Optimal Parallel Algorithms for the 3D Euclidean Distance Transform on the CRCW and EREW PRAM Models *

Yuh-Rau Wang†‡, Shi-Jinn Horng§†, Yu-Hua Lee†, and Pei-Zong Lee§

†Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan.
‡Department of Electronic Engineering, Lan Yang Institute of Technology, I-Lan, Taiwan.
§Institute of Information Science, Academia Sinica, Taipei, Taiwan.

January 18, 2002

Abstract

In this paper, we develop parallel algorithms for the three-dimensional Euclidean distance transform (3D_EDT) on both the CRCW and EREW PRAM computation models. Based on the dimensionality reduction technique and the solution for proximate points problem, we achieve the optimality of the 3D_EDT computation. For an \( N \times N \times N \) binary image array, our parallel algorithms for 3D_EDT computation are in \( O(\log \log N) \) time using \( \frac{N^3}{\log \log N} \) CRCW processors or in \( O(\log N) \) time using \( \frac{N^3}{\log \log N} \) EREW processors. We then extend it to the n-dimensional space to compute the n-dimensional Euclidean distance transform (nD_EDT) of a binary image array of size \( N^n \). The n-dimensional Euclidean distance transform of a binary image of size \( N^n \) can be computed in \( O(n \log \log N) \) time using \( \frac{N^n}{\log \log N} \) CRCW processors or in \( O(n \log N) \) time using \( \frac{N^n}{\log \log N} \) EREW processors. As for applications, our parallel 3D_EDT algorithm can be used to build up Voronoi diagram and Voronoi polyhedra, to find all maximal empty spheres and the largest empty sphere, and to compute the distance-based medial axis transform in a 3-D binary image. All of these parallel algorithms can be performed in \( O(\log \log N) \) time using \( \frac{N^n}{\log \log N} \) CRCW processors or in \( O(\log N) \) time using \( \frac{N^n}{\log \log N} \) EREW processors. Our algorithms for the nD_EDT can be also applied to extend those applications described above to an n-dimensional space. To the best of our knowledge, all results derived in this paper are the best that never found before.

Keywords: Computer Vision, Distance Transform, Euclidean Distance Transform, Image Processing, Parallel Algorithm, Voronoi Diagram, Proximate point, EREW PRAM Model, CRCW PRAM Model, all maximal empty spheres, the largest empty sphere, medial axis transform.

1 Introduction

Given a computational problem \( P \), let a sequential time complexity of \( P \) be \( T^*(n) \). A sequential algorithm is termed time-optimal [24] if its time bound \( O(T^*(n)) \) cannot be improved. The main parallel complexity measure for evaluating the performance of an algorithm is the amount \( W(n) \) of work performed by this algorithm, denoted by the product \( p \times T_p(n) \), where \( T_p(n) \) is the time complexity of a parallel algorithm using \( p \) processors. A parallel algorithm is termed optimal or work-optimal [24] if the work \( W(n) \) required by this algorithm satisfies \( W(n) = \Theta(T^*(n)) \), where \( T^*(n) \) is the running time of the fastest sequential

---

\*The correspondence address is : Prof. Shi-Jinn Horng, Department of Electrical Engineering, National Taiwan University of Science and Technology, 43, Section 4, Kee-Lung Road, Taipei, Taiwan, R. O. C. Tel: (02)2737-6700, FAX: (02)2737-6699, E-mail: horng@mouse.ee.ntust.edu.tw or yrwang@mail.fit.edu.tw. The work was partly supported by National Science Council under the contract no. NSC-89-2213-E011-007. Part of this work was carried out while the second author was visiting the Institute of Information Science, Academia Sinica, Taipei, Taiwan, during July 1, 2001 to September 30, 2001.
algorithm for the problem. A parallel algorithm is work-time (WT) optimal or optimal in the strong sense [24] if it is work-optimal and, in addition, its running time $T_p(n)$ cannot be improved by any other work-optimal parallel algorithm in that model. Therefore, the running time of a WT optimal algorithm represents the ultimate speed that can be achieved in that model.

Processing digital image data from three-dimensional (3-D, for short) objects is an important task in the image processing and the computer vision fields. All of those application areas such as medical image processing, industrial inspection, and robot guidance are known to be intractable and computationally intensive due to the complexity of the elementary operations and large amount of data to be processed. Developing efficient and powerful parallel algorithms for elementary operations is very important. The distance transform ($DT$, for short) first introduced by Rosenfeld and Pfaltz [37], [38] is one of the most important elementary operations and is extensively applied in the image processing area for extracting useful information from the received digital image. It is usually used for skeleton extraction, shape filtering [23], object expanding or shrinking [4], image matching, image compression [31], computer vision, 2-D or 3-D computer graphics [21], [42], thinning, and computing Voronoi diagram [5], [46]. See [9], for an overview of applications using the distance transform. A number of different $DT$s have been developed.

The Euclidean distance transform ($EDT$, for short) is based on the Euclidean metrics, whose application is better than those of $DT$ based on other metrics. Because the $EDT$ is a global operation, it is prohibitively time consuming when performs this operation on a binary image array, especially on a 3-D binary image array. In order to provide the efficient transform computations, parallelism should be employed.

Recently, many papers were introduced for constructing the Euclidean distance transform of a 2-D binary image of size $N \times N$. Hirata [22], and Breu et al. [11] presented $O(N^2)$ time sequential algorithms for computing the Euclidean distance transform. On the EREW PRAM model, Lee et al. [28] presented an $O(\log^2 N)$ time algorithm using $n^2$ processors; Pavel and Akl [34] presented an algorithm running in $O(\log N)$ time using $N^2$ processors. Clearly, these two algorithms are not work-optimal, but Chen [13] presented a work-optimal $O(N \log N)$ time algorithm using $\frac{N}{\log N}$ EREW processors. Fujiwara et al. [19] also presented a work-optimal algorithm running in $O\left(\frac{\log N}{\log \log N}\right)$ time using $\frac{N^2 \log \log N}{\log N}$ CRCW processors and a work-time optimal algorithm running in $O(\log N)$ time using $\frac{N^2}{\log N}$ EREW processors, respectively; Hayashi et al. [20] further presented a work-time optimal algorithm running in $O(\log \log N)$ time using $\frac{N^2}{\log \log N}$ CRCW processors. In recent years, there are many researches in the 3-D image processing based on the distance transform approach [6], [12], [36], [47]. Several sequential 3-D distance transform algorithms were introduced by [8], [10], [23]. Up to now, very few parallel algorithms are developed for the 3-D Euclidean distance transform [27].

The contribution of this paper is to develop work-time optimal parallel algorithms for the three-dimensional Euclidean distance transform ($3D_{EDT}$) on both the CRCW and EREW PRAM computation
models, each of size $N \times N \times N$. Based on the dimensionality reduction technique and the solution for 2-D proximate points problem, we achieve the optimality of the $3D_{\text{EDT}}$ computation. For an $N \times N \times N$ binary image array, our parallel algorithms can be run in $O(\log \log N)$ time using $\frac{N^3}{\log \log N}$ CRCW processors. Based on the simulation from the CRCW PRAM model to the EREW PRAM model, our parallel algorithms can be also implemented in $O(\log N)$ time using $\frac{N^3}{\log N}$ EREW processors. To build up the Euclidean distance transform, every voxel must be accessed at least once. Thus, $\Omega(N^3)$ work is required for any algorithms to solve the $3D_{\text{EDT}}$ operation. Obviously, our algorithms are work-optimal. Based on the fundamental results derived from [15], [20], [43], any algorithm that solves an instance of size $N$ of the proximate points problem must take $\Omega(\log \log N)$ time in the worst case if $N \log^{O(1)} N$ CRCW processors are available, or $\Omega(\log N)$ time on the EREW even if an infinite number of processors are available. Therefore, both of our CRCW and EREW algorithms proposed in this paper are work-time optimal. In this paper, we also apply the technique used for the $3D_{\text{EDT}}$ and proximate points problem to compute the $nD_{\text{EDT}}$ of a binary image array of size $N^n$. The $n$-dimensional Euclidean distance transform of a binary image of size $N^n$ can be computed in $O(n \log \log N)$ time using $\frac{N^n}{\log \log N}$ CRCW processors or in $O(n \log N)$ time using $\frac{N^n}{\log N}$ EREW processors. As for applications, our parallel $3D_{\text{EDT}}$ algorithm can be used to build up Voronoi diagram and Voronoi polyhedra, to find all maximal empty spheres and the largest empty sphere, and to compute the distance-based medial axis transform in a 3-D binary image. All of these parallel algorithms can be performed in $O(\log \log N)$ time using $\frac{N^3}{\log \log N}$ CRCW processors or in $O(\log N)$ time using $\frac{N^3}{\log N}$ EREW processors. Likewise, we can apply the algorithms for the $nD_{\text{EDT}}$ to solve the problems listed above in an $n$-D binary image. To the best of our knowledge, all results developed in this paper are the best results that never found before.

The remainder of this paper is organized as follows: In Section 2, we introduce the PRAM models and some notations upon which our algorithms are based. In Section 3, several important theorems, that are the essential concepts of our parallel algorithms, are derived. In Section 4, we describe our parallel algorithms named ALGORITHM $2D_{\text{EDT, CRCW}}$ and ALGORITHM $3D_{\text{EDT, CRCW}}$ in details. We also extend our parallel algorithms to $n$-dimensional space. In Section 5, we use the algorithms for $3D_{\text{EDT}}$ computation to find all maximal empty spheres and the largest empty sphere, and to compute the distance-based medial axis transform in a 3-D binary image. Finally, some concluding remarks are included in the last section.

2 Preliminaries and Notation

2.1 The PRAM Models

The Parallel Random Access Machine (PRAM, for short) consists of a number of identical processors, memory access unit, and a shared-memory. The shared-memory stores data and serves as the communication medium for the processors. At each step, every processor performs the same instruction, with
a number of processors marked out. The concurrent read concurrent write PRAM (CRCW, for short) and exclusive read exclusive write PRAM (EREW, for short) models which will be used in this paper for parallel computation are two types of the PRAM models. The CRCW PRAM can be further divided into a number of submodels, according to the rule used to resolve memory write conflicts, which occur when more than one processor attempt to write into the same memory location simultaneously. This gives rise to submodels such as the Common CRCW PRAM, the Priority CRCW PRAM, and so on. The Common CRCW PRAM processors are allowed to write in a memory location if they are attempting to write the same value. Because a single step execution of the N-processor CRCW can be simulated by an N-processor EREW in \(O(\log N)\) time, all of the algorithms developed on the CRCW model in this paper will be simulated on the EREW PRAM model with a slowdown of \(O(\log N)\) times. See [2], [24] for details.

Without loss of generality, throughout this paper, the Common CRCW PRAM submodel is used as our CRCW PRAM computational model and we only analyze the complexity of algorithms implemented on the CRCW PRAM model. The complexity analysis for the algorithm to be implemented on the EREW PRAM model can be discussed similarly.

**Lemma 1** [24] A single step execution of the N-processor CRCW can be simulated by an N-processor EREW in \(O(\log N)\) time.

### 2.2 Definitions

Figure 1 (a) illustrates the directions of axes in two dimensions \(X, Y\), and Figure 1 (b) depicts the directions of axes in three dimensions \(X, Y, Z\). Throughout this paper, a pixel (short for picture element) is used to represent a point in a 2-D plane and a voxel (short for volume element) is used to represent a point in a 3-D (or \(n\)-D) space. A pixel or a voxel is represented by its Cartesian coordinate. In a 2-D plane, we assume that a pixel \(p\) is represented by its Cartesian coordinate \((X_p, Y_p)\). The 2-D Euclidean distance between the planar pixels \(p\) and \(q\) is denoted by \(d_{p,q} = \sqrt{(X_p - X_q)^2 + (Y_p - Y_q)^2}\). In a 3-D space, we assume that a voxel \(p\) is represented by its Cartesian coordinate \((X_p, Y_p, Z_p)\). The 3-D Euclidean distance between the voxels \(p\) and \(q\) is denoted by \(d_{p,q} = \sqrt{(X_p - X_q)^2 + (Y_p - Y_q)^2 + (Z_p - Z_q)^2}\). In an \(n\)-D space, we assume that a voxel \(p\) is represented by its coordinate \((p_1, p_2, ..., p_n)\). The \(n\)-D Euclidean distance between the voxels \(p\) and \(q\) is denoted by \(d_{p,q} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}\). The Euclidean distance transform is an operation that converts an image consisting of black and white pixels (or voxels) to an image where each pixel (voxel) has a value that represents the Euclidean distance to its nearest 1-pixel. The Euclidean distance transforms in arbitrary dimensions are formally defined as follows:

**Definition 1** Let \(a_{i, j}\) denote a pixel of a two-dimensional binary image array with coordinate \((i, j)\). Let \(P = \{(i, j) \mid a_{i, j} = 0 \text{ or } 1, \ 1 \leq i, j \leq N\}\) denote a two-dimensional binary image array of size \(N \times N\). Let \(B_{2D} = \{(x, y) \mid a_{x, y} = 1, \ 1 \leq x, y \leq N\}\) represent the set of the 1-pixels of the 2-D binary image.
Then the two-dimensional Euclidean distance transform (2D_EDT) of pixel \( a_{i,j} \) with respect to its nearest 1-pixel \( a_{x,y} \) is denoted by \( \text{edt}_{i,j} = \min_{(x,y) \in \mathcal{B}_{2D}} \{ \sqrt{(i-x)^2 + (j-y)^2} \}, \) for all \( i, j = 1, 2, \ldots, N. \) □

**Definition 2** Let \( a_{i,j,k} \) denote a voxel of a three-dimensional binary image array with coordinate \((i, j, k)\). Let \( \mathcal{V} = \{(i, j, k) \mid a_{i,j,k} = 0 \text{ or } 1, \ 1 \leq i, j, k \leq N \} \) denote a three-dimensional binary image array of size \( N \times N \times N \). Let \( \mathcal{B}_{3D} = \{(x, y, z) \mid a_{x,y,z} = 1, \ 1 \leq x, y, z \leq N \} \) represent the set of the 1-voxels of the 3-D binary image. Then the three-dimensional Euclidean distance transform (3D_EDT) of voxel \( a_{i,j,k} \) with respect to its nearest 1-voxel \( a_{x,y,z} \) is denoted by \( \text{edt}_{i,j,k} = \min_{(x,y,z) \in \mathcal{B}_{3D}} \{ \sqrt{(i-x)^2 + (j-y)^2 + (k-z)^2} \}, \) for all \( i, j, k = 1, 2, \ldots, N. \) □

**Definition 3** A voxel in an n-dimensional space can be represented by its coordinate \((x_1, x_2, \ldots, x_n)\), where \( x_1, x_2, \ldots, x_n \) represent the value of each axis. Let \( p \) denote a voxel of an n-dimensional binary image array with coordinate \((p_1, p_2, \ldots, p_n)\). Let \( q \) denote a voxel of an n-dimensional binary image array with coordinate \((q_1, q_2, \ldots, q_n)\). Let \( \mathcal{H} = \{p \mid p = (p_1, p_2, \ldots, p_n) = 0 \text{ or } 1, \ 1 \leq p_1, p_2, \ldots, p_n \leq N \} \) denote an n-dimensional binary image array of size \( N^n \). Let \( \mathcal{B}_{nD} = \{q \mid q = (q_1, q_2, \ldots, q_n) = 1, \ 1 \leq q_1, q_2, \ldots, q_n \leq N \} \) represent the set of the 1-voxels of the n-D binary image. Then the n-dimensional Euclidean distance transform (nD_EDT) of voxel \( p \) with respect to its nearest 1-voxel \( q \) is denoted by \( \text{edt}_p = \min_{q \in \mathcal{B}_{nD}} \{ \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} \}, \) for all \( 1 \leq p_1, p_2, \ldots, p_n \leq N. \) □

**Definition 4** Let \( \mathcal{V} = \{(x, y, z) \mid 1 \leq x, y, z \leq N \} \) be a three-dimensional binary image array of size \( N \times N \times N \) voxels. A specified plane \( \Gamma_k \) as shown in Figure 8 can be represented as \( \{(x, y, k) \mid 1 \leq x, y \leq N, \ k \text{ is a fixed integer} \} \) in a 3-D coordinate system and a non-specified plane \( \Gamma \) can be simply represented as \( \mathcal{V} = \{ (x, y) \mid 1 \leq x, y \leq N \} \) in a 2-D coordinate system if \( k \) is well understood. Then, \( \mathcal{V} \) can be represented by \( \mathcal{V} = \{ \Gamma_k \mid 1 \leq k \leq N \} \) also. That is, \( \mathcal{V} \) consists of \( N \) planes with each plane consisting of a 2-D \( N \times N \) binary image array. These \( N \) planes are denoted by planes \( \Gamma_k, 1 \leq k \leq N \) (i.e., planes \( \Gamma_1, \Gamma_2, \ldots, \Gamma_N \)). We can generate a vertical column parallel to Z-axis by picking up a pixel with fixed X- and Y-coordinates \((x, y)\) from every \( \Gamma_k \) plane, \( 1 \leq k \leq N \). Because there are \( N \times N \) pixels on each \( \Gamma_k \) plane, so, we can generate total \( N \times N \) vertical columns, each of them with different combination of \( x, y \) values. For the sake of convenience, we denote each of these vertical columns as \( Z_{i,j} \)-column and each of these \( \Gamma_k \)
planes as Z-plane.

**Definition 5** Let \( \mathcal{H} = \{ (x_1, x_2, ..., x_n) \mid 1 \leq x_1, x_2, ..., x_n \leq N \} \) be an \( n \)-dimensional binary image array of size \( N^n \). A specified \((n-1)\)-dimensional binary image array \( U_k \) of size \( N^{n-1} \) can be represented as \( \{ (x_1, x_2, ..., x_{n-1}, k) \mid 1 \leq x_1, x_2, ..., x_{n-1} \leq N, \text{ where } k \text{ is a fixed integer} \} \) in an \( n \)-dimensional coordinate system and a non-specified \((n-1)\)-dimensional binary image array \( U \) can be simply represented as \( \{ (x_1, x_2, ..., x_{n-1}) \mid 1 \leq x_1, x_2, ..., x_{n-1} \leq N \} \) in an \((n-1)\)-dimensional coordinate system if \( k \) is well understood. Then, \( \mathcal{H} \) can be represented by \( \mathcal{H} = \{ U_k \mid 1 \leq k \leq N \} \) also. That is, \( \mathcal{H} \) can be decomposed into \( U_1, U_2, ..., U_k, ..., U_N \). We can generate a column (or line) parallel to \( X_n \)-axis by picking up a voxel with fixed \( x_1, x_2, ..., x_{n-1} \) coordinate values from every \( U_k \), \( 1 \leq k \leq N \). Because there are \( N^{n-1} \) voxels on each \( U_k \), so, we can generate total \( N^{n-1} \) vertical columns, each of them with distinct combination of \( x_1, x_2, ..., x_{n-1} \) values. For the sake of convenience, we denote each of these columns as \( X_{x_1, x_2, ..., x_{n-1}} \)-column.

**Definition 6** Let \( R = (x_R, y_R, z_R) \) be a voxel of \( V \), where \((x_R, y_R, z_R)\) is the coordinate of \( R \). Let \( N_{x_R}, y_R, z_R \) denote the nearest 1-voxel of the voxel \( R \) with respect to all 1-voxels in \( V \). Let \( N_{x_R}(\Gamma_k) \) or \( N_{x_R}(\Gamma_k), y_R, z_R(\Gamma_k) \), where \( 1 \leq k \leq N \), denote the nearest 1-voxel of the voxel \( R \) with respect to all 1-voxels in plane \( \Gamma_k \). Let \( (X_{N_{x_R}(\Gamma_k)}, Y_{N_{x_R}(\Gamma_k)}, Z_{N_{x_R}(\Gamma_k)}) \) denote the coordinate of \( N_{x_R}(\Gamma_k) \). Clearly, \( Z_{N_{x_R}(\Gamma_k)} = k \).

**Definition 7** Given a sequence \( b_1, b_2, ..., b_N \) of zeros and ones, to determine the closest 1-pixel to every pixel in this sequence is defined as the 1-D nearest-one problem. Let \( P = \{ P_k \mid 1 \leq k \leq N \} \) be a set of \( N \) pixels in the plane sorted by the coordinate of \( X \)-axis. Without loss of generality, we assume that all pixels in \( P \) have distinct \( X \)-coordinates. A point \( P_k \), where \( 1 \leq k \leq N \), is a proximate point of \( P \) if there exists
Figure 3: An illustration of the set of proximate points $\mathcal{P}_L = \{P_1, P_3, P_4, P_5, P_6\}$.

Figure 4: An illustration of the set of proximate points $\mathcal{P}_R = \{P_7, P_8, P_9, P_{12}\}$. 
Figure 5: Illustrating the contact points between two sets of proximate points \( \mathcal{P}_L \cup \mathcal{P}_R \).

A point on the X-axis closer to \( P_k \) than to any other points in \( \mathcal{P} \). The proximate points problem is to determine all proximate points in \( \mathcal{P} \). Let’s define the proximate points problem of size \( N \) with all these \( N \) points located in a 2-D space to be the 2-D proximate points problem. For example, in Figure 2, there are twelve pixels in \( \mathcal{P} \) are distributed in a 2-D space and \( P_1, P_3, P_4, P_9 \), and \( P_{12} \) are proximate points of \( \mathcal{P} \). If the proximate points problem of size \( N \) with all these \( N \) points is located in a 3-D space, then we define this problem to be the 3-D proximate points problem. If the proximate points problem of size \( N \) with all these \( N \) points located in an \( n \)-D space, then we define this problem to be the \( n \)-D proximate points problem.

For every point \( P_i \) of \( \mathcal{P} \), the locus of all the points in the plane that are closer to \( P_i \) than to any other points in \( \mathcal{P} \) is defined to be the Voronoi cell of \( P_i \) and is denoted by \( \text{Vor}(i) \). The collection of all the Voronoi cells of points in \( \mathcal{P} \) partitions the plane into the Voronoi diagram of \( \mathcal{P} \). The Voronoi diagram of \( \mathcal{P} \) partitions the X-axis into proximate intervals. Let us represent X-axis as \( \{ Q'_k \mid Q'_k = (k, 0), \text{where } 1 \leq k \leq N \} \). Then the proximate interval \( I_i \) can be represented as \( \{ Q'_k \mid Q'_k \text{ is closer to } P_i \text{ than to any other points in } \mathcal{P}, \text{where } 1 \leq k \leq N \} \). That is, the proximate interval \( I_i \) is the locus of all the points \( Q'_k \) if \( P_i \) is a proximate point.

For example, in Figure 6, any one of intervals \( I_1, I_3, I_4, I_9, I_{12} \) is a proximate interval. A point \( Q'_k \) is termed as a boundary point if two proximate intervals \( I_i \) and \( I_j \) are adjacent and intersected at point \( Q'_k \). Clearly, point \( Q'_k \) is equidistant to proximate points \( P_i \) and \( P_j \).

The following example is a briefly description for finding proximate points. Assume that there are a set of twelve pixels as shown in Figure 2. Finding the proximate points out of these twelve pixels can be computed by using divide and conquer algorithm as follows: First, we divide these twelve pixels into two sets as shown in Figure 3 and Figure 4. Then, divide the set \( \{P_1, P_2, P_3, P_4, P_5, P_6\} \) in Figure
3 into \( \{P_1, P_2, P_3\} \) and \( \{P_4, P_5, P_6\} \), and divide the set \( \{P_7, P_8, P_9, P_{10}, P_{11}, P_{12}\} \) as shown in Figure 4 into \( \{P_7, P_8, P_9\} \) and \( \{P_{10}, P_{11}, P_{12}\} \). Now let’s focus on the set \( \{P_1, P_2, P_3\} \) in Figure 3. We draw a perpendicular bisector for line segment \( \overline{P_1P_2} \) and assume that this perpendicular bisector intersects X-axis at coordinate \((X_{P_1}, 0)\). We also draw a perpendicular bisector for line segment \( \overline{P_2P_3} \) and assume that this perpendicular bisector intersects X-axis at coordinate \((X_{P_2}, 0)\). The relations among X-coordinates of \( P_1, P_2, P_3 \) are \( X_{P_1} < X_{P_2} < X_{P_3} \). Then, if point \((X_{P_2}, 0)\) is at the left hand side of point \((X_{P_1}, P_2)\), that is, \( X_{P_2} < X_{P_1} \), we say that pixel \( P_2 \) is dominated by its left hand side neighbor \( P_1 \) and right hand side neighbor \( P_3 \). All the sets \( \{P_1, P_2, P_3\}, \{P_4, P_5, P_6\} \) and \( \{P_{10}, P_{11}, P_{12}\} \) will be processed in the same way as described above. Clearly, \( P_{11} \) is dominated by its left hand side neighbor \( P_{10} \) and right hand side neighbor \( P_{12} \). Any pixel dominated by its both side neighbors will cease to be a proximate point. Next, we merge the sets \( \{P_1, P_3\} \) and \( \{P_4, P_5, P_6\} \) as shown in Figure 3 and apply the same procedure as stated above to get the set of proximate points \( \mathcal{P}_L = \{P_1, P_3, P_4, P_5, P_6\} \). Also, we merge the sets \( \{P_7, P_8, P_9\} \) and \( \{P_{10}, P_{12}\} \) as shown in Figure 4 and get the set of proximate points \( \mathcal{P}_R = \{P_7, P_8, P_9, P_{12}\} \). Clearly, \( P_{10} \) is dominated by its left hand side neighbor \( P_9 \) and right hand side neighbor \( P_{12} \) during this merge procedure. Finally, we merge \( \mathcal{P}_L = \{P_1, P_3, P_4, P_5, P_6\} \) and \( \mathcal{P}_R = \{P_7, P_8, P_9, P_{12}\} \) as shown in Figure 5 and get the set of proximate points \( \mathcal{P}_L \cup \mathcal{P}_R = \{P_7, P_8, P_9, P_{12}\} \) as shown in Figure 6, with \( P_1 = P_4 \) and \( P_j = P_9 \). Here \( P_i \) and \( P_j \) are the contact points between \( \mathcal{P}_L \) and \( \mathcal{P}_R \). Clearly, no pixels in \( \mathcal{P}_L \) to the right of \( P_4 \) can be the proximate points of \( \mathcal{P}_L \cup \mathcal{P}_R \). Also, no pixels in \( \mathcal{P}_R \) to the left of \( P_9 \) can be the proximate points of \( \mathcal{P}_L \cup \mathcal{P}_R \). Each one of the survived proximate points owns its corresponding proximate interval. For example, in Figure 6, any one of intervals \( I_1, I_3, I_4, I_9, I_{12} \) is a proximate interval.

Figure 6: Illustrating the proximate points of \( \mathcal{P}_L \cup \mathcal{P}_R = \{P_1, P_3, P_4, P_9, P_{12}\} \).
3 Essential Concepts

**Theorem 1** [27] Let $Q = (x_Q, y_Q, z_2)$ be a voxel with coordinate $(x_Q, y_Q, z_2)$ located at plane $\Gamma_{z_2}$. Let $\mathcal{N}_Q(\Gamma_{z_2}) = (x_{\mathcal{N}_Q}(\Gamma_{z_2}), y_{\mathcal{N}_Q}(\Gamma_{z_2}), z_2)$, where $\mathcal{N}_Q(\Gamma_{z_2}) \in \Gamma_{z_2}$, be the nearest 1-voxel of the voxel $Q$ at plane $\Gamma_{z_2}$. Let $R$ be anyone 1-voxel other than $\mathcal{N}_Q(\Gamma_{z_2})$ at plane $\Gamma_{z_2}$. Let $P$ be a voxel with coordinate $(x_Q, y_Q, z_1)$ located at plane $\Gamma_{z_1}$. That is, $P$ and $Q$ are located in the same Z-axis. Then, $\mathcal{N}_P(\Gamma_{z_2}) = \mathcal{N}_Q(\Gamma_{z_2})$. That is, the nearest 1-voxel at plane $\Gamma_{z_2}$ of the voxel $P$ is the same as the nearest 1-voxel at plane $\Gamma_{z_2}$ of the voxel $Q$.

**Proof:** See Figure 7 for the following description. Assume $|QN_Q(\Gamma_{z_2})| = a$, $|QR| = b$, $|PQ| = c$. Because $\mathcal{N}_Q(\Gamma_{z_2})$ is the nearest 1-voxel of $Q$ at plane $\Gamma_{z_2}$, so $|QN_Q(\Gamma_{z_2})| < |QR|$: that is, $a < b$. Because $PQ \perp QN_Q(\Gamma_{z_2})$ and $PQ \perp QR$, so $|PN_Q(\Gamma_{z_2})| = \sqrt{a^2 + c^2} < |PR| = \sqrt{b^2 + c^2}$. Thus, the nearest 1-voxel at plane $\Gamma_{z_2}$ of $P$ is $\mathcal{N}_Q(\Gamma_{z_2})$, i.e., $\mathcal{N}_P(\Gamma_{z_2}) = \mathcal{N}_Q(\Gamma_{z_2})$. The conclusion implies $\mathcal{N}_Q(\Gamma_{z_1}) = \mathcal{N}_P(\Gamma_{z_1})$ also.

**Theorem 2** [27] Let $Q_k = (X_{Q_k}, Y_{Q_k}, k)$ be a sequence of voxels located at plane $\Gamma_k$, $1 \leq k \leq N$, with each of them in the same $Z_{i,j}$-column which is parallel to Z-axis as defined in Definition 4. That is, there are $Q_1$, $Q_2$, ..., $Q_N$ voxels, each of them located at planes $\Gamma_1$, $\Gamma_2$, ..., $\Gamma_N$ respectively, with $X_{Q_1} = X_{Q_2} = X_{Q_3} = X_{Q_N}$ and $Y_{Q_1} = Y_{Q_2} = Y_{Q_3} = Y_{Q_N}$, where $1 \leq k \leq N$. Let $\mathcal{N}_{Q_k}$ be the nearest 1-voxel of voxel $Q_k$ in the 3-D space. Then $\mathcal{N}_{Q_k} \in \{\mathcal{N}_Q(\Gamma_i) \mid 1 \leq k, r \leq N\}$. It means that the nearest 1-voxel of $Q_k$ must be one of $\mathcal{N}_Q(\Gamma_1), \mathcal{N}_Q(\Gamma_2), ... \mathcal{N}_Q(\Gamma_N)$.

**Proof:** See Figure 8 for the illustration of this theorem. According to Theorem 1, we get $\mathcal{N}_{Q_1}(\Gamma_2) = \mathcal{N}_{Q_2}(\Gamma_2) = \mathcal{N}_{Q_3}(\Gamma_2) = ... = \mathcal{N}_{Q_N}(\Gamma_2)$. Therefore, $\mathcal{N}_{Q_k}(\Gamma_2)$ is the same for any $Q_k$ where $1 \leq k \leq N$.
The coordinate of \( Q_k = (X_{Q_k}, Y_{Q_k}, k) \)
The coordinate of \( N_{Q_k}(\Gamma_k) = (X_{N_{Q_k}(r_k)}, Y_{N_{Q_k}(r_k)}, k) \)

This is also sustained for all \( Q_k \). Then \( N_{Q_k} \in \{ N_{Q_r}(\Gamma_r) | 1 \leq k, r \leq N \} \) is proved. \( \square \)

**Theorem 3** Let \( Q_k = (X_{Q_k}, Y_{Q_k}, k) \) be a sequence of voxels each located at plane \( \Gamma_k \), 1 \( \leq k \leq N \), with all of the voxels in the same \( Z_{i,j} \)-column which is parallel to \( Z \)-axis as defined in Definition 4. That is, there are \( Q_1, Q_2, ..., Q_N \) voxels, each of them located at planes \( \Gamma_1, \Gamma_2, ..., \Gamma_N \) respectively, with \( X_{Q_1} = X_{Q_2} = X_{Q_k} = X_{Q_N} \) and \( Y_{Q_1} = Y_{Q_2} = Y_{Q_k} = Y_{Q_N} \), where \( 1 \leq k \leq N \). Suppose the nearest 1-voxel \( N_{Q_k} \) of voxel \( Q_k \) is located at plane \( \Gamma_j \) in the 3-D space. Instead of finding the nearest 1-voxel of voxel \( Q_k \) from the original 3-D space, first map the voxel \( Q_k \) and its nearest 1-voxel \( N_{Q_k} \) to the pixel \( Q_k' \) with coordinate \((k, 0)\) and the pixel \( P_j' \) with coordinate \((j, (Q_jN_{Q_j}(\Gamma_j)))\) respectively of a 2-D space. Then based on the mapping, the nearest 1-voxel of voxel \( Q_k \) located in the 3-D space can be found as the pixel \( P_j' \) located in the newly created 2-D space. That is, for decreasing the computational complexity of the 3-D nearest 1-voxel problem, we can downgrade the 3-D coordinate system as shown in Figure 8 to a 2-D coordinate system as shown in Figure 9.

**Proof:** Assume each voxel of a 3-D space is \( Q_k \) with coordinate \((X_{Q_k}, Y_{Q_k}, k)\), and the nearest 1-voxel of \( Q_k \) is at plane \( \Gamma_j \) as shown in Figure 8. According to Theorem 1, \( N_{Q_k} = N_{Q_k}(\Gamma_j) = N_{Q_j}(\Gamma_j) = N_{Q_j}(\Gamma_j) = \ldots = N_{Q_1}(\Gamma_k) = N_{Q_k}(\Gamma_k) \).
(X_{NQ_j}(r_j'), Y_{NQ_j}(r_j'), j). Assume d_{3D} is the distance between Q_k and \( N_{Q_k} \), then \( d_{3D}^2 = |Q_kN_{Q_k}|^2 = (X_{Q_k} - X_{NQ_j}(r_j))^2 + (Y_{Q_k} - Y_{NQ_j}(r_j))^2 + (k-j)^2 \). After Figure 8 is converted into Figure 9, the voxel \( Q_k \) is mapped into the pixel \( P_k' \) with coordinate \((k, 0)\), the voxel \( N_{Q_k} \) is mapped into the pixel \( P_j' \) with coordinate \((j, |Q_jN_{Q_j}(r_j)|)\), where \(|Q_jN_{Q_j}(r_j)| = \sqrt{(X_{Q_j} - X_{NQ_j}(r_j))^2 + (Y_{Q_j} - Y_{NQ_j}(r_j))^2}\). Assume the distance between these two mapped pixels \( P_k' \) and \( P_j' \) on the 2-D plane is \( d_{2D} \), then \( d_{2D}^2 = |Q_jN_{Q_j}(r_j)|^2 = (X_{Q_k} - X_{NQ_j}(r_j))^2 + (Y_{Q_k} - Y_{NQ_j}(r_j))^2 + (k-j)^2 \). By definition, \( Q_k \) and \( Q_j \) are located at the same \( Z_{i,j} \)-column, which is parallel to \( Z \)-axis, we get \( X_{Q_k} = X_{Q_j} \) and \( Y_{Q_k} = Y_{Q_j} \). So, \( d_{3D} = d_{2D} \). □

Following Theorem 3, this leads to the following corollary.

**Corollary 1** The 3-D Euclidean distance transform problem of a binary image of size \( N \times N \times N \) can be decomposed into \( N \) planes of 2-D Euclidean distance transform problem each consisting of a binary image of size \( N \times N \). After the nearest 1-voxel of each voxel \((i, j, k)\) is found in each independent plane \( \Gamma_k \), the problem of finding the nearest 1-voxel of the voxel located in each \( Z_{i,j} \)-column is reduced to the 2-D proximate points problem of size \( N \). □

The above theorem and corollary can be extended to an \( n \)-dimensional space as follows:

**Theorem 4** Let \( Q_k = (x_{1Q_k}, x_{2Q_k},..., x_{n-1Q_k}, k) \) be a sequence of voxels each located at \((n-1)\)-dimensional space \( U_k \), \( 1 \leq k \leq N \), with all of the voxels in the same \( X_{x_1,x_2,...,x_{n-1}} \)-column which is parallel to \( X_n \)-axis as defined in Definition 5. That is, there are \( Q_1, Q_2,..., Q_N \) voxels, each of them located at the \((n-1)\)-dimensional spaces \( U_1, U_2,..., U_N \) respectively, with \( x_{1Q_1} = x_{1Q_2} = x_{1Q_N}, x_{2Q_1} = x_{2Q_2} = x_{2Q_k} = x_{2Q_N},..., x_{n-1Q_1} = x_{n-1Q_2} = x_{n-1Q_k} = x_{n-1Q_N}, 1 \leq k \leq N \). Suppose the nearest 1-voxel \( N_{Q_k} \) of voxel \( Q_k \) is located at the \((n-1)\)-dimensional space \( U_j \) with coordinate \( N_{Q_k} = (x_{1N_{Q_k}}, x_{2N_{Q_k}},..., x_{n-1N_{Q_k}}, j) \) in the \( n \)-dimensional space. Instead of finding the nearest 1-voxel of voxel \( Q_k \) from the original \( n \)-D space, first map the voxel \( Q_k \) and its nearest 1-voxel \( N_{Q_k} \) to the pixel \( P_k' \) with coordinate \((k, 0)\) and the pixel \( P_j' \) with coordinate \((j, |Q_jN_{Q_j}(U_j)|)\) respectively of a 2-D space. Then based on the mapping, the nearest 1-voxel of voxel \( Q_k \) located in the \( n \)-D space can be found as the pixel \( P_j' \) located in the newly created 2-D space. That is, for decreasing the computational complexity of the \( n \)-D nearest 1-voxel problem, we can downgrade the \( n \)-dimensional coordinate system to a 2-D coordinate system.

**Proof:** See Theorem 3. □

**Corollary 2** The \( n \)-dimensional Euclidean distance transform problem of a binary image of size \( N^n \) can be decomposed into \( N \) hyperplanes of \((n-1)\)-dimensional Euclidean distance transform problem each consisting of a binary image of size \( N^{n-1} \). After the nearest 1-voxel \( N_{Q_k}(U_k) \) of each voxel \( Q_k = (x_1, x_2,..., x_{n-1}, k) \) is found in each independent \((n-1)\)-dimensional space \( U_k \), the problem of finding the nearest 1-voxel \( N_{Q_k} \) of the voxel \( Q_k \) located in each \( X_{x_1,x_2,...,x_{n-1}} \)-column, which is parallel to \( X_n \)-axis, is reduced to the 2-D proximate points problem of size \( N \). □

12
4 Optimal Parallel Algorithms for 3D_EDT

4.1 The Sketch of Algorithm

The idea of our algorithm is based on the dimensionality reduction technique. That is, instead of computing the 3D_EDT from a 3-D space directly, we compute it from the 1-D space, then the 2-D space, finally the 3-D space in sequence. As stated before, the 1D_EDT problem is reduced to the 1-D nearest-one problem. The high level description of the proposed algorithm consists of the following three phases. First, suppose we compute the 1D_EDT for the pixels along the Y-axis for plane \( \Gamma_k \). Then, after this phase, for plane \( \Gamma_k \) (i.e., the same Z-coordinate) and column \( i \) (i.e., the same X-coordinate) every voxel \((i, j, k)\), \(1 \leq i, j, k \leq N\), is assigned a coordinate \((i, N_j, k)\) where \( N_j \) is the Y-coordinate of the 1-D nearest 1-voxel of voxel \((i, j, k)\) located at the same plane \( \Gamma_k \) with the same column \( i \). If no 1-voxel is in \{\((i, 1, k), (i, 2, k), ..., (i, N, k)\}\), let \( N_j = +\infty \). Next, we compute the 2D_EDT for the pixel along the X-axis for plane \( \Gamma_k \). The 2D_EDT problem for row \( j \) is reduced to the 2-D proximate points problem as defined in Definition 7. Based on the intermediate result as derived in the previous phase and Definition 7, after this phase, for plane \( \Gamma_k \), every voxel \( Q_i = (i, j, k) \) located in row \( j \) finds the nearest 1-voxel \( N_{Q_i} \) with coordinate \((X_{N_{Q_i}}, Y_{N_{Q_i}}, k)\) (or assigned a Euclidean distance \( \text{edt}_{i, j, k} = \sqrt{(i - X_{N_{Q_i}})^2 + (j - Y_{N_{Q_i}})^2} \)). Finally, we compute the 3D_EDT. By Theorem 3 and Corollary 1, the 3D_EDT problem for a \( Z_{i,j} \)-column is reduced to the 2-D proximate points problem again. As stated in the previous phase, every voxel \( Q_k = (i, j, k), 1 \leq i, j, k \leq N, \) located in a \( Z_{i,j} \)-column got the information about its nearest 1-voxel \( N_{Q_k}(\Gamma_k) \) located in its original plane \( \Gamma_k \). Based on the intermediate result, Theorem 3 and Definition 7, after this phase, every voxel \( Q_k = (i, j, k) \) located in a \( Z_{i,j} \)-column finds the nearest 1-voxel \( N_{Q_k} \) with coordinate \((X_{N_{Q_k}}, Y_{N_{Q_k}}, Z_{N_{Q_k}})\) (or assigned a Euclidean distance \( \text{edt}_{i, j, k} = \sqrt{(i - X_{N_{Q_k}})^2 + (j - Y_{N_{Q_k}})^2 + (k - Z_{N_{Q_k}})^2} \)). The following is the detailed description of our
algorithms.

4.2 Detailed Description of Algorithm

According to the definition of Definition 1, a 2-D binary image array of size $N \times N$ is represented by $a_{i, j} = 0$ or $1$, for $i, j = 1, 2, \ldots, N$, and $(1, 1)$ is assumed to be the bottom-left corner pixel of the image. Clearly, there are $N$ rows in this 2-D binary image array; in row $j$, there are total $N$ pixels with coordinates $(1, j)$, $(2, j)$, ..., $(N, j)$. As defined before, $B_{2D} = \{(x, y) \mid a_x, y = 1, 1 \leq x, y \leq N\}$ is used to represent the set of the 1-pixels of the binary image. For every pixel in row $j$, we can determine its nearest 1-pixel in column $i$. That is, for pixel $a_{i, j}$, we find its nearest 1-pixel from the pixels with coordinates $(i, 1)$, $(i, 2)$, ..., $(i, N)$. This becomes a 1-D nearest-one problem as defined in Definition 7.

Based on Lemma 2 as derived by JáJá and Vishkin, we can solve the 1-D nearest-one problem and it is then stated in Lemma 3.

**Lemma 2** [24], [41] The task of computing the prefix maxima (prefix minima) of an $N$-item sequence can be performed in $O(\log \log N)$ time using $N \frac{\log \log N}{\log N}$ CRCW processors or in $O(\log N)$ time using $N \frac{\log N}{\log \log N}$ EREW processors.

**Lemma 3** The 1-D nearest-one problem of size $N$ can be solved in $O(\log \log N)$ time using $N \frac{\log \log N}{\log N}$ CRCW processors or in $O(\log N)$ time using $N \frac{\log N}{\log \log N}$ EREW processors.

Let $d_j$ denote the distance between a pixel $Q_i = (i, j)$ and its 1-D nearest-one pixel $N_{Q_i} = (i, N_j)$ located at the same plane $\Gamma$ with the same column $i$, then $d_j = |j - N_j|$. If no 1-pixel is in $\{(i, 1), (i, 2), \ldots, (i, N)\}$, let $N_j = +\infty$. For column $i$ of a 2-D image plane, after we solve the 1-D nearest-one problem for this column, every pixel located at this column has gotten the distance to its 1-D nearest-one pixel located at the same column. There are $N$ columns for a $2-D N \times N$ binary image array. After we apply Lemma 3 for all the $N$ columns in parallel, every pixel located at a column of the 2-D image plane has computed the distance to its 1-D nearest-one pixel located at the same column. For plane $\Gamma$, by mapping the pixel $Q_i$ located at $(i, j)$ and its corresponding 1-D nearest-one pixel $N_{Q_i}$ located at $(i, N_j)$ to the pixels $Q_i'$ located at $(i, 0)$ and $P_i'$ located at $(i, d_j)$ of the newly created coordinate system respectively, the $2D_{EDT}$ problem is reduced to the proximate points problem with $P_j = \{P_i' = (i, d_j) \mid 1 \leq i \leq N\}$ as defined in Definition 7. The proximate points problem can be solved by Theorem 5 [20]. Corollary 3 is a direct result from Theorems 3, 4, Corollaries 1, 2 and Theorem 5.

**Theorem 5** [20] The 2-D proximate points problem of size $N$ can be solved in $O(\log \log N)$ time using $N \frac{\log \log N}{\log N}$ CRCW processors or in $O(\log N)$ time using $N \frac{\log N}{\log \log N}$ EREW processors.

**Corollary 3** For each $Z_{i,j}$-column, $1 \leq i, j \leq N$, of the 3-D proximate points problem of size $N$ in the 3-D space, it can be solved in $O(\log \log N)$ time using $N \frac{\log \log N}{\log N}$ CRCW processors or in $O(\log N)$ time using...
For each $X_{x_1,x_2,...,x_{n-1}}$-column, $1 \leq x_1,x_2,...,x_{n-1} \leq N$, of the n-D proximate points problem of size $N$ in the n-D space, it can be solved in $O(\log \log N)$ time using $\frac{N}{\log \log N}$ CRCW processors or in $O(\log N)$ time using $\frac{N}{\log N}$ EREW processors also.

Having solved the 2-D proximate points problem in parallel, we can determine how many pixels of $\mathcal{P}_j$ stated above are proximate points and to which proximate interval $I_i$ the pixel $Q'_i$ is belonged. Now we have gotten the coordinate of the 2-D nearest 1-pixel for every pixel of row $j$ on the 2-D image plane. The same procedure will be executed in parallel for every row $j$, $1 \leq j \leq N$, to find the 2-D nearest 1-pixel for every pixel $(i,j)$ of row $j$; therefore, two-dimensional Euclidean distance transform can be computed. The parallel algorithm for computing $2D_{EDT}$ is stated as follows:

**ALGORITHM $2D_{EDT_{CRCW}}$**

**Input:** A 2-D $N \times N$ binary image array, each pixel being represented by $a_{i,j} = 0$ or 1, for $1 \leq i, j \leq N$.

**Output:** A 2-D $N \times N$ array, each pixel $(i,j)$ being represented by $(X_{N_i,j}, Y_{N_i,j})$, for $1 \leq i, j \leq N$, or $\text{edt}_{i,j}$, the Euclidean distance between the pixel $(i,j)$ and its nearest 1-pixel $N_{i,j}$.

1. **for** $i := 1$ **to** $N$ **pardo**
   
   2. Apply Lemma 3 to solve the 1-D nearest-one problem for every column $i$ of a 2-D $N \times N$ binary image array.
   
3. **end**;

4. **for** $j := 1$ **to** $N$ **pardo**
   
5. Construct the set $\mathcal{P}_j$ of pixels for every row $j$, where $\mathcal{P}_j = \{P'_i = (i,d_j) \mid 1 \leq i \leq N\}$, $1 \leq j \leq N$, for the proximate points problem computation.

6. Apply Theorem 5 to solve the proximate points problem, for every row $j$ in parallel.

7. **end**;

The parallel algorithm for computing $3D_{EDT}$ consists of two major phases: the **plane phase** and the **vertical phase**. During the plane phase (i.e., 2-D phase), for each plane (i.e., Z-plane) $\Gamma_k$, $1 \leq k \leq N$, we find the nearest 1-pixel for each pixel in the plane. Clearly, there are total $N$ independent planes each of size $N \times N$. This is a $2D_{EDT}$ problem which can be implemented by **ALGORITHM $2D_{EDT_{CRCW}}$**. At the end of the plane phase, each pixel in plane $\Gamma_k$ has the coordinate of its nearest 1-pixel over the entire plane or the Euclidean distance between the pixel $(i,j)$ and its nearest 1-pixel $N_{i,j}$. Let $Q_k = (i,j,k)$, $1 \leq i, j, k \leq N$, be a voxel with coordinate $(i,j,k)$ located at plane $\Gamma_k$, and $\mathcal{N}_{Q_k}(\Gamma_k) = (x, y, k) \in \Gamma_k$.
be the nearest 1-voxel of the voxel \( Q_k \) at plane \( \Gamma_k \) as shown in Figure 8. Then the distance from \( Q_k \) to its nearest 1-voxel \( N_{Q_k}(\Gamma_k) \) in the same \( \Gamma_k \) plane is \( \sqrt{(i-x)^2 + (j-y)^2} \). A 3-D binary image array \( V \) is defined according to Definition 3. That is, \( V \) consists of \( N \) planes each consisting of a 2-D \( N \times N \) binary image array. Therefore, a voxel with coordinate \((x, y, k)\) can be represented by a pixel with coordinate \((x, y)\) on \( \Gamma_k \) plane.

Now, in the vertical phase (i.e., 3-D phase), we first generate a vertical \( Z_{i,j} \)-column, which is parallel to \( Z \)-axis, by picking up a pixel with fixed \( X \)- and \( Y \)-coordinates from every \( \Gamma_k \) plane. Because there are \( N \times N \) pixels on each \( \Gamma_k \) plane, we can construct total \( N \times N \) vertical columns, each of them with different combination of \( x, y \) values. We then use the \( Z_{i,j} \)-column to find out where the nearest 1-voxel of each voxel is located. In Theorem 2, we have proved that for every voxel \( Q_k = (X_{Q_k}, Y_{Q_k}, k) \), the nearest 1-voxel of voxel \( Q_k \) must be one of \( N_{Q_1}(\Gamma_1), N_{Q_2}(\Gamma_2), \ldots, N_{Q_N}(\Gamma_N) \). In Theorem 3, we have proved that for computing the 3-D nearest 1-voxel of voxel \( Q_k \), the voxel \( Q_k \) and voxel \( N_{Q_k}(\Gamma_j) \) in the 3-D space as shown in Figure 8 can be mapped into the pixel \( Q'_k \) on X-axis with coordinate \((k, 0)\) and the pixel \( P'_j \) with coordinate \((j, |Q_j|N_{Q_j}(\Gamma_j))\) in the 2-D plane as shown in Figure 9. Based on Corollary 1, the 3-D nearest 1-voxel problem for a \( Z_{i,j} \)-column is reduced to the proximate points problem. All \( N \times N \) \( Z_{i,j} \)-columns will be processed in parallel. Then, apply Theorem 5 to solve the proximate points problem for all \( N \times N \) \( Z_{i,j} \)-columns in parallel. After this step, we have gotten the coordinate of the nearest 1-voxel of every voxel on the 3-D space. Assume that the coordinate of the nearest 1-voxel of the voxel \( a_{i, j, k} \) is \((x, y, z)\), then the distance from this voxel to its nearest 1-voxel is \( \sqrt{(i-x)^2 + (j-y)^2 + (k-z)^2} \). The parallel algorithm for computing 3D_EDT is stated as follows:

**ALGORITHM 3D_EDT_CRCW**

**Input:** A 3-D \( N \times N \times N \) binary image array, each voxel being represented by \( a_{i, j, k} = 0 \) or 1, for \( 1 \leq i, j, k \leq N \).

**Output:** A 3-D \( N \times N \times N \) array, each voxel \( a_{i, j, k} \) being represented by \((X_{N_{i, j, k}}, Y_{N_{i, j, k}}, Z_{N_{i, j, k}})\) (i.e., the coordinate of \( N_{i, j, k} \)) or \( \text{edt}_{i, j, k} \) (i.e., the Euclidean distance between the voxel \( a_{i, j, k} \) and its nearest 1-voxel \( N_{i, j, k} \)), for \( 1 \leq i, j, k \leq N \).

1.0 The Plane Phase

1.1 for \( k := 1 \) to \( N \) pardo

1.2 Apply ALGORITHM 2D_EDT_CRCW on each Z-plane \( \Gamma_k \), \( 1 \leq k \leq N \).

1.3 end;

2.0 The Vertical Phase
2.1 for \( i, j := 1 \) to \( N \) pardo

2.2 Construct all \( Z_{i,j} \)-columns, \( 1 \leq i, j \leq N \) in parallel.

\{ By picking up a pixel with fixed \( X \)- and \( Y \)-coordinates from every \( \Gamma_k \) plane, there are total \( N \times N \) \( Z_{i,j} \)-columns to be constructed, each of them with different combination of \( x, y \) values. \}

2.3 Map each voxel on \( Z_{i,j} \)-column together with its nearest 1-voxel \( N_{Q_k}(\Gamma_k) \) from the original 3-D coordinate system to the newly created 2-D coordinate system.

\{ Based on Theorem 3, map the voxel \( Q_k \) with coordinate \( (i, j, k) \) and its nearest 1-voxel \( N_{Q_k} \) in the 3-D space as shown in Figure 8 to the pixel \( Q'_k \) with coordinate \( (k, 0) \) and the pixel \( P'_k \) with coordinate \( (k, |Q_k N_{Q_k}(\Gamma_k)|) \) in the 2-D plane as shown in Figure 9. \}

2.4 Apply Theorem 5 to solve the proximate points problem for all \( N \times N \) \( Z_{i,j} \)-columns in parallel.

Assume the coordinate of the nearest 1-voxel of the voxel \( a_{i, j, k} \) is \( (x, y, z) \), then the Euclidean distance from this voxel to its nearest 1-voxel is \( \sqrt{(i-x)^2 + (j-y)^2 + (k-z)^2} \).

2.5 end;

4.3 Time Complexity Analysis

The complexity analysis of ALGORITHM 2D.EDT_CRCW is stated as follows: We apply Lemma 3 in step 2 to solve the 1-D nearest-one problem for every column of a 2-D \( N \times N \) binary image array in parallel. This can be performed in \( O(\log \log N) \) time using \( \frac{N^2}{\log \log N} \) CRCW processors. Step 5 constructs the set \( P_j \) of pixels for each row \( j \), where \( P_j = \{ P'_i = (i, d_j) | 1 \leq i \leq N \} \). Any pixel in the set \( P_j \) is the candidate of the proximate points. Step 6 applies Theorem 5 to solve the proximate points problem for every row in parallel. Both step 5 and step 6 can be performed in \( O(\log \log N) \) time using \( \frac{N}{\log \log N} \) CRCW processors. Therefore, processing all \( N \) rows in parallel for step 4 will take \( O(\log \log N) \) time using \( \frac{N^2}{\log \log N} \) CRCW processors. Following the previous discussion, we have the following result.

**Theorem 6** [20] The Euclidean distance transform of a binary image of size \( N \times N \) can be computed in \( O(\log \log N) \) time using \( \frac{N^2}{\log \log N} \) CRCW processors or in \( O(\log N) \) time using \( \frac{N^2}{\log N} \) EREW processors.

By Definition 7, for every pixel \( (i, j), 1 \leq i, j \leq N \), of a 2-D binary image array \( P \), the union of all the pixels in the plane that are closer to pixel \( (i, j) \) than to any other pixels in \( P \) is defined to be the Voronoi polygons of \( (i, j) \) and is denoted by \( \text{Vor}(i, j) \). That is, if we assign every pixel \( (i, j) \) the coordinate \( (X_{N_{i,j}}, Y_{N_{i,j}}) \) of \( N_{i,j} \), instead of the Euclidean distance \( \text{edt}_{i,j} \) to its nearest 1-pixel, then \( \text{Vor}(i, j) \) becomes
the set of all pixels assigned the coordinate \((i, j)\) and the output of ALGORITHM 2D \(_{\text{EDT,CRCW}}\) turns out to be the set of \(\{\text{Vor}(i, j) \mid 1 \leq i, j \leq N\}\). This concludes the following corollary.

**Corollary 4** The Voronoi polygons and Voronoi diagram of a binary image of size \(N \times N\) can be constructed in \(O(\log \log N)\) time using \(\frac{N^2}{\log \log N}\) CRCW processors or in \(O(\log N)\) time using \(\frac{N^2}{\log N}\) EREW processors. □

We then analyze the complexity of ALGORITHM 3D \(_{\text{EDT,CRCW}}\). In the plane phase, we apply ALGORITHM 2D \(_{\text{EDT,CRCW}}\) on each \(Z\)-plane \(\Gamma_k, 1 \leq k \leq N\), in parallel. Because all \(N\) planes are processed in parallel, it can be performed in \(O(\log \log N)\) time using \(\frac{N^2}{\log \log N}\) CRCW processors by Theorem 6. The complexity of the vertical phase is analyzed in the following. Step 2.2 constructs all \(N \times N\) \(Z_{i,j}\)-columns in parallel in \(O(1)\) time using \(N \times N\) CRCW processors. Step 2.3 maps each voxel on a \(Z_{i,j}\)-column together with its nearest 1-voxel from a 3-D coordinate system to a 2-D coordinate system in \(O(\log \log N)\) time using \(\frac{N}{\log \log N}\) CRCW processors. It will take \(O(\log \log N)\) time using \(\frac{N^3}{\log \log N}\) CRCW processors for processing these \(N \times N\) \(Z_{i,j}\)-columns in parallel. From Theorem 5, an instance of the proximate points problem of size \(N\) can be solved in \(O(\log \log N)\) time using \(\frac{N}{\log \log N}\) CRCW processors. Therefore, step 2.3 applies Theorem 5 to solve the proximate points problem for all \(N \times N\) \(Z_{i,j}\)-columns in parallel and it can be performed in \(O(\log \log N)\) time using \(\frac{N^3}{\log \log N}\) CRCW processors. Because both the plane phase and vertical phase of ALGORITHM 3D \(_{\text{EDT,CRCW}}\) can be performed in \(O(\log \log N)\) time using \(\frac{N^3}{\log \log N}\) CRCW processors, we get the following theorem and corollary.

**Theorem 7** The Euclidean distance transform of a binary image of size \(N \times N \times N\) can be computed in \(O(\log \log N)\) time using \(\frac{N^3}{\log \log N}\) CRCW processors or in \(O(\log N)\) time using \(\frac{N^3}{\log N}\) EREW processors. □

**Corollary 5** The Voronoi polyhedra and Voronoi diagram of a binary image of size \(N \times N \times N\) can be constructed in \(O(\log \log N)\) time using \(\frac{N^3}{\log \log N}\) CRCW processors or in \(O(\log N)\) time using \(\frac{N^3}{\log N}\) EREW processors. □

The dimensionality reduction technique and the solution for proximate points problem can be also utilized to compute the \(nD_{\text{EDT}}\) of a binary image array of size \(N^n\). By applying Corollary 2, the \(n\)-dimensional Euclidean distance transform problem of a binary image array of size \(N^n\) can be decomposed into \(N\) hyperplanes of \((n-1)\)-dimensional Euclidean distance transform problem each consisting of a binary image of size \(N^{n-1}\). After the nearest 1-voxel of each voxel \((x_1, x_2, ..., x_{n-1}, k)\) is found in each independent \((n-1)\)-dimensional space \(U_k\), the problem of finding the nearest 1-voxel of the voxel located in each \(X_{x_1,x_2,...,x_{n-1}}\)-column can be reduced to the 2-D proximate points problem of size \(N\). Totally, there are \(N^{n-1}\) pillars of \(X_{x_1,x_2,...,x_{n-1}}\)-column.

Like the 3D \(_{\text{EDT}}\) problem, the \(nD_{\text{EDT}}\) computation can be computed from the \(1-D\) space, then the \(2-D\) space, then the \(3-D\) space, ..., finally the \(n-D\) space in sequence. Basically, it takes \(O(\log \log N)\) time using
loglogN CRCW processors for the 1D_EDT computation. Suppose it takes $O((i-1)\log\log N)$ time using $N^{(i-1)}_\loglogN$ CRCW processors for the $(i-1)D_EDT$ computation. We will prove that it takes $O(i\log\log N)$ time using $N^{i}_\loglogN$ CRCW processors for the $iD_EDT$ computation. By Corollary 2, the $iD_EDT$ problem for a binary image of size $N^i$ can be decomposed into $N^i$ hyperplanes of $(i-1)D_EDT$ problem each consisting of a binary image of size $N^{(i-1)}$. After the nearest 1-voxel $N_{Q_k}(U_k)$ of each voxel $Q_k = (x_1, x_2, ..., x_{i-1}, k)$ is found in each independent $(i-1)$-dimensional space $U_k$, the problem of finding the nearest 1-voxel $N_{Q_k}$ of the voxel $Q_k$ located in each $X_{x_1, x_2, ..., x_{i-1}}$-column is reduced to the 2-D approximate points problem of size $N$. By Corollary 3, it takes $O(\log\log N)$ time using $N^{i}_\loglogN$ CRCW processors for a column computation. It then takes $O(i\log\log N)$ time using $N^{i}_\loglogN$ CRCW processors for $N^{(i-1)}$ columns computation. Therefore, it takes $O(i\log\log N)$ time using $N^{i}_\loglogN$ CRCW processors for the $iD_EDT$ computation. We skip the details for the analysis of the EREW PRAM model. This leads to the following theorem.

**Theorem 8** The $n$-dimensional Euclidean distance transform of a binary image of size $N^n$ can be computed in $O(n\log\log N)$ time using $N^{n}_\loglogN$ CRCW processors or in $O(n\log N)$ time using $N^{n}_\loglogN$ EREW processors.

\[\square\]

5 Applications

As described in Section 1, the Euclidean distance transform is a global operation. It is one of the most important elementary operations. Our parallel algorithms for EDT computation can be used in all the application areas such as medical image processing, industrial inspection, and robot guidance where the EDT is used as their elementary operations.

5.1 All Maximal Empty Spheres and The Largest Empty Sphere

Here we apply our ALGORITHM 3D_ EDT_CRCW in the following to find all maximal empty spheres and the largest empty sphere in a 3-D $N \times N \times N$ binary image array $V$. An empty sphere is a sphere whose interior contains only 0-voxel. A maximal empty sphere is an empty sphere that contains in no other empty sphere. The all maximal empty spheres problem is to find all the maximal empty spheres in a 3-D binary image. The largest empty sphere problem is to find the largest empty sphere from all the maximal empty spheres.

**ALGORITHM All_Maximal_Empty_Spheres**

**Input:** A 3-D $N \times N \times N$ binary image array $V$, each voxel being represented by $a_i, j, k = 0$ or 1, for $1 \leq i, j, k \leq N$.

**Output:** Each voxel $(i, j, k)$ is represented by its labelled $r_{i, j, k}$ to represent that there exists a maximal empty sphere with radius $r_{i, j, k}$ and origin $(i, j, k)$. The largest empty sphere of all the maximal empty
spheres of a binary image of size $N \times N \times N$ is represented by radius $\max r_{i,j,k}$ and origin $(i, j, k)$.

1.1 for $i, j, k := 1$ to $N$ pardo

1.2 Apply ALGORITHM 3D_EDT_CRCW to compute the 3D_EDT of $V$.

{ After this step, each voxel is assigned the coordinate of the nearest 1-voxel $N_{i, j, k}$ for
$1 \leq i, j, k \leq N$, or $edt_{i,j,k}$, the Euclidean distance between the voxel $(i, j, k)$ and its nearest
1-voxel $N_{i, j, k}$. }

1.3 end;

2.1 for $i, j, k := 1$ to $N$ pardo

2.2 Compute $r_{i,j,k} = \min\{i - 1, j - 1, k - 1, N - i, N - j, N - k, \text{edt}_{i,j,k}\}$ for every voxel
$(i, j, k)$.

{ It means that the largest radius of an empty sphere centered at every voxel $(i, j, k)$ is $r_{i,j,k}$. }

2.3 end;

3.1 for $i, j, k := 1$ to $N$ pardo

3.2 Check whether there exists a neighboring voxel $(i', j', k')$ such that the sphere with radius
$r_{i,j,k}$ and origin $(i, j, k)$ is covered by the sphere with radius $r_{i',j',k'}$ and origin $(i', j', k')$.

If no such sphere exists, label the sphere with radius $r_{i,j,k}$ and origin $(i, j, k)$ as a maximal
empty sphere.

{ A voxel with coordinate $(i, j, k)$ has 26 neighboring voxels with coordinate $(i', j', k')$, where
$|i - i'| \leq 1$ and $|j - j'| \leq 1$ and $|k - k'| \leq 1$. }

3.3 end;

4.1 for $i, j, k := 1$ to $N$ pardo

4.2 Compute $\max r_{i,j,k} = \max \{ r_{i,j,k} \mid \text{where } r_{i,j,k} \text{ is the radius of the labelled voxel } (i, j, k)$
computed in step 3\}. Then the voxel $(i, j, k)$ with the largest radius $\max r_{i,j,k}$ is the
largest empty sphere in $V$.

4.3 end;
Now we analyze the complexity of ALGORITHM All_Maximal_Empty_Spheres. Step 1 can be performed in $O(\log \log N)$ time using $\frac{N^3}{\log \log N}$ CRCW processors. For both step 2 and step 3, we first partition the $N^3$ voxels of $V$ into $\log \log N$ subsets, each subset containing $\frac{N^3}{\log \log N}$ voxels. Then, assign $\frac{N^3}{\log \log N}$ CRCW processors to a subset with each processor assigned to a voxel $(i, j, k)$ to compute $r_{i,j,k}$ by finding the minimum value from the set $\{i-1, j-1, k-1, N-i, N-j, N-k, \text{edt}_{i,j,k}\}$ in parallel for step 2 and to check if radius $r_{i,j,k}$ is the maximum among its 26-neighbors in parallel for step 3. For each processor, we use the optimal sequential algorithm for finding the minimum (or maximum) described in [24]. In the worst case, for each processor it takes $O(\log \log 7) = O(1)$ time to compute the $r_{i,j,k}$ and $O(\log \log 26) = O(1)$ time to check if the $r_{i,j,k}$ is the maximum. Sequentially applying the procedures described above for all $\log \log N$ subsets takes $O(\log \log N)$ time. Clearly, both steps 2 and 3 can be performed in $O(\log \log N)$ time using $\frac{N^3}{\log \log N}$ CRCW processors. In step 4, we will find the maximum of all the labelled $r_{i,j,k}$ computed in step 3. In the worst case, the number of the labelled $r_{i,j,k}$ could be $O(N^3)$. It can be solved in $O(\log \log N)$ time using $\frac{N^3}{\log \log N}$ CRCW processors by applying Lemma 4 described in [24]. This concludes the following theorem.

**Lemma 4** [24] Finding the maximum of $N$ elements can be done optimally in $O(\log \log N)$ time using $\frac{N}{\log \log N}$ CRCW processors.

**Theorem 9** The task of labelling all the maximal empty spheres and the largest empty sphere of all the maximal empty spheres of a binary image of size $N \times N \times N$ can be performed in $O(\log \log N)$ time using $\frac{N^3}{\log \log N}$ CRCW processors or in $O(\log N)$ time using $\frac{N^3}{\log \log N}$ EREW processors.

### 5.2 The Distance-Based Medial Axis Transform

Another application is to compute the medial axis transform (MAT, for short) introduced by Blum [7] in parallel in a 3-D $N \times N \times N$ binary image array $V$. The MAT is a powerful shape descriptor that has good properties for data reduction and allows a full reconstruction of the original shape. There are two types of medial axis transforms [18]. One is the distance-based MAT, which is defined as a recovering of the object by maximal digital disks in a 2-D plane or maximal digital spheres in a 3-D space included in the object. A maximal digital disk (or sphere) is a digital disk (or sphere) that contains in no other digital disk (or sphere). The other is the block-based MAT, which is a recovering of the object by maximal square blocks of pixels in a 2-D plane or maximal cube blocks of voxels in a 3-D space. A maximal square (or cube) block is a square (or cube) block that contains in no other square (or cube) block. Figure 10 illustrates the distance-based MAT in a 2-D plane. Throughout the paper, the MAT will be defined as the distance-based MAT and we will focus on the parallel computation of the distance-based MAT in a 3-D space. If every voxel of the medial axis of an object is labelled with the radius of its corresponding maximal sphere, then the object can be exactly rebuilt using the information of its axis [40].
Figure 10: An illustration of the distance-based medial axis transform in a 2-D plane.

**ALGORITHM** \texttt{DB\_MAT\_CRCW}

**Input:** A 3-D $N \times N \times N$ binary image array $V$, each voxel being represented by $a_{i,j,k} = 0$ or 1, for $1 \leq i, j, k \leq N$.

**Output:** Each voxel $(i, j, k)$ of the medial axis of an object is labelled with the radius $r_{i,j,k}$ of its corresponding maximal sphere.

1.1 for $i, j, k := 1$ to $N$ pardo

1.2 Complement the value of every voxel $(i, j, k)$ of $V$.

\{ That is, a 1-voxel (i.e., $a_{i,j,k} = 1$) is changed to 0-voxel (i.e., $a_{i,j,k} = 0$), and vice versa. We denote the complemented 3-D $N \times N \times N$ binary image array as $V'$ \}

1.3 end;

2.1 for $i, j, k := 1$ to $N$ pardo

2.2 Apply ALGORITHM \texttt{All\_Maximal\_Empty\_Spheres} to find all maximal empty spheres for every voxel in the 3-D $N \times N \times N$ binary image array $V'$.

\{ After this step, each voxel $(i, j, k)$ of $V'$ is assigned a maximal empty sphere with radius $r_{i,j,k}$ and origin $(i, j, k)$. The radius $r_{i,j,k}$ of the maximal empty sphere of each voxel $(i, j, k)$ of $V'$ is exactly the radius of the maximal sphere of the voxel on the medial axis of an object in $V$. \}

2.3 end;

Clearly, both step 1 and step 2 can be performed in $O(\log \log N)$ time using $\frac{N^3}{\log \log N}$ CRCW processors. This concludes the following theorem.
**Theorem 10** The task of computing the distance-based medial axis transform of a binary image of size $N \times N \times N$ can be performed in $O(\log \log N)$ time using $\frac{N^3}{\log \log N}$ CRCW processors or in $O(\log N)$ time using $\frac{N^3}{\log N}$ EREW processors.

6 Conclusions

The main contribution of this paper is that we reduce the 3D\_EDT problem to the proximate points problem. That is, instead of computing the 3D\_EDT from the original 3-D space directly, we compute it by the proximate points computation. Based on this reduction, the result shows that the 3D\_EDT for an image of size $N \times N \times N$ can be computed in $O(\log \log N)$ time using $\frac{N^3}{\log \log N}$ CRCW processors and in $O(\log N)$ time using $\frac{N^3}{\log N}$ EREW processors respectively. To the best of our knowledge, both algorithms are work-time optimal and are the best results that never found before. We then extend the dimensionality reduction technique and the solution for proximate points problem to compute the $nD\_EDT$ of a binary image array of size $N^n$. The n-dimensional Euclidean distance transform of a binary image of size $N^n$ can be computed in $O(n \log \log N)$ time using $\frac{N^n}{\log \log N}$ CRCW processors or in $O(n \log N)$ time using $\frac{N^n}{\log N}$ EREW processors. Our algorithms developed for the 3D\_EDT are also applied to build up Voronoi diagram and Voronoi polyhedra, to find all maximal empty spheres and the largest empty sphere, and to compute the distance-based medial axis transform in a 3-D binary image. All of these parallel algorithms can be also performed in $O(\log \log N)$ time using $\frac{N^3}{\log \log N}$ CRCW processors or in $O(\log N)$ time using $\frac{N^3}{\log N}$ EREW processors. Obviously, our algorithms for $nD\_EDT$ can be also applied to build up Voronoi diagram, to find all maximal empty n-dimensional spheres and the largest empty n-dimensional sphere, and to compute the distance-based medial axis transform in an n-dimensional binary image. Interestingly, the algorithms are conceptually simple and of practical as well.

References


[38] A. Rosenfeld and J. L. Pfalz, Distance function on digital pictures, Pattern Recognition, vol. 1, 33-61 (1968).


