

제한된 영역에서의 이동 및 고정 객체를 위한 시공간 분할 트리

윤 종 선* · 박 현 주**

The Separation of Time and Space Tree for Moving or Static Objects in Limited Region

Jong-sun Yoon* · Hyun-ju Park**

Abstract

Many indexing methods were proposed so that process moving object efficiently. Among them, indexing methods like the 3D R-tree treat temporal and spatial domain as the same. Actually, however, both domain had better process separately because of difference in character and unit. Especially in this paper we deal with limited region such as indoor environment since spatial domain is limited but temporal domain is grown. In this paper we present a novel indexing structure, namely STS-tree(Separation of Time and Space tree), based on limited region. STS-tree is a hybrid tree structure which consists of R-tree and one-dimensional TB-tree. The R-tree component indexes static object and spatial information such as topography of the space. The TB-tree component indexes moving object and temporal information.

Keywords : Index, Limited Region, R-tree, TB-tree, Moving Object

논문접수일 : 2004년 10월 20일 논문개재확정일 : 2004년 12월 5일

* 이 연구는 2004학년도 한밭대학교 교내학술연구비의 지원으로 연구되었음.

* 한밭대학교 정보통신공학과 석사과정

** 한밭대학교 정보통신공학과 부교수

1. 서 론

최근 무선 통신의 발달과 휴대폰, PDA, 노트북과 같은 휴대용 단말기의 사용이 일반화되면서 다양한 위치기반 서비스(LBS : Location Based Service)가 이루어지고 있다. 위치기반 서비스란 이동통신망을 기반으로 사람이나 사물의 위치를 정확하게 파악하고 이를 활용하는 응용 시스템 및 서비스를 말한다. GPS를 이용한 내비게이션, 주차정보 시스템과 같은 교통정보 서비스와 친구 찾기 같은 이동통신 어플리케이션 등을 제공하는 것처럼 광범위한 실외에서의 위치기반 서비스는 이미 보편화되었다. 그러나 GPS는 실내에서 사용할 수 없고, 오차범위가 너무 크므로 실내와 같은 제한된 영역에서는 사용할 수 없었다. 하지만 최근 오차범위가 적은 위치 추적 기술이 개발되면서 이를 이용해 놀이동산에서의 미아 찾기, 마트나 백화점에서의 고객의 이동 경로에 따른 상품 진열, 공장에서 이동 장비의 위치 찾기 그리고 전화 중에 고립되거나 실종된 소방관의 위치 찾기 등의 비교적 좁고 한정된 영역에서 그 이용범위가 점차 확대되고 있다. 이런 환경은 기존의 이동 객체 색인 기법에서 고려하고 있는 환경과 여러 가지 차이점이 있다. 예를 들어, 기존의 색인 기법에서 고려하는 환경은 공간의 제약이 없으므로 객체의 이동 범위가 광범위 했으나 본 논문에서 고려하는 환경은 영역에 제한을 둔다. 또한 건물의 도면 등을 이용해 공간에 대한 위상정보를 미리 얻는 것이 가능하다. 따라서 실내와 같은 제한된 영역의 특징을 이용해 보다 효율적으로 정보를 저장하고 빠르게 검색할 수 있는 새로운 색인 기법이 필요하다.

색인 기법은 크게 과거의 정보, 현재의 위치 그리고 현재와 미래의 위치에 대해 색인하는 방법으로 나누어 볼 수 있다[M. F. Mokbel et al.,

2003]. 본 논문에서는 과거의 정보를 다루는 색인 기법을 고려한다. 과거의 정보는 이동 객체의 움직임에 따라 연속적으로 자신의 위치에 대한 정보를 저장해야 하므로 색인의 크기가 계속해서 증가한다. 색인의 크기를 줄이기 위한 두 가지 접근법이 있는데 그 중 하나는 특정 시간에 위치를 추출하는 샘플링기법이고 다른 하나는 이동 객체가 그들의 속도나 방향을 바꿀 때만 변화를 개신하는 방법이다. 그러나 이런 기법은 측정된 위치 사이에서의 객체의 이동에 관해서는 질의응답을 할 수 없게 된다. 전체의 움직임을 얻기 위해 가장 간단한 접근 방법으로 선형 보간법을 사용한다. 본 논문에서 고려하는 환경에서의 객체는 주로 사람이므로 객체의 움직임이 자유롭다. 따라서 속도 등을 측정하는 것 보다는 샘플링 기법이 더 효율적이다.

과거의 정보를 다루는 색인 기법 중에서 이동 객체를 검색하는 방법에는 크게 타임스탬프 질의, 영역 질의, 궤적 질의, 복합 질의 등으로 나눌 수 있다. 이들 질의를 처리하기 위해 여러 기법들이 제안되었으나 모든 종류의 질의를 다 효율적으로 처리하는 기법은 없었다. 예를 들어, HR-tree[M. A. Nascimento et al., 1998]는 타임스탬프 질의는 매우 효율적으로 처리하지만, 영역 질의에는 비효율적이고, 3D R-tree[Y. Theodoridis et al., 1996]는 3차원 영역 질의에서 뛰어난 성능을 보이지만 타임스탬프 질의에서 좋지 않은 성능을 보인다는 문제점이 있으며 궤적을 보존하지 않으므로 궤적 추출비용이 큰 단점을 가진다. TB-tree[D. Pfoser et al., 2000]는 궤적 추출을 효과적으로 하기 위해 단말 노드에 동일한 이동 객체의 궤적을 모아두어 궤적 질의 성능은 우수하지만 공간 인접성을 고려하지 않아 사장 영역이 크고 단말 노드간의 중복이 심해 영역 질의를 처리하기에 적합하지 않다. 따라서 본 논문에서는 특정 형태의 질의가

아닌 모든 종류의 질의에서 고른 성능을 보이는 색인 기법을 제안하고자 한다.

제안하는 STS-tree는 객체를 고정 객체와 이동 객체로 구분하여 각각 다른 색인에 저장하는 방법이다. 고정 객체는 MBR이 고정된 R-tree를 사용하고 이동 객체는 시간차원만을 고려한 TB-tree 사용해 저장한다. 이때, 이동 객체의 공간 영역을 포함하는 R-tree의 MBR 식별자를 TB-tree의 단말노드 엔트리에 유지해서 검색 시 이용한다. 또한 R-tree에 저장되는 고정 객체는 위치의 변화가 거의 없으므로 하드 디스크에 저장하고, 이동 객체는 위치가 연속적으로 변하므로 빠른 삽입과 검색을 위해 메모리에 유지한다.

논문의 구성은 다음과 같다. 2장에서는 기존의 과거정보에 대한 색인기법에 대해서 알아보고 그 중 R-tree와 TB-tree에 대해서 자세히 살펴본다. 그리고 3장에서는 고려하는 환경의 차이로 인한 기존 연구와의 다른점을 설명한 후, 본 연구에서 제안한 STS-tree의 구조에 대해서 살펴본다. 4장에서는 STS-tree의 타임스탬프, 영역, 궤적, 복합 질의 방법에 대해 예를 들어 설명하고, 삽입 알고리즘과 영역 질의 알고리즘을 설명한다. 또, 5장에서는 제안하는 STS-tree와 R-tree 그리고 TB-tree의 삽입과 검색 성능을 비교한 후, 6장에서 제안하는 색인 기법의 효율성에 대해 기술하고 마지막으로 결론을 맺고 향후 연구 방향을 제시한다.

2. 관련 연구

과거의 정보를 다루는 방법[M. F. Mokbel et al., 2003]에서 첫 번째로, 시간의 차원을 고려한 방법에는 R-tree와 TSB-tree의 결합인 RT-tree[X. Xu et al., 1990], 그리고 공간의 차원에 시간의 차원을 더해 시간과 공간 질의에 구분을

없앤 3D R-tree, R-tree와 다른 삽입/분할 알고리즘을 가진 R-tree의 확장인 STR-tree[D. Pfoser et al., 2000] 등이 있다.

두 번째, 과거의 정보를 각 시간마다 분리된 R-tree로 구축하는 overlapping · 다중버전 구조에는 R-tree의 구조에 overlapping B-tree의 아이디어를 합한 MR-tree[X. Xu et al., 1990]가 있다. 또, MR-tree와 유사하게 각 타임스탬프마다 R-tree를 유지하는 HR-tree(Historical R-tree)와 HR-tree에서 엔트리들의 중복을 피하기 위해 설계된 HR+-tree[Y. Tao et al., 2001a]가 있으며, 다중버전 B-tree에 기반한 MV3R-tree [Y. Tao et al., 2001b]가 있다. MV3R-tree에서는 두 개의 tree를 구축하는데 그 중 MVR-tree는 타임스탬프 질의를 처리하고, 3D R-tree는 긴 간격 질의를 처리한다.

마지막으로, 과거 궤적을 저장하고 검색하기 위해 제안된 구조로는 TB-tree(Trajectory-Bundle tree), SETI(Scalable and Efficient Trajectory Index)[V. P. Chakka et al., 2003] 그리고 SEB-tree(Start/End timestamp B-tree) [Z. Song et al., 2003]등이 있다. 이중에서 SETI는 공간적인 특성에 초점을 두었고, STR-tree의 확장인 TB-tree는 잎 노드에서 같은 궤적을 따르는 부분만을 담고 있다는 특징이 있다.

이 중에서 본 논문에서 이용하는 R-tree와 TB-tree 두 가지를 자세히 살펴보겠다.

2.1 R-tree

R-tree는 공간 객체를 색인하기 위해 최소 경계 사각형(MBR : Minimum Bounding Rectangle)을 사용한 B-tree의 확장이다[A. Guttman, 1984]. 색인은 계층적으로 이루어져 있으므로 적은 노드들만을 방문함으로써 공간 검색을 가능하게 한다. 트리에서의 노드는 디스크의 페이

지에 해당한다. 모든 노드는 최소 $m(m \leq M/2)$ 개에서 최대 M 개의 엔트리들을 담고 있고 리프 노드는 데이터 객체에 대한 포인터를 포함하고 있다. 또한 R-tree는 높이 균형 트리이므로 모든 리프 노드들이 같은 레벨에 위치해 있다. 색 인은 완전히 동적인 성질을 가지므로 삽입, 삭제가 탐색과 혼합되어 행해질 수 있고 주기적인 재구성도 필요하지 않다. 또한, 디스크 I/O 수를 최소화 할 수 있도록 디스크 특성에 적합하게 설계되어 있다.

객체의 삽입은 트리를 순회하여 삽입될 리프 노드를 선택하여 이루어지는데, 객체를 포함했을 때 면적확장이 가장 적은 노드가 선택된다. 그러나 선택한 노드에 객체를 삽입할 공간이 없으면 분할이 발생한다. 이때 분할 방법의 선택에 따라 색인의 성능이 좌우된다. 객체의 검색은 노드의 엔트리가 검색 윈도우에 겹쳐지는지를 검사하여 겹쳐진다면 자식 노드를 반복적으로 방문함으로써 이루어진다. 그러나 R-tree는 MBR의 중복이 허용되므로 각 레벨에서 많은 엔트리들이 검색 윈도우와 겹쳐짐으로써 색인 성능을 떨어뜨릴 수 있는 단점이 있다.

2.2 TB-tree(Trajectory-Bundle tree)

TB-tree는 한 리프 노드에 같은 궤적에 속하는 세그먼트들만을 포함해서 궤적을 보존(Trajectory Preservation)하는 접근 방법이다 [D. Pfoser et al., 2000]. 그러므로 저장 공간의 오버헤드가 없으며 궤적 질의에 탁월한 성능을 갖는다.

객체의 삽입은 삽입할 라인 세그먼트가 속한 궤적 중 선행 세그먼트가 삽입된 리프 노드를 찾음으로써 이루어진다. 이때, 루트에서부터 삽입할 라인 세그먼트와 겹쳐지는 자식 노드를 단계적으로 검사하여 새 세그먼트와 연결된 세그

먼트를 포함하는 리프 노드를 선택한다. 노드에 빈공간이 있으면 바로 삽입하고, 노드가 가득 찬 경우에는 분할을 해야 하는데 이것은 전체 궤적 보존의 원칙을 위반하는 것이므로 대신 새로운 리프 노드를 생성해서 객체를 삽입한다. 이때, 가득 차지 않은 부모 노드를 찾을 때까지 트리를 거슬러 올라간 후 새 노드를 삽입하기 위해 가장 오른쪽 경로를 선택한다. 부모 노드에 빈 공간이 있으면 새 리프 노드를 삽입하고 부모 노드가 모두 가득 차 있으면 레벨 1에 새로운 리프 노드를 갖는 새로운 노드를 생성한다. 그러므로 TB-tree는 왼쪽에서 오른쪽으로 성장한다. 이렇게 생성된 리프 노드를 이전 리프 노드가 가리키도록 한다. 따라서 궤적에 관한 질의를 처리하기 위해서는 리프 노드에 연결된 링크드 리스트의 포인터를 따라감으로써 구할 수 있으므로 궤적 추출이 효율적이다. 그러나 TB-tree는 궤적을 보존하는 정책으로 인해 근접성과 같은 공간적인 특성을 고려하지 않는다는 문제가 있으며, 단말 노드의 MBR이 커지고 객체가 많을수록 MBR의 겹침 현상이 두드러져 질의 성능이 나빠지는 단점을 가진다.

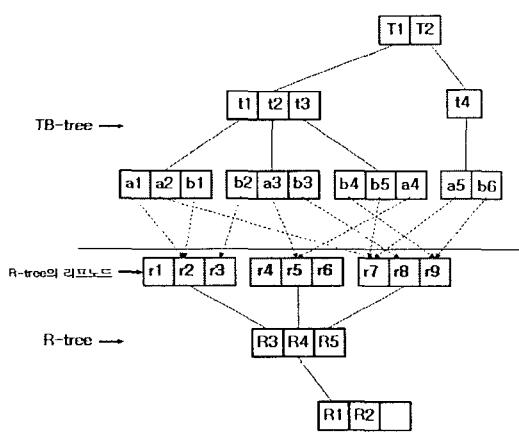
3. STS-tree 구조

본 연구에서는 벌딩 또는 놀이동산과 같은 제한된 영역을 대상으로 하므로 기존 연구와는 몇 가지 차이점이 있다.

우선, 제한된 공간에서의 객체를 살펴보면 방, 엘리베이터, 정수기 등과 같이 위치가 변하지 않는 고정 객체와 사람이나 이동장비와 같은 이동 객체로 나누어 볼 수 있다. 고정 객체에 비해 이동 객체는 지속적인 생신이 이루어져야 하므로 이를 객체를 하나의 색인에 저장하는 것은 비효율적이다. 따라서 이동 객체와 고정 객체를 나누어서 각각의 색인에 저장하는 기법을

제안한다. 고정된 객체는 고정된 공간 객체를 저장하기에 가장 적합한 R-tree를 이용한다. 가능하다면 R-tree는 공간의 평면도 등 위상정보를 이용해 미리 구성한다. 반면, 이동 객체는 과거의 정보를 저장하기 위해 수정된 TB-tree를 사용한다.

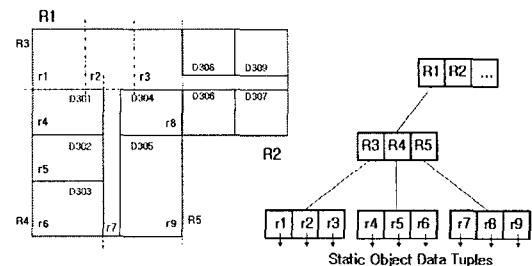
다음으로, 논문에서 고려하는 환경은 공간이 제한되어 있으므로 기존의 방법에서와는 달리 공간(x, y)축과 시간(t)축이 모두 성장하는 것이 아니라 x, y축은 한정된 반면 t축은 지속적으로 성장한다. 이와 같이 각 차원의 특징이 서로 다르므로 기존의 방법에서처럼 시간차원과 공간차원을 함께 유지하는 것보다는 분리하는 것이 더 효율적이다. 그리고 공간정보는 변하지 않으므로 매번 공간에 대한 MBR을 수정하는 것은 효율적이지 않다. 따라서 TB-tree는 시간차원만을 고려하고 공간에 대해서는 R-tree의 MBR#만을 유지함으로써 시간과 공간을 분리한다. 이 때, TB-tree의 MBR은 하위 노드들의 lifetime을 포함하도록 구성한다. <그림 1>은 STS-tree의 전체 구조이다. 그림에서처럼 STS-tree는 R-tree와 TB-tree로 이루어져 있고, TB-tree의 리프 노드 엔트리는 R-tree의 리프 노드 엔트리의 MBR 정보를 가진다.



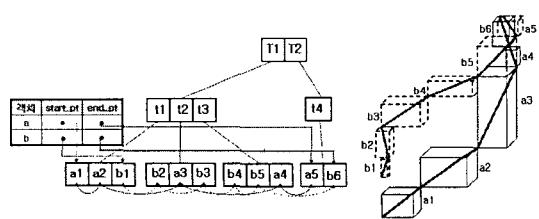
〈그림 1〉 STS-tree 구조

또한, TB-tree가 1차원(시간차원)으로 유지되어 있으므로 기존의 TB-tree에서처럼 한 노드에 하나의 궤적에 관계된 객체만을 유지하게 되면 새로운 객체가 들어올 때마다 노드를 생성해야 하고, 따라서 MBR의 겹침 현상이 발생한다. 그러므로 MBR의 겹침을 줄이고 시간의 순서대로 객체를 유지하기 위해 하나의 노드에 여러 가지 다른 객체들을 담을 수 있도록 수정한다.

R-tree에 유지되는 고정 객체들은 거의 수정되지 않고 그 형태를 유지하는 반면, TB-tree에 유지되는 이동 객체들은 그들의 위치가 연속적으로 변한다. 연속적으로 이동하는 객체의 위치 정보를 삽입하기 위해, 현재 이동 객체의 위치 데이터는 객체의 다음 위치 데이터가 들어오기 전에 데이터베이스에 저장되어야 한다. 그러나 기존의 방법에서처럼 하드디스크에 저장한다면 디스크의 입출력 시간이 오래 걸리므로 적합하지 않다. 따라서 본 연구에서는 TB-tree를 메모리에, R-tree는 하드디스크에 저장하고 TB-tree를 주기적으로 디스크에 백업한다. 그리고 R-tree의 내부노드 구조는 TB-tree의 삽입과 검색 시 계속 검색되어야 하므로 메모리에 유지한다.



〈그림 2〉 고정 객체를 저장하기 위한 R-tree



〈그림 3〉 이동 객체를 유지하기 위한 TB-tree

<그림 2>와 <그림 3>은 STS-tree의 세부 구조인 고정 객체를 위한 R-tree와 이동 객체를 위한 TB-tree의 구조를 나타낸다. <그림 3>에서 와 같이 TB-tree는 추가적으로 객체별 시작 포인터와 마지막 포인터를 유지하는 테이블을 갖는다[심춘보 외, 2004]. 시작 포인터와 마지막 포인터는 객체의 궤적정보 검색을 위해 사용된다.

각 트리의 노드 구조를 살펴보면, R-tree의 단말 노드와 비 단말 노드는 단말 노드 엔트리에 영역 식별자인 MBR#가 추가된다는 것을 제외하고는 기준과 같고, TB-tree의 비단말 노드와 단말 노드 엔트리에 대해서는 아래 <표 1>과 <표 2>에 나타나 있다.

<표 1> TB-tree의 비단말 노드 엔트리

구성 요소	설명
MBR	시간차원만을 고려한 구간 [t, t')
child	하위 노드를 가리키는 포인터

<표 2> TB-tree의 단말 노드 엔트리

구성 요소	설명
객체 ID	객체 식별자
MBR	시공간에 대한 MBR (x, y, t), (x', y', t+1)
MBR#	객체의 공간정보를 나타내는 R-tree의 MBR 식별자
pre	궤적정보 유지를 위해 객체의 이전 위치 정보를 가리키는 포인터
next	궤적정보 유지를 위해 객체의 다음 위치 정보를 가리키는 포인터

4. STS-tree 알고리즘

4.1 삽입

<표 3>은 삽입할 객체의 속성을 나타내고 있다. 객체의 삽입은 객체가 고정 객체인지 이동 객체인지에 따라 구별된다. 사실상 고정 객체는 말 그대로 위치가 고정되어서 거의 변하지 않으

므로 삽입 및 삭제가 거의 이루어지지 않는다. 고정 객체라면 공간의 위상정보를 이용해 미리 구축된 R-tree에 삽입된다. 그러나 R-tree의 리프 노드가 가득차서 분할이 일어난다면 R-tree의 리프 노드들을 갖고 있는 TB-tree도 모두 생신되어야 하므로 R-tree의 리프 레벨은 노드당 최소/최대 객체의 제한을 두지 않는다. 삽입할 객체가 이동 객체라면 우선 searchRtree 알고리즘을 사용해서 R-tree를 검색한 후, 삽입할 이동 객체의 공간 속성이 속하는 R-tree 리프 노드를 리턴 받는다. searchRtree 알고리즘은 기존 R-tree의 검색 알고리즘과 같다. searchRtree를 통해 리턴 받은 리프 노드의 MBR#를 삽입할 객체의 MBR#로 설정한다. 객체의 삽입은 시간 순으로 이루어지므로 가장 마지막에 삽입된 객체가 속한 노드에 공간이 있다면 그 노드에 삽입되고 그렇지 않다면 TB-tree의 삽입방법에 따라 새로운 노드를 생성한 후 삽입한다. 객체별 시작과 끝 포인터를 담고 있는 테이블에서 삽입할 객체에 해당하는 마지막 객체(end_pt)가 가리키는 객체)를 반환받아 링크를 설정한다.

<표 3> 객체의 속성

구성 요소	설명
객체 ID	객체 식별자
MBR	시공간에 대한 MBR
type	고정 객체인지 이동 객체인지 표시

Algorithm STS_Insert(E)

```

N = searchRtree(E, R)
/*R : R-tree의 Root
N : 객체 E의 공간 속성에 해당하는 R-tree의 노드*/
if(E.type == STATIC)
    /*#define STATIC 1*/
    Insert E to N
else {
    NE = createNewEntry(E)
}

```

```

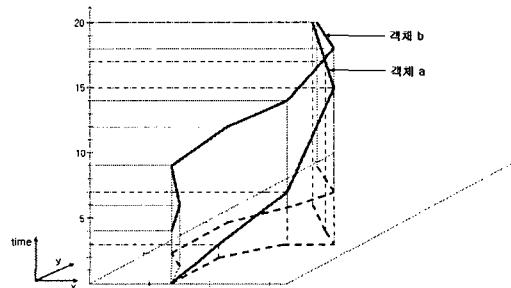
/*createNewEntry : 새로운 TB-tree의 단말 노드
   엔트리 생성한 후 E의 속성을 새로운 엔트리의
   속성으로 설정*/
N.MBR# = N.MBR#
LN = getLastNode()
/*LN : 마지막으로 객체가 삽입된 노드*/
if(LN.entryNum == M){
    /*M : 노드에 삽입될 수 있는 최대 엔트리 수*/
    NN = createNewNode()
    /*새로운 단말 노드 생성*/
    setLastNode(NN)
    LN = NN
}
Insert NE to LN
if((EP =getEndPtr(E))!= NULL) {
    /*getEndPtr : 객체별 시작과 끝 포인터를 담고
       있는 테이블에서 끝 포인터에 해당하는 객체 리
       턴*/
    NE.pre = EP
    EP.next = NE
}
setEndPtr(NE)
}

```

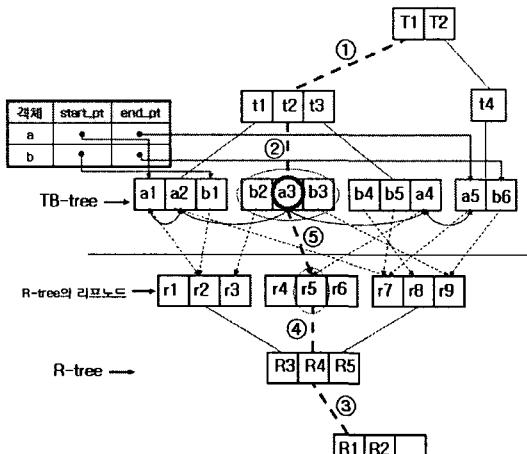
〈그림 4〉 삽입 알고리즘

4.2 검색

질의의 종류를 나누는 방법은 여러 가지가 있을 수 있지만, 여기서는 타임스탬프 질의(timestamp query), 영역 질의(range query), 궤적 질의(trajectory-based query), 그리고 이 두 가지를 결합한 복합 질의(combined query) 이렇게 네 가지로 나누어서 생각하겠다[전봉기 외, 2002]. 타임스탬프 질의는 특정 시간에 주어진 공간 원도우에 속하는 이동 객체들을 검색하는 질의이며, 영역 질의는 주어진 시간 동안에 공간 원도우에 속하는 이동 객체들을 검색하는 3차원 시공간 질의이다. 또, 궤적 질의는 특정한 이동 객체의 이동 경로를 검색하는 질의이다. <그림 5>에서 <그림 7>을 이용해 네 가지 질의 방법이 각각 어떻게 처리되는지 예를 들어 살펴보자.



〈그림 5〉 객체 a와 b의 궤적



〈그림 6〉 STS-tree의 예

R1	R2	엔트리	MBR
r1, r2, r3, r4, r5, r6	D308, D309	T1, T2, T3, T4	[0, 18), [17, 20), [0, 7), [6, 15), [12, 18), [17, 20)
r3, r4, r5, r6	D304, D306, D307		
r4, r5	D305		
r6	r7, r8, r9		
	R5		

〈비단말 노드 엔트리의 MBR 정보〉

〈그림 7〉 공간정보와 비단말 노드 엔트리의 MBR 정보

영역 질의는 TB-tree의 검색을 통해 시간의 조건에 알맞은 객체를 찾은 후, R-tree를 검색해 공간의 조건에 맞는 리프 노드의 MBR#를 찾는다. 그리고 TB-tree의 검색 결과로 나온 객체들의 MBR#를 비교해서 조건에 맞는 객체를 찾을 수 있다. 예를 들면, “7분에서 11분 동안에 D302호에 머물렀던 객체를 검색하라.”가

L인 TB-tree의 T1과 t2를 거쳐 7분에서 11분 사이에 존재하는 객체인 b2, a3, b3을 구한 후 (①~②), 아랫부분인 R-tree를 통해 D302호가 속한 리프 노드의 MBR#인 r5를 구한다(③~④). 그리고 b2, a3, b3 중 MBR#가 r5인 객체 a3을 구한다(⑤). 또 다른 영역 질의인 시간 축에 대한 영역 질의를 고려해 보자. 예를 들어, “16분에서 17분 사이에 객체 a는 어디에 있나?”와 같은 질의가 있다. 이 질의는 역시 TB-tree를 통해서 16분에서 17분 사이에 존재하는 객체 b5, a4 중에서 a4의 MBR#인 r5를 구한다. 여기서 r5는 D302라는 것을 알 수 있으므로 시간 t3동안 객체 a는 ‘D302호’에 있다는 것을 쉽게 알 수 있다.

TB-tree의 리프 노드는 시간의 순서로 이루어져 있으므로 타임스탬프 질의는 간단히 처리될 수 있다. 타임스탬프 질의는 영역 질의의 시간의 차원이 0인 특별한 경우일 뿐이므로 같은 방법으로 검색한다. 예로, “15분에 복도에 있는 객체는?”이라는 질의가 있다. 우선 TB-tree에서 15분은 t3에 속하므로 T1과 t3을 통해 15분에 있는 객체 a4와 b5를 찾는다. 또 복도의 MBR#는 r7이다. 객체 a4의 MBR 식별자는 r5이고, b5는 r7이므로 15분에 복도에 있는 객체는 ‘b’라는 것을 찾아낼 수 있다.

케적 질의는 단순히 객체별 포인터 테이블에서 시작 객체를 구한 후, 포인터를 따라가면 구할 수 있다. 예를 들어, “객체 b는 어디서 어디로 이동하였는가?”라는 질의는 테이블에서 객체 b의 시작 포인터를 따라 b1을 찾고 next 포인터를 따라서 b2-b3-b4-b5-b6을 구할 수 있다. 따라서 객체 b는 ‘현관-현관-D304호-D305호-복도-D305호’로 이동하였다는 것을 알 수 있다.

마지막으로 복합 질의는 예를 들어 “7분에서 11분 사이에 D302호에 머물렀던 객체의 이전 5

분간의 이동 경로를 검색하라.”가 있을 수 있다. 이 질의에서 “7분에서 11분 사이에 D302호에 머물렀던 객체”를 찾는 질의를 내부 질의영역 (inner range)이라고 하고, “이전 5분간의 이동 경로”는 외부 질의영역(outer range)이라고 한다. 내부 질의영역에 속하는 객체를 위의 영역 질의의 방법에 따라 객체의 아이디인 a3을 저장하고, pre 포인터를 이용해 외부 질의영역에 속하는 a2-a1을 찾음으로써 ‘복도-현관’이라고 응답할 수 있다.

검색 알고리즘 중 케적 질의는 비교적 간단하고 복합 질의는 케적 질의와 영역 질의의 결합이므로 여기서는 시간과 공간 영역에 대한 영역 질의의 알고리즘만 기술한다. 알고리즘은 <그림 8>과 같다. 우선 TB-tree를 검색하는 findEntry 알고리즘을 통해서 주어진 시간 영역에 속한 엔트리들을 리턴 받는다. 다음으로 R-tree의 검색 알고리즘을 수정한 findNode를 통해 공간 영역이 속한 MBR의 번호를 리턴 받는다. 그리고 만약 검색하는 공간의 영역이 MBR과 영역이 동일한 경우에는 단순히 리턴 받은 엔트리들의 MBR#만을 비교함으로써 원하는 값을 얻을 수 있지만, 검색 영역이 MBR보다 작은 경우에는 다시 한 번 영역을 비교함으로써 원하는 값을 얻을 수 있다. findEntry 알고리즘은 <그림 9>, 그리고 findNode 알고리즘은 <그림 10>과 같다.

```

Algorithm RangeSearch(rangeS, rangeT)
E = findEntry(TB, rangeT)
/*TB : TB-tree의 Root
E : 시간 조건을 만족하는 엔트리들의 집합*/
num = findNode(R, rangeS) ;
/*R : R-tree의 Root
num : 공간 조건을 만족하는 MBR 번호*/
for(i = 0, j = 0 ; i < E.length ; i++)
  if(E[i].MBR# == num)
    ANSI[j++] = E[i]
  if(isEqual == true)
    /*isEqual : 전역 변수. MBR과 검색 영역이 같으*/

```

```

    면 true 검색영역이 더 작으면 false */
    return ANS1
  for(k = 0, c = 0 ; k < j ; k++)
    if(contain(rangeS, S[k]) == true)
      /*contain(a, b) a에 b가 포함되면 true*/
      ANS2[c++] = ANS1[k]
  return ANS2
}

```

〈그림 8〉 영역 질의 알고리즘

```

Algorithm findEntry(N, T)
if(N.level != LEAF)
  /*#define LEAF 0*/
  for(i=0 ; i < N.entryNum ; i++)
    if(overlap(E[i], T) == true)
      /*E : 노드 N에 속하는 엔트리*/
      findEntry(E[i] → child, T)
  else
    for(j=0, c=0 ; j < N.entryNum ; j++) {
      if(contain(T, E[j]) == true)
        ANS[c++] = E[j]
      else if(E[j].t+1 >= T.t'+1)
        return ANS
    }
}

```

〈그림 9〉 시간 영역에 대해 TB-tree를 검색하는 알고리즘

```

Algorithm findNode(N, S)
if(N == LEAF)
  for(i=0 ; i < N.entryNum ; i++) {
    if(equal(S, E[i]) == true) {
      isEqual=true
      return N.MBR#
    }
    else if(contain(E[i], S) == true) {
      isEqual=false
      return N.MBR#
    }
  }
  else
    for(j=0 ; j < N.entryNum ; j++) {
      if(contain(S, E[j]) == true)
        findNode(E[j] → child, S)
    }
}

```

〈그림 10〉 공간 영역에 대해 R-tree를 검색하는 알고리즘

5. 성능 평가

이 장에서는 STS-tree와 STS-tree에서 이용

한 R-tree와 TB-tree의 삽입과 영역 질의, 궤적 질의 그리고 복합 질의를 수행한 결과를 비교한다. 실험은 윈도우 XP에서 CPU Pentium IV 2.4GHz, 512MB의 메인 메모리를 장착한 PC에서 수행했다. 삽입 데이터는 GSTD(Generate Spatio Temporal Data)[Y. Theodoridis et al., 1996]를 사용해 생성했다. 이동 객체의 초기 분포는 가우시안 분포를 사용했고, 생성된 데이터 중 공간 좌표가 음수가 아닌 양수 값으로 생성된 값만을 골라서 평가했다. 모든 색인은 디스크에 저장하지 않고 메모리에 유지해서 삽입과 검색 성능을 평가했다. 또한 R-tree에 시간 도메인을 추가했다. 각 트리의 팬아웃(fanout)을 살펴보면 R-tree는 31, TB-tree는 35이고, STS-tree의 단말 노드는 25, 그리고 중간 노드는 83이다. STS-tree의 수정된 TB-tree의 중간 노드는 공간 도메인을 유지하지 않아도 되는 반면, 단말 노드는 시간과 공간 도메인을 모두 유지해야 하므로 중간 노드와 단말 노드의 팬아웃을 다르게 설정했다.

5.1 삽 입

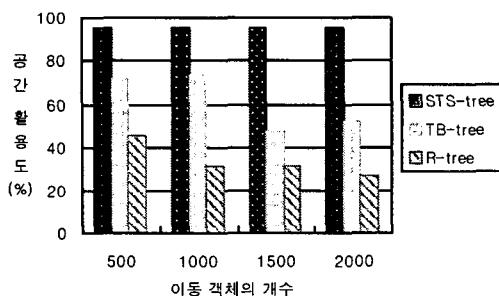
삽입은 <표 4>와 같이 이동 객체의 개수를 500개에서 2,000개까지의 네 가지의 데이터 집합을 이용해, 공간 활용도와 삽입 시간으로 나누어 평가했다. GSTD에서 일정 시간 간격마다 점 객체를 생성하기 때문에 한 객체에 대해 이전 시간의 점 객체를 이용해 선분을 만들어 삽입했다.

먼저 <그림 11>에서 공간 활용도를 살펴보면, STS-tree가 가장 우수하고 TB-tree 그리고 R-tree 순위 것을 볼 수 있다. 이것은 STS-tree의 경우, 한 노드가 가득 차지 않을 경우에는 새로운 노드를 생성하지 않으므로 객체의 종류의 개수와 상관없이 공간 활용도가 95%로 우수

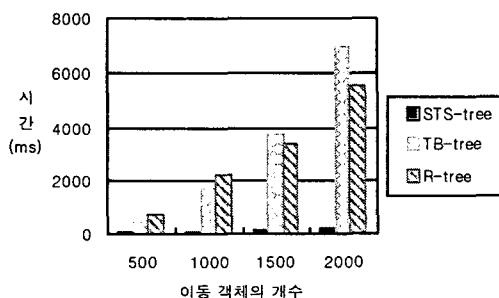
했으나, TB-tree는 하나의 노드에 한 종류의 객체만이 삽입될 수 있으므로 객체의 종류가 많아 질수록 계속해서 새로운 노드를 생성하므로 사장 영역이 증가해 공간 활용도가 좋지 않았다. 또한 R-tree도 계속적인 분할로 인해 객체의 개수가 많아질수록 공간 활용도가 좋지 않았다.

〈표 4〉 삽입 데이터 집합

이동 객체의 개수	객체 당 평균 스냅샷의 수	전체 선분의 개수
500	32	16239
1000	33	32625
1500	33	49079
2000	33	65092



〈그림 11〉 이동 객체의 개수에 따른 공간 활용도의 비교



〈그림 12〉 이동 객체의 개수에 따른 삽입 시간의 비교

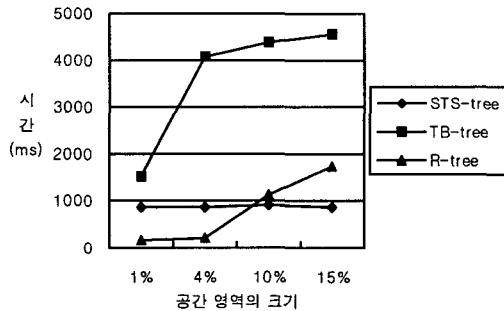
다음으로, 〈그림 12〉는 이동 객체의 삽입 시간을 측정한 결과이다. TB-tree와 R-tree는 객체를 삽입하기 전에 그 객체가 삽입될 알맞은 노드를 선택해야 하므로 삽입 시간이 큰 것을 알 수 있다. 특히, TB-tree는 객체의 종류가 많아질수록 노드의 개수가 증가하므로 탐색 시간이 길어져서 삽입 시간이 크게 증가한다. 그러나 STS-tree는 객체의 종류와 공간, 시간 좌표와 상관없이 이전 객체가 삽입된 노드에 삽입되면 되므로 삽입 전의 탐색이 없다. 따라서 STS-tree의 삽입 성능이 가장 우수한 것으로 평가되었다.

5.2 영역질의

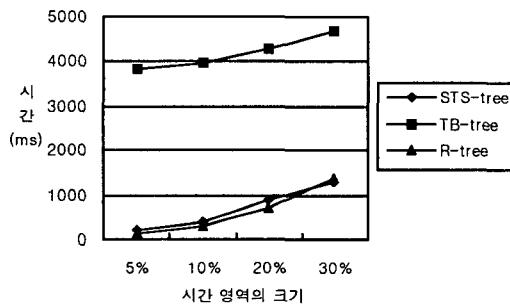
영역질의 성능을 평가하기 위해, 먼저 시간 도메인의 크기를 20%로 고정시키고 공간 도메인의 크기를 1%에서 15%의 네 가지 단계로 나누어 평가했다. 이때, 객체의 개수는 2,000개이고 삽입되는 선분의 개수는 65,092개다. 〈그림 13〉은 영역질의를 1,000번 수행했을 때의 결과이다. R-tree는 루트에서부터 검색 윈도우와 겹쳐지는 노드들만을 따라서 탐색하므로 영역질의 성능이 가장 우수 했으나 사장 영역이 매우 크므로 검색 영역의 크기가 증가할수록 성능이 나빠지는 것을 발견할 수 있었다. STS-tree는 공간 활용도가 우수하므로 색인의 높이가 다른 색인들 보다 낮아서 영역질의 성능이 비교적 우수한 것을 볼 수 있다. 공간 도메인의 크기가 증가해도 STS-tree의 성능이 일정한 것은 STS-tree는 질의 시 시간 도메인의 조건에 알맞은 객체들을 먼저 골라낸 후에 그 객체의 MBR#를 비교하므로 공간 영역의 크기에 영향을 받지 않기 때문이다. TB-tree는 공간 특성을 고려하지 않고 삽입되었으므로 영역질의 성능이 매우 떨어지는 것을 볼 수 있다.

다음으로, 공간 도메인의 크기를 4%로 고정하고 시간 도메인의 크기를 5%에서 30%로 변화시키면서 성능을 측정해 보았다. 이동 객체의 개수는 2,000개이고 영역 질의를 1,000번 수행

하였다. 결과는 <그림 14>와 같이 R-tree와 STS-tree는 성능이 비슷하고, TB-tree는 성능이 떨어지는 것을 볼 수 있다.



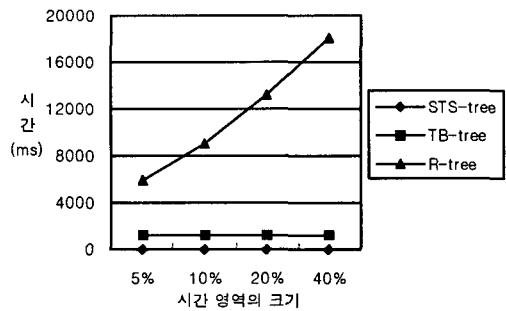
<그림 13> 공간 영역의 크기에 따른 영역질의 시간의 비교



<그림 14> 시간 영역의 크기에 따른 영역 질의 시간의 비교

5.3 궤적 질의

궤적 질의는 하나의 객체에 대해 시간 도메인을 5%에서 40%로 변화시키면서 측정했다. 영역질의와 마찬가지로 객체의 개수는 2,000개이고, 질의는 각각 10,000번씩 수행했다. 결과는 <그림 15>와 같다. R-tree는 궤적을 보존하지 않으므로 성능이 좋지 않은 것을 볼 수 있고, TB-tree와 STS-tree가 성능이 우수한데 STS-tree가 조금 더 우수한 것은 STS-tree가 객체별 궤적 테이블을 유지해서 객체의 시작 포인터를 바로 얻을 수 있는데 반해, TB-tree는 우선 객체가 어디에 있는지 검색해야 하므로 약간의 차이가 나는 것을 볼 수 있다.



<그림 15> 시간 영역의 크기에 따른 궤적 질의 시간의 비교

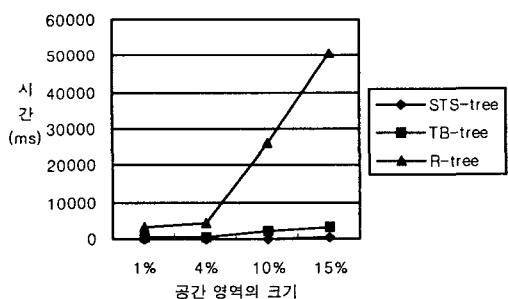
5.4 복합 질의

복합질의는 내부 질의 영역의 시간 도메인과 공간 도메인을 변화시키면서 측정했다. <그림 16>은 내부 질의 영역의 공간 도메인의 크기에 따른 복합질의 성능을 비교한 것이다. 이동 객체의 개수는 2,000개로 동일하고, 내부 질의 영역의 시간 도메인은 20%이며, 외부 질의 영역은 30%이다. 그리고 <그림 17>은 내부 질의 영역의 시간 도메인의 크기에 따른 복합 질의의 성능을 비교했다. 내부 질의 영역의 공간 도메인은 4%이며, 외부 질의 영역은 30%이다. <그림 16>과 <그림 17>은 복합 질의를 100번 수행한 결과이다. 내부 질의 영역의 크기가 커질수록 궤적을 검색해야 하는 객체의 수가 증가하므로 궤적 질의 성능이 떨어지는 R-tree가 가장 영향을 많이 받아 성능이 크게 떨어지는 것을 볼 수 있다. 또한 영역질의와 궤적질의에서 고른 성능을 보인 STS-tree가 복합 질의에서도 가장 좋은 성능을 보인다.

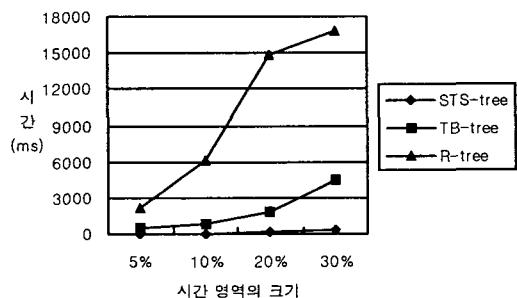
6. STS-tree의 효율성

본 논문에서 고려한 환경은 실내와 같은 제한된 영역이다. 앞에서 살펴본 바와 같이 제한된 영역과 그동안 다른 색인 기법에서 다루었던 일반적인 환경과는 여러 가지 차이점이 있다. 이런

차이점을 고려한 STS-tree는 우선 공간이 제한되어 있다는 특징을 이용해 R-tree의 리프 노드를 고정시킴으로써 분할과 재구성을 하지 않으므로 구현이 용이하고 저장단계가 단순하다. 또한, 고정 객체와 이동 객체를 분리해서 저장함으로써 이동 객체를 검색하는 경우는 TB-tree를, 고정 객체를 검색하는 경우는 R-tree만을 검색함으로써 검색 시간을 단축시킨다. 그리고 시간과 공간을 분리하여 시간정보는 TB-tree에, 공간 정보는 R-tree에 저장함으로써 노드 간의 겹침이 적으며, 영역 질의를 할 때 시간정보를 이용해 TB-tree에서 객체들을 필터링한 후, 그 객체들에 대해서만 공간정보를 검색하므로 이동 객체에 대한 시공간 영역 질의 성능이 향상된다. 또한, R-tree는 고정되어 있고 TB-tree는 시간의 순서로 저장되므로 사장 영역이 적다는 장점이 있다.



〈그림 16〉 내부 질의 영역의 공간 영역의 크기에 따른 복합 질의 시간의 비교



〈그림 17〉 내부 질의 영역의 시간 영역의 크기에 따른 복합 질의 시간의 비교

7. 결론 및 향후 연구

오차 범위가 작은 무선 통신 기법이 발전함에 따라 제한된 영역에서의 객체의 위치를 이용한 다양한 서비스가 이루어지고 있다. 본 논문에서는 이를 보다 빠르고 효율적으로 처리하기 위해 STS-tree를 제안했다. STS-tree는 제한된 영역에서의 객체를 이동하는 객체와 고정된 객체로 분리해서 각각 R-tree와 TB-tree에 저장한다. 또한 제한된 영역에서는 공간의 확장 없이 시간 영역만이 계속 확장된다는 점을 이용해 시간과 공간의 차원을 나누어서 TB-tree는 시간차원만을 이용한 1차원으로 구축하고, R-tree의 공간정보를 그대로 사용함으로써 특성이 다른 두 차원을 분리했다. 따라서 궤적정보를 유지하는 색인 기법에서 영역 질의를 처리할 때 비효율적이었던 것에 비해, 제안된 트리를 사용하면 영역 질의 뿐 아니라, 궤적 질의, 그리고 이 둘을 결합한 복합 질의까지 효율적으로 처리할 수 있다.

향후 연구 과제로는 제안한 STS-tree는 시간과 시공간에 대한 영역 질의는 효율적으로 처리할 수 있는데 반해, “D302호에 객체 a는 몇 시에 있었나?”와 같은 이동 객체의 공간에 대한 영역 질의를 처리하기 적당하지 않으므로 이에 대한 개선이 필요하다. 또한, 객체의 위치를 나타내는 하나의 선분은 한 MBR안에 있다고 가정하고 있으므로 두 개 이상의 MBR에 속해있는 선분을 처리하는 방법에 대한 연구가 필요하다.

참 고 문 현

- [1] 심춘보, 강홍민, 엄정호, 장재우, “위치 기반 서비스에서 이동 객체의 궤적을 위한 TB-트리의 확장”, 정보과학회 2004년 춘계학술대회, 제31권 1호, 2004년 4월, pp. 142-144.
- [2] 전봉기, 임덕성, 홍봉희, “이동체 데이터베이스를 위한 색인 기법”, 데이터베이스연구, 제

- 18권 4호, 2002년 12월, pp. 23-35.
- [3] Guttman, A., "R-Trees : A Dynamic Index Structure for Spatial Searching", Proc. of the ACM International Conference on Management of Data, June 1984, pp. 47-57.
- [4] Pfoser, D., Jensen, C. S. and Theodoridis, Y., "Novel Approaches in Query Processing for Moving Objects", Proc. of the International Conference on Very Large Data Bases, Sept. 2000, pp. 395-406.
- [5] Nascimento, M. A. and Silva, J. R. O., "Towards historical R-trees", Proc. of the ACM Symp. on Applied Computing, Feb. 1998, pp. 235-240.
- [6] Mokbel, M. F., Ghanem, T. M. and Aref, W. G., "Spatio-temporal Access Methods", IEEE Data Engineering Bulletin, Vol. 26, No. 2, 2003, pp. 40-49.
- [7] Chakka, V. P., Everspaugh, A. and Patel, J. M., "Indexing Large Trajectory Data Sets with SETI", Proc. of the 2003 CIDR Conference, Jan. 2003.
- [8] Xu, X., Han, J. and Lu, W., "RT-Tree : An Improved R-Tree Indexing Structure for Temporal Spatial Databases", Proc. of the International Symp. on Spatial Data Handling, July 1990, pp. 1040-1049.
- [9] Tao, Y. and Papadias, D., "Efficient Historical R-trees", Proc. of the International Conference on Scientific and Statistical Database Management, July 2001a, pp. 223-232.
- [10] Tao, Y. and Papadias, D., "MV3R-Tree : A Spatio-Temporal Access Method for Time-stamp and Interval Queries", Proc. of the International Conference on Very Large Data Bases, Sept. 2001b, pp. 431-440.
- [11] Theodoridis, Y., Silva, J. R. O. and Nascimento, M. A., "On the Generation of Spatio-temporal Datasets", SSD, LNCS 1651, 1999, pp. 147-164.
- [12] Theodoridis, Y., Vazirgiannis, M. and Sellis, T., "Spatio-Temporal Indexing for Large Multimedia Applications", Proc. of the IEEE Conference on Multimedia Computing and Systems, June 1996.
- [13] Song, Z. and Roussopoulos, N., "SEB-tree : An Approach to Index Continuously Moving Objects", Mobile Data Management, Jan. 2003, pp. 340-344.

□ 저자소개



윤종선

현재 (주)내셔널그리드에서 응용소프트웨어 개발자로 일하고 있고, 한밭대학교 정보통신대학원에서 공학석사(2005)를 취득했다. 주요 관심분야는 모바일 데이터베이스와 그리드 컴퓨팅 등이 있다.



박현주

현재 한밭대학교 정보통신컴퓨터공학부 부교수로 재직 중이다. 서울시립대학교 전산통계학사(1990), 서울대학교 계산통계학 석사(1992), 박사학위(1997)를 취득하였다. 주요 연구 분야는 모바일 데이터베이스, 파일 시스템, 임베디드 소프트웨어 등이다.