공간 해싱: 공간 객체에 대한 동적 색인 구조†

김용환*, 황수찬
한국항공대학교 컴퓨터공학과

Spatial Hashing : Dynamic Index Structure for Spatial Objects

Yonghwan Kim*, Soochan Hwang
Department of Computer Engineering, Han-Kuk Aviation University

요 약

최근에 활발히 연구되고 있는 지리 정보 시스템 등은 2차원 이상의 공간 속성을 갖는 공간 객체들로 구성되며 데이타 양이 매우 방대하여 효율적인 공간 색인 기법이 요구되고 있다. 그러나, 기존의 공간 색인 기법들은 공간 객체의 크기와 밀도 차이, 공간 연산의 종류에 따라 각각 큰 성능차를 보이며 때로는 이용이 불가능한 경우도 있다. 이와 같은 문제점들을 해결하기 위해서는 공간 객체의 크기와 밀도 차이에 독립적인 하나의 색인 구조로 다양한 공간 연산들을 효율적으로 지원할 수 있는 공간 색인 기법이 필요하다. 본 논문에서는 이와 같은 문제를 해결할 수 있는 새로운 공간 색인 기법인 공간 해싱(spatial hashing)을 제안하고 관련 연산들을 정의하였다. 공간 해싱은 각 객체의 영역을 MBR로 단순화하고 그 MBR의 좌상점(Left-Top point)과 우하점(Right-Bottom point)만을 이용해 객체의 영역 정보와 위치 정보를 확장성 해싱을 이용하여 유지하는 색인 기법이다.

1. 서론

지리정보 시스템 같은 공간 데이타베이스 응용 분야는 공간 객체로 구성된 데이타베이스를 유지한다. 이러한 공간 객체에 필수적인 연산으로는 공간 객체의 삽입, 삭제, 영역 탐색 및 공간 조인(spatial join) 등이 있다. 공간 연산은 객체의 어떤 성질에 대한 연산인가에 따라 크게 기하학적(geometrical) 연산과 위상학적(topological) 연산으로 구분한다. 기하학적 연산은 공간 객체 자체의 기하학적 성질에 근거한 연산으로서 최소 거리, 넓이, 둘레, 부피, 높이, 기울기 등의 연산을 들 수 있다. 위상학적 연산은 각 공간 객체의 위상학적 특성에 근거한 연산으로 포함(contain), 교차(intersect), 겹침(overlap) 등의 조건에 의한 연산을 들 수 있다[1].

지금까지 다차원 공간 데이타에 대한 연산들을 효율적으로 지원하기 위해서 다양한 공간 색인 기법들이 연구되어 왔다[2, 3, 6, 7, 8, 9, 10]. 하지만 기존의 공간 색인 기법들은 공간 객체의 크기 차이와 밀도(응집도) 차이에 그 성능이 크게 영향을 받거나 색인을 구성할 수 없는 경우가 있다. 또한 공간 연산의 종류에 따라 각각 큰 성능차를 보이며 때로는 이용이 불가능한 경우도 있다. 따라서 다양한 공간 연산들을 모두 지원하기 위해서는 하나의 응용에 대해 여러 개의 공간 색인 구조를 구성해야 한다는 부담이 있다.

이와 같은 기존 공간 색인 기법으로는 공간 데이타베이스에 대한 다양한 연산을 효율적으로 지원할 수 없으며 이로 인해 응용 시스템의 전체 성능이 크게 저하되고 색인의 구현 비용이 높아지는 원인이 된다. 이러한 문제를 해결하기 위해서는 공간 객체들의 밀도와 크기에 제한을 받지 않고 다양한 공간 연산들을 효율적으로 지원할 수 있는 색인 기법이 필요하다. 따라서 본 논문에서는 이와 같은 요구를 만족시킬 수 있는 공간 색인 기법으로 공간 해싱(spatial hashing) 기법을 제안한다.

본 논문에서 제시한 공간 해싱 기법은 효율적인 색인 구조를 위해 각 객체의 영역을 MBR(Minimum Bounding Rectangle)로 단순화하고 그 MBR의 좌상점(Left-Top point)과 우하점(Right-Bottom point)만을 이용해 객체의 영역 정보와 위치 정보를 확장성 해싱(extendible hashing)을 이용하여 표현하는 기법이다.

본 논문의 구성은 2장에서 공간 해싱의 기본 구조를 설명하고 3장에서 이 색인 구조를 이용한 공간 연산에 대해 기술한다. 마지막으로 4장에서 결론 및 추후 연구 방향을 제시한다.

2. 공간 해싱(spatial hashing)

본 논문에서 제안하는 공간 색인 구조로서의 공간 해싱(spatial hashing)은 다음과 같은 특성을 갖는다.

- 삽입과 삭제에 대해 동적으로 색인 구조를 유지한다.
- 공간 순서화 기법을 적용해 색인 구조를 안정화한다.
- 죽은 공간, 비정방형성에 의한 불필요한 I/O를 제거한다.
- 본래 객체의 영역 정보, 위치 정보를 유지하므로 공간 변환을 하지 않는다.

이와 같은 특성을 가진 공간 해싱은 객체를 Z-순서화 하여 그 Z-값을 확장성 해싱(extendible hashing)[4]의 모조키로 사용한다.

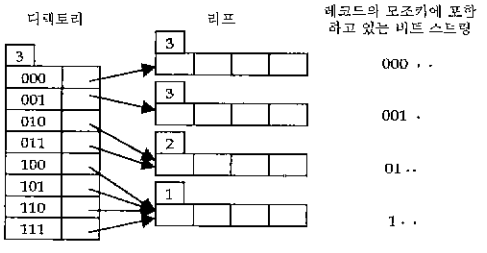

그림 1 확장성 해싱 화일

확장성 해싱 화일은 두 단계의 조직을 사용한다는 면에서 동적

† 이 논문은 한국과학재단의 '98 특정기초연구비의 지원에 의한 것임

해싱 화일과 비슷하다. 확장성 해싱은 디렉토리(directory)와 리프의 집합을 사용한다. 각 리프는 레코드들이 저장되는 버켓과 정수값을 갖는 헤더로 구성되고 디렉토리는 정수값(d)을 갖는 헤더와 리프에 대한 2^d개의 포인터로 구성된다. 물론 이 포인터 값은 반드시 상이해야 되는 것은 아니다. 그림 1은 통상적인 형태를 보여주고 있다.

공간 해싱은 각 공간 객체에 대해서 최소 경계 사각형(MBR, Minimum Bounding Rectangle)을 사용하여 단순화하며 확장성 해싱과 같이 디렉토리와 리프의 두 레벨로 되어있다.

리프는 (N, I, 꼭지점 구별자)형태를 이룬다. 여기서 N은 리프 페이지에 있는 엔트리의 수를 나타내며 I는 n차원의 사각형으로써 객체 MBR의 영역을 의미한다.

$$I = (I_0, I_1, I_2, \ldots, I_{n-1})$$

여기서 n은 차원의 수이며 각 I_i는 객체 MBR의 각 꼭지점 중 원점에 가장 가까운 꼭지점부터 각 축 상의 최대점에 위치한 꼭지점을 의미한다. 예를 들어 2차원인 경우 객체 MBR의 최하점(LB, Left Bottom)이 원점에 가장 가까울 때, 우하점(RB, Right Bottom)이 최하점으로부터의 X축으로 최대점에 위치한다. 또한 좌상점(LT, Left Top)이 좌하점부터의 Y축 상의 최대점에 위치한다. 그러므로 이 때 I는 아래와 같다.

$$I = (LT, RB)$$

꼭지점 구별자는 현재의 엔트리가 나타내는 값이 I의 $I_0 \sim I_{n-1}$중 어떤 점인지를 의미한다.

디렉토리는 (d, Z', LP)의 형태를 갖는다. 여기서 LP는 리프 페이지에 대한 포인터를 의미한다. Z'은 현재의 영역 사각형을 Z-순서화 방법에 따라 분할하였을 때 현재의 영역이 어떤 영역을 의미하는지를 나타낸다. Z-값은 그림 2처럼 전체 영역을 4분하였을 때 좌상을 00, 좌하를 01, 우상을 10, 우하를 11, 이렇게 2비트(x,y)로 나타낸다. 확장성 해싱에서의 모조키처럼 사용된다. d는 디렉토리의 크기를 나타내는 헤더값이다. 또한 현재 영역 사각형의 순서화 레벨을 의미한다.

이렇게 정의된 공간 해싱 화일을 구성하기 위해 각 MBR의 I를 구성하는 I_i점들에 대하여 Z-순서화를 사용해 순서화를 수행한다. 이 때 순서화는 d/n만큼의 레벨까지 수행한다. 본 논문에서는 이 레벨을 '리프 레벨'이라 정의하였다. 여기서 d는 디렉토리의 헤더값을, n은 차원을 의미한다. 이렇게 순서화를 수행하여 얻은 Z-값을 디렉토리의 모조키로 사용한다. I_i점들은 디렉토리에서 모조키가 위치한 엔트리의 포인터가 가리키고 있는 리프 페이지에 저장된다.

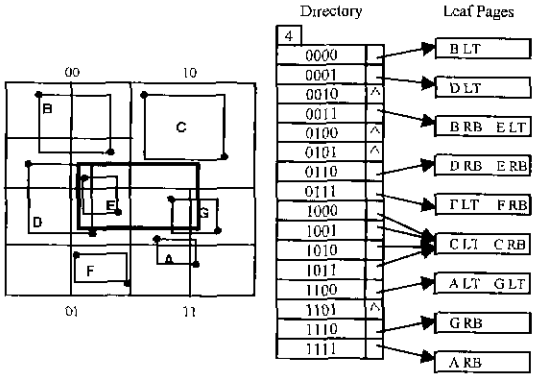

그림 2 공간 해싱의 예

그림 2는 이러한 공간 해싱의 예이다. 객체 B의 경우, d가 4이므로 2레벨까지 Z-순서화를 수행하면 객체 B의 LT는 Z-값이 0000이 되고, RB는 0011이 된다. 디렉토리에서 모조키 0000이 있는 엔트리의 포인터가 가리키는 리프 페이지에는 객체 B의 LT가, 0010이 가리키는 리프 페이지에는 RB가 저장되어 있다.

3. 공간 연산

공간 객체의 삽입과 삭제 또는 공간 연산을 위해 주어진 탐색 윈도우 등에 대해서 공간 해싱은 이 새로운 객체 MBR의 LT와 RB를 위한 Z-값을 구해야 한다. Z-값은 대상 공간의 원점과 각 축과의 최대 길이에 의해 원하는 레벨까지 쉽게 구할 수 있는데 이 값이 바로 디렉토리의 모조키를 의미하게 된다.

3.1 공간 해싱의 삽입/삭제

공간 해싱에서의 삽입은 기존의 순서화 되지 않은 색인 구조에서 필요한 트리상의 삽입될 위치를 찾기 위한 겹침 영역 탐색이나 내부 노드 영역 사각형의 부가적인 최적화가 필요하지 않다. 공간 해싱은 삽입된 공간 객체의 MBR의 좌상이나 우하점이 위치할 Z-영역을 나타내는 디렉토리의 모조키가 지시하는 리프 페이지에 여유 공간이 없으면 그 영역에 대한 분할이 수행된다. 또한 디렉토리의 크기를 2^n배로 늘리고 각 리프 페이지에 대한 포인터들도 조절한다. 여기에서 n은 차원을 의미한다.

삭제의 경우는 확장성 해싱에서와 같이 버디 버켓 개념을 이용한다. 삭제될 공간 객체 MBR의 좌상,우하점이 제거되었을 때 버디 페이지에 있는 엔트리와의 합이 임계치 이하이면 병합한다. 또한 상위 Z-영역들에 하나 이하의 리프 페이지만 존재하면 디렉토리의 크기를 2^n배로 줄인다.

3.2 공간 해싱의 공간 탐색 연산

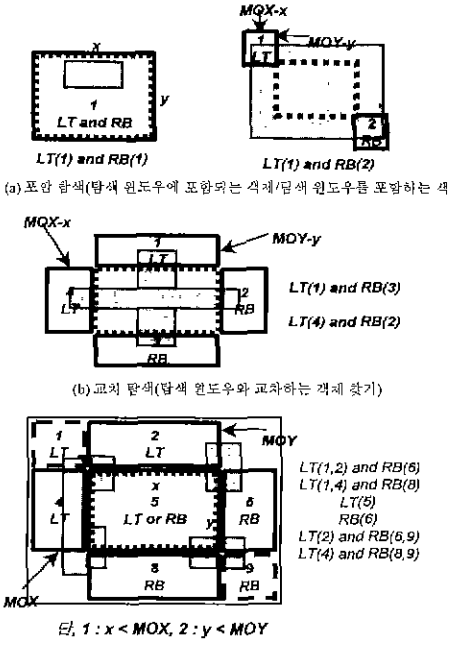

그림 3 '탐색-영역'의 정의

일반적으로 공간 탐색 연산은 여과(filtering) 남개의 점제

(refinement) 단계로 구성된 2단계 질의 처리 기법으로 수행된다 [5]. 공간 해싱의 공간 탐색 연산은 이러한 2단계 질의 처리 기법에서 여과 단계에 해당한다고 볼 수 있다. 2차원인 경우 포함, 겹침, 교차와 같은 탐색 조건을 탐색 윈도우와 각 객체 MBR의 좌상과 우하점에 의해 검사한다. 이 때 탐색은 미리 정해진 영역만을 탐색하게 된다. 그림 3의 굵은 실선으로 표시된 사각형의 영역들은 굵은 점선으로 표시된 '탐색 윈도우'에 대하여 포함, 교차, 겹침의 유무를 좌상, 우하점으로 획인 할 수 있기 위한 탐색 대상 영역을 나타낸다. 본 논문에서는 이와 같이 미리 정의된 탐색 대상 영역을 '탐색-영역'이라 정의하였다.

공간 해싱의 탐색 연산은 포함, 교차, 겹침 연산 모두 그림 3에서 정의한 '탐색-영역'의 기법에 의해 포함 연산만으로 구성할 수 있다. 예를 들어 교차 연산인 경우 그림 3-(b)에 표시한 굵은 실선으로 된 4개의 '탐색-영역'에 대한 4개의 포함 연산으로 구성된다.

영역 탐색을 위해 그림 3에서 정의된 '탐색-영역'을 검색하기 위해서 공간 해싱의 탐색 기법은 2단계로 구성된다. 첫 번째로 검색할 영역을 최소화하기 위해서 탐색-영역의 꼭지점을 Z-순서화 했을 때의 Z-값(탐색 영역의 LT는 00, LB는 01, RT는 10, RB는 11이다)과 전체 영역을 분할하였을 때 각 꼭지점이 위치한 영역의 Z-값을 가지고 검색해야 할 영역을 계산하는 단계 즉, 키를 생성하는 단계이다. 본 논문에서는 이 단계를 '전-단계'라 정의하였다. 전-단계를 수행하는 알고리즘은 다음과 같다.

① 탐색-영역이 4개의 Z-영역에 걸쳐 존재할 때까지 전체 영역을 분할한다. 만약 리프 레벨까지 분할했는데도 탐색-영역이 4개의 Z-영역에 존재하지 않으면 리프 레벨에서의 각 꼭지점의 Z-값이 탐색할 영역 Z'이 된다. 그림 2에서 굵은 실선으로 된 사각형은 탐색-영역을 의미한다. 이 예에서는 처음 분할할 때 탐색영역이 4개의 Z-영역에 존재한다.

② 리프 레벨까지 분할하여 각 꼭지점(LT, LB, RT, RB)이 위치한 영역의 Z-값을 구한다. 예에서 RB의 Z-값은 '1110'이다.

③ ②번에서 구한 Z-값 중에서 ①번에서 분할한 레벨까지의 Z-값 Z'_0과 그 이외의 부분 Z_1를 구분한다. 예에서 Z'_0(RB)는 '11'이고 Z_1(RB)는 '10'이다.

④ 탐색 영역의 꼭지점을 Z-순서화 했을 때의 값(V)과 ③번에서 구한 Z와의 비트 연산을 수행하는 다음 법칙을 적용하여 탐색할 영역 Z'을 계산한다.

- $V_x \oplus Z_x = 1$ 이면 $temp_x = Z_x$
- $V_x \oplus Z_x = 0$ 이면 $temp_x = 0, 1$
- $temp_y$ 역시 마찬가지이다.
- $Z'_1 = temp_x temp_y$
- $Z' = Z'_0 Z'_1$

그림 2의 예에서 V_x(RB) $\oplus$ Z_x(RB) = 1 $\oplus$ 1 = 0이므로 $temp_x$(RB) = Z_x(RB) = 0, 1이고, V_y(RB) $\oplus$ Z_y (RB) = 1 $\oplus$ 0 = 1이므로 $temp_y$(RB) = Z_y(RB) = 0이다. 그래서 Z'_1 (RB) = $temp_x$(RB) $temp_y$(RB) = 00, 10이다. 따라서, Z'(RB) = Z'_0(RB) Z'_1(RB) = 1100, 1110이 된다. 마찬가지로 Z'(LT) = 0011이고, Z'(LB) = 0110이며, Z'(RT) = 1001, 1011이 된다.

두 번째로는 전-단계에서 구한 탐색할 영역의 Z-값을 모조키로 하여 리프 페이지에서 객체를 찾아 포함 여부를 결정하는 단계이다. 이 단계를 본 논문에서는 '후-단계'라 정의하였다. 알고리즘은 다음과 같다.

① 전-단계에서 계산된 영역 Z'을 모조키 K로 사용한다. 그림 2의 예에서 모조키 K는 0011, 0110, 1001, 1011, 1100, 1110이다.

② 디렉토리에서 K가 위치한 곳에 있는 포인터를 찾는다.

③ 그 포인터를 따라 리프 페이지로 이동한다. 예의 디렉토리에서 0011이 위치한 곳의 포인터는 B.RB, E.LT가 저장되어 있는 리프 페이지를 지시하고 있다.

④ 그 리프 페이지에 있는 점들 중에서 실제로 탐색-영역에 포함되는 엔트리를 추출한다. 예의 경우 각 키에 대해서 탐색 영역에 포함된 점들을 계산해보면 E.LT, E.RB, G.RB가 추출되어 진다.

⑤ 추출된 엔트리들 중에서 LT와 RB가 모두 있는 객체를 찾는다. 예에서는 객체 E가 검색되어 진다.

4. 결론

기존의 공간 색인 구조는 공간 객체들의 크기와 밀도 차이, 공간 연산의 종류 등에 이용이 제한이 된다. 따라서 하나의 질의에 대해서도 대부분 둘 이상의 공간 색인 구조를 구성해야 한다. 또한 색인 구조를 트리 형태로 구성할 때, 트리의 노드를 방문해야 하는 오버헤드를 감수해야만 한다. 본 논문에서는 이러한 문제점을 해결하기 위해 공간 객체의 영역 정보와 위치 정보를 모두 유지하면서 해싱의 특성인 빠른 접근을 보장하는 공간 해싱을 제안하였다.

포함과 교차 연산 및 겹침 연산에 대해서는 전체 탐색 영역을 제한하여 미리 정의된 '탐색-영역'만을 탐색하므로 위상학적 영역 탐색 연산에서 높은 성능을 보일 것이고, 순서화의 특성을 이용하므로 공간 조인에서도 좋은 성능이 기대된다.

공간 해싱은 리프 페이지의 크기를 작게 하면 Z-영역의 레벨이 커지면서 디렉토리의 깊이도 증가한다. 반대로 리프 페이지의 크기를 크게 하면, 디렉토리의 깊이가 감소하지만 불필요한 객체에 대한 I/O가 발생할 수 있다. 따라서 최적화된 리프 페이지의 크기에 대한 추가적인 연구가 필요하다.

참고문헌

[1] R.Laurini and D.Thompson, Fundamentals of Spatial Information System, Academic Press, 1992

[2] Hanan Samet, The Design and Analysis of Spatial Data Structures, Addison Wesley, 1990

[3] Hongjun Lu and Beng-Chin Ooi, "Spatial indexing: Past and Future", IEEE Data Engineering, Vol.16, No 3, pp 16-22, 1993

[4] Ronald Fagin, Jurg Nievergelt, Nicholas Pippenger, H. Raymond Strong, "Extendible Hashing - A Fast Access Method for Dynamic Files", TODS 4(3), pp.315-344, 1979

[5] H.P.Kriegel, Thomas Brinkhoff, and Ralf Schneider, "Efficient Spatial Query Processing in Geographic Database System", IEEE Data Engineering Bulletin, Vol 16, No 3, pp 10-15, 1993

[6] Hongjun Lu and Beng-Chin Ooi, "Spatial indexing: Past and Future", IEEE Data Engineering, Vol 16, No 3, pp 16-22, 1993

[7] B Seeger and H P Kriegel, "The Buddy-tree: An Efficient and Robust Access Method for Spatial Database Systems", Proc Of 16th Intl Conf VLDB, pp 590-601, 1990

[8] Hanan Samet, Applications of Spatial Data Structures, Addison Wesley, 1995

[9] B.Seeger and H.P.Kriegel, "Techniques for Design and Implementation of Efficient Spatial Access Method", Proc. Of 14th Intl On VLDB, pp 10-17, 1988

[10] M L Lo and C V Ravishankar, "Spatial Joins Using Seeded Trees", Proc. ACM SIGMOD '94, pp 209-220, 1994