R-트리를 이용한 직선상의 공간 객체 검색*

김유진○ 편동현 표창우
홍익대학교
afreet@naver.com, pyo@hongik.ac.kr, peterpyun@empal.com

Searching Spatial Objects on a Straight Line using R-tree

Yujin Kim○ Dong Hyun Pyun Changwoo Pyo
Hongik University

요 약

두 지점 사이의 직선경로 위에 위치하는 공간 객체를 검색하기 위한 효율적인 알고리즘을 제안하였다. 이 알고리즘은 공간 객체의 인덱스 구조로 R-트리를 사용하였고, 한정 직사각형 (bounding rectangle)의 대각선 정보를 사용하여 검색하는 방식이다. R-트리를 구성할 때 각 노드에 대응하는 한정 직사각형의 2개의 대각선 정보를 추가해 놓으면 검색의 정확성을 높여 시간을 줄일 수 있다. 공간 객체의 밀집도가 높은 경우 10% ~ 30% 검색 시간 감축 효과를 볼 수 있다.

1. 서 론

최근 GIS 발전과 함께 RFID, GPS 등에 대한 많은 관심이 있다. 이와 함께 공간 객체를 효율적으로 다루기 위한 SAM(Spatial Access Method)에 대한 연구가 진행되었다[1]. SAM은 공간검색과 공간 객체의 범위질의 등에 사용되는 접근 방법이다. SAM은 공간 인덱싱과 클러스터링을 고려하여 구현된다. 인덱스 구조는 공간객체에 대한 검색 연산 수행 성능을 결정하는 주요 인자이다. 효율적인 인덱스 방법으로 R-트리[2], SS-트리[3], 하이브리드트리[4], 쿼드트리[5]와 같은 다양한 구조가 알려져 있다.

초기 제안된 구조들은 검색 대상은 지역 형태 또는 Grid파일 형태로 표현한다[6]. 이들을 기반으로 성능을 향상시키기 위한 이들의 변형이 등장하였다. 변형의 방법은 트리의 깊이를 줄이거나 겹치는 부분을 감소시키는 것이었다. 이와 같이 공간 객체의 표현과 그 구조의 성능은 점차 향상되었다. 그러나 직선경로 상의 공각 객체 검색이 필요한 경우, 지금까지 제안된 구조는 직선을 지역형태로 감싸서 표현하고 검색하여야한다.

본 논문에서는 특정 두 지점 사이의 유한 직선 위에 놓이는 공간 객체를 검색하기 위한 효율적인 알고리즘을 제안하였다. 공간 인덱스 중 가장 널리 사용되는 R-tree를 기반으로 대각선 정보를 추가하여 새로운 인덱스 방법을 개발하였다. 직선의 방정식을 이용하여 직선을 표현하고, 직선의 교점을 이용하여 검색을 수행한다.

본 논문에서는 2차원 데이터에 대해서 기술 할 것이다.

* 이 논문은 2008학년도 홍익대학교 학술연구진흥비에 의하여 지원되었음

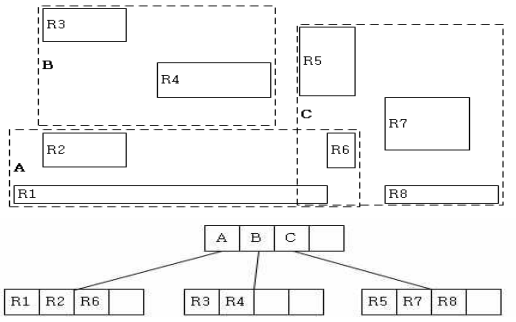
2. 관련연구

SAM의 하나로 제안된 R-트리는 널리 사용되는 공간 객체 인덱스 방법 중 하나이다. R-트리는 사각형을 이용하여 물체의 대략적인 형태를 보존하고, 사각형의 포함관계에 따라 계층적인 구조를 형성한다. B-트리의 높이 균형 트리 특성을 계승한 구조이다. 또한 저장 공간의 효율성을 위해 노드 이용률을 보장하기 위해 노드 내 최소, 최대 엔트리 수를 제한하고 있다. 각 엔트리는 최소 한정 직사각형(Minimum Bounding Rectangle, MBR) 정보를 가지고 있으며, MBR은 자식노드의 모든 객체를 포함하는 최소 사각형이다. [그림 1] 은 R-트리의 구조를 보여준다.

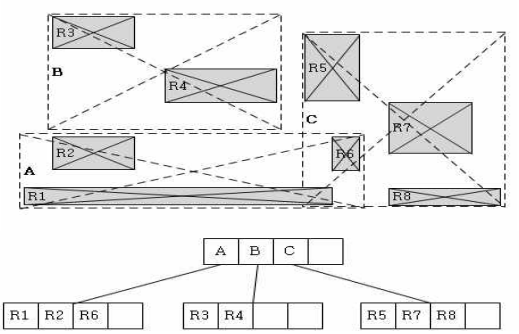
R-트리는 MBR의 포함관계에 따라 자식노드를 방문하여 검색을 수행한다. 엔트리의 사각형이 검색 대상과 겹칠 경우, 엔트리가 가리키는 자식노드를 방문하여, 자식노드의 모든 엔트리에 대해 검색 대상과 겹치는지 여부를 확인한다. 이 과정은 재귀적으로 수행된다. 검색 과정이 리프노드에 도달하면, 검색 대상과 겹치는 객체를 반환한다. 하나의 노드를 방문하면 노드내의 모든 엔트리에 대한 검사를 수행해야 하기 때문에 R-트리의 검색 성능은 방문하는 노드 수에 따라 증가한다.

R-트리는 2차 이상의 공간객체에 대해 좋은 성능을 가지지만, 공간데이터의 차수가 5차 이상이 되면 성능이 급격이 악화된다. 고차원의 공간객체를 다루고자 한다면 SS-트리가 적합한 것으로 알려져 있다[4]. R-트리가 알려진 이후 이를 발전시킨 인덱스가 많이 등장하였다 (R+-트리[7], R*-트리[8], QR-트리[9] 등).

3. 유한 직선 상의 객체 검색



[그림 1] R-tree



[그림 2] DR-tree

모든 사각형은 내부에 대각선을 가지고 있으며, 이는 직선으로 직선상 데이터 검색에 중요한 역할을 한다. R-트리 각 노드에 대응하는 MBR의 2개의 대각선 정보를 추가한 인덱스 구조를 DR-트리라고 정의한다. [그림 2]는 DR-트리를 예시하고 있다. 직선은 꺽이거나 휘어지지 않는 성질을 가지고 있다. 이런 직선의 성질 때문에 사각형 외부의 직선이 사각형을 관통하려면 사각형의 두 대각선 중 적어도 하나와 마주치게 된다[그림 3]. 따라서 대각선과 외부 직선 간 교점의 존재 여부에 따라 직선이 사각형과 겹침 여부를 결정 할 수 있다.

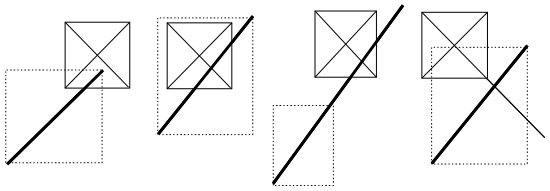
대각선과 직선은 직선의 방정식으로 표현된다. 직선의 방정식은 무한한 길이를 갖는 직선을 표현하기 때문에 평행한 두 직선이 아니라면 교점은 항상 존재 할 것이다. 따라서 교점의 존재 여부가 아니라 교점의 사각형 내부에 존재여부에 따라 겹침 여부를 결정 할 수 있다. 이 교점은 직선의 영역 안에도 포함되어야 한다. 두 직선의 방정식을 다음과 같이 정의한다.

$$y = m1x + \delta1$$
$$y = m2x + \delta2$$

여기서 m1, m2는 각 직선의 기울기, $\delta1$,$\delta2$는 각 직선의 y절편이다. 위 두식을 이용하여 교점(Ix, Iy)의 값을 구하면 아래와 같다.

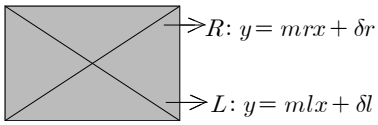
$$Ix = \frac{\delta2 - \delta1}{m1 - m2} \ , \ Iy = m1 * Ix + \delta1$$

사각형에서 제공되는 두 대각선과 검색직선을 직선의 방정식화 하여 표현하고 교점을 계산하여 검색에 이용한다.



(A) (B) (C) (D)

[그림 3] 한정 사각형과 검색직선의 관계



[그림 4] 한정 사각형의 대각선 표현

정점이 한 사각형 내에 포함되려면 모든 축에 대해서 정점의 좌표가 사각형 좌표 내에 포함되어야한다. 한 지점(x)이 범위를 가지는 두 지점(Xl, Xh)에 포함되기 위한 조건은 다음과 같다.

(Xl − x)(Xh − x) < 0

모든 축의 좌표에 대해 위의 식이 성립되면, 정점(x)는 사각형 내에 존재한다.

지금까지 제시한 정보를 이용하기 위해서 R-트리의 사각형은 대각선 정보를 추가로 포함하고 있어야 한다. 형태는 아래와 같다.

Rectangle<xh, xl, yh, yl, mr, ml, δr, δl>

R-트리와 마찬가지로 네 점의 정보가 필요하다(xh, xl, yh, yl,). 오른쪽으로 증가하는 기울기를 가진 대각선의 정보 mr, δr와 감소하는 기울기를 가진 대각선의 정보 ml, δl이 추가 되었다[그림 4].

주어진 두 지점을 검색 직선으로 저장 할 수 있는 구조가 필요하다. 이는 Rectangle 구조에서 두 대각선 중 하나의 대각선만 유효한 값을 가지게 하여서 표현할 수 있다. 또는 직선에 대한 자료구조를 따로 정의하여도 문제 되지 않는다. 아래와 같이 기술 한다.

Line <xh, xl, yh, yl, m, δ>

이제 변경된 자료구조를 이용하여 검색을 수행하기 위한 검색 조건을 확인한다. 대각선과 직선을 방정식화 하여 표현하고, 교점의 포함여부에 따라 겹침을 확인한다. 방정식화 하여 표현된 직선은 무한 직선이기 때문에 범위가 지정되어 있어야 하며 범위에 대한 검사가 수행되어야 한다. 직선과 대각선의 기울기가 평행인 경우를 제외하면, 직선과 대각선을 가진 사각형과의 관계는 [그림 3]의 총 4가지로 정리 된다.

직선의 두 끝점을 각각 (a, b), (p, q)로 둔다. 사각형

의 정보는 위에서 제안한 자료구조를 이용한다. 두 대각선과 직선의 교점은 각각 Ir(Ir.x, Ir.y)와 Il(Il.x, Il.y)로 계산된다. [그림 3]의 (A)와 같이 직선의 끝점이 사각형 내부에 존재한다면, 이 직선이 사각형과 겹치는 것은 자명하다. 조건식은 아래와 같다.

(1) (xh - a)(xl - a) < 0 and
    (yh - b)(yl - b) < 0
(2) (xh - p)(xl - p) < 0 and
    (yh - q)(yl - q) < 0

[그림 3]의 (C)와 같이 교점이 직선 영역 외부에 존재하면 겹치는 것이 아니다. 또한 교점이 검색 대상 밖에 존재한다면([그림 3] 의 (D)) 해당 객체는 검색되지 않는다. 직선 영역 내에 존재하면서 대상 객체 내에 존재 할 조건은 다음과 같다.

(3) { (Ir.x - a)(Ir.x - p) < 0 and
    (Ir.y - b)(Ir.y - q) < 0 } and
    { (Ir.x - xh)(Ir.x - xl) < 0 and
    (Ir.y - yh)(Ir.y - yl) < 0}
(4) { (Il.x - p)(Il.x - p) < 0 and
    (Il.y - q)(Il.y - q) < 0 } and
    { (Il.x - xh)(Il.x - xl) < 0 and
    (Il.y - yh)(Il.y - yl) < 0 }

위 조건을 만족시키는 경우는 [그림 3]의 (B)와 같은 형태이다. (3), (4)의 조건을 검사하기 위하여 교점을 계산하여야 한다. 이 때, 직선과 대각선의 기울기가 평행이라면, 교점은 존재 하지 않는다. 따라서 기울기가 평행이 아닐 때 (3), (4) 조건을 검사한다. (1), (2)가 성립하지 않는 경우에는 (3), (4)의 조건을 검사하여 겹침 여부를 검사한다.

4. 검색 알고리즘

검색 알고리즘은 Search()함수와 Overlap(), ContainPoint()함수로 구성된다. Search()함수에서 검색을 시작하고, Search()함수 내에서 Overlap()함수를 호출한다. Overlap()함수를 사각형과 직선과의 겹침 여부를 검사한다. 이때, 사각형의 대각선과의 교점을 이용한다. 교점의 포함 여부를 확인하기 위해 ContainPoint()함수를 이용한다. Search()의 매개변수인 N과 L에 루트노드와 검색 직선을 할당하여 함수를 호출한다.

Search()함수의 동작은 다음과 같다. 노드 N의 첫 번째 엔트리를 E에 할당하고 리프노드가 아니라면, E의 사각형 R과 탐색직선 L이 겹치는지를 확인한다. 만약 겹친다면 E의 자식노드를 N으로 L을 검색직선으로 두고 Search()함수를 호출한다. 만약 겹치지 않는다면 N의 다음 엔트리에 대한 검사를 수행한다. 이와 같이 N의 모든 엔트리에 대해 검색을 수행한다. 만약 N이 리프노드일 경우, 검색직선 L과 N의 엔트리 E의 사각형 R이 검색직선 L과 겹친다면 결과를 반환한다.

```
Search(Node *N, Line *L) {
  Entry E;
  Rectangle R;
  for( E = N의 첫 번째 엔트리
        ; N의 마지막 엔트리일 때 까지
        ; E = E의 다음 엔트리) {
    if ( N is not leaf ) {
      R = E의 Rectangle;
      if ( Overlap (R, L) )
        Search (R.childNode, L);
    }
    else { // N is leaf Node
      R = E의 Rectangle;
      if ( Overlap (R, L) )
        return R or count;
    }
  }
}
```

함수 Overlap()은 한정 사각형과 탐색직선을 매개변수로 받아 동작한다. 사각형 R이 검색직선의 시작점이나 끝점을 포함하고 있다면 겹치는 것이다. 포함하고 있지 않다면, 대각선과의 교점의 기울기가 평행한지 여부를 검사한다. 평행하지 않을 때, 교점을 계산하고, 해당 교점의 포함여부를 알아본다. 교점 Ir이 직선의 영역에 속하는지 보고, 속하면 이 교점이 R에 포함되는지 확인한다. 만약 포함된다면 사각형R과 직선L은 겹치는 것이다. 만약 포함되지 않는다면, Il에 대해 Ir과 같은 방법으로 직선과 사각형의 포함여부를 확인한다. 교점이 포함된다면 겹치는 것이므로 TRUE를 반환하고, 포함되지 않는다면 직선과 사각형은 겹치는 것이 아니므로 FALSE를 반환한다.

```
Overlap( Rectangle R, Line L) {
  if ( ContainPoint( R, L.a, L.b) or
     ContainPoint( R, L.p, L.q))
    return TRUE;

  if (R.mr != L.m) {
    Ir.x = (δr - δ) / (m - mr);      //오른쪽으로 증가하는
    Ir.y = m*Ir.x + δ;               //대각선과의 교점
```

```
    if( ContainPoint( L, Ir.x, Ir.y) &&
        ContainPoint( R, Ir.x, Ir.y))
      return TURE;
  }
  if (R.ml != L.m) {
    Il.x = (δl − δ) / (m − ml);      //왼쪽으로 증가하는
    Il.y = m*Il.x + δ;    //대각선과의 교점
    if( ContainPoint(L, Il.x, Il.y) &&
       ContainPoint( R, Il.x, Il.y))
      return TURE;
  }

  return FALSE;
}
```

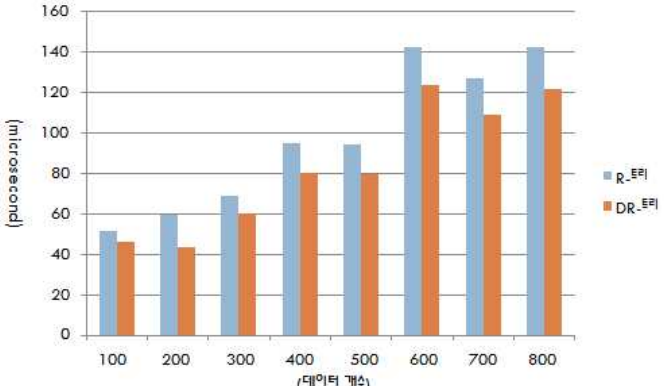
함수 Contain()은 Overlap()함수에 의해 호출되며 한정 사각형R과 정점 x, y좌표를 매개변수로 받아 동작한다. R에 정점이 포함되는지 확인한다. X축에 대해 정점의 x좌표가 R의 x 좌표 사이에 존재하고, Y축에 대해 정점의 y좌표가 R의 y좌표 사이에 존재한다면, 사각형 내부에 정점이 포함되는 것이므로 TRUE를 반환한다. 그렇지 않을 경우 FALSE를 반환한다.

```
ContainPoint(Rectangle or Line R,
                float x, float y) {
  if ((R.xh - x)*(R.xl - x) < 0 and
    (R.yh - y)*(R.yl - y) < 0 )
    return TRUE;
  else return FALSE;
}
```

R-트리의 검색 복잡도는 n이 노드의 개수일 때, O(log n)이다. DR-트리는 R-트리의 구조를 기반으로 확장하였으나, 트리깊이나 구성방법을 변화시킨 것이 아니기 때문에 검색 복잡도는 O(log n)으로 동일하다.

R-트리의 검색성능은 방문하는 노드의 수에 따라 결정된다. DR-트리의 경우, 정확도의 증가로 인해 검색 시 방문해야 할 노드의 개수가 감소한다.

R-트리와 DR-트리에 동일한 검색 알고리즘을 적용하였을 때, 실험결과는 [그림 5]와 같다. DR-트리의 검색 시간이 10~30% 감축되었다. 직선 정보를 가지고 있는 DR-트리와 달리 R-트리는 직선 상 검색을 위해서 직선 정보를 계산해야 한다. R-트리는 노드를 방문 할 때 마다 직선 정보를 계산해야 하는 오버헤드가 존재하기 때문에 [그림 5]와 같은 결과가 나온 것으로 보인다.



[그림 5] R-트리와 DR-트리의 검색속도

5. 결 론

5차원 미만의 공간에서 객체를 효과적으로 인덱스 할 수 있는 R-트리를 확장하여 새로운 알고리즘을 제시하였다. R-트리의 구조에 대각선 정보를 추가 저장 하였고, 추가 정보를 통해서 직선 검색을 효율적으로 수행할 수 있었다. R-트리의 기본 형태를 유지하였기 때문에 구현이 용이하고, R-트리의 여러 변형에 적용시켜 성능을 더욱 향상 시킬 수 있을 것이다.

제안된 새로운 알고리즘은 두 지점 간 직선 탐색에 효율성과 정확성을 제공한다. 이를 이용하면 지도상의 위치표현, 센서배치, 천문학에서의 활용 등에서 두 지점 사이의 탐색에 활용하여 긍정적이 효과를 기대할 수 있다. 또한 GPS를 이용한 위치기반 서비스(LBS)나 GIS의 RFID를 포함한 분야에 응용되어 사용될 수 있을 것이다.

6. 참고문헌

[1] A.U. Frank. Properties of Geographic Data: Requirements for Spatial Access Methods. *Advances in Spatial Database*. 525:225-234, 1991

[2] A. Guttman. R-tree: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD international Conference on Management of Data,* pages 47-57, 1984

[3] D.A. White, R. Jain. Similarity Indexing with the SS-tree. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 516-523, 1996

[4] K. Chakrabarti, S. Mehrotra. The Hybrid tree: An Index Structure for High Dimensional Feature Spaces. In

Proceedings of the 15th International Conference on Data Engineering, pages 440-447, 1999

[5] R. A. Finkel, J. L. Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica,* 4:11-9, 1974

[6] H. Samet. Spatial Data Structures. *Modern Database System: the Object Model, interoperability, and beyond*, pages 361-385, 1995

[7] T.K. Sellis, N. Roussopoulos, C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the 13th International Conference on Very Large Data Bases,* pages 507-518, 1987

[8] N. Beckmann, H.-P. Kriegel. R. Schneider and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangle. *In Proceedings ACM SIGMOD Conference*, pages 322-331, 1990

[9] Y.-C. Fu, Z.-U. Hu, W. Guo, D.-R. Zhou, QR-Tree: A Hybid Spatial Index Structure. In *Proceedings of thre Second International Conference on Machine Learning and Cybernetics*, pages 459-463, 2003