HBR-Tree를 이용한 실시간 모바일 GIS의 개발†

Development of a Real-Time Mobile GIS using the HBR-Tree

이기영*, 윤재관**, 한기준**
Ki-Yamg Lee, Iae-Kwan Yun, Ki-Joon Han

요 약 최근 들어 무선 인터넷이 발전하고, PDA, HPC의 보급이 늘어남에 따라 GIS(Geographic Information System)와 관련된 연구 및 개발이 점차적으로 위치 기반 서비스(LBS: Location Based Service)를 제공하기 위한 실시간 모바일 GIS로 변화해 가고 있다. LBS를 효과적으로 제공하기 위해서는 이동 객체의 동적인 상황을 효과적으로 처리할 수 있는 실시간 GIS 플랫폼과 위치 데이타의 특성을 반영한 위치 인덱스가 필요하다. 위치 데이타는 이전의 GIS에서 사용되는 것과 동일한 데이타 타입(예, 점)이 사용되지만 위치 데이타의 관리는 이전 GIS와는 다른 처리 방식을 사용해야 한다. 이를 위하여 본 논문에서는 대용량의 위치 데이타를 효과적으로 처리할 수 있는 HBR-tree를 이용한 실시간 모바일 GIS의 개발에 대하여 연구하였다.

본 연구에서 개발된 실시간 모바일 GIS는 HBR-tree와 실시간 GIS 플랫폼으로 구성되어 있다. HBR-tree는 R-tree와 공간 해쉬가 결합된 위치 인덱스이다. 그러므로, 위치 데이타가 빈번하게 변경되더라도 갱신 연산은 HBR-tree의 동일한 해쉬 테이블에서 일어나기 때문에 다른 트리 기반 인덱스에 비하여 갱신 연산이 적으며, 검색 연산은 R-tree의 검색 메커니즘을 이용하기 때문에 공간 데이타를 신속하게 검색할 수 있다. 본 논문에서 실시간 GIS 플랫폼은 주기억 장치 데이타베이스 시스템의 기능이 확장된 실시간 GIS 엔진, 공간 및 비공간 데이타를 서버와 클라이언트로 전송하기 위한 미들웨어, 그리고 모바일 장치에서 동작하는 모바일 클라이언트로 구성되어 있다. 특히, 본 논문에서는 실험적 방법을 사용하여 HBR-tree와 실시간 GIS 엔진의 성능 평가 결과에 대해서도 기술하였다.

ABSTRACT Recently, as the growth of the wireless Internet, PDA and HPC, the focus of research and development related with GIS(Geographic Information System) has been changed to the Real-Time Mobile GIS to service LBS. To offer LBS efficiently, there must be the Real-Time GIS platform that can deal with dynamic status of moving objects and a location index which can deal with the characteristics of location data. Location data can use the same data type(e.g., point) of GIS, but the management of location data is very different. Therefore, in this paper, we studied the Real-Time Mobile GIS using the HBR-tree to manage mass of location data efficiently.

The Real-Time Mobile GIS which is developed in this paper consists of the HBR-tree and the Real-Time GIS platform. HBR-tree, we proposed in this paper, is a combined index type of the R-tree and the spatial hash. Although location data are updated frequently, update operations are done within the same hash table in the HBR-tree, so it costs less than other tree-based indexes. Since the HBR-tree uses the same search mechanism of the R-tree, it is possible to search location data quickly. The Real-Time GIS platform consists of a Real-Time GIS engine that is extended from a main memory database system, a middleware which can transfer spatial, aspatial data to clients and receive location data from clients, and a mobile client which operates on the mobile devices. Especially, this paper described the performance evaluation conducted with practical tests of the HBR-tree and the Real-Time GIS engine respectively.

주요어 : LBS, HBR-tree, 위치 인덱스, 실시간 GIS 플랫폼, 실시간 모바일 GIS
Key word : LBS, HBR-tree, Location Index, Real-Time GIS Platform, Real-Time Mobile GIS

† 본 연구는 한국과학재단 목적기초연구(과제번호: R01-2001-000-00540-0) 지원으로 수행되었음.

* 서울보건대학 인터넷정보과 kylee@shjc.ac.kr
** 건국대학교 컴퓨터공학과 {jkyun, kjhan@db.konkuk.ac.kr}

1. 서 론

최근 컴퓨터의 대용량화 및 고성능화에 따라 다양한 형태의 응용 프로그램이 연구 및 개발되고 있다. 이전까지의 이러한 연구 및 개발은 주로 GIS에 대한 데이타베이스의 측면을 기준으로 많이 진행되었지만, 무선 인터넷, PDA, HPC의 발전에 따라 연구의 방향이 점차적으로 LBS를 제공하기 위한 실시간 모바일 GIS의 형태로 발전되고 있다[1][2]. LBS를 효과적으로 제공하기 위해서는 이동 객체의 동적인 상황(예, 개개인의 위치 파악)을 효과적으로 처리할 수 있는 실시간 GIS 플랫폼과 위치 데이타의 특성을 반영한 위치 인덱스가 필요하다[3][4].

기존의 이동 객체의 위치 데이타를 처리하기 위한 인덱스 기법으로는 R-tree 계열의 인덱스 구조를 확장한 방법과 1차원의 해쉬 함수를 2차원의 공간으로 확장한 공간 해쉬 인덱스가 있다. 그러나, 실제 모바일 환경에서 모바일 사용자의 위치를 얻기 위한 위치 측위 기술들은 다수의 모바일 사용자들에 대하여 동일한 시공간 정보, 즉 같은 시간에 같은 공간적인 영역을 갖는 것으로 처리하고 있다[5]. 이러한 동일한 시공간 정보를 갖는 다수의 이동 객체들을 처리하기 위하여 기존의 R-tree 기반의 인덱스 기법을 이용할 경우 이동 객체가 여러 리프 노드에 분산 저장되어 인덱스 크기가 커지게 된다는 문제가 발생된다. 이로 인해 삽입 및 삭제 시에 동일한 MBR을 갖는 객체들의 저장하거나 삭제할 위치를 검색하기 위한 디스크 I/O 횟수 증가가 발생하게 된다. 또한, 질의 시 노드의 중복 방문으로 인한 디스크 I/O 횟수의 증가로 많은 비용이 든다는 문제점을 가지고 있다[6].

공간 해쉬 인덱스를 이용하는 경우에는 공간 해쉬 함수를 이용하여 빠른 삽입 및 검색이 가능하고 공간 해쉬 함수에 의해서 충돌이 발생하는 경우에는 인접한 위치 데이타임을 판단할 수 있기 때문에 현재 위치 인덱스를 구성하기에 적합하지만 위치 데이타가 존재하지 않는 빈 공간(Dead Space)이 많이 발생하게 되면 메모리의 낭비가 발생하기 때문에 비효율적이게 된다[7]. 본 논문에서는 이러한 기존의 시공간 및 위치 인덱스들의 단점을 극복하는 효율적인 이동 객체의 현재 위치 인덱스 방법인 HBR-tree를 제시하였고, 또한 HBR-tree의 성능 평가를 위해 R-tree와 공간 해쉬 인덱스와 비교하였다.

LBS에서 주로 사용되는 위치 데이타는 한번 저장이 되면 거의 변경이 없는 기존의 GIS와는 달리 이동 객체의 위치 데이타가 획득될 때마다 변경 연산이 수행되어야 한다[8][9]. 그러나, 이러한 변경 연산에 이전의 디스크 기반 GIS를 사용할 경우에는 복구를 위한 리두 로그를 생성하고 체크포인트가 발생될 때마다 동기화를 시키기 위해서 디스크를 사용해야 하기 때문에 대용량으로 발생되는 위치 데이타의 변경 연산을 적용하기에는 부족한 점이 많다. 이를 해결하기 위해서 본 논문에서는 HBR-tree를 이용한 실시간 모바일 GIS의 개발하였고, 또한 기존의 디스크 기반 시스템(PostgreSQL, ORACLE, ZEUS)과 성능을 비교해 우수성을 입증하였다.

본 논문의 구성은 다음과 같다. 제 2 장에서는 관련 연구로써 공간 해쉬 인덱스와 R-tree 인덱스를 분석하고 이전에 개발된 실시간 데이타베이스 시스템에 대하여 설명한다. 제 3 장에서는 본 논문에서 제시하는 HBR-tree에 대해 언급하고, 제 4 장에서는 시스템의 개발에 대하여 기술한다. 제 5 장에서는 성능 평가 결과를 보여주고, 마지막으로 제 6 장에서는 결론 및 향후 연구 과제에 대해 언급한다.

2. 관련 연구

본 장에서는 기존의 공간 인덱스인 공간 해쉬 인덱스와 R-tree 인덱스에 대해서 분석하고, 본 연구에서 확장한 실시간 데이타베이스 시스템에 대하여 설명한다.

2.1 기존 공간 인덱스의 분석

공간 해쉬 인덱스는 이동 객체의 위치를 키 값으로 해쉬하는 방법을 사용하여 비교적 간단한 과정으로 인덱스가 가능하다는 장점이 있다[10]. 그러나, 데이타 집합이 비정규 분포(특정 지역에 밀집)일 경우 특정 영역 셀에 지속적인 오버플로우가 발생하여 인덱스의 성능이 저하되는 문제가 발생한다. 이동 객체는 주기적으로 이동하므로 밀집 지역을 빈번하게 발생시키기 때문에 공간 해쉬 인덱스의 성능 저하가 발생된다.

R-tree는 높이 균형 트리 구조이기 때문에 데이타의 분포와 관계없이 일반적으로 검색에 우수한 성능을 나타낸다. 그러나, 이동 객체의 계속되는 위치 이동으로 인해 인덱스의 변경이 발생하고, 인덱스의 빈번한 변경으로 전체적인 인덱스의 성능 저하가 발생한다. 이와 같은 문제의 원인은 공간 인덱스가 변경이 극히 적은 정적 데이타를 기반으로 설계되었기 때문에 검색

에는 효과적이지만 삽입이나 갱신이 빈번한 위치 데이타의 연산에는 적합하지 않은 구조이기 때문이다.

이동 객체의 위치 변경으로 인한 빈번한 인덱스의 갱신 문제는 공간 해쉬 함수를 이용하여 줄일 수 있다. 이동 객체의 위치 변경은 대부분 기존 공간 객체의 변경으로 볼 수 있다. 기존의 공간 인덱스는 대부분 갱신 연산보다는 검색 연산을 중점적으로 고려하였으나, 이동 객체 데이타베이스에서는 빈번한 갱신 연산을 고려한 인덱스 구조가 필요하다. 일반적으로 트리 기반의 인덱스 구조에서 이동 객체의 위치를 변경하기 위해서는 이동 객체를 삭제 후 재삽입을 하는 연산으로 처리된다. 그러나, 트리 기반의 인덱스에서는 삭제와 재삽입 연산이 인덱스의 병합과 분할을 초래하기 때문에 위치 변경이 빈번한 이동 객체를 위한 인덱스로는 적합하지 않다.

일반적으로 이동 객체에 대한 현재 위치 인덱스를 구성할 경우에 이동 객체의 특성에 따라서 그룹을 지을 수 있다. 즉, 지하철을 타고 가는 경우, 버스를 타고 가는 경우, 고속도로를 가는 경우, 인도를 걸어가는 경우 등 대부분의 경우는 하나의 그룹이 동일한 영역을 이동하다가 새로운 이동 객체가 그룹에 추가되거나 다른 영역으로 이동하는 이동 객체가 그룹에서 이탈하게 된다. 그러므로, 그룹에 대한 MBR을 이용하여 공간 해쉬 테이블을 구성하고 이에 대한 MBR 정보를 R-tree에 추가하게 되면 기존의 공간 해쉬 인덱스의 단점을 해결할 수 있고, 또한 R-tree의 문제점인 인덱스의 빈번한 변경으로 인한 전체적인 인덱스의 성능 저하가 발생하는 문제점도 해결할 수 있다.

2.2 실시간 데이타베이스 시스템

본 연구에서 사용된 주기억 장치 데이타베이스 시스템은 메모리 내에서의 고성능 데이타 저장 및 검색을 지원함으로써 연산들에 대한 예측 가능한 빠른 응답 시간을 보장한다[11]. 여러 프로세스들의 동시 접근이 가능하도록 데이타베이스가 공유 메모리에 상주하기 때문에 여러 개의 프로세스가 한 데이타베이스에 접근할 수 있고, 또한 한 프로세스가 여러 데이타베이스에 접근할 수 있다. 특히, 3단계의 패스워드에 의한 보안과 2가지의 잠금을 지원하며, 유연성 및 확장 가능성이 뛰어난 특징이 있다.

실시간 데이타베이스 시스템의 실시간 스케쥴링은 운영체제의 우선 순위 선점 스케쥴링에 기반하여 프로세스 차원에서 관리되며, 프로세스 메모리 잠금과 공유 메모리 잠금을 이용해 동시성 제어를 하고 있다. 또한, 실시간 데이타베이스 시스템는 B-Tree 인덱스와 해쉬 인덱스를 제공하고 있고, 실시간 데이타베이스 시스템에서의 데이타베이스는 다중 사용자가 데이타베이스를 접근하는 모드인 공유(shared) 모드와 한 프로세스만 데이타베이스 관리자 암호로 접근할 수 있는 배타(exclusive) 모드를 제공한다.

그리고, 실시간 데이타베이스 시스템에서는 시스템 정보가 공유 메모리에서 관리되기 때문에 여러 프로세스가 하나의 데이타베이스를 효과적으로 사용할 수 있다. 또한, 레코드나 열들 중의 하나의 값 또는 입력 영역 내에 값의 변화가 있으면 발생하는 이벤트를 지원하며, 메모리에 있는 데이타를 화일로 출력하는 언로드, 화일에 있는 데이타를 메모리에 적재하는 로드 유틸리티를 제공하고 있다. 본 논문에서는 실시간 데이타베이스 시스템을 확장하여 LBS에서 사용되는 위치 데이타 및 공간 데이타를 효율적으로 관리할 수 있도록 하였다.

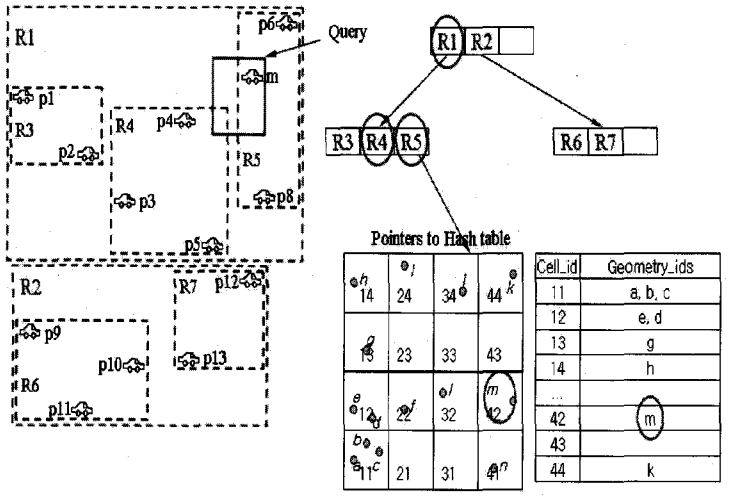
3. HBR-tree 인덱스

본 장에서는 위치 기반 서비스에서 시간의 흐름에 따라 연속적으로 변화하면서 발생되는 이동 객체의 위치 데이타에 대한 현재 위치 인덱스 방법인 HBR-tree에 대하여 설명한다.

3.1 HBR-tree의 구조

HBR-tree는 기존의 R-tree의 장점과 공간 해쉬 인덱스의 장점을 동시에 수용하고 있다. HBR-tree는 R-tree와 공간 해쉬 인덱스에서 확장된 개념으로 HBR-tree의 구조는 <그림 1>과 같다.

HBR-tree에서 영역에 대한 질의가 들어오면 R-tree에서 해당하는 MBR을 검색하게 된다. 그림 1에서와 같이 R1의 전체 영역 중 질의에 포함되는 R4와 R5를 검색한 후 공간 해쉬 테이블에서 m을 구하게 된다. 일반적인 위치 데이타의 갱신은 공간 해쉬 테이블에서 일어나고, 이에 대한 전체적인 MBR의 검색은 R-tree에서 일어나게 되므로 빈번한 이동 객체의 변경에도 빠른 처리가 가능하게 된다.



〈그림 1〉 HBR-tree의 구조

3.2 공간 해쉬 테이블의 생성

HBR-tree에서는 전체 공간 도메인을 공간 해쉬 함수를 통해 작은 범위로 축소시키게 된다. 예를 들면, 서울의 지도를 가로, 세로의 일정한 영역으로 나누었을 때 각각의 격자는 공간 해쉬 테이블의 하나의 셀(cell)에 매핑된다. 즉, d-차원을 구성하는 각 i축(단, $1 \le i \le d$)을 m_i 구간으로 나누어 전체 공간 도메인을 $k = m_1 \times m_2 \times \ldots \times m_d$개 셀들의 집합으로 구성되는 공간으로 변환한다. d-차원 공간의 임의의 좌표는 $(v_1, v_2, \ldots, v_d)$와 같이 표현된다. 해쉬 테이블에서는 각 축의 값에 대한 공간 해쉬 함수 값들인 $(f_1(v_1), f_2(v_2), \ldots, f_d(v_d))$을 해당 좌표의 셀 주소(cell address)로서 사용한다. HBR-tree에서 사용하는 i축을 위한 공간 해쉬 함수 $f_i(v_i)$는 다음과 같다 (단, $1 \le i \le d$).

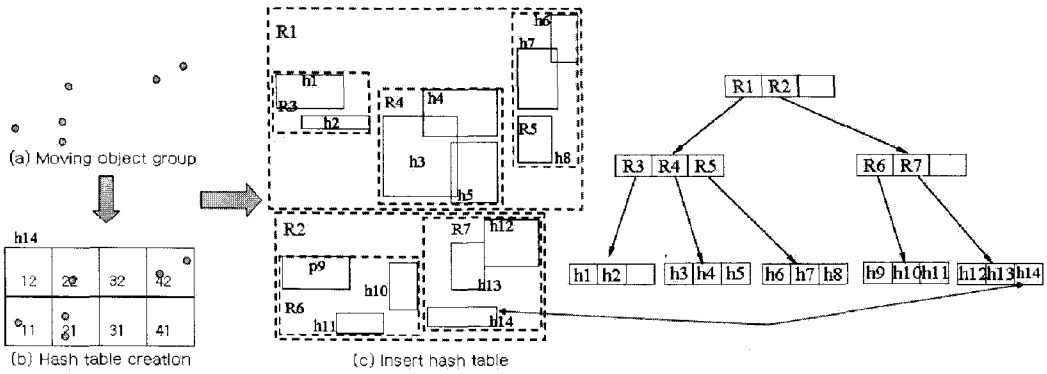
$$f_i(v_i) = \frac{v_i - \min_i}{\left\lceil \frac{\max_i - \min_i + 1}{m_i} \right\rceil}$$

$\max_i$와 $\min_i$는 각각 공간 데이타베이스에 존재하는 모든 공간 객체들의 i축의 값에 대한 최대값과 최소값이다. m_i는 i축을 구성하는 구간의 수이다. 즉, i축은 m_i개로 나뉘어진다. 그러므로, 셀 주소는 $(0,0,\ldots,0) \sim (m_1, m_2, \ldots, m_d)$의 범위에 존재하게 된다. m_i의 값이 $\Delta i(\max_i - \min_i + 1)$에 근접한다면, 즉 m_i의 값이 커지면 셀의 크기가 작아지기 때문에 이동 객체는 밀집 상태가 되어 오버플로우가 많이 발생하게 된다. m_i의 값이 1에 근접한다면, 즉 m_i의 값이 작아지면 셀의 크기가 커져서 셀에 속하게 되는 이동 객체가 많아지게 되므로 인접한 이동 객체를 찾는 것이 쉽지만 메모리 공간을 많이 차지하게 된다. 그러므로, m_i는 이동 객체의 밀집도에 따라서 결정하는 것이 바람직하다.

3.3 HBR-tree의 생성

본 논문에서는 이동 객체가 일종의 그룹을 이룬 형태로 입력되는 것을 가정하고 있다. 실세계에서의 이동 객체는 일반적으로 그룹을 이루면서 동일한 궤적으로 이동한다. 즉, "2호선 지하철을 이용하는 시민들"은 지하철이 이동하는 궤적에 따라 위치 데이타가 발생되게 된다. 그러므로, 이러한 이동 객체의 위치 데이타는 위치 획득 단계에서 그룹으로 나누어 HBR-tree에 입력되게 된다.

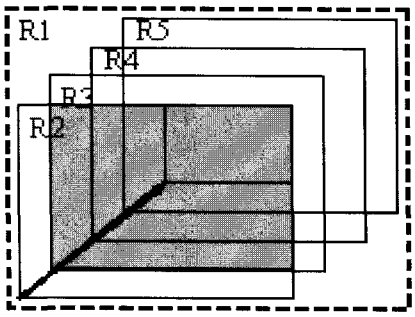
특정 시간에 〈그림 2(a)〉와 같은 새로운 위치 데이타 그룹이 삽입이 되면 〈그림 2(b)〉와 같이 위치 데이타 그룹에 해당하는 공간 해쉬 테이블이 만들어진다. 위치 데이타 그룹에 의해 생성된 공간 해쉬 테이블 h14는 그림 2(c)와 같이 HBR-tree에 삽입이 된다. 삽입된 h14는 R7의 자식 노드에 삽입이 된다. 만약 h14에 소속된 위치 데이타를 검색하고자 한다면 루트 노드에서 R2를 검색하여 R6, R7을 찾고, R6, R7을 검색하여 h14를 찾은 후 공간 해쉬 함수를 이용하여 위치 데이타를 검색하면 된다.



〈그림 2〉 이동 객체의 추가

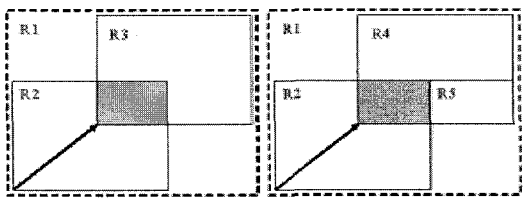
3.4 Lazy 갱신 및 해쉬 영역 분할

이동 객체의 그룹이 이동할 경우에 위치 획득 시간이 짧으면 〈그림 3〉과 같이 HBR-tree의 갱신이 빈번하게 일어나야 하는 경우가 발생한다. 〈그림 3〉에서는 이동 객체 그룹(예: 버스, 지하철)이 이동을 함에 따라 R2, R3, R4, R5의 공간 해쉬 테이블에 대한 MBR이 계속적으로 발생되는 경우를 예를 든 것이다. 이 경우에 HBR-tree에 대한 인덱스 갱신이 자주 발생되므로 비효율성이 발생하게 되는데 이를 해결하기 위해서는 위치 획득 간격을 늘리거나 Lazy 갱신과 같은 방법을 수행하여 HBR-tree의 빈번한 트리 재조정을 막아야 한다. 본 논문에서는 Lazy 갱신을 이용하여 인덱스 갱신이 빈번하게 일어나는 것을 해결하였다.



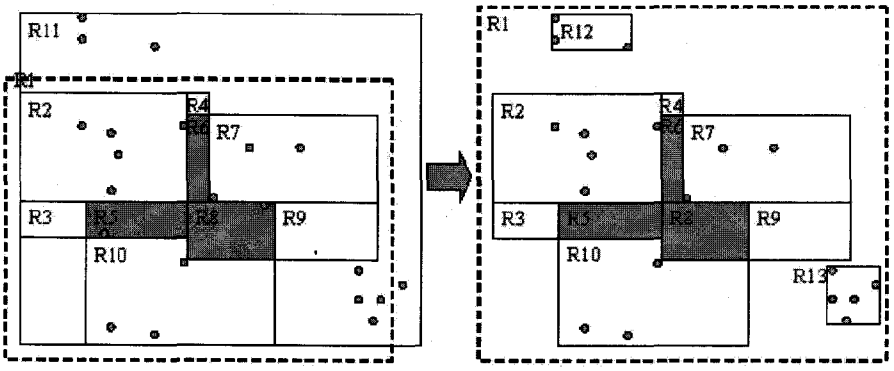
〈그림 3〉 이동 객체 그룹 이동시 MBR 발생 예

이동 객체의 위치 데이타를 획득하였을 때 공간 해쉬 테이블이 〈그림 4〉와 같이 구성된다면 회색의 영역에 대하여 메모리 중복이 발생하게 된다. 이 경우에 중복되는 영역은 공간 해쉬 테이블에서 동일한 영역을 동시에 가지기 때문에 불필요한 메모리 공간을 사용하게 되므로 HBR-tree에서는 분할(Split)이 수행된다. 그림 4에서는 이와 같은 공간 해쉬 테이블의 MBR이 중복되었을 경우 중복된 영역 R3가 분할되어 R4와 R5로 나뉘는 것을 보여준다.



〈그림 4〉 공간 해쉬 테이블의 MBR 중복 예 1

공간 해쉬 테이블에 대한 MBR의 중복은 〈그림 5〉와 같은 경우도 있을 수 있다. 현재 이미 구성된 MBR R1에 R10에 대한 MBR이 삽입되었을 경우에 R11에 있는 위치 데이타가 이미 구축된 HBR-tree에 소속되는 경우에는 이미 구축된 공간 해쉬 테이블에 위치 데이타를 삽입하고 구축된 HBR-tree에 포함되지 않는 경우에는 소속되지 않는 위치 데이타가 형성할 수 있는 최소의 MBR을 별도로 구성한다. 그림 5의 경우에는 R11이 추가되면서 R2, R3, R4, R5, R6, R7, R8, R9, R10의 공간 해쉬 테이블에 소속된 위치 데이타는 이미 생성된 HBR-tree에 삽입되고, 이에 소속되지 않은 위치 데이타는 별도의 공간 해쉬 테이블을 만들고 그에 대한 MBR인 R12와 R13이 HBR-tree에 새로이 삽입된다.



〈그림 5〉 공간 해쉬 테이블의 MBR 중복 예 2

4. 시스템의 개발

본 장에서는 HBR-tree를 이용한 실시간 모바일 GIS의 개발에 대해서 설명한다. 즉, 시스템의 구동 환경과 구성, 그리고 시스템의 주요 구현 기법에 대하여 설명한다.

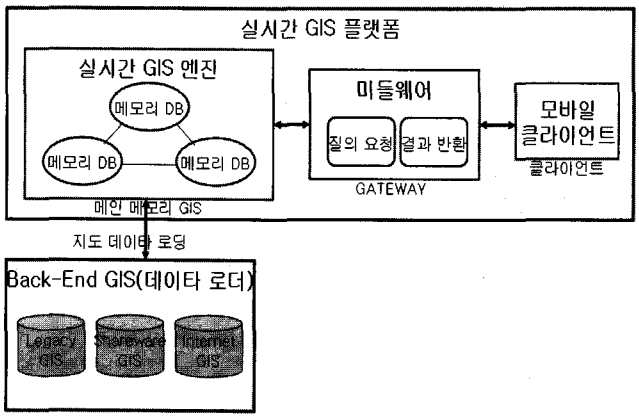
4.1 시스템의 구동 환경

일반적으로 디스크를 사용하는 기존의 GIS와는 달리 주기억 장치를 사용하는 실시간 GIS 엔진에서는 한번에 주기억 장치에 적재할 수 있는 양이 제한되어 있기 때문에 대용량의 지도 데이타를 모두 적재하기에는 문제가 있다. 그러므로, 본 논문에서는 기존의 GIS를 백엔드로 사용하고 데이타 로더를 이용하여 실시간 GIS 엔진에 데이타를 적재한다.

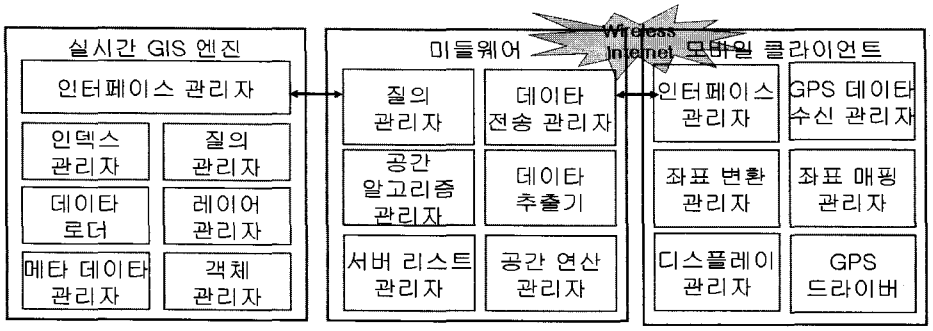
실시간 GIS 엔진은 주기억 장치의 용량에 따라 저장할 수 있는 데이타의 양이 결정되므로 하나의 시스템이나 프로세스에 저장할 수 있는 데이타의 양을 작은 규모로 처리한다. 즉, 작은 규모의 실시간 GIS 엔진을 여러 개 분산하여 구동한 후에 각 실시간 GIS 엔진과 데이타 통신을 수행하여 질의를 처리한다. 미들웨어는 실시간 GIS 엔진과 모바일 클라이언트 사이에서 질의를 전달하고 결과를 반환하며, 모바일 클라이언트는 실시간 GIS 엔진의 인터페이스 관리자와 연결되어 질의 및 데이타를 전송한다. 본 논문에서의 실시간 모바일 GIS의 구동 환경은 〈그림 6〉과 같다.

4.2 시스템의 구성

본 논문에서 개발한 실시간 모바일 GIS는 〈그림 7〉과 같이 인터페이스 관리자, 질의 관리자, 인덱스 관리자, 데이타 로더, 레이어 관리자, 메타 데이타 관리자, 객체 관리자로 구성된 실시간 GIS 엔진, 질의



〈그림 6〉 실시간 모바일 GIS의 구동 환경



〈그림 7〉 실시간 모바일 GIS의 시스템 구성

관리자, 데이타 전송 관리자, 공간 알고리즘 관리자, 데이타 추출기, 서버 리스트 관리자, 공간 연산 관리자로 구성된 미들웨어, 그리고 인터페이스 관리자, GPS 데이타 수신 관리자, 좌표 변환 관리자, 좌표 매핑 관리자, 디스플레이 관리자, GPS 드라이버로 구성된 모바일 클라이언트로 구성되어 있다.

실시간 GIS 엔진의 인터페이스 관리자는 서로 다른 시스템에서 서로 다른 사용자들이 실시간 GIS 엔진에 접근할 수 있도록 API를 제공한다. 질의 관리자는 공간, 비공간, 위치 데이타를 처리하기 위한 공간 연산을 제공한다. 인덱스 관리자에서는 공간, 비공간, 위치 데이타를 위한 인덱스를 제공하고, 메타 데이타 관리자는 테이블에 대하여 속성 정보, 객체에 대한 정보를 관리한다. 객체 관리자는 공간, 비공간, 위치 데이타를 삽입, 갱신, 삭제, 그리고 검색하는 기능을 수행한다. 생성되는 객체는 공유 메모리에 저장되며 메타 데이타 관리자에서 객체에 대한 offset, length, 데이타 타입 등을 관리하게 된다. 데이타 로더는 주기억 장치에 공간, 비공간, 위치 데이타를 필요할 때마다 적재하여 주는 기능을 한다. 그리고, 레이어 관리자는 공간 및 위치 데이타에 중요도를 부여하고 관리하기 위하여 사용된다.

미들웨어는 모바일 클라이언트에서 요구된 사항을 질의로 생성하여 실시간 GIS 엔진으로 전송하고, 그 처리 결과를 모바일 클라이언트로 전송한다. 미들웨어는 모바일 클라이언트에서 전달되는 질의를 처리하기 위한 질의 관리자, 공간 데이타를 이용한 연산을 하기 위한 공간 연산 관리자, 최단, 최적 경로 탐색과 같은 질의 처리를 위한 공간 알고리즘 관리자, 질의 결과로 반환될 데이타를 추출하기 위한 데이타 추출기, 등록된 실시간 GIS 엔진을 관리하기 위한 서버 리스트 관리자, 모바일 클라이언트로 데이타를 전송하기 위한 데이타 전송 관리자로 구성되어 있다.

모바일 클라이언트의 GPS 드라이버는 GPS 수신기가 위성으로부터 NMEA0183 프로토콜을 통해 수신한 위치정보를 GPS 데이타 수신 관리자로 전송한다. 디스플레이 관리자는 주기억 장치 GIS에서 전송된 데이타를 화면에 디스플레이 한다. 좌표 매핑 관리자는 실지 공간 데이타를 화면에 표시하기 위한 좌표로 매핑하는 기능을 한다. 좌표 변환 관리자는 좌표 값을 다른 좌표 체계로 변환하는 기능을 한다. GPS 데이타 수신 관리자는 GPS 드라이버로부터 입력받은 데이타를 파싱하여 자신의 위치 정보를 표시하는데 필요한 위도, 경도, 고도를 산출해 낸다. 인터페이스 관리자는 모바일 클라이언트의 기능을 사용자가 쉽게 사용할 수 있도록 GUI 환경을 제공한다.

4.3 시스템의 주요 구현기법

(1) 가변길이 데이타 관리 기법

일반적으로 고정 길이를 가지는 공간 데이타의 경우에는 그 칼럼의 길이가 테이블이 생성될 때 결정된다. 그러나, 가변 길이를 가지는 공간 데이타의 경우에는 칼럼의 길이가 포함된 점의 개수에 따라서 그 길이가 결정된다. 즉, 데이타 입력이 발생한 후에야 전체 길이를 알 수 있다. 본 논문에서는 실시간 GIS 엔진에서 공간 데이타 타입이 있는 테이블이 생성될 경우 입력 영역이라고 하는 공유 메모리 영역을 새로이 생성한다. 그리고, 메타 데이타 관리자에서는 입력되는 가변 길이 데이타에 대한 offset과 length 정보를 유지 관리한다. 이와 같은 방법을 사용하게 되면 실제 가변 길이 데이타는 별도로 저장되기 때문에 비공간 데이타에 대한 접근만이 일어날 경우에는 실제 공간 데이타에 대한 접근이 필요하지 않으므로 좀 더 빠른 데이타 검색을 수행할 수 있게 된다.

(2) 점진적 전송 기법

LBS를 사용하는 사용자는 주로 특정 위치에 대한 데이타를 사용하는 빈도수는 높은 반면에 특정 위치에서 먼 데이타는 관심이 없는 경향이 있다. 이를 고려한다면 무선 인터넷을 이용하여 데이타를 전송할 경우 사용자에게 필요한 공간 및 비공간 데이타를 먼저 전송한 후 거리가 먼 데이타를 전송하는 방법을 사용하게 되면 사용자의 질의에 대한 결과를 신속하게 디스플레이 할 수 있게 된다. 이를 위해서 사용되는 방법이 공간 및 비공간 데이타에 대한 점진적 전송 기법이다.

점진적 전송 기법에는 2가지의 방법이 있다. 첫번째는 위치를 기반으로 하여 인접한 위치에 있는 공간 및 비공간 데이타를 먼저 보내고 난 후에 먼 거리의 위치에 있는 데이타를 보내는 위치 기반 점진적 전송 기법이다. 두번째는 건물의 중요도를 중심으로 하여 중요도가 높은 공간 및 비공간 데이타를 먼저 보내고 난 후에 중요도가 낮은 데이타를 보내는 중요도 기반 점진적 전송 기법이다. 본 논문에서는 효율적인 위치 기반 서비스를 제공하기 위해서는 이러한 2가지 형태의 점진적 전송 기법을 모두 지원하고 있다.

(3) 좌표 변환 기법

일반적으로, GIS에서 사용되는 좌표는 8 바이트의 double 형을 가지는 공간 데이타 값을 사용한다. 이러한 공간 데이타 값을 맵 뷰어나 모바일 클라이언트에서 디스플레이하기 위해서는 실수와 정수에 대한 매핑 방법을 많이 사용한다. 실세계 좌표인 (x, y)를 모바일장치의 좌표인 (u, v)로 옮길 때 다음의 식을 사용하게 된다.

$$[u,v]=\left[(x-x_{\min})\left(\frac{u_{\max}-u_{\min}}{x_{\max}-x_{\min}}\right)+u_{\min},(y-y_{\min})\left(\frac{v_{\max}-v_{\min}}{y_{\max}-y_{\min}}\right)+v_{\min}\right]$$

여기서 우리가 ISOTROPIC(등방위) 모드를 쓴다고 가정하여 축척의 역수를 l로 표시하고 기준점을 (x_0, y_0) (u_0, v_0)로 치환하여 계산하면 다음과 같은 식을 사용할 수 있다.

$$[u,v] = [(x - x_0)/l + u_0,(y - y_0)/l + v_0]$$

여기서 (x_0, y_0) (u_0, v_0)는 좌표변환의 중심점이 된다.

(4) GPS 드라이버의 구현 기법

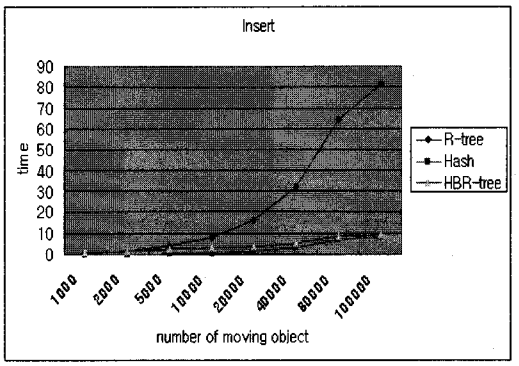
모바일 클라이언트가 GPS에 연결되었을 때 GPS로부터 데이타를 얻기 위하여 NMEA0183 라이브러리가 사용된다. 모바일 클라이언트에서는 현재 위치를 알기 위해서 GPGGA 항목이 기본적으로 사용되나, 최근의 GPS는 GPRMC 항목을 많이 사용한다. 본 논문에서는 사용자의 위치를 확인하기 위하여 GPRMC 항목을 사용하고, 위성의 정보를 화면에 디스플레이하기 위하여 GPGSV 항목을 사용한다. 특히, GPS로부터 입력되는 데이타들은 WGS84 좌표계를 따르고 있으므로 일반적으로 많이 사용되는 TM, UTM 좌표로 변환하여야 한다.

5. 시스템의 성능 평가

본 장에서는 실험적 방법을 사용하여 실시간 모바일 GIS의 주요 구성요소인 HBR-tree와 실시간 GIS 엔진의 성능 평가 결과에 대하여 설명한다.

5.1 HBR-tree의 성능 평가

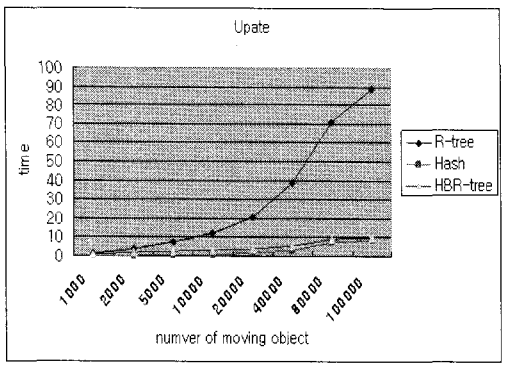
본 논문에서 HBR-tree의 성능 평가를 위해 R-tree와 공간 해쉬 인덱스와 비교하였고, 이를 위해 사용된 시스템의 사양은 다음과 같다. CPU는 펜티엄 III 800, RAM의 용량은 256MB, 그리고 운영 체제로는 RedHat 9.0 기반 하의 개인용 컴퓨터에서 수행하였다. 또한, 실험을 보다 객관화하기 위해 실험 프로그램을 제외한 모든 프로그램은 가동하지 않은 상태에서 실험을 수행하였다. 〈그림 8〉은 이동 객체의 개수에 따라 삽입을 수행한 결과이다.



〈그림 8〉 삽입에 대한 성능 평가

HBR-tree는 삽입의 처음에는 공간 해쉬 테이블을 이용하여 트리를 구성하기 때문에 시간이 많이 걸리지만 공간 해쉬 테이블을 이용한 트리가 점차적으로 구성이 되고 나면 그 이후에는 갱신을 위한 비용이 적기 때문에 공간 해쉬 인덱스만 사용하였을 경우의 속도에 근접하게 되고 R-tree에 비해서는 월등히 좋은 성능을 보이고 있다.

갱신 연산은 이동 객체의 수가 증가함에 따라 검색을 수행한 후에 그 값을 변경하는 방법을 사용하였다. 갱신에 대한 성능 평가 결과는 〈그림 9〉와 같다. 갱신 연산은 이동 객체의 위치 획득이 5분 단위로 발생한다고 가정하고 성능 평가를 수행하였다.

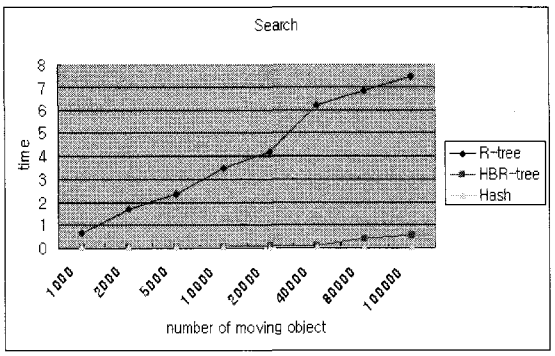


〈그림 9〉 갱신에 대한 성능 평가

HBR-tree는 트리를 검색하여 이동 객체의 위치 데이타가 포함된 공간 해쉬 테이블을 찾고 나면 공간 해쉬 테이블에서 검색 연산을 하기 때문에 이동 객체의 위치 데이타가 포함된 공간 해쉬 테이블을 찾는 시간을 제외하고는 공간 해쉬 인덱스와 유사한 성능을 보여주고 R-tree에 비해서는 월등히 좋은 성능을 보이고 있다.

검색은 이동 객체가 1,000~100,000개만큼 삽입된 상태에서 동일한 검색 영역을 설정한 후 그 영역에 해당하는 이동 객체를 검색하는데 소요되는 시간을 측정하였다. 〈그림 10〉은 검색에 대한 성능 평가 결과이다.

HBR-tree는 이동 객체가 포함되어 있는 공간 해쉬 테이블을 트리에서 찾게 되면 공간 해쉬 함수를 이용하여 이동 객체가 소속되는 셀을 바로 찾을 수 있기 때문에 빠른 검색이 가능하다. 공간 해쉬 인덱스는 공간 해쉬 테이블에서만 검색을 하면 되기 때문에 R-tree나 공간 해쉬 인덱스에 비해서 가장 빠른 성능을 보이고 있다.



〈그림 10〉 검색에 대한 성능 평가

5.2 실시간 GIS 엔진의 성능 평가

본 연구에서 개발된 실시간 GIS 엔진은 주기억 장치 데이타베이스 시스템을 모바일 GIS에 적합한 형태로 확장한 것이다. 실시간 GIS 엔진은 주기억 장치 데이타베이스 시스템을 확장한 시스템이기 때문에 이를 사용하기 위해서는 우선 데이타베이스, 테이블, 인덱스를 생성한 후 프로세스를 실행하여야 데이타베이스에 접근할 수 있다. 실시간 GIS 엔진에서는 대화형 질의 방식으로 이러한 데이타베이스, 테이블, 인덱스를 생성하지만 스크립트의 형태로 한번에 처리하는 방법도 사용될 수 있다.

〈그림 11〉은 위치 기반 서비스를 위한 데이타베이스, 테이블, 인덱스를 생성하는 스크립트이다. 생성될 데이타베이스 이름은 spatial_schema이고, 데이타베이스 관리자를 위한 암호는 lbs, 읽기/쓰기 접근 암호는 lbs1, 읽기 전용 암호는 lbs2로 설정되어 있다. spatial_schema 데이타베이스에서 가질 수 있는 최대의 테이블 수는 15개, 최대 열의 개수는 40, 최대 인덱스의 개수는 7, 최대 입력 영역의 개수는 1, 접근할 수 있는 프로세스의 개수는 50, 그리고 설정된 이벤트의 최대 개수는 8개로 지정되어 있다.

〈그림 11〉의 스크립트에서는 3개의 테이블, 4개의 인덱스, 그리고 1개의 입력 영역을 지정하고 있다. location01t 테이블은 특정 시간에 id 값을 가지는 객체의 위치 값과 오류를 저장하기 위한 테이블이다. apartment 및 road_in_6 테이블은 각각 아파트와 6M 이내 도로를 저장하기 위한 테이블이며 각각 초기 100,000개의 레코드를 저장할 수 있도록 지정되어 있다. 만약 100,000개 이상의 레코드가 저장될 경우에는 본 논문에서의 알고리즘에 의해서 데이타베이스

```
#!/bin/sh
DBNAME="location_schema"
cat > $DBNAME.sch <<EndOfData_1
#########################################################################
CREATE SCHEMA $DBNAME DBA=lbs WRITE=lbs1 READ=lbs2
                TABLE=15      COLUMN=40     INDEXES=7
                INPUTAREAS=1  SESSION=50    EVENT=8

CREATE INPUTAREA userarea  700
CREATE TABLE location01t 10000 ( id INTEGER,  current_time CHARACTER(7),
                                 cor_x RD_POINT, cor_y RD_POINT,
                                 err INTEGER )
CREATE TABLE apartment 10000 ( m_name CHARACTER(8), m_geo  POINT)
CREATE TABLE road_in_6 10000 ( m_name CHARACTER(8), m_geo    POLYLINE)

CREATE INDEX  location_index_id ON  location01t(id) UNIQUE PRIMERATIO=2.0
CREATE INDEX  location_index_time ON  location01t(current_time) PRIMERATIO=2.0
CREATE INDEX  apartment_index ON  apartment (m_name) UNIQUE  PRIMERATIO=2.0
CREATE INDEX  road_in_6_index ON  road_in_6(m_name)  UNIQUE PRIMERATIO=2.0
#########################################################################
EndOfData_1
```

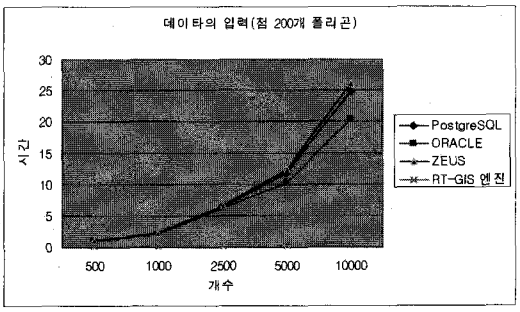
〈그림 11〉 데이타베이스, 테이블, 인덱스의 생성

의 크기가 자동으로 증가하게 된다.

실시간 GIS 엔진에서는 비공간 데이타를 위해서 B-Tree 인덱스와 해쉬 인덱스를 사용한다. 인덱스 생성 부분에서 PRIMERATIO는 해쉬 인덱스를 사용할 경우에 기본 비율에 대한 값을 설정하는 부분이다. 기본 비율의 값이 높을수록 키의 충돌이 적어지기 때문에 빠른 검색을 할 수 있다.

데이타베이스, 테이블, 인덱스를 생성한 후 삽입, 검색, 삭제, 갱신에 대한 성능 평가는 1GHz 듀얼 CPU, 1GB의 메모리, 512MB의 공유 메모리를 설정한 시스템 상에서 PostgreSQL, ORACLE, ZEUS, 실시간 GIS 엔진을 사용하여 공간 데이타의 삽입, 검색, 삭제, 갱신 작업에 대해 소요되는 시간을 측정하였다. 성능 평가를 하기 위해서 사용된 데이타는 200개의 점을 포함하고 있는 폴리곤을 사용하였다. 〈그림 12〉는 폴리곤의 수가 증가함에 따라 공간 데이타의 삽입에 소요된 시간을 보여준다.

일반적으로 디스크 기반 시스템(PostgreSQL, ORACLE, ZEUS)은 트랜잭션을 기반으로 하고 있기 때문에 인스턴스가 사용하는 메모리와 디스크의 동기화가 매우 중요하다. 그러므로, 일정한 시간이 지나면 메모리의 내용을 디스크로 쓰게 된다. 이로 인하여 점 200개로 구성된 폴리곤을 삽입하는 과정에서 디스크에 접근하지 않는 실시간 GIS 엔진은 기존의 디스크 기반 시스템에 비하여 20배~25배에 이르는 성능을 보이고 있다.



〈그림 12〉 공간 데이타의 삽입

실시간 GIS 엔진에서 메모리 할당 공식은 테이블일 경우 〈그림 13〉과 같은 공식을 사용한다. 〈그림 12〉의 성능 평가에 사용된 테이블은 〈그림 11〉의 road_in_6 테이블이고, 공식에 의하여 차지하는 메모리의 양은 테이블의 칼럼 수가 2개이고 튜플의 길이는 m_name이 8바이트, m_geo가 16바이트 * 200이므로 3,208 바이트를 차지

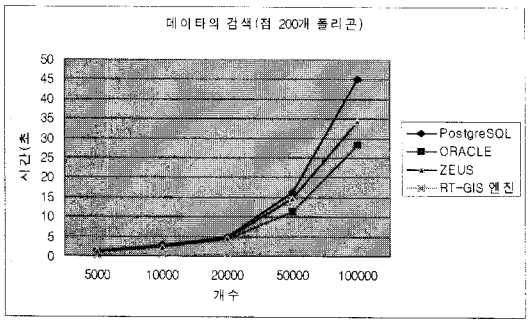
하게 된다. 그리고, 최대 튜플의 수는 10,000개이므로 road_in_6 테이블이 차지하는 메모리 공간은 56+(12*(2))+(3,208+8)*10,000 = 32,160,080 바이트이다. 그러므로, 대략 32 메가 바이트 정도의 용량으로 200개의 점을 가지는 폴리곤을 10,000개까지 삽입할 수 있다.

1GB의 물리적 메모리가 있을 경우 공유 메모리의 사용량을 실험해 보면 별도의 메모리 할당이나 프로세스가 동작하지 않는다고 가정할 때 1/2(512MB)를 사용할 경우가 가장 적당하고, 3/4(768MB) 이상을 공유 메모리로 사용할 경우에는 프로세스가 사용하는 메모리의 공간에 따라 스와핑이 발생한다. 그러므로, 512MB를 공유 메모리로 잡는다면 200개의 점을 가지는 폴리곤을 최대 160,000개까지 삽입할 수 있게 된다.

객체	공식
테이블	56(테이블 헤더)+(12*(테이블의 칼럼 수))+(튜플 길이+8)*(최대 튜플의 수)

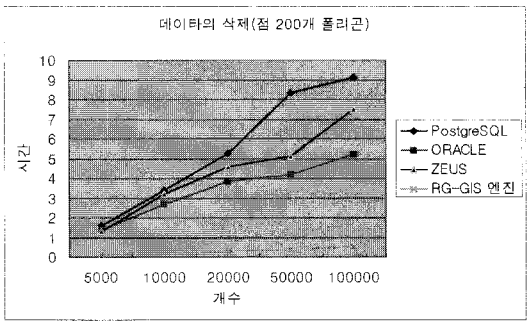
〈그림 13〉 테이블의 메모리 할당 공식

〈그림 14〉는 공간 데이타의 검색에 대한 성능 평가 결과이다. 공간 데이타의 검색에서도 road_in_6 테이블을 사용하였다. 검색은 5,000~100,000개의 폴리곤에 대해 전체 테이블 검색을 수행하여 걸리는 시간을 측정하였다. 일반적으로 디스크 기반 시스템은 메모리에 검색되는 내용을 블럭 단위로 적재하기 때문에 검색되어야 하는 데이타의 양이 많을수록 급격하게 검색 시간이 증가하였지만 실시간 GIS 엔진은 검색해야 하는 양이 많아도 급격한 시간의 증가는 보이지 않고 완만하게 증가하고 있다.



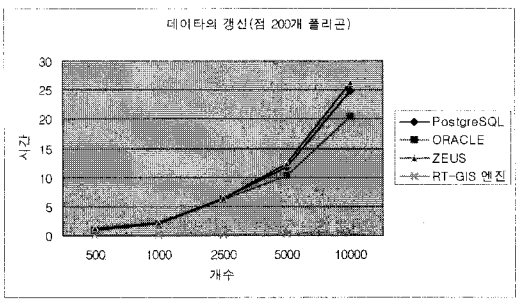
〈그림 14〉 공간 데이타의 검색

〈그림 15〉는 공간 데이타의 삭제에 대한 성능 평가 결과를 보여준다. 공간 데이타를 삭제할 경우에 기존의 디스크 기반 시스템에서는 "DELETE FROM road_in_6"을 사용하였다. 그리고, 실시간 GIS 엔진에서는 테이블의 처음에서 시작하여 순환문을 돌면서 MdRmTpl 함수를 호출하여 튜플들을 삭제하였다. 공간 데이타의 삭제 역시 디스크 기반 시스템에서는 롤백을 위한 리두 로그를 기록하는 시간이 있기 때문에 실시간 GIS 엔진에 비하여 많은 시간이 소요되었다. 그러나, "DROP TABLE road_in_6"를 이용하였을 경우에는 복구에 관련된 별도의 작업을 수행하지 않기 때문에 실시간 GIS 엔진과 유사한 시간이 소요되었다.



〈그림 15〉 공간 데이타의 삭제

〈그림 16〉은 공간 데이타 갱신 연산에 대한 성능 평가 결과를 보여준다. 갱신 연산에서는 검색에 사용되는 시간을 줄이기 위하여 road_in_6 테이블에 B-tree 인덱스를 사용하였다. 그리고, B-tree 인덱스의 키 속성은 m_name을 사용하였다. 실시간 GIS 엔진과 기존의 디스크 기반 시스템이 동일하게 B-tree



〈그림 16〉 공간 데이타의 갱신

를 제공하기 때문에 인덱스에 의한 검색 시간은 동일하다고 가정하였다. 갱신 연산은 삽입 연산과 유사한 성능 평가 결과를 보여주고 있다.

삽입, 검색, 삭제, 갱신의 성능 평가 결과에서도 볼 수 있듯이 본 논문에서 개발한 실시간 GIS 엔진을 사용할 경우 기존의 디스크 기반 시스템에 비하여 공간 데이타의 처리 속도가 매우 신속하게 이루어짐을 알 수 있다. 그러므로, 실시간 GIS 엔진은 공간 데이타의 요청이 많은 시스템이나 위치 기반 서비스와 같이 공간 및 위치 데이타의 삽입, 검색, 삭제, 갱신이 빈번하게 일어나는 경우에 매우 적합하다.

6. 결론 및 향후 연구 과제

최근 PDA, HPC와 같은 모바일 장치의 급속한 발전과 무선 인터넷의 사용이 증가함에 따라 GPS를 기반으로 한 LBS에 대한 관심이 점차적으로 확대되고 있다. 이러한 추세에 따라 국내외적으로 다양한 LBS가 생겨나고 있으며, 이동 통신 업체를 중심으로 하여 위치 추적, 경로 안내, 지역 안내 등과 같은 실생활에 밀접한 서비스를 제공하고 있다.

이와 같은 LBS에서 사용되는 위치 데이타는 기존의 GIS에서 사용되었던 변화가 적은 대용량의 정적인 데이타보다는 특정 시간에 갱신이 빈번한 동적인 데이타가 주로 사용된다. 그러므로, 기존의 GIS, 공간 인덱스, 저장 시스템을 이용하여 LBS에서 처리해야 할 이동 객체의 위치 데이타를 관리하는 것은 상당히 비효율적이다. 이를 위해서 본 논문에서는 이동 객체의 위치 데이타를 효과적으로 처리할 수 있는 HBR-tree를 이용한 실시간 모바일 GIS에 대하여 연구하였다.

본 연구에서의 HBR-tree는 기존의 공간 인덱스를 위치 데이타 처리에 사용하는 경우 갱신 비용이 크다는 단점을 해결하기 위하여 본 논문에서 제시한 위치 인덱스이다. HBR-tree는 2차원 해쉬 인덱스의 장점과 R-tree 계열 인덱스의 장점을 취합하여 개발된 인덱스로써 이동 객체의 변경 연산을 효과적으로 처리할 수 있다.

본 논문에서 개발한 실시간 모바일 GIS는 주로 실시간 GIS 엔진, 미들웨어, 모바일 클라이언트로 구성되며, 실시간 GIS 엔진은 이동 객체의 빈번한 갱신 연산을 효과적으로 처리하기 위한 시스템이고, 미들웨어는 모바일 클라이언트의 질의를 실시간 GIS 엔진에 전달하고 처리 결과를 모바일 클라이언트에 전송하는 역할을 한다. 모바일 클라이언트는 Windows CE, Embedded Linux와 같은 모바일 운영 체제에서 동작하며 이동 객체의 위치 데이타를 획득하고 질의를 요청하며 결과를 화면에 디스플레이한다. 특히 본 논문에서는 실험적 방법을 사용하여 HBR-tree와 실시간 GIS 엔진의 성능 평가 결과에 대해서도 기술하였다.

본 연구의 향후 연구 과제는 대용량으로 발생하는 이동 객체의 과거 위치 데이타를 효과적으로 저장할 수 있도록 HBR-tree를 확장하고, 또한 실시간 GIS 엔진에서 저장하기 어려운 대용량의 과거 위치 데이타를 효과적으로 저장할 수 있는 위치 저장 시스템을 개발하는 것이다.

참고문헌

[1] Forlizzi, L., Gueting, R.H., Nardelli, E., and Schneider, M., "A Data Model and Data Structures for Moving Objects Databases," Proceeding of the ACM SIGMOD Conference, 2000, pp.319-330.

[2] Niedzwiadek, H., "OpenLS-1 Interoperability Project: An Overview," LBS Forum Foundation Meeting and Celebration Seminar, 2002, pp.30-51.

[3] 윤재관, 김동오, 한기준, "LBS를 위한 실시간 GIS 엔진의 설계 및 구현," 한국정보과학회 학술발표논문집, 29권2호, 2002, pp.244-246.

[4] OpenLS Initiative, A Request for Technology In Support of an Open Location Services (OpenLS) Testbed, http://www.openls.org, 2000.

[5] Pfoser, D., Theodoridis, Y., and Jensen, C. S., Indexing Trajectories in Query Processing for Moving Objects, Chorochronos Technical Report, 1999.

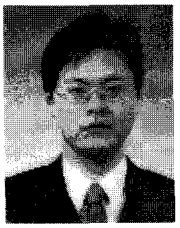
[6] Saltenis, S., Jensen, C. S., Leutenegger, S. T., and Lopez, M. A., "Indexing the Positions of Continuously Moving Objects," Proceeding. of the ACM SIGMOD Conference, 2000, pp.331-342.

[7] 전봉기, 홍봉희, "이동체 데이타베이스를 위한 해쉬 기반의 공간 색인," 한국정보과학회 학술발표논문집, 15권28호, 2001, pp.205-207.

[8] Pitoura, E., and Samaras, G., Data Management for Mobile Computing, Kluwer

Academic Publishers, 1998.
[9] Prakash, R., and Singhal, M., A Dynamic Approach to Location Management in Mobile Computing Systems, Dept. of Computer and Information Science, The Ohio State Univ., Technical Report, OSU-CISRC-4/96-TR22, 1996.
[10] Lee, K.Y., Kim, D.O., Yun, J.K., Han, K.H., "A Real-time Mobile GIS based on the HBR-tree," Proc. of 33rd International Conference on Computers & Industrial Engineering, 2004.
[11] 신명철, 박인하, 이강준, 한기준, "HP 실시간 데이타베이스 시스템을 위한 회복 시스템," 한국 데이타베이스 학술대회 논문집, 15권1호, 1999, pp.74-81.

이기영
1984년 숭실대학교 전자계산학과 졸업(학사)
1984~1991년 한국해양연구원 (KORDI) 연구원
1988년 건국대학교 대학원 컴퓨터공학과 졸업(석사)
1998년 건국대학교 대학원 컴퓨터공학과 박사수료
1991년~현재 서울보건대학 인터넷정보과 부교수
관심분야: 모바일 GIS, LBS, 공간 데이타 마이닝, 공간 데이타 웨어하우스



윤재관
1997년 건국대학교 컴퓨터공학과 졸업(학사)
1999년 건국대학교 대학원 컴퓨터공학과 졸업(석사)
2003년 건국대학교 대학원 컴퓨터공학과 졸업(박사)
2003년~현재 건국대학교 컴퓨터공학부 강의교수
관심분야: 실시간 데이타베이스, LBS, 위치 데이타 인덱스, 분산 위치 저장 시스템

한기준
1979년 서울대학교 수학교육학과 졸업(학사)
1981년 한국과학기술원(KAIST) 전산학과 졸업(석사)
1985년 한국과학기술원(KAIST) 전산학과 졸업(박사)
1990년 Stanford 대학 전산학과 visiting scholar
1985년~현재 건국대학교 컴퓨터공학부 교수
2004년~현재 개방형지리정보시스템학회 회장
2004년~현재 한국정보시스템감리사협회 회장
관심분야: 데이타베이스, GIS, LBS, 텔레매틱스, 정보시스템 감리