제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

제한된 영역에서의 이동 및 고정 객체에 대한 색인 기법

윤종선*, 박현주*
*한밭대학교 정보통신 공학과
e-mail:yjs2398@hanmail.net

An indexing method for moving or static objects in limited region

Jong-sun Yoon*, Hyun-ju Park*
*Department of Information Communication Engineering, Hanbat National University

요 약

이동 객체를 효율적으로 처리하기 위해 여러 가지 색인 기법들이 제안되었다. 이들 중 3D R-tree와 같은 색인 기법은 시간과 공간을 동일한 차원으로 생각하고 있으나, 실제로 이 두 차원은 단위와 성격이 다르므로 분리해서 처리해야 한다. 특히 본 논문에서 고려하는 환경은 실내와 같은 한정된 공간이므로, 이런 환경에서는 시간과 공간이 같이 성장하는 것이 아니라 공간은 한정되어 있는 반면 시간 차원만이 성장한다. 따라서 R-tree와 1차원(시간차원)의 TB-tree 두 개의 색인을 유지하여, 공간정보와 고정된 객체는 R-tree에, 시간 정보와 이동 객체는 TB-tree에 저장하는 시공간 분리 트리(STS-tree : Separation of Time and Space tree)를 제안한다.

1. 서론

최근 무선 통신의 발달과 휴대용 단말기의 사용이 일반화되면서 다양한 위치기반 서비스(Location Based Service : LBS)가 이루어지고 있다. GPS를 이용한 광범위한 실외에서의 위치기반 서비스는 이미 보편화되었고 ZigBee 등 보다 오차범위가 적은 통신 기법이 개발되면서 이를 이용해 실내와 같이 한정된 영역으로 그 이용범위가 점차 확대되고 있다. 이런 환경은 기존의 기법이 다루고 있는 실외환경과 여러 가지 차이점이 있으므로 제한된 영역에서 보다 효율적으로 정보를 저장하고 검색할 수 있는 색인 기법이 필요하다.

색인 기법은 크게 과거의 정보, 현재의 위치 그리고 현재와 미래의 위치에 대해 색인하는 방법으로 나누어 볼 수 있다[1]. 본 논문에서는 과거의 정보를 다루는 색인 기법을 고려한다. 과거의 정보는 이동 객체의 움직임에 따라 연속적으로 자신의 위치에 대한 정보를 저장해야 하므로 색인의 크기가 계속해서 증가한다. 색인의 크기를 줄이기 위한 두 가지 접근법이 있는데 그 중 하나는 특정 시간에 위치를 추출하는 샘플링기법이고 다른 하나는 이동 객체가 그들의 속도나 방향을 바꿀 때만 변화를 갱신하는 방법이다. 그러나 이런 기법은 측정된 위치 사이에서의 객체의 이동에 관해서는 질의응답을 할 수 없게 된다. 전체의 움직임을 얻기 위해 가장 간단한 접근방법으로 선형 보간법을 사용한다.

본 논문에서 고려하는 환경에서의 객체는 주로 사람이므로 객체의 움직임이 자유롭다. 따라서 속도 등을 측정하는 것 보다는 샘플링 기법이 더 효율적이다. 또한 제안하는 STS-tree는 하드디스크에 저장하는 고정 R-tree와 메모리에 유지하는 시간차원만을 고려한 TB-tree를 사용해 각각 고정 객체와 이동 객체를 저장하는 방법이다.

논문의 구성은 다음과 같다. 2장에서는 기존의 과거정보에 대한 색인기법에 대해서 알아보고 그 중 R-tree와 TB-tree에 대해서 자세히 살펴본다. 그리고 3장에서는 본 연구에서 제안한 STS-tree의 구조에 대해서 살펴보고, 4장에서는 STS-tree의 삽입과

검색 알고리즘에 대해서 설명한다. 그리고 마지막으로 결론을 기술한다.

2. 관련연구

과거의 정보를 다루는 방법[1]에서 첫 번째로, 시간의 차원을 고려한 방법에는 R-tree와 TSB-tree의 결합인 RT-tree, 그리고 공간의 차원에 시간의 차원을 더해 시간과 공간 질의에 구분을 없앤 3D R-tree[2], R-tree와 다른 삽입/분할 알고리즘을 가진 R-tree의 확장인 STR-tree 등이 있다.

두 번째, 과거의 정보를 각 시간마다 분리된 R-tree로 구축하는 overlapping · 다중버전 구조에는 R-tree의 구조에 overlapping B-tree의 아이디어를 합친 MR-tree가 있다. 또, MR-tree와 유사한 HR-tree(Historical R-tree)와 HR-tree에서 엔트리들의 중복을 피하기 위해 설계된 HR+-tree가 있으며, 다중버전 B-tree에 기반한 MV3R-tree가 있다. MV3R-tree에서는 두 개의 tree를 구축하는데 그 중 MVR-tree는 타임스탬프 질의를 처리하고, 3D R-tree는 긴 간격 질의를 처리한다.

마지막으로, 과거 궤적을 저장하고 검색하기 위해 제안된 구조로는 TB-tree (Trajectory-Bundle tree), SETI(Scalable and Efficient Trajectory Index) 그리고 SEB-tree(Start/End timestamp B-tree)등이 있다. 이 중 STR-tree의 확장인 TB-tree는 잎 노드에서 같은 궤적을 따르는 부분만을 담고 있다는 특징이 있다.

이 중에서 본 논문에서 이용하는 R-tree와 TB-tree 두 가지를 자세히 살펴보겠다.

2.1 R-tree

R-tree는 공간 객체를 색인하기 위해 최소 경계 사각형(Minimum Bounding Rectangle : MBR)을 사용한 B-tree의 확장이다[3]. 색인은 계층적으로 이루어져 있으므로 적은 노드들만을 방문함으로써 공간 검색을 가능하게 한다. 리프 노드는 최소 m (m<=M/2)개에서 최대 M개의 엔트리들을 담고 있고 데이터 객체에 대한 포인터를 포함하고 있다. 또한 R-tree는 높이 균형 트리이므로 모든 리프 노드들은 같은 레벨에 위치해 있다. 색인은 디스크 I/O 수를 최소화 할 수 있도록 디스크 특성에 적합하게 설계되어 있다.

2.2 TB-tree (Trajectory-Bundle tree)

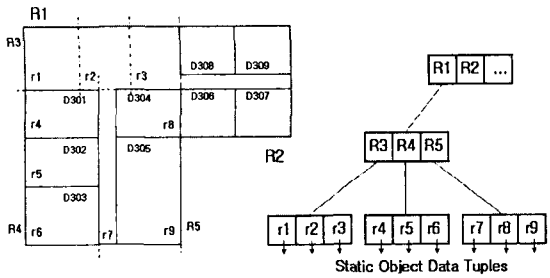
TB-tree는 한 리프 노드에 같은 궤적에 속하는 세그먼트들만을 포함해서 궤적을 보존(Trajectory Preservation)하는 접근 방법이다[4]. 그러므로 저장 공간의 오버헤드가 없으며 궤적 질의에 탁월한 성능을 갖는다. 객체의 삽입은 선행 객체가 삽입된 리프 노드를 찾은 후 노드에 빈공간이 있으면 바로 삽입하고, 노드가 가득 찬 경우에는 분할을 해야 하는데 이것은 전체 궤적 보존의 원칙을 위반하는 것이므로 대신 새로운 리프 노드를 생성해서 객체를 삽입한다. 이렇게 생성된 리프 노드를 이전 리프노드가 가리키도록 한다. TB-tree는 궤적을 보존하는 정책으로 인해 단말 노드의 MBR이 커지고 객체가 많을수록 MBR의 겹침 현상이 두드러져 질의 성능이 나빠질 수 있다.

3. STS-tree 구조

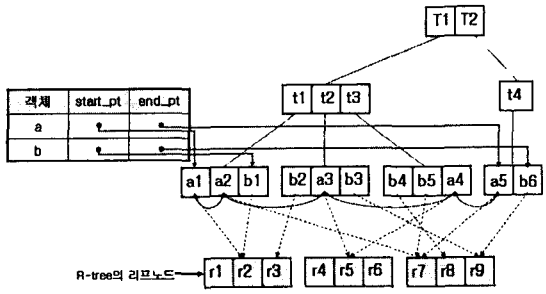
본 연구에서는 빌딩 또는 놀이동산과 같은 제한된 영역을 대상으로 하므로 기존 연구와는 몇 가지 차이점이 있다.

우선, 제한된 공간에서의 객체를 살펴보면 방, 엘리베이터, 정수기 등과 같이 위치가 변하지 않는 고정 객체와 사람이나 이동장비와 같은 이동 객체로 나누어 볼 수 있다. 고정 객체에 비해 이동 객체는 지속적인 갱신이 이루어져야 하므로 이들 객체를 하나의 색인 기법에 저장하는 것은 비효율적이다. 따라서 이동 객체와 고정 객체를 나누어서 각각의 트리에 저장한다. 고정된 객체는 고정된 공간 객체를 저장하기에 가장 적합한 R-tree를 이용한다. 가능하다면 R-tree는 공간의 평면도 등을 이용해 미리 구성한다. 또 이동 객체는 과거의 정보를 저장하기 위해 수정된 TB-tree를 사용한다.

다음으로, 논문에서 고려하는 환경은 공간이 제한되어 있으므로 기존의 방법에서와는 달리 공간(x, y)축과 시간(t)축이 모두 성장하는 것이 아니라, x, y축은 한정된 반면 t축은 지속적으로 성장한다. 이와 같이 각 차원의 특징이 서로 다르므로 기존의 방법에서처럼 시간차원과 공간차원을 함께 유지하는 것보다는 분리하는 것이 더 효율적이다. 그리고 공간정보는 변하지 않으므로 매번 공간에 대한 MBR을 수정하는 것은 효율적이지 않다. 따라서 TB-tree는 시간차원만을 고려하고 공간에 대해서는 R-tree의 MBR#만을 유지함으로써 시간과 공간을 분리한다. 이때 TB-tree의 MBR은 하위 노드들의 lifetime을 포함하도록 구성한다.



(그림 1) 고정 객체를 저장하기 위한 R-tree



(그림 2) 이동 객체를 유지하기 위한 TB-tree

또한, TB-tree가 1차원(시간 차원)으로 유지되어 있으므로 기존의 TB-tree에서처럼 한 노드에 하나의 객체만을 유지하게 되면 새로운 객체가 들어올 때마다 노드를 생성해야 하고, 따라서 MBR의 겹침 현상이 발생한다. 그러므로 하나의 노드에 여러 가지 다른 객체들을 담을 수 있도록 수정한다.

R-tree에 유지되는 고정 객체들은 거의 수정되지 않고 그 형태를 유지하는 반면, TB-tree에 유지되는 이동 객체들은 그들의 위치가 연속적으로 변한다. 이동객체의 위치 데이터는 다음 데이터가 들어오기 전에 현재 데이터가 데이터베이스에 저장되어야 한다. 그러나 기존의 방법에서처럼 하드디스크에 저장한다면 디스크의 입출력 시간이 오래 걸리므로 적합하지 않다. 따라서 본 연구에서는 TB-tree를 메모리에, R-tree는 하드디스크에 저장하고 TB-tree를 주기적으로 디스크에 백업한다. 그리고 R-tree의 내부 노드 구조는 TB-tree의 삽입과 검색 시 계속 검색되어야 하므로 메모리에 유지한다.

각 트리의 노드 구조를 살펴보면, R-tree의 단말 노드와 비 단말 노드는 단말 노드 엔트리에 영역 식별자인 MBR#가 추가된다는 것을 제외하고는 기존과 같다. TB-tree의 단말 노드 엔트리는 {객체ID, (x_i, y_i, t_i), $(x_{i+1}, y_{i+1}, t_{i+1})$, *MBR#*, *pre*, *next*}를 담고 있고, 비 단말 노드 엔트리는 {t_j, t_{j+1}, 포인터}를 담고 있다. 여기서 MBR#는 고정된 R-tree의 MBR의 식별자를 가리키며, pre와 next는 각각 이전과 다음 객체를 가리키는 포인터이다. 또, 포인터는 하위노드를 가리키는 포인터이다.

추가적으로 객체별 시작 포인터와 마지막 포인터를 유지하는 테이블을 갖는다[5]. 시작 포인터는 객체의 궤적 정보 검색을 위해 사용되고, 마지막 포인터는 TB-tree에서의 객체 삽입 시 매번 TB-tree를 검색하여 선행자를 찾아서 링크를 설정하는 것이 비효율적이므로 테이블 검색을 통해 바로 값을 얻어 링크를 설정하기 위해 사용된다.

(그림 1)과 (그림 2)는 각각 고정 객체를 위한 R-tree와 이동 객체를 위한 TB-tree의 구조를 나타낸다.

4. STS-tree 알고리즘

4.1 삽입

```
Algorithm insert(E)
  N = searchRtree(E, R)
    /* R : R-tree의 루트 */
  if(E.type == STATIC)
    /* #define STATIC 1 */
    Insert E to N
  else {
    NE = createNewEntry(E)
    NE.MBR# = N.MBR#
    LN = getLastNode()
     /*LN : 마지막으로 객체가 삽입된 노드*/
    if(LN.entryNum == M){
      /* M : 노드에 삽입될 수 있는 최대 엔트리수*/
      NN = createNewNode()
      setLastNode(NN)
      LN = NN
    }
    Insert NE to LN
    if((EP =getEndPtr(E))!= NULL) {
      NE.pre = EP
      EP.next = NE
    }
    setEndPtr(NE)
  }
```

(그림 3) 삽입 알고리즘

삽입할 객체는 (객체 ID, MBR, type)의 속성을 갖는다. 여기서 type은 객체가 이동 객체인지 고정 객체인지를 나타낸다. 객체의 삽입은 객체가 고정 객체인지 이동 객체인지에 따라 구별된다. 고정 객체라면 기존의 R-tree의 삽입방법과 같은 방법으로 삽입된다. 그러나 R-tree의 리프노드가 가득차서 분할이 일어난다면 R-tree의 리프노드들을 갖고 있는 TB-tree도 모두 갱신되어야 하므로 R-tree의 리프 레벨은 노드 당 최소/최대 객체의 제한을 두지 않는다. 이동 객체라면 우선 R-tree의 기존 검색 알고리즘을 사용해서 R-tree를 검색한 후 리프노드를 리턴받는다. 그 리프 노드의 MBR#를 삽입할 객체의 MBR#로 설정한다. 객체의 삽입은 시간 순으로 이루어지므로 가장 마지막에 삽입된 객체가 속한 노드에 공간이 있다면 그 노드에 삽입되고 그렇지 않다면 TB-tree의 삽입방법에 따라 새로운 노드를 생성한다. 객체별 시작과 끝 포인터를 담고 있는 테이블에서 삽입할 객체에 대한 마지막 객체를 반환받아 링크를 설정한다.

4.2 검색

질의의 종류를 나누는 방법은 여러 가지가 있을 수 있지만, 여기서는 영역질의(range query), 궤적질의(trajectory-based query), 그리고 이 두 가지를 결합한 복합질의(combine query) 이렇게 세 가지로 나누어서 생각하겠다. 영역질의는 time-slice, time-interval로 나누어서 생각할 수도 있지만 time-slice질의는 time-interval 질의의 시간차원이 0인 범위에 대한 특별한 경우를 의미하므로 하나로 생각한다.

영역질의는 앞의 (그림 1)과 (그림 2)에서 예를 들면, "시간 t2동안 D302호에 머물렀던 객체를 검색하라."가 있을 수 있다. 이때, 우선 R-tree를 통해 D302호가 속한 리프 노드의 MBR#인 r5를 구한다. 그리고 TB-tree를 통해서 t2동안에 존재하는 객체인 b2, a3, b3을 구한 후, 그 객체들의 MBR#가 r5인 객체 a3을 구한다.

궤적질의는 단순히 객체별 포인터 테이블에서 시작 객체를 구한 후, 포인터를 따라가면 구할 수 있다.

마지막으로 복합질의는 예를 들어 "시간 t2동안 S에 머물렀던 객체의 이전 10분간의 이동 경로를 검색하라."가 있을 수 있다. 이 질의에서 "시간 t2동안에 D302호에 머물렀던 객체"를 찾는 질의를 내부 질의영역(inner range)이라고 하고, "이전 10분간의 이동경로"는 외부 질의영역(outer range)이라고 한다. 내부 질의영역에 속하는 객체를 위의 영역질의의 방법으로 찾아서 객체의 아이디인 a3을 저장하고, pre 포인터를 이용해 외부 질의영역에 속하는 궤적을 찾음으로써 처리한다.

5. 결론

본 논문에서 제안한 STS-tree는 제한된 영역에서의 객체를 이동하는 객체와 고정된 객체로 분리해서 각각 R-tree와 TB-tree에 저장함으로써 보다 효율적이다. 또한 제한된 영역에서는 공간의 확장 없이 시간 영역만이 계속 확장된다는 점을 이용해 시간과 공간의 차원을 나누어서 TB-tree는 시간차원만을 이용한 1차원으로 구축하고, R-tree의 공간정보를 그대로 사용함으로써 특성이 다른 두 차원을 분리했다. 따라서 궤적 정보를 유지하는 색인 기법에서 영역질의를 처리할 때 비효율적이었던 것에 비해, 제안된 트리를 사용하면 영역질의 뿐 아니라, 궤적질의, 그리고 이 둘을 결합한 복합질의까지 효율적으로 처리할 수 있을 것으로 기대된다.

참고문헌

[1] M. F. Mokbel, T. M. Ghanem, and W. G. Aref "Spatio-temporal Access Methods" IEEE Data Engineering Bulletin, 26(2), pp.40-49, 2003.

[2] Y. Theodoridis, M. Vazirgiannis, and T. Sellis. "Spatio-Temporal Indexing for Large Multimedia Applications" In Proc. of the IEEE Conference on Multimedia Computing and Systems, 1996.

[3] A. Guttman. "R-Trees: A Dynamic Index Structure for Spatial Searching" In Proc. of the ACM International Conference on Management of Data, pp.47-57, 1984.

[4] D. Pfoser, C.S. Jensen, and Y. Theodoridis "Novel Approaches in Query Processing for Moving Objects" In Proc. of the International Conference on VLDB, pp.395-406, 2000.

[5] 심춘보, 강홍민, 엄정호, 장재우 "위치 기반 서비스에서 이동 객체의 궤적을 위한 TB-트리의 확장" 정보과학회 2004년 춘계학술대회, 31(1), pp.0142-0144, 2004.