직사각형으로 분할된 영역 비교 알고리즘*

정 해 재**

요 약

CAD나 화상 처리와 같은 응용에서는 다각형으로 구성된 다양한 기하 객체를 다룬다. x축과 y축에 평행한 변을 가지는 다각형은 효율적인 조작을 위해 단순 다각형인 직사각형으로 흔히 분할된다. 그러나 동일한 기하 객체 또는 영역이라 할지라도 분할 방법에 따라 직사각형의 수, 크기 및 모양에 있어 전혀 다르게 된다. 따라서 CAD 또는 화상과 같은 장면으로부터 추출된 두 개의 직사각형 집합이 동일한 영역을 나타내는지 판단하는 알고리즘이 요구된다.

본 고에서는 직사각형들로 구성된 두 집합을 효율적으로 비교하는 알고리즘을 제안한다. 제안된 알고리즘은 기존의 훑기에 근거한 알고리즘에 비해 간단할 뿐만 아니라, 균형이진트리를 이용한 알고리즘에서의 중첩 직사각형 수 $O(n^2)$을 O(nlogn)으로 줄인다.

A Comparison Algorithm of Rectangularly Partitioned Regions

Haejae Jung**

ABSTRACT

In the applications such as CAD or image processing, a variety of geometric objects are manipulated. A polygon in which all the edges are parallel to x- or y-axis is decomposed into simple rectangles for efficient handling. But, depending on the partitioning algorithms, the same region can be decomposed into a completely different set of rectangles in the number, size and shape of rectangles. So, it is necessary an algorithm that compares two sets of rectangles extracted from two scenes such as CAD or image to see if they represent the same region.

This paper proposes an efficient algorithm that compares two sets of rectangles. The proposed algorithm is not only simpler than the algorithm based on sweeping method, but also reduces the number $O(n^2)$ of overlapped rectangles from the algorithm based on a balanced binary tree to O(nlogn).

Key words : Geometry, Sweep, Partition, Rectangle

---

* 이 논문은 2005학년도 안동대학교 학술연구 조성비에 의해 연구되었음.

** 안동대학교 정보통신공학과

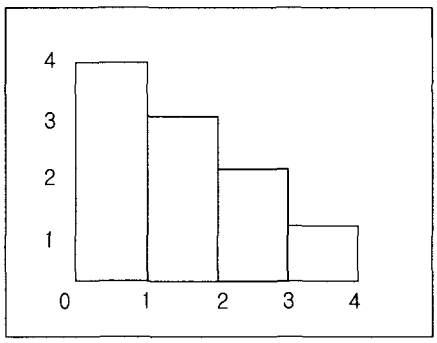
1. 서 론

CAD 또는 영상 처리와 같은 응용에서는 어떤 영역을 구성하는 복잡한 다각형을 처리하기 위해 사각형이나 사다리꼴과 같은 간단한 다각형으로 분할을 한다. 예를 들면, x와 y축에 평행한 변(edge)으로만 구성된 다각형의 경우 직사각형으로 분할될 수 있으며, 그렇지 않은 다각형의 경우 사다리꼴로 분할될 수 있다. 이렇게 분할된 상태에서 도형의 삽입, 삭제, 또는 검색과 같은 연산을 하거나, 레이(ray) 추적을 효율적으로 할 수 있다.

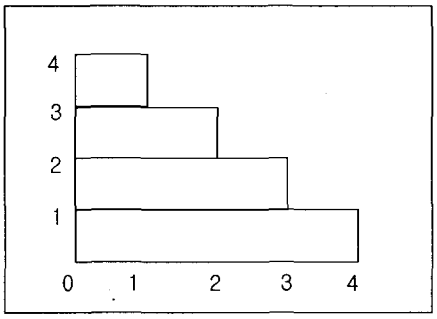
CAD와 같은 응용 분야 프로그램은 하나의 도면에 사다리꼴 또는 직사각형으로 분할된 다각형으로 표현되는 수많은 기하 객체를 다룬다[7]. 이러한 경우, 프로그램이 수정될 때마다 그 결과 도면 또는 도면의 관심 부분(예를 들면, 게이트)이 수정 전의 프로그램 결과와 동일한 지를 육안으로 검증하기가 극히 어렵다. 이것을 자동적으로 검증하는 하나의 방법은 프로그램 수정 전의 결과 도면과 수정 후의 결과 도면을 비교하는 알고리즘을 이용하는 것이다. 즉, 각 도면의 관심 부분에 속한 모든 사다리꼴 또는 직사각형을 추출하여 서로 비교하는 것이다.

이와 유사하게, 화상(image)을 아주 작은 단위인 화소 단위로 처리하는 것은 효율적이지 못하다. 따라서 화상을 효율적으로 처리하기 위해 유사한 화소 값을 가지는 이웃하는 화소들을 묶어 큰 영역을 형성하여 처리의 효율을 개선할 수 있다. 이러한 영역은 분할 알고리즘에 따라 그 영역을 구성하는 직사각형의 크기와 모양이 다르게 나타난다. 서로 다른 분할 알고리즘을 이용하여 직사각형으로 분할된 두 화상 전체 또는 화상의 일부 관심 영역이 동일한 것인지를 판단하기 위해서는 그를 구성하는 영역들을 서로 비교하여야 한다.

지금까지 2차원에서의 기하 도형 분할에 대한 많은 연구가 이루어져 왔다[2-6]. 그러나 3차원에서의 분할은 NP 문제인 것으로 증명되어 휴리스틱 알고리즘이 연구되고 있다[8, 9].



(가) 세로 중심 분할



(나) 가로 중심 분할

(그림 1) 동일 영역에 대한 서로 다른 분할

사용되는 분할 알고리즘에 따라, 동일한 영역이라 할지라도 다양한 분할된 형태를 나타낼 수 있다. 예를 들면, (그림 1)에 나타난 바와 같이 동일한 영역에 대해 분할하는 방법에 따라 그 영역을 구성하는 직사각형의 크기와 모양이 서로 전혀 다르다. (그림 1)의 (가)는 세로를 중심으로 (나)는 가로를 중심으로 분할된 단순한 예를 보여 주고 있다.

일차원에서의 두개의 집합 비교 알고리즘이 n 개의 데이터에 대해 최소한 $\Omega(n\log n)$ 시간이 걸리는 것과 같이, 이차원에서의 기하 객체 비교 또한 이 시간 복잡도를 가진다. 이차원 기하 비교 알고리즘은 기하 관련 알고리즘 분야에서 많이 쓰이는 훑기 방법이나, [1]에서 제안된 균형이진 트리 및 볼록 다각형(convex hull)을 이용한 방법을 사용할 수 있다[1-4].

기하 객체 비교는 잘 알려진 훑기 방법을 이용

한 선분 교차점 찾기 알고리즘을 응용한 공통부분 계산에 의해 이루어질 수 있으며, 이때 각 객체의 변을 선분으로 이용한다[1, 2, 4, 12-14]. 두 개의 도면으로부터 관심 부분을 추출한 많은 직사각형으로 구성된 두 집합을 비교하기 위해서도 두 다각형간 공통부분 계산에서와 같이 훑기 방법을 이용하여 계산할 수 있다. 위에서 아래로 훑기 방법을 사용할 경우, 사건(event)은 직사각형의 윗변과 아래 변이 되고, 각 사건에 훑기 선이 도달할 때마다 두 집합을 비교한다. 훑기 선이 각 직사각형의 윗변에 도달했을 때 그 직사각형은 활성 상태가 되고, 아랫변에 도달했을 때 비활성 상태가 된다. 훑기 방법을 이용한 비교 알고리즘은 각 집합이 동일하게 n개의 직사각형을 가진다고 가정할 경우 시간 복잡도는 $O(n^2 \log n)$이 된다. 그러나 훑기 방법은 두 개의 자료 구조를 유기적으로 잘 이용할 것을 요구하고 있다.

[1]에서 제안된 이진 트리 및 볼록 다각형을 이용한 영역 비교 알고리즘은 최선의 경우 훑기 알고리즘을 사용하는 것보다 훨씬 빠르지만, 최악의 경우 $O(n^2)$개의 중첩 직사각형이 생긴다.

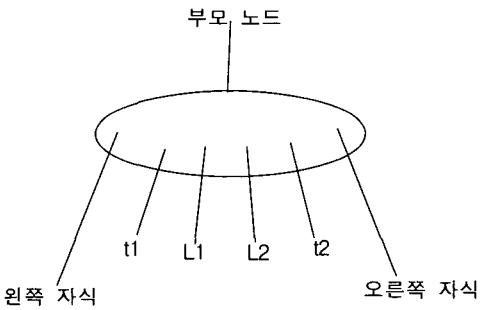
본 논문에서는 직사각형들로 구성된 두 집합이 주어 졌을 때, 그 두 개의 집합이 동일한 기하 객체들을 나타내는 지를 판단하는 영역에 근거한 비교 알고리즘을 제안한다. 세그먼트 트리(segment tree)를 확장 응용하는 제안된 알고리즘은 [1]에서 제안된 알고리즘을 사용할 경우 발생하는 중첩 직사각형 $O(n^2)$개를 O(nlogn)개로 줄인다. 다음 2장에서는 제안된 자료 구조 및 비교 알고리즘을 설명하고, 3장에서 결론을 맺는다.

2. 영역 비교 알고리즘

본 절에서는 제안된 영역 비교 알고리즘을 위해 세그먼트 트리를 확장하여 2차원의 직사각형을 표현 및 영역 비교를 할 수 있도록 한 확장 세그먼트 트리(augmented segment tree)에 대해 2.1에서 기술한 후, 각 집합에 있는 직사각형들을 정렬된 순서대로 확장 세그먼트 트리에 삽입을 하여 확장 세그먼트 트리를 초기화하는 직사각형 삽입 알고리즘을 2.2에서 설명한다. 초기화된 확장 세그먼트 트리의 각 노드를 순회하면서 두 집합을 비교하여 두 집합이 동일한 영역을 나타내는 지를 검사하는 영역 비교 알고리즘을 2.3에서 설명한다.

2.1 확장 세그먼트 트리

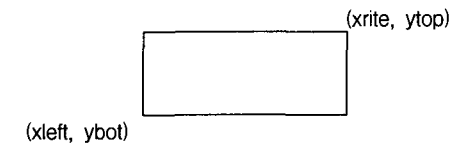
직사각형을 표현하도록 확장된 확장 세그먼트 트리는 세그먼트 트리와 같이 정적 이진 트리 구조이며, 그 노드 구조는 (그림 2)에 나타나 있다. 두 집합 S1과 S2의 직사각형들은 각각 리스트 L1과 L2에 연결된다. (그림 2)의 노드에 있는 두 연결 리스트 t1과 t2는 영역 비교 및 제거를 수행하면서 자식 노드 수준에서는 중첩되지 않아서 올라오는 합병된 직사각형들을 연결하는 리스트이다. 즉 각 노드에서 직사각형 집합 S1은 L1과 t1에 의해 표현되고 집합 S2는 L2와 t2에 의해 표현된다.




```
class Node {
  Node      parent, lchild, rchild;
  Rectangle t1, L1, L2, t2;
}
```

(그림 2) 확장 세그먼트 트리의 노드 구조

(그림 2)의 각 노드에 연결되는 직사각형은 다음 (그림 3)과 같은 구조를 가진다.




```
class Rectangle {
    Rectangle next;    // link
    int  ybot, ytop;   // y: bottom & top
    int  xleft, xrite;
}
```

(그림 3) 직사각형 좌표 및 연결 구조

2.2 직사각형 삽입 알고리즘

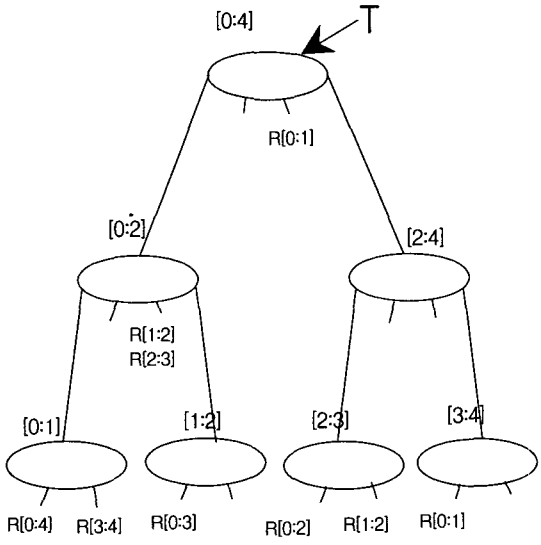
확장 세그먼트 트리를 이용하여 영역을 비교하기 위해, 영역을 구성하는 직사각형을 확장 세그먼트 트리에 삽입한다. 삽입하기 전에 직사각형 집합 각각을 ybot에 대해 오름차순으로 정렬한 후, 세그먼트 트리 알고리즘에서와 같이 직사각형을 구간 [xleft, xrite]에 대해 확장 세그먼트 트리에 삽입한다. 이 때, 직사각형 r은 인수 flag의 값에 따라 각 노드의 연결 리스트 L1 또는 L2에 삽입되는데, S1의 직사각형은 모두 L1에 연결되고 S2의 직사각형은 모두 L2에 연결된다. 따라서 이들 연결 리스트에 있는 직사각형들은 ybot에 대해 오름차순으로 연결되기 때문에, 두 집합의 영역은 선형 시간에 비교가 이루어진다.

```
void insertRect( Rectangle r,
                 Node v, int flag )
{
  if( r.xleft <= v.left && v.rite <= r.xrite ) {
     if( flag == 1 )  append r to v.L1;
     else             append r to v.L2;
  } else {
    // m : mid-point of node v
    m = ⌊ (v.left + v.rite)/2 ⌋ ;
    if( r.xleft < m ) insertRect( r, v.lchild );
    if( m < r.xrite ) insertRect( r, v.rchild );
  }

  return;
}
```

(알고리즘 1) 직사각형 삽입 알고리즘

(정리 1) 한 개의 직사각형을 확장 세그먼트 트리에 삽입하는데 O(logn) 시간이 걸린다. n은 직사각형의 개수이다.

(증명) 세그먼트 트리에서와 같이 직사각형의 구간 [xleft, xrite]이 노드 구간을 포함하는 경우 삽입이 이루어지므로 O(logn) 시간이 걸리고, 각 노드의 연결 리스트에 추가하는데 상수 시간이 걸린다. 따라서 직사각형을 삽입하는데 걸리는 총 삽입 시간은 O(logn)이 된다. ◆



(그림 4) (그림 1)의 직사각형 삽입 후

(그림 1)의 모든 직사각형을 (알고리즘 1)을 이용하여 삽입한 후의 확장 세그먼트 트리는 (그림 4)와 같다. (그림 4)의 각 노드위에 표시된 구간은 묵시적으로 표현되는 노드의 x구간을 나타내고, 삽입된 직사각형 R[xleft : xrite, ybot : ytop]은 그 직사각형의 y구간 R[ybot : ytop]만으로 간단히 연결 리스트에 표현되어 있다. 이 그림에서 연결 리스트 t1과 t2는 직사각형 삽입 시 사용되지 않으므로 생략되어 있다.

예를 들어, 직사각형 R[0 : 3, 1 : 2]의 경우, 노드

구간 [0:2]와 [2:3]을 나타내는 노드에 삽입되고, 구간 [0:2]를 나타내는 노드에는 두개의 직사각형 R[1:2]와 R[2:3]이 ybot에 대해 오름차순으로 연결되어 있다.

2.3 영역 비교 알고리즘

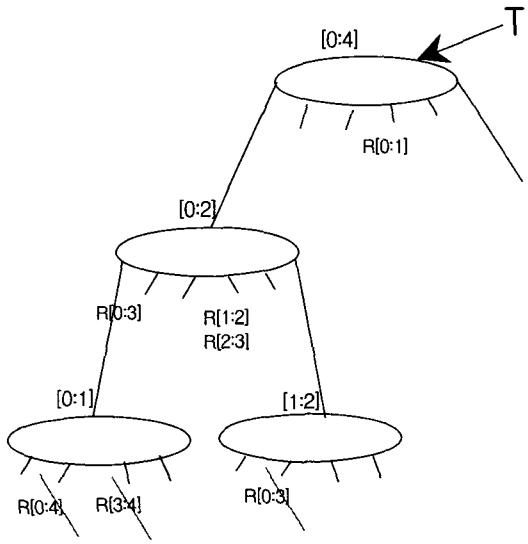
두 집합에 있는 모든 직사각형이 삽입된 후, 두 집합이 동일 영역을 나타내는 지를 검사하기 위해 트리의 각 노드를 후위 순회(post-order traversal) 순서로 방문하여 중첩 영역(overlapped region)을 제거한다.

```
void  removeOverlappedRegion( Node v )
{
  if( v == NULL )  return;

  removeOverlappedRegion( v.lchild );
  removeOverlappedRegion( v.rchild );

  // remove overlapped regions
  // and promote the rest.
  리스트 ((L1, t1), (L2, t2))의
                중첩 직사각형 제거;
  중첩되지 않는 (L1, t1)의 직사각형을
                부모노드의 t1에 연결;
  중첩되지 않는 (L2, t2)의 직사각형을
                부모노드의 t2에 연결;
}
```

(알고리즘 2) 중첩영역 제거 알고리즘

즉, (알고리즘 2)에 나타난 바와 같이, 각 노드에서 S1을 나타내는 리스트 (t1, L1)과 S2를 나타내는 (L2, t2)에 연결된 직사각형들을 서로 비교하여 중첩되는 경우 트리로부터 모두 제거하고, 중첩되지 않는 나머지 직사각형은 부모 노드의 연결 리스트 t1과 t2에 연결한다.

앞서 (알고리즘 1)에서 설명한 삽입 알고리즘은 모든 직사각형을 연결리스트 L1과 L2에만 연결하므로, 리프 노드의 경우 L1과 L2만 비교하면 되고, 내부 노드의 경우에는 네 개의 연결 리스트에 있는 모든 직사각형을 비교해야 한다. 이러한 비교는 확장 세그먼트 트리에 삽입하기 전에 각 셋의 모든 직사각형을 정렬하였으므로 직사각형의 개수에 비례하여 선형 시간이 걸린다.

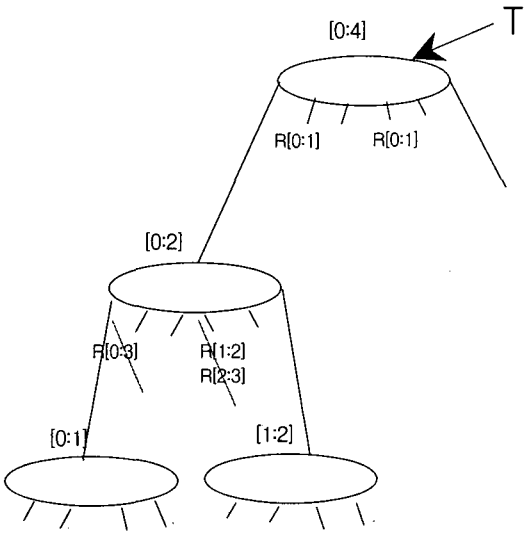


(그림 5) 노드 [0:2]에서 비교 전의 트리

(그림 5)는 구간 [0:2]를 나타내는 노드, 즉 노드 [0:2]의 두 서브트리를 처리한 직후의 확장 세그먼트 트리를 보여주고 있다. 후위 순회에 따라 노드 [0:1]의 두 직사각형을 검사한다. 두 직사각형의 R[3:4]가 중첩되어 제거되고, 중첩되지 않는 부분 R[0:3]이 부모 노드인 노드 [0:2]의 리스트 t1에 연결된다. 노드 [1:2]에서는 집합 S1의 직사각형만 있으므로, 노드 [1:2]의 R[0:3]은 부모 노드 [0:2]의 연결 리스트 t1의 R[0:3]과 합병되어 직사각형의 x 범위가 노드 [0:2]의 범위가 된다.

(그림 6)은 노드 [0:2]에서의 중첩 직사각형 제거 후의 결과를 보여주고 있다. 셋 S1에 속하는 리스트 t1의 직사각형 R[0:3]과 셋 S2에 속하는 리스트 L2의 직사각형 R[1:2] 및 R[2:3]의 중첩 영역을 제거한 나머지 부분인 R[0:1]이 부모 노드

인 루트 노드의 t1에 연결된다. 두 셋이 동일 영역을 나타내는 경우, 올라온 이 직사각형은 루트 노드의 오른 쪽 서브 트리의 처리 후 올라올 직사각형과의 합병에 의해 루트 노드 x 구간 [0:4]를 형성하여 집합 S2의 직사각형과 중첩 비교에 의해 트리로부터 제거된다.



(그림 6) 노드 [0:2]에서 중첩 영역 제거 후

결과적으로 두 집합 S1과 S2가 동일한 영역을 나타내면, 중첩 영역 직사각형 제거 알고리즘 removeOverlappedRegion()의 수행 후, 트리 루트 노드의 4개 리스트 t1, L1, L2, 및 t2 모두 빈(empty) 리스트가 된다. 그렇지 않고 두 셋이 서로 다른 영역을 나타낸다면, 루트 노드의 4개 리스트는 중첩되지 않는 영역을 가지게 된다.

(정리 2) 두 집합의 중첩 직사각형 영역을 제거하는 알고리즘 removeOverlappedRegion()은 O($n$log$n$) 시간 복잡도를 가진다. 여기서 두 집합이 각각 n개의 직사각형을 포함하고 있다고 가정한다.

(증명) n개의 직사각형에 대해 확장 세그먼트 트리는 O(log$n$)의 높이를 가진다. 하나의 직사각형은 세그먼트 트리에서처럼 O(log$n$)개의 노드에 삽입될 수 있고 한 레벨에는 많아야 두개의 노드에 삽입될 수 있다. 따라서 트리의 각 레벨은 많아야 O(n)개의 직사각형을 포함하고 각 직사각형을 처리하는데 O(*1*) 시간이 걸리고 트리의 높이가 O(log$n$)이므로, 알고리즘 removeOverlappedRegion()의 총 시간 복잡도는 O($n$log$n$)이 된다. ◆

(알고리즘 3)은 상기 두 알고리즘을 이용하여 영역을 비교하는 알고리즘을 보이고 있다. 알고리즘에서 보는 바와 같이, 두 셋을 먼저 정렬하고, 각 셋의 직사각형을 insertRect() 알고리즘을 이용하여 확장 세그먼트 트리에 모두 삽입한다. 그 후, removeOverlappedRegion() 함수를 이용하여 트리에 삽입된 두 집합의 중첩 직사각형 영역을 제거한 후, 루트 노드의 4개의 리스트 t1, L1, L2, t2를 검사한다. 4개의 리스트가 모두 비어 있을 경우, 두 집합의 영역은 동일함을 의미한다.

```
void compareRegion( Rectangle S1[],
                    Rectangle S2[] )
{
    sort( S1 );  sort( S2 );

    // T : the root node of
    // the augmented segment tree
    for( Rectangle r : S1 )
        insertRect( r, T, 1 );
    for( Rectangle r : S2 )
        insertRect( r, T, 2 );

    result = removeOverlappedRegion( T );

    if( 루트노드 T의 4 리스트 == empty )
        display "Same Regions !";
    else
        display "Different Regions !";
}
```

(알고리즘 3) 영역 비교 알고리즘

(정리 3) 제안된 영역 비교 알고리즘 compareRegion()은 O(*n*log*n*) 시간 복잡도를 가진다. 여기서 각 셋에 *n*개의 직사각형이 있다고 가정한다.

(증명) (알고리즘 3)에서 보는 바와 같이 두 셋을 정렬하는데 각각 O(*n*log*n*) 시간이 걸리고, 두 개의 for 문을 통하여 모든 직사각형을 삽입하는데 O(*n*log*n*) 시간이 걸린다. 또한 두 셋의 중첩 영역을 제거하는 알고리즘 removeOverlapped Region() O(*n*log*n*) 시간 복잡도를 가지므로, 영역 비교 알고리즘의 총 시간 복잡도는 O(nlogn)이 된다. ◆

3. 결 론

본 고에서는 세그먼트 트리를 확장한 확장 세그먼트 트리를 이용하여 직사각형으로 구성된 두 영역을 비교하는 알고리즘을 제안했다. 제안된 알고리즘은 [1]에서 제안된 알고리즘의 중첩 영역의 개수를 $O(n^2)$에서 O(*n*log*n*)로 줄이므로, 효과적으로 영역 비교를 수행한다.

제안된 알고리즘은 x 또는 y축에 평행한 사다리꼴로 분할된 영역에 대해서도 쉽게 적용할 수 있다. 이 경우 시간 복잡도는 직사각형으로 분할된 영역 비교와 동일하다.

참 고 문 헌

[1] 정해재, "k 사다리꼴 셋의 영역 중심 비교 알고리즘", 한국정보처리학회 논문지, 10-A(6), pp. 665-670, 2003.

[2] F. P. Preparata and M. I. Shamos, "Computational Geometry : An Introduction", Springer-Verlag, New York, 1988.

[3] D. Mehta and S. Sahni(ed.), "Handbook of Data Structures and Applications", Chapman & Hall/CRC, New York, 2005.

[4] J. O'Rourke, "Computational Geometry in C (2nd edition)", Cambridge University Press, New York, pp. 264-268, 1998.

[5] S. Nahar and S. Sahni, "Fast Algorithm for Polygon Decomposition", IEEE Transactions on Computer-Aided Design, Vol. 7, No. 4, pp. 473-483, Apr. 1988.

[6] Takao Asano, Tetsuo Asano, and Hiroshi Imai, "Partitioning a Polygon Region Into Trapezoids", JACM, Vol. 33, No. 2, pp. 290-312, 1986.

[7] J. Ousterhout, "Corner Stitching : A Data-Structuring Technique for VLSI Layout Tools", IEEE Transactions on CAD", Vol. 3, No. 1, pp. 87-99, 1984.

[8] Haejae Jung, "Algorithms for External Beam dose Computation", Ph.D. thesis, University of Florida, 2000.

[9] V. J. Dielissen and A. Kaldewaij, "Rectangular Partition is Polynomial in Two Dimensions but NP-complete in Three", Information Processing Letters, Vol. 38, No. 1, pp. 1-6, Apr. 1991.

[10] E. Horowitz, S. Sahni, and D. Mehta, "Fundamentals of Data Structures in C++", W. H. Freeman, San Francisco, 1995.

[11] D. Sleator and R. Tarjan, "Self-adjusting Binary Search Trees", Journal of the Association for Computing Machinary, Vol. 32, No. 3, pp. 652-686, 1985.

[12] J. Bentley, and T. Ottmann, "Algorithms for Reporting and Counting Geometric Intersections", IEEE Transactions on Computer, C-28, pp. 643-647, 1979.

[13] I. J. Balaban, "An Optimal Algorithm for

Finding Segment Intersections", Proc. 11th Annual ACM Symposium on Computational Geometry, pp. 211-219, 1995.

[14] K.L. Clarkson, and P.W. Shor, "Applications of Random Sampling in Computational Geometry, II", Discrete & Computational Geometry, Vol. 4, pp. 387-421, 1989.

정 해 재

1984년 경북대학교 전자계산학과 (공학사)

1987년 서울대학교 컴퓨터 공학과 (공학석사), DBMS

2000년 플로리다대학교(UF) 컴퓨터정보학과(공학박사), 알고리즘

1988년~1995년 한국전자통신연구원 선임연구원

2001년~2002년 Numerical Technologies Inc., Staff Engineer

2003년~2005년 성신여자대학교 컴퓨터정보학부 교수

2005년~현재 안동대학교 공과대학 정보통신 공학과 교수

관심분야 : 고성능 정보처리, 자료구조 및 알고리즘, 계산기하, 네트워크 알고리즘