Updating Distance Maps when Objects Move

Terrance E. Boult
Columbia University Department of Computer Science
New York City, New York, 10027. tboult@cs.columbia.edu

ABSTRACT

Using a discrete distance transform one can quickly build a map of the distance from a goal to every point in a digital map. Using this map, one can easily solve the shortest path problem from any point by simply following the gradient of the distance map. This technique can be used in any number of dimensions and can incorporate obstacles of arbitrary shape (represented in the digital map) including pseudo-obstacles caused by unattainable configurations of a robotic system.

This paper further extends the usefulness of the digital distance transform technique by providing an efficient means for dealing with objects which undergo motion. In particular, an algorithm is presented that allows one to update only those portions of the distance map that will potentially change as an object moves. The technique is based on an analysis of the distance transform as a problem in wave propagation. The regions that must be checked for possible update when an object moves are those that are in its "shadow", or in the shadow of objects that are partially in the shadow of the moving object. The technique can handle multiple goals, and multiple objects moving and interacting in an arbitrary fashion.

The algorithm is demonstrated on a number of synthetic two dimensional examples.

1 INTRODUCTION

Given a digital map of distances to a goal (or goals), one can easily solve the shortest path problem to the goal(s) from any other point by simply following the gradient of the distance map [Montanari 68]. This technique can be used in any number of dimensions and can incorporate arbitrary obstacles (represented in the digital map) as well as pseudo-obstacles caused by unattainable configurations of a robotic system.

Through the years, researchers have proposed algorithms which compute digital distances assuming various neighborhood and approximations, e.g. see [Lee 61], [Rosenfeld and Pfaltz 66], [Rosenfeld and Pfaltz 68], [Barrow et al. 77], [Danielsson 80], [Borgefors 84a], [Borgefors 84b]. and [Verbeek et al. 86].

Unfortunately, the above referenced techniques for digital distance transforms required complete precomputation of the distance after any object undergoes an unpredictable motion (all of the above can be trivially extended to handle periodic motion). However, it is obvious that, in general, even if an object moves unpredictably, there will large portions of the digital distance map that are unaffected.

The major advantages of an algorithm that updates only those portions of the distance map which are actually effected by a moving object are twofold. The most obvious advantage is the savings in computational effort. The less obvious, but possibly equally important, is that if it can be quickly determined that the current location of the "robot" is not in an effected region, the "robot" may be allowed to continue movement toward its goal while the distance map is updated in the background. Thus the robot motion would not be unnecessarily halted if an object entered its envelope but did not effect its goals.

Another possible use of the technique would be as part of high-level path planning in hostile environments. For this application, a system would hypothesize obstacle locations at some future time and could use the "update" algorithm to determine which regions of space could possibly (but not necessarily) interfere with the movement to the current goal. A simple variant of the algorithm could be used to actually predict the minimal velocity of a given obstacle before it could have a potential effect.

The next section presents some background on the constrained distance transform. Following that are sections describing the algorithm, describing the experimental testing, and then discussing the limitations of the current algorithm, and avenues for future research.

2 BACKGROUND: THE CONSTRAINED DISTANCE TRANSFORM

There have been many sequential approaches to the computation of Digital distance maps. These techniques differ mostly in their neighborhood definition, weighting schema, and "sweeping" algorithm. The algorithm for determining regions which will potentially be updated requires knowledge of the neighborhood and weighting functions. This section briefly introduces the constrained distance transform as presented in [Dorst and Verbeek 86] and [Verbeek et al. 86], both of which dependent heavily on the distance transform discussed in [Borgefors 84a].

	n11		n10	
n12	n4	n3	n2	n9
	n5	n0	n1	
n13	n6	n7	n8	n16
	n14		n15	

Location of 16 Neighbors

	d3		d3	
d3	d2	d1	d2	d3
	d1	0	d1	
d3	d2	d1	d2	d3
	d3		d3	

Weights for 16 neighbors

Figure 1: Example showing the numbering of the 16 neighbors and their associated weights for a 2D CDT. For 16 neighbor real valued approximation to euclidean distance, the weights are $d1 = 1$, $d2 = \sqrt{2}$ and $d3 = \sqrt{5}$. The minimal error integer approximation is given by $d1 = 5$, $d2 = 7$ and $d3 =$ [illegible]. For 8 neighbor the real valued approximation uses $d1 = 1$, $d2 = \sqrt{2}$ and $d3 = 0$ and the best integer approximation uses $d1 = 3$, $d2 = 4$ and $d3 = 0$.

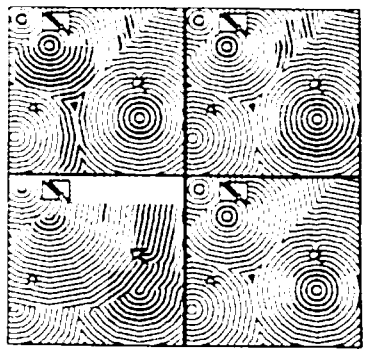

Figure 2: DDM after various passes of constrained distance transform for a scene containing three goal points, and four objects. The lower left shows the DDM after the first pass. Lower right is after the second, upper left after the third and upper right is after the final (sixth) pass. Distances are represented as iso-distance contours. See text for more details.

In its general form, the constrained distance transform (hereafter CDT) allows computation in arbitrary dimensions, and with various neighborhoods. For simplicity, only the calculation of two dimensional digital distance maps (hereafter DDMs) is considered herein.

The general two dimensional algorithm begins with a definition of neighborhoods and weighting functions, see figure 1. This discussion assumes 16 neighbors as in the right of figure 1 and uses the integer approximations to the euclidean distance provided in that figure. Given that neighbors $n_1 \ldots n_{16}$ all have values, $v_1 \ldots v_{16}$ which are the distances from that point to some goal point, then the distance from n_0 is simply $min_{i=0..16}(v_i + w_i)$, where w_i is the associated weighting function.* To allow obstacles in the distance map, one can generalize the values $v_1 \ldots v_{16}$ to include the value $+\infty$ whenever the associated pixel is part of an object.

The serial version CDT algorithm is best described as two alternating passes.† The exact number of alternations required depends on the number, shape and relative location of the obstacles. The odd passes sweep left to right over the image from top to bottom. As the pass proceeds, it updates the value at every non-infinite (i.e. non-obstacle) pixel, say DDM[i,j] according to the equation

$$DDM[i,j] = \min_{i = 0,2,3,4,5,10,11,12} (v_i + w_i) \tag{1}$$

where the v_i are the distance values of the associated neighborhood of DDM[i,j].‡ The even passes sweep from right to left, and from bottom to top . They update the value at every non-infinite (i.e. non-obstacle) pixel according to

$$DDM[i,j] = \min_{i = 0,1,6,7,8,13,14,15,16} (v_i + w_i). \tag{2}$$

Because these passes alternate, the information first flows down and to the right but with some spreading on an angle to the left, see 2. On the alternate pass the information flows up and to the left, with some angular dispersion of information to the right. If there are multiple sources, or obstacles in certain positions, multiple passes will be required to spread the distance information throughout the image.

Given a DDM, the calculation of the shortest path from the current location to a goal point is achieved by simply following the gradient of the map.** A mobile robot actually never needs to calculate the entire path; motion may be determined locally. This property allows the DDM technique to easily and efficiently deal with (although not anticipate) slippage and other problems with the robots inertial guidance. If the robot system has some means of determining its current location, its can continually plan the "shortest" path to its goal even though it can not actually follow that path accurately.

While the expressed purpose of the CDT is to allow one to easily solve the shortest-path problem, it can also be used to deal with the any-path problem. This is accomplished by planning the shortest path to a single "goal" (the one used to compute the DDM) from both the starting point and desired ending point. If there exists a path from the starting point to the "goal" (this can be easily determined from the DDM), then this technique will always find a path. This type of path planning, while generating inefficient path, might be used as a simple mechanism of dealing with unexpected motions required by real time errors in an assembly task.

*The inclusion of zero is necessary because the point under consideration might be a goal point.

†One can actually do the computation in four (or 8 or 16) different passes. For four passes, one computes the above passes as well as one that sweeps top to bottom, left to right and another that sweeps bottom to top, right to left. It has been reported elsewhere, [Verbeek et al. 86], that this reduces the number of passes required to compute the CDT with obstacles.

‡If one uses only eight neighbors, the sum is modified by dropping terms above $i = 8$. This can greatly decrease the running time of the algorithm, but also increases the error of the transform as compared to the true euclidean distance.

**However, due to the nature of digital distances, from any given point there may be two different directions of travel which result in two equi-distance paths. In such cases, the gradient is not uniquely defined, and the algorithm may choose direction of motion path, possibly using some other criterion to make the decision

To summarize, the advantages of the CDT over other techniques for path planning include:

- It is fast and simple to compute. See the experimentation section for example timings.
- It can handle arbitrary shaped obstacles. In fact, they need not even be simply connected. (Multiply connected obstacles can arise when considering path planning in configuration-space.)
- If can be very efficiently updated if objects move.
- It is a better distance measure than true euclidean distance for those simple robotic carts which have limited angular movement (say 16 or 32 possible directions of motion).
- It can even be used to find "good" paths in situations where the robot cannot accurately follow the path (assuming that sensory data can be used to monitor the actual location).
- It is easily extended into higher dimensions (with enough memory).

3 Intelligent Updating of a DDM when Objects Move

This section describes the algorithm for intelligent updating of a digital distance map when some number of objects in a scene have moved. It is assumed that all objects are given by polyhedral approximations. The algorithm for updating is done in two stages, computation of effected regions, and actual update. The latter stage employs the CDT from section 2 with a slight modification so that it does not require the scanning of the entire map, but only sections of said map.

If one considers iso-distance contours in a truly euclidean map without obstacles, the contours form circles. The shortest path from any point to the goal is then a straight line. If however, an obstacle is present, the shortest path is a straight line to the goal if and only if the goal is in "the line of sight" from the current location. Otherwise, the shortest path is a series of lines, to the obstacles, around the obstacles, and to the goal. If one examines the difference between a map without obstacles, and the map with a single obstacle, the difference forms a figure similar to a drawing of the shadowing of light by a planet, with regions of shadow, umbra, and penumbra, see fig 3. However, most of the map remains unaffected, with the percentage of unaffected area depending on the size and location of the obstacle. This type of treatment is easily extended to handle multiple goals. However, the multiple goals are not exactly the same as multiple light sources. The system can be viewed as two logically distinct phases. One phase is the calculation of potential update regions, and the other is the actual update of those regions. If care is taken, spatially disconnected potential update regions may be updated in parallel. These phases are discussed assuming a two dimensional problem using a N by N digital distance map. Note that the algorithm extends into arbitrary dimensions, although with an increase in complexity.

The actual update phase of the algorithm is a simple extension to the constrained distance transform which uses the defined potential update region (defined by an array) to limit the extent of the sweeps in each update pass. Care must be taken to insure that the boundaries of the region are correctly handled. The cost of this phase is generally a small fraction of the cost of a total constrained distance calculation.

The other phase of algorithm, i.e., the one that calculates the potential update region, assumes that each object is uniquely numbered and is described by an array of vertices. It is assumed that obstacles are also explicitly represented in the DDM as boundary pixels connecting the vertices. Boundary values are assumed to be coded such that they can be distinguished from distance values, and such that the index of the associated obstacle can be determined from the pixel value.

In addition to the $N \times N$ DDM and list of objects (stored as vertices), the algorithm uses two auxiliary arrays. The first of these, referred to as the update array, is a $2 \times N$ array of integers which, for each row of the DDM, stores the starting and ending column of the potential region for update. This implies that the update region is defined by raster lines, which is of course a simplification of the true update region. If the extra area updates becomes significant, the algorithm is easily modified to handle more complex descriptions of the update region.

The second auxiliary array, referred to as the status array, is used to store markers indicating which objects has effected, may effect, or will effect the potential update region. When an object moves, its original position (if it existed, which is not assumed) as well as the new position of the object is used to extend the potential update region. Thus the addition of new objects is a simple type of motion.

The calculation of the potential update region is now discussed. Assume that the DDM is correct for the current list of obstacles. Then this phase begins by initializing the *status array* to indicate that for each object which has moved both its old and new positions will contribute to the potential update region.

The calculation of the potential update region is done iteratively, in six steps. These are:

Step 1 For each vertex in the objects definition, the algorithm uses purely local operations to determines which neighbor is closest to the goal, and the direction of travel from that neighbor to the goal. This neighbor is the initial location for expansion.

Step 2 While the current location has not encountered a boundary or another object, and it is not a local minima (this is possible only if the location is a goal, or if it is equi-distant from multiple goal points), then the current location is moved to a neighbor such that the gradient of the distance map at the neighbor leads to the current location. As this path is calculated, each step is checked against the definition of the update region, and if needed, the region is enlarged. Care must be taken to insure proper operation around obstacle vertices.

Step 3 If during the processing of step 2, the path encountered another object, then if said object has not already been added to the region (as determined by the marking array), the object may effect the region, and the object is so marked.

Step 4 With the expansion from a single vertex complete, the system moves on to the next vertex of the current object (assuming that it has one) and goes to step 1. If the current object is completed, the system moves on to the step 5.

Step 5 This step checks to determine if the objects which were labeled as "may effect" will actually effect the region. This is accomplished by determining if the any of the vertices of the object which are in the current update region, could "cast a shadow", i.e. is the local gradient into the object (then no shadow) or around it (shadow). If any vertex could cast a shadow, the status of the object is moved up to "will" effect, otherwise the potential update region is expanded to include any portion of the object boundary in shadow, and then the object's status is reset to having no potential. If any objects remain which will effect the update region, the system returns to step one, otherwise it goes to step 6.

Step 6 At this point, the system has enlarged the potential update region to the size required for the moved objects. It is now necessary to make a single pass over this region and reset all distance values to a large value, so that the actual update pass will correctly operate.

4 EXPERIMENTATION

Since the main goal of this research was the study and development of the algorithm to calculate the actual update region, little attempt was made to optimize the implementation of the update algorithm. However, the speed with which such calculations can be accomplished is of some importance in determining the value of the algorithm. For that reason, this section presents measures of the efficiency of the algorithm on various synthetic examples. While it would be nice to express the running time of the algorithm in terms of some of the input parameters, this seems impractical because the times are dependent on the interactions of the location(s) of the goal(s) as well as the spatial extent and layout of the obstacles.

One alternative is to present machine times, but these are too site dependent to be meaningful. Thus, we present two measures of the efficiency of the algorithm. These are the percentage of the area of the DDM which must be updated, and the relative time spent on each phase of the algorithm. The relative times will be presented in units, η, where one η is the "average" cost of one pass of the constrained distance transformation on a 256 by 256 DDM. Each measurement is an average of 4 runs of the update algorithm. For those readers interested in the actual running time, η, was roughly 4.5 seconds of elapsed time (wall-clock time, not cpu time) and was measured on a Vax750, with a system load of approximately 1. Other researchers have reported running times of 2-3 seconds per pass, see [Dorst and Verbeek 86] and [Verbeek et al. 86]. Note that even these relative measures have dubious meaning, since they depend on the relative portion of the image occupied by the object(s) in motion.

The examples in this section are all completed using 16 neighbor connections on a 256 by 256 (32 bit wide) two dimensional map. The distances were calculated using the integer approximation described in Fig. 1. All examples assume the edges of the map are obstacles. The

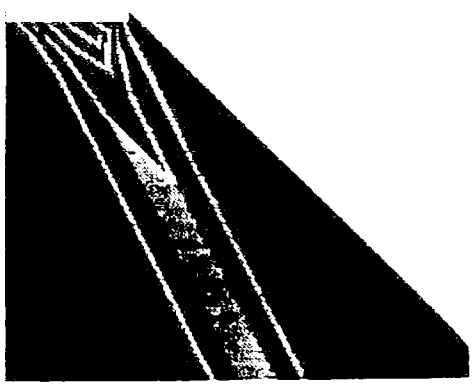

Figure 3: Example showing "shadow" regions for simple objects with a single goal. Obtained by computing the difference between DDM with no obstacles and a DDM with one obstacle.

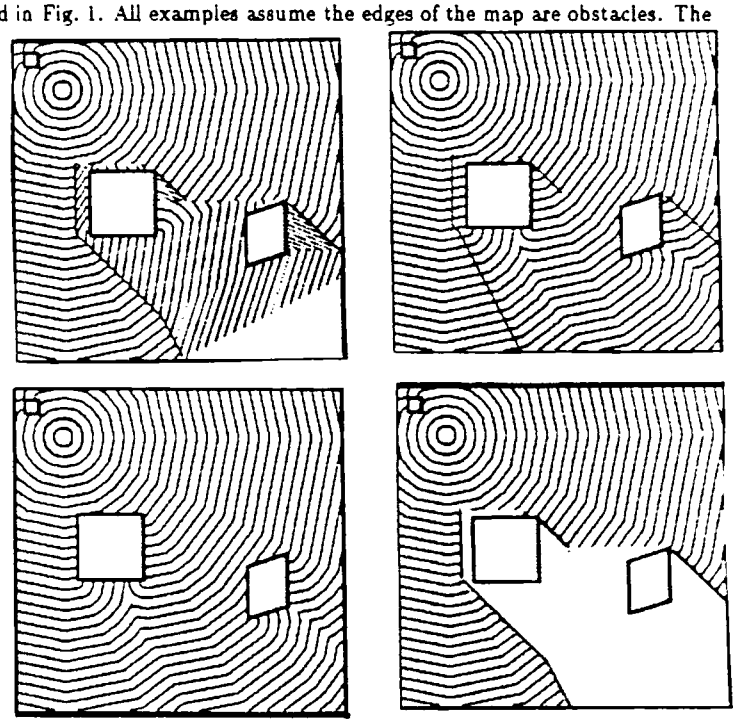

Figure 4: The results of "update" algorithm for a simple scene with 3 objects and one goal. The lower left is the DDM for original object configuration, lower right is the region of potential update (white region with thin black outline), upper left is after 1 pass of update CDT and upper right the final result of the update.

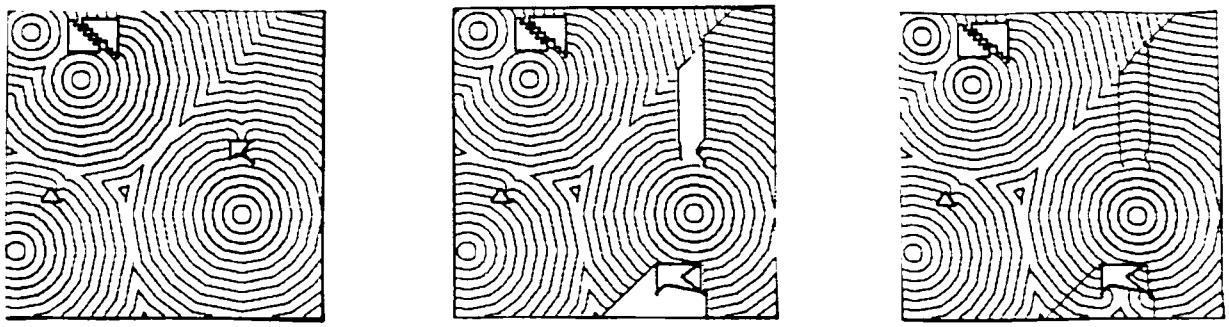

Figure 5: Example containing four goal points, and four objects, where object movement was on a large scale. Also note the termination of region as it enters the area effected by a different goal node.

examples are presented using iso-distance contours, and black outlined objects. The update regions represented as white space (no contour lines) with thin black outlines showing the beginning and end of the update region for each grid line.

The first example of the update algorithm is on a simple scene with three convex objects and a single goal point, see Fig 4. Note how the update region intersects a secondary object, and thus the object is included in determining the potential update region. The calculation of the DDM for these objects takes 5 η. When the large square in the central portion of the scene moves slightly down and to the left, the calculation of the update region and preparation of that region for the update requires .18 η. The percentage of the total area occupied by the update region is 38%. The time required by the actual update in this area is 1.6 η. Thus the update algorithm was approximately 3.1 times faster than recalculation with the CDT which requires 5 η (i.e. passes).

The second example of the update algorithm is on the scene presented in Fig 5, and Fig 2, which has four goal points and 4 objects. The object which moved, did so on a large scale and was also deformed as it moved. Note that as the update region (generated by the upper object position) is naturally terminated by the algorithm as it enters the region of influence of a second goal point. The calculation of the DDM for the original object position takes 5 η. The calculation of the update region takes .0008 η, and the preparation for updating takes .0002 η. The percentage of the total area occupied by the update region is 6.5%. The cost of the update in this area is .2 η. Thus the update

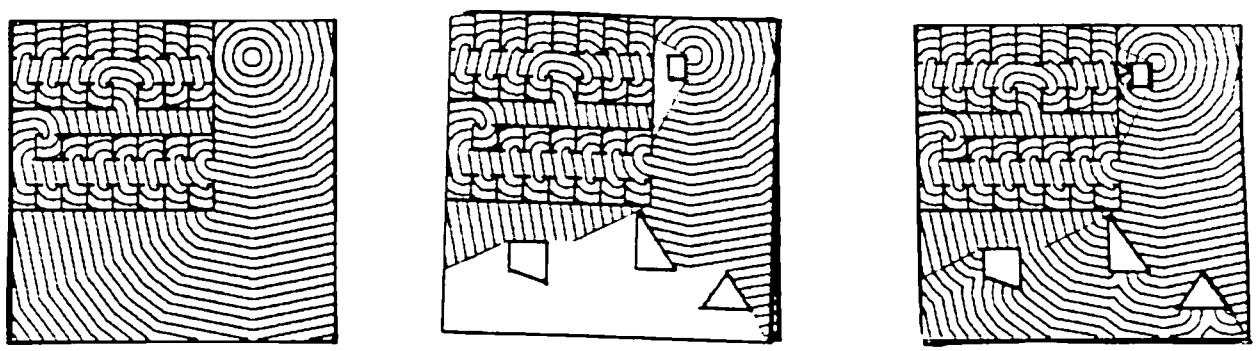

Figure 6: Example with a complex "room" scene. In this case, the motion consisted of adding 4 obstacles. Note that because the object "shadow" of the object nearest the goal falls entirely on one side of a "room wall", the wall does not effect the potential update region.

algorithm delivered approximately a 25 fold speedup.

The final example of the update algorithm is on the complex "room" scene in Fig 6. In this case there is no motion, but rather 4 objects are added to the scene. This example demonstrates the capability of the system to use pre-stored DDMs for complex scene, and then update them if a few objects are added. This technique be especially useful in configuration space where large obstacles (the robot's self intersection) will always exist, and will generate complex DDMs. This example also demonstrates the algorithm's ability not to generate update potentials for large objects when a small shadow falls on them. Again, this is especially important in configuration space where many "large" objects will exist. The calculation of the DDM for the original object position (or the new position) objects takes 9 η. The calculation of the update region takes .0008 η, and the preparation for updating takes .001 η. The percentage of the total area occupied by the update region is 25.8%. The cost of the update in this area is .99 η. Therefore the update algorithm delivered approximately a 9 fold speedup.

In another experimental setup, an object was placed so as to increase the cost of every distance in the "room" portion of the example in Fig 6. This results in the entire room portion of the scene being added to the update region. Thus the update would still require substantial computation time (9 passes over that region). However, the algorithm can determine the update region in only .01 η. If the path planning were currently occurring outside the update region, the path planning could continue as the DDM is updated in the background. This ability

becomes increasingly important in higher dimensions, and as the resolution of the DDM is increased.

5 Conclusions, Limitations and Future Work

This paper presented an efficient algorithm for the updating of digital maps when objects undergo motion, or are added to a precomputed DDM. The algorithm can intelligently deal with multiple goal points, arbitrary object interaction, including not increasing the update region when a a small object moves in front of a large object. In situations where a small percentage of the DDM is effected, the algorithm can save significant computation time. Additionally, the algorithm can quickly decide if a given location will be potentially effected by the movement.

The major limitations of this approach are actually limitations of the DDM. These include: possible exorbitant memory requirements (especially for higher dimensions), the use of discrete distances, and the difficulty of representing obstacles in configuration space.

The current realization of the algorithm requires very large amounts of memory for higher dimensional problem. The interested reader may consult [Verbeek et al. 86] for an estimation of memory requirement in various dimensions and resolutions. While the DDM requires considerable memory, much of it is free space, and should be capable of being represented more compactly. Future work will explore the possibility of using quad-trees, oct-trees or related structures, e.g., see [Soetadji 86], [Kambhampati and Davis 85].

The algorithm presented simply provides a means for updating the DDM after object motion is known. However, if the DDM is being used to represent configuration space, the calculation of obstacle motions is highly non-trivial, and may actually dominate the cost of updating the DDM. One avenue for future work is to investigate the use of rough but conservative approximations to the configuration space volume of an obstacle, e.g. see [Lozano-Pérez 87]. Then if the rough obstacle has no potential to effect the current path, the detailed calculations can be done in the background.

It is interesting to note that the regions that are updated, generally can be divided into three regions, the "fringe", the "umbra" and the "penumbra", see Fig. 3. Oddly, the pointwise difference between the distance in the penumbra region, and the distance in the same location without the associated object present is constant. Similarly, the difference between a large port of the umbra region is extremely regular. Future research will attempt to determine if this observation can be exploited to further reduce the complexity of updates.

Acknowledgments

This work was supported in part by Darpa grant #N00039-84-C-0165. The author would also like to thank Leo Dorst for introducing him to the use of the CDT, and Peter Allen for his comments on this manuscript.

References

[Barrow et al. 77] H.G. Barrow, J.M. Tenenbaum, R. C. Bolles, and H.C. Wolf. Parametric correspondence and chamfer matching: two new techniques for image matching. In *Proceedings of the Fifth International Joint Conference on Arificial Intelligence*, pages 659–663, Cambridge, MA, 1977.

[Borgefors 84a] G. Borgefors. Distance transforms in arbitrary dimensions. *Computer Vision, Graphics, and Image Processing*, 27:321–345, 1984.

[Borgefors 84b] G. Borgefors. Distance transforms in digital images. *Computer Vision, Graphics, and Image Processing*, 34:344–371, 1984. Also available as FOA report C 30401-E1, National Defence Research Institute, Linkoping Sweeden.

[Danielsson 80] P.E. Danielsson. Eculedian distance mapping. *Computer Vision, Graphics, and Image Processing*, 14:227–248, 1980.

[Dorst and Verbeek 86] L. Dorst and P.W. Verbeek. The constrained distance transformation: a pseudo-euclidean, recursive implementation of the Lee-algorithm. In I.T. Young Et Al., editor, *SIGNIAL PROCESSING III: Theories and Applications*, Elsevier Science Publishers B.V. (North-Holland), 1986.

[Kambhampati and Davis 85] S. Kambhampati and L.S. Davis. Multi-resolution path planning for mobile robots. In *Proceedings of the DARPA Image Understanding Workshop*, pages 421–432, DARPA, December 1985.

[Lee 61] C.Y. Lee. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, 346–365, September 1961.

[Lozano-Pérez 87] Tomás Lozano-Pérez. A simple motion-planning algorithm for general robot manipulators. *IEEE Journal of Robotics and Automation*, RA-3(3):224–238, June 1987.

[Montanari 68] U. Montanari. A method for obtaining skeletons using a quasi-eculidean distance. *Journal of the ACM*, 15:600–624, 1968.

[Rosenfeld and Pfaltz 66] A. Rosenfeld and J. Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM*, 13:471–494, 1966.

[Rosenfeld and Pfaltz 68] A. Rosenfeld and J. Pfaltz. Distance functions on digital pictures. *Pattern Recognition*, 1(1):33–61, 1968.

[Soetadji 86] T. Soetadji. Cube based representation of free space for the navigation of an autonomous mobile robot. In L.O. Hertzberger and F.C.A. Groen, editors, *Proceedings of the International Intelligent Autonomous Systems conference*, pages 546–561, North-Holland, Amsterdam, Netherlands, December 1986.

[Verbeek et al. 86] P.W. Verbeek, L. Dorst, B.J.H. Verwer, and F.C.A. Groen. Collision avoidance and path finding through constrained distance transformation in robot state space. In L.O. Hertzberger and F.C.A. Groen, editors, *Proceedings of the International Intelligent Autonomous Systems conference*, pages 627–634, North-Holland, Amsterdam, Netherlands, December 1986.