SQR-Tree : 효율적인 공간 질의 처리를 위한 하이브리드 인덱스 구조

SQR-Tree : A Hybrid Index Structure for Efficient Spatial Query Processing

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

요 약 대표적인 트리 기반 공간 인덱스 구조는 크게 R-Tree와 같은 데이타 분할 기반 인덱스 구조와 KD-Tree와 같은 공간 분할 기반 인덱스 구조로 구분되며, 최근에는 이들의 장점을 결합한 하이브리드 인덱스 구조에 대한 연구가 활발히 진행되고 있다. 그러나, 기존 연구에서는 공간 객체가 삽입되는 노드의 분할 경계 확장이 다른 이웃 노드에 연쇄적으로 전파되어 노드간 겹침이 증가하고 질의 처리 비용이 높아지는 문제가 있다. 본 논문에서는 이러한 문제를 해결하기 위하여 효율적인 질의 처리를 위한 하이브리드 인덱스 구조인 SQR-Tree를 제시한다. SQR-Tree는 크기를 갖는 공간 객체 처리에 적합하도록 Quad-Tree를 확장한 SQ-Tree(Spatial Quad-Tree)와 SQ-Tree의 리프 노드마다 연계되어 실제로 공간 객체를 저장하는 R-Tree가 결합된 인덱스 구조이다. SQR-Tree는 노드마다 하위 노드를 포함하는 MBR을 가지고 있기 때문에 노드의 분할 경계 확장이 독립적으로 이루어지도록 하여 노드간 겹침을 줄였다. 그리고 SQR-Tree에서 공간 객체는 분할된 데이타 공간마다 존재하는 여러 R-Tree에 분산 저장되며 SQ-Tree가 분할된 데이타 공간을 식별하는 기능을 수행한다. 따라서 공간 질의 처리시 질의 영역에 해당하는 R-Tree만 접근하면 되기 때문에 질의 처리 비용을 줄일 수 있다. 마지막으로 실험을 통해 SQR-Tree의 우수성을 입증하였다.

키워드 : 대용량 공간 데이타, 하이브리드 인덱스 구조, SQ-Tree, R-Tree

Abstract Typical tree-based spatial index structures are divided into a data-partitioning index structure such as R-Tree and a space-partitioning index structure such as KD-Tree. In recent years, researches on hybrid index structures combining advantages of these index structures have been performed extensively. However, because the split boundary extension of the node to which a new spatial object is inserted may extend split boundaries of other neighbor nodes in existing researches, overlaps between nodes are increased and the query processing cost is raised. In this paper, we propose a hybrid index structure, called SQR-Tree that can support efficient processing of spatial queries to solve these problems. SQR-Tree is a combination of SQ-Tree(Spatial Quad-Tree) which is an extended Quad-Tree to process non-size spatial objects and R-Tree which actually stores spatial objects associated with each leaf node of SQ-Tree. Because each SQR-Tree node has an MBR containing sub-nodes, the split boundary of a node will be extended independently and overlaps between nodes can be reduced. In addition, a spatial object is inserted into R-Tree in each split data space and SQ-Tree is used to identify each split data space. Since only R-Trees of SQR-Tree in the query area are accessed to process a spatial query, query processing cost can be reduced. Finally, we proved superiority of SQR-Tree through experiments.

Keywords : Large Spatial Data, Hybrid Index Structure, SQ-Tree, R-Tree

† 본 연구는 국토해양부 첨단도시기술개발사업 - 지능형국토정보기술혁신 사업과제의 연구비지원(07국토정보C05)에 의해 수행되었습니다.

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

1. 서 론

최근 LBS(Location Based Services), Telematics, ITS(Intelligent Transport Systems)와 같은 GIS 응용 분야에서는 다양한 지오센서(Geosensor)의 활용으로 수집되는 공간 데이타의 양이 급증하고 있으며 대용량 공간 데이타를 빠르게 처리할 수 있는 공간 인덱스의 필요성이 높아지고 있다[9,10,13,19]. 널리 사용되고 있는 트리 기반 공간 인덱스 구조는 크게 R-Tree[7]와 같은 데이타 분할(Data Partitioning) 기반 인덱스 구조와 KD-Tree[4]와 같은 공간 분할(Space Partitioning) 기반 인덱스 구조로 구분된다[12,16,18].

공간 분할 기반 인덱스 구조는 데이타 공간을 서로 겹치지 않는 하위 공간으로 분할하므로 구조가 단순하고 생성이 용이하며 크기를 갖지 않은 점 데이타 처리에 적합하지만 사각형 데이타와 같이 크기를 갖는 공간 데이타 처리에는 적합하지 않은 단점을 가지고 있다[1,12]. 반면에 데이타 분할 기반 인덱스 구조는 계층적인 포함(Containment) 관계를 이용하여 구성되고 크기를 갖지 않은 점 데이타뿐만 아니라 크기를 갖는 공간 데이타 처리에 모두 적합하지만 노드간의 겹침(Overlap)이 발생하기 때문에 검색시 다중 검색 경로를 갖는 단점을 가지고 있다[8,16,17].

최근 대용량 공간 데이타의 효율적인 처리를 위해 공간 분할 기반 인덱스 구조와 데이타 분할 기반 인덱스 구조의 장점을 살리고 단점을 상호 보완하기 위해 두 인덱스 구조의 특성을 결합한 하이브리드 인덱스 구조에 대한 연구가 활발히 진행되고 있다[6,11,12]. 이러한 하이브리드 인덱스 구조에는 KDB-Tree[14]의 노드 연쇄 분할 문제(Cascading Split Problem)를 해결하기 위해 데이타 분할 기반 인덱스 구조의 특성인 노드 사이의 겹침을 적용한 Hybrid-Tree[6], 크기를 갖는 공간 데이타에 대한 인덱싱을 지원할 수 있도록 Hybrid-Tree를 확장한 SH-Tree(Spatial Hybrid-Tree)[11] 등이 있다. 그러나, SH-Tree는 공간 객체가 삽입되는 노드의 분할 경계 확장이 다른 이웃 노드에 연쇄적으로 전파되어 노드간 겹침이 증가하고 질의 처리 비용이 높아지는 문제가 있다.

본 논문에서는 이러한 문제를 해결하기 위해 공간 객체 삽입시 노드의 분할 경계 확장이 독립적으로 이루어지도록 노드를 구성함으로써 노드간 겹침을 줄이고 대용량 공간 데이타에 대한 질의 처리 성능을 향상시키는 인덱스 구조인 SQR-Tree를 제시한다. SQR-Tree는 SQ-Tree(Spatial Quad-Tree)와 R-Tree를 결합한 하이브리드 인덱스 구조인데, SQ-Tree는 크기를 갖는 공간 데이타를 처리할 수 있도록 Quad-Tree[4]를 확장한 인덱스 구조이다. SQR-Tree는 노드마다 하위 노드를 포함하는 MBR을 가지고 있기 때문에 노드의 분할 경계 확장이 독립적으로 이루어진다.

SQR-Tree는 전체 데이타 공간을 Quad-Tree 형태로 분할하는 SQ-Tree와 SQ-Tree의 리프 노드마다 연계되어 실제로 공간 객체를 저장하는 R-Tree들로 구성된다. SQR-Tree에서는 SQ-Tree가 분할된 데이타 공간을 식별하는 기능을 수행한다. SQR-Tree에서는 SQ-Tree로 분할된 데이타 공간마다 R-Tree가 존재하므로 공간 객체가 여러 R-Tree에 분산 저장되기 때문에 트리 높이를 낮출 수 있다. 또한, 공간 질의 처리시 질의 영역에 해당하는 R-Tree만 접근하면 되기 때문에 질의 처리 비용을 줄일 수 있다. 마지막으로 실험을 통해 기존 연구보다 SQR-Tree가 우수함을 입증하였다.

본 논문의 구성은 다음과 같다. 2장의 관련 연구에서는 비교 대상인 R-Tree와 SH-Tree에 대해 분석한다. 3장에서는 본 논문에서 제시한 SQR-Tree를 구성하는 SQ-Tree와 SQR-Tree의 구조에 대해 기술한다. 4장에서는 SQR-Tree의 알고리즘을 설명한다. 그리고 5장에서는 성능 평가를 통해 SQR-Tree의 우수성을 입증한다. 마지막으로 6장에서는 결론에 대해 언급한다.

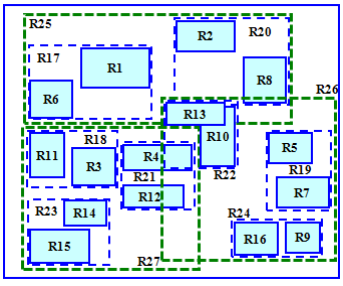
2. 관련 연구

본 장에서는 비교 대상인 R-Tree와 SH-Tree에 대해 설명한다.

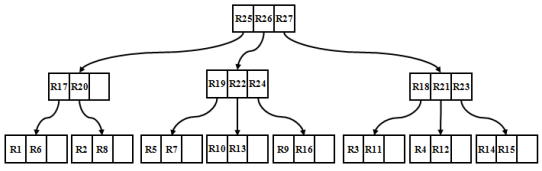
2.1 R-Tree

R-Tree는 B-Tree를 공간 인덱스에 맞게 변형한 것으로서, 공간 객체를 표현하기 위해 MBR을 사용하는 높이 균형 트리이다[7]. R-Tree는 중간 노드와 리프 노드로 구성되고, 공간 객체에 대한 모든 참조는 리프 노드에만 존재한다. R-Tree의 중간 노드는 (p, $RECT$)로 나타낼 수 있는데, 여기서 p는

자식 노드에 대한 포인터이고, *RECT*는 자식 노드에 대한 MBR(즉, 노드 MBR)이다. 또한 R-Tree의 리프 노드는 (*oid*, *RECT*)로 나타낼 수 있는데, 여기서 *oid*는 공간 객체가 저장된 디스크 페이지에 대한 포인터이고, *RECT*는 공간 객체에 대한 MBR(즉, 공간 객체 MBR)이다. 그림 1은 R-Tree의 예를 보여준다.



(a) 공간 객체 MBR과 노드 MBR



(b) R-Tree

그림 1. R-Tree 예

그림 1(a)는 공간 객체 MBR과 노드 MBR을 보여준다. 즉, 사각형 R1~R16은 공간 객체 MBR을 보여주고, 사각형 R17~R27은 노드 MBR을 보여준다. 그림 1(b)는 그림 1(a)에 대한 R-Tree를 보여준다. 이와 같이 R-Tree에서는 노드 MBR이 서로 겹치기 때문에 루트 노드에서 리프 노드에 도달하는 검색 경로가 하나 이상 존재할 수 있으며, 공간 객체 검색시 검색 경로 상의 모든 노드를 접근해야 한다. R-Tree에서 노드 접근의 증가는 디스크 I/O의 증가를 의미하며 결국 검색 비용이 높아지게 된다. R-Tree의 성능을 향상시키기 위해 다양한 변형 구조가 제시되었다[3,5,15].

2.2 SH-Tree

본 논문에서 비교 대상으로 하는 SH-Tree[11]는 크기를 갖는 공간 데이타 처리가 가능하도록 Hybrid-Tree[6]를 확장한 인덱스 구조이다. SH-Tree는 크기를 갖는 공간 데이타에 대한 인덱싱이 가능하도록 하기 위해 Hybrid-Tree 구조에 SKD-Tree[12]의 삽입 알고리즘을 적용함으로써 공간 객체가 삽입되는 시점에 공간 객체가 완전히 포함되도록 노드의 분할 경계가 확장된다. 그리고, SH-Tree는 Hybrid-Tree의 노드 분할 알고리즘을 적용함으로써 검색시 디스크 접근 횟수를 줄일 수 있도록 하였다. SH-Tree는 내부 노드를 제외하면 Hybrid-Tree와 동일한 구조를 가지고 있는데, 그림 2는 SH-Tree의 내부 노드를 보여준다.

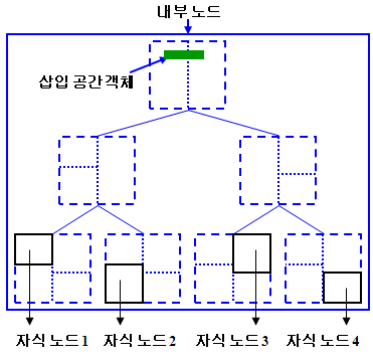


그림 2. SH-Tree의 내부 노드

SH-Tree의 내부 노드는 자식 노드를 구분하는 KD-Tree를 포함하고 있는데, 그림 2에서는 4개의 자식 노드를 가지고 있다. 그리고 내부 노드에 포함된 자식 노드들 사이에는 겹침이 허용된다. 그림 2에서 보듯이 SH-Tree의 내부 노드에 삽입되는 공간 객체가 모든 자식 노드에 완전히 포함되지 않는 경우가 발생할 수 있다. 이러한 경우에 SH-Tree에서는 삽입되는 공간 객체를 완전히 포함하도록 해당 자식 노드의 분할 경계가 확장된다. 그러나 SH-Tree에서는 자식 노드들이 분할 경계를 서로 공유하기 때문에 특정 노드의 분할 경계가 확장되면 그 분할 경계를 공유하는 다른 노드의 분할 경계까지도 확장되면서 노드 간의 겹침이 증가되는 연쇄 겹침 문제(Cascading Overlap Problem)가 발생한다. 그림 3은 SH-Tree에서 연쇄 겹침 문제가 발생하는 예를 보여준다.

그림 3에서 사각형 R1~R10은 공간 객체이고, 사각형 C1~C4는 SH-Tree의 노드 영역이다. 그림 3에서 보듯이 공간 객체 R10이 삽입되면 공간 객체 R10의 중심점(Centroid)이 노드 영역 C2 내에 포함되기 때문에 공간 객체 R10은 노드 영역 C2에 해당

하는 노드에 저장된다. 그러나 공간 객체 R10은 노드 영역 C2 내에 완전히 포함되어 있지 않기 때문에 그림 3과 같이 노드 영역 C2 내에 공간 객체 R10을 완전히 포함되도록 노드 분할 경계가 확장된다. 그러나 노드 영역 C1과 C2는 아래쪽 분할 경계를 서로 공유하기 때문에 공간 객체 R10을 포함시키기 위해 노드 영역 C2의 분할 경계가 확장되면 노드 영역 C1의 분할 경계도 같이 확장되어 노드 간의 겹침이 증가한다. 이러한 노드 간 겹침의 증가는 질의 처리시 검색 경로를 증가시키기 때문에 질의 처리 비용이 증가되는 원인이 된다[11,17,19].

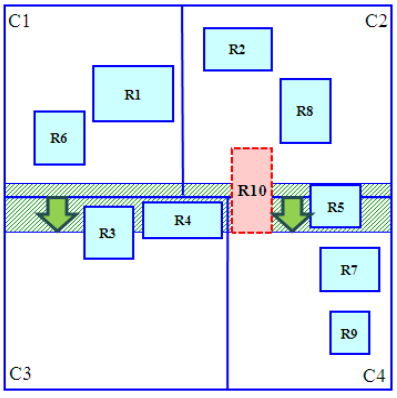


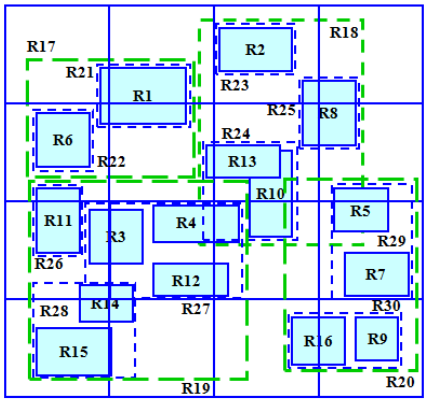
그림 3. 연쇄 겹침 문제의 발생 예

3. SQR-Tree

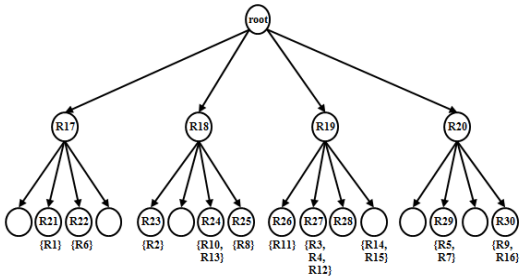
본 장에서는 본 논문에서 제시하는 SQR-Tree를 구성하는 SQ-Tree와 SQR-Tree의 전체 구조에 대해 기술한다.

3.1 SQ-Tree

연쇄 겹침 문제는 노드들이 분할 경계를 서로 공유하기 때문에 발생한다. 그러므로 이를 해결하기 위해서는 각 노드는 독립적인 분할 경계를 갖도록 하여 특정 노드의 분할 경계 확장이 다른 노드의 분할 경계 확장에 영향을 미치지 않도록 해야 한다. 본 논문에서는 노드마다 독립적인 분할 경계를 갖는 Quad-Tree를 확장한 SQ-Tree(Spatial Quad-Tree)를 제시한다. SQ-Tree의 기본 구조는 Quad-Tree와 유사하나, SQ-Tree 노드는 노드마다 가지고 있는 모든 공간 객체를 포함하는 MBR(즉, 노드 MBR)을 갖는다. 그림 4는 그림 1(a)와 동일한 경우에 대한 SQ-Tree의 예를 보여준다.



(a) 공간 객체 MBR 및 노드 MBR

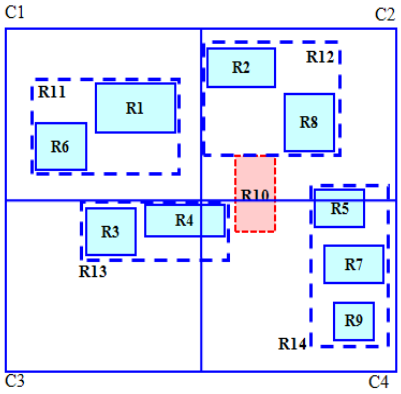


(b) SQ-Tree

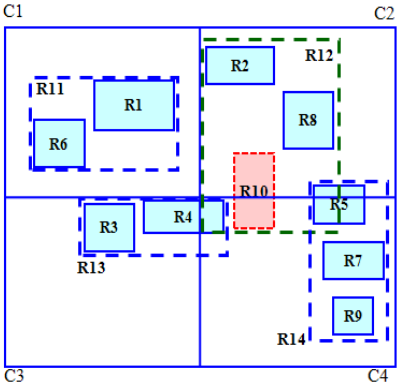
그림 4. SQ-Tree의 예

그림 4(a)에서 실선 사각형 R1~R16은 공간 객체 MBR이고, 점선 사각형 R17~R30은 SQ-Tree 노드의 노드 MBR이다. 그림 4(b)는 그림 4(a)에 대한 SQ-Tree를 보여준다. 삽입시 공간 객체가 저장되는 노드는 공간 객체의 중심점에 따라 결정되고, 저장할 노드가 결정되면 해당 노드의 노드 MBR은 공간 객체를 포함하도록 확장된다. 그림 5는 SQ-Tree에서 공간 객체가 삽입되는 과정을 보여준다.

그림 5는 공간 객체 R10이 삽입되는 과정을 보여준다. 그림 5에서 사각형 C1~C4를 SQ-Tree의 노드 영역이라 할 때, 공간 객체 R10의 중심점은 노드 영역 C2 내에 포함되므로 공간 객체 R10은 노드 영역 C2에 해당하는 노드에 삽입된다. 그림 5(b)는 공간 객체 R10을 포함하기 위해 노드 영역 C2의 노드 MBR이 확장되는 것을 보여준다. 이와 같이 SQ-Tree에서는 공간 객체 MBR가 삽입될 때, 독립적으로 분할 경계가 확장될 수 있기 때문에 SH-Tree와 같은 연쇄 분할 문제가 발생하지 않는다.



(a) 공간 객체 MBR(R10) 삽입



(b) 노드 MBR 확장

그림 5. SQ-Tree에서 공간 객체 삽입

그림 6은 SQ-Tree의 노드 구조를 보여준다. 그림 6(a)에서 보듯이 SQ-Tree 노드는 노드 레벨과 4개의 엔트리로 구성되어 있다. 그림 6(b)는 자식 노드에 대한 노드 MBR과 포인터로 구성되어 있는 중간 노드의 엔트리 구조를 보여준다. 그리고 그림 6(c)는 자신과 연계된 R-Tree의 루트 노드에 대한 노드 MBR과 포인터로 구성되어 있는 리프 노드의 엔트리 구조를 보여준다.

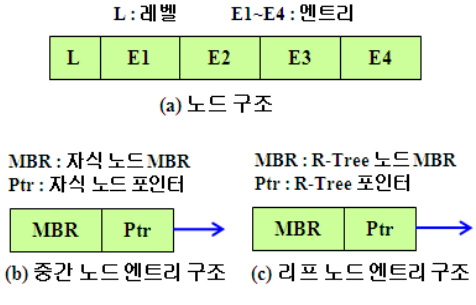


그림 6. SQ-Tree의 노드 구조

이처럼 SQ-Tree에서는 노드마다 노드 MBR을 두어 공간 객체 삽입시 해당 노드의 노드 MBR만을 확장함으로써 연쇄 겹침 문제가 발생하지 않도록 한다. 따라서 연쇄 겹침 문제로 인한 노드간 겹침을 줄일 수 있다. 그러나 Quad-Tree를 기반으로 하는 SQ-Tree는 메모리 기반 인덱스 구조이기 때문에 대용량 공간 데이타 처리에는 적합하지 못하다. 따라서, 본 논문에서는 SQ-Tree는 R-Tree를 식별해주는 기능을 수행하고, 실제 공간 객체는 SQ-Tree의 리프 노드마다 연계되는 R-Tree에 저장되도록 한다.

3.2 SQR-Tree

SQR-Tree는 SQ-Tree와 R-Tree의 결합 인덱스 구조로서 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. SQR-Tree가 실제로 인덱스로 사용시에는 SQ-Tree는 분할된 데이타 공간을 구분해주는 단순한 구조를 갖기 때문에 메인 메모리에 상주하고, 실제 공간 객체가 저장되는 R-Tree는 디스크에 저장하게 된다. 그림 7은 그림 4에 대한 SQR-Tree의 예를 보여준다.

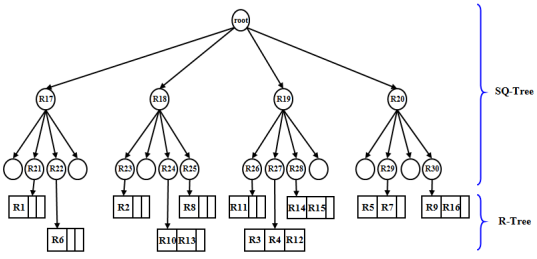


그림 7. SQR-Tree의 예

그림 7에서 보듯이 SQR-Tree의 상위 구조는 SQ-Tree로 구성되고 하위 구조는 SQ-Tree의 리프 노드마다 연계된 R-Tree들로 구성된다. 이처럼 SQR-Tree는 데이타 공간을 여러 하위 공간으로 분할하고 SQ-Tree를 통해 분할된 하위 공간을 검색할 수 있다. 따라서, 공간 질의 처리시 SQR-Tree는 SQ-Tree를 통한 1차적인 필터링을 수행함으로써 여러 R-Tree 중에서 필요한 R-Tree만을 접근

할 수 있다. 또한, SQR-Tree는 노드마다 독립적인 경계 확장이 가능하기 때문에 연쇄 겹침 문제가 발생하지 않아서 노드간 겹침을 줄일 수 있다.

4. 알고리즘

본 절에서는 SQR-Tree를 위한 삽입, 삭제, 검색 알고리즘에 대해 설명한다.

4.1 삽입 알고리즘

SQR-Tree는 SQ-Tree와 R-Tree를 결합한 구조이기 때문에 삽입 과정은 두 인덱스 구조 모두에서 이루어진다. 새로운 공간 객체가 삽입되면 먼저 상위 구조인 SQ-Tree에서 삽입 과정이 수행된다. SQR-Tree에서 공간 객체는 실제로 R-Tree에 저장되기 때문에 SQ-Tree는 단순히 공간 객체를 삽입할 R-Tree를 찾는 경로를 제공한다. 삽입되는 공간 객체는 자신의 중심점에 따라 하위 노드를 결정하여 해당되는 리프 노드에 도달할 때까지 따라 내려간다. 해당되는 리프 노드에 도달하면 이와 연계되어 있는 R-Tree에서 공간 객체 삽입이 실제로 이루어진다. 그림 8은 SQR-Tree에서 공간 객체를 삽입하는 알고리즘을 보여준다.

```
Algorithm : Insert(M, ID, 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, ID, CN);
   Else
7:    RN ← R-Tree node pointed to by I;
8:    CombineMbr(M, N);
9:    InsertRTree(R, ID, RN);
   End If
```

그림 8. 삽입 알고리즘

그림 8의 삽입 알고리즘에서 입력값은 삽입되는 공간 객체의 MBR과 식별자, 그리고 접근하고 있는 SQ-Tree 노드이다. 먼저 삽입되는 공간 객체의 중심점을 구하고 SQ-Tree 노드에서 공간 객체의 중심점을 포함하는 자식 노드를 선택한다. 만일 SQ-Tree 노드가 내부 노드이면 삽입되는 공간 객체 MBR을 포함하도록 SQ-Tree 노드의 노드 MBR을 확장하고, 리프 노드에 도달할 때까지 삽입 알고리즘을 재귀적으로 수행한다. 만일 SQ-Tree 노드가 리프 노드이면(즉, 리프 노드에 도달하면) 삽입되는 공간 객체 MBR을 포함하도록 SQ-Tree 노드의 노드 MBR을 확장하고, SQ-Tree 노드에 연계되어 있는 R-Tree에서 공간 객체를 삽입한다.

4.2 삭제 알고리즘

SQR-Tree에서 삭제 과정은 상위 구조인 SQ-Tree에서 시작되며, SQ-Tree는 삭제할 공간 객체가 저장되어 있는 R-Tree를 찾는 경로를 제공한다. 삭제할 공간 객체의 중심점을 포함하는 하위 노드를 따라 리프 노드까지 도달하면 이와 연계되어 있는 R-Tree에서 공간 객체 삭제가 실제적으로 이루어진다. 그림 9는 SQR-Tree에서 공간 객체를 삭제하는 알고리즘을 보여준다.

그림 9의 삭제 알고리즘에서 입력값은 삭제할 공간 객체의 MBR과 식별자, 그리고 접근하고 있는 SQ-Tree 노드이다. 먼저 삭제할 공간 객체의 중심점을 구하고 SQ-Tree 노드에서 공간 객체의 중심점을 포함하는 자식 노드를 선택한다. 만일 SQ-Tree 노드가 내부 노드이면 리프 노드에 도달할 때까지 삭제 알고리즘을 재귀적으로 수행한다. 만일 SQ-Tree 노드가 리프 노드이면 연계되어 있는 R-Tree에서 공간 객체를 삭제한다. 이때, 공간 객체가 삭제되면 SQ-Tree 노드의 노드 MBR 크기가 줄어들 수 있다. 만일 SQ-Tree 노드의 노드 MBR의 크기가 줄어들면 최악의 경우 루트 노드까지 거슬러 올라가면서 노드 MBR이 조정될 수 있다.

```
Algorithm : Delete(M, ID, 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, ID, CN)) Then
6:        If (MBR size of N is decreased) Then
7:           AdjectMbr(N);
          End If
       End If
    Else
8:     RN ← R-Tree node pointed to by I;
9:     DeleteRTree(M, ID, RN);
10:    If (MBR size of N is decreased) Then
11:       AdjectMbr(N);
       End If
    End If
```

그림 9. 삭제 알고리즘

4.3 검색 알고리즘

SQR-Tree에서는 검색 과정도 마찬가지로 SQ-Tree와 R-Tree 모두에서 이루어진다. 질의 윈도우가 결정되면 먼저 상위 구조인 SQ-Tree에서 검색 과정이 시작된다. SQ-Tree에서 자식 노드 MBR과 질의 윈도우가 서로 겹치는지를 검사하여 접근할 하위 노드를 결정하고, 이 과정을 반복 수행하여 리프 노드에 도달하게 된다. 리프 노드에 도달하면 이와 연계되어 있는 R-Tree에서 공간 객체 검색이 실제적으로 이루어진다. 그림 10은 SQR-Tree에서 공간 객체를 검색하는 알고리즘을 보여준다.

그림 10의 검색 알고리즘에서 입력값은 검색을 위한 질의 윈도우와 접근하고 있는 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. 만일 SQ-Tree 노드가 리프 노드이면(즉, 리프 노드에 도달하면) 이와 연계되어 있는 R-Tree에서 질의 윈도우로 검색을 수행한다.

```
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 ← R-Tree node pointed to by point;
10:       RESULT ← SearchRTree(QW, RN);
        End If
      End For
    End If
11: Return RESULT;
```

그림 10. 검색 알고리즘

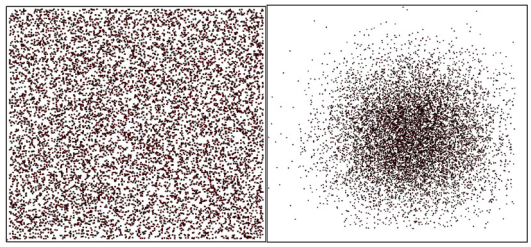
5. 성능 평가

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

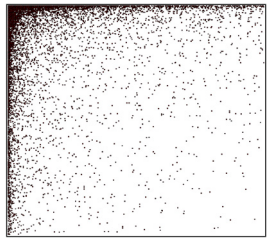
5.1 실험 환경

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 R-Tree, SH-Tree, SQR-Tree를 구현하였다.

본 실험에서는 Uniform, Gauss, Skew 분포를 갖는 공간 데이타를 생성하여 사용하였고, 모든 공간 객체를 포함하는 사각형 영역을 전체 데이타 공간으로 가정하였다. 또한 Uniform, Gauss, Skew 분포를 갖는 공간 데이타는 한 변의 길이가 전체 데이타 공간의 한 변의 길이의 0.01%가 되는 정사각형으로 20만개~100만개를 생성하였다. 그림 11은 본 실험에서 사용된 공간 데이타를 보여준다.



(a) Uniform (b) Gauss

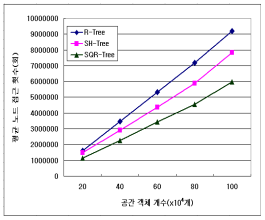


(c) Skew

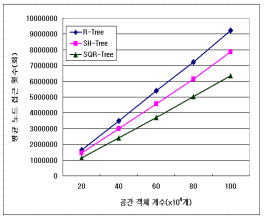
그림 11. 실험에 사용된 공간 데이타

5.2 실험 결과

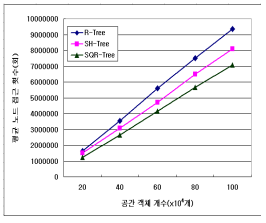
인덱스 생성 실험에서는 한 변의 길이가 전체 데이타 공간의 한 변의 길이의 0.01%가 되는 정사각형인 공간 데이타를 20만개에서 100만개까지 20만개씩 증가시키면서 Uniform, Gauss, Skew 분포인 경우에 R-Tree, SH-Tree, SQR-Tree의 평균 노드 접근 횟수를 비교하였다. 그림 12는 인덱스 생성을 수행한 실험 결과를 보여준다.



(a) Uniform

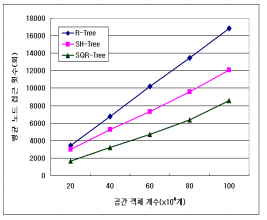


(b) Gauss

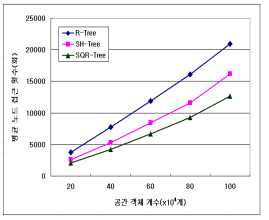


(c) Skew

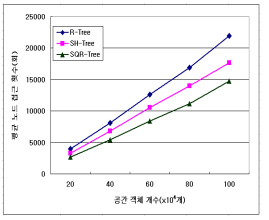
그림 12. 인덱스 생성 실험 결과



(a) Uniform



(b) Gauss



(c) Skew

그림 13. 검색 실험 결과

인덱스 생성 실험에서는 그림 12와 같이 SQR-Tree의 성능이 가장 우수한 것으로 나타났다. 이는 SQR-Tree가 공간 객체를 여러 R-Tree에 분산 저장함으로써 삽입 비용이 줄어들었기 때문이다. Uniform 분포에서는 SQR-Tree가 SH-Tree 보다 평균 30%, R-Tree 보다 평균 50% 성능이 향상되었다. 그리고 Gauss 분포에서는 SQR-Tree가 SH-Tree 보다 평균 25%, R-Tree 보다 평균 45% 성능이 향상되었다. 마지막으로 Skew 분포에서는 SQR-Tree가 SH-Tree 보다 평균 15%, R-Tree 보다 평균 35% 성능이 향상되었다.

검색 실험에서는 범위 질의를 수행하였으며, 범위 질의에서는 전체 데이타 공간의 1%~5%가 되는 질의 사각형으로 질의를 수행하였다. 성능 비교를 위해 1,000번의 검색 질의를 수행하는 동안 R-Tree, SH-Tree, SQR-Tree의 평균 노드 접근 횟수를 비교하였다. 그림 13은 범위 질의를 수행한 실험 결과를 보여준다.

그림 13과 같이 범위 질의에서도 모든 경우에 SQR-Tree의 성능이 가장 우수한 것으로 나타났고, 공간 데이타 개수가 증가하면 성능 차이는 더욱 커지는 것으로 나타났다. 이는 SQR-Tree가 SH-Tree 보다 노드 간 겹침을 줄이고 트리 높이를 낮춤으로써 검색 비용을 줄였기 때문이다. Uniform 분포에서는 SQR-Tree가 SH-Tree 보다 평균 25%, R-Tree 보다 평균 90% 성능이 향상되었다. 그리고 Gauss 분포에서는 SQR-Tree가 SH-Tree 보다 평균 20%, R-Tree 보다 평균 70% 성능이 향상되었다. 마지막으로 Skew 분포에서는 SQR-Tree가 SH-Tree 보다 평균 15%, R-Tree 보다 평균 50% 성능이 향상되었다.

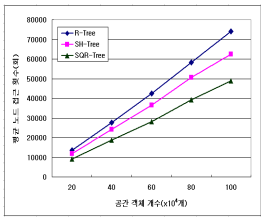
삭제 실험에서는 전체 공간 객체의 1%의 공간 객체를 삭제하는 동안 R-Tree, SH-Tree, SQR-Tree의 평균 노드 접근 횟수를 비교하였다. 그림 14는 삭제 질의 실험 결과를 보여준다.

그림 14와 같이 삭제 질의 실험 결과에서도 SQR-Tree의 성능이 가장 우수한 것으로 나타났다. Uniform 분포에서는 SQR-Tree가 SH-Tree 보다 평균 30%, R-Tree 보다 평균 50% 성능이 향상되었다. 그리고 Gauss 분포에서는 SQR-Tree가 SH-Tree 보다 평균 25%, R-Tree 보다 평균 35% 성능이 향상되었다. 마지막으로 Skew 분포에서는 SQR-Tree가 SH-Tree 보다 평균 10%, R-Tree 보다 평균 20% 성능이 향상되었다.

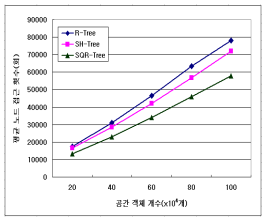
위의 실험 결과와 같이 SQR-Tree는 다양한 공간 분포를 갖는 대용량 공간 데이타에 대해 검색, 삽입, 삭제 성능이 SH-Tree, R-Tree 보다 우수한 것으로 나타났다.

6. 결 론

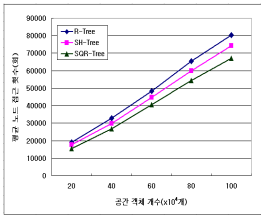
본 논문에서는 SH-Tree의 연쇄 겹침 문제를 해결하고 대용량 공간 데이타를 효율적으로 처리하기 위하여 SQR-Tree를 제안하였다. SQR-Tree는 SQ-



(a) Uniform



(b) Gauss



(c) Skew

그림 14. 삭제 실험 결과

Tree와 R-Tree를 결합한 인덱스 구조로서 SQ-Tree를 이용하여 노드마다 독립적으로 분할 경계를 확장할 수 있기 때문에 연쇄 겹침 문제가 발생하지 않는다. 그리고, SQR-Tree에서는 공간 객체가 여러 R-Tree에 분산 저장되기 때문에 공간 질의 처리시 해당 R-Tree만을 접근하면 되기 때문에 공간 질의 처리 성능이 향상된다. 특히, SQR-Tree는 R-Tree의 큰 변경없이 구현이 가능하고 다양한 R-Tree 변형에도 쉽게 적용할 수 있는 장점이 있다. 실험을 통해 SQR-Tree의 성능이 기존 인덱스보다 우수함을 입증하였다.

그러나 Gauss, Skew 분포와 같이 공간 객체 분포가 균등하지 않은 환경에서는 SQ-Tree의 높이가 높아지면서 R-Tree가 증가함에 따라 성능이 저하되는 문제가 있다. 향후에는 비균등 분포에서도 질의 처리 성능을 향상시키도록 SQR-Tree의 구조를 개선하는 것이 필요하겠다.

참 고 문 헌

[1] H. K. Ahn, N. Mamoulis, and H. M. Wong, 2001, A Survey on Multidimensional Access Methods, Technical Report, UU-CS-2001-14.

[2] 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.

[3] 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.

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

[5] S. Berchtold, D. A. Keim, and H. Kriegel, 1996, “The X-Tree : An Index Structure for High-Dimensional Data,” Proc. of 22th Int. Conf. on Very Large Data Bases, pp. 28-39.

[6] K. Chakrabarti, and S. Mehrotra, 1999, “The Hybrid Tree: An Index Structure for High Dimensional Feature Spaces,” Proc. of the Int. Conf. on Data Engineering, pp. 440-447.

[7] 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.

[8] 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.

[9] 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.

[10] H. Y. Lin, P. W. Huang, and K. H. Hsu, 2007, “A New Indexing Method with High Storage Utilization and Retrieval Efficiency for Large Spatial Databases,” Information and Software Technology, vol. 49, no. 8, pp. 817-826.

[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] B. C. Ooi, K. J. McDonell, and R. Sacks-Davis, 1987, “Spatial KD-Tree: An Indexing Mecha-

nism for Spatial Databases," Proc. of Int. Conf. on Computer Software and Applications, pp. 433-438.

[13] C. G. Park, H. K. Kang, J. J. Kim, and K. J. Han, 2009, "A Hash-based R-Tree for Fast Search of Large Spatial Data," Proc. of the 1st Int. Conf. on Emerging Databases, pp. 45-50.

[14] J. T. Robinson, 1981, "The K-D-B-Tree: A Search Structure for Large Multi-dimensional Dynamic Indexes," Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp. 10-18.

[15] 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.

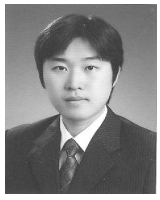
[16] X. Wu, and C. Zang, 2009, "A New Spatial Index Structure for GIS Data," Proc. of the 3rd Int. Conf. on Multimedia and Ubiquitous Engineering, pp. 471-476.

[17] 강홍구, 김정준, 신인수, 한기준, 2008,"대용량 공간 데이타의 빠른 검색을 위한 해시 기반 R-Tree," 공동 추계학술대회 논문집, 한국지형공간정보학회, pp. 82-89.

[18] 김학철, 2010, "다차원 공간의 효율적인 그리드 분할을 통한 디클러스터링 알고리즘 성능향상 기법," 한국공간정보시스템학회 논문지, 제12권, 제1호, pp. 37-48.

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

논문접수 : 2011.02.08
수 정 일 : 2011.03.16
심사완료 : 2011.03.20



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 공간데이타베이스, GIS, LBS, USN

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

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