# LR 트리 : 지도 일반화를 지원하는 공간 데이터를 위한 공간 인덱싱

권 준 희† · 윤 용 익††


## 요 약

지리정보시스템은 처리 속도의 향상과 비쥬얼라이징의 개선이 필요하다. 이를 위해서는 맵 일반화와 레벨별 상세화 개념이 요구된다. 기존의 공간 인덱싱은 지도 일반화를 지원하지 않거나 지원하더라도 모든 지도 일반화 연산을 제공하지 않는다는 문제점을 가진다. 본 논문에서는 이를 위해 모든 일반화 연산을 지원하는 새로운 인덱스 구조인 LR트리를 제안한다. 또한 LR트리를 검색, 삽입, 삭제하기 위한 알고리즘을 기술하고, 성능 분석을 수행한다. 성능 분석을 통해 제안된 인덱스 구조가 지도 일반화를 지원하는데 있어 다른 공간 인덱싱 기법보다 우수한 성능을 나타냄을 보인다.


# The LR-Tree : A spatial indexing of spatial data supporting map generalization

Joon-Hee Kwon† · Yong-Ik Yoon††


## ABSTRACT

GIS (Geographic Information Systems) need faster access and better visualization. For faster access and better visualization in GIS, map generalization and levels of detail are needed. Existing spatial indexing methods do not support map generalization. Also, a few existing spatial indexing methods supporting map generalization do not support all map generalization operations. We propose a new index structure, i.e. the LR-tree, supporting all map generalization operations. This paper presents algorithms for the searching and updating the LR-tree and the results of performance evaluation. Our index structure works better than other spatial indexing methods for map generalization.


**키워드 : 지리정보시스템(GIS), 공간 인덱싱(Spatial Indexing), 레벨별 상세화(Levels Of Detail), 맵 일반화(Map Generalization), R트리(R-tree), 축척(Scale)**

## 1. 서 론

최근의 지리정보시스템에 있어서 필요한 연구분야 중 하나는 처리 속도의 향상과 비쥬얼라이징의 개선이다. 이는 지리정보시스템에 있어서의 중요한 데이터인 공간 데이터가 본질적으로 대용량 데이터라는 점과 지도라는 특수한 데이터 형태로 데이터를 표현한다는데 있다.

공간 데이터는 전통적인 종이 지도를 수치 지도화한 것이다. 전통적인 종이 지도에는 축척 개념이 존재하여 대상 영역과 목적에 따라 상세화 정도를 다르게 한 지도가 존재한다. 이는 수치 지도로 변환된 후에도 마찬가지로 생각해 볼 수 있다. 또한 지리정보시스템에 있어 많이 사용되고 있는 확대, 축소 연산에 있어서도 객체를 확대할 때 객체를 단순히 크게 보여주는 것만으로는 높은 품질의 지도를 기대할 수 없다. 따라서, 각 객체가 확대될 때 각 객체는 보다 상세하게 보여져야 하고, 중요도가 떨어지는 객체가 보여져야만 한다[1]. 지도 제작에 있어서 이러한 문제는 오랫동안 논의되어 왔으며 지도 일반화(map generalization)문제로 알려져 왔다[2]. 즉, 지도 일반화를 통해 레벨별로 사용자가 인식할 수 있는 수준의 데이터만을 보임으로써 비쥬얼라이징의 향상을 기대할 수 있게 되는 것이다.

공간 데이터를 보다 효과적으로 처리하기 위한 연구 중 하나로는 공간 인덱싱 기법에 대한 연구를 들 수 있다. 대용량 공간 데이터 검색은 일반적으로 공간 인덱싱 기법을 이용해 이루어지게 되는데 알려져 있는 다양한 공간 인덱싱 기법이 존재한다.

그러나, 상세화 정도에 따른 공간데이터를 검색하고자 할 때 기존의 공간 인덱싱 기법을 사용하는 경우 문제가 발생하게 된다. 기존의 공간 인덱싱 기법을 사용할 경우의 접근

† 준 회 원 : 숙명여자대학교 대학원 컴퓨터 과학과
†† 종신회원 : 숙명여자대학교 정보학부 교수
논문접수 : 2001년 11월 17일, 심사완료 : 2002년 4월 1일

방법은 상세화 레벨별로 각각의 인덱싱 구조로 저장하거나, 혹은 한개의 인덱싱 구조에 저장한 후 이를 뷰 관점에서 다르게 가져와 사용하는 방법이 있다. 첫째, 각각의 인덱싱 구조로 저장하는 경우는 각 레벨별 데이터를 별개로 취급하여 통합된 데이터로 처리하기 힘들게 된다. 둘째, 한개의 인덱싱 구조에 저장한 후 뷰 관점에서 다르게 가져와 사용하는 방법은 뷰를 처리하는 방법이 어렵고 각 레벨별 데이터 검색에 따른 비효율성 문제가 발생한다. 또한 이 방법들은 모두 데이터베이스에 중복(redundancy)문제와, 수정 오퍼레이션을 통해 일관성 문제를 초래한다[3].

기존의 인덱싱 구조에 따르는 문제점을 극복하기 위해서는 레벨별 상세화를 지원하는 공간 데이터를 위한 별도의 공간 인덱싱 기법이 요구된다. 그러나, 이러한 기존 연구는 비교적 활발하지 못했으며 몇 가지 연구에 있어서도 지도 일반화 연산자 중 일부만을 지원하고 있어 지원되는 연산자 이외의 일반화 연산자를 사용하는 경우 사용이 불가능하다. 본 논문에서는 이러한 문제점을 극복하고자 어떠한 일반화 연산자를 거쳐 만들어진 레벨별 데이터라도 효율적으로 액세스할 수 있도록 하는 공간 인덱싱 기법인 LR트리를 제안하고자 한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 관련 연구를 살펴보고, 제 3장에서는 지도 일반화를 지원하는 공간데이터를 위한 공간 인덱싱 기법인 LR트리 모델과 알고리즘을 기술한다. 제 4장에서는 제안된 인덱싱 기법에 대한 적용 사례를 보이며, 제 5장에서 기존의 공간 인덱싱 기법과 제안된 LR트리 모델에 대한 성능을 분석하며 제 6장에서 결론을 맺는다.

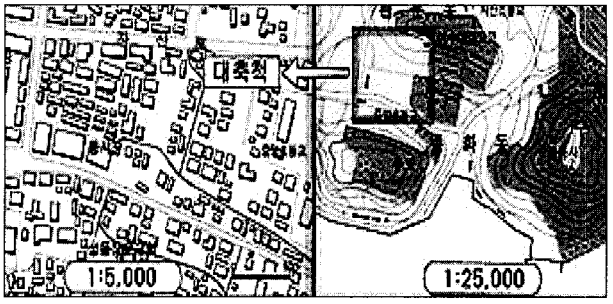
## 2. 관련 연구

### 2.1 지도 일반화

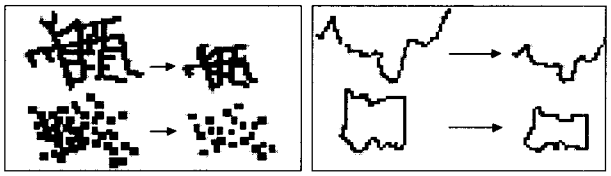
지도는 주어진 축척에 따라 그려진다. 동일한 정보라 하더라도 다른 축척에 따라 다르게 그려져야 하며 그 이유는 다음과 같다[4]. 첫째, 서로 다른 축척별로 데이터 표현 방법이 달라진다. 둘째, 선택된 축척에 따라 각 데이터들은 보여지거나 보여지지 않기 때문에 나타나거나 사라질 수 있으며 혹은 통합되거나 통합되지 않을 수도 있다. 셋째, 보여지는 데이터들의 모양이 더 간단하거나 더 상세하게 변할 수 있다. 넷째, 현재 축척에서 해당 정보가 필요하지 않다.

이를 위해 소축척에서 나타나는 지도가 대축척에서 나타나는 지도와 상세화된 내용에 있어 달라져야 하며 이러한 개념을 상위 레벨에서 하위 레벨 관점에서 보면 레벨별 상세화(Level Of Detail)로 정의되고, 하위 레벨에서 상위 레벨 관점에서 바라보면 지도 일반화(map generalization)로 정의된다. 또한 이러한 개념을 지원하기 위해서는 다중 표현(multiple representaion)이 요구된다[3-5].

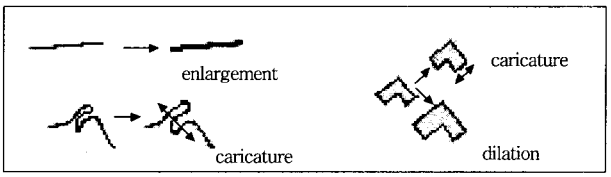
레벨별 상세화 혹은 지도 일반화의 개념은 지리정보시스템의 기능 중 확대 혹은 축소 연산시 나타나고, 이는 지능화된 확대(intelligent zoom)연산으로 알려져 있다[6]. 지능화된 확대 연산은 확대시 데이터의 크기만을 확대하는 것이 아니라 데이터를 상세화하는 것이며 (그림 2.1)과 같다.



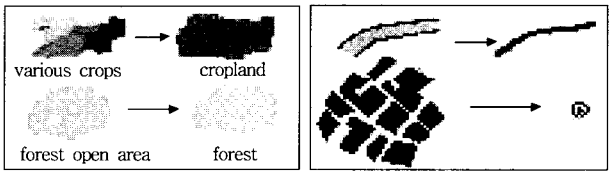

(그림 2.1) 지능화된 확대 연산



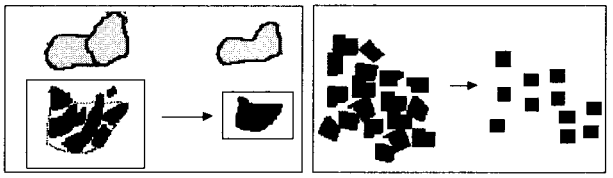
(a) Selection (b) Simplification



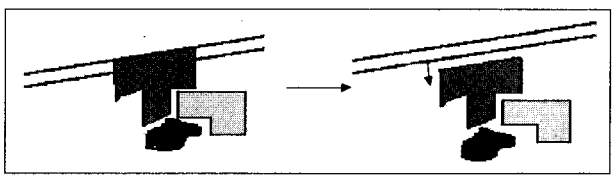

(c) Exaggeration




(d) Classification (e) symbolization



(f) Aggregation (g) Typification



(h) Anamorphose

(그림 2.2) 지도 일반화 연산자

이러한 지도 일반화의 목적은 인간의 해석 능력과 분석 능력에 적당하도록 상세화된 정도를 감소하고 객체의 밀도를 줄이는데 있다. 이러한 지도 일반화는 다음과 같은 연산자를 통해 이루어진다[7].

(a) 선택(selection) : 중요성에 기반해서 레벨별로 필요한 지도 피처를 추출한다.
(b) 단순화(simplification) : 선이나 면을 구성하는 점의 개수를 감소하거나 굴곡이 많던 라인을 보다 매끄럽게 표현한다. 예를 들면, 해안선 표현을 들 수 있다.
(c) 확장(exaggeration) : 지도 피처 중 의미 있는 특징을 강조해서 나타낸다.
(d) 분류(classification) : 여러개의 각 객체들을 공통적인 속성에 따라 그룹화 한다.
(e) 심볼화(symbolization) : 면이 선으로, 선이 점으로 변하는 차원의 변화를 의미한다.
(f) 통합(aggregation) : 인접해 있는 여러개의 객체가 하나의 객체로 표현된다.
(g) 대표화(typification) : 유사한 형태를 가진 많은 수의 별개의 객체들을 동일하고 전형적인 형태를 가지는 작은 수의 객체로 나타낸다.
(h) 형태변형(anamorphose) : 인접성 충돌 문제를 해결하기 위해 객체 집합을 변형하여 나타낸다.

### 2.2 공간 인덱싱 기법

공간 데이터베이스는 1차원 데이터 뿐 아니라 2차원 이상의 다차원 데이터베이스이다. 따라서 기존의 데이터 처리 방법은 적합하지 않으며 이를 위한 별도의 처리 방법이 있어야 한다. 이러한 연구 분야 중 하나가 공간 인덱싱이다.

공간 인덱싱 기법은 크게 트리 기반과 해쉬 기반 방법으로 분류될 수 있다[8-11]. 트리 기반 방법은 계층화된 검색 트리를 기반으로 하며 대표적으로 R트리[12-14]와 Quad트리[15, 16] 등이 있다. 해쉬 기반 방법은 그리드 파일을 기반으로 하며 대표적으로 그리드 파일[17], R파일[18] 등이 있다. 이 중 해쉬 기반 방법은 데이터 분포에 의존적이며 오버플로우 발생 및 이에 따라 효율이 저하되는 문제점을 가진다. 따라서 많은 공간 데이터베이스에서는 이러한 해쉬 기반 방법보다 트리 기반을 선호하고 있으며 이 중에서 R 트리 방법이 가장 많이 사용되고 있다. 본 논문에서는 이러한 점을 고려하여 R 트리를 기반으로 하고 있으며 따라서 본 절에서는 R트리를 중심으로 설명하도록 한다.

R트리는 트리 기반 방법 중 최소 사각형(MBR : Minimum Bounding Rectangle)에 기반한 방법으로 현재 가장 널리 쓰이고 있는 인덱싱 기법이다. 데이터 구조는 중간 노드와 리프 노드로 구성된 높이 균형 트리이다. 데이터 객체는 리프 노드에 저장되고 중간 노드는 하위 레벨에 해당하는 모든 사각형을 완전히 둘러싸는 사각형이다. M이 각 노드의 최대 엔트리 수라 할 때 R트리는 다음과 같은 속성을 가진다.

(a) 루트노드는 리프노드가 아니라면 적어도 2개의 자식을 가진다.
(b) 모든 중간 노드는 루트가 아니라면 M/2에서 M개 사이의 자식을 가진다.
(c) 모든 리프 노드는 루트가 아니라면 M/2에서 M개 사이의 자식을 가진다.
(d) 모든 리프는 같은 레벨에 나타난다.

### 2.3 제한된 지도 일반화를 지원하는 공간 인덱싱

지도 일반화를 지원하는 공간 데이터 구조로는, 자동화된 일반화를 지원하기 위한 자료 구조들이 많이 알려져 있으며 대부분 지도 일반화 중 단순화 연산자를 제공한다. 그러나 이러한 자료 구조는 데이터 삽입, 삭제에 따라 이를 동적으로 반영하지 못한다는 점에서 공간 데이터 처리에 한계점을 가진다. 따라서 이러한 문제점을 극복하기 위해서 지도 일반화를 지원하는 공간 인덱싱을 지원해야만 하며 이렇게 알려진 기법으로는 Reactive트리[1], Priority Rectangle파일[19]이 있다. 이 중, Reactive트리는 R트리에 기반하며 Priority Rectangle파일은 그리드 파일에 기반한다.

그러나, 이러한 공간 인덱싱은 다음과 같은 몇가지 문제점을 가진다. 첫째, 지도 일반화 연산자 중에서 선택 연산자 혹은 단순화 연산자라는 한가지 일반화 연산자만을 제공한다. 따라서 두 가지 이상의 일반화 연산자를 사용하고자 하는 경우에는 사용할 수 없다는 문제점을 가진다. 둘째, 레벨별 데이터간 링크가 존재하지 않는다. 즉, 한 객체에 대한 다중 표현으로 레벨별 데이터를 간주하는 것이 아니라 서로 별개의 데이터로 취급한다는 문제점을 가진다.

Reactive트리[1]는 R트리에 기반한 지도 일반화를 지원하는 공간 인덱싱 기법으로 R트리와 유사한 속성을 가지고 있으나 다음과 같은 몇가지 차이점이 있다. 첫째, 실객체가 리프 노드 뿐 아니라, 중간 노드에도 나타난다. 둘째, R트리의 노드 형태에 중요도 값이 추가된다. 세째, 모든 리프가 같은 레벨에 나타나지 않는다. 즉, 같은 레벨에 있는 모든 노드는 같은 중요도 값을 가진 엔트리를 포함한다. 보다 중요한 엔트리는 상위 레벨에 저장된다. Reactive트리의 중요한 개념은 객체당 중요도를 부여하고, 이렇게 부여된 값에 따라 중요도가 높은 객체가 상위 레벨에 위치하여 중요도가 높은 객체는 보다 빠른 검색이 가능하다는데 있다. 그러나, 선택 연산자만을 제공한다는 점과 서로 다른 레벨의 객체들간 겹침이 없어야 한다는 문제점을 가진다.

Priority Rectangle파일[19]은 그리드 파일 기법의 변형 중 하나인 R파일에 기반한 지도 일반화를 지원하는 공간 인덱싱 기법 중 하나이다. 이 기법은 R파일과 유사하나 우선순위를 부여했다는 점이 다르다. Priority Rectangle파일은 디렉토리 구조에 기반하며 이러한 디렉토리가 우선순위가 서로 다른 블록을 레퍼런스하는 방식으로 구성된다. 이 때, 우선순위가 높은 블록은 우선순위가 낮은 블록보다는 항상

일찍 발견된다는 속성을 가진다. Priority Rectangle파일의 중요한 개념은 객체당 중요도를 부여하고, 이렇게 부여된 값에 따라 중요도가 높은 객체가 순서적으로 앞의 버켓에 위치하여 중요도가 높은 객체는 보다 빠른 검색이 가능하다는데 있다. 그러나, 단순화 연산자만을 제공한다는 점과 공간 인덱싱 중 그리드 파일의 문제점을 그대로 가진다는 문제점을 가진다.

## 3. LR트리 모델

본 장에서는 모든 지도 일반화 연산자를 지원하는 새로운 공간 인덱싱 기법을 설명하고 이를 위한 검색, 삽입, 삭제 알고리즘을 소개한다. 제안된 공간 인덱싱 기법은 R트리에 기반하며 레벨화 되었다는 점에서 LR트리(Leveled R-tree)로 명명한다.

### 3.1 객체와 객체 다중표현

본 논문에서는 지리정보 객체에 대한 레벨별 상세화를 나타내는 다중 표현을 다루고 있다. 이에 대한 정의와 관계를 다음과 같이 정의한다.

**[정의 1]** *객체(Object)*

지리 정보의 다른 표현에 대한 궁극적인 한가지 실체(reality)를 '*객체(Object)*'라고 정의한다. *객체*는 지리 정보의 표현에 대한 동일한 한가지 사실(fact)을 의미하며 한 *객체*에 대한 다른 표현은 지리 정보 이외에는 동일한 정보를 공유한다. ■

**[정의 2]** *객체 다중표현(Multiple Representation of Object)*

지리 정보의 한 *객체*에 대한 여러개의 다른 지리정보 표현을 '*객체 다중표현(Multiple Representation of Object)*'이라고 정의한다. 지리 정보는 다른 추상화 레벨에 따라 다른 지리정보 표현을 가지며 이는 한 *객체*에 대한 뷰(view)로 볼 수 있다. ■

**[정의 3]** *객체* O와 *객체 다중 표현* MR과의 관계

O(Object)를 *객체*의 집합이라 하고, MR(Multiple Representation of Object)을 *객체 다중표현* 집합이라 할 때, O와 MR과의 관계는 다음과 같이 정의된다.

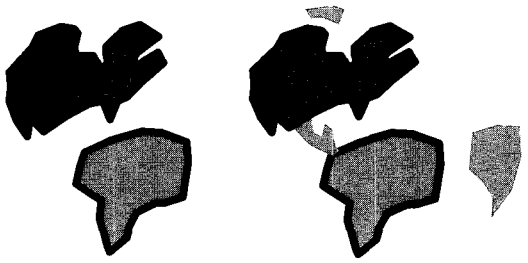
$$O = \bigcup_{i=0}^{n} O_i$$

여기서 n은 *객체* 개수이다.

$$O_i = \bigcup_{j=0}^{mr} MR_{ij}$$

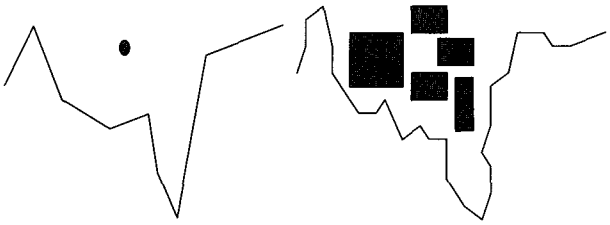
여기서 mr은 *객체* $O_i$에 대한 *객체 다중표현* 개수이다. ■

객체 다중표현의 상세화 유형은 크게 2가지로 나뉘어질 수 있다. 이를 위해 모든 일반화 연산자에 대해 상세화 레벨에 따른 변화를 살펴보면 다음과 같다. 첫째, 선택 연산자는 상세화된 데이터로부터 일부 데이터 객체를 추출하여 이를 간략화된 데이터로 생성하는 연산자이다. 즉, 선택 연산자를 사용하여 레벨별 데이터가 생성된 경우에는, 상세화된 데이터는 간략화된 데이터와 상세화된 데이터에서 새롭게 추가된 데이터로 이루어진다. (그림 3.1)에서 굵은 선으로 나타난 간략화된 레벨의 데이터가 상세화된 데이터에서 동일하게 나타남을 알 수 있다. 둘째, 선택 연산자 이외의 연산자, 즉, 단순화, 확장, 분류, 심볼화, 통합, 대표화, 형태변형 연산자는 상세화된 데이터로부터 간략하게 수정된 데이터를 생성하는 연산자로 요약될 수 있다. 즉, 선택 연산자 이외의 연산자를 사용하여 레벨별 데이터가 생성된 경우에는, 상세화된 데이터는 간략화된 데이터를 공유하지 않고 상세화된 데이터로만 구성된다. (그림 3.2)를 살펴보면 간략화된 데이터와 동일한 데이터가 상세화된 데이터에는 나타나지 않음을 알 수 있다. 이를 토대로 본 논문에서는 레벨별 객체 다중표현간 상세화 유형을 다음과 같이 정의한다.



(a) 간략화된 레벨 (b) 상세화된 레벨

(그림 3.1) 선택 연산자에 의한 데이터의 예



(a) 간략화된 레벨 (b) 상세화된 레벨

(그림 3.2) 선택 연산자 이외의 연산자에 의한 데이터의 예

**[정의 4]** *객체 다중표현 상세화 유형*

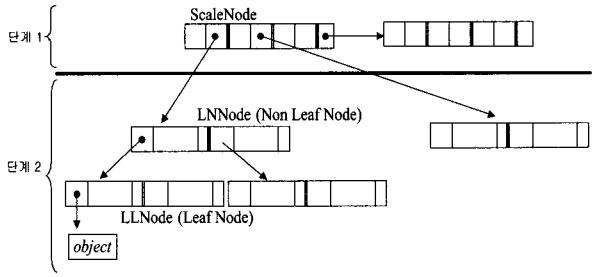
- *교체(replacement)* : 간략화된 레벨의 객체 다중표현이 상세화된 레벨의 객체 다중표현으로 교체되어 상세화된다.
- *추가(addition)* : 간략화된 레벨의 객체 다중표현을 추가해서 상세화된 레벨의 객체 다중표현이 상세화된다. ■

### 3.2 R트리 기반의 2단계 인덱스 구조

본 논문에서는 R트리 기반의 공간 인덱스를 사용한다.

이는 R트리가 가장 널리 사용되는 공간 인덱스이기 때문에 사용된 것으로 새로운 인덱스 구조를 만들어내는데 비해 비용면에서 보다 효율적이라는 장점을 가진다.

(그림 3.3)은 2단계 공간 인덱스 구조를 보여주고 있다. 1단계에서는 각 객체에 대한 객체 다중표현이 나타나는 축척값으로 구성되고, 2단계에서는 각 축척값에 따른 객체 다중표현을 나타내도록 트리가 구성된다. 따라서, 축척값에 따른 각 트리는 높이 균형 트리 속성을 가지고 전체 트리 측면에서는 높이 불균형 트리 속성을 가지게 된다.




(그림 3.3) 2단계 공간 인덱스 구조

### 3.3 LR트리의 특징

LR트리는 기존의 공간 인덱싱에 비해 레벨별 상세화를 지원하여 객체 다중표현을 나타낼 수 있다는 특징을 가진다. 한 레벨별로 상세화된 공간 데이터를 저장하기 위해서는 크게 2가지 방법이 존재한다. 첫 번째 방법은, 레벨별로 상세화된 공간 데이터를 레벨별로 각각 별개의 공간 데이터로 저장한다. 두 번째 방법은, 가장 상세화된 공간 데이터만을 저장하고 자동화된 일반화 연산자를 통해 레벨별 데이터를 얻는다. 이 중 두 번째 방법은 첫 번째 방법에 비해 저장 공간 측면에서 효율적이라는 장점을 가지지만 다음과 같은 단점을 가진다. 첫째, 자동화된 일반화 연산자에 대한 연구가 현재까지 완전하지 않아 선택과 단순화 등의 몇 가지 연산자로 연구가 집중된다. 둘째, 일반화 연산자를 거치게 됨으로써 처리 속도가 떨어질 수 있다는 문제점을 가진다. 따라서 본 논문에서는 첫 번째 접근방법을 사용하여 일반화 연산을 거쳐 미리 만들어진 데이터를 그 대상으로 한다. 여기에 비해 그 동안의 레벨별 상세화 데이터를 위한 공간 인덱싱 기법은 두 번째 접근방법을 기반으로 하고 있어 모든 일반화 연산자 중 선택과 단순화 연산자만을 다룬다는 문제점을 가진다.

이에 대해 레벨별 상세화 데이터 지원을 목적으로 하지 않는 기존의 공간 인덱싱 기법을 사용하는 경우는 접근 방법에 따라 다음과 같은 문제점을 가진다. 첫 번째 방법은 레벨별로 각각 다른 인덱스에 저장하는 방법에 의해 단계화된 데이터를 지원하는 것이다. 이러한 접근 방법은 동일한 데이터가 존재하는 경우 이를 각 레벨별로 중복 저장하는 문제가 발생한다. 두 번째 방법은 레벨별 데이터를 하나의 인덱스 구조에 저장하는 것이다. 이 방법은 데이터의 중복은 발생하지 않지만 각 레벨에 관계없이 모든 레벨의 데이터를 검색하는데 따른 검색 속도 저하의 문제가 발생한다. 본 논문에서는 2가지 접근 방법의 장점을 취해 별개로 각각 저장된 여러개의 인덱스를 한 개의 인덱스 구조로 통합한다.

LR트리의 인덱스 구조를 설명하면 다음과 같다. LR트리의 인덱스 구조는 크게 축척값을 나타내는 ScaleNode와 각 축척값에 따른 객체 다중표현을 나타내는 LNNode와 LLNode로 구성된다.

ScaleNode는 다음과 같은 형태로 이루어진다.

$$(entries,\ p_s)$$

여기서 $p_s$ 는 여러개의 ScaleNode가 존재하는 경우, 다음 ScaleNode를 포인트하는 포인터이며, *entries*는 다음과 같은 형태를 가진다.

$$(s,\ l,\ p_l)$$

여기서 $s$는 레벨을 결정하는 축척값이고, $l$은 레벨값을 의미하며 $p_l$은 LNNode 혹은 LLNode에 대한 포인터이다.

LNNode는 객체 다중표현에 대한 중간 노드를 의미하며 다음과 같은 형태를 가진다.

$$(p,\ RECT,\ l)$$

여기서 $p$는 LNNode의 자식 노드를 포인트하는 포인터이고, *RECT*는 하위 노드의 엔트리에 존재하는 모든 사각형을 포함하는 최소 사각형이다. $l$은 하위 노드의 엔트리에 존재하는 모든 레벨값을 나타낸다.

LLNode는 객체 다중표현에 대한 리프 노드를 의미하며 다음과 같은 형태를 가진다.

$$(id,\ RECT,\ l)$$

여기서 *id*는 객체 다중표현을 포인트하는 포인터이고, *RECT*는 *id*에 의해 포인트된 객체 다중표현을 포함하는 최소 사각형이다. $l$은 해당 객체가 나타나는 모든 레벨값을 나타낸다.

제안된 인덱스 구조의 속성은 다음과 같다.

(a) LNNode와 LLNode는 객체 다중표현에 대해 레벨값을 제외하고는 R트리와 동일한 속성을 가진다.

(b) ScaleNode를 제외한 모든 노드의 엔트리 수는 $M_l/2$와 $M_l$사이이며, ScaleNode의 엔트리 수는 1과 $M_S$사이이다. 이 때, $M_l$은 LNNode와 LLNode가 가질 수 있는 최대 엔트리 수이며, $M_s$는 ScaleNode가 가질 수 있는 최대 엔트리 수이다.

(c) ScaleNode의 축척값은 큰 값에서 작은 값 순으로 정렬되며, ScaleNode의 $p_s$에 포인트된 ScaleNode의 모든 엔트리 내 축척값은 현재 ScaleNode의 모든 축척값보다 늘 작은 값을 가진다.

(d) ScaleNode의 엔트리 내 $p_l$에 포인트된 트리는 높이

균형 트리이다.

(e) ScaleNode가 루트가 된 전체 트리는 높이 균형 트리가 아니다.

(f) 데이터는 ScaleNode 내각 엔트리 ($s$, $l$, $p_l$)에 대해, 해당 데이터가 나타나는 가장 작은 단계값 $l$에 해당하는 엔트리의 $p_l$에 의해 포인트된 노드에 의해 루트가 된 트리에 나타난다.

(g) LNNode와 LLNode의 각 엔트리 ($p$, $RECT$, $l$)에 대해, 단계값 $l$은 $p$에 의해 포인트된 모든 엔트리의 $l$에 대해 비트 OR 연산자를 수행한 결과이다.

### 3.4 LR트리 검색

R트리에서 레벨화된 데이터를 검색하기 위해서는 레벨별로 별도 구성된 R트리에 대해 각각 검색 알고리즘을 호출하거나, 해당 레벨에 대한 데이터를 추출하기 위해 검색 전 혹은 후에 별도의 레벨별 데이터 정보나 검색이 필요하다. 그러나, LR트리에 있어서는 레벨화된 데이터를 검색하기 위해 한가지 인덱싱에 대해 별도의 전처리나 후처리 필요 없이 데이터를 검색할 수 있다. 따라서 레벨화된 데이터를 검색할 때 R트리에 비해 사용이 용이하다는 장점을 가진다.

LR트리의 검색 알고리즘은 R트리와 유사하다. 즉, 우선 검색 대상이 되는 레벨값과 동일한 레벨값을 가지는 노드를 ScaleNode로부터 검색한다. 다음으로, 이렇게 검색된 각 트리에서 검색 대상이 되는 레벨값과 동일한 레벨값을 가지는 모든 노드를 검색하여 최종적으로 해당 데이터가 발견된다. 이를 나타내면 (Algorithm 1)과 (Algorithm 1.1)과 같다.

```
Algorithm 1 : Search (LR, W, S)
/*
Input
  LR : LR tree rooted at node ScaleNode
  W : search window W
  L : search scale S
Output
  All data objects in scale S overlapping W
*/
1.  result = {}
2.  for (each ScaleNode of LR)
3.    for (each entry(s, l, pl) of ScaleNode)
4.      if (s >= S)
5.        result = result + SearchLevelTree(LT, W, l), where LT
          is the node pointed by pl
6.      else goto line 8
7.      endif
8.    endfor
9.  endfor
10. return result
End of Algorithm1.
```

(Algorithm 1) Search (LR, W, S)

```
Algorithm 1.1 : SearchLeveltree (LT, W, L)
/*
Input
  LT : LR tree rooted at node LNNode or LLNode
  W : search window
  L : search level value L
Output
  All data objects in level L overlapping W
*/
1.  if (LT is not a leaf node)
2.    for (each entry ( p, RECT, l ) of LT)
3.      if ( bitwiseAND (l, L ) = L)
4.        if (RECT overlaps W)
5.          SearchLRTree(CHILD, W∩Rect, L), where CHILD is
                        the node pointed by p
6.        endif
7.      endif
8.    endfor
9.  else
10.     for (each entry (p, RECT, l) of LT)
11.       if ( bitwiseAND(l, L) = L) return all objects
                                    overlapping W
12.       endif
13.     endfor
14. endif
End of Algorithm1.1.
```

(Algorithm 1.1) SearchLeveltree (LT, W, L)

### 3.5 LR트리 삽입

LR트리 내에 객체와 객체 다중표현을 삽입하는 알고리즘은 (Algorithm 2), (Algorithm 2.1), (Algorithm 2.2)와 같다. 삽입 방법을 간략히 기술하면 다음과 같이 2가지 경우로 요약된다. 첫 번째 경우는, 간략화된 데이터로부터 데이터가 수정되는 경우이다. 이는 레벨값을 처리하는 부분을 제외하고는 R트리와 동일하다. 두 번째 경우는, 간략화된 데이터를 공유하여 사용하는 경우이다. 이 때는 해당 데이터를 소유하고 있는 가장 간략화된 데이터의 레벨값에 해당 단계의 레벨값을 비트OR 연산자에 의해 추가한다.

```
Algorithm 2 : Insert (LR, E, S, AS)
/*
Input
  LR : LRtree rooted at node ScaleNode
  E : index entry E (id, RECT)
  S : scale
  AS : smaller scale if data is added from smaller scale,
       otherwise UNDEFINED
Output
  The new LR tree that results after the insertion
*/
1.  if (CS = UNDEFINED)
2.    if (an entry(s, l, pl) does not exist in entry of ScaleNodes
                        of LR, where s = S)
3.      Insert1stStepTree(LR, S)
4.    endif
5.    find an entry (l, pl) in ScaleNodes of LR, where s = S
6.    Insert2ndStepTree(LT, E, L), where LT is the node pointed
                        by pl
7.  else
8.    if (an entry(s, l, pl) does not exist in entry of ScaleNodes
        of LR, where s = AS) return failure
9.    endif
```

10. find an entry ($s$, $l$, $p_l$) in ScaleNodes of LR, where $s$ = AS
11. search a leaf node LN in which to place a new index entry E from tree pointed by $p_l$
12. perform bitwiseOR ($l$, CL) calcuation for $l$ of entries ($p$, *RECT*, $l$) of LN
13. propagate changes by bitwiseOR calculation, ascending from a leaf node LN to the root node of LR
14. **endif**

**End of Algorithm 2.**

(Algorithm 2) Insert (LR, E, S, AS)

**Algorithm 2.1 : Insert1stStepTree** (LR, S)
/*
*Input*
*LR : LRtree rooted at node ScaleNode*
*S : scale*
*Output*
*The LR tree that results after the insertion of scale S*
*/
1. create a new LLNode, NewLT
2. create level value L, where L is the result of shift left 1 from the largest level value
3. insert an entry (S, L, $p_l$) in ScaleNodes of LR, where $p_l$ is pointing the NewLT and each entry is sorted from small scale to large scale

**End of Algorithm 2.1**

(Algorithm 2.1) Insert1stStepTree (LR, S)

**Algorithm 2.2 : Insert2ndStepTree** (LR, E, L)
/*
*Input*
*LR : LRtree rooted at node ScaleNode*
*E : index entry (id, RECT)*
*L : level value*
*Output*
*The new Leveled R-tree that results after the insertion of E in level L*
*/
1. find a leaf node LN in which to place a new index entry E
2. **if** (LN has room for another entry) insert (E, L) in a LN
3. **else** SplitNode(LN, E, L)
4. **endif**
5. AdjustLRTree(LN, LR), where the method is similar to algorithm 'AdjustTree' of R tree. The difference with AdjustTree algorithm is propagating changes by bitwiseOR($l$, L) calculation ascending from leaf nodes to LR, where $l$ is an element of entries ($p$, *RECT*, $l$) of LLNode and LNNode.
6. **if** (root is splited)
   create a new root whose children are the two resulting nodes, propagating changes by bitwiseOR calculation for $l$ of entries($p$, *RECT*, $l$) of two resulting nodes.
7. **endif**

**End of Algorithm 2.2.**

(Algorithm 2.2) Insert2ndStepTree (LR, E, L)

### 3.6 LR트리 삭제

LR트리로부터 데이터를 삭제하는 방법은 기존의 R트리와 달리 2가지 경우로 나뉜다. 이는 객체와 객체 다중표현이라는 2가지 유형의 데이터가 존재하기 때문이다.

이 중 객체를 삭제하는 경우는 DeleteObject 알고리즘으로 나타난다. DeleteObject 알고리즘은 객체와 이에 해당하는 모든 객체 다중표현을 LR트리로부터 삭제한다. 객체 다중표현을 삭제하는 경우는 Delete알고리즘으로 나타난다. Delete 알고리즘은 객체 다중표현 중 삭제하고자 하는 객체 다중표현을 입력 인자로 하여 원하는 데이터가 삭제된다.

**Algorithm 3 : DeleteObject(LR, E)**
/*
*Input*
*LR : LRtree rooted at node ScaleNode*
*E : index entry (id, RECT)*
*Output*
*The new LR tree that results after deletion*
*/
1. find leaf node LN containing E in LR
2. remove E from LN
3. CondenseLRTree(LN, LR), where the method is similar to algorithm 'CondenseTree' of R-tree.
   The difference with CondenseTree algorithm is propagating changes by bitwiseOR($l$, L) calculation ascending from a leaf node LN to the node LR, where $l$ is an element of entries($p$, *RECT*, $l$) of the LLNode and LNNode.
4. **if** (root node of an Leveled R-tree has only one child)
5. make the child the new root node of an Leveled R-tree
6. **endif**

**End of Algorithm 3.**

(Algorithm 3) DeleteObject (LR, E)

**Algorithm 4 : Delete(LR, E, S)**
/*
*Input*
*LR : Leveled R-tree rooted at node*
*E : index entry (id, RECT)*
*S : scale*
*Output*
*The new LR tree that results after the deletion*
*/
1. find a entry ($s$, $l$, $p_l$) in ScaleNodes of LR, where $s$ = S
2. find leaf node LN containing E in LR
3. perfom bitwiseAND ($l$, bitwiseNOT(L)) calculation for entry (*id*, *RECT*, $l$) of LN, where e = E
4. propagate changes of level value ascending from LN to LR

**End of Algorithm 4.**

(Algorithm 4) Delete (LR, E, S)

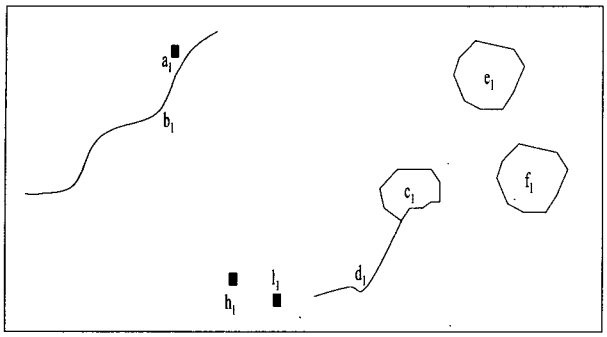
## 4. LR트리 적용 사례

본 장에서는 3장에서 설명한 LR트리 모델과 알고리즘을 바탕으로 2개의 레벨로 구성된 입력 데이터를 예로 들어 기존의 인덱스 구조를 사용한 경우와 LR트리를 사용한 경우에 대한 적용 사례를 설명한다.

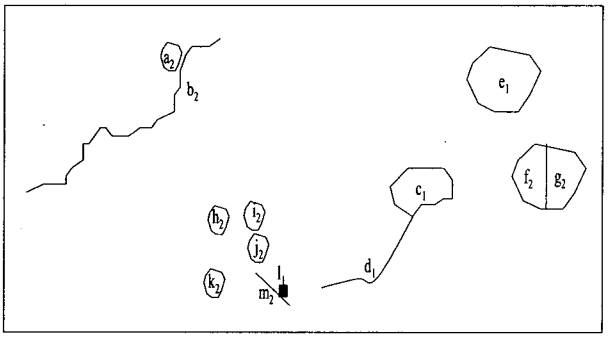
### 4.1 레벨별 입력 데이터

입력 데이터는 2개의 레벨을 가진 데이터로 가정한다. 즉, 1 : 50,000 축척 지도, 1 : 10,000 축척 지도로 나뉘어지며 이를 (그림 4.1)로 나타내었다. 본 논문에서 사용한 입력

데이터는 [20]에서 제시한 예제를 수정하여 사용하였다.

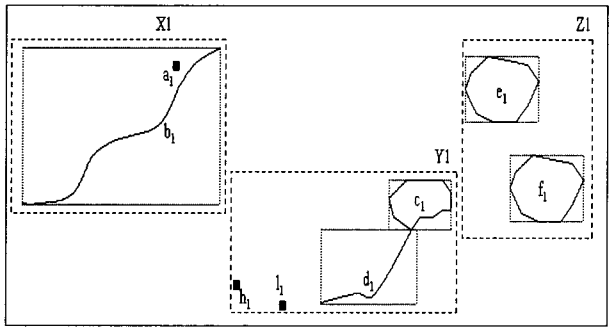


(a) 레벨 1 (1 : 50,000)

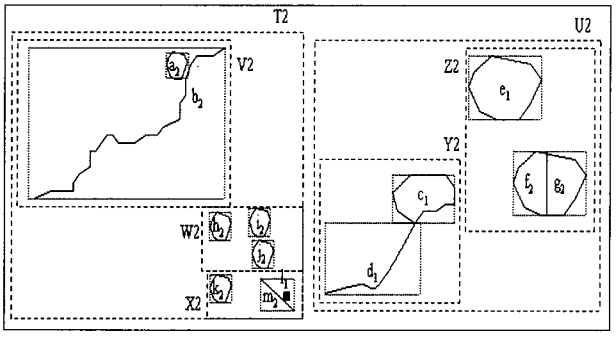


(b) 레벨 2 (1 : 10,000)

(그림 4.1) 레벨별 데이터의 예



(a) 레벨 1



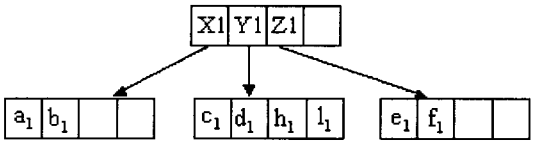
(b) 레벨 2

(그림 4.2) (그림 4.1)에 대한 레벨별 R트리로 만들어진 사각형

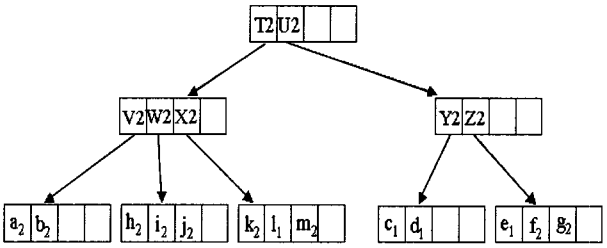
(그림 4.1)을 살펴보면 레벨 1에 존재하는 데이터 $a_1$은 레벨 2에서는 데이터 $a_2$로 수정된다. 이는 지도 일반화 연산자 중 심볼화 연산자의 결과이다. 그러나, 이러한 데이터는 지도 일반화를 지원하는 기존의 공간 인덱스 즉, Reactive 트리와 Priority Rectangle파일로는 표현이 불가능하다. 그러므로 여기서는 지도 일반화 지원을 목적으로 하지 않은 기존의 공간 인덱스와 LR 트리를 (그림 4.1)에 의해 적용한 예제를 통해 비교해 보도록 한다.

### 4.2 레벨별 R트리

레벨화된 데이터를 지원하는데 있어 레벨화된 지리정보 데이터 지원을 목적으로 하지 않는 기존의 공간 인덱스 기법을 사용하는 방법 중 각각 다른 인덱스에 저장하는 방법에 의해 레벨화된 데이터를 지원하는 경우에 대해 (그림 4.1)의 예제를 적용하면 (그림 4.2)와 (그림 4.3)과 같다. 이를 통해, 레벨 1과 레벨 2에 존재하는 객체 $c_1$,$d_1$,$e_1$,$l_1$이 각각 별개의 인덱스 구조에 별도로 중복 저장되어 있음을 알 수 있다.




(a) 레벨 1

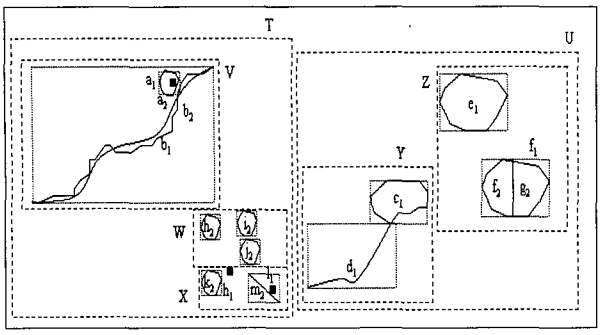



(b) 레벨 2

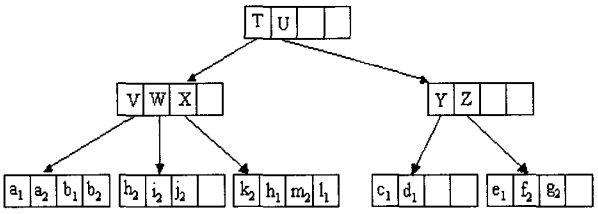
(그림 4.3) (그림 4.2)에 대한 레벨별 R트리

### 4.3 한 개의 R트리

레벨화된 데이터를 지원하는데 있어 레벨화된 지리정보 데이터 지원을 목적으로 하지 않는 기존의 공간 인덱스 기법을 사용하는 방법 중 하나의 인덱스 구조에 저장하는 방법에 의해 레벨화된 데이터를 지원하는 경우에 대해 (그림 4.1)의 예제를 적용하면 (그림 4.4)와 (그림 4.5)와 같다. 이때, 사각형 U 내에 있는 레벨 2의 모든 객체를 검색한다고 가정하면, 레벨 2에 있는 객체를 포인트하는 엔트리를 가지고 있지 않은 Y에 의해 포인트된 노드도 방문하고 있음을 알 수 있다. 이로부터, 각 레벨에 관계없이 모든 레벨의 데이터를 검색하는데 따른 검색 속도 저하의 문제가 발생함을 이해할 수 있다.

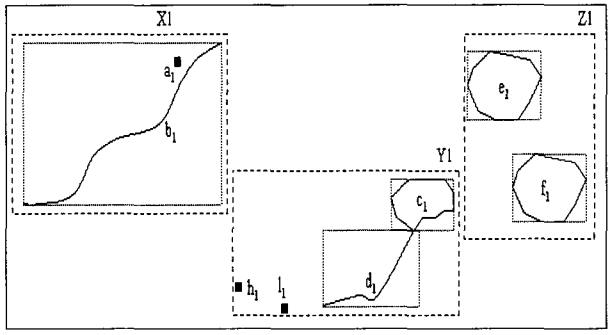



(그림 4.4) (그림 4.1)에 대한 한 개의 R트리로 만들어진 사각형

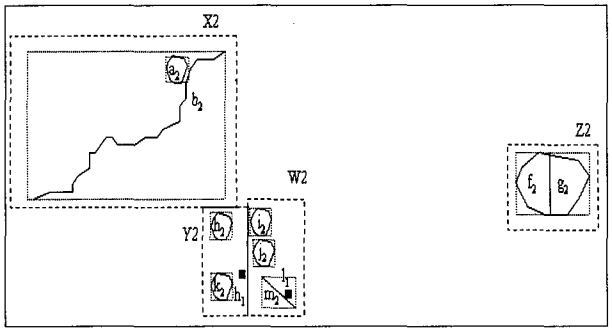



(그림 4.5) (그림 4.4)에 대한 한 개의 R트리

4.4 LR트리



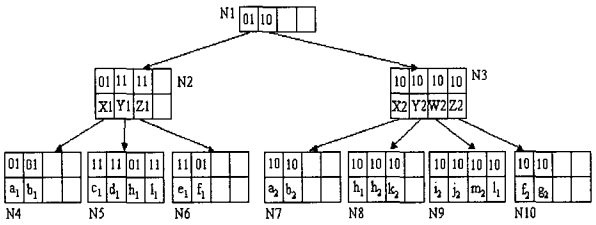

(a) 레벨 1




(b) 레벨 2

(그림 4.6) (그림 4.1)에 대한 LR트리로 만들어진 사각형

레벨화된 데이터를 지원하는데 있어 제안된 LR트리에 대해 (그림 4.1)의 예제를 적용하면 (그림 4.6)과 (그림 4.7)과 같다. (그림 4.6)과 (그림 4.7)을 통해 레벨 1과 레벨 2에 존재하는 데이터 $c_1$,$d_1$,$e_1$,$l_1$이 레벨 1에만 저장됨을 알 수 있다. 이를 위해 레벨 1의 노드 N2, N5, N6에 단계값 '11'이 기존의 R트리와 달리 추가되었다. 단계값 '11'은 단계 1의 단계값 '01'과 단계 2의 단계값 '10'의 비트OR(bitwise-OR) 연산의 결과로 단계값 1과 단계값 2에서 모두 사용 가능한 노드임을 나타낸다. 따라서, 단계 2에 해당하는 데이터를 검색하고자 할 때에는 단계 1의 노드 중 단계 2의 단계값 '11'을 가지고 있는 노드만을 방문함으로써 데이터의 중복 저장 없이도 검색 속도의 향상을 기대할 수 있게 된다.




(그림 4.7) (그림 4.6)에 대한 LR트리

## 5. 성능 평가

본 장에서는 LR트리의 성능을 평가한다. 이를 위해 메모리 용량과 검색 속도 측면에서 지도 일반화를 지원하지 않는 공간 인덱싱 기법인 R트리와 비교한다. 이는 기존의 공간 인덱싱 중 모든 일반화 연산자를 제공하는 공간 인덱싱 기법이 존재하지 않아 비교가 불가능하므로 지도 일반화를 지원하지 않는 공간 인덱싱으로 같은 효과를 만들어 내기 위해서이다. 그러나, R트리를 사용할 경우는 레벨별 데이터 간 데이터 중복 문제가 발생하여 수정 오퍼레이션시 데이터 일관성 문제가 발생할 수 있다는 측면에서 R트리를 사용하여 지도 일반화를 지원한다는 점은 문제점을 가진다.

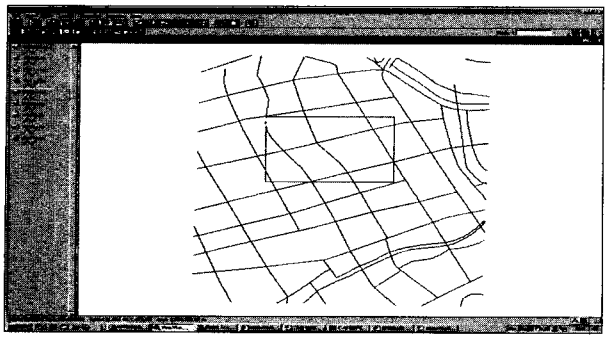
지도 일반화를 지원하지 않는 공간 인덱싱에 있어서 지도 일반화를 지원하는 효과를 만들기 위해서는 다음과 같은 2가지 경우와의 비교를 통해 성능을 평가하여 LR트리의 성능 우수성을 증명하도록 한다.

(a) 레벨별로 각각의 R트리 사용
(b) 한 개의 R트리 사용

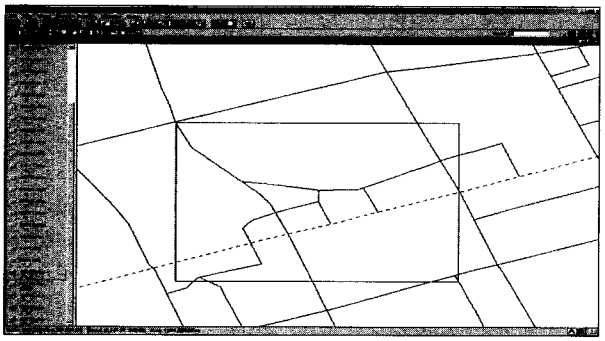
본 논문에서 제시한 LR트리에 대한 구현 환경은 다음과 같다. 구현 언어는 윈도우즈에서 구동되는 Cygwin 기반 하에 GNU C++을 사용했다. Cygwin은 래드 햇(Red Hat)에 의해 개발된 Windows 운영체제에서의 Unix환경이다. Cygwin은 표준 Unix와 Linux 환경을 그대로 따르므로 Unix와 Linux 환경으로의 이식이 원활히 이루어져 운영체제에 관계없이 동일한 개발이 가능하다는 장점을 가진다. 실험을 위해 본 논문에서는 R트리와 LR트리 모두를 구현하였으며, 구현된 프로그램은 Windows 2000 운영체제에서 펜티엄 III 800EB와 256MB 메인 메모리 환경 하에서 실험하였다.

### 5.1 실험 데이터

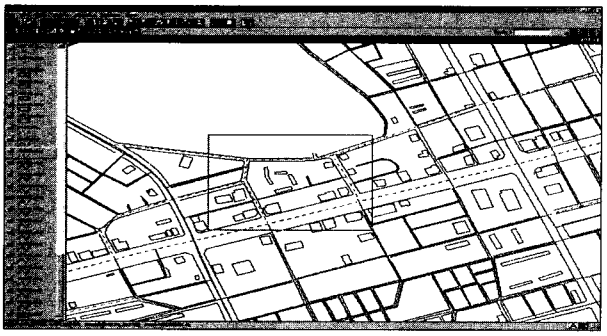
본 논문에서 사용된 데이터는 서울시 강남구 지역을 대상으로 4개의 축척에 대한 약 10만 건의 방대한 실제 데이터를 대상으로 하였다. 각 레벨별 데이터를 디스플레이한 결과는 (그림 5.1)과 같다. <표 5.1>은 이러한 실제 데이터를 요약하여 설명한다. <표 5.1>에서는 데이터 객체의 수 및 간략화된 레벨의 데이터와 상세화된 레벨의 데이터간 교체되거나 추가된 데이터의 수를 보여준다. 본 논문에서는 1 : 250,000 축척을 레벨 1, 1 : 50,000 축척을 레벨 2, 1 : 5,000축척을 레벨 3, 1 : 1000 축척을 레벨 4로 하여 실험을 수행하였다.



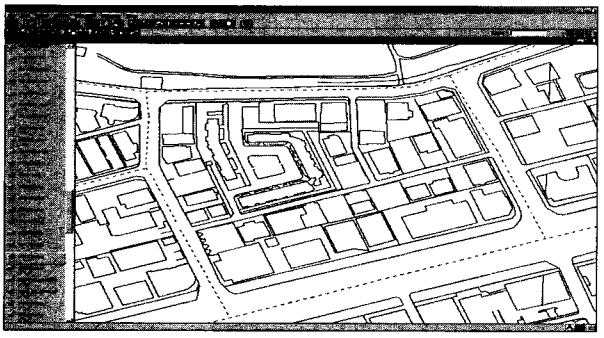
(a) 1 : 250,000 축척



(b) 1 : 50,000 축척



(c) 1 : 5,000 축척



(d) 1 : 1000 축척

(그림 5.1) 실험 데이터

〈표 5.1〉 실험 데이터 요약

| 단계값 | 축 척 | 전체 객체 수 | 간략화된 단계의 객체로부터 교체된 객체수 | 간략화된 단계의 객체가 추가된 객체수 | |
|---|---|---|---|---|---|
| | | | | 객체의 최소 단계값 | 객체수 |
| 1 | 1 : 250000 | 37 | 37 | 0 | |
| 2 | 1 : 50000 | 393 | 390 | 1 | 3 |
| 3 | 1 : 5000 | 17265 | 17228 | 1 | 3 |
| | | | | 2 | 34 |
| 4 | 1 : 1000 | 80420 | 74982 | 1 | 3 |
| | | | | 2 | 34 |
| | | | | 3 | 5401 |

### 5.2 실험 질의문

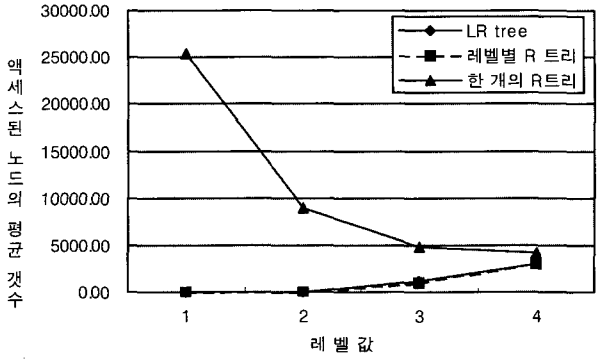
LR트리의 성능을 보다 객관적으로 평가하기 위해, 본 논문에서는 4개의 다른 윈도우 질의 영역에 대해 많은 수의 윈도우 질의를 랜덤하게 생성하였다. 레벨별 데이터를 검색하는데 있어서, 각 객체는 일정 윈도우 크기 범위에서만 해당 레벨의 객체가 나타난다. 이 때 윈도우 크기와 레벨값의 관계를 살펴보면, 윈도우 크기가 커지면 레벨값은 적어지게 된다. 본 논문에서는 다양한 크기와 위치의 윈도우 질의로 실험을 수행하기 위해, 4개의 각각 다른 레벨에 대해 10,000개씩의 윈도우 질의를 생성하였다.

### 5.3 실험 결과

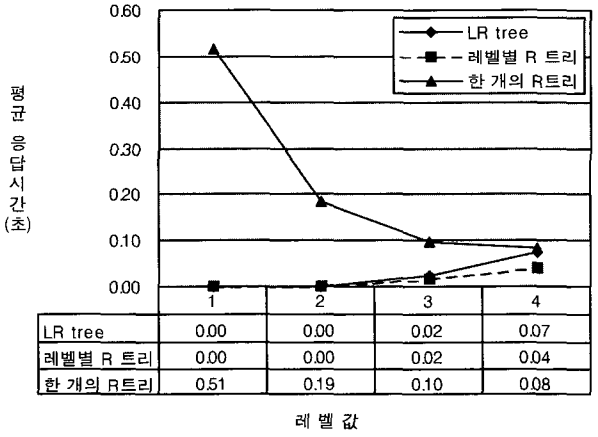
5.2절의 실험 모델에 따라 R트리와 제안된 인덱스 기법 각각에 대해 검색 속도의 평균값과 총 메모리 용량을 계산하였다. 이에 대한 결과는 (그림 5.2), (그림 5.3), 그리고 (그림 5.4)와 같다.

검색 속도를 비교하기 위해 본 논문에서는 액세스된 노드의 수와 응답 속도를 측정하였다. (그림 5.2)에서는 레벨별로 액세스된 노드 수의 변화를 꺾은선 그래프로 각 인덱스 구조에 따라 비교하여 나타내었다. 이를 통해 다음과 같은 결과를 얻는다. 첫째, LR트리는 하나의 R트리로 구축된 경우보다 늘 성능이 우수하다. 둘째, LR트리는 레벨별로 별도 구축된 R트리의 집합과 성능면에서 거의 동일하다. (그림 5.3)은 레벨별로 실험 질의문에 의해 소요된 평균 응답 속도를 보여준다. (그림 5.2)와의 차이점은 LR트리의 응답 속도가 레벨별로 별도 구축된 R트리의 응답 속도와 약간의 차이를 보여준다는 점이다. 이는 LR트리에서 레벨값을 찾는데 걸리는 시간에 해당한다. 그러나 그 차이는 0.03초에

불과하므로 무시될 수 있다. (그림 5.2)와 (그림 5.3)의 레벨 값 변화에 따른 결과를 살펴보면 다음과 같다. LR트리와 레벨별 R트리는 하나의 R트리와 비교할 때 레벨값이 작을수록 액세스된 노드의 평균 개수와 응답시간이 감소하여 모든 레벨에 있어 상대적으로 고른 성능을 유지하고 있음을 알 수 있다. 이는 레벨별 데이터 검색에 있어 LR트리와 레벨별 R트리는 전체 데이터 개수와 무관하게 검색하고자 하는 레벨의 데이터만을 검색하는데 비해 하나의 R트리는 모든 레벨의 데이터를 검색하기 때문으로 해석될 수 있다.



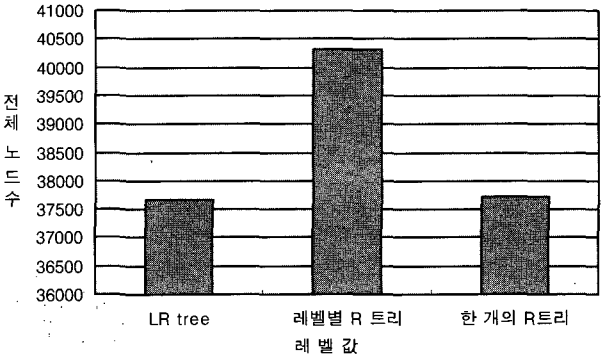

(그림 5.2) 실험 질의문에 대해 액세스된 노드의 평균 개수 비교




| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| LR tree | 0.00 | 0.00 | 0.02 | 0.07 |
| 레벨별 R 트리 | 0.00 | 0.00 | 0.02 | 0.04 |
| 한 개의 R트리 | 0.51 | 0.19 | 0.10 | 0.08 |

(그림 5.3) 실험 질의문에 대한 평균 응답 속도 비교

(그림 5.4)는 메모리 용량에 대한 성능 평가 결과를 보여준다. 메모리 용량을 비교하기 위해 본 논문에서는 각 인덱스에서 소요된 전체 노드 수를 측정하였다. 이를 위해, 각 인덱스에 데이터를 삽입할 때마다 액세스된 노드 수를 계산하였다. 이렇게 계산된 결과는 각 인덱스 구조별로 소요된 전체 노드 수를 막대 그래프로 표시하였다. 이를 통해 다음과 같은 결과를 얻는다. 첫째, LR트리는 다른 인덱스 구조 중 소요되는 메모리 용량이 가장 작다. 둘째, 검색 속도 측면에서 우수한 성능을 보였던 단계별 R트리의 집합은 최악의 결과를 얻었다. 이는 단계별 데이터간 중복되는 데이터를 중복하여 저장하는데 따른 결과이다. 이에 비해 LR트리는 중복되는 데이터를 통합관리하여 이러한 문제를 해결하였다.




(그림 5.4) 각 인덱스에서 소요된 전체 노드 수 비교

(그림 5.2), (그림 5.3), (그림 5.4)를 통해 LR트리는 알려진 기존의 공간 인덱스 기법에 비해 검색 속도와 메모리 용량 모두에서 우월함을 알 수 있다. 따라서, 지도 일반화를 위한 공간 인덱스 구조로서 매우 효율적인 인덱스 구조임을 알 수 있다.

## 6. 결 론

기존의 공간 인덱싱 기법은 지도 일반화를 지원하지 않거나, 지원하더라도 많은 제한이 있었다. 즉, 기존의 공간 인덱싱 기법을 사용하여 지도 일반화를 지원하기 위해서는 데이터 중복과 이에 따른 일관성 문제가 발생하게 되며, 지도 일반화를 지원하는 공간 인덱싱을 사용하는 경우는 모든 지도 일반화 연산자를 나타낼 수 없다는 문제점을 가진다. 그러나, 지도 일반화 기법은 지도의 비쥬얼라이징이나 효율적인 검색을 위해 그 필요성이 증가하고 있어 새로운 처리 기법이 요구되어 왔다. 이러한 요구사항에 따라 본 논문에서는 모든 일반화 연산자를 지원하는 공간 인덱싱 기법인 LR트리를 제안하였다.

LR트리는 기존 R트리 인덱스 구조의 확장으로 2단계 인덱스 구조 방식을 가진다. 즉, 1단계에서는 각 객체에 대한 객체 다중표현이 나타나는 축척값으로 구성되고, 2단계에서는 각 축척값에 따른 객체 다중표현을 나타내도록 트리가 구성된다. 본 논문에서는 제안된 새로운 공간 인덱싱 기법인 LR트리의 구조 및 검색, 삽입, 삭제 알고리즘을 보이고 이를 예제를 통해 적용해 보였다. 또한, 제안된 인덱싱 기법을 지도 일반화를 지원하지 않는 R트리로 같은 효과를 내도록 하였을 때와 성능 모델을 통해 비교하여 메모리 용량과 검색 속도 측면에서 모두 우월함을 보였다. 향후 연구과제로는 객체 다중표현간 관계(relation) 측면에서의 일관성을 고려하지 않은 현재의 모델로부터 관계 측면에서의

일관성을 고려하는 방법에 대한 연구가 필요하다.

## 참 고 문 헌


[1] P. V. Oosterom, "The Reactive-Tree : A Storage Structure for a Seamless, Scaless Geographic Database," Proceedings of Auto-Carto, 10, pp.393-407, 1991.

[2] "Automatic generalization on geographic data," project report, VBB Viak, 1997.

[3] C. Davis, A. Laender, "Multiple Representations in GIS : Materialization, Geometric, and Spatial Analysis Operations," Proceedings of the 7th International Symposium on Advances in GIS, Kansas City, MO, pp.60-65, 1999.

[4] S. Spaccapietra, C. Paren, and C. Vangenot, "GIS Databases : From Multiscale to MultiRepresentation," Proceedings of the 4th International Symposium, SARA-2000, Horseshoe Bay, Texas, USA, pp.26-29, 2000.

[5] M. Garland, "Multiresolution Modeling : Survey&Future Opportunities," State of the Art Report, Eurographics, pp.111-131, 1999.

[6] S. Timpf, "Cartographic objects in a multi-scale data structure," Geographic Information Research : Bridging the Atlantic. 1(1), London, Taylor and Francis, pp.224-234, 1997.

[7] A. Ruas, "Multiple Representation and Generalization," Lecture Notes for, "sommarkurs : kartografi," 1995.

[8] V. Gaede, O. Gunther, "Multidimensional Access Methods," ACM Computing Surveys, 30(2), pp.170-231, 1998.

[9] O. Gunther, J. Bilmes, "Tree-Based Access Methods for Spatial Databases : Implementation and Performance Evaluation," IEEE Transactions on Knowledge and Data Engineering, 3(3), pp.342-356, 1991.

[10] H. Lu, B. C. Ooi, "Spatial Indexing : Past and Future," IEEE Data Engineering Bulletin. 16(3), pp.16-21, 1993.

[11] H. Samet, "The Design and Analysis of Spatial Data Structures," Addison-Wesley, Reading, MA. 1990.

[12] A. Guttman, "R-trees : A Dynamic Index Structure for Spatial Searching," Proceedings of the ACM SIGMOD International Conference on Management of Data, Boston, MA., pp.47-54, 1984.

[13] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-tree : A Dynamic Index for Multidimensional Objects," Proceedings of the 13th International Conference on VLDB, Brighton, England, pp.507-518, 1987.

[14] N. Beckmann, H. P. Kriegel, R. Schneider, R, and B. Seeger, "The R* tree : An Efficient and Robust Access Method for Points and Rectangles," Proceedings of ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, pp.322-331, 1990.

[15] H. Samet, "The Quadtree and Related Hierarchical Data Structure," ACM Computing Surveys, 16(2), pp.187-260, 1984.

[16] H. Samet, R. E. Webber, "Storing a Collection of Polygons using Quadtrees," ACM Transactions. Graph, 4(3), pp.182-222, 1985.

[17] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, "The Grid file : An Adaptable, Symmetric Multikey File Structure," ACM Transactions on Database Systems. 9(1), pp.38-71, 1984.

[18] A. Hutflesz, H-Werner Six, and P. Widmayer, "The R-file : An Efficient Access Structure for Proximity Queries," Proceedings of IEEE 6th International Conference on Data Engineering, Los Angeles, CA, pp.372-379, 1990.

[19] B. Becker, P. Widmayer, "Spatial Priority Search : An Access Technique for Scaless Maps," Proceedings of ACM SIGMOD, Denver, Colorado, pp.128-137, 1991.

[20] B. Michela, M. J. Egenhofer, "Progressive Vector Transmission," Proceedings of ACM 7th International Symposium on Advances in Geographic Information Systems, Kansas City, USA, pp.152-157, Nov., 1999.



**권 준 희**

e-mail : kwonjh24@hotmail.com

1992년 숙명여자대학교 전산학과 졸업 (학사)

1994년 숙명여자대학교 대학원 전산학과 졸업(석사)

1999년 숙명여자대학교 대학원 컴퓨터과학과 박사과정 수료

1994년~현재 쌍용정보통신 GIS기술팀 과장

2000년 전자계산조직응용기술사

관심분야 : 공간데이터베이스, GIS, 멀티미디어 데이터베이스, 컴포넌트, 객체지향 모델링 및 방법론



**윤 용 익**

e-mail : yiyoon@sookmyung.ac.kr

1983년 동국대학교 통계학과 졸업(학사)

1985년 한국과학기술원 전산학과 졸업 (석사)

1994년 한국과학기술원 전산학과 졸업 (박사)

1985년~1997년 한국전자통신연구원 책임연구원

1997년~현재 숙명여자대학교 정보과학부 교수

관심분야 : 멀티미디어 분산시스템, 멀티미디어 통신, 실시간 처리 시스템, 분산 미들웨어 시스템, 데이터베이스 시스템, 실시간 OS/DBMS