Journal of Korea Game Society 2013 Jun; 13(3): 47-54
http://dx.doi.org/10.7583/JKGS.2013.13.3.47

게임 프로그래밍

게임 환경에 적합한 연속적인 k-개의 이웃 객체 찾기 알고리즘 비교 분석

이재문
한성대학교 멀티미디어공학과
jmlee@hansung.ac.kr

Comparison of Algorithms to find Continuous k-nearest Neighbors to be Appropriate under Gaming Environments

Jae Moon Lee
Dept. of Multimedia Engineering, Hansung University

요 약

대부분의 연속된 k-개의 이웃 찾기 알고리즘은 차량, 핸드폰 등 이동하는 객체에 대하여 주기적으로 모니터링을 하는 위치 기반 서비스에서 연구되어 왔다. 이러한 연구들은 쿼리 포인트가 이동 객체에 비하여 매우 적을 뿐만 아니라 쿼리 포인트가 움직이지 않고 고정된 환경을 가정한다. 게임 환경에서 k-개의 이웃을 찾아야 하는 경우는 무리 짓기, 군중 시뮬레이션 및 로봇과 같이 이동 객체가 주변의 이웃 객체를 인식하여 다음 이동을 계산하여야 할 때이다. 따라서 모든 이동 객체가 쿼리 포인트가 되고, 그 결과 이동 객체와 쿼리 포인트의 수가 동일하며, 쿼리 포인트도 움직이게 된다. 본 논문에서는 이러한 게임 환경에서 기존의 위치기반 서비스에서 연구된 k-개의 이웃 찾기 알고리즘들을 적용하여 어떤 알고리즘이 어떤 조건에서 적합한지에 대한 성능을 분석한다.

ABSTRACT

In general, algorithms to find continuous k-nearest neighbors has been researched on the location based services monitoring periodically the moving objects such as vehicles and mobile phone. Those researches assume the environment that the number of query points is much less than that of moving objects and the query points are not moved but fixed. In gaming environments, cases to find k-nearest neighbors are when computing the next movement considering the neighbors such as flocking, crowd and robot simulations. Thus, every moving object becomes a query point so that the number of query point is same to that of moving objects and the query points are also moving. In this paper, we analyze the performance of the existing algorithms focused on location based services how they operate under the gaming environments.

Received: May 21, 2013 Accepted: Jun. 13, 2013
Corresponding Author: Jae Moon Lee(Hansung University)
E-mail: jmlee@hansung.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

주어진 정점에서 k-개의 가장 가까운 이웃을 찾는 문제(kNN: k-nearest neighbor)는 공간 데이터베이스 중심으로 연구되어 왔다[1]. 대부분의 연구는 R-트리 같은 공간 액세스 방법으로 데이터 객체들을 색인화 하고, 탐색 공간을 줄이기 위하여 휴리스틱을 사용하였다. 객체가 끊임없이 이동하는 상황 하에 kNN을 찾는 다양한 방법(CkNN: continuous k-nearest neighbor)도 연구 되어왔다[4,5,6,9]. 이러한 방법들은 주로 k-개의 가장 가까운 이웃을 찾기 위한 적당한 정보를 제공하는데 시간적, 공간적인 제약을 받는다.

위치 기반 서비스(LBS: Location Based Service)[7,11]를 모티브로 [4,5,6]에서는 CkNN에 대한 효율적인 방법을 제안 하였다. 움직이는 객체에서 CkNN을 찾기 위해 기존의 모든 알고리즘은 그리드 인덱스 구조를 사용했다. [4]에서는 움직이는 객체에 대하여, 객체 인덱스와 질의 인덱스를 사용하여 CkNN 모니터링 방법을 제안하였다. 특히, 객체 인덱스에 대해 오버하울(overhaul) 방법과 증가(Incremental) 방법을 제안하였다. [4]에서는 움직이는 객체가 균일하게 분포되어져 있지 않을 때 전체적인 성능을 향상시키기 위한 객체 인덱스의 계층적인 버전도 제안하였다. [6]에서는 개념적 분할 방법(CPM: Conceptual Partitioning Monitoring)을 적용하여 보다 효율적으로 CkNN을 찾는 알고리즘을 제안하였다. CPM은 가상으로 각 쿼리 포인트로 부터 그들과 가까운 사각형 안의 셀들을 구성한다. 이 알고리즘은 가상 사각형들을 이용하여 최소한의 셀을 방문한다.

CkNN에서 지금까지의 대부분의 연구는 움직이는 객체에 대하여 쿼리 포인트가 고정되어 있는 환경에서 효율적으로 동작하는 알고리즘이다[4,5,6]. 또한 쿼리 포인트의 수도 객체 수에 비하여 매우 작은 수로 하였다[4,5,6]. 이것은 LBS 지향적 서비스에 초점을 맞추어 연구되었기 때문이다. 그러나 본 연구에서와 같이 게임에서의 무리 짓기, 군중 시뮬레이션이나 로봇과 같이 움직이는 객체가 하나의 에이전트이고 에이전트가 주변 상황을 판단하기 위하여 kNN이 필요한 경우에는 모든 에이전트들에 대하여 매 일정한 시간 간격으로 kNN을 구하여야 한다[2,3,8,10]. 즉, 이러한 환경은 각각의 객체가 쿼리 포인트가 된다. 따라서 객체의 수와 쿼리 포인트의 수가 동일하며, 객체와 쿼리 포인트는 동일하게 지속적으로 움직이게 된다. 본 논문은 이러한 게임 환경에서 기존의 알고리즘들이 어떻게 동작하는지 성능 분석을 하는 것이다.

2장에서는 기존의 CkNN 알고리즘들을 소개하였고, 3장에서는 게임 환경에서 알고리즘들의 성능을 분석한다. 마지막으로 4장에서는 결론을 논한다.

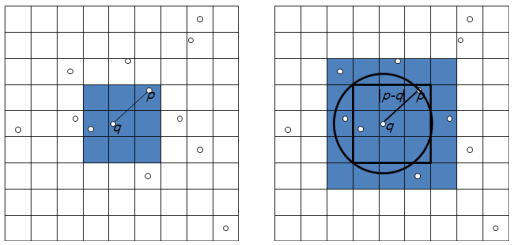
2. 기존 연구

2.1 YPK-CkNN

움직이는 객체에서 CkNN을 찾기 위해서 기존의 모든 알고리즘은 그리드 인덱스 구조를 사용한다. 그리드를 사용하는 방법은 매우 단순하다. 먼저 제어하고자 하는 공간을 $\delta \times \delta$ 그리드로 분할한다. 다음 모든 움직이는 객체들을 그들의 위치를 포함하는 셀에 저장한다. 객체가 지속적으로 움직여 다른 셀의 위치로 이동하게 되면 이전에 저장되었던 셀에서는 삭제가 일어나고 새로운 셀에 저장되게 된다. 이렇게 하면 가까운 이웃들은 쿼리 포인트가 저장된 셀내에 있거나 인접한 셀에 있기 때문에 kNN을 찾기 위하여 전체 객체들을 모두 평가하는 것이 아니라 인접 셀에 있는 객체들만 평가하면 된다.

[4]에서 제안한 YPK-CkNN 방법은 그리드를 사용하는 가장 기본적인 방법이다. [Fig. 1]을 고려하여 보자. 여기서 k는 2라고 가정한다. [Fig. 1]의 왼쪽과 같이 쿼리 포인트(q)를 포함하는 셀에서 시작하여 순차적으로 인접 셀을 탐색하되 2개의 객체가 포함될 때 까지 인접 셀들을 확장하여 탐색한다. 그런 후 [Fig. 1]에서 q에서 가장 먼 거리에

있는 객체(p)를 찾고, 그 객체까지의 거리($|p-q|$)를 구한다. 그런 후 q를 중심으로 $2|p-q|$ 크기의 정사각형과 겹치는 모든 셀에 저장된 객체들을 평가하여 k-개의 가장 가까운 이웃들을 찾는다.



[Fig. 1] Grid method in YPK-CkNN

overhaul(inputs: *q*, *G*, *k* output: kNN)
1. By using the grid *G*, find the rectangle which centers the cell including *q* and contains at least *k* objects. 2. Find the object *p* which is in the above rectangle and is farthest from *q*. 3. By using the grid *G*, find all the object in the circle with the radius $\|p-q\|$ centered at q. 4. Find and return the *k*-nearest neighbours from the objects found in "3".

[Fig. 2] YPK-Overhaul Algorithm

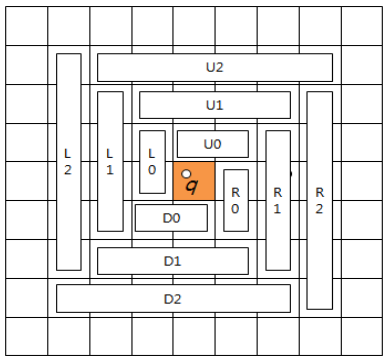
[Fig. 2]는 [4]에서 제안한 오버하울 알고리즘이다. 이 방법의 장점은 쿼리 포인트와 인접한 셀에 저장된 객체들만 평가하면서 k-개의 가장 가까운 이웃을 찾을 수 있다는 것이다.

[4]에서는 오버하울 방법의 성능 개선을 위하여 추가적인 방법을 제안 하였는데 시간 t-1에서 구해진 kNN을 시간 t에서 이들을 이용하여 새로운 kNN을 찾자는 것이다. overhaul에서 넓이가 2|p-q|인 정사각형이 중요한 것이 아니라 쿼리 포인트 q를 중심으로 반경 |p-q|안에 적어도 k개 이상의 객체가 존재한다는 것이 중요하다. 따라서 임의의 반경내에 k개 이상의 객체가 존재한다면 그 반경을 사용하여 오버하울 방법의 3, 4를 실행하여도 된다는 것이다. 이러한 사실에 착안하여 [4]에서는 t-1에서 구한 kNN을 별도로 저장하여 시간 t까지 유지한다. 그런 후 t에서 t-1에서 구한 kNN을 시험하여 가장 먼 거리에 있는 에이전트를 구하고 그 거리를 이용하여 오버하울 방법의 3, 4를 실행하는 것이다. [4]에서는 이를 증가 방법 명명하였다. 다음은 이에 대한 알고리즘이다.

incremental(inputs: *q*, *G*, *k*, kNN(*t*-1) output: kNN)
1. Compute $\|kNN(t-1)-q\|$. 2. By using the grid *G*, find all the object in the circle with the radius $\|kNN(t-1)-q\|$ centered at q. 3. Find and return the *k*-nearest neighbours from the objects found in "2".

[Fig. 3] YPK-Incremental Algorithm

상기 알고리즘에서 kNN(t-1)는 시간 t-1에서 구해진 q의 kNN이며, |kNN(t-1)-q|는 시간 t에서 kNN(t-1) 모두를 포함하는 원의 최소 반경이다. 증가 방법은 오버하울 방법에 비하여 간단하다. 또한 객체들의 이동 속도가 느린 경우 매우 좋은 성능을 나타낸다[4,6]. 그러나 객체들의 이동 속도가 큰 경우 t-1에서 찾아진 kNN이 멀리 이동할 가능성이 높아지고 따라서 |kNN(t-1)-q|의 값이 커질 가능성이 매우 높다. 이러한 경우 증가 방법에서 평가하여야 하는 셀 수와 객체의 수가 많아짐으로 성능이 나빠진다. [4]에서도 실험적으로 이러한 사실을 보였다.



[Fig. 4] DIR in CPM-CkNN

2.2 CPM-CkNN

[6]에서 제안한 CPM-CkNN 방법은 YPK-CkNN 방법에서 불필요한 셀의 액세스를 줄이기 위하여 CPM-CkNN 방법을 제안하였다. CPM-CkNN 방법에서는 [Fig. 4]와 같은 여러 셀을 포함하는 DIR(direction)을 정의하였다. [Fig. 4]에서 U0, U1, ..., L2 등이 DIR이다. DIR은 레벨을 가지는데 그림에서와 같이 쿼리 포인트로부터 가까운 DIR로부터 순차적으로 0에서부터 시작하여 1씩 증가하면서 정해진다. CMP-CkNN 방법에서 모든 셀은 쿼리 포인트와의 거리를 가지고 있는데 그 거리는 쿼리 포인트와 셀에서 임의의 지점 가운데 가장 가까운 거리로 정의한다. 따라서 셀에 포함된 어떠한 객체도 이 거리보다 가까울 수 없다. DIR도 쿼리 포인트의 거리를 갖는데 이 거리는 DIR에 포함된 셀들 중 가장 가까운 거리가 DIR의 거리가 된다. 이러한 거리를 mindist<DIRi, q>로 표현하였다.

[6]에서 제안한 알고리즘은 기본적으로 셀과 DIR에 대하여 힙(heap) 자료 구조를 사용한다. 즉, 검색 대상이 될 가능성이 있는 셀이나 DIR을 그들의 거리에 따라 힙에 저장하고 검색이 필요한 경우 힙으로부터 삭제하여 필요한 객체들을 평가한다. [Fig. 5]는 [6]에서 제안한 NN_Computation 알고리즘이다.

NN_Computation(inputs: *q*, *G*, *k* output: kNN)

1. Insert the cell <Cq, 0> tp MinHeap.
2. For four directions based on cell Cq, insert <*DIR0*, *mindist(DIR0, q)*> to *MinHeap*.
3. Remove the root from *MinHeap*.
3.1 If the root is a cell, store each object in the cell to *bestNN* and compute *best_dist* if necessary.
3.2 Otherwise, insert all the cells in *DIR(i)* and *DIR(i+1)* to *MinHeap*.
3.3 If the distance of the root in *MinHeap* is greater than *best_dist*, return. Otherwise go to 3.

[Fig. 5] CPM-NN_computation Algorithm

상기 알고리즘에서 bestNN은 k-개의 가장 가까운 이웃을 저장하는 배열이며, best_dist는 초기값은 무한대이고 3.1에서 k개의 객체가 bestNN에 저장된 경우 bestNN에 있는 모든 객체를 포함하는 최소 반경이다. 상기 알고리즘은 맨 처음 시작할 때 사용하는 알고리즘이다. [6]에서는 [Fig. 5]의 3.1에서 사용된 셀들을 모든 쿼리 포인트 별로 <vist_list>를 유지하도록 하였다. 따라서 두 번째부터는 주어진 쿼리 포인트에 대하여 MinHeap을 이용하는 것이 아니라 <vist_list>를 이용하여 셀을 액세스한다. 이것은 쿼리 포인트가 고정된 경우 별도의 힙 자료구조를 사용하지 않기 때문에 성능을 개선한다. 또한 [6]에서는 각 쿼리 포인트별로 영향력 지역(influence region)을 유지하는데, 이것은 영향력 지역으로 들어오는 객체와 나가는 객체를 조사하여 계산 비용을 최소화 한다. 그러나 [6]에서는 움직이는 쿼리 포인트에 대해서는 <vist_list>나 영향력 지역을 모두 무시하고 NN_computation 알고리즘만 사용하도록 하였다.

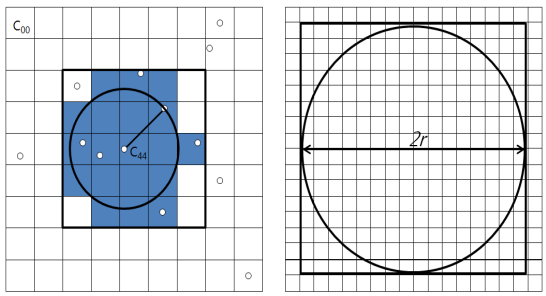
2.3 YPK-CkNN과 CPM-CkNN 차이

YPK-CkNN과 CPM-CkNN 알고리즘의 가장 큰 차이는 k-개의 가장 가까운 이웃을 찾기 위하여 액세스하는 셀의 차이이다. [Fig. 6]은 두 방법에 의하여 액세스된 셀들을 표시하고 있다. YPK-CkNN 방법에서는 큰 사각형에 포함된 셀 25개를 액세스해야 하지만, CPM-CkNN 방법에서는 그림에서와 같이 배경색이 있는 19개의 셀만 액세스하면 된다.

이러한 현상은 그리드가 매우 좁게 되어 셀의 크기가 매우 작고 그리고 사각형이 매우 큰 경우 두 방법의 차이를 일반화 할 수 있다. 예를 들어 [Fig. 6](b)에서와 같이 YPK-CkNN에 의하여 액세스되는 셀들이 사각형 $(2r)\times(2r)$내에 존재한다고 하면, CPM-CkNN에 의하여 액세스되는 셀들은 사각형 $(2r)\times(2r)$내에 존재하는 가장 큰 원내에 존재하는 셀들이 된다. 따라서 YPK-CkNN와 CPM-CkNN의 셀 액세스 비율은 다음과 같다.

$$\frac{number\ of\ cells\ in\ CPM}{number\ of\ cells\ in\ YPK}\times 100\%=\frac{\pi r^2}{4r^2}\times 100\%=78.5\% \quad \text{(eq. 1)}$$

(eq. 1)은 CPM-CkNN은 YPK-CkNN에 비하여 최대 21.5%의 셀 액세스를 줄일 수 있다는 것을 의미한다. 이러한 경우는 매우 이상적인 경우이다. 예를 들어, CPM-CkNN에서 [Fig. 6](b)의 원에 걸치는 사각형에 포함된 모든 객체들도 액세스되어야 하므로 상기 계산에서 걸치는 사각형의 모든 면적이 (eq. 1)의 분자에 포함되어야 한다. 따라서 이러한 경우 (eq. 1)의 비율은 보다 높아질 것이다. 특히 [Fig. 6](b)에서 $2r$이 매우 작은 경우, 이러한 걸치는 사각형까지 고려하면 (eq. 1)의 거의 100%가 될 것이다. 이 경우 CPM-CkNN의 성능 개선 효과는 더욱 줄어든다. 나중에 실험에서도 보이겠지만 이러한 현상은 k값이 작거나, 객체의 수가 매우 많은 경우 r이 매우 작게 되며, 이 경우 YPK-CkNN이 더 좋은 성능을 보인다.



[Fig. 6] Cells to be accessed in YPK-CkNN and CPM-CkNN

3. 게임 환경에서 성능분석

본 논문에서는 앞장에서 소개된 세 가지 알고리즘을 게임 환경, 즉, 모든 움직이는 객체가 쿼리 포인트가 되는 환경에서 실험적으로 성능을 분석한다. 실험을 위하여 세 가지 알고리즘을 C++를 사용하여 구현하였다. 실험 환경은 인텔 i7 CPU와 8GB 메모리를 가진 컴퓨터에서 윈도우즈 7 운영체제상에서 수행되었다. 실험에 적용되는 파라미터는 다음 표와 같다.

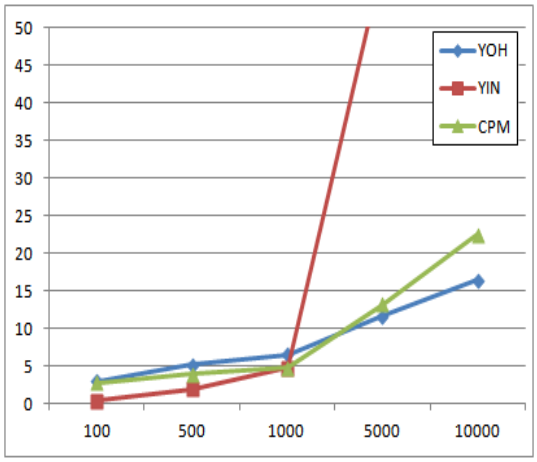
[Table 1] Parameters used in experiments

Parameter	Default	Range
Number of Objects (n)	1000	100, 500, 1000, 5000, 10000
Number of Query Points	same to number of objects	
Number of Neighbors(k)	16	2, 4, 8, 16, 32, 64
Velocity of Objects(v)	normal	slow normal, fast

실험에 사용된 공간의 크기는 1000x1000 픽셀의 크기로 하였다. [Table 1]에서 객체의 속도는 느림, 보통, 빠름으로 하였는데 이 속도는 [6]에서와 같이 각각 프레임당 4, 20 및 100 픽셀의 속도로 움직이는 것으로 정하였다. 실험 데이터의 생성은 각 객체에 대하여 2차원 공간에 대하여 난수를 생성하여 초기 위치를 정하였으며, 이후 일정한 시간 간격으로 정의된 속도로 임의의 방향으로 이동하게 하였다. 성능 분석에서 각 알고리즘의 이름은 오버하울 방법, 증가 방법 및 CPM의 실행 시간을 각각 YOH, YIN, CPM으로 명명하였다. 모든 실험에서 각 알고리즘은 100번 반복적으로 수행되었으며, 측정 시간은 초이다. 예를 들어 YOH가 76이라는 의미는 특정 환경에서 YOH를 반복하여 100실행한 시간의 합이고 단위는 초 단위이다.

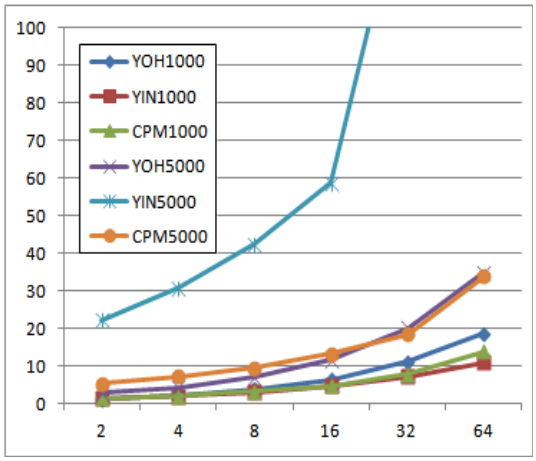
첫 번째 실험은 객체 수의 변화에 따른 성능 비교이다. 즉 [Table 1]에서와 같은 객체의 수의 변화에 따른 성능을 측정한다. 여기서 이웃 에이전트의 수는 16으로 고정하였으며, 객체의 스피트는 보통으로 고정하였다. 실험의 결과는 [Fig. 7]에서 보인다. n값이 작은 경우에는 증가 방법이 가장 좋은 성능을 나타낸다. 그러나 객체의 수가 증가함에 따라 성능이 급격히 저하됨을 볼 수 있다. 이것은 객

체의 속도가 느리지 않을 뿐만 아니라 객체가 많아지면 단위 면적당 객체의 수가 많아지고 따라서 [Fig. 3]의 2번 항에서 찾아지는 객체의 수가 k보다 훨씬 많아지기 때문이다. 오버하울과 CPM의 성능은 객체의 수가 약 2500에서 서로 다른 성능을 나타낸다. 그림에서 볼 수 있듯이 객체의 수가 2500보다 적은 경우에는 CPM이 좋은 성능을 나타내고 있으나, 그 이상의 경우에는 오히려 오버하울이 더 좋은 성능을 나타낸다. 이것은 (eq. 1)이 적용되지 않은 경우로 판단된다. 즉, 고정된 공간과 고정된 k 값으로 된 환경에서 객체의 수가 증가하면 [Fig. 6]에서 보이는 사각형과 원의 크기가 작아지고 원 주변에 걸치는 사각형 때문에 (eq. 1)의 값이 거의 100%에 육박하기 때문이다. 그럼에도 불구하고 CPM의 성능이 유난히 저하되는 것은 (eq. 1)의 이익은 없고 heap-자료구조의 오버헤드는 그대로 남아 있기 때문이다.



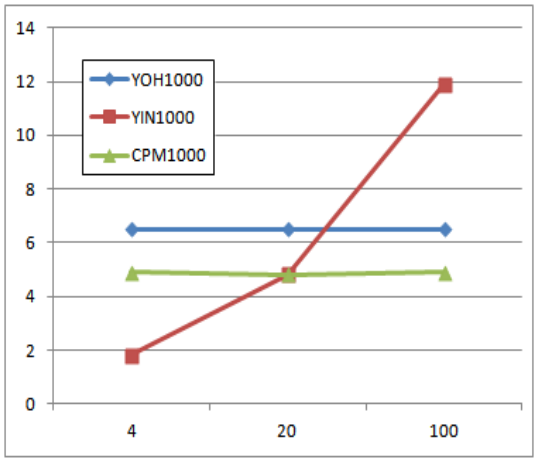
[Fig. 7] Performance comparison according to n with k=16 and v=보통

두 번째 실험은 이웃 객체 수의 변화에 따른 성능 비교이다. 이 실험에서 객체의 수는 1000과 5000으로 고정하였으며, 객체의 스피드는 보통으로 고정하였다. 실험의 결과는 [Fig. 8]에서 보인다. 객체의 수가 1000인 경우 k 값이 작은 경우에는 세 방법의 성능이 유사하다. k 값이 증가함에 따라 증가 방법이 가장 좋은 성능을 나타내며, 다음으로 CPM에 좋은 성능을 나타낸다. 객체의 수가 5000인 경우에는 증가 방법이 가장 나쁜 성능을 나타낸다. 이것은 앞의 분석에서 설명하였듯이 단위 면적당 객체의 수가 많아지면 따라서 [Fig. 3]의 2번 항에서 찾아지는 객체의 수가 k보다 훨씬 많아지기 때문이다. 객체의 수가 5000인 경우 k가 증가함에 따라 CPM의 성능이 YOH에 비하여 개선되고 있음을 볼 수 있는데 그 이유는 k가 크다는 의미는 [Fig. 6]에서 사각형과 원이 커지므로 (eq. 1)의 값이 거의 78.5%가 되기 때문이다.



[Fig. 8] Performance comparison according to k with n=1000, 5000 and v=보통

마지막 실험은 속도의 변화에 따른 성능 비교이다. 객체의 속도에 따른 성능 개선 비교이다. 속도는 [Table 1]에서와 같이 느림, 보통 빠름 세 가지로 변화를 주었다. 결과는 [Fig. 9]에 나타나 있다. 알고리즘의 특성상 알 수 있듯이 YOH와 CPM은 객체의 속도에 관계가 없다. 따라서 [Fig. 9]에서 볼 수 있듯이 YOH와 CPM 은 거의 일정한 값을 갖는다. 그런 반면 YIN은 객체의 속도가 증가함에 따라 급격히 성능 저하가 나타나는데 이것은 [Fig. 3]에서 |kNN(t-1)-q|의 값이 커지기 때문이다.



[Fig. 9] Performance comparison according to v with k=16 and n=1000

상기 세 가지 실험의 결과를 분석하면 객체 수가 적은 경우에는 증가 방법(YIN)이 성능이 우수하다. 반면 객체의 수가 아주 많아지는 경우는 오버하울 방법(YOH)이 우수하다. 이것은 [6]에서의 결과와 부분적으로 상반된다. 그것의 이유는 쿼리 포인트 수의 차이라고 판단한다. 게임 환경에서는 움직이는 객체를 무한히 증가 시킬 수는 없다. 또한 게임 환경에서는 객체들의 이동에 대한 위치 계산도 많은 시간을 소요하지만 각 객체를 렌더링하는 시간도 많기 때문에 일반적으로 움직이는 객체를 1000개 이상 적용하는 경우는 많지 않다. 따라서 게임 환경에서는 객체의 속도가 비교적 느린 경우 YPK의 증가 방법이 적합하고, 그렇지 않은 경우 CPM 방법이 적합한 것으로 판단된다.

4. 결 론

k-개의 가장 가까운 이웃 찾기 문제는 다양한 환경에서 연구되고 있다. 대표적인 환경이 위치 기반 서비스이다. 이러한 서비스를 중심으로 많은 연구가 진행되어 왔다. 그러나 위치 기반 서비스 환경에서는 비록 객체는 이동하는 환경이나 쿼리 포인트는 대부분 고정된 환경이다. 따라서 움직이는 객체가 쿼리 포인트인 무리 짓기, 군중 시뮬레이션 및 로봇 분야등 게임 환경은 위치 기반 서비스와 다른 환경이다.

본 논문에서는 기존의 위치기반 서비스 환경에 적합하도록 연구된 알고리즘들을 게임 환경에 적용하여 어떤 알고리즘이 어떤 조건에서 가장 적합한지를 분석하여 보았다. 실험적 분석 결과로는 객체의 속도가 비교적 느린 경우 YPK의 증가 방법이 적합하고, 그렇지 않은 경우 CPM 방법이 적합한 것으로 판단된다.

ACKNOWLEDGMENT

This research was financially supported by Hansung University.

REFERENCES

[1] Guttman, A., “R-trees: a dynamic index structure for spatial searching”, ACM SIGMOD Rec., 14(2):47-57, 1984.
[2] Reynolds, C. W., “Flocks, Herds, and Schools: A Distributed Behavioral Model”, SIGGRAPH, 21(4), pp. 25-34, 1987.
[3] Mat Buckland, “Programming Game AI by Example”, ISBN 1556220782, Wordware Publications, 2005.
[4] Yu, X., Pu, K.Q., Koudas, N., “Monitoring k-Nearest Neighbor Queries over Moving Objects”, 21st Int. Conf. on Data Engineering, p.631-642, 2005.
[5] Xiong, X., Mokbel, M.F., Aref, W.G., “SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-Temporal Databases. 21st Int. Conf. on Data Engineering”, p.643-654, 2005.
[6] Mouratidis, K., Hadjieleftheriou, M., Papadias, D., “Conceptual Partitioning: an Efficient Method for Continuous Nearest Neighbor Monitoring”, Proc. ACM SIGMOD Int. Conf.

on Management of Data, p.634-645, 2005.
[7] Junglas, I.A., Watson, R.T., "Location-based services", Commun. ACM, 51(3):65-69, 2008.
[8] Jae Moon Lee, "An Efficient Algorithm to Find k-Nearest Neighbors in Flocking Behavior", Information Processing Letters, 110, pp. 576-579, 2010.
[9] Jun Min Park, Jae Moon Lee, "Performance Analysis for finding continuous k-nearest neighbors of moving objects under game environments", Proc. of Korea Game Society, Spring, 2013.
[10] Jae Moon Lee, Young Mee Kwon, "A Model of Pursuing Energy of Predator in Single Predator-Prey Environment", Journal of Korea Game Society 2013 Feb; 13(1): 41-48.
[11] Sangchul Kim and Zhong Yong Che, "A Method for Tennis Swing Recognition Using Accelerator Sensors on a Smartphone", Journal of Korea Game Society 2013 Apr; 13(2): 29-38.

이 재 문(Lee, Jae Moon)

1986년 한양대학교 전자공학과(학사)
1988년 한국과학기술원 전기및전자공학과(석사)
1992년 한국과학기술원 전기및전자공학과(박사)
1994년-현재 한성대학교 멀티미디어공학과 교수

관심분야 : 기계학습, 게임프로그래밍, 감성컴퓨팅