On the complexity of constrained distance transforms and digital distance map updates in two dimensions

Terrance E. Boult
Columbia University Department of Computer Science
New York City, New York, 10027. tboult@cs.columbia.edu
Technical Report CUCS-454-89

June 21, 1989

Abstract

Using digital distance maps, one can easily solve the shortest path problem from any point by simply following the gradient of the distance map. Other researchers have developed techniques to quickly compute such maps. One such technique, the constrained distance transform, is described and a computational complexity analysis is presented.

Unfortunately, algorithms for digital distance maps have all assumed a static environment. This paper extends the usefulness of the planar digital distance maps by providing an efficient means for dealing with obstacles in motion. In particular, an algorithm is presented that allows one to compute what portions of a map will probably be effected by an obstacle's motion. The regions that must be checked for possible update when an obstacle moves are those that are in its "shadow", or in the shadow of obstacles that are partially in the shadow of the moving obstacle. The technique can handle multiple fixed goals, multiple obstacles moving and interacting in an arbitrary fashion. A complexity analysis and short verification is presented.

The algorithm is demonstrated on a number of synthetic two dimensional examples, and example timing results are reported.

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 [1]. 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 both parallel and sequential algorithms which compute digital distances assuming various neighborhood and approximations, e.g. see [2], [3], [4], [5], [6], [7], [8], and [9]. Unfortunately, these techniques for digital distance transforms required complete recomputation of the distance after any obstacle underwent any motion. However, it is obvious that, in general, as an obstacle moves there will be 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 obstacle are twofold. The most obvious advantage is the savings in computational effort. 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 obstacle entered its work 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 digital distance maps, including their computation with the constrained distance transform, and a description of our simple modification to that algorithm. Also included in that section is a complexity analysis of the constrained distance algorithm, which is a necessary part of the complexity analysis of our update algorithm. Following the background section is a description of the update algorithm with an informal verification of correctness and a complexity analysis. That is followed by a section presenting some experimental tests, and then a section discussing the limitations of the current algorithm, and avenues for future research.

2 Background: digital distance maps

This section discusses some background and motivation for the use of digital distance maps (hereafter DDMs). Then we discuss a technique for the computation of DDMs, due to [9], called the the constrained distance transform (hereafter CDT). We also present a complexity analysis of that algorithm.

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. Due to the discrete nature of digital distances, from any given point there may be two different directions of travel which result in 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.

An important observation is that while planning a path (say for a mobile robot) one does not need to calculate the entire path; optimal direction of travel 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 robot's inertial guidance. If the robot system has some means of determining its current location, it can continually plan the "shortest" path to its goal even though it can not actually follow that path accurately.

While it is obvious that the DDM can easily solve the shortest-path problem, it can often be used to deal with the find-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 both the starting and ending points have distance values in the DDM, then this technique will find a connecting path. This type of any-path planning, while generating inefficient paths, might be used as a simple mechanism for dealing with unexpected motions required by real time errors in an assembly task. In general, a failure of this process to find a path does not imply that no path exists. However, if one can show that the DDM does not contain disconnected regions (easily determined if there is only one goal), then the approach finds a path if and only if one exists.

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

Location of 16 Neighbors

	d_3		d_3	
d_3	d_2	d_1	d_2	d_3
	d_1	0	d_1	
d_3	d_2	d_1	d_2	d_3
	d_3		d_3	

Distance Weights for the 16 neighbors

Figure 1: Example showing the numbering of the 16 neighbors and their associated distances (weights) for a 2D CDT. For 16 neighbor real valued approximation to Euclidean distance, the weights are $d_1 = 1$, $d_2 = \sqrt{2}$ and $d3 = \sqrt{5}$. The minimal error integer approximation is given by $d_1 = 5$, $d_2 = 7$ and $d3 = 11$. For 8 neighbors, the real valued approximation uses $d_1 = 1$, $d_2 = \sqrt{2}$ and $d3 = 0$ and the best 8-neighbor integer approximation uses $d_1 = 3$, $d_2 = 4$ and $d3 = 0$.

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

- It is fast and simple to compute. (See the experimentation section for example timings.)
- It can handle arbitrarily shaped obstacles. In fact, they need not even be simply connected.
- 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)*
- If can be updated efficiently if obstacles move.

There have been many approaches to the computation of digital distance maps. Parallel algorithms proceeds in a "brush-fire" type approach and sequential algorithms "sweep" over the map in passes, e.g., see [2], [4], [5], [6], [7], and [9]. The sequential techniques differ mostly in their neighborhood definition, weighting schema, and "sweeping" algorithm.

Let us now briefly introduce one technique for DDM computation, the constrained distance transform as presented in [10] and [9], both of which dependent heavily on the distance transforms discussed in [7].

In its general form, the constrained distance transform (CDT) allows computation in arbitrary dimensions, and with various neighborhoods. For simplicity, only the calculation of two dimensional 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 left 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. (The inclusion of $(v_0 + w_0)$ is necessary because the point under consideration might be a goal point.) 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 obstacle.

The CDT algorithm assumes that the map is initialized with obstacles at $+\infty$, goal points at 0, and all other points at an initial value greater than the distance of any point in the scene from a goal (as you will

*For example, see [9]. We will only consider the planar case.

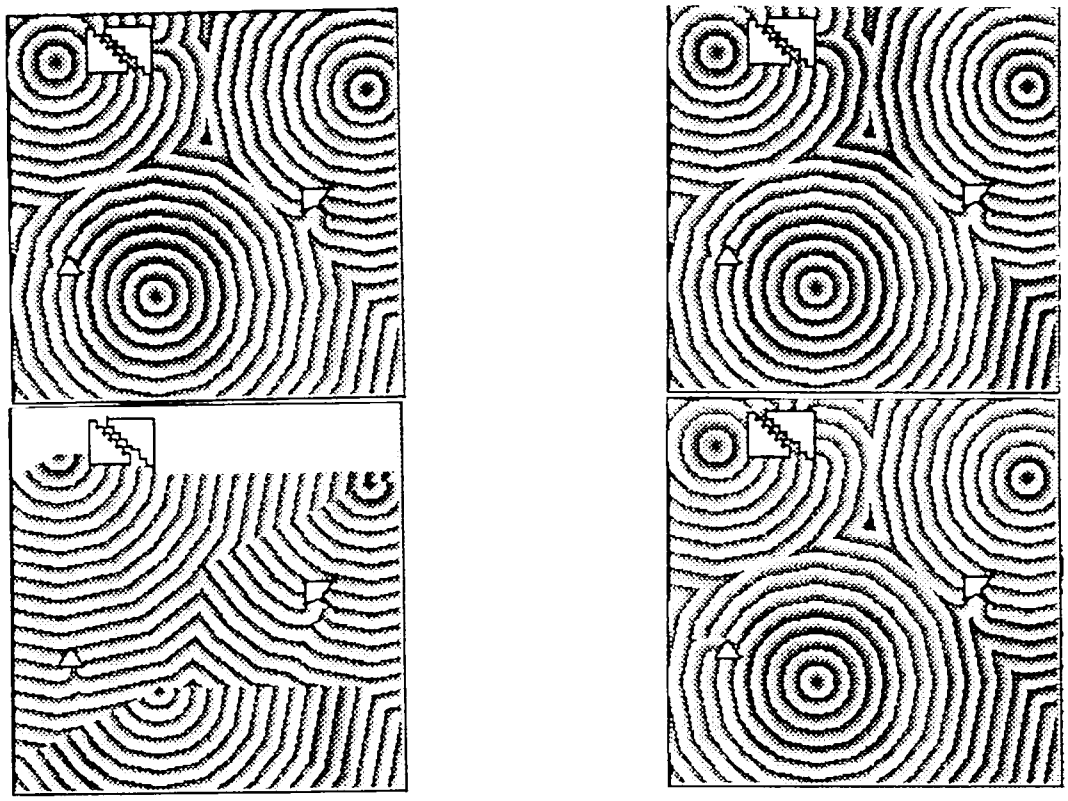

Figure 2: DDM after various passes of constrained distance transform for a scene containing three goal points, and four obstacles. The lower left shows the DDM after the first pass. Lower right is after the second, upper left after the third and upper right is after the final (sixth) pass. Distances modulo 64 are represented as intensities. See text for more details.

see the CDT will only reduce the computed distance at a point.) While previous papers have not explicitly mentioned it, the CDT assumes that it is given a region of interest, i.e., region of computation, in the past this has been the entire image. We assume the region of interest is given by a polygonal boundary. This boundary is not necessarily treated as an obstacle, i.e. points outside the region may be used in the computation of distances. However, if points outside the region are used, it must be assumes that the exterior points contain estimates greater than or equal to the correct distance.

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 over the region of computation from left to right, and 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

$$DDM[i,j] = \min_{i=0,2,3,4,5,9,10,11,12}(v_i + w_i) \quad (1)$$

where the v_i are the distance values of the associated neighborhood of DDM[i,j]. 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.

The even passes sweep from right to left, bottom to top. They update each non-obstacle point in the

[†]One can actually do the computation in four (or 8 or 16) different passes. It has been reported elsewhere, [9], that using four passes reduces the total number of passes required to compute the CDT with obstacles. However, as will be shown later, the worst case complexity is not reduced as the number of passes (or directions) increases.

region of interest according to

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

Because these passes alternate, the information first flows down and to the right but with some spreading on an angle to the left, see Figure 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.

The odd and even passes of the CDT are alternated until there is no change in the image. The result is a DDM. Later in this section we will present a complexity analysis of the CDT in terms of the size of the image, and the number of obstacles.

2.1 Our modifications to the CDT algorithm

To make the CDT algorithm more useful for update algorithm, we have made two modifications. The subsection describes these modifications and their implications.

The first modification is that the algorithm does not sweep over the entire image, but rather over a polygonal region of interest. The points outside this region are not necessarily treated as obstacles, but rather w assume that the distance estimates outside the region are greater than or equal to the correct distances. In the implementation used for the examples in this paper, the region of interest is restricted to a set of rasters, i.e., each row of the DDM is either entirely outside the region of interest, or there is exactly one starting point and one ending point for the region of interest in that row. (This was convenient for implementation, but does unnecessarily increase the size of the region of interest.) The implications of this modification are that we can apply the CDT to subsets of the image at a lower cost, see section 2.2.

The second modification is that we allow the region of interest to dynamically change. In particular, if in some pass of the CDT, a portion of a particular row, say j, is modified by the pass, and if j is also on the boundary of the region of interest, then the points outside the region of interest which have neighbors in the modified portion of j, may potentially change. In this case, we increase the size of the region of interest to include those regions of potential change. The second type of dynamic behavior for the region of interest is shrinking. Assume that rows j through $j+i$ are not changed in some pass of the CDT. Then it is easily shown that either they will not effect later computations, or their role in later computations will be caused by changes external to those rows. Thus, we can safely remove rows j through $j+i$ from the region of interest. If some portion of those rows should be needed for later computation, the above described mechanism for growing the regions will insure that the region is correctly included in the region of interest. The correctness of these modifications are not included in this paper, but follow directly from the proof on the complexity of the algorithm in Section 2.2.

The impact of the second modification to the CDT again effects the computational efficiency of the algorithm. Region growing also has the important effect of allowing us to specify a region smaller than the actual update region, so long as we can insure that the "missed" region directly borders the current region of interest, and that the distance estimates in the non-included region are greater than or equal to the true distances.

2.2 The complexity of the CDT algorithm

This section examines the (sequential) computational complexity of the constrained distance transform. The analysis of the complexity produces, as a by product, a verification of the correctness of the CDT

algorithm.

The complexity of each pass is obviously $\Omega(nm)$ where m is the number of cells in the region of interest (the entire map in [10] and [9]) and n is the size of the neighborhood used in the algorithm (n=16 in our case). If the size of the region of interest is constant at m, then the overall complexity of the CDT is obviously $\Omega(\rho mn)$ where ρ is the number of passes required to correctly compute the distance estimate at all points in m.

We now show that, in the worst case, the number of passes required by the CDT algorithm is $\Omega(v)$ where v is the number of distinct vertices (assuming edges defining the obstacles intersect only at vertices). Thus the worst case complexity of the CDT algorithm is $\Omega(vnm)$. We note, however, that in practice we find the number of passes to be $\ll v$. The proof is to derive first an upper bound for the cost of the CDT, then to present a family of obstacles such that they will provide a lower bound on the complexity. Since the upper and lower bounds agree we obtain the above result.

2.2.1 Upper bound on the number of passes of the CDT

Theorem 1 *Let O be a set of obstacles defined with v distinct vertices, such that edges only intersect at vertices. Let M be a region of interest such that each point in M currently contains an estimate of the distance to the goal which is greater than or equal to the correct distance. Assume that the distance between points is spatially invariant. Then, the constrained distance transform will require $\leq 2 + v$ passes to correctly compute the distance from each point to the nearest goal.*

To prove this above theorem, it is convenient to first prove two lemmas. The proof of the lemmas prove, and provide the bulk of the proof of correctness of the CDT algorithm.

Lemma 1 *Given that DDM at point p contains the correct distance to a goal, then the "odd" pass of the CDT algorithm correctly determines the distances to all points q in the DDM such that there exists a minimal distance path from p to q where each step of the path is to one of the neighbors $n_j, j = 1, 6, 7, 8, 13, 14, 15, 16$. (This is assuming that the current distance estimate stored at each point in the region of interest is greater than or equal to the correct distance.)*

Proof.

The proof is by induction on g, the number of steps (not the distance) in the minimal distance path from p to q.

For $g = 0$ the lemma is trivial.

Assume the lemma is true for all q meeting the conditions of the lemma with the minimal distance path from p to q involving $\leq g - 1$ steps. Let q^* be a point such that there exists a minimal distance path from p to q^* requiring exactly g steps, where each step of the path is to one of the neighbors $n_j, j = 1, 6, 7, 8, 13, 14, 15, 16$. Obviously, all the points on the minimal distance path from p to q^* satisfy the conditions of the lemma, and thus, by induction, the odd pass of the CDT correctly computes the distance to each. Recall that in the odd pass we will compute the distance to q^* as $\min_{i=0,2,3,4,5,9,10,11,12}(v_i + w_i)$. Now note that by the above assumptions, at least one of the points on the path from p to q^* will be one neighbors of q^* involved in the minimum used to compute its distance. Call this point q'. For example, let q' be the element of the path just before q^*, and assume that the step to q^* was a step to the neighbor n_{13} of q', then q' is neighbor n_9 of q^*.

Since we assumed that the original values of all points in the area were greater than or equal to the correct distance, and since by induction the distance computation at q' is correct, the odd pass of the CDT will correctly determine the distance to q^*. ∎

Lemma 2 *Given that DDM at point p contains the correct distance to a goal, then the "even" pass of the CDT algorithm correctly determines the distances to all points q in the DDM such that there exists a minimal distance path from p to q where each step of the path is to one of the neighbors* $n_j, j = 2, 3, 4, 5, 9, 10, 11, 12$. *(This is assuming that the current distance estimate stored at each point in the region of interest is greater than or equal to the correct distance.)*

The proof is essentially the same as the proof of Lemma 1.

Proof of Theorem 1.

Assume the theorem does not hold. Then there exists some point, say r, such that the computation of the correct distance from a goal to r requires $t > 2 + v$ passes. It is well known that the shortest path from a point to a goal is a collection of straight lines connecting, the point, obstacle vertices, and a goal point. Noting that a obstacle vertex can be encountered only once on the shortest path from r to a goal, and recalling that there are v vertices, this allows one to conclude that the shortest path from r to a goal point consists of $l \leq v + 1$ straight lines. Call these line segments $s_i, i = 1 \ldots, l$, with the index i increasing with the distance from the goal point.

Since each s_i involves only one direction, we known, by lemmas 1 and 2, that distance information will be correctly spread along each s_i in *at most* one pass per segment. We say at most one pass, because it is possible for s_i and s_{i+1} to involve directions such that one pass may correctly propagate information along both segments, e.g., a 0° segment connected to a $-45°$ segment would both have information correctly propagated in the same odd pass. In practice, this situation is rather common, and is one of the reasons that in the number of passes is generally much less that worst case bound. Note that it is likely that s_i is not a line in one of the allowed primitive directions. However, it is straightforward to show that any segment s_i line can be approximated by a digital line using only primitive directions from one of the subsets used in lemmas 1 and 2. Since the components of this approximation come from the same subset of primitive directions, information would propagate through the entire digital approximation in a single pass.

From the above we see that after information starts propagating it will require at most l passes to propagate distance information from a goal point to the point r. If the initial direction of s_1 is not through a neighbor $n_j, j = 1, 6, 7, 8, 13, 14, 15, 16$ of the goal point, that the first pass will not propagate any information, otherwise information begins propagating on the first pass. Thus we can conclude that, t, the number of passes requires satisfies $t \leq l + 1$. However, by assumption, $t > 2 + v \geq l + 1$. Thus we have a contradiction, and the theorem holds.

∎

2.2.2 Lower bound on the number of passes of the CDT

The lower bound on the number of passes, in the worst case, is obtained by developing a family of scenes where the CDT algorithm is required to take $\frac{v}{2}$ passes.

Before we can discuss the description of the family of scenes which provides the lower bound on the number of passes of the CDT, we note that the number of allowed vertices is not independent of the size

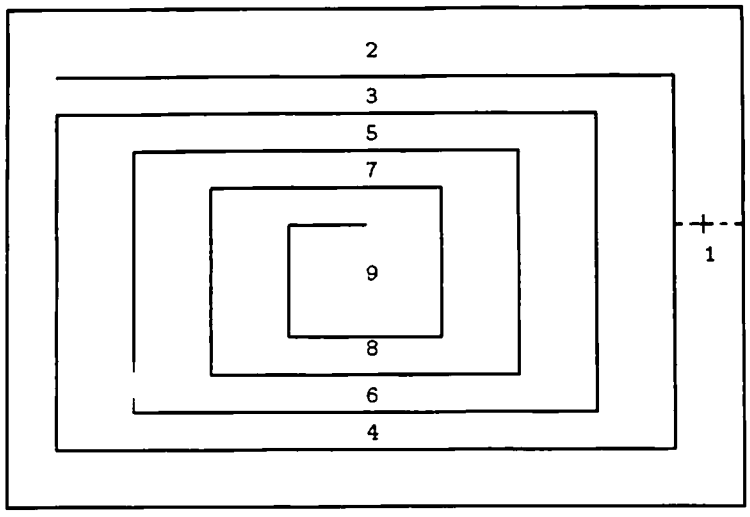

Figure 3: A scene showing that the CDT can require $\frac{v}{2}$ passes to complete distance map. Goal is +. Regions (bordered by edges and dotted lines) are labeled by the pass number which determines distances in that region.

of the region of interest. Thus, when we we speak of the number of passes required being $O(v)$, we assume that the region of interest is sufficiently finely divided (and large enough) that the vertices, the edges, and the path to the goal can all be represented.

The family of scenes for the lower bound are based on a "spiraling square" pattern, as typified in figure 3. As can be easily verified, each pass of any CDT will be able to make only one 90° turn (it will partially turn a second, leaving enough information that the next pass will not have to turn that corner either). That this is true for any CDT follows directly from the fact that each CDT pass must propagate information in less than 180°. As a result, we can conclude that each pass (after up to 3 initial passes to get things started) will propagate information around 2 corners. Given that the spiraling square with v vertices has $v-5$ corners, we can conclude that any CDT can be required to take at least $3+\frac{v-5}{2} > \frac{v}{2}$ passes.

2.2.3 The Computational Complexity of the CDT

Recall from before that we had shown that if the size of the region of interest is constant at m, and the neighborhood size n, then the overall complexity of the CDT is obviously $\Omega(\rho mn)$ where ρ is the number of passes required to correctly compute the distance estimate at all points in m.

Combining the results of the two previous sections we see that the lower bound on the number of passes is $\frac{v}{2}$, and the upper bound is $v+2$. Thus the number of passes is $\Omega(v)$, and the overall complexity of the CDT for fixed distance costs is $\Omega(vmn)$. Again we comment that in practice, the experienced number of passes has been considerably less than predicted by the worst case.

One might consider that making restrictions on the obstacles, e.g., only allowing convex polygons, might reduce the worst case complexity results. However, unless restrictions are also placed on the locations of obstacles, then for many restricted classes of obstacles a similar construction to the lower bound will result in the bounds differing only by a constant (e.g., for convex obstacles replace each line segment in the construction by a thin rectangle.) An analysis of the expected number of passes would, however, be interesting. Results on the expected number of turns in a shortest path for planar obstacles would directly

result in upper bounds on the expected cost of the CDT.

3 Intelligent updating of a DDM when obstacles move

This section describes the algorithm for intelligent updating of a digital distance map when some number of obstacles in a scene have moved. We begin with an informal introduction to the algorithm, then present a more formal definition and verification.

3.1 Informal description of algorithm

If one considers iso-distance contours in a plane, the contours form "circles" and the shortest path from any point to the goal is radial 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 figure 10. However, most of the map remains unaffected, with the percentage of unaffected area depending on the size and location of the obstacle. Thus, if we wanted to recompute a DDM with the new obstacle, only the "shadow" area would need consideration. Similarly, if an obstacle disappeared, only the region it used to shadow would need to be updated. This type of treatment is easily extended to handle multiple goals, although "distance shadow" interaction is not analogous to regular shadow interaction.

To compute the shadow, the system follows the negative of the gradient of the DDM from each vertex until it terminates on the map boundary, hits another obstacle, or reaches a local minima. If the shadow is terminated by hitting another obstacle, the system determines if it needs to add that obstacle to the potential update region as well.

The above shows what happens when an object suddenly appears from nowhere, or disappears from sight. An object that moves can be considered an object suddenly disappearing and reappearing in another location. The algorithm takes an object that moved, computes its shadow in its original and moved locations, and then takes the union of these regions. Each object that is encountered in the shadow expansions may also generate a shadow, though it is not necessary that it does so. As an analogy, consider a shadow falling on a large table, if the shadow falls entirely on the top it does not effect the area below, however, if it covers the edge of the table, it may (depending on the light angle) effect the lighting below parts of the table.

Once the update region is computed, we initialize this region of interest and then recompute the distances using our modification of the CDT so that it operates only on the region of potential update, which it may dynamically change. Initialization is important because the correct operation of the CDT assumes the distance estimates are greater than or equal to the correct value.

3.2 A more formal definition of the algorithm

This section includes a more formal description and verification of the algorithm. We begin with a detailed description then follow with the verification.

We assume an $N \times N$ DDM and a list of obstacles are stored both as filled regions in the DDM, and also as lists of vertices. In addition the algorithm uses auxiliary arrays.

The first of these arrays, 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 introduced by this assumption 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 obstacles have effected, may effect, or will effect the potential update region. The computation of these state variables is discussed momentarily.

The final auxiliary array, the terminator array, is used to store the location of the termination of each shadow line. These values are needed to allow us to connect these locations to finalize the computation of the update region.

Before we discuss the computation of the potential update region we comment on obstacle motion. We are assuming that some external mechanism is providing the location of each obstacle at discrete time events. When an obstacle is not in the same position in two successive maps, its location before and after the motion are used to update the DDM. This formulation is convenient because it removes the need for matching obstacles, there is no need to know the obstacle motion, only its initial and final positions.

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 obstacle which has moved both its "old" and "new" positions will contribute to the potential update region, and are treated independently.

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

Step 1a (deletion of obstacle) For each vertex in the obstacles definition, the algorithm uses neighborhood operations to determine the distance of the vertex to the goal. We need to do this because the vertex is treated as part of the obstacle, and thus does not usually have a distance value. Using this "hypothesized" distance value, the algorithm determines which of the neighbors would have smaller distance values if they went through the vertex, see Figure 4. The neighbor, who would like to go through the vertex, and whose direction of motion is furthest from the "inside" of the object is considered the point of expansion. (This corresponds to the negative of the local gradient at the vertex. While in general, there might be two negative gradient directions, one of them would not have its distance effected by the vertex distance and so there is no ambiguity in this case).

Step 1b (obstacle addition) For addition of an obstacle, the potential vertex location is determined, and the negative of the local gradient is computed at that location (with ambiguous choices broken in favor of the one "closer" to edge of the obstacle). This becomes the point and direction of expansion.

Step 2 If the current location has not encountered a boundary or another obstacle, and if 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 in the negative gradient direction (i.e., to a neighbor such that the gradient of the distance map at the neighbor leads to the current location.) Given that the DDM often has two "gradient" directions, the direction which generates a larger updated region is chosen, see Figure 4. As the expansion path is calculated, each step is checked against the definition of the

[‡]While not part of our implementation note that if care is taken, spatially disconnected potential update regions might be updated in parallel.

update region, and if needed, the region is enlarged. If the point cannot be expanded, the current point is labeled a termination and we move on to step 3.

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

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

Step 5 Previous processing may have resulted in a number of terminated paths which must be connected to complete the expansion of the update region. Each of the termination points has associated with it knowledge of which direction is the "inside" of the region. One then connects the terminations by following the (boundary,object face, object shadow, equi-distant contour) where the termination occurred in the specified direction until it either connects to another termination or the vertex which generated it.** Example can be found in Figure 7. As the connections are determined, the potential updates regions are adjusted to include the new area.

Step 6 This step checks to determine if the obstacles which were labeled as "may effect" will actually effect the region. This is accomplished by checking if the obstacle has any vertices which are contained, or directly adjoining the potential update region. If any vertex of an obstacle is within (or on the boundary of) the potential update region then the status of the obstacle is moved up to "will" effect, or else the obstacle's status is reset to having no potential. If any obstacles remain which will effect the update region, the system returns to step one, otherwise it goes to step 7.

Step 7 At this point, the system has enlarged the potential update region to the size required for the moved obstacles. 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.

3.3 Verification of the algorithm

We now present a semi-formal verification of the idea underlying the algorithm. While the algorithm is based on the use of digital distance maps, the following discussion will assume a continuous formulation because it simplifies many of the arguments. In particular, we take advantage of the fact that in a continuous distance map, points with multiple shortest paths to a goal form a set of measure zero. By contrast, there are almost always multiple shortest paths in a digital distance map (because of the limited directions of movement).

We present arguments to show the algorithm correctly determines the distance map when the distance to any vertex of an obstacle undergoes a known change. The modification to the digital distance map is comprised of two parts, the determination of a potential update region, and the application of our modified CDT.

Without loss of generality, we shall assume the obstacles in question are simply lines $L_i = (l_{i_1}, l_{i_2}). l_{i_j} \in \mathbb{R}^2$. Because of the discrete nature of a DDM, the only obstacles that can exist are polygons, which are easily represented as a collection of lines.

In the discussion to follow, we discuss both the single goal and the multiple goal case. We show that in the single goal case the system always determines the complete region of potential update, while in the

**The latter case occurs for non-convex obstacles.

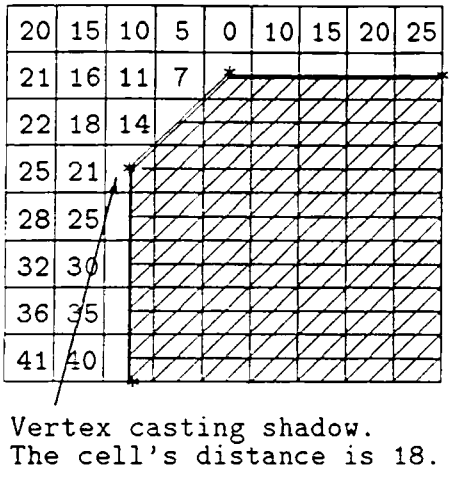

Figure 4: Example showing a part of a DDM and a vertex casting a shadow. The cell containing the vertex is a distance of 18 from a goal (using the current DDM to compute distance).

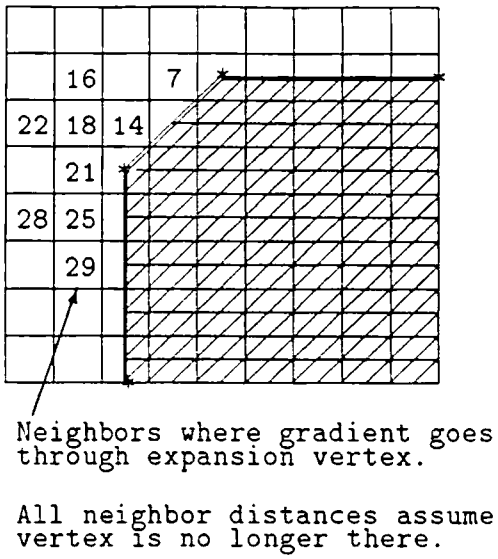

Figure 5: Example showing distances of neighbors of the shadowing vertex assuming it is removed from the scene. The neighbors of potential expansion (where the gradient of the DDm goes through the shadowing vertex's cell) are also marked.

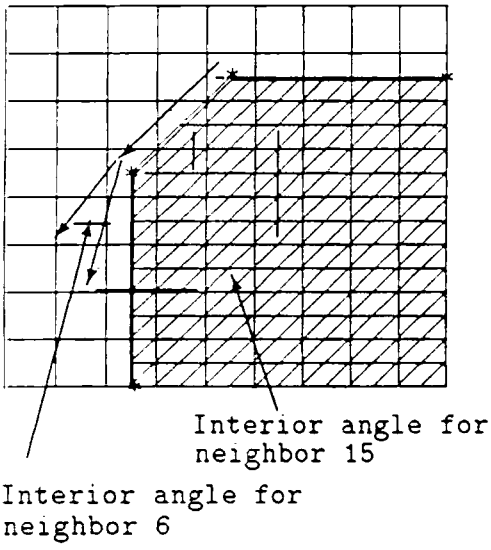

Figure 6: Example showing angles used to determine which neighbor for expansion generates a larger update region (i.e., the one with a larger angle). Vertex used to determine angle is adjacent to the shadowing vertex and in the direction of decreasing distance to the goal.

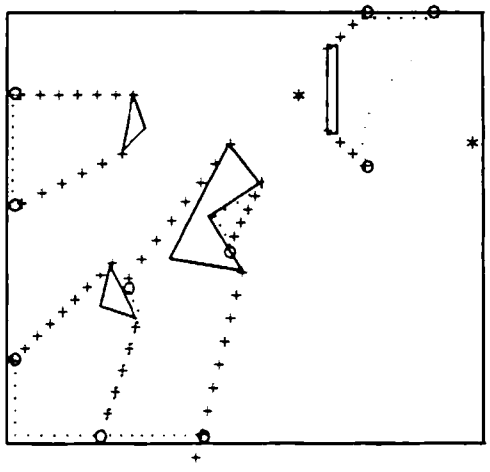

Figure 7: Scene showing two goals (*), shadows (lines of +), typical terminations (marked as o), and the closure of the terminations (dotted lines). (Note that this is not system output.)

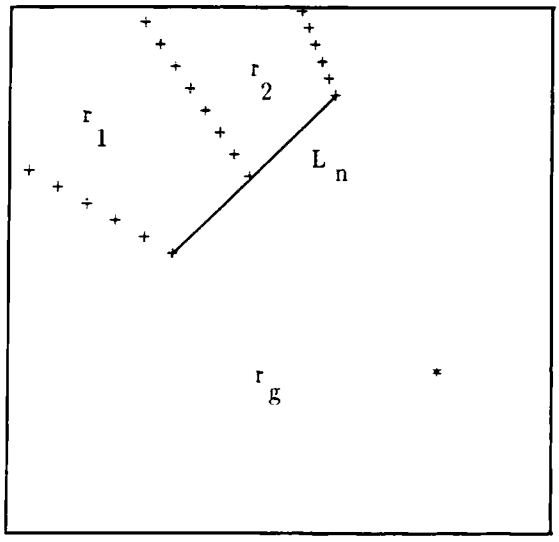

Figure 8: Scene showing regions of distance map for a single obstacle. Goal is *, shadows are lines of +, see text for more details.

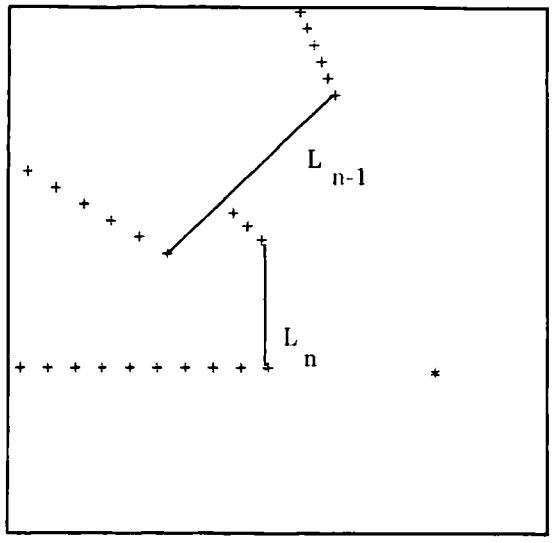

Figure 9: Scene showing interaction of object shadows. Only the L_n moved, but its shadow caused L_{n-1} to also cast a shadow.

multiple goal case, the system may determine a region smaller than the actual region of potential update. However, we also show that the portion of the potential update region not determined by the algorithm has the property that its distance to a goal is non-increasing. This fact, coupled with the modified constrained distance transform discussed in the previous sections, which will increase its region of interest (i.e., domain of application) and as long as the distances are decreasing, this will insure that the system correctly updates the distance from any point to the nearest goal.

As in previous sections, we shall use the term shadow of an obstacle to refer the region of potential update, and the shadow of a point to refer to the line extending from that point in direction of the negative of the gradient of the distance map.

The verification of the algorithm is by induction on n, the number of obstacles in the scene.

Basis step, n = 1;.

In this case, L_n is the only obstacle. The distance map for the scene can be partitioned into three regions, r_g, r_1, r_2, where all points in r_g have their shortest path directly to the goal point and points in r_i have shortest paths to the goal which pass through the point L_{n_i}. Points on the boundary of a region (a set of measure zero), can be considered in either region. Figure 8 shows a possible configuration.

We note that for the single goal case, the boundary between regions r_g and r_1 is exactly the line generated by extending the negative of the gradient vector of the the distance map at the point L_{1_1}. Similarly the boundary between regions r_g and r_2 can be obtained from the negative of the gradient of the map at the point L_{1_2}. The boundary between regions r_1 and r_2 is not a simple as the other region boundaries (it is not drawn correctly in 8). While this boundary might be useful (if the distance to one vertex changed while the other remained fixed), it is considerably more difficult to determine, and thus we will not consider it further.

If we were to change the distance from L_{1_1} and/or L_{1_2} to a goal, then the potential update region would

obviously be region contained in $r_1 \bigcup r_2$. Note that this region can be determined simply by following the negative of the gradient from the vertices of L until they terminate on the edge of the map. The procedure, which is used by our algorithm, separates the map into two regions. The region of potential update is the one containing points that pass through the vertices of L_n. Because of the discrete nature of the DDM, we note that the algorithm will have a choice of negative gradients leaving a vertex. The algorithm chooses the gradient that is closest to the "inside" of the region because points on the other gradient have alternative shortest paths that do not require going through the vertex.

Note that if multiple goals are present in the scene, then the negative of the gradient would not necessarily terminate at the edge of the distance map, but rather may reach a point where where the distance to a second goal was exactly equal to the distance to the goal around vertices of L_n. In this case, our algorithm would stop, and after the shadows of both vertices of L_n were determined, the algorithm would start at one vertex and follow the ridge (of points equal distance to at least 2 goals) to connect the two termination points. While there may be points outside this region whose minimal distance to a goal would change (in particular, points that above were called equal distant might now be closer to the goal which was obscured by L_n), it should be obvious that the minimal distance for such points would only decrease and so the modified CDT algorithm would correctly update these points as well.

This completes the basic step of the verification of the algorithm. We now move on to the induction step.

Induction step.

We assume the algorithm is correct for $n-1$ obstacles, and show that the algorithm is correct for a scene with n obstacles.

Let us assume that (at least) obstacle L_n has moved. As before, we assume that movement is handled by forming the disjunction of the potential update regions for the obstacle in its original and final position. Thus, without loss of generality, we examine the determination of the potential update region assuming that the distance to the vertices L_{n_1}, L_{n_2} has changed and show that the algorithm correctly determines the complete update region.

There are four possibilities to consider:

1. The shadows of L_{n_1} and L_{n_2} terminate at the edge of the DDM. In this case the region between the shadows of those lines, where the lines are connected by the edge of the DDM or by following a equi-distant contour, is the potential update region. (Note that this region may wholly contain other obstacles.)

2. The shadow of L_{n_1} (or L_{n_2}) intersect another obstacle, say L_j. In this case, by induction we can compute the update region for L_j. Obviously, the update region is simply the union of the update region for L_j and the area between that region and the shadow line of L_{n_2} (respectively L_{n_1}), see Figure 9. The shadow line and the update region of L_j are connected by either the edge of the DDM or an equi-distant contour.

3. The shadows of L_{n_1} and L_{n_2} intersect other obstacles, say L_i and L_j respectively. In this case, by induction we can compute the update region for L_i and L_j. Obviously, for all n obstacles, the update region is simply the union of the update regions associated with L_i and L_j and the area between them, see Figure 9.

4. The final possibility is that, in the multiple goal case, one or both of the shadow lines from L_{n_1} or L_{n_2} (as in case 1 and 2) terminate by reaching a point equi-distant from two or more goals. As before

the algorithm completes the region by following the ridge from the point of termination to either the other termination point or until the ridge intersects another part of the update region.

Unfortunately, this process does not insure that all points that should be updated will be included in the update region: there may be points beyond the ridge which are now closer to the goal than they previously were. Because the distance at those points is nonincreasing, the modified CDT algorithm insures, as we saw before, that the distance to all points will be correctly computed.

This completes the verification of our algorithm.

3.4 Complexity of the update algorithm

As should be apparent, the algorithm has two main phases, determination of the region of update, and the actual update. We discuss the complexity of each.

To discuss the complexity, we will need a few definitions. As before, we let v be the total number of obstacle vertices, m be the number of cells of the DDM in the region of interest, n be the number of neighbors of each cell, and ρ be the number of passes needed for the CDT to compute the DDM. For the motion of obstacles, let v' be the number of vertices on the obstacles that have moved, v'' be the number of vertices in the update region, m' be the number of cells in the potential update region, and ρ' be the number of passes needed for the CDT to compute the correct distance values for the potential update region. Clearly we have $v' \leq v'' \leq v, m' \leq m, \rho' \leq \rho$. Empirically we have found that $v' \approx v'' \ll v$, and $m' \ll m$. In addition, we let l be the maximum of the length (in steps) of the longest gradient path in the potential update region, and the length of the longest equi-distant contour. Obviously $l \leq m'$, and generally $l \leq \sqrt{m'}$.

The computation of the update requires following the negative of a gradient from a vertex until it terminates, and doing this for all the vertices of the objects that move. Because of object intersections, the number of gradient paths extended may approach v''. In addition, there is the connection of the termination points (again limited by v''). Thus determination of the region takes time $O(lnv'')$.

The application of the CDT to compute the new distance values is simply $O(\rho' m' n)$, which as shown in the section on the complexity of the CDT, is less than or equal to $O(v'' m')$.[††]

Thus the total cost of the update is $(Ov''n(m' + l))$. Obviously, the amount of saving of total recomputation of the DDM is dependent on the relative size of the update region as compared to the initial region of interest, on the number of vertices in the update region, and on the length of the paths in the update region. Note, that the savings are multiplicative, so that if we have a update region that is $\frac{1}{10}$ of the original image, and contains $\frac{1}{20}$ of the original vertices, the (worst case) savings is approximately factor of 200.

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.

[††] Unfortunately, we can not say $\rho < v'$. For example, a single line moving in front of the spiraling square in figure 3 would have $v' = 2$, but the computation of the modified distances would take $O(\frac{1}{2})$ passes.

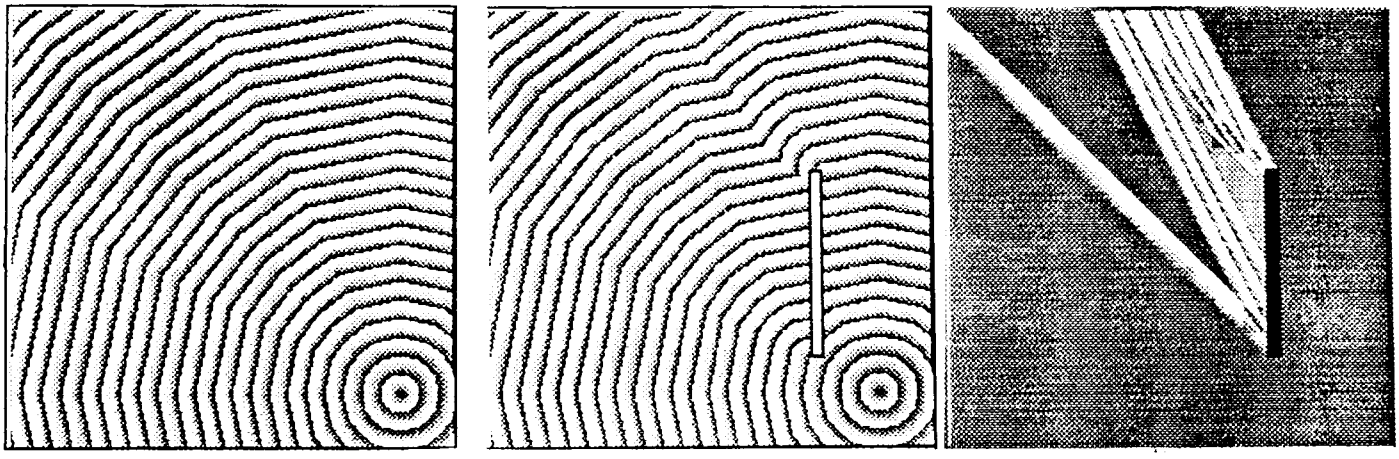

Figure 10: Example showing "shadow" regions for simple obstacles with a single goal. The left figure is the DDM with no obstacle, the middle the DDM with a simple obstacle, and the right image was obtained by computing the pointwise difference of the two images. (Distances and differences shown modulo 64)

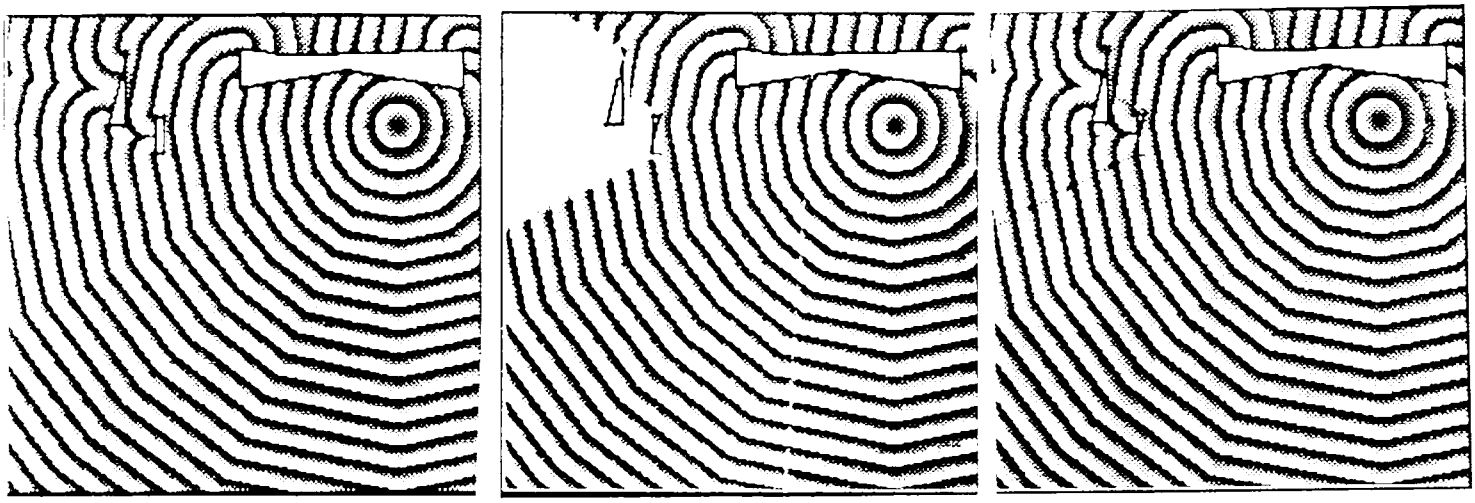

Figure 11: The results of "update" algorithm for a simple scene with 3 obstacles and one goal. The left is the DDM for the original obstacle configuration, the middle is the region of potential update (white region with thin black outline), and the right is the final result of the update.

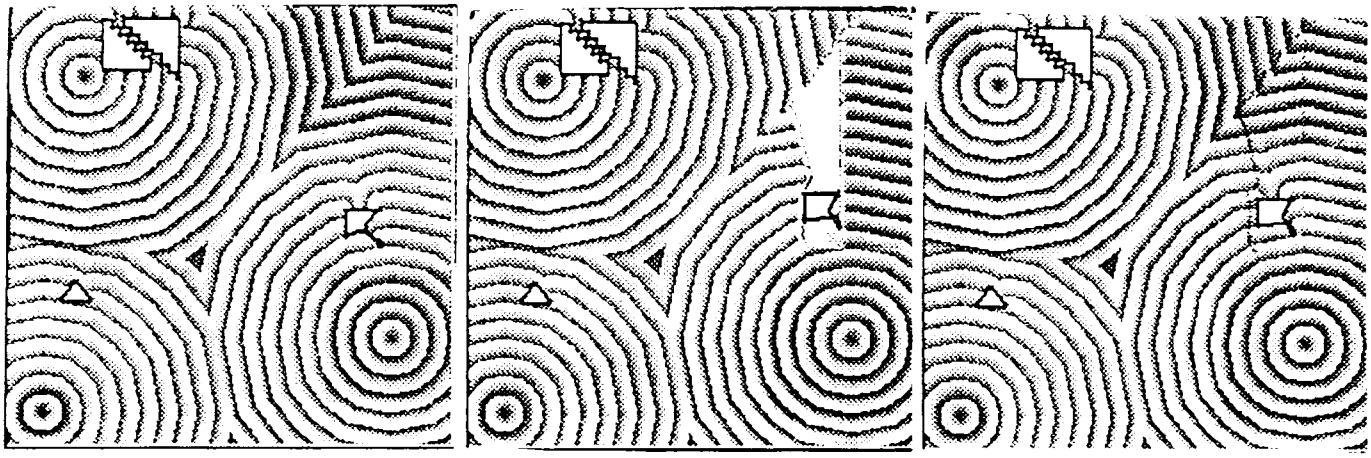

Figure 12: Example containing three goal points, and four obstacles, where obstacle movement was on a large scale. Also note the termination of region as it enters the area effected by a different goal node.

One approach to efficiency measures is to present run-times, but these are too machine/implementation dependent to be meaningful. Thus, we present two more abstract measures of the efficiency of the algorithm: 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.[‡‡] Note that even these relative measures have dubious meanings, since they depend on the relative portion of the image occupied by the obstacle(s) in motion. The real usefulness depends on what this proportion is for a typical scene in one's application area.

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 examples are presented distance encoded as grayvalues modulo 64, and black outlined obstacles. 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 obstacles and a single goal point, see Fig 11. Note how the update region intersects a secondary obstacle, and thus the obstacle is included in determining the potential update region. The calculation of the DDM for these obstacles takes 5 η. The small square in the lower left portion of the scene moves slightly down, to the left and compresses. The calculation of the update region and preparation of that region for the update requires .0019 η (preparation amounts to reinitizialiation of the area to a large distance value). The percentage of the total area occupied by the update region is 9.9%. The time required by the actual update in this area is .57 η. Thus the update algorithm was 8.7 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 12, and Fig 2, which has

[‡‡] 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. On our Sun-4 with a system load of 1.99, η was about 1 second (wall-clock time). Other researchers have reported running times of 2-3 seconds per pass, see [10] and [9].

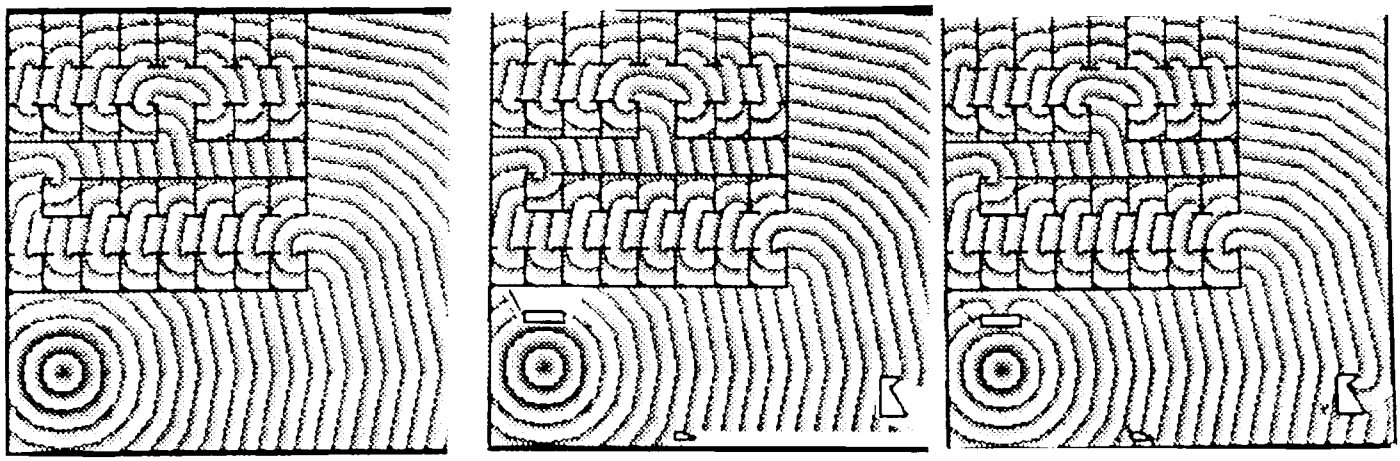

Figure 13: Example with a complex "room" scene. In this case, the motion consisted of adding 3 obstacles. Note that because the obstacle "shadow" of the obstacle nearest the goal falls entirely on one side of a "room wall", the wall does not effect the potential update region.

four goal points and 4 obstacles. The obstacle which moved, did so on a large scale and was also deformed as it moved. Note the update region (generated by the upper obstacle 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 obstacle 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 algorithm delivered approximately a 25 fold speedup.

The final example of the update algorithm is on the complex "room" scene in Fig 13. In this case there is no motion, but rather 3 obstacles are added to the scene. This example demonstrates the capability of the system to use pre-stored DDMs for a complex scene, and then update them if a few obstacles are added. This example also demonstrates the algorithm's ability not to generate update potentials for large obstacles when a small shadow falls on them. The calculation of the DDM for the original obstacle position (or the new position) obstacles takes 9 η. The calculation of the update region takes .0006 η, and the preparation for updating takes .0007 η. The percentage of the total area occupied by the update region is 4.1%. The cost of the update in this area is .2 η. Therefore the update algorithm delivered approximately a 45 fold speedup.

In another experimental setup, an obstacle was placed in order to increase the cost of every distance in the "room" portion of the example in Fig 13. This resulted in the entire room portion of the scene being added to the update region. Thus the update still required substantial computation time (9 passes over that region). However, the algorithm was able to determine the update region in only .01 η (remember many more obstacles will effect the area now). If the "robot" was currently moving outside the update region, the path planning/motion could continue as the DDM was updated in the background. This ability becomes increasingly important as the resolution of the DDM is increased.

5 Summary

This paper presented an efficient algorithm for the updating of digital maps when obstacles underwent motion, or are added to a precomputed DDM. The algorithm can intelligently deal with multiple goal points, arbitrary obstacle interaction, including not increasing the update region when a small obstacle moves in front of a large obstacle. 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. While the algorithm presented used a constrained distance transform to compute the DDM, any algorithm could have been used and the update algorithm would still be useful.

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 finer resolution or higher dimensional problem. The interested reader may consult [9] 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 [11], [12].

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. 10. Oddly, the pointwise difference between the distance in the penumbra region, and the distance in the same location without the associated obstacle present is constant. Similarly, the difference between a large part 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.

An important area for path planning is in configuration spaces. While the DDM ideas have been extended, the extension of the updating algorithm awaits further work. While many of the ideas extend (shadows etc.) there are a few nontrivial representational/ computational issues to be resolved to extend the idea into three dimensions and beyond.

The algorithm presented simply provides a means for updating the DDM after obstacle motion is known. However, if the DDM is being used to represent configuration space, the calculation of obstacles (i.e., configuration space obstacles) is highly non-trivial, and may actually dominate the cost of computing/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 [13]. Then if the rough obstacle has no potential to effect the current path, the detailed calculations can be done in the background.

Acknowledgments

This work was supported in part by Darpa grant #N00039-84-C-0165, the author was also supported by NSF grant IRI8800370. 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. We also thank the reviewers who provided useful criticisms of the ordinal version of the article.

References

[1] U. Montanari, "A method for obtaining skeletons using a quasi-eculidean distance," *J. ACM*, vol. 15, pp. 600–624, 1968.

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

[3] A. Rosenfeld and J. Pfaltz, "Sequential operations in digital picture processing," *J. ACM*, vol. 13, pp. 471–494, 1966.

[4] A. Rosenfeld and J. Pfaltz, "Distance functions on digital pictures," *Pattern Recognition*, vol. 1, no. 1, pp. 33–61, 1968.

[5] H. Barrow, J. Tenenbaum, R. Bolles, and H. Wolf, "Parametric correspondence and chamfer matching: two new techniques for image matching," in *Proceedings of the Fifth International Joint Conference on Arificial Intelligence*, (Cambridge, MA), pp. 659–663, 1977.

[6] P. Danielsson, "Eculedian distance mapping," *Computer Vision, Graphics, and Image Processing*, vol. 14, pp. 227–248, 1980.

[7] G. Borgefors, "Distance transforms in arbitrary dimensions," *Computer Vision, Graphics, and Image Processing*, vol. 27, pp. 321–345, 1984.

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

[9] P. Verbeek, L. Dorst, B. Verwer, and F. Groen, "Collision avoidance and path finding through constrained distance transformation in robot state space," in *Proceedings of the International Intelligent Autonomous Systems conference*, (L. Hertzberger and F. Groen, eds.), pp. 627–634, Amsterdam, Netherlands: North-Holland, December 1986.

[10] L. Dorst and P. Verbeek, "The constrained distance transformation: a pseudo-euclidean, recursive implementation of the lee-algorithm," in *SIGNAL PROCESSING III: Theories and Applications*, (I. Y. E. Al., ed.), Elsevier Science Publishers B.V. (North-Holland), 1986.

[11] T. Soetadji, "Cube based representation of free space for the navigation of an autonomous mobile robot," in *Proceedings of the International Intelligent Autonomous Systems conference*, (L. Hertzberger and F. Groen, eds.), (Amsterdam, Netherlands), pp. 546–561, North-Holland, December 1986.

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

[13] T. Lozano-Pérez, "A simple motion-planning algorithm for general robot manipulators," *IEEE Journal of Robotics and Automation*, vol. RA-3, pp. 224–238, June 1987.