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].

E. Andres et al. (Eds.): DGCI 2005, LNCS 3429, pp. 228–239, 2005.

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

(a) Mercury (b) Lena (c) Ball

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

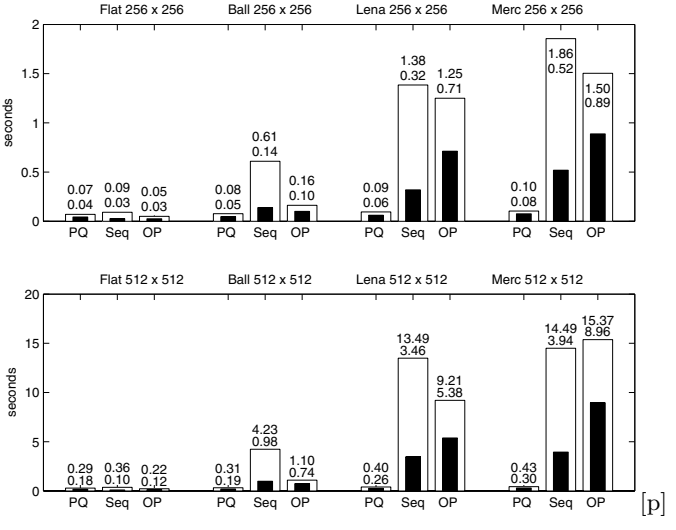

Fig. 2. Average run times of DTOCS (black bar) and WDTOCS (white bar) using Priority Queue, Sequential and Ordered Propagation algorithms

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

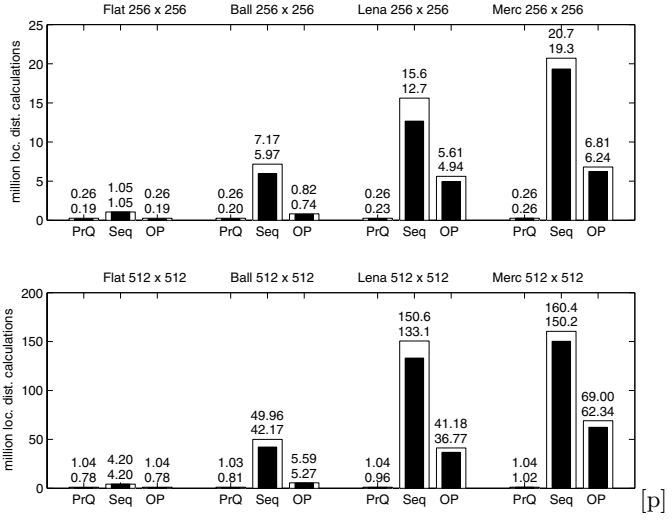

Fig. 3. Average number of local distance calculations needed in DTOCS (black bar) and WDTOCS (white bar) using Priority Queue, Sequential and Ordered Propagation algorithms

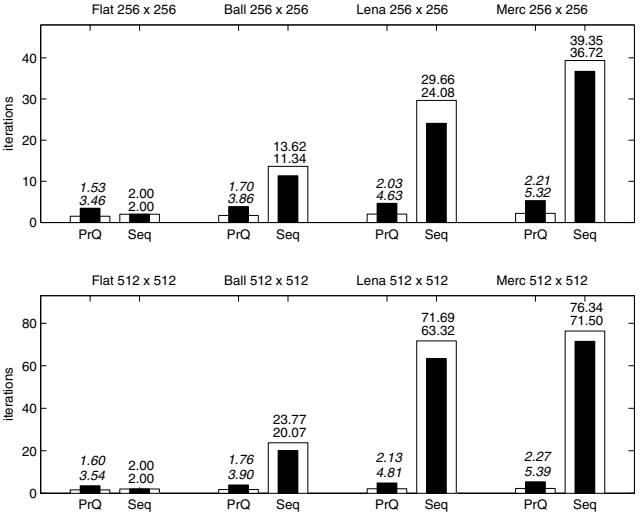

Fig. 4. Average number of iterations needed in sequential DTOCS (black bar) and WDTOCS (white bar). The number of iterations indicated for the pixel queue algorithm is a comparison number calculated from the run times

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 superiorior. 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.

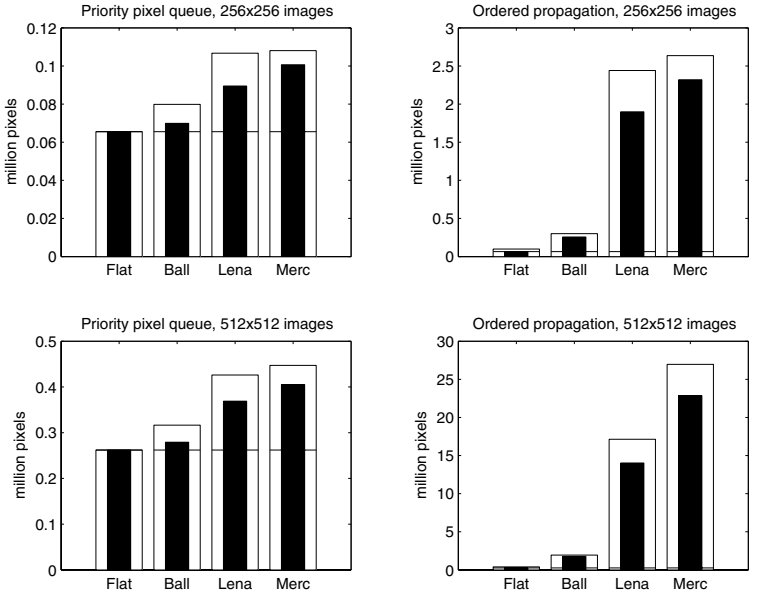

Fig. 5. Number of pixel enqueuings in DTOCS (black bar) and WDTOCS (white bar) for the priority queue (left) and the ordered propagation (right). The horizontal line on each bar indicates the number of pixels in the image, so the section of the bar above the line shows how many pixels get enqueued repeatedly.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

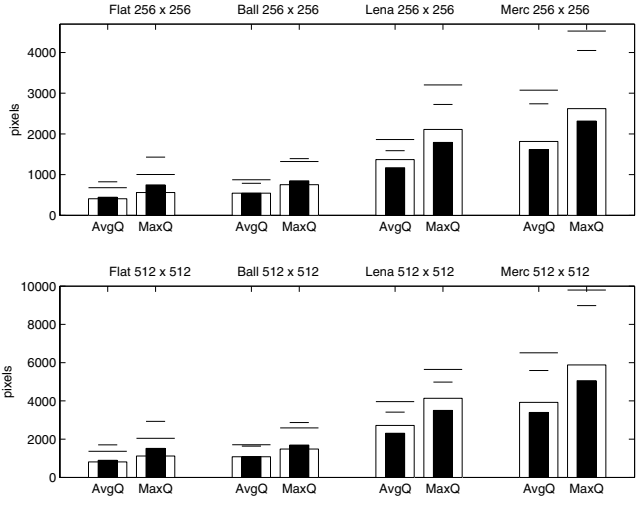

Fig. 6. Average and maximum queue lengths in DTOCS (black bar) and WDTOCS (white bar). The horizontal lines above the bars indicate the maximum values, i.e. the largest average queue length and largest maximum queue length

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, particularly 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.