JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 21, No. 4 : 753~758 Apr. 2017

광선 슈팅 문제를 위한 볼록 레이어 트리

김수환*

A Convex Layer Tree for the Ray-Shooting Problem

Soo-Hwan Kim*

Department of Embedded Software, Busan University of Foreign Studies, Busan 46234, Korea

요 약

광선 슈팅 문제는 주어진 기하 객체들에 대해서 직선을 따라서 이동하는 광선이 처음으로 부딪히는 객체의 점을 찾는 문제이다. 광선은 보통 질의의 형태로 주어지기 때문에, 이 문제의 일반적인 해법은 다음과 같다. 먼저, 전처리 과정으로, 주어진 객체들에 대한 자료구조를 구축한다. 그 다음, 이 자료구조를 이용하여 각 질의에 대한 답을 빠르게 구한다. 본 논문에서는 x축 상에 놓인 수직 선분들 집합에 대한 광선 슈팅 문제를 고려한다. 본 논문에서는 입력으로 주어진 n개의 수직 선분들에 대해 볼록 레이어 트리라고 부르는 새로운 자료구조를 제시한다. 이것은 수직 선분들의 볼록 외피들의 레이어로 구성되는 이진 트리이다. 이 트리는 $O(n\log n)$ 시간과 $O(n)$ 공간의 알고리즘으로 구축되며 구현이 용이하다. 또한 이 자료구조를 사용하여 각 질의를 $O(\log n)$ 시간에 수행하는 알고리즘을 제시한다.

ABSTRACT

The ray-shooting problem is to find the first intersection point on the surface of given geometric objects where a ray moving along a straight line hits. Since rays are usually given in the form of queries, this problem is typically solved as follows. First, a data structure for a collection of objects is constructed as preprocessing. Then, the answer for each query ray is quickly computed using the data structure. In this paper, we consider the ray-shooting problem about the set of vertical line segments on the x-axis. We present a new data structure called a convex layer tree for n vertical line segments given by input. This is a tree structure consisting of layers of convex hulls of vertical line segments. It can be constructed in $O(n\log n)$ time and $O(n)$ space and is easy to implement. We also present an algorithm to solve each query in $O(\log n)$ time using this data structure.

키워드 : 이진 트리, 계산기하학, 볼록 외피, 광선 슈팅

Key word : Binary tree, Computational geometry, Convex hull, Ray-shooting

Received 01 February 2017, **Revised** 03 February 2017, **Accepted** 21 March 2017

* **Corresponding Author** Soo-Hwan Kim(E-mail:shkim@bufs.ac.kr, Tel:+82-51-509-6243)
Department of Embedded Software, Busan University of Foreign Studies, Busan 46234, Korea

Open Access https://doi.org/10.6109/jkiice.2017.21.4.753 print ISSN: 2234-4772 online ISSN: 2288-4165

ⓒThis is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License(http://creativecommons.org/li-censes/ by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Copyright Ⓒ The Korea Institute of Information and Communication Engineering.

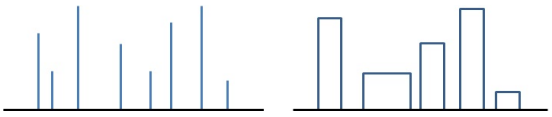
Ⅰ. 서 론

광선 슈팅 문제(ray-shooting problem)는 이차원 평면 또는 삼차원 공간 상에 주어진 객체 집합에 대해서, 질의로서 주어진 어떤 한 점에서 출발한 광선이 처음으로 부딪히는 객체의 점을 찾는 문제이다. 광선 슈팅 문제는 계산기하학(computational geometry) 분야에서 다루어지는 주요 연구 주제 중의 하나로서 최근까지 많은 연구 결과가 발표되고 있으며[1-3], 컴퓨터 그래픽스와 로보틱스 등의 응용 분야뿐만 아니라 계산기하학의 다른 문제를 해결하는 핵심 기술로 활용된다[4-6].

광선 슈팅 문제의 해는 전형적으로 전처리과정과 질의 알고리즘으로 구성된다. 전처리과정은 주어진 객체 집합으로 자료구조를 구축하는 단계를 말하고, 질의 알고리즘은 전처리과정에서 구축된 자료구조를 이용하여 효율적으로 질의에 대한 답을 찾는 단계를 말한다. 광선 슈팅 문제의 해의 효율성은, 질의 알고리즘의 수행 시간뿐만 아니라 전처리과정에서 자료구조를 구축하는데 걸리는 수행 시간과 구축된 자료구조의 기억 공간의 크기에 의해 결정된다. 자료구조의 공간복잡도와 질의 알고리즘의 시간복잡도는 보통 상충관계(trade-off)에 있다.

Pellegrini[7]는 n개의 삼각형으로 구성된 객체 집합에 대해서, $O(n^{4+\epsilon})$ 공간의 자료구조를 사용하여 효율적인 $O(\log n)$ 질의 시간을 갖는 알고리즘을 제시하였고, Agarwal과 Matoušek[8]는 삼각형 객체 집합에 대해서 거의 선형 공간의 자료구조를 사용하지만 질의 시간이 $O(n^{3/4})$로 다소 비효율적인 알고리즘을 제시하였다. 좌표축에 평행한 n개의 직사각형으로 구성된 객체 집합에 대해서, Berg[9]는 $O(n^{2+\epsilon})$ 공간의 자료구조를 사용하여 $O(\log n)$ 질의 시간을 갖는 알고리즘을 제시하였고, Sharir와 Shaul[10]는 $O(n^{1+\epsilon})$ 공간의 자료구조를 사용하면서 $O(n^{2/3})$ 질의 시간을 갖는 알고리즘을 제시하였다.

본 논문에서는 x축 상에 놓인 n개의 수직 선분들로 구성된 객체 집합에 대한 광선 슈팅 문제를 고려한다. 이들 수직 선분들의 집합은 밑변이 x축에 위치한 직사각형들의 집합을 단순화한 구조로 볼 수 있다(그림 1). 따라서 수직 선분 집합에 대한 알고리즘을 직사각형 집합에 대한 알고리즘으로 어렵지 않게 변형할 수 있다.



(a) vertical lines (b) rectangles

Fig. 1 Examples of geometric objects.

본 논문에서는 이들 수직 선분들에 대해 질의로 주어진 광선이 처음으로 부딪히는 수직 선분의 점을 찾는 알고리즘을 제시한다. 질의로 주어지는 광선은 출발점의 좌표와 이동 방향, 즉 기울기로 표현된다. 따라서 광선을 반직선(half line)으로 해석할 수 있다. 앞으로의 기술에서 광선과 반직선을 구분 없이 사용할 것이다. 본 논문에서 다루는 기하 객체, 즉 수직 선분은 앞서 연구된 기하 객체들보다 제약된 구조이기 때문에, 비효율적이거나 구현이 매우 복잡한 기존의 알고리즘을 그대로 적용하는 것이 바람직하지 않다. 본 논문에서는 기존 알고리즘보다 훨씬 구현이 용이하며 효율적인 단순한 알고리즘을 제시한다. 이를 위해 각 노드가 볼록 외피들에 대응되는 새로운 자료구조인 볼록 레이어 트리(convex layer tree)를 도입하고, 이를 구축하는 알고리즘을 제시할 것이다. 2장에서는 시간복잡도와 공간복잡도가 각각 $O(n\log n)$인 볼록 레이어 트리를 소개하고 이를 사용한 $O(\log^2 n)$ 시간의 질의 알고리즘을 제시한다. 3장에서는 볼록 레이어 트리의 공간복잡도를 $O(n)$으로 개선할 수 있음을 보이고, 이 볼록 레이어 트리를 사용하여 질의 알고리즘도 수행 시간이 $O(\log n)$으로 개선될 수 있음을 보인다. 마지막 4장에서는 결론과 앞으로의 연구 과제를 소개한다.

Ⅱ. 볼록 레이어 트리

본 논문에서 제시하는 볼록 레이어 트리는 이진 트리로서, 각 노드는 특정 구간, 즉 연속된 수직 선분들 그룹에 대한 볼록 외피 정보를 가진다. 볼록 외피는 점들을 둘러싸는 가장 작은 볼록 다각형이다. 트리의 루트는 전체 수직 선분들에 대한 볼록 외피에 대응되고, 루트의 왼쪽 자식은 수직 선분들의 왼쪽 절반에 대한 볼록 외피, 오른쪽 자식은 오른쪽 절반에 대한 볼록 외피에 대응된다(그림 2 참조). 재귀적으로, 트리의 각 내부 노드는 부모 노드에 속한 수직 선분들에 대해서 왼쪽 절

반 또는 오른쪽 절반에 대한 볼록 외피에 대응된다. 트리의 각 잎 노드는 입력으로 주어진 각 수직 선분에 대응된다. 그림 2에서 볼록 레이어 트리의 루트인 노드 A는 8개 수직 선분들 전체에 대한 볼록 외피에 대응되고, 왼쪽 자식 노드 B는 왼쪽 4개 수직 선분들에 대한 볼록 외피, 오른쪽 자식 C는 오른쪽 4개 수직 선분들에 대한 볼록 외피에 대응된다.

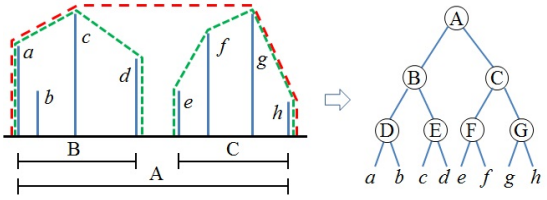


Fig. 2 Convex hulls of vertical segments and the corresponding convex layer tree.

위에서 정의한 볼록 레이어 트리의 구축에 관련된 공간복잡도와 시간복잡도를 다음 보조정리에서 기술한다.

보조정리 1. n개의 수직 선분에 대한 볼록 레이어 트리의 기억 공간의 크기는 $O(n\log n)$이다.

(증명) 수직 선분의 개수가 n이면, 볼록 레이어 트리는 n개의 잎 노드를 가지므로 총 노드 수는 $2n-1$이며 트리의 높이는 $\lceil \log_2 n \rceil$이다. 볼록 레이어 트리의 각 노드에 볼록 외피의 모든 정보를 저장한다면, 볼록 레이어 트리의 각 레벨의 노드들에 저장되는 볼록 외피들의 크기의 합은 $O(n)$이 된다. 따라서 총 기억 공간의 크기는 $O(n\log n)$이다. □

보조정리 2. n개의 수직 선분에 대한 볼록 레이어 트리를 구축하는데 필요한 수행 시간은 $O(n\log n)$이다.

(증명) 입력으로 주어진 수직 선분들을 x좌표의 오름차순으로 정렬하여 배열 S[]에 저장하고, 볼록 레이어 트리의 각 노드는 볼록 외피를 저장할 배열 CH, 왼쪽 자식에 대한 링크 left, 오른쪽 자식에 대한 링크 right로 구성된다고 하자. 배열 S[]에 저장된 선분들에 대한 볼록 레이어 트리는 다음의 재귀 함수 makeCLTree(0, n-1)를 호출하여 구할 수 있다.

```
/* 함수 makeCLTree(start, end)는 S[start]부터 S[end]까
   지 구간의 배열에 저장된 수직 선분들에 의한  볼록
   레이어 트리를 재귀적으로 구축한다.
*/
Node makeCLTree(int start, int end) {
    Node node = createNode();
    if (start == end) {
        node.CH에 수직 선분 S[start]를 저장한다;
        return node;
    }
    int mid = (start+end)/2;
    node.left = makeCLTree(start, mid);
    node.right = makeCLTree(S, mid+1, end);
    노드 node.left와 node.right에 저장된 두 볼록
        외피의 볼록 외피를 node.CH에 저장한다;
    return node;
}
```

볼록 레이어 트리의 잎 노드에 저장되는 수직 선분도 (퇴화된) 볼록 외피로 처리한다. 두 볼록 다각형의 볼록 외피를 선형 시간에 구할 수 있으므로, 위에 기술한 분할정복법(divide and conquer) 알고리즘은 볼록 레이어 트리를 $O(n\log n)$ 시간에 구축한다. 정렬된 배열 S[]의 구축에 $O(n\log n)$ 시간이 소요되므로, 총 수행 시간은 $O(n\log n)$이다. □

볼록 레이어 트리를 사용하여 질의에 대한 답을 구하는 알고리즘은 다음과 같다. 질의로 주어진 광선의 출발점은 수직 선분들보다 위쪽에 위치한다고 가정하자.

1. 루트 노드의 볼록 외피와 질의로 주어진 반직선 R과의 교차 여부를 판별한다. 만일 서로 교차하지 않으면, 어떤 수직 선분과도 교차하지 않으므로 "교차점 없음"을 출력하고 알고리즘을 종료한다.
2. 만일 루트가 잎 노드이면, 노드에 저장된 수직 선분과 R과의 교차점을 질의에 대한 답으로 출력하고 알고리즘을 종료한다.
3. 루트의 왼쪽 자식의 볼록 외피와 R, 그리고 루트의 오른쪽 자식의 볼록 외피와 R과의 교차 여부를 각

각 판별한다. 둘 중 어떤 것과도 교차하지 않으면 "교차점 없음"을 출력하고 알고리즘을 종료한다.
4. 둘 중 하나만 교차한다면, 해당 자식 노드를 루트로 하는 부트리에서 재귀적으로 알고리즘을 수행한다.
5. 만일 둘 다 교차한다면, 그 중 y좌표가 큰 교차점을 가진 볼록 외피에 대응되는 노드를 루트로 하는 부트리에서 재귀적으로 알고리즘을 수행한다.

다음의 보조정리는 위 질의 알고리즘의 시간복잡도가 $O(\log^2 n)$ 임을 보여준다.

보조정리 3. 볼록 레이어 트리를 사용한 질의 알고리즘의 수행 시간은 $O(\log^2 n)$ 이다.

(증명) 질의 알고리즘은 이진 트리의 재귀 성질을 바탕으로 설계된 것으로 옳게 동작한다는 것을 직관적으로 알 수 있다. 각 볼록 외피와 반직선과의 교차점은 바이토닉 순서(bitonic sequence)에서 이진 탐색을 수행하는 것과 같으므로 $O(\log n)$ 시간에 구할 수 있다. 최악의 경우에 이진 탐색을 트리의 모든 레벨에서 수행해야 하므로, 알고리즘의 수행 시간은 $O(\log^2 n)$ 이다. □

Ⅲ. 개선된 알고리즘

먼저 볼록 레이어 트리의 기억 공간의 크기를 개선하기 위한 방안을 소개한다. 각 내부 노드에 대응되는 볼록 외피는 자식 노드에 대응되는 볼록 외피, 즉 두 분리된 볼록 다각형의 볼록 외피이므로, 두 볼록 다각형에 대한 위쪽 접선(supporting line)이 지나는 두 접점 사이의 선분만 새롭게 볼록 외피의 에지로 추가되며, 나머지 에지들은 기존의 두 볼록 다각형의 에지들 중에서 선택된다(그림 3 참조). 그러므로 볼록 외피 전체에 대한 정보가 아니라 두 접점에 대한 정보만 내부 노드에 저장하더라도 충분하다. 잎 노드에는 입력으로 주어진 수직 선분에 대한 정보가 저장되므로, 볼록 레이어 트리의 $2n-1$개 각 노드에는 하나의 선분, 즉 두 점에 대한 정보만 저장한다. 그러면 볼록 레이어 트리의 기억 공간의 크기가 $O(n)$으로 개선된다. 따라서 다음 정리가 성립한다.

정리 1. n개의 수직 선분에 대한 개선된 볼록 레이어 트리의 기억 공간의 크기는 $O(n)$ 이다.

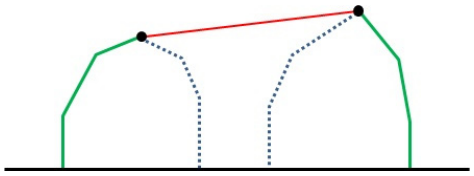


Fig. 3 The convex hull of two disjoint convex polygons

앞 절에서 기술한 재귀 함수 makeCLTree에서 각 노드에 볼록 외피를 직접 저장하는 것 대신에 두 접점 정보만 저장하고, 트리의 노드와 함께 구축된 볼록 외피를 추가로 리턴하도록 수정하면, 개선된 볼록 레이어 트리도 $O(n\log n)$ 시간에 구축된다.

개선된 볼록 레이어 트리에서 수행할 질의 알고리즘은 기본적으로 앞 절의 질의 알고리즘의 절차를 따르지만, 더 복잡한 사례 연구가 필요하다. 질의로 주어지는 반직선을 R이라고 하고, 볼록 레이어 트리의 루트 노드 $root$에 저장된 선분을 s라고 하자. 만일 $root$가 잎 노드라면 s와의 교차점을 직접 구하여 질의에 대한 답을 출력할 수 있다. 그렇지 않은 경우, 반직선 R과 선분 s의 교차 여부에 대한 상황을 다음의 4가지 경우로 분류할 수 있다(그림 4 참조).

① R과 s가 평행한 경우
② R이 s의 왼편을 지나는 경우
③ R이 s의 오른편을 지나는 경우
④ R과 s가 서로 교차하는 경우

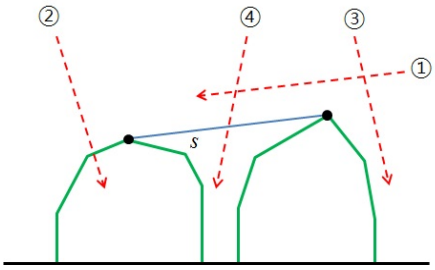


Fig. 4 Four cases for the situation of intersection between R and s.

경우 1: R과 s가 평행한 경우에 R은 노드 $root$에 대응하는 볼록 외피 CH와는 절대 교차하지 않는다. 그 이

유는, 볼록 다각형의 성질에 의해 CH의 모든 점은 s를 양 방향으로 연장한 직선의 아래쪽 영역에 놓여 있기 때문이다. 이 경우에는 더 이상의 검사가 필요하지 않으므로 알고리즘을 곧바로 종료한다.

경우 2: R이 s의 왼편을 지난다는 것은, R과 s가 직접 교차하지 않지만 s를 왼쪽으로 연장한 반직선과 R이 만나는 경우를 말한다. 이 경우는 노드 $root$의 왼쪽 자식에 대응되는 볼록 외피만 고려하면 된다. 그 이유는 반직선 R이 오른쪽 자식에 대응되는 볼록 외피와 교차하기 위해서는 반드시 왼쪽 볼록 외피를 먼저 통과해야하기 때문이다. 이 경우, $root$의 왼쪽 자식을 루트로 하는 부트리에 대해 재귀적으로 수행을 계속한다.

경우 3: R이 s의 오른편을 지나는 경우는 경우 2와 대칭적이므로, 노드 $root$의 오른쪽 자식에 대응되는 볼록 외피만 고려하면 된다. 따라서 노드 $root$의 오른쪽 자식을 루트로 하는 부트리에 대해 재귀적으로 수행을 계속한다.

경우 4: R과 s가 교차한다는 정보만으로는 R이 노드 $root$의 왼쪽 자식과 오른쪽 자식에 대응하는 각 볼록 외피 중 어느 쪽을 먼저 만나는지 판단할 수 없다. 이것은 $root$의 두 자식 노드를 모두 검사해야 한다는 것을 의미한다. $root$의 두 자식 노드를 모두 검사하면서 질의를 로그 시간에 수행하기 위해서는 부가적인 관찰과 사례 연구가 필요하다. 루트 노드 $root$의 왼쪽 자식을 $child_1$, 오른쪽 자식을 $child_2$라고 하자. 두 노드가 보유하고 있는 선분을 각각 s_1과 s_2라고 하고, 각 노드에 대응되는 볼록 외피를 CH_1과 CH_2라고 하자. 반직선 R과 두 선분 s_1과 s_2와의 교차 여부에 대한 상황을 다음의 4가지 경우로 다시 분류할 수 있다(그림 5 참조).

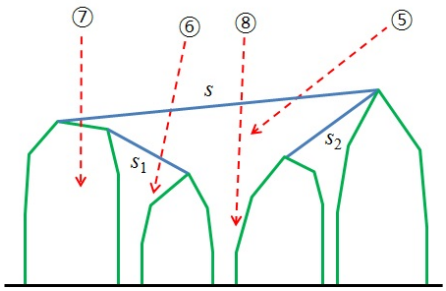


Fig. 5 Four cases for the situation of intersection between R and each of s_1 and s_2.

⑤ R이 s_1 또는 s_2와 평행한 경우
⑥ R이 s_1 또는 s_2와 교차하는 경우
⑦ R이 s_1의 왼편 또는 s_2의 오른편을 지나는 경우
⑧ R이 s_1의 오른편과 s_2의 왼편을 지나는 경우

경우 5: R이 두 선분과 모두 평행하다면, R은 어떤 볼록 외피와도 만나지 않으므로 알고리즘을 곧바로 종료한다. 만일 R이 어느 한 선분과 평행하다면, R은 그 선분이 속한 볼록 외피와는 만나지 않으므로, 다른 쪽 선분을 가진 노드를 루트로 하는 부트리에 대해 재귀적으로 수행을 계속한다.

경우 6: R이 두 선분 모두와 교차한다면, y좌표가 더 큰 교차점을 가진 선분이 속한 볼록 외피가 다른 쪽 볼록 외피보다 항상 먼저 R과 만나게 된다. 따라서 이 선분을 보유한 노드를 루트로 하는 부트리에 대해 재귀적으로 수행을 계속한다.R이 두 선분 중 하나만 교차한다면, 교차하는 선분을 보유한 노드를 루트로 부트리에 대해 수행을 계속한다.

경우 7: R이 s_1의 왼편을 지나는 경우 오른쪽 볼록 외피 CH_2를 더 이상 고려할 필요가 없다. 따라서 노드 $child_1$을 루트로 부트리에 대해 재귀적으로 수행을 계속한다. 마찬가지로 R이 s_2의 오른편을 지나는 경우는 노드 $child_2$를 루트로 부트리에 대해 재귀적으로 수행을 계속한다.

경우 8: R이 s_1의 오른편과 s_2의 왼편을 지나면, 경우 2와 경우 3에 의해 노드 $child_1$의 오른쪽 자식에 대응되는 볼록 외피와 $child_2$의 왼쪽 자식에 대응되는 볼록 외피만 고려하면 된다. 따라서 두 노드, 즉 $child_1$의 오른쪽 자식과 $child_2$의 왼쪽 자식에 대해 경우 4와 같은 방식으로 수행을 계속한다.

위의 사례 연구를 통해서, 질의 알고리즘에 분할정복법 적용이 가능하다는 것을 보였다. 경우 1부터 8까지 각 경우는 모두 상수 시간에 수행된다. 이 수행은 볼록 레이어 트리의 각 레벨마다 한 번씩 수행되므로 총 수행 시간은 $O(\log n)$이다. 지금까지의 내용으로 다음 정리가 성립한다는 것을 알 수 있다.

정리 2. x축 상에 n개의 수직 선분이 놓여 있을 때, 이

수직 선분들을 $O(n\log n)$ 시간과 $O(n)$ 공간으로 볼록 레이어 트리를 구축할 수 있다. 또한, 볼록 레이어 트리를 사용하여 질의로 주어진 광선이 처음으로 만나는 수직 선분의 점을 $O(\log n)$ 시간에 찾을 수 있다.

Ⅳ. 결 론

본 논문에서는 x축 상에 놓인 n개의 수직 선분들에 대한 광선 슈팅 문제를 해결하기 위한 효율적이며 구현이 쉬운 자료 구조인 볼록 레이어 트리를 제시하였다. 이 트리는 $O(n\log n)$ 시간과 $O(n)$ 공간으로 구축된다. 이 트리를 사용하여 광선 슈팅 문제의 질의를 $O(\log n)$ 시간에 해결하는 알고리즘을 제시하였다. 이 연구 결과는 삼각형이나 사각형 객체를 대상으로 하는 기존 연구 결과와 비교할 때, 본 논문에서 다룬 수직 선분 객체는 상대적으로 단순한 기하 객체이지만 자료구조의 공간복잡도와 질의 알고리즘의 시간복잡도 면에서 효율적이다. 앞으로의 연구 과제로는 x축 상에 놓이지 않은 수직 선분들에 대한 광선 슈팅 문제를 효율적이며 구현하기 쉬운 자료구조와 알고리즘을 개발하는 것이다.

ACKNOWLEDGMENTS

This work was supported by the research grant of the Busan University of Foreign Studies in 2017.

REFERENCES

[1] M. Pellegrini, “Ray shooting and lines in space,” in *Handbook of Discrete and Computational Geometry*, New York, CRC Press, ch. 37, pp. 599-614, 1997.

[2] D. Chen and H. Wang, “Visibility and ray shooting queries in polygonal domains,” *Computational Geometry: Theory and Applications*, vol. 48, no. 2, pp. 31-41, Feb. 2015.

[3] M. Ishaque, B. Speckmann, and C. D. Toth, “Shooting permanent rays among disjoint polygons in the plane,” *SIAM*(*Society for Industrial and Applied Mathematics*) *Journal on Computing*, vol. 41, no. 4, pp. 1005-1027, Aug. 2012.

[4] Y. Zheng and K. Yamane, “Ray-shooting algorithms for robotics,” *IEEE Transactions on Automation Science and Engineering*, vol. 10, no. 4, pp. 862-874, July 2013.

[5] A. A. Behmanesh, S. Pourbahrami, and B. Gholizadeh, “Reducing render time in ray tracing by pixel averaging,” *International Journal of Computer Graphics & Animation*, vol. 2, no. 4, pp. 15-24, Oct. 2012.

[6] D. Z. Chen and H. Wang, “Weak visibility queries of line segments in simple polygons,” *Computational Geometry: Theory and Applications*, vol. 48, no. 6, pp. 443-452, Aug. 2015.

[7] M. Pellegrini, “Ray shooting on triangles in 3-space,” *Algorithmica*, vol. 9, no. 5, pp. 471-494, May 1993.

[8] P. K. Agarwal and J. Matoušek, “On range searching with semialgebraic sets,” *Discrete & Computational Geometry*, vol. 11, no. 4, pp. 393-418, Apr. 1994.

[9] M. de Berg, *Ray Shooting, Depth Orders and Hidden Surface Removal*, LNCS, vol. 703, New York, Springer-Verlag, 1993.

[10] M. Sharir and H. Shaul, “Ray shooting amid balls, farthest point from a line, and range emptiness queries,” in *Proceedings of 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 525-534, 2005.

김수환(Soo-Hwan Kim)

1987년 서울대학교 계산통계학과 학사.
1989년 한국과학기술원 전산학과 석사.
1995년 한국과학기술원 전산학과 박사.
1992년~현재 부산외국어대학교 임베디드소프트웨어학과 교수
※관심분야 : 순차 및 병렬 알고리즘, 계산기하학, 그래프 이론 등