변환공간 뷰를 기반으로한 공간 조인
(Spatial Join based on the Transform-Space View)

이 민 재 † 한 욱 신 †† 황 규 영 †††
(Min-Jae Lee) (Wook-Shin Han) (Kyu-Young Whang)

요 약 공간 조인이란 서로 겹치는 관계를 가지는 공간 객체의 쌍들을 찾는 질의이다. 색인 기반 공간 조인에는 원공간 색인인 R 트리가 널리 사용된다. 원공간 색인이란 원공간상에서 표현된 공간 객체를 색인하는 구조로, 이를 활용한 조인은 크기를 가지는 공간 객체를 다루기 때문에 정형적인 방법이 아닌 휴리스틱에 의존하는 단점을 가진다. 반면, 변환공간 색인은 원공간 상의 공간 객체를 변환공간 상의 크기가 없는 점 객체로 변환하여 색인한 후에 이들을 다루기 때문에, 이를 활용한 공간 조인은 상대적으로 단순하고 정형적인 방법을 사용하는 장점을 가진다. 그러나, 이 방법은 R 트리와 같이 원공간 객체를 색인하는 원공간 색인에는 적용될 수 없는 문제점을 가진다. 본 논문에서는 이 두 방법의 장점만을 취하는 새로운 방법을 제안한다. 즉, **변환공간 뷰**(*transform-space view*)라는 새로운 개념과 이를 사용한 공간 조인 알고리즘인 **변환공간 뷰 조인 알고리즘**(*transform-space view join algorithm*)을 제안한다. 변환공간 뷰란 원공간 색인에 대한 가상의 변환공간 색인으로서, 이미 구축된 원공간 색인을 구조적으로 변경하지 않고서 별도의 추가비용 없이 가상의 변환공간 색인으로 해석할 수 있게 한다. 실험 결과, 변환공간 뷰 조인 알고리즘은 R 트리를 원공간에서 조인하는 알고리즘들과 비교하여 디스크 액세스 횟수 측면에서 최대 43.1%까지 더 좋은 성능을 보인다. 본 논문의 가장 중요한 공헌은 R 트리와 같이 널리 사용되는 원공간 색인을 변환공간 뷰라는 새로운 개념을 통하여 변환공간에서 해석하여 사용할 수 있음을 보인 것이다. 우리는 이 새로운 개념이 다양한 공간 질의 처리 알고리즘들이 변환공간에서 새롭게 개발될 수 있는 프레임워크를 마련했다고 믿는다.

키워드 : 변환공간 뷰, 공간 조인, 변환 기법, 지리 정보 시스템

Abstract Spatial joins find pairs of objects that overlap with each other. In spatial joins using indexes, original-space indexes such as the R-tree are widely used. An *original-space index* is the one that indexes objects as represented in the original space. Since original-space indexes deal with sizes of objects, it is difficult to develop a formal algorithm without relying on heuristics. On the other hand, *transform-space indexes*, which transform objects in the original space into points in the transform space and index them, deal only with points but no sizes. Thus, spatial join algorithms using these indexes are relatively simple and can be formally developed. However, the disadvantage of transform-space join algorithms is that they cannot be applied to original-space indexes such as the R-tree containing original-space objects. In this paper, we present a novel mechanism for achieving the best of these two types of algorithms. Specifically, we propose a new notion of the *transform-space view* and present the *transform-space view join algorithm*(*TSVJ*). A transform-space view is a virtual transform-space index based on an original-space index. It allows us to interpret on-the-fly a pre-built original-space index as a transform-space index without incurring any overhead and without actually modifying the structure of the original-space index or changing object representation. The experimental result shows that, compared to existing spatial join algorithms that use R-trees in the original space, the TSVJ improves the number of disk accesses by up to 43.1%. The most important contribution of this paper is to show that we can use original-space indexes, such as the R-tree, in the transform space by interpreting them through the notion of the transform-space view. We believe that this new notion provides a framework for developing various new spatial query

· 본 연구는 첨단정보기술연구센터를 통하여 한국과학재단으로부터 지원을 받았음
† 비 회 원 : 한국과학기술원 전자전산학과
mjlee@mozart.kaist.ac.kr
†† 비 회 원 : 경북대학교 컴퓨터공학과 교수
wshan@knu.ac.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
kywhang@mozart.kaist.ac.kr
논문접수 : 2002년 12월 27일
심사완료 : 2003년 6월 4일

processing algorithms in the transform space.

Key words : Transform-Space View, Spatial Join, Transformation Technique, GIS

1. 서 론

공간 조인이란 주어진 공간 조건을 만족하는 모든 공간 객체의 쌍들을 찾는 공간 질의의 한 종류로서, 공간 데이타베이스 시스템의 기본 연산 중의 하나이다[1]. 공간 조인의 예로는 서로 교차하는 도로와 강의 쌍을 찾는 연산을 들 수 있다. 공간 조인은 많은 디스크 I/O와 처리시간을 필요로 하므로, 이를 효과적으로 처리하는 알고리즘들에 대한 많은 연구들이 진행되어 왔다[1-7].

지금까지 제안된 공간 조인 알고리즘들은 공간 색인을 사용하는 것과 사용하지 않는 것으로 나눌 수 있다. 색인을 사용하지 않는 공간 조인 알고리즘들로는 파티션 기반 공간 머지 조인(Partition Based Spatial Merge Join)[6]과 공간 해쉬 조인(Spatial Hash Join)[5], 시드 조인(Seed Join)[4], 그리고 슬롯 인덱스 공간 조인(Slot Index Spatial Join)[8] 등이 있다. 이들 공간 조인 방법들은 조인되는 어느 파일에도 공간 색인이 구축되어 있지 않거나, 조인되는 한 파일에만 공간 색인이 있는 경우에는 유용하다. 그러나, 공간 색인이 이미 구축되어 있는 경우, 공간 색인을 사용하는 알고리즘보다 느린 문제점을 가지고 있다. 따라서, 본 논문에서는 색인을 사용한 조인 알고리즘을 논의 대상으로 한다.

색인을 사용하는 공간 조인 알고리즘들은 사용되는 색인의 종류에 따라 원공간 색인 조인 알고리즘과 변환공간 색인 조인 알고리즘으로 나뉜다. 원공간 색인 조인 알고리즘은 원공간상의 크기를 가지는 공간 객체들을 색인하여 공간 조인을 수행하는 알고리즘이다. 변환공간 색인 조인 알고리즘은 원공간 상의 크기를 가지는 공간 객체들을 변환공간 상의 크기를 갖지 않는 점들로 변환하여 색인한 후 공간 조인을 수행하는 알고리즘이다.

원공간 색인 조인 알고리즘으로는 R 트리[9, 10]를 기반으로 한 알고리즘들이 있다. Brinkhoff 등은 두 개의 R 트리를 깊이 우선 탐색(depth-first traversal)하면서 여러 가지 휴리스틱을 사용하여 공간 조인을 수행하는 깊이 우선 탐색 R 트리 조인 알고리즘[1]을 제안하였다. 이 알고리즘은 공간 조인의 성능을 비교할 때 기준으로서 사용되는 대표적인 알고리즘이다. Huang 등은 두 개의 R 트리를 넓이 우선 탐색(breadth-first traversal)하면서 공간 조인을 수행하는 넓이 우선 탐색 R 트리 조인 알고리즘[3]을 제안하였다. Huang 등이 제안한 알고리즘은 디스크 액세스 순서와 메모리 관리 방법에 따라 Combo1과 Combo2라는 두 가지 방법으로 나뉜다. Combo1은 사용하는 버퍼의 크기가 작을 때 Brinkhoff의 알고리즘보다 성능이 우수하나, 버퍼의 크기가 클 때는 Brinkhoff의 알고리즘보다 성능이 떨어지는 문제점을 가지고 있고, Combo2는 그 반대의 문제점을 가진다. Brinkhoff 등과 Huang 등이 제안한 알고리즘들은 모두 정형적인 분석 없이 휴리스틱에 의존하는 문제점을 가진다. Brinkhoff의 알고리즘은 local plane sweeping, pinning, 그리고 local Z-ordering이라는 휴리스틱을 사용하고, Huang의 알고리즘은 디스크 페이지 액세스 순서를 결정하는 여러가지 휴리스틱들을 사용한다.

변환공간 색인 조인 알고리즘으로는 MLGF[11,12]를 기반으로 하여 Song 등이 제안한 변환 기반 공간 조인 알고리즘[7]이 있다. 이 알고리즘은 변환공간의 특성을 사용하여 전역 최적화를 수행하기 때문에 Brinkhoff 알고리즘보다 항상 성능이 우수하거나 비슷한 성능을 보인다.

공간 색인을 사용하는 조인 알고리즘들은 각각의 장단점을 가진다. 원공간 색인 조인 알고리즘은 R 트리와 같이 널리 사용되는 원공간 색인을 활용한다는 장점이 있는 반면, 크기를 가진 객체를 다루므로 공간 조인 알고리즘이 상대적으로 복잡하며 정형적인 방법이 아닌 휴리스틱에 의존하는 단점을 가진다. 이에 반해 변환공간 색인 조인 알고리즘은 크기가 없는 점만을 다루므로 공간 조인 알고리즘이 상대적으로 단순하며 정형적인 방법이라는 장점을 가지는 반면, R 트리와 같이 널리 사용되는 원공간 색인에 적용될 수 없는 단점을 가진다.

본 논문에서는 이 두 방법의 장점만을 취하는 새로운 방법을 제안한다. 즉, 원공간 색인에 대한 가상의 변환공간 색인인 **변환공간 뷰**(*transform-space view*)를 제안하고, 이를 사용하여 원공간 색인을 변환공간에서 조인하는 공간 조인 알고리즘인 **변환공간 뷰 조인 알고리즘**(*transform-space view join algorithm*)을 제안한다. 변환공간 뷰란 본 논문에서 제안하는 새로운 개념으로 이미 구축된 원공간 색인을 구조적으로 변경하지 않고서 별도의 추가비용 없이 가상의 변환공간 색인으로 해석하게 해준다. 따라서, 변환공간 뷰를 통해 성능 좋은 변환공간 색인 조인 알고리즘을 R 트리와 같이 널리 사용되는 원공간 색인에 적용할 수 있다. 본 논문에서는 변환공간 뷰 조인 알고리즘으로서 Song 등[7]이 제안한 변환공간 조인 알고리즘을 개선한 것을 사용한다. 변환공간 뷰는 영역과 객체를 MBR(minimum bounding rectangle)들로 관리하는 모든 원공간 색인에 적용이 가

능하다.[1)]

본 논문의 공헌은 다음의 세 가지로 요약된다. 첫째, 원공간 색인을 가상의 변환공간 색인으로 해석하는 새로운 개념인 변환공간 뷰를 제안한다. 이 개념을 통해 널리 사용되는 R 트리와 같은 원공간 색인에 변환공간 색인 조인 알고리즘을 적용할 수 있다. 둘째, Song 등이 제안한 변환 기반 공간 조인 알고리즘을 향상 시켜 변환공간 뷰를 사용한 공간 조인 알고리즘을 제안한다. 셋째, 실험을 통해 R 트리를 원공간 색인 조인 알고리즘을 사용하여 공간 조인하는 것보다 변환공간 색인 조인 알고리즘을 사용하여 변환공간 뷰를 통해 조인하는 것이 성능측면에서 더 좋음을 보인다.

본 논문의 구성은 다음과 같다. 제 2장에서는 관련 연구로서 원공간 색인 조인 알고리즘과 변환공간 색인 조인 알고리즘에 대해 설명한다. 제 3장에서는 변환공간 뷰의 개념을 제안하고, 제 4장에서는 변환공간 뷰에 기반한 공간 조인 알고리즘을 제안한다. 제 5장에서는 제안하는 공간 조인 알고리즘의 성능 평가를 수행하고 그 결과를 제시한다. 마지막으로 제 6장에서는 결론을 내린다.

2. 관련 연구

본 장에서는 색인을 사용하는 공간 조인 알고리즘들에 대해 설명한다. 제 2.1절에서는 원공간 색인을 사용하는 공간 조인 알고리즘을 설명하고, 제 2.2절에서는 변환공간 색인을 사용하는 공간 조인 알고리즘에 대해 설명한다.

2.1 원공간 색인 조인 알고리즘

원공간 색인 조인 알고리즘으로는 Brinkhoff 등이 제안한 **깊이 우선 탐색 R 트리 공간 조인**(*Depth-First Traversal R-tree Spatial Join: DFRJ*) 알고리즘[1]과 Haung 등이 제안한 **넓이 우선 탐색 R 트리 공간 조인**(*Breadth First Traversal R-tree Spatial Join: BFRJ*) 알고리즘[3]이 있다. 이 알고리즘들은 원공간 색인으로 R 트리를 사용하므로, 먼저 R 트리 구조에 대해서 간략하게 설명한다. R 트리는 다차원 데이타를 저장하는 균형이 잡힌(height-balanced) 색인 구조이다. R 트리의 비단말 페이지(non-leaf page)는 <*mbr, ref*> 형태의 엔트리들로 구성된다. 여기서, *ref*는 자식 페이지를 가리키는 포인터이고, *mbr*은 자식 페이지의 모든 엔트리들의 MBR(Minimum Bounding Rectangle)을 모두 포함하는 MBR이다. 단말 페이지(leaf page)는 <*mbr, oid*>형태의 엔트리들로 구성되는데, 여기서 *oid*는 데이타베이스에 저장되어 있는 공간 객체를 가리키는 식별자이고, *mbr*은 공간 객체의 MBR이다.

Brinkhoff의 DFRJ 알고리즘은 깊이 우선 탐색을 통해 주어진 두 R 트리의 모든 단말 페이지의 MBR 들을 서로 비교하여 겹침(overlap)이 발생하는 MBR 쌍들을 찾는다. 이때, 상위 비단말 페이지의 MBR 들 사이에 겹침이 발생하는 페이지들만을 비교 함으로써 비교 대상을 줄인다[1]. 즉, 어떤 두 비단말 페이지에서 겹침이 발생하지 않으면, 각 비단말 페이지들로부터 재귀적으로 연결된 단말 페이지들의 집합 사이에는 어떠한 겹침도 발생하지 않는다는 사실을 활용한다. 공간 조인 시에 같은 디스크 페이지들이 디스크에서 여러 번 읽히는 것을 최대한 방지하기 위해 페이지 액세스 순서를 정하는 것이 필요하다. Brinkhoff의 DFRJ 알고리즘에서는 디스크 액세스 횟수를 줄이기 위해 local plane sweeping과 pinning과 같은 휴리스틱 방법들을 사용한다. 그러나, 이들 휴리스틱 방법들은 겹침 관계에 있는 단 하나의 비단말 페이지 쌍이 가리키는 자식 페이지들의 액세스 순서만을 고려하므로, 전역 최적값(global optimum)을 찾지 못하고 지역 최적값(local optimum)만을 찾는 단점을 가진다[7].

하나 이상의 비단말 페이지 쌍들이 가리키는 자식 페이지들에 대한 순서를 함께 고려할 수 있다면, 전역 최적값에 가까운 값을 구할 수가 있다는 점에 착안하여, Huang의 BFRJ 알고리즘[3]에서는 겹침 관계에 있는 페이지 쌍들의 정보를 넓이 우선 탐색을 통해 가능한 많이 구한 후, 구해진 페이지 쌍들의 순서를 정하여 공간 조인을 수행한다. 그러나, 많은 페이지 쌍들의 정보들을 저장하기 위한 데이타 구조의 비용으로 인해, 작은 버퍼 크기에서는 DFRJ에 비해 성능이 떨어지는 문제점을 가지고 있다.

2.2 변환공간 색인 조인 알고리즘

변환공간 색인 조인 알고리즘은 점 객체를 색인하는 변환공간 색인의 특성을 사용한다. 변환공간 색인은 변환기법을 사용하여 원공간의 크기를 갖는 공간 객체를 변환공간상의 크기가 없는 점으로 변환하고, 이를 다차원 점 색인으로 색인한다. 이때 사용되는 대표적인 변환기법으로는 구석점 변환기법[18, 19]이 있다. 구석점 변환기법은 n차원 원공간상의 공간 객체를 $2n$차원 변환공간상의 점 객체로 변환하는데, n차원 원공간의 각 차원 축에 대한 원공간 객체의 최소값과 최대값을 사용하여 $2n$차원 점 객체의 좌표값을 구성한다. 예를 들어, 1차원 원공간에서 공간 객체의 최소값과 최대값이 각각 lx와 rx인 공간 객체는 2차원 변환공간상의 점 객체 (lx, rx)로 변환된다.

1) 변환공간 뷰의 적용이 가능한 원공간 색인의 예로는 R-tree[10], R*-tree[9], X-tree[13], SKD-Tree[14], 그리고 GBD-Tree[15] 등이 있다. MBR들이 아닌 임의의 다각형들을 사용하여 영역과 객체를 관리하는 원공간 색인들에는 변환공간 뷰를 적용할 수 없는데, 이러한 원공간 색인들로는 P-Tree[16]과 Cell-Tree[17] 등이 있다.

다른 변환기법으로는 n차원 원공간상의 객체를 1차원 변환공간상의 점으로 변환하는 기법들[20]이 있다. 변환된 점들은 1차원 액세스 방법을 통해 색인된다. 이 기법들은 우선 공간을 그리드 셀들로 나눈 뒤, 각 셀들에 Z 순서나 힐버트 순서로 결정된 번호를 부여한다. 변환된 점들은 자신을 포함하는 셀의 번호에 의해 B^+-tree에 색인된다. 그러나, 이 기법들에서 그리드 구조가 서로 다른 경우, 한쪽이 같은 구조로 새로 색인되기 전까지는 조인될 수 없는 문제점을 가진다. 이 때문에 이 방법들은 공간 조인에 적당하지 않는 것으로 알려져 있다[21].

Song 등은 MLGF[11, 12]를 색인 구조로 사용하는 **변환 기반 공간 조인**(*Transform-Based Spatial Join: TBSJ*) 알고리즘을 제안하였다[7]. 이 알고리즘은 구석점 변환기법으로 변환된 점 객체들을 색인하는 두 MLGF R과 S를 공간 조인한다. 먼저 MLGF의 구조를 간략하게 설명한다. MLGF는 다차원 점 데이타를 저장하는 균형이 잡힌(height-balanced) 색인 구조이다. MLGF의 비단말 페이지는 <*region, ref*>형태의 엔트리들로 구성되는데, 여기서 *ref*는 자식 페이지를 가리키는 포인터이고, *region*은 자식 페이지의 모든 엔트리들의 영역을 모두 포함하는 영역이다. 단말 페이지는 <*point, oid*>의 형태의 엔트리들로 구성되는데, 여기서 *oid*는 데이타베이스에 저장되어 있는 점 객체를 가리키는 식별자이고, *point*는 점 객체의 좌표값을 나타낸다.

Song의 TBSJ 알고리즘은 R과 S의 모든 단말 페이지들의 점 객체들을 서로 비교하여 겹침이 발생하는 점 객체들의 쌍을 찾는다. TBSJ 알고리즘은 모든 단말 페이지들을 고려하기 때문에 전역 최적화를 수행한다. 이 때, 변환공간에서 서로 인접한 R의 두 영역과 공간 조인되는 S의 두 영역들 사이에는 매우 큰 겹침이 존재하는 특성을 가지는데, 이러한 특성과 LRU 버퍼를 사용하여 디스크 페이지의 액세스 횟수를 줄인다. 자세한 방법은 제 4.1절에서 설명한다.

Song의 TBSJ 알고리즘은 전역 최적화와 위에서 언급한 특성을 활용하여 페이지 순서를 정형적으로 분석하여 사용하기 때문에 Brinkhoff의 DFRJ보다 항상 좋은 성능을 보인다[7]. 그러나 변환공간 색인 조인 알고리즘은 변환공간상의 점 객체들만을 다루는 변환공간 색인에 대해서만 적용이 가능하므로, 크기를 가지는 객체를 다루는 원공간 색인에는 직접 적용하지 못한다. 그러므로, R 트리와 같이 널리 사용되는 원공간 색인에는 사용될 수 없는 것으로 알려져 있다.

3. 변환공간 뷰

변환공간 뷰(*transform-space view*)는 원공간 색인에 대한 가상의 변환공간 색인으로, 변환기법을 사용하여 정의된다. 여기서 원공간 색인은 n차원 원공간상의 공간 객체를 색인하는 구조이고, 변환공간 색인은 $2n$차원 변환공간상의 점 객체를 색인하는 구조이다. 변환공간 뷰를 사용하면 그림 1과 같이 원공간 색인 구조를 변경하지 않고도 가상의 변환공간 색인인 변환공간 뷰를 통해 변환공간 색인 조인 알고리즘을 원공간 색인에 직접 적용할 수 있다. 그림에서 두 원공간 색인 R과 S는 변환공간 뷰 TV(R)과 TV(S)로 동적으로 해석되며, TV(R)과 TV(S)에 변환공간 색인 조인 알고리즘이 적용된다.

변환공간 뷰는 원공간 색인의 각 구성 요소를 구석점 변환공간 색인의 구성 요소로 매핑함으로써 정의된다. 우선, 원공간 색인인 R 트리의 구조를 변환공간 색인인 MLGF의 구조로 매핑하는 방법을 예를 들어 설명한다. 제 2장에서 소개하였듯이, R 트리의 비단말 페이지의 엔트리는 <*mbr, ref*>의 형태이고, MLGF의 비단말 페이지의 엔트리는 <*region, ref*>의 형태이다. 여기서, *mbr*은 n차원 영역을 나타내고, *region*은 $2n$차원 영역을 나타낸다. 따라서, R 트리에서의 *mbr*은 MLGF에서의 *region*으로 매핑된다. R트리의 단말 페이지의 엔트리는 <*mbr, oid*>의 형태이고, MLGF의 비단말 페이지 엔트리는 <*point, oid*>의 형태이다. 여기서, *mbr*은 n차

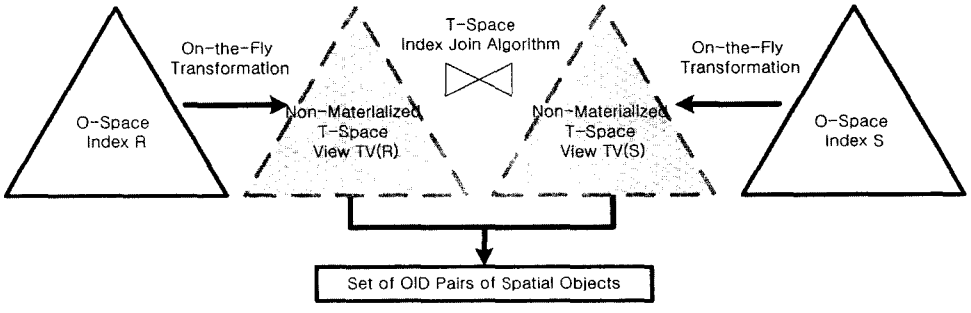


그림 1 변환공간 뷰를 통한 변환공간 질의 처리 과정

원 객체를 나타내고, *point*는 *2n*차원 점을 나타낸다. 따라서, R 트리에서의 *mbr*은 MLGF에서의 *point*로 매핑된다. 이와 같은 과정을 통해 R 트리를 MLGF와 유사한 구조를 가지는 변환공간 색인으로 해석할 수 있다.

*n*차원 원공간 색인을 *2n*차원 변환공간 뷰로 정의하는 정형적인 방법은 다음과 같다. 변환기법으로는 구석점 변환기법을 사용한다.

정의 1 *n*차원 원공간 색인에서 원공간 객체와 원공간 영역이 각각 $[l_1, r_1]$ $[l_2, r_2]$ × ... × $[l_n, r_n]$의 형태로 표현된다고 하자. 여기서 l_i는 원공간의 *i*번째 차원축에 대한 객체 혹은 영역의 최소값을 뜻하며, r_i는 최대값을 뜻한다. *n*차원 원공간 색인에 대한 *2n*차원 변환공간 뷰는 다음과 같이 정의된다.

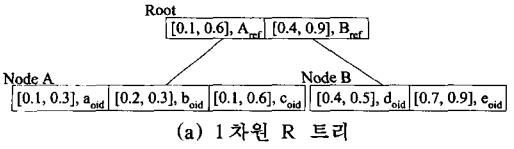
(a) 원공간 객체의 매핑: *n*차원 원공간 색인에서 원공간 객체 $[l_1, r_1] \times [l_2, r_2] \times ... \times [l_n, r_n]$은 *2n*차원 변환공간 뷰의 점 $<l_1, r_1, l_2, r_2, ..., l_n, r_n>$으로 매핑된다.

(b) 원공간 영역의 매핑: *n*차원 원공간 색인에서 원공간 영역 $[l_1, r_1] \times [l_2, r_2] \times ... \times [l_n, r_n]$은 *2n*차원 변환공간 뷰의 영역 $[l_1, r_1] \times [l_1, r_1] \times [l_2, r_2] \times [l_2, r_2] \times ... \times [l_n, r_n] \times [l_n, r_n]$으로 매핑된다.

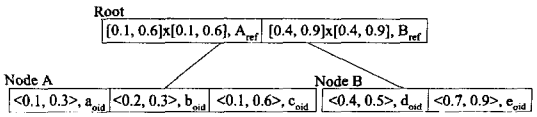
정의 1의 **원공간 객체의 매핑**은 구석점 변환 기법의 정의를 그대로 따른 것이고, 정의 1의 **원공간 영역의 매핑**은 원공간 색인에서의 객체와 영역 사이의 관계가 변환공간 색인에서도 유지되도록 구석점 변환 기법의 정의를 확장하여 정의한 것이다. 즉, 원공간 영역의 매핑은 원공간 영역에 포함될 수 있는 모든 객체들이 변환공간 점 객체들로 매핑되는 경우, 매핑된 모든 변환공간 점 객체들이 매핑된 변환공간 영역에 포함됨을 보장하도록 변환공간 영역을 정의한다. 이러한 포함 관계의 유지는 변환공간 뷰가 올바른 탐색 구조를 가지기 위해 필요하다. 예를 들어, 만약 1차원 원공간 영역 $[lx, rx]$에 포함될 수 있는 임의의 객체 $[lx', rx']$ $(lx \le lx' \le rx' \le rx)$을 2차원 변환공간 점 객체 P $<lx', rx'>$으로 매핑하면, P는 $lx \le lx' \le rx$와 $lx \le rx' \le rx$의 관계에 의해 2차원 변환공간 영역 $[lx, rx] \times [lx, rx]$의 안에서만 존재한다. 따라서, 원공간 영역과 원공간 객체들 사이의 관계가 정의 1의 원공간 영역의 매핑에 의한 변환 후에도 보존된다.

Example 1: 그림 2는 1차원 원공간 색인인 R 트리에 대한 2차원 변환공간 뷰를 나타낸다. 그림 2(a)는 1차원 R 트리 구조를 표현한 것이고, 그림 2(b)는 이로부터 정의된 2차원 변환공간 뷰를 표현한 것이다. 그림 2(c)는 (a)의 객체들과 영역들을 1차원 원공간상에서 표현한 것이며, 그림 2(d)는 (b)의 점 객체들과 영역들을 2차원 변환공간상에서 표현한 것이다. 그림 2(a)의 원공간 색인은 두 개의 단말 페이지 A, B와 Root로 구성되고 단말 페이지들에는 원공간 객체 a, b, c, d, e가 저장된다. 원공간 객체와 영역의 변환공간 점과 영역으로의 매핑은 다음과 같다. 그림 2(c)에서 [0.1, 0.3]으로 표현된 원공간 객체 a는 정의 1(a)에 의해 그림 2(d)에서 변환공간 점 <0.1, 0.3>으로 매핑된다. 나머지 원공간 객체 b, c, d, e들도 이와 마찬가지 방법으로 매핑된다. 그림 2(c)의 [0.1, 0.6]으로 표현된 단말 페이지 A의 영역은 정의 1(b)에 의해 그림 2(d)에서 변환공간 영역 [0.1, 0.6] × [0.1, 0.6]으로 매핑된다. [0.4, 0.9]로 표현된 단말 페이지 B의 영역은 변환공간 영역 [0.4, 0.9] × [0.4, 0.9]으로 매핑된다. □

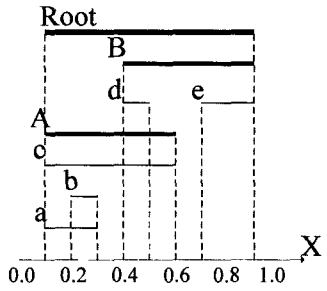
정의 1을 통해 변환공간 뷰를 사용하는 알고리즘은 원공간 색인을 변환공간 뷰로 별도의 추가비용 없이 매



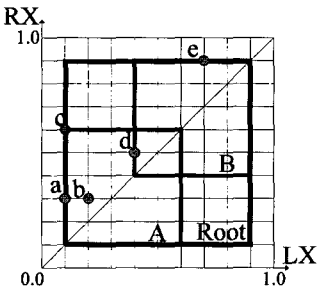
(a) 1차원 R 트리



(b) 1차원 R 트리로부터 정의된 2차원 변환공간 뷰



(c) 1차원 원공간에서 R 트리 구조의 표현



(d) 2차원 변환공간에서 변환공간 뷰 구조의 표현

그림 2 원공간 색인 구조에 대한 변환공간 뷰로의 매핑 예

핑하여 사용할 수 있다. 왜냐하면, 정의 1의 **원공간 객체의 매핑**과 **원공간 영역의 매핑**은 원공간 객체와 영역을 구성하는 좌표값들의 재배치만으로 이루어 지는데, 이 재배치된 좌표값들을 알고리즘을 구현한 코드상에서 사용하는 것만으로 원공간 색인을 변환공간 뷰로 볼 수 있기 때문이다. 따라서, 좋은 성능을 가지는 변환공간 조인 알고리즘을 원공간 색인에 추가비용 없이 적용할 수 있다.

정의 1에 의하면 변환공간 뷰에서의 영역들은 항상 정사각형이고 대각선 위에 존재하는 특성을 가진다. 그렇기 때문에, 변환공간 뷰에서는 가로로 인접한 영역들이 존재하지 않는다. 이러한 특징은 변환공간 뷰를 사용한 변환공간 뷰 조인 알고리즘의 설계에 활용된다. 자세한 내용은 제 4.2절에서 설명한다.

4. 변환공간 뷰 조인 알고리즘

본 장에서는 변환공간 색인 조인 알고리즘[7]과 이를 변환공간 뷰에 적용하는 방법을 설명한다. 제 4.1절에서는 Song 등이 제안한 TBSJ 알고리즘[7]에 대해 설명하고, 제 4.2절에서는 TBSJ를 확장하여 변환공간 뷰 조인 알고리즘을 제안한다.

4.1 변환기반 공간 조인 알고리즘(TBSJ)

TBSJ 알고리즘은 구석점 변환 기법에 의해 원공간상의 객체들로부터 변환된 점 객체들을 색인하는 두 변환공간 색인 R과 S를 공간 조인한다. 설명의 편의를 위해 본 절에서는 1) 변환공간은 2차원이고, 2) 색인은 변환공간을 동일한 크기를 가지는 그리드 셀들로 분할한다고 가정한다. 여기서, 각 그리드 셀들은 색인의 단말 페이지와 일대일 대응된다.

먼저, 공간 조인 윈도우(spatial join window)의 개념을 설명한다. 다음으로, 공간 조인 윈도우들 사이의 특성을 정리하고, 이 특성을 이용하는 TBSJ 알고리즘을 정의한다.

정의 2 [7] 서로 공간 조인되는 색인 R과 S의 변환공간을 TS(R)과 TS(S)라 하자. 이때, 2차원 TS(R)에 있는 사각형 영역 P에 대한 **공간 조인 윈도우**(*spatial join window*) *SJW(P)*는 사각형 영역 P에 포함될 수 있는 모든 객체들과 교차관계를 가지는 객체들이 존재하는 TS(S)내의 최소 영역이다. **공간 조인 윈도우 페이지**(*spatial join window pages*) *SJWP(P)*는 SJW(P)와 대응되는 페이지들의 집합이다.

아래의 정리 1은 주어진 영역 P에 대한 SJW(P)가 변환공간에서 어떤 영역으로 표현되는 지를 나타낸다. 정리 1은 보조정리 1과 2에 의해 유도되는데, 보조정리 1은 SJW(P)가 P의 좌상점과 교차관계를 가지는 영역임을 나타내는 것이고, 보조정리 2는 좌상점과 교차관계를 가지는 영역의 표현 방법을 정리한 것이다.

보조정리 1 [7] TS(R)내의 사각형 영역 P의 좌상점을 q라 하자. 이때, 영역 P에 대한 2차원 SJW(P)는 좌상점 q와 대응되는 원공간 객체와 원공간에서 교차관계를 가질 수 있는 모든 객체들이 존재하는 TS(S)내의 최소 영역이다.

증명: 사각형 영역 P에 포함되는 임의의 점 $<lx', rx'>$을 u라 하자. 점 u는 영역 P안에 존재하므로, $lx \le lx'$와 $rx' \le rx$의 관계가 성립한다. 점 u는 변환공간상의 대각선 위에 존재하므로 $lx' \le rx'$의 관계가 성립하므로, $lx \le lx' \le rx' \le rx$의 관계가 성립한다. 이 관계에 의해 점 q와 대응되는 원공간 객체 $\hat{q}$은 점 u와 대응되는 원공간 객체 $\hat{u}$을 항상 포함한다. 원공간 객체 $\hat{q}$은 영역 P에 존재하는 모든 원공간 객체를 포함하는 가장 큰 객체이므로, TS(S)내의 객체들이 원공간 객체 $\hat{u}$과 교차하기 위해서는 $\hat{q}$과 반드시 교차해야 한다. □

보조정리 2 [7] 2차원 TS(R)내의 점 $q=<lx, rx>$와 대응되는 원공간 객체 $\hat{q}$과 원공간에서 교차관계를 가질 수 있는 모든 객체들이 존재하는 TS(S)내의 최소 영역은 $[0, rx] \times [lx, 1]$이다.

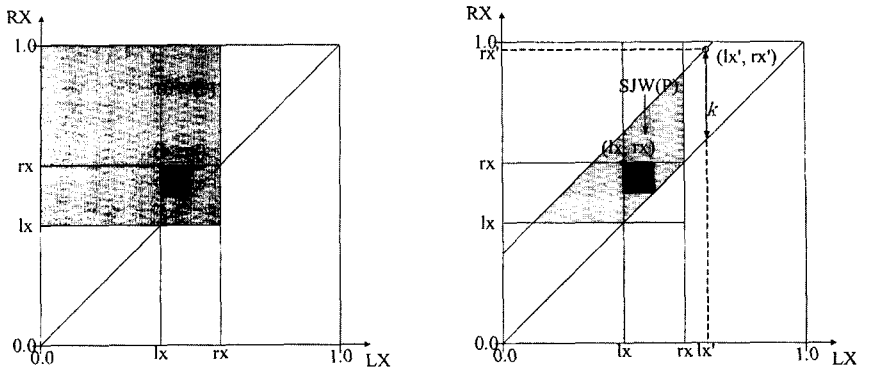
증명: TS(R)내의 점 q와 대응되는 원공간 객체 $\hat{q}$과 점 $u =<lx', rx'>$과 대응되는 원공간 객체 $\hat{u}$이 서로 교차관계에 있을 때, 이 두 객체들은 서로 떨어져 있지 않은 관계를 가지므로 $not\ (lx > rx'\ or\ lx' > rx)$의 관계를 만족한다. 여기서 $lx > rx'$과 $lx' > rx$는 두 객체 $\hat{q}$과 $\hat{u}$이 서로 떨어져 있음을 뜻한다. 이 관계를 풀어 쓰면 $(lx \le rx'\ and\ lx' \le rx)$이 된다. 이 관계에 의해 lx'은 구간 $[0, rx]$에 존재하고, rx'은 구간 $[lx, 1]$에 존재하므로, $\hat{q}$과 교차하는 모든 객체들이 존재하는 TS(S)내의 최소 영역은 $[0, rx] \times [lx, 1]$이다. □

정의 1 [7] 2차원 TS(R)내의 사각형 영역 P의 좌상점이 $<lx, rx>$일 때, 2차원 SJW(P)는 TS(S)내의 영역 $[0, rx] \times [lx, 1]$이다.

증명: 정의 2와 보조정리 1과 2에 의해 성립함. □

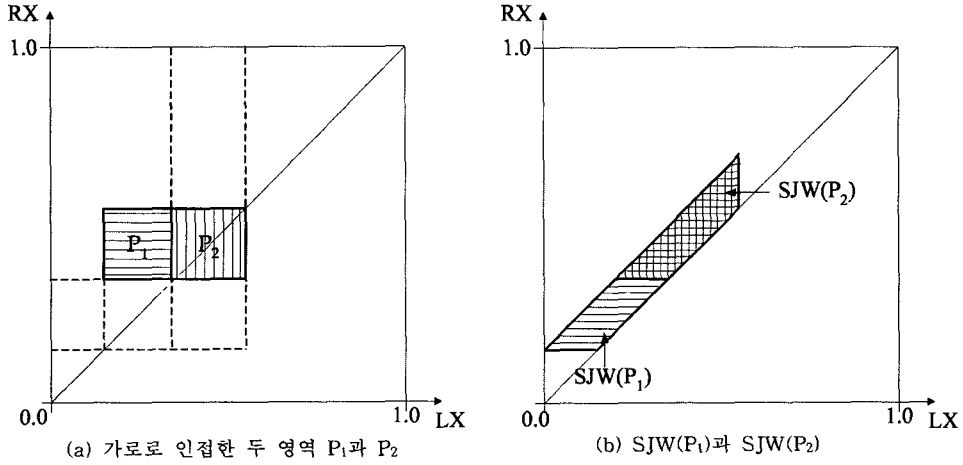
Example 2: 그림 3은 TS(R)내의 영역 P에 대한 TS(S)내의 영역 SJW(P)를 나타낸다. 그림 3(a)에서 영역 P는 길은 사각형으로, SJW(P)는 옅은 사각형으로 표시된다. □

그림 3(b)는 변환공간에서 점 객체가 존재하는 영역인 TOPA(Transformed Objects Placing Area)에 의해 그 크기가 제한된 SJW(P)를 나타낸다. TOPA는 대각선 위의 얇은 띠 형태를 한다. TOPA가 대각선 위에 존재하는 이유는 변환공간 상의 점 객체 $<lx, rx>$를 구성하는 lx와 rx가 각각 원공간 객체의 최소값과 최대값이므로, 항상 $lx \le rx$의 관계가 존재하기 때문이다. TOPA가 얇을 띠 형태를 하는 이유는 대각선과 점 객



(a) 공간 조인 윈도우 (b) TOPA로 크기가 제한된 공간 조인 윈도우

그림 3 공간 조인 윈도우



(a) 가로로 인접한 두 영역 P_1과 P_2 (b) SJW(P_1)과 SJW(P_2)

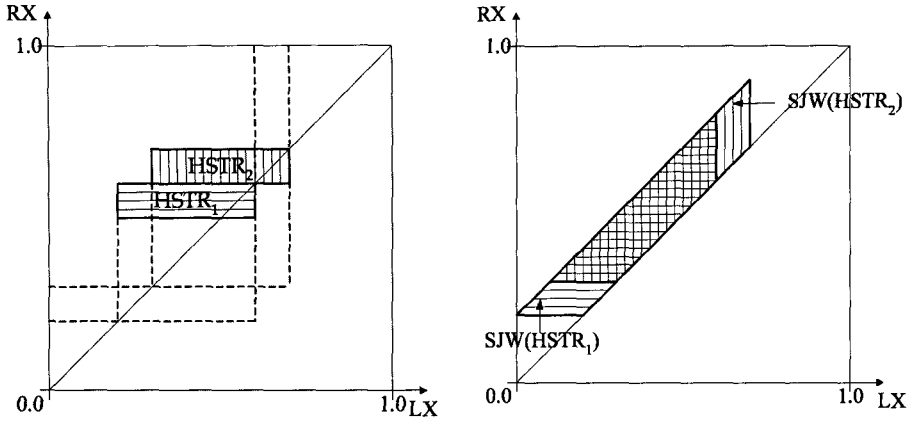
그림 4 가로로 인접한 두 영역들의 공간 조인 윈도우들

체와의 수직 거리는 원공간 객체의 크기를 뜻하므로, 가장 큰 원공간 객체의 크기가 k인 경우, 모든 점 객체들은 대각선과의 수직 거리가 k이상이 될 수 없기 때문이다. GIS와 같은 공간 데이타베이스 응용에서 공간 객체의 크기는 원공간의 크기에 비해 매우 작아 $k \ll 1$이기 때문에 TOPA의 두께는 매우 얇다.

TS(R)내의 인접한 두 영역들에 대한 TS(S)내의 두 공간 조인 윈도우들은 변환공간에서 서로 많이 겹치는 특성을 가진다. 이 특성은 두 가지로 나뉜다. 첫 번째 특성은 가로로 인접한 두 영역들의 공간 조인 윈도우들은 서로 포함관계를 가지는 것이다. 그림 4(a)에서와 같이 가로로 인접한 두 영역들 중 대각선에서 먼 쪽을 P_1, 가까운 쪽을 P_2라고 할 때, 이 두 영역의 SJW들 사이에는 그림 4(a)에서와 같이 SJW(P_1) ⊃ SJW(P_2)인 관계가 성립한다. 만약 P_1과 SJW(P_1)을 조인한 후에 P_2와 SJW(P_2)를 조인하면, SJW(P_2)는 이미 버퍼에 있으므로 별도의 디스크 액세스 없이 조인을 처리할 수 있다. 그러므로, TBSJ 알고리즘에서는 가로로 인접한 영역들을 가로띠로 정의하고 가로띠에 속한 영역들을 함께 조인한다. 정의 3은 가로띠를 정의한다.

정의 3 [7] 2차원 **가로띠**(*Horizontal Strip*) *HSTR*은 세로축에 대한 투영 범위가 동일한 그리드 셀들을 포함하는 집합으로 정의된다. **가로띠 페이지**(*Horizontal Strip Pages*) *HSP*(*HSTR*)은 가로띠 HSTR에 포함된 페이지들의 집합이다.

두 번째 특성은 세로로 인접한 두 가로띠들의 공간 조인 윈도우들 사이에는 큰 겹침이 존재하는 것이다. 그림 5에서와 같이 세로로 인접한 두 가로띠 $HSTR_1$과 $HSTR_2$의 공간 조인 윈도우 SJW($HSTR_1$)과 SJW ($HSTR_2$)는 서로 크게 겹친다. 만약, $HSTR_1$과 SJW



(a) 세로로 인접한 두 가로띠 $HSTR_1$과 $HSTR_2$ (b) $SJW(HSTR_1)$과 $SJWP(HSTR_2)$

그림 5 세로로 인접한 두 가로띠들의 공간 조인 윈도우들

($HSTR_1$)을 조인한 후에 $HSTR_2$와 $SJW(HSTR_2)$를 조인하면, $SJW(HSTR_2)$의 대부분은 이미 버퍼에 있으므로 적은 수의 디스크 액세스로 조인을 처리할 수 있다. 그러므로, TBSJ 알고리즘에서는 가로띠들을 세로로 인접하게 선택하면서 조인한다.

위의 두 가지 특성을 사용하여 TBSJ 알고리즘을 그림 6과 같이 정의한다. 알고리즘은 변환공간 색인 R과 S, 그리고 가로띠의 선정 순서인 SFC를 입력으로 받는다. 여기서, SFC는 **공간 채움 곡선**(*Space Filling Curve*)[22]이며, 공간상의 영역들의 순서를 결정하는데 사용된다. 여기서는 가로띠의 순서를 결정하기 위해 사용된다. 2차원 변환공간의 경우, 세로축(RX)을 SFC로서 사용하여 가로띠들을 세로로 인접하게 선택한다. 알고리즘은 세 개의 루프(loop)로 구성되는데, Line 1의 첫 번째 루프는 R 내의 각각의 가로띠 HSTR를 SFC에 따라 인접하게 선택한다. Line 2의 두 번째 루프는 각각의 가로띠 HSTR에 대해 SJWP(HSTR)를 구하고, 이에 포함되는 페이지를 선택하여 *s_page*로 가리킨다. Line 3의 세 번째 루프는 HSTR에 속하는 페이지 집합 HSP(HSTR)에서 각 페이지들을 선택하여 *r_page*로 가리킨다. Line 5는 선택한 두 페이지 *s_page*와 *r_page*를 *JoinPages* 함수를 사용하여 서로 조인한다.

알고리즘 TBSJ의 Line 2와 3에서 페이지를 선택하는 순서를 조정함으로써, 디스크 액세스 횟수를 더 줄 일 수 있다. 이를 위해 Line 2에서는 페이지들을 선택하는 순서로 행기준(row major)으로 아래에서 위(bottom-up)로 선택하는 순서와 위에서 아래(top-down)로 선택하는 순서를 번갈아 가며 사용하고, Line 3에서는 대각선으로부터 먼 쪽에서 가까운 쪽(far-near)으로 선택하는 순서와 가까운 쪽에서 먼 쪽(near-far)으로 선택하는

```
Algorithm TBSJ(R, S, SFC)
1  for each horizontal strip HSTR in R ordered by SFC
2      for each page s_page of SJWP(HSTR) in S,
3          for each page r_page in HSP(HSTR)
4              if s_page is a page of SJWP(r_page)
5                  JoinPages(r_page, s_page)
6              end
7          end
8      end
9  end
```

그림 6 Transform-Based Spatial Join(TBSJ) 알고리즘

순서를 번갈아 가며 사용한다. 이렇게 순서를 정하면, 이전 루프에서 읽은 내용이 버퍼 안에 들어 있을 가능성을 높아져 다음 루프에서의 디스크 액세스 횟수를 줄인다. 자세한 방법은 참고문헌 [7]에 나와 있다.

4.2 변환공간 뷰 조인 알고리즘

그림 7은 본 논문에서 제안하는 변환공간 뷰 조인 알고리즘(Transform-Space View Join Algorithm: TSVJ)을 나타낸 것으로 제 4.1절에서 정의한 변환 기반 공간 조인 알고리즘을 변환공간 뷰에 맞게 수정한 알고리즘이다. TSVJ는 변환공간 뷰 TV(R)과 TV(S), 공간 채움 곡선 SFC를 입력으로 받는다. 2차원 변환공간의 경우에는 세로축을 SFC로 사용하여 공간을 세로로 인접하게 선택한다. 일반적으로 *2n*차원인 경우에는 *2n*차원 가로띠와 직각을 이루는 *n*차원 공간에서의 공간 채움 곡선을 SFC로 사용한다.

그림 7에서 TSVJ는 TV(R)의 한 단말 페이지와 대응되는 영역을 SFC 순서로 선택하고, 이를 하나의 가로띠를 사용한다. 여기서, 하나의 영역을 하나의 가로띠로

사용하는 이유는 1) 정의 1에 의해 판명된 변환공간 뷰의 영역들이 가지는 특성에 의해 변환공간에서는 가로로 인접한 영역들이 존재하지 않기 때문이며, 2) R-tree와 같은 원공간 색인들에서는 서로 겹치는 영역들간에 객체들이 중복되어 저장되지 않기 때문이다.[2] 다음으로, TV(R)의 모든 영역이 선택될 때까지, 선택된 영역과 대응된 페이지와 공간 조인 윈도우에 포함된 페이지들을 서로 조인한다.

이 과정을 좀 더 자세히 설명하면 다음과 같다. TSVJ는 먼저 SFC 순서에 따라 영역들을 선택하는 우선 순위 큐(priority queue)구조인 *sfc_priority_queue*를 생성한다(Line 1). 다음으로, *sfc_priority_queue*에 TV(R)의 루트 페이지 영역을 집어넣는다(Line 2). 다음으로, *sfc_priority_queue*로부터 영역 *r_region*을 하나 뽑고(Line 4), 이와 대응되는 페이지를 *r_page*라 부른다(Line 5). 이를 *sfc_priority_queue*가 빌 때까지 반복한다(Line 3). 만약 *r_page*가 단말 페이지라면 알고리즘은 *r_page*를 SJWP(*r_region*)에 속하는 모든 페이지들과 서로 조인시킨다(Line 6-9). 만약 *r_page*가 비단말 페이지이고 SJWP(*r_region*)이 빈 집합(empty set)이 아니라면 알고리즘은 *r_page*에 속한 자식 페이지들의 모든 영역들을 *sfc_priority_queue*에 추가시킨다 (Line 10-14). 이때, SJWP(*r_region*)이 빈 집합인지 확인함으로써, 앞으로 조인되지 않을 모든 *r_region*들을 미리 버린다. 조인 과정 중에서 *sfc_priority_queue*의 크기는 매우 작은 크기를 가지는데, 이는 *sfc_priority_ queue*안에 들어 있는 영역들이 SFC 순서에 따라 선택되기 전까지는 자식 페이지들의 영역들로 확장되지 않은 채 하나의 영역으로 들어 있기 때문이다. 본 논문의 실험에 의하면 *sfc_priority_queue*의 크기는 4 Kbytes 이하였다.

지금까지는 2차원 변환공간에서의 공간 조인 윈도우와 공간 채움 곡선에 기반한 2차원 TSVJ를 설명하였다. *2n*차원 TSVJ는 공간 조인 윈도우와 공간 채움 곡선을 *2n*차원으로 확장함으로써 정의된다. *2n*차원 공간 조인 윈도우는 *n*개의 2차원 공간 조인 윈도우의 카티션 곱(cartesian product)으로 정의되는데, 이는 정의 2, 보조정리 1, 2와 정리 1을 확장하여 정의 4, 보조정리 3, 4와 정리 2로 유도된다. *2n*차원 공간 채움 곡선으로는 **힐버트 순서**(*Hilbert order*)[23]가 사용된다. 힐버트 순서는 공간을 가장 잘 인접하게 순서화하는 것으로 알려져 있어 TSVJ에 적합하다[7].

정의 4 *2n*차원 TS(R)내의 초월 사각형(hyper-rectangle) P에 대한 **공간 조인 윈도우**(*spatial join window*) SJW(P)는 초월 사각형 P에 포함될 수 있는 모든 객체들과 교차관계를 가질 수 있는 모든 객체들이 존재하는 TS(S)내의 최소 영역이다.

```
Algorithm TSVJ(TV(R), TV(S), SFC)
    /* create sfc_priority_queue , which is a priority queue whose
       elements are ordered by SFC */
1   SFCPriorityQueue sfc_priority_queue(SFC)
    /* enqueue the region of the root page of TV(R) into
       sfc_priority_queue */
2   sfc_priority_queue.Enqueue(GetRootPageRegion(TV(R)))
3   while not sfc_priority_queue.IsEmpty() do
        /* dequeue a region r_region from sfc_priority_queue */
4       r_region = sfc_priority_queue.Dequeue()
5       let r_page be the page corresponding to r_region
        /* if r_page is a leaf page, join r_page with every page in
           SJWP(r_region) */
6       if r_page is a leaf page then
7           for each leaf page s_page in SJWP(r_region) in TV(S) do
8               JoinPage(r_page, s_page)
9           end
        /* if r_page is a non-leaf page, and SJWP(r_region ) is not
           empty, enqueue all the regions of the children in
           r_page into sfc_priority_queue */
10      else if r_page is a non-leaf page and SJWP(r_region) in
            TV(S) ≠ø then
11          for each region child_page_region in r_page do
12              sfc_priority_queue.Enqueue(child_page_region)
13          end
14      end
15  end
```

그림 7 Transform-Space View Join (TSVJ) 알고리즘

보조정리 3 *2n*차원 TS(R)내의 초월 사각형 (hyper rectangle) 영역 P에서, *2i-1*번째 차원축의 최소값을 ld_i, *2i*번째 차원축의 최대값을 rd_i라 할 때, 점 $<ld_1, rd_1, ld_2, rd_2, ..., ld_n, rd_n>$을 *q*라 하자. 이때, 영역 P에 대한 SJW(P)는 점 *q*와 대응되는 원공간 객체 $\hat{q}$과 원공간에서 교차관계를 가질 수 있는 모든 객체들이 존재하는 TS(S)내의 최소 영역이다. *n*이 1인 경우, 점 $q=<ld_1, rd_1>$은 2차원 사각형 영역 P의 좌상점에 해당되므로, 보조정리 3은 보조정리 1과 같은 의미를 가진다.

증명: *2n*차원 초월 사각형 영역 P에 포함되는 임의의 점 $<ld_1', rd_1', ld_2', rd_2', ..., ld_n', rd_n'>$을 *u*라 하자. 점 *u*는 영역 P안에 존재하므로, $1 \le i \le n$에 대해 $ld_i \le ld_i'$와 $rd_i' \le rd_i$의 관계가 성립한다. 점 *u*는 변환공간상의 대각선 위에 존재하므로 $ld_i' \le rd_i'$의 관계가 성립하므로, 모든 $1 \le i \le n$에 대해 $ld_i \le ld_i' \le rd_i' \le rd_i$의 관계가 성립한다. 이 관계에 의해 점 *q*와 대응되는 원공간 객체 $\hat{q}$은 점 *u*와 대응되는 원공간 객

2) 겹치는 영역들에 객체들을 중복 저장하는 원공간 색인에서는 중복을 제거하는 추가 연산을 수행해야 한다. 이러한 추가 연산은 원공간 색인들을 직접 조인할 때도 필요한 연산이다.

체 $\hat{u}$을 항상 포함한다. 원공간 객체 $\hat{q}$은 영역 P에 존재하는 모든 원공간 객체를 포함하는 가장 큰 객체이므로, TS(S)내의 객체들이 원공간 객체 $\hat{u}$과 교차하기 위해서는 $\hat{q}$과 반드시 교차해야 한다. □

보조정리 4 *2n*차원 TS(R)내의 점 $q=<ld_1, rd_1, ld_2, rd_2, ..., ld_n, rd_n>$과 대응되는 원공간 객체 $\hat{q}$과 원공간에서 교차관계를 가질 수 있는 모든 객체들이 존재하는 TS(S)내의 최소 영역은 $[0, rd_1] \times [ld_1, 1] \times [0, rd_2] \times [ld_2, 1] \times ... \times [0, rd_n] \times [ld_n, 1]$이다. n이 1인 경우, 보조정리 4는 보조정리 2와 같은 의미를 가진다.

증명: TS(R)내의 점 q와 대응되는 원공간 객체 $\hat{q}$과 점 $u=<ld_1', rd_1', ld_2', rd_2', ..., ld_n', rd_n'>$과 대응되는 원공간 객체 $\hat{u}$이 서로 교차관계에 있을 때, 이 두 객체들은 서로 떨어져 있지 않은 관계를 가지므로 *not* $(ld_1 > rd_1' \text{ or } ld_1' > rd_1 \text{ or } ... \text{ or } ld_i > rd_i' \text{ or } ld_i' > rd_i \text{ or } ... \text{ or } ld_n > rd_n' \text{ or } ld_n' > rd_n)$의 관계를 만족한다. 여기서 $ld_i > rd_i'$과 $ld_i' > rd_i$는 원공간의 i번째 차원 축에 대해 두 객체 $\hat{q}$과 $\hat{u}$이 서로 떨어져 있음을 뜻한다. 이 관계를 풀어 쓰면 $(ld_1 \le rd_1' \text{ and } ld_1' \le rd_1 \text{ and } ... \text{ and } ld_i \le rd_i' \text{ and } ld_i' \le rd_i \text{ and } ... \text{ and } ld_n \le rd_n' \text{ and } ld_n' \le rd_n)$이 된다. 이 관계에 의해 ld_i'은 구간 $[0, rd_i]$에 존재하고, rd_i'은 구간 $[ld_i, 1]$에 존재하므로, $\hat{q}$과 교차하는 모든 객체들이 존재하는 TS(S)내의 최소 영역은 $[0, rd_1] \times [ld_1, 1] \times [0, rd_2] \times [ld_2, 1] \times ... \times [0, rd_n] \times [ld_n, 1]$이다. □

정리 2 *2n*차원 TS(R)내의 초월 사각형 영역 P에서, 점 $<ld_1, rd_1, ld_2, rd_2, ..., ld_n, rd_n>$을 q라 하자. 여기서, ld_i는 $2i-1$번째 차원축에 대한 P의 최소값이고, rd_i는 $2i$번째 차원축에 대한 P의 최대값이다. 이때, 영역 P에 대한 ***2n*차원 공간 조인 윈도우** SJW(P)는 *2n*차원 구석점 변환공간 TS(S)내의 영역 $[0, rd_1] \times [ld_1, 1] \times [0, rd_2] \times [ld_2, 1] \times ... \times [0, rd_n] \times [ld_n, 1]$이다. n이 1인 경우, 정리 2는 정리 1과 같은 의미를 가진다.

증명: 정의 4와 보조정리 3과 4에 의해 성립함. □

5. 실 험

본 장에서는 TSVJ와 원공간 조인 알고리즘들의 성능을 비교한다. 원공간 조인 알고리즘으로는 R 트리를 사용한 대표적인 공간 조인 알고리즘인 Brinkhoff의 DFRJ[1]와 Huang의 BFRJ[3]를 선택하였다. BFRJ는 Combo1과 Combo2로 구분되는데, 버퍼크기가 매우 작을 때를 제외하고는 Combo2의 성능이 Combo1에 비해 많이 좋기 때문에 본 논문에서는 Combo2를 비교 대상으로 선택하였다. 본 장에서 비교하는 알고리즘들을 정리하면 표 1과 같다.

본 논문에서는 성능 평가의 척도(measure)로서 디스크 액세스 횟수를 사용한다. I/O 비용이 전체 조인 성능을 좌우하기 때문에 CPU 비용은 척도에서 제외되었다. 실험 데이타로는 균일(uniform), 지수(exponential), 그리고 실제(real) 분포를 가지는 데이타들을 사용한다. 이 데이타들은 Song 등[7]이 사용한 실험 데이타와 동일하다. 균일 분포를 가지는 실험 데이타는 두 데이타 집합 U1과 U2로 구성되고, 지수 분포를 가지는 실험 데이타는 두 데이타 집합 E1과 E2로 구성된다. 각 데이타 집합은 13만개의 MBR들로 구성된다. U1과 U2내의 MBR들의 중심점들은 전체 공간에 대해 균일 분포를 가지고, E1과 E2내의 MBR들의 중심점들은 각 축에서의 평균값이 1/4인 지수 분포를 가진다. 데이타 집합들내의 MBR들의 크기는 각 축에 대해 0에서 k사이의 균일 분포를 가진다. k의 값은 균일, 지수, 실제 분포에 대한 공간 조인 결과 수가 비슷한 값을 가지도록 조정된 값을 사용한다. U1과 U2의 공간 조인 결과 수는 87,434개이고 k는 1/440이다. E1과 E2의 공간 조인 결과 수는 87,346개이고 k는 1/920이다. 실제 분포를 가지는 실험 데이타는 두 데이타 집합 tiger1과 tiger2로 구성된다. 이 데이타는 Brinkhoff 등[1]과 Huang 등[3]이 사용한 실험 데이타와 동일한 것으로서, tiger1은 캘리포니아 지역의 도로 131,461개의 MBR로 이루어진 데이타 집합이고 tiger2는 강과 철도 128,971개의 MBR로 이루어진 데이타 집합이다. tiger1과 tiger2의 공간 조인 결과 수는 다른 데이타 분포의 공간 조인 결과와 비슷한 86,094개이다.

표 1 실험에 사용된 알고리즘들

이 름	의 미
TSVJ	변환공간 뷰 조인 알고리즘
DFRJ	깊이 우선 탐색 R 트리 공간 조인 알고리즘 [1]
BFRJ	넓이 우선 탐색 R 트리 공간 조인 알고리즘: Combo2 [3]

실험에서 사용한 공간 색인은 Brinkhoff 등[1]이 사용한 것과 동일한 R^* 트리를 사용하였다. 사용된 R^* 트리의 페이지 크기는 1 KBytes이다.

5.1 DFRJ와 TSVJ의 비교

그림 8은 각각 균일 분포, 지수 분포, 실제 분포를 가지는 데이타에 대해 TSVJ와 DFRJ의 성능을 비교한 것이다. 그래프에서 맨 밑에 있는 수평선은 최적 즉, 각 디스크 페이지들이 단 한번만 디스크로부터 읽혔을 때의 디스크 액세스 횟수를 나타낸다. 실험 결과, 제안하는 TSVJ는 모든 데이타 분포에 대해 디스크 액세스 횟수 측면에서 DFRJ보다 좋은 성능을 보인다.

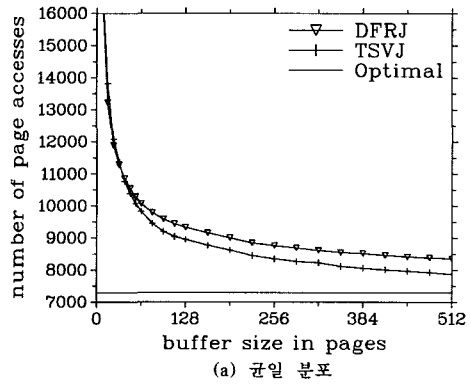
작은 버퍼를 70~140 페이지(조인 대상인 두 색인의 1~2% 크기)에 해당하는 버퍼로 정의하고, 큰 버퍼를 650~730 페이지(9~10%)에 해당하는 버퍼로 정의할

때, 그림 8에 의하면, TSVJ는 작은 버퍼 크기에서 DFRJ와 비교하여 균일 분포에서 디스크 액세스 횟수를 2.7~3.7% 줄였고, 지수 분포에서 3.4~4.1% 줄였으며, 실제 분포에서 13.4~15.4% 줄였다. 큰 버퍼 크기에서는 균일 분포에서 1.8~2.0% 줄였고, 지수 분포에서 5.3~5.5% 줄였으며, 실제 분포에서 9.2~9.7% 줄였다.

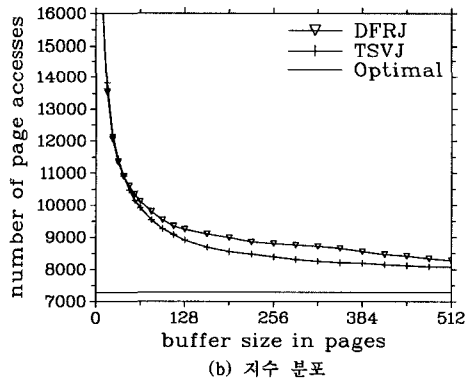
TSVJ가 DFRJ보다 좋은 이유는 다음과 같이 생각할 수 있다. TSVJ는 변환공간상의 특성을 활용하여 페이지와 공간 조인 윈도우의 순서를 정형적으로 제어하여 전역 최적화를 수행하는 최적화된 알고리즘이다. 반면, DFRJ는 plane-sweeping, pinning, 그리고 local z-ordering등의 휴리스틱 기법들만을 사용하여 디스크 페이지들의 순서를 제어하는 지역적 최적화만을 수행하기 때문에 디스크 비용이 상대적으로 크다.

5.2 BFRJ와 TSVJ의 비교

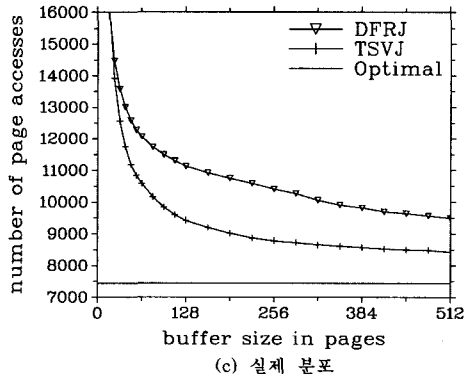
BFRJ는 pin/unpin이라는 방법으로 LRU 버퍼를 제어



(a) 균일 분포

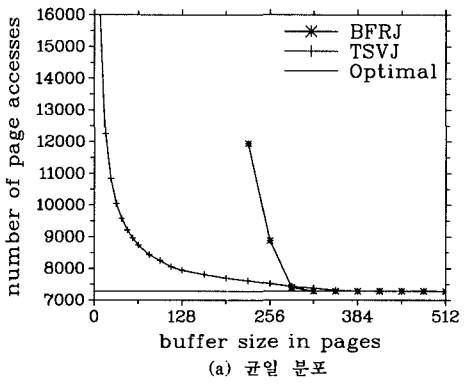


(b) 지수 분포

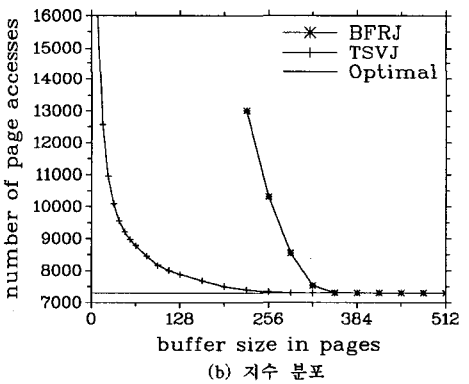


(c) 실제 분포

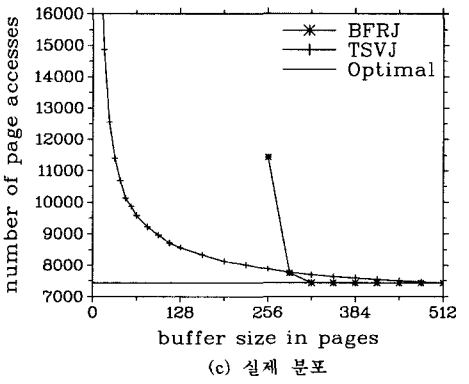
그림 8 균일, 지수, 실제 분포 데이타에 대한 DFRJ와 TSVJ의 디스크 액세스 횟수



(a) 균일 분포



(b) 지수 분포



(c) 실제 분포

그림 9 균일, 지수, 실제 분포 데이타에 대한 BFRJ와 TSVJ의 디스크 액세스 횟수

하여 버퍼의 효율을 높이는 방법을 사용한다. Pin 연산은 버퍼에 다시 읽힐 것이라고 예상되는 페이지들을 다시 읽힐 때까지 버퍼에 고정하는 연산이고, unpin 연산은 pin 연산을 해제하는 연산이다. 공평한 비교를 위해 TSVJ에도 이 기법을 적용한다.

그림 9는 pin/unpin기법을 사용하는 LRU 버퍼에서 균일 분포, 지수 분포, 실제 분포를 가지는 데이타에 대해 TSVJ과 BFRJ의 성능을 비교한 것이다. 실험 결과, 제안하는 TSVJ은 모든 데이타 분포에 대해 디스크 액세스 횟수 측면에서 BFRJ보다 좋은 성능을 보인다. 일부 버퍼 크기에서 약간 성능이 떨어지는데 그 차이는 미미하다.

그림 9에 의하면, 디스크 액세스 횟수 측면에서 작은 버퍼 크기(조인 대상인 두 색인의 1~2% 크기)의 성능 비교는 수행하지 못했다. 이는 주어진 실험 데이타에 대해 BFRJ가 동작하기 위해서는 약 200 페이지 이상의 버퍼가 필요한데, 작은 버퍼 크기는 이 보다 작기 때문이다. 작은 버퍼 크기 대신에 210~290 페이지(조인 대상인 두 색인의 3~4% 크기)에 해당하는 버퍼를 사용하여 비교하면, TSVJ는 BFRJ와 비교하여 균일 분포에서 디스크 액세스 횟수를 -0.5~36.3% 줄였고, 지수 분포에서 14.6~43.1% 줄였으며, 실제 분포에서 -0.3~31.1% 줄였다. 큰 버퍼 크기는 원패스 버퍼 크기보다 크므로 모든 분포에서 동일한 성능, 즉 최적 성능을 보였는데, 균일 분포와 지수 분포에서는 352 페이지 (조인 대상인 두 색인의 5% 크기)이상에서, 실제 분포에서는 480 페이지 (조인 대상인 두 색인의 6.8% 크기) 이상에서 동일한 성능을 보였다. 그러나, 버퍼크기가 작을 때는 큰 성능 차이를 보였는데, 이러한 성능 차이는 여러 조인 연산이 동시에 발생하는 다사용자 환경에서 특히 중요하다. 왜냐하면, 작은 버퍼 크기에서 성능이 좋은 알고리즘은 한정된 버퍼 크기에서 더 많은 조인들을 그렇지 않은 알고리즘들보다 더 좋은 성능으로 수행할 수 있기 때문이다. TSVJ가 BFRJ보다 좋은 이유는 TSVJ가 변환공간상의 특성을 활용하여 페이지와 공간 조인 윈도우의 순서를 정형적으로 제어하는 전역 최적화를 수행하기 때문이다.

6. 결 론

본 논문에서는 첫째, **변환공간 뷰**라는 새로운 개념을 제안하였다. 이 개념은 원공간 색인에 대한 가상의 변환공간 색인으로서, 그 중요성은 R 트리와 같이 널리 사용되는 원공간 색인을 구조적으로 변경하지 않고서도 별도의 추가비용 없이 변환공간 색인으로 해석할 수 있게 해 줌에 있다. 이 변환공간 뷰는 영역과 객체를 MBR(minimum bounding rectangle)들로 관리하는 모든 원공간 색인(예를 들어, R-tree[10], R*-tree[9], X-tree[13], SKD-Tree[14], GBD-Tree[15] 등)에 적용될 수 있다.

둘째, 변환공간 뷰를 사용한 **변환공간 뷰 조인** (*transform-space view join: TSVJ*) 알고리즘을 제안하였다. 이 알고리즘은 Song 등[7]이 제안한 변환기반 공간 조인 알고리즘을 개선한 것이다. 또한, 정의 4, 보조정리 3과 4, 정리 2를 통해 알고리즘을 $2n$차원으로 일반화하였다.

셋째, 분석과 실험을 통해, TSVJ의 우수성을 보였다. R 트리를 원공간에서 조인하는 알고리즘들과 비교하여 TSVJ는 디스크 액세스 횟수를 최대 43.1%까지 줄였다.

본 논문의 가장 큰 공헌은 R 트리와 같이 널리 사용되는 원공간 색인을 변환공간 뷰라는 새로운 개념을 통하여 변환공간에서 해석하여 사용할 수 있음을 보인 것이다. 우리는 이 새로운 개념이 다양한 공간 질의 처리 알고리즘들이 변환공간에서 새롭게 개발될 수 있는 프레임워크를 마련했다고 믿는다.

참 고 문 헌

[1] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient Processing of Spatial Joins Using R-Trees," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 237-246, May 1993.

[2] O. Günther, "Efficient Computation of Spatial Joins," In *Proc. the Ninth Int'l Conf. on Data Engineering*, pp. 50-59, 1993.

[3] Y.-W. Huang and N. Jing, "Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations," In *Proc. the 23rd Int'l Conf. on Very Large Data Bases*, pp. 396-405, 1997.

[4] M.-L. Lo and C. V. Ravishankar, "Spatial Joins Using Seeded Trees," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 209-220, May 1994.

[5] M.-L. Lo and C. V. Ravishankar, "Spatial Hash-Joins," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 247-258, June, 1996.

[6] J. M. Patel and D. J. Dewitt, "Partition Based Spatial-Merge Join," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 259-270, June 1996.

[7] J. Song, K. Whang, Y. Lee, M. Lee, and S. Kim, "Spatial Join Processing Using Corner Transformation," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 11, No. 4, pp. 688-695, 1999.

[8] N. Mamoulis and D. Papadias, "Slot Index Spatial Join," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 15, No. 1, pp. 211-231, 2003.

[9] N. Beckmann, H.-P. Kriegel, and R. Schneider, "The R*-tree: An Efficient and Robust Access

Method for Points and Rectangles," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 322-331, 1990.

[10] A. Guttman, "R-trees: a Dynamic Index Structure for Spatial Searching," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp 47-57, 1984.

[11] J. Lee, Y. Lee, K. Whang, and I. Song, "A Region Splitting Strategy for Physical Database Design of Multidimensional File Organizations," In *Proc. the 23rd Int'l Conf. on Very Large Data Bases*, pp. 416-425, 1997.

[12] K. Whang and R. Krishnamurthy, Multilevel Grid Files, IBM Research Report RC 11516, 1985.

[13] S. Berchtold, D. Keim, and H.-P. Kriegel, "The X-tree: An Index Structure for High-Dimensional Data," In *Proc. the 22nd Int'l Conf. on Very Large Data Bases*, pp. 28-39, 1996.

[14] B. C. Ooi, K. J. Mcdonell, and R. Sacks-Davis, "Spatial kd-tree: An Indexing Mechanism for Spatial Databases," In *Proc. IEEE Computer Software and Applications Conference*, pp. 433-438, 1987.

[15] Y. Ohsawa and M. Sakauchi, "A New Tree Type Data Structure with Homogeneous Node Suitable for a Very Large Spatial Database," In *Proc. the Sixth IEEE Int'l Conf. on Data Engineering*, pp. 296-303. 1990.

[16] H. V. Jagadish, "Spatial Search with Polyhedra," In *Proc. the 16th IEEE Int'l Conf. on Data Engineering*, pp. 311-319, 1990.

[17] O. Günther, "The Cell Tree: An Object-Oriented Index Structure for Geometric Databases," In *Proc. the 15th IEEE Int'l Conf. on Data Engineering*, pp. 598-605, 1989.

[18] K. Hinrichs and J. Nievergelt, "The Grid File: A Data Structure Designed to Support Proximity Queries on Spatial Objects," In *Proc. Int'l Workshop on Graph Theoretic Concepts in Computer Science*, pp. 100-113, 1983.

[19] B. Seeger and H.-P. Kriegel, "Techniques for Design and Implementation of Efficient Spatial Access Methods," In *Proc. the 14th Int'l Conf. on Very Large Data Bases*, pp. 360-371, 1988.

[20] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addision-Wesley Publishing Company, Inc., 1990.

[21] V. Gaede and O. Gnther, Multidimensional Access Methods, *ACM Computer Surveys*, Vol. 30, No. 2, pp. 170-231, 1998.

[22] H. V. Jagadish, Linear Clustering of Objects with Multiple Attributes, In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 332-342, May 1990.

[23] J. K. Lawder and P. J. H. King, Querying Multi-Dimensional Data Indexed Using the Hilbert Space-filling Curve, *SIGMOD Record*, Vol.30, No.1, pp. 19-24, 2001.

이 민 재

1995년 2월 한국과학기술원 전자전산학과 전산학전공 학사. 1997년 2월 한국과학기술원 전자전산학과 전산학전공 석사 1997년 3월~현재 한국과학기술원 전자전산학과 전산학전공 박사 과정. 관심분야는 지리 정보 시스템, 객체 관계형 데이타베이스 시스템

한 욱 신

2003년 3월~현재 전임강사, 컴퓨터공학과, 경북대학교. 2002년 9월~2003년 2월 연구교수, 첨단정보기술연구센터, KAIST 2001년 9월~2002년 8월 포스트닥, 첨단정보기술연구센터, KAIST. 1996년 3월~2001년 8월 Ph.D, 전산학과, KAIST 1994년 3월~1996년 2월 MS, 전산학과, KAIST. 1990년 3월~1994년 2월 BS, 컴퓨터공학과, 경북대학교. 2002년 7월 방문연구원, 미국 HP Labs. 2001년 7월~2001년 8월 Visiting Scholar, 미국 HP Labs

황 규 영

1973년 서울대학교 전자공학과 졸업(B.S). 1975년 한국과학기술원 전기 및 전자학과 졸업(M.S.). 1982년 Stanford University(M.S.). 1983년 Stanford University(Ph.D.). 1975년~1978년 국방과학연구소(ADD), 선임연구원. 1983년~1990년 IBM T.J. Watson Research Center, Research Staff Member. 1992년~1994년 한국정보과학회 데이타베이스 연구회(SIGDB) 운영위원장. 1995년 한국정보과학회 이사 겸 논문지 편집위원장. 1999년~2000년 한국정보과학회 부회장. Editior: the VLDB Journal, 1990년~현재 Editior: Distributed and Parallel Databases: An International Journal, 1991년~1995년 Editor: International Journal of Geographical Information Systems, 1994년~현재 Associate Editor: IEEE Data Engineering Bulletin, 1990년~1993년 IEEE Transactions on Knowledge and Data Engineering, 2002년~현재. 1998년~2004년 Trustee, The VLDB Endowment. 1999년~2005년 Steering Committee Member, DASFAA. 1999년~현재 한국과학기술원 전자전산학과 전산학전공 교수 1990년~현재 첨단정보기술연구센터(과학재단 우수연구센터) 소장. 관심분야는 데이타베이스 시스템, 멀티미디어, GIS