MEMORY EFFICIENT PROPAGATION-BASED ALGOTITHMS FOR INFLUENCE ZONE TRANSMISSION

C. I. Cotsaces and I. Pitas

Department of Informatics
University of Thessaloniki
Thessaloniki 540 06, GREECE
E-mail:pitas@zeus.csd.auth.gr

ABSTRACT

The influence zone transform is a fundamental tool in morphological and qualitative digital image processing. Because of its inherent geodesic properties, it is most efficiently computed using propagation front or grassfire based methods. However, when the image processed is too large to be contained in available memory, the random access nature of these algorithms makes them exceptionally inefficient. In order to alleviate this problem, we have developed two algorithms that greatly reduce the memory requirements of the transform. The first is designed specifically for computing the influence zone transform on surfaces, without storing the volume enclosing the surface. The second performs the transform using only the propagation fronts, and without storing any part of the region that is being processed. Both methods use much less memory than the ones in the literature, and thus enable the transform to be performed on much larger images than before. However, since all three algorithms use a significant number of set-access operations, they are considerably slower that their classical counterparts. Several techniques have been developed in this work in order to minimize the effect of these set operations. These include fast search methods, double propagation fronts, directional propagation, and others.

1. INTRODUCTION

The concept of distance [1] is fundamental to most areas of image processing and analysis. The *geodesic* distance functions are particularly useful in many tasks that require the description of distance within a specific region. There are a number of methods that can segment an image according to a distance criterion. Among the simplest of these is the influence zones transform [2], which classifies points of the image according to their geodesic distance from a predefined number of markers. It is commonly accepted that the above method is by far best performed by propagation fronts. Here we shall concentrate on the use of propagation-based algorithms on images that cannot fit in available memory. Such are the three-dimensional images, whose third dimension greatly increases the image size. Three dimensional images are used mostly in biomedical applications [**?**], but also in geophysics, in industrial quality control and elsewhere. Large images are also met where exceptionally high resolution is required, as in digital photography, topography and geodesy, and in other applications. In the following the description of the algorithms will be given for the case of of 3-D images. However, the algorithms are equally applicable in two or multiple dimensions.

When processing large images, it is impossible to keep the entire image in the computer memory (RAM). Usually, the image is stored in the permanent storage medium (magnetic disk, WORM, CDROM, network storage) and the algorithm reads parts of the image that are needed in the computation. The part of the image that is read each time has to be small enough to fit in available memory. Each point in the image is read as many times as the algorithm processes it. However, seek time exceeds read time by orders of magnitude, so it is important to read the data sequentially, when this is possible. t is also important that each point of the image be read as few times as possible. Most image processing transforms are local, and therefore can be implemented by sequential algorithms [6]. These process the image using a predefined scanning order. Consequently the image may be read from the permanent storage medium in segments that have size equal to the available memory, thus minimizing the delay caused by the seek operations. However, in propagation-based methods for the influence zones transform there is no predefined order for processing the image. Therefore, the use of propagation front algorithms in large images presents serious problems.

In the following, we shall present two algorithms that solve the above problems by minimizing memory requirements. First, in Section 2, the theoretical foundations of the influence zones transform are briefly reviewed. Then the existing algorithms are described. In Section 3, an algorithm designed to perform the influence zone transform on surfaces or other regions of interest is presented. in Section 4, we describe a second algorithm which processes complete domains without the storage of the underlying image, but only using the propagation fronts.

2. THEORETICAL BACKGROUND AND ALGORITHMS

2.1. Basic definitions

Let D denote and image region $D \subset \mathcal{Z}^n$. A path of length n in D is defined as a sequence of points $(\mathbf{p_0}, \mathbf{p_1}, \ldots, \mathbf{p_n})$, $\mathbf{p_i} \in D$ such that $\mathbf{p}_{i+1}$ is adjacent to $\mathbf{p}_i$. The discrete

geodesic distance from point $\mathbf{p}$ to point $\mathbf{p}'$ is defined as the minimum length of all paths starting at $\mathbf{p}$ and ending at $\mathbf{p}'$ and will be denoted $\text{dist}(\mathbf{p}, \mathbf{p}')$. Furthermore, the distance from a set $S \subset D$ to a point $\mathbf{p}$ is defined as the minimum of all distances from points of S to $\mathbf{p}$. We define a *growth* of S within D as the set $S(1) = S \cup G(1)$ where $G(1) = \{\mathbf{p} \in D - S,\ \mathbf{p} \in N_C(\mathbf{p}'),\ \mathbf{p}' \in S\}$. It contains only all points that are adjacent to S. The n-th growth of S, $S(n)$ is the growth of $S(n-1)$. Similarly, it contains all points that have a distance of n from S. In morphological terms, if the connectivity C is seen as a structuring element, the n-th growth of set S is expressed as $(S \oplus nC^s) - S$.

2.2. Influence zone transform

Supposing that there are m sets $S_1, \dots S_m,\ S_i \in D, S_i \cup S_j = 0, i \neq j$ (called *markers*), we define the *influence zone* of S_i as the set:

$$IZ_D(S_i) = \{\mathbf{p} \in D,\ \text{dist}(S_i, \mathbf{p}) < \text{dist}(S_j, \mathbf{p}),\ \forall j \neq i\}$$

The skeleton by influence zones of D is defined as:

$$SKIZ_D = \{\mathbf{p} \in D : \exists i \neq j,\ \text{dist}(S_i, \mathbf{p}) = \text{dist}(S_j, \mathbf{p})\}$$

A point $p \in D$ belongs to $IZ_D(S_i)$ if and only if:

$$p \in S_i(n),\ p \notin S_j(m),\ \forall j \neq i,\ \forall m \leq n$$

By defining as $D(n) = \{\mathbf{p} \in D, \exists i : \mathbf{p} \in S_i(n),\ \nexists j :\ \mathbf{p} \in S_j(m)\ m > n\}$, it is clear that if $\mathbf{p} \in D(n),\ \mathbf{p} \in SKIZ_D$ then for all $\mathbf{p}'$ such that $\mathbf{p}' \in N_C(\mathbf{p}),\ \mathbf{p}' \in D(n+1)$ hold that $\mathbf{p}' \in SKIZ_D$. Thus, the influence zone transform can yield very thick skeletons. It is consequently necessary to modify its computation in such a way as to minimize the number of skeleton points without sacrificing consistency. This can be achieved by modifying D, defining $D'(n) = D(n) - SKIZ_D(n)$ and calculating $D(n+1)$ using $D'(n)$. This effectively eliminates the effect described above, without altering the classification of points already in influence zones.

2.3. Algorithms using propagation fronts

There are many algorithms for computing the influence zones, some of them parallel, other sequential. The fastest, though, are those that use *propagation fronts* (also called grassfires or wavefronts) to process each point in D only once, that is at the time a zone's growth reaches it. These propagation fronts are implemented by a data strucure (queue, list, stack, array) that stores all points in the edge of the influence zone. In the following we shall refer to that structure as a propagation front, without specifically stating how it is implemented.

These algorithms have the following steps:

1. Label the points of the influence zone of each marker S_i with a unique label in a label image.
2. Put the points of each marker in the propagation front.
3. For each top point of each propagation front, find all adjacent points that are not labeled in the label image, put them in the propagation front and label them.
4. Remove each point from a propagation front after it has been processed, then go to step 3.

3. INFLUENCE ZONE TRANSFORM ON SURFACES AND OTHER REGIONS OF INTEREST

In various applications in image processing, graphics and other fields, it is often necessary to perform a geodesic influence zone transform on a surface. For example, this is the case when computing the Delauney triangulation of a surface [8], based on previously calculated geodesic influence zones (Voronoi tesselation).

However, in the case of the algorithms described in Section 2.3, in order to store the surface in RAM, the storage of the enclosing rectangular volume is needed. If the surface is not approximately planar, the size of the volume that needs to be stored in memory exceeds the size of the surface by orders of magnitude. Thus, the volume containing a surface may be too large to fit in available memory, although the surface itself is not. Because the transformation is performed on an image domain too large to fit in memory, the performance of the classical algorithms drops drastically. The proposed solution to this problem is to store only the surface points, and not the whole volume.

3.1. Storage of the region of interest

The most common method to represent an image or a volume is in the form of a bitmap — an array which contains the image information, whether it is greyvalues or labels. However, because of the one-to-one correspondence between memory and image, the shape of a region stored as a bitmap can only be a parallelepiped. Thus, in order to store an arbitrarily shaped region or surface, this method will need to be abandoned and alternative ones be sought.

The obvious method for the representation of an arbitrarily shaped point set is to store each point separately, defining it with its coordinates and value, in an array. The disadvantage of this method is that accessing the value of a point, given its coordinates, requires a search through the entire image region that is stored in memory. The solution that is applied to solve this problem is two sided. Firstly, the minimization of the seek time of a point in the image is attempted. Secondly, ways to limit the search operations of the algorithm are sought.

In order to minimize the seek time of a point in a region stored in an array of points in coordinate form is to sort this array, and then to perform sorted search operations on it. The ordering that is going to be used in the following is the one commonly used in the storage of images. In the three-dimensional case we sort first by the z-coordinate, then by the y-coordinate and then by the x-coordinate. Hence, this ordering will be called the *basic order*. It will be shown that this choice of ordering will be beneficial because of its correspondence with the way images are stored.

3.2. Use of double propagation fronts

The only image access operations performed by the propagation front based algorithm are those that find the image

points adjacent to a propagation front point and, depending on their labels, may add them to the propagation front and label them accordingly. Since the size of a propagation front is much smaller than the size of the image region, searching for a point in it is much faster than searching for it in the image. This only holds if the points in the propagation front are sorted, facilitating search operations of logarithmic computational complexity. Thus, if the points in the current propagation fronts are sorted before the propagation begins and each point adjacent to them is sought among them before being sought in the image, significant performance gains can be achieved. In order to reject points from previous propagations, the immediately previous propagation fronts must be kept, and points must be searched in them as well as in the current propagation fronts. These previous propagations constitute what we will call the *second* or *backup level* of the propagation fronts.

So, the final method used for the minimization of search operations on the entire image region is storing the two levels of the propagation fronts. Thus, each point adjacent to a point that belongs to the current propagation front is first sought in these two levels of the propagation fronts and, if it is not found there, it is sought in the image. On average, half of these adjacent points are found in the propagation fronts and the rest must be searched for both in the image and in the propagation fronts. Thus, the speed of this part of the algorithm is effectively doubled.

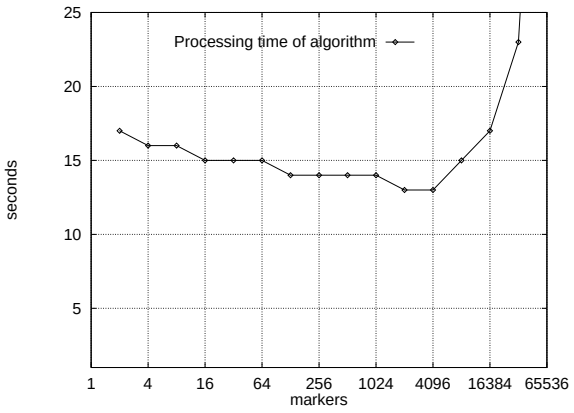

Figure 1: Processing time for the domain storage algorithm.

3.3. Results

A test image was created in order to verify the algorithm experimentally. It consists of a spherical surface having a radius of 127 enclosed in a $256 \times 256 \times 256$ volume. In this sphere a number of point markers were randomly inserted, ranging in number from 2 to 65536. The tests were performed on a PC with a Pentium processor at 90 MHz and 32 Mbytes of RAM, running Microsoft Windows NT Server.

The speed of the algorithm remained constant at 13 to 14 seconds for most of the tests and only rse significantly when using more than 16384 marker points, as can be seen in Figure 1. Above that number, the number of markers

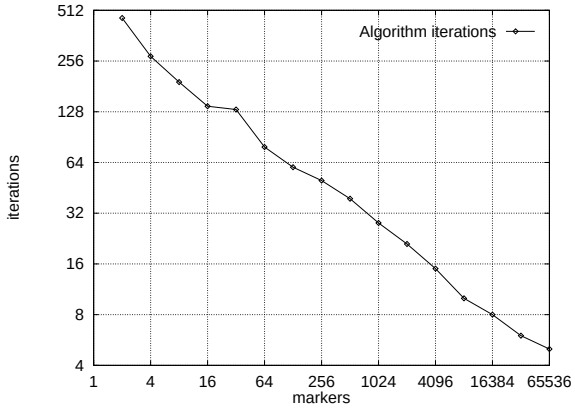

Figure 2: Maximum number of iterations for the domain storage algorithm.

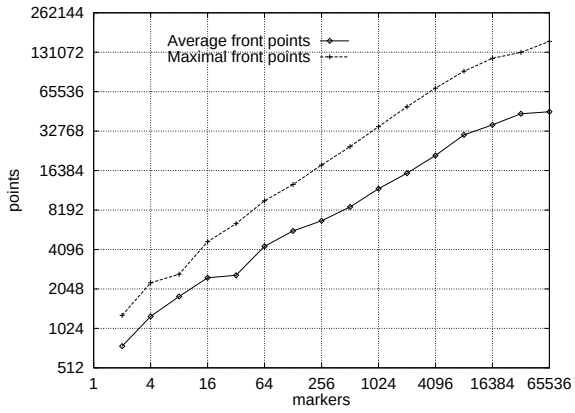

Figure 3: Maximum and average number of points in propagation fronts for the domain storage algorithm.

becomes comparable to the number of points in the surface, and the entire algorithm degenerates. The execution time remains constant although the number of iterations decreases with the number of markers as can be seen in Figure 2, meaning that the time spent on each iteration increases with marker number. he number of points in the propagation fronts increased with marker number, but they never exceeded the number of points in the region, as can be seen in Figure 3. In most cases, the average number of points in the fronts remained well over one order of magnitude below the number of domain points.

If we assume a uniform distribution of markers on the image, the number of points belonging to each influence zone can be approximated as $P_{zone} = P_{Region}/n$ where n the number of markers and P_{Region} the constant number of points in the region. If the region is modeled as a volume, the radius of the influence zone is $r = k_1 \sqrt[3]{P_{zone}}$. Therefore the number of points on the front is $P_{front} = k_2(P_{Region}/n)^{2/3}$, and thus the total number of points in fronts $P_{tot} = k_3 \sqrt[3]{n}$. If the region is modeled as a plane,

the radius is $r = k_4\sqrt{P_{zone}}$ and the number of points in fronts is $P_{front} = k_5\sqrt{P_{Region}/n}$ and the total number $P_{tot} = k_6\sqrt{n}$. Because the region used is almost exactly the surface of a sphere, the results for the above quantities should be closer to that of the planar case. Indeed the measurements shown in Figure 3 were modeled by the least-squares method to be as follows:

- The maximum number of points stored in propagation fronts during the execution of the algorithm is modelled by $\max(P_{tot}) = kn^{0.47}$.
- The average number of points stored in propagation fronts during execution was modelled by $\text{avg}(P_{tot}) = kn^{0,40}$.

Also, again as expected, the number of propagation steps, as shown in Figure 2 was modelled as $\max(r) = kn^{-0,42}$.

4. INFLUENCE ZONES TRANSFORM WITHOUT IMAGE STORAGE

The algorithm presented in Section 3.1 is of no help in the case that the area to be segmented *is* a large image. In this case, the need to store the influence zone labels leads to serious performance problems. The only way to avoid these problems is to avoid the storage of the image in memory and to store only the propagation fronts and compute the transform solely with operations on them.

Unfortunately, without the image, there is no simple way to know whether a point belongs to the skeleton or to an influence zone (and to which) or to neither. The influence zone algorithm requires that each point adjacent to a propagation front point is labeled as an influence zone or skeleton point and, accordingly, is appended to the propagation fronts or not. Thus, we have to find some other way to determine whether such an adjacent point has already been visited by the propagation fronts, or is a new propagation front point or is a collision point that belongs to the skeleton.

4.1. Determining collisions

The first step towards determining the status of a point adjacent to a propagation front is to establish whether it has already been visited by that propagation front. As described in Section 3.2, a way to achieve this is to use double (or backed-up) propagation fronts. Because the maintenance and operation of the double fronts is inexpensive in both memory and computation terms, and because they do not require image reference operations, they are a natural choice for this task.

However, the double fronts cannot determine whether a point has been visited by *another* propagation front, or whether it is the point where two propagation fronts collide. Having determined that a new point has not been already visited by its respective propagation front, an way must be found to check whether it appears in any other new point set or propagation front. In order to achieve this, a *minimum heap* is used [9]. The heap's function is to extract quickly the minimum element it contains, and to accept new elements in any order. The computational cost for the extraction of the minimum of the heap is $O(\log(i))$, and the cost for the insertion of an arbitrary element is also $O(\log(i))$ where i is the number of elements in the heap.

The heap is used to receive the points from the propagation fronts and the sets of new points that are to be appended to each of them. The size of the heap is $2n$ points and the computational cost of the insertion $O(\log(n))$, where n is the number of markers. This is because both the fronts and the new point sets are sorted on the basic order beforehand. Thus, if their first points are inserted into the heap, the next point extracted from the heap will be the minimum of all the points left in them.

Thus, by successively extracting points from the heap, and then inserting into the heap the next point from where the extracted point originated from, we can guarantee to receive all the points in the basic order. Duplicate points will be in succession and can be identified as such. If there is a number of identical points from different new point sets, and there is no point from a propagation front among them, it means that this point is a skeleton point. In this case, all the points are removed from their respective sets and appended to the set of skeleton points. If there is a point originating from a propagation front among a number of identical points, all other points are removed from their respective sets. Thus, the determination of the collisions between fronts can be achieved with only $O(\log(n))$ computational complexity per point.

4.2. Skeleton maintenance and result output

In order to have a proper computation of the influence zones transform, the skeleton points must be properly computed. Because the propagation step does not continue through skeleton points, and because of the geodesic nature of the transform, all skeleton points need to be kept in order to ensure the correct propagation of the fronts. So, the skeleton points found by using the heap at the end of each propagation step are temporarily kept in a list. When the heap has processed all points this list is sorted and merged into an array containing all skeleton points found in previous propagation steps. This array is then used as an input to the heap, in order to determine further collisions.

Also, the fact the image is not kept in memory means that the result is not explicitly available. However, it is possible to store the result of the transformation to disk, as the algorithm is running. This is best done at the end of each propagation step, when all points pass through the heap and receive their final labels. At that time, the points that have been reached in the current propagation step are written, in coordinate form, to intermediate disk file corresponding to their z-coordinate. Because the points are coming out of the heap in the basic order (which was chosen to be based first on their z-coordinate), they are written to disk continuously and, therefore, without delays. After the end of the algorithm, these files are read and the points they contain are written to their final positions in the output file, which is in bitmap format.

4.3. Results

We have used our algorithm to perform the influence zone transform on a $256 \times 256 \times 256$ image, which contained from 2 to 16385 markers. The machine used for the tests was a

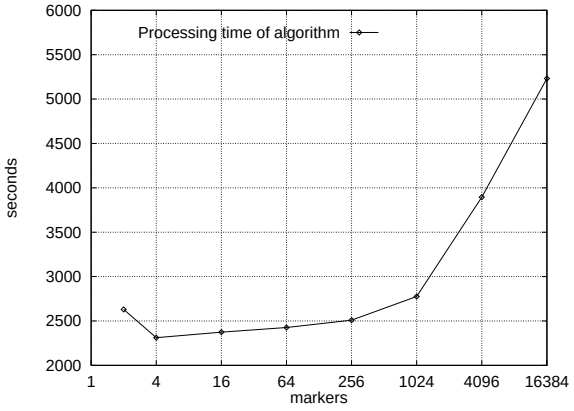

Figure 4: Processing time for the fronts-only algorithm.

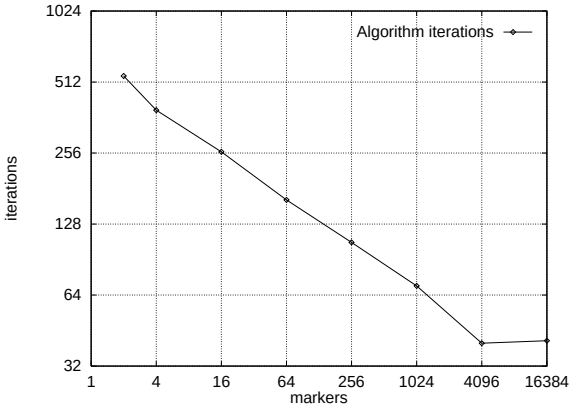

Figure 5: Maximum number of iterations for the fronts-only algorithm.

PC with a Pentium processor at 90 MHz and 64 Mbytes of RAM, running Microsoft Windows NT Server. The performance of the algorithm was satisfactory, rising very slowly for small numbers of markers, but faster when the number of markers goes above 1024. The memory requirements of the fronts were naturally small, but greatly increased with marker density. Again, the number of iterations were dependant on the number of markers. The above are demonstrated in Figures 4, 5 and 6.

5. CONCLUSIONS

We have presented two algorithms to reduce greatly the memory requirements of the influence zone algorithm, thus making them computable on large images. The first of these algorithms which computes the influence zone transform on surfaces, was demonstrated to have in satisfactory speed and to achieve very large memory savings. The performance of the second, which uses only the propagation fronts to perform the transform, is somewhat less satisfactory in speed, but still achieves significant memory gains with reasonable

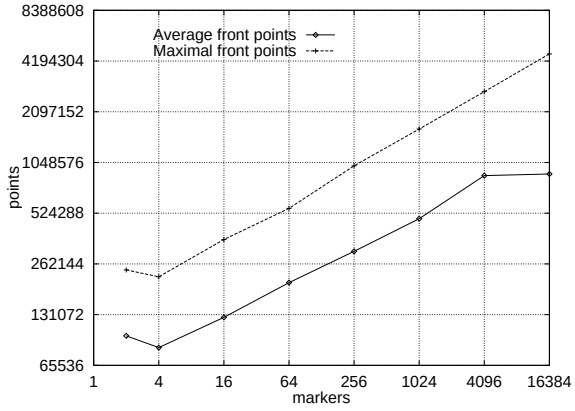

Figure 6: Maximum and average number of points in propagation fronts for the fronts-only algorithm.

slowdown.

6. REFERENCES

[1] A. Rosenfeld and J. Pfalz, "Distance Functions on Digital Pictures", *Pattern Recognition*, vol. 1, no. 1, pp. 33–61, 1968.

[2] C. Lantuejoul and F. Maisonneuve, "Geodesic Methods in Quantitative Image Analysis" *Pattern Recognition*, vol. 17, no. 2, pp. 177–187, 1984.

[3] F.P. Preparata and M.I. Shamos, *Computational Geometry*, Springer-Verlag, 1985.

[4] G. Borgefors "Distance Transformations in Digital Images" *Computer Vision, Graphics, Image Proc.*, vol. 34, pp. 344–371, 1986.

[5] I. Pitas, "Performance Analysis and Parallel Implementation of Voronoi Tessellation Algorithms Based on Mathematical Morphology", *IEEE Pattern Anal. Machine Intel.*, submitted for publication, April 1996.

[6] I. Pitas, *Digital Image Processing Algorithms*, Prentice Hall, 1993.

[7] B.J.H. Verwer, P.W. Verbeek and S.T. Dekker, "An Efficient Uniform Cost Algorithm Applied to Distance Transforms", *IEEE Trans. Pattern Anal. Machine Intel.*, vol. 11, no. 4, pp. 425–429, April 1989.

[8] L. Vincent, "Graphs and Mathematical Morphology", *Signal Processing*, vol 16, pp. 365–388, 1989.

[9] M. Atkinson, J. Sack, N. Santoro and T. Strotthotte, "Max-Min Heaps and Generalized Priority Queues", *Communications of ACM*, vol. 29, no. 10, pp.996–1000, 1986.

[10] P. Salembier and A. Oliveras, "Practical Extensions of Connected Operations", *Math. Morph. and Appl. to Image and Signal Proc.*, P. Maragos *et al.*, Eds. Kluwer, pp. 97–118, 1996.