2

DISTANCE TRANSFORM ALGORITHMS AND THEIR IMPLEMENTATION AND EVALUATION

George J. Grevera

Saint Joseph's University
Philadelphia, Pennsylvania, USA

Consider an n-dimensional binary image consisting of one or more objects. A value of 1 indicates a point within some object and a value of 0 indicates that that point is part of the background (i.e., is not part of any object). For every point in some object, a distance transform assigns a value indicating the distance from that point within the object to the nearest background point. Similarly for every point in the background, a distance transform assigns a value indicating the minimum distance from that background point to the nearest point in any object. By convention, positive values indicate points within some object and negative values indicate background points. A number of elegant and efficient distance transform algorithms have been proposed, with Danielsson being one of the earliest in 1980 and Borgefors in 1986 being a notable yet simple improvement. In 2004 Grevera proposed a further improvement of this family of distance transform algorithms that maintains their elegance but increases accuracy and extends them to n-dimensional space as well. In this paper, we describe this family of algorithms and compare and contrast them with other distance transform algorithms. We also present a novel framework for evaluating distance transform algorithms and discuss applications of distance transforms to other areas of image processing and analysis such as interpolation and skeletonization.

1. INTRODUCTION

Consider a binary image, I, consisting of one or more objects. Since this is a binary image, each point is either within the bounds of some object (interior point) or is part of the background and is not part of any object (exterior point).

George J. Grevera, BL 215, Mathematics and Computer Science, 5600 City Avenue, Saint Joseph's University, Philadelphia, PA 19131, USA. Phone: (610) 660-1535; Fax: (610) 660-3082. ggrevera@sju.edu.

We note that some points within objects are noteworthy in that they are positioned on the border of the object with at least one of the outside (background) points. Adopting terminology from digital topology [1], we call these points elements of the set of points that form the immediate interior (II) of some object. Similarly, some background points are notable in that they are positioned on the border or interface of some object as well. Once again adopting terminology from digital topology, we call these background points elements of the set of points that form the immediate exterior (IE). Together, the union of the sets II and IE form a set of points called border points (or boundary elements), B. We may now define a distance transform as an algorithm that given I produces a transformed image, I', by assigning to each point in I the minimum distance from that point to all border points.

A number of issues arise when dealing with distance transform algorithms. The first and probably the most important issue is one of accuracy. Does the distance transform produce results with minimal errors (or is the distance transform error free)? If that is the case, it is important to develop a methodology to verify this claim (and we will do so in this chapter). Even for those algorithms that are theoretically proven to be error free, from a software engineering standpoint it is important to be able to validate the implementation of the algorithm. A second important issue concerns the computation time required by the method. It is relatively straightforward to develop an exhaustive method that requires a great deal of processing time. It is important to evaluate processing time as well. And yet another issue is with regard to the dimensionality of I. Since medical imagery is of three or four dimensions, it is important in the medical arena for a distance transform algorithm to generalize to dimensions higher than two. Another issue that arises when dealing with medical images is that of anisotropic sampling. Medical images are typically acquired as three-dimensional volumes of data (stacks of slices) with the sampling within each plane or slice at a higher rate than the sampling across slices. This yields data with finer spacing between neighboring pixels (or more generally, voxels) within a slice (e.g., 0.5 mm) than between neighboring pixels between slices (e.g., 1.0 mm).

Distance transform algorithms are, like most other computationally intensive algorithms, of interest in and by themselves and have been the subject of at least one PhD dissertation [2]. Many distance transform algorithms have been proposed, with [3] and [4] most likely being the earliest. In general, distance transform algorithms exhibit varying degrees of accuracy of the result, computational complexity, hardware requirements (such as parallel processors), and conceptual complexity of the algorithms themselves. In [5], the author proposed an algorithm that produces extremely accurate results by propagating vectors that approximate the distance in 2D images by sweeping through the data a number of times by propagating a local mask in a manner similar to convolution. In [6] the author presented the Chamfer distance algorithm (CDA), which propagates scalar, integer values to efficiently and accurately calculate the distance transform of 2D and 3D images (again in a

manner similar to convolution). Borgefors [6] also presented an error analysis for the CDA for various neighborhood sizes and integer values. More recently in [7] an analysis of 3D distance transforms employing 3x3x3 neighborhoods of local distances was presented. In [8] an analysis of the 2D Chamfer distance algorithm using 3x3, 5x5, and larger neighborhoods employing both integer and real values was presented. Marchand-Maillet and Sharaiha [9] also present an analysis of Chamfer distance using topological order as opposed to the approximation to the Euclidean distance as the evaluation criteria. Because of the conceptual elegance of the CDA and because of its widespread popularity, we feel that the CDA family of algorithms is important and worthy of further study.

Of course, distance transforms outside of the Chamfer family also have been presented. A technique from Artificial Intelligence, namely $A*$ heuristic search [10], has been used as the basis for a distance transform algorithm [11]. A multiple-pass algorithm using windows of various configurations (along the lines of [5] and other raster scanning algorithms such as the CDA) was presented in [12] and [13]. A method of distance assignment called ordered propagation was presented in [14]. The basis of that algorithm and others such as $A*$ (used in [11]) is to propagate distance between pixels, which can be represented as nodes in a graph. These algorithms typically employ sorted lists to order the propagation among the graph nodes. Guan and Ma [15] and Eggers [16] employ lists as well. In [17] the authors present four algorithms to perform the exact, Euclidean, n-dimensional distance transform via the serial composition of n-dimensional filters. Algorithms for the efficient computation of distance transforms using parallel architectures are presented in [18] and [19]. In [19] the authors present an algorithm that consists of two phases, with each phase consisting of both a forward scan and a backward scan. In the first phase columns are scanned; in the second phase rows are scanned. They note that since the scanning of a particular column (or row) is independent of the scanning of the other columns (or rows), each column (row) may be scanned independently (i.e., in parallel). A distance transform employing a graph search algorithm is also presented in [20].

Since the early formulation of distance transform algorithms [3, 4], applications employing distance transforms have also become widespread. For example, distance transforms have been used for skeletonization of images [21, 22, 23, 24]. Distance transforms are also useful for the (shape-based) interpolation of both binary images [25, 26] as well as gray image data [27]. In [28] the authors employ distance transform information in multidimensional image registration. An efficient ray tracing algorithm also employs distance transform information [29]. Distance transforms have also been shown to be useful in calculating the medial axis transform, with [30, 31] employing the Chamfer distance algorithm specifically. In addition to the usefulness of distance transforms for the interpolation of 3D gray medical image data [32, 33], they have also been used for the automatic classification of plant cells [34] and for measuring cell walls [35]. The Chamfer distance was also employed in a method to characterize spinal cord atrophy [36].

Because distance transforms are applicable to such a wide variety of problems, it is important to develop accurate and efficient distance transform algorithms.

2. DISTANCE TRANSFORM ALGORITHMS

Before we begin our discussion of algorithms in detail, we note that our descriptions will be limited to the two-dimensional case (i.e., where the input image, I, is two dimensional). Some algorithms readily generalize (or have been generalized) to higher dimensions. We will endeavor to point out those more general algorithms. Furthermore, all of the algorithms that will be described do not require any special purpose hardware such as parallel processors.

All of the distance transform algorithms that will be described can be said to rely on the determination of border points, so we begin with a method to determine them. Recall that we define the border points, B, as the union of the sets *II* and *IE*. To determine the set of border points, we must first determine these sets. A point $p = (x, y)$ is an element of an object iff $I(p)$ = 1. Similarly, a point $q = (x, y)$ is an element of the background iff $I(q) = 0$. But not all object points are elements of *II* (nor are all background points elements of *IE*). Only those object points that are on the border of an object are elements of *II* (and similarly for *IE*). To determine if an object point, $p = (x, y)$, is an element of *II*, we consider the neighborhood of p to be the set of all points $N(p) = \{(x + dx, y + dy) \mid -1 <= dx <= 1$ and $-1 <= dy <= 1\}$. In practice, we typically restrict the definition of $N(p)$ to include only those nearby elements with the same x or y coordinates as p (the so-called 4-adjacency (connectedness or connectivity) versus the less restrictive 8-adjacency) as follows: $N(p) = \{(x + dx, y + dy)| - 1 <= dx <= 1$ and $-1 <= dy <= 1$ and $|dx + dy| = 1\}$. If there exists at least one point q in $N(p)$ such that q is an element of the background, then p is an element of *II*. Similarly, to determine if a background point, $q = (x, y)$, is an element of *IE*, we consider the neighborhood of q. If there exists at least one point p in $N(q)$ such that p is an element of an object, then q is an element of the *IE*. The algorithm for this determination follows:

```
for (y=1; y<ySize-1; y++)
   for (x=1; x<xSize-1; x++)
        if ( I(x-1,y) != I(x,y) or I(x+1,y)] != I(x,y) or
             I(x,y-1) != I(x,y) or I(x,y+1)  != I(x,y))
            then (x,y) is a 4-adjacent border element.
        if ( I(x-1,y-1) != I(x,y) or I(x+1,y-1)] !=I(x,y) or
             I(x-1,y+1) != I(x,y) or I(x+1,y+1)  !=I(x,y) )
           then (x,y) is a remaining 8-adjacent border element.
```

where xSize is the number of columns in I and I', and ySize is the number of rows. We note that some distance transform algorithms including [5] restrict the definition of border points to elements of *II* only. Our framework easily accommodates this

via a simple change to the algorithm above as illustrated below. We point out, however, that this definition will not preserve the property of symmetry under complement [37]. Consider the complement C of the binary image I such that $C(p) = 1$ if $I(p) = 0$ and $C(p) = 0$ otherwise. A distance transform that preserves symmetry under complement produces the same result given either C or I (i.e., $C'(p) = I'(p)$ for all p, although the sign may be opposite by convention. In that case, $|C'(p)| = |I'(p)|$.

```
for (y=1; y<ySize-1; y++)
    for (x=1; x<xSize-1; x++)
        if (I(x,y)==1)    //restrict border points to II only
            if ( I(x-1,y) != I(x,y) or I(x+1,y)] !=I(x,y) or
                 I(x,y-1) != I(x,y) or I(x,y+1)  !=I(x,y) )
                then (x,y) is a 4-adjacent border element.
            if ( I(x-1,y-1) != I(x,y) or I(x+1,y-1)] != I(x,y) or
                 I(x-1,y+1) != I(x,y) or I(x+1,y+1) != I(x,y) )
                then (x,y) is a remaining 8-adjacent border element.
```

Note that, without loss of generality, we assume that no object extends to the edge of the discrete matrix in which it is represented. Otherwise, the description of the algorithms would be unnecessarily complicated by additional boundary condition checks. If it is the case that an object extends to the edge of the matrix, one may simply embed that matrix and the objects that are represented within it in a larger matrix with an additional layer of surrounding background elements.

2.1. A Simple Distance Transform Algorithm (Simple)

Arguably the simplest distance transform follows. First, we assign each border element a distance value of 0: $I'(s) = 0$, where s is in B. Then for each t not in B, we assign $I'(t) = \min\{d(s,t) | s \text{ in } B \text{ and } t \text{ not in } B\}$, where $d(s,t)$ is the Euclidean distance from s to t. This algorithm is very simple, both conceptually and computationally. It is also error free. Furthermore, it is also very easy to extend this algorithm to higher dimensions as well as to anisotropic data. Unfortunately, if for each t we must search I to determine every s in B, we have an algorithm that is the least computationally efficient of those that will be discussed. We subsequently refer to this algorithm as Simple. Pseudo code for this algorithm follows.

```
//iterate over all (non border element) points
 for (y=1; y<ySize-1; y++) {
     for (x=1; x<xSize-1; x++) {
         if (I'(x,y)!=0) {  //only consider non border elements
             //t=(x,y)
             //now iterate over all border elements
             for (y1=0; y1<ySize; y1++) {
                 for (x1=0; x1<xSize; x1++) {
```

```
                    if (I'(x1,y1)==0) {
                        //s=(x',y')
                //calculate the distance to this border element
                        d = sqrt( (x-x1)*(x-x1) + (y-y1)*(y-y1) );
                //is it better than what's already been assigned?
                        if (d < I'(x,y)) {
                //yes, then update the distance from this
                //point, t, to the border element, s
                            I'(x,y) = d;
                        }
                    }  //end if
                  }  //end for x1
              }  //end for y1
          }  //end if
      }  //end for x
  }  //end for y
```

2.2. A Simple Distance Transform Algorithm Employing a List (SimpleList)

A straightforward modification to the simple algorithm yields a surprisingly effective method. Instead of exhaustively and repeatedly searching I to determine every s in B for every t not in B, we employ an additional data structure, a list (actually using the vector class which can be indexed as implemented in the generics in the standard C++ library) to represent B. The C++ vector class is used because it may be indexed like an array, but unlike an array it can grow in size dynamically. One does not need to know a priori the number of points to allocate for the array. Then for each t not in B, we search the list $L = B$ and assign $I'(t) = \min\{d(s,t)|s \text{ in } L \text{ and } t \text{ not in } B\}$. Like the simple algorithm that does not employ a list, this algorithm is also very simple, both conceptually and computationally. It too is also error free. Furthermore, it is also very easy to extend this algorithm to higher dimensions as well as to anisotropic data. Unlike the simple algorithm that does not employ a list, this algorithm is very efficient when the size of the list is small. We subsequently refer to this algorithm as SimpleList. Pseudo code for this algorithm follows.

```
//iterate over all (non border element) points
for (y=1; y<ySize-1; y++) {
    for (x=1; x<xSize-1; x++) {
        if (I'(x,y)!=0) {  //only consider non border elements
        //at this stage, we have a point that is not an element of
        //the border. iterate over all border elements in the list.
            for (i=0; i<list.size(); i++) {
                x1 = list[i]->x;
```

```
                y1 = list[i]->y;
                //calculate the distance to this border element
                d = sqrt( (x-x1)*(x-x1) + (y-y1)*(y-y1));
                //is it better than what's already been assigned?
                if (d < I'(x,y)) {
                    //yes, then change to this border element
                    I'(x,y) = d;
                }
            }
        }  //end if
    }  //end for x
}  //end for y
```

This method is also important for testing other methods as it will form the basis of our testing procedure.

2.3. Danielsson's [5] 4SED and 8SED Algorithms (and Grevera's Improved 8SED Algorithm)

Although not completely error free, Danielsson's distance transform algorithms were early contributions and are important steps in the development of subsequent algorithms such as the Chamfer distance and Dead Reckoning. 8SED is efficient, reasonably accurate, conceptually easy to understand, and is still in widespread use today. These algorithms begin by initially assigning a distance value of 0 for all p in B and a value of infinity for all p not in B. Then the algorithms sweep through I' using a number of passes and various local "windows" in a manner somewhat similar to convolution [38] from digital signal processing. The current distance assignment to each point under consideration, u, is compared to the current assignments to its neighbors plus the distance, a, from the specific neighbor, n, to u. If the current distance assignment, $I'(u)$, is greater than $I'(n) + a$, then $I'(u)$ is updated to $I'(n) + a$, which results in minimizing the distance to u. The difference between 4SED, 8SED, and Grevera's improved 8SED algorithms are in the number of sweeps and in the neighborhood windows that are checked during the minimization process. 4SED is the least accurate but is the fastest. Grevera's improved 8SED produces more accurate results at the expense of increased processing time although the increase in time is not significant. Pseudo code for the 4SED algorithm follows.

```
//perform the first pass ("first picture scan")
for (y=1; y<=ySize-1; y++) {
    for (x=0; x<=xSize-1; x++)  check( x,   y-1, dy  );
    for (x=1; x<=xSize-1; x++)  check( x-1, y,   dx  );
    for (x=xSize-2; x>=0; x--)  check( x+1, y,   dx  );
}
```

```
//perform the final pass ("second picture scan")
for (y=ySize-2; y>=0; y--) {
    for (x=0; x<=xSize-1; x++)  check( x,   y+1, dy  );
    for (x=1; x<=xSize-1; x++)  check( x-1, y,   dx  );
    for (x=xSize-2; x>=0; x--)  check( x+1, y,   dx  );
}
```

where *check* compares, and updates if necessary, the current distance $I'(u)$, $u = (x, y)$ to the specified neighbor and offset from neighbor to u, $I'(n) + a$. Pseudo code for the 8SED algorithm follows.

```
//perform the first pass ("first picture scan")
for (y=1; y<=ySize-1; y++) {
    for (x=0; x<=xSize-1; x++) {
        if (x>0) {  //** boundary condition not checked in original
                    //   but needed
            check( x-1, y-1, dxy );
        }
        check( x,   y-1, dy  );
        if (x<xSize-1) {  //** not checked in original but needed
            check( x+1, y-1, dxy );
        }
    }
    for (x=1; x<=xSize-1; x++)  check( x-1, y,   dx  );
    for (x=xSize-2; x>=0; x--)  check( x+1, y,   dx  );
}

//perform the final pass ("second picture scan")
for (y=ySize-2; y>=0; y--) {
    for (x=0; x<=xSize-1; x++) {
        if (x>0) {  //** not checked in original but needed
            check( x-1, y+1, dxy );
        }
        check( x,   y+1, dy  );
        if (x<xSize-1) {  //** not checked in original but needed
            check( x+1, y+1, dxy );
        }
    }
    for (x=1; x<=xSize-1; x++)  check( x-1, y, dx );
    for (x=xSize-2; x>=0; x--)  check( x+1, y, dx );
}
```

Pseudo code for Grevera's improved 8SED algorithm follows. Note: * indicates a difference from the original 8SED algorithm.

```
//perform the first pass ("first picture scan").
for (y=1; y<ySize-1; y++) {
    for (x=0; x<=xSize-1; x++) {  //* from 4sed
        check( x,   y-1, dy  );
    }
    for (x=1; x<=xSize-1; x++) {
        check( x-1, y,   dx  );
        check( x-1, y-1, dxy );   //*
    }
    for (x=xSize-2; x>=0; x--) {
        check( x+1, y,   dx  );
        check( x+1, y-1, dxy );   //*
    }
}

//perform the final pass ("second picture scan")
for (y=ySize-2; y>=0; y--) {
for (x=0; x<=xSize-1; x++) {  //* from 4sed
        check( x,   y+1, dy  );
    }
    for (x=1; x<=xSize-1; x++) {
        check( x-1, y,   dx  );
        check( x-1, y+1, dxy );   //*
    }
    for (x=xSize-2; x>=0; x--) {
        check( x+1, y,   dx  );
        check( x+1, y+1, dxy );   //*
    }
}
```

2.4. Borgefors' [6] CDA (Including Chessboard, Cityblock, and Euclidean 3x3 Window)

Borgefors' Chamfer distance algorithm (CDA) is arguably the most popular distance transform method. Like Danielsson's algorithms, it is not completely error free, but it is efficient, reasonably accurate, and conceptually even easier to understand than Danielsson's algorithms. Furthermore, it has also been extended from 2D to 3D [7, 40], and simple modifications produce other distance transforms such as chessboard, cityblock, and Euclidean with a 3x3 window. Like Danielsson's algorithm, the CDA (including chessboard, cityblock, and Euclidean) begins by initially assigning a distance value of 0 for all p in B and a value of infinity for all q not in B. Then the CDA sweeps through using two passes. The first pass is from top to bottom and left to right and the second pass is from bottom to top and right to left. Again, various local "windows" are used in a manner similar to convolution

[38] from digital signal processing using 3x3 windows for CDA 3x3, chessboard, cityblock, and Euclidean, 5x5 windows for CDA 5x5, and 7x7 windows for CDA 7x7. The current distance assignment to each point under consideration, u, is compared to the current assignments to its neighbors plus the distance, a_n, for that specific neighbor n taken from Figure 1 from the specific neighbor, n, to u. If the current distance assignment, $I'(u)$, is greater than $I'(n)+a_n$, then $I'(u)$ is updated to $I'(n) + a_n$, which results in minimizing the distance to u. Resulting distance transform errors diminish with increasing window size while computation cost increases with increasing window size. Note that different window configurations (but of the same size) are employed for the forward and backward passes.

Pseudo code for CDA 3x3, cityblock, chessboard, and Euclidean 3x3 appears below. *dx*, *dy*, and *dxy* are assigned a_n values according to the table entries in Figure 1 corresponding to the desired method. CDA 5x5 and CDA 7x7 are analogous.

```
//perform the first (forward) pass
for (y=1; y<ySize-1; y++) {
    for (x=1; x<xSize-1; x++) {
        check( x-1, y-1, dxy );
        check( x,   y-1, dy  );
        check( x+1, y-1, dxy );
        check( x-1, y,   dx  );
    }
}
//perform the final (backward) pass
for (y=ySize-2; y>=1; y--) {
    for (x=xSize-2; x>=1; x--) {
        check( x+1, y,   dx  );
        check( x-1, y+1, dxy );
        check( x,   y+1, dy  );
        check( x+1, y+1, dxy );
    }
}
```

where *check* compares, and updates if necessary, the current distance $I'(u)$, $u = (x, y)$ to the specified neighbor and offset from neighbor to u, $I'(n) + a_n$.

Borgefors cleverly demonstrated: (i) using a small window and propagating distance in this manner introduces errors in the assigned distance values even if double precision floating point is used to represent distance values, (ii) these errors may be minimized by using values other than 1 and $\sqrt{2}$ for the distances between neighboring pixels, and, surprisingly, (iii) using integer window values such as 3 and 4 yields more accurate results than using window values of 1 and $\sqrt{2}$ and does

	forward pass	backward pass
CDA 3×3	4 3 4 3 u - - - -	- - - - u 3 4 3 4
city block	- 1 - 1 u - - - -	- - - - u 1 - 1 -
chessboard	1 1 1 1 u - - - -	- - - - u 1 1 1 1
CDA 5x5	- 11 - 11 - 11 7 5 7 11 - 5 u - - - - - - - - - - - -	- - - - - - - - - - - - u 5 - 11 7 5 7 11 - 11 - 11 -
CDA 7×7	- 43 38 - 38 43 - 43 - 27 - 27 - 43 38 27 17 12 17 27 38 - - 12 u -	- u 12 - - 38 27 17 12 17 27 38 43 - 27 - 27 - 43 - 43 38 - 38 43 -
Euclidean 3x3	$\sqrt{2}$ 1 $\sqrt{2}$ 1 u - - - -	- - - - u 1 $\sqrt{2}$ 1 $\sqrt{2}$

Figure 1. Various windows used by the Chamfer distance algorithm. 'u' indicates the center of the window. '-' indicates that the point is not used (considered) during that pass of the algorithm.

so with much better performance (when implemented using integer arithmetic), and (iv) larger windows with appropriate values minimize errors even further at increased computational cost.

2.5. Grevera's [37] Dead Reckoning Algorithm

The Dead Reckoning Algorithm (DRA) is a straightforward modification to the CDA that, employing equal-sized windows, produces more accurate results at a slightly increased computational cost. Furthermore, it has been demonstrated [37] that DRA using only a 3x3 window typically produces more accurate results than CDA with a 7x7 window with similar execution times.

In addition to I', which for a given point, (x, y), is the minimum distance from (x, y) to the nearest border point, the DRA introduces an additional data structure, $P(x, y) = (x', y')$, which is used to indicate the actual border point (x', y') in B such that $I'(x, y)$ is minimum. This is similar to the method employed by Danielsson [5], where 4SED employs three minimization iterations in both the forward and backward passes. Our method as in the CDA employs only one iteration in each pass. Note that as the CDA progresses, $I'(x, y)$ may be updated many times. In the DRA, each time that $I'(x, y)$ is updated, $P(x, y)$ is updated as well. We note that the order in which the 'if' statements in the pseudo code for this algorithm are evaluated may influence the assignment of $P(x, y)$ and subsequently, the value assigned to $I'(x, y)$. Regardless, our results demonstrate that our algorithm remains more accurate using only a 3x3 neighborhood than CDA using a 7x7 neighborhood. Although the DRA employs a 3x3 (or larger) window to guide the update/minimization of distance process as does CDA, the actual values assigned to I' are not the same as CDA. DRA uses instead the actual Euclidean distance from the border to the point (x, y) at the center of the window. Using only a 3x3 window, the DRA typically determines a more accurate estimation of the exact Euclidean distance within the framework of the CDA. Pseudo code for the DRA is the same as CDA except for a modification to *check*. *check* compares, and updates if necessary, the current distance $I'(u)$, $u = (x, y)$ to the specified neighbor and offset from neighbor to u, $I'(n) + a_n$ as before in the CDA but if $I'(u) >= I'(n) + a_n$, $I'(u)$ is not assigned $I'(n) + a_n$ but is assigned distance from u to $P(n)$ and $\underline{P}(u) = P(n)$.

2.6. Dijkstra's Graph Algorithm

Dijkstra's shortest path algorithm [39] for determining minimum cost paths in graphs can be adapted to distance transform algorithms as well. To accomplish this we simply map the input binary image, I, to a graph, $G = (V, E)$, where V is the set of vertices (the set of discrete (x, y) locations in I and E is the set of edges defined as follows. Consider some point, $p = (x, y)$, in V and the (8-connected) neighborhood $N(p) = \{(x-1, y), (x+1, y), (x, y-1), (x, y+1), (x-1, y-1), (x+1, y-1), (x-1, y+1), (x+1, y+1)\} = \{(x+dx, y+dy) | -1 <= dx <= 1$

and $-1 <= dy <= 1$ and $|dx + dy| <= 2\}$. E consists of all edges from points p to each of its neighbors and define the cost (distance) associated as either 1 or $\sqrt{2}$ depending on the Euclidean distance from p to the particular neighbor. As in previous algorithms, this method begins as many of the previous ones with initially assigning a distance value of 0 for all p in B and a value of infinity for all p not in B. Those points p for which $I'(p) = 0$ are also initially placed on an ordered list L that is sorted from smallest to largest according to the distance assignment, $I'(p)$. The algorithm then proceeds as follows:

```
while (L is not empty) {
    remove from L, p such that I'(p) is minimal
      among all elements of L;
    consider each neighbor n in N(p);
    if (I'(p)+d(p,n) < I'(n)) {
        I'(n) = I'(p)+d(p,n);
        put n in L according to I'(n);
    }
}
```

where $d(p, n)$ is the Euclidean distance from p to n. Because of the regularly discretized nature of I, $d(p, n)$ is either 1 or $\sqrt{2}$ since n is in $N(p)$. When this algorithm terminates, $I'(p)$ will contain the minimal distance from p to an element of B in terms of the minimal summation of edge costs associated with a path from any element of B to p. In that respect, this method is similar to DRA. Subsequently, we will refer to Dijkstra's algorithm adapted and applied to the distance transform problem as ModifiedDijkstra (MD).

We will now describe a variant of the MD algorithm. Recall that DRA is the same as CDA except for a modification to the *check* procedure where *check* compares the current distance assignment $I'(u)$, $u = (x, y)$ to the specified neighbor and offset from neighbor to u, $I'(n) + a_n$ as before, but if $I'(u) >= I'(n) + a_n$, $I'(u)$ is not assigned $I'(n) + a_n$ but is assigned the distance from u to $P(n)$. We can also modify MD to perform in this manner. We call this method ModifiedDijkstraDeadReckoning (MDDR) using 8-connected neighborhoods.

None of these algorithms (based on Dijkstra's graph algorithm) are error free. To develop an error-free graph-based algorithm, we note that Dijkstra's algorithm determines cost (distance) as a discrete sequence of edges in the graph between two vertices. Since we are using this discrete space as a model of an underlying continuous space, small errors may be introduced. Therefore, the first time that a vertex is encountered may not be the optimal distance assignment for that point. We must allow for a vertex to be revisited (as in $A*$ heuristic search [10]) by maintaining a list (vector) of border element assignments (instead of a single, first assignment). We call this method DijkstraVectors (DV), and experimental results have shown this algorithm to be error free.

2.7. Ragnemalm's [14] CSED Algorithm

Ragnemalm's CSED algorithm is similar to Dijkstra's algorithm except that instead of employing a single ordered list, L, it avoids maintaining L in sorted order by using two lists, $L1$ and $L2$, and a limit or threshold, l , on the current p in $L1$ such that $I'(p) < l$. In this manner, propagation of distance values is ordered along approximate isocontours by repeatedly sweeping through $L1$ by examining each p in $L1$ in turn. If the current $I'(p) < l$, then we consider the neighborhood of p. Otherwise, we move p from $L1$ to $L2$ for future consideration. Initially, all elements of $N(b)$, b in B, are placed on $L1$. Similar to the DRA, which introduced an additional data structure, $P(x, y) = (x', y')$, which is used to indicate the actual border point (x', y') in B such that $I'(x, y)$ is minimum, CSED uses an additional data structure $\underline{P}(x, y) = (d\underline{x}, d\underline{y})$ such that $(x', y') = (x, y) + (d\underline{x}, d\underline{y})$. ($P(x, y)$ is an element of B where $\underline{P}(x, y)$ is the displacement from (x, y) to an element of B). Additionally, we will refer to the x component of p in $\underline{P}$ as $\underline{p}x$ and similarly for y. Ragnemalm's algorithm also performs a more intelligent propagation among the 8-connected neighbors than Dijkstra's algorithm as well.

Pseudo code for this algorithm follows. Like the algorithms based on Dijkstra's algorithm (except for DV), CSED is not error free.

```
for all p {
    if I'(p)=0 then
        consider each neighbor n in N(p)
        check( p,  0,  1 );
        check( p,  1,  1 );
        check( p,  1,  0 );
        check( p,  1, -1 );
        check( p,  0, -1 );
        check( p, -1, -1 );
        check( p, -1,  0 );
        check( p, -1,  1 );
}
swap( l1, l2 );
l = 1;
while L1 is not empty
    for each p in L1
        remove p from L1
        if I'(p) > l then
            put p on L2;
        else if px(p)=0 then
            check( p, 0,sgn(py(p)) );
                                          //vertical
        else if py(p)=0 then
            check( p, sgn(px(p)), 0 );
```

```
                                        //horizontal
        else if |px(p)| = |py(p)| then
            check( p, sgn(px(p)), sgn(py(p)) );
                                        //diagonal
            check( p, sgn(px(p)), 0          );
                                        //horizontal
            check( p, 0,          sgn(py(p)) );
                                        //vertical
        else if |Px(p)| > |Py(p)| then
            check( p, sgn(px(p)), sgn(py(p)) );
                                        //diagonal
            check( p, sgn(px(p)), 0          );
                                        //horizontal
        else
            check( p, sgn(px(p)), sgn(py(p)) );
                                        //diagonal
            check( p, 0,          sgn(py(p)) );
                                        //vertical
    L1 = L2;
```

where $sgn(k) = -1$ if $k < 0$, 0 if $k = 0$, and 1 if $k > 0$ and *check* is defined as follows:

```
check( p, dx, dy )
    let n = (px+dx, py+dy);
    let d = sqrt( dx*dx + dy*dy );
    if (I'(n) > I'(p)+d) {
        I'(n) = I'(p)+d;
        put n in L2;
    }
```

Ragnemalm also presents an error free version of the CSED algorithm. Unfortunately, our implementation of that algorithm, which we believe is faithful to the description in their paper, allows a few points to remain initialized at infinity in our tests. This severely skews the results. Therefore, the software that accompanies this article includes our implementation of the error free version but the results of executing that implementation will not be included in this paper.

3. EVALUATING DISTANCE TRANSFORM ALGORITHMS

Distance transforms may be evaluated according to a variety of criteria. As mentioned previously, the accuracy of the result is arguably the most important measure. Even algorithms that purport to be error free should be evaluated to ensure that the implementation is indeed error free. To evaluate accuracy we

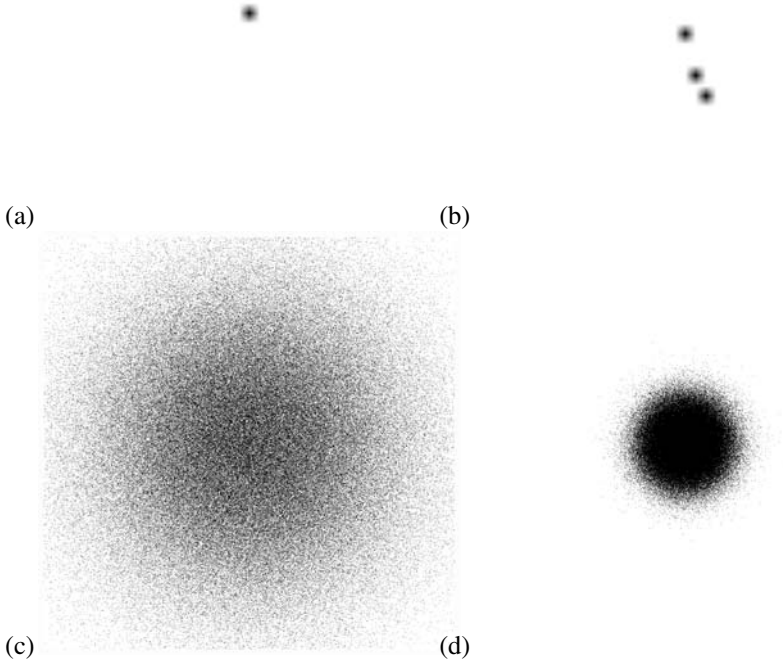

Figure 2. Sample test images consisting of (a) a single, solitary point-object, (b) a configuration of three single point-objects that is a known problematic configuration, (c) and (d) randomly generated test images by sampling from a normal distribution (with different standard deviations).

need to (i) choose a suite of test cases (input binary images), (ii) develop a "gold standard" or "ground truth" for each of the test cases, (iii) choose a set of metrics to compare the result of a distance transform method with ground truth, and then (iv) compare the results of a method under test with the gold standard using the metrics.

The simplest test case consists of an image that contains a solitary object consisting of a single point at the center of the image as shown in Figure 2a. Another test case has been described [2] as being extremely problematic for algorithms that sweep through the data using local windows (such as 4SED, 8SED, CDA,

DRA, and others). It consists of the three single point-objects as shown in Figure 2b. Although input images consisting of a few solitary point-objects are useful for understanding algorithms, they are not reflective of real-world objects. To simulate real-world objects, we also include images consisting of randomly generated objects by sampling from a normal distribution, as shown in Figures 2c and 2d.

With regard to a gold standard, we chose the SimpleList algorithm because it is straightforward, easy to verify, and is exhaustive in its determination of the correct distance assignments. The Simple algorithm could be used instead of SimpleList but in practice, Simple is too slow to be useful. (For example, for a rather small image of 300x300 consisting of a single center point object, SimpleList required 0.03 s of CPU time on a 2-GHz Pentium 4 under Linux. Simple required 71.98 s. The remaining methods required less then 1 s.)

The result of the distance transform, I', may be regarded as a grey image where the grey value at each location is the distance value assigned by the particular algorithm. Given I', the result of some distance transform algorithm, and $I'_{\mathrm{SimpleList}}$ (the result of applying SimpleList to I), we can compute the magnitude of the differences between I' and $I'_{\mathrm{SimpleList}}$ and determine the RMS (root mean squared) error as well as the location of the (magnitude of the) single largest difference between I' and $I'_{\mathrm{SimpleList}}$. Additionally, we also calculate the number of pixels that exhibit any difference whatsoever (regardless of the magnitude of the difference) and express this as a percentage of the whole. More qualitative insights can also be gained by viewing difference images ($|I' - I'_{\mathrm{SimpleList}}|$) or by simply thresholding I' to create a binary image and viewing the result as shown in Figure 7 as applied to the input binary image consisting of a single center point. The expected thresholded result should appear as a circular isocontour with radius equal to the distance from the center point. Which isocontour is observed depends upon the selected threshold value. Note that the thresholded results of CDA 3x3, CDA 5x5, Chessboard, Cityblock, Euclidean 3x3, and MD exhibit significant visible errors in the form of polygonal approximations to the circular isocontour. The more accurate of these methods exhibit polygons with more sides (while the least have less sides). Chessboard has only four sides, while the thresholded results of CDA 3x3, Cityblock, Euclidean 3x3, and MD have eight sides. Careful examination of the thresholded results of CDA 5x5 and CDA 7x7 yields 16- and 20-sided polygons for the selected threshold, respectively. The remaining, most accurate methods do not have any noticeable artifacts. In addition to accuracy, it is also important to report the CPU time required to perform the distance transform as well.

4. RESULTS OF EVALUATION

All experiments were performed on a Dell 3.6-GHz Pentium 4 system with 2 GB of RAM running Redhat Linux version 2.6.9 and using g++ version 3.4.2.

The times reported are user mode CPU time plus kernel mode CPU time. We feel that this is a better measure than simple elapsed time, especially on modern, multiprogrammed operating systems. No other users were logged onto the system during the tests. Four input test images were employed to evaluate the various distance transform algorithms: (1) a solitary object consisting of a single solitary point at the center of the image (Figure 2a), (2) the extremely problematic image consisting of 3 point-objects (Figure 2b), (3) a randomly generated set of objects created by sampling from a normal distribution with a mean of the center of the image an a standard deviation of 0.20 (Figures 2c), and (4) another randomly generated set of objects created by sampling from a normal distribution with a different standard deviation of 0.05 (Figure 2d). Each of the input test images were 1000x1000 pixels in size. As previously mentioned, the SimpleList algorithm was used as the gold standard. RMS error as well as the magnitude of the single largest difference are reported as well. The results of the evaluation are shown in Table 1 for the central single-point object and three single-point objects, and 2 for two sets of randomly generated objects.

5. CONCLUDING REMARKS

Although the results in Table 1 appear promising for the gold standard method, SimpleList, with regards to CPU time, Table 2 demonstrates that SimpleList is not practical for most applications because its time is two to three orders of magnitude worse than other methods. The best-performing methods with regard to CPU time took as little as 0.1 seconds. Of these fastest methods, DRA 3x3 exhibited minimal error for the randomly generated images.

With regard to accuracy, DV and SimpleList were the only methods that exhibited 0 errors. The performance of SimpleList precludes it from being used in practice but the performance of DV is quite good for practical use. For applications that can tolerate small errors, the modified 8SED algorithm had a very low error rate and excellent performance.

6. ACKNOWLEDGMENTS

The authors gratefully acknowledge NIH Grant R01-EB004395-01 for support of this work.

Table 1. Results of various distance transform algorithms applied to an image consisting of a solitary object consisting of a single point at the center of the image (left) and 3 single-point objects (right). RMSE is the root mean squared error, max err is the value of the magnitude of the largest difference, diff is the percentage of the total number of points that are different, and time is the CPU time in seconds

	Central single-point object				3 single-point objects			
	RMSE	max err	diff	time	RMSE	max err	diff	time
CDA 3×3	14.7	39.6	95.7	0.1	14.7	39.6	95.8	0.1
CDA 5×5	3.9	10.0	95.7	0.1	3.9	9.9	95.7	0.1
CDA 7×7	2.5	6.8	95.7	0.3	2.4	6.9	95.7	0.3
Chessboard	67.5	202.6	95.8	0.1	67.4	203.0	95.8	0.1
Cityblock	135.0	286.7	95.8	0.1	134.2	287.3	95.8	0.1
CSED	0.0	0.0	0.0	0.2	4E-05	0.0	1E-04	0.4
DRA 3×3	0.0	0.0	0.0	0.1	0.5	2.8	13.7	0.1
DRA 7×7	0.0	0.0	0.0	0.5	0.1	0.9	2.7	0.5
DV	0.0	0.0	0.0	1.8	0.0	0.0	0.0	2.8
8SED	0.0	0.0	0.0	0.2	1E-02	0.2	1.1	0.2
8SED modified	0.0	0.0	0.0	0.2	1E-02	0.1	1.1	0.1
Euclidean 3×3	22.5	43.9	95.8	0.1	22.4	44.1	95.8	0.1
4SED	0.4	1.0	47.8	0.2	0.5	3.0	48.3	0.2
MD	22.5	43.9	95.8	1.4	22.4	44.1	95.8	1.4
MDDR	0.0	0.0	0.0	1.4	1E-02	0.1	1.1	1.4
SimpleList	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.4

Table 2. Results of various distance transform algorithms applied to an image (left) consisting of a randomly generated set of objects created by sampling from a normal distribution with a mean of the center of the image an a standard deviation of 0.20, and (right) another randomly generated set of objects created by sampling from a normal distribution with a different standard deviation of 0.05

	First randomly generated objects				Second randomly generated objects			
	RMSE	max err	diff	time	RMSE	max err	diff	time
CDA 3×3	0.1	1.0	20.3	0.1	7.0	26.6	87.6	0.1
CDA 5×5	2E-02	0.3	20.3	0.1	1.7	5.9	87.5	0.1
CDA 7×7	1E-02	0.2	20.3	0.3	1.1	4.1	87.6	0.3
Chessboard	0.3	5.0	20.9	0.1	42.3	136.3	88.3	0.1
Cityblock	0.5	5.8	20.7	0.1	65.8	192.7	88.1	0.1
CSED	0.0	0.0	0.0	0.5	2E-05	2E-02	4E-04	0.4
DRA 3×3	4E-03	0.6	3E-02	0.1	1.0	18.2	1.7	0.1
DRA 7×7	4E-05	4E-02	1E-04	0.5	1E-02	1.6	0.1	0.5
DV	0.0	0.0	0.0	4.0	0.0	0.0	0.0	3.0
8SED	3E-04	0.1	3E-03	0.2	5E-03	0.3	0.2	0.2
8SED modified	3E-04	0.1	2E-03	0.2	5E-03	0.3	0.2	0.2
Euclidean 3×3	0.1	1.3	10.9	0.1	9.6	28.0	86.5	0.1
4SED	0.2	2.8	10.2	0.2	0.4	3.4	48.4	0.2
MD	0.1	1.3	10.9	3.1	9.6	28.0	86.5	2.0
MDDR	3E-04	0.1	3E-03	3.2	7E-03	0.3	0.4	1.9
SimpleList	0.0	0.0	0.0	5498.2	0.0	0.0	0.0	854.6

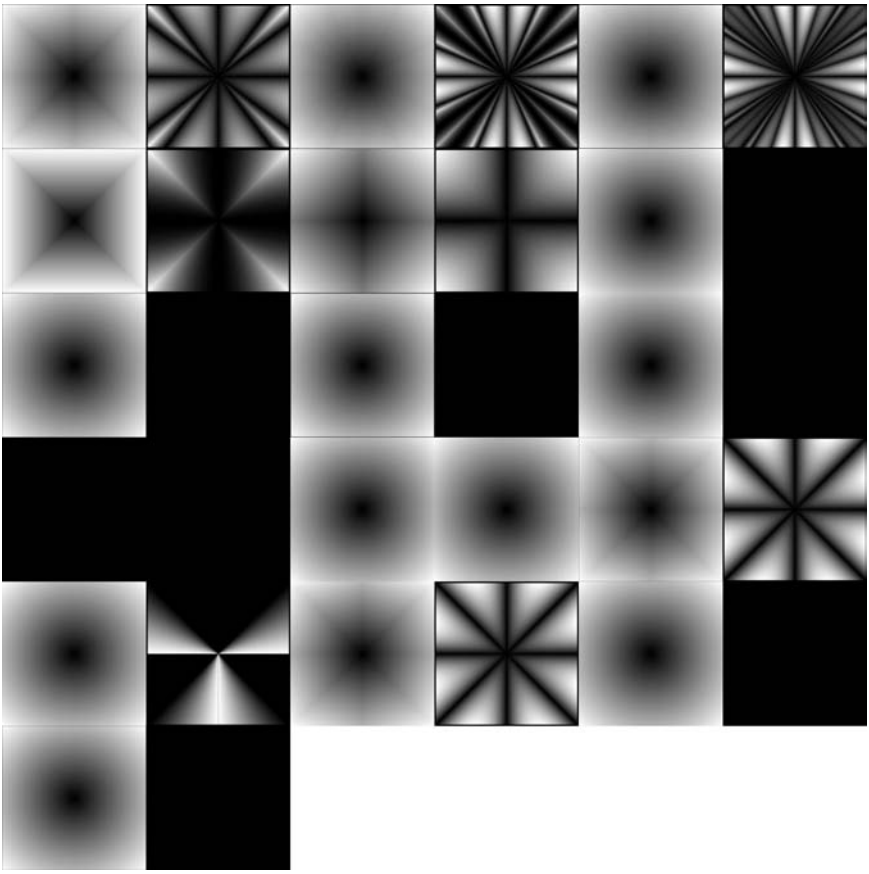

Figure 3. Results of various distance transform algorithms applied to an image consisting of a solitary object consisting of a single point at the center of the image, and the magnitude of the differences for the method. Top row, left to right: CDA 3x3 and differences, CDA 5x5 and differences, CDA 7x7 and differences. Second row, left to right: Chessboard and differences, Cityblock and differences, CSED and difference. Third row, left to right: DRA 3x3 and differences, DRA 7x7 and differences, DV and differences. Fourth row, left to right: 8SED and differences, 8SED modified and differences, Euclidean 3x3 and difference. Fifth row, left to right: 4SED and differences, MD and differences, MDDR and differences. Sixth row: SimpleList and differences.

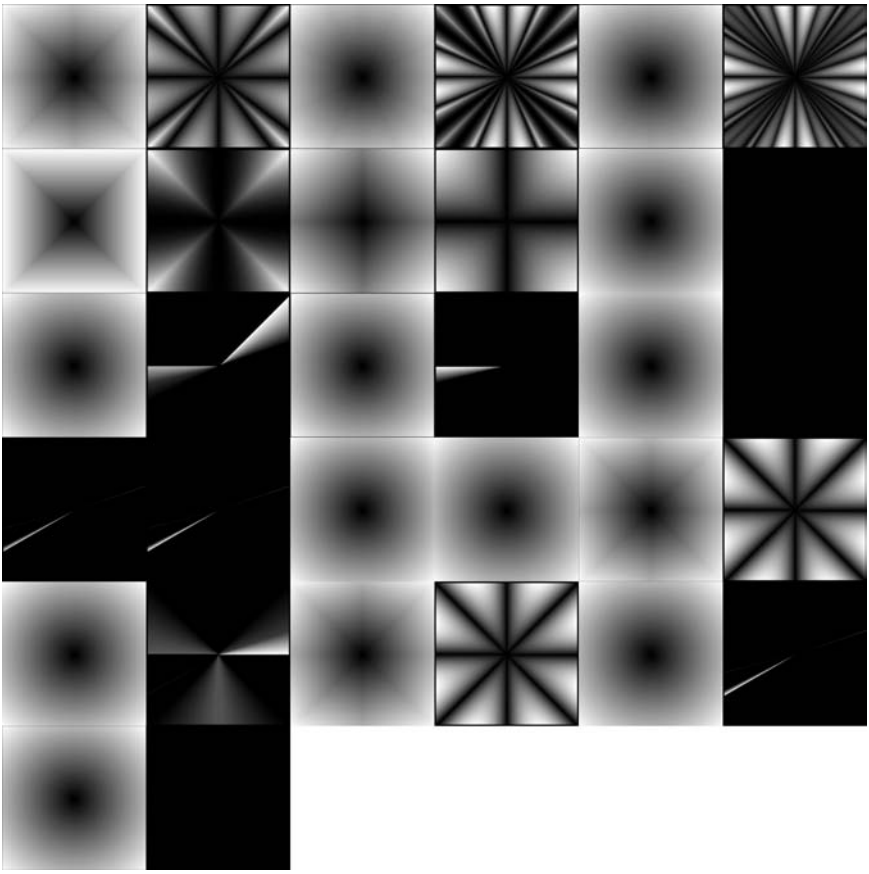

Figure 4. Results of various distance transform algorithms applied to an image consisting of 3 single-point objects. Top row, left to right: CDA 3x3, CDA 5x5, CDA 7x7. Second row, left to right: Chessboard, Cityblock, CSED. Third row, left to right: DRA 3x3, DRA 7x7, DV. Fourth row, left to right: 8SED, 8SED modified, Euclidean 3x3. Fifth row, left to right: 4SED, MD, MDDR. Sixth row: SimpleList.

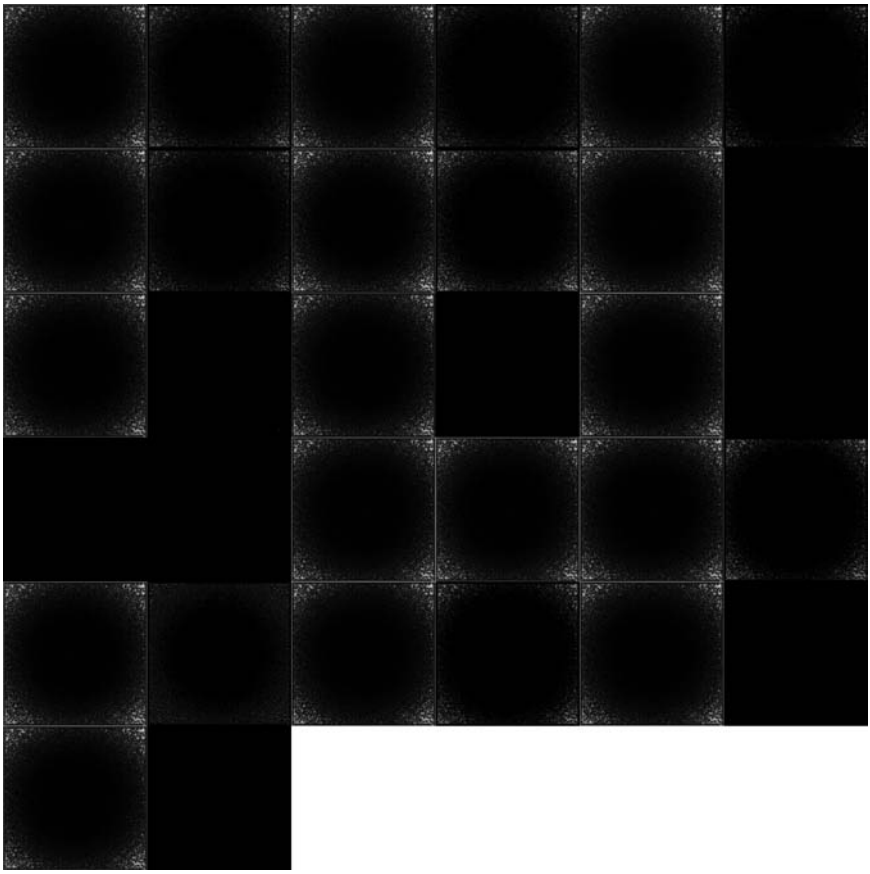

Figure 5. Results of various distance transform algorithms applied to an image consisting of a randomly generated set of objects created by sampling from a normal distribution with a mean of the center of the image an a standard deviation of 0.20. Top row, left to right: CDA 3x3, CDA 5x5, CDA 7x7. Second row, left to right: Chessboard, Cityblock, CSED. Third row, left to right: DRA 3x3, DRA 7x7, DV. Fourth row, left to right: 8SED, 8SED modified, Euclidean 3x3. Fifth row, left to right: 4SED, MD, MDDR. Sixth row: SimpleList.

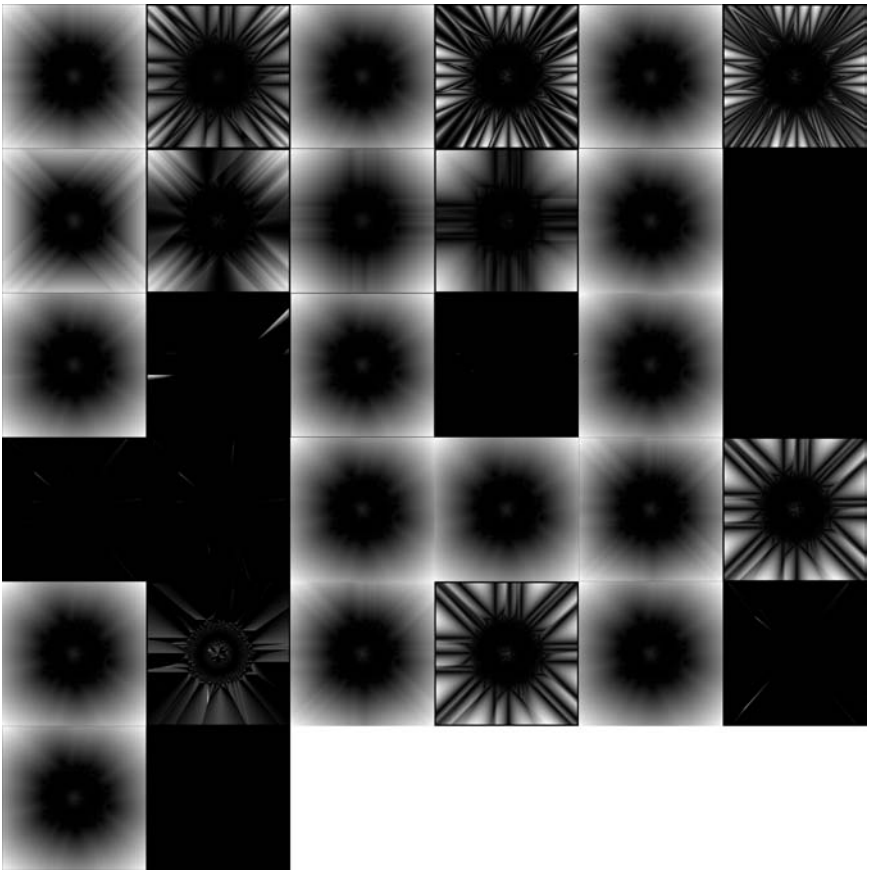

Figure 6. Results of various distance transform algorithms applied to an image consisting of a randomly generated set of objects created by sampling from a normal distribution with a mean of the center of the image an a standard deviation of 0.05. Top row, left to right: CDA 3x3, CDA 5x5, CDA 7x7. Second row, left to right: Chessboard, Cityblock, CSED. Third row, left to right: DRA 3x3, DRA 7x7, DV. Fourth row, left to right: 8SED, 8SED modified, Euclidean 3x3. Fifth row, left to right: 4SED, MD, MDDR. Sixth row: SimpleList.

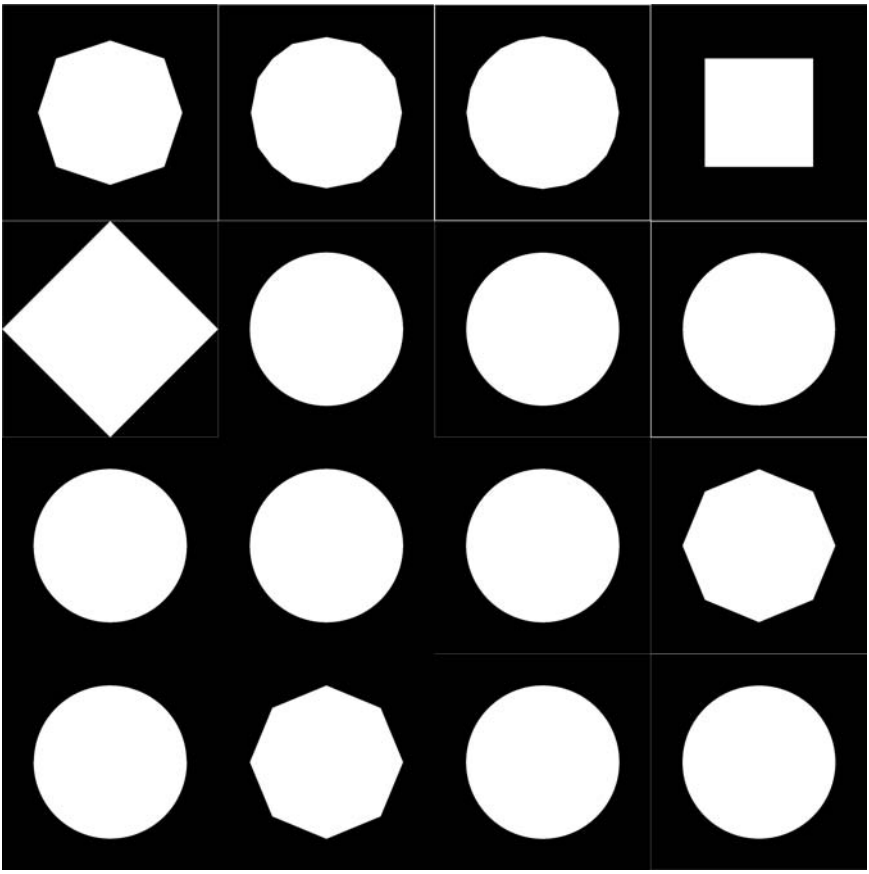

Figure 7. Thresholded results of various distance transform algorithms applied to an image consisting of a solitary object consisting of a single point at the center of the image. Top row, left to right: CDA 3x3, CDA 5x5, CDA 7x7, Chessboard. Second row, left to right: Cityblock, CSED, DRA 3x3, DRA 7x7. Third row, left to right: DV, 8SED, 8SED modified, Euclidean 3x3. Fourth row, left to right: 4SED, MD, MDDR, SimpleList.

7. APPENDIX

The accompanying CD contains implementations (C++ classes that compile and run under Windows with VC++ 6, and Linux and Solaris with g++) of the following distance transform algorithms in an extensible framework. Updates to this software can be found at http://www.sju.edu/~ggrevera.

- Chamfer2D_3x3
- Chamfer2D_5x5
- Chamfer2D_7x7
- Chessboard2D
- Cityblock2D
- CSED
- DeadReckoning_3x3
- DeadReckoning_7x7
- DijkstraVectors
- EightSED
- EightSED_modified
- errorfreeCSED
- Euclidean2D
- FourSED
- ModifiedDijkstra
- Simple
- SimpleList

Other support classes include:

- **CLUT —** CLUT (Color LookUp Table) class for writing some color TIFF image files
- **Timer —** Timer class for reporting elapsed time and CPU time
- **Normal —** Normal class which samples random numbers from a normal distribution using the Box-Muller transform

- **TIFFWriter —** This class contains methods that write 8-bit color rgb images or float, double, 8-bit, or 16-bit grey images
- **DistanceTransform —** an abstract base class from which all distance transform classes inherit

A VC++ 6 workspace is included along with Windows executables. The Makefile works under both Linux and Solaris. The main.cpp file creates binary test images, applies each of the distance transforms in turn to the test images, evaluates the results, and creates TIFF images of the results.

8. REFERENCES

1. Udupa JK. 1994. Multidimensional digital boundaries. *Comput Vision Graphics Image Process: Graphical Models Image Process* **56**(4):311–323.
2. Cuisenaire O. 1999. Distance transformations: fast algorithms and applications to medical image processing. PhD thesis. Université Catholique de Louvian.
3. Rosenfeld A, Pfaltz JL. 1968. Distance functions on digital pictures. *Pattern Recognit* **1**(1):33–61.
4. Montanari U. 1968. A method for obtaining skeletons using a quasi-Euclidean distance. *J Assoc Comput Machin* **15**:600–624.
5. Danielsson P-E. 1980. Euclidean distance mapping. *Comput Graphics Image Process* **14**:227–248.
6. Borgefors G. 1986. Distance transformations in digital images. *Comput Vision Graphics Image Process* **34**:344–371.
7. Borgefors G. 1996. On digital distance transforms in three dimensions. *Comput Vision Image Understand* **64**(3):368–376.
8. Butt MA, Maragos P. 1998. Optimum design of chamfer distance transforms. *IEEE Trans Image Process* **7**(10):1477–1484.
9. Marchand-Maillet S, Sharaiha YM. 1999. Euclidean ordering via Chamfer distance calculations. *Comput Vision Image Understand* **73**(3):404–413.
10. Nilsson NJ. *Artificial intelligence: a new synthesis*. San Francisco: Morgan Kaufmann, 1998.
11. Verwer BJH, Verbeek PW, Dekker ST. 1989. An efficient uniform cost algorithm applied to distance transforms. *IEEE Trans Pattern Anal Machine Intell* **11**(4):425–429.
12. Leymarie F, Levine MD. 1992. Fast raster scan distance propagation on the discrete rectangular lattice. *Comput Vision Graphics Image Process: Image Understand* **55**(1):84–94.
13. Satherley R, Jones MW. 2001. Vector-city vector distance transform. *Comput Vision Image Understand* **82**:238–254.
14. Ragnemalm I. 1992. Neighborhoods for distance transformations using ordered propagation. *Comput Vision Graphics Image Process: Image Understand* **56**(3):399–409.
15. Guan W, Ma S. 1998. A list-processing approach to compute Voronoi diagrams and the Euclidean distance transform. *IEEE Trans Pattern Anal Machine Intell* **20**(7):757–761.
16. Eggers H. 1998. Two fast Euclidean distance transformations in Z2 based on sufficient propagation. *Comput Vision Image Understand* **69**(1):106–116.
17. Saito T, Toriwaki J-I. 1994. New algorithms for euclidean distance transformation of an n-dimensional digitized picture with application. *Pattern Recognit* **27**(11):1551–1565.
18. Boxer L, Miller R. 2000. Efficient computation of the Euclidean distance transform. *Comput Vision Image Understand* **80**:379–383.
19. Meijster A, Roerdink JBTM, Hesselink WH. 2000. A general algorithm for computing distance transforms in linear time. In *Mathematical morphology and its applications to image and signal processing*, pp. 331–340. Ed. J Goutsias, L Vincent, DS Bloombers. New York: Kluwer.

20. Lotufo RA, Falcao AA, Zampirolli FA. 2000. Fast Euclidean distance transform using a graph-search algorithm. *SIBGRAPI* **2000**:269–275.
21. da Fontoura Costa L. 2000. Robust skeletonization through exact Euclidean distance transform and its application to neuromorphometry. *Real-Time Imaging* **6**:415–431.
22. Pudney C. 1998. Distance-ordered homotopic thinning: a skeletonization algorithm for 3D digital images. *Comput Vision Image Understand* **72**(3):404–413.
23. Sanniti di Baja G. 1994. Well-shaped, stable, and reversible skeletons from the (3,4)-distance transform. *J Visual Commun Image Represent* **5**(1):107–115.
24. Svensson S, Borgefors G. 1999. On reversible skeletonization using anchor-points from distance transforms. *J Visual Commun Image Represent* **10**:379–397.
25. Herman GT, Zheng J, Bucholtz CA. 1992. Shape-based interpolation, *IEEE Comput Graphics Appl* **12**(3):69–79.
26. Raya SP, Udupa JK. 1990. Shape-based interpolation of multidimensional objects. *IEEE Trans Med Imaging* **9**(1):32–42.
27. Grevera GJ, Udupa JK. 1996. Shape-based interpolation of multidimensional grey-level images. *IEEE Trans Med Imaging* **15**(6):881–892.
28. Kozinska D. 1997. Multidimensional alignment using the Euclidean distance transform. *Graphical Models Image Process* **59**(6):373–387.
29. Paglieroni DW. 1997. Directional distance transforms and height field preprocessing for efficient ray tracing. *Graphical Models Image Process* **59**(4):253–264.
30. Remy E, Thiel E. 2000. Computing 3D medial axis for Chamfer distances. *Discrete Geom Comput Imagery* pp. 418–430.
31. Remy E, Thiel E. 2002. Medial axis for chamfer distances: computing look-up tables and neighbourhoods in 2D or 3D. *Pattern Recognit Lett* **23**(6):649–662.
32. Grevera GJ, Udupa JK. 1998. An objective comparison of 3D image interpolation methods. *IEEE Trans Med Imaging* 17(4):642–652.
33. Grevera GJ, Udupa JK, Miki Y. 1999. A task-specific evaluation of three-dimensional image interpolation techniques. *IEEE Trans Med Imaging* **18**(2):137–143.
34. Travis AJ, Hirst DJ, Chesson A. 1996. Automatic classification of plant cells according to tissue type using anatomical features obtained by the distance transform. *Ann Botany* **78**:325–331.
35. Van Der Heijden GWAM, Van De Vooren JG, Van De Wiel CCM. 1995. Measuring cell wall dimensions using the distance transform. *Ann Botany* **75**:545–552.
36. Schnabel JA, Wang L, Arridge SR. 1996. Shape description of spinal cord atrophy in patients with MS. Comput Assist Radiol ICS **1124**:286–291.
37. Grevera GJ. 2004. The “dead reckoning” signed distance transform. *Comput Vision Image Understand* **95**:317-333.
38. Oppenheim AV, Schafer RW, Buck JR. 1999. *Discrete-time signal processing*, 2d ed. Englewood Cliffs: Prentice Hall.
39. Cormen TH, Leiserson CE, Rivest RL, Stein C. 2001. *Introduction to algorithms*, 2d ed. Cambridge: MIT Press.
40. Svensson S, Borgefors G. 2002. Digital distance transforms in 3D images using information from neighbourhoods up to 5x5x5. *Comput Vision Image Understand* **88**:24-53.