MR-Tree: 효율적인 공간 검색을 위한 매핑 기반 R-Tree

MR-Tree: A Mapping-based R-Tree for Efficient Spatial Searching

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 최근, u-GIS 환경에서 다양한 지오센서(Geosensor)의 활용으로 수집되는 공간 데이타의 양이 급증하면서 대용량 공간 데이타의 효율적인 검색을 위한 공간 인덱스의 중요성이 높아지고 있다. 특히, 공간 데이타의 검색 성능을 높이기 위해 R-Tree를 기반으로 한 공간 인덱스에 대한 연구가 활발히 진행되고 있다. 그러나 기존 연구는 R-Tree에서 노드 사이의 겹침이나 트리의 높이를 줄임으로써 어느 정도 검색 성능을 향상시켰지만 트리 순회(tree traversal)에서 발생하는 불필요한 노드 접근 문제를 효율적으로 해결하지 못하고 있다. 본 논문에서는 이러한 문제를 해결하기 위하여 대용량 공간 데이타의 효율적인 검색을 위한 매핑 기반 R-Tree인 MR-Tree(Mapping based R-Tree)를 제안한다. MR-Tree는 R-Tree 순회 없이 리프 노드를 직접 접근하도록 하는 매핑 트리를 이용함으로써 검색 성능을 향상시킨다. 매핑 트리는 데이타 공간을 차원에 따라 반복적으로 분할한 각 파티션(Partition)과 연계되는 R-Tree 리프 노드의 MBR과 포인터를 이용하여 구성된다. 특히, MR-Tree는 기존 R-Tree에 큰 변경 없이 구현이 가능하고, 다양한 R-Tree 변형에도 쉽게 적용할 수 있으며, 또한 매핑 트리를 메인 메모리에 상주시킴으로써 검색 시간을 단축시킬 수 있다. 마지막으로 실험을 통해 기존 인덱스보다 MR-Tree 성능의 우수성을 보였다.

키워드 : 공간 데이타, 공간 검색, 매핑 트리, R-Tree, MR-Tree

Abstract Recently, due to rapid increasement of spatial data collected from various geosensors in u-GIS environments, the importance of spatial index for efficient search of large spatial data is rising gradually. Especially, researches based R-Tree to improve search performance of spatial data have been actively performed. These previous researches focus on reducing overlaps between nodes or the height of the R-Tree. However, these can not solve an unnecessary node access problem efficiently occurred in tree traversal. In this paper, we propose a MR-Tree(Mapping-based R-Tree) to solve this problem and to support efficient search of large spatial data. The MR-Tree can improve search performance by using a mapping tree for direct access to leaf nodes of the R-Tree without tree traversal. The mapping tree is composed with MBRs and pointers of R-Tree leaf nodes associating each partition which is made by splitting data area repeatedly along dimensions. Especially, the MR-Tree can be adopted in various variations of the R-Tree easily without a modification of the R-Tree structure. In addition, because the mapping tree is constructed in main memory, search time can be greatly reduced. Finally, we proved superiority of MR-Tree performance through experiments.

Keywords : Spatial Data, Spatial Search, Mapping Tree, R-Tree, MR-Tree

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

[*] 한국인터넷진흥원 융합보호 R&D팀 선임연구원 hkkang@kisa.or.kr
[**] 건국대학교 컴퓨터공학부 박사과정 isshin@db.konkuk.ac.kr
[***] 건국대학교 컴퓨터공학부 강의교수 jjkim9@db.konkuk.ac.kr(교신저자)
[****] 건국대학교 컴퓨터공학부 교수 kjhan@db.konkuk.ac.kr

1. 서 론

u-GIS란 유비쿼터스 시대에서 사용자들의 다양한 요구에 적합한 맞춤형 정보 서비스를 제공하기 위하여 공간 데이타와 센서 데이타의 융·복합 저장 및 처리를 위해 독립적으로 발전해온 GIS 기술과 USN 기술을 접목한 것을 의미한다[11,16]. 이러한 u-GIS 환경에서는 위치 및 경계 데이타를 포함한 다양한 공간 데이타를 수집하는 RFID, GPS, Digital Camera 센서 같은 지오센서(Geosensor)의 활용으로 저장되는 공간 데이타 양이 급속히 증가하고 있으며 이러한 대용량 공간 데이타의 빠른 처리를 위한 공간 인덱스의 중요성이 높아지고 있다[4,7,8,17].

기존의 공간 트리 인덱스는 크게 공간 분할(Space Partitioning) 기반 인덱스[5,6]와 데이타 분할(Data Partitioning) 기반 인덱스[1,9,10]로 나눌 수 있다. 공간 분할 기반 인덱스는 데이타 공간을 서로 겹치지 않는 하위 공간으로 분할하는 것이 특징으로 크기를 갖지 않는 점 데이타 처리에 효율적이지만 크기를 갖는 공간 데이타 처리에는 적합하지 않다. 반면에, 데이타 분할 기반 인덱스는 계층적인 포함(Containment) 관계를 이용하여 인덱스 구조를 생성하는 것이 특징으로 노드 사이의 겹침(Overlap)을 허용하기 때문에 점 데이타와 크기를 갖는 공간 데이타의 처리가 모두 가능하다.

특히, 가장 널리 사용되고 있는 데이타 분할 기반 인덱스인 R-Tree[1]를 기반으로 검색 성능을 향상시키기 위한 연구가 활발히 진행되고 있다. 제시된 대표적인 인덱스에는 R+-Tree[12], R*-Tree[9], X-Tree[10], QR-Tree[13,14], iQR-Tree[3] 등이 있다. 이들 인덱스는 노드 사이의 겹침이나 트리 높이를 줄임으로써 검색 성능을 향상시켰다. R+-Tree는 공간 객체를 서로 겹치는 여러 노드에 중복 저장하여 내부 노드 사이의 겹침을 없앴고 R*-Tree는 분할시 노드 재삽입(Reinsertion)을 이용하여 노드 사이의 겹침을 줄였다. 그리고 X-Tree는 노드 크기가 큰 슈퍼 노드(Super Node)를 적용하여 노드 사이의 겹침을 줄였다. QR-Tree와 iQR-Tree는 Quad-Tree와 R-Tree의 결합 구조로서 전체 데이타 공간을 여러 하위 공간으로 분할하고 분할된 데이타 공간마다 R-Tree를 구성하여 전체적인 트리 높이를 줄였다. 그러나 이러한 인덱스들은 트리 순회(tree traversal) 과정에서 발생하는 불필요한 노드 접근 비용을 효율적으로 해결하지 못하는 문제를 가지고 있다[2,15].

본 논문에서는 이러한 문제를 해결하기 위해서 대용량 공간 데이타의 빠른 검색을 지원하는 인덱스인 MR-Tree(Mapping based R-Tree)를 제안한다. MR-Tree에서는 데이타 공간은 차원에 따라 서로 다른 크기를 갖는 여러 파티션(Partition)들로 분할되고[6], 파티션은 분할 과정에서 계층 구조를 갖는다. 그리고 생성된 파티션과 대응되는 R-Tree 리프 노드의 MBR과 포인터를 이용하여 매핑 트리를 구성한다. 매핑 트리는 분할된 데이타 공간을 나타내는 파티션과 R-Tree 리프 노드를 매핑시키는 기능을 가진다.

따라서 MR-Tree는 검색시 R-Tree의 루트 노드에서 리프 노드까지의 트리 순회 없이 검색 영역에 해당하는 파티션을 찾은 후 매핑 트리를 통해 해당 R-Tree 리프 노드를 직접 접근하기 때문에 검색 성능을 향상시킬 수 있다. 또한, MR-Tree에서는 메모리에 적합한 구조인 매핑 트리를 메인 메모리에 상주시킴으로써 검색시 응답 시간을 단축시킨다. 특히, MR-Tree는 R-Tree의 큰 변경없이 구현이 가능하고 다양한 R-Tree 변형에도 쉽게 적용할 수 있는 장점이 있다. 마지막으로 실험을 통해 기존 연구와 비교하여 MR-Tree의 검색 성능이 우수함을 입증하였다.

본 논문의 구성은 다음과 같다. 2장의 관련 연구에서는 R-Tree와 iQR-Tree에 대해 설명한다. 3장에서는 파티션, 매핑 트리, MR-Tree 구조, 알고리즘에 대해 기술한다. 4장에서는 다양한 분포를 갖는 대용량 공간 데이타를 이용한 실험을 통해 MR-Tree의 성능을 평가한다. 마지막으로 5장에서는 결론에 대해 언급한다.

2. 관련 연구

본 장에서는 널리 사용되고 있는 R-Tree와 R-Tree의 검색 성능을 향상시킨 iQR-Tree에 대해 설명한다.

2.1 R-Tree

R-Tree는 B-Tree를 공간 인덱스에 맞게 변형한 것으로서, 공간 객체를 표현하기 위해 MBR을 사용

하는 높이 균형 트리이다[1]. 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, RECT는 자식 노드에 대한 MBR(즉, 노드 MBR)이다. 또한 R-Tree의 리프 노드는(oid, RECT)로 나타낼 수 있는데, 여기서 oid는 공간 객체가 저장된 디스크 페이지에 대한 포인터이고, RECT는 공간 객체에 대한 MBR(즉, 공간 객체 MBR)이다. 그림 1은 R-Tree의 예를 보여준다.

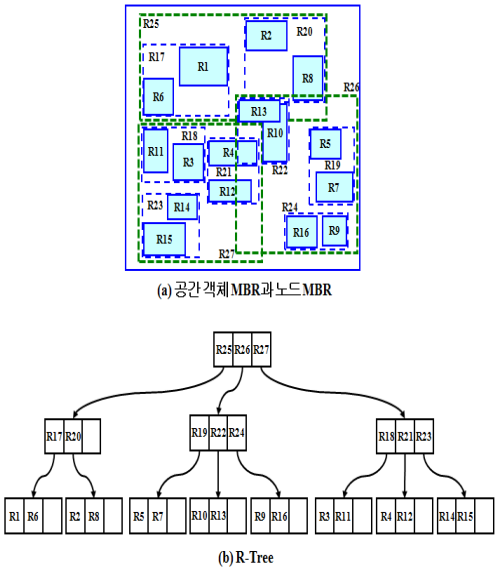


그림 1. R-Tree 예

그림 1(a)는 공간 객체 MBR과 노드 MBR을 보여준다. 즉, 실선 사각형 R1~R16은 공간 객체 MBR을 보여주고, 점선 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[9,10]. R-Tree에서 노드 접근의 증가는 디스크 I/O의 증가를 의미하며 결국 검색 비용이 높아지게 된다. R-Tree의 검색 비용을 줄이기 위해 다양한 변형 구조가 제시되었다. 대표적인 변형 구조로서 R+-Tree, R*-Tree, X-Tree, QR-Tree, iQR-Tree 등이 있다.

2.2 iQR-Tree

최근 제안된 iQR-Tree는 Quad-Tree와 R-Tree의 결합 인덱스인 QR-Tree[14]를 확장한 인덱스이다[3]. 기존 QR-Tree에서는 각각의 Quad-Tree 리프 노드가 R-Tree에 연계되어 있는 구조를 가지고 있으며, 실제로 공간 객체는 R-Tree에 저장된다. 그러나, 삽입되는 공간 객체가 Quad-Tree의 하위 분할 경계에 걸치는 경우에는 공간 객체를 쪼개서(Clipping) 여러 하위 노드에 중복 저장되어 인덱스 성능이 저하되는 문제를 가지고 있다. iQR-Tree에서는 각각의 Quad-Tree 리프 노드뿐만 아니라 내부 노드도 R-Tree에 연계시킴으로써 QR-Tree의 중복 저장 문제를 해결한다. 그림 2는 iQR-Tree의 예를 보여준다.

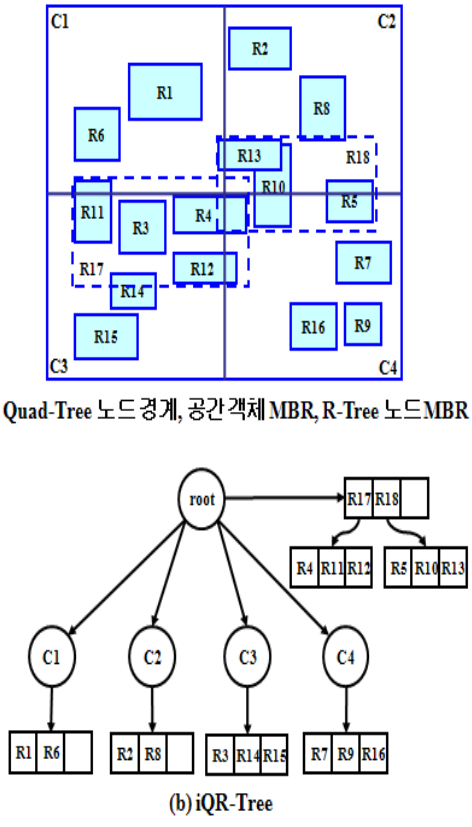


그림 2. iQR-Tree 예

그림 2(a)는 Quad-Tree 노드 경계, 공간 객체 MBR, R-Tree 노드 MBR을 보여준다. 즉, 사각형 C1~C4는 Quad-Tree 노드 경계를 보여주고, 실선 사각형 R1~R16은 공간 객체 MBR을 보여주며, 점

선 사각형 R17과 R18은 R-Tree 노드 MBR을 보여준다. 그림 2(b)는 그림 2(a)에 대한 iQR-Tree를 나타낸다. 그림 2(b)에서 원(Circle)은 Quad-Tree의 노드를 나타내고 있으며, 각각의 Quad-Tree 노드는 R-Tree와 연계되어 있다.

iQR-Tree에서는 삽입되는 공간 객체가 Quad-Tree의 하위 분할 경계와 겹치는 경우에는 공간 객체를 포함할 수 있는 상위 노드와 연계되는 R-Tree에 저장되고, 그렇지 않은 경우에는 하위 노드와 연계되는 R-Tree에 저장된다. 예를 들어, 그림 2에서와 같이 공간 객체 MBR R4, R5, R10, R11, R12, R13은 Quad-Tree 루트 노드의 하위 분할 경계와 겹치기 때문에 대응되는 공간 객체는 루트 노드와 연계된 R-Tree에 저장되고 나머지 공간 객체는 하위 노드와 연계되는 R-Tree에 저장된다.

iQR-Tree는 단순히 데이타 공간을 구분해주는 Quad-Tree를 메인 메모리에 구성하고, 실제 공간 객체를 저장하는 R-Tree를 디스크에 구성되는 이중 구조를 가지고 있다. iQR-Tree는 분할된 데이타 공간마다 R-Tree를 가지므로 트리 높이가 낮아지고, 검색 영역에 해당하는 R-Tree만 접근함으로써 검색 성능이 향상된다. 그러나 Quad-Tree의 레벨이 높아지면 R-Tree 간의 겹침이 급격히 증가하기 때문에 검색 성능이 크게 저하되는 문제가 있다 [3,13,14].

3. MR-Tree

본 장에서는 파티션, 매핑 트리, MR-Tree 구조, 그리고 알고리즘에 대해 기술한다.

3.1 파티션

MR-Tree에서는 R-Tree의 리프 노드를 직접 접근하게 해주는 매핑 트리를 구성하기 위해 데이타 공간을 서로 다른 크기의 여러 파티션으로 분할한다. 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. 그림 3은 X-Y 2차원 평면에서 파티션 생성 과정을 보여준다.

그림 3(a)는 전체 데이타 공간을 나타내는 파티션에 오버플로우가 발생하여 X축을 분할 축으로 하여 같은 크기를 갖는 두 개의 하위 파티션으로 분할된 것을 보여준다. 그림 3(b)는 그림 3(a)의 좌측 파티션에 오버플로우가 발생하여 Y축을 분할 축으로 하여 같은 크기를 갖는 두 개의 하위 파티션으로 분할된 것을 보여준다. 그림 3(c)와 그림 3(d)도 마찬가지로 분할 축을 재순환(Recycle)하면서 연속적으로 파티션이 생성되는 예를 보여준다.

중간 파티션은 분할 경계와 겹치는 R-Tree 리프 노드와 연계되며, 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 연계되며, 파티션 내에 완전히 포함되는 R-Tree 리프 노드가 없는 경우를 포함하여 최대 하나의 R-Tree 리프 노드와 연계될 수 있다. 만일 리프 파티션에 새로운 R-Tree 리프 노드가 추가되어 오버플로우가 발생되면 리프 파티션은 분할된다. 각 파티션은 자신과 연계되는 모든 R-Tree 리프 노드를 포함하는 MBR (즉, 파티션 MBR)을 가진다.

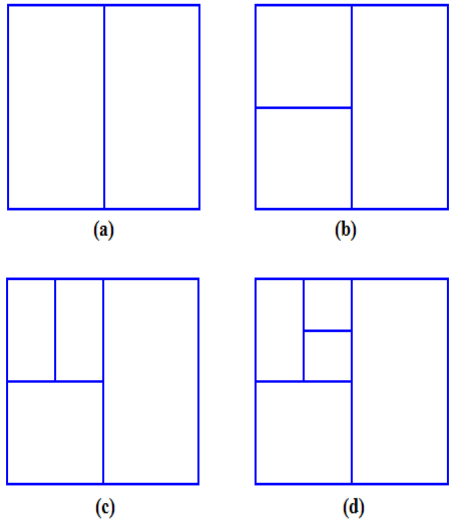


그림 3. 파티션 생성 과정

이러한 파티션의 주요 특징을 정리하면 다음과 같다.

1) 데이타 공간은 서로 다른 크기를 가지는 파티션들로 분할된다.

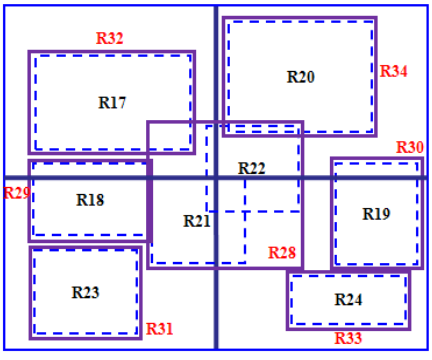
2) 파티션은 계층적인 형태를 가진다.

3) 각 파티션은 R-Tree 리프 노드와 연계된다.

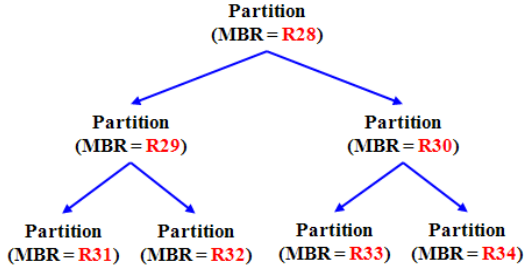
4) 각 파티션은 연계된 모든 R-Tree 리프 노드를 포함하는 파티션 MBR을 가진다.

그림 4는 그림 1의 R-Tree에 대한 R-Tree 리프 노드 MBR, 파티션 MBR, 그리고 파티션 계층 구조를 보여준다.

그림 4(a)는 R-Tree 리프 노드 MBR과 파티션 MBR을 보여준다. 즉, 점선 사각형 R17~R24는 R-Tree 리프 노드 MBR을 보여주고, 실선 사각형 R28~R34는 파티션 MBR을 보여준다. 그림 4(b)는 그림 4(a)에 대한 파티션 계층 구조를 나타낸다. 그림 4(b)와 같이 중간 파티션은 3개 존재하고, 이들은 각각 파티션 MBR R28, R29, R30을 갖는다. 중간 파티션은 2개 이상의 R-Tree 리프 노드와 연계될 수 있다. 예를 들어, 파티션 MBR R28을 갖는 파티션은 R-Tree 리프 노드 R21, R22에 연계되어 있다. 여기서 파티션 MBR R28을 갖는 파티션은 최상위 파티션으로 전체 데이타 공간을 나타낸다. 그리고 그림 4(b)에서 리프 파티션은 4개 존재하고, 이들은 각각 파티션 MBR R31, R32, R33, R34를 갖는다. 리프 파티션은 최대 하나의 R-Tree 리프 노드와 연계될 수 있다.



(a) R-Tree 리프 노드 MBR과 파티션 MBR

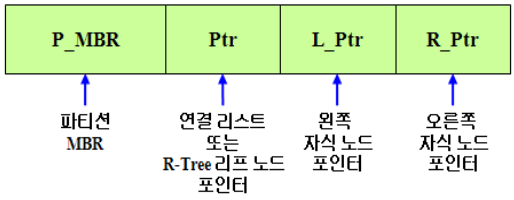


(b) 파티션 계층 구조

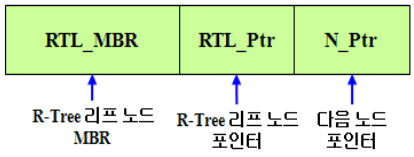
그림 4. R-Tree 리프 노드 MBR, 파티션 MBR, 파티션 계층 구조

3.2 매핑 트리

매핑 트리는 파티션마다 연계되는 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 연결해주고, 연결 리스트 노드는 파티션과 연계된 2개 이상의 R-Tree 리프 노드를 연결해준다. 그림 5는 매핑 트리 노드 구조와 연결 리스트 구조를 보여준다.



(a) 매핑 트리 노드 구조



(b) 연결 리스트 노드 구조

그림 5. 매핑 트리 노드 구조와 연결 리스트 노드 구조

그림 5(a)에서 보는 바와 같이 매핑 트리 노드는 파티션 MBR, 연결 리스트 또는 단일 R-Tree 리프 노드 가리키는 포인터, 왼쪽/오른쪽 자식 노드를 가리키는 2개의 포인터로 구성된다. 이때, 매핑 트리 노드가 연결 리스트 노드를 가리키는 포인터를 갖는 경우는 파티션과 연계되는 R-Tree 리프 노드가 2개 이상인 경우이고, 단일 R-Tree 리프 노드를 가리키는 포인터를 갖는 경우는 파티션과 연계되는 R-Tree 리프 노드가 1개인 경우이다. 그리고 그림 5(b)에서 보는 바와 같이 연결 리스트 노드는 R-Tree 리프 노드 MBR, R-Tree 리프 노드 포인터, 그리고 연결 리스트의 다음 노드를 가리키는 포인터로 구성된다. 그림 6은 그림 4에 대한 매핑 트리 예를 보여준다.

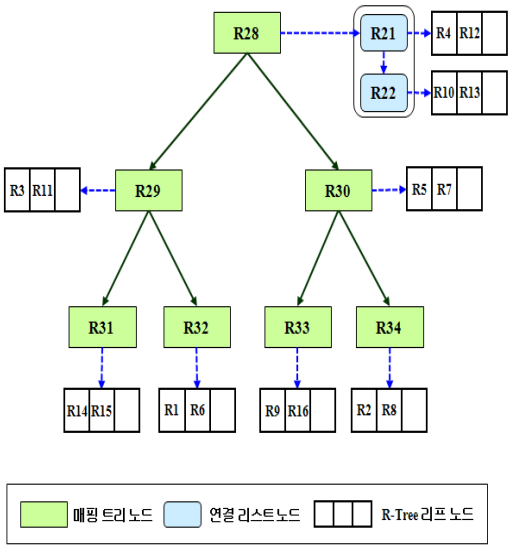


그림 6. 매핑 트리 예

그림 6과 같이 중간 파티션은 2개 이상의 R-Tree 리프 노드와 연계될 수 있기 때문에 매핑 트리는 연결 리스트를 통해 여러 R-Tree 리프 노드를 가리키게 되고, 반면에 리프 파티션은 최대 한 개의 R-Tree 리프 노드와 연계될 수 있기 때문에 매핑 트리는 연결 리스트 없이 해당 R-Tree 리프 노드를 가리키게 된다. 그림 6의 매핑 트리에서 파티션 MBR R28에 대응되는 매핑 트리 노드는 2개 R-Tree 리프 노드의 MBR과 포인터를 가지는 연결 리스트를 가리키는 포인터를 가지고 있으나, 다른 매핑 트리 노드는 모두 하나의 R-Tree 리프 노드를 직접 가리키는 포인터를 가지고 있다.

3.3 MR-Tree 구조

MR-Tree는 기존 R-Tree에 R-Tree 리프 노드를 직접 접근할 수 있는 매핑 트리를 적용한 인덱스 구조이다. 그림 7은 그림 1의 R-Tree에 매핑 트리를 적용한 MR-Tree의 전체 구조를 보여준다.

그림 7과 같이 MR-Tree는 R-Tree와 매핑 트리로 구성되어 있고, 매핑 트리는 파티션마다 연계된 R-Tree 리프 노드를 가리키고 있다. MR-Tree는 R-Tree에 기반한 인덱스 구조이지만 검색은 R-Tree의 루트 노드가 아닌 매핑 트리의 루트 노드에서 시작된다. MR-Tree에서는 매핑 트리의 검색 결과에 해당되는 매핑 트리 노드 또는 연결 리스트 노드는 R-Tree 리프 노드를 가리키고 있기 때문에 검색시 R-Tree의 트리 순회없이 R-Tree 리프 노드를 직접 접근할 수 있다.

반면에 MR-Tree에서 공간 객체의 삽입과 삭제는 검색과는 달리 R-Tree 루트 노드에서 시작된다. MR-Tree에 삽입되는 공간 객체는 최종적으로 R-Tree 리프 노드에 삽입되는데, 이때 새로운 공간 객체 추가로 해당 R-Tree 리프 노드 MBR이 커질 수 있다. MR-Tree에서는 공간 객체 삽입시 R-Tree 리프 노드 MBR이 변경되는 경우에만 매핑 트리의 해당 매핑 트리 노드 또는 연결 리스트 노드가 갱신된다. 공간 객체 삭제시에도 마찬가지로 R-Tree 리프 노드 MBR이 변경되는 경우에만 매핑 트리의 해당 매핑 트리 노드 또는 연결 리스트 노드가 갱신된다. 특히, 공간 객체 삽입이나 삭제시에 R-Tree 리프 노드의 최대 엔트리 개수가 초과되거나 미달될 수 있다. 만약에 MR-Tree에서 R-Tree 리프 노드의 최대 엔트리 개수가 초과되거나 미달되면 분할이나 병합이 발생된다.

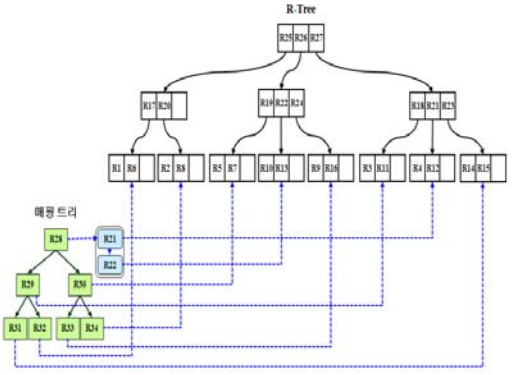


그림 7. MR-Tree의 전체 구조

3.4 알고리즘

MR-Tree에서 매핑 트리의 삽입, 삭제는 R-Tree의 삽입, 삭제 과정과 연결된다. MR-Tree의 R-Tree에서 공간 객체가 리프 노드에 삽입되거나 삭제되면 단순히 R-Tree 리프 노드 MBR의 크기가 변경될 수도 있지만, 노드 분할이나 합병으로 R-Tree 리프 노드가 새롭게 추가되거나 삭제될 수도 있다. 이처럼 R-Tree 리프 노드 MBR이 생성, 변경, 삭제되는 경우에 매핑 트리에서는 해당되는 R-Tree 리프 노드 MBR과 포인터에 대한 삽입, 변경, 삭제가 수행된다. 만일 R-Tree에 공간 객체 삽

입, 삭제시 R-Tree 리프 노드 MBR에 대한 변경이 없으면 매핑 트리는 갱신될 필요가 없다.

그림 8은 MR-Tree에서 새로운 R-Tree 리프 노드를 삽입하는 알고리즘을 보여준다.

```
Algorithm : Insert(R, P, MTN)
Begin
 1: If(MTN is leaf) Then
 2:   If(MTN overflows) Then
 3:     Split(MTN);
 4:     Insert(R, P, MTN);
      Else
 5:     InsertRTN(R, P);
      End If
    Else
 6:   If (R overlaps split line of current partition) Then
 7:     InsertRTN(R, P);
 8:   Else If (left or bottom partition containes R) Then
 9:     Insert(R, P, LeftBottom(MTN));
10:   Else If (right or top partition containes R) Then
11:     Insert(R, P, RightTop(MTN));
      End If
    End If
End
```

그림 8. 삽입 알고리즘

그림 8의 삽입 알고리즘에서 입력값은 삽입할 R-Tree 리프 노드 MBR과 포인터, 그리고 매핑 트리 노드이다. 먼저 매핑 트리 노드가 리프 파티션이면 오버플로우가 되는지 검사한다. 만일 매핑 트리 노드가 오버플로우이면 매핑 트리 노드를 분할하고 다시 R-Tree 리프 노드 삽입하기 위해 Insert()를 호출한다. 그렇지 않고 매핑 트리 노드가 오버플로우가 되지 않으면 해당 매핑 트리 노드에 R-Tree 리프 노드를 삽입하도록 InsertRTN()를 호출한다.

매핑 트리 노드가 중간 파티션이면 R-Tree 리프 노드 MBR이 파티션의 분할 경계와 서로 겹치는지 검사한다. 만일 R-Tree 리프 노드 MBR이 파티션의 분할 경계와 서로 겹치면 해당 매핑 트리 노드에 R-Tree 리프 노드를 삽입하기 위해 Insert()를 호출한다. 그렇지 않으면, 현재 매핑 트리 노드의 두 개 자식 노드에 R-Tree 리프 노드 MBR이 포함되는지 검사하고 R-Tree 리프 노드 MBR을 포함하는 매핑 트리 노드로 이동하여 삽입 알고리즘을 재귀적으로 수행한다.

그림 9는 매핑 트리에서 기존의 R-Tree 리프 노드를 삭제하는 알고리즘을 보여준다.

그림 9의 삭제 알고리즘에서 입력값은 삭제할 R-Tree 리프 노드 MBR과 포인터, 그리고 매핑 트리 노드이다. 먼저 매핑 트리 노드로부터 파티션 MBR을 읽어서 변수 M1에 저장한다. 만일 변수 M1에 저장된 파티션 MBR이 삭제할 R-Tree 리프 노드 MBR을 완전히 포함하면 매핑 트리 노드에서 입력값인 R-Tree 리프 노드 MBR과 포인터와 같은 R-Tree 리프 노드를 찾아 삭제하기 위해 DeleteRTN()을 호출한다. 이때, 매핑 트리 노드가 리프 파티션이고 언더플로우가 발생하면 매핑 트리 노드를 삭제하기 위해 Merge()를 호출한다. 만일 변수 M1에 저장된 파티션 MBR이 삭제할 R-Tree 리프 노드 MBR을 완전히 포함하지 않으면 현재 매핑 트리 노드의 두 개 자식 노드에 R-Tree 리프 노드 MBR이 포함되는지 검사하고 R-Tree 리프 노드 MBR을 포함하는 매핑 트리 노드로 이동하여 삭제 알고리즘을 재귀적으로 수행한다.

```
Algorithm : Delete(R, P, MTN)
Begin
 1: M1 ← partition MBR in MTN;
 2: If(M1 containes R) Then
 3:   DeleteRTN(R, P);
 4:   If(MTN is leaf and underflows) Then
 5:     Merge(MTN);
      End If
    Else
 6:   If (left or bottom partition containes R) Then
 7:     Delete(R, P, LeftBottom(MTN));
 8:   Else If (right or top partition containes R) Then
 9:     Delete(R, P, RightTop(MTN));
      End If
  End If
End
```

그림 9. 삭제 알고리즘

그림 10은 MR-Tree에서의 검색 알고리즘을 보여준다.

MR-Tree의 검색은 트리 순회 없이 매핑 트리를 통해 R-Tree 리프 노드를 직접 접근하여 수행된다. 그림 10의 검색 알고리즘에서 입력값은 질의 사각형과 매핑 트리 노드이다. 먼저 매핑 트리 노드로부터 파티션 MBR을 읽어서 변수 M1에 저장한다. 만일 변수 M1에 저장된 파티션 MBR과 질의 사각형이 서로 겹치면 해당 파티션에 연계된 R-Tree 리프 노드의 MBR과 질의 사각형이 서로 겹치는지 검사한다. 그리하여 질의 사각형과 서로 겹치는 실제 R-Tree 리프 노드를 바로 접근하여 공간 객체를 검색하기 위해 SearchRTN()을 호출한다. 만일

파티션 MBR과 질의 사각형이 서로 겹치지 않으면 현재 매핑 트리 노드의 두 개 자식 노드에서 검색 알고리즘을 재귀적으로 수행한다.

```
Algorithm : Search(QW, MTN)
Begin
 1: RESULT ← Ø;
 2: M1 ← partition MBR in MTN;
 3: If (MBR is null) Then
 4:   Return RESULT;
    End If
 5: If (M1 overlaps QW) Then
 6:   P1 ← pointer in MTN;
 7:   RESULT ← SearchRTN(QW, P1);
    End If
 8: RESULT ← Search(QW, LeftBottom(MTN));
 9: RESULT ← Search(QW, RightTop(MTN));
10: Return RESULT;
End
```

그림 10. 검색 알고리즘

4. 성능 평가

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

4.1 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, iQR-Tree, MR-Tree를 구현하였다. 본 실험에서는 Uniform, Gauss, Skew 분포를 갖는 공간 데이타를 생성하여 사용하였고, 공간 객체를 모두 포함하는 사각형 영역을 전체 데이타 공간으로 가정하였다. Uniform, Gauss, Skew 분포를 갖는 공간 데이타로는 한 변의 길이가 전체 데이타 공간의 한 변의 길이의 0.01%가 되는 정사각형을 20만개~100만개를 생성하여 사용하였다. 그림 11은 실험에서 사용된 공간 데이타를 보여준다.

4.2 실험 결과

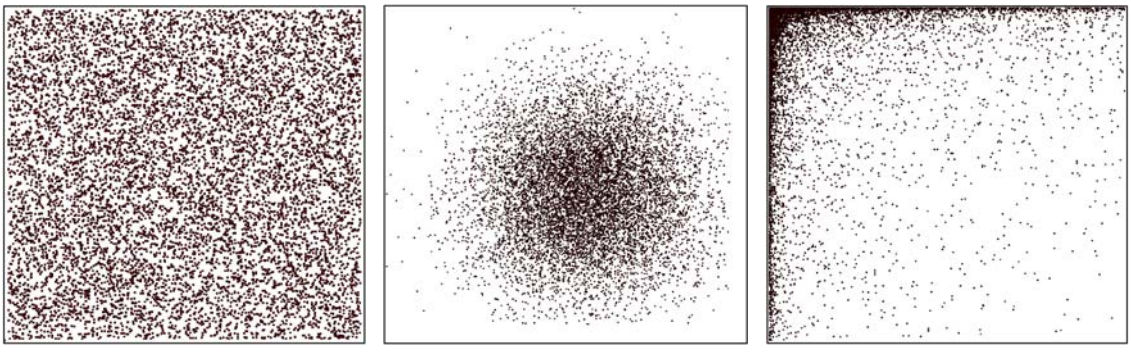
삽입 실험에서는 공간 데이타를 20만개씩 증가시키면서 100만개까지 삽입하는 동안 R-Tree, iQR-Tree, MR-Tree의 평균 노드 접근 횟수를 비교하였다. 그림 12는 삽입을 수행한 성능 결과를 보여준다.

그림 12와 같이 삽입에서는 MR-Tree의 성능이 iQR-Tree보다 다소 저하되는 것으로 나타났다. 이는 iQR-Tree가 MR-Tree보다 트리 높이가 낮고 공간 객체를 여러 R-Tree에 분산 저장함으로써 삽입 비용을 줄인 반면에 MR-Tree는 매핑 트리 갱신 비용이 필요하기 때문이다. 전체적으로 MR-Tree는 iQR-Tree보다 평균 10%의 성능 저하를 보였고, R-Tree와는 거의 비슷한 성능을 보였다.

삭제 실험에서는 전체 데이타 공간의 1%가 되는 질의 윈도우와 서로 겹치는 공간 객체를 모두 삭제하는 동안 R-Tree, iQR-Tree, MR-Tree의 평균 노드 접근 횟수를 비교하였다.

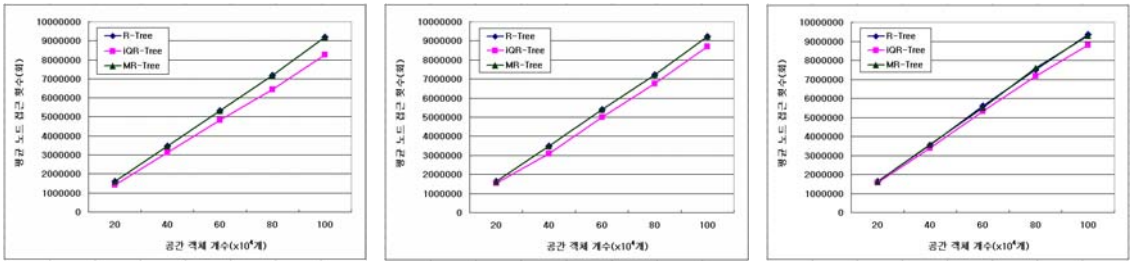
그림 13은 삭제를 수행한 실험 결과를 보여준다.

그림 13과 같이 삭제에서는 MR-Tree의 성능이 iQR-Tree보다 다소 저하되는 것으로 나타났다. 이는 iQR-Tree가 공간 객체를 여러 R-Tree에 분산 저장함으로써 삭제 비용을 줄인 반면에 MR-Tree



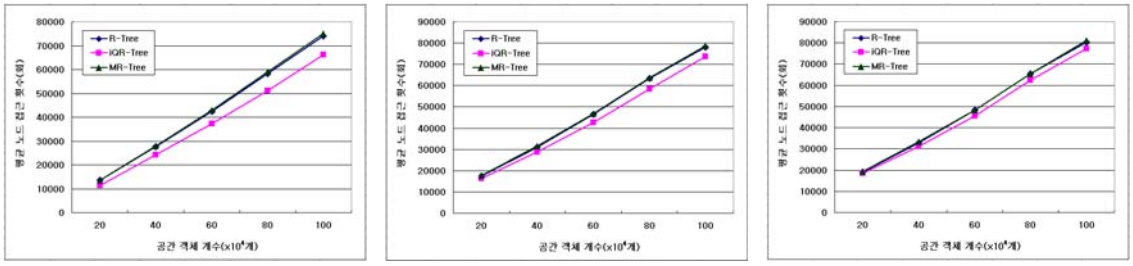
(a) Uniform (b) Gauss (c) Skew

그림 11. 실험에 사용된 공간 데이타



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

그림 12. 삽입 실험 결과



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

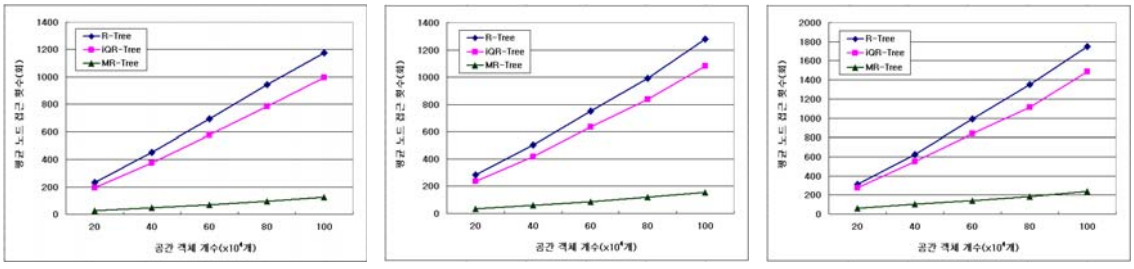
그림 13. 삭제 실험 결과

는 매핑 트리갱신 비용이 추가되었기 때문이다. 전체적으로 MR-Tree는 iQR-Tree보다 평균 10%의 성능 저하를 보였고, R-Tree와는 거의 비슷한 성능을 보였다.

검색 실험에서는 점 질의와 범위 질의를 수행하였다. 범위 질의에서는 전체 데이타 공간의 1%~5%가 되는 질의 사각형으로 질의를 수행하였다. 검색 성능 비교를 위해 1,000번의 검색 질의를 수행하는 동안 R-Tree, iQR-Tree, MR-Tree의 평균 노드 접근 횟수를 비교하였다.

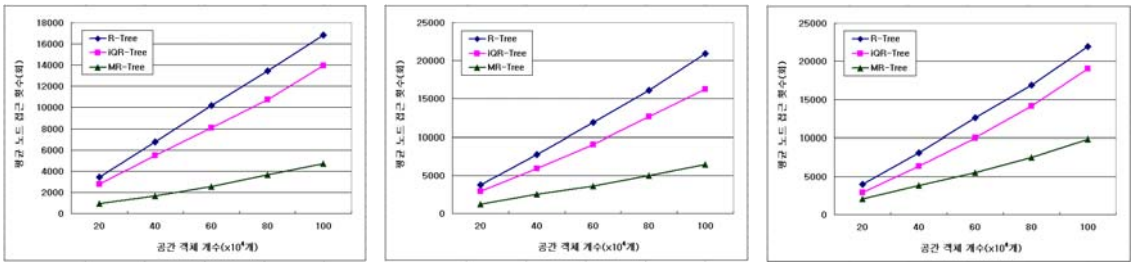
그림 14는 점 질의를 수행한 성능 결과를 보여준다.

그림 14와 같이 점 질의에서는 모든 경우에 MR-Tree의 성능이 가장 우수한 것으로 나타났다. 이는 MR-Tree인 경우에 매핑 트리를 통해 R-Tree 리프 노드를 직접 접근함으로써 트리 순회가 필요없어서 발생하는 노드 접근을 줄일 수 있었기 때문이다. Uniform 분포에서 MR-Tree가 iQR-Tree보다 평균 700%, R-Tree보다 평균 860%의 성능 향상을 보였다. 그리고, Gauss 분포에서는 MR-Tree가 iQR-Tree보다 평균 630%, R-Tree보다 평균 740%의 성능 향상을 보였다. 마지막으로 Skew 분포에서는 MR-Tree가 iQR-Tree보다 평균 460%, R-Tree보



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

그림 14. 점 질의 실험 결과



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

그림 15. 범위 질의 실험 결과

다 평균 560%의 성능 향상을 보였다. MR-Tree는 데이타 밀집도가 높은 Gauss, Skew 분포 보다는 Uniform 분포에서 검색 성능이 더 좋게 나타났다.

그림 15는 동일한 공간 데이타에 대해서 범위 질의를 수행한 결과를 보여준다.

그림 15와 같이 범위 질의는 점 질의보다 검색 공간이 크기 때문에 접근하는 노드가 증가하였으나 모든 경우에서 MR-Tree의 성능이 가장 우수한 것으로 나타났다. Uniform 분포에서 MR-Tree가 iQR-Tree보다 평균 67%, R-Tree보다 평균 73%의 성능 향상을 보였다. 그리고 Gauss 분포에서는 MR-Tree가 iQR-Tree보다 평균 59%, R-Tree보다 평균 68%의 성능 향상을 보였다. 마지막으로 Skew 분포에서는 MR-Tree가 iQR-Tree보다 평균 41%, R-Tree보다 평균 53%의 성능 향상을 보였다.

이처럼 성능 평가에서 MR-Tree는 다양한 공간 분포를 갖는 대용량 공간 데이타에 대해 삽입 성능과 갱신 성능에서는 R-Tree와는 비슷한 성능을 보여주었고, iQR-Tree 보다는 약 10%의 성능 저하를 보여주었다. 그러나 검색 성능에서는 MR-Tree는 점 질의에서 iQR-Tree와 R-Tree 보다 각각 460%~700%와 560%~860%의 성능 향상을 보여주었고, 범위 질의에서 iQR-Tree와 R-Tree 보다 각각 41%~67%와 53%~73%의 성능 향상을 보여주었다.

5. 결 론

본 논문에서는 R-Tree에서 검색시 불필요한 노드 접근을 줄이기 위해 리프 노드를 직접 접근하게 해주는 매핑 트리를 이용하는 인덱스인 MR-Tree를 제안하였다. MR-Tree는 데이타 공간을 차원에 따라 서로 다른 크기를 갖는 여러 파티션으로 분할하고, 매핑 트리를 통해 각 파티션을 R-Tree 리프 노드에 연계시키고, 검색시 매핑 트리를 이용하여 검색 영역에 해당하는 파티션과 연계된 R-Tree 리프 노드를 직접 접근할 수 있도록 한다.

MR-Tree는 트리 순회 없이 매핑 트리를 이용하여 R-Tree의 리프 노드를 직접 접근하기 때문에 대용량 공간 데이타에 대한 효율적인 검색이 가능하다. 또한, 메모리 구조에 적합한 매핑 트리를 메인 메모리에 상주시킴으로써 R-Tree의 검색 시간을 크게 단축시킨다. 특히, MR-Tree는 R-Tree의 큰 변경없이 구현이 가능하고 다양한 R-Tree 변형에도 쉽게 적용할 수 있는 장점이 있다.

실험을 통해 다양한 공간 분포를 갖는 대용량 공간 데이타에서 MR-Tree의 검색 성능이 기존 인덱스보다 우수함을 증명하였다. 그러나 MR-Tree는 공간 객체의 삽입, 삭제시 매핑 트리를 갱신하는 비용이 발생함으로써 삽입, 삭제 성능이 다소 저하되는 문제가 있다. 향후 연구에서는 검색 성능뿐만 아니라 삽입, 삭제 성능도 고려하여 MR-Tree를 확장하는 것이 필요하다.

참 고 문 헌

[1] 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.

[2] 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.

[3] 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.

[4] H. Y. Lin, 2008, “Efficient and Compact Indexing Structure for Processing of Spatial Queries in Line-based Databases”, Data and Knowledge Engineering, vol. 64, no. 1, pp. 365-380.

[5] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, 1984, “The Grid File: An Adaptable, Symmetric Multikey File Structure”, ACM Transactions on Database Systems, vol. 9, no. 1, pp. 38-71.

[6] 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.

[7] 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.

[8] M. Moreau, and W. Osborn, 2008, “An Insertion Strategy for a Two-Dimensional Spatial Access Method”, Proc. of the Canadian Conf. on Computer Science and Software Engineering, pp. 129-131.

[9] 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.

[10] 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.

[11] S. K. Kim, S. H. Baek, D. W. Lee, W. I. Chung, G. B. Kim, and H. Y. Bae, 2009, “Data Source Management Using Weight Table in u-GIS DSMS”, Journal of Korea Spatial Information System Society, vol. 11, no. 2, pp. 27-33.

[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] Y. C. Fu, Z. Y. Hu, W. Guo, and D. R. Zhou, 2003, “QR-Tree: A Hybrid Spatial Index Structure”, Proc. of the 2nd Int. Conf. on Machine Learning and Cybernetics, pp. 459-463.

[14] 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.

[15] 강홍구, 김정준, 신인수, 한기준, 2008, “대용량 공간 데이타의 빠른 검색을 위한 해시 기반 R-Tree”, 2008 공동추계학술대회 논문집, 한국공간정보시스템학회, pp. 82-89.

[16] 박종현, 2008, “Ubiquitous Sensor Network and u-GIS 기술 특집”, 인터넷정보학회지, 제9권, 제1호, pp. 3-3.

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

논문접수 : 2010.07.15
수 정 일 : 2010.10.14
심사완료 : 2010.10.18

강 홍 구
2002년 건국대학교 컴퓨터공학 공학사
2004년 건국대학교 대학원 공학석사
2009년 건국대학교 대학원 공학박사
2009년. 3월~2010년 6월 건국대학교 컴퓨터공학부 강의교수
2010년 7월~현재 한국인터넷진흥원 융합보호 R&D팀 선임연구원
관심분야는 GIS, LBS, 시공간 데이타베이스, 모바일 데이타베이스, Sensor Network

신 인 수

2006년 건국대학교 컴퓨터공학 공학사
2008년 건국대학교 대학원 공학석사
2008년~현재 건국대학교 컴퓨터공학 박사과정
관심분야는 공간 데이타베이스, GIS, LBS, 텔레매틱스

김 정 준

2003년 건국대학교 컴퓨터공학 공학사
2005년 건국대학교 대학원 공학석사
2010년 건국대학교 대학원 공학박사
2010년 9월~현재 건국대학교 컴퓨터공학부 강의교수
관심분야는 공간 데이타베이스, 시공간 데이타베이스, GIS, LBS, 텔레매틱스, 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