그리드 기반의 질의 색인을 통한 효율적인 연속 영역 질의 처리

박 용 훈[†] · 복 경 수[††] · 유 재 수[†††]

요 약

본 논문에서는 기존 그리드 기반의 질의 색인 기법을 변형하여 보다 적은 저장 공간을 사용하면서 보다 빠른 연산을 수행하는 연속 영역 질의 처리 기법을 제안한다. 제안하는 기법의 주요 특징은 두 가지 이다. 첫째, 각 질의에 비트 식별자를 부여하고 그리드의 각 셀은 이러한 비트 식별자의 조합으로 이루어진 비트 열을 이용하여 질의들의 겹침 정보를 반영한다. 이러한 비트 열을 통해 셀이 어떤 질의들에 포함되어져 있는지 빠르게 판단할 수 있으며, 두 셀 사이의 각 셀을 포함하는 질의 식별자들을 비교하지 않고 비트 열만을 비교하여 질의들의 포함관계를 알아내어 불필요한 연산을 줄일 수 있다. 둘째, 셀들을 그룹단위로 관리하여 불필요하게 비트 열의 길이가 증가하여 저장 공간을 낭비하고 비트 열의 비교 연산 시간이 증가하는 문제를 해결한다. 제안하는 기법이 기존 연속 영역 질의 처리 기법에 비해 우수함을 성능 평가를 통해 입증한다.

키워드 : 연속 영역 질의, 질의 색인, 이동 객체 그리고 위치 기반 서비스

An Efficient Continuous Range Query Processing Through Grid based Query Indexing

Yong-Hun Park[†] · Kyoung Soo Bok[††] · Jae Soo Yoo[†††]

ABSTRACT

In this paper, we propose an efficient continuous range query processing scheme using a modified grid based query indexing to reduce storage spaces and to accelerate processing time. The proposed method has two major features. First, each query has a bit identifier and each cell in a grid has a bit pattern that consists of the bit identifiers of the queries. The bit patterns present the relationship between cells and queries. Using the bit patterns, we can compute quickly what queries overlap a cell in a grid and reduce the number of unnecessary operations by comparing the bit patterns without comparing the query identifiers when we compute the relation between cells and queries. Second, the management of cells in the grid by groups prevents from wasting the storage space through the increase of the length of the bit pattern and increasing the comparison costs of bit patterns. We show through the performance evaluation that the proposed method outperforms the existing methods.

Key Words : Continuous Range Queries, Query Indexing, Moving Objects, Location Based Services

1. 서 론

이동 객체에 대한 연속 질의를 처리하기 위해 객체 색인(object indexing)을 이용하는 많은 연구들이 진행되어 왔다[1, 2, 3, 4]. 객체 색인 기법은 연속 질의를 처리하기 위해 객체의 위치 변화에 따라 주기적으로 질의를 재실행하여 질의 결과를 매번 계산을 해야만 한다. 그러나 변경된 객체의 개수와 무관하게 전체 색인을 대상으로 매번 질의를 처리하기 때문에 많은 통신비용을 소모하며 질의들 사이의 포함 관계 또는 객체와 질의의 포함 관계를 판별하기 위해 많은 연산 시간을 소요한다. 또한, 주기적인 재실행을 하기 때문에 주기 간에 갱신된 객체에 대해 질의 결과에 대한 반영이 주기적으로 이루어져 질의 결과들에 대한 정확성을 보장하지 못한다. 질의 색인(query indexing) 기법은 기존의 객체 색인과 달리 질의 자체를 색인으로 구성하기 때문에 객체의 갱신 과정에서 객체의 갱신이 질의 결과에 영향을 미치는지 아닌지를 빠르게 판단할 수 있으며 질의 결과에 영향을 미치는 갱신에 한해서만 해당 질의의 결과를 변경하는 연산을 수행한다[5, 6, 7]. 이로 인해 객체의 갱신으로 인해 영향을 받는 모든 질의에 대한 동시적인 처리가 가능하다.

※ 이 논문은 2007년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원(지방연구중심대학육성사업/충북BIT연구중심대학육성사업단)과 정부(과학기술부)의 재원으로 한국과학재단(특정기초연구 과제번호 R01-2006-000-10809000) 지원에 의해 연구되었음.

† 준 회 원 : 충북대학교 전기전자컴퓨터공학부 정보통신공학과 박사과정
†† 준 회 원 : 한국과학기술원 전산학과 POSTDOC 연구원
††† 종신회원 : 충북대학교 전기전자컴퓨터공학부 정교수 (교신저자)

논문접수 : 2006년 11월 1일, 심사완료 : 2007년 1월 12일

최근 몇 년간 연속 영역 질의 처리를 위해 많은 연구가 진행되어 왔다. 질의의 이동성 고려 여부에 따라 연속 영역 질의 처리에 대한 기존 연구는 크게 두 가지로 분류할 수 있다. 첫 번째는 이동하는 객체에 대해 질의가 고정되어 있는 상황에서의 연속 영역 질의 처리[6, 7, 8, 9, 10]이고 두 번째는 질의 자체가 이동하는 것을 고려한 연속 영역 질의 처리[11, 12]이다. 본 논문은 질의 자체가 이동하는 않는 정적인 질의만을 고려하고 있다. 연속 영역 질의를 처리하기 위해서는 질의 색인 기법이 필요하다. 최근까지 연속 영역 질의를 처리하기 위한 질의 색인 기법에 관련된 많은 연구들이 진행되었다. 연속 영역 질의를 처리하기 위해 객체의 이동에 따라 질의 결과에 변화를 디스크 기반 질의 색인을 통해 수행할 경우 빈번한 I/O가 발생하며, 이로 인해 질의 처리에 많은 시간이 소요된다. 디스크 기반 색인 기법으로는 R-tree와 안정 영역(safe region) 개념을 이용한 Q-Index[9], 질의의 이동을 함께 고려한 SINA[11] 그리고 질의 영역을 구분하기 위해 color개념을 도입한 cGridex[10]를 들 수 있다. 객체의 갱신에 따른 질의 결과의 변화를 빠르게 처리하기 위해서는 질의 색인을 메모리에 유지하는 메모리 기반의 색인구조가 필요하다. 메모리 기반 색인 기법으로는 셀 기반 색인(cell-based index)인 CQI-Index[6] 그리고 가상 셀 기반 색인(virtual tile-based index)인 VCR-Index[7]와 CES-Index[8]가 있다.

메모리 기반의 색인구조와 디스크 기반의 색인구조 사이에는 두 가지 큰 차이점이 있다. 메모리는 속도는 빠르지만 가격적인 측면이나 하드웨어 적인 측면에서 용량이 제한되어 있어서 색인의 유지비용을 최대한 줄이는 것이 중요하다. 디스크 기반의 색인 기법은 색인의 응답 속도를 빠르게 처리하기 위해 디스크의 I/O횟수를 줄이는 것이 중요하지만 메모리 기반의 색인구조에서는 더 이상 디스크를 이용하지 않기 때문에 불필요한 연산을 줄이는 것이 중요하다. 따라서 메모리 기반에서 고려해야 할 점은 색인의 저장 공간 비용과 불필요한 연산을 줄이는 것이다. 기존의 메모리 기반 질의 색인 기법들은 접근 방법에 따른 성능상의 한계를 지니고 있다. 첫 번째로 객체가 어떤 질의의 범위로부터 벗어났는지 아니면 어떤 질의의 범위로 새롭게 포함되었는지를 파악하기 위해 객체가 이전에 포함되었던 질의들과 현재 포함하는 질의들을 질의 식별자를 이용하여 비교한다. 이때, 객체의 식별자를 이용하여 비교 연산을 하여야하기 때문에 많은 연산 비용을 소요한다. 두 번째로 메모리 기반 색인 구조가 대부분 그리드 기반으로 구성되기 때문에 그리드에 포함된 질의를 유지하기 위해서는 다수의 그리드에 질의의 식별자를 중복적으로 유지하여 저장 공간의 낭비를 초래한다.

본 논문에서는 연속 영역 질의 처리를 위해 기존에 제안된 메모리 기반 질의 색인 기법의 문제점을 획기적으로 개선하여 보다 적은 유지비용으로 보다 빠른 연산을 수행하는 질의 색인 구조를 제안한다. 제안하는 질의 색인 기법은 객체의 이동에 따른 질의 결과를 변화를 빠르게 처리하는데 초점을 두었으며 메모리 기반의 그리드 구조로 구성한다. 질의 색인을 기반으로 연속 질의를 처리하기 위한 연산 비용을 최소화하기 위해 제안하는 연속 질의 처리 기법은 비트 식별자(Bit Identifier)라는 개념을 사용한다. 비트 식별자는 각 질의에 부여된 식별자로 그리드 색인의 각 셀은 질의들의 겹침 정보를 반영하는 비트 열을 가지고 있다. 이러한 비트 열을 통해 객체가 어떤 질의에 포함되었는지 파악할 수 있으며 셀 사이의 비트 열을 비교하여 질의들 간의 포함 관계를 파악할 수 있다. 이러한 처리 과정은 대량의 객체에 대해 질의를 처리할 경우 큰 성능 향상을 보이며 기존 메모리 기반 기법에 비해 연산 비용을 감소시킬 수 있다. 제안하는 연속 질의 처리 기법은 저장 공간과 비트 열의 길이를 효율적으로 관리하기 위해 셀들을 그룹 단위로 관리하고 각 그룹은 그룹에 포함된 질의들을 관리한다. 이로 인해, 질의 식별자들의 중복 저장을 최소화하여 색인의 유지비용을 감소시킨다.

본 논문의 나머지 구성은 다음과 같다. 먼저 2장에서는 관련 연구로서 기존에 제안된 메모리 기반의 연속 영역 질의 색인 기법들에 대해서 설명하고, 3장에서는 본 논문에서 제안하는 질의 색인 기법과 질의 색인을 통해 연속 영역 질의를 처리 기법을 구체적으로 설명한다. 4장에서는 기존에 제안된 연속 질의 처리 기법과의 성능 평가를 통해 제안하는 연속 영역 질의 기법의 우수성을 입증하고 마지막 5장에서는 논문의 결론 및 향후 연구에 대해 기술한다.

2. 관련 연구

2.1 메모리 기반 연속 영역 질의 처리

연속 영역 질의 처리를 위한 기법은 크게 메모리 기반의 기법과 디스크 기반의 기법으로 구분할 수 있다. 최근 들어 메모리의 가격 하락과 계속적인 이동 객체의 변화에 따른 실시간 처리를 수행하기 위해 연속 질의 분야에 메모리 기반의 기법들이 많은 관심을 모으고 있다. 본 논문은 메모리 기반의 기법에 대해서 연구를 수행하였고 관련 연구로서 기존에 제안된 메모리 기반의 연속 영역 질의 처리 기법에 대해서 기술한다.

D. V. Kalashnikov는 이동 객체를 대상으로 하는 연속 영역 질의 처리를 위한 메모리 기반의 질의 색인 기법인 CQI-Index(Cell-based Query Indexing)[6]을 제안했다. CQI-Index는 전체 색인 공간이 일정한 크기의 셀로 분할되어진 그리드를 기반으로 하고 각 셀은 겹치는 질의들에 대한 식별자를 유지하는 질의 리스트 관리한다. 질의 삽입 시 겹치는 모든 셀에 존재하는 질의 리스트에 질의의 식별자를 삽입하고, 삭제 시 해당 셀들에 존재하는 질의 리스트로부터 질의 식별자를 삭제한다. 질의 리스트는 셀을 완전히 겹치는 질의를 위한 완전 리스트(Full List) 그리고 셀을 부분적으로 겹치는 질의를 위한 부분 리스트(Part List)로 분류하여 관리한다. CQI-Index는 객체의 위치가 갱신될 때마다 부분 리스트에서 실제 객체의 위치가 질의에 포함되는지를 분석해야 하기 때문에 많은 연산 비용을 소요한다.

CQI-Index 기법이 제안되어진 이후 K. L. Wu는 이동 객

체를 대상으로 연속 영역 질의 처리를 위한 메모리 기반의 새로운 질의 색인 기법인 VCR(Virtual Construct Rectangle)-Index[7]을 제안했다. VCR-Index는 CQI-Index에서 검색 속도에 문제가 되었던 부분 리스트(part list)를 없애기 위해 전체 그리드의 셀을 세분화 하여 질의와 부분적으로 겹침이 발생하지 않도록 하고, 셀을 세분화 하면서 발생하는 저장 공간의 낭비를 줄이고 검색 속도를 개선하기 위해 가상 셀(Virtual Cell)개념을 도입한다. VCR-Index는 가상 셀로서 VCR(Virtual Construct Rectangle)을 제안한다. VCR-Index는 객체의 위치를 포함하는 셀과 그 셀을 포함하는 VCR들을 이용하여 질의 검색을 수행한다. 각 VCR은 질의 식별자를 유지하는 질의 리스트 관리하고 객체의 위치를 포함하는 모든 VCR들의 질의 리스트를 이용하여 해당 객체가 어떤 질의에 포함되는지 판별한다. 객체의 이동시 객체의 이전 위치를 포함하는 VCR들과 객체의 현재 위치를 포함하는 VCR들을 비교하여 만약 동일한 VCR이 있으면 그 VCR에서 유지하는 질의는 객체의 이동에 대해서 영향을 받지 않는 것으로서 비교 대상에서 제외한다. 이러한 방식으로 질의 검색 시 불필요한 비교작업을 줄임으로서 처리속도의 향상을 가져왔다.

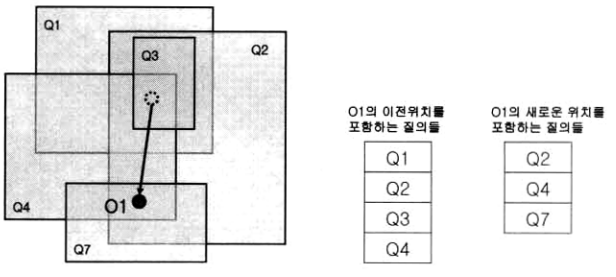
K. L. Wu는 VCR과는 다른 방식으로 구성된 가상 셀인 CES(Containment Encoded Squares)를 이용하는 향상된 질의 색인 기법인 CES(Containment Encoded Squares)-Index[8]를 제안했다. CES-Index는 VCR-Index와 질의 처리 방식은 비슷하지만 가상 셀 사이의 관계를 계층적으로 구성함으로서 객체의 위치를 포함하는 질의 검색 시 검사해야 할 가상 셀의 개수를 감소시켰으며 이로 인해 질의 처리 속도를 향상시켰다. 본 논문에서는 가장 최신에 제안된 CES-Index와 비교하여 성능 평가를 수행한다.

2.2 기존 연구의 문제점

2.2.1 질의 식별자를 이용한 질의 처리에 의한 연산 시간 소모

연속 영역 질의를 효율적으로 처리하기 위해서는 객체가 위치 갱신을 할 때, 그 객체에 대해서 질의 결과에 영향을 받는 질의들을 빠르게 파악하는 것이 중요하다. 질의 결과에 대한 갱신이 필요한 질의를 알아내는 과정은 객체의 위치가 갱신될 때마다 수행되기 때문에 많은 연산 시간을 소요하기 때문이다. 기존의 연속 질의 처리 기법은 질의 색인을 통해 이동 객체가 이전에 포함되었던 질의 리스트들과 현재 포함되는 질의 리스트들을 계산하여 어떤 질의에서 벗어나고 어떤 질의에 새로 포함되는지를 계산한다. 이러한 수행 과정에서 질의 식별자들을 하나씩 비교해야하기 때문에 많은 연산을 소모한다. 뿐만 아니라, 실제 객체의 갱신으로 질의 결과에 영향을 미치지 않는 경우에도 매번 동일한 연산을 수행하기 때문에 많은 연산 비용을 소모한다.

(그림 1)은 이동 객체의 위치 갱신에 따른 질의와의 관계를 나타낸 것이다. 이동 객체 O1의 이전 위치를 포함하는 질의들은 Q1, Q2, Q3 그리고 Q4 이고, O1의 새로운 위치를 포함하는 질의들은 Q2, Q4, 그리고 Q7 이다. O1이 이동을 함으로써 Q1과 Q3에는 더 이상 포함되지 않고 Q7에는 새



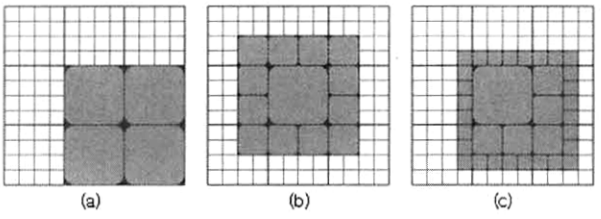
(그림 1) 객체 O1의 이동에 따른 질의와의 포함관계

롭게 포함되며 Q2와 Q4에는 여전히 포함된다. 이때, O1에 대해서 삭제를 수행하는 질의는 Q1과 Q3이 되고, 삽입을 수행하는 질의는 Q7이 된다. 결과적으로 O1의 이동에 영향을 받는 질의는 Q1, Q3, Q7이 된다. 이러한 결과를 연산하기 위해 기존의 기법은 Q2, Q4에 대한 불필요한 비교 연산을 필요로 한다.

VCR-Index와 CES-Index는 다양한 크기의 가상 셀을 이용하여 이런 불필요한 연산을 줄였다. 하지만 VCR-Index의 경우 한 위치를 포함할 가능성이 있는 가상 셀의 개수가 너무 많아서 실제 포함하는 가상 셀을 검색하는데 오히려 많은 연산비용을 소모한다. VCR-Index의 이러한 단점을 해결한 CES-Index의 경우 트리 구조를 이용하여 한 위치를 포함할 가능성이 있는 가상 셀의 개수를 줄여 질의 처리 성능을 향상 시켰지만 이것은 비교할 가상 셀의 개수를 줄이기 위해서 가상 셀 간의 중복된 질의 식별자로 인한 불필요한 연산을 허용 한데서 온 결과이다.

2.2.2 중복적인 질의 식별자 저장으로 인한 저장 공간 낭비

연속 영역 질의를 처리하기 위한 질의 색인은 질의를 관리하기 위해 질의 식별자를 유지하는데 일반적으로 메모리 기반 색인 구조는 그리드 구조를 사용하기 때문에 하나의 질의가 여러 셀에 포함되어 있을 경우 질의 식별자를 중복적으로 유지한다. CQI-Index의 경우에는 동일한 크기의 셀들로 구성된 그리드 구조를 이용하고 있기 때문에 각 셀은 포함하는 질의들의 리스트를 유지하여 질의와 셀의 겹침 관계를 표현한다. 그래서 셀의 크기를 지정하는 것이 저장 공간의 효율성과 처리 속도의 성능을 결정하는 중요한 요소가 된다. 하지만 셀에서 유지하는 부분 리스트 때문에 오는 연산 비용을 해결하기 위해 셀을 세분화 시킬 경우 질의의 크기에 따라 질의 식별자의 중복 저장에 의한 필요 저장 공간의 증가가 발생한다. VCR-Index과 CES-Index의 경우 그리드 구조를 이용하지만 CQI-Index처럼 동일한 크기의 셀이 아닌 다양한 크기를 나타내는 가상의 셀을 이용하여 질의 식별자를 저장함으로써 질의 식별자에 대한 중복적인 저장 횟수를 줄였다. VCR-Index의 경우 다양한 크기의 셀들에 대한 개수가 너무 많기 때문에 그것을 유지하는 비용이 크다. 그리고 CES-Index의 경우 계층적인 구조를 이용하여 만들어 지는 다양한 크기의 셀 개수를 제한하였다. 그러나 질의의 위치에 따라서 질의를 분할하는 셀의 개수가 다르고, 만들어 지는 가상 셀을 제한하여 VCR-Index에서의 개수 이



(그림 2) CES를 이용한 질의 분할의 예

상의 가상 셀로 질의를 분할한다. 많은 수의 가상 셀로 분할이 된다는 것은 그 만큼 질의 식별자의 중복적인 저장이 일어남을 나타낸다.

(그림 2)는 CES를 이용한 동일한 크기의 질의를 분할한 예를 나타낸다. (a), (b), (c)는 모두 동일한 크기의 질의지만 질의의 위치에 따라서 분할된 가상 셀의 개수가 다르다는 것을 볼 수 있다. 적은 개수의 가상 셀들로 분할되는 것이 그만큼 질의 식별자의 중복 저장 횟수를 줄이고, 되도록 이면 큰 가상 셀들로 분할되는 것이 질의 처리 시 중복된 질의 식별자에 대한 처리 횟수가 줄기 때문에 질의 처리를 효율적으로 수행한다. 이러한 기준으로 (a)의 경우가 (B)와 (C)에 비해 가장 이상적으로 분할된 형태이고, 이들 중 (C)가 가장 비효율적인 형태로 분할된 경우이다.

2.2.3 색인의 초기 구축비용으로 인한 저장 공간 소모

기존의 그리드 구조를 가지는 색인에서는 초기에 색인을 구축하기 위한 기본적인 저장 공간을 할당할 수밖에 없었다. 그 비용 때문에 전체 그리드 공간을 세밀하게 분할하는데 제약을 가지고 있었다. CQI-Index에서는 각 셀에서 겹치는 질의 정보를 유지하기 위한 포인터를 유지한다. 이러한 포인터 정보는 32bit 컴퓨터에서는 4byte(=32bit)이며, 비록 셀에서 유지할 질의 정보가 없더라도 색인을 구축하기 위한 초기 비용으로 소모된다. 그리드 공간을 n×m로 분할했을 때 초기 구축비용은 n×m×4byte 로 표현할 수 있다. 이러한 공식은 세분하게 분할할수록 초기 구축비용은 기하급수 적으로 증가한다는 것을 나타낸다. 예를 들어, 만약 그리드 공간을 512×512로 분할하면 1Mbyte, 1024×1024로 분할하면 4Mbyte, 2048×2048로 분할하면 16Mbyte를 초기 구축비용으로 소모한다. VCR-Index와 CES-Index 기법의 경우에는 CQI-Index 보다 초기 구축비용을 많이 필요로 한다. 이것은 그리드 공간에서 분할된 가장 작은 셀뿐만 아니라 다양한 크기의 가상 셀들을 추가적으로 만들기 때문이다. CES-Index는 VCR-Index 보다 만드는 가상 셀의 수가 적기 때문에 VCR-Index 보다는 적은 초기 구축비용을 가진다.

3. 제안하는 연속 영역 질의 처리

연속 질의 처리를 위한 기존 질의 색인 기법의 문제점을 해결하여 보다 효율적인 질의 색인을 만들기 위해 본 논문에서는 새로운 질의 색인 기법을 제안한다. 또한, 제안하는 질의 색인 기법을 이용한 연속 영역 질의 처리 기법을 제안한다. 먼저 제안하는 질의 색인 기법에 대해 설명하고 제안하는 질의 색인 기법을 이용한 연속 질의 처리 기법을 설명한다.

3.1 질의 색인 기법의 특징

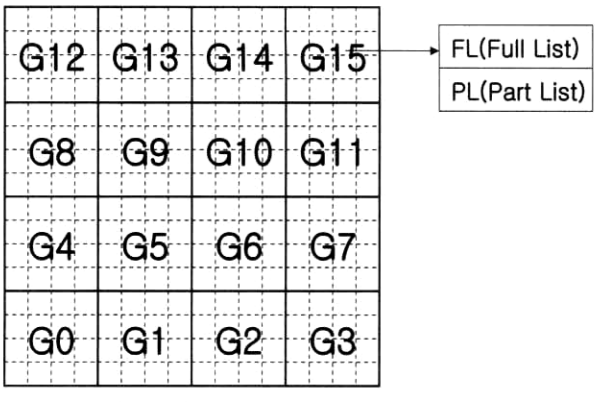
본 논문은 그리드 기반의 질의 색인 구조에서 발생하는 질의 식별자의 중복 저장 문제와 객체의 갱신으로 인해 비교해야 할 질의 수가 증가 하는 문제를 해결하기 위해 여러 셀들을 그룹 단위로 묶어서 관리하는 질의 색인 기법을 제안한다. 즉, 그룹 내에서 여러 셀들이 어떤 질의에 겹치더라도 한 그룹에서 일괄적으로 유지를 함으로서 중복적인 질의 식별자 저장 횟수를 줄이는 것이다. 그룹 안에서의 질의관리를 보다 적은 저장 공간을 사용하여 유지하고, 객체의 위치 갱신으로 영향을 받는 질의를 쉽게 파악하기 위해 비트 식별자를 활용한 방법을 제안한다. 이러한 방법은 갱신된 이동 객체에 대해서 결과 집합의 갱신이 필요한 질의들을 계산하는 과정에서 불필요한 연산을 줄인다. 과거 위치를 만족시키는 질의들과 새로운 위치를 만족시키는 질의들의 식별자를 서로 비교하지 않고 그룹 내의 각 셀에서 질의와의 포함관계를 나타내는 비트 열을 이용한 연산을 사용하여 객체의 위치 변화에 따른 질의 결과의 변화를 빠르게 처리하기 위해서이다.

제안하는 기법은 그리드 형태의 색인을 기반으로 하고 질의의 형태는 좌측 하단과, 우측 상단의 두 점을 갖는 직사각형의 형태이다. 셀은 질의를 표현할 수 있는 작은 단위로 분할되고, 질의는 셀에 대해서 부분적으로 겹치지 않는다. 그리고 각 연속 영역 질의에 대한 초기 결과 집합은 VCR-Index[7]와 CES-Index[8]등의 기존 영역 질의 처리 알고리즘과 동일하다.

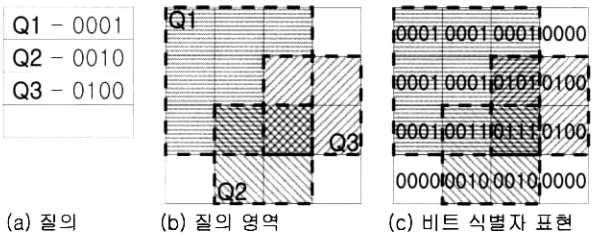
질의 색인에서 각 그룹은 전체 그리드를 분할된 정사각형 영역으로 가로와 세로의 크기는 2^n으로서 모두 동일한 크기이다. 한 그룹에서는 FL(Full List)와 PL(Part List) 두 가지 질의 리스트를 관리한다. FL은 그룹과 겹치는 질의 중에서 완전히 그룹을 포함하는 질의에 대한 리스트이다. PL은 그룹과 겹치는 질의 중에서 부분적으로 그룹과 겹치는 질의에 대한 리스트이고 뿐만 아니라 그룹이 관리하고 있는 셀들에 대한 정보도 같이 유지한다. (그림 3)는 2^4x2^4 크기의 전체 그리드를 구성하는 2^2x2^2 크기의 그룹에 대해서 표현하고 있다.

(그림 4)는 그룹 내에서 비트 식별자를 이용한 질의의 관리와 각 셀과 질의와의 겹침 관계를 표현하는 비트 열에 대해서 나타내고 있다. (a)는 그룹과 겹치는 질의들의 식별자와 각 질의들의 비트 식별자를 나타내고, (b)는 그룹 내의 질의 영역을 표현한 것이다. (c)는 그룹을 구성하는 각 셀들은 비트 열로 표현하고 이들과 질의와의 포함관계를 나타낸다. 예를 들어, Q1의 비트 식별자가 0001, Q2의 비트 식별자가 0010 그리고 Q3의 비트 식별자가 0100이라고 할 때, 아무런 질의와도 겹치지 않는 셀의 비트 열은 0000, Q1하고만 겹치는 부분의 셀의 비트 열은 0001, Q3과 Q1에 겹치는 부분의 셀은 0101이다. 반대로 셀의 비트 열을 이용하여 겹침 관계에 있는 질의들을 식별할 수 있다.

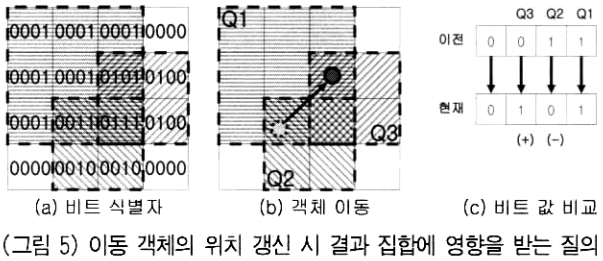
각 셀의 비트 열을 이용하여 결과 집합의 갱신이 필요한 질의들을 계산한다. 비트열의 각 자리의 비트 값은 그 자리가 1인 비트 식별자를 가지는 질의를 나타내기 때문에 객체



(그림 3) 전체 그리드를 구성하는 그룹



(그림 4) 비트 식별자



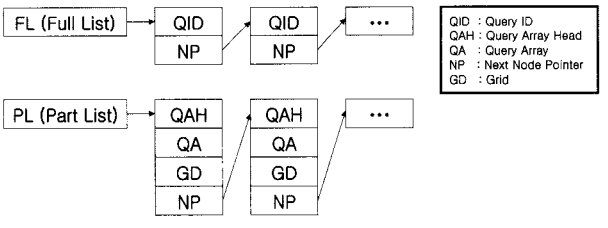
(그림 5) 이동 객체의 위치 갱신 시 결과 집합에 영향을 받는 질의

의 이동으로 인한 결과 집합의 갱신이 필요한 질의들을 알아내기 위해서 그 객체들의 이전 위치를 포함하는 셀의 비트열과 현재 위치를 포함하는 셀의 비트열을 각 자리의 비트 값 단위로 비교한다. 만약 이전 셀에서 1이고 현재 셀에서 0이면 그 식별자의 질의에서 객체는 벗어난 것이고 이전 셀에서 0이고 현재 셀에서 1이면 객체는 그 식별자의 질의로 들어간 것이다. 만약 같다면 그 객체의 이동이 해당 질의의 결과 집합에 아무런 영향을 주지 않는다.

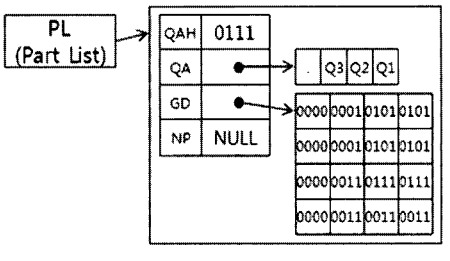
(그림 5)는 그리드 상에서 객체의 이동을 나타낸 것이다. (a)는 각 셀의 비트 열을 나타내고, (b)는 객체의 이동을 나타낸다. (c)는 각 비트 값을 비교하여 결과 집합의 갱신이 필요한 질의를 알아내는 것을 그림으로 나타낸 것이다. 객체의 이전 위치와 현재 위치 셀의 비트 열은 각각 0011과 0101이다. 비트 열을 비교한 결과로서 Q3은 객체를 결과 집합에 삽입해야 하고, Q2는 객체를 결과 집합에서 삭제해야 한다.

3.2 FL(Full List)과 PL(Part List)의 구조

색인의 전체 영역은 동일한 크기의 여러 그룹들로 구성되어져 있고, 각 그룹에서의 질의 리스트와 포함하는 셀의 정



(그림 6) FL과 PL의 구조

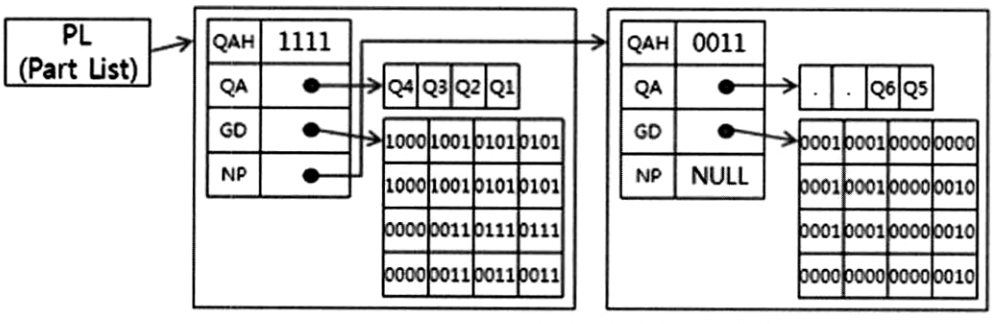


(그림 7) Part List 에서 비트열의 길이보다 적은 수의 질의 관리

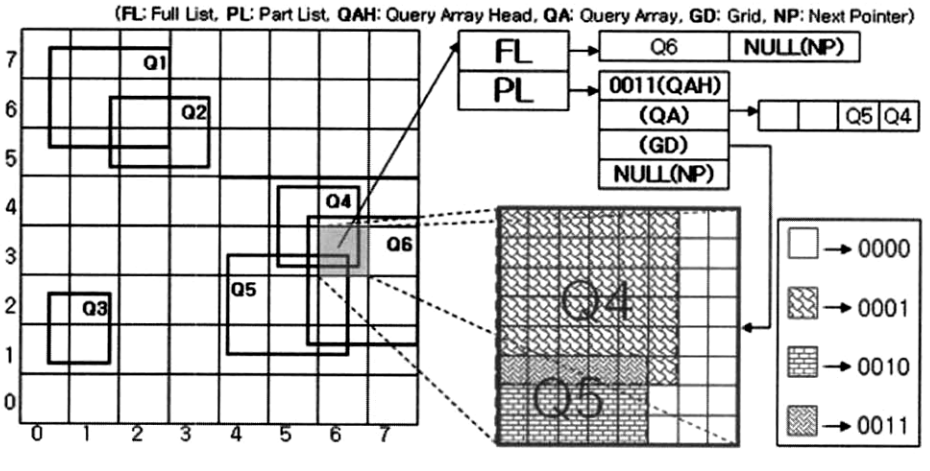
보를 FL(Full List)와 PL(Part List)로 관리한다. FL은 그룹영역과 겹치는 질의 중에서 완전히 그룹을 포함하는 질의에 대한 리스트이고, PL은 그룹영역과 겹치는 질의 중에서 부분적으로 그룹과 겹치는 질의에 대한 리스트이다. 그룹은 포함하는 셀들에 대한 정보를 함께 유지한다.

(그림 6)은 각 그룹에서 관리하는 FL과 PL의 구조에 대해서 보여주고 있다. PL의 경우 QAH(Query Array Head), QA(Query Array), GD(Grid) 그리고 NP(Next Node Pointer)로 이루어진다. QA는 그룹을 부분적으로 포함하는 질의의 식별자를 저장하고, 일단 부분적으로 그룹을 포함하는 질의가 삽입이 되면 그룹 내에서 포함하는 영역을 구분하기위해 사용되어질 그 질의에 대한 비트 식별자가 지정된다. QAH는 그 배열상의 어느 위치에 질의의 식별자가 있는지를 비트 열로 나타냄으로서 질의의 삽입, 삭제 시 QA의 관리를 보다 쉽게 하기 위한 것으로, 검색 시에도 유용하게 이용을 한다. GD는 크기가 1인 셀들로 구성되어져 있으며, 그 셀에서는 그룹에 부분적으로 겹치는 질의들과의 포함정보를 비트 열로 유지한다. NP는 QA가 질의 식별자들에 의해서 꽉 찼을 경우 다음 노드로 확장을 시키거나 또는 검색 시에 사용되는 다음 노드를 가리키는 포인터로서, 일반적인 자료구조의 리스트에서 쓰이는 다음 노드를 가리키기 위한 포인터와 동일한 역할을 한다.

(그림 7)은 Part List의 한 노드에서 관리하는 비트열의 길이보다 적은 수의 질의를 삽입했을 때의 경우를 그림으로 표현한 것이다. 비트열 길이가 4이고 삽입된 질의가 3개이므로 길이가 4인 비트 열에서 3개의 질의를 모두 표현할 수 있다. (그림 8)은 Part List 에서 비트열의 길이보다 많은 수의 질의를 삽입했을 때를 보여준다. 비트열 길이가 4이고 삽입된 질의가 6개이므로 길이가 4인 비트열을 이용한 표현이 불가능하기 때문에 비트 열의 길이가 동일한 한 개의 노드를 PL에 추가하여 관리한다. 그리고 노드는 NP를 이용하여 연결한다. 이러한 방식으로 질의 개수의 증가에 따른 색



(그림 8) Part List 에서 비트열의 길이보다 많은 수의 질의 관리



(그림 9) 전체 색인 구조

인의 확장성을 보장한다.

FL의 경우 QID(Query ID) 그리고 NP(Next Node Pointer)로 이루어진다. 이동객체가 그룹 안에서 이동을 했다면 FL의 질의들에 대해서는 계산에서 제외한다. 이는 그룹을 완전히 포함하는 질의들이기 때문에 그룹 안에서의 이동은 그 질의들의 결과에 아무런 영향을 미치지 않기 때문이다.

3.3 질의 색인 구조

본 논문에서 제안하는 기법은 기존의 CES 기법과 셀을 그룹으로 관리한다는 면에서 비슷한 접근 방식을 가지고 있다. 그러나 색인 구조를 비교해 보면 비트 식별자를 이용한 질의 관리와 셀에서 유지하는 비트 열을 이용한 질의와의 포함관계를 표현하기 위한 구조로서 CES와는 매우 상이한 구조를 지니는 것을 알 수 있다. (그림 9)는 연속 영역 질의 처리를 위해 본 논문에서 제안하는 기법의 전체적인 색인 구조를 표현하고 있다. 전체 그리드는 크기가 8×8인 64개의 그룹으로 구성되어져 있고, 6개의 질의(Q1~Q6)가 등록된 상태이다. 예들 들어, (6, 3) 위치에 있는 그룹으로 그룹과 겹치는 질의는 총 3개이다. 그중에서 Q6은 그룹을 완전히 포함하므로 FL에 등록되어 있다. 또한, Q4와 Q5는 그룹을 부분적으로 겹치므로 PL에 삽입된다. PL에 삽입 되어진 질의는 QA에 입력되고, 각 질의는 QA 내의 위치에 따른 비트 식별자를 갖는다. Q4의 비트 식별자는 0001이고, Q5의 질의 식별자는 0010이다. PL의 GD는 포함하는 셀을 관리하는데, Q4와 Q5에 포함되는 셀의 비트 열에 각각 질의의 비트 식별자 값을 더한다. 그래서 아무런 질의에도 포함되지 않는 셀의 비트 열은 0000, Q4에 또는 Q5에 포함되는 셀의 비트 열은 각각 0001과 0010이다. 그리고 Q4와 Q5에 모두 포함되는 셀의 비트 열은 0011 이다.

3.4 알고리즘

본 절에서는 제안하는 질의 색인 기법의 질의의 삽입, 삭제 알고리즘과 또한 그 색인 기법을 이용한 연속 영역 질의 처리에 대한 알고리즘에 대해서 설명한다. <표 1>은 알고리즘을 표현하기 위한 기호들에 대해 설명한다.

3.4.1 질의 삽입

본 논문에서 제안하는 색인 구조에서 질의의 삽입은 삽입되는 질의의 영역과 겹치는 모든 그룹에 대해서 한 번씩 수행 하고 질의가 그룹을 완전히 포함하는 경우와 부분적으로 겹치는 경우를 구분하여 완전히 겹치는 경우에는 FL(Full List) 그리고 부분적으로 겹치는 경우에는 PL(Part List)에 삽입한다. PL에 삽입이 되면 그룹 내의 그 질의와 겹치는 모든 셀의 비트 열에 그룹 셀 내에서 할당된 질의의 비트 식별자를 OR 연산해서 그 셀들이 삽입된 질의에도 포함된다는 것을 나타낸다.

〈표 1〉 알고리즘을 위한 기호

기호	설명
g	그룹 셀
g.FL	그룹 셀 g의 FL(FULL List)
g.PL	그룹 셀 g의 PL(Part List)
q	질의
c	비트 열
Pnew(O)	객체 O의 현재 위치
Pold(O)	객체 O의 이전 위치
G(p)	위치 p를 포함하는 그룹 셀
C(p, pl)	부분 리스트 pl에서 위치 p를 포함하는 셀의 비트 열
OL(q)	질의 q의 결과 집합을 가지는 객체 리스트
QL(p)	위치 p를 포함하는 질의 리스트
Bit(c, i)	비트 열 c의 i번째 비트 값
BIT_ID_SIZE	PL내에서 비트 식별자의 길이

```
    Algorithm InsertQuery(lx, ly, hx, hy)
01  q = (lx, ly, hx, hy);
02  for each g in ∀g ∈ (G(q)) {
03    if( g is fully contained by q )
04      InsertQueryIntoFL(g, q);
05    else
06      InsertQueryIntoPL(g, q);
07  }
```

(그림 10) InsertQuery()

```
    Algorithm InsertQueryIntoFL(Group g, Query q);
01  create new FL node;
02  add the new FL node to g.FL;
03  fl_node = the new FL node;
04  fl_node.QID = q;
```

(그림 11) InsertQueryIntoFL()

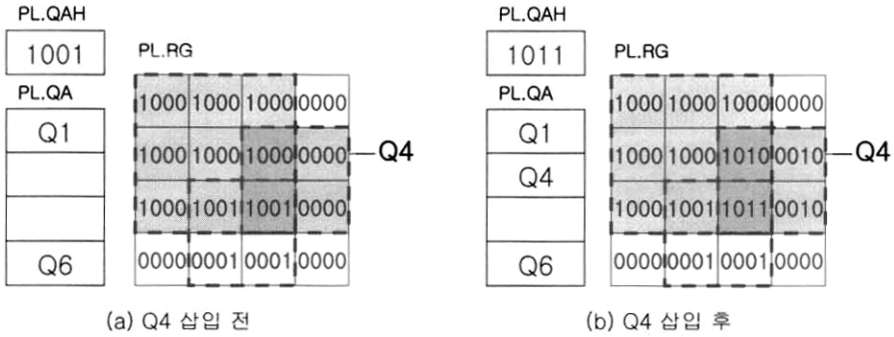
(그림 10)은 질의 삽입을 수행하는 알고리즘이다. 질의는 직사각형의 형태로 내려지고, 그 영역은 좌측 하단 좌표(l_x, l_y)와 우측 상단의 좌표(h_x, h_y)로 표현한다. 질의가 삽입되면 질의 영역과 겹치는 그룹이 질의에 의해서 완전히 포함되면 그룹의 FL에 질의를 삽입하기 위해 InsertQueryIntoFL()을 수행하고, 그렇지 않으면 PL에 질의를 삽입하기 위해 InsertQueryIntoPL()을 수행한다. FL은 자료구조의 일반적인 리스트 구조로서 질의 식별자를 저장하는 것이 목적이며, (그림 11)에서 나타내는 것과 같이 InsertQueryIntoFL()은 FL에 질의 식별자를 삽입한다.

```
    Algorithm InsertQueryIntoPL(Group g, Query q);
01  if (g.PL == NULL){
02    create new PL node;
03    add the new PL node to g.PL;
04  }
05  pl_node = g.PL;
06  while ((pl_node is full by queries) and (pl_node.NP ≠ NULL))
07    pl_node = pl_node.NP;
08  if (pl_node ≠NULL){
09    i = 0;
10    while (i-th bit value of pl_node.QAH is not 0) i++;
11    pl_node.QA[i] = q;
12    pl_node.QAH = pl_node.QAH || 2^i;
13    for each c in ∀c ∈ C(q, pl_node) do c = c || 2^i;
14  }else{
15    create new PL node;
16    add the new PL node to g.PL;
17    pl_node = the new PL node;
18    pl_node.QA[0] = q;
19    pl_node.QAH = pl_node.QAH || 2^0;
20    for each c in ∀c ∈ C(q, pl_node) do c = c || 2^0;
21  }
```

(그림 12) InsertQueryIntoPL()

(그림 12)는 질의를 PL에 삽입하는 알고리즘이다. 질의를 삽입할 그룹 g의 PL이 NULL이면 새로운 PL을 생성한다. (줄01~04). 그룹 g의 PL에 연결된 모든 PL 노드의 QAH를 검색하여 질의를 삽입할 공간이 있는 노드를 찾는다. (줄06~07). 질의를 삽입할 PL 노드를 찾으면 질의를 삽입할 위치 정하기 위해 그 노드의 QAH에 비트 값이 0인 위치인 i를 알아낸다(줄10). 그리고 그 노드의 QA에서 i 번째 자리에 질의 식별자를 삽입하고, 질의의 비트 식별자는 i 번째 비트 값만 1인 비트 열로 결정된다(줄11~12). 질의의 비트 식별자를 그 노드의 QAH과 OR연산을 수행하여 값을 변경하고, 그 노드의 GD 내에서 질의와 겹치는 모든 셀의 비트 열과 OR연산을 수행하여 삽입된 질의와의 겹침 정보를 추가한다(줄13).

(그림 13)은 한 그룹의 PL에 질의가 삽입되는 경우의 예를 나타낸다. 이 예는 한 그룹과 부분적으로 겹치는 Q4를 그 그룹의 PL에 삽입하는 것이고 (a)는 삽입 전의 상태, (b)는 삽입 후의 상태를 나타낸다. Q4가 입력되면 그룹의 QA에 어느 공간이 비었는지 알기 위해서 QAH를 보고 배열 QA의 두 번째 자리가 비었음을 알아내고 Q4를 배열 QA에 삽입한다. Q4가 두 번째 자리에 입력이 되었으므로 비트 식



(a) Q4 삽입 전 (b) Q4 삽입 후

(그림 13) PL에 질의 Q4 삽입

```
    Algorithm DeleteQuery(Query q)
01    for each g in ∀g ∈ (G(q))
02      if ( g is fully contained by q )
03        DeleteQueryFromFL(g, q);
04      else
05        DeleteQueryFromPL(g, q);
```

(그림 14) DeleteQuery()

```
    Algorithm DeleteQueryFromFL(Group g, Query q);
01  fl_node = g.FL;
02  while (fl_node ≠ NULL){
03    if (fl_node.qid == q){
04      remove fl_node from g;
05      break;
06    }
07    fl_node = fl_node.NP;
08  }
```

(그림 15) DeleteQueryFromFL()

```
    Algorithm DeleteQueryFromPL(Group g, Query q)
01  pl_node = g.PL;
02  while (pl_node ≠ NULL){
03    for (i=0 ; i<BIT_ID_SIZE ; i++){
04      if (i-th bit value is 0) continue;
05      if (pl_node.QA[i] == q){
06        pl_node.QAH = pl_node.QAH ^ 2^i;
07        if (pl_node.QAH == 0){
08          remove pl_node from g.PL;
09          return;
10        }
11        for each c in ∀c ∈ C(q, pl_node) do c = c ^ 2^i;
12        return;
13      }
14    }
15    pl_node = pl_node.NP;
16  }
```

(그림 16) DeleteQueryFromPL()

별자는 0010이 된다. 이 비트 식별자를 QAH의 비트 열인 1001에 OR연산을 수행하면 QAH는 1011이 된다. 그리고 질의의 비트 식별자를 GD상의 Q4와 겹치는 모든 셀의 비트 열에 OR연산을 수행한다.

3.4.2 질의 삭제

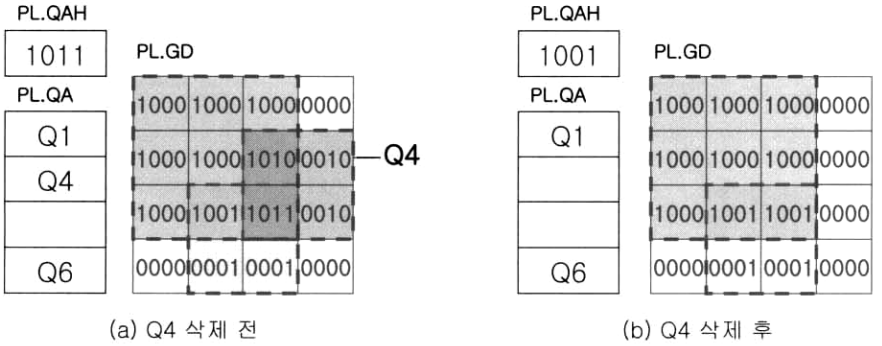
삭제는 기본적으로 질의 삽입 과정과 거의 동일하게 수행되어 진다. 삽입 과정에서는 QA에 질의 식별자를 기록하고 QAH에 비트 식별자 값을 OR연산을 수행했지만 삭제 과정에서는 XOR연산을 수행한다. 겹치는 모든 셀의 비트 열에도 XOR연산을 수행한다.

(그림 14)는 질의 삭제를 수행하는 알고리즘이다. 질의 q의 영역과 겹치는 모든 그룹에 대해서 질의에 의해서 완전히 포함되면 그룹의 FL에서 질의를 삭제하기 위해 DeleteQueryFromFL()을 수행하고, 부분적으로 포함되면 그룹의 PL에서 질의를 삭제하기 위해 DeleteQueryFromPL()을 수행한다. (그림 15)에서 나타내는 것과 같이 DeleteQueryFromFL()은 FL에서 질의 식별자를 삭제한다.

(그림 16)은 한 그룹의 PL에서 질의를 삭제하는 알고리즘이다. 그룹 g의 PL에 연결된 모든 노드의 QA에서 삭제할 질의 q를 검색하여 질의 q의 비트 식별자를 알아낸다(줄05). 그리고 질의 q를 포함하는 노드의 QAH에 질의 q의 비트 식별자와 XOR 연산을 수행하여 QAH의 값을 갱신한다(줄06). 만약 QAH의 값이 0이면, 그 노드에는 아무런 질의도 포함되지 않으므로 그룹의 PL에서 삭제하고(줄07~08), QAH가 0이 아니면 그 노드의 GD내에서 질의 q와 겹치는 모든 셀의 비트 열에 질의 q의 비트 식별자와 XOR 연산을 수행하여 각 셀의 비트 열을 갱신한다(줄10).

질의가 삭제되는 경우 질의 밀도가 낮은 PL 노드가 발생할 수 있다. 이러한 경우 노드의 병합을 수행 할 경우 매번 질의의 삭제 시 질의 밀도를 비교하는 연산과 노드를 병합하는 연산이 추가적으로 발생하게 되므로 질의 처리 시간이 지연될 수 있다. 따라서 본 논문에서는 질의의 삭제에 의해 빈 PL 노드가 발생할 때까지 PL노드를 유지하고 PL 노드의 모든 질의가 삭제되면 해당 노드의 삭제를 수행한다.

(그림 17)의 (a)와 같이 QA상에서 Q4가 두 번째에 위치하므로 Q4의 비트 식별자는 0010이 된다. 이 비트 식별자를 QAH의 값인 1011에서 XOR 연산을 수행하면 1001이 된다. 그리고 GD에서 Q4와 겹치는 모든 셀의 비트 열에서 Q4의 비트 식별자인 0010을 XOR 연산을 수행한다. 결과는 (그림 17)의 (b)에서 나타내고 있다.



(a) Q4 삭제 전 (b) Q4 삭제 후

(그림 17) PL에서의 질의 삭제

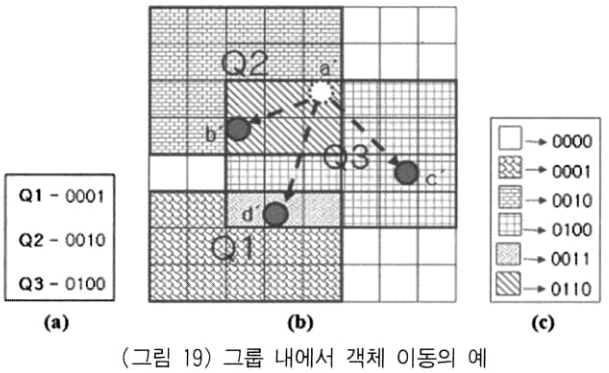
3.4.3 연속 영역 질의 처리

연속 영역 질의 처리는 두 가지로 분류되어 진행이 된다. 하나는 이동 객체가 그룹 내에서 이동을 한 경우와 나머지 한 가지는 그룹 밖으로 이동을 한 경우이다. 이동 객체의 그룹 내에서의 이동은 셀 간의 비트 열을 이용한 연산으로 결과 집합의 갱신이 필요한 질의를 파악한다. 하지만 객체가 그룹 밖으로 이동을 한 경우에는 새로운 위치에서 포함되는 모든 질의에 대해서 삽입 과정을 수행하고, 이전에 포함되는 모든 질의에 대해서 삭제 과정을 수행하여 질의들의 결과 집합을 갱신한다.

(그림 18)은 이동객체가 새로운 위치 값을 보내왔을 때 연속 영역 질의가 처리하는 알고리즘이다. 만약 객체가 그룹 안에서 이동 했으면 다시 말해서 $G(p_{new}) = G(p_{old})$이면, 이전 위치의 비트 열과 새로운 위치의 비트 열을 비교 한다(줄08). 만약 그 값이 다르면 객체의 이동에 영향을 받는 질의가 존재하는 것이므로, 각 자리의 비트 값을 비교하여 영향을 받는 질의의 비트 식별자를 찾아내고, 비트 식별자에 해당하는 질의의 결과 집합 갱신을 수행한다(줄08~13). 만약 이전 위치의 비트 값이 0이고, 새로운 위치의 비트 값이 1이면 해당 질의 식별자에 해당하는 질의에 객체가 새롭게 들어갔음을 의미하여 결과 집합에 객체의 삽입을 수행하고 (줄11), 만약 이전 위치의 비트 값이 1이고, 새로운 위치의 비트 값이 0이면 해당 질의 식별자에 해당하는 질의에서 객체가 벗어났음을 의미 하여 결과 집합에서 객체의 삭제를 수행한다(줄13). 그리고 이러한 과정을 PL의 마지막 노드까지 수행한다.

```
    Algorithm CQProcessing(Object O)
01    p_new = P_new(O);
02    p_old = P_old(O);
03    if (G(p_new) = G(p_old) ) {  // an object has moved within a group
04      g = G(p_old);
05      pl_node = g.PL;
06      while (pl_node ≠ NULL ){
07        c_old = C(p_old, pl_node); c_new = C(p_new, pl_node);
08        if (c_old ≠ c_new)
09          for ( i=0 ; i<BIT_ID_SIZE ; i++)
10            if ( (Bit(c_old, i) = 0) && (Bit(c_new, i) = 1) )
11              insert O into OL(q), q = pl_node.QA[i];
12            else if ( (Bit(c_old, i) = 1) && (Bit(c_new, i) = 0) )
13              remove O from OL(q), q = pl_node.QA[i];
14        pl_node = pl_node.NP;
15      }
16    } else {      // an object has moved out of a group
17      insert O into OL(q), ∀q ∈ QL(p_new);
18      remove O from OL(q), ∀q ∈ QL(p_old);
19    }
```

(그림 18) CQProcessing()



(그림 19) 그룹 내에서 객체 이동의 예

(그림 19)는 한 그룹 내에서 연속 영역 질의 처리의 예를 나타낸다. (a)는 그룹과 부분적으로 겹치는 질의들을 나타내고, (b)는 객체의 다양한 이동을 표현하고, (c)는 그룹 내의 질의와의 포함관계를 나타내는 셀들에 대한 비트 열을 표현한다. a′는 이동 객체의 이전 위치를 나타내고, b′, c′, d′는 이동 객체의 다양한 이동의 예를 표현하기 위한 새로운 위치들이다. 만약 객체가 a′에서 b′로 이동했다면, a′위치에 해당하는 비트 열과 b′ 위치에 해당하는 비트 열이 동일하므로 더 이상 아무런 연산을 수행하지 않는다. 만약 객체가 a′에서 c′또는 d′로 이동했다면 이전 위치의 비트 열과 새로운 위치의 비트 열이 다르기 때문에 영향을 받는 질의가 존재한다는 것으로서, 각 자리의 비트 값을 비교하여 갱신이 필요한 질의들의 결과 집합을 갱신한다.

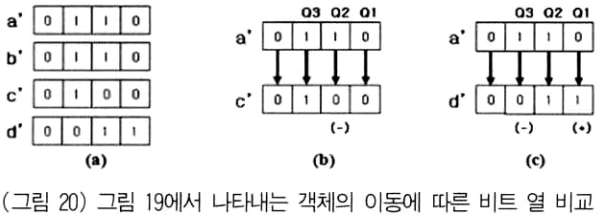
(그림 20)은 객체의 이동으로 영향을 받는 질의를 찾기 위한 비트 열을 비교하는 것을 나타낸 것이다. (a)는 (그림 17)에서 나타낸 객체의 각 위치에 따른 비트 열을 나타내고, (b)는 객체가 a′에서 c′로 이동했을 경우, 비트 식별자가 0010인 질의 Q2로부터 객체가 벗어났음을 나타내고, (c)는 객체가 a′에서 d′로 이동했을 경우, 비트 식별자가 0001인 질의 Q1에 새롭게 포함되고, 비트 식별자가 0100인 질의 Q3로부터 벗어났음을 나타낸다.

4. 성능 평가

4.1 평가 환경

메인 메모리 기반의 그리드 구조 색인 기법이 R-트리를 이용하는 것보다 탐색하는 시간과 트리를 유지하는 저장 공간 및 처리 비용을 감소시킨다[7]. 그러므로 본 논문에서는 R-트리 기반의 알고리즘과의 비교는 수행하지 않는다. [7]에서는 VCR기법이 CQI 기법 보다 질의 처리 속도 및 저장 공간 측면에서 모두 우수함을 보여주었고, [8]에서는 CES 기법이 VCR 기법과 비교하여 질의 처리 속도가 향상되었음을 보여 주었다. 따라서 본 논문에서는 CES 기법과 성능비교를 수행하였고 성능 평가 인자는 CES 기법에서의 것과 유사하게 설정하였다.

제안하는 연속 영역 질의 처리 기법의 우수성을 입증하기 위해 펜티엄4 CPU 3G, RAM 1G, 윈도우즈 XP 운영체제에



(그림 20) 그림 19에서 나타내는 객체의 이동에 따른 비트 열 비교

〈표 2〉 성능 평가 인자

인자	내용
질의의 크기	1 ~ 80
객체의 이동 속도	0 ~ 20
그룹의 크기	4× 4 ~ 64× 64
질의의 개수	1000 ~ 16000
객체의 개수	4000 ~ 64000

서 C언어로 구현하여 성능 평가를 수행한다. 성능 평가를 위해 전체 그리드 크기를 512×512로 하고 8,000개의 질의와 64,000개의 객체를 생성한다. 성능 평가에 많은 영향을 미치는 질의의 크기와 이동 객체의 속도는 다양한 값으로 변화하면서 성능 평가를 수행한다. <표 2>는 성능 평가를 위해 사용된 인자들을 나타낸 것이다. <표 2>에 나타난 인자의 크기 1은 최소 크기 셀의 측면 길이이다. 본 논문에서 제안하는 색인 기법은 편의상 BID로 나타내었다.

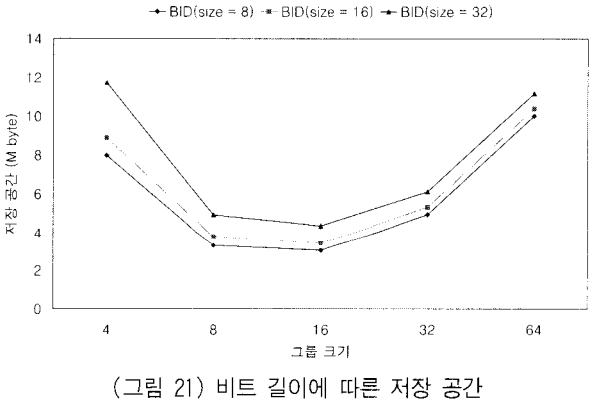
4.2 성능 비교

BID 기법에서 그룹 크기는 성능을 결정하는 아주 중요한 요소이다. 그룹의 크기가 증가할수록 부분적으로 겹치는 질의의 수가 증가하므로 그룹 내의 셀들이 유지하는 비트 열의 크기가 질의의 수만큼 증가된다. 비트 열의 크기가 너무 크면 객체의 이동에 따른 갱신 처리 시 비트 열 비교 연산의 증가와 저장 공간의 낭비를 초래할 수 있다. 또한 그룹의 크기가 감소할수록 부분적으로 겹치는 질의의 수는 감소되어 비트 열의 크기는 작아지지만, 그룹들의 중복적인 질의 식별자 저장 횟수가 증가되고 그룹을 벗어나는 이동 객체의 갱신이 많아지므로 객체의 이동에 따른 갱신 처리 속도도 저하된다.

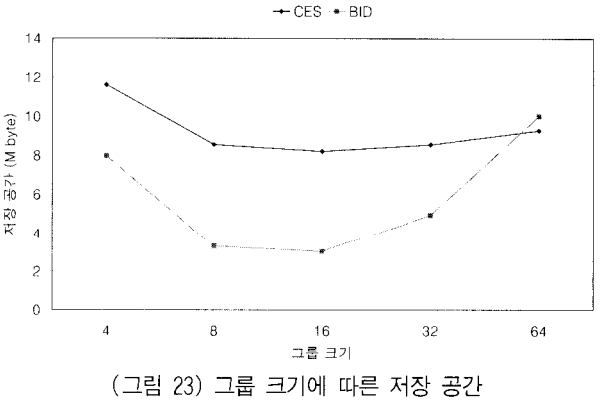
(그림 21)과 (그림 22)는 제안하는 질의 색인에 대해 비트 식별자의 길이와 그룹의 크기 설정이 저장 공간과 처리 속도에 미치는 영향을 나타낸 것이다. 비트 식별자의 크기가 8, 16, 32인 경우 그룹의 크기를 4 ~ 64로 변화하면서 성능을 비교한 결과이다. 비트 식별자의 크기가 저장 공간과 처리속도에 큰 영향을 주지는 않지만 그룹의 크기는 많은 영향을 주고 있다. 저장 공간이나 처리 속도 면에서 그룹 크기가 16일 때 가장 좋은 성능을 보이고 있다.

(그림 23)과 (그림 24)는 기존에 제안된 CES 기법과 비트 식별자의 크기가 8일 때 제안하는 BID 기법과의 그룹 크기별 저장 공간과 처리속도 비교를 나타낸 것이다. BID 기법은 CES 기법에 비해 적은 저장 공간을 이용하여 색인을 유지하고 그룹 크기가 16인 경우 저장 공간을 가장 적게 이용하는 것이 나타났다. 하지만 그룹 크기가 64인 경우와 같이 그룹 크기가 너무 커져 버리면 오히려 더 많은 저장 공간을 소모하게 된다. 처리 속도 면에는 그룹 크기와 상관없이 BID 기법이 전체적으로 빠른 처리를 수행하였고 CES 기법의 경우 그룹 크기가 처리속도에 많은 영향을 미치는 반면 BID 기법은 그룹 크기에 큰 영향을 미치지 않는다는 것을 알 수 있다.

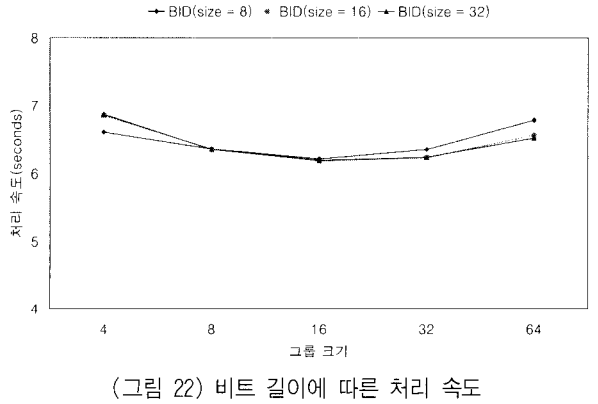
(그림 25)와 (그림 26)은 기존 CES 기법과 제안하는 BID



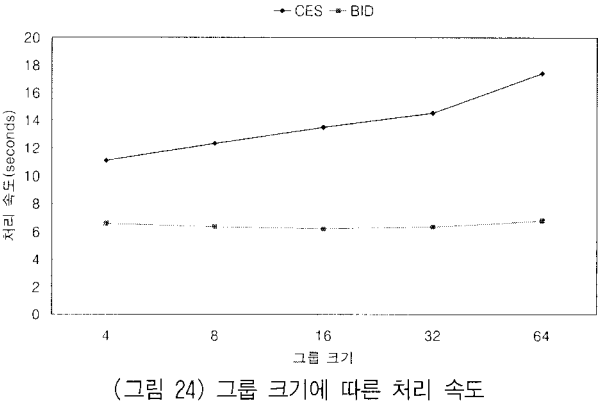
(그림 21) 비트 길이에 따른 저장 공간



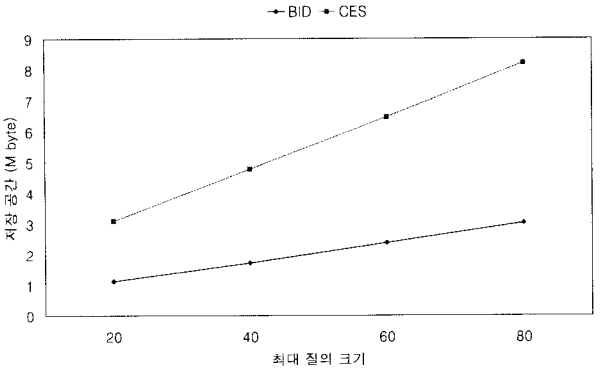
(그림 23) 그룹 크기에 따른 저장 공간



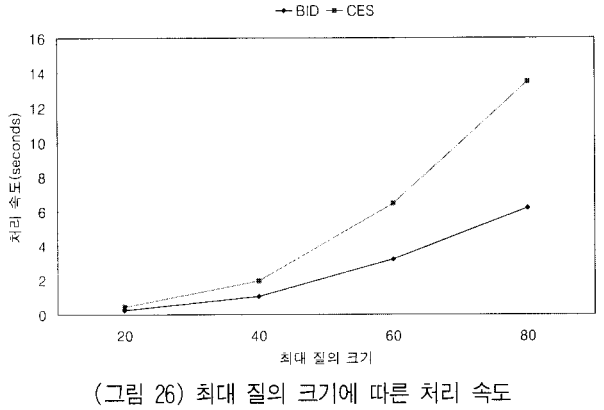
(그림 22) 비트 길이에 따른 처리 속도



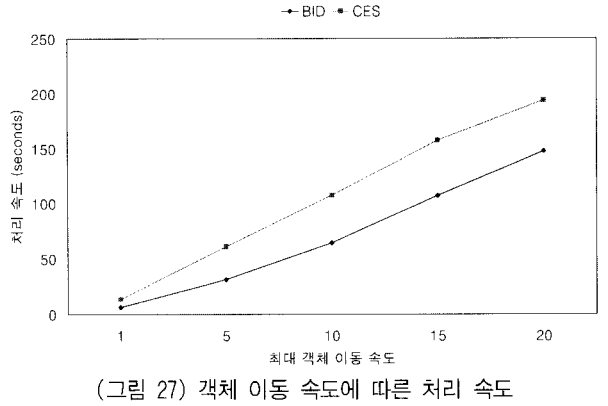
(그림 24) 그룹 크기에 따른 처리 속도



(그림 25) 최대 질의 크기에 따른 저장 공간



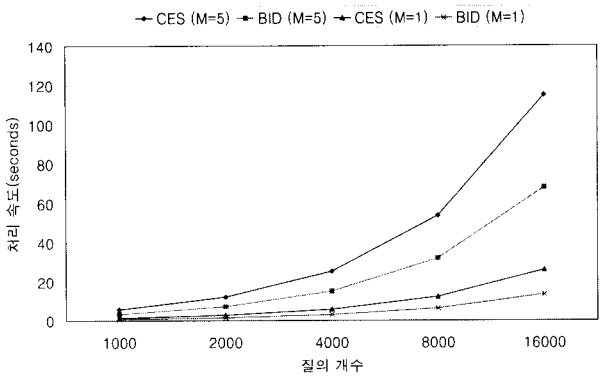
(그림 26) 최대 질의 크기에 따른 처리 속도



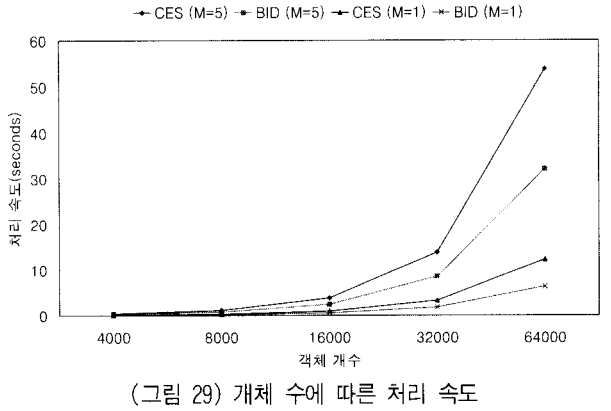
(그림 27) 객체 이동 속도에 따른 처리 속도

기법에 대해 비트 식별자를 8비트로 할 때 최대 질의 크기별 저장 공간과 처리 속도 비교를 나타낸다. 질의 크기가 증가될수록 저장되는 정보가 증가되고 질의 처리를 수행하는데 비교되는 정보의 개수가 증가되기 때문에 처리 속도는 저하된다. 성능 평가 결과 두 기법 모두 질의 크기에 영향을 받지만 BID기법이 CES 기법에 비하여 모든 경우에 우수한 성능을 나타낸다.

(그림 27)은 기존 CES 기법과 제안하는 BID 기법에 대해 비트 식별자를 8비트로 할 때 객체의 속도가 연속 범위 질의 처리에 미치는 영향에 대해서 나타낸 것이다. 제안하는 BID 기법은 보다 적은 연산 비용으로 질의 처리를 수행할 수 있도록 객체가 그룹 안에서 이동을 했을 때 비트 열을 비교를 통해 결과에 영향을 미치는 질의를 찾는다. 따라서 객체가 빠르게 이동 할수록 그룹을 벗어나는 경우가 많이 발생하기 때문에 추가적인 연산 비용이 소모된다. 성능 평가 결과 모든 경우에 BID 기법이 CES 기법보다 빠른 처리를 수행하였고 객체의 이동 속도에는 BID 기법과 CES 기법 모두 큰 영향을 받고 있는 것으로 나타났다.



(그림 28) 질의 수에 따른 처리 속도



(그림 29) 개체 수에 따른 처리 속도

(그림 28)과 (그림 29)는 질의의 개수와 객체의 개수에 따른 처리 속도를 비교한다. 이 평가는 각 기법의 확장성에 대한 평가를 위해 수행된 것으로서 질의 개수는 1000~16000개 그리고 객체 개수는 4000~64000개를 그리고 객체의 최대 속도 M은 1과 5인 경우를 기준으로 하였다. 모든 경우에 BID 기법이 CES 기법 보다 대략 40%정도의 성능 개선을 보였다.

본 논문에서는 다양한 관점에서 제안하는 기법의 성능을 가장 성능이 우수한 CES 기법과 비교하여 하였다, 성능 평가 결과 제안하는 BID 기법이 CES 기법 보다 모든 환경에서 저장 공간 측면과 처리 속도 측면 모두 향상시킨 것을 확인할 수 있었다. 처리 속도의 경우 모든 면에서 CES 기법 보다 향상되었음이 확인되었고 그룹 크기에도 큰 영향을 받지 않는 것으로 나타났다. 저장 공간의 경우 그룹의 크기에 영향을 많이 받고 최악의 경우에는 CES 기법 보다 더 많은 저장 공간을 소모한다. 따라서 적용 애플리케이션의 질의 크기와 객체의 속도에 따른 적절한 그룹의 크기 지정은 필수적이다. 결과적으로 소모되는 저장 공간만을 고려하여 그룹의 크기를 지정하면 모든 부분에서 기존의 기법들 보다 좋은 성능이 보장한다.

5. 결론 및 향후 연구

본 논문에서는 이동 객체를 대상으로 하는 연속 영역 질의 처리를 위해 기존에 제안된 여러 기법들의 단점을 자세히 분석하고, 기존 기법에 비해 보다 적은 공간을 사용하면서 빠른 연속 질의를 처리하기 위한 질의 처리 기법을 제안하였다. 연속 질의 처리는 객체의 갱신에 의해 질의 결과의 변경을 처리하기 위해 영향을 받는 질의를 빠르게 알아내는 것이 중요하다. 제안하는 연속 질의 처리 기법은 질의 색인 기법을 위해 메인 메모리 기반의 그리드 구조를 사용하고 그리드로 분할된 셀들은 그룹 단위로 관리한다. 본 논문에서는 질의를 나타내기 위해 비트 식별자를 이용하여 저장 공간의 효율과 빠른 처리 속도 제공하고 각 셀에서 유지하는 질의와의 포함관계를 나타내는 비트 열을 이용하여 객체의 위치 갱신이 영향을 주는 질의를 빠르게 파악할 수 있다. 향후 연구로 연속 영역 질의와 함께 연속 질의로 많이 사용되는 연속 k-최근접 질의(continuous k-nn query)에 대한 연구를 계속적으로 수행할 예정이다.

참 고 문 헌

[1] P. K. Agarwal, L. Arge, and J. Erickson. "Indexing moving objects," Proc. ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp.175-186, 2000.

[2] G. Kollios, D. Gunopulos, and V. J. Tsotras. "On indexing mobile objects," Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp.261 272, 1999.

[3] D. Pfoser, C. S. Jensen, and Y. Theodoridis. "Novel approaches to the indexing of moving object trajectories," Proc. International Conference on Very Large Data Bases. pp.395-406, 2000.

[4] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. "Indexing the positions of continuously moving objects," Proc. ACM SIGMOD International Conference on Management of Data, pp.331-342, 2000.

[5] Y. Cai and K. A. Hua. "An Adaptive Query Management Technique for Real-time Monitoring of Spatial Regions in Mobile Database Systems," Proc. International Conference on Performance, Computing and Communication, pp.259-266, 2002.

[6] D. V. Kalashnikov, S. Prabhakar, and S. E. Hambrusch, "Main Memory Evaluation of Monitoring Queries Over Moving Objects," Distributed and Parallel Databases, Vol.15, No.2, pp.117-135, 2004.

[7] K. L. Wu, S. K. Chen, and P. S. Yu, "Processing continual range queries over moving objects using VCR-based query indexes," Proc. IEEE International Conference on Mobile and Ubiquitous Systems : Networking and Services, pp.226-235, 2004.

[8] K. L. Wu, S. K. Chen, and P. S. Yu, "On Incremental Processing of Continual Range Queries for Location-Aware Services and Applications", Proc. Annual International Conference on Mobile and Ubiquitous Systems, pp.261 269, 2005.

[9] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch, "Query Indexing and Velocity Constrained Indexing : Scalable Techniques for Continuous Queries on Moving Objects," IEEE Transactions on Computers, Vol.51, No.10, pp.1124-1140, 2002.

[10] X. Wang, Q. Zhang, W. Sun, W. Wang, and B. Shi, "cGridex: Efficient Processing of Continuous Range Queries over Moving Objects," Proc. International Conference on Advances in Web-Age Information Management, pp.345-356, 2005.

[11] M. F. Mokbel, X. Xiong, and W. G. Aref, "SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases," Proc. ACM SIGMOD International Conference on Management of Data, pp.623-634, 2004.

[12] B. Gedik, K. L. Wu, P. S. Yu, and L. Liu. "Motion adaptive indexing for moving continual queries over moving objects," Proc. ACM International conference on Information and Knowledge Management, pp.427-436, 2004.

박 용 훈

e-mail : yhpark@netdb.chungbuk.ac.kr
2005년 호원대학교 정보통신공학과 및 건축공학과 (공학사)
2007년 충북대학교 정보통신공학과 (공학석사)
2007년 3월~현 재 충북대학교 정보통신공학과 박사과정
관심분야 : 데이터베이스 시스템, 정보검색, 위치기반 서비스, 시공간 데이터베이스 등

복 경 수

e-mail : ksbok@dbserver.kaist.ac.kr
1998년 충북대학교 수학과 (이학사)
2000년 충북대학교 정보통신공학과 (공학석사)
2005년 충북대학교 정보통신공학과 (공학박사)
2005년 3월~현 재 한국과학기술원 전산학과 연수연구원
관심분야 : 자료 저장 시스템, 이동 객체 데이터베이스, 시공간 색인 구조, 센서 네트워크 및 RFID 등

유 재 수

e-mail : yjs@cbucc.chungbuk.ac.kr
1989년 전북대학교 컴퓨터공학과 (공학사)
1991년 한국과학기술원 전산학과 (공학석사)
1995년 한국과학기술원 전산학과 (공학박사)
1995년~1996년 8월 목포대학교 전산통계학과 전임강사
1996년 8월~현 재 충북대학교 정보통신공학과 교수
관심분야 : 데이터베이스 시스템, 정보검색, 멀티미디어 데이터베이스, 분산 객체 컴퓨팅 등