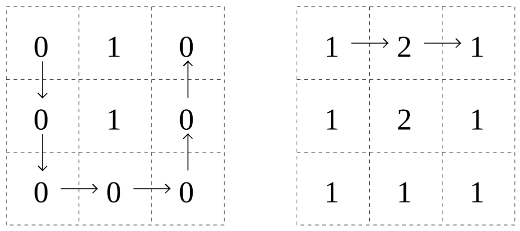


Figure 2.8: Examples of minimal cost paths, where the gray-value corresponds to the cost of traversing the pixel.

algorithm presented in [7] is based on this idea. Another example of utilizing DTs of edge images is a robot application, where metal sheets are picked from a pile. The best locations for the magnetic grip are selected based on the fact that the largest distance values appear at the centers of sheets, which are not covered by other sheets [22].

Applying DTs to edge images is problematic, as most edge detectors produce errors, both false detections, and missed real edges. Salience DTs are based on introducing strength values describing the importance, or salience, of edges. The underlying assumption is that false detections produce weaker edges, and the most interesting edges get strong salience values. Various approaches to incorporating the edge strength are presented and compared in [49]. The edge weight can be propagated together with the distance values during the transformation, so that final distance values can be normalized with the corresponding weight. Pixels close to important edges get low values in the salience DT. The idea of propagating additional information together with the distance values is utilized in the nearest neighbor transformation presented in Section 4.4.

Alternatively, the edge pixels can be initialized with the negative of the weight magnitude, so that smaller distances propagate from strong edges. Consequently, weak edges located near strong edges have little effect on the resulting DT [49]. Similar results can be obtained using gray-level DTs on an image, where strong edges are indicated with low gray-values, and weak edges with higher gray-values, resulting in smaller gray-level distances propagating from the strong edges. The FDT is a way to achieve the same in a more intuitive way, as the likelihood of a pixel belonging to an edge can be encoded directly as the value of the pixel. The DTOCS can also be applied similarly by using an input image, where the gray-level difference between weak edges and the background is greater than the difference between strong edges and the background. Distances are then more likely to propagate from the strong edges, decreasing the effect of the weaker edges on the resulting DT.

This chapter reviewed the basic ideas and notions behind DTs, including the DTOCS, which is used and developed further in this work. Modifications of the DTOCS, which produce improved distance approximations, are discussed in the next chapter. The distance propagation mechanism for calculating the distances was also introduced here, but transformation algorithms are discussed in more detail in Chapter 4, which presents the new priority pixel queue algorithm developed in this work. The DTOCS can be viewed as an extension of the chessboard DT for binary images, as it calculates distances in a similar way. Issues concerning gray-level DTs also affect this work significantly, as the local distances of varying magnitude result in curved paths in all gray-level DTs. Even though the shortest paths found by the DTOCS are inherently different from the minimal cost paths found by other gray-level DTs, there are many similarities. In some applications, like the obstacle avoidance problems discussed in Section 5.2, either approach can be used, if the gray-level image representing the scene with obstacles is created appropriately. The DTOCS can be used to approximate true distances along a surface represented as a range image, but the applications presented in Chapter 6 rely on the DTOCS as a measure of the amount of gray-level variation in images, which are not necessarily height maps.

Chapter III

DTOCS Modifications

In this chapter, the distance approximations produced by the DTOCS and the WDTOCS are improved by redefining the local distances using so called chamfer metrics. Chamfer metrics produce more accurate approximations of the Euclidean DT than the traditional chessboard and city block metrics, and are beneficial also when introduced to the DTOCS setting. The Optimal DTOCS developed in this work is the best approximation found so far for distances along a gray-level surface. However, slightly better approximations can be obtained by transforming the gray-level image into a 3D umbra, and calculating distances on its top surface using 3D chamfer distances. Other DTOCS modifications based on chamfer distances are also discussed. Furthermore, a generalization of the DTOCS and the WDTOCS to anisotropic grids is presented, and issues concerning scaling of the gray-level surfaces and the distances calculated along them are discussed.

3.1 Chamfer Metrics

In chamfer DTs the local distance a between pixels that are edge neighbors, and the local distance b between vertex neighbors, can be any real numbers [5]. The locally Euclidean distance introduced in Section 2.1 is a chamfer distance, and is sometimes called the chamfer-Euclidean or the quasi-Euclidean distance. Neighborhoods larger than 3×3 can also be used. Figure 3.1 shows the 5×5 neighborhood, which includes the so called knight's move with the local distance value c. The knight's move is not meaningful for the chessboard or the city block metrics, but can be defined to improve the distance approximations obtained using chamfer distances.

Borgefors [6] derived local distance values, which minimize the maximum difference from the Euclidean distance. The minimization is based on the fact that in a square grid, there always exists a minimal path consisting of at most two straight line segments between two points [37], provided that the minimal path is based on a regular DT, where a diagonal step is longer than a horizontal or vertical step, but shorter than two such steps ($a < b < 2a$). The Euclidean distance, which is the length of the straight line between two points, is approximated using the sum of horizontal or vertical steps, which form one

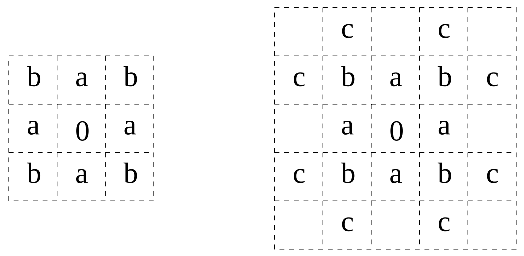


Figure 3.1: The 3×3 and the 5×5 neighborhood in chamfer distance calculations.

line segment, and diagonal steps, which form the other line segment. The error compared to the Euclidean distance varies depending on the slope of the straight line between the pixels, that is, on the relative number of diagonal steps on the minimal paths. The geometrical basis for the optimization is visualized in Figure 3.2. The distance from the origin to a pixel on the vertical line $x = M$ is formulated as the sum of the diagonal and the horizontal step lengths. An example of such a path is indicated using a solid line. The difference between the length of the path and the Euclidean distance indicated with a dotted line is:

$$Diff(y) = y(b-a) + Ma - \sqrt{M^2 + y^2}\,,\ 0 \leq y \leq M \tag{3.1}$$

The absolute maximum of $Diff(y)$ is minimized, resulting in local distances [6]:

$$\begin{aligned} a_{opt} &= (\sqrt{2\sqrt{2}-2}+1)/2 &&\approx 0.95509 \\ b_{opt} &= \sqrt{2} + (\sqrt{2\sqrt{2}-2}-1)/2 &&\approx 1.36930 \end{aligned} \tag{3.2}$$

The optimal coefficients, which equal the optimal parameter values for length estimates of type $l = a\Delta x + (b-a)\Delta y$ [3], are used in the Optimal DTOCS developed in this work (Section 3.2). The local distance coefficients minimizing the maximum error from the Euclidean distance can be derived similarly for the 5×5 neighborhood [8]. It should be noted that the coefficients a_{opt} and b_{opt} are not identical to the ones used in the 3×3 neighborhood. The values used in the 5×5 neighborhood are $a_{opt} \approx 0.986$, $b_{opt} \approx 1.414$ and $c_{opt} \approx 2.208$. The value for b_{opt} can be selected from a small range producing the same maximum error, and the locally Euclidean distance $\sqrt{2}$ included in the range is used in a 5×5 modification of the Optimal DTOCS presented in Section 3.2.

Verwer [76] minimized the maximum error based on distances to a Euclidean circle, claiming that Borgefors' approach of using distances between the origin and points on a vertical line ($x = M$) causes the relative error in the diagonal direction to dominate the optimization compared to the same relative error in another direction. Slightly different coefficients minimizing the average difference from the Euclidean distance can be found in [82]. In [65] effective errors are minimized by checking all integer weights a from 2 to 255 for the straight neighbor distance. Other chamfer distances in the large masks are obtained by rounding the Euclidean distance scaled with a to the nearest integer. Alternating city block, d_4, and chessboard distances, d_8, in the octagonal distance transformation [48] results in a better approximation of the Euclidean distance. Thiel and

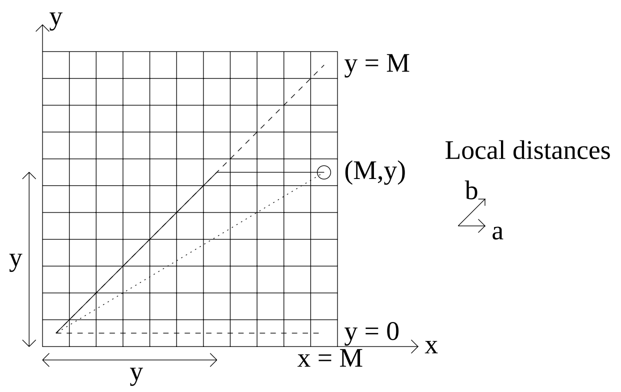


Figure 3.2: The geometry of the chamfer DT in the 3×3 neighborhood [6].

Montanvert [65] formulate the chamfer distance $d_C = d_4 dy + d_8(dx - dy)$ as a linear combination of chessboard and city block distances, and state that it induces a distance only if the elementary displacements dx and dy fulfill the inequalities $dx \geq dy \geq 0$ and $dx > 0$ within one octant of the neighborhood, called the influence cone. In general, the differences between chamfer distances optimized based on different criteria are very small, and all produce reasonable approximations for the Euclidean distance. Distances can also be approximated using circular propagation, which produces discrete circles, if the propagation starts from a single point [30].

The quasi-Euclidean distance calculations by Montanari [37] already contain the idea of using larger neighborhoods than the 8 immediate neighbors of a pixel. Figure 3.3 shows the steps available in one octant of a 7×7 neighborhood. The possible slopes of the pixel displacements follow the so called Farey sequence, that is, the ordered sequence of all rational numbers between 0 and 1 with denominators less than or equal to n. In the example where $n = 3$, the sequence is: 0, $\frac{1}{3}$, $\frac{1}{2}$, $\frac{2}{3}$ and 1. The distances in the 5×5 neighborhood shown in Figure 3.1 can be defined using the Farey sequence for $n = 2$, as the possible slopes are 0 (local distance a), $\frac{1}{2}$ (local distance c), and 1 (local distance b). The Euclidean lengths of the displacements, or approximations of them, can be used as step lengths. Using larger neighborhoods and longer Farey sequences produce closer approximations of the Euclidean distance values. Setting $n = m - 1$ in an image of size $m \times m$ would produce Euclidean distances, but that would lead to brute force calculation of distances between all feature and all non-feature pixels, which is computationally expensive. In earlier DT research the direct approach was denounced impossible, but the recently presented Fast Exact Euclidean Distance Transform [53], and its weighted modification [72] make the brute force method feasible by restricting the number of pixels taken into consideration. However, a direct approach based on the coordinates of the path end-points does not exist for the DTOCS, similarly as it is impossible to determine the driving distance between two cities based only on their earth coordinates. The distances need to be calculated along the curved paths, which follow the varying height surface.

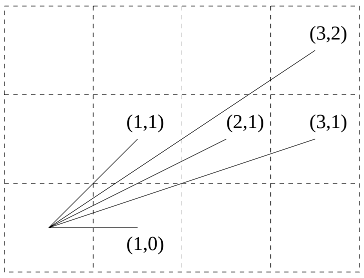


Figure 3.3: Connections between pixels in a 7×7 neighborhood for calculation of quasi-Euclidean distance values.

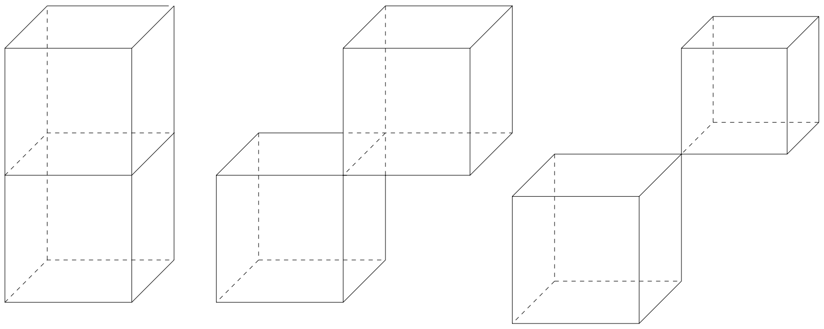


Figure 3.4: The neighborhood relations between two voxels in 3D space: face neighbors (left), edge neighbors (middle) and vertex neighbors (right).

Chamfer distances can be defined also in 3D. A voxel in 3D space has 26 immediate neighbors, 6 face neighbors, 12 edge neighbors and 8 vertex neighbors, as visualized in Figure 3.4. The regularity criteria for DTs can be applied in the cubic grid to select appropriate local distances a, b and c to the three types of neighbors. Borgefors [10] derived values $a \approx 0.926$, $b \approx 1.341$ and $c \approx 1.658$, which minimize the maximum difference from the Euclidean distance. The resulting DT is compared with the DTOCS and its variations in Section 3.2. Alternatively, the local distances can be optimized by minimizing the mean squared error from the real Euclidean length before discretization [4].

3.2 Optimal DTOCS

In the WDTOCS, locally Euclidean distances are used instead of chessboard distances, which are used to define the DTOCS. The lengths of the digital paths leading to the smallest distance values are calculated just as in the chamfer or quasi-Euclidean distance transformations of binary images. However, usually the objective is to approximate lengths of true geodesics on a real surface rather than lengths of digital paths on the discrete approximation of the surface. In binary DTs, the difference between the true distance between two points and the DT approximation of it, can be mathematically analyzed as in [6]. Such a thorough analysis is not possible for the DTOCS. The shortest path can be of any shape, so there is no basic assumption to follow, as in the case of binary DTs, where a minimal path consisting of a horizontal or vertical, and a diagonal line segment, always exists. Furthermore, there are numerous local distance values, compared to typically one or two different values in binary DTs. In the DTOCS, the number of different local distance values that can occur in the image equals the number of gray-levels available, and in the WDTOCS and the Optimal DTOCS, this number is doubled, as the same height difference can occur between edge neighbors and vertex neighbors.

As paths are curved, the local distances can not be optimized based on lines of different slopes, which is the basis for the optimization of the local distance coefficients for binary DTs. However, the error can be decreased by minimizing the error in the horizontal image plane. The Optimal DTOCS utilizes the coefficients, which minimize the maximum difference from the Euclidean DT for binary DTs [6] listed in Section 3.1. Replacing the horizontal local distance components 1 and $\sqrt{2}$ of the WDTOCS with these optimal distance coefficients results in the local distance definition of the Optimal DTOCS introduced in *Publication IV*:

$$d(p_i, p_{i-1}) = \begin{cases} \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + a_{opt}^2} & , \; p_{i-1} \in N_4(p_i) \\ \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + b_{opt}^2} & , \; p_{i-1} \in N_8(p_i) \setminus N_4(p_i) \end{cases} \tag{3.3}$$

In the case of real world images, which are only digital approximations of the surfaces they represent, there is no correct path length available for comparison, unless true geodesics can be measured on the original surface. In *Publication IV* distance definitions are compared using a mathematically defined synthetic surface, a gray-scale half-sphere (see Figs. 6. and 7. in *Publication IV*), and results of a similar experiment are shown in Figure 3.5. The distance values calculated from the “poles” of the sphere should, in theory, have equal values along the “equator” line, approximating one quarter of the sphere circumference. Due to the discretization of the sphere, the values vary significantly, and depend greatly on the distance definition used. The DTOCS underestimates lengths of paths with many diagonal displacements. For example, all the routes between the “poles”, which bypass the sphere along the background, are of the same undervalued length. The WDTOCS, on the other hand, overestimates the sphere circumference along the whole “equator” line. This is mostly due to the fact that only eight step directions are available, so there is little flexibility in how the paths are formed. Increasing the local neighborhood size provides new propagation directions. For example, in a 5×5 neighborhood, steps can be taken in 16 different directions.

In Figure 3.5 distances on the same sphere image are calculated using the Optimal DTOCS with the 3×3 neighborhood, and with the 5×5 neighborhood, in which the optimal coefficients a_{opt}, b_{opt} and c_{opt} derived in [8] are used as distance components in the image plane. It can be seen that increasing the neighborhood size decreases the error somewhat, but in the case of curved DTOCS paths, larger neighborhoods are risky, as very narrow obstacles can be missed. Also, there is no guarantee that larger

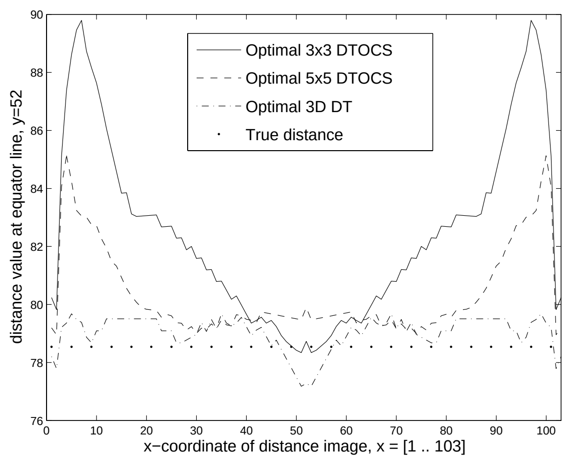


Figure 3.5: Comparison between the 3D DT, and the Optimal DTOCS with 3×3 and 5×5 neighborhood.

neighborhoods produce a more accurate approximation for the lengths of curved paths, as straightness within the larger neighborhood is assumed in the optimization of local distances [28].

A common approach to solving problems in gray-level morphology is transforming the gray-level image into an umbra, that is, a voxel image representing the surface relief, and then using well known binary morphological operations in 3D. For example, the convex hull of a gray-level image can be calculated by filling concavities on the 3D surface representation of the image [38]. The equivalence of gray-level images and umbrae is discussed in [63], and visualized in Figure 1.1. Here, the DTOCS distances are compared to values obtained using the 3D DT with the optimal local distances to the 26 immediate neighbors of a voxel derived by [10], which are listed in Section 3.1. The 3D DT is calculated on the top surface of the umbra, whose height z at each point (x, y) in the horizontal plane corresponds to the gray-value of the corresponding pixel. The top surface is defined as the set of voxels connected to the background. It can be seen in Figure 3.5 that the 3D DT produces more accurate results, that is, the distance values along the equator line show less variation than the corresponding values produced by the Optimal DTOCS. The larger 5×5 neighborhood improves the DTOCS results, but not sufficiently. The 3D distance transformation has more propagation directions, so the formation of paths is more flexible. The digital 3D approximation of the path could further be refined by using more accurate length estimators of 3D curves [28], or by so called curve shortening flow [27]. The memory requirement is considerably higher, when a 3D representation of the surface is used, and consequently, the computational cost is higher. Thus, the DTOCS provides efficient 2D solutions to some inherently 3D problems, as discussed in *Publication IV.*

3.3 Chamfer Variations of the DTOCS

Floating point operations can be executed efficiently with modern computers, but in earlier work on image processing, integer operations have been a clear preference. Borgefors multiplied the optimal local distances a_{opt} and b_{opt} with various integer factors, rounded the resulting distances, and found that even though the approximations improve with increasing scaling factors, factor 3 producing local distances 3 and 4 is sufficiently accurate to be recommended for use [6]. These local distance values have been introduced as the horizontal distance components in the 3-4-DTOCS presented in *Publication IV*. When the height difference is scaled with 3 as well, the distance values approximate true distances on a surface more accurately than the chessboard DTOCS. The main benefit is that straight steps are preferred, so detours along diagonal steps are eliminated. Examples of how such detour paths are formed in the DTOCS, with equal distances to edge neighbors and vertex neighbors, can be seen, for example, in Figures 3 (a) and 5 (a) in *Publication II*.

The $\sqrt{2}$-DTOCS adds locally Euclidean distances in the horizontal plane with gray-level height differences. The resulting distance values correspond to the lengths of digital paths consisting of horizontal steps in the xy-plane and vertical steps in the direction of the z-coordinate of the 3D surface the gray-level height map represents. If approximations of true geodesics are needed, the 3-4-DTOCS is more accurate, and also more efficient, as only integer operations are needed. As pointed out in [64], efficiency is not the only, or even the primary reason to prefer integer DTs. More importantly, the integer result is easier to use in many basic applications, like skeletonization, or identifying centers of maximal discs, and reconstructing shapes based on them. Also, the integer DTs are metrics, whereas the limited accuracy in representing floating point distances can violate the metrics criteria, as explained in Section 2.3.

Attempts to find integer approximations for the local distances of the WDTOCS and the Optimal DTOCS have failed so far. Table 3.1 lists local distance values for various height differences. For each height or gray-level difference, there is one local distance value for the DTOCS, and two for the real valued modifications. It can be seen that the difference between the straight and the diagonal local distance decreases when the height difference increases. Using rounded local distance values as integer approximations would eliminate the distinction between neighbors with the same gray-level value, and neighbors differing by one gray-level. Also, the small but still significant difference between straight and diagonal neighbors would be lost. Even though the diagonal and the straight local distance asymptotically approach each other, the small difference ensures that straight paths are shorter than detour paths along diagonal steps, if no height difference is involved.

Multiplying all local distances by a factor, and then rounding the results, might produce some usable approximations of local distance values. However, the difference between straight and diagonal steps inevitably vanishes at some point. For example, using a factor 2 results in local distances of 6 and 7 for a height difference of 3, but local distance values for a height difference of 4 both round to 8. Analyzing the effect of different scaling factors would require statistics on how often each gray-level difference appears in the image, so optimizing the local distance values would become image dependent.

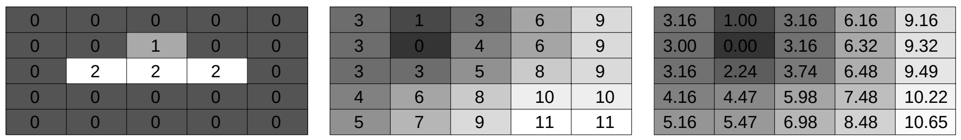
Table 3.1: Local distance values for DTOCS, WDTOCS and Optimal DTOCS.

Height diff	DTOCS	WDTOCS		Optimal	
0	1	1.000	1.414	0.955	1.369
1	2	1.414	1.732	1.383	1.696
2	3	2.236	2.449	2.216	2.424
3	4	3.162	3.317	3.148	3.298
4	5	4.123	4.243	4.112	4.228
5	6	5.099	5.196	5.090	5.184
:					
10	11	10.050	10.100	10.046	10.093
:					
20	21	20.025	20.050	20.023	20.047
:					
40	41	40.012	40.025	40.011	40.023
:					
80	81	80.006	80.012	80.006	80.012
:					
160	161	160.003	160.006	160.003	160.006
:					
255	256	255.002	255.004	255.002	255.004

3.4 DTOCS for Anisotropic Grids

When calculating distances along gray-level surfaces, it is important to consider the scaling of the image. To approximate actual distances, the difference between consecutive gray-levels must correspond to the image resolution in the xy-plane. For instance, if the straight distance between pixel centers in the horizontal plane is 1 mm, the image should be constructed so that one gray-level corresponds to 1 mm. This is not only an issue for the DTOCS. If distances are calculated on the 3D umbra representation of the surface, the resolution of the z-coordinate indicating the height of the umbra should be the same as the resolution in the xy-plane. Instead of scaling the original image, a scaled version of the DTOCS can be implemented. If one horizontal pixel displacement corresponds to r gray-levels, the horizontal distance component in the DTOCS definition, Equation 2.6, can be replaced with r. Similarly, if one gray-level corresponds to the length of r steps in the horizontal plane, the height difference component in the local distance definition can be multiplied by r.

Some scaling in the flat image plane might also be needed, if the resolution of the original image is not the same in the x- and y-direction. In an anisotropic grid, the step length in the x-direction can be r_x and the step length in the y-direction r_y. One approach to transforming such images is interpolating additional values in the direction with the lower resolution, but this leads to a considerably larger image size. A more efficient solution is modifying the local distance definitions. The DTOCS local distances, which



0	0	0	0	0
0	0	1	0	0
0	2	2	2	0
0	0	0	0	0
0	0	0	0	0

(a) Original image

3	1	3	6	9
3	0	4	6	9
3	3	5	8	9
4	6	8	10	10
5	7	9	11	11

(b) DTOCS

3.16	1.00	3.16	6.16	9.16
3.00	0.00	3.16	6.32	9.32
3.16	2.24	3.74	6.48	9.49
4.16	4.47	5.98	7.48	10.22
5.16	5.47	6.98	8.48	10.65

(c) WDTOCS

Figure 3.6: Example of the DTOCS and the WDTOCS in a rectangular grid, where the pixels are three times longer in the x-direction than in the y-direction.

include the possibility for scaling the height with a factor r_z, become:

$$d(p_i, p_{i-1}) = \begin{cases} r_z|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})| + r_x \ , \ p_{i-1} \text{ neighbor of } p_i \text{ in } x\text{-direction} \\ r_z|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})| + r_y \ , \ p_{i-1} \text{ neighbor of } p_i \text{ in } y\text{-direction} \\ r_z|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})| + \max(r_x, r_y) \ , \ p_{i-1} \text{ diagonal neighbor of } p_i \end{cases} \tag{3.4}$$

Similarly, the WDTOCS can be generalized to rectangular grids:

$$d(p_i, p_{i-1}) = \begin{cases} \sqrt{r_z^2|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + r_x^2} \ , \ p_{i-1} \text{ neighbor of } p_i \text{ in } x\text{-direction} \\ \sqrt{r_z^2|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + r_y^2} \ , \ p_{i-1} \text{ neighbor of } p_i \text{ in } y\text{-direction} \\ \sqrt{r_z^2|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + r_x^2 + r_y^2} \ , \ p_{i-1} \text{ diagonal neighbor of } p_i \end{cases} \tag{3.5}$$

A small example, where the step length r_x is three, and the step length r_y and the height scaling r_z are one, is shown in Figure 3.6. The pixels in the original image are, in practise, rectangular, as visualized in Figure 3.6 (a). The DTOCS and the WDTOCS calculated in the rectangular grid are shown in Figures 3.6 (b) and (c).

This combination of scaling height differences, and scaling horizontal local distance components produces a DT, which is applicable for any resolution, that is, for step lengths r_x and r_y in the horizontal plane, and for the unit height difference r_z vertically. The weights derived by Sintorn and Borgefors [58] for binary DTs in rectangular grids could be exploited to modify the Optimal DTOCS for images with a different resolution in the x- and y-direction. The weights minimize the error from the Euclidean distance for linear trajectories, similarly as the coefficients a_{opt} and b_{opt} described in Section 3.1. The discrete distance operator for rectangular grids in [15] is optimized based on circular trajectories, resulting in slightly different local distances. Weighted 3D DTs for elongated voxel grids have been developed in [59]. In [20] a method for systematic optimization of chamfer distances in 3D anisotropic grids is presented.

The length units do not necessarily have to be integers. Even the DTOCS, which is originally based on chessboard distances, can easily be modified to transform images containing floating point height values, like profilometer data. The horizontal step lengths r_x and r_y defined by the image resolution can also be any real numbers. The DTOCS calculates the local distance as the sum of the horizontal and the vertical step, while the Euclidean distance between the pixel centers is calculated in the WDTOCS. The generalization of the DTOCS and the WDTOCS for anisotropic grids will be utilized in future work on surface roughness measurement based on ideas presented in Section 6.2.

In conclusion, the DTOCS and the WDTOCS, which are the basis for this work, can be improved significantly by using chamfer distances as the horizontal distance component in the local distance definition. They can also be generalized to anisotropic grids, but so far, the DTOCS and the WDTOCS for anisotropic grids have not been implemented using local distance weights producing better approximations for the Euclidean distance in the image plane. Table 3.2 lists the distance definitions used in this work, and their important properties. The integer DTs are metrics, and the floating point DTs are by definition metrics, but the limited number of bits available for representing them may result in violations of the metrics criteria. Also, if the surface, along which distances are calculated is represented using floating point values, or the resolution of the image requires using non-integer step lengths, the resulting floating point distances are not metrics according to the strict definition.

Table 3.2: Variations of the DTOCS and their properties.

Algorithm	Local distance	regularity	Metric ?
DTOCS	chessboard in the image plane + height difference	semi-regular in the image plane	metric (integer)
WDTOCS	Euclidean	regular in the image plane	floating point
$\sqrt{2}$**-DTOCS**	Euclidean in the image plane + height difference	regular in the image plane	floating point
3-4-DTOCS	chamfer in the image plane + scaled height diff.	regular in the image plane	metric (integer)
DTOCS anisotropic	scaled chessboard in the image plane + scaled height diff.	semi-regular in the image plane	metric (integer)
WDTOCS anisotropic	Euclidean from scaled differences	regular in the image plane	floating point

Chapter IV

Transformation Algorithms

Distance transformation algorithms are based on propagating local distance values across the image, as described in Section 2.2. In Euclidean DTs, the propagated information is usually a combination of horizontal and vertical pixel differences [55], or in other words, a vector pointing towards the nearest feature pixel [18], [17]. The vector norm gives the distance value, when the transformation is finished. The propagation can proceed either in parallel or sequentially, and the propagation order can be predefined, as in the SLT, or it can be determined during the transformation based on previously calculated distance values. The forward scan of a SLT typically proceeds row-wise from top to bottom using a mask that propagates distances to the mask center pixel from its top and left neighbors. The reverse scan proceeds with a mask designed to propagate distances from pixel neighbors, which are located below or to the right of the current pixel. Alternative mask configurations requiring three or four raster-scans have also been used [40], and the Euclidean distance transformation by Ragnemalm uses four masks in a 4-scan algorithm [44]. Also, a DT may be obtained by merging results of scanning rows and columns separately [39]. In recursive and ordered propagation algorithms, as well as in pixel queue approaches, the transformation begins at the reference pixels, and proceeds towards pixels further away in the calculation area. Instead of propagating single distance values, or vectors in the case of Euclidean DTs, segments of the propagating distance boundary can be processed, resulting in a chain propagation algorithm [79].

This chapter explains the main approaches to distance propagation, and relates them to the DTOCS setting, where paths typically are curved. In the earlier parts of this research, which resulted in *Publications I-IV*, the SLT was used for calculating the DTOCS. The sequential approach was found to be inefficient when calculating distances in situations, where paths become long. For example, in the Route DTOCS algorithm finding the shortest route between two points, distances need to be calculated from one point to every other point in the image (Chapter 5). Producing the long curved paths require several iterations of the sequential algorithm. This chapter presents a new more efficient approach, the priority pixel queue DTOCS algorithm, which was introduced in *Publication V*, and extended to produce the nearest neighbor transform in addition to the distance map in *Publication VI.*

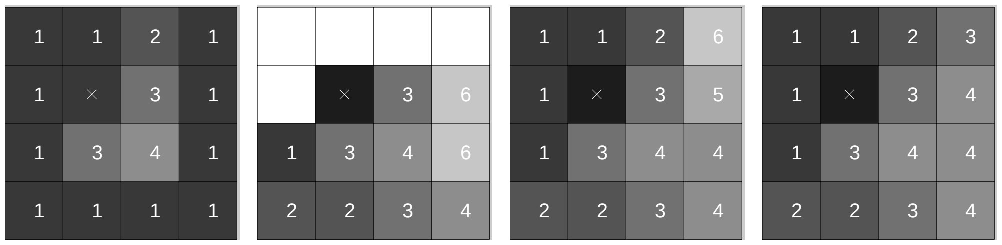
4.1 Sequential and Parallel Local Transformations

Even before the pioneering work in DTs by Rosenfeld and Pfaltz, digital images were processed using local operations on picture neighborhoods [47]. Parallel local operations affect each pixel independently, without taking into account the new values obtained for the neighboring pixels in the same parallel scan. The method is based on the idea of having a parallel computer with one processing element for each pixel. Some specialized architectures, like pyramid machines [11], can be used to efficiently implement parallel transforms, but advanced image processing today is typically done on sequential computers. SLT algorithms also transform each pixel using its neighborhood, but as the pixels are processed sequentially rather than in parallel, the new values are used as soon as they become available. In [47], the parallel and sequential approaches are shown to be mathematically equivalent.

The sequential implementation of the DTOCS is based on the modification of the SLT algorithm by Rosenfeld and Pfaltz [47] introduced in Section 2.2. In the DTOCS, any pixels can be reference pixels, from which distances are calculated. To define the calculation area, a binary image $\mathcal{F}$ is needed for the transformation in addition to the original gray-level image $\mathcal{G}$. The calculation area X in $\mathcal{F}$ is initialized to max (the maximal representative number of memory) and the complement area, X^C, to 0. The zero valued pixels indicate the source points for the distance calculations, that is, the feature pixels, when distances are calculated from the nearest feature, or the background pixels, when distances are calculated from the background into the object. Neither the calculation area, nor the reference pixel set needs to be connected. The original gray-level image is only used for calculating the local distance values between pixels, and remains intact through the transformation. The binary image, which originally defines the calculation area, is transformed into a gray-level image containing distance values.

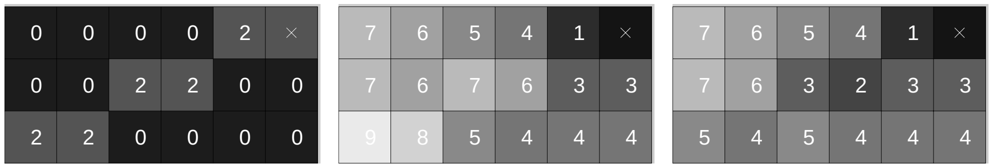
The DTOCS algorithm proceeds according to the distance propagation defined by Equations 2.4 and 2.5 in Section 2.2. The distance between pixel p_c and its neighbor p in one of the two masks shown Figure 2.1, is $\Delta(p) = d(p, p_c)$ according to the distance definition used, for example, Equation 2.6 for the DTOCS and Equation 2.7 for the WDTOCS. The Δ-notation is used to differentiate distances inside the calculation mask from local distances along a digital path. The result of the first pass is used as input for the second pass, that is, the binary image indicating the reference pixels in the first pass is replaced with the unfinished distance image in the second pass, and also in subsequent iterations. Similarly, the original SLT algorithm in [47] processes the same distance image both in the forward and the reverse raster-scan, modifying the values obtained by the first scan in the the second scan [47], but repeated iterations can not be performed. The gray-weighted distance transformation algorithm by Rutovitz [50] transforms the original image in the forward pass, and a copy of the original image in the reverse pass. The result is obtained as the minimum of the two transformed images.

Figure 4.1 shows a simple example of how the sequential transformation proceeds when calculating the DTOCS. Gray-values or distance values are shown on each square representing a pixel. The reference pixel marked with 'x' has gray-value 1 in the original image, Figure 4.1 (a), and value 0 in the distance images, Figures 4.1 (b)–(d). In the result of the first forward pass, Figure 4.1 (b), the pixels shown in white have not yet been updated, that is, they still have value max. After the reverse pass, Figure 4.1 (c),



(a) Original image (b) Distance image after 1. pass (forward) (c) Distance image after 2. pass (reverse) (d) Final distance image after 3. pass (forward)

Figure 4.1: An example of a sequential distance transformation.



(a) Original image (b) Distance image after 2. pass (reverse) (c) Distance image after 6. pass (reverse)

Figure 4.2: A sequential distance transformation requiring one pass for each pixel on the longest path.

all pixels have obtained a distance value, but not necessarily the correct one. The final DT, Figure 4.1 (d), is obtained in the forward pass of the second iteration.

The two passes of the DTOCS are repeated until the distance values converge, that is, until no changes appear in the distance image. The size and the complexity of the transformed domain affect the convergence. In practise, the resolution of the image is a significant factor. In 2D convex integer domains, two or at most four passes produce the final DT, but for non-convex domains of complex shape, the number of passes necessary is generally greater [40]. As demonstrated in Section 2.4, varying height surfaces are inherently non-convex domains, for which sequential distance transformations typically require numerous iterations, and the number of iterations needed is quite unpredictable. The number of pixels in the longest path is the theoretical upper limit for the number of iterations. Figure 4.2 presents an example, where this limit is reached. The first forward pass does not update any pixels on the diagonal path from the reference pixel in the top right corner to the pixel in the bottom left corner of the image. In each subsequent pass, correct distance values propagate only one step along the path. Figure 4.2 (b) is the result after the first two passes. Four pixels on the path have distance values obtained along paths, which are not minimal. The final distance values along the globally minimal path are reached after three two-pass iterations, Figure 4.2 (c). A similar example is used in [40] to demonstrate that even convex binary domains may require more than two passes of a sequential transformation.

In practical applications, the number of iterations rarely reaches the theoretical upper limit. Convergence is reached in about 10–15 two-pass iterations in the experiments presented in *Publication III* and in [69], but examples requiring approximately 80 iterations can be found in *Publication V*, which presents convergence experiments on different types of images at different resolution. As expected, a flat surface containing constant gray-values can be transformed in two iterations, one for the transformation itself, and another to detect convergence, just like most convex objects in the case of binary DTs. In fact, the flat surface is a convex object according to the definition of convexity given in [40], but introducing gray-level variation generally makes the surface non-convex, with the exception of some special cases, like smooth gray-level slopes.

4.2 Recursive and Ordered Propagation

Instead of propagating distances in a predefined scanning order, the calculation can be initiated at the reference pixels, propagating into the calculation area. Piper and Granum [40] presented recursive and ordered propagation algorithms, and demonstrated their efficiency compared to the SLT approach, which they call the propagated distance transformation (PDT). The recursive propagation is essentially a depth-first search, whereas ordered propagation proceeds as a breadth-first search. Both start at the reference pixels, and the difference is in how they process the neighbors, which obtain new distance values. The recursive propagation starts at one reference pixel, and recursively processes each of its neighbors. The propagation is highly dependent on the order in which the neighbors are processed. A fixed ordering can lead to the following unfortunate scenario, visualized by the example in Figure 4.3:

The top right pixel in the 3$\times$3 neighborhood is always processed first, and the propagation starts from the bottom left corner of the image. The propagation shoots diagonally across the image to the top right corner processing only pixels on the diagonal, and at each step the recursion depth increases by one. The the propagation continues in the direction of the next available neighbor to be processed. In this example, neighbor pixels are processed in a clockwise order, starting from the top right pixel. Figure 4.3 (b) shows the intermediate result after 9 recursive calls of the propagation. If the recursion depth is not a problem, and the distance transformation propagates smoothly, that is, the local distances are the same between all neighbor pixels, or vary only somewhat, everything may work out fine. But if varying local distances are used, as in the DTOCS, a shorter path to one of the already processed pixels is likely to exist. Then all the distance values, which have propagated from that pixel, are erroneous, and need to be recalculated. If the error appears early in the transformation, most of the image may already be processed by the time the recursion returns to correct the value of the pixel, from which the propagation of the erroneous distance values originated. Vast amounts of reprocessing must be performed after updating the value. In Figure 4.3 (c) the pixel with value 1 in the bottom row has just obtained its correct value, and the pixels to the right of it still have erroneous values. Still after 125 recursive calls, the top left pixel is untouched, and after obtaining the final result after 130 recursive calls, several recursive calls are still needed to check that all distance values are final. In this example, the recursive propagation function was called 344 times while processing the image with only 25 pixels.

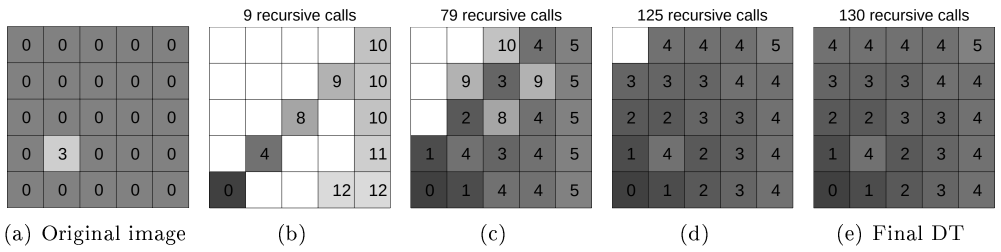


Figure 4.3: Recursive propagation with a fixed processing order for pixel neighbors. The white pixels in the intermediate results (b)–(d) have not yet received a distance value. The number of recursive calls needed to produce each result is indicated above each image.

The recursive propagation can be enhanced by exploiting the fact that distances typically propagate as fronts. Piper and Granum [40] devised an ordering, where the "back-direction" of the previous propagation step is used, that is, if the distance value propagated from the straight left pixel neighbor, the diagonal neighbors to the left of the current pixel are explored first. In the scenario described above, this method would be clearly beneficial, as distance values would slowly proceed from the corner of the image, alternating different propagation directions, and concentrating first on the area near the reference pixel. Some reprocessing may still be needed, but the risk of having to reprocess most of the image area repeatedly is considerably smaller. Cyclic and random ordering are additional options for determining which neighbor to recursively process next, but less efficient than the specific order based on the propagation direction [40].

Ordered propagation utilizes a queue data structure instead of recursion. When a pixel obtains a new distance value, its neighbors are enqueued to wait for processing in a first-in-first out order. Pixel queue algorithms utilizing a first-in-first-out queue, for example, in [80] and [57], are implementations of ordered propagation, or breadth-first search. In the case of binary DTs, ordered propagation effectively performs successive dilations from the reference set [40]. The contour processing method for dilation, erosion, propagation and skeletonization by Vliet and Verwer [73], counts the number of pixels enqueued at each iteration so that the whole propagating front can be dequeued in the following iteration. In the case of the DTOCS, distances do not propagate as such continuous fronts, as paths across image areas with much variation contain fewer pixels than paths across smoother areas. If neighbors of updated pixels are reprocessed, the ordered propagation, as well as the recursive propagation, produces correct results also for the DTOCS. The ordered propagation typically requires less reprocessing than the recursive propagation. The main problem with both approaches is that distance values are propagated forward without knowing for sure whether they are final. The priority pixel queue algorithm solves this problem, eliminating the need for reprocessing, and recalculation of distance values.

4.3 Priority Pixel Queue Distance Transformation

The priority pixel queue DTOCS algorithm developed in this work eliminates repetition of local distance calculations by processing pixels in order of increasing distance values. In sequential mask operations, and in recursive and ordered propagation of varying local distance values, recalculations are inevitable, as distances do not propagate as smooth fronts. Figure 4.4 shows an example, where distances from one reference pixel, marked with 'x', are calculated along a varying height surface. The pixels shown in white in Figure 4.4 (b) have a distance value equal to 80 in the image of size 128×128. No continuous equal distance curves are formed due to the highly varying local distances. In Figure 4.4 (c) a continuous border is found by marking pixels which have at least one edge neighbor with a distance value greater than 80, and at least one edge neighbor with a distance value less than or equal to 80. The boundary limits the area in which the pixels are processed by the time the distance reaches the given value, so all pixels inside the border have a distance value of less than 80. Priority ordering is needed to cope with this kind of propagation, which proceeds further in smooth areas than in areas with much variation.

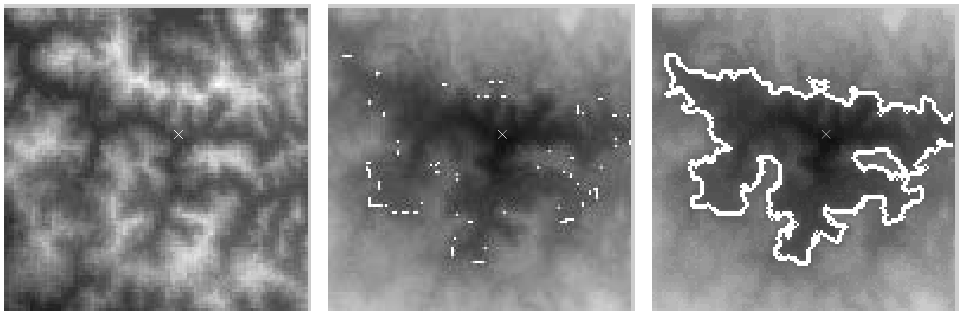

(a) Original image (b) Equal distance pixels (c) Equal distance border

Figure 4.4: Example of equal distance pixels and border in the DTOCS.

Various priority ordering approaches to distance transformations have been presented. The geodesic time DT by Soille utilizes a priority value, which is incremented by one, once all pixels with the current distance value have been processed [62, pp. 238–239]. As it lists all possible distance values, it can be used only for numerable distance values. The same applies to bucket sorting approaches [78], [17]. The contour processing algorithm by Ragnemalm can handle floating point distance values as well as integers, as priority ordering is achieved by maintaining an upper limit for distance values to be propagated at each iteration [42]. Once applicable pixels are processed, the limit is increased by the smallest possible local distance value. However, as local distance values vary greatly in the DTOCS, scanning the contour set becomes inefficient. In the case of binary DTs, with one or maybe two possible local distance values, most of the contour set can be processed at each iteration. When transforming a highly varying image, only a fraction of the pixels in the contour set is applicable after each update of the priority limit, so numerous scans of the contour set are needed. In the worst case, all pixels, which are currently in the propagating contour, have different distance values, and only one pixel is found applicable during each scan of the contour set.

In the new priority pixel queue algorithm developed in this work, priority ordering is maintained using a queue implemented as a minimum heap. A minimum heap is a binary tree, where the children of a node always have a larger value than their parent. Consequently, the root of the tree contains the minimum value. Addition of a new node, or updating the tree, when the root node is removed, takes $\mathcal{O}(\log n_q)$ time, where n_q is the length of the queue. Descriptions of the heap data structure can be found in almost any basic book on algorithms, for example, [16]. The priority pixel queue algorithm presented in *Publication V* is as follows:

```
1. Initialize the binary image $\mathcal{F}(x) = \begin{cases} 0 & , x \in X^C \quad \text{(reference pixels)} \\ max & , x \in X \quad \text{(calculation area)} \end{cases}$
2. Put pixels with $\mathcal{F}(x) = 0$ to *priority queue* Q.
3. While Q not empty
       $p = dequeue(Q)$, $\mathcal{F}_q(p)$ was the smallest distance in Q.
       If $\mathcal{F}_q(p) > \mathcal{F}(p)$ (obsolete value), continue from step 3.
       $\mathcal{F}(p)$ becomes $\mathcal{F}^*(p)$ (value is final).
       For neighbors $x$ of $p$ with $\mathcal{F}(x) > \mathcal{F}^*(p)$
           Compute local distance $d(p,x)$ from the original image $\mathcal{G}$
           If $\mathcal{F}^*(p) + d(p,x) < \mathcal{F}(x)$
               Set $\mathcal{F}(x) = \mathcal{F}^*(p) + d(p,x)$
               $enqueue(x)$
           end if
       end for
   end while
```

The basic idea of the algorithm is that reference pixels are enqueued to the minimum heap, and then dequeued for processing in priority order. Distance values are propagated from the dequeued pixel to its neighbors, and neighbors, which obtain a new distance value are enqueued. The priority ordering ensures that only final distance values are propagated further. If a shorter path is found to a pixel, which has already been enqueued, the distance value is replaced. Previous instances of the pixel in the queue become obsolete, and can be discarded. When the queue is empty, all distance values are final. The scenario described for the recursive propagation, where updating a distance value can lead to significant amounts of reprocessing, can not occur in the priority queue transformation. In fact, no local distance is calculated more than once. Numerous redundant local distance calculations, including the reverse directions of already calculated local distances, are eliminated by not calculating distances to neighbors, which already have distance values less or equal to the value they could obtain in the next propagation step. Local distances are calculated only in the possible propagation directions, as in the directed masks used by Ragnemalm [43] and in the bucket sorting algorithm by Verwer et al [77], which eliminate the direction, from which the propagation arrived.

The priority pixel queue algorithm can handle floating point distances as well as numerable ones, and the magnitude of the distance values is not a problem, unless arithmetic precision of the computer becomes a factor. In approaches based on enumerating distance values, for example, the priority queue designed for the geodesic time DT [62, pp. 238–239] or the contour processing approach [42], the number of iterations needed is proportional to the largest distance. In the chessboard DT, the maximum distance is the length or the width of the image. The DTOCS can, in the worst case, result in a long curved path, where each local distance equals the difference between the smallest and the largest gray-level in the image plus one. The same problem applies to any gray-level DT, and huge distance values are even more likely to appear in DTs summing gray-values rather than gray-level differences along the path. Thus, the priority ordering would be beneficial for any gray-level DT.

The main results from *Publication V* are shown in Figure 4.5. Distances were calculated on four images, using two different resolutions for each image. In each run, one of 244 pre-selected pixels was used as the reference pixel, from which distances were calculated. The results are averages of the 244 runs for each image, and each version of the algorithm. The complexity of the images increases from the left to the right. The image named 'Flat' contains only one constant gray-value, representing the smoothest surface possible. The other three, which are shown in Figure 1. in *Publication V*, are increasingly complex, so that the image named 'Merc' contains the most variation. It can be seen that the number of local distance calculations needed in the priority pixel queue algorithm (PQ in Figure 4.5) is almost the same in all images of the same resolution. The upper limit is known based on the fact that no local distance is calculated more than once. The SLT requires numerous repetitions, as each local distance calculation is repeated twice in each iteration consisting of one forward and one reverse pass, and the number of iterations needed grows with the complexity of the image. The results include the extra iteration needed to detect convergence, so the number of local distance calculations indicated for the 'Flat' image is the result of two iterations. The ordered propagation (OP) does some reprocessing of distance values, when new shorter paths to previously processed pixels are found. The amount of reprocessing needed increases with the complexity of the image, but not as much as in the SLT.

The running times of the tests, which can be seen in Figure 2 of *Publication V*, indicate that the running time of the priority pixel queue algorithm does increase slightly, when the complexity of the image increases. The number of queue operations increases, and also the length of the queue representing the propagating border increases, so queue operations take more time. The length of the priority queue is generally of a smaller magnitude than the problem size, that is, the number of pixels in the image. The longest queues in the experiments in *Publication V* contained only a few percent of the pixels in the image. The statistics of the Mercury surface test image, visualized in Figure 6 in *Publication V*, are as follows: the longest queue in tests with the image of size 256×256 contained $4527 \approx 6.9$ % of the pixels in the image. In the images with increased resolution, the longest queues were relatively shorter ($9794 \approx 3.7$ % of 512×512 pixels, and $15755 \approx 2.7$ % of 768×768 pixels), suggesting that the queue lengths grow sub-linearly with the size of the image. A contributing reason for the sub-linear growth

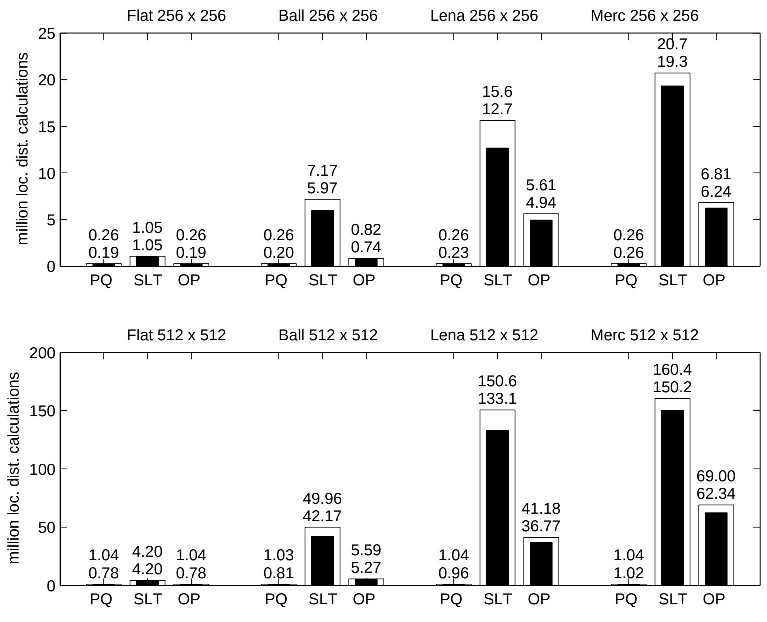


Figure 4.5: Average number of local distance calculations needed in DTOCS (black bar) and WDTOCS (white bar) using Priority Queue, Sequential and Ordered Propagation algorithms.

is the use of resized versions of the same images, as increasing the image resolution does not increase the complexity of the underlying surface. However, the experimental finding is supported by the fact that the propagating distance front, which consists of the pixels in the queue at any moment in the transformation, is a 1D structure in a 2D image. For example, in a chessboard distance transformation starting from one pixel in the center of a square image, the propagating front expands outward until it reaches the edges of the image (see Figure 4.6). Consequently, the longest queue contains about $4n$ pixels in an image of size n^2. If the the number of pixels increases by a factor of k^2, the circumference of the square image, which provides an estimate of the maximum queue length, increases only by a factor of k. The propagating front can become very curved in the DTOCS, and may momentarily contain more pixels than the image circumference, but having one less dimension than the image, its maximum length should not increase linearly with the resolution of the image.

The propagating front can surround disconnected areas, if the set of reference pixels is disconnected. An example can be seen in Figure 4.7, where intermediate results of a distance transformation initiated from two reference pixels are shown. The white areas indicate pixels, which have not yet been processed, and the borders of the gray areas indicate the pixels in the propagating border. The distance values propagating from each reference pixel are mixed in the priority queue. When the segments of the propagating front meet, the transformation is final.

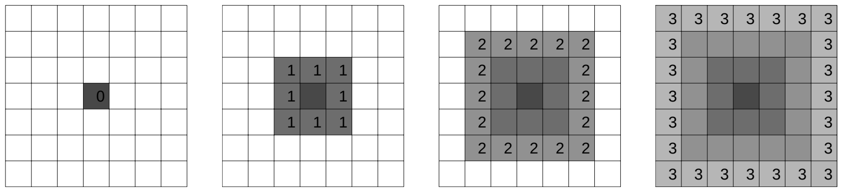


Figure 4.6: The propagating front in a priority queue chessboard DT.

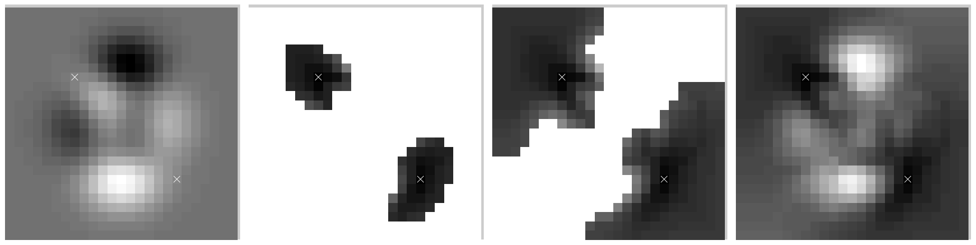


Figure 4.7: The propagating front in a priority queue DTOCS transformation. The original image is shown to the left, and final DT to the right. The two reference pixels, from which the DT is initiated, are marked in each image.

In [73], an image pattern is presented, where the image consists of 3×3 pixel squares, and in each square, the center pixel is a feature pixel, as shown in Figure 4.8. In such a case, every ninth pixel is initially enqueued. Once all feature pixels have been dequeued, all their neighbors, that is, 8/9 of the pixels in the image are in the queue. An even longer queue can occur if more than 8/9 of the pixels are reference pixels, but then the queue length decreases rapidly. Due to pathological cases like these, the implementation must be able to handle queue lengths up to the number of pixels in the image. The memory requirement of the algorithm is about three times the image size, as the original image and the distance image, as well as the priority queue, must be handled simultaneously. The sequential algorithm can be implemented using slightly less memory, that is, twice the image size, plus a small constant amount for the mask operations.

The time complexity of the pixel queue algorithm is discussed in *Publication V*. The time complexity is in $\mathcal{O}(n \log n_q)$, where n is the number of pixels in the image, and n_q the length of the queue, as the number of queue operations, enqueue and dequeue, is in the order of $\mathcal{O}(n)$, and each queue operation is in $\mathcal{O}(\log n_q)$. The queue length n_q varies throughout the transformation, but typically it is of a smaller magnitude than n, so the worst case complexity of the priority pixel queue algorithm, $\mathcal{O}(n \log n)$ is a gross overestimate. In most cases, the priority pixel queue algorithm is superior compared to the sequential approach, as presented in *Publication V*.

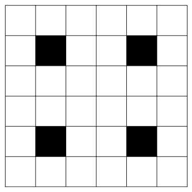


Figure 4.8: An image pattern, which results in a pixel queue containing 8/9 of the image pixels, after the black reference pixels have been dequeued and processed.

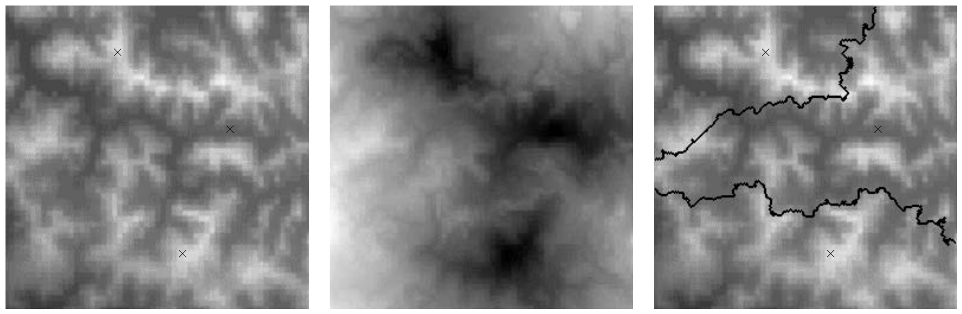
4.4 Nearest Neighbor Transformation

The nearest neighbor transformation produces a tessellation image, in which each pixel is marked with the identity of its nearest reference pixel. Thus, the image is divided into regions so that each reference pixel is surrounded by pixels, which are closer to it than any other reference pixel. Pixels at region borders, however, can be equally close to two or more reference pixels. The nearest neighbor transform (NNT) is a discrete version of the Voronoi diagram. The Voronoi diagram for a finite set of seed points in a plane divides the plane to polygons, so that each point within a polygon is closer to the seed point inside the polygon than any other seed point. The polygon of a seed point p_i is defined as $V(p_i) = \{r \,|\, r \in R^2$ and $d(r, p_i) \leq d(r, p_j), j \neq i\}$ [33]. Voronoi diagrams can be explained using the analogy of expanding waves:

"If a pebble is dropped into a still pond, circular waves move out from the point of impact. If n pebbles are dropped simultaneously, the places where the wave fronts meet define the Voronoi diagram." [13]

DTs are closely related to Voronoi diagrams. In [2], the Voronoi diagram of a binary digital image is computed by selecting edge pixels from the skeleton of the background, called the exoskeleton, produced by a distance transformation from the foreground. The Euclidean DT of a binary image in two or more dimensions can be calculated in linear time from the corresponding Voronoi diagram [12], [36]. In gray-level DTs, there is no such straightforward link from the Voronoi diagram to the DT, as values along the paths define the distances. The NNT based on the DTOCS is described in *Publication VI.* It divides the image into areas so that each pixel is assigned to the seed, which is nearest according to the distance along the varying height surface.

Figure 4.9 shows a terrain height map divided into regions based on DTOCS distances from three seed points. The implementation of the nearest neighbor transformation is a simple extension of the priority pixel queue distance transformation algorithm. Each reference pixel is assigned a seed number between 1 and n_f, where n_f is the number of features, and the seeds are marked into the tessellation image. The seed values are propagated together with the distance values. The mechanism could also be used in a sequential algorithm, as discussed in [2], but the implementation of the propagation according to Equations 2.4 and 2.5 would have to be modified to propagate the identity of the mask pixel providing the minimum distance value. If a pixel is enqueued repeatedly, that is, it obtains a new lower distance value, its seed value is replaced with the seed value



(a) Original image with seeds (b) DTOCS image (c) Nearest neighbor regions

Figure 4.9: Nearest neighbor region boundaries on a terrain height map.

of the pixel, from which the new distance value propagated. The final seed value identifies the feature, from which the propagation of the final distance value originated. The idea of propagating seed values simultaneously with distance values is used for Dirichlet tessellation of binary images in [6], but priority ordering is achieved using a parallel distance transformation. The chain propagation distance transformation algorithm by Vincent produces the Euclidean skeleton by influence zones (SKIZ), which is, in practise, a NNT, by propagating labels together with the distance chains [79]. The geodesic SKIZ by Soille is a NNT based on the gray-level DT calculating minimal cost paths [62, p. 227]. The morphological Voronoi tessellation algorithm for 3D volumes presented in [25] grows each region by one voxel in all directions by propagating tessellation seeds.

A nearest neighbor transformation of a gray-level surface can produce regions, which vary considerably in size, even if the seed points are equally spaced across the image, as in Figure 1 in *Publication VI*. The region border between a seed point in a rough area and a seed point in a smooth area is formed closer to the seed point in the rough area, as paths can propagate further across image areas with less variation. This could lead to some applications in segmentation, for example, separating highly varying textures from smoother textures. In the segmentation algorithm called seeded region growing [1], pixels are assigned to areas based on similar gray-level values. As local distances between pixels with almost equal gray-values are small, the NNT based on the DTOCS could be useful in a similar segmentation approach. Within the scope of this work, the NNT is used only for producing the roughness map described in 6.2.

This chapter discussed the various approaches to distance propagation, and introduced the priority pixel queue algorithm, which was developed in this work. The priority pixel queue approach is beneficial in any DT with highly varying local distances, and curved paths. The shortest route algorithm presented in the next chapter was implemented using the SLT, but the priority queue transformation produces the same results more efficiently.

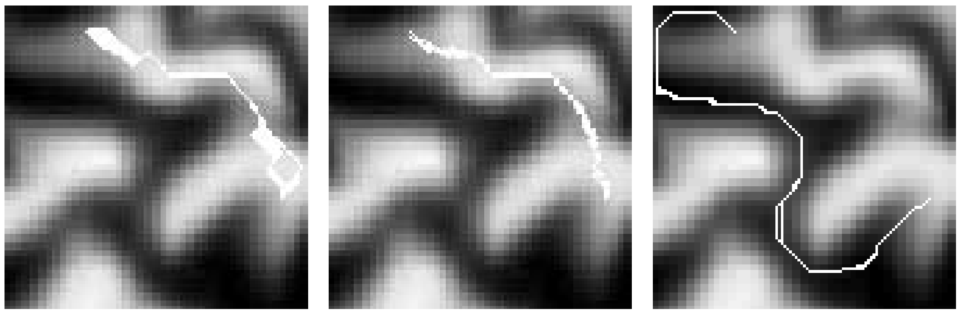
Chapter V

Shortest Routes on Surfaces

DTs generally produce only distance values, and the path along which the distance is calculated remains unknown. There can be more than one shortest path between two points even in the case of regular distance definitions. In the DTOCS, which does not follow the regularity principles, the number of equally short paths is quite unpredictable. For example, the shortest path between pixels with coordinates $(0, 0)$ and $(x, 0)$ can form a straight horizontal line, or consist of only diagonal steps and pass through pixel $(x/2, x/2)$, as in Figure 2.5, or combine both types of steps. The Route DTOCS algorithm developed in this work can be used to find and visualize all minimal paths between two points (*Publication II*, *Publication IV*) or between two point sets (*Publication III*). In addition, a distinct shortest path can be extracted from the route, which is defined as the set of points on any minimal path. The shortest routes found by the Route DTOCS correspond to minimal geodesics defined in geography, that is, they are approximations of shortest routes along varying height surfaces, like terrains.

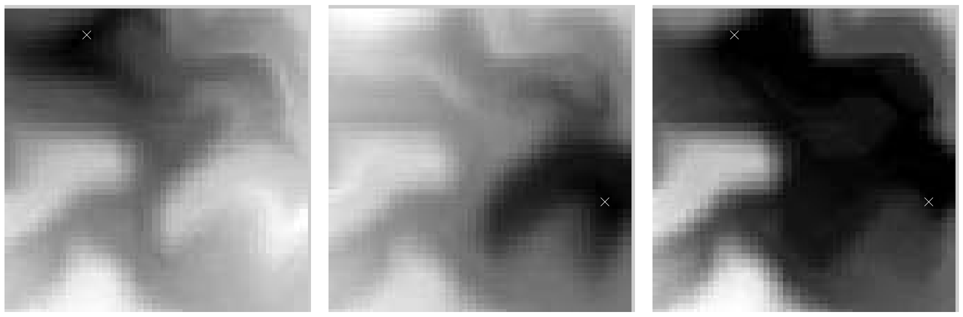
5.1 Route DTOCS

The Route DTOCS algorithm for calculating shortest routes along varying height surfaces, first introduced in [23], and developed further in *Publication II*, *Publication III* and *Publication IV*, is based on calculating two distance maps, one for each end-point or end-point set of the route. The idea of combining two distance maps to find the shortest route between two points has previously been used both in the DT world [61], and in the world of level sets [26]. Approaches utilizing modifications of the Dijkstra graph search are presented, for example, in [27], [51]. If local distances are defined using gray-level sums rather than differences, a minimal cost path is found. In such DTs, like the GRAYMAT [34] and the geodesic time [61], the gray-value represents the cost of traversing the pixel. The minimal cost paths, which follow low gray-values in the image, can be used for example to find faint linear structures in noisy images [81]. A comparison between DTOCS, WDTOCS and GRAYMAT routes shown in Figure 5.1 illustrates the difference between minimal cost paths and shortest routes along surfaces. Figure 4 in *Publication IV* shows a similar example.



(a) DTOCS route (b) WDTOCS route (c) GRAYMAT route

Figure 5.1: Shortest routes between two points based on the distance along the surface (DTOCS and WDTOCS), and the minimal cost distance (GRAYMAT).



(a) Distance image $\mathcal{F}_a^*(x)$ (b) Distance image $\mathcal{F}_b^*(x)$ (c) Route distance image $\mathcal{D}_R(x)$

Figure 5.2: The intermediate steps in the Route DTOCS algorithm producing the DTOCS route shown in Figure 5.1 (a).

The Route DTOCS algorithm is based on the fact that a minimal path consists of minimal sub-paths. For each pixel x on a minimal path between pixels a and b, the sub-paths between a and x, and between b and x are minimal as well (see Lemma 1. in [40]). To find the shortest route, the lengths of all sub-paths starting from the end-points, a and b, are determined by calculating two DTs, $\mathcal{F}_a^*(x)$ and $\mathcal{F}_b^*(x)$. The two DTs used to produce the DTOCS route in Figure 5.1 (a) are visualized in Figures 5.2 (a) and (b). In the resulting distance images, the value of pixel x corresponds to the length of the shortest path between x and the reference pixel a (resp. b). The sub-paths themselves are not known, but knowing their lengths is enough. Summing the two distance images produces a route distance image $\mathcal{D}_R(x)$, as in Figure 5.2 (c), where the value of pixel x is the sum of the lengths of the sub-paths between a and x, and b and x. Consequently, the value of x is the length of the shortest path between a and b, which passes through pixel x. This means that if point x has a minimal value in the route distance image, it lies on a minimal path between a and b. Thus, the route $\mathcal{R}(a,b)$ is defined as the set of points, for which the route distance value is minimal:

$$\mathcal{R}(a,b) = \{\, x \mid \mathcal{D}_R(x) = \min_x \mathcal{D}_R(x)\} \tag{5.1}$$

To summarize, the Route DTOCS algorithm is:

1. Calculate the distance image $\mathcal{F}_a^*(x)$ from the source point a
2. Calculate the distance image $\mathcal{F}_b^*(x)$ from the destination point b
3. Calculate the route distance image $\mathcal{D}_R(x) = \mathcal{F}_a^*(x) + \mathcal{F}_b^*(x)$
4. Mark pixels x with $\mathcal{D}_R(x) = \min_x \mathcal{D}_R(x)$ as pixels on the shortest route $\mathcal{R}(a,b)$

The correctness of the Route DTOCS algorithm can be proven based on the triangle inequality, that is, $d(a,b) \leq d(a,x) + d(x,b)$, as the DTOCS distance d is a metric. The equality holds only for pixels x, which lie on a minimal path between a and b. The pixels sharing the minimal value in the route distance image, Figure 5.2 (c), are marked as route pixels in Figure 5.1. Instead of finding the absolute minimum, a threshold can be used to find the points, which have a sufficiently low route distance value. This is necessary when using the WDTOCS or the Optimal DTOCS, where the calculation accuracy of floating point operations becomes a factor. The Route DTOCS can also be defined for routes between sets of points. As DTs calculate distances to the nearest reference, the two sub-path distance images can be calculated starting from point sets A and B. In distance image $\mathcal{F}_A^*(x)$ the value of pixel x corresponds to the length of the shortest path between x and the pixel in set A, which is nearest to x according to the distance definition used. The route set then contains the pixels, which lie on any minimal path between the route end-point sets.

The routes are typically wide, that is, there can be several minimal paths between two points, and the width can increase even further in the case of finding routes between sets. If there are equally short paths on both sides of an obstacle in the gray-level surface, the route can appear to consist of two or more separate routes. The experiments measuring route lengths across a gray-scale half sphere in *Publication II* (Fig. 5) and *Publication IV* (Fig. 6) demonstrate this effect, as the routes lie symmetrically on both sides of the sphere. The minimal route between sets may reach several points in the end-point sets, that is, minimal paths can exist between more than one pair of points. An alternative method for finding the shortest path between sets is an exhaustive search for the shortest path from each point in set A, and from each point in set B. If the end-point sets are large, an all-pairs algorithm may be more efficient than several shortest path searches from single sources. The complexity of the Route DTOCS algorithm compared to all-pairs approaches is discussed in *Publication III*.

The Route DTOCS is mainly just a visualization tool, that is, it can be used to visualize the shortest routes, but not directly to obtain a chain coded shortest path. If a distinct path is needed, it must be extracted from the route set. The simple extraction algorithm described in *Publication III* and *Publication IV* is based on backtracking towards decreasing distance values from point b in distance image $\mathcal{F}_a^*(x)$, or from point

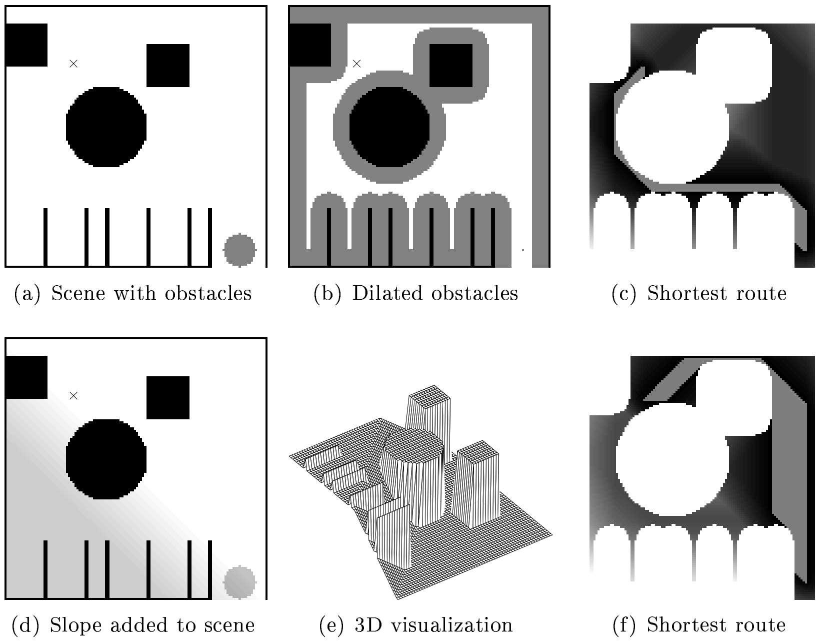
a in distance image $\mathcal{F}_b^*(x)$. In binary distance transforms the backtracking can proceed recursively from the current path pixel to the neighbor with the smallest distance value. In other words, it can follow the direction of the steepest descending slope of the distance function. Due to the varying local distances in the DTOCS, the neighbor with the smallest distance value may not belong to a minimal path. Thus, the route distance image is used as a reference, and the backtracking proceeds towards decreasing distance values within the route set. Alternatively, the backtracking can utilize the distance image, and the original image, to find an appropriate neighbor. The distance value of the neighbor must be smaller than the distance value of the current pixel. In addition, the local distance between the pixels must match their distance value difference, in order for the pixels to lie on the same minimal path.

5.2 Path Planning and Obstacle Avoidance

The most obvious applications for the Route DTOCS algorithm are in path planning and obstacle avoidance. Shortest routes along a varying height terrain may be needed in planning permanent constructions, like railroads and highways, or in hiking and orienteering. In different types of scenes, like city maps or factory floor plans, obstacles can be marked with very high values. In finding the fastest route through a city, some lower level obstacles, like locations of traffic lights, can be marked as bumps on otherwise smooth roads. Constrained distance transforms, which are otherwise applicable for path planning, can not handle different levels of obstacles. The Route DTOCS can be used to implement flexible obstacle avoidance, even in scenes, where the accessible areas are of varying height.

Figure 5.3 shows a synthetic example, where a disk shaped robot navigates among obstacles in a room. The obstacles are first dilated using a structuring element having the same size and shape as the robot shown in the bottom right corner of Figures 5.3 (a) and (d). Dilating the obstacles to make sure that routes are wide enough to accommodate the moving object is known as the growing obstacles approach [35]. Alternatively, the free regions can be eroded, as in the 3D path planning approach by Shih and Wu [56], where the shortest path is traced from the Euclidean distance image calculated in the constrained 3D domain. Once the obstacles are dilated, paths can be planned for a single pixel representing the center of the robot. The shortest route to the destination marked with 'x' is found using the WDTOCS distance definition in the Route DTOCS algorithm. In the first case a constrained binary DT can be used as well, but the DTOCS approach shows its strength in the modified example. A slope is added to the floor of the room, as visualized in Figures 5.3 (d) and (e). A different shortest route is found, as following the flat surface around the obstacles yields a shorter path than climbing up and down the slope. If it is more convenient to introduce the additional cost factors using pixel values rather than differences, the route method can be implemented with a gray-level DT calculating minimal cost paths, as in [61]. DTOCS distances along an area of constant gray-values are the same as geodesic time distances along an area with gray-value 1, so in a scene consisting of obstacles and smooth paths, the same routes can be found with both DT approaches, if the height map is modified accordingly.

The route distance image, from which the shortest route is extracted, contains the route lengths for all pixels in the image, that is, the route distance value of pixel x corresponds

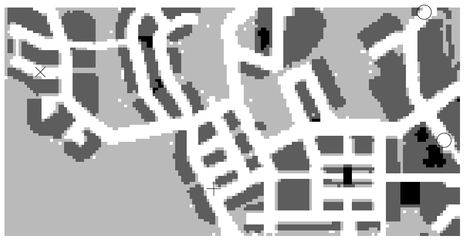

(a) Scene with obstacles (b) Dilated obstacles (c) Shortest route

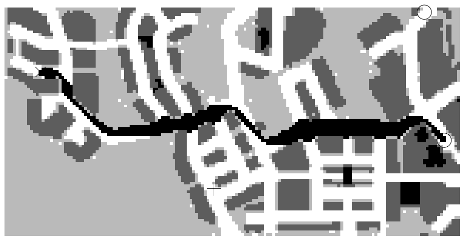
(d) Slope added to scene (e) 3D visualization (f) Shortest route

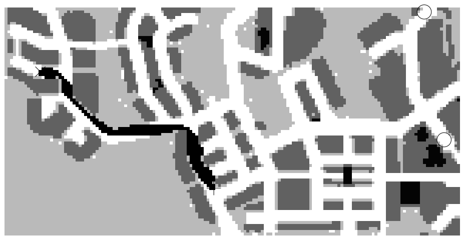
Figure 5.3: Robot navigation using Route DTOCS and dilated obstacles.

to the length of the shortest path from the source to the destination, which passes through point x. If the shortest route found by the algorithm is not applicable for some reason, an alternative route can be found by selecting an intermediate point with an acceptably low route distance value, and running the Route DTOCS algorithm twice, from the source to the intermediate point, and from the intermediate point to the destination. An example of such a detour route is shown in Figure 9 in *Publication IV*, and another in Figure 5.4, where the shortest route along a street map is found using the WDTOCS. Figure 5.4 (d) demonstrates clearly that there can be several equally short paths in the same route set. The destination of the route in Figure 5.4 was selected as the nearest of two destinations by using the set of alternative destination points as the route end-point set. Figure 3 in *Publication III* illustrates the method using three alternative destinations.

The route consists of pixels on any minimal path, and typically several alternative paths exist. The algorithm for extracting a single path described in *Publication III* and *Publication IV* uses the route set as a reference in making sure the backtracking proceeds in the right direction. The alternative approach of using the original image as a reference may be more flexible, as the need to produce the second distance map containing sub-path lengths from the destination is eliminated. For example, in a mobile robot application, one distance map can be calculated using the location of one or several docking stations as reference points. Then at any time, a path can be backtracked from the current location of the robot to its nearest docking station. In addition to planning paths for mobile robots in real space, DTs can be used more generally for robots, for example, for robot arms, to find accessible paths in the state space [74].


(a) Original street map


(b) Shortest route from source to destination


(c) Shortest route from source to intermediate point

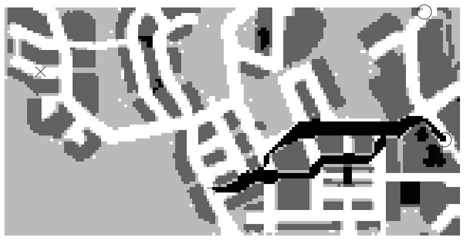

(d) Shortest route from intermediate point to destination

Figure 5.4: A path planning example, where the shortest route from the source 'x' to the nearest of two alternative destinations 'o', is replaced with an alternative route via a desired intermediate point '+'.

This chapter presented the Route DTOCS algorithm for finding and visualizing shortest routes on a gray-level surface, and listed some application ideas, where it can be used. If the surface, along which distances are calculated, represents a real surface, like a terrain height map, the routes approximate geodesics on the surface. Similar approaches utilizing other gray-level DTs find low-lying paths, as local distances are defined based on the gray-levels themselves, not their differences. If the Route DTOCS algorithm is applied to an image, which is not a height map, the route contains pixels on a minimal cost path. The distance in the horizontal plane is combined with the cost introduced by the gray-level differences, and the interpretation of the resulting route depends on the application.

Chapter VI

Applications

So far, the DTOCS has mostly been applied to problems, where the distance along the surface is used as a measure of surface variation. The first application, for which the DTOCS was originally developed [68], was image compression. The compression method codes the image by storing only a fraction of its pixels. The DTOCS and a distance threshold is used in order to concentrate more of these so called control points into areas with more gray-level variation. In the most recent application, which is still under development, the DTOCS is used to evaluate the roughness of a surface. The underlying idea is the same as in the compression method, that is, distance values are used as a measure of the amount of variation in the gray-level image. The amount of gray-level variation is assumed to correlate with the roughness of the surface the image represents.

6.1 Image Compression

Image compression aims at coding digital images using as few bits as possible. In lossless compression, an image identical to the original image must be obtained when decompressing the data. Lower storage size, and consequently higher compression ratios, can be achieved when some data loss is allowed. Lossy compression is an optimization problem with at least two conflicting objectives, as the image quality typically deteriorates with increasing compression ratios. Additional objectives can include minimizing the complexity of the compression and decompression algorithms. A lossy compression method utilizing the DTOCS is based on the idea of storing more information from image locations with more variation. Control points are stored from locations, where the difference between the DTOCS distance and the corresponding chessboard distance exceeds a predefined threshold. The gray-values and relative locations of the control points are coded using Huffman coding, or arithmetic coding [21]. The image is then reconstructed from the control points by interpolation. In the experiments in *Publication I*, linear interpolation based on a Delaunay triangulation is used, but in [66] a non-linear interpolation scheme is presented. The Delaunay triangulation is used only in the decoding, or reconstruction, whereas the fractal image compression algorithm presented in [19] uses Delaunay triangulation to find self-similar parts within an image for efficient encoding.

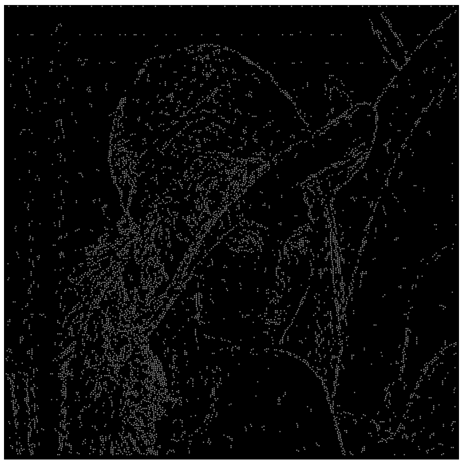
A detailed description and analysis of the DTOCS compression algorithm can be found in [68], which also includes various small improvements to the basic algorithm, for example, selecting control points only at locations with coordinates divisible by two, and storing patterns of control points instead of single control points. Both modifications attempt to store more information without increasing the number of bytes needed to encode the positions of the control points. In *Publication I*, the compression method was improved by selecting control points from the derivative image instead of the original image. Distances calculated along the original gray-level surface cause the control points to be placed at locations, where the gray-value has changed enough. The gray-values in the derivative image are calculated as the magnitude of the gradient $G_{MAG} = \sqrt{G_X^2 + G_Y^2}$, where G_X and G_Y are the horizontal and vertical derivative components. The gray-level difference used in the DTOCS local distance corresponds to the change in surface curvature, so control points are placed at locations, where the curvature changes.

Figure 6.1 illustrates the compression and reconstruction method. The control points selected by calculating distances on the original image are shown in Figure 6.1 (a), and the control points selected from the derivative image are shown in Figure 6.1 (c). The images reconstructed from the control points are shown in Figures 6.1 (b) and (d). The example is produced with a high distance threshold resulting in quite few control points, and the interpolation artefacts are clearly visible in the reconstructed images. Increasing the number of control points by decreasing the distance thresholds results in images of better quality, as visualized by the descending error curves in Figures 4-6 (a) in *Publication I.* The control points in Figures 6.1 (a) and (c) are concentrated at edges and other areas with high gray-level variation. This effect, which is the basic idea behind DTOCS compression, is intensified in the derivative method, enhancing the quality of the decompressed image, particularly in areas with rapid gray-level changes. Theoretical basis and experimental evidence for the improvement are given in *Publication I.*

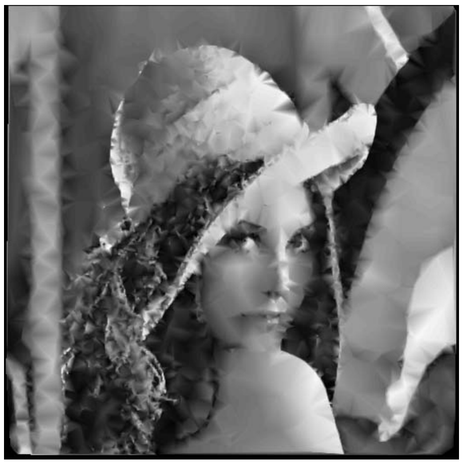
The implementation of the compression method utilizes the SLT algorithm, and boundaries, at which the distance value exceeds a given threshold, need to be extracted separately. Finding the boundary is trivial using the priority pixel queue algorithm developed in this work. As distance values propagate in priority order, the boundary consists of pixels, which obtain a value exceeding the threshold during the transformation. When a pixel is found to belong to the boundary, the propagation can be stopped by not enqueueing the pixel neighbors. Excess processing is eliminated, as the transformation terminates, when all boundary points have been found. Boundary tracing is needed only in the second phase of each iteration. A 1D distance transformation is performed along the boundary, and control points are placed at locations, where the difference between the DTOCS distance and the corresponding straight distance calculated from the previously selected control point exceeds a threshold. The new control points can immediately be enqueued into the priority queue as reference pixels for the next iteration.

6.2 Measuring Surface Roughness

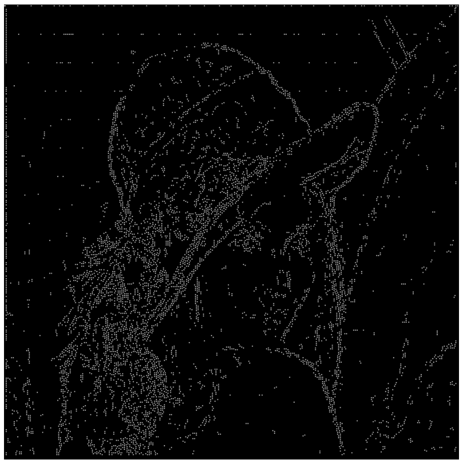
As DTOCS distances along a highly varying surface are larger than distances along a smooth surface, it is quite natural to try to use the DTOCS to evaluate surface roughness. The underlying idea is quite similar as in the image compression method. In fact, the number of control points found by the DTOCS compression algorithm could be used



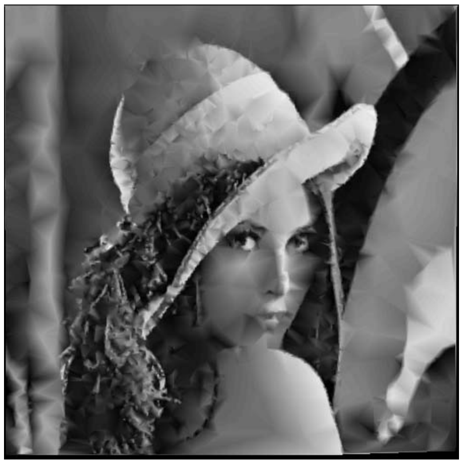
(a) Control points from original



(b) Reconstructed image (original)

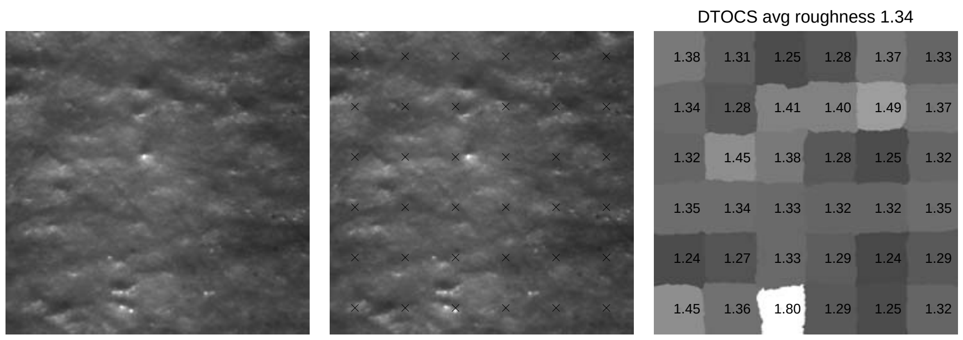


(c) Control points from derivative



(d) Reconstructed image (derivative)

Figure 6.1: Example of DTOCS image compression. Image (b) is reconstructed from the 6944 control points shown in image (a), and image (d) from the 6847 control points shown in image (c). The original image contains 512×512 pixels, so only about 2.6 % of the gray-values plus their relative locations are stored in the compressed images.



(a) Paper surface image (b) Seeds for DT and NNT (c) Roughness map

Figure 6.2: A microscopic image of a paper surface, the seed point grid used in the roughness measurement method, and the resulting DTOCS roughness map.

as an estimate on how much variation there is in the image. The idea behind the surface roughness measurement method, which is still under development, is introduced in *Publication VI*. A combination of the DT and the NNT is used to obtain estimates of surface roughness both globally, and locally within regions of the image, as illustrated in Figure 6.2. First, a grid of seed points is selected, as in Figure 6.2 (b), from which the distance transformation and the corresponding nearest neighbor transformation are initiated. Region borders appear where the propagating fronts meet. Within each nearest neighbor region, the distance values are normalized with, for example, the chessboard distance to the seed of the region. The larger the normalized distance values are, the more variation there is inside the region. The average of the normalized distances in a region estimates the roughness locally, and a roughness map can be visualized by coloring each region with its roughness value, as in Figure 6.2 (c). The average of all normalized distances can be used as an estimate of the global roughness.

The method can be applied to microscopic images of paper, and also on profilometer data representing the surface relief of the paper. The profilometer data is of unequal resolution, so the generalization of the DTOCS or the WDTOCS for anisotropic grids is needed. Better known approaches for measuring the roughness of paper include statistical analysis, like kurtosis [24], and analysis based on the fractal dimension [31]. The main problem is that roughness properties differ greatly between different paper types. The DTOCS seems to measure small scale roughness quite well, but waviness, which is roughness of larger scale, has a clearly smaller effect on the distance values measured along the surface, as distance values along a smooth slope are only slightly larger than distances along a flat plane. This means that the average of normalized DTOCS distances can not be used as the only measure for roughness. For the method to be useful in the paper industry, the DTOCS roughness measure needs to be combined with cost factors designed to extract waviness properties. Only the idea for the method has been published (*Publication VI*), and results will be presented in future work.

Chapter VII

Discussion

The basis for this thesis was the Distance Transform on Curved Space, and the objective was to analyze and improve the transform, and to find applications for it. The starting point was the method, as opposed to research projects, which start with a problem, and aim towards developing a method to solve it. The original plan was to focus on applications, but the topic evolved towards theoretical advances.

The gray-level differences, which are used in the DTOCS local distance definition, do not necessarily have to correspond to differences in altitude. In terrain navigation, difficult areas, like swamps, can be marked with higher values. In obstacle avoidance problems, like the labyrinth application in *Publication II* and *Publication IV*, one option is to scale the original image so that obstacles are high enough, but another is to scale the gray-level difference used in the local distance definition so that the high cost of moving from the legal path to an obstacle prevents illegal shortcuts. The local distances can be redefined to contain a cost factor, like energy expenditure [51]. Steep slopes can introduce a higher cost than the same distance along gradually shifting planes. In the DTOCS such a cost factor could be included by redefining the local distance using a non-linear function, like the squared gray-level difference. In the priority pixel queue algorithm, where the propagation order coincides with the paths, different costs could be defined for upward and downward pointing local distances. The only restrictions are that local distances must be non-negative for the algorithm to work, and strictly positive to be considered metrics. Extensions to 3D images may be considered in future work, if an application is found, where the gray-level difference between neighbor voxels is meaningful. A minimal cost path according to some other gray-level DT in 3D space could, for example, be used to find the path with the lowest density through a 3D object, where the value of a voxel indicates its density. If the DTOCS was used in a similar way, the path with the least density variation would be found.

The publications included in this thesis are in chronological order. *Publication I* provides a link to the history of the DTOCS research, as the transform was originally developed for image compression [68]. The experiments only demonstrate how one version of the DTOCS compression algorithm improves when the derivative image is used as input instead of the original image. No comparison to other compression methods was made. Even if the compression algorithm does not lead to a real application, the idea behind it might be of interest in some other image processing problems, for example, detecting image areas with high variation. The basic idea of comparing the curved DTOCS distance with the corresponding straight distance could be useful in measuring features from range images in various applications, for example, in face recognition. The curved distance between the inner corners of the eyes measured from a 3D model of a face represented as a height map could provide some information about the profile of the nose. The possibility of using the DTOCS in such applications will be explored in future work.

Publication II describes the Route DTOCS algorithm originally presented in [23], and redefines it with the WDTOCS to produce more accurate approximations of distances and routes. The Optimal DTOCS defined in *Publication IV* produces the most accurate distance approximations found so far. To further improve them, methods used for length estimation of digital curves could be applied. A recent evaluation of length estimators of digital curves in 2D can be found in [14], and some of the estimators can be extended to 3D. Paths in the DTOCS setting are essentially 3D curves, even though they are represented in 2D. Ideas presented for obstacle avoidance and robot navigation will be implemented and tested in the future. A distance transformation using the destination or alternative destinations of the robot as reference pixels, produces a map, where the distance to the nearest destination is known in each location. The path to the destination can be found by backtracking on the distance map.

The idea to use DTs to find routes between point sets is not as new as *Publication III* insinuates. In [81], the minimal cost path between two regions of an image is computed to find faint linear features running from one region to the other. However, *Publication III* demonstrates how the scaling of the original image can affect the paths, and contains discussion about the complexity of the sequential distance transformation, which is relevant for the later work presented in *Publication V*.

Publication V presents the priority pixel queue algorithm for calculating the DTOCS. Propagating distances in priority order is not a new idea, but gray-level distance transformations implemented using a minimum heap have not been found in the literature. In [28], distances along digitized 3D surfaces represented as voxel images are calculated using a minimum heap, but the advantage of using priority ordering instead of ordered propagation is probably quite small in binary images, where local distances vary only based on the propagation direction within the neighborhood. In the DTOCS and other gray-level DTs, where local distances vary considerably, the minimum heap provides an efficient, and a very flexible approach, as it can handle any distances, including floating point values. *Publication V* also includes experiments on the convergence properties of the sequential transformation, and the ordered propagation. It is well known that transforming non-convex domains, like gray-level surfaces, requires several iterations of the sequential transform, but no records on how high the iteration number can become have been found. The experiments made for *Publication V* demonstrate a clear trend, that is, the number of iterations grow both with the resolution of the image and with the complexity of the surface, and tens or hundreds of iterations may be needed. The ordered propagation was demonstrated to be applicable for smooth and simple surfaces, but clearly inferior to the priority pixel queue in the case of complex images.

The priority pixel queue distance transformation is very similar to the fast marching algorithm for calculating forward propagating level sets, and it would be interesting to make a comparison between DTs and evolving interfaces defined within the mathematical framework of level sets described, for example, in [54]). It is clear that there are many similarities, but DTs are defined directly for the discrete geometry inherent for digital image processing, so the calculations are simpler. In [75], a straightforward connection between wavefronts and DTs is found by showing that the gray-weighted distance transform can be used to approximate the Eikonal equation, which is a core concept in level set theory. The speed of the propagating front is used in the calculation of level sets, and the local distance used in the DTOCS can be viewed as the inverse of the speed, since large local distances slow down the propagation.

In *Publication VI*, the priority pixel queue transformation algorithm is extended to produce the NNT simultaneously with the distance map. Discrete approximations of Voronoi or Dirichlet tessellations have been done for binary images in 2D and 3D, but the DTOCS divides the image to regions in a new way. Possible applications for NNTs of gray-level images could be in cost minimization. For example, an area of varying height terrain could be divided into regions surrounding two or more support points, so that maintenance tasks could be assigned to the nearest support point. If the division should be based on actual distances along the terrain, the DTOCS NNT should be used. If the pixel values represent the cost of traversing the terrain, a NNT based on calculating minimal cost paths should be used instead. Some applications for NNTs of gray-level surfaces might be found in the telecommunications field, like planning the locations of base stations for mobile communication.

To summarize, the contributions of this thesis are as follows:

- The Route DTOCS algorithm for finding shortest routes on surfaces.
- The new variations of the DTOCS with more accurate local distance definitions: the $\sqrt{2}$-DTOCS, the 3-4-DTOCS and the Optimal DTOCS.
- The generalization of the DTOCS and the WDTOCS to anisotropic grids.
- The new efficient propagation algorithm based on a priority pixel queue.
- The experimental analysis on the convergence of the sequential and ordered propagation algorithms in complex domains.
- The nearest neighbor transform based on distances along varying height surfaces.

Hopefully, one result of this thesis is taking gray-level DTs another step closer to real applications in modern image processing. Most of the basic research in the field has been done in the 1980s, when computers possessed only a fraction of the computation power available today. Any home computer can now be used to transform large images in little time. High resolution images can be processed, and large distance values, or distance values of floating point accuracy, present no problems. Even the input image can contain floating point values, like profilometer data, instead of gray-levels, so the scope of possible applications widens. The DTOCS and the WDTOCS generalized to anisotropic grids need to be refined using more accurate distance definitions if they are to be used for approximating real distances along surfaces. In measuring surface roughness, the coarse approximation is probably sufficient, as the distance values are used only to estimate the amount of variation in the surface. The research on using the DTOCS and its modifications to estimate the surface roughness of paper based on profilometer data or microscopic images of the paper surface is currently going on, and future work will focus on finding more applications.

Chapter VIII

Errata

The word "noise" has been left out in all three figure captions presenting signal to noise ratios of reconstructed images in *Publication I*.

The reference list in *Publication II* contains the article "Salience Distance Transforms" by Rosin and West [49], which is not cited in the text.

In *Publication IV* the concept of Geodesic Time is described as the sum of gray-values along a path. Even though the inventor Pierre Soille himself uses that definition in the abstract of [61], the exact definition of the path length t_f is the sum of the mean of the gray-values taken two at a time along the path $\mathcal{P}$:

$$t_f(\mathcal{P}) = \sum_{i=1}^{l} \frac{f(p_{i-1}) + f(p_i)}{2} = \frac{f(p_0)}{2} + \frac{f(p_l)}{2} + \sum_{i=1}^{l-1} f(p_i)$$

Reference [9] in *Publication IV* is by G. Borgefors (erroneous initial).

Bibliography

[1] Adams, R., and Bischof, L. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence 16*, 6 (1994), 641–647.

[2] Arcelli, C., and Sanniti di Baja, G. Computing Voronoi diagrams in digital pictures. *Pattern Recognition Letters 4*, 4 (1986), 383–389.

[3] Beckers, A. L. D., and Smeulders, A. W. M. A comment on "A Note on 'Distance Transformations in Digital Images'". *Computer Vision, Graphics, and Image Processing 47* (1989), 89–91.

[4] Beckers, A. L. D., and Smeulders, A. W. M. Optimization of length measurements for isotropic distance transformations in three dimension. *CVGIP: Image Understanding 55*, 3 (1992), 296–306.

[5] Borgefors, G. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics, and Image Processing 27* (1984), 321–345.

[6] Borgefors, G. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing 34* (1986), 344–371.

[7] Borgefors, G. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence 10*, 6 (1988), 849–865.

[8] Borgefors, G. Another Comment on "A Note on 'Distance Transformations in Digital Images'". *CVGIP: Image Understanding 54*, 2 (1991), 301–306.

[9] Borgefors, G. Applications using distance transforms. In *Aspects of Visual Form Processing*, C. Arcelli, L. Cordella, and G. Sanniti di Baja, Eds. World Scientific, Singapore, 1994, pp. 83–108.

[10] Borgefors, G. On digital distance transforms in three dimensions. *Computer Vision and Image Understanding 64*, 3 (1996), 368–376.

[11] Borgefors, G., Hartmann, T., and Tanimoto, S. L. Parallel distance transforms on pyramid machines: theory and implementation. *Signal Processing 21*, 1 (1990), 61–86.

[12] BREU, H., GIL, J., KIRKPATRICK, D., AND WERMAN, M. Linear time Euclidean distance transform algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence 17*, 5 (1995), 529–533.

[13] CHEW, L. P., AND DYRSDALE, R. L. Voronoi diagrams based on convex distance functions. In *First Annual Symposium on Computational Geometry* (Baltimore, Maryland, 1985), ACM Press, pp. 235–244.

[14] COEURJOLLY, D., AND KLETTE, R. A comparative evaluation of length estimators of digital curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 2 (2004), 252–258.

[15] COQUIN, D., AND BOLON, P. Discrete distance operator on rectangular grids. *Pattern Recognition Letters 16*, 9 (1995), 911–923.

[16] CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. *Introduction to Algorithms.* MIT Press, 1990.

[17] CUISENAIRE, O., AND MACQ, B. Fast Euclidean distance transformation by propagation using multiple neighborhoods. *Computer Vision and Image Understanding 76*, 2 (1999), 163–172.

[18] DANIELSSON, P.-E. Euclidean distance mapping. *Computer Graphics and Image Processing 14* (1980), 227–248.

[19] DAVOINE, F., ANTONINI, M., CHASSERY, J.-M., AND BARLAUD, M. Fractal image compression based on Delaunay triangulation and vector quantization. *IEEE Transactions of Image Processing 5*, 2 (1996), 338–346.

[20] FOUARD, C., AND MALANDAIN, G. 3-D chamfer distances and norms in anisotropic grids. *Image and Vision Computing 23*, 2 (2005), 143–158.

[21] FRYDRYCH, M., AND TOIVANEN, P. Arithmetic coding with sliding window for control-point based image compression. In *International Conference on Image Processing and its Applications (IPA)* (1999), pp. 601–604.

[22] IKONEN, L., KÄLVIÄINEN, H., AND OINONEN, O. A computer vision approach for robotized handling of sheets in a manufacturing cell. In *Scandinavian Conference on Image Analysis (SCIA)* (Lappeenranta, Finland, 1997), M. Frydrych, J. Parkkinen, and A. Visa, Eds., pp. 309–315.

[23] IKONEN, L., TOIVANEN, P., AND TUOMINEN, J. Shortest route on gray-level map using Distance Transform on Curved Space. In *Scandinavian Conference on Image Analysis (SCIA)* (Gothenburg, Sweden, 2003), J. Bigun and T. Gustavsson, Eds., pp. 305–310.

[24] JOHANSSON, J.-O. Measuring homogenity of planar point-patterns by using kurtosis. *Pattern Recognition Letters 21*, 13-14 (2000), 1149–1156.

[25] KAPOUTSIS, C. A., VAVOULIDIS, C. P., AND PITAS, I. Morphological iterative closest point algorithm. *IEEE Transactions on Image Processing 8*, 11 (1999), 1644–1646.

[26] Kimmel, R., Amir, A., and Bruckstein, A. M. Finding shortest paths on surfaces using level sets propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 17*, 6 (1995), 635–640.

[27] Kimmel, R., and Kiryati, N. Finding shortest paths on surfaces by fast global approximation and precise local refinement. *International Journal of Pattern Recognition and Artificial Intelligence 10* (1996), 643–656.

[28] Kiryati, N., and Székely, G. Estimating shortest paths and minimal distances on digitized three-dimensional surfaces. *Pattern Recognition 26*, 11 (1993), 1623–1637.

[29] Kiselman, C. O. Regularity properties of distance transformations in image analysis. *Computer Vision and Image Understanding 64*, 3 (1996), 390–398.

[30] Kulpa, Z., and Kruse, B. Algorithms for circular propagation in discrete images. *Computer Vision, Graphics, and Image Processing 24* (1983), 305–328.

[31] Kuparinen, T., Rodionov, O., Toivanen, P., Mielikainen, J., Bochko, V., Korkalainen, A., Parviainen, J., and Vartiainen, E. Fractal dimension analysis and statistical processing of paper surface images towards surface roughness measurement. In *Scandinavian Conference on Image Analysis (SCIA)* (Joensuu, Finland, 2005), H. Kalviainen, J. Parkkinen, and A. Kaarna, Eds., pp. 1218–1227.

[32] Lantuejoul, C., and Maisonneuve, F. Geodesic methods in quantitative image analysis. *Pattern Recognition 17*, 2 (1984), 177–187.

[33] Lee, T. D., and Preparata, F. P. Computational geometry - a survey. *IEEE Transactions on Computers C-33*, 12 (1984), 1072–1101.

[34] Levi, G., and Montanari, U. A grey-weighted skeleton. *Information Control 17* (1970), 62–91.

[35] Lin, P. L., and Chang, S. A shortest path algorithm for a nonrotating object among obstacles of arbitrary shapes. *IEEE Transactions on Systems, Man, and Cybernetics 23*, 3 (1993), 825–833.

[36] Maurer, C. R., Qi, R., and Raghavan, V. A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25*, 2 (2003), 265–270.

[37] Montanari, U. A method for obtaining skeletons using a quasi-Euclidean distance. *Journal of the Association for Computer Machinery 15*, 4 (1968), 600–624.

[38] Nyström, I., Borgefors, G., and Sanniti di Baja, G. 2D grey-level convex hull computation: A discrete 3D approach. In *Scandinavian Conference on Image Analysis (SCIA)* (Gothenburg, Sweden, 2003), J. Bigun and T. Gustavsson, Eds., pp. 763–770.

[39] Paglieroni, D. W. Distance transforms: Properties and machine vision applications. *CVGIP: Graphical Models and Image Processing 54*, 1 (1992), 56–74.

[40] Piper, J., and Granum, E. Computing distance transformations in convex and non-convex domains. *Pattern Recognition 20*, 6 (1987), 599–615.

[41] Preteux, F. On a distance function approach for gray-level mathematical morphology. In *Mathematical Morphology for Image Processing*, E. R. Dougherty, Ed. Marcel Dekker, Inc., 1993, pp. 323–349.

[42] Ragnemalm, I. Contour processing distance transforms. In *5th International Conference on Image Analysis and Processing (ICIAP)* (Positano, Italy, 1989), World Scientific, pp. 204–212.

[43] Ragnemalm, I. Neighborhoods for distance transformations using ordered propagation. *CVGIP: Image Understanding 56*, 3 (1992), 399–409.

[44] Ragnemalm, I. The Euclidean distance transform in arbitrary dimensions. *Pattern Recognition Letters 14* (1993), 883–888.

[45] Rosenfeld, A., and Kak, A. C. *Digital Picture Processing*, second ed., vol. 2. Academic Press, 1982.

[46] Rosenfeld, A., Kong, T. Y., and Wu, A. Y. Digital surfaces. *CVGIP: Graphical Models and Image Processing 53*, 4 (1991), 305–312.

[47] Rosenfeld, A., and Pfaltz, J. L. Sequential operations in digital picture processing. *Journal of the Association for Computing Machinery 13*, 4 (1966), 471–494.

[48] Rosenfeld, A., and Pfaltz, J. L. Distance functions on digital pictures. *Pattern Recognition 1* (1968), 33–61.

[49] Rosin, P. L., and West, G. A. W. Salience distance transforms. *Graphical Models and Image Processing 57*, 6 (1995), 483–521.

[50] Rutovitz, D. Data structures for operations on digital images. In *Pictorial Pattern Recognition*. Thompson Book, Washington, 1968, pp. 105–133.

[51] Saab, Y., and VanPutte, M. Shortest path planning on topographical maps. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans 29*, 1 (1999), 139–150.

[52] Saha, P. K., Wehrli, F. W., and Gomberg, B. R. Fuzzy Distance Transform: Theory, algorithms, and applications. *Computer Vision and Image Understanding 86* (2002), 171–190.

[53] Schouten, T., and van den Broek, E. Fast Exact Euclidean Distance (FEED) Transformation. In *17th International Conference on Pattern Recognition (ICPR)* (Cambridge, UK, 2004), J. Kittler and M. Petrou, Eds., pp. 594–597.

[54] Sethian, J. A. *Level Set Methods and Fast Marching Methods*, 2nd ed. Cambridge University Press, 1999.

[55] Shih, F. Y., and Wu, Y.-T. Fast Euclidean distance transformation in two scans using a 3 x 3 neighborhood. *Computer Vision and Image Understanding 93* (2004), 195–205.

[56] Shih, F. Y., and Wu, Y.-T. Three-dimensional Euclidean distance transformation and its application to shortest path planning. *Pattern Recognition 37* (2004), 79–92.

[57] Silvela, J., and Portillo, J. Breadth-first search and its application to image processing problems. *IEEE Transactions on Image Processing 10*, 8 (2001), 1194–1199.

[58] Sintorn, I.-M., and Borgefors, G. Weighted distance transforms in rectangular grids. In *International Conference on Image Analysis and Processing (ICIAP)* (Palermo, Italy, 2001), E. Ardizzone and V. Di Gesú, Eds., pp. 322–326.

[59] Sintorn, I.-M., and Borgefors, G. Weighted distance transforms for images using elongated voxel grids. In *Discrete Geometry for Computer Imagery (DGCI)* (Berlin, Germany, 2002), A. Braquelaire, J. Lachaud, and A. Vialard, Eds., pp. 244–254.

[60] Sladoje, N., Nyström, I., and Saha, P. K. Measurements of digitized objects with fuzzy borders in 2D and 3D. *Image and Vision Computing 23*, 2 (2005), 123–132.

[61] Soille, P. Generalized geodesy via geodesic time. *Pattern Recognition Letters 15*, 12 (1994), 1235–1240.

[62] Soille, P. *Morphological Image Processing: Principles and Applications*, second ed. Springer-Verlag, 2003.

[63] Sternberg, S. R. Grayscale morphology. *Computer Vision, Graphics, and Image Processing 35* (1986), 333–355.

[64] Strand, R., and Borgefors, G. Distance transforms for three-dimensional grids with non-cubc voxels. *Computer Vision and Image Understanding 100* (2005), 294–311.

[65] Thiel, E., and Montanvert, A. Chamfer masks: Discrete distance functions, geometrical properties and optimization. In *11th International Conference on Pattern Recognition (ICPR)* (The Hague, The Netherlands, 1992), vol. III, Conference C: Image, Speech and Signal Analysis, pp. 244–247.

[66] Toivanen, P. Image compression by selecting control points using Distance Function on Curved Space. *Pattern Recognition Letters 14* (1993), 475–482.

[67] Toivanen, P. The Euclidean Distance Transform on Curved Space (EDTOCS) with application to image processing. In *European Signal Processing Conference (EUSIPCO)* (Edinburgh, Scotland, 1994), M. Holt, C. Cowan, P. Grant, and W. Sandham, Eds., pp. 983–986.

[68] Toivanen, P. *New Distance Transforms for Gray-Level Image Compression.* PhD thesis, Lappeenranta University of Technology, 1996.

[69] Toivanen, P. New geodesic distance transforms for gray-scale images. *Pattern Recognition Letters 17* (1996), 437–450.

[70] Toivanen, P., Vepsäläinen, A., and Parkkinen, J. Image compression using distance transform on curved space (DTOCS) and Delaunay triangulation. *Pattern Recognition Letters 20* (1999), 1015–1026.

[71] Trucco, E., and Verri, A. *Introductory Techniques for 3-D Computer Vision.* Prentice Hall, 1998.

[72] van den Broek, E. L., Schouten, T. E., Kisters, P. M. F., and Kuppens, H. Weighted Distance Mapping (WDM). In *IEE International Conference on Visual Information Engineering (VIE)* (Glasgow, Scotland, 2005), pp. 157–164.

[73] van Vliet, L. J., and Verwer, B. J. H. A contour processing method for fast binary neighbourhood operations. *Pattern Recognition Letters 7*, 1 (1988), 27–36.

[74] Verbeek, P. W., Dorst, L., Verwer, B. J. H., and Groen, F. C. A. Collision avoidance and path finding through constrained distance transformation in robot state space. In *International Conference on Intelligent Autonomous Systems* (1986), T. Kanade, F. Groen, and L. Hertzberger, Eds., pp. 627–634.

[75] Verbeek, P. W., and Verwer, B. J. H. Shading from shape, the eikonal equation solved by grey-weighted distance transform. *Pattern Recognition Letters 11* (1990), 681–690.

[76] Verwer, B. J. H. Local distances for distance transformations in two and three dimensions. *Pattern Recognition Letters 12* (1991), 671–682.

[77] Verwer, B. J. H., van Vliet, L. J., and Verbeek, P. W. Binary and grey-value skeletons - metrics and algorithms. *International Journal of Pattern Recognition and Artificial Intelligence 7*, 5 (1993), 1287–1308.

[78] Verwer, B. J. H., Verbeek, P. W., and Dekker, S. T. An efficient uniform cost algorithm applied to distance transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11*, 4 (1989), 425–429.

[79] Vincent, L. Exact Euclidean distance function by chain propagations. In *Computer Vision and Pattern Recognition* (Maui, Hawaii, 1991), pp. 520–525.

[80] Vincent, L. New trends in morphological algorithms. In *Proc. SPIE/SPSE, Nonlinear Image Processing II* (San Jose, California, 1991), vol. 1451, pp. 158–170.

[81] Vincent, L. Minimal path algorithms for the robust detection of linear features in gray images. In *International Symposium on Mathematical Morphology* (Amsterdam, 1998), Kluwer, pp. 331–338.

[82] Vossepoel, A. M. A note on "Distance Transformations in Digital Images". *Computer Vision, Graphics, and Image Processing 43* (1988), 88–97.

Publications

Publication I

Ikonen, L., Kyrki, V., Toivanen, P., Kälviäinen, H.,
Image Compression Using Derivative Information with Distance Transforms

IMAGE COMPRESSION USING DERIVATIVE INFORMATION WITH DISTANCE TRANSFORMS

Leena Ikonen, Ville Kyrki, Pekka J. Toivanen[1]*, and Heikki Kälviäinen*
Dept. of Information Technology,
Lappeenranta University of Technology,
P.O. Box 20, 53851 Lappeenranta,
FINLAND
Tel: +358 5 6213448; fax: +358 5 6243456
e-mail: Pekka.Toivanen@lut.fi

ABSTRACT

In this paper, a new image compression method is presented using the Distance Transform on Curved Space (DTOCS) and derivative information in finding positions for control points. In previous work it has been shown that the control points are not in exactly optimal positions. This paper presents theoretical considerations according to which the new method enhances the decompressed image quality particularly in the areas of rapid changes. The obtained results shown verify the correctness of the theoretical considerations. The reconstructed image quality is clearly better measured by error criteria. Also visually the difference is significant.

1 INTRODUCTION

Image compression techniques are commonly divided into first and second generation methods [6]. The second generation methods can further be divided into two groups. The first group is characterized by the use of local operators. Pyramidal coding [1] and anisotropic nonstationary predictive coding [11] are the main examples of this groups of methods. The second group methods attempt to describe an image in terms of contour and texture. Directional decomposition-based coding [4] and segmentation-based coding [5] are two major examples of this second group of methods.

A recently introduced image compression method is based on the linear wavelet theory. With this method it is possible to obtain both time and space resolution at the same time giving better compression ratios than classical methods [2].

In [10] the min-max-Medial Axis Transfrom is used for image compression giving 1.0 and 2.5 bits per pixel for noisy chromosome images. Such rates are comparable with those typically obtained in interpolative and transform coding schemes.

[1]currently at
Linköping University
ITN / Campus Norrköping
601 74 Norrköping
Sweden

Commonly distance transforms are used in feature extraction in pattern matching and learning. Their use in image compression is very rare. The proposed gray-level image compression method is based on the use of the Distance Transform on Curved Space (DTOCS), which is a distance transform for gray-level images. The definition of the DTOCS and a sequential two-pass algorithm to calculate it is presented in [8]. Previously the DTOCS has been applied to gray-level image compression [9] with fairly good results. In [9], the DTOCS is used as part of a compression algorithm to find as optimal as possible locations for control points, i.e., points that are considered fundamental for the reconstruction of the image. The algorithm is applied directly to the original image. In this paper, the original image filtered with a derivative filter before the compression algorithm. giving somewhat better results thane those in [9].

This paper is divided as follows. Section 1 is an introduction on the subject. Section 2 presents the underlying theory of using derivative information with the DTOCS in compression algorithm. Section 3 presents the compression algorithm. Results are shown in section 4. Finally, section 5 is a conclusion.

2 THEORY

The Distance Transform on Curved Space (DTOCS) is a distance transform for gray-level images [8]. It can be utilized in control point based image compression [9]. The compression algorithm presented in [9] is essentially a 2-dimensional algorithm. However, the main problems in it, related to finding curves along which new control points are placed, are 1-dimensional. Therefore, and for simplicity, the following distance transform formulas are presented for a 1-dimensional case. The new method presented in this paper corrects the main problem also for the 2-dimensional algorithms.

Let $a \in Z$ and $b \in Z$. The DTOCS distance transform can be represented in one-dimension as

$$d_1(a,b) \quad = \quad \sum_{x=a+1}^{b} (\|f(x) - f(x-1)\| + 1) \qquad (1)$$

$$= \sum_{x=a+1}^{b} (\|f(x) - f(x-1)\| + b - a) \quad ,(2)$$

where $d(a,b)$ represents the distance between points a and b, and $f(x)$ is the signal to transform. Likewise, the chessboard distance transform can be defined as in [8], as the binary image distance transform on digital grid. Thus, the 1-dimensional function for the chessboard distance is

$$d_0(a,b) = \sum_{x=a+1}^{b} 1 = b - a \ . \quad (3)$$

The difference between the distances is thus

$$\delta(a,b,f) = d_1(a,b) - d_0(a,b) \quad (4)$$
$$= \sum_{x=a+1}^{b} (\|f(x) - f(x-1)\|) \ . \quad (5)$$

Equation 4 can be understood as the sum of the absolute values of discrete first derivatives from a to b, when the derivative of $f(x)$ is defined as

$$f'(x) = f(x) - f(x-1). \quad (6)$$

A similar distance transform can be defined for the derivative of the signal using the Equation 5. The derivative distance δ_1 is defined as

$$\delta_1(a,b,f) = D(a,b,f') \quad (7)$$
$$= \sum_{x=a+1}^{b} (\|f'(x) - f'(x-1)\|) \ , \quad (8)$$

where $f'(x)$ is defined as above.

The compression method based on the DTOCS tries to place the control points in a way to preserve maximal amount of image information in decompression, as the decompression is perfomed using linear interpolation. New control points are placed in such locations, where the value of the difference function δ is greater than some predefined threshold. See Figure 1.(a). The open circles denote the positions for the control points given by the existing algorithms [9]. No points are in the ideal bending points.

This problem can be solved by using the derivative distance function δ_1. Figure 1 presents an example of this approach. Figure 1.(a) shows an example signal, Figure 1.(b) is the derivative of the signal shown in Figure 1.(a). The black dots in Figure 1.(a) in the bending points of the signal denote the ideal positions for the control points from the reconstruction point of view, in which linear interpolation is used. Since the signal is monotonously increasing, its derivative is zero or positive. The control points of the compression are placed along the increasing part of the function slope with equal distances. It can be seen that the distance function responds to changes in the function value. If the points are placed using the derivative, the places are determined by the changes in the second derivative, i.e., the curvature of the signal. The new distance measure δ_1 is actually the cumulative sum of the absolute differences of the derivative. It is presented in Figure 1.(c), which shows that the only points of interest are those, where the value of the first derivative changes, i.e., the bending points of the original signal, marked with black dots in Figure 1.(a). This is unlike the original method, where the points are selected whenever the signal value is changing enough. Thus, in the new method, more points are devoted to areas, where the curvature changes rapidly. Theoretically, this should enhance the decompressed image quality, since the decompression is perfomed by linear interpolation.

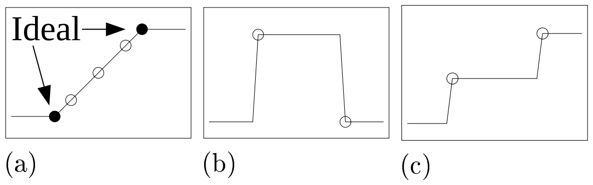


Figure 1. a) Input signal and original distance function, b) Derivative of the input, c) Distance function δ_1.

This approach of using the derivative as the base signal can be extended to two-dimensional images. In this study, the image was first filtered with 3×3 spatial Sobel filters presented in [3]. Next, the magnitude of the gradient was calculated using the formula $G_{MAG} = \sqrt{G_X^2 + G_Y^2}$, where G_{MAG} is the magnitude, G_X and G_Y are the corresponding horizontal and vertical components. The result was then used with the standard DTOCS compression algorithm to select the control points.

3 THE COMPRESSION ALGORITHM

Let us make the following definitions for the compression algorithm. Let $G(x,y)$ be the original gray-level image and $F(x,y)$ be a binary image which determines the area of calculation. Let $H(x,y)$ be the original image filtered using the derivative filter. In this paper, $F(x,y)$ = maximum integer if $G(x,y) > 0$, and $F(x,y) = 0$ otherwise. The found control points with their gray values are stored in $C = \{c_1, c_2, ..., c_N\}$. Let the chessboard distance be denoted by d_D and the distance generated by the DTOCS on image H be D_H, correspondingly.

The compression algorithm:

1. Select a proper threshold ϵ. Pick randomly some initial control points $c_i = (x_i, y_i, G(x_i, y_i))$ from the borders of the original image and $\forall c_i : c_i \leftarrow 0$. Put the picked points to $C = \{c_i\}$. $F \setminus C \leftarrow maxint$.

2. Filter the original image G using a suitable derivative filter to obtain the derivative image H.

3. Calculate the DTOCS and the chessboard binary distance transform. Then scan the images until a point is found for which $|D_H(x,y) - d_H(x,y)| > \epsilon$.
4. Resume scanning and form a curve R on those points for which $|D_H(x,y) - d_H(x,y)| \geq \epsilon$ holds. The curve R ends when a point $|D_H(x,y) - d_H(x,y) < \epsilon|$ is found.
5. Go along the curve R and calculate the DTOCS and the chessboard distance transform for each curve point in R. If $|D_H(x,y) - d_H(x,y)| > \epsilon$ put a control point to (x,y) in image C. Set the corresponding pixels in C to $G(x,y)$ and all the processed pixels in F to 0. Set all the other pixels in F to maximum integer. Goto 5 until the entire curve R has been handled.
6. Goto phase 3.
7. Code every control point using Huffman coding for both its relative distance from the previous point and for its gray value.

The decompression starts by unfolding the Huffman code. Delaunay triangles [7] are formed among the control points. The gray value of each point inside a triangle is calculated in the following way. Let $G(x)$ be the gray value of a pixel $x = (x_x, x_y)$ in the image. A, B, C are the three corner points of the Delaunay triangle and a, b, c are the three weights of pixel x in triangle ΔABC. We have a set of equations, Equations 9-12, from which $\mathcal{G}(x)$ can be calculated for all pixels $x = (x_x, x_y) \in \Delta ABC$:

$$\begin{aligned} x_x &= aA_x + bB_x + cC_x && (9) \\ x_y &= aA_y + bB_y + cC_y && (10) \\ a + b + c &= 1 && (11) \\ G(x) &= aG(A) + bG(B) + cG(C) && (12) \end{aligned}$$

4 RESULTS

The obtained results show that for all numbers of control points, i.e., for all compression ratios, the error criteria show lower values and the images look better than in previous work [9]. Figure 1.(a) shows the original Lena image of size $512 \times 512 \times 8$ bits. Figure 1.(b) shows the original airplane image and Figure 1.(c) the original peppers image of the same size. Figure 2.(a) depicts a decompressed Lena image which has been compressed using a standard DTOCS-based compression algorithm presented in [9]. It has 13459 control points. The signal-to noise ratio is 22.32 dB and the compression ratio is 1:25 using the star pattern presented in [9]. Without the stars with same number of points the ratio would be 1:17. Figure 2.(b) shows the same for the airplane image and Figure 2.(c) for the peppers image. Figure 3.(a) shows the decompressed Lena image using the derivative filter. Figures 3.(b) and 3,(c) show the same for the airplane and the peppers image.

In Figures 4-6 the dashed lines represent the results obtained with the original image without derivative filtering. The solid line represents the results with derivative filtering. Figure 4.(a) shows the mean average error versus number of control points for the Lena image. It can be seen that for normal amounts of control points the new method gives lower error. Figure 4.(b) depicts peak signal-to noise ratio. Also it is clearly higher with the new method. Visually the difference is significant, since the use of the derivative filter corrects many defects near the edges of the image. This can be seen when comparing Figures 2.(a) and 3.(a), Figures 2.(b) and 3.(b), and Figures 2.(c) and 3.(c). Figure 5.(a) shows the mean average error and Figure 5.(b) the peak signal-to noise ratio for the airplane image. In the airplane image, the new method reproduces little details better than the original method. Finally, Figure 6.(a) depicts the mean average error and Figure 6.(b) depicts the peak signal-to noise ratio for the peppers image.

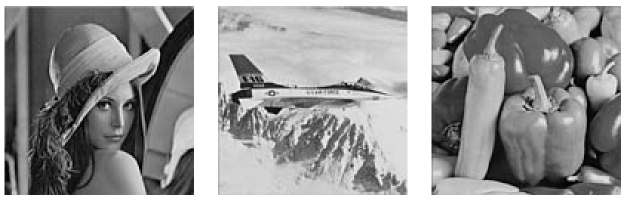

(a) (b) (c)

Figure 1. The original images.

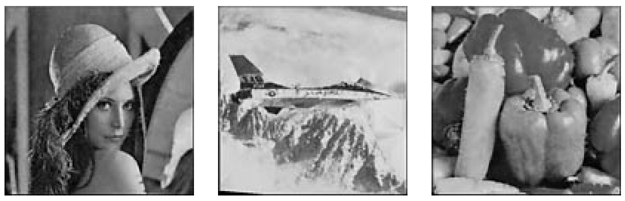

(a) (b) (c)

Figure 2. Decompressed images without derivative filtering.

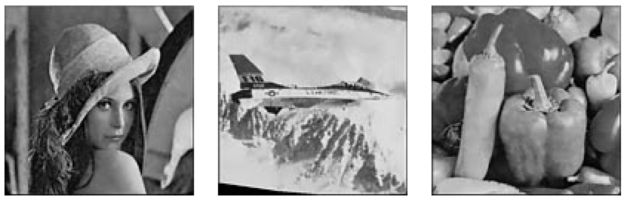
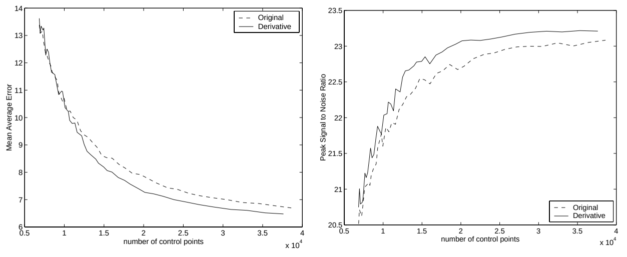
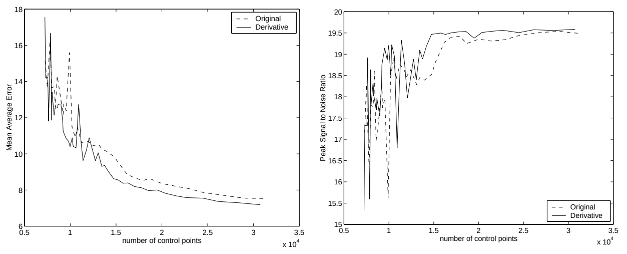

(a) (b) (c)

Figure 3. Decompressed images with derivative filtering.



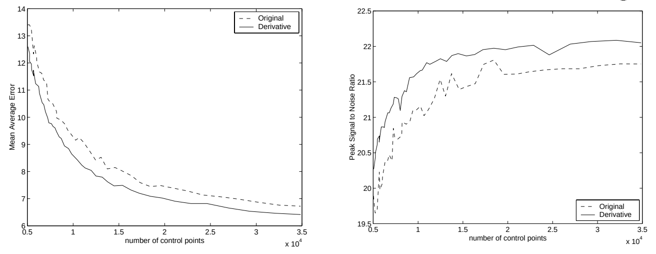
(a) (b)

Figure 4.(a) Mean average error,(b) Peak signal to ratio vs. number of control points for the Lena image.



(a) (b)

Figure 5.(a) Mean average error,(b) Peak signal to ratio vs. number of control points for the airplane image.



(a) (b)

Figure 6.(a) Mean average error,(b) Peak signal to ratio vs. number of control points for the peppers image.

5 CONCLUSION

In this paper, a new image compression method is presented. It is based on the use of derivative information together with the Distance Transform on Curved Space (DTOCS) and the chessboard distance transform. The compression algorithm is applied to an image which has first been filtered by a derivative filter. The underlying theory is presented in analytical form, according to which the decompressed image quality is enhanced particularly in the areas of rapid changes. The improvement of the reconstructed image quality is clear with all compression ratios. This is verified by calculating peak signal to noise ratios and mean average errors, and by visual inspection.

References

[1] P. J. Burt and E. H. Adelson, "The Laplacian Pyramid as a Compact Code", *IEEE Transactions on Communication*, pp. 532-540, 1983.

[2] R. A. DeVore, B. Jawerth and B. J. Lucier, "Image Compression Through Wavelet Transform Coding", *IEEE Transactions in Information Theory*, Volume 38, No. 2, March 1992.

[3] Rafael C. Gonzales, and Richard E. Woods: Digital Image Processing. Addison-Wesley Publishing Company, Inc. 1993.

[4] A. Ikonomopoulos and M. Kunt, "High compression image coding via directional filtering", *Signal Processing 8 (2)*, pp. 179-203, 1985.

[5] M. Kocher and M. Kunt, "Image data compression by contour-texture modelling", in Proceedings of the *SPIE International Conference on the Applications of Digital Image Processing*, Geneva, Switzerland, pp. 131-139, April 1983.

[6] M. Kunt, M. Benard and R. Leonardi, "Recent Results in High-Compression Image Coding", *IEEE Transactions on Circuits and Systems*, Volume 34, No. 11, pp. 1306-1336, 1987.

[7] V. T. Rajan, "Optimality of the Delaunay Triangulation in R^d", *Discrete and Computational Geometry*, Number 12, pp. 189-202, 1994.

[8] P. J. Toivanen, "New Geodesic Distance Transforms for Gray-Level Images", *Pattern Recognition Letters*, Number 17, pp. 437 - 450, 1996.

[9] P. J. Toivanen, A. M. Vepsäläinen, and J. P. S. Parkkinen, "Image Compression using the Distance Transform on Curved Space (DTOCS) and Delaunay Triangulation", *Pattern Recognition Letters*, Number 20, pp. 1015-1026, 1999.

[10] S. Wang and A. Y. Wu, and A. Rosenfeld, "Image Approximation from Gray Scale Medial Axes", *IEEE Transactions on Pattern Analysis and Machine Intelligence* Volume PAMI-3, No. 6, 1981.

[11] R. Wilson, H. E. Knutsson, and G. H. Granlund, "Anisotropic nonstationary image estimation and its applications: Part II -Predictive image coding", *IEEE Transactions on Communication 31*, pp. 398-406, 1983.

Publication II

Ikonen, L., Toivanen, P.
Shortest Route on Height Map Using Gray-Level Distance Transforms

Shortest Route on Height Map Using Gray-Level Distance Transforms

Leena Ikonen and Pekka Toivanen

Lappeenranta University of Technology
P.O.Box 20, 53851 Lappeenranta, Finland
leena.ikonen@lut.fi

Abstract. This article presents an algorithm for finding and visualizing the shortest route between two points on a gray-level height map. The route is computed using gray-level distance transforms, which are variations of the Distance Transform on Curved Space (DTOCS). The basic Route DTOCS uses the chessboard kernel for calculating the distances between neighboring pixels, but variations, which take into account the larger distance between diagonal pixels, produce more accurate results, particularly for smooth and simple image surfaces. The route opimization algorithm is implemented using the Weighted Distance Transform on Curved Space (WDTOCS), which computes the piecewise Euclidean distance along the image surface, and the results are compared to the original Route DTOCS. The implementation of the algorithm is very simple, regardless of which distance definition is used.

1 Introduction

Finding the shortest path between two points on a three dimensional surface is a common optimization problem in many practical applications, e.g. robotic and terrain navigation, highway planning, and medical image analysis. By considering the digitized surface as a graph, variations of Dijkstra's classical path search algorithm become feasible (e.g. [4], [10]). A dynamic programming-based algorithm for computing distances of fuzzy digital objects is presented in [9].

This article presents an algorithm for finding optimal routes, or so called minimal geodesics, between two points on a gray-level height map. Other distance map approaches for path optimization include level sets propagation [3], and morphological grassfire algorithms [5]. Our algorithm is based on the Distance Transform on Curved Space (DTOCS presented in [13]), which calculates distances on a gray-level surface, when the gray-levels are understood as height values of the image surface. The Route DTOCS, first presented in [2], is developed further by using distance definitions, which give more accurate values for the global distances compared to the original chessboard distance transform. Particularly the piecewise Euclidean distance calculated with the Weighted Distance Transform on Curved Space (WDTOCS [13]) produces reliably optimal routes.

2 Definitions for Route DTOCS and WDTOCS

In the distance image produced by the DTOCS or the WDTOCS, every pixel in the calculation area X has a value which corresponds to the distance of that pixel to the nearest background pixel in X^C. The definition of the DTOCS for any calculation area X can be found in [13]. In the Route DTOCS the same distance metrics apply, but the complement area X^C is restricted to a single point, and the distance can be calculated according to the following, slightly simplified, definitions.

A discrete gray-level image is a function $\mathcal{G} : Z^2 \to N$, where N is the set of positive integers.

Definition 1. *Let $N_8(p)$ denote the set of all 8 neighbors of pixel p in Z^2. Pixels p and q are 8-connected if $q \in N_8(p)$. Let $N_4(p)$ denote the set of 4-connected neighbors, and $N_8(p) \setminus N_4(p)$ the set of diagonal neighbors. A discrete 8-path from pixel p to pixel s is a sequence of pixels $p = p_0, p_1, ..., p_n = s$, where every p_i is 8-connected to p_{i-1}, $i = 1, 2, ..., n$..*

Definition 2. *Let $\Psi(x, y)$ denote the set of all possible discrete 8-paths linking points $x \in X$ and $y \in X^C$. Let $\gamma \in \Psi(x, y)$ and let γ have n pixels. Let p_i and p_{i+1} be two adjacent pixels in path γ. Let $\mathcal{G}(p_i)$ denote the gray value of pixel p_i.*

The length of the path γ is defined by $\Lambda(\gamma) = \sum_{i=1}^{n-1} d(p_i, p_{i+1})$, where the definition of $d(p_i, p_{i+1})$, i.e. the distance between neighbor pixels p_i and p_{i+1} on the path, depends on the distance transform used. The Weigthed Distance Transform on Curved Space (WDTOCS) uses the Euclidean distance calculated with the Pythagoras' theorem from the height difference and the horizontal displacement of the two pixels:

$$d(p_i, p_{i+1}) = \begin{cases} \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i+1})|^2 + 1} \ , \ p_{i+1} \in N_4(p_i) \\ \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i+1})|^2 + 2} \ , \ p_{i+1} \in N_8(p_i) \setminus N_4(p_i) \end{cases} \tag{1}$$

In the chessboard DTOCS the distance is defined as the height (gray-level) difference between the pixels, plus one for the horizontal displacement:

$$d(p_i, p_{i+1}) = |\mathcal{G}(p_i) - \mathcal{G}(p_{i+1})| + 1 \tag{2}$$

The distance can also be defined using separate height and pixel-to-pixel displacements as in DTOCS, but using the accurate horizontal distance between diagonal neighbors:

$$d(p_i, p_{i+1}) = \begin{cases} |\mathcal{G}(p_i) - \mathcal{G}(p_{i+1})| + 1 \quad , \ p_{i+1} \in N_4(p_i) \\ |\mathcal{G}(p_i) - \mathcal{G}(p_{i+1})| + \sqrt{2} \ , \ p_{i+1} \in N_8(p_i) \setminus N_4(p_i) \end{cases} \tag{3}$$

Definition 3. *The distance image $\mathcal{F}^*(x)$ when $X^C = \{y\}$ is*

$$\mathcal{F}^*(x) = \begin{cases} \min\limits_{\gamma \in \Psi}(\Lambda(\gamma)) \ , \ x \in X \\ 0 \qquad\qquad\quad , \ x \in X^C \end{cases} \tag{4}$$

The same distance image definition is used for the WDTOCS, the DTOCS and for the distance transform using $\sqrt{2}$ as the horizontal displacement between diagonal neighbors. The definition of neighbor-distances $d(p_i, p_{i+1})$ used in calculating the path length $\Lambda(\gamma)$ determines which version of the distance transform is produced by the algorithm.

3 The Distance Transformation Algorithms

The two-pass algorithm (see [13]) for calculating the DTOCS or the WDTOCS image $\mathcal{F}^*(x)$ is a sequential local operation (see [7]). The algorithm requires two images: the original gray-level image $\mathcal{G}(x)$ and a binary image $\mathcal{F}(x)$, which determines the region(s) in which the transformation is performed. The calculation area X in $\mathcal{F}(x)$ is initialized to *max* (the maximal representative number of memory) and the complement area X^C to 0.

The first computation pass proceeds using the mask $M_1 = \{p_{nw}, p_n, p_{ne}, p_w\}$ in figure 1 rowwise from the top left corner of the image, substituting the middle point $\mathcal{F}(p_c)$ with the distance value

$$\mathcal{F}_1^*(p_c) = \min[\mathcal{F}(p_c), \min_{p \in M_1}(\Delta(p) + \mathcal{F}_1^*(p))] \tag{5}$$

The distance $\Delta(p)$ between pixels p_c and p is calculated according to the definition of the distance transformation that is used:

$$\text{WDTOCS:} \ \Delta(p) = \begin{cases} \sqrt{|\mathcal{G}(p) - \mathcal{G}(p_c)|^2 + 1}\ , & p \in N_4(p_c) \\ \sqrt{|\mathcal{G}(p) - \mathcal{G}(p_c)|^2 + 2}\ , & p \in N_8(p_c) \setminus N_4(p_c) \end{cases} \tag{6}$$

$$\text{DTOCS:} \ \Delta(p) = |\mathcal{G}(p) - \mathcal{G}(p_c)| + 1 \tag{7}$$

$$\sqrt{2}\text{-DTOCS:} \ \Delta(p) = \begin{cases} |\mathcal{G}(p) - \mathcal{G}(p_c)| + 1 & , p \in N_4(p_c) \\ |\mathcal{G}(p) - \mathcal{G}(p_c)| + \sqrt{2}\ , & p \in N_8(p_c) \setminus N_4(p_c) \end{cases} \tag{8}$$

The backward pass uses the mask $M_2 = \{p_e, p_{sw}, p_s, p_{se}\}$ in figure 1 replacing the distance value $\mathcal{F}_1^*(p_c)$ calculated by the forward pass with the new value

$$\mathcal{F}^*(p_c) = \min[\mathcal{F}_1^*(p_c), \min_{p \in M_2}(\Delta(p) + \mathcal{F}^*(p))] \tag{9}$$

If the original gray-level map is complex, the two calculation passes may have to be repeated several times to get the perfect distance map (see [11]). The distance image $\mathcal{F}^*(x)$ is used instead of the binary image $\mathcal{F}(x)$ for the next computation pass repeatedly until the DTOCS algorithm has converged to the globally optimal distances.

4 The Shortest Route Algorithm

The shortest route algorithm is based on calculating two distance maps, one for each endpoint of the desired route. Assuming we have a gray-level map $G(x)$

p_{nw}	p_n	p_{ne}
p_w	(p_c)	

	(p_c)	p_e
p_{sw}	p_s	p_{se}

Fig. 1. The masks for calculating the DTOCS. The left mask M_1 is used in the forward calculation pass, and the right mask M_2 in the backward pass.

and want to find an optimal route from point a with gray-level value (i.e. height) $G(a)$ to point b with value $G(b)$, we initialize the binary images $\mathcal{F}_a(x)$ and $\mathcal{F}_b(x)$ with $X_a^C = \{a\}$ and $X_b^C = \{b\}$ respectively. Using these two images, we calculate the distance images $\mathcal{F}_a^*(x)$ and $\mathcal{F}_b^*(x)$ with one of the distance transformation algorithms. In the resulting distance maps each value corresponds to the distance between point x and point a (or b respectively) along an 8-connected path that is optimal according to the distance definition of the used algorithm, WDTOCS, DTOCS or $\sqrt{2}$-DTOCS. It can be noted that $\mathcal{F}_a^*(b)$ as well as $\mathcal{F}_b^*(a)$ equals the length of the shortest route between points a and b, but the route itself can not be seen in the separate maps. Using the two maps we define the route distance:

$$\mathcal{D}_{\mathcal{R}}(x) = \mathcal{F}_a^*(x) + \mathcal{F}_b^*(x) \tag{10}$$

For each point x the value $\mathcal{D}_{\mathcal{R}}(x)$ is the length of the shortest path from point a to b that passes through point x. The value $\mathcal{F}_a^*(x)$ is the shortest distance from a to x, and $\mathcal{F}_b^*(x)$ is the shortest distance from x to b, and these optimal subpaths form an optimal path (see [6]). The equal distance propagation curves in [3] are combined similarly to form minimal geodesics. Now the optimal route from a to b is the set of points, for which the route distance is minimal. We define the route:

$$\mathcal{R}(a,b) = \{\, x \mid \mathcal{D}_{\mathcal{R}}(x) = \min_x \mathcal{D}_{\mathcal{R}}(x)\} \tag{11}$$

There can be several optimal paths, and the set $\mathcal{R}(a,b)$ contains all points that are on any optimal path, so this method does not provide an analytical description of a distinct route (e.g. a sequence of pixels). However, the routes can be visualized by marking the set of pixels $\mathcal{R}(a,b)$ on the original image. In WDTOCS and $\sqrt{2}$-DTOCS real values are used in the calculations, but the route distance $\mathcal{D}_{\mathcal{R}}(x)$ is rounded up to nearest integer before finding the points with the minimal distance. To summarize, **the shortest route algorithm** is:

1. Calculate the distance image $\mathcal{F}_a^*(x)$ from source point a
2. Calculate the distance image $\mathcal{F}_b^*(x)$ from destination point b
3. Calculate the route distance $\mathcal{D}_{\mathcal{R}}(x) = \mathcal{F}_a^*(x) + \mathcal{F}_b^*(x)$
4. Mark points with $\mathcal{D}_{\mathcal{R}}(x) = \min_x \mathcal{D}_{\mathcal{R}}(x)$ as points on optimal route $\mathcal{R}(a,b)$

5 Experiments and Results

This section demonstrates how the shortest route algorithm works, and compares the results of implementations with different distance definitions. Figure

2 presents a step by step application of the algorithm. Figure 2 a) is the original gray-level image. Figures 2 b) and 2 c) show the DTOCS-images $\mathcal{F}_a^*(x)$ and $\mathcal{F}_b^*(x)$ calculated from the endpoints a and b (marked with 'x'). As the distance function is symmetrical, it does not matter which endpoint corresponds to a and which to b. Figure 2 d) shows the route distance image, i.e. the sum of the DTOCS-images. Images 2 b)–d) are scaled to gray-levels, but original distance values, which can be beyond 255, are used in the calculation of $\mathcal{D}_{\mathcal{R}}(x)$. Figure 2 e) presents the final result, i.e. the points in set $\mathcal{R}(a, b)$. Figure 2 f) presents the same route calculated with the WDTOCS. It can be seen that for the complex image surface representing varying terrain the route is very similar, but sharper than the route by the DTOCS.

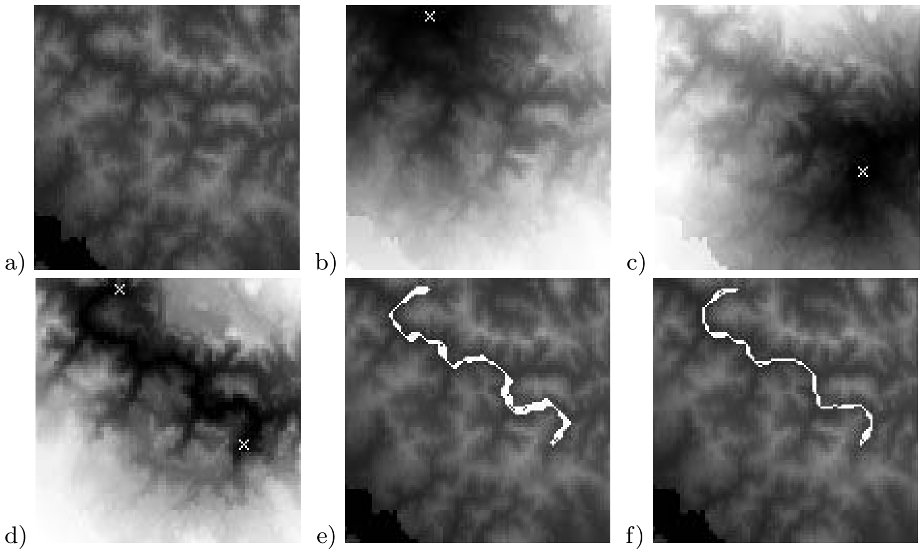


Fig. 2. a) Original image, b) distance from source point, c) distance from destination point, d) sum of distance images, e) route by DTOCS, f) route by WDTOCS.

A sample application, where the shortest route idea is used to solve a labyrinth, was presented in [2]. Figure 3 a) shows the route through a labyrinth produced by the original Route DTOCS. The algorithm needs a threshold segmented image, where labyrinth paths get value zero and walls get a very high value. Then the shortest path from the entrance to the exit of the labyrinth is the route through the labyrinth. It can be seen in figure 3 a) that the route makes seemingly extra 90° corners when calculated with the chessboard DTOCS.

The explanation to this problem is visualized in figure 4. The route from point A to B that passes through point x is just as short as as the intuitively optimal straight route, as there are as many pixel-to-pixel displacements on both routes. Consequently, there are several optimal discrete 8-connected paths through the labyrinth, and as the route is defined as the set of all points that are on any optimal path, the visualized route becomes wide. Figure 3 b) shows how the route width decreases when the longer distance between diagonal pixels is taken into account according to equation 3.

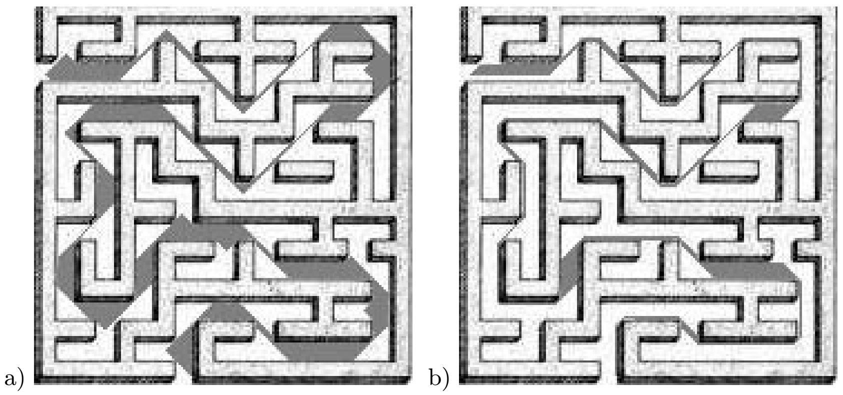


Fig. 3. a) Route through labyrinth by DTOCS b) Route through labyrinth by DTOCS with $\sqrt{2}$ diagonal distances.

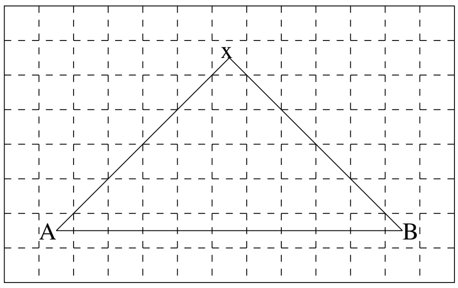


Fig. 4. Two of several possible routes from point A to B on a flat image surface according to the chessboard distance definition. The DTOCS distance is the same along the route through point x as along the straight line, as there are as many pixels on both routes (each square represents a pixel).

Tests with a gray-scale-ball image show similar results. The routes between the endpoints of the horizontal diameter of the half-sphere are too wide, when calculated with the basic Route DTOCS (figure 5 a), but introducing the $\sqrt{2}$-factor to the diagonal neighbor distances makes the routes as optimal as can be expected of discrete 8-connected paths (figure 5 b). Using the Euclidean neighbor distances of the WDTOCS changes the result dramatically, i.e. the algorithm finds the route across the half-sphere rather than around it (figure 5 c). The differing route lengths are partly a result of the digitization of the sphere function. Figure 6 shows a cross-section and a horizontal projection of a digital ball with few pixels. The digitization error is smaller but still present when using a higher resolution ball image. Another big factor is that the variation in surface height increases the WDTOCS-distances less than the distances of the transforms that add the height difference to the horizontal displacement. The DTOCS-distance across the ball along the route the WDTOCS algorithm finds optimal (as in figure 5 c) would be clearly longer than the WDTOCS-distance, as each neighbor-distance $\sqrt{d^2+1}$ is replaced with $d+1$, where d is the height difference of the neighbor pixels.

When gray-level variations, i.e. height differences are large, the effect the horizontal displacements have on the distance value decreases in WDTOCS, whereas it stays constant in DTOCS and $\sqrt{2}$-DTOCS. The application determines which approach is better. If the transformation is used to approximate actual distances along a real surface, using the piecewise Euclidean distance of WDTOCS is justified. If the gray-level differences represent a different type of cost than the horizontal displacements, the transformations adding horizontal and vertical distances may work better and be more easily scalable. To modify the effect the height differences have on the distance transform, the original image can be scaled before applying the transformation. Alternatively, a weighting factor can be added to the height difference in equations 1, 2 and 3.

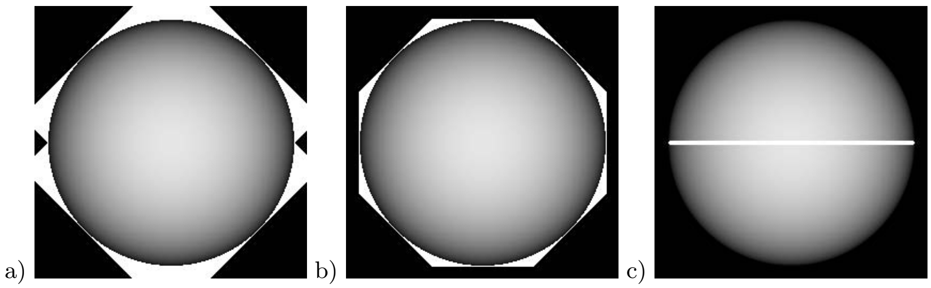


Fig. 5. a) Route by DTOCS, b) Route by DTOCS with $\sqrt{2}$ diagonal distance, c) Route by WDTOCS.

6 Discussion

In previous work, the DTOCS algorithm has mostly been used to calculate local distances. For example in image compression (see e.g. [12]) distance values are used to measure the variation of the image surface. More control points need to be stored from image areas, where local distances are high, i.e where gray-level values change rapidly. In such applications the chessboard distance transform works well enough, and the use of integer approximations of distance values is justified to save computation time and space. However, the route optimization algorithm computes global distances across the whole image, and the approximation error of the chessboard distance accumulates. Particularly on smooth and simple image surfaces, the chessboard Route DTOCS performs poorly, and using the WDTOCS produces more reliable optimal routes.

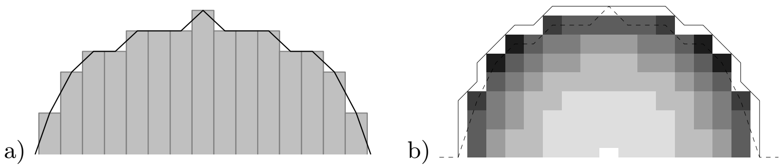


Fig. 6. a) Cross section of digititized ball with the WDTOCS route across the ball. The height of the bars corresponds to gray-level values. b) flat projection of digitized ball with the $\sqrt{2}$-DTOCS route around the ball, and the shape of the WDTOCS-route marked with dashed line for comparison. Each square represents a pixel.

The distance transform using $\sqrt{2}$ as the diagonal pixel-to-pixel displacement is an interesting hybrid of chessboard and Euclidean distance definitions, as the locally Euclidean distance is used as the horizontal pixel-to-pixel displacement, but the height difference is calculated just as in the chessboard DTOCS. The theoretical basis for this hybrid distance transform may not be as solid as for the DTOCS and the WDTOCS, but in route optimization it can give some interesting results. For example in the labyrinth application the horizontal displacements form the desired route, and the values of the gray-level differences are not significant, as long as distances along low paths are clearly shorter than distances over high walls. Other obstacle avoidance problems can be solved using the route optimization algorithm, and treating the horizontal and vertical displacements differently can be practical.

Using the piecewise Euclidean distances of WDTOCS gives the most accurate approximations for distances along the image surface. If the slightly heavier computation of floating point values instead of integers is not a problem, the WDTOCS algorithm should be used to get the best results in route optimization. A question for future research is whether we can define integer kernel distances, which approximate the Euclidean distance more accurately than the DTOCS. Borgefors [1] showed that using local distances 3 and 4 for square and diagonal neighbors in binary images actually gives a better approximation of Euclidean distance along the horizontal image plane than the distances 1 and $\sqrt{2}$ used here. Extending the ideas to gray-level images requires further investigation into how the height differences affect, and how they should affect the distance transformation.

References

1. Borgefors, G.: Distance Transformations in Digital Images. Computer vision, Graphics, and Image Processing, 34 (1986) 344–371
2. Ikonen, L., Toivanen, P., Tuominen, J.: Shortest Route on Gray-Level Map using Distance Transform on Curved Space. Proc. of Scandinavian Conference on Image Analysis (2003) 305–310
3. Kimmel, R., Amir, A. and Bruckstein A.: Finding Shortest Paths on Surfaces Using Level Sets Propagation. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 17, no. 6 (1995) 635–640

4. Kimmel, R. and Kiryati, N.: Finding Shortest Paths on Surfaces by Fast Global Approximation and Precise Local Refinement. International Journal of Pattern Recognition and Artificial Intelligence, vol 10 (1996) 643–656
5. Lin P., Chang S.: A Shortest Path Algorithm for a Nonrotating Object Among Obstacles of Arbitrary Shapes. IEEE Transactions on Systems, Man, and Cybernetics, vol 23, no 3 (1993) 825–833
6. Piper J., Granum E.: Computing Distance Transformations in Convex and Non-Convex Domains. Pattern Recognition, vol 20, no 6 (1987) 599–615
7. Rosenfeld A., Pfaltz, J. L.: Sequential Operations in Digital Picture Processing. Journal of the Association for Computing Machinery, vol 13, no 4 (1966) 471–494
8. Rosin, P., West, G.: Salience Distance Transforms. Graphical Models and Image Processing, vol 56, no 6 (1995) 483–521
9. Saha P. K., Wehrli F. W., Gomberg B. R.: Fuzzy Distance Transform: Theory, Algorithms and Applications. Computer Vision and Image Understanding 86 (2002) 171–190
10. Saab, Y. and VanPutte M.: Shortest Path Planning on Topographical Maps. IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans, vol 29, no 1 (1999) 139–150
11. Toivanen, P. J.: Convergence properties of the Distance Transform on Curved Space (DTOCS). Proc. of Finnish Signal Processing Symposium (1995) 75–79
12. Toivanen, P. J.: Image Compression by Selecting Control Points Using Distance Function on Curved Space, Pattern Recognition Letters 14 (1993) 475–482
13. Toivanen, P.: New geodesic distance transforms for gray-scale images. Pattern Recognition Letters, 17 (1996) 437–450

Publication III

IKONEN, L., TOIVANEN, P.,
Shortest Routes Between Sets on Gray-Level Surfaces

SHORTEST ROUTES BETWEEN SETS ON GRAY-LEVEL SURFACES

L. Ikonen and P. Toivanen

Lappeenranta University of Technology, Laboratory of Information Processing
P.O. Box 20, 53851 Lappeenranta, Finland
{leena.ikonen, pekka.toivanen} @lut.fi

This article presents a distance transform based algorithm for finding shortest routes along gray-level surfaces. Finding the shortest path between two point sets using a point-to-point shortest path algorithm would require a minimum path search for each possible end-point pair. The new distance transform approach finds the route consisting of all optimal paths between two point sets as easily as the route between a fixed pair of points, in near-linear time. It is an efficient 2D solution to the 3D path optimization problem

Introduction

The shortest path along a surface is needed in many types of applications. Paths can be computed on images representing actual surfaces, like height maps in terrain navigation applications, or on surfaces representing a cost function. Shortest paths could be found with a graph search, but the complexity $O(n^2)$ of the basic Dijkstra algorithm is a problem in large images, where each pixel represents a vertex. As edges exist only between neighbor pixels, more efficient sparse graph algorithms are applicable, but the complexity is still $O(n \log n)$ for computing one single source shortest path. Shortest paths between sets of points could be found by computing single source shortest paths for each point in the source point set, or, if the point sets are large, more efficiently with an all-pairs algorithm. Our algorithm computes the optimal route between sets of points in near-linear time by combining distance transform maps. Here the route is defined as the set of points, which belong to any optimal path. If a distinct path is needed, it can be extracted with a simple backtracking algorithm.

Shortest Route Algorithm

The Distance Transform on Curved Space (DTOCS) computes distances along a gray-level surface, when gray-levels are understood as height values [9]. The distance maps describing the distance of each point to the nearest background or feature point are produced with a sequential local transformation similar to the Chamfer algorithms presented in [1]. The DTOCS just adds the difference between gray-levels *G(p)* and *G(p_c)* to the local horizontal distance between the neighbor pixels *p* and *p*$_c$ (mask center pixel):

$$\Delta(p) = |\mathcal{G}(p) - \mathcal{G}(p_c)| + 1 \tag{1}$$

The Weighted DTOCS (WDTOCS) uses piece-wise Euclidean local distances computed with the Pythagoras' theorem from the horizontal displacement and the height difference:

$$\Delta(p) = \begin{cases} \sqrt{|\mathcal{G}(p) - \mathcal{G}(p_c)|^2 + 1} & , \; p \in N_S(p_c) \\ \sqrt{|\mathcal{G}(p) - \mathcal{G}(p_c)|^2 + 2} & , \; p \in N_D(p_c), \end{cases} \tag{2}$$

where $N_S(p_c)$ denotes the set of square neighbors of pixel p_c, and $N_D(p_c)$ the set of diagonal neighbors. The WDTOCS produces clearly more accurate global distances than the DTOCS, which underestimates the length of diagonal steps compared to straight steps. The sequential distance propagation passes need to be repeated until the transformation converges to the globally optimal distance map $F^*(x)$.

The Route DTOCS algorithm, first presented in [5] and refined by using the WDTOCS in [4], requires two distance maps $F^*_a(x)$ and $F^*_b(x)$. The route end-point *a* (resp. *b*) is the feature, from which all distances are computed. From the distance maps, a route distance image is calculated by a simple addition:

$$\mathcal{D}_R(x) = \mathcal{F}_a^*(x) + \mathcal{F}_b^*(x) \qquad (3)$$

The value $D_R(x)$ is the distance between the route end-points along the shortest path passing through point x. Consequently, the points with the minimal route distance value form the desired route, so the definition for the shortest route between the two points a and b is:

$$\mathcal{R}(a,b) = \{\, x \mid \mathcal{D}_R(x) = \min_x \mathcal{D}_R(x)\} \qquad (4)$$

The same idea has been used in [7] for a distance transform adding gray-level values along the path. Such a distance definition produces minimal cost paths, which are inherently different than routes along surfaces.

Finding Routes between Point Sets

The original Route DTOCS and WDTOCS were designed to find the route between two single points, but it is quite natural to extend the idea to find routes between point sets, or areas of the image. In fact, the single point-pair route is just a special case of a more general route between sets. The DTOCS and the WDTOCS, like common distance transforms for binary images, compute the distance to the nearest feature, i.e. the distance to the nearest point or points in the feature point set. The route between point sets is found by calculating distance maps $F^*_A(x)$ and $F^*_B(x)$, where A and B are the point sets between which we want to find an optimal route. The points with minimal values in the resulting route distance image form the optimal route between the point sets. The end-points of the routes are the points, which belong both to the route set, and one of the original end-point sets. There can be several optimal paths, so the resulting routes are rarely one pixel wide.

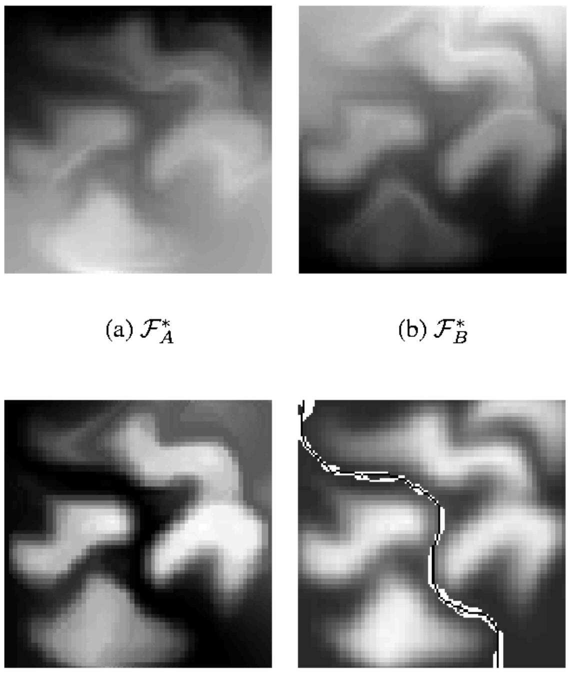
Extracting a Distinct Path

Finding all optimal paths as one route can be sufficient in many applications, but extracting a distinct path, which can be described for example with a chain code, can also be necessary. A path can be found from a binary distance transform map simply by following decreasing distance values from the target point towards the source point. However, in the DTOCS setting, the step lengths vary, and the neighbor pixel with the smallest distance value is not necessarily the next pixel on an optimal path. The route set defined by the Route DTOCS can be utilized in backtracking a path. The backtracking starts from a destination point b belonging to the route set. The next path pixel is chosen among the neighbors, which also belong to the route, and have a smaller distance value on the map F^*_A than the current pixel. As there usually is more than one optimal path, different paths are found depending on which neighbor is chosen. To extract all paths, recursive backtracking from each feasible neighbor pixel would be required.

Experiments

Figure 1 demonstrates how the shortest route algorithm proceeds, when finding the shortest route between two point sets on the image seen with the resulting route in figure 1(d). Here one set is the top line of the image, and the distances from that set to all other points in the image can be seen in figure 1(a). The other set is the bottom line of the image producing the distance map in figure 1(b). The route distance image, i.e. the sum of the two distance images can be seen in figure 1(c). The resulting route can be seen in figure 1(d), where also an example of a distinct path extracted with the path backtracking algorithm is included.

Another example can be seen in figure 2. The route between a point near the center of the twirl to anywhere on the rightmost column of the image is found using the WDTOCS. In figure 2(a) the surface is used as it is, and in figure 2(b) it is scaled with two, so that the largest height value 253 becomes 506, doubling also the height differences. The effect the height differences should have depends on the application. If the image represents an actual surface, and real distances need to be calculated, the height values should be scaled so that one unit in gray-level difference corresponds to the horizontal distance between square neighbor pixels.


(a) $\mathcal{F}^*_A$ (b) $\mathcal{F}^*_B$

(c) $\mathcal{D}_R$ (d) WDTOCS Route

Fig. 1. Finding the shortest route between any point on the top line of the image to any point on the bottom line.

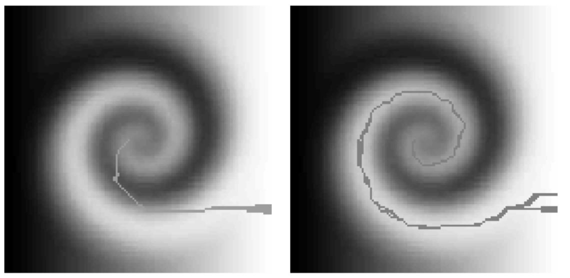
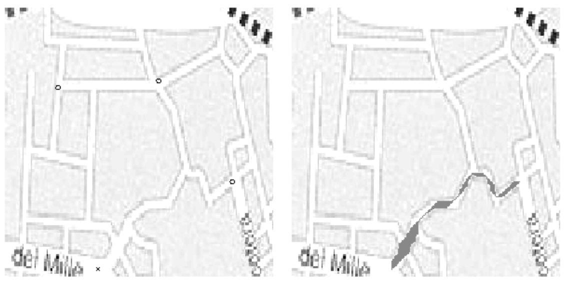

(a) (b)

Fig. 2. Route from center to rightmost column

The end-point sets do not have to be connected. Figure 3 presents a navigation example, where the objective is to find the nearest of three destinations, marked with 'o' in figure 3(a), starting from one source point marked with 'x'. The alternative destination points form one end-point set. Figure 3(b) shows the result, i.e. the route to the nearest destination. The lengths of the shortest routes to the other destinations can be read directly from the distance map computed with the source point as ``feature'', but to visualize the routes a new distance map from each destination would be needed. In this street map example the gray-levels do not correspond to height, but the image is scaled so that the height difference between roads and other areas are clear enough to make sure all paths follow the roads. The image could also be segmented to a binary image with roads having one value, and background another. Then either the two values in the binary image can be significantly different (e.g. one zero and the other very large), or the height difference in the local distance can be scaled with a large value to prevent illegal shortcuts. Applying the Route DTOCS to this kind of an obstacle avoidance problem is very similar to the idea of the constrained distance transform, where a path along a ternary image, with feature, non-feature and constraint pixels are found with a binary distance transform modified to avoid the constraint pixels, see e.g. [2] for a good example. The DTOCS approach makes it possible to use several different levels of obstacles, i.e. ones that can absolutely not be crossed marked with infinite values, and others that can be crossed but at a higher cost. Only the input image must be edited and the algorithm works without any changes. The constrained distance transform can cope only with the absolute obstacles marked with constraint pixels.



(a) (b)

Fig. 3. Path planning example, where the route to the nearest of three destinations is computed.

Complexity Analysis

The complexity analysis of the algorithm is not straightforward. One two-pass iteration of the WDTOCS is clearly of O(n), where n is the number of pixels. Adding the distance maps as well as finding the minimum can also be done in linear time. The problematic part is estimating the number of WDTOCS-iterations needed to produce the final distance maps, as it depends on the size and the complexity of the surface. The theoretical worst case situation would be that the distance to the pixel with most pixels on its shortest path to nearest feature propagates only one pixel forward at each iteration, i.e. the number of pixels on the path with most pixels is an upper limit on the number of iterations. In practice, much fewer iterations are usually required. The example in figure 1 needed 15 iterations for the first distance map and 13 iterations for the second, the examples in figure 2 needed 11 and 12 iterations for the left image and 15 and 13 for the right image, and the example in figure 3 needed 14 and 12 iterations. The numbers include the extra iteration needed to detect convergence.

When comparing this method to a graph search algorithm, the gray-level image corresponds to a sparse graph with a large amount of vertices. The complexity of a sparse graph modification of Dijkstra is in $O(n \log n)$ for computing the path between one end-point pair. This is comparable to the complexity of the Route DTOCS if the number of iterations needed for a globally optimal distance map is around $\log(n)$, which is a reasonable estimate (e.g. image size $n = 128*128 \rightarrow \log(n) = 14$). So the Route DTOCS computes all optimal routes between two point sets with a similar complexity as a graph search calculates only one shortest path candidate between the sets.

Discussion

This paper presented the Route DTOCS, a distance transform based method for finding the shortest route along a gray-level surface. The shortest route between sets of points can be found as easily as the shortest route between single end-points. The algorithm is very simple, and runs in near-linear time. In some practical applications the estimate of the shortest route achieved by one or a constant small number of iterations of the distance transformation could be accurate enough, which would make the running time linear.

Distances along a gray-level surface could be computed also by transforming the 2D gray-level image to an umbra or surface relief in 3D, see e.g. [8]. Distances along the top surface of the umbra can be computed with well known binary distance transformations in 3D, see e.g. [3], constrained to the surface. Also graph search combined with estimation of lengths of digital curves could be used [6]. However, keeping the computation in 2D with the DTOCS approach has its benefits, as the increase in problem size, which is inevitable when transforming to 3D, can be avoided. With the most naive representation of the umbra the problem size is multiplied with the number of gray-levels in the used range (e.g. problem size $m*n$ can grow to 255*m*n). Even if only the top surface of the umbra is stored, the number of voxels is considerably higher than the number of pixels in the original image, unless the surface is very smooth. Paths also consist of fewer pixels in 2D than voxels in 3D. The Route DTOCS for finding routes between sets of points is thus a simplification in two ways - finding all paths as easily as a single source path, and solving a 3D problem in a less complex 2D setting.

References

1. G. Borgefors, Distance Transformations in Digital images. Computer Vision, Graphics and Image Processing, 34:344-371, 1986.
2. G. Borgefors, Applications using distance transforms. In Aspects of Visual Form Processing, pages 83-108. World Scientific, Singapore 1994.
3. G. Borgefors, On digital distance transforms in three dimensions. Computer Vision and Image Understanding, 64(3):368-376, 1996.
4. L. Ikonen and P. Toivanen, Shortest Route on Height Map using Gray-Level Distance Transforms. In Discrete Geometry for Computer Imagery (DGCI), pages 308-316, 2003.
5. L. Ikonen, P. Toivanen and J. Tuominen, Shortest Route on Gray-Level Map using Distance Transform on Curved Space. In Scandinavian Conference on Image Analysis (SCIA), pages 305-310, 2003.
6. N. Kiryati and G. Szekely, Estimating Shortest Paths and Minimal Distances on Digitized Three-dimensional Surfaces. Pattern Recognition, 26(11):1623-1637, 1993.
7. P. Soille, Generalized geodesy via geodesic time. Pattern Recognition Letters, 15(12):1235-1240, 1994.
8. S. Sternberg, Grayscale Morphology. Computer Vision, Graphics, and Image Processing, 35:333-355, 1986.
9. P. Toivanen, New geodesic distance transforms for Gray-scale images. Pattern Recognition Letters, 17:437-450, 1996.

Publication IV

Ikonen, L., Toivanen, P.,
Shortest Routes on Varying Height Surfaces using Gray-Level Distance Transforms

Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

Image and Vision Computing 23 (2005) 133–141

www.elsevier.com/locate/imavis

Shortest routes on varying height surfaces using gray-level distance transforms

Leena Ikonen*, Pekka Toivanen

Department of Information Technology, Lappeenranta University of Technology, P.O. Box 20, Lappeenranta FIN-53851, Finland

Received 16 January 2004; received in revised form 28 April 2004; accepted 29 June 2004

Abstract

The distance transform on curved space (DTOCS) and its locally Euclidean modification weighted DTOCS (WDTOCS) calculate distances along gray-level surfaces. This article presents the Route DTOCS algorithm for finding and visualizing the shortest route between two points on a gray-level height map, and also introduces new distance definitions producing more accurate global distances. The algorithm is very simple to implement, and finds all optimal paths between the two points at once. The Route DTOCS is an efficient 2D approach to finding routes on a 3D surface. It also provides a more flexible solution to obstacle avoidance problems than the constrained distance transform.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Distance transforms; Gray-level distance transforms; Shortest paths; Minimal geodesics; Obstacle avoidance

1. Introduction

Finding the shortest path between two points along a surface of varying height is an optimization problem in many practical applications, for example in terrain navigation and medical image analysis The shortest path is a widely researched problem, with many known solutions. Variations of Dijkstra's path search algorithm can be used by modifying the path optimization problem to a graph search, as presented, for example, in Refs. [1,2]. A dynamic programming-based algorithm for computing distances of fuzzy digital objects is presented in Ref. [3].

This article presents an algorithm for finding shortest routes, or so called minimal geodesics, between two points on a surface described with a gray-level height map. The algorithm is based on gray-level distance transforms, which are variations of the distance transform on curved space (DTOCS) presented in Ref. [4]. Other distance map approaches for path optimization include level sets propagation as in Ref. [5], and morphological grassfire algorithms as in Ref. [6]. The DTOCS calculates distances on a varying height surface represented as a gray-level image, where gray-levels correspond to height values. The Route DTOCS algorithm, first presented in Ref. [7], and refined in Ref. [8], utilizes the distance maps to find optimal routes. The original chessboard algorithm underestimates diagonal local distances, and better results are achieved using the integer approximations three and four for optimal neighbor distances well known from Ref. [9]. The weighted distance transform on curved space (WDTOCS) with Euclidean local distances presented in Ref. [4] produces more accurate global distances than the integer approximations, and the new Optimal WDTOCS introduced in this article improves the results even further by utilizing local distances proven to be optimal for binary distance transforms by Ref. [9]. The simple Route DTOCS finds all optimal paths between two points at once, even if there are alternative paths with the same length. The algorithm does not extract a single digital path, which could be described for example with a chain code. Instead it finds the route, which we define as the set pixels belonging to any optimal path. If a distinct path is needed, it can be extracted from the distance maps with a simple backtracking algorithm.

An option for computing distances along a gray-level surface would be to transform the 2D gray-level image to

* Corresponding author.
E-mail address: leena.ikonen@lut.fi (L. Ikonen).

0262-8856/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.imavis.2004.06.010

a binary umbra or surface relief in 3D. A pixel with gray-value z at location (x, y) in the 2D image corresponds to the point (x, y, z) in 3D space, and coordinate z is the height of the umbra at that point (Ref. [10]). Distances along the top surface of the umbra can be computed with binary distance transformations in 3D constrained to the surface (Ref. [11]) or with graph search techniques combined with estimation of digital curve lengths (Ref. [12]). However, keeping the computation in 2D with the DTOCS approach has its benefits, as the increase in problem size, which is inevitable when transforming to 3D space, can be avoided. With the most naive representation of the umbra the problem size is multiplied with the number of gray-levels in the used range (e.g. problem size $m \times n$ can grow to $255 \times m \times n$). Even if only the top surface of the umbra is stored, the number of voxels is considerably higher than the number of pixels in the original image, unless the surface is very smooth. Also the paths consist of fewer pixels in 2D than voxels in 3D, as each neighbor pixel is reachable with one step, whose length depends on the height difference. The same connection in 3D requires a number of voxel steps corresponding to the gray-level difference.

We will compare the accuracy of different distance definitions, and show with an example that at least for some types of surfaces, the WDTOCS produces more accurate distance values than the best 3D distance transforms. The WDTOCS and the new Optimal WDTOCS can actually smooth down some of the discretization error in digital images, and produce good estimates of distances on the continuous surfaces the images represent.

2. Definitions for distance transforms

In the distance image produced by the DTOCS or the WDTOCS algorithm, every pixel in the calculation area X has a value which corresponds to the distance of that pixel to the nearest background pixel in X^C. The definition of the DTOCS for any calculation area X can be found in Ref. [4]. In the Route DTOCS, the same distance metrics apply, but the complement area X^C consists of a single point, which slightly simplifies the definitions.

Definition 1. Let $N_8(p)$ denote the set of all eight neighbors of pixel p. Pixels p and q are 8-connected if $q \in N_8(p)$. Let $N_4(p)$ denote the set of 4-connected neighbors, and $N_8(p) \backslash N_4(p)$ the set of diagonal neighbors. A discrete 8-path from pixel p to pixel s is a sequence of pixels $p=p_0, p_1, \ldots, p_n=s$, where every p_i is 8-connected to p_{i-1}, $i=1,2,\ldots,n$.

Definition 2. Let $\Psi(x,y)$ denote the set of all discrete 8-paths linking points $x \in X$ and $y \in X^C$. Let $\gamma \in \Psi(x,y)$ and let γ have n pixels. Let p_i and p_{i-1} be two adjacent pixels in path γ. Let $\mathcal{G}(p_i)$ denote the gray value of pixel p_i. The length of the path γ is $\Lambda(\gamma) = \sum_{i=1}^{n} d(p_i, p_{i-1})$, where the distance $d(p_i, p_{i-1})$ between neighbor pixels p_i and p_{i-1} on the path is defined according to the distance transform used.

In the chessboard DTOCS, the neighbor pixel distance is defined as the height difference between the pixels, plus one for the horizontal displacement:

$$d(p_i, p_{i-1}) = |\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})| + 1 \tag{1}$$

Obviously, the DTOCS underestimates the length of diagonal steps. Distance estimates can be improved by using neighbor distances 3 and 4 shown to be the best for binary distance transforms in Ref. [9]. To treat height differences and horizontal displacements in a similar manner, the gray-level difference is scaled with three. We call this new distance transform the 3-4-DTOCS:

$$d(p_i,p_{i-1}) = \begin{cases} 3|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})| + 3, & p_{i-1} \in N_4(p_i) \\ 3|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})| + 4, & p_{i-1} \in N_8(p_i) \backslash N_4(p_i) \end{cases} \tag{2}$$

The WDTOCS uses the piecewise Euclidean distance calculated with the Pythagoras' theorem from the height difference and the local distance in the xy-plane:

$$d(p_i,p_{i-1}) = \begin{cases} \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + 1}, & p_{i-1} \in N_4(p_i) \\ \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + 2}, & p_{i-1} \in N_8(p_i) \backslash N_4(p_i) \end{cases} \tag{3}$$

The WDTOCS defines the horizontal displacement between diagonal pixels as $\sqrt{2}$, which despite being the locally Euclidean distance, does not produce the best approximations of global distances. The algorithm can be improved by introducing these optimal local distances derived in Ref. [9]:

$$a_{\text{opt}} = (\sqrt{2\sqrt{2}-2} + 1)/2 \approx 0.95509 \text{ for square neighbors}$$

$$b_{\text{opt}} = \sqrt{2} + (\sqrt{2\sqrt{2}-2} - 1)/2 \approx 1.36930 \text{ for diagonal neighbors}$$

As the WDTOCS algorithm itself requires floating point computation, the accurate optimal values can easily be introduced without increasing computation time. The new distance transform Optimal WDTOCS is defined as:

$$d(p_i,p_{i-1}) = \begin{cases} \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + a_{\text{opt}}^2}, & p_{i-1} \in N_4(p_i) \\ \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + b_{\text{opt}}^2}, & p_{i-1} \in N_8(p_i) \backslash N_4(p_i) \end{cases} \tag{4}$$

Definition 3. The distance image $\mathcal{F}^*(x)$ is

$$\mathcal{F}^*(x) = \begin{cases} \min\limits_{\gamma \in \Psi}(\Lambda(\gamma)), & x \in X \\ 0, & x \in X^C \end{cases} \tag{5}$$

The same distance image definition applies for all the variations of the DTOCS. The definition of the local distances $d(p_i,p_{i-1})$, which add up to the path length $\Lambda(\gamma)$, determines which transform the algorithm produces.

3. The distance transformation algorithms

The two-pass algorithm for calculating the distance image $\mathcal{F}^*(x)$ is performed by applying sequential local operations as presented in Ref. [13], computed similarly as the Chamfer distance maps in Ref. [9]. In addition to the original gray-level image $\mathcal{G}(x)$ a binary image $\mathcal{F}(x)$ is needed to determine the region or regions in which the transformation is performed. The calculation area X in image $\mathcal{F}(x)$ is initialized to *max* (the maximal representative number of memory) and the complement area X^C to 0. In DT literature describing distance as the distance from nearest feature, e.g. Ref. [9], the features correspond to our complement or background area.

The first computation pass proceeds using the mask $M_1=\{p_{nw},p_n,p_{ne},p_w\}$ in Fig. 1 rowwise from the top left corner of the image, substituting the middle point $\mathcal{F}(p_c)$ with the distance value

$$\mathcal{F}_1^*(p_c) = \min[\mathcal{F}(p_c), \min_{p\in M_1}(\Delta(p)+\mathcal{F}_1^*(p))] \tag{6}$$

The distance $\Delta(p)$ between pixels p_c and p is calculated according to the definition of the distance transformation that is used:

DTOCS :

$$\Delta(p) = |\mathcal{G}(p)-\mathcal{G}(p_c)|+1$$

$3-4-$DTOCS :

$$\Delta(p) = \begin{cases} 3|\mathcal{G}(p)-\mathcal{G}(p_c)|+3, & p\in N_4(p_c) \\ 3|\mathcal{G}(p)-\mathcal{G}(p_c)|+4, & p\in N_8(p_c)\backslash N_4(p_c) \end{cases}$$

WDTOCS :

$$\Delta(p) = \begin{cases} \sqrt{|\mathcal{G}(p)-\mathcal{G}(p_c)|^2+1}, & p\in N_4(p_c) \\ \sqrt{|\mathcal{G}(p)-\mathcal{G}(p_c)|^2+2}, & p\in N_8(p_c)\backslash N_4(p_c) \end{cases}$$

Opt. WDTOCS :

$$\Delta(p) = \begin{cases} \sqrt{|\mathcal{G}(p)-\mathcal{G}(p_c)|^2+a_{\text{opt}}^2}, & p\in N_4(p_c) \\ \sqrt{|\mathcal{G}(p)-\mathcal{G}(p_c)|^2+b_{\text{opt}}^2}, & p\in N_8(p_c)\backslash N_4(p_c) \end{cases}$$

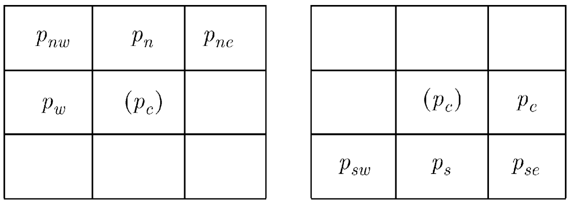


Fig. 1. The masks for calculating the DTOCS. The left mask M_1 is used in the forward calculation pass, and the right mask M_2 in the backward pass.

The backward pass with mask $M_2=\{p_e,p_{sw},p_s,p_{se}\}$ in Fig. 1 replaces the distance value $\mathcal{F}_1^*(p_c)$ calculated by the forward pass with:

$$\mathcal{F}^*(p_c) = \min[\mathcal{F}_1^*(p_c), \min_{p\in M_2}(\Delta(p)+\mathcal{F}^*(p))] \tag{7}$$

The two calculation passes may have to be repeated several times, using the distance image $\mathcal{F}^*(x)$ as the 'binary image' for the next computation pass, until the algorithm converges to the globally optimal distances, Ref. [14]. The number of passes required for convergence is typically around 10, but depends on the size and the complexity of the image surface, and on the number of pixels on the paths. The worst case scenario would be an image and a path requiring one iteration for each pixel in the path.

4. The Route DTOCS algorithm

The shortest route algorithm requires two distance maps, one for each endpoint of the route. Assuming we have a gray-level image $G(x)$ and want to find an optimal route from point a with gray-level value $G(a)$ to point b with value $G(b)$, we initialize the binary images $\mathcal{F}_a(x)$ and $\mathcal{F}_b(x)$ with $X_a^C=\{a\}$ and $X_b^C=\{b\}$, respectively. Using these two images, which in practice have the single pixel a or b set to 0 and all other pixels to *max*, we individually calculate the two distance images $\mathcal{F}_a^*(x)$ and $\mathcal{F}_b^*(x)$. In the resulting distance maps each value corresponds to the distance between point x and point a (or b, respectively) along an optimal 8-connected path. It can be noted that the values $\mathcal{F}_a^*(b)$ and $\mathcal{F}_b^*(a)$ equal the length of the shortest route, but the route itself is not known. Using the two DTOCS images, we define the route distance:

$$\mathcal{D}_R(x) = \mathcal{F}_a^*(x)+\mathcal{F}_b^*(x) \tag{8}$$

For each point x the value $\mathcal{D}_R(x)$ is the length of the shortest path from point a to b that passes through point x. The value $\mathcal{F}_a^*(x)$ is the shortest distance from a to x, and $\mathcal{F}_b^*(x)$ is the shortest distance from x to b, and the optimal subpaths form an optimal path as proved in Ref. [15]. In Ref. [5] equal distance propagation curves are combined similarly to form minimal geodesics. The same idea for computing minimal paths was presented in Ref. [16], but with the distance defined as the sum of gray-values along the path. The resulting minimal cost routes are inherently different than the actual routes along a surface found by the Route DTOCS.

The set of points, for which the route distance is minimal, form the shortest route from a to b, i.e. the route definition is:

$$\mathcal{R}(a,b) = \{x \mid \mathcal{D}_R(x) = \min_x \mathcal{D}_R(x)\} \tag{9}$$

To summarize, the shortest route algorithm is:

(1) Calculate the distance image $\mathcal{F}_a^*(x)$ from source point a
(2) Calculate the distance image $\mathcal{F}_b^*(x)$ from destination point b

(3) Calculate the route distance $\mathcal{D}_R(x) = \mathcal{F}^*_a(x) + \mathcal{F}^*_b(x)$
(4) Mark points with $\mathcal{D}_R(x) = \min_x \mathcal{D}_R(x)$ as points on optimal route $\mathcal{R}(a,b)$

In the WDTOCS, floating point values are used in the calculations, so there must be some tolerance in finding points with the minimum value. In the experiments, the route distance values were rounded up to the nearest integer. Instead of finding points with the single minimum value, a threshold can be used to find points on paths that are short enough.

There can be several optimal paths, and the route set $\mathcal{R}(a,b)$ contains points on any optimal path, and can directly be used only for visualizing the route. If a single path, i.e. a sequence of pixels, is needed, one can be extracted with a simple backtracking algorithm:

(1) Start from pixel b on distance map $\mathcal{F}^*_a$
(2) Move from current pixel x_n to its neighbor x_{n-1}, which fills the criteria:
    (a) Decreasing distance: $\mathcal{F}^*_a(x_{n-1}) < \mathcal{F}^*_a(x_n)$
    (b) Point on route: $x_{n-1} \in \mathcal{R}(a,b)$, i.e. $\mathcal{F}^*_a(x_{n-1}) + \mathcal{F}^*_b(x_{n-1}) = \mathcal{D}_R(x_{n-1})$
(3) Repeat from step 2 until point a with distance value 0 is reached.

The distinct path found by backtracking depends on how step (2) of the algorithm is implemented, as there can be several neighbor pixels, which can be the next point along an optimal digital path. Recursive backtracking from each neighbor fulfilling the path pixel criteria would be required to find all the paths. The first criterion for acceptable path pixel guarantees that the path search proceeds from the destination point b towards the source point a. The second criterion is needed to make sure the tracking stays on a globally optimal path. The most obvious solution to the path tracking problem would be to start from the destination point b on distance map $\mathcal{F}^*_a$ and follow decreasing distance values towards the source point a, as in the path finding example for a constrained binary DT in Ref. [17]. This approach, however, does not work in the DTOCS setting, as step lengths vary. Checking that the local distance matches the distance value difference between the two path pixels would be needed to get correct paths. The same thing is achieved by the second criterion for acceptable path pixel in the backtracking algorithm.

5. Route optimization results

The algorithm was tested on several height map images, and this section presents some results. Each step of the algorithm is included in the example in Fig. 2 to demonstrate how the algorithm proceeds. Fig. 2(a) is the original gray-level image, and Fig. 2(b) and (c) show the DTOCS-images $\mathcal{F}^*_a(x)$ and $\mathcal{F}^*_b(x)$ calculated from the end points a and b (marked with 'x'). Fig. 2(d) shows the route distance image, i.e. the sum of the DTOCS-images, and Fig. 2(e) the optimal route by the DTOCS. It can be seen in Fig. 2(f) that the corresponding WDTOCS-route is sharper than the DTOCS-route.

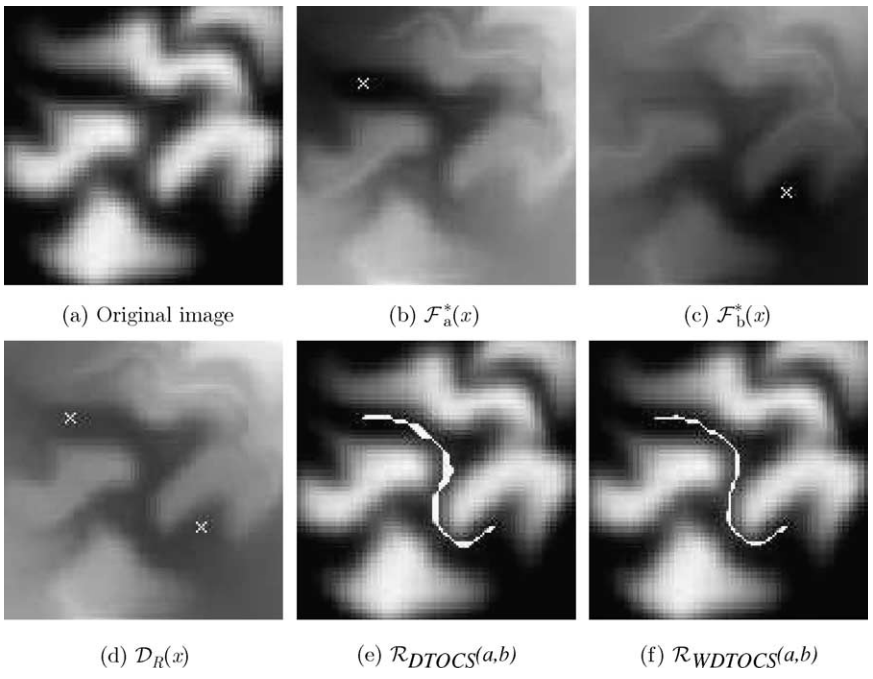


Fig. 2. (a) Original image, (b) DTOCS distance from source point, (c) DTOCS distance from destination point, (d) sum of distance images, (e) DTOCS route and (f) corresponding WDTOCS route.

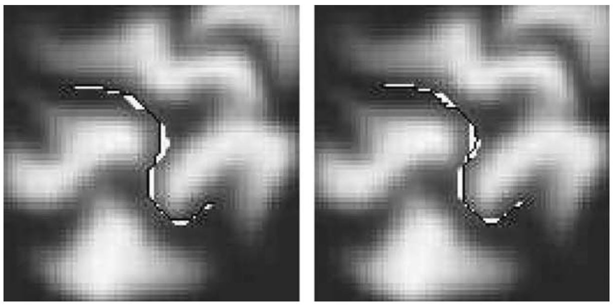


Fig. 3. Two different optimal paths on the same DTOCS route.

Fig. 3 shows two different results of the path tracking algorithm, i.e. two distinct optimal digital paths connecting the end-points of the same DTOCS route as in Fig. 2(e). The difference is caused by a different choice of path pixel among neighbors fulfilling the path pixel criteria in step 2 of the path tracking algorithm. Both paths consist of pixels belonging to the route set.

The effect of narrowing the route with more accurate local distances can be seen in Fig. 4(a) and (b). Fig. 4(c) is included to demonstrate clearly how the routes computed with the DTOCS transforms differ from routes that could be computed with other gray-level distance transforms. Here the GRAYMAT algorithm was used, which is similar to the geodesic time distance used in Ref. [16], i.e. it adds gray-level values along the path. In a terrain navigation application such a distance transform would be quite meaningless. It would prefer routes passing via deep valleys, even though the real shortest route follows a high ridge, as in the example image. The distances are inherently different in minimal cost paths, where gray-levels correspond to the cost of traversing the pixel, and in the DTOCS paths, where actual distances along the varying height surface are estimated using gray-level differences.

A sample application of solving a labyrinth with the Route DTOCS (Fig. 5) also shows how using accurate local distances improve the results. The labyrinth image was segmented into low ‘paths’ and very high ‘walls’, and the Route DTOCS was used to find a path through the labyrinth. In the WDTOCS route, the extra 90° corners seen in the DTOCS route are eliminated. Using the Route DTOCS in an obstacle avoidance problem like this is very similar to the idea of the Constrained DT, Ref. [15]. The constraint pixels are marked with very high values, and paths avoid the obstacles without any changes in the algorithm. The DTOCS approach also provides a possibility for several levels of obstacles, i.e. some that can absolutely not be crossed marked with infinite values, and others that can be crossed but with a higher cost. This is not a feasible option with constrained distance transforms.

The test image in Fig. 6 is a gray-scale ball constructed as a digitization of a mathematical sphere. The height or gray-value at the center corresponds to the radius of the sphere, and the values decrease toward the edges reaching zero at the circumference of the sphere. The tests demonstrate the same differences between the distance transform definitions as the previous examples.

The routes between the ‘north pole’ and the ‘south pole’ of the half-sphere are too wide, when calculated with the chessboard DTOCS (Fig. 6(e)), but introducing the factors 3 and 4 to the square and diagonal neighbor distances makes the route as narrow as can be expected of discrete 8-connected paths around the ball (Fig. 6(f)). Using the WDTOCS with Euclidean local distances changes the result dramatically, i.e. the algorithm finds the route across the ball rather than around it (Fig. 6(g)). The route by the new Optimal WDTOCS looks quite similar (Fig. 6(h)), but at closer look, the route distance values are more accurate.

As the distances between the poles of a sphere should be the same, πr, along any ‘longitude’, one way to estimate how well the different transforms approximate the distances is to compare these route lengths. Fig. 7 shows the lengths of all routes passing through the ‘equator’ of the sphere, i.e. the route distance values $\mathcal{D}_R(x)$ along the middle row of Fig. 6(a)–(d). It can be seen that all the distance transformations produce overestimates of the correct route length, except the DTOCS, which overestimates distances across the ball but underestimates distances around it. The 3-4-DTOCS produces the largest overestimates of distances across the sphere, but the route lengths around the sphere along the horizontal plane are quite accurate, as they should be, according to the results on best neighbor distances in Ref. [9]. The error of the 3-4-DTOCS increases

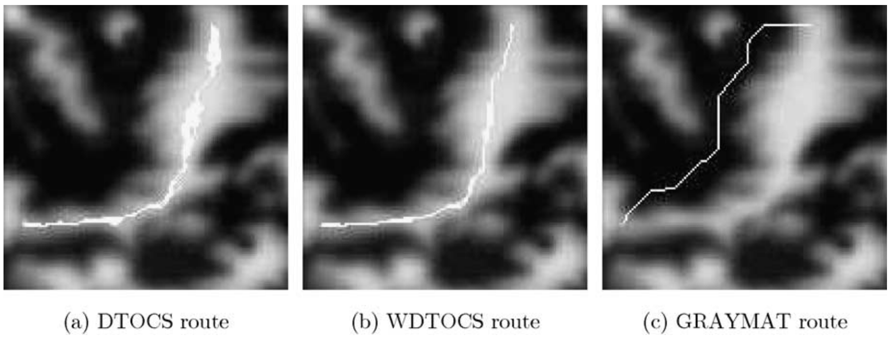

(a) DTOCS route (b) WDTOCS route (c) GRAYMAT route

Fig. 4. Terrain height map representing a ridge surrounded by valleys.

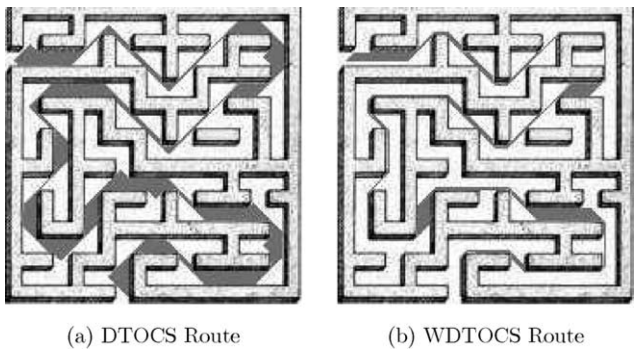


Fig. 5. An obstacle avoidance example: labyrinth solving with DTOCS.

quickly, when there is some height variation involved. The WDTOCS route lengths are clearly more accurate than the integer approximations produced by the DTOCS and the 3-4-DTOCS. The Optimal WDTOCS produces the best results, approximating both the route length across the ball and around the ball more accurately than the other transforms. The route length straight across the ball with no diagonal displacements is very close to the accurate value πr, i.e. the optimal weight a_{opt} for the local horizontal distance almost eliminates the digitization error.

When height differences increase, the effect the horizontal displacements have on the distance values decreases in the WDTOCS and the Optimal WDTOCS, whereas it stays constant in the DTOCS and the 3-4-DTOCS. The application determines which approach is better. If the transformation is used to approximate actual distances along a real surface, the piecewise Euclidean distances of the WDTOCS produces more accurate distance estimates, and the Optimal WDTOCS can decrease the error in the horizontal component of the distance values. If, on the other hand, the height differences represent another type of cost than the horizontal displacements, the DTOCS and the 3-4-DTOCS, in which the height difference is treated as a separate term, can work better and be more easily scalable. To modify the effect the height variation has on the distance values, the original image can be scaled before applying the transformation. Alternatively, the height difference in Eqs. (1)–(4) can be weighted with a suitable factor.

Fig. 8 demonstrates a simple surface, where the WDTOCS algorithm gives more accurate distance values than could be achieved with a 3D distance transform on the corresponding umbra. The image represents a slope, where the height difference between neighbor pixels is 2. The WDTOCS path from the point (x,y) with gray-level value z to the point $(x+2,y)$ with gray-level value $z+4$ consists of two steps of length $\sqrt{2^2+1}$, i.e. the length of the path is $2\sqrt{5}\approx 4.47$. The corresponding path length using the 3D distance transform with local distances 3 and 4 by Ref. [11] would require four steps, two of length 3 and two of length 4, and scaled down with 3, the distance becomes $14/3\approx 4.67$. Even with the accurate optimal values for neighbor distances a_{opt} and b_{opt}, the distance is over-estimated (≈ 4.65) compared to the WDTOCS value. If the image is a digitization of a surface with a constant slope representable with a rational number (whole pixels), the WDTOCS distance is an errorfree estimate. At best, the WDTOCS and the Optimal WDTOCS can smooth away some error caused by digitization of an object, and compute distances, which are good approximations of geodesic distances on the original continuous surface.

An obstacle avoidance problem similar to the labyrinth is presented in Fig. 9, showing an example of route optimization through a city. The gray-levels do not correspond to actual heights, but the differences between

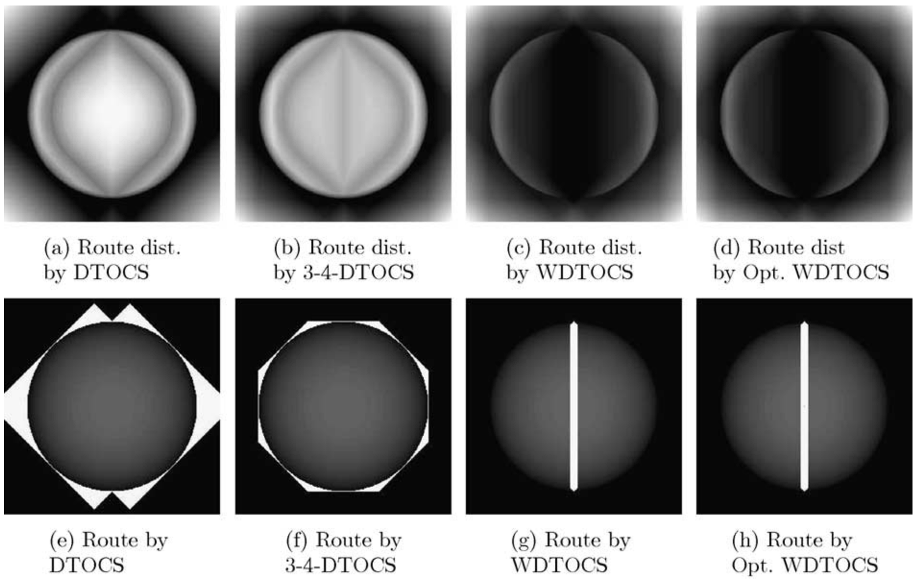


Fig. 6. Route distances and resulting routes with various distance definitions: DTOCS, 3-4-DTOCS, WDTOCS and Optimal WDTOCS.

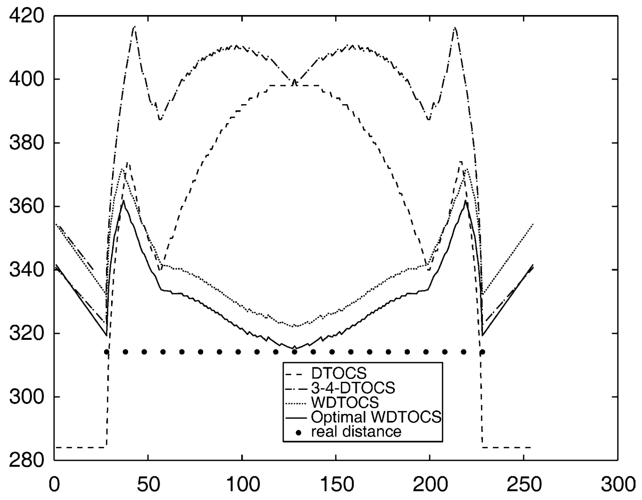


Fig. 7. Comparison of route lengths from pole to pole on a digital sphere using different distance transform definitions. The correct route length is π(r is also plotted for comparison.

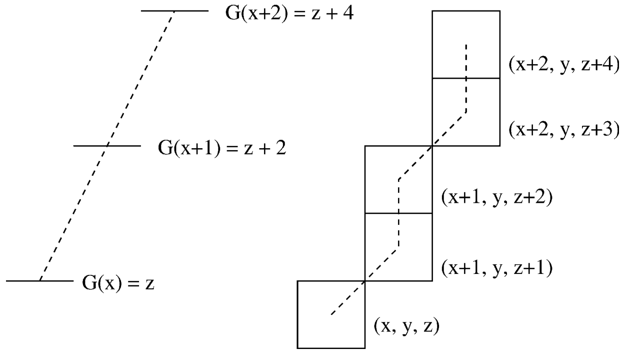


Fig. 8. A simple slope represented with gray-level values (left) and with a top surface of an umbra in 3D, marked with coordinates of the voxels (right).

roads and other image areas must be large enough to make sure the shortest paths follow the roads. The example demonstrates how the Route DTOCS algorithm, in addition to finding all shortest paths between two points, simultaneously computes all route lengths between the points. For example, here the length of the optimal route is 193 units. The length of the shortest route passing through the point marked with ‘x’ in Fig. 9(a) is the value of the route distance at that point, i.e. 281 units. If the value is good enough (it is worth making the detour) optimal subroutes can be computed by calculating the Route DTOCS distance from the source point to the detour point, and from the detour point to the destination, as can be seen in Fig. 9(b) and (c).

6. Discussion

The DTOCS algorithm has previously been used to approximate distances locally, in small parts of the image. In image compression, Ref. [18], distance values provide an estimate on how much variation there is on the image surface. More control points are stored from image areas, where gray-level values change rapidly. In such applications, the chessboard distance transform works well enough. However, the route optimization algorithm computes global distances across the whole image, and the approximation error of the chessboard distance accumulates. If the slightly heavier computation of floating point values instead of integers is not a problem, the WDTOCS or the Optimal WDTOCS algorithm should be used to get the best results in route optimization. A question for future research is whether we can define integer kernel distances, which approximate the Euclidean distance well enough. The local horizontal distances 3 and 4 for square and diagonal pixel neighbors improve the results significantly compared to the chessboard DTOCS, but not enough to achieve the performance of the WDTOCS. Ideas from 3D distance transforms as in Ref. [11] could be applied, but the problem with the DTOCS approach is that there is a large number of different step lengths, i.e. two for each possible height difference value. When height differences are large, the relative effect of the horizontal displacement is so small that in practice integer approximations of square and diagonal local distances would be the same, unless very large integer factors are used.

Improving global distances by increasing the mask size is also an option for the DTOCS algorithms, but cannot be applied without some risks. A very narrow peak, in practice a one pixel wide obstacle, could be missed when using

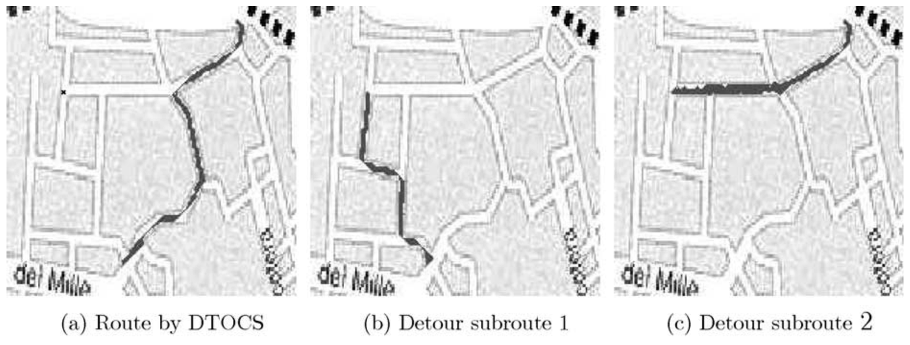


Fig. 9. Navigation example, where the route via a detour point is found.

a 5×5 mask with the 'knight's move' (Ref. [17]). The same problem makes analysis of distance transform regularity not directly applicable to the DTOCS algorithms. Distance transforms are defined to be regular or semi-regular if a straight line represents the distance between two pixels (Ref. [11]). In the DTOCS transforms, the straight line distance can be blocked with pixels that are higher or lower than the surrounding pixels, and the shortest path goes around the obstacle. This feature complicates analysis of the transform, but can also be useful in some applications. Obstacle avoidance can easily be implemented with several different levels of obstacles, separating completely illegal paths, and paths that should be avoided but not at any cost. Constrained distance transforms can only handle one type of constraints, pixels which cannot be crossed.

The Route DTOCS presented in this article is an exceptionally simple method for finding shortest routes along a surface. Despite the simple implementation, the time complexity analysis of the algorithm is not straightforward. Each iteration of the DTOCS and its variations can be done in linear time, as well as the addition of the distance images, and finding the points with the minimal route distance value. But repeating the distance transformation passes until the globally optimal distance map is produced introduces an unpredictable factor to the complexity, as the size and variance of the image surface, as well as the lengths of the paths affect the convergence. The worst case scenario would require as many iterations as there are pixels on the path with the most pixels. Typically convergence is reached in about 10 iterations, or for very complex surfaces in about 20–30. This is still a small number compared to the number of pixels in the images, i.e. the complexity is considerably lower than the $\mathcal{O}(n^2)$ of the basic Dijkstra algorithm, when the n pixels represent n vertices in the graph search. The graphs corresponding to digital images are sparse, however, as there are edges only between neighbor pixels, and the time complexity of Route DTOCS is comparable to more efficient shortest path algorithms applicable for sparse graphs.

A question for future research is how much the route distance values change during subsequent iterations of the distance transformation, i.e. in some applications the first estimate of the route computed with just one or a few iterations may be good enough. This provides a possibility for progressive path optimization. Another option to get a more efficient method would be to abandon the chamfering, and compute the distance transform for example with a modification of the ordered propagation algorithm in Ref. [15] or the pixel-queue algorithm in Ref. [19]. The implementation would, however, be more complex for the DTOCS than for binary distance transforms, as steps have different lengths. The pixel queue would have to be implemented as a priority queue to process the pixel with the smallest current distance first. Otherwise paths with many short steps might be missed, and wrong distance values via paths with few longer steps might end up in the final distance map.

The DTOCS algorithms process two-dimensional images, which essentially represent three-dimensional surfaces. As distance transforms for three-dimensional images are well known, and thoroughly researched, it is a tempting—and a quite feasible—option to transform the gray-level height map back to its original domain, a surface in 3D space. We suggest the opposite approach—computing distances along 3D surfaces using 2D gray-level images, as a more efficient and in some cases more accurate method than the 3D distance transforms. Further research on the new Optimal DTOCS is required to estimate how well it performs generally compared to the well-established 3D distance transforms with optimal local distances.

Acknowledgements

Thanks to two anonymous reviewers, whose critical comments helped improve this article considerably, and provided valuable pointers for future research.

References

[1] R. Kimmel, N. Kiryati, Finding shortest paths on surfaces by fast global approximation and precise local refinement, International Journal of Pattern Recognition and Artificial Intelligence 10 (1996) 643–656.
[2] Y. Saab, M. VanPutte, Shortest path planning on topographical maps, IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans 29 (1) (1999) 139–150.
[3] P. Saha, F. Wehrli, B. Gomberg, Fuzzy distance transform: theory. Algorithms and applications, Computer Vision and Image Understanding 86 (2002) 171–190.
[4] P. Toivanen, New geodesic distance transforms for gray-scale images, Pattern Recognition Letters 17 (1996) 437–450.
[5] R. Kimmel, A. Amir, A. Bruckstein, Finding shortest paths on surfaces using level sets propagation, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (6) (1995) 635–640.
[6] P. Lin, S. Chang, A shortest path algorithm for a nonrotating object among obstacles of arbitrary shapes, IEEE Transactions on Systems, Man, and Cybernetics 23 (3) (1993) 825–833.
[7] L. Ikonen, P. Toivanen, J. Tuominen, Shortest route on gray-level map using distance transform on curved space, in: Scandinavian Conference on Image Analysis (SCIA), 2003 pp. 305–310.
[8] L. Ikonen, P. Toivanen, Shortest route on height map using gray-level distance transforms, in: Discrete Geometry for Computer Imagery (DGCI), 2003 pp. 308–316.
[9] C. Borgefors, Distance transformations in digital images, Computer Vision, Graphics, and Image Processing 34 (1986) 344–371.
[10] S. Sternberg, Grayscale Morphology, Computer Vision, Graphics, and Image Processing 35 (1986) 333–355.
[11] G. Borgefors, On digital distance transforms in three dimensions, Computer Vision and Image Understanding 64 (3) (1996) 368–376.
[12] N. Kiryati, G. Székely, Estimating shortest paths and minimal distances on digitized three-dimensional surfaces, Pattern Recognition 26 (11) (1993) 1623–1637.

[13] A. Rosenfeld, J.L. Pfaltz, Sequential operations in digital picture processing, Journal of the Association for Computing Machinery 13 (4) (1966) 471–494.

[14] P. Toivanen, Convergence properties of the distance transform on curved space (DTOCS), in: Finnish Signal Processing Symposium, 1995 pp. 75–79.

[15] J. Piper, E. Granum, Computing distance transformations in convex and non-convex domains, Pattern Recognition 20 (6) (1987) 599–615.

[16] P. Soille, Generalized geodesy via geodesic time, Pattern Recognition Letters 15 (12) (1994) 1235–1240.

[17] G. Borgefors, Applications using distance transforms in: Arcelli, Cordella, S. di Baja (Eds.), Aspects of Visual Form Processing, World Scientific, Singapore, 1994, pp. 83–108.

[18] P. Toivanen, Image compression by selecting control points using distance function on curved space, Pattern Recognition Letters 14 (1993) 475–482.

[19] J. Silvela, J. Portillo, Breadth-first search and its application to image processing problems, IEEE Transactions on Image Processing 10 (8) (2001) 1194–1199.

Publication V

Ikonen, L.,
Pixel Queue Algorithm for Geodesic Distance Transforms

Pixel Queue Algorithm for Geodesic Distance Transforms

Leena Ikonen

Lappeenranta University of Technology,
Department of Information Technology,
PO Box 20, 53851 Lappeenranta, Finland
leena.ikonen@lut.fi

Abstract. Geodesic distance transforms are usually computed with sequential mask operations, which may have to be iterated several times to get a globally optimal distance map. This article presents an efficient propagation algorithm based on a best-first pixel queue for computing the Distance Transform on Curved Space (DTOCS), applicable also for other geodesic distance transforms. It eliminates repetitions of local distance calculations, and performs in near-linear time.

1 Introduction

Distance transformations were among the first operations developed for digital images. Sequential local transformation algorithms for binary images were presented already in the 1960s [8], and similar chamfering techniques have been used successfully in 2D, 3D and even higher dimensions, see e.g. [2], [3], [1]. By modifying the definitions local distances, the chamfering can be applied to gray-level distance transforms as well. The Distance Transform on Curved Space (DTOCS) and its locally Euclidean modification Weighted DTOCS (WDTOCS), which compute distances to nearest feature along a surface represented as a gray-level height map, have been implented as mask operations [12].

Instead of propagating local distances in a predefined scanning order, the distance transformation can begin from the set of feature pixels, and propagate to points further away in the calculation area. A recursive propagation algorithm was presented in [7], where the distance value propagates from the previously processed neighbor. If the new value is accepted into the distance map, i.e. it is smaller than the previous distance value of the same pixel, the procedure is repeated recursively for each neighbor. The efficiency of the recursive propagation is highly dependent on the order in which the neighbors are processed. An unwise or unlucky choice of propagation order causes numerous repetitions of distance calculations, as shorter paths are found later on in the transformation. The ordered propagation algorithm, also presented in [7], eliminates some of the repetitions. First the boundary of the feature set, and then neighbors of already processed pixels, are placed in a queue, from which they are then taken to be processed in order. Similar pixel queue algorithms are also presented in [9] and [14].

The recursive and ordered propagation, and pixel queue algorithms, can be seen as applications of graph search, where each pixel represents a vertex, and edges exist between neighbor pixels. Local distances can be defined as weights of connecting edges. The recursive propagation proceeds as a depth-first-search, and first-in-first-out pixel queue algorithms are applications of breadth-first-search. This article presents a best-first-search algorithm for computing gray-level distance transforms based on a priority queue, which is implemented efficiently as a minimum heap. A distance transform algorithm utilizing the priority queue idea was presented in [13]. Bucket sorting is used to find the pixel with the smallest current distance. The algorithm is applicable only for integer distances, as a separate storing bucket is needed for each distance value. Our heap based priority queue works for any distance values, including the real valued modifications of the DTOCS. Experiments demonstrate that convergence of the sequential transformation as well as the ordered propagation algorithm is highly dependent on the image size and complexity, whereas the near-linear pixel queue algorithm slows down only slightly with increasing surface variance.

2 Distance Transforms

The Distance Transform on Curved Space (DTOCS) calculates distances along a gray-level surface, when gray-levels are understood as height values. Local distances are defined using gray-level differences. The basic DTOCS simply adds the gray-level difference to the chessboard distance in the horizontal plane, i.e. the distance between neighbor pixels is:

$$d(p_i, p_{i-1}) = |\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})| + 1 \tag{1}$$

where $\mathcal{G}(p)$ denotes the gray-value of pixel p, and p_{i-1} and p_i are subsequent pixels on a path. The locally Euclidean Weighted DTOCS (WDTOCS) is calculated from the height difference and the horizontal distance using Pythagoras:

$$d(p_i, p_{i-1}) = \begin{cases} \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + 1}\ , & p_{i-1} \in N_4(p_i) \\ \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + 2}\ , & p_{i-1} \in N_8(p_i) \setminus N_4(p_i) \end{cases} \tag{2}$$

The diagonal neighbors of pixel p are denoted by $N_8(p) \setminus N_4(p)$, where $N_8(p)$ consists of all pixel neighbors in a square grid, and $N_4(p)$ of square neighbors. More accurate global distances can be achieved by introducing weights, which are proven to be optimal for binary distance transforms, to local distances in the horizontal plane. The Optimal DTOCS is defined in [6] as

$$d(p_i, p_{i-1}) = \begin{cases} \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + a_{opt}^2}\ , & p_{i-1} \in N_4(p_i) \\ \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + b_{opt}^2}\ , & p_{i-1} \in N_8(p_i) \setminus N_4(p_i) \end{cases} \tag{3}$$

where $a_{opt} = (\sqrt{2\sqrt{2} - 2} + 1)/2 \approx 0.95509$ and $b_{opt} = \sqrt{2} + (\sqrt{2\sqrt{2} - 2} - 1)/2 \approx 1.36930$ as derived in [2] by minimizing the maximum difference from the Euclidean distance that can occur between points on the binary image plane.

3 Pixel Queue Transformation Algorithm

Pixel queue algorithms are simple to implement for binary distance transforms. With equal step lengths the distances propagate smoothly from the feature set outwards, and the distance corresponds to the number of steps. As step lengths vary in the DTOCS transformations, several short steps along a smooth area of the image can create a shorter path than just one or a few steps along an area with high variance. Distances can not propagate as pixel fronts moving outwards from the feature set, or a path with a few long steps might be found instead of a shorter path consisting of many short steps. Both recursive and ordered propagation algorithms can compute the correct global distances also in the DTOCS setting, if neighbors of updated pixels are processed whether or not they have been processed before. However, this is very inefficient, as numerous repetitions of local distance calculations are needed. The new efficient **pixel queue algorithm** utilizes a priority queue implemented as a minimum heap:

```
1. Define binary image $\mathcal{F}(x) = 0$ for each pixel $x$ in feature set, and $\mathcal{F}(x) =$
   $max$ for each $x$ in calculation area.
2. Put feature pixels (or boundary) to *priority queue* Q.
3. While Q not empty
       $p = dequeue(Q)$, $\mathcal{F}_q(p)$ was the smallest distance in Q.
       If $\mathcal{F}_q(p) > \mathcal{F}(p)$ (obsolete value), continue from step 3.
       $\mathcal{F}(p)$ becomes $\mathcal{F}^*(p)$ (value is final).
       For neighbors $x$ of $p$ with $\mathcal{F}(x) > \mathcal{F}^*(p)$
           Compute local distance $d(p,x)$ from original image $\mathcal{G}$
           If $\mathcal{F}^*(p) + d(p,x) < \mathcal{F}(x)$
               Set $\mathcal{F}(x) = \mathcal{F}(p) + d(p,x)$
               $enqueue(x)$
           end if
       end for
   end while
```

The initialization of the queue can be implemented in two different ways without affecting the result. Only feature boundary pixels need to be enqueued in the initial step, but enqueueing all feature pixels yields the same result. Processing non-boundary feature pixels does not cause any changes in the distance image, and hence no further enqueueings of neighbor pixels. The application determines which approach is more efficient, e.g., if distances from the background into a small object are calculated, the external boundary of the object should be used rather than enqueueing the whole background.

The best-first approach eliminates repetition of local distance calculations. Using the priority queue ensures that the propagation always proceeds from a point, which already has its final distance value. As local distances, which by definition are non-negative, are added to distance values taken from the queue, the currently smallest distance can never decrease further. So once a pixel is dequeued, it will not be enqueued again. However, as step lengths vary, a distance value that has propagated from a point with a final optimal value, may still be replaced with a smaller one. Small local distances can create new shorter paths. This will cause the same pixel to be enqueued repeatedly, first with a larger distance value and then with smaller ones, before the first instance has been dequeued. Once the final distance value is dequeued, other instances of the pixel in the queue become obsolete, and could be removed. However, it is easier to just discard them when they are dequeued in the normal priority queue order. Not processing neighbors x of point p, which already have a distance value smaller or equal to $\mathcal{F}(p)$, eliminates a significant amount of local distance calculations, including the reverse directions of previously calculated distances, i.e., if $d(p_i, p_j)$ is calculated during the transformation, $d(p_j, p_i)$ will never be needed.

The local distances are treated similarly as in the pixel queue transformation in [9]. The current pixel is considered the source point, and new distance values are assigned to all neighbors, for which the path via the source point is the shortest found so far. The recursive and ordered propagation algorithms in [7] as well as the sequential transforms view the current point as the destination with each neighbor as a possible source. Local distances from all neighbors within mask must be calculated to obtain one new distance value. The "greedy" approach of calculating distances forward from a source point was tested also for the sequential algorithm, but the effect on convergence was insignificant.

4 Complexity Analysis

The forward and reverse pass of a sequential local transformation can be done in linear time, as there is a constant number of operations per pixel. The problem with the complexity analysis is that the number of passes needed varies a lot depending on the size and the complexity of the image surface. Smooth and simple images can usually be transformed in just a few iterations, but it is possible to construct example images, which require one iteration for each pixel on the path with the most pixels. Typical values for test images in our previous works have been about 10-15 two-pass-iterations, which for an image of size 128×128 is in the ballpark of $\log n = 14$, which would make the whole algorithm about $\mathcal{O}(n \log n)$. However, with larger images and more complex surfaces, the number of iterations needed increases. The Experiments section will present 512×512 example images converging in about 70 iterations, which is clearly more than $\log n = 18$.

The priority queue transformation propagates local distances from each pixel only when it is dequeued with its final distance value. This means that each local distance in the image is computed only once, or some not at all, if neighbor pixels can be discarded due to already smaller distance values. Sequential algorithms recalculate each local distance at each iteration, which can be very costly, especially in transformations requiring heavier floating point calculations, like the WDTOCS. Updating the priority queue adds a factor to the computation time, as each enqueueing and dequeueing takes $\mathcal{O}(\log n_q)$ time, where n_q is the number of pixels in the queue. The value n_q varies through the transformation representing the boundary of the area, where distances are already calculated.

An upper limit on the complexity can be estimated using the fact that at each step after dequeueing one pixel, at most 7 pixels can be enqueued. The path through the current point must come from somewhere, so at least one neighbor must already have its final value. At each step one pixel value becomes final, so the number of efficient steps is $n - n_f$, where n_f is the number of feature pixels. Even with the extra enqueueings, and dequeueings of obsolete pixels, the number of steps is in $\mathcal{O}(n)$, which makes the complexity of the whole algorithm $\mathcal{O}(n \log n_q)$, or worst case complexity $\mathcal{O}(n \log n)$. The theoretically maximal queue length, about $6n$, is a gross overestimate, as distances propagate locally as pixel fronts, which means that in practise only about half the neighbors of a pixel are enqueued with new distance values. Also after the $n - n_f$ efficient steps leaving one final distance value, the queue should be empty, and certainly not at its maximum length. Experimental results will provide a more realistic estimate on the number of queue operations and the average length of the queue.

5 Experiments

The priority queue algorithm was tested on gray-level images with varying surface complexity to compare with the sequential local transformation, and also with the ordered propagation algorithm implemented with a first-in-first-out pixel queue, like in [9]. The distance images were compared to make sure they were identical - and at first they were not. The sequential implementation calculated distances only at points, where the whole mask fit on the image, so errors appeared in areas, where the shortest path from the feature passed via edge pixels. Instead of modifying the mask at the edges, the border effects were corrected by adding an extra row or column to each edge before the mask transformation, copying the edge values to the corresponding extra row or column. With this correction the distance images were identical for the DTOCS, and within calculation accuracy tolerance for the WDTOCS. The pixel queue algorithms propagate distances to existing neighbors, so distances near edges are calculated correctly without tricks.

The performance of the algorithms was compared using the images seen in Fig. 1. The Mercury height map, Fig. 1 a), and the Lena image, Fig. 1 b), represent highly varying surfaces. The Lena image is obviously not an actual height map, but is used similarly in these tests. The Ball image, Fig. 1 c), is constructed as a digitization of the sphere function, i.e. the highest gray-value in the center corresponds to the radius of the sphere. The fourth test image, Flat, consists of a constant gray-value representing the smoothest surface possible. Testpoint grids were created (see example on the Ball image, Fig. 1 c), and distances from one testpoint to everywhere else in the image were calculated. The grids contained 244 points, and averages calculated from these 244 independent runs are visualized in figures 2 - 6. The sequential algorithm was faster only for the integer DTOCS on the Flat images. The larger and more complex the surfaces were, the more clearly the pixel queue algorithm outperformed the sequential transformation, and also the ordered propagation. The ordered propagation was

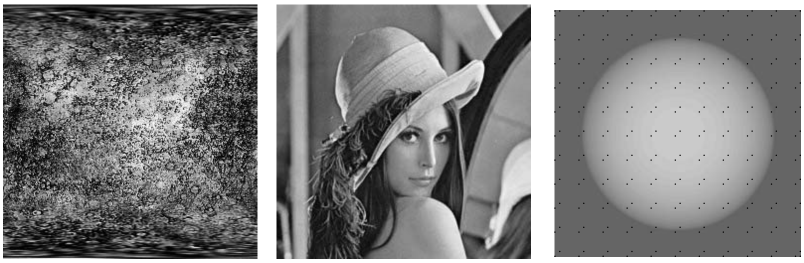
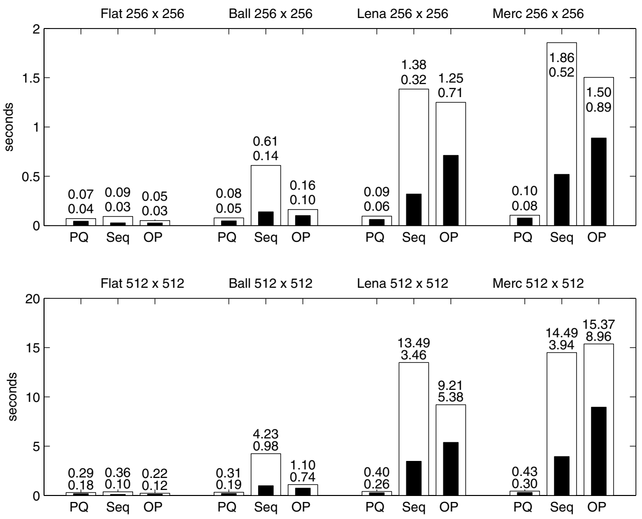


Fig. 1. Test images used. An example of a test point grid is shown on the Ball image

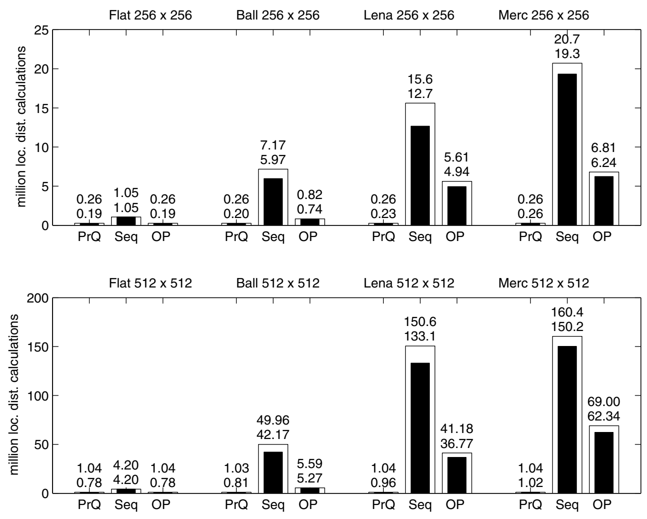


[p]

Fig. 2. Average run times of DTOCS (black bar) and WDTOCS (white bar) using Priority Queue, Sequential and Ordered Propagation algorithms

slightly faster than the sequential algorithm in most cases, as despite numerous repeated pixel enqueuings, processing all pixels several times in the sequential transformations is more costly. For very smooth surfaces where distances proceed evenly as pixel fronts, the ordered propagation is faster than the priority queue, as first-in-first-out queue operations take constant time.

The run times (Fig. 2), and the number of local distance calculations (Fig. 3) are proportional to the number of iterations in the sequential algorithms, and the number of iterations needed grows with the size and the complexity of the image (Fig. 4). The pixel queue transformation eliminates a lot of computation by calculating only those local distances, which are needed. If each local distance was calculated exactly once, the $256 * 256$ images would require 260610



[p]

Fig. 3. Average number of local distance calculations needed in DTOCS (black bar) and WDTOCS (white bar) using Priority Queue, Sequential and Ordered Propagation algorithms

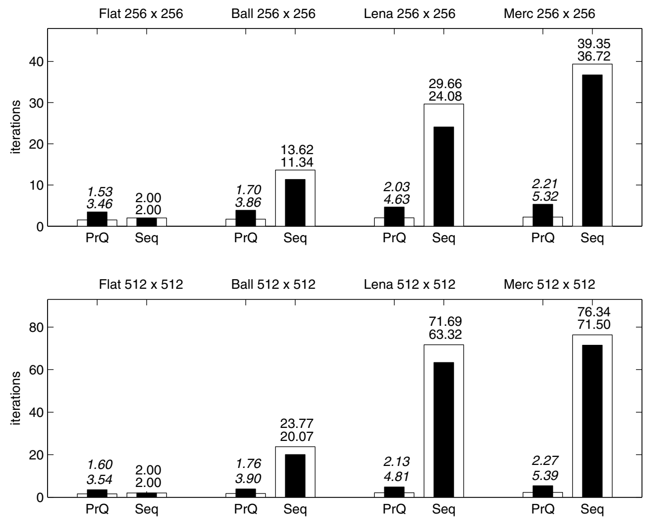


Fig. 4. Average number of iterations needed in sequential DTOCS (black bar) and WDTOCS (white bar). The number of iterations indicated for the pixel queue algorithm is a comparison number calculated from the run times

local distances, and the $512 * 512$ images 1045506 ($rows * (columns - 1)$ horizontal, $columns * (rows - 1)$ vertical, and $2 * (rows - 1) * (columns - 1)$ diagonal distances). Each iteration of the sequential transformation calculates each of these local distances twice, once in both directions. Some local distance calculations could have been eliminated from the first iteration by scanning the image to the feature pixel without calculating distances, saving about half an iteration.

The only source for repetition in the priority pixel queue algorithm is the calculation accuracy of floating point distance transforms. A distance value may be considered new, and consequently the pixel enqueued, even if it is smaller than the previous value only because of computation accuracy. Despite adding a threshold to the comparisons (the new value must be 0.001 smaller to be accepted as new), a few pixels ended up being enqueued repeatedly in the complex surfaces, e.g., the number of enqueueings minus the number of obsolete pixels found from the queue was at most 262190 for the $512 * 512$ Mercury surface of 262144 pixels. In the WDTOCS transformations of the smooth images, and of course in all the DTOCS transformations, the number of enqueueings minus the number of obsolete pixels equals the number of pixels.

The running times of C-implementations of the algorithms on a Linux computer with an AMD Athlon 1.678 GHz processor indicate that particularly for the floating point WDTOCS distances the pixel queue algorithm is superior-ior. The speed of the priority queue operations, enqueue and dequeue, is not affected by the choice of floating point versus integer distances, so the relative cost of repeating the local distance calculations in numerous iterations is higher when using floating point values. In addition, the WDTOCS typically requires a few more iterations, causing even more repetitions. For example for the Mercury height map of size $512 * 512$ the speedup of the pixel queue transform compared to the sequential transform is $3.94/0.30 \approx 13$ for the integer DTOCS and $14.49/0.43 \approx 34$ for the floating point WDTOCS. The Optimal DTOCS was not tested here, as one integer and one floating point distance transform were enough to demonstrate the efficiency of the pixel queue algorithm. The advantage would be even more clear in the case of the Optimal DTOCS, which requires an additional multiplication operation to calculate each local distance.

The number of iterations marked for the pixel queue algorithm in Fig. 4 is calculated as the number of sequential iterations that could have been performed in the time consumed by the pixel queue algorithm. As the running time for one iteration should be constant for a certain image size and local distance definition, the comparison number can be used to estimate how much the performance of the pixel queue algorithm depends on the complexity of the image surface. The value ranged in the DTOCS tests of $512 * 512$ images from 3.54 (Flat image) to 5.39 (complex Mercury surface), while the number of iterations of the sequential DTOCS ranged from 2 to 71.50. This means that the running time of the pixel queue algorithm is much better predictable. One larger image, the Mercury $768 * 768$ surface, was tested to provide experimental basis to the claim of near-linear complexity. The average runtimes were 0.66 and 1.01 seconds for the priority queue DTOCS and WDTOCS, and 9.52 and 41.72 seconds for the sequential DTOCS and WDTOCS. Compared to the $256 * 256$ images, the corresponding $512 * 512$ images took about 4 times longer to transform with the priority pixel queue algorithm, and the fact that the $768 * 768$ Mercury image took about 9 times longer than the $256 * 256$ image suggests a continuing linear trend.

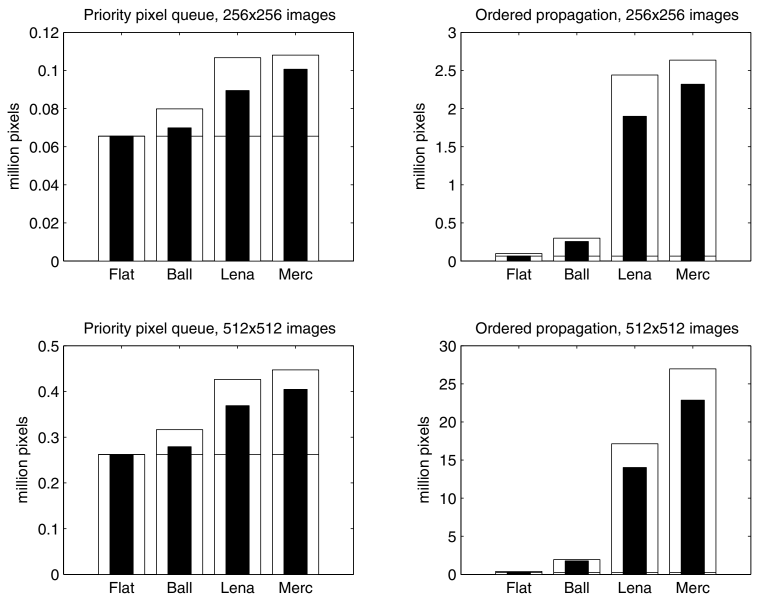


Fig. 5. Number of pixel enqueuings in DTOCS (black bar) and WDTOCS (white bar) for the priority queue (left) and the ordered propagation (right). The horizontal line on each bar indicates the number of pixels in the image, so the section of the bar above the line shows how many pixels get enqueued repeatedly. Notice the different scales

More statistics on the pixel queue transformation are shown in Fig. 5 and Fig. 6. The number of enqueued pixels, i.e. the number of enqueue and dequeue operations, is somewhat larger than the number of pixels. The more complex the surface, the more pixels get enqueued repeatedly when new shorter paths are found. The number of pixel enqueuings in the ordered propagation algorithm is in a larger magnitude, and also grows very rapidly with the size and complexity of the image (see Fig. 5). The average and maximum queue lengths (Fig. 6) are calculated from the average and maximum queue lengths recorded at each run. The largest average and the largest maximum queue length for each test image is indicated as lines on top of the bars. The average queue lengths for the $768 * 768$ Mercury surface not included in the graphs were 5295 for the DTOCS and 6073 for the WDTOCS, and the longest queue encountered contained 14069 pixels $\ll n = 768 * 768 = 589824$. In general, the queue lengths seem to grow

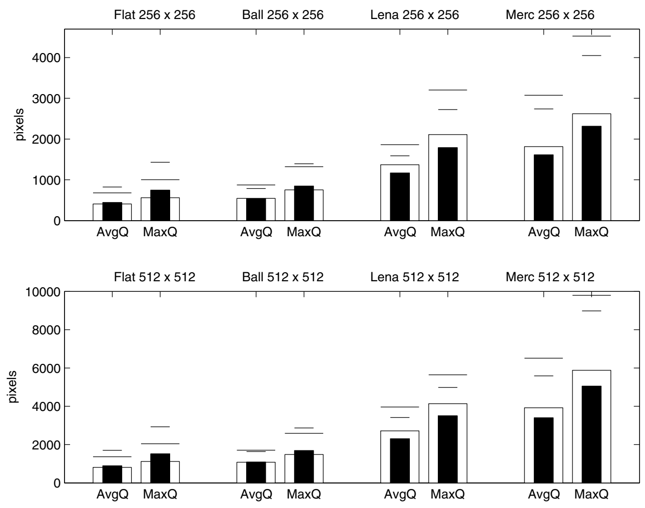


Fig. 6. Average and maximum queue lengths in DTOCS (black bar) and WDTOCS (white bar). The horizontal lines above the bars indicate the maximum values, i.e. the largest average queue length and largest maximum queue length

sublinearly with the size of the image. As queue lengths are in a clearly smaller magnitude than the number of pixels, the algorithm is in practise linear.

6 Discussion

The DTOCS algorithms have been presented as geodesic distance transforms without proper explanation on how and why they may be called geodesic. The DTOCS distances resemble geographical geodesic distances. Discrete paths follow the gray-level surface like the shortest path between two cities follow the surface of the geoid. In image analysis the term geodesic distance refers to a situation where paths linking image pixels are constrained to remain within a subset of the image plane [11]. In the DTOCS setting paths can cross any areas of the image, but path lengths can become huge. The DTOCS can be used in the same manner as constrained distance transforms, marking constraint pixels with values differing so much from the rest of the image plane that the shortest paths will never cross those pixels. In such a situation the distances propagate similarly as in a geodesic, i.e. constrained, distance transform. The pixel queue algorithm could be used to calculate both types of transforms, as well as gray-level distance transforms calculating minimal cost paths, e.g. the geodesic time transform with distances defined as the sum of gray-values along the path [10].

The presented pixel queue algorithm was demonstrated to be efficient, outperforming the sequential algorithm in almost all test cases. The running times do grow a bit with increased surface complexity, but not nearly is much as the running times of the sequential transformation. The complexity of the algorithm is $\mathcal{O}(n \log n_q)$, but as $n_q \ll n$ it performs in near-linear time. The number of local distance calculations is minimized, i.e. each local distance in the image is computed at most once, which is a clear benefit compared to the iterated sequential transforms, particulary if the local distances require heavier floating point computations.

Previous DTOCS experiments have been made on quite small images. The experiments here demonstrate the expected effect of increasing the image size, i.e. the number of iterations needed for convergence becomes quite unpredictable. The sequential transformation may still be useful, but in applications with high resolution images, the pixel queue algorithm is more efficient. Another benefit of the pixel queue approach is that distances are calculated exactly where they are needed. If, for example, an image of an object on a background is transformed, the sequential transformation calculates unnecessary distances on the background. The pixel queue algorithm naturally proceeds from the border into the object. Also, as distance values are known to be final once they are dequeued, a real time application could utilize some values before the whole transformation is done. If the feature set is disconnected, the distance values propagated from each feature will be mixed in the priority queue, but distance values near each feature will be calculated early in the transformation. When the propagating fronts meet, the transformation is final. This idea could be utilized for developing a tesselation method.

Another approach, which ensures that obtained distance values are immediately final is presented in [4]. The parallel implementation is based on the fact that in binary distance transforms each pixel with distance value N must have a neigbor with distance value $N - a$ or $N - b$, where a and b are the local distances to square and diagonal neighbors. Pixels with a 0-valued neighbor are updated first, and then pixels with a neighbor of each possible successively increasing distance value. Thus, the distance values propagate similarly as in the pixel queue transformation presented here.

Pixel queue algorithms can be implemented also in higher dimensions. For binary voxel images in 3D, as well as for binary images in 2D, where distances propagate as smooth fronts, ordered propagation with a first-in-first-out queue would probably work as well or even better than the priority queue approach. However, if the voxels have values other than 0 and 1, and path lengths are defined using voxel values on the path resulting in varying local distances, the priority queue algorithm could be useful. Larger neighborhoods, for example $5*5$ in 2D or $5*5*5$ in 3D, could be introduced to the pixel queue algorithm, but in the DTOCS setting larger neighborhoods need to be used with care, as they can result in illegal paths across very narrow obstacles.

The pixel queue algorithm could easily be modified to record the path of the shortest distance, by storing the direction from which the path propagated to each pixel. However, only the first found path would be recorded even though there are usually several equally short paths. The Route DTOCS algorithm for finding the route between two points [6] or point sets [5] requires two distance maps, one for each end-point set. The route consists of points on any optimal path, and a distinct path can be extracted using backtracking. In shortest route applications large complex images with long paths are typical, so the priority pixel queue algorithm improves the method significantly.

References

1. G. Borgefors. Distance Transformations in Arbitrary Dimensions. *Computer Vision, Graphics, and Image Processing*, 27:321–345, 1984.
2. G. Borgefors. Distance Transformations in Digital Images. *Computer Vision, Graphics, and Image Processing*, 34:344–371, 1986.
3. G. Borgefors. On digital distance transforms in three dimensions. *Computer Vision and Image Understanding*, 64(3):368–376, 1996.
4. G. Borgefors, T. Hartmann, and S. L. Tamimoto. Parallell distance transforms on pyramid machines: theory and implementation. *Signal Processing*, 21(1):61–86, 1990.
5. L. Ikonen and P. Toivanen. Shortest routes between sets on gray-level surfaces. In *Patter recognition and Image Analysis (PRIA)*, pages 244–247, St. Petersburg, Russia, October 2004.
6. L. Ikonen and P. Toivanen. Shortest routes on varying height surfaces using gray-level distance transforms. *Image and Vision Computing*, 23(2):133–141, February 2005.
7. J. Piper and E. Granum. Computing Distance Transformations in Convex and Non-convex Domains. *Pattern Recognition*, 20(6):599–615, 1987.
8. A. Rosenfeld and J. L. Pfaltz. Sequential Operations in Digital Picture Processing. *Journal of the Association for Computing Machinery*, 13(4):471–494, October 1966.
9. J. Silvela and J. Portillo. Breadth-first search and its application to image processing problems. *IEEE Transactions on Image Processing*, 10(8):1194–1199, 2001.
10. P. Soille. Generalized geodesy via geodesic time. *Pattern Recognition Letters*, 15(12):1235–1240, 1994.
11. P. Soille. *Morphological Image Processing: Principles and Applications.* Springer-Verlag, 2 edition, 2003 and 2004.
12. P. Toivanen. New geodesic distance transforms for gray-scale images. *Pattern Recognition Letters*, 17:437–450, 1996.
13. Ben J. H. Verwer, Piet W Verbeek, and Simon T. Dekker. An efficient uniform cost algorithm applied to distance transforms. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11(4):425–429, April 1989.
14. L. Vincent. New trends in morphological algorithms. In *Proc. SPIE/SPSE*, volume 1451, pages 158–170, 1991.

Publication VI

Ikonen, L., Toivanen, P.,
Distance and Nearest Neighbor Transforms of Gray-Level Surfaces using Priority Pixel Queue Algorithm

Distance and Nearest Neighbor Transforms of Gray-Level Surfaces Using Priority Pixel Queue Algorithm

Leena Ikonen and Pekka Toivanen

Laboratory of Information Processing, Department of Information Technology,
Lappeenranta University of Technology, P.O.Box 20, 53851 Lappeenranta, Finland
{leena.ikonen, pekka.toivanen}@lut.fi

Abstract. This article presents a nearest neighbor transform for gray-level surfaces. It is based on the Distance Transform on Curved Space (DTOCS) calculated using an efficient priority pixel queue algorithm. A simple extension of the algorithm produces the nearest neighbor transform simultaneously with the distance map. The transformations can be applied for example to estimate surface roughness.

1 Introduction

Nearest neighbor, or nearest feature transforms, are closely related to distance transforms, and should preferably be achieved using the same algorithm. In the case of binary images, distance transforms can be derived from nearest neighbor transforms, but not vice versa [8]. Distance transformations were among the first image processing algorithms. Rosenfeld [10] presented a sequential local transformation algorithm for calculating distances in binary images in 1966, and similar chamfering techniques have been applied widely in the field, e.g., [1], [9], [13]. The transformations propagate local distance values across the image with a mask operation, which may have to be iterated several times to achieve globally optimal distances for gray-level images.

Alternatives to chamfering include ordered and recursive propagation [9], and pixel queue algorithms [11], [16]. The recursive propagation proceeds like a depth first search, while ordered propagation and pixel queue algorithms are applications of breadth first search. The efficiency of the depth first search is highly dependent on the propagation order, and breadth first approaches eliminate some of the repetition of distance calculations caused by finding shorter paths later on in the transformation. Gray-level distance transforms with varying local distances can be calculated correctly with ordered or recursive propagation, if neighbors of updated pixels are reprocessed. However, this is very inefficient for gray-level distance transforms of complex surfaces with highly curved paths. The ordered propagation seems to be more efficient in calculating the Distance Transform on Curved Space (DTOCS) [13] than the chamfering approach, but the priority pixel queue algorithm, which corresponds to a best first search, clearly outperforms both in large complex images [4]. A priority pixel queue idea by Verwer [15] is implemented with bucket sorting, which is applicable only with integer distances. Bucket sorting is also utilized in the transformation algorithm by Cuisenaire and Macq [3], where Euclidean distance values are obtained by gradually increasing the propagation neighborhood. The priority queue algorithm for calculating the geodesic time by Soille [12] also enumerates all possible distance values. The priority value is increased with one when no pixels with the current priority value are found in the queue. As the geodesic time sums gray-values along digital paths, the distance values can become very large, which also means a lot of priority values must be tested. Our minimum heap based transformation is applicable for any positive distances, including floating point distance values, and processes only distance values, which are needed. The priority queue approach enables easy implementation of the nearest neighbor transform, which can be calculated simultaneously with the distance transform. The unified distance transformation algorithm by Paglieroni [8] also calculates the nearest neighbor and distance transformation simultaneously using horizontal and vertical scans in a parallel architecture.

This article is organized as follows. The distance transforms are presented in Section 2 and the pixel queue transformation algorithm in Section 3. Section 4 presents the nearest neighbor transform, and Sections 5 and 6 some application ideas. Section 7 contains conclusions and discussion.

2 The Distance Transforms

The Distance Transform on Curved Space (DTOCS) calculates distances along a gray-level surface, when gray-levels are understood as height values. Local distances, which are summed along digital paths to calculate the distance transform values, are defined using gray-level differences:

$$d(p_i, p_{i-1}) = |\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})| + 1 \tag{1}$$

where $\mathcal{G}(p)$ denotes the gray-value of pixel p, and p_{i-1} and p_i are subsequent pixels on a path. The locally Euclidean Weighted DTOCS (WDTOCS) is calculated from the height difference and the horizontal distance using Pythagoras:

$$d(p_i, p_{i-1}) = \begin{cases} \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + 1}\ , & p_{i-1} \in N_4(p_i) \\ \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + 2}\ , & p_{i-1} \in N_8(p_i) \setminus N_4(p_i) \end{cases} \tag{2}$$

The diagonal neighbors of pixel p are denoted by $N_8(p) \setminus N_4(p)$, where $N_8(p)$ consists of all pixel neighbors in a square grid, and $N_4(p)$ of square neighbors. More accurate global distances can be achieved by introducing weights, which are proven to be optimal for binary distance transforms, to local distances in the horizontal plane. The Optimal DTOCS is defined in [5] as

$$d(p_i, p_{i-1}) = \begin{cases} \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + a_{opt}^2}\ , & p_{i-1} \in N_4(p_i) \\ \sqrt{|\mathcal{G}(p_i) - \mathcal{G}(p_{i-1})|^2 + b_{opt}^2}\ , & p_{i-1} \in N_8(p_i) \setminus N_4(p_i) \end{cases} \tag{3}$$

where $a_{opt} = (\sqrt{2\sqrt{2}-2}+1)/2 \approx 0.95509$ and $b_{opt} = \sqrt{2}+(\sqrt{2\sqrt{2}-2}-1)/2 \approx 1.36930$ as derived by Borgefors [1] by minimizing the maximum difference from the Euclidean distance that can occur between points on the binary image plane.

3 Pixel Queue Distance Transformation Algorithm

The DTOCS has previously been calculated with a mask operation, which has to be iterated several times before the distance map converges [13]. The larger and more complex the image surface is, the more iterations are needed, whereas the pixel queue approach slows down only slightly with increased surface complexity [4]. The efficient pixel queue algorithm eliminates repetition of local distance calculations by using a priority queue implemented as a minimum heap:

```
1. Define binary image $\mathcal{F}(x) = 0$ for each pixel $x$ in feature set, and $\mathcal{F}(x) = max$ for each non-feature $x$.
2. Put feature pixels to *priority queue* Q.
3. While Q not empty
       $p = dequeue(Q)$, $\mathcal{F}_q(p)$ was the smallest distance in Q.
       If $\mathcal{F}_q(p) > \mathcal{F}(p)$ (obsolete value), continue from step 3.
       $\mathcal{F}(p)$ becomes $\mathcal{F}^*(p)$ (value is final).
       For neighbors $x$ of $p$ with $\mathcal{F}(x) > \mathcal{F}^*(p)$
           Compute local distance $d(p,x)$ from original image $\mathcal{G}$.
           If $\mathcal{F}^*(p) + d(p,x) < \mathcal{F}(x)$
               Set $\mathcal{F}(x) = \mathcal{F}(p) + d(p,x)$
               $enqueue(x)$
           end if
       end for
   end while
```

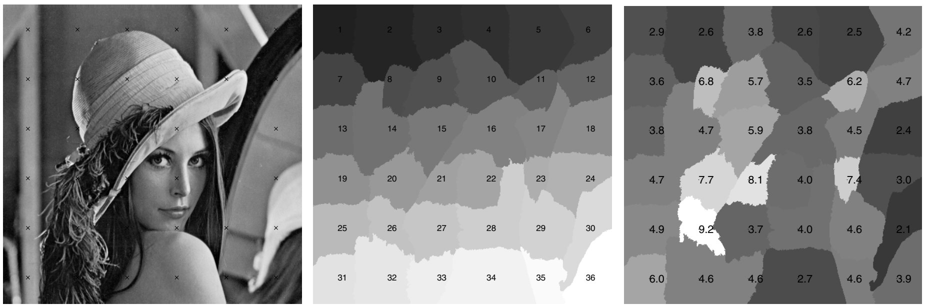
If the feature point sets are large and connected, it can be beneficial to enqueue only the feature boundary pixels in step 2. of the algorithm, but the same result is achieved as when enqueuing all feature pixels. The priority queue approach for calculating distances ensures that distance values are final when they are dequeued, and propagated further. Repeated enqueuings are possible if a new shorter path is found, but previous instances of the pixel in the queue can be eliminated based on obsolete distance values. The local distance calculation between two pixels is never repeated, as only pixels with final distance values can be source points, and pixel pairs, where the destination point already has a smaller distance value than the source point are also eliminated. The complexity of the pixel queue algorithm is in $\mathcal{O}(n \log n_q)$, where n_q is the length of the queue, which varies throughout the transformation. Typically, $n_q \ll n$, so the algorithm is in practise near-linear [4].

4 Nearest Neighbor Transform

The nearest neighbor transform can be viewed as a discretized version of the Voronoi diagram dividing the image to polygons around the feature or site points, so that each pixel belongs to the region of the closest site. In fact, Voronoi diagrams can be used to calculate Euclidean distance transforms for binary images [2], including voxel images in arbitrary dimensions [7]. The nearest neighbor transform assigns to each pixel the identity of its nearest feature pixel. The nearest site is here determined according to DTOCS distances, i.e. distances along the varying height surface, but the same algorithm works for any distance transforms with non-negative distance values. As local distances based on gray-values can vary a lot, the nearest neighbor transform can result in any shapes of regions around each site.

The nearest neighbor transformation produces a tesselation image, which is initialized to zero at non-feature pixels, and to a unique seed value $1..n_f$ at each of the n_f feature pixels. A simple extension of the priority pixel queue algorithm calculates the nearest neighbor transform simultaneously with the distance transform. When a pixel with a new distance value is enqueued, the corresponding pixel in the tesselation image gets the seed value of the pixel from which the distance value propagated. If the same pixel is enqueued repeatedly, the seed value is replaced with the new one. The final seed value identifies the feature pixel from which the propagation path of the final distance value originated. Points equally distant from two or more seed points will end up in the region from which the distance propagated first. A similar region growing algorithm for Voronoi tesselation of 3D volumes resolves collisions of neighboring regions using Euclidean distances [6], but in the DTOCS with curved paths, the Euclidean distance between the pixels does not correspond to the real distance the transformation approximates.

An example of a nearest neighbor transform can be seen in Fig. 1. The familiar 'Lena' image represents a varying height surface, and a nearest neighbor transform using an evenly spaced grid of seed points is calculated. The original image is shown with the seed points in Fig. 1 a), and the resulting nearest neighbor transform is shown in Fig. 1 b) with seed values marked on each region. As distances are calculated along the surface with the DTOCS, seed points in areas with more variation are surrounded by small regions (e.g. seed value 23).



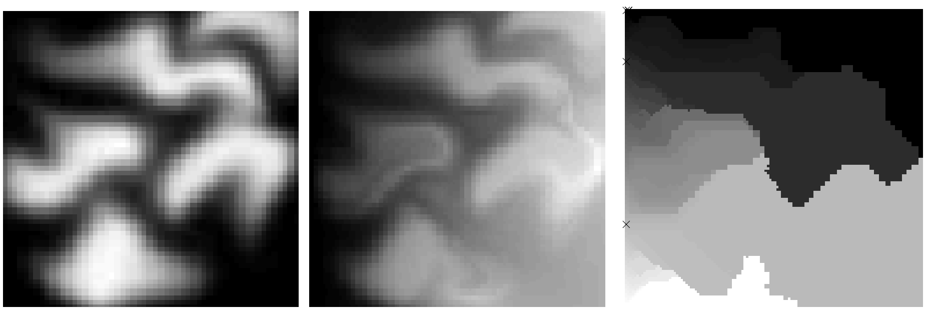
a) Orig. with grid points b) Nearest Neighbor Transf. c) DTOCS roughness map

Fig. 1. Nearest neighbor transform and roughness map from an even grid of points

In smooth areas distances can propagate further, covering more pixels. Region borders are more likely to appear near locations, where there are abrupt changes in gray-values, causing large local distances. This can be seen for example in regions 21, 16 and particularly 11, where the brim of Lena's hat is clearly visible. This suggests that the nearest neighbor transform could be applied to segmenting highly varying textures from smoother ones. The roughness map in Fig. 1 c) will be explained in Section 6.

5 Propagation Visualization

The nearest neighbor transform can be used to visualize propagation of distance values. The points in the feature set can be numbered as seed points for the nearest neighbor transform, and when the distance values propagate, the seed values propagate as well. When the distance map is final, the tesselation map shows from which feature point each distance value has propagated. On a varying image surface with several feature points, some feature seed values propagate only in a small area, or not at all, if distance values spread fast from points in the vicinity of the point. The order in which feature pixels are enqueued, and in which neighbors of the dequeued pixel are processed, affect the propagation order. Equal distances could be achieved along several different paths, but the seed values indicate via which points the values have in practise propagated.



a) Original image b) DTOCS image c) Nearest neighbor transf.

Fig. 2. Example surface with its distance transform and propagation tesselation

Figure 2 shows a height map, its distance image and the propagation tesselation map, when the feature point set consists of all points in the leftmost column. The color of an area in Fig. 2 c) identifies the feature point from which the distance value has propagated. It can be seen that the number of different seed values propagating decreases towards the end of the distance transformation, i.e. when the highest distance values towards the right edge of the image are reached, only two different seed values are left of the original 128 feature point values used in the $128 * 128$ surface image. The feature points with the three furthest spread seed values are marked with 'x' on the tesselation image.

6 Roughness Measurement

The distance and nearest neighbor transforms can be combined into a method estimating the roughness of a gray-level surface. Figure 1 c) shows an example of a roughness map, where the values marked on each region of the nearest neighbor transform indicate the average roughness of that region. The values are calculated as the average of normalized distance values within each region. The normalized values are obtained by dividing the curved DTOCS distances calculated from a grid of feature points with the corresponding straight distances. The straight distance, or chessboard distance, is simply the larger of the coordinate differences between the point in question and its nearest neighbor grid point. The more variation there is around the grid point, the larger are the normalized distances, and subsequently the roughness value of the region. An estimate of the global roughness of the image surface can be calculated as the average of all normalized distances. In future works, the method will be applied to measuring roughness of paper from microscopic gray-level images.

7 Discussion

The main contribution of this paper is the new nearest neighbor transform algorithm for gray-level surfaces based on the priority pixel queue distance transformation. The algorithm is very simple, and fast, as its complexity is near-linear. The nearest neighbor transform is calculated simultaneously with the distance transformation, and the value of a pixel is known to be final once it is dequeued. This means that intermediate results can be used in time critical applications, or if a complete distance transformation is not needed. For instance, if the distance transforms are used to find a route along a surface, as presented in [5], the transformation starts from the source point, and can be interrupted once the destination point is reached, that is, when the destination point is dequeued. Obviously, the path of the shortest distance could be recorderd during the distance transformation by storing the direction from which the distance propagates to each pixel, but only a single path would be found, whereas the Route DTOCS algorithm [5] finds points on any path. The nearest neighbor transform could also be utilized in some shortest path problems. The actual path is not found, but the nearest of several destinations can be selected by calculating the nearest neighbor transform with the alternative destinations as features, and then selecting the destination with the seed value, which the source point obtained in the transformation.

An application idea of using the distance and nearest neighbor transformations for surface roughness evaluation was also presented. Generally, the DTOCS is well suited for measuring the amount of variation in a gray-level image. The first application of the DTOCS involved selecting control points for image compression [14]. To store information from locations, where gray-levels change, the control points were selected from boundaries, at which the curved DTOCS distances normalized with the corresponding chessboard distances exceed a given threshold. The priority queue approach could produce the boundary in a straightforward manner by not enqueuing pixels after reaching the threshold. The implementation utilizing sequential DTOCS has to search for the boundary in the transformed image. In general, equal distance curves can be found easily with the priority queue approach, and also limiting the transformation to some maximum distance value is trivial, unlike in mask operations, where the whole image must be processed to be sure distance values are globally optimal.

The curved DTOCS paths are similar to paths formed in constrained distance transforms, see for instance [9], and in fact, the DTOCS can be used as a constrained distance transform. Constraint pixels are marked with values differing so much from other image areas, that paths to other pixels will generally not cross them. The same idea can also be implemented by multiplying the gray-level difference used in the local distance definition by a large factor. A maximum distance value can be set, so that the transformation finishes without calculating the distances to the constraint pixels, which otherwise would get huge values. The DTOCS can be applied in obstacle avoidance problems with several levels of obstacles, for example areas that can be crossed with a higher cost in addition to completely constrained areas. All accessible areas in an obstacle avoidance setting could be found by using the DTOCS or the nearest neighbor transform with a maximum allowed distance, and at the same time, the shortest path to the destination could be found in a navigation application.

References

1. Gunilla Borgefors. Distance Transformations in Digital Images. *Computer Vision, Graphics, and Image Processing*, 34:344–371, 1986.
2. Heinz Breu, Joseph Gil, David Kirkpatrick, and Michael Werman. Linear time Euclidean distance transform algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5):529–533, May 1995.
3. O. Cuisenaire and B. Macq. Fast Euclidean distance transformation by propagation using multiple neighborhoods. *Computer Vision and Image Understanding*, 76(2):163–172, November 1999.
4. Leena Ikonen. Pixel queue algorithm for geodesic distance transforms. In *Discrete Geometry for Computer Imagery (DGCI)*, pages 228–239, Poitiers, France, April 2005.
5. Leena Ikonen and Pekka Toivanen. Shortest routes on varying height surfaces using gray-level distance transforms. *Image and Vision Computing*, 23(2):133–141, February 2005.
6. C. A. Kapoutsis, C. P. Vavoulidis, and I. Pitas. Morphological iterative closest point algorithm. *IEEE Transactions on Image Processing*, 8(11):1644–1646, November 1999.
7. Calvin R. Maurer, Rensheng Qi, and Vijay Raghavan. A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):265–270, February 2003.
8. David W. Paglieroni. Distance Transforms: Properties and Machine Vision Applications. *CVGIP: Graphical Models and Image Processing*, 54(1):56–74, January 1992.

9. Jim Piper and Erik Granum. Computing Distance Transformations in Convex and Non-convex Domains. *Pattern Recognition*, 20(6):599–615, 1987.
10. Azriel Rosenfeld and John L. Pfaltz. Sequential Operations in Digital Picture Processing. *Journal of the Association for Computing Machinery*, 13(4):471–494, October 1966.
11. Jaime Silvela and Javier Portillo. Breadth-first search and its application to image processing problems. *IEEE Transactions on Image Processing*, 10(8):1194–1199, 2001.
12. Pierre Soille. *Morphological Image Processing: Principles and Applications.* Springer-Verlag, 2 edition, 2003.
13. Pekka J. Toivanen. New geodesic distance transforms for gray-scale images. *Pattern Recognition Letters*, 17:437–450, 1996.
14. Pekka J. Toivanen, Ari M. Vepsäläinen, and Jussi P. S. Parkkinen. Image compression using distance transform on curved space (DTOCS) and Delaunay triangulation. *Pattern Recognition Letters*, 20:1015–1026, 1999.
15. Ben J. H. Verwer, Piet W. Verbeek, and Simon T. Dekker. An efficient uniform cost algorithm applied to distance transforms. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11(4):425–429, April 1989.
16. Luc Vincent. New trends in morphological algorithms. In *Proc. SPIE/SPSE*, volume 1451, pages 158–170, 1991.