韓國 컴퓨터情報學會 論文誌
第13卷 第1號, 2008. 1.

2008-13-1-1-2

효율적인 최근접 질의 처리를 위한 Voronoi 다이어그램 기반 그리드 검색 구조

권 동 섭*

A Voronoi Diagram-Based Grid Structure for Efficient Nearest Neighbor Query Processing

Dongseop Kwon*

요 약

최근접 질의(nearest-neighbor query)는 멀티미디어 시스템이나 GIS 시스템과 같은 여러 가지 응용 분야에서 사용되는 중요한 질의 처리 기법 중 하나이다. 최근접 검색 기법들을 위한 다양한 연구가 제안되었으나 이러한 기법들은 질의 수행 시 데이터를 검색하여 최근접 질의를 처리하므로 성능의 한계가 있었다. 본 논문에서는 정적인 데이터에 대하여 Voronoi 다이어그램을 이용한 전처리를 통하여 최근접 질의의 결과를 미리 계산하고 이 결과를 그리드 기반 검색 구조를 이용하여 저장하는 기법을 제안한다. 이 기법은 데이터 자체를 색인하는 기존의 기법과는 달리, 질의의 결과를 미리 색인하므로 대량의 데이터에 대해서도 기존의 기법보다 빠르게 최근접 질의를 처리할 수 있다.

Abstract

Nearest-neighbor searches are essential operations in various applications such as multimedia systems and GIS systems. Although numbers of research works for nearest-neighbor search have been proposed, they have a limitation on the performance since they process queries on the fly with indexes on data. This paper proposes a new nearest-neighbor search algorithm based on a grid-based data structure, which preprocesses and stores the result of nearest-neighbor queries using Voronoi diagrams over static data. While traditional techniques try to index data itself, the proposed technique attempts to index the result of the queries. Therefore, it performs nearest-neighbor queries more efficiently.

▸ Keyword : Voronoi 다이어그램(Voronoi diagram), 데이터베이스(Database), 최근접 검색질의(Nearest-neighbor search queries)

• 제1저자 : 권동섭
• 접수일 : 2007. 11. 15, 심사일 : 2008. 1. 8, 심사완료일 : 2008. 1.25.
* 명지대학교 컴퓨터소프트웨어학과 조교수

Ⅰ. 서 론

최근 멀티미디어 데이터베이스 응용이나 GIS(Geographic Information System), 데이터마이닝 등의 새로운 데이터베이스 응용 분야의 중요성이 증가함에 따라 최근접 검색(nearest neighbor search)은 중요한 데이터베이스 연산으로 부각되었다. 지금까지 다차원 데이터에 대한 최근접 검색 기법은 일반적으로 전체 데이터를 특정한 공간 인덱싱 구조에 저장하고 이 인덱스를 이용하여 질의와 가장 유사한 데이터를 검색하는 방법이었다. 하지만, 이러한 인덱스에 기반한 기법들은 데이터의 차원이 높아지고 크기가 커질수록 인덱스 구조의 효율성이 급격히 떨어지는 단점이 존재한다는 것이 밝혀졌다[1,2]. 최근접 질의 처리 기법의 대안으로서 데이터 자체를 인덱싱하는 대신 최근접 질의의 결과를 계산, 저장하여 두고 이를 이용하여 빠르게 최근접 질의를 처리하고자 하는 방법[3]이 발표되었다. 이러한 방법은 처음 인덱스 생성시 많은 계산이 필요하지만, 실제 최근접 질의 시에는 더 빠르게 결과를 사용자에게 돌려줄 수 있는 장점이 있다.

본 논문에서는 읽기가 주류를 이루는 대량의 저차원 데이터를 대상으로 한 새로운 최근접 검색 질의 수행 기법을 제안하였다. 읽기가 주류를 이루는 경우 데이터의 수정이나 삽입, 삭제에 대한 부담이 없으므로, 미리 대량의 데이터에 대해서 많은 계산을 해 둠으로써 실제 최근접 질의 수행 시에는 기존의 기법들보다 빠르게 질의를 수행할 수 있다. 본 논문에서는 Voronoi 다이어그램[4]을 이용하여 미리 최근접 검색의 결과를 격자를 이용하여 분할 저장하여 둔다. 그리고, 이 인덱스를 이용하여 최근접 질의를 빠른 시간에 처리할 수 있는 방법을 제시한다. 이 방법을 이용하면 최근접 질의의 결과를 구하기 위해 디스크에 접근하는 회수를 줄일 수 있으므로 대량의 데이터에 대해서도 빠른 응답 시간을 보장할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 소개한다. 3장에서는 본 논문에서 제안하는 새로운 최근접 검색 기법인 VDGrid 기법을 소개한다. 4장에서는 실험을 통하여 VDGrid 기법을 다른 기법과 비교 분석한다. 마지막으로 5장에서 결론을 맺고 추후 연구과제를 언급한다.

Ⅱ. 관련 연구

2.1 Voronoi 다이어그램

본 논문에서는 최근접 검색의 결과를 미리 계산하기 위하여 데이터에 대한 Voronoi 다이어그램[4]을 이용한다. Voronoi 다이어그램에 대한 정의는 다음과 같다.

정의 1. Voronoi 셀, Voronoi 다이어그램

DB를 d차원의 점의 집합이라고 하자. 다음과 같이 원소의 개수가 m개인 DB의 부분집합 A가 존재하고($A \subset DB$, $m = |A|$), d차원 상의 거리 함수 f_d가 정의되어 있을 때, A에 대한 차수가 m인 Voronoi 셀, $VoronoiCell_m(A)$는 다음과 같이 정의된다.

$$VoronoiCell_m(A) = \begin{cases} x \in R^d \mid \forall (P_i \in A)\ \forall (P_j \in DB - A) : \\ f_d(x, P_i) \le f_d(x, P_j) \bigwedge \end{cases}$$

그리고, DB에 대한 차수가 m인 Voronoi 다이어그램, $VoronoiDiagram_m(DB)$는 다음과 같이 정의된다.

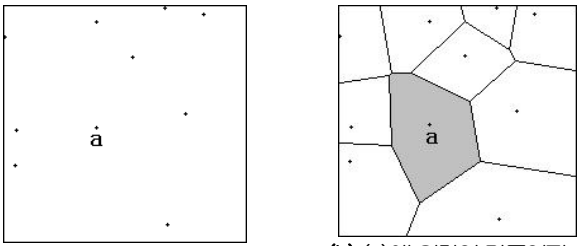
$$VoronoiDiagram_m(DB) = \{\, VoronoiCell_m(A) \mid A \subset DB \bigwedge$$

일반적으로 DB에 속하는 점들을 사이트(site)라고 부른다. 위의 정의를 차수가 1인 경우를 예로 들어 설명하면, 차수가 1인 Voronoi 셀은 가장 가까운 점이 같은 모든 점의 집합이고, 차수가 1인 Voronoi 다이어그램은 차수가 1인 Voronoi 셀의 집합이다. 예를 들어 2차원의 유클리디안 공간에서 〈그림 2-1〉 (a)에서 점 a에 대한 Voronoi 셀, $VoronoiCell_1(a)$은 〈그림 2-1〉 (b)의 음영으로 나타난 영역이고, 〈그림 2-1〉 (a)의 10개의 점에 대한 Voronoi 다이어그램은 〈그림 2-1〉 (b)와 같다.

d차원의 유클리디안 공간에서 n개의 점 사이트에 대한 차수가 1인 Voronoi 다이어그램을 만드는 알고리즘의 최대 복잡도는 $\Theta(n^{\lceil d/2 \rceil})$이다. 그리고, 이 Voronoi 다이어그램은 $O(n \log n + n^{\lceil d/2 \rceil})$에 생성될 수 있다.[4]

2.2 최근접 검색 기법

지금까지 다차원의 데이터를 처리하기 위하여 많은 다차원 접근 방법(multidimensional access method)[5]이 제안되었다. 이러한 방법들 가운데 널리 이용되는 R-tree[6]를 이용한 대표적인 k-최근접 검색(k-nearest neighbor search)기법은 차원 감소 기법을 이용한 방법[7][8]이다.



(a) 10개의 점 (b) (a)에 의하여 만들어진 1차 Voronoi 다이어그램

〈그림 2-1〉 Voronoi 다이어그램의 예
〈Figure 2-1〉 An example of Voronoi diagram

이 방법은 k개의 후보군이 가질 수 있는 거리의 최대값을 유추하고 이 값에 의해 범위 검색을 수행하여 후보군을 선택한다. 그리고, 이 후보군의 거리를 실제 비교하여 k개의 최근접 객체를 찾는다. 하지만, 이 방법은 실제 필요한 회수보다 많은 디스크 접근을 하게 되는 단점이 있다. 이러한 단점을 극복하기 위하여 다단계 k-최근접 검색 기법[9]이 제안되었다. 이 방법은 우선 순위 큐를 이용하여 다단계로 R-tree를 순회하며 k개의 최근접 객체를 찾는다. 이 방법은 점진적인 순위 매김 기법을 적용함으로써 최소한의 후보만을 검색해보고 최근접 검색을 수행할 수 있는 장점이 있다. 이러한 기법은 다차원 접근 방법을 이용하여 최근접 검색시의 디스크 접근 회수를 줄이지만, 차원이 증가하고 데이터의 크기가 커짐에 따라 인덱스의 효율이 급격히 떨어지고 질의 처리시 디스크 접근 회수가 급격히 증가하게 되는 단점이 있다. 그리고, 최근접 검색 자체가 다른 질의에 비하여 복잡한 처리 과정이 필요하므로 데이터의 개수가 많은 경우 여전히 다른 연산에 비하여 많은 수의 디스크 접근이 필요하다.

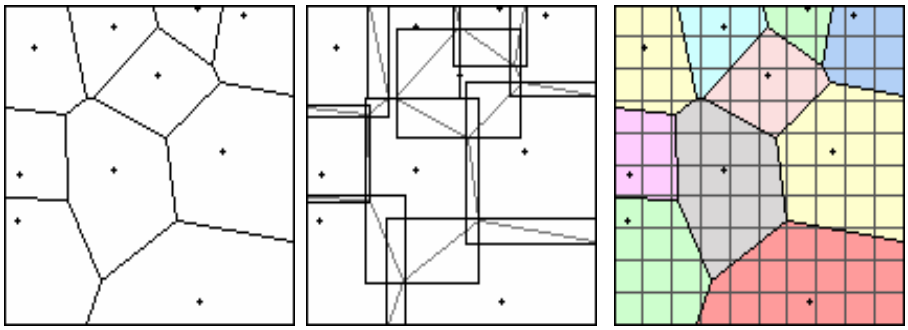
2.3 MBR-근사(MBR-Approximation) 기법

지금까지 대부분의 최근접 검색에 관한 연구는 데이터를 인덱싱하고 이를 이용하여 최근접 검색을 수행하는 방법을 사용하고 있다. 이러한 기존 방법들과는 달리 미리 모든 최근접 검색의 해를 구해두고 이를 이용하여 최근접 검색을 수행하는 새로운 접근 방법[3]이 제안되었다. 이 방법은 미리 주어진 데이터에 대한 Voronoi 다이어그램을 구한다. 그리고, 각 Voronoi 셀을 MBR(minimum boundary region)로 근사하여 이 MBR을 R*-tree[10]에 저장한다. 질의점이 어느 Voronoi 셀에 속하는지만 결정하면 최근접 질의의 결과가 되므로, 질의가 주어지면 이 R*-tree에 점 질의를 수행하여 질의점이 속하는 MBR만 찾으면 최근접 질의의 결과를 찾을 수 있다. 이때 MBR이 서로 겹칠 수 있으므로 실제로는 질의점이 속하는 MBR들 사이에서 실제 거리 비교를 통하여 최근접점을 찾을 수 있다. 이 방법은 미리 Voronoi 다이어그램을 구해야 되기 때문에 생성 시간이 많이 들지만, 최근접 검색 질의를 점 검색 질의로 바꾸어 수행할 수 있으므로 기존의 기법들 보다 빠르게 최근접 검색 질의를 수행할 수 있다. 특히 고차원에서는 기존의 기법에 비해 월등히 뛰어난 수행 성능을 보인다. 하지만, 이러한 MBR-근사 기법은 Voronoi 셀을 MBR로 근사하면서 중첩(overlap) 영역이 많이 발생하게 되고 그 결과로 거리 비교를 해야 하는 후보가 늘어나기 때문에 성능이 떨어지게 된다. 이러한 문제를 해결하기 위하여 고차원에서 근사시킨 MBR을 분해하는 기법을 함께 제안했지만, 실험 결과 10차원 미만의 저차원에서는 기존의 기법에 비해 성능 향상이 거의 없었다. 그리고, 2차원의 데이터의 경우 중첩 영역이 많을 경우 오히려 기존의 기법에 비해 성능이 떨어지는 경우도 발생하였다.

III. VDGrid 기법

저차원의 데이터에서 MBR-근사 기법이 가지는 단점을 보완하기 위하여 본 논문에서는 VDGrid (Voronoi Diagram-Grid) 기법을 제안하였다. VDGrid 기법이란 최근접 검색의 해가 되는 Voronoi 다이어그램을 구하여 이를 미리 정해진 수의 격자(grid)로 저장하고, 이 정보를 이용하여 최근접 검색을 수행하는 기법이다. VDGrid 기법의 특성상 최근접 검색의 대상 데이터와 최근접 검색 방법은 다음과 같이 제한된다.

- 저차원 데이터 : 공간을 격자로 분할하여야 하므로 고차원에서는 너무 많은 개수의 격자가 필요하다. 따라서 VDGrid 기법은 저차원의 데이터를 대상으로 한다. 본 논문에서는 저차원 가운데서도 2차원의 데이터를 대상으로 하여 검색 기법을 제안한다. 하지만, 본 논문에서



(가) 10개의 점에 의한 Voronoi 다이어그램 (나) (가)에 의한 MBR-근사기법의 예 (다) (가)에 의한 10x10 VGrid의 예

〈그림 3-1〉 Voronoi 다이어그램과 MBR-근사 기법, VDGrid 기법
〈Figure 3-1 Voronoi Diagram, MBR-Approximation, and VDGrid〉

제안하는 알고리즘들은 같은 방법으로 3차원이나 4차원 정도의 저차원 데이터에 확장 적용 가능하다.

- 점 데이터 : 다각형이나 선분과 같은 객체에 대해서는 Voronoi 다이어그램을 생성할 수 없으므로 VDGrid 기법을 수행할 수 없다. VDGrid 기법은 점 데이터만을 대상으로 한다.
- 1-최근접 검색(1-nearest neighbor search) : 본 논문에서는 1-최근접 검색 질의에 대한 처리를 위한 알고리즘을 제시한다. VDGrid 기법은 미리 최근접 검색의 해를 구하여 두는 방법이기 때문에, k-최근접 검색을 수행하기 위해서는 미리 Voronoi 다이어그램을 계산할 때 차수가 1인 Voronoi 다이어그램을 생성하는 대신 차수가 k인 Voronoi 다이어그램을 생성하고 이를 이용하여 VDGrid를 생성하면 된다.
- 정적인 데이터 : 본 논문에서는 최근접 검색의 대상이 되는 데이터는 정적인 것으로 가정한다. Voronoi 다이어그램을 생성하고 VDGrid를 만드는 데는 많은 시간이 필요할 뿐 아니라, Voronoi 다이어그램의 특징상 하나의 데이터가 추가되거나 삭제되면 Voronoi 다이어그램을 다시 생성해야 하므로 VDGrid 기법은 동적인 환경에는 적합하지 않다. 물론, 동적인 Voronoi 다이어그램 생성 기법이 있으므로 동적인 환경을 어느 정도 지원할 수 있다. 이에 대해서는 7절에서 언급하도록 하겠다. 하지만, 기본적으로 VDGrid는 정적인 데이터에 적합한 기법이다. GIS 응용 같은 여러 가지 응용분야에서 정적인 대량의 데이터에 대한 최근접 검색이 필요한 경우가 있으므로 이러한 환경에서 VDGrid 기법을 적용할 수 있다.

3.1 VDGrid 기법의 정의

Grid 기법은 전체 공간을 미리 정의한 수의 격자로 나누고, 각 격자를 지나는 Voronoi 셀을 하나의 노드에 저장하는 방법이다. 이 때 각 격자의 크기는 동일하다. 예를 들어, 〈그림 3-1〉의 (가)와 같은 Voronoi 다이어그램이 있을 때, MBR-근사 기법은 〈그림 3-1〉의 (나)와 같이 각 Voronoi 셀을 MBR로 근사하여 R*-tree에 저장하게 된다. 하지만, 그림에서 볼 수 있듯이 Voronoi 다이어그램은 각 셀간에 겹치는 영역이 전혀 없지만, MBR-근사 기법은 각 MBR간에 겹치는 영역이 많이 발생하게 된다. 겹치는 영역이 많아지면 최근접 검색의 후보도 많아지기 때문에, 실제 10개의 점 자체를 R*-tree에 저장하여 두고 최근접 검색을 수행하는 기존의 기법과 큰 차이가 나지 않는다. 〈그림 3-1〉의 (다)는 10×10 VDGrid의 예이다. 전체 영역을 10×10의 격자로 분할하여 각 격자에 어떠한 Voronoi 셀이 지나가는지를 저장해두면 실제 최근접 질의가 들어왔을 때 질의점이 지나는 격자에 해당하는 Voronoi 셀만 최근접 검색의 후보로 고려하면 된다. 각 격자에 해당하는 Voronoi 셀의 정보를 하나의 디스크 페이지에 저장한다면 VDGrid 기법은 한 번의 디스크 접근만으로 최근접 검색의 결과를 얻을 수 있다.

3.2 VDGrid의 자료 구조

VDGrid의 각 격자는 하나의 노드에 저장된다. VDGrid의 전체 격자의 수는 미리 정해져 있으므로 기본 노드의 수는 처음부터 정해지게 된다. 하나의 노드는 하나의 디스크 페이지에 기록되게 된다. 그리고, 이 기본 노드들은 디스크에 선형적으로 저장된다. 따라서, 특정한 격자에 해당하는 노드의 위치는 즉시 알 수 있다. 노드와 노드 내의 엔트리 구조는 〈그

노드 자료구조	엔트리 자료구조

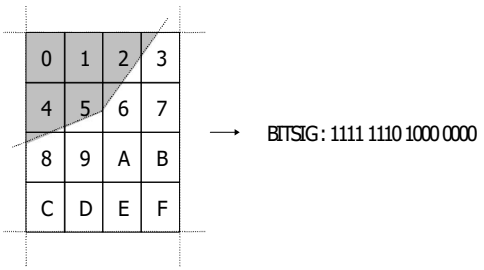
structure Node {
NumOfEntry n;
Entry data[MAX_ENTRY];
}
	structure Entry {
	Point p
	ObjectID oid
	Bit-Signature BITSIG
	}

〈그림 3-2〉 노드와 엔트리의 자료구조
〈Figure 3-2 Data structures for a node and an entry〉

림 3-2〉와 같다.

노드에는 하나의 격자를 지나는 Voronoi 셀에 대한 정보를 저장하게 된다. 하나의 노드는 하나의 디스크 페이지에 저장되어야한다. 그러므로 엔트리 개수의 최대값은 미리 정해져 있다. 하나의 엔트리는 하나의 Voronoi 셀에 대한 정보를 지니게 된다. 엔트리는 Voronoi 셀의 사이트(점)의 위치정보와 점에 해당하는 객체 ID, 그리고 Voronoi 셀에 대한 Bit-Signature로 구성된다. 2차원의 경우 일반적으로 점 데이터 8Byte, 객체 ID 4Byte, Bit-Signature 2Byte가 필요하므로 하나의 엔트리의 크기는 14Byte가 되고, 디스크 페이지의 크기가 4KB라면 하나의 노드는 292개의 엔트리를 가진다.

Voronoi 셀의 점 데이터와 객체 ID만으로 엔트리를 구성해도 되지만, 좀 더 효율적인 알고리즘 구성을 위하여 Bit-Signature라는 부가적인 정보를 엔트리에 추가하였다. 실제 하나의 격자 내에 여러 개의 Voronoi 셀이 지나간다 하더라도, 각 Voronoi 셀은 서로 겹치는 영역이 없으므로, 대부분의 Voronoi 셀은 격자의 일부분만을 지나게 된다. 이러한 경우를 위하여 Voronoi 셀이 격자의 어떠한 부분을 지나는 지를 저장하여 둔 것이 Bit-Signature이다. 예를 들어, 2차원의 데이터에 대해서 Bit-Signature를 2바이트 사용한다고 하자. 그러면 Bit-Signature는 16비트이므로 하나의 격자를 16부분(4 ×4)으로 분할할 수 있다. 〈그림 3-3〉과 같이 하나의



〈그림 3-3〉 Bit-Signature의 예
〈Figure 3-3 An example of a bit-signature〉

격자가 있을 때 어떤 Voronoi 셀이 지나는 영역이 음영으로 표시된 부분이라고 한다면, 하나의 격자를 그림과 같이 16부분으로 분할하고 Voronoi 셀이 지나는 부분은 1, 지나지 않는 부분은 0으로 나타내면 Bit-Signature는 1111 1110 1000 0000으로 나타난다. 이러한 Bit-Signature는 오버플로우 처리에 사용되는데 이는 4절에서 설명하겠다. 이 외에도 질의점의 Bit-Signature와 Voronoi 셀의 Bit-Signature를 AND 연산으로 비교하면 질의점과 만날 수 없는 Voronoi 셀을 미리 걸러낼 수 있기 때문에 실제 거리를 비교하는 회수를 줄일 수 있다. 따라서 최근접 검색 시 CPU 연산 시간이 줄일 수 있는 장점이 된다.

3.3 VDGrid의 생성 알고리즘

VDGrid는 〈그림 3-4〉의 알고리즘과 같은 방법으로 생성할 수 있다. VDGrid를 생성하기 위해서는 먼저 Voronoi 다이어그램을 생성하고, 이 Voronoi 다이어그램의 각 Voronoi 셀이 어떠한 격자를 지나는지를 검사하여 지나는 격자에 해당하는 노드에 Voronoi 셀의 정보를 저장하면 된다. 하나의 격자와 Voronoi 셀이 만나는지 검사해보는 방법은 여러 가지가 있을 수 있지만 여기서는 간단하게 Voronoi 셀을 격자로 클리핑하는 방법을 사용하였다. Sutherland-Hodgman의 폴리곤 클리핑 알고리즘[11]을 사용하여 Voronoi 셀을 클리핑하고 클리핑 결과가 있으면 겹치는 영역이 있는 것이므로 노드에 Voronoi 셀에 해당하는 정보를 저장하면 된다. Voronoi 셀에 해당하는 Bit-Signature를 계산하는 것도 같은 방법을 사용한다. 노드 내의 엔트리의 개수가 최대값에 도달하였으면 노드에는 더 이상의 엔트리를 추가할 수 없다. 이때 엔트리를 추가하려하면 오버플로우가 발생하게 되어 오버플로우에 대한 처리를 하여야 한다.

3.4 오버플로우 처리 기법

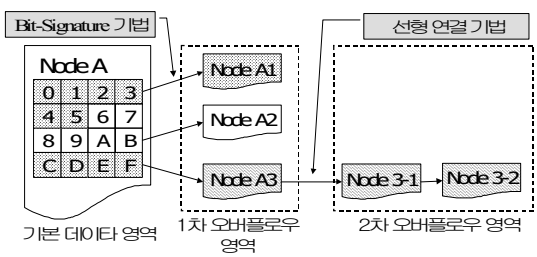
노드에 더 이상 엔트리를 저장할 공간이 없으면 오버플로우

```
Procedure BuildVDGrid(PointsData) {
  VoronoiDiagram ← ComputeVoronoiDiagram(PointsData);     // Voronoi 다이어그램 계산
  For each VoronoiCell in VoronoiDiagram {
    MBR ← GetMBR(VoronoiCell);
    For each Grid in MBR
      If (Overlap(VoronoiCell,Grid)) {
        Write VoronoiCell to GridNode(Grid);       // 오버플로우가 필요한 경우,
      }                                            // 오버플로우 처리
    }
  }
}
```

〈그림 3-4〉 VDGrid 생성 알고리즘
〈Figure 3-4 Algorithm for creating a VDGrid〉

처리를 해야 한다. 오버플로우 처리 방법은 Bit-Signature를 이용하는 방법과 선형 연결 기법 두 가지가 있다. 〈그림 3-5〉는 이 두 방법을 이용하여 오버플로우를 처리하는 예이다.

- Bit-Signature 기법 : 기본 데이터 영역의 노드에 오버플로우가 발생한 경우, 오버플로우가 발생한 노드를 Bit-Signature에 따라 분할하여 1차 오버플로우 노드에 저장한다. 한 비트마다 하나의 노드를 연결할 수 있으므로 2차원의 경우 2바이트의 Bit-Signature를 사용한다면 최대 16개의 노드를 연결할 수 있다. 이러한 방식으로 오버플로우를 처리하면 Bit-Signature를 비교하여 바로 다음 오버플로우 노드를 찾을 수 있게 되고, 최근접 검색의 후보는 다음 오버플로우 노드에 들어 있으므로 거리 비교를 해야하는 후보의 수는 늘어나지 않게 된다.
- 선형 연결 기법 : 1차 오버플로우 노드에서 다시 오버플로우가 발생한 경우, 더 이상 Bit-Signature에 따라 노드를 분할할 수 없으므로 이 때는 1차 오버플로우 노드에 선형으로 2차 오버플로우 노드를 계속 연결한다. 이는 해쉬에서 선형 체인을 이용하여 충돌이 일어나는 데이터를 처리하는 방법과 유사하다. 이러한 연결노드가 늘어나면 질의 수행 시 디스크 접근 회수가 증가할 뿐 아니라, 연결된 노드의 모든 엔트리가 거리 비교의 후보가 되므로 질의 수행 성능이 떨어진다. 따라서 오버플로우가 너무 많이 발생하면 격자를 더 작게 나누어 인덱스를 재구성하는 것이 바람직하다.



〈그림 3-5〉 오버플로우 처리
〈Figure 3-5 handling overflows〉

3.2절에서 언급했듯이 2차원의 경우 하나의 노드는 292개의 엔트리를 가지므로 Bit-Signature에 의하여 16개까지 노드가 분할 될 경우 하나의 노드에 4,672개의 엔트리가 저장될 수 있는데, 이 경우 격자의 크기는 1/16로 줄어드는 셈이 된다. 따라서, 처음 VDGrid를 생성할 때 선택한 격자의 개수가 너무 작지 않다면 일반적으로 기본 데이터 영역과 1차 오버플로우 영역 정도로도 대부분의 경우 데이터를 VDGrid에 저장할 수 있다.

3.5 최근접 검색 알고리즘

VDGrid의 최근접 검색기법은 〈그림 3-6〉의 알고리즘과 같다. 이 경우 오버플로우가 없다면 한번의 디스크 접근으로 최근접 검색을 수행할 수 있다. 오버플로우가 있을 경우 연결된 오버플로우 노드만큼 디스크 접근이 더 필요하다. 이에 비하여 R-tree계열의 인덱스를 이용한 경우 최소한 트리의 깊이 이상 디스크 접근이 필요하고, 겹치는 영역이 많거나 데이

```
Procedure NNsearchVDGrid (QueryPoint) {
    CandidateSet ← ∅;
    GridID ← GetGridID(QueryPoint);
    Node ← ReadNode(GetNodeID(GridID));
    while (TRUE) {
      CandidateSet ← CandidateSet ∪ ReadData(Node);
      if (HaveOverflowLink(Node)) Node ← ReadNextNode(Node);
      else break;
    }
    return FindNearestNeighbor(CandidateSet);
}
```

〈그림 3-6〉 VDGrid 최근접 검색 알고리즘
〈Figure 3-6 Algorithm for searching nearest-neighbor in VDGrid〉

터가 많은 경우 여러 번의 트리 탐색이 필요할 수도 있다. 특히 최근접 검색은 그 특성상 트리를 여러 번 탐색해 봐야 할 확률이 높다. MBR-근사 기법도 같은 수의 MBR을 R-tree에 저장해야 하므로 트리의 높이는 데이터를 직접 저장했을 때와 같다. 오히려 입력된 데이터 개수만큼의 Voronoi 셀을 MBR로 근사하여 트리에 저장해야 하므로 겹치는 영역이 증가하여 수행 성능을 저하시킬 수 있다. 따라서 VDGrid기법은 일반적인 공간 인덱스들의 최근접 검색 기법에 비해 빠르게 최근접 검색 질의를 처리할 수 있다.

3.6 격자 개수의 선택

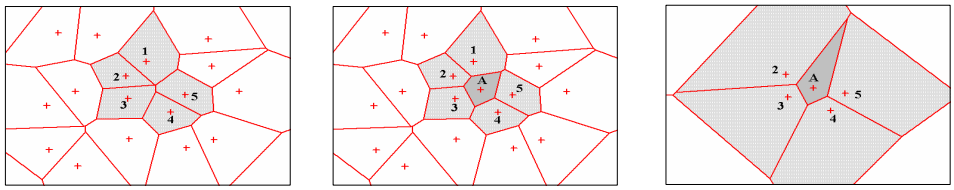
VDGrid의 검색 속도는 오버플로우의 개수와 밀접한 관계를 가진다. 따라서 오버플로우의 개수를 일정 수준 이하로 줄여야만 VDGrid 기법의 장점을 살릴 수 있다. 따라서 격자의 개수를 어떻게 선택하느냐가 VDGrid의 성능에 가장 중요한 요소가 된다. VDGrid는 정적인 데이터를 대상으로 하므로 미리 전체 데이터의 개수나 분포를 파악 최적의 VDGrid를 생성할 수 있다.

그리고, VDGrid의 최근접 검색시 디스크 접근 회수는 오버플로우 노드의 연결 개수와 같으므로 오버플로우의 최대 연결 개수를 구하면 최근접 검색 시간의 최대값을 추정할 수 있다. 그리고, 전체 격자의 평균 오버플로우 노드 연결 개수를 계산하면 평균 검색 시간도 추정할 수 있다. 그러므로, VDGrid의 경우 미리 오버플로우의 연결 개수를 n개 이하가 되도록 격자의 개수를 선택하여 VDGrid 인덱스를 생성한다면 디스크 접근 회수는 반드시 n회 이하이므로 최근접 질의의 수행 시간의 최대 한계를 미리 정의할 수 있다. 따라서, 최근접 질의의 응답시간이 매우 중요한 응용에서는 VDGrid를 이용하면 최근접 질의에 대한 응답 시간을 보장할 수 있다.

3.7 동적인 데이터의 지원

기본적으로 VDGrid 기법은 정적인 데이터에 적합하지만, 동적인 데이터의 지원이 완전히 불가능한 것은 아니다. 동적인 환경에서 Voronoi 다이어그램을 생성하는 기법[12]이 이미 소개되어 있지만, 실제 대량의 데이터의 경우 이러한 기법을 적용하는 것은 적합하지 않다. 여기서는 추가적인 데이터의 삽입을 처리할 수 있는 간단한 방법을 제시하겠다.

일반적으로 Voronoi 다이어그램은 전체 데이터에 대해서 계산하여야 하지만, 특정 점에 해당하는 Voronoi 셀은 그 셀과 이웃하는 셀의 점만으로도 계산할 수 있다. 예를 들어 〈그림 3-7〉의 (a)와 같은 상황에서 A점이 삽입되면 〈그림 3-7〉의 (b)와 같은 Voronoi 다이어그램이 생성된다. 이 두그림을 비교하여 보면 A점의 Voronoi 셀과 이웃하는 1, 2, 3, 4, 5 점에 해당하는 Voronoi 셀만 변경된 것을 알 수 있다. 동적인 Voronoi 다이어그램을 정확히 계산할 수 있다면 A의 Voronoi 셀을 VDGrid에 삽입하고, 1, 2, 3, 4, 5 점에 해당하는 Voronoi 셀만 VDGrid에서 변경하면 된다. 하지만, 크기가 줄어든 Voronoi 셀에 대한 변화를 무시하더라도 최근접 검색의 결과는 동일하다. 단지 줄어들지 않은 영역에서 최근접 검색의 후보수가 증가할 뿐이다. 따라서, A에 해당하는



(a) A점 삽입 이전의 Voronoi 다이어그램 (b) A점 삽입 이후의 Voronoi 다이어그램 (c) 가장 가까운 4개의 점만으로 계산한 Voronoi 다이어그램

〈그림 3-7〉 A점 삽입에 따른 Voronoi 다이어그램의 예
〈Figure 3-7 An example of a voronoi diagram after inserting point A〉

Voronoi 셀만 계산하여 삽입하면 된다. 실제 인접한 점을 모두 찾는 것 역시 쉽지 않다. 하지만 인접한 점을 모두 찾지 않더라도 최근접 검색의 결과는 동일하다. 만일 2, 3, 4, 5의 점과 A점 만으로 Voronoi 다이어그램을 구하더라도 〈그림 3-7〉의 (c)와 같이 A에 해당하는 Voronoi 셀의 크기가 커질 뿐이므로 늘어난 영역에서 후보수가 증가할 뿐이다. 따라서 가까운 몇 개의 점을 찾아 이를 이용하여 A의 Voronoi 셀을 구하고 이 셀을 VDGrid에 추가하면 된다. 물론, 이렇게 근사된 Voronoi 셀들이 많아지면, 그만큼 VDGrid가 비효율적이게 되고 검색시 불필요한 후보의 개수가 증가하게 된다. 따라서, 너무 많은 수의 점이 추가되면 전체 VDGrid를 새로 생성할 필요가 있다.

IV. 실험결과

이 장에서는 2차원의 점 데이터를 대상으로 하여 VDGrid 기법을 구현하고, Voronoi 셀을 MBR로 근사시키는 MBR-근사 기법, 그리고 R*-tree에 데이터를 저장하여 최근접 검색을 수행하는 기법과 VDGrid 기법을 비교 분석하였다. 이 실험에서 디스크 페이지의 크기는 4KB로 가정하였다. R*-tree는 Berchtold가 공개한 소스 코드[13]를 이용하였고, Voronoi 다이어그램 생성은 Qhull[14] 프로그램을 이용하였다. 실험 데이터로는 다음과 같은 두가지 데이터를 사용하였다.

- 데이터 A : 데이터 A는 균등 분포 데이터로서 전체 영역에 같은 확률로 분포된 2차원의 점 데이터이다. 점들이 일정한 영역에 편중되어 있지 않고 전체 영역에 고르게 분포되어 있다. 본 데이터는 실험을 위하여 난수를 이용하여 임의로 생성한 데이터이다.
- 데이터 B : 데이터 B는 특정한 영역에 많은 점이 편중된 데이터이다. 실험에 사용한 데이터는 Leutenegger가 공개한 CFD(Computation Fluid Dynamics) 2차원 점 데이터[15]이다. 이 데이터는 점의 분포가 공간의 중앙으로 매우 심하게 집중되어 있다. 실험에서는 점의 개수가 52,510개인 데이터와 208,688개인 데이터 두가지를 사용했다.

4.1 VDGrid 생성 실험

데이터 A와 B를 대상으로 VDGrid 생성시간을 R*-tree의 생성시간과 비교해 보았다. 데이터 A는 100×100의 VDGrid를 사용하였고, 데이터 B에 대하여는 50×50의 VDGrid를 이용하였다. 생성시간은 〈표 4-1〉과 같다.

〈표 4-1〉 VDGrid와 R*-tree 생성 시간 비교
〈Table 4-1 Creation time of VDGrid and R*-tree〉

데이터 종류	인덱스 종류	점 개수	생성 시간	비고
데이터 A (균등 분포 데이터)	VDGrid (100×100)	50만개	4.9분	오버플로우 없음
		100만개	9.2분	오버플로우 없음
	R*-tree	100만개	1.6분	트리 높이 3
데이터 B (편중 분포 데이터)	VDGrid (50×50)	52,510	31초	17개 격자에서 오버플로우 발생
		208,688	82초	35개 격자에서 오버플로우 발생
	R*-tree	52,510	23초	
		208,688	44초	

실험결과 VDGrid의 생성 시간이 R*-tree의 생성시간보다 많이 걸리는 볼 수 있다. 이는 VDGrid는 Voronoi 다이어그램을 생성하고 각 격자와 겹치는 지를 검사해야 하기 때문이다. 특히 100×100개의 격자를 사용한 경우가 50×50개의 격자를 사용하였을 때보다 월등히 많은 생성 시간이 필요했는데, 이는 단지 점의 개수가 많아진 것 뿐 아니라, 격자의 개수가 많아지면 Voronoi 셀과 겹치는지를 비교해야 하는 격자의 개수도 크게 늘어나기 때문이다. 실험 결과 균등 분포 데이터는 점의 개수가 많더라도 충분히 오버플로우 없이 생성할 수 있었다. 그리고, 분포에 편중이 심한 경우에도 오버플로우의 개수를 충분히 줄일 수 있었다. 참고로 편중 분포 데이터의 경우 100×100의 VDGrid를 사용하면 모두 오버플로우 없이 생성할 수 있었고, 50×50의 VDGrid를 사용하였을 때 점의 개수 52,510개에서는 오버플로우 노드의 깊이가 2였고, 208,688개일 때는 오버플로우 노드의 깊이가 3이였다.

4.2 최근접 질의 수행

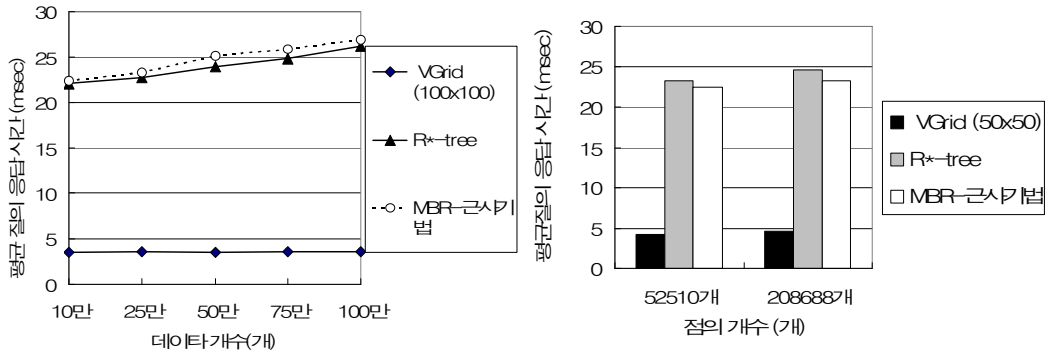
4.1절에서 생성한 VDGrid 인덱스와 R*-tree, MBR-근사 기법에 대해 임의로 생성한 10,000개의 점을 가지고 최근접 질의를 수행하였다. 〈그림 4-1〉은 이 때의 평균 최근접 검색 질의 응답 시간이다. 〈그림 4-1〉의 (a) 경우, VDGrid는 오버플로우가 발생하지 않았기 때문에 각 경우에 모두 한 번의 디스크 접근으로 최근접 질의를 수행할 수 있다. 따라서 질의 응답 시간은 거의 동일하였다. 이 때, R*-tree의 경우 트리의 높이는 3이었고, 실험 결과 평균 4~5회의 디스크 접근이 필요하였다. MBR-근사 기법 역시 R*-tree와 거의 유사한 응답 시간을 보여 주었다. 〈그림 4-1〉의 (b) 경우 VDGrid에 오버플로우 노드가 존재하므로 평균 응답 시간이 10% 정도 증가하으나 다른 기법 역시 데이터의 편중이 심하므로 질의 수행시간이 증가하였다. 실험 결과, 데이터 분포의 편중이 심한 경우에도 오버플로우 노드의 연결 개수가 많지 않고, 오버플로우가 발생하는 격자가 개수가 작으므로 성능이 크게 떨어지지는 않았다. VDGrid를 이용하면 다른 기법의 1/5 정도의 수행시간에 최근접 검색을 수행할 수 있었다.

4.3 실험 결과 분석

실험 결과를 살펴보면 VDGrid의 경우 인덱스의 생성 시간이 다른 기법에 경우 많이 걸리는 것을 알 수 있다. 특히 격자 개수가 많은 경우 6배 이상 생성시간이 걸리기도 하였다. 하지만, VDGrid 기법을 이용하면 최근접 검색 질의를 다른 기법의 1/5 정도의 시간에 수행할 수 있었다. 이는 다른 기법의 경우 평균 4~5회의 디스크 접근이 필요했지만, VDGrid 기법의 경우 평균 1번 정도의 디스크 접근으로 최근접 검색의 결과를 찾을 수 있었기 때문이다. 이는 VDGrid 기법의 경우 일반적으로 1번의 디스크 접근으로 최근접 검색의 결과를 찾을 수 있는데 반하여 다른 기법은 4~5회의 디스크 접근이 필요하기 때문이다. 물론, 편중된 데이터의 경우 오버플로우가 발생하므로 검색 시간이 10% 정도 증가하였다.

V. 결론

본 논문에서는 최근접 검색 알고리즘에 대한 새로운 접근 방법으로서 최근접 검색의 해인 Voronoi 다이어그램을 이용하여 해답 공간 자체를 인덱싱하는 VDGrid 기법을 제안하였다. 최근접 검색의 결과를 미리 계산해두는 방법이므로 처음



(a) 데이타 A(균등분포) (b) 데이타 B(편중분포)

〈그림 4-1〉 최근접 질의 수행 시간
〈Figure 4-1〉 Execution time for nearest-neighbor search

VDGrid 인덱스를 생성할 때는 부가적인 계산 시간이 필요하지만, 실제 최근접 검색을 수행할 때는 기존 기법보다 빠르게 최근접 검색의 결과를 구할 수 있다. 따라서 데이터의 변화가 작고 최근접 질의의 응답 시간이 중요한 응용에서는 효과적으로 사용될 수 있다. 본 논문에서는 VDGrid 기법을 위한 자료구조와 생성 알고리즘, 최근접 검색 알고리즘을 제안하였다. 비록 VDGrid 기법이 미리 검색 결과를 계산하여야 하므로 데이터의 변화가 없는 경우에 적합하지만, 데이터의 삽입과 같은 동적인 변화가 있을 경우 지원할 수 있는 방법도 제시하였다. 또한, 실험을 통하여 VDGrid 기법이 기존의 기법에 비해 효율적으로 최근접 검색을 수행할 수 있음을 보였다. 추후 연구 관제로 효율적인 VDGrid를 생성하기 위하여 데이터의 개수와 분포에 따른 VDGrid 최적화에 관한 연구가 필요하다. 그리고, 동적인 변화가 심한 경우에도 이용할 수 있도록 데이터의 삽입, 삭제시 VDGrid를 효율적으로 유지하는 방법에 대한 추가적인 연구가 필요하다.

참고문헌

〔1〕 Roger Weber, Hans-J. Schek, and Stephen Blott(1998), "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," Proc. of the 24th VLDB Conference, pp.194-205

〔2〕 Stefan Berchtold, Christian Bohm, Hans-Peter Kriegel (1997), "A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space," Proc. of the 16th PODS Conference, pp.78-86

〔3〕 S. Berchtold, B. Ertl, D. A. Keim, H. P. Kriegel, T. Seidl(1998), "Fast Nearest Neighbor Search in High-dimensional Space," Proc. of the 14th ICDE Conference, pp.209-218

〔4〕 M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf(1997), "Computational Geometry - Algorithms and Applications, " Springer, pp.145-161

〔5〕 V. Gaede and O. G nther(1998), "Multidimensional Access Methods," ACM Computing Surveys, Vol. 30, No. 2(June), pp.170-231

〔6〕 Guttman, A.(1984), "R-trees: A dynamic index structure for spatial searching", Proc. of ACM SIGMOD, pp.47-57

〔7〕 F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, Z. Protopapas(1996), "Fast Nearest Neighbor Search in Medical Image Databases," Proc. of the 22th VLDB Conference, pp.215-226

〔8〕 김범수, 송병호, 이석호(1998), "내용 기반 이미지 검색을 위한 점진적 여과 알고리즘", 한국정보과학회 논문지(B), 제 25권 제 1호, pp.64-71

〔9〕 T. Seidl, H. P. Kriegel(1998), "Optimal Multi-Step k-Nearest Neighbor Search," Proc. of SIGMOD, pp.154-165

〔10〕 Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger(1990), "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," Proc. of ACM SIGMOD, pp.322-331

〔11〕 James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes(1990), "Computer Graphics - Principles and Practice," Addision-Wesley Publishing Company, 2nd Ed., pp.124-127

〔13〕 R*-tree source, http://www.research.att.com/~berchtol/my_software.html

〔14〕 Qhull Library, http://www.geom.umn.edu/locate/qhull

〔15〕 Multi Dimensional Data Sets, http://www.cs.du.edu/~leut/MultiDimData.html

저 자 소 개

권 동 섭

서울대학교 컴퓨터공학 학사, 석사, 박사

삼성전자 SW연구소 책임연구원

현재: 명지대학교 컴퓨터소프트웨어학과 조교수