Morphological Operations in Recursive Neighbourhoods

Pieter P. Jonker

Pattern Recognition Group, Faculty of Applied Sciences,
Delft University of Technology, Lorentzweg 1, 2628 CJ Delft,
The Netherlands
pieter@ph.tn.tudelft.nl

Abstract. This paper discusses the use of Recursive Neighbourhoods in Mathematical Morphology. Its two notable applications are the recursive erosion / dilation, as well as the detection of foreground-background changes to be used in skeletonization. The benefit of the latter over an extension of the neighbourhood or the use of sub-cycles is emphasized. Two applications are presented that use the recursive neighbourhood in a 3D surface and a 3D curve anchor-skeleton variant.

1 Introduction

In previous papers [3], [4], [5], we have described a general principle for morphological operations on cubic tessellated binary images X_N. For the sake of clarity we will briefly repeat this in sections 1 and 2, and in examples use images of dimensions 2 and 3. In section 3 we elaborate on the principle of recursive neighbourhoods. The use of recursive neighbourhoods is not new, however, its principle of work and its possibilities are often not well understood. In section 3 we show that from the four alternatives to circumvent the problem of detecting the topology change of a two pixel/voxel thick structure in a 3^N neighbourhood, as is necessary in skeletonization, the recursive neighbourhood approach is generally the fastest. The recursive neighbourhood detects change, either foreground changed to background or background changed to foreground. This change detection can be fruitfully used in a number of applications, but also give rise to problems, e.g., excessive protrusions in skeletons, when the recursive neighbourhood is not correctly used. In chapter 4 we present two examples of its use. The essence of both examples is the use of a surface, and a curve skeleton, forced through anchor points, whereas its surface and curve protrusions are recursively eroded due to the use of the recursive neighbourhood.

From [3], [4], [5], we derived that using a recursive neighbourhood is based on scanning an image with a set of masks that comprises a structuring element S. The masks contain foreground, background and don't-cares. On each element of the image X an inexact match ($\cong$) between all masks of the set and the neighbourhood extracted

A. Braquelaire, J.-O. Lachaud, and A. Vialard (Eds.): DGCI 2002, LNCS 2301, pp. 187–196, 2002.

from the image can be performed, whereas the result written to the output image Y is the union of all matches. A second input image Z can be used in the same match to perform dyadic operations. Dyadic operations can be used to locally enable or disable the morphological operation, using the points from a second input image.

To elaborate on this: Let x, y, and s be the elements of images X, Y, and S. Let S, e.g., be a 3 x 3 structuring element and M_k an equivalent 3 x 3 neighbourhood around pixel x_k. For binary morphology, the transformation $Y \leftarrow X \cong S$, is informally:

If for any pixel x_k *in an input image* X*, its neighbourhood* M_k *matches inexactly with a structuring element* S*, the pixel* y_k *of output image Y is set to one. If* M_k *doesn't match* S*,* y_k *is set to zero.*

In the inexact neighbourhood match the foreground pixels in S should match with foreground pixels in M_k at the same positions AND the background pixels in S should match with background pixels in M_k at the same positions, whereas in the don't care positions of S a match is not required. This inexact match can be extended to:

If for any pixel x_k *in an input image* X*, its neighbourhood* M_k *matches one mask* S_i^S *from a set of masks* S^S *the pixel* y_k *of output image Y is set to one, else to zero.*

Meaning that the union of all mask matches is taken. A further extension is:

If for any pixel x_k *in an input image* X*, its neighbourhood* M_k *matches any mask* S_i^S *from a given set of masks* S^S *the pixel* y_k *of output image* Y *is set to one, else set to zero, OR if its neighbourhood matches any mask* S_i^R *from a given set of masks* S^R*, the pixel* y_k *is set to zero, else to one.*

The image transformation $Y \leftarrow X \cong S$ is now implemented with a set of masks S consisting of a subset S^S (the SET-masks) and a subset S^R (the RESET-masks), one of which may be empty. This means that either a pixel y_k is set to zero, if one of the RESET masks fits, or the pixel is set to one, if one of the SET mask fits, where the SET masks are chosen to dominate over the RESET masks. A second input image Z can be used to locally enable/disable the transformation, yielding a dyadic operation. If a "mask-bit" z of a mask of the set S is set to don't care, the transformation is enabled. If z is do-care, then if the pixel z_k of Z matches with z, the operation is disabled, else enabled. Operations that use Z to locally mask-off, or insert seeds, are the propagation operation and the anchor-skeleton. Finally, an operation on an image can be done by performing a transformation with a mask-set once, twice or more, or until the image is idempotent (does not change) under the operation.

In many cases practical use can be made of the intermediate results in the output image. This is called **(spatial) recursion**. For instance, if the transformation is performed by a raster scan over the image, i.e., from top-left to bottom-right, the fact that some neighbours of pixel y_k have already obtained a new value can be utilized. For this purpose, the neighbourhood M_k of pixel x_k is as well as the structuring element S is extended. Both the Local Neighbourhood (LN) and the Recursive Neighbourhood

(RN) can be used in a neighbourhood matching procedure, referred to as *local neighbourhood* operations (LNO) and *recursive neighbourhood operations* (RNO).

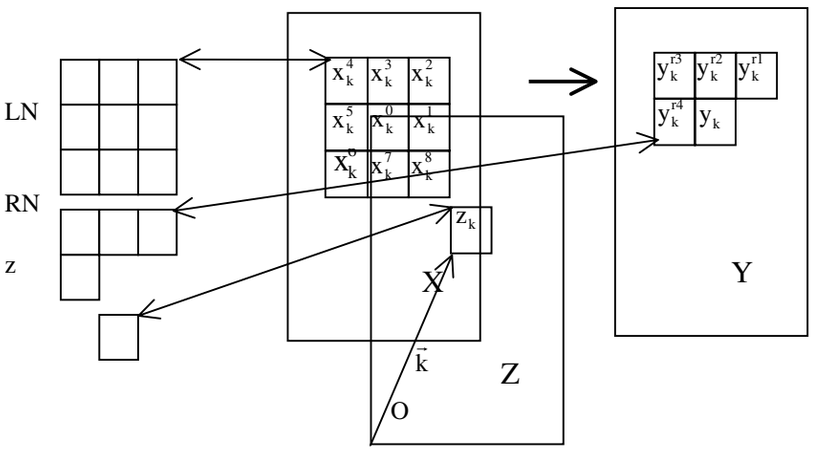

Fig. 1. Drawing conventions for a neighbourhood match. Here shown for 2D images and 3 x 3 neighbourhoods. The LN is taken from input image X, the RN from output image Y. For dyadic operations a second input image Z is used.

Figure 1 shows the matching process with a single mask from a set S, whereas the neighbourhood M_k is extracted from the three different images. Note that when using a software raster scan over the image, it is beneficial for RNOs to scan in all odd scans from top-left to bottom-right and in all even scans from bottom-right to top-left. In this case, the RN should be transposed to match with the scan direction.

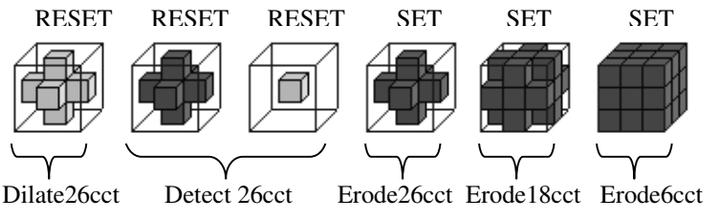

Fig. 2. Structuring elements for simple operations in X_3. Opaque elements are don't cares, light grey are background elements, and dark grey are foreground elements.

Figure 2 shows some structuring elements or mask-sets for simple Local Neighbourhood Operations in X_3, while Figure 3 shows an example of a dyadic Recursive Neighbourhood Operation; the propagation operation, a recursive conditional dilation: Objects in an image are recursively dilated (the first mask), wherever foreground in image Z and background in X was found (the second mask).

2 Skeletonization, Shape Primitives

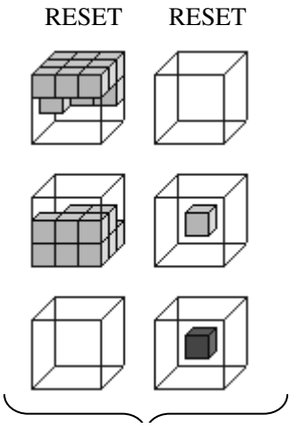

Fig. 3. A dyadic RNO in X_3; the propagation operation

Erosions can be described as a match on foreground area in X_2 and on foreground volume in X_3. Skeletonization can be seen as conditional erosion [3]. Figure 4 shows that in X_3 a volume is eroded to a curved surface -the surface skeleton-, where after the surface is eroded to a space curve -the curve skeleton-. The condition for the erosion is that surfaces, or curves, should not be eroded. Those conditions are also known as shape primitives in X_N. The sets of shape primitives and how they can be found are described in [5]. In 3D one set is called *Surf26*, which represents the surface primitives: On all possibilities of a curved surface to interesect a 3^3 neighbourhood, one of these masks (or a rotated and/or mirrored version) will hit. The set *Curv26* represents likewise the space curve primitives: On all possibilities of a space curve to interesect a 3^3 neighbourhood one of these masks (or a rotated and/or mirrored version) will hit. Iterating over an image X_3 with the erosion mask *Erode26cct* of Figure 2 and a set of surface primitives (*Surf26*) and the set of curve primitives (*Curv26*) yields a skeleton. *Erode26cct* erodes surfaces from volumes (it hits only on the core of the volumes, not on their boundaries). The set *Surf26* detects surfaces and thus may be used to prevent the erosion of surfaces (the masks only hit on the core of the surface, not on the surface boundaries). The set *Curv26* detects curves and can be used to prevent the erosion of curves (the masks only hit on the kernel of the curves). Consequently, as volumes, surfaces and curves cannot be eroded from their kernels, they are eroded from their boundaries. So only closed surfaces and curves will remain.

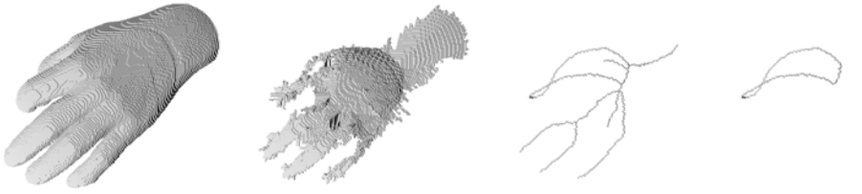

Fig. 4. Original[1], surface skeleton, curve skeleton, and the topological kernel.

To prevent the erosion of boundaries, from the set *Curv26* a set *Curv26e* can be derived that contains all surface edge situations. Similarly, a set *Curve26e* can be made from the set *Curv26* containing all curve end situations [4],[5].

[1] Courtesy Dr. K. Katada, Fujita Health University Japan

3 On the Necessity and Use of Recursive Neighbourhoods

A recursive neighbourhood can be used for a number of reasons.

First it can be used to erode or dilate objects or background in one or two passes over the image. For instance the masks of Figure 3 can be used to quickly select objects. With objects stored in image X and seed voxels stored in image Z, starting from the seed voxel, a wave front dilation starts over objects that are connected to the seed. Figure 5 shows how such dilation evolves in a 2D situation. The dilation mask (5a) is assumed to make a raster scan over the image from top-left to bottom down. A part of an image X is shown in 5b…5e. In 5b and 5c the mask does not fit and hence the pixels are reset to foreground in the output image Y. When the scan runs over the image at one row lower, shown in Figure 5d and 5e, in situation 5d the pixel swaps value from background to foreground but the swap in 5e depends on which neighbourhood is used. Using the recursive neighbourhood (partially using output image Y), the pixels swaps value, as the mask does not match, but when using the Local Neighbourhood (only using input image X) the mask fits and the pixel remains background. Of course, an unconditioned recursive dilation in a downward scan followed one in an upward scan over the image would fill the whole image. Likewise, a recursive erosion would erode a convex object in two of such scans over the image. Hence its application is found in conditional erosions and dilations.

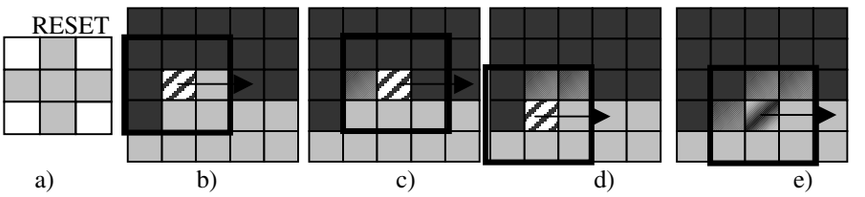

Fig. 5. Recursive dilation

Secondly, the Recursive Neighbourhood can be used to detect change within a skeletonization procedure[2].

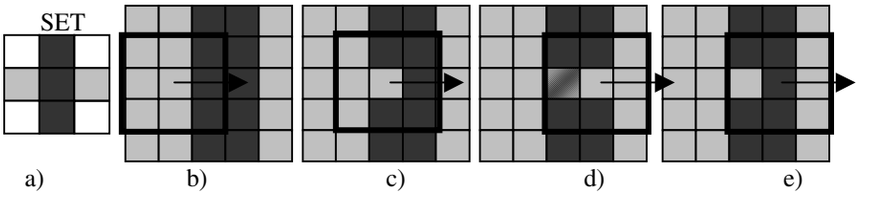

Fig. 6. Topology check using the local (b, c, d) or recursive (b,c,e) neighbourhood

The problem with a topology check (e.g., a mask) that works on a 3^N neighbourhood is that it cannot detect two element (pixel, voxel, …) thick structures. Figure 6 shows a situation in which the mask of Figure 6a is used to detect a straight single pixel thick

[2] The first one known to me that used it for this purpose was Hilditch (1969) [2]

vertical line. This curve primitive (6a) is one of the masks that are used as topology check for a 2D skeleton. A raster scan is assumed. The sequence 6b, c, d shows what happens if the local neighbourhood is used for a topology check. 6b, c, e shows what happens if the recursive neighbourhood is used. In situation 6d the topology is broken, in 6e it is preserved as now the output image is partly used in the mask match.

An alternative to the use of the recursive neighbourhood is to use only the Local Neighbourhood, but to erode from one direction at a time -explicitely or implicitely- [1]. This technique is called sub-cycling. In the example of Figure 6, the tests of 6c and 6d would then have been done in subsequent cycles through the image, first an erosion from the left, and secondly an erosion from the right. The same effect can be obtained by first processing all odd lines of the image followed by the processing of all even lines in the image. This is often done on parallel hardware such as massively parallel processor arrays [7]. In literature [10], often the terminology sequential algorithm versus parallel algorithm is used for the use of the recursive neighbourhood versus the local neighbourhood with sub-cycles. However, both approaches can be implemented on sequential as well as parallel machines. A third method is the extension of the neighbourhood -for instance in 2D to 3 x 4- to detect the two element thick structures [8]. This lets the number of masks (or topology tests), as well as the number of image elements to be tested grow considerably. When a recursive neighbourhood can be addressed, this is the fastest method, as it diminishes the number of tests within the neighbourhood, the number of topology tests as well as the total number of cycles through the image.

When the skeleton is made, the topology checks can be split into checks on the interior of the object and checks on the boundary. In 3D playing with the subsets: erosion mask (*Erode26cct*), surface detection mask-set (*Surf 26*), curve detection mask-set (*Curv26*), the mask-sets for surface ends (*Surf26e*) and curve ends (*Curv26e*) skeleton variants can be made. The set {*Erode26cct*, *Surf26*, *Surf26e*, *Curv26*} is used for the surface skeleton. The set {*Erode26cct*, *Surf26*, *Curv26*, *Curv26e*} was used to obtain the curve skeleton. The set {*Erode26cct*, *Surf26*, *Curv26*} was used to obtain the last skeleton of Figure 4. The recursive neighbourhood should be used, only for the *Surf 26* and *Curv26* sets and not for *Erode26cc* and the *Surf26e* and *Curv26e* sets.

If the recursive neighbourhood is used for the erosion mask, the erosion will not be performed boundary by boundary like peeling an onion skirt, but the erosion will immediately propagate over the object. The final skeleton will not be on the medial axis, but will lay on the bottoms of the objects, when raster scanning. If the recursive neighbourhood and/or sub-cycling method is also used for the object boundary conditions, e.g., *Surf26e* and *Curv26e,* this will lead to the sprouting of spurious protrusions. This can be best explained when using the sub-cycle technique. Due to the first sub-cycle, locally a noisy boundary may be formed, which may be cancelled out in the subsequent sub-cycle by an erosion from another direction. However, a boundary detection condition mask, may decide that this is the beginning of a protrusion and decides to keep it. So object core conditions should be treated differently from object boundary conditions, to avoid excess sprouting of protrusions. But even then, in the object core conditions, e.g., (*Surf26*, *Curv26*), the recursive neighbourhood should

only be checked to verify if a foreground is changed into background, where a background is expected in the mask. This is shown in Figure 7.

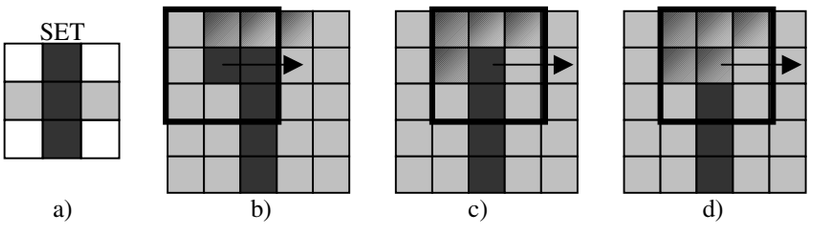

Fig. 7. Using the recursive neighbourhood for foreground

Suppose the mask of Figure 7a is used to detect a skeleton curve in 2D. The pixel in 7b is reset to background as the mask does not fit. The pixel in 7c is correctly set to foreground; the mask fits when foreground in the mask (the North pixel) is checked in the input image X and the background in the mask (the West pixel) is verified using the output image Y. 7d Shows what happens if both are verified using the output image Y. The topology is broken. The use of the full recursive neighbourhood in the sense of Figure 7b, can be profitably when a skeleton without end conditions is made. For instance in a skeleton without end-voxel conditions, yielding the topological kernel, as shown in Figure 4d, the space curve conditions are verified in the full recursive neighbourhood. The benefit of this is that the non-closed curves are eroded recursively, i.e., in a few passes through the image as the erosion propagates over the skeleton branches within a single cycle through the image. This makes this skeleton fast.

4 Advanced Examples

In this section two examples are presented of applications that make use of 3D morphological operations, most notable of variants of the 3D skeleton operator and the propagation operation. Both use the recursive neighbourhood in various ways.

Consider Figure 8. The original image[3] is taken by an acquisition system -based on multiple cameras- that produces a stream of 3D object data [11]. As the system produces imperfect data, each image of the stream has to be processed before rendering texture and colour onto it. The lower images in Figure 8 are cross sections of the legs.

The procedure used to clean-up the original is the following sequence:

1) dilate the object with a 26 connected contour
2) dilate the object with a 6 connected contour, twice
3) erode a 26 connected contour from the object
4) erode a 6 connected contour from the object

[3] Courtesy Prof. Matsuyama, Kyoto University Japan
This is an old stream; currently his 3D streams have a far better quality [11]

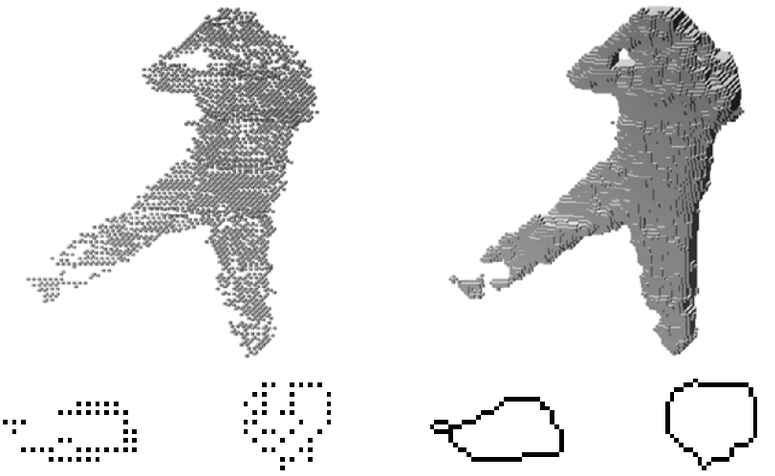

Fig. 8. Aerobic girl original Aerobic girl processed

5) iteratively erode a 6 connected contour from the object, (*Erode6cct*) but use also the mask-sets *Surf26* and *Surf26e*, thus performing a few cycles of the surface skeleton. Anchor the original object into the skeleton; with a logic OR operation, after each skeleton cycle the original data points are re-inserted, thus forcing the skeleton through the original data points. Step 1 through 4 performs a 3D closing operation. Due to step 5 the result is a closed single voxel thick surface that goes through the original data points. Like alternating 4 and 8 connected in 2D, the alternation of the 26 and 6 connected erosions/dilations in 3D leads to a better erosion/dilation metric [3].
6) propagate the edge of the image into the image (*Propagation26cct)*, stop at the object boundary and invert the result. This makes the object solid.
7) extract the contour of the solid object. Steps 6 and 7 remove all inner data points of the object.
8) take a surface skeleton without boundary conditions using {*Erode26cct, Surf26*}. This finds the closed surface contour only, i.e., it removes sprouting surfaces. The full recursive neighbourhood is used in *Surf26* and consequently the boundary surfaces are eroded recursively and two skeleton cycles are sufficient.

In Robot Soccer [6],[9] two teams of each 4 autonomous robots play soccer in a field of 5 x 10 meter. One of the issues is to quickly plan collision free paths for the robot. Moreover, if a robot should play together with a teammate, his presence is required on a certain point in space AND time, e.g., to intercept a passed ball. A 3D image X_3 represents the universe of the robot with two space dimensions (x, y) and one time dimension (t). Note that if the orientation of the robot should also be taken into account, the problem should be solved in X_4 (x, y, θ, t). Figure 9 shows 12 pictures of a soccer field. The soccer field is in this example divided into cubic voxels of 0.3 x 0.3 x 0.6 [m·m·s]. The robot speed is {0, 1, 2} · [0.5 m/s]. The image size [x, y, t] is

17 x 35 x 35 voxels. The origin of the field is in the centre of the image at t = 0. Assume an attacker robot with ball wants to score, while the goal is defended by two defender robots. The attacker is on point (-6,-6,0) and tries to get to point (6,-2,34) to score, and the defenders are at points (15,7,0) and (10,-6,0) and are perceived by the attacker robot to head for the points (-10,6,34) and (0,0,34). The path planning procedure of the attacker robot consists of the following steps.

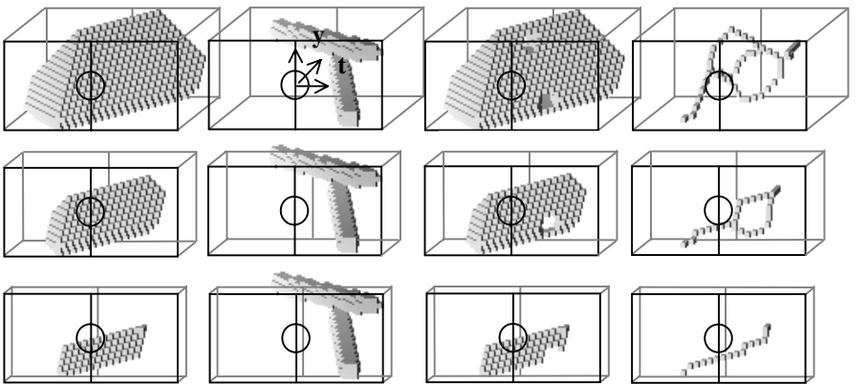

Fig. 9. Search space moving objects reduced search space collision free path

1) Recursively dilate the start point towards the positive time direction. The resulting cone indicates the points in the field that can be reached by the robot in positive time direction, when using speed 1. See Figure 10a.

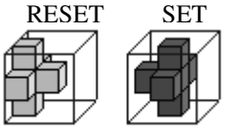

Fig. 10. a) Fully recursive neighbourhood (reset) mask to propagate a cone in space-time and b) a fully recursive neighbourhood (set) mask to erode objects in space-time

2) Recursively dilate the goal point towards the negative time direction
3) AND the results of 1) and 2). The top row, left column of Figure 9 shows the result, the initial search space.
4) From the two defender robots a speed and heading is perceived by the vision system of the attacker robot. It assumes that for the duration of the path those robots will linearly continue their path. The top row, second column in Figure 9 shows two objects, the linear paths of the defender robots in space-time, dilated one step, to introduce a safety margin.
5) The EXCLUSIVE OR of search space and the moving object images yields the reduced search space of the top row, third column of Figure 9.
6) The curve skeleton without end-curve conditions {*Surf26*, *Curv26*}, with start and endpoint of the attacker robot anchored into the skeleton, yield the final collision free path. Note that both *Surf26* and *Curv26* are used in the full recursive neighbourhood. This enables the fast erosion of sprouting surfaces and space curves. A problem in the planning is that the path to be found may not

run backwards in time. To prevent this, first all skeleton scans over the image are only applied in the forward time direction. Secondly, the erosion mask is adapted, so that it does not erode backward in time (see Figure 10b). Thirdly, masks in *Curv26* that have configurations that allow connectivity's backward in time are omitted. Thus paths that run backward in time are never found, as their topology is not preserved.

7) As the curve skeleton finds all possibilities to go from start to goal, a simple algorithm is: go left at a branch point if $y > 0$, right if $y \leq 0$, leading the attack over the wings. Alternatively, the distance can be propagated over the branches, to find the shortest of the branches.
8) If no fixed end-time t_{end} is assumed for the attacker robot, t_{end} can be iteratively reduced, until no path is found anymore. The three rows of Figure 9 show the paths at end-times $t_{end} = 34$, $t_{end} = 22$ and finally $t_{end} = 16$, the minimum time for a collision free path. A start value for the end-time is $t_{end}=x_{end}-x_{start}+y_{end}-y_{start}$.
9) For $t_{end} = 34$, a path is found in 84 msec on a PIII, 600 Mhz .

References

1. Arcelli C., Cordella L., Levialdi S. (1975) Parallel thinning of binary pictures. Electronic letters 11: pp 140-149
2. Hilditch C.J. (1969) Linear Skeletons from Square Cupboards. In: B. Meltzer and D. Mitchie (eds.), Machine Intelligence, Vol. 4, Edinburgh University Press, 404-420
3. Jonker P.P. (1992) Morphological Image Processing: Architecture and VLSI design. Kluwer Dordrecht, ISBN 90-2012766-7
4. Jonker P.P. (2000), Morphological Operations on 3D and 4D Images: From Shape Primitive Detection to Skeletonization. Proc. 9th Int. Conf., DGCI 2000 (Uppsala, Dec.13-15), Lecture Notes in Computer Science, vol. 1953, Springer, Berlin, 2000, 1371-391.
5. Jonker P.P. (2002), Skeletons in N dimensions using Shape Primitives, Pattern Recognition Letters, April 2002.
6. P.P. Jonker, J. Caarls, and W. Bokhove (2001), Fast and Accurate Robot Vision for Vision based Motion Proc. 4th Int. Workshop on Robocup (Melbourne, Australia, Aug.31-Sep.1, Springer Verlag, Berlin, 149-158.
7. Kyo, S., Okazaki, S., Fujita, Y., Yamashita, N (1997), A Parallelizing Method for Implementing Image Processing Tasks on SIMD Linear Processor Arrays, Proceedings of the IEEE workshop on Computer Architecture for Machine Perception (CAMP 97): 180-184
8. Ma C.M. (1994) On Topology Preservation in 3D Thinning. CVGIP-IU, 59(3), 328-339
9. www. robocup.org , www.robocup.nl
10. Rosenfeld A, Pfalz J.L. (1966) Sequential operations in digital picture processing. Journal of the ACM, 13(4): 471-494
11. Wu X, Wada T, Matsuyama T (2001) Real-time Active 3D Object Shape Reconstruction for 3D Video. Proc. of the 4th Int. Workshop on Cooperative Distributed Vision, March 22-24, Kyoto, Japan: 455-474. cdvws@vision.kuee.kyoto-u.ac.jp