SQMR-tree: 대용량 공간 데이타를 위한 효율적인 하이브리드 인덱스 구조

SQMR-tree: An Efficient Hybrid Index Structure for Large Spatial Data

신 인 수* 김 정 준** 강 홍 구*** 한 기 준****
In-Su Shin Joung-Joon Kim Hong-Koo Kang Ki-Joon Han

요 약 본 논문에서는 기존에 제시된 MR-tree와 SQR-tree의 장점을 결합하여 대용량 공간 데이타를 보다 효율적으로 처리할 수 있는 하이브리드 인덱스 구조인 SQMR-tree(Spatial Quad MR-tree)를 제시한다. MR-tree는 R-tree에 R-tree 리프 노드를 직접 접근해주는 매핑 트리를 적용한 인덱스 구조이고, SQR-tree는 SQ-tree (Spatial Quad-tree)와 SQ-tree의 리프 노드마다 실제로 공간 객체를 저장하는 R-tree가 결합된 인덱스 구조이다. SQMR-tree는 SQR-tree를 기본 구조로 SQR-Tree의 R-tree마다 매핑 트리가 적용된 형태를 가진다. 따라서, SQMR-tree는 SQR-tree와 같이 공간 객체가 여러 R-tree에 분산 저장되며 질의 영역에 해당하는 R-tree만 접근하면 되기 때문에 공간 질의 처리 비용을 줄일 수 있다. 또한, SQMR-tree는 MR-tree와 같이 매핑 트리를 통해 트리 검색 없이 R-tree 리프 노드의 빠른 접근이 가능하기 때문에 검색 성능을 향상시킬 수 있다. 마지막으로 실험을 통해 SQMR-tree의 우수성을 입증하였다.

키워드 : 대용량 공간 데이타, 하이브리드 인덱스 구조, R-tree, SQMR-tree

Abstract In this paper, we propose a hybrid index structure, called the SQMR-tree(Spatial Quad MR-tree) that can process spatial data efficiently by combining advantages of the MR-tree and the SQR-tree. The MR-tree is an extended R-tree using a mapping tree to access directly to leaf nodes of the R-tree and the SQR-tree is a combination of the SQ-tree(Spatial Quad-tree) which is an extended Quad-tree to process spatial objects with non-zero area and the R-tree which actually stores spatial objects and are associated with each leaf node of the SQ-tree. The SQMR-tree consists of the SQR-tree as the base structure and the mapping trees associated with each R-tree of the SQR-tree. Therefore, because spatial objects are distributedly inserted into several R-trees and only R-trees intersected with the query area are accessed to process spatial queries like the SQR-tree, the query processing cost of the SQMR-tree can be reduced. Moreover, the search performance of the SQMR-tree is improved by using the mapping trees to access directly to leaf nodes of the R-tree without tree traversal like the MR-tree. Finally, we proved superiority of the SQMR-tree through experiments.

Keywords : Large Spatial Data, Hybrid Index Structure, R-tree, SQMR-tree

1. 서 론

오늘날 IT 기술을 기반으로 급속하게 발전하고 있는 LBS, Telematics, ITS 등과 같은 GIS 응용 분야는 RFID, GPS, Digital 카메라, CCTV 등과 같은 Geosensor 기술과 접목되면서 수집되고 저장되는 공간 데이타 양이 급속히 증가하고 있으며 이러한 대용량 공간 데이타를 효율적으로 처리할 수 있는

† 본 연구는 국토해양부 첨단도시기술개발사업 - 지능형국토정보기술혁신 사업과제의 연구비지원(10국토정보J71)에 의해 수행되었습니다.

* 건국대학교 컴퓨터공학과 박사과정 isshin@db.konkuk.ac.kr
** 건국대학교 컴퓨터공학과 강의교수 jjkim9@db.konkuk.ac.kr(교신저자)
*** 한국인터넷진흥원(KISA) 인터넷침해대응센터 침해예방단 선임연구원 redball@kisa.or.kr
**** 건국대학교 컴퓨터공학과 교수 kjhan@db.konkuk.ac.kr

공간 인덱스의 중요성이 높아지고 있다[6, 7, 8, 15, 16, 17]. 대표적인 트리 기반 공간 인덱싱 기법은 크게 R- tree와 같은 데이타 분할(Data Partitioning) 기반 인덱스 구조[2, 5]와 KD-tree와 같은 공간 분할(Space Partitioning) 기반 인덱스 구조[1, 3, 12]로 구분된다. 최근에는 이들의 특성을 결합한 하이브리드 인덱스 구조에 대한 연구가 활발히 진행되고 있다[8, 9, 10, 13, 17].

그러나, 이들 하이브리드 인덱스 구조는 노드 사이의 겹침이 증가하면 질의 처리시 노드 접근 비용이 증가하는 문제가 발생한다[4, 11]. 따라서, 기존 하이브리드 인덱스 구조의 질의 처리 성능을 향상시키기 위해서는 노드 사이의 겹침을 최소화하고 노드 접근 비용을 최소화하는 것이 필요하다. 이를 위해 R-tree를 기반으로 KD-tree[3]나 Quad-tree[1]와 같은 공간 분할 특성을 이용하여 성능을 향상시키는 연구가 활발히 진행되고 있고 있으며, 대표적인 인덱스 구조에는 MR-tree[14], SQR-tree[15]이 있다.

MR-tree는 R-tree 리프 노드를 직접 접근하게 해주는 매핑 트리와 R-tree를 결합한 하이브리드 인덱스 구조인데, MR-tree는 기존 인덱스 구조에 비해 검색 성능이 우수하지만 삽입 및 삭제 성능은 그렇지 못한 단점이 있다. SQR-tree는 SQ-tree[15]와 R-tree를 결합한 하이브리드 인덱스 구조인데, SQR-tree는 기존 인덱스 구조에 비해 검색, 삽입, 삭제 성능이 모두 우수하지만 MR-tree에 비해 검색 성능이 낮은 결과를 나타내고 있다.

본 논문에서는 MR-tree와 SQR-tree의 장점을 결합하여 대용량 공간 데이타를 보다 효율적으로 처리할 수 있는 하이브리드 인덱스 구조로서 SQMR-tree(Spatial Quad MR-tree)를 제시한다. SQMR-tree는 SQR-tree를 기본 구조로 하여 R-tree마다 매핑 트리가 적용된 형태를 가진다. 따라서, SQMR-tree는 SQR-tree와 같이 공간 객체가 여러 R-tree에 분산 저장되며 공간 질의 처리시 질의 영역에 해당하는 R-tree만 접근하면 되기 때문에 질의 처리 비용을 줄일 수 있다. 또한, SQMR-tree는 MR-tree와 같이 매핑 트리를 이용하여 트리 검색 없이 R-tree 리프 노드의 빠른 접근이 가능하기 때문에 검색 성능이 크게 향상된다. 특히, SQMR-tree는 R-tree의 큰 변경 없이 구현 가능하고 다양한 R-tree 변형에도 쉽게 적용할 수 있는 장점을 가지고 있다. 마지막으로 실험을 통해 SQMR-tree가 기존 연구와 비교하여 성능이 우수함을 입증하였다.

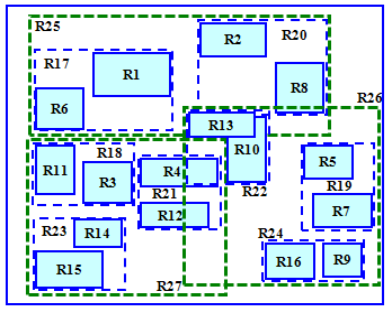
본 논문의 구성은 다음과 같다. 2장의 관련 연구에서는 MR-tree와 SQR-tree에 대해 분석한다. 3장에서는 본 논문에서 제시하는 SQMR-tree에 대해 기술하고, 4장에서는 SQMR-tree의 알고리즘에 대해 설명한다. 5장에서는 대용량 공간 데이타를 이용한 실험을 통해 SQMR-tree의 성능을 평가한다. 마지막으로 6장에서는 결론에 대해 언급한다.

2. 관련 연구

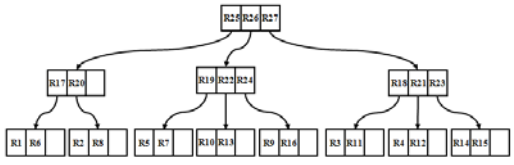
본 장에서는 R-tree를 기반으로 제시된 MR-tree와 SQR-tree에 대해 분석한다.

2.1 MR-tree

R-tree는 B-Tree를 공간 인덱싱에 적합하도록 변형한 인덱스로서 공간 객체를 표현하기 위해 MBR(Minimum Bounding Rectangle)을 사용하는 높이 균형 트리이다[5]. 그림 1은 R-tree 예를 보여준다.



(a) 공간 객체 MBR과 노드 MBR



(b) R-tree

그림 1. R-tree 예

그림 1(b)는 그림 1(a)에 대한 R-tree를 보여준다. R-tree는 중간 노드와 리프 노드로 구성되어 있으며 공간 객체에 대한 참조는 리프 노드에만 존재한다. 또한 R-tree는 노드 MBR이 서로 겹치기 때문에 검

색시 루트에서 리프 노드까지 도달하는 검색 경로가 하나 이상 존재할 수 있으며, 공간 객체 검색시 검색 경로 상의 모든 노드를 접근해야 한다[4].

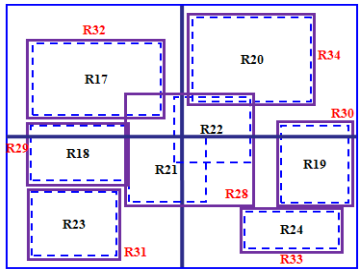
MR-tree는 R-tree에서 트리 검색 없이 R-tree 리프 노드를 직접 접근할 수 있는 매핑 트리를 이용함으로써 검색 성능을 향상시킨 인덱스 구조이다[14]. MR-tree에서는 매핑 트리를 구성하기 위해 데이타 공간을 서로 다른 크기의 여러 파티션으로 분할하며, 분할된 파티션들은 분할 과정에서 계층적인 형태를 갖는다.

계층 구조에서 파티션은 분할되지 않은 최하위 파티션(즉, 리프 파티션)과 분할된 파티션(즉, 중간 파티션)으로 구분될 수 있다. 중간 파티션은 분할 경계와 겹치는 R-tree 리프 노드와 연계되는데, 분할 경계와 겹치는 R-tree 리프 노드가 없는 경우를 포함하여 다수의 R-tree 리프 노드와 연계될 수 있다. 반면에 리프 파티션은 파티션 내에 포함되는 R-tree 리프 노드와 연계되는데, 파티션 내에 완전히 포함되는 R-tree 리프 노드가 없는 경우를 포함하여 최대 하나의 R-tree 리프 노드와 연계될 수 있다. 만일 리프 파티션에 포함되는 새로운 R-tree 리프 노드가 추가되면 리프 파티션은 분할된다.

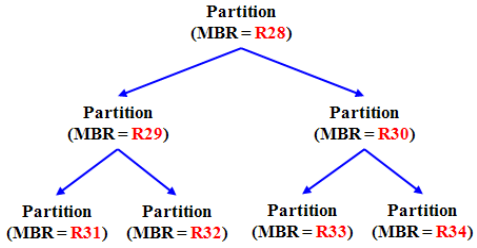
또한 각 파티션은 자신과 연계되는 모든 R-tree 리프 노드를 포함하는 MBR(즉, 파티션 MBR)을 가진다. 그림 2는 그림 1의 R-tree에 대한 R-tree 리프 노드 MBR, 파티션 MBR, 그리고 파티션 계층 구조를 보여준다.

그림 2(a)에서 점선 사각형 R17~R24는 R-tree 리프 노드 MBR을 보여주고, 실선 사각형 R28~R34는 파티션 MBR을 보여준다. 그림 2(b)는 그림 2(a)에 대한 파티션 계층 구조를 나타낸다. 그림 2(b)에는 중간 파티션은 R28, R29, R30 3개가 존재한다. 중간 파티션은 2개 이상의 R-tree 리프 노드 MBR과 연계될 수 있다. 그리고 그림 2(b)에는 리프 파티션은 R31, R32, R33, R34 4개가 존재한다. 리프 파티션은 최대 하나의 R-tree 리프 노드와 연계될 수 있다.

매핑 트리는 파티션마다 해당하는 R-tree의 리프 노드를 직접 접근하는데 사용되며 파티션, R-tree 리프 노드의 MBR과 포인터를 이용하여 생성된다. 매핑 트리는 매핑 트리 노드와 연결 리스트 노드로 구성된다. 매핑 트리 노드는 파티션마다 생성되며 파티션과 R-tree 리프 노드를 연결해 주고, 연결 리스트 노드는 파티션과 연계된 2개 이상의 R-tree 리프 노드를 연결해 준다. 그림 3은 그림 1의 R-tree에 매핑 트리를 적용한 MR-tree의 전체 구조를 보여준다.



(a) R-tree 리프 노드 MBR과 파티션 MBR



(b) 파티션 계층 구조

그림 2. R-tree 리프 노드 MBR, 파티션 MBR, 파티션 계층 구조

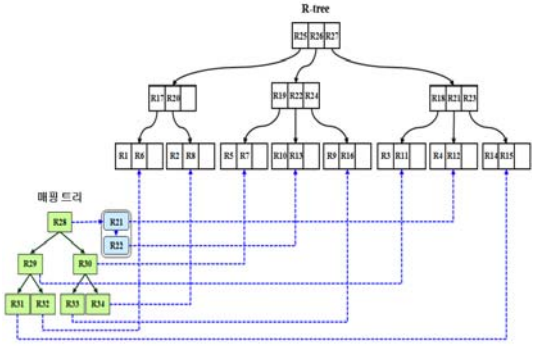


그림 3. MR-tree의 전체 구조

그림 3과 같이 MR-tree는 R-tree와 매핑 트리로 구성되어 있다. 그림 3의 매핑 트리에서 파티션 MBR R28인 매핑 트리 노드는 2개 R-tree 리프 노드의 MBR과 포인터를 가지는 연결 리스트(즉, R21과 R22로 구성됨)를 가리키는 포인터를 가지고 있다. 다른 매핑 트리 노드는 모두 하나의 R-tree 리프 노드를 직접 가리키는 포인터를 가지고 있다.

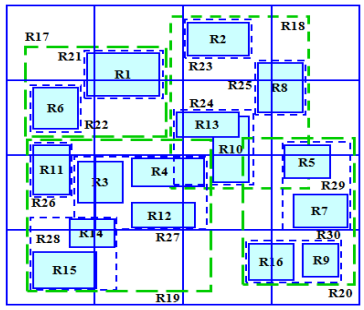
MR-tree의 매핑 트리는 파티션마다 연계된 R-tree 리프 노드를 가리키고 있어서 검색시 매핑 트리를 이용하여 트리 검색 없이 R-tree 리프 노드를 직접 접근할 수 있다. 그리고, MR-tree는 공간 객체 삽입 및 삭제시 R-tree 리프 노드 MBR이 변경되는 경우에만 해당 매핑 트리 노드 또는 연결 리스트 노드를 갱신하면 된다. 그러나, MR-tree는 갱신시 매핑 트리에 갱신 내용을 반영해야 하기 때문에 갱신 성능이 R-tree 보다 저하되는 단점을 가지고 있다.

2.2 SQR-tree

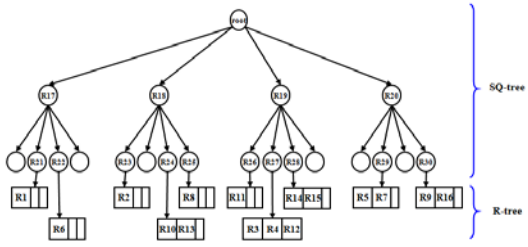
SQR-tree는 면적을 갖는 공간 객체 처리에 적합하도록 Quad-tree를 확장한 SQ-tree(Spatial Quad-tree)와 SQ-tree의 리프 노드마다 연계되어 공간 객체를 저장하는 R-tree가 결합된 인덱스 구조이다[15]. SQR-tree는 2단계 필터링을 적용한다. 먼저 분할된 데이타 공간마다 존재하는 R-tree를 식별하는 SQ-tree가 1단계 필터링을 수행하고, SQ-tree의 리프 노드마다 연계되면서 실제로 공간 데이타를 저장하는 R-tree가 2단계 필터링을 수행한다. 그리고 SQR-tree 구현시 SQ-tree는 분할된 데이타 공간을 구분해주는 단순한 구조를 갖기 때문에 메인 메모리에 상주하고, 실제 공간 객체가 저장되는 R-tree는 디스크에 저장하게 된다. 그림 4는 그림 1에 대한 SQR-tree의 전체 구조를 보여준다.

그림 4(a)에서 실선 사각형 R1~R16은 공간 객체 MBR을 보여주고 점선 사각형 R17~R30은 SQ-tree의 노드 MBR이다. 그림 4(b)는 그림 4(a)에 대한 SQR-tree를 보여준다. 그림 4(b)에서 보듯이 SQR-tree의 상위 구조는 SQ-tree로 구성되고 하위 구조는 SQ-tree의 리프 노드마다 연계된 R-tree들로 구성된다. 이처럼 SQR-tree에서는 데이타 공간을 여러 하위 공간으로 분할하고 SQ-tree를 통해 분할된 하위 공간을 검색할 수 있다. 따라서, 공간 질의 처리시 SQR-tree는 SQ-tree를 통해 여러 R-tree 중에서 필요한 R-tree만을 접근할 수 있다.

SQR-tree는 다양한 분포를 갖는 공간 데이타에 대해서도 효율적인 인덱싱이 가능하고, SQ-tree를 통해 검색할 데이타 공간을 찾는 선행 필터링을 수행함으로써 R-tree의 접근을 줄이고, 공간 객체를 여러 R-tree에 분산 저장함으로써 트리 높이를 낮출 수 있다. 따라서 SQR-tree는 기존 인덱스보다 검색, 삽입, 삭제 성능이 우수하나, MR-tree 보다 검색 성능이 저하되는 단점을 보이고 있다.



(a) 공간 객체 MBR 및 SQ-tree의 노드 MBR



(b) SQR-tree

그림 4. SQR-tree의 예

3. SQMR-tree

본 장에서는 본 논문에서 제시한 SQMR-tree에 대하여 설명한다.

3.1 SQMR-tree 전체 구조

본 논문에서는 SQR-tree와 MR-tree의 장점을 결합한 새로운 하이브리드 인덱스 구조인 SQMR-tree (Spatial Quad MR-tree)를 제안한다. SQMR-tree는 SQR-tree를 기본 구조로 가지며 SQR-tree에 포함된 R-tree마다 매핑 트리가 적용된 인덱스 구조이다. 즉, SQMR-tree에서는 상위 구조가 SQ-tree이고, 하위 구조가 MR-tree로 되어 있다. SQMR-tree는 SQ-tree의 적용으로 노드 분할 경계의 독립적인 확장이 가능하고, MR-tree의 적용으로 다양한 분포의 공간 데이타에 대한 효율적인 처리가 가능하다.

SQMR-tree에서는 SQ-tree는 단순히 데이타 공간을 분할하고 MR-tree를 찾아가는 선행 필터링을 제공하고, MR-tree는 SQ-tree의 리프 노드마다 연계되어 공간 객체를 저장한다. 그리고 SQMR-tree

구현시 메모리 구조에 적합한 SQ-tree와 매핑 트리는 메인 메모리에 구성되고, 실제 데이타를 저장하는 R-tree는 디스크에 구성된다. 그림 5는 그림 4에 대한 SQMR-tree의 전체 구조를 보여준다.

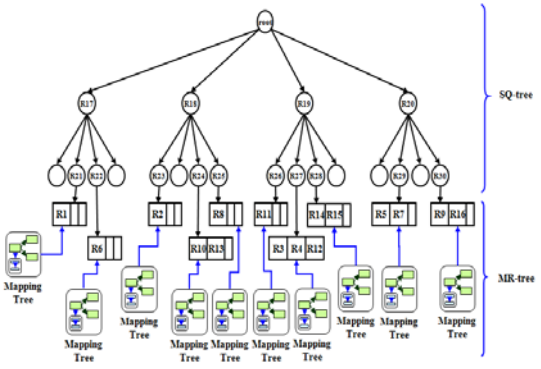


그림 5. SQMR-tree의 전체 구조

그림 5와 같이 SQMR-tree에서 상위 구조는 SQ-tree이고, SQ-tree의 리프 노드는 MR-tree와 연계되어 있다. SQMR-tree는 SQ-tree를 통해 데이타 공간을 하위 공간들로 분할하고, 각 하위 공간에 포함되는 공간 객체는 해당 MR-tree에 저장된다. SQMR-tree에서는 단순히 데이타 공간을 분할하고 MR-tree를 찾아가는 경로를 제공하는 SQ-tree가 메인 메모리에 구성되어 있기 때문에 질의 영역에 해당되는 MR-tree를 빠르게 접근할 수 있다. 또한, MR-tree에서 R-tree 리프 노드를 직접 접근하게 해주는 매핑 트리도 메인 메모리에 구성되어 있기 때문에 R-tree 리프 노드의 접근 속도를 향상시킬 수 있다. 따라서 SQMR-tree는 SQR-tree이나 MR-tree보다 검색 성능을 향상시킬 수 있다.

SQMR-tree는 SQR-tree와 유사하게 2단계 필터링을 수행한다. 먼저 단순히 데이타 공간을 분할하는 SQ-tree가 1단계 필터링을 수행하고, SQ-tree의 리프 노드마다 연계되면서 실제로 공간 데이타를 저장하는 MR-tree가 2단계 필터링을 수행한다.

3.2 MR-tree의 파티션 생성 영역

SQMR-tree는 SQ-tree의 리프 노드마다 MR-tree가 연계되어 있는 구조를 가지고 있다. 그러므로 각각의 MR-tree는 R-tree 리프 노드를 직접 접근하게 해주는 매핑 트리를 구성하기 위해 파티션을 생성해야 한다. 즉, MR-tree는 자신이 해당되는 SQ-tree 리프 노드의 영역에 파티션을 생성해야 한다. 그림 6은 SQMR-tree에서 MR-tree의 파티션 생성 영역을 보여준다.

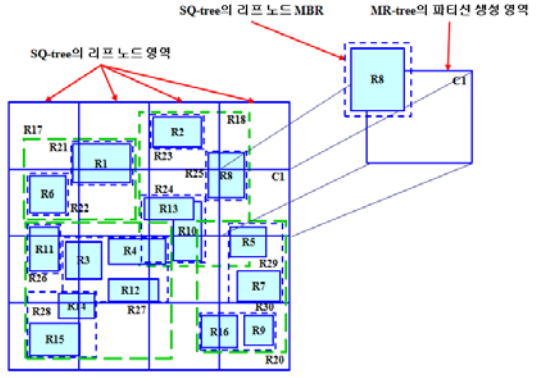


그림 6. 파티션 생성 영역

그림 6은 서로 겹치지 않는 16개의 SQ-tree 리프 노드 영역을 보여준다. 예를 들어, SQ-tree 리프 노드 영역 C1은 다른 SQ-tree 리프 노드 영역과 서로 겹치지 않는다. SQMR-tree에서 서로 겹치지 않는 각각 SQ-tree 리프 노드 영역은 하나의 MR-tree를 생성하기 위한 파티션 생성 영역이 될 수 있다.

비록 그림 6과 같이 SQMR-tree의 특정 MR-tree에서 공간 객체 MBR(예를 들어 R8)이 파티션 생성 영역을 벗어나는 경우가 발생하지만 공간 객체 MBR은 자신의 중심점 위치에 따라 포함되는 SQ-tree 노드가 결정된다. 그런데, 공간 객체 MBR의 절반 이상은 반드시 파티션 생성 영역에 존재하고 마찬가지로 공간 객체 MBR을 포함하고 있는 SQ-tree의 노드 MBR의 절반 이상도 반드시 파티션 생성 영역에 존재하게 된다. 따라서 각각의 SQ-tree 리프 노드 영역은 대응되는 MR-tree에 대한 파티션 생성 영역으로 사용될 수 있다.

4. 알고리즘

본 장에서는 SQMR-tree의 삽입, 삭제, 그리고 검색 알고리즘에 대해 설명한다. 공간 객체가 삽입, 삭제, 검색되면 먼저 상위 구조인 SQ-tree에 대해서 수행된다. 그러나 SQMR-tree에서 실제로 공간 객체는 MR-tree에 저장되기 때문에 SQ-tree는 단순히 공간 객체를 삽입, 삭제, 검색할 MR-tree를 찾는 경로를 제공한다.

4.1 삽입 알고리즘

그림 7은 SQMR-tree에서 공간 객체를 삽입하는 알고리즘을 보여준다.

```
Algorithm : Insert(M, OID, N)
1: C ← GetCentroid(M);
2: I ← FindQuadrant(C, N);
3: If (N is a internal node) Then
4:    CN ← child node pointed to by I;
5:    CombineMbr(M, N);
6:    Insert(M, OID, CN);
   Else
7:    MRN ← MR-tree node pointed to by I;
8:    CombineMbr(M, N);
9:    InsertMRtree(M, OID, MRN);
   End If
```

그림 7. 삽입 알고리즘

그림 7의 삽입 알고리즘에서 입력값은 삽입되는 공간 객체의 MBR과 식별자, 그리고 SQ-tree 노드이다. 먼저 삽입되는 공간 객체의 중심점을 구하고, SQ-tree 노드에서 이 중심점을 포함하는 자식 노드를 선택한다. 만일 SQ-tree 노드가 내부 노드이면 삽입되는 공간 객체 MBR을 포함하도록 노드 MBR을 확장하고 리프 노드에 도달할 때까지 삽입 알고리즘을 재귀적으로 수행한다. 반면에 SQ-tree 노드가 리프 노드이면 삽입되는 공간 객체 MBR을 포함하도록 SQ-tree 리프 노드의 노드 MBR을 확장하고, 연계되어 있는 MR-tree에서 공간 객체를 삽입한다. MR-tree에서 공간 객체 삽입은 R-tree에서 먼저 수행된다. 공간 객체가 R-tree 리프 노드에 도달하면 R-tree 리프 노드 MBR이 확장되거나 노드 분할로 R-tree 리프 노드 MBR이 삭제되는 경우에만 해당 R-tree와 연계된 매핑 트리가 갱신된다.

4.2 삭제 알고리즘

그림 8은 SQMR-tree의 삭제 알고리즘을 보여준다.

그림 8의 삭제 알고리즘에서 입력값은 삭제할 공간 객체의 MBR과 식별자, 그리고 SQ-tree 노드이다. 먼저 삭제할 공간 객체의 중심점을 구하고, SQ-tree 노드에서 이 중심점을 포함하는 자식 노드를 선택한다. 만일 SQ-tree 노드가 내부 노드이면 리프 노드에 도달할 때까지 삭제 알고리즘을 재귀적으로 수행한다. 반면에 SQ-tree 노드가 리프 노드이면 연계되어 있는 MR-tree에서 공간 객체가 삭제된다. 이때, MR-tree에서 공간 객체가 삭제된 R-tree 리프 노드의 MBR 크기가 변경되면 해당 R-tree 리프 노드와 연계된 매핑 트리가 갱신되고, SQ-tree에서도 리프 노드의 MBR 크기가 변경되면 루트 노드까지 노드 MBR이 조정된다.

```
Algorithm : Delete(M, OID, N)
1: C ← GetCentroid(M);
2: I ← FindQuadrant(C, N);
3: If (N is a internal node) Then
4:    CN ← child node pointed to by I;
5:    If (Delete(M, OID, CN)) Then
6:       If (MBR size of N is decreased) Then
7:          AdjectMbr(N);
          End If
       End If
    Else
8:    RN ← MR-tree node pointed to by I;
9:    DeleteMRtree(M, OID, RN);
10:   If (MBR size of N is decreased) Then
11:      AdjectMbr(N);
      End If
   End If
```

그림 8. 삭제 알고리즘

4.3 검색 알고리즘

그림 9는 SQMR-tree의 검색 알고리즘을 보여준다.

그림 9의 검색 알고리즘에서 입력값은 질의 윈도우와 SQ-tree 노드이다. 만일 SQ-tree 노드가 내부 노드이면 자식 노드 MBR과 질의 윈도우가 겹치는지 검사하여 해당 자식 노드를 접근한다. 이러한 과정은 리프 노드에 도달할 때까지 재귀적으로 수행된다. SQ-tree 리프 노드에 도달하면 연계된 MR-tree에서 질의 윈도우로 검색을 수행한다.

5. 성능 평가

본 장에서는 다양한 분포 형태를 갖는 대용량 공간 데이타를 가지고 실험을 수행하여 SQMR-tree의 성능을 평가한다.

```
Algorithm : Search(QW, N)
 1: RESULT ← Ø;
 2: If (N is a internal node) Then
 3:   For each entry (mbr, point) of N Do
 4:     If (mbr overlaps QW) Then
 5:       CN ← child node pointed to by point;
 6:       Search(QW, CN);
        End If
      End For
    Else
 7:   For each entry (mbr, point) of N Do
 8:     If (mbr overlaps QW) Then
 9:       RN ← MR-tree node pointed to by point;
10:       RESULT ∪ SearchMRtree(QW, RN);
        End If
      End For
    End If
11: Return RESULT;
```

그림 9. 검색 알고리즘

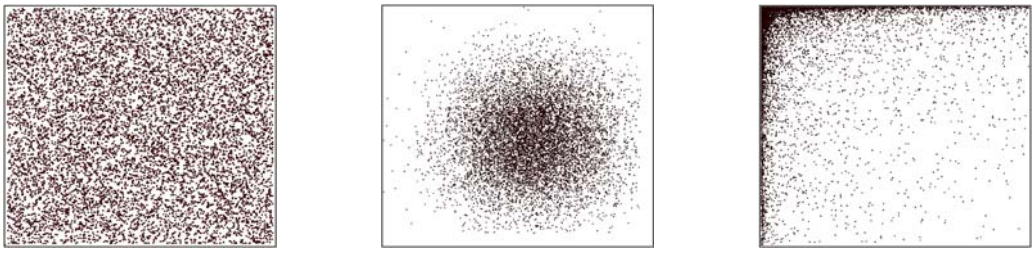
5.1 실험 환경

실험에 사용된 시스템의 하드웨어 사양은 Intel Core 2.33GHz CPU, 1GB RAM이고, 운영체제는 Windows XP를 사용하였다. 그리고 성능 평가를 위해 Visual C++ 6.0을 이용하여 MR-tree, SQR-tree, SQMR-tree를 구현하였다. 본 실험에서는 Uniform, Gauss, Skew 분포를 갖는 공간 데이타를 사용하였고 공간 객체를 모두 포함하는 사각형 영역을 전체 데이타 공간으로 가정하였다. Uniform, Gauss, Skew 분포를 갖는 공간 데이타는 한 변의 길이가 전체 데이타 공간의 한 변의 길이의 0.01%가 되는 정사각형으로 20만개~100만개를 생성하여 사용하였다. 그림 10은 실험에서 사용된 공간 데이타를 보여준다.

5.2 실험 결과

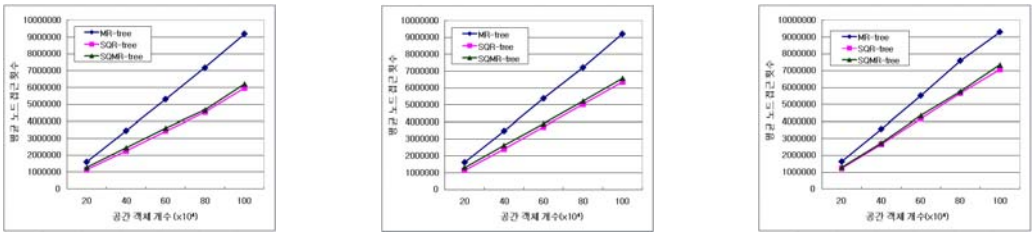
인덱스 생성 실험에서는 Uniform, Gauss, Skew 분포를 갖는 공간 데이타를 20만개에서 100만개까지 20만개씩 증가시키면서 인덱스를 생성하는 동안 MR-tree, SQR-tree, SQMR-tree의 평균 노드 접근 횟수를 비교하였다. 그림 11은 인덱스 생성을 수행한 실험 결과를 보여준다.

그림 11과 같이 인덱스 생성 실험 결과에서는 SQMR-tree의 성능이 SQR-tree 보다는 다소 저하되지만 MR-tree 보다는 크게 향상되는 것으로 나타났다. 이는 SQMR-tree가 공간 객체를 여러 MR-tree에 분산 저장하여 삽입 비용을 줄임으로써 MR-tree 보다는 인덱스 생성 성능이 향상되었지만,



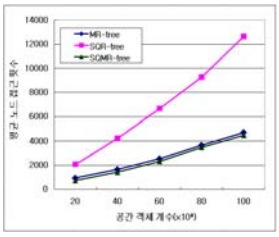
(a) Uniform (b) Gauss (c) Skew

그림 10. 실험에 사용된 공간 데이타

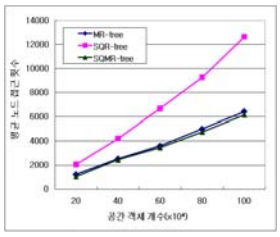


(a) Uniform (b) Gauss (c) Skew

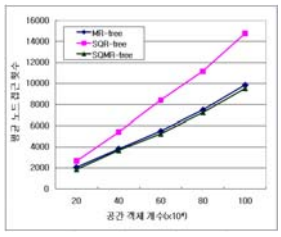
그림 11. 인덱스 생성 실험 결과



(a) Uniform



(b) Gauss



(c) Skew

그림 12. 범위 검색 질의 실험 결과

MR-tree마다 매핑 트리를 유지하는 비용이 발생함으로써 SQR-tree 보다는 삽입 성능이 저하되었기 때문이다.

검색 실험에서는 범위 검색 질의를 수행하였으며, 범위 검색 질의에서는 전체 데이타 공간의 1%~5%가 되는 질의 사각형을 사용하였다. 성능 비교를 위해 1,000번의 검색 질의를 수행하는 동안 MR-tree, SQR-tree, SQMR-tree의 평균 노드 접근 횟수를 비교하였다. 그림 12는 범위 검색 질의를 수행한 실험 결과를 보여준다.

그림 12와 같이 범위 검색 질의 실험 결과에서는 모든 경우에서 SQMR-tree의 성능이 가장 우수한 것으로 나타났다. 이는 SQMR-tree가 SQ-tree를 통해 여러 MR-tree 중에서 질의 객체에 해당하는 MR-tree만을 접근하고, 접근한 MR-tree에서는 매핑 트리를 통해 R-tree 리프 노드를 직접 접근할 수 있기 때문이다.

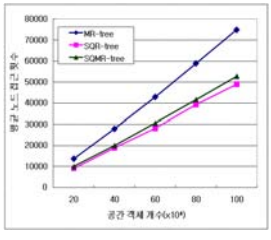
삭제 질의 실험에서는 전체 데이타 공간의 1%가 되는 질의 윈도우와 서로 겹치는 공간 객체를 모두 삭제하는 동안 MR-tree, SQR-tree, SQMR-tree의 평균 노드 접근 횟수를 비교하였다. 그림 13은 실험 데이타에 대한 삭제 질의를 수행한 실험 결과를 보여준다.

그림 13에서 보듯이 삭제 질의 실험 결과는 인덱스 생성 실험 결과와 유사하게 SQMR-tree의 성능이 SQR-tree 보다는 다소 저하되지만 MR-tree 보다는 크게 향상되는 것으로 나타났다. 이는 SQMR-tree가 SQ-tree를 통해 여러 MR-tree 중에서 삭제 질의 윈도우와 겹치는 MR-tree만을 접근하여 삭제 비용을 줄임으로써 MR-tree 보다는 삭제 성능이 향상되었지만, MR-tree마다 매핑 트리를 유지하는 비용이 발생함으로써 SQR-tree 보다는 삭제 성능이 저하되었기 때문이다.

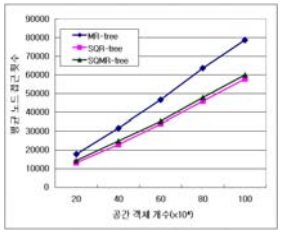
결론적으로 비록 인덱스 생성과 삭제 질의 실험 결과에서는 SQMR-tree가 SQR-tree보다 성능이 다소 저하되는 것으로 나타났으나 범위 검색 질의 실험 결과는 SQMR-tree가 MR-tree와 SQR-tree 보다 우수한 성능을 보였다.

6. 결 론

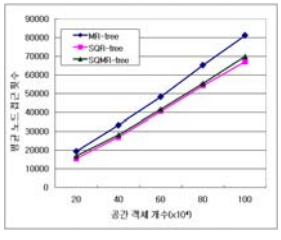
본 논문에서는 R-tree 기반 인덱스인 MR-tree와 SQR-tree의 장점을 결합하여 대용량 공간 데이타의 효율적인 질의 처리를 제공하는 SQMR-tree를 제안하였다. SQMR-tree에서는 공간 객체가 여러 MR-tree에 분산 저장되고 질의 처리시 해당되는 MR-tree만을 접근하면 되기 때문에 질의 처리 성능이 향상된다. 그리고, MR-tree에서 R-tree 리프 노드를



(a) Uniform



(b) Gauss



(c) Skew

그림 13. 삭제 질의 실험 결과

직접 접근하게 해주는 매핑 트리를 이용함으로써 검색 성능이 크게 향상된다. 특히, R-tree를 기반으로 생성된 SQMR-tree는 R-tree 구조의 큰 변형없이 구현이 가능하고 기존의 다양한 R-tree 변형에도 쉽게 적용할 수 있는 장점을 가지고 있다.

그러나 갱신시 갱신 부분을 매핑 트리에 반영하는 비용이 발생하기 때문에 SQR-tree에 비해 갱신 성능이 저하되는 단점을 가지고 있다. 또한 Gauss, Skew 분포와 같이 공간 객체 분포가 균등하지 않은 환경에서는 SQ-tree의 높이가 높아지면서 MR-tree가 증가함에 따라 성능이 저하되는 문제가 있다. 향후에는 비균등한 분포에서도 질의 처리 성능을 향상시키도록 SQMR-tree의 구조를 개선하는 연구가 필요하다.

참 고 문 헌

[1] D. A. Beckley, M. W. Evens and V. K. Raman, 1985, “Multikey Retrieval from K-d Trees and Quad-trees”, Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp. 291-301.

[2] N. Beckmann, H. P. Kriegel, R. Schneider and B. Seeger, 1990, “The R*-tree: An Efficient and Robust Access Method for Points and Rectangles”, Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 322-331.

[3] J. L. Bentley, 1975, “Multidimensional Binary Search Trees Used for Associative Searching”, Communications of the ACM, vol. 18, no. 9, pp. 509-517.

[4] H. Bo and W. Qiang, 2007, “A Spatial Indexing Approach for High Performance Location Based Services”, The Journal of Navigation, vol. 60, no. 1, pp. 83-93.

[5] A. Guttman, 1984, “R-trees: A Dynamic Index Structure for Spatial Searching”, Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp. 47-57.

[6] X. Huang, S. H. Baek, D. W. Lee, W. I. Chung and H. Y. Bae, 2009, “UIL : A Novel Indexing Method for Spatial Objects and Moving Objects”, Journal of Korea Spatial Information System Society, vol. 11, no. 2, pp. 19-26.

[7] Y. J. Jung, K. H. Ryu, M. S. Shin and S. Nittel, 2010, “Historical Index Structure for Reducing Insertion and Search Cost in LBS”, Journal of Systems and Software, vol 83, no. 8. pp. 1500-1511.

[8] W. Li and B. Timo, 2004, “Comparative Analysis of the Efficiency of R-tree Based Indexing Strategies for Information Retrieval”, Proc. of the Int. Conf. on Information and Knowledge Engineering, pp. 180-184.

[9] Y. Manolopoulos, E. Nardelli, A. Papadopoulos and G. Proietti, 1996, “QR-tree: A Hybrid Spatial Data Structure”, Proc. of the 1st Int. Conf. on Geographic Information Systems in Urban, Regional and Environmental Planning, pp. 3-7.

[10] K. Michal, S. Vaclav, P. Jaroslav and Z. Pavel, 2006, “Efficient Processing of Narrow Range Queries in Multi-dimensional Data Structures”, Proc. of the 10th Int. Database Engineering and Applications Symposium, pp.69-79.

[11] B. S. Nam and A. Sussman, 2004, “A Comparative Study of Spatial Indexing Techniques for Multidimensional Scientific Datasets”, Proc of Int. Conf. on Scientific and Statistical Database Management, pp. 171-180.

[12] T. K. Sellis, N. Roussopoulos and C. Faloutsos, 1987, “The R+-tree: A Dynamic Index for Multi-dimensional Objects”, Proc. of the 13th Int. Conf. on Very Large Data Bases, pp. 507-518.

[13] X. Wu and C. Zang, 2009, “A New Spatial Index Structure for GIS Data”, Proc. of the 3rd Int. Conf. on Multimedia and Ubiquitous Engineering, pp. 471-476.

[14] 강홍구, 김정준, 신인수, 한기준, 2010, “MR-tree: 효율적인 공간 검색을 위한 매핑 기반 R-tree”, 한국공간정보학회지, 제18권, 제4호, pp. 109-120.

[15] 강홍구, 김정준, 한기준, 2011, “SQR-tree : 효율적인 공간 질의 처리를 위한 하이브리드 인덱스 구조”, 한국공간정보학회지, 19권, 제2호, pp. 47-56.

[16] 김학철, 2010, “다차원 공간의 효율적인 그리드 분할을 통한 디클러스터링 알고리즘 성능향상 기

법", 한국공간정보시스템학회 논문지, 제12권, 제1호, pp. 37-48.

[17] 이득우, 강홍구, 이기영, 한기준, 2009, "DGR-tree : u-LBS에서 POI의 검색을 위한 효율적인 인덱스 구조", 한국공간정보시스템학회 논문지, 제11권, 제3호, pp. 55-62.

논문접수 : 2011.05.09
수 정 일 : 1차 2011.07.22 / 2차 2011.08.23
심사완료 : 2011.08.25

신 인 수

2006년 건국대학교 컴퓨터공학 공학사
2008년 건국대학교 대학원 공학석사
2008년~현재 건국대학교 컴퓨터공학 박사과정
관심분야는 시공간 데이타베이스, 모바일 데이타베이스, Geo Semantic Web

김 정 준

2003년 건국대학교 컴퓨터공학 공학사
2005년 건국대학교 대학원 공학석사
2010년 건국대학교 대학원 공학박사
2010년~현재 건국대학교 컴퓨터공학부 강의교수
관심분야는 공간 데이타베이스, 시공간 데이타베이스, GIS, LBS, 텔레매틱스, USN, Semantic Web

강 홍 구

2002년 건국대학교 컴퓨터공학 공학사
2004년 건국대학교 대학원 공학석사
2009년 건국대학교 대학원 공학박사
2009년~2010년 건국대학교 컴퓨터공학부 강의교수
2010년~현재 한국인터넷진흥원(KISA) 인터넷침해대응센터 침해예방단 연구개발팀 선임연구원
관심분야는 공간 데이타베이스, GIS, LBS, USN

한 기 준

1979년 서울대학교 수학교육학 이학사
1981년 한국과학기술원(KAIST) 공학석사
1985년 한국과학기술원(KAIST) 공학박사
1985년~현재 건국대학교 컴퓨터공학부 교수
1990년 Stanford 대학 전산학과 Visiting Scholar
2000년~2002년 한국정보과학회 데이타베이스 연구회 운영위원장
2004년~2006년 한국공간정보시스템학회 회장
2004년~2008년 한국정보시스템감리사협회 회장
관심분야는 공간 데이타베이스, GIS, LBS, 텔레매틱스, 정보시스템 감리