韓國地形空間情報學會論文集 第3卷 第2號 1995年 12月 pp. 137-149

# GOOD 2.0 : 공간 인덱스를 사용한 지리 데이타 관리기†

## GOOD 2.0 : a Geographical Data Manager using Spatial Indices

오병우* 한기준**
OH, Byoung-Woo, HAN, Ki-Joon

## 要 旨

점차 중요성이 증가하고 있는 지리 정보 시스템의 효율적인 검색을 위해서는 공간 인덱스가 필요하다. 이를 위하여 본 논문에서는 기존에 개발한 지리 정보 시스템을 위한 데이타 관리기인 GOOD 1.0에 공간 인덱스를 처리할 수 있는 공간 인덱스 처리 모듈을 추가하여 GOOD 2.0을 설계 및 구현한다. 즉, 공간 인덱스로는 R tree 및 R* tree를 지원하여 효율적인 검색이 가능하도록 한다. 그리고, 효율성의 향상 정도를 측정하기 위해 성능 평가를 실시하고 결과를 분석한다. 성능 평가 시에는 다양한 환경 요소들을 고려하여 지리 정보 시스템 관리자가 해당 도메인에 적합한 공간 인덱스를 선택하는데 기초 자료로서 사용될 수 있도록 한다.

## ABSTRACT

A spatial index is necessary to support an efficient search in a geographical information system (GIS) that is important in these days. In this paper, we design and implement a geographical data manager, called GOOD (Geo-Object Oriented Data Manager) 2.0, by extending GOOD 1.0 with a spatial index processing module. That is, R-tree and R*-tree are used as a spatial index in this paper to make an efficient search possible. In addition, this paper conducts a performance evaluation to measure the improvement in search efficiency and analyzes the results of the performance evaluation. When the performance evaluation is carried out, we consider various environment factors to allow an GIS administrator to use the results as a basic data in selecting an appropriate spatial index.

## 1. 서론

지리 정보 시스템(GIS: Geographical Information System)은 복잡하고 대용량인 지리 데이타를 종합적이고 효율적으로 다룰 수 있도록 해준다는 점에서 60년대 이후 데이타베이스의 중요한 연구 분야로서 자리잡고 현재 많은 연구가 진행 중이다[5, 9]. 지리 정보 시스템은 공간 및 비공간 데이타를 모두 다루어야 하는 교육, 기상, 경찰 업무, 행정, 군사, 국토 수치 정보, 대기 오염, 기상의 모니터 데이타, 인공위성 데이타, 국세 조사, 도로, 전기, 가스, 통신, 상하수도, 도시개발 계획, 개인 부동자산 관리 등의 매우 광범위하고 다양한 응용 환경에서 사용될 수 있는 중요한 정보 시스템이다[16].

† 본 연구는 '94년도 한국과학재단 핵심전문연구 (과제번호:941-0900-079-2)와 '95년도 한국통신 장기기초연구 (협약번호:95-16)의 지원에 의한 결과임.

* 건국대학교 전자계산학과 박사과정 ** 건국대학교 전자계산학과 교수

지리 정보 시스템에서는 공간 데이타를 처리하기 위한 가변 길이 필드, 공간 데이타의 출력을 위한 그래픽스, 지리 객체의 비공간 데이타를 처리하기 위한 멀티미디어 데이타 타입 등과 같이 기존의 관계형 데이타베이스 관리 시스템에서는 지원하지 못하는 많은 특성을 필요로 한다.

특히, 지리 정보 시스템에서 공간 데이타의 신속한 접근을 위한 공간 인덱스는 지리 정보 시스템의 효율성을 결정짓는 중요한 구성 요소로서 많은 연구들이 선행되어 R tree, R+ tree, R* tree, kd tree, Spatial kd tree, Buddy tree, LSD tree, PLOP hashing, Grid file, Bang file 등이 개발되었다 [1, 4, 6, 10, 12, 13].

본 논문에서는 이미 개발된 지리 정보 시스템을 위한 데이타 관리기인 GOOD (Geo-Object Oriented Data Manager) 1.0[14]에 공간 인덱스 기능을 추가하여 접근 효율성을 높일 수 있도록 하고, 또한 효율성의 향상을 확인하기 위해 성능 평가를 실시한다. GOOD 1.0에 공간 인덱스 기능을 추가한 GOOD 2.0에서는 기존의 데이타 처리 모듈, 이미지 처리 모듈, 그리고 수입/수출 모듈에 공간 인덱스 처리 모듈을 추가하여 전체 시스템을 설계 및 구현한다. 공간 인덱스로는 R tree[6]와 R* tree[1]를 채택하여 기존의 순차 검색에 의한 공간 질의 뿐만아니라 R tree 또는 R* tree에 의한 검색도 가능하게 한다. 이를 위하여 GOOD 1.0의 클래스 유지 관리를 위한 자료구조에 현재의 공간 인덱스를 기억하는 변수와 트리 구조를 참조하는 OID 변수를 추가한다. 또한, 가능한 GOOD 1.0의 함수들은 그대로 유지하고, 클래스 생성, 객체 삽입, 공간 질의 등의 함수 내부는 크게 개선하여 사용자에게 투명성을 제공한다. 그리고, 공간 인덱스의 변경 등과 같은 함수들도 추가로 설계 및 구현한다.

본 논문에서는 GOOD 2.0의 성능 평가를 위하여 4가지 데이타 집합을 사용한다. 즉, 객체의 크기가 큰 균일 분포, 객체의 크기가 작은 균일 분포, 객체의 크기가 큰 밀집 분포, 그리고 객체의 크기가 작은 밀집 분포이다. 이러한 데이타 집합들에 대해 객체의 갯수, 그리고 R tree와 R* tree의 버켓 크기를 변경시키면서 GOOD 2.0의 객체 삽입과 검색을 수행하고 성능 결과를 분석한다. 그리하여 다양한 경우에 대한 성능 평가를 통해 지리 정보 시스템 관리자가 해당 도메인에 적합한 공간 인덱스를 선택하는데 기초 자료로써 사용될 수 있도록 한다.

본 논문의 구성은 다음과 같다. 제 2 장에서 관련 연구로서 GOOD 1.0의 특징과 장단점 및 본 논문에서 사용하는 R tree와 R* tree에 대해 살펴보고, 제 3 장에서 GOOD 2.0의 설계 및 구현에 대해 언급한다. 그리고, 제 4 장에서는 GOOD 2.0의 성능 평가를 통해 얻은 효율성 향상에 대한 정량적인 결과를 분석한다. 마지막으로 제 5 장에서는 결론 및 이후의 연구 과제에 대해 언급한다.

## 2. 관련 연구

본 장에서는 기존의 GOOD 1.0의 특징과 사용된 개념들을 정의하고, 구성 및 장단점에 대해 살펴본다. 그리고, 공간 인덱스로 사용하는 R tree 및 R* tree의 구조 및 특징들에 대해 언급한다.

### 2.1 GOOD 1.0[14]

GOOD 1.0은 범용의 지리 데이타 관리기로서 지리 데이타 즉, 공간 및 비공간 데이타를 다루는 모든 응용 프로그램에서 사용이 가능하다. 특히, 지리 데이타는 대용량이므로 지리 정보 시스템에서는 분산 처리가 필요하다. 그러므로 GOOD 1.0은 물리적인 저장을 위해서 EXODUS Storage Manager[2]를 사용하여 분산 처리를 지원한다. 또한, API(Application Programming Interface)로써 호출 가능 함수들의 라이브러리(Callable Function Library)를 사용하여 응용 프로그램에서 별도의 내재 언어를 습득할 필요없이 바로 링크시켜 사용할 수 있다.

GOOD 1.0에서는 도메인, 클래스, 객체 등의 객체 지향 개념을 지원하며 이들 각각에 대한 설명은 다음과 같다.

- 도메인 : 다른 호스트에 분산되어 저장된 여러 지리 정보 시스템의 단위

- 클래스 : 한 종류의 지리 데이타가 갖는 공통적인 특성을 나타내는 공간 및 비공간 애트리뷰트들의 집합
- 객 체 : 임의의 클래스에 속하는 하나의 고유한 지리 데이타

GOOD 1.0은 공간 및 비공간 데이타의 처리를 위한 데이타 처리 모듈, X-Windows에서의 객체 및 배경 지도의 출력을 위한 이미지 처리 모듈, 그리고 외부 지리 정보 시스템 또는 화일 시스템과의 데이타 상호 변환을 위한 수입/수출 처리 모듈로 나뉘어 진다. 각각의 모듈에 대한 상세한 설명은 GOOD 2.0의 설계 및 구현에서 설명하겠다.

GOOD 1.0을 사용하면 다음과 같은 장점을 얻을 수 있다. 첫째, 분산 처리가 가능해 데이타의 중복없이 해당 도메인의 데이타만을 유지 관리하면 되므로 저장 공간을 크게 줄일 수 있다. 둘째, 다양한 데이타 타입을 지원하므로 사용자들이 원하는 클래스의 특성을 충분히 표현할 수 있다. 셋째, 각 클래스는 별도의 논리 구조 및 물리 구조로 유지 관리됨으로 효율적인 접근이 가능하다. 넷째, 다중 사용자의 사용에 의해 데이타의 공용성 및 활용성을 높일 수 있다. 마지막으로, API로써 호출 가능한 함수들을 사용하므로 별도의 인터프리트 시간 및 새로운 질의어 습득 기간이 필요 없어 응용 프로그램의 개발 비용 및 시간을 단축시킬 수 있다.

GOOD 1.0에서는 비공간 질의와 공간 질의를 모두 지원한다. 특히, 공간 질의로는 Furthest, Closest, Contain, Included, Intersect, Adjacent, Range 연산을 제공한다. 그러나, GOOD 1.0의 공간 질의 처리는 모두 순차 검색에 의해 이루어지므로 효율적인 검색이 불가능하였다. 본 논문에서는 효율적인 공간 질의 처리를 위하여 공간 인덱스 기능을 추가한 GOOD 2.0을 제시한다. GOOD 2.0에서는 공간 인덱스로써 R tree 및 R* tree를 사용한다.

## 2.2 공간 인덱스

효율적인 공간 데이타의 검색을 위하여 이제까지 많은 공간 인덱스들에 대한 연구가 진행되었다. 본 논문에서는 공간 인덱스로서 R tree 및 R* tree를 사용하므로 본 절에서는 이들 특징에 대해 간략히 살펴본다.

### 2.2.1 R tree[6]

R tree는 다차원 사각형들을 자르거나 보다 높은 차원의 점으로 변환하지 않고 완전한 객체로 저장하는 공간 인덱스이다. 중간 노드는 (*CP*, *Rectangle*) 형태의 엔트리들을 포함하고 있는데, 여기서 *CP*는 R tree내에서 자식 노드의 주소이고, *Rectangle*(즉, 디렉토리 사각형)은 그 자식 노드내에 있는 엔트리들인 모든 사각형들에 대한 MBR(Minimum Bounding Rectangle)이다. 이때 노드내의 엔트리 갯수를 버켓의 크기라 한다. 단말 노드는 (*Oid*, *Rectangle*) 형태의 엔트리들을 포함하는데, 여기서 *Oid*는 데이타베이스내에서 공간 객체를 기술하는 한 레코드를 가리키고, *Rectangle*(즉, 데이타 사각형)은 그 공간 객체의 MBR이다.

M을 한 노드에 들어갈 수 있는 엔트리 갯수의 최대값(즉, 버켓의 크기)이라 하고 m을 최소값이라고 하면 R tree의 특징은 다음과 같다. 단, $m \leq M / 2$ 이다.

- 단말 노드가 아닌 루트 노드는 적어도 2개의 자식 노드를 갖는다.
- 루트 노드가 아닌 모든 중간 노드는 m에서 M개의 자식 노드를 갖는다.
- 루트 노드가 아닌 모든 단말 노드는 m에서 M개의 자식 노드를 갖는다.
- 모든 단말 노드는 같은 레벨에 존재한다.

R tree에서의 검색은 B+ tree에서의 검색 방법과 유사하지만 한 노드에서 방문하여야 하는 서브트리가 하나 이상이다. 즉, 서브트리는 서로 격리되어 있지 않고 중첩될 수 있다. R tree에서의 삽입도 B+ 트리와 같이 단말 노드에 객체를 참조하는 포인터를 추가하고 오버플로우시에 분할된다. 즉, 객체를 삽입할 때 노드에 M개의 엔트리가 있어 더 이상 삽입할 수 없

다면 M+1개의 엔트리를 2개의 노드에 분할하여 저장한다. 분할을 위한 알고리즘들은 분할로 생긴 두 사각형 면적의 합이 최소가 되도록 한다.

2.2.2 R* tree[1)]

R* tree는 R tree의 특성을 그대로 유지하면서 여러 특성들을 최적화시킨다. R tree는 각각의 중간 노드내에 있는 디렉토리 사각형의 면적을 최소화하는 공간 인덱스인 반면에 R* tree에서는 다음과 같은 특성을 갖는다.

- 디렉토리 사각형에 의해서 커버되는 면적이 최소화된다.
- 디렉토리 사각형들간의 중복이 최소화된다.
- 디렉토리 사각형의 둘레가 최소화된다.
- 기억장소 활용율이 최적화된다.

일반적으로 R* tree를 사용한 검색이 모든 환경에서 R tree보다 우수하며, 또한 평균 삽입 비용에서도 R* tree가 R tree보다 우수하다[1)]. 반면에 R* tree의 구현 비용은 R tree의 구현 비용보다 약간 크다. R* tree에서도 R tree와 같이 중복되는 디렉토리 사각형들이 허용되므로 특정 질의인 경우에 유일한 탐색 경로가 보장되지 않는다.

본 논문에서는 지리 정보 시스템을 위한 데이타 관리기인 기존의 GOOD 1.0에 공간 인덱스 기능(즉, R tree와 R* tree)을 첨가하여 GOOD 2.0을 구현하고, 다양한 경우에 대해 공간 인덱스를 사용하지 않는 삽입, R tree를 사용한 삽입, R* tree를 사용한 삽입을 실제 수행하여 성능을 분석한다. 또한, 순차 검색, R tree를 사용한 검색, R* tree를 사용한 검색도 실제 수행하여 성능을 분석한다.

## 3. GOOD 2.0의 설계 및 구현

GOOD 1.0의 구조 및 함수들을 최대한 유지하도록 GOOD 2.0을 설계 및 구현하여 GOOD 1.0에서 GOOD 2.0으로 변환시에 개발 비용 및 시간이 최소화되도록 하였다. 본 장에서는 GOOD 2.0의 전체적인 구조와 모듈의 설계 및 구현에 대해 설명하고, 특히 공간 인덱스 처리 모듈에 대해서 설명한다.

### 3.1 GOOD 2.0의 구조

본 논문에서 제시하는 GOOD 2.0의 구조는 그림 3.1과 같다. GOOD 2.0 데이타 관리기는 지리 데이타의 저장을 위한 EXODUS 저장 관리기를 기반으로 하여 데이타 처리 모듈, 이미지 처리 모듈, 수입/수출 처리 모듈, 그리고 공간 인덱스 처리 모듈로 구성된다.

3.1.1 EXODUS 저장 관리기

GOOD 2.0은 저장 관리기로서 EXODUS 저장 관리기[2)]를 사용한다. EXODUS 저장 관리기는 미국의 위스콘신 대학에서 개발한 객체 지향 데이타베이스 관리 시스템의 개발을 위한 개발 도구이다. EXODUS 저장 관리기는 API로써 호출 가능 함수 라이브러리를 사용한다. 즉, 함수 호출을 통해 상위 계층에 루트 엔트리, 화일, 객체, 색인 등에 대한 삽입, 삭제, 검색 등을 제공한다. 또한, 트랜잭션, 클라이언트/서버 구조, 두-단계 록킹(two-phase locking), 버전, 버퍼 관리, 대형 객체 등도 지원한다.

3.1.2 GOOD 2.0 데이타 관리기

GOOD 2.0 데이타 관리기는 데이타 처리 모듈, 이미지 처리 모듈, 수입/수출 처리 모듈, 그리고 공간 인덱스 처리 모듈로 구성되며 각각의 모듈에 대해서 다음절에서 자세히 설명한다.

### 3.2 데이타 처리 모듈

데이타 처리 모듈은 지리 정보 시스템에 저장된 전반적인 데이타를 처리하는 모듈로서 상위 계층에서 하나의 지리 데이타를 공간 및 비공간 애트리뷰트 값을 가진 하나의 객체로서 생각할 수 있도록 해준다. 데이타 처리 모듈은 초기화 처리, 도메인 처리, 클래스 처리, 애트리뷰트 처리, 객체 처리, 공간 질의 처리, 비공간 질의 처리로 나뉘어 지며 각각의 설명은 다음과 같다.

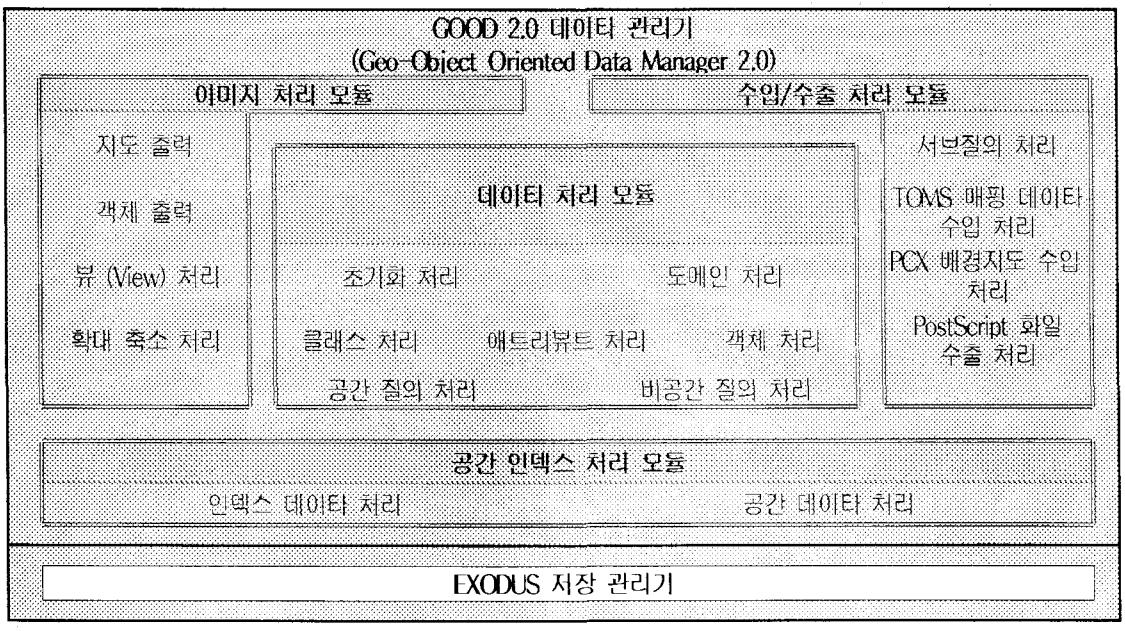



그림 3.1 GOOD 2.0의 구조

- 초기화 처리 : 초기화 처리는 GOOD 2.0을 사용하기 위해 시스템을 기동시키는 역할을 수행한다. 즉, 내부 변수들을 초기화시킨다. 특히, GOOD 2.0은 저장 관리기로서 EXODUS 저장 관리기를 사용하므로 클라이언트 부분을 초기화하고, R tree 및 R* tree를 위한 내부 변수도 초기화한다.
- 도메인 처리 : GOOD 2.0은 저장 관리기에서 제공하는 클라이언트/서버 구조를 사용하므로 도메인 처리를 통해 다른 호스트 또는 동일한 호스트에 저장된 특정 도메인의 지리 데이타에 대한 접근을 제공한다.
- 클래스 처리 : 클래스 처리는 클래스의 삽입, 갱신, 삭제 그리고 검색을 담당한다. GOOD 2.0에서는 클래스의 처리를 위해 공간 인덱스에 관한 정보가 필요하다. 그러므로, 공간 접근 방법(SAM: Spatial Access Method)이 무엇인지를 기억하는 변수와 트리 구조를 참조하는 OID 변수를 유지 관리한다.
- 애트리뷰트 처리 : 애트리뷰트 처리는 해당 클래스에 속하는 애트리뷰트의 삽입, 갱신, 삭제, 그리고 검색을 담당한다. 이를 위해 애트리뷰트 이름, 애트리뷰트 타입, 애트리뷰트 크기를 유지 관리한다.
- 객체 처리 : 객체 처리는 해당 클래스의 실제 지리 데이타를 생성, 삭제, 그리고 해당 객체의 애트리뷰트 값의 갱신 또는 검색을 담당한다. GOOD 2.0에서는 갱신 또는 검색하려는 애트리뷰트의 타입이 공간 타입이면 해당 클래스의 공간 접근 방법(순차 또는 공간 인덱스)을 참조하여 대응되는 접근 방법을 통한 갱신 또는 검색이 수행된다. 그리하여, 사용자에게는 동일한 API를 제공하지만 함수 내부의 처리는 공간 인덱스를 사용하여 보다 효율적인 갱신 및 검색이 가능하게 된다.
- 공간 질의 처리 : GOOD 2.0에서는 Furthest, Closest, Contain, Included, Intersect, Adjacent, Range의 공간 연산을 제공한다. 공간 연산의 결과를 얻기 위해 먼저 후보 객체들을 공간 인덱스를 사용하여 얻는다. 그리고, 이를 바탕으로 정확한 공간 연산을 수행한다.
- 비공간 질의 처리 : 비공간 질의 처리는 비공간 애트리뷰트에 대한 연산(즉, <, >, <=, >=, =, <> 등의 연산)의 처리를 담당한다.

데이타 처리 모듈에서 제공하는 함수들은 그림 3.2와 같다.

```
int     GInitialize(int argc, char *argv[]);
int     GClose();
int     GGetDomainList(GDOMAINLIST **dlist);
int     GOpenDomain(GDOMAINDATA *domaindata);
int     GInsertClass(GCLASSDATA *classdata);
int     GUpdateClass(GCLASSDATA *classdata);
int     GDeleteClass(GCLASSDATA *classdata);
int     GSearchClass(GCLASSDATA *classdata, GCLASSDEF *cdef, int *offset, int *found);
int     GGetClassList(GCLASSLIST **clist);
int     GInsertAttribute(GCLASSDATA *classdata, GATTRDATA *attrdata);
int     GUpdateAttribute(GCLASSDATA *classdata, GATTRDATA *attrdata);
int     GDeleteAttribute(GCLASSDATA *classdata, GATTRDATA *attrdata);
int     GGetAttributeList(GCLASSDATA *classdata, GATTRLIST **alist);
int     GCreateObject(GCLASSDATA *classdata, char *objectname, GOBJECTDATA *objectdata);
int     GDeleteObject(GCLASSDATA *classdata, GOBJECTDATA *objectdata);
int     GGetClassObjectList(GCLASSDATA *classdata, GOBJECTLIST **olist);
int     GGetObjectAttribute(GCLASSDATA *classdata, GOBJECTDATA *objectdata, GATTRDATA *attrdata, char **data);
int     GSetObjectAttribute(GCLASSDATA *classdata, GOBJECTDATA *objectdata, GATTRDATA *attrdata, char *data);
int     GGetAspatialQueryObjectList(GCLASSLIST *clist, GOBJECTLIST *in_olist, GATTRDATA *attrdata,
                                    GASPATIALOP operator1, char *data1, GASPATIALOP operator2, char *data2,
                                    GOBJECTLIST **out_olist);
int     GGetPointQueryObjectList(GCLASSLIST *clist, int x, int y, GOBJECTLISST **olist);
int     GGetRegionQueryObjectList(GCLASSLIST *clist, int x1, int y1,  int x2, int y2, GOBJECTLIST **olist);
int     GGetFurthestObjectList(GOBJECTDATA *objectdata, GCLASSLIST *clist, GOBJECTLIST *in_olist,
                                    GOBJECTLIST **out_olist);
int     GGetClosestObjectList(GOBJECTDATA *objectdata, GCLASSLIST *clist, GOBJECTLIST *in_olist,
                                    GOBJECTLIST **out_olist);
int     GGetContainObjectList(GOBJECTDATA *objectdata, GCLASSLIST *clist, GOBJECTLIST *in_olist,
                                    GOBJECTLIST **out_olist);
int     GGetIncludedObjectList(GOBJECTDATA *objectdata, GCLASSLIST *clist, GOBJECTLIST *in_olist,
                                    GOBJECTLIST **out_olist);
int     GGetIntersectObjectList(GOBJECTDATA *objectdata, GCLASSLIST *clist, GOBJECTLIST *in_olist,
                                    GOBJECTLIST **out_olist);
int     GGetAdjacentObjectList(GOBJECTDATA *objectdata, GCLASSLIST *clist, GOBJECTLIST *in_olist,
                                    GOBJECTLIST **out_olist);
int     GGetRangeObjectList(GOBJECTDATA *objectdata, long range, GCLASSLIST *clist, GOBJECTLIST *in_olist,
                                    GOBJECTLIST **out_olist);
```

그림 3.2 데이타 처리 모듈의 함수

### 3.3 이미지 처리 모듈

이미지 처리 모듈은 지리 정보 시스템에 저장된 공간 데이타를 화면으로 출력하는 기능을 담당한다. GOOD 2.0에서 이미지 처리 모듈은 비트맵과 벡터 데이타의 그래픽스 출력을 위하여 X-Windows를 사용하며 지도 출력, 객체 출력, 뷰 처리, 확대 축소 처리로 나뉘어 진다. 이들 각각에 대한 설명은 다음과 같다.

- 지도 출력 : 지도출력은 비트맵 배경지도를 출력하여 시각적인 효과를 증대시키는 역할을 담당한다.
- 객체 출력 : 객체출력은 벡터형태로 저장된 객체의 공간 데이타를 출력하는 역할을 담당한다.
- 뷰 처리 : 뷰 처리는 특정 비공간 애트리뷰트의 값을 지도 위에 색으로서 출력하여 사용자들이 쉽게 비공간 데이타를 분석할 수 있도록 한다.
- 확대 축소 처리 : 확대 축소 처리는 특정 배율로 비트맵 배경지도 및 객체를 확대 또는 축소하여 사용자가 이해하기 쉬운 원하는 결과를 쉽게 얻을 수 있도록 한다.

이미지 처리 모듈에서 제공하는 함수들은 그림 3.3과 같다.

```
int     GiGetMap(Display *display, GC gc, int sizex, int sizey, Drawable pixmap);
int     GiGetMapHeader(GMAPHEADER *header);
int     GiGetMapSizeXY(int *sizex, int *sizey);
int     GiDrawObject(GOBJECTDATA *odata, Display *display, GC gc, Drawable pixmap);
int     GiDrawObjectList(GOBJECTLIST *olist, Display *display, GC gc, Drawable pixmap);
int     GiGetViewMap(GCLASSDATA *classdata, GOBJECTLIST *olist, GATTRDATA *attrdata, int order,
                     Display *display, GC gc, int red, int green, int blue, Drawable pixmap);
int     GiSetZoomFactor(float zoomx, float zoomy);
int     GiGetZoomFactor(float *zoomx, float *zoomy);
```

그림 3.3 이미지 처리 모듈의 함수

## 3.4 수입/수출 처리 모듈

수입/수출 처리 모듈은 GOOD 2.0과 다른 지리 정보 시스템 또는 화일 시스템간의 데이타를 교환할 경우 이를 처리하기 위해 확장 가능한 모듈이다. 수입/수출 처리 모듈은 서브질의 처리, TOMS 매핑 데이타 수입 처리, PCX 배경지도 수입 처리, PostScript 화일 수출 처리로 나뉘어 지며 각각에 대한 설명은 다음과 같다.

- 서브질의 처리 : 서브질의 처리는 서브질의의 결과를 저장하고 복원하는 역할을 담당한다. 복원할 때는 질의의 성격에 따라 합집합 중첩, 교집합 중첩, 단순 복원이 사용된다.
- TOMS 매핑 데이타 수입 처리 : TOMS 매핑 데이타 수입 처리는 VAX/VMS 운영체제의 Informap에서 작성된 매핑 데이타를 수입하는 역할을 담당한다.
- PCX 배경지도 수입 처리 : PCX 배경지도 수입 처리는 컬러 스캐너로 입력된 PCX 그래픽 화일의 배경지도를 수입하는 역할을 담당한다.
- PostScript 화일 수출 처리 : PostScript 화일 수출 처리는 질의에 의한 결과를 운영체제와는 독립적인 PostScript 화일로 수출하는 역할을 담당한다.

수입/수출 처리 모듈에서 제공하는 함수들은 그림 3.4와 같다.

```
int     GeSaveMap(char *filename, Display *display, GC gc, GOBJECTLIST *olist, Drawable pixmap);
int     GeLoadMap(char *filename, char load_option, Display *display, GC gc,
                  GOBJECTLIST *in_olist, Drawable in_pixmap,
                  GOBJECTLIST **out_olist, Drawable out_pixmap);
int     GeTOMS2PS_Init(char *psfilename, FILE **psfp);
int     GeTOMS2PS_Kind1(int areacode, int tablenum, FILE *psfp);
int     GeTOMS2PS_Kind2(int areacode, int tablenum, FILE *psfp);
int     GeTOMS2PS_Kind3(int areacode, int tablenum, FILE *psfp);
int     GeTOMS2PS_Kind4(int areacode, int tablenum, FILE *psfp);
int     GeTOMS2PS_Kind5(int areacode, int tablenum, FILE *psfp);
int     GeTOMS2PS_Close(FILE *psfp);
int     GeImportPS(char *inname);
int     GeImportPCX(char *inname);
int     GeExportPS(char *outname, int startx, int starty, int sizex, int sizey, GOBJECTLIST *olist, Drawable pixmap);
```

그림 3.4 수입/수출 처리 모듈의 함수

### 3.5 공간 인덱스 처리 모듈

공간 인덱스 처리 모듈은 효율적인 공간 데이타의 검색을 위해 공간 인덱스를 생성, 갱신, 삭제하고 공간 데이타를 삽입, 갱신, 삭제, 검색하는 역할을 담당한다. 본 논문에서는 EXODUS 저장 관리기를 위한 공간 인덱스를 사용하여 공간 인덱스 처리 모듈을 설계 및 구현하였다. 공간 인덱스로 R tree 및 R* tree를 제공하는 공간 인덱스 처리 모듈은 인덱스 데이타 처리와 공간 데이타 처리로 나뉘어 지며 각각에 대한 설명은 다음과 같다.

- 인덱스 데이타 처리 : 인덱스 데이타 처리는 공간 인덱스를 생성 또는 삭제하며, 다른 형태의 공간 인덱스로 전환하는 인덱스 자체에 대한 처리를 담당한다.
- 공간 데이타 처리 : 공간 데이타 처리는 생성된 공간 인덱스에 공간 데이타를 삽입, 갱신, 삭제하는 역할을 담당하며, 상위 계층에 함수로 제공되지 않고 내부 함수(예를 들면, 객체의 애트리뷰트 갱신 함수)들에서만 호출된다.

공간 인덱스 처리 모듈에서 제공하는 함수들은 그림 3.5와 같다.

```
int     GsSwitchSAMType(GCLASSDATA *cdata, int samtype);
int     GsInsertObject(GCCLASSDATA *cdata, GOBJECTDATA *odata, int x1, int y1, int x2, int y2);
int     GsDeleteObject(GCCLASSDATA *cdata, GOBJECTDATA *odata);
int     GsSearchObject(GCCLASSDATA *cdata, GOBJECTLIST **odata, int x, int y);
```

그림 3.5 공간 인덱스 처리 모듈의 함수

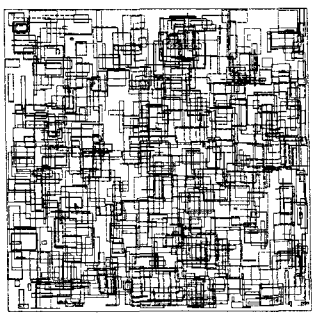
## 4. GOOD 2.0의 성능 평가

본 논문에서는 GOOD 2.0의 효율성 향상을 측정하기 위해 성능 평가를 실시하였다. 먼저 예상되는 객체의 분포에 따른 데이타 집합을 정의하고 각각의 데이타 집합에 대해 객체 삽입과 점 질(point query)의 검색에 소모되는 시간의 평균을 구한다. 이때 더욱 정밀한 성능 평가를 위하여 객체의 갯수와 트리 버켓의 크기를 변경하면서 GOOD 2.0을 실행하여 시간을 측정한다. 이처럼 다양한 경우에 대한 성능 평가 결과를 제공함으로써 지리 정보 시스템 관리자가 해당 도메인에 대한 공간 인덱스의 선택할 때 (즉, 순차, R tree, R* tree 중 어떤 방법을 사용할 것인지를 결정할 때) 기초 자료로써 사용될 수 있도록 한다.

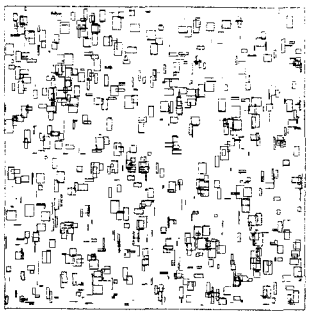
### 4.1 실험 환경

본 논문에서는 SUN SparcStation을 사용하고, 운영체제로는 SunOS 4.1.3을 사용하여 성능 평가를 실시하였다. 또한, 객체의 분포에 따른 GOOD 2.0의 성능 평가를 위하여 다음과 같은 데이타 집합을 고려하였다.

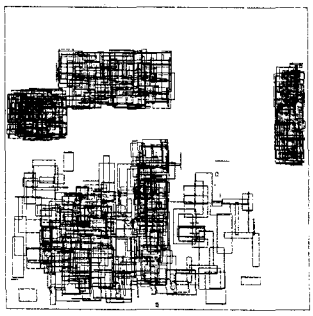
- 데이타 집합 1 : 그림 4.1 (a)와 같이 MBR의 크기가 전체 크기의 10% 이하인 객체들이 골고루 균일 분포하는 데이타 집합을 의미한다.
- 데이타 집합 2 : 그림 4.1 (b)와 같이 MBR의 크기가 전체 크기의 30% 이하인 객체들이 골고루 균일 분포하는 데이타 집합을 의미한다.
- 데이타 집합 3 : 그림 4.1 (c)와 같이 MBR의 크기가 전체 크기의 10% 이하인 객체들이 특정 지역들에 모여 밀집 분포하는 데이타 집합을 의미한다.
- 데이타 집합 4 : 그림 4.1 (d)와 같이 MBR의 크기가 전체 크기의 30% 이하인 객체들이 특정 지역들에 모여 밀집 분포하는 데이타 집합을 의미한다.



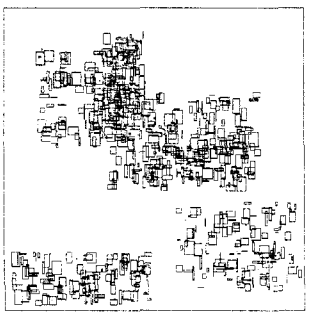
(a) 객체의 크기가 큰 균일 분포



(b) 객체의 크기가 작은 균일 분포



(c) 객체의 크기가 큰 밀집 분포



(d) 객체의 크기가 작은 밀집 분포

그림 4.1 데이타 집합의 예제

## 4.2 객체의 갯수에 따른 성능 평가

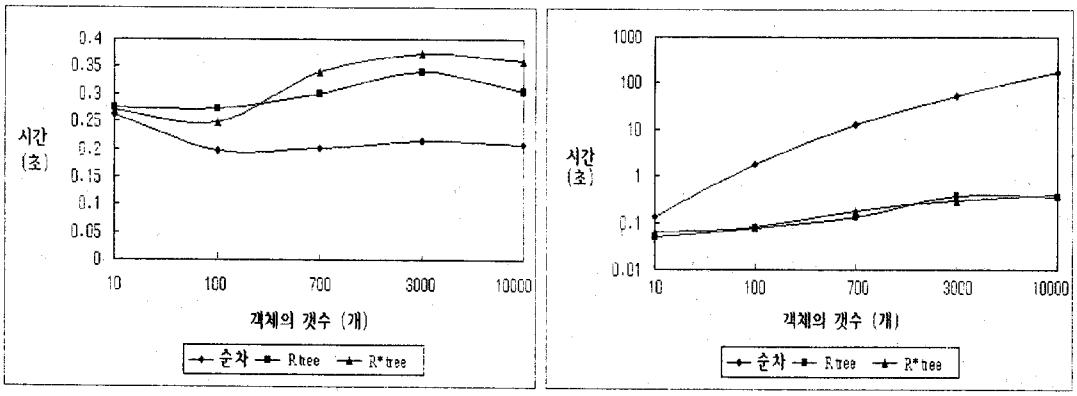
본 논문에서는 객체의 갯수에 따른 GOOD 2.0의 접근 방법에 관한 효율성을 측정하기 위해 각각의 데이타 집합에 대해 객체의 갯수를 10, 100, 700, 3000, 10000개로 변경시키면서 객체의 삽입과 검색에 소모되는 시간을 측정하였다. 표 4.1은 한개의 객체를 삽입하는데 소모된 평균 시간(즉, 전체 삽입 시간을 객체의 갯수로 나눈 시간)을 보여준다. 이를 통해 공간 인덱스를 사용하지 않는 객체의 삽입이 전체적으로 우수하다는 것을 알 수 있다. 특히, 객체의 갯수가 증가함에 따라 공간 인덱스를 사용하지 않는 방법이 공간 인덱스를 사용하는 방법보다 시간에 적게 걸린다는 결과를 얻었다. 그리고, 데이타 집합 3과 데이타 집합 4와 같이 객체가 밀집 분포인 경우에는 R tree를 사용한 삽입 시간이 R* tree를 사용한 삽입 시간 보다 적게 걸린다는 결과를 얻었다. 표 4.2는 한개의 객체를 검색하는데 소모된 평균 시간(즉, 전체 검색 시간을 검색 회수로 나눈 시간)을 보여준다. 이를 통해 공간 인덱스를 사용하는 검색이 객체의 갯수가 증가함에 따라 월등히 우수함을 알 수 있다. 그리고, 객체의 갯수가 많으면서 밀집 분포인 경우(즉, 데이타 집합 3과 데이타 집합 4)에서는 R* tree를 사용한 검색이 R tree를 사용한 검색보다 효율적이라는 결과를 얻었다. 그림 4.2는 모든 데이타 집합에서 객체의 갯수에 따라 삽입 및 검색에 소모된 평균 시간을 보여준다. 이를 통해 삽입시에는 공간 인덱스를 사용하지 않는 방법이 우수하며, 검색시에는 객체의 갯수가 증가함에 따라 R* tree를 사용한 검색이 우수해 진다는 결과를 얻었다.

표 4.1 객체의 갯수에 따른 객체 삽입 평균 시간

| 방법 / 갯수 | 데이타 집합 1 | | | 데이타 집합 2 | | | 데이타 집합 3 | | | 데이타 집합 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 순차 | R tree | R* tree | 순차 | R tree | R* tree | 순차 | R tree | R* tree | 순차 | R tree | R* tree |
| 10 | 0.275 | 0.269 | 0.259 | 0.248 | 0.270 | 0.311 | 0.283 | 0.262 | 0.259 | 0.243 | 0.297 | 0.251 |
| 100 | 0.189 | 0.233 | 0.238 | 0.195 | 0.236 | 0.245 | 0.201 | 0.306 | 0.255 | 0.206 | 0.313 | 0.256 |
| 700 | 0.218 | 0.274 | 0.329 | 0.150 | 0.279 | 0.339 | 0.217 | 0.319 | 0.357 | 0.216 | 0.322 | 0.332 |
| 3000 | 0.212 | 0.333 | 0.367 | 0.214 | 0.337 | 0.367 | 0.215 | 0.337 | 0.378 | 0.217 | 0.351 | 0.382 |
| 10000 | 0.202 | 0.308 | 0.348 | 0.207 | 0.312 | 0.357 | 0.212 | 0.297 | 0.377 | 0.213 | 0.301 | 0.358 |

표 4.2 객체의 갯수에 따른 객체 검색 평균 시간

| 방법 / 갯수 | 데이타 집합 1 | | | 데이타 집합 2 | | | 데이타 집합 3 | | | 데이타 집합 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 순차 | R tree | R* tree | 순차 | R tree | R* tree | 순차 | R tree | R* tree | 순차 | R tree | R* tree |
| 10 | 0.193 | 0.034 | 0.061 | 0.195 | 0.046 | 0.057 | 0.130 | 0.051 | 0.066 | 0.029 | 0.069 | 0.062 |
| 100 | 1.697 | 0.064 | 0.072 | 1.690 | 0.040 | 0.059 | 1.947 | 0.135 | 0.120 | 1.777 | 0.060 | 0.074 |
| 700 | 11.710 | 0.210 | 0.300 | 13.250 | 0.074 | 0.089 | 12.026 | 0.172 | 0.263 | 12.133 | 0.080 | 0.078 |
| 3000 | 51.001 | 0.101 | 0.101 | 51.286 | 0.190 | 0.264 | 50.404 | 0.969 | 0.662 | 50.655 | 0.232 | 0.179 |
| 10000 | 168.803 | 0.280 | 0.309 | 168.861 | 0.410 | 0.306 | 169.285 | 0.350 | 0.732 | 173.126 | 0.419 | 0.295 |




(a)삽입 (b) 검색

그림 4.2 객체의 갯수에 따른 삽입 및 검색 평균시간

### 4.3 버켓의 크기에 따른 성능 평가

본 논문에서는 버켓의 크기에 따른 R tree 및 R* tree를 사용한 GOOD 2.0의 접근 성능을 측정하기 위해 각각의 데이타 집합에 대해 버켓의 크기를 4, 10, 60, 100, 140으로 변경시키면서 객체의 삽입과 검색에 소모되는 시간을 측정하였다. 표 4.3은 한개의 객체를 삽입하는데 소모된 평균 시간을 보여준다. 이를 통해 모든 데이타 집합에 대해서 버켓의 크기와 거의 무관하게 R tree를 사용한 삽입 시간이 R* tree를 사용한 삽입 시간 보다 적게 걸린다는 결과를 얻었다.

표 4.4는 한개의 객체를 검색하는데 소모된 평균 시간을 보여준다. 이를 통해 객체의 크기가 작은 경우(즉, 그림 4.3은 모든 데이타 집합에서 버켓의 크기에 따라 삽입 및 검색에 소모된 평균 시간을 보여준다. 이를 통해 버켓의 크기와는 무관하게 삽입시에는 R tree를 사용한 삽입이 우수하며, 검색시에는 일반적으로 R*tree를 사용한 검색이 우수하다는 결과를 얻었다. 데이타 집합 2와 데이타 집합 4)에서는 버켓의 크기가 증가함에 따라 R* tree를 사용한 검색 시간이 R tree를 사용한 검색 시간 보다 적게 걸린다는 결과를 얻었다.

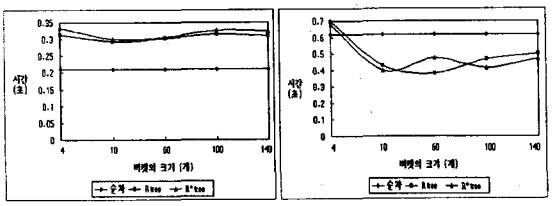
표 4.3 버켓의 크기에 따른 객체 삽입 평균 시간

| 방법 / 크기 | 데이타 집합 1 | | | 데이타 집합 2 | | | 데이타 집합 3 | | | 데이타 집합 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 순차 | R tree | R* tree | 순차 | R tree | R* tree | 순차 | R tree | R* tree | 순차 | R tree | R* tree |
| 4 | 0.203 | 0.301 | 0.325 | 0.226 | 0.305 | 0.336 | 0.206 | 0.322 | 0.334 | 0.207 | 0.324 | 0.337 |
| 10 | 0.203 | 0.291 | 0.303 | 0.226 | 0.270 | 0.297 | 0.206 | 0.312 | 0.316 | 0.207 | 0.295 | 0.288 |
| 60 | 0.203 | 0.302 | 0.311 | 0.226 | 0.293 | 0.302 | 0.206 | 0.319 | 0.309 | 0.207 | 0.298 | 0.304 |
| 100 | 0.203 | 0.311 | 0.347 | 0.226 | 0.318 | 0.322 | 0.206 | 0.315 | 0.329 | 0.207 | 0.313 | 0.309 |
| 140 | 0.203 | 0.314 | 0.326 | 0.226 | 0.300 | 0.318 | 0.206 | 0.313 | 0.318 | 0.207 | 0.313 | 0.336 |

표 4.4 버켓의 크기에 따른 객체 검색 평균 시간

| 방법 / 크기 | 데이타 집합 1 | | | 데이타 집합 2 | | | 데이타 집합 3 | | | 데이타 집합 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 순차 | R tree | R* tree | 순차 | R tree | R* tree | 순차 | R tree | R* tree | 순차 | R tree | R* tree |
| 4 | 0.590 | 0.798 | 0.779 | 0.627 | 1.097 | 1.141 | 0.641 | 0.829 | 0.644 | 0.614 | 0.061 | 0.148 |
| 10 | 0.590 | 0.864 | 0.710 | 0.627 | 0.060 | 0.085 | 0.641 | 0.673 | 0.749 | 0.614 | 0.117 | 0.046 |
| 60 | 0.590 | 0.717 | 0.915 | 0.627 | 0.077 | 0.089 | 0.641 | 0.595 | 0.755 | 0.614 | 0.125 | 0.144 |
| 100 | 0.590 | 0.999 | 0.810 | 0.627 | 0.203 | 0.111 | 0.641 | 0.538 | 0.600 | 0.614 | 0.128 | 0.135 |
| 140 | 0.590 | 0.962 | 1.030 | 0.627 | 0.163 | 0.105 | 0.641 | 0.711 | 0.636 | 0.614 | 0.168 | 0.116 |

그림 4.3은 모든 데이터 집합에서 버켓의 크기에 따라 삽입 및 검색에 소모된 평균시간을 보여준다. 이를 통해 버켓의 크기와는 무관하게 삽입시에는 R tree를 사용한 삽입이 우수하며, 검색시에는 일반적으로 R*tree를 사용한 검색이 우수하다는 결과를 얻었다.



(a)삽입 (b) 검색

그림 4.3 버켓의 크기에 따른 삽입 및 검색 평균 시간

## 5. 결 론

지리 정보 시스템에서 효율적인 공간 데이타의 검색을 위해서는 공간 인덱스가 필요하다. 이를 위하여 본 논문에서는 이미 개발된 지리 정보 시스템을 위한 데이타 관리기인 GOOD 1.0에 공간 인덱스 기능을 추가하여 효율성을 높일 수 있도록 하였다. 즉, 기존의 데이타 처리 모듈, 이미지 처리 모듈, 그리고 수입/수출 모듈에 공간 인덱스 처리 모듈을 추가하여 GOOD 2.0을 설계 및 구현하였다.

그리고, 효율성의 향상을 정량적으로 증명하기 위하여 GOOD 2.0의 성능 평가를 실시하였다. 성능 평가를 통해 다음과 같은 결과를 얻을 수 있었다. 첫째, 공간 인덱스를 사용하지 않는 객체의 삽입이 전체적으로 우수하다. 둘째, R tree를 사용한 삽입 시간은 R* tree를 사용한 삽입 시간 보다 적게 걸린다. 셋째, 객체의 갯수가 증가함에 따라 R* tree를 사용한 검색 시간이 R tree를 사용한 검색 시간 보다 적게 걸린다. 특히, 밀집 분포일 경우에서는 R* tree를 사용한 검색이 우수하다. 넷째, 크기가 작은 경우에는 버켓의 크기가 증가함에 따라 R* tree를 사용한 검색이 R tree를 사용한 검색 보다 우수하다.

이후의 연구 과제로는 R tree 및 R* tree 이외의 다른 공간 인덱스에 대한 연구가 있다. 즉, 다양한 공간 인덱스로부터 특정한 경우에 적합한 공간 인덱스를 선택하기 위한 연구가 필요하다. 그리고, DEM (Digital Elevation Model), TIN (Triangulated Irregular Network) 등과 같이 점차 3 차원 지리 데이타의 처리가 늘어나고 있으므로 3 차원 전용의 공간 인덱스에 대한 연구도 필요하다. 또한, 시공간 지리 정보 시스템을 위한 버전 개념을 지원하는 공간 인덱스에 대한 연구도 필요하다.

## 참 고 문 헌


1. Beckmann, N., Kriegel, H.P., Schneider, R., and Seeger, B., "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," Proc. of ACM SIGMOD Int. Conf., May 1990, pp. 322-331.
2. Carey, M, DeWitt, D., and Shekita, E., "Storage Management for Objects in EXODUS," in Object-Oriented Concepts, Databases and Applications, W. Kim and F. Lochovsky, eds., Addison-Wesley, 1989.
3. Faloutsos, C., and Kamel, I., "High Performance R-trees," IEEE Data Engineering, Vol.16, No.3, Sept. 1993, pp. 28-33.
4. Freeston, M., "The BANG File: A New Kind of Grid File," Proc. of ACM SIGMOD Int. Conf., May 1987, pp. 260-269.
5. Gunther, G., and Buchemann, A., "Research Issues in Spatial Databases," ACM SIGMOD Record, Vol.19, No.4, Dec. 1988, pp.61-68.
6. Guttman, A., "R-trees: A Dynamic Index Structure for Spatial Searching," Proc. of ACM SIGMOD Int. Conf., June 1984, pp. 47-57.
7. Henrich, A., Six, H.W., and Widmayer, P., "The LSD tree: Spatial Access to Multidimensional

Point and Non-point Objects," Proc. of 15th VLDB Int. Conf., Aug. 1989, pp. 45-53.

8. Lu, H., and Ooi, B.C., "Spatial Indexing: Past and Future," IEEE Data Engineering, Vol.16, No.3, Sept. 1993, pp. 16-21.
9. Medeiros, C.B., and Pires, F., "Databases for GIS," Proc. of ACM SIGMOD Int. Conf., Mar. 1994, pp. 107-115.
10. Nievergelt, J., Hinterberger, H., and Sevcik, K.C., "The Grid File: An Adaptable, Symmetric Multikey File Structure," ACM Trans. on Database Systems, Vol.9, No.1, 1984, pp. 38-71.
11. Samet, H., *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1989.
12. Seeger, B., and Kriegel, H., "The Buddy-tree: An Efficient and Robust Access Method for Spatial Database Systems," Proc. of 16th VLDB Int. Conf., Aug. 1990, pp. 590-601.
13. Sellis, T., Roussopoulos, N., and Faloutsos, C., "The R+-tree: A Dynamic Index for Multi-dimensional Objects," Proc. of 13th VLDB Int. Conf., Sep. 1987, pp. 507-518.
14. 오병우, 이우영, 한기준, "GOOD : 지리 정보 시스템을 위한 데이타 관리기," 한국정보과학회, 데이타베이스 연구회, 데이타베이스 연구회지, Vol.10, No.2, 1994, pp. 3-22.
15. 오병우, 한기준, "EGG (EXODUS - GOOD - GRACE) : A Geographic Information System Using Object-Oriented Concepts," Proc. of the 17th International Cartographic Conference, Barcelona, Spain, Sept. 1995, pp. 2735-2739.
16. 오병우, 한기준, "지리 정보 시스템을 위한 사용자 인터페이스," 한국정보과학회, 정보과학회지, Vol.13, No.3, 1995, pp. 18-29.