Comput. & Graphics Vol. 16, No. 3, pp. 289–294, 1992
Printed in Great Britain.

0097-8493/92 $5.00 + .00

Technical Section

A PARALLEL ALGORITHM FOR FINDING CONGRUENT REGIONS*

CHIN-LAUNG LEI and HORNG-TWU LIAW
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan 106, R.O.C.

Abstract—In this paper, we study the problem for finding all the regions, which are congruent to a testing region R, in an input planar figure F. In a shared memory system with m processors, we propose an efficient MAX$\{O(mn), O(n \log n)\}$ time parallel algorithm, where n, m are the numbers of edges of F and R, respectively. Furthermore, our algorithm does not require to read from or write into the same memory location simultaneously, hence it can be implemented on an exclusive-read, exclusive-write (EREW) model.

1. INTRODUCTION

Finding the congruent regions among geometric objects is a popular topic in computational geometry. In general, this problem arises in pattern recognition, computer vision, etc. Recently, some researchers have devoted themselves to investigating this problem [1–4].

Roughly speaking, two planar regions R and S are congruent if there exists a mapping, including a proper geometrical translation and/or rotation, which makes R onto S. A formal definition will be given in Section 2. In [3], they defined the congruent regions finding problem as follows: Given a planar figure F and a testing region R, determine whether R is congruent to any region of F, and then find all of them if they indeed exist. Figure 1 shows an example of this finding problem. Only the shadow regions bounded by edges (v_0, v_7), (v_7, v_8), (v_8, v_9), (v_9, v_0) and bounded by (v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1) are congruent to the testing region R. The value labeled with each edge represents the length of that edge.

The investigation of VLSI technology has made progress in parallel operation that reveals a high degree of parallelism in multiprocessor systems. Basically, there are two different architectural models for multiprocessor systems. One of them is a tightly-coupled system where communication is through a shared memory. Thus, we also say that this system is a shared-memory multiprocessor system. The other one is a loosely-coupled system where communication is done via an interconnection network, that is, this is a message-passing multiprocessor system.

In a shared-memory parallel system, each processor can read from or write into any memory location, depending on whether concurrent read from or concurrent write into a memory is allowed or not. Therefore, a shared-memory parallel system can be further divided into the following four models:

1. Exclusive-Read, Exclusive-Write (EREW) model. No two processors are allowed to read from or write into the same memory location simultaneously.
2. Concurrent-Read, Exclusive-Write (CREW) model. Processors are allowed to read from the same memory location, but no two processors are allowed to write into the same memory location simultaneously.
3. Exclusive-Read, Concurrent-Write (ERCW) model. Processors are allowed to write into the same memory location but no two processors are allowed to read from the same memory location simultaneously.
4. Concurrent-Read, Concurrent-Write (CRCW) model. Multiple processors are allowed to read from and write into the same memory location simultaneously.

Among the schemes[1–4] mentioned above, only the method proposed in [3] adopts a parallel approach to solve this problem. In this paper, we propose a method rather in EREW model than in CREW model to find the congruent regions. Our algorithm requires only MAX$\{O(mn), O(n \log n)\}$ computation time, where n, m are the numbers of edges of the input planar figure F and the testing region R, respectively. Furthermore, Shih, Lee, and Yang's[3] results may contain some repetitive congruent regions. In our algorithm, we have solved this problem. The rest of this paper is organized as follows: In Section 2, some essential definitions and notations of the geometric objects are described. In Section 3, we propose an efficient parallel algorithm for finding the congruent regions and analyze the time complexity. Finally, concluding remarks are given in Section 4.

2. PRELIMINARIES

Before we embark on our study of an efficient parallel algorithm for finding congruent regions based on a shared-memory system, we first give some definitions, notations, and properties of the geometric objects.

A graph $G = (V, E)$ consists of a set V of elements called vertices and a set E of unordered pairs of members of V called edges. The vertices of the graph are shown as points, while the edges are shown as lines connecting pairs of points. The edge between the pair of vertices v_a and v_b is denoted by (v_a, v_b). Here, we call the vertices v_a and v_b the endpoints of the edge (v_a, v_b), and we say the edge (v_a, v_b) is incident with the vertices v_a and v_b. In addition, if an edge does not

* This work was supported in part by the National Science Council of the Republic of China under Grant NSC 81-0416-E-002-20.

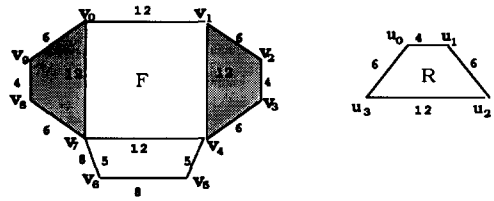

Fig. 1. The input figure F and the testing region R.

exhibit a direction, it is called an undirected edge. Thus, an undirected edge $e = (v_a, v_b)$ can also be represented by (v_b, v_a), that is, $(v_a, v_b) = (v_b, v_a)$. On the other hand, a directed edge has only one direction. Let $\vec{e} = \langle v_a, v_b \rangle$ be a directed edge, where v_a is the tail vertex and v_b is the head vertex. Therefore, $\langle v_a, v_b \rangle \neq \langle v_b, v_a \rangle$. A path of directed edges is a sequence of directed edges that are connected with their endpoints. Without loss of generality, let $P = [\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n]$ denote a path of directed edges, where $\vec{e}_1 = \langle v_1, v_2 \rangle$, $\vec{e}_2 = \langle v_2, v_3 \rangle, \ldots, \vec{e}_n = \langle v_n, v_{n+1} \rangle$. Furthermore, a directed cycle is a path of directed edges $P = [\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_m]$, where $\vec{e}_1 = \langle v_1, v_2 \rangle$, $\vec{e}_2 = \langle v_2, v_3 \rangle, \ldots, \vec{e}_m = \langle v_m, v_1 \rangle$, and $v_i \neq v_j$ when $i \neq j$. Besides, a region consists of a directed cycle and the interior of directed cycle, which is the right-hand side when we walk along the directed edge clockwisely. Here, we renewedly define the angle of each directed edge. For simplicity, let $\vec{e} = \langle v_1, v_2 \rangle$. First, we translate this directed edge and let the tail vertex v_1 be at the Cartesian coordinate origin, that is, the tail vertex v_1 is regarded as a Cartesian coordinate origin. The angle between x-axis and $\vec{e}$ (or simply the angle of $\vec{e}$) is defined as the angle lying to the right of us when we walk along the positive x-axis to the tail of $\vec{e}$. In this paper, we use $A(\vec{e})$ to denote it. Figure 2 shows the edge $\vec{e} = \langle v_1, v_2 \rangle$ and the angle $A(\vec{e})$. In addition, the angle between two adjacent edges $\vec{e}_1 = \langle v_1, v_2 \rangle$ and $\vec{e}_2 = \langle v_2, v_3 \rangle$, denoted by $A(\vec{e}_1, \vec{e}_2)$, is defined as the angle lying to the right of us when we walk along $\vec{e}_1$ to the tail of $\vec{e}_2$. Furthermore, let $\vec{e}'_1 = \langle v_2, v_1 \rangle$ be the opposite direction of $\vec{e}_1$. Thus, $A(\vec{e}_1)$ and $A(\vec{e}'_1)$ differs by 180°, that is, $A(\vec{e}'_1) = A(\vec{e}_1) - 180$ or $A(\vec{e}'_1) = A(\vec{e}_1) + 180°$. Without loss of generality, we assume that any angle is nonnegative and less than 360°. Thus, $A(\vec{e}'_1) = (A(\vec{e}_1) - 180) \bmod 360°$. $A(\vec{e}_1, \vec{e}_2)$, the angle between $\vec{e}_1$ and $\vec{e}_2$, can be obtained by the following theorem.

Theorem 1. Let $\vec{e}_1 = \langle v_1, v_2 \rangle$, and $\vec{e}_2 = \langle v_2, v_3 \rangle$, then $A(\vec{e}_1, \vec{e}_2) = (A(\vec{e}_2) - A(\vec{e}'_1)) \bmod 360°$.

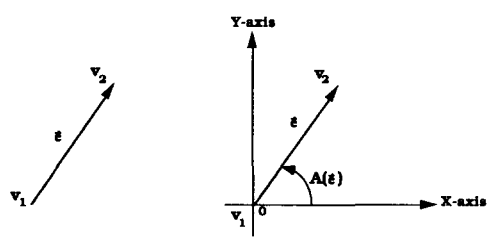

Fig. 2(a). The directed edge $\vec{e} = \langle v_1, v_2 \rangle$; (b) The angle $A(\vec{e})$ of the directed edge $\vec{e}$.

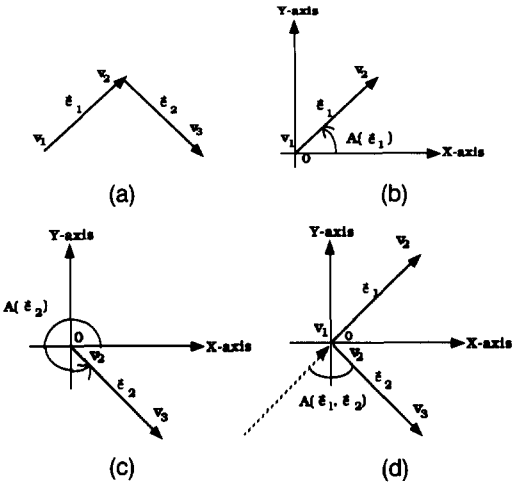

Fig. 3(a). The adjacent edges $\vec{e}_1$ and $\vec{e}_2$; (b) the angle $A(\vec{e}_1)$ of the directed edge $\vec{e}_1$; (c) The angle $A(\vec{e}_2)$ of the directed edge $\vec{e}_2$; (d) the angle $A(\vec{e}_1, \vec{e}_2)$ between the adjacent edges $\vec{e}_1$ and $\vec{e}_2$.

Proof. Since $\vec{e}_1$ and $\vec{e}_2$ intersect at the vertex v_2, the angle between them is equal to the angle between $\vec{e}'_1$ and $\vec{e}_2$, that is, $A(\vec{e}_1, \vec{e}_2) = A(\vec{e}_2) - A(\vec{e}'_1)$. From our assumption, any angle must be nonnegative and less than 360°. Hence, $A(\vec{e}_1, \vec{e}_2) = (A(\vec{e}_2) - A(\vec{e}'_1)) \bmod 360°$. □

Theorem 2. If $A(\vec{e}_1, \vec{e}_2)$ and $A(\vec{e}_1)$ are given, then $A(\vec{e}_2) = (A(\vec{e}'1) + A(\vec{e}_1, \vec{e}_2)) \bmod 360°$.

Proof. From theorem 1, $A(\vec{e}_1, \vec{e}_2) = (A(\vec{e}_2) - A(\vec{e}'_1)) \bmod 360°$. Thus, $A(\vec{e}_2) - A(\vec{e}'_1) = A(\vec{e}_1, \vec{e}_2) + 360k$, for some integer k. That is, $A(\vec{e}_2) = A(\vec{e}'_1) + A(\vec{e}_1, \vec{e}_2) + 360k$. Since $A(\vec{e}_2)$ is also nonnegative and less than 360°, $A(\vec{e}_2) = (A(\vec{e}'_1) + (A(\vec{e}_1, \vec{e}_2)) \bmod 360°$. □

Figure 3 shows the adjacent edges $\vec{e}_1 = \langle v_1, v_2 \rangle$, $\vec{e}_2 = \langle v_2, v_3 \rangle$ and the angles of $A(\vec{e}_1)$, $A(\vec{e}_2)$, $A(\vec{e}_1, \vec{e}_2)$. Furthermore, we use $|\vec{e}|$ to represent the length of any directed edge $\vec{e}$ in this paper.

Let $\vec{e}_a$, $\vec{e}_b$, $\vec{e}_x$, and $\vec{e}_y$ be four directed edges and the head vertices of $\vec{e}_a$ and $\vec{e}_x$ be the tail vertices of $\vec{e}_b$ and e_y, respectively. If there exists a condition such that $|\vec{e}_a| = |\vec{e}_x|$, $|\vec{e}_b| = |\vec{e}_y|$ and $A(\vec{e}_a, \vec{e}_b) = A(\vec{e}_x, \vec{e}_y)$, then we say that $\vec{e}_a$ is connected forwardly to $\vec{e}_b$ with respect to $\vec{e}_x$ and $\vec{e}_y$. Besides, if $\vec{e}_a$ is connected forwardly to $\vec{e}_b$, then we also say that $\vec{e}_b$ is connected backwardly to $\vec{e}_a$. Furthermore, if two regions $A = [\vec{e}_{A_0}, \vec{e}_{A_1}, \ldots, \vec{e}_{A_{m-1}}]$ and $B = [\vec{e}_{B_0}, \vec{e}_{B_1}, \ldots, \vec{e}_{B_{m-1}}]$ are congruent, then there must exist some j such that $\vec{e}_{A_{(i \bmod m)}}$ is connected forwardly to $\vec{e}_{A_{((i+1) \bmod m)}}$ with respect to $\vec{e}_{B_{((i+j) \bmod m)}}$ and $\vec{e}_{B_{((i+j+1) \bmod m)}}$, where $0 \leq i, j \leq m - 1$.

In this paper, our approach is inspired by the method found in [3] but improves upon it. In the next section, we present an efficient parallel algorithm for finding all the congruent regions. In [3], they have shown that a path of directed edges $[\vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{m-1}]$ with $|\vec{e}_i| = |\vec{t}_i|$, $0 \leq i \leq m - 1$, and $A(\vec{e}_j, \vec{e}_{j+1}) = A(\vec{t}_j,$

$\vec{t}_{j+1}$), $0 \le j \le m-2$, defines a region that is congruent to a testing region $[\vec{t}_0, \vec{t}_1, \ldots, \vec{t}_{m-1}]$. Thus, the kernel part of our proposed algorithm is to judge whether the specified candidate region from the input planar figure F and the testing region R possess the same edges and angles.

3. AN EFFICIENT PARALLEL ALGORITHM

We are now ready to propose a parallel algorithm on an EREW shared-memory computer. The algorithm presented as algorithm Parallel-Congruent makes the following assumptions. Throughout this paper, we use m processing elements (PEs) to execute our algorithm, where m is the number of edges of testing region R. Initially, we duplicate any edge of the input planar figure $F = \{e_0, e_1, \ldots, e_{n-1}\}$ into two directed edges with opposite direction. Thus, we get a new set $D = \{\vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{2n-1}\}$. In addition, the edge $\vec{t}_i$ of the testing region $R = [\vec{t}_0, \vec{t}_1, \ldots, \vec{t}_{m-1}]$ is stored into PE_i, where $i = 0, 1, \ldots, m-1$. After some steps of our algorithm, there are m sets of directed edges $D_0, D_1, \ldots, D_{m-1}$ stored in shared memory, where $D_j = \{\vec{e}_i : |\vec{e}_i| = |\vec{t}_j|, 0 \le i \le 2n-1\}, 0 \le j \le m-1$. Furthermore, each element of D_j is denoted as $D_j[k]$, $1 \le k \le n_j$, where n_j is the number of edges in D_j and each $D_j[k]$ contains six fields EDGE, TAIL, HEAD, ANGLE, FLINK, and BLINK. The fields TAIL and HEAD store the tail vertex and head vertex of the edge, respectively. The field ANGLE stores the angle of the edge in D_j. The field FLINK is pointed to D_{j+1}, BLINK is pointed to D_{j-1}. If $D_j[k]$.FLINK and $D_j[k]$.BLINK are pointed to an element of D_{j+1} and D_{j-1}, respectively, then $D_j[k]$.EDGE can be forwardly and backwardly connected to $D_{j+1}[D_j[k]$.FLINK$]$.EDGE and $D_{j-1}[D_j[k]$.BLINK$]$.EDGE, respectively. In the following algorithm, we need to sort the order of a pair sequence. For convenience, we define the order of any two pairs as follows: $(a, b) < (c, d)$ if and only if $a < c$ or ($a = c$ and $b < d$). Without loss of generality, we assume that m is even and $D_j[k]$.FLINK and $D_j[k]$.BLINK are null initially. The algorithm, described in what follows as algorithm Parallel-Congruent proceeds in stages.

Algorithm Parallel-Congruent

Input: The set of edges $E = \{e_0, \ldots, e_{n-1}\}$ of the input planar figure F and the cycle of directed edges $[\vec{t}_0, \ldots, \vec{t}_{m-1}]$ of the testing region R.

Output: All the regions of planar figure F that are congruent to the testing region R.

Step 1: Duplicate the set of edges $E = \{e_0, e_1, \ldots, e_{n-1}\}$ of F to the set of directed edges $D = \{\vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{2n-1}\}$.

Step 2: Load $\vec{t}_j$ into the processing element PE_j, where j=0, 1, ..., $m-1$.

Step 3: Calculate the values $FA_i = A(\vec{t}_i, \vec{t}_{(i+1) \bmod m})$ and store it into PE_i, where $i = 0, 1, \ldots, m-1$.

Step 4: Load the set of directed edges $D = \{\vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{2n-1}\}$ into the processing elements by the following operations.
For $i = 0$ to $2n-1$ do
For $j = 0$ to $m-1$ do in parallel
$k_i = (j+i) \bmod 2n$
$\vec{e}_{k_j}$ enters PE_j
If $|\vec{e}_{k_j}| = |\vec{t}_j|$ then PE_j copies $\vec{e}_{k_j}$ into the set D_j
end
end

Step 5: For $j = 0$ to $m-1$ do in parallel
Calculate the relative angle of each edge in D_j and let the value of the angle be $A_j(\vec{e}_{k_j})$, $0 \le k_j \le 2n-1$.
end

Step 6: For $j = 0$ to $m-1$ do in parallel
Sort the pairs $(v_{\vec{e}_{k_j}}, A_j(\vec{e}_{k_j}))$, $0 \le k_j \le 2n-1$, where $v_{\vec{e}_{k_j}}$ is the tail vertex of edge $\vec{e}_{k_j}$ in D_j.
end

Step 7: For $j = 0, 2, 4, \ldots, m-2$ do in parallel
(7.1) For $k = 1$ to n_j do // n_j is the number of edges in D_j //
$A_j(\vec{e}_{k_j}') = (A_j(\vec{e}_{k_j}) - 180°) \bmod 380°$
$A_{j+1}(\vec{e}_{p_{j+1}}) = (FA_j + A_j(\vec{e}_{k_j}')) \bmod 360°$ // $0 \le k_j$, $p_{j+1} \le 2n-1$ //
Let this edge $\vec{e}_{k_j}$ be pointed by L and use binary search to find the pair $(v, A_{j+1}(\vec{e}_{p_{j+1}}))$ in D_{j+1}, where v is both the head vertex of edge $\vec{e}_{k_j}$ in D_j and the tail vertex of certain edge in D_{j+1}. Let R be a pointer to this found edge in D_{j+1}. If such R exists, then $D_j[k]$.FLINK $\leftarrow R$ and $D_{j+1}[D_j[k]$.FLINK$]$.BLINK $\leftarrow L$.
If such R does not exist, then $D_j[k]$.EDGE $\leftarrow 0$.
end
end

Step 8: For $j = 1, 3, 5, \ldots, m-3$ do in parallel
Perform step (7.1) of Step 7.

Step 9: All processing elements perform this step iteratively until there is no more edge in each D_j, $0 \le j \le m-1$, which can be deleted in one iteration.
(9.1) For j = 0, 2, 4, ..., $m-2$ do (a) and (b) in parallel
(a) For $k = 1$ to n_j do
If $D_{j+1}[D_j[k]$.FLINK$]$.EDGE=0 then $D_j[k]$.EDGE $\leftarrow 0$
end
(b) For $k = 1$ to n_{j+1} do
If $D_j[D_{j+1}[k]$.BLINK$]$.EDGE=0 then D_{j+1}[2k].EDGE $\leftarrow 0$
end
end
(9.2) For $j = 1, 3, 5, \ldots, m-3$ do in parallel

PE_j and PE_{j+1} simultaneously perform steps (a) and (b) of step 9, respectively.

end

Step 10: (10.1) Flink all the remainder regions into some sets $A_0, \ldots, A_k$, $0 \le k \le 2n-1$, in the shared memory, from D_0 to D_{m-1} via appropriate pointers.

(10.2) Rotate each remainder region A_i, $0 \le i \le k$, so that the smallest labeled vertex is at the first position.

(10.3) Sort $A_0, \ldots, A_k$ with the first vertex.

(10.4) Erase the repetitive regions and output all the proper congruent regions.

Now, let us discuss this algorithm. The algorithm Parallel-Congruent uses m processing elements, where m is the number of edges of the testing region R. Step 1 requires $O(n)$ time to duplicate the set of edges of the input planar figure F, where n is the number of edges of F. In Step 2, the directed edges $[\vec{t}_0, \vec{t}_1, \ldots, \vec{t}_{m-1}]$ are loaded into the processing elements that requires $O(m)$ computation time. Similarly, Step 3 requires $O(m)$ time to calculate the relative angles of all the directed edges $[\vec{t}_0, \vec{t}_1, \ldots, \vec{t}_{m-1}]$. In Step 4, the set of directed edges $D = \{\vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{2n-1}\}$ are loaded into the processing elements and compared with the directed edge $\vec{t}_j$ on PE_j, where $j = 0, 1, \ldots, m-1$. Since there exist $2n$ directed edges, this step requires $O(n)$ computation time. In the worst case, there exist $O(n)$ directed edges in D_j when Step 4 is completed. In Step 5, each processing element PE_j, $j = 0, 1, \ldots, m-1$, calculates the relative angles of all the edges in D_j. Since there may exist $O(n)$ edges in D_j in the worst case, it takes $O(n)$ computation time. Step 6 sorts all the pairs in D_j, where each pair is composed by a tail vertex and an angle of certain edge. From [5] we know that any algorithm for sorting n elements must require at least $\Omega(n \log n)$ operations in the worst case. In this step, we sort the tail vertices in first pass, and then sort the angles in second pass. Each of these two passes is completed in $O(n \log n)$ time. Therefore, the whole step takes $O(n \log n)$ time. In Step 7, for each edge in D_j, $j = 0, 2, \ldots, m-2$, it uses binary search scheme to find the ordered pair $(v\ A_{j+1}(\vec{e}_{p_{j+1}}))$ in D_{j+1}, where v is not only the head vertex in D_j but also the tail vertex in D_{j+1}, and $A_{j+1}(\vec{e}_{p_{j+1}})$ is the angle of certain edge in D_{j+1}, $0 \le p_{j+1} \le 2n-1$. First, we adopt binary search to find v in D_{j+1}, and then adopt binary search to find $A_{j+1}(\vec{e}_{p_{j+1}})$ in D_{j+1} again. Since there may exist $O(n)$ edges in D_{j+1} in the worst case, it requires $O(\log n)$ time to search the tail vertex and the angle of certain edge in D_{j+1}. Therefore, the whole step requires $O(n \log n)$ computation time. Step 8 is similar to Step 7, and it takes $O(n \log n)$ computation time. In Step 9, we use the concept of the odd-even transposition sort [6, 7] to delete some unsuitable edges in D_j. Shih, Lee, and Yang [3] have shown that this step must perform $O(m)$ iterations in the worst case to complete this odd-even scheme. Therefore, this step takes $O(mn)$ computation time in the worst case. Furthermore, there may exist $O(n)$ remainder regions after Step 9. Thus, using m processing elements, Step 10.1 requires $O(n)$ computation time to feed these regions into $A_0, \ldots, A_k$, $0 \le k \le 2n-1$, in shared memory. Step 10.2 requires $O(m)$ time to rotate each A_i, $0 \le i \le k$. Thus, it takes $O(n/m)*O(m) = O(n)$ computation time to complete this step. Step 10.3 sorts $A_0, \ldots, A_k$ with the first vertex. Many papers have been published on parallel sorting scheme for last three decades. Whichever we choose, this step does not exceed $O(n \log n)$ computation time. Moreover, there may still exist $O(n)$ regions after Step 10.3. Thus, it takes $O(mn)$ computation time to erase and sequentially output all the congruent regions. Consequently, $\mathrm{MAX}\{O(mn), O(n \log n)\}$ time is required in Step 10.

From the above analysis, we observe that our algorithm is dominated by Step 6, 7, 8, 9, and 10. Thus, the time complexity of this proposed algorithm is $\mathrm{MAX}\{O(mn), O(n \log n)\}$.

The following example illustrates how the congruent regions are found by the algorithm Parallel-Congruent.

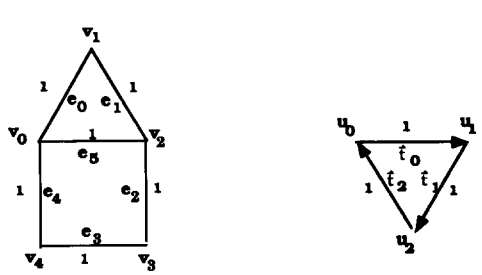

Fig. 4(a). The input planar figure F; (b) The testing region R.

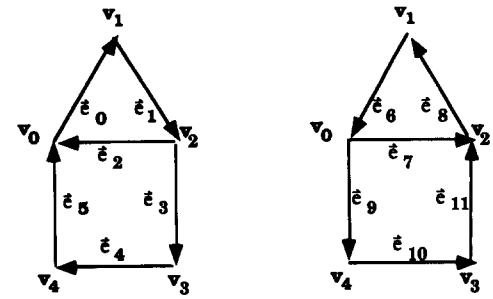

Fig. 5. All the directed edges of figure F.

D_0

EDGE	TAIL	HEAD	ANGLE
$\vec{e}_0$	v_0	v_1	60
$\vec{e}_1$	v_1	v_2	300
$\vec{e}_2$	v_2	v_0	180
$\vec{e}_3$	v_2	v_3	270
$\vec{e}_4$	v_3	v_4	180
$\vec{e}_5$	v_4	v_0	90
$\vec{e}_6$	v_1	v_0	240
$\vec{e}_7$	v_0	v_2	0
$\vec{e}_8$	v_2	v_1	120
$\vec{e}_9$	v_0	v_4	270
$\vec{e}_{10}$	v_4	v_3	0
$\vec{e}_{11}$	v_3	v_2	90

D_1

EDGE	TAIL	HEAD	ANGLE
$\vec{e}_0$	v_0	v_1	60
$\vec{e}_1$	v_1	v_2	300
$\vec{e}_2$	v_2	v_0	180
$\vec{e}_3$	v_2	v_3	270
$\vec{e}_4$	v_3	v_4	180
$\vec{e}_5$	v_4	v_0	90
$\vec{e}_6$	v_1	v_0	240
$\vec{e}_7$	v_0	v_2	0
$\vec{e}_8$	v_2	v_1	120
$\vec{e}_9$	v_0	v_4	270
$\vec{e}_{10}$	v_4	v_3	0
$\vec{e}_{11}$	v_3	v_2	90

D_2

EDGE	TAIL	HEAD	ANGLE
$\vec{e}_0$	v_0	v_1	60
$\vec{e}_1$	v_1	v_2	300
$\vec{e}_2$	v_2	v_0	180
$\vec{e}_3$	v_2	v_3	270
$\vec{e}_4$	v_3	v_4	180
$\vec{e}_5$	v_4	v_0	90
$\vec{e}_6$	v_1	v_0	240
$\vec{e}_7$	v_0	v_2	0
$\vec{e}_8$	v_2	v_1	120
$\vec{e}_9$	v_0	v_4	270
$\vec{e}_{10}$	v_4	v_3	0
$\vec{e}_{11}$	v_3	v_2	90

Fig. 6. The contents of D_0, D_1 and D_2.

D_0

EDGE	TAIL	HEAD	ANGLE
$\vec{e}_7$	v_0	v_2	0
$\vec{e}_0$	v_0	v_1	60
$\vec{e}_9$	v_0	v_4	270
$\vec{e}_6$	v_1	v_0	240
$\vec{e}_1$	v_1	v_2	300
$\vec{e}_8$	v_2	v_1	120
$\vec{e}_2$	v_2	v_0	180
$\vec{e}_3$	v_2	v_3	270
$\vec{e}_{11}$	v_3	v_2	90
$\vec{e}_4$	v_3	v_4	180
$\vec{e}_{10}$	v_4	v_3	0
$\vec{e}_5$	v_4	v_0	90

D_1

EDGE	TAIL	HEAD	ANGLE
$\vec{e}_7$	v_0	v_2	0
$\vec{e}_0$	v_0	v_1	60
$\vec{e}_9$	v_0	v_4	270
$\vec{e}_6$	v_1	v_0	240
$\vec{e}_1$	v_1	v_2	300
$\vec{e}_8$	v_2	v_1	120
$\vec{e}_2$	v_2	v_0	180
$\vec{e}_3$	v_2	v_3	270
$\vec{e}_{11}$	v_3	v_2	90
$\vec{e}_4$	v_3	v_4	180
$\vec{e}_{10}$	v_4	v_3	0
$\vec{e}_5$	v_4	v_0	90

D_2

EDGE	TAIL	HEAD	ANGLE
$\vec{e}_7$	v_0	v_2	0
$\vec{e}_0$	v_0	v_1	60
$\vec{e}_9$	v_0	v_4	270
$\vec{e}_6$	v_1	v_0	240
$\vec{e}_1$	v_1	v_2	300
$\vec{e}_8$	v_2	v_1	120
$\vec{e}_2$	v_2	v_0	180
$\vec{e}_3$	v_2	v_3	270
$\vec{e}_{11}$	v_3	v_2	90
$\vec{e}_4$	v_3	v_4	180
$\vec{e}_{10}$	v_4	v_3	0
$\vec{e}_5$	v_4	v_0	90

Fig. 7. The result of Fig. 6 after sorting.

Example

Figure 4(a) shows an input planar figure F, and Fig. 4(b) shows a testing region R. By executing the Algorithm Parallel-Congruent, we easily find all the regions that are congruent to the testing region R. The input planar figure F consists of a set of edges $E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$ and the testing regions R is defined as $[t_0, t_1, t_2]$. For simplicity, assume that $|e_0| = |e_1| = |e_2| = |e_3| = |e_4| = |e_5| = |\vec{t}_0| = |\vec{t}_1| = |\vec{t}_2| = 1$.

Step 1 duplicates the set of edges $E = \{e_0, e_1, \ldots, e_5\}$ of the input figure F to the set of directed edges $D = \{\vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{11}\}$. Figure 5 shows all the directed edges of figure F.

In Step 2, all the directed edges $\vec{t}_0$, $\vec{t}_1$, and $\vec{t}_2$ are loaded into the processing elements PE_0, PE_1, and PE_2, respectively.

Step 3 calculates the angle of $FA_0 = A(\vec{t}_0, \vec{t}_1) = A(\vec{t}_1) - (A(\vec{t}_0) - 180° + 360°)$, $FA_1 = A(\vec{t}_1, \vec{t}_2) = A(\vec{t}_2) - (A(\vec{t}_1) - 180°)$, and $FA_2 = A(\vec{t}_2, \vec{t}_0) = A(\vec{t}_0) - (A(\vec{t}_2) - 180° + 360°) + 360°$. Without loss of generality, let $FA_0 = 240° - (0 - 180° + 360°) = 60°$, $FA_1 = 120° - (240° - 180°) = 60°$, and $FA_2 = 0 - (120° - 180° + 360°) + 360° = 60°$. Then, we save FA_0, FA_1, and FA_2 into PE_0, PE_1, and PE_2, respectively. In Step 4, if $|\vec{e}_{k_j}| = |\vec{t}_j|$ then PE_j copies $\vec{e}_{k_j}$ into the set D_j, where $0 \le k_j \le 11$, $0 \le j \le 2$. Therefore, $D_0 = \{\vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{11}\}$, $D_1 = \{\vec{e}_0, \vec{e}_1, \ldots,$

D_0

BLINK	EDGE	TAIL	HEAD	ANGLE	FLINK
0	0	v_0	v_2	0	0
0	$\vec{e}_0$	v_0	v_1	60	$\vec{e}_1$
0	0	v_0	v_4	270	0
0	0	v_1	v_0	240	0
0	$\vec{e}_1$	v_1	v_2	300	$\vec{e}_2$
0	0	v_2	v_1	120	0
0	$\vec{e}_2$	v_2	v_0	180	$\vec{e}_0$
0	0	v_2	v_3	270	0
0	0	v_3	v_2	90	0
0	0	v_3	v_4	180	0
0	0	v_4	v_3	0	0
0	0	v_4	v_0	90	0

D_1

BLINK	EDGE	TAIL	HEAD	ANGLE	FLINK
0	0	v_0	v_2	0	0
$\vec{e}_2$	$\vec{e}_0$	v_0	v_1	60	$\vec{e}_1$
0	0	v_0	v_4	270	0
0	0	v_1	v_0	240	0
$\vec{e}_0$	$\vec{e}_1$	v_1	v_2	300	$\vec{e}_2$
0	0	v_2	v_1	120	0
$\vec{e}_1$	$\vec{e}_2$	v_2	v_0	180	$\vec{e}_0$
0	0	v_2	v_3	270	0
0	0	v_3	v_2	90	0
0	0	v_3	v_4	180	0
0	0	v_4	v_3	0	0
0	0	v_4	v_0	90	0

D_2

BLINK	EDGE	TAIL	HEAD	ANGLE	FLINK
0	0	v_0	v_2	0	0
$\vec{e}_2$	$\vec{e}_0$	v_0	v_1	60	0
0	0	v_0	v_4	270	0
0	0	v_1	v_0	240	0
$\vec{e}_0$	$\vec{e}_1$	v_1	v_2	300	0
0	0	v_2	v_1	120	0
$\vec{e}_1$	$\vec{e}_2$	v_2	v_0	180	0
0	0	v_2	v_3	270	0
0	0	v_3	v_2	90	0
0	0	v_3	v_4	180	0
0	0	v_4	v_3	0	0
0	0	v_4	v_0	90	0

Fig. 8. Tables of D_0, D_1 and D_2 that contain FLINK and BLINK.

$\vec{e}_{11}\}$ and $D_2 = \{\vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{11}\}$. Step 5 calculates the relative angle of each edge in D_j, $0 \le j \le 2$, and let the angle be $A_j(\vec{e}_{k_j})$, $0 \le k \le 11$. For simplicity, we show them in terms of three tables, each table contains four fields EDGE, TAIL, HEAD, and ANGLE. The field EDGE represents the edge name in D_j, $0 \le j \le 2$. The fields TAIL and HEAD are the tail vertex and head vertex of certain edge, respectively. Furthermore, the field ANGLE stores the relative angle. In order to continue our example, each angle in field ANGLE is given a suitable value. Figure 6 shows these three tables.

In Step 6, we sort all the pairs $(v_{\vec{e}_{k_j}}, A(\vec{e}_{k_j}))$, $0 \le k_j \le 11$, where $v_{\vec{e}_{k_j}}$ is the tail vertex of edge $\vec{e}_{k_j}$ in D_j, $0 \le j \le 2$. Figure 7 shows the result, after sorting, of Fig. 6.

In Steps 7 and 8, binary search scheme is adopted to find the appropriate pair with respect to the testing region R. Two pointers FLINK and BLINK are used to point to the suitable edge, respectively. For ease and simplicity, we replace these pointers with appropriate edges. Figure 8 shows this result. Furthermore, in Step 9, the scheme of the odd-even transposition sort is adopted to delete unsuitable edges in each D_j at each iteration until no more edge can be deleted in each D_j.

In Step 10, all the remainder regions are loaded into three sets A_0, A_1, and A_2, that is, $A_0 = \{\vec{e}_0, \vec{e}_1, \vec{e}_2\}$, $A_1 = \{\vec{e}_1, \vec{e}_2, \vec{e}_0\}$, and $A_2 = \{\vec{e}_2, \vec{e}_0, \vec{e}_1\}$. Then, these three sets A_0, A_1, and A_2 are rotated, respectively. Therefore, $A_0 = \{\vec{e}_0, \vec{e}_1, \vec{e}_2\}$, $A_1 = \{\vec{e}_0, \vec{e}_1, \vec{e}_2\}$, and $A_2 = \{\vec{e}_0, \vec{e}_1, \vec{e}_2\}$. Now, these three sets are sorted in terms of the first vertex. Finally, we erase the possible repetitive sets and output all the remainder regions, which are congruent to the testing region R. Therefore, only the region bounded by edges $\vec{e}_0$, $\vec{e}_1$, and $\vec{e}_2$ is congruent to the testing region R.

4. CONCLUDING REMARKS

In this paper, we propose an efficient parallel algorithm for finding the congruent regions. Compare with previous papers. In [8, 9], they limit the input form to be polygons. In [3], they adopts a CREW shared-memory model. In Step 4 of our algorithm, each processing element PE_j, $j = 0, 1, \ldots, m-1$, read the different edge $\vec{e}_i$, $i = 0, 1, \ldots, 2n-1$, on the same time. Therefore, we modify their CREW model to EREW model in this paper. Furthermore, in [3], their proposed algorithm requires $O(n^2)$ computation time for finding the congruent regions by using m processing elements. Whereas, the time complexity of our algorithm only takes $\text{MAX}\{O(mn), O(n \log n)\}$ computation time using the same number of processing elements. In addition, the results in [3] may contain some repetitive congruent regions. Here, we have solved this serious problem completely in our proposed algorithm. Moreover, our algorithm can be easily modified to handle the situation where the number of available processing elements is not the same as the number of edges in the testing region. Up until now, there still existed some interesting open problems, for instance, can we propose a parallel algorithm for finding all the 3-dimensional regions, which are congruent to a planar or 3-dimensional testing regions R? In addition, there is a region similarity problem, which is similar to this congruent region problem, in computational geometry. The difference between them is that the similarity problem must also consider the scaling factors[1, 8, 9, 10].

Acknowledgement—The authors would like to thank the referees for their valuable comments and suggestions.

REFERENCES

1. H. Alt, K. Mehlhorn, H. Wagener, and E. Weizi, Congruence, similarity, and symmetrics of geometric objects In: *Proc. 3rd ACM Ann. Symp. on Computational Geometry*, 308–315 (1987).
2. M. D. Atkinson, An optimal algorithm for geometrical congruence, *Journal of Algorithms*, **8**, 159–172 (1987).
3. Z. C. Shih, R. C. T. Lee, and S. N. Yang, A parallel algorithm for finding congruent regions, *Parallel Computing*, **13**, 135–142 (1990).
4. K. Sugihara, An n log n algorithm for determining the congruity of polyhedra, *Journal of Computer and System Sciences* **29**, 36–47 (1984).
5. E. Horowitz and S. Sahni, *Fundamentals of Data Structures*, Computer Science Press, Rockville, MD (1976).
6. G. Baudet and D. Stevenson, Optimal sorting algorithms for parallel computers, *IEEE Transactions on Computer*, **27**(1) 84–87 (1978).
7. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, Sorting and Searching, Addison-Wesley, Reading, MA (1973).
8. S. G. Akl and G. T. Toussaint, An improved algorithm to check for polygon similarity, *Information Processing Letters*, 7(3) 127–128 (1978).
9. A. Byka, On polygon similarity, *Information Processing Letters*, **9**(1) 23–25 (1979).
10. M. J. Atallah, Checking similarity of planar figures, *International Journal of Computer Information Science*, **13**(4) 279–290 (1984).