k 사다리꼴 셋의 영역 중심 비교 알고리즘

정 해 재†

요 약

반도체 생산을 위한 마스크 자동 생성과 같은 기하 객체를 다루는 응용에서는, 사다리꼴로 분할된 수 많은 다각형으로 구성된 도면에 새로운 다각형을 추가하거나 삭제하기 위해 사다리꼴 삽입, 삭제, 및 검색 연산을 한다. 동일한 다각형에 대해 분할된 사다리꼴은 사용된 분할 알고리즘에 따라 모양, 크기 등에 있어서 다르게 된다. 사다리꼴로 구성된 기하 객체를 다루는 프로그램을 검증하는 것과 같은 예에서는 구성된 도면의 관심 부분을 나타내는 여러 사다리꼴 셋을 비교하는 알고리즘이 필요하다. 본 논문에서는 k개 도면의 관심 영역으로부터 각각 추출된 사다리꼴로 구성된 k 셋이 주어졌을 때, 그 k 셋이 형성하는 기하 도형들이 동일한지 아닌지를 비교하는 새로운 알고리즘을 제시한다. 제시된 알고리즘은 각 셋이 공히 n개의 사다리꼴을 포함하고 있다고 가정할 때, $O(2^{k-2}n^2(\log n+k))$ 시간 복잡도를 가진다. 제시된 알고리즘은 입력 셋의 수 k($<<n$)가 적을 때 훑기 중심 알고리즘과 동일한 시간 복잡도 $O(n^2 \log n)$를 가지며, 특히 k 셋이 동일하거나 대부분 동일한 사다리꼴들로 구성되어 있을 경우 훑기 중심 알고리즘보다 kn배까지 빠른 것은 나타났다.

A Region-based Comparison Algorithm of k sets of Trapezoids

Haejae Jung†

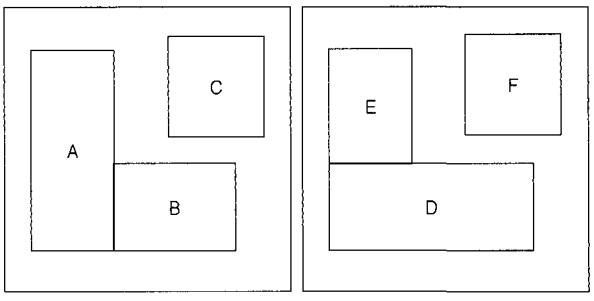
ABSTRACT

In the applications like automatic masks generation for semiconductor production, a drawing consists of lots of polygons that are partitioned into trapezoids. The addition/deletion of a polygon to/from the drawing is performed through geometric operations such as insertion, deletion, and search of trapezoids. Depending on partitioning algorithm being used, a polygon can be partitioned differently in terms of shape, size, and so on. So, it's necessary to invent some comparison algorithm of sets of trapezoids in which each set represents interested parts of a drawing. This comparison algorithm, for example, may be used to verify a software program handling geometric objects consisted of trapezoids. In this paper, given k sets of trapezoids in which each set forms the regions of interest of each drawing, we present how to compare the k sets to see if all k sets represent the same geometric scene. When each input set has the same number n of trapezoids, the algorithm proposed has $O(2^{k-2}n^2(\log n+k))$ time complexity. It is also shown that the algorithm suggested has the same time complexity $O(n^2 \log n)$ as the sweeping-based algorithm when the number k($<<n$) of input sets is small. Furthermore, the proposed algorithm can be kn times faster than the sweeping-based algorithm when all the trapezoids in the k input sets are almost the same.

키워드 : 다각형(Polygon), 볼록 다각형(Convex Hull), 사다리꼴(Trapezoid), 분할(Partition), 공통(Intersection), 중첩(Overlap), 동적 균형 이진 트리(Dynamic Balanced Binary Search Tree), 훑기(Sweeping)

1. 서 론

CAD, 컴퓨터 그래픽스, 또는 이미지 처리와 같은 응용에서는 복잡한 다각형을 처리하기 위해 사각형이나 사다리꼴과 같은 간단한 다각형으로 분할을 한다. 예를 들면, x와 y 축에 평행한 선으로만 구성된 2차원 화상의 경우 직사각형으로 분할될 수 있으며, 다각형의 경우 사다리꼴로 분할될 수 있다. 이렇게 분할된 상태에서 도형의 삽입, 추출, 또는 삭제와 같은 연산을 하거나, 레이(ray) 추적을 효율적으로 할 수 있다. 지금까지 2차원에서의 기하 도형 분할에 대한 많은 연구가 이루어져 왔다[1-5]. 그러나, 3차원에서의 분할 문제는 NP 문제인 것으로 증명되어 휴리스틱 알고리즘이 연구되고 있다[6, 7].



(a) 세로 우선 분할 (b) 가로 우선 분할

(그림 1) 동일한 다각형을 가진 도면의 서로 다른 분할 예

† 종신회원 : 성신여자대학교 컴퓨터정보학부 교수
논문접수 : 2003년 4월 1일, 심사완료 : 2003년 11월 10일

사용되는 분할 알고리즘에 따라, 동일한 도형이라 할지라도 다양한 분할 모양을 나타낼 수 있다. 예를 들면, (그림 1)에 나타난 바와 같이, (그림 1)(a)의 A, B, C로 구성된 도형과 (그림 1)(b)의 D, E, F로 구성된 도형이 동일하다 할지라도, (그림 1)(a)는 세로를 (그림 1)(b)는 가로를 우선하여 분할하였기 때문에 도형을 구성하는 각각의 도형인 사다리꼴 즉 직사각형은 그 크기와 모양에 있어서 전혀 다르다.

CAD와 같은 응용 분야 프로그램은 하나의 도면에 사다리꼴로 분할된 다각형으로 표현되는 수 많은 기하 객체를 다룬다[4]. 이러한 경우, 프로그램이 수정될 때마다 그 결과 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 결과와 수정 후의 결과 도면을 비교하는 알고리즘을 이용하는 것이다. 즉, 각 도면의 관심 부분에 속한 모든 사다리꼴을 추출하여 서로 비교하는 것이다. 결과가 다를 경우, 각 모듈 별로 따로 수정된 실행 코드로부터 얻은 각각의 결과 도면의 관심 부분을 모두 비교함으로써, 그 프로그램의 어느 모듈이 수정 전의 도형과 다른 도형을 발생시켰는지를 추적할 수 있다.

이러한 기하 객체 비교는 다각형 간의 부울(boolean) 연산인 공통 부분 계산에 의해 이루어질 수 있다. 두 다각형 간의 공통 부분 계산은 훑기 방법에 근거한 선분 교차점 찾기 알고리즘을 응용하여 계산 할 수 있으며, 이때 다각형을 형성하는 각 변을 선분으로 이용한다[1, 10-12]. k 도면으로부터 관심 부분을 추출한 많은 사다리꼴로 구성된 k 셋을 비교하기 위해서도 두 다각형간 공통 부분 계산에서와 같이 훑기 방법을 이용하여 계산할 수 있다. 위에서 아래로 훑기 방법을 사용할 경우, 사건(event)은 사다리꼴의 윗변과 아래 변이 되고, 각 사건에 훑기 선이 도달했을 때마다 k 셋을 비교한다. 훑기 선이 각 사다리꼴의 윗변에 도달했을 때 그 사다리꼴은 활성 생태가 되고, 아랫변에 도달했을 때 비활성 상태가 된다. [1]에서 기술된 두 다각형 공통 부분 계산 알고리즘과 유사하게, 훑기 방법을 이용한 비교 알고리즘은 다음과 같이 기술된다.

① 각 사건(사다리꼴의 윗변 및 아래 변)의 y값에 대해 정렬.
② 훑기 선을 사건의 가장 큰 사건의 y값으로 초기화.
③ 각 셋에 대해, 훑기 선과 모든 활성 상태의 사다리꼴의 공통 부분을 계산한다. 이 경우 공통 부분은 수평 선분들로 구성된다.
④ 구한 공통 부분을 셋 끼리 서로 비교한다. 비교 결과 서로 다르면 입력 셋들이 서로 동일하지 않으므로 종료한다.
⑤ 훑기 선을 현재 훑기 선 값보다 적은 가장 큰 y 사건 값으로 바꾸고, 위의 단계 3과 단계 4를 반복한다. 이러한 반복은 제일 적은 y값을 가지는 사건을 처리한 후 종료한다.

기술한 훑기 방법을 이용한 비교 알고리즘은 각 셋이 공히 n개의 사다리꼴을 가지는 k개의 셋에 대해 $O((kn)^2 \log(kn))$ 시간 복잡도를 가진다. 단계 1과 단계 2는 각각 $O(kn \log(kn))$ 및 $O(1)$ 시간 걸린다. 어떤 사건에 도달한 훑기 선에서 kn 개의 사다리꼴이 활성 상태일 수 있고, 각 활성 사다리꼴과 훑기 선의 공통 부분은 상수 시간에 계산되므로, 단계 3은 $O(kn)$ 시간이 걸린다. 단계 4의 k 개의 셋을 서로 비교하기 위해 정렬이 필요하므로 $O(kn \log(kn))$ 시간 걸린다. 단계 3과 단계 4는 kn번 반복될 수 있으므로, 총 시간 복잡도는 $O((kn)^2 \log(kn))$이 된다.

본 논문에서는 사다리꼴을 구성 요소로 하는 k(≥ 2)개의 셋이 주어 졌을 때, 그 k개의 셋이 동일한 기하 객체들을 나타내는 지를 판단하는 영역에 근거한 비교 알고리즘을 제시한다. 2장에서 문제에 대한 정의를 기술하고, 3장에서는 제시된 비교 알고리즘을 설명하며, 4장에서는 본 고에서 제시한 알고리즘과 앞서 기술된 훑기에 근거한 알고리즘의 성능 비교를 한 후, 5장에서 결론을 맺는다.

2. 문제 정의

본 논문에서는 k개의 도면이 있을 경우, 그 k개의 도면에 있는 관심 영역이 모두 동일한지를 판단하는 새로운 알고리즘을 제안한다. 이때, 각 도면의 관심 영역은 사다리꼴로 분할되어 있다고 가정한다. 제안되는 알고리즘의 입력과 출력은 다음과 같이 정의된다.

- 입력 : 각 도면의 관심 영역을 구성하는 사다리꼴을 담고 있는 k개의 배열.
- 출력 : 모든 배열이 동일한 관심 영역을 나타내면 참을 출력하고, 아니면 거짓을 출력.

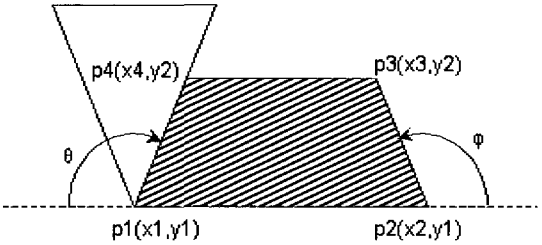
따라서, 문제는 다음과 같이 정의 된다.

[문제 정의] *사다리꼴로 구성된 k개의 셋이 주어졌을 때, k 셋이 모두 동일한지를 판단한다.*

3. 영역 중심 비교 알고리즘

(그림 2)의 빗금 친 사다리꼴에 나타난 바와 같이, 윗변과 아랫변이 평행한 사다리꼴은 4개의 꼭지점 p1, p2, p3, 및 p4로 표현될 수 있는데, 본 논문에서는 (x1, y1)을 p1, (x2, y1)을 p2, (x3, y2)를 p3, (x4, y2)를 p4라 한다. 또한,

사다리꼴은 y축에 대해 평행한 두 선분을 가진 것과 x축에 평행한 두 선분을 가진 두 종류로 나눌 수 있지만, 특별한 언급이 없는 한 본 논문에서는 x축에 평행한 두 선분을 가진 사다리꼴을 단순히 사다리꼴이라 부르기로 한다.



(그림 2) 사다리꼴 좌표

영역 비교를 위하여 동적 균형 이진 탐색 트리들 중 하나를 이용할 것인데, 이를 BST라 한다[8, 9]. 먼저, BST를 이용하기 위해서, 우리는 사다리꼴을 정렬하기 위한 키를 결정하여야 한다. 본 논문에서는 y1값에 대해 먼저 정렬하고, 동일한 y1값에 대해서는 x1 값에 대해 정렬한다. 그러나, (그림 2)의 왼쪽 사다리꼴 아랫변이 하나의 점으로 되어 오른쪽 사다리꼴의 (x1, y1)과 겹쳐져 있는 경우를 구분하기 위하여 p4를 이용한다. 따라서, 키를 <y1, x1, x4, y2>로 하여 각도 θ 값에 따라 정렬 함으로써, 이러한 문제를 해결할 수 있다.

먼저, k개의 셋에 포함되어 있는 모든 사다리꼴을 BST에 삽입하여 BST를 초기화 한 후, 제일 적은 키 값을 가진 사다리꼴을 한번에 k개씩 추출하여 서로 비교한다. 추출된 k개의 사다리꼴은 모두 동일한 p1 값을 가져야 하고, 좌변은 동일한 선 상에 있어야 한다. 그렇지 않으면, 비교 알고리즘은 거짓을 리턴하게 되다. 키가 같으면, 추출된 사다리꼴의 공통 부분을 찾아, 추출된 각각의 사다리꼴에서 공통 부분을 제외한 나머지 부분을 다시 사다리꼴로 분할하여 BST에 삽입한다. 그 공통 부분의 윗변은 하나의 수평선을 형성하는데, 이를 ymax라 한다. 이 값은 추출시 가장 작은 y2값을 ymax로 함으로써 쉽게 결정할 수 있다. 문제는 k 사다리꼴의 우변으로부터 형성될 공통 부분의 경계를 어떻게 빨리 찾을 것인가 하는 것이다.

사다리꼴의 우변은 선분으로 이루어져 있고, 모두 [y1, ymax] 구간을 통과하므로 공통 부분은 최대 k개의 선분으로 이루어진 볼록 다각형을 형성하게 된다[1, 5]. 그러나, 일반적인 볼록 다각형을 찾는 알고리즘은 $O(k \log k)$ 시간 복잡도를 가지는데, 본 논문에서는 이 시간 복잡도를 선형 시간으로 만들어 영역 비교 알고리즘의 전체 시간 복잡도를 줄이고자 한다. 여기서 관찰할 것은 영역 비교 알고리즘은 BST를 사용하고 있기 때문에, 이 BST만 잘 이용하면 k개의 사다리꼴을 (그림 2)에 나타난 우변에 의해 형성되는 각도 에 대해서 정렬된 순서로 추출할 수 있다. 즉, 사다리꼴의 우변을 나타내는 k개의 선분을 에 대해 정렬된 순서로 가지고 있다면, 선형 시간에 볼록 다각형인 공통 부분을 구할 수 있다. 이를 위해, 앞서 정의된 BST 키에 p2와 p3 값을 추가 함으로써, k개의 사다리꼴을 추출할 때 사다리꼴의 우변을 항상 에 대해 정렬된 순서대로 획득할 수 있다. 따라서, 키는 다음과 같이 정의된다.

BST 키 : <y1,x1, x4,y2, x2,y1, x3,y2 >

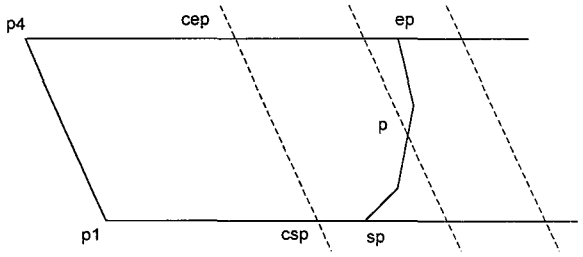
```
int  find_CH ( ymin, ymax, ls[0..k-1], bndpt[] )
{ // ls[]에 k개의 선분이 있음.
    ymin과 ls[0], ymax와 ls[0]의 교차점을 각각 sp, ep라 하자.
    bndpt[]에 sp와 ep를 추가 ;
    for ( i = 1; i < k; i++ ) {
        csp = ls[i]와 ymin의 교차점 ;
        cep = ls[i]와 ymax의 교차점 ;
        if ( csp가 sp의 왼쪽에 있음 ) {
            sp = csp ;   ep = cep ;
            bndpt[0] = sp ;   bndpt[1] = ep ;
            continue ;
        }
        if ( cep가 ep의 오른쪽에 있음 ) continue ;
        경계선으로 형성된 선분을 시계 방향으로 가면서 교차점
        p를 찾음 ;
        벡터<p, cep>의 오른쪽에 있는 모든 선분을 bndpt[]에서
        삭제 ;
        p와 cep를 새로운 경계선으로 bndpt[]에 추가 ;
    } // end of for loop
    return bndpt[]에 있는 점의 수 ;
} // end of find_CH()
```

(그림 3) 구간 [ymin, ymax]를 통과하는 k개의 선분에 의한 볼록 다각형 계산

φ에 대해 정렬된 k개의 선분이 주어졌을 때, (그림 4)와 같이 볼록 다각형을 형성하는 오른쪽 경계선을 선형 시간에 찾을 수 있다. (그림 3)의 find_CH() 알고리즘은 x축과 평행한 두 개의 선 ymin, ymax와 φ에 대해 정렬된 k개의 선분을 가진 배열 ls[]를 입력으로 받아, k 사다리꼴 공통 부분의 오른쪽 경계선을 이루는 점을 bndpt[] 배열에 담아 호출 함수에 전달한다. 리턴 값은 bndpt[]에 저장된 점의 갯수이다. (그림 4)에서 보는 바와 같이, sp로부터 ep에 이르는 선분들이 지금까지 찾아 bndpt[]에 저장된 경계선이라 하자.

점선으로 된 현재 선분과 sp에서 ep에 이르는 경계선에 의해 형성되는 새로운 경계선은 세가지 경우로 나뉘어 계산될 수 있다. 만일 제일 왼쪽의 점선과 같이 csp가 sp의 왼쪽에 있다면, 찾고자 하는 경계선은 csp에서 cep에 이르는 선분이 된다. 현재 선분이 제일 오른쪽의 점선이면, cep가 ep의 오른쪽에 있으므로 이 점선은 볼록 다각형 형성에 영향을 주지 못하게 된다. 그렇지 않을 경우에는, (그림 4)의 중간에 있는 점선과 같은 현재 선분이 이미 발견된 경

계선과 교차하게 되므로, 그 교차점을 찾아야 한다. 이를 위해, 우리는 ep로 시작하여 시계 방향으로 진행하면서 현 선분과 교차하는 선분을 찾는다. 찾은 교차점 p로부터 cep로 향했을 때, 그 오른쪽에 있는 모든 점을 bndpt[]에서 제거하고, 교차점 p와 cep를 bndpt[]에 추가하여 새로운 볼록 다각형의 경계선으로 한다.



(그림 4) 볼록 다각형 경계선

[정리 1] [ymin, ymax]을 통과하고 φ에 대해 정렬된 k개 선분의 볼록 다각형은 선형 시간 O(k)에 구할 수 있다. 여기서, ymin < ymax.

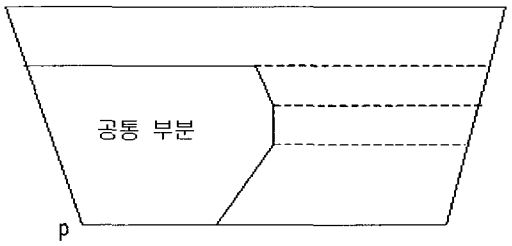
(증명) (그림 3)의 알고리즘에 따라, 각 선분은 최대 한번 bndpt[] 배열에 삽입되고, 최대 한번 삭제된다. 따라서, (그림 3)의 알고리즘은 O(k) 시간 복잡도를 가진다. ◆

(그림 5)의 same_trapset() 알고리즘은 (그림 3)의 find_CH() 알고리즘을 이용하여 사다리꼴로 구성된 k 셋을 비교한다. 매개 변수로 표현된 각각의 입력 배열은 각 도면으로부터 추출된 사다리꼴을 포함하고 있으며, 모든 셋이 동일한 영역을 나타내면 참 값을 리턴하고 그렇지 않으면 거짓을 리턴한다.

```
bool same_trapset ( set1[], set2[], ···, setk[] )
{
    <p1, p4, p2, p3>를 키로 하여,
    k 셋의 모든 사다리꼴을 BST에 삽입 ;  // BST 초기화
    while( BST is NOT empty ) {
       BST로부터 k개의 사다리꼴을 추출하여 배열 ls[]에 저장 ;
       if ( k 사다리꼴의 p1 값이 모두 동일하지 않거나,
          좌변이 모두 동일 선상에 있지 않으면 ) return FALSE ;
       if (추출된 k 사다리꼴이 모두 동일하면) continue ;
       ymax = 추출된 k 사다리꼴의 y2 값들 중 최소값 ;
       m = find_CH( y1, ymax, ls[0..k-1], bndpt[] ) ;
       // 공통 부분의 오른쪽 경계선을 형성하는 m개의 점이
          bndpt[]에 있음.
       for ( 추출된 각각의 사다리꼴에 대해 ) {
          공통 부분에 포함되지 않는 부분을 사다리꼴로 분할 ;
          분할된 모든 사다리꼴을 BST에 삽입 ;
       } // end of for
    } // end of while
    return TRUE ;
} // end of same_trapset()
```

(그림 5) 사다리꼴로 구성된 k 셋 영역 비교 알고리즘

(그림 5)의 알고리즘에서 나타난 바와 같이, k개의 입력 배열에 있는 모든 사다리꼴은 동적 균형 이진 탐색 트리 BST에 삽입된다. 그 후, while문에서 사다리꼴을 k개씩 정렬된 순서로 추출한다. 추출된 k 사다리꼴의 공통 부분인 볼록 다각형을 구한 후, 추출된 사다리꼴 각각에 대해 공통 부분을 제외한 부분을 다시 사다리꼴로 분할하여 BST에 삽입한다. 예를 들어, (그림 6)의 경우 추출된 큰 사다리꼴에서 공통 부분을 제외한 부분에서 4개의 새로운 사다리꼴이 생성되어 BST에 삽입된다. 이러한 작업을 반복하여 BST가 비게 되면 참 값을 리턴한다. 그러나, p1 값이나 사다리꼴의 좌변이 동일 선상에 있지 않으면 거짓을 리턴한다.

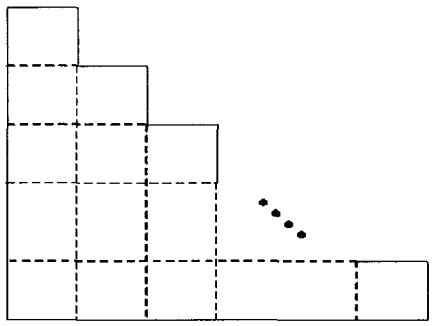


(그림 6) 공통 부분은 제외한 부분의 사다리꼴 분할

기술된 (그림 5)의 영역 비교 알고리즘 성능 분석을 위해 중첩 사다리꼴의 갯수를 살펴보면 다음과 같다.

[정리 2] 중첩 사다리꼴 갯수 : 입력 k (≥2)개의 배열에 공히 n개의 사다리꼴이 있다고 하자. 이때, k 개의 배열에 있는 모든 사다리꼴이 중첩되었을 때 발생하는 중첩 사다리꼴 갯수는 최소 kn개에서 최대 $2^{k-2}n^2$개이다.

(증명) k 셋이 모두 동일한 n개의 사다리꼴을 포함할 경우 중첩 사다리꼴의 갯수는 kn개가 된다.



(그림 7) 가로와 세로로 분할된 영역의 중첩

최대의 경우를 살펴보자. k = 2일 경우, (그림 7)에서 보는 바와 같이 하나는 가로로 다른 하나는 세로로 분할했을 경우 가능한 중첩 사다리꼴(사각형)의 수는 n^2이다. k > 2인 경우, 앞서 중첩 분할된 사다

리꼴보다 많아야 2배씩 증가하게 된다. 예를 들면, 처음 두 셋 중의 하나(세로 또는 가로로 분할된 셋)를 약간 이동하여 중첩시키면 2배 증가 한다. 따라서, 최대 중첩 사다리꼴의 갯수는 $2^{(k-2)}n^2$이 된다. 이때, k는 2, 3, 4, …인 정수이다. ◆

위의 [정리 2]를 이용하여 (그림 5)의 영역 비교 알고리즘을 분석하면 다음과 같은 시간 복잡도를 얻는다.

[정리 3] 시간 복잡도 : 사다리꼴로 분할된 영역 비교 알고리즘은 $O(2^{(k-2)}n^2(\log n+k))$ 시간 복잡도를 가진다. 여기서, k(≥2)는 입력 셋의 갯수를 나타내고, 각 입력에 공히 n개의 사다리꼴을 가지고 있는 것으로 가정하였다.

(증명) N을 중첩 사다리꼴 갯수라 하자. BST에 삽입 또는 삭제시 사용되는 각도 θ와 φ는 상수 시간에 계산될 수 있다. 따라서, 하나의 사다리꼴을 BST에 삽입 또는 삭제시 O(logN) 시간이 걸리므로 BST 초기화에 O(NlogN) 시간이 소요된다. 알고리즘의 while문이 실행될 때마다 k개의 사다리꼴이 추출되므로 while문은 O(N/k) 시간이 걸리고, k개의 사다리꼴을 추출하는데 O(klogN) 시간이 걸리며, 볼록 다각형을 찾는 find_CH()은 O(k) 시간이 걸린다.

그리고, while문 내부에 있는 for문은 추출된 각각의 사다리꼴에 대하여 분할을 위해 O(k) 시간이 걸리고, 분할된 O(k)개의 사다리꼴을 삽입하는데 O(klogN) 시간이 걸려, for문의 시간 복잡도는 O(k(k + klogN))이 된다. 그러나, for문의 분할을 위해 소요된 시간 O(k)를 분할 생성된 각각의 사다리꼴에 할당하고, 분할된 각각의 사다리꼴을 BST에 삽입하는데 걸리는 시간 O(klogN)을 O(k) 개의 생성된 각각의 사다리꼴에 할당할 수 있다. 따라서, for문의 시간 복잡도는 O(k(1+logN))이 된다. 따라서, 비교 알고리즘 전체 시간 복잡도는 O(NlogN + (N/k)(klogN + k + k(1 + logN)) 즉, O(NlogN)이 된다. 그러므로, [정리 2]의 중첩 사다리꼴 최대 갯수를 N에 대입하면, (그림 5)의 알고리즘에 대한 시간 복잡도는 $O(2^{(k-2)}n^2(\log n+k))$가 된다. ◆

4. 성능 비교

본 논문에서 제시된 비교 알고리즘은 최악의 경우 서론에서 보인 훑기에 근거한 비교 알고리즘보다 느린 것으로 나타났다. 그러나, 실제 응용에서는 대부분의 경우 입력 셋의 수 k는 n에 비해 매우 적은 상수로 간주된다. 또한, 하나의 소프트웨어 개발에는 여러 종류의 분할 알고리즘을 동시에 사용하지 않는다. 이러한 관점에서 두 알고리즘을 비교하면 다음과 같다.

- 셋의 수 k($<<n$)를 상수로 간주할 수 있는 경우, 두 알고리즘의 시간 복잡도는 $O(n^2\log n)$이 되어 동일한 시간 복잡도를 가진다.
- [정리 2]에서 보인 바와 같이 중첩 사다리꼴의 갯수가 kn인 경우, 제안된 알고리즘은 $kn\log(kn)$의 시간 복잡도를 갖게 되어 훑기 알고리즘보다 kn배까지 빠르다.

5. 결 론

본 논문에서는 사다리꼴로 분할된 k 도면의 관심 부분으로부터 추출된 k 셋을 비교하는 영역에 근거한 새로운 알고리즘을 제안하였다.

본 논문에서 제시된 알고리즘은 입력 셋의 수가 적을 때, 훑기 방법에 근거한 알고리즘과 동일한 시간 복잡도를 가진다. 특히, k개의 입력 셋이 동일하거나 거의 동일한 사다리꼴을 포함하고 있을 경우에는 훑기 방법에 근거한 알고리즘보다 약 배까지의 성능 향상을 얻을 수 있다. 이 두 가정은 실제 응용에 있어서 사실이다. 서론의 예를 보면, 하나의 과제 구성 모듈은 대부분 많아야 7개 정도로 구성된다. 또한, 반도체 생산을 위한 마스크 자동 생성 프로그램은 여러 가지 분할 알고리즘을 사용하기 보다는 하나의 분할 알고리즘을 사용하여 분할한 후 필요에 따라 기하 객체를 추가, 삭제, 또는 수정을 한다.

참 고 문 헌

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

[2] S. Nahar and S. Sahni, "Fast algorithm for polygon decomposition," IEEE Transactions on Computer-Aided Design, 7(4), pp.473-483, Apr., 1988.

[3] Takao Asano, Tetsuo Asano, Hiroshi Imai, "Partitioning a polygon region into trapezoids," JACM, 33(2), pp.290-312, 1986.

[4] J. Ousterhout, "Corner Stitching : A Data-Structuring Technique for VLSI Layout Tools," IEEE Transactions on CAD," 3(1), pp.87-99, 1984.

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

[6] Haejae Jung, "Algorithms for external beam dose computation," Ph.D. thesis, University of Florida, 2000.

[7] V. J. Dielissen and A. Kaldewaij, "Rectangular partition is polynomial in two dimensions but NP-complete in three," Information Processing Letters, 38(1), pp.1-6, Apr., 1991.

[8] E. Horowitz, S. Sahni and D. Mehta, "Fundamentals of Data Structures in C++," W. H. Freeman, San Francisco, 1995.
[9] D. Sleator and R. Tarjan, "Self-adjusting binary search trees," Journal of the Association for Computing Machinary, 32(3), pp.652-686, 1985.
[10] J. Bentley, T. Ottmann, "Algorithms for reporting and counting geometric intersections," IEEE Transactions on Computer, C-28, pp.643-647, 1979.
[11] I. J. Balaban, "An optimal algorithm for finding segment intersections," Proc. 11th Annual ACM Symposium on Computational Geometry, pp.211-219, 1995.
[12] K. L. Clarkson and P. W. Shor, "Applications of random sampling in computational geometry, II," Discrete & Computational Geometry, 4, pp.387-421, 1989.

정 해 재

e-mail : hjjung@cs.sungshin.ac.kr
1984년 경북대학교 전자공학과(전산전공)(공학사)
1987년 서울대학교 컴퓨터공학과(공학석사)
2000년 University of Florida 컴퓨터정보학과(공학박사)
1988년 1995년 한국전자통신연구원 선임연구원
2001년~2002년 Numerical Technologies Inc. USA, Staff Engineer
2002년~2003년 서울대학교 컴퓨터신기술연구소 객원연구원
2003년~현재 성신여자대학교 컴퓨터정보학부 초빙교수
관심분야 : 컴퓨터 알고리즘 및 자료 구조, 계산 기하, 컴퓨터 비젼