효율적인 Nearest Surrounder 질의 처리 방법

최정임°, 정재화°, 김종완°°, 임석진°°, 강상원°°, 정순영°
고려대학교 컴퓨터교육학과°, 고려대학교 컴퓨터학과°°
{jungimism, bigbearian, jsy}@comedu.korea.ac.kr, wany@korea.ac.kr, {seokjin, swkang}@disys.korea.ac.kr

Efficient Nearest Surrounder Queries Processing

Jungim Choi°, Jaehwa Chung°, Jongwan Kim°°, SeokJin Im°°, Sang-Won Kang°°, Soonyoung Jung°
Department of Computer Science of Education°, Department of Computer Science and Engineering°° Korea University, Anam-Dong, Sungbuk-Ku, 136-713, Seoul, Korea

요 약

지금까지 질의 점을 중심으로 최근접 객체(Nearest Neighbor : NN)를 찾는 다양한 연구가 진행되었다. 하지만 이 방법은 질의 점과 객체의 거리만을 고려하기 때문에 질의 점을 둘러싸고 있는 객체들을 찾을 수 없다는 문제점이 있다. 이것을 해결하기 위해서 제안 된 것이 최근접 주변객체(Nearest Surrounder : NS) 질의 처리 이다. 최근접 주변 객체는 질의 점을 둘러싸고 있으면서 가장 가까운 객체들을 찾는 것에 대한 연구이다. 기존의 NS를 찾는 방법은 객체 인덱싱을 위하여 R-tree를 사용하며, 질의 점과 최소경계 사각형(minimum bounding rectangle : MBR)이 이루는 각의 범위를 계산한다. 계산 수행 결과 각 MBR들이 이루는 각의 범위가 겹치는 부분이 발생하면 해당 각 범위 내에서 질의 점으로부터 최소거리에 있는 MBR을 선택해야 하므로 범위별 질의 점과 MBR들의 최대 최소 거리를 구해야 한다. 이러한 범위별 계산 과정은 계산 비용을 높이는 단점이 있다. 따라서 본 논문에서는 NS를 필요로 하는 영역에서 각 범위별 겹쳐지는 MBR들의 꼭지점 좌표만을 비교한다. 이것은 기존 연구에서 계산 비용을 높이는 공통 각 계산 절차를 개선하고, 최대 최소 거리 계산 수행은 생략하여 NS를 찾는다. 제안 기법을 위해 논문에서 사용하는 각 알고리즘은 이전 연구보다 나은 계산비용 절감 효과를 가져 올 수 있다.

1. 서 론

수 십 년 동안 최근접 객체(Nearest Neighbor : NN)[1]를 찾는 다양한 연구가 진행되었다. 그러나 NN은 질의 점과 객체의 거리만을 기준으로 결과 객체를 찾기 때문에 적용할 수 없는 상황이 있다. 그 첫 번째 예로 전쟁터를 들 수 있다. 전쟁터에서는 직선 거리상 가장 가까이에 있는 적을 찾는 것 보다 자신을 둘러싸고 있는 적을 찾는 것이 더 중요하다. 두 번째 예는 로봇 축구에 적용될 수 있다. 현재 공을 가지고 있는 로봇이 공을 패스 하려고 할 때 가장 가까이에 있는 같은 팀 로봇이 상대 팀의 로봇에 의해 가려져 있다면 그 로봇에게는 공을 패스하지 말아야 한다. 즉 공을 패스 했을 때 공을 직접적으로 받을 수 있는 같은 팀원을 찾아야 한다. 이 경우 역시 직선거리상 가까운 객체보다 주변에 있는 객체를 찾는 것이 더 중요하다. 마지막으로 무선통신을 하려는 상황에서도 NN은 적용하기 어렵다. 만약 무선 통신의 수단이 장애물을 통과할 수 없는 전파, 예를 들어 적외선과 같은 것이라면 단순히 NN을 통해서는 직접 통신을 할 수 없다. 따라서 질의 점을 중심으로 어느 방향, 즉 어느 polar angle 범위에 가장 가까운 객체가 존재하는지를 고려한 최근접 주변 객체(Nearest Surrounder : NS)에 대한 필요성이 대두되었다[2].

요약하자면, 주어진 객체 집합에서 NN은 질의 점과 가장 가까운 객체를 결과로 반환한다. 반면 NS는 질의 점의 어느 방향, 즉 어느 polar angle 범위에 어떤 객체가 있는지를 결과로 반환한다. 따라서 NS의 결과는 <객체식별자, [시작각, 끝각)>의 집합으로 구성된다.

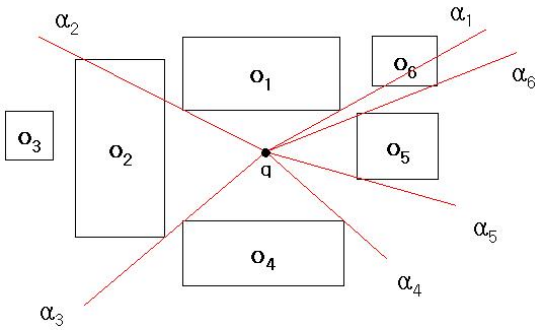


그림 1 NS의 개념

객체를 R-tree[3]를 사용하여 인덱스 하고, 객체는 움직이지 않는다고 가정한다. q는 질의 점, o_i는 R-tree에 인덱스 된 객체, α_i는 각도를 나타낸다고 하자. 그러면 그림 1에서 q의 NS 결과는 {<o_1, [α_1, α_2)>, <o_2, [α_2, α_3)>, <o_4, [α_3, α_4)>, <o_5, [α_5, α_6)>, <o_6, [α_6, α_1)>} 가 된다[1].

이러한 NS 결과를 구하기 위해 제안된 기존의 알고리즘은 다음의 단계를 거친다. 우선 질의 점과 객체가 이루는 각 범위를 구한다. 여기서 구한 각 범위가 겹치는 객체를 찾아서 공통되는 각 범위를 구한다. 이 범위 내에서 질의 점과 객체의 최대 거리와 최소 거리를 계산한다. 그리고 그 거리 값을 기반으로 NS를 결정한다. 이 방법은 많은 연산을 요구하기 때문에 계산 비용이 높다. 그리고 겹치는 각 범위를 발견하면 해당 범위에 대해서만 NS가 누구인지 판단하는 절차를 거치기 때문에 {<o_1, [α_1, α_2)>, <o_1, [α_2, α_3)>}와 같은 결과가 나올 수 있다. 하지만 이 결과는 {<o_1, [α_1, α_3)>}로 수정해야 한다. 즉, 기존의 NS 알고리즘은 계산 비용이 높고 결과를 수정해 주어야 하는 상황이 발생할 수 있다. 이 문제를 해결하는 방법을 본 논문에서 제안하고자 한다.

본 논문에서는 질의 점과 꼭지점이 이루는 각을 킷값으로 하여 우선순위 큐에 넣는다. 그리고 큐에 있는 엔트리를 순차적으로 탐색하는 방법을 선택하였다. 그럼으로써 공통되는 각을 일일이 계산하지 않아도 된다. 따라서 계산 비용이 줄어든다. 또 공통되는 각 범위에 있는 두 객체 중 NS를 선택할 때 거리를 계산하는 대신 객체의 꼭지점의 좌표값의 크기를 비교한다. 그럼으로써 누가 더 질의 점에 가까이 있는지를 찾아낸다. 이 방법 역시 복잡한 계산을 덜어주어 알고리즘의 성능을 향상시킬 수 있다.

2장에서는 기존에 진행되었던 연구들에 대하여 설명하고 3장에서는 본 논문이 제안하는 기법과 알고리즘에 대하여 설명하겠다. 마지막으로 4장에서 결론과 향후 과제를 기술하겠다.

2. 기존 연구

NN Search[1]는 질의 점에서 가장 가까운 객체를 찾는 알고리즘이다. 일반적으로 기존 연구에서 NN Search는 객체를 간략하게 표현하기 위하여 MBR로 근사하고 R-Tree로 객체를 인덱싱한 환경에서 최근접 객체를 검색한다. 그렇게 하기 위해서 MINDIST(Minimum Distance)와 MINMAXDIST(Minimax Distance) 라는 개념을 도입한다. MINDIST는 질의점과 가장 가까운 MBR의 경계선이 질의 점과 떨어진 거리를 의미한다. 이 MINDIST 값을 통해서 질의 점과 가장 가까운 객체를 찾을 수 있다. 그리고 방문하는 객체의 수를 최소화 하기 위해서 MINMAXDIST를 사용한다. 만약 MBR 안에 큰 사장 공간(dead space)이 있다면 NN은 MBR의 MINDIST보다 더 멀리 있을 수 있다. 이 문제를 해결하기 위해서 MINMAXDIST를 사용한다. MINMAXDIST는 방문하지 않을 MBR들을 결정할 때 사용된다. 만약 어떤 MBR의 MINDIST가 어떤 상한선보다 높으면 방문하지 않는다. MINMAXDIST는 모든 객체와 질의 점이 이루는 최대 거리 값의 최소값이다. MINMAXDIST 값보다 작거나 같은 범위 내에서는 반드시 객체가 존재함을 보장한다.

하지만 이러한 기존의 NN Search 방법은 객체가 질의 점을 기준으로 어느 방향에 있는지에 대한 정보를 알 수 없다. 즉, 질의 점을 둘러싸고 있고, 각 방향에 대하여 가장 가까운 객체들을 찾는 방법을 제시하지 못한다. 이 문제를 해결하기 위해서 NS 알고리즘[2]이 제안되었다.

기존의 NS를 구하는 알고리즘[2]은 각도와 거리를 계산해야 한다. 우선 질의 점과 객체 간의 angular bound를 계산하는데 이것은 그림 2에서 표시된 것처럼 질의 점 q와 객체가 이루는 각의 범위를 말한다. 여기서 o_1의 angular bound는 [α_{start}, α_{end})이며 α_{start}, ≤ α_{end}와 같은 조건을 만족한다. 객체는 R-tree[3]를 기반으로 인덱스 되어 있는데, 만약 객체가 자식(child) 객체를 가지고 있는 경우 부모(parent) 객체는 자식 객체의 angular bound를 포함한다[2]. 그림 2에서 보면, o_1의 자식 객체인 o_{1-A}의 angular bound는 o_1의 angular bound에 포함된다. 즉, o_1의 angular bound, [α_{start}, α_{end})는 o_{1-A}의 angular bound인 [$\alpha_{start'}$, α_{end})를 포함하며 이 관계는 부모 · 자식 객체에서 항상 만족한다.

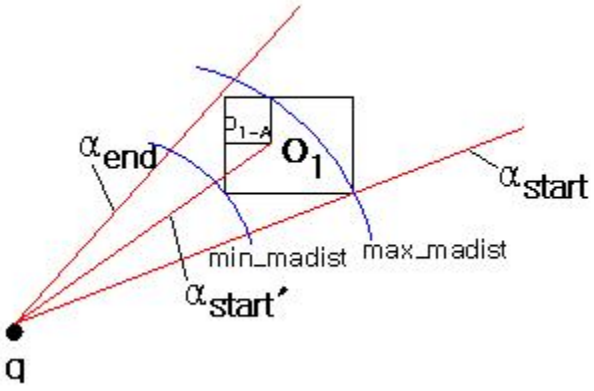


그림 2 angular bound와 min_madist, max_madist의 개념

객체의 minimum angular distance(min_madist)는 angular bound에서 질의 점과 객체 간 가장 가까운 거리를 말한다. maximum angular distance(max_madist)는 주어진 angular bound에서 질의 점 객체 간 가장 먼 거리를 말한다. 이 값은 R-tree[3] 인덱스 구조에서 다음을 만족한다[2].

· *parent->min_madist ≤ child->min_madist*
· *parent->max_madist ≥ child->max_madist*

두 객체의 angular bound가 겹칠 때, 공통되는 각 범위 내에서 어떤 객체를 NS로 선택할지 결정할 때 min_madist와 max_madist를 사용하게 된다.

그림 3에서 두 객체 o_1과 o_2의 angular bound는 [α_{start2}, α_{end1})에서 겹친다. 이 범위에서 'o_2의 min_madist ≤ o_1의 min_madist'를 만족하기 때문에 [α_{start2}, α_{end1}) 범위에서는 o_2가 NS가 된다.

max_madist는 방문하지 않을 객체를 선택할 때 사용된다. 그림 3에서 공통되는 각 범위 [α_{start3}, α_{end1})에서 'o_1의 max_madist ≤ o_3의 min_madist'를 만족하기 때문에 o_3은 NS가 아니다. 이러한 방법을 부모 객체에 적용 시키면 방문하지 않아도 되는 MBR을 결정할 수 있다[2].

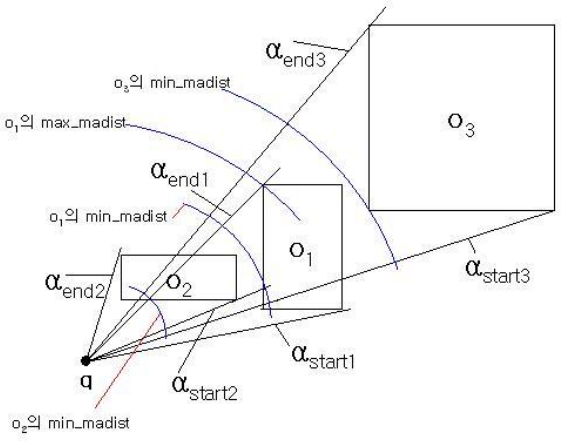


그림 3 min_madist, max_madist 값을 이용한 NS 결정

그런데 기존의 NS 알고리즘이 공통각을 구하는 방법이 간단하지 않다. NS 알고리즘은 우선순위 큐를 초기화한 후 공통 각 범위을 찾는다. 우선 R-tree[3]의 root를 우선순위 큐에 넣는다. 킷값은 angular bound의 α_{start}값이다. 우선순위 큐의 헤드가 단말 노드가 아니면 그 노드의 자식을 큐에 넣는다. 그런데 이러한 방법은 우선순위 큐의 제일 앞에 있는 객체를 시작으로, 큐에 있는 객체들이 공통 각 범위를 가지는지를 판단해야 한다. 그런데 우선순위 큐는 α_{start}값으로만 저장되어 있기 때문에 어떤 객체가 공통되는 각 범위를 가지고 있는지 쉽게 찾을 수 없다. 게다가 공통되는 각 범위를 가진 두 객체에 대하여 min_madist, max_madist를 구해야 한다. min_madist와 max_madist를 구하는 것은 많은 계산 비용을 요구한다.

그림 3의 o_1은 공통되는 각 범위가 [α_{start2}, α_{end1})와 [α_{start3}, α_{end1})가 있다. [α_{start2}, α_{end1})에 대해서는 min_maidst를 구해야 하고 [α_{start3}, α_{end1})에 대해서는 max_madist를 구해주어야 한다. 그런데 [α_{start2}, α_{end1})에서 min_madist를 정확하게 구하는 것은 간단하지 않다. 우선 질의 점과 o_2의 오른쪽 아래 좌표 값을 통해 구한 직선의 방정식과 o_1의 왼쪽 경계선이 만나는 교점을 구해야 한다. 그 교점과 질의 점과의 거리가 min_madist가 되는 것이다. 이것은 많은 계산비용을 요구한다.

이러한 문제를 해결하기 위해서 본 논문은 우선순위 큐를 정렬하는 방법을 바꾸어서 겹치는 각 범위를 쉽게 구하도록 했다. 이렇게 만든 우선순위 큐는 겹치는 각도가 있는 객체를 한 번에 찾아낼 수 있기 때문에 계산 비용이 절감된다. 또한 복잡한 min_madist, max_madist를 계산하는 대신 객체의 꼭지점의 좌표 값의 크기를 비교함으로써 NS를 찾는 방법을 제안하고자 한다.

3. 제안 기법

3.1 개선된 NS를 위한 우선순위 큐

NS를 찾는 것은 주변 객체의 경계선을 찾아내는 것으로 생각할 수 있다. 그리고 새로운 NS를 찾는 것은 그 객체의 꼭지점에서 시작된다.

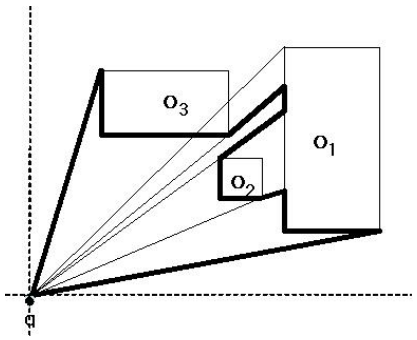
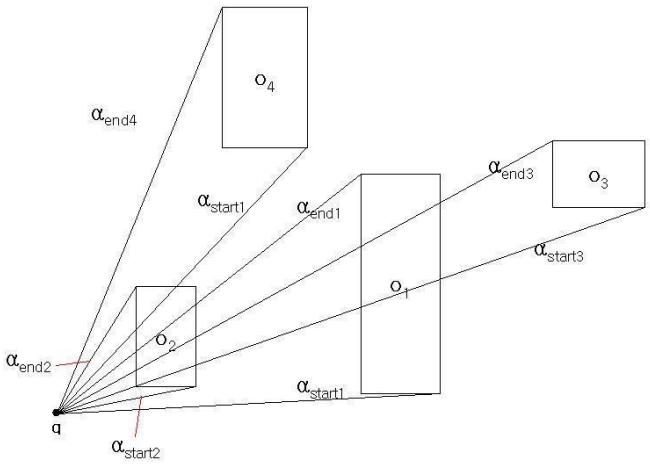


그림 4 새로운 NS가 발견되는 지점

그림 4를 보면 굵은 선으로 표시 된 것이 NS를 결정짓는 선이다. 그런데 이 선은 객체의 경계선과, 객체의

꼭지점과 질의 점을 이은 선으로 이루어져 있다. 즉 새로운 NS가 발견되는 곳은 객체의 꼭지점이라는 점에 착안하여 우선순위 큐에 각도에 대한 오름차순으로 정렬하되 한 객체에 대하여 두 개의 꼭지점을 고려하도록 한다.

앞서 설명한 새로운 우선순위 큐를 만드는 예제는 그림 5를 보면 알 수 있다.



(a) 질의 점과 객체의 분포도

	A	B	C	D	E	F	G	H
객체	o_1	o_2	o_3	o_3	o_1	o_4	o_2	o_4
각도	α_{start1}	α_{start2}	α_{start3}	α_{end3}	α_{end1}	α_{start4}	α_{end2}	α_{end4}

(b) 그림 (a)를 우선순위 큐에 저장한 결과

그림 5 객체 공간과 해당 공간을 우선순위 큐에 저장한 결과

우선순위 큐는 질의 점 q와 R-tree의 root를 인자로 받아 root의 모든 자식 객체에 대하여 α_{start}, α_{end}를 계산한다. 그리고 각각의 α_{start}, α_{end}를 킷값으로 우선순위 큐에 저장한다. 이것을 알고리즘으로 표현하면 다음과 같다.

Algorithm initial priority queue (q, root)
input : 질의점 q, R-tree의 root
output : root의 자식들과 q가 이루는 a_{start}와 a_{end} 값들로 정렬된 우선순위 큐 Q

```
1.    for (root의 모든 자식 객체에 대하여) {
2.      q와의 a_start와 a_end를 계산
3.      우선순위 큐 Q에 오름차순으로 삽입
4.    }
```

알고리즘 1 initial priority queue

3.2 공통되는 각을 가진 객체 찾기

그림 5(b)를 보면 공통되는 각 범위인, [α_{start2}, α_{end1}) 또는 [α_{start3}, α_{end3})를 가진 객체를 한 번에 찾을 수 있다. 첫 번째 열(A)의 객체가 o_1이므로 이것과 같은 객체 id를 가진 또 다른 열(E) 사이에 있는 모든 객체가 o_1과 공통되는 각을 가진 객체가 된다. 이 예에서는 o_2와 o_3이 발견된다. o_2를 포함하는 것은 열(B) 하나만 있고 o_3은 두 개의 열(C), (D)가 있다.

o_2처럼 하나의 열이 있다는 것은 그림 6의 (a), (b) 상황 중 하나를 의미하고 o_3처럼 두 개의 열이 있다는 것은 그림 6의 (c), (d) 상황을 의미한다.

o_2처럼 열이 한 개 있다는 것은 그림 6(a)처럼 B열에서부터 바로 다음 열인 C지점 까지 o_2가 NS임을 의미하거나 그림 6(b)처럼 현재 NS인 o_1의 α_{end1}이 발생하는 E열이 있는 지점까지 o_2는 NS가 될 수 없음을 의미한다.

o_3처럼 열이 두 개 있다는 것은 그림 6(c)처럼 o_3이 o_1에 의해 완전히 가려지거나 그림 6(d)처럼 o_3이 o_1의 일부를 가린다는 것을 의미한다. 만약 그림 6(c)로 판단되면 C열과 D열을 우선순위 큐에서 지워야 한다.

이것을 판단하는 것은 객체쌍(o_1,o_2 또는 o_1,o_3)의 꼭지점의 좌표 값의 대소 관계를 비교함으로써 구할 수 있다. 자세한 것은 3.3에서 설명한다.

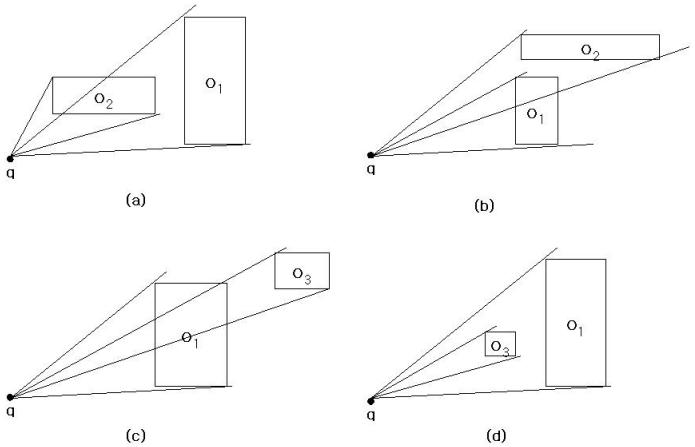


그림 6 두 객체의 각이 겹칠 때 예상 가능한 객체의 배열

지금까지 설명한 것을 알고리즘으로 표현하면 다음과 같다.

이 알고리즘 2는 우선순위 Q의 마지막 entry까지 수행함으로써 끝이 난다. 각도를 기준으로 정렬된 우선순위 큐를 사용한다. 3번째 줄에서 공통 각을 가진 객체를 찾고, 5번째 줄에서 해당 객체 쌍을 select NS(알고리즘 3)로 넘긴다. select NS의 결과 값을 통해서 두 객체 중에서 적합한 것을 반환한다. 반환된 결과를 토대로, 그림 6의 (a), (b), (c)에 해당된다고 여겨지면, 우선순위 큐에

서 entry를 삭제하고(8, 11, 15번째 줄) 다음 entry로 기준을 옮겨서 해당 알고리즘을 다시 수행한다. 만약 현재 검사하고자 하는 객체가 단말노드가 아니라면(29번째 줄) 해당 단말 노드의 자식을 우선순위 Q에 넣도록 한다.

```
Algorithm common angle object processing (n)
input : 우선순위 큐의 entry, n
output : Nearest Surrounder result

1.      if (n이 leaf node이면) {
2.        next ← Q의 다음 entry
3.        while (next의 id와 n의 id가 다르면) {
4.          if (next가 leaf node이면) {
5.            ns ← select NS (n, next)
6.            if (ns == n이면) { // 그림 6 (b), (c) 경우
7.              // b의 경우
8.              우선순위 Q에서 next를 삭제
9.              next ← Q의 다음 entry
10.             // c의 경우
11.             next와 같은 id를 가진 entry를 Q에서 삭제
12.             next ← Q의 다음 entry
13.           } else { // 그림 6 (a), (d) 경우
14.             // a의 경우
15.             n과 같은 id를 가진 entry를 Q에서 삭제
16.             n ← Q의 다음 entry
17.             // d의 경우
18.             n ← Q의 다음 entry
19.           }
20        } else {
21.           if (그림 6(a), (b), (d)이면) {
22.             next의 자식 노드를 Q에 삽입
23.           } else {
24.             next와 같은 id를 가진 entry 삭제
25.           }
26.       }
27.       next ← n 다음 entry
28.     }
29.   } else {
30.     자식 노드를 Q에 삽입
31.   }
```

알고리즘 2 common angle object processing

3.3 공통되는 각을 가진 객체 중 NS를 선택하는 방법

그림 6의 (c), (d)의 경우 o_1과 o_3의 꼭지점의 좌표 값의 대소 관계를 비교한다. 그런데 기준이 되는 객체가 어느 사분면에 있느냐에 따라 각기 다르게 적용할 필요가 있다. 기준이 되는 객체는 보다 작은 α값(그것이 α_{start}이든 α_{end}이든)을 갖는 우선순위 큐 엔트리의 객체이다. 예를 들어 이 경우에서는 o_1을 기준으로 그림 7과 같이 영역을 나누어야 한다. o_1의 네 꼭지점이 영역을 나누는 지표가 된다. 이렇게 나눈 영역 중에서 o_3의 오른쪽 아래 점이 o_1을 기준으로 어느 부분에 속하는지를 판단해야 한다. 그런데 항상 오른쪽 아래 점을 보는 것이 아니라 현재 기준이 되는 객체가 몇 사분면에 위치했느냐에 따라 오른쪽윗점을 볼 수도 있고(2사분면인 경우), 왼쪽윗점을 볼 수도 있다(3사분면인 경우).

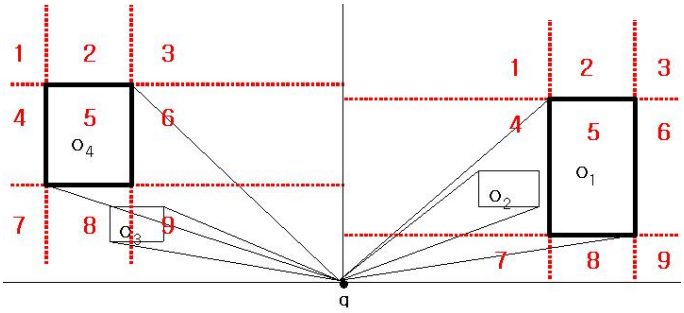


그림 7 현재 NS인 것을 기준으로 데이터 영역을 9개로 나눔

o_1의 각 범위가 [0, 90) 사이에 있고, o_2의 꼭지점이 4, 7, 8 영역 중 하나에 있다면 o_2는 해당 범위에서 NS가 된다. 마찬가지로 o_4의 각 범위가 [90, 180) 사이에 있다면 6, 8, 9 영역 중 하나에 있는 객체, o_3이 NS가 된다.

이상에서 설명한 것을 알고리즘으로 표현하면 다음과 같다.

```
Algorithm select NS (n1, n2)
input : 우선순위 큐의 원소 n1, n2
output : Nearest Surrounder node

1.      if ( 0 < n1->각도 < 90 ) {
2.        p1 ← n1의 오른쪽아래점
3.        p2 ← n2의 오른쪽아래점
4.        if (p1 기준, p2가 4번, 7번, 8번 영역 중 하나에 있으면)
5.          return n2
6.        else
7.          return n1
8.      } else if ( 90 < n1->각도 < 180 ) {
9.        p1 ← n1의 오른쪽윗점
10.       p2 ← n2의 오른쪽윗점
11.       if (p1 기준, p2가 6번, 8번, 9번 영역 중 하나에 있으면)
12.         return n2
13.       else
14.         return n1
15.     } else if ( 180 < n1->각도 < 270 ) {
16.       p1 ← n1의 왼쪽윗점
```

```
17.       p2 ← n2의 왼쪽윗점
18.       if (p1 기준, p2가 2번, 3번, 6번 영역 중 하나에 있으면)
19.         return n2
20.       else
21.         return n1
22.     } else if ( 270 < n1->각도 < 360 ) {
23.       p1 ← n1의 왼쪽아래점
24.       p2 ← n2의 왼쪽아래점
25.       if (p1 기준, p2가 1번, 2번, 4번 영역 중 하나에 있으면)
26.         return n2
27.       else
28.         return n1
29.     }
```

알고리즘 3 select NS

기준이 되는 객체(n1)와 질의점이 이루는 각이 몇 사분면에 속하는지 파악하고(1, 8, 15, 22번째 줄) n1과 공통 각을 가지는 n2가 n1에 대하여 어느 부분에 있는지를 파악한 후 NS를 결정한다.

4. 결론 및 향후 과제

이 논문은 새로운 질의 형태인 NS를 구하는 기존의 방법이 복잡하고 계산이 많다는 것을 개선하기 위해서 연구되었다. 기존의 NS를 찾는 방법은 공통 각을 찾기 위해서 비교 연산을 여러 번 해야 하는 복잡함이 있고 주어진 각 범위 내에서 거리를 구할 때 많은 연산 단계를 거쳐야 한다. 하지만 본 논문에서 제안한 방법은 새로운 NS가 발견 될 것으로 예상되는 지점에서 객체 간의 꼭지점 좌표 값의 대소를 비교함으로써 NS를 찾는다. 이 방법은 기존의 NS과 비교했을 때 연산 단계를 줄임으로써 성능 향상을 기대할 수 있다.

향후 연구 과제로 현재 진행 중인 구현을 통해 얼마만큼의 계산비용이 절감되는지를 확인할 것이다. 또 본 논문은 질의 점과 객체가 움직이지는 않는 환경에서 알고리즘을 전개하였으므로 앞으로 질의 점과 객체가 움직이는 환경에서 NS를 구하는 연구를 진행 할 계획이다.

5. 참고문헌

[1] N.Roussopoulos, S.Kelley, and F. Vincent, “Nearest neighbor Queries,” *In SIGMOD*, pp.71-79, San Jose, California, May 22-25, 1995.

[2] Ken C.K. Lee, Wang-Chien Lee, Hong Va Leong, "Nearest Surrounder Queries," *International Conference on Data Engineering (ICDE'06)*, pp. 85-94, Atlanta, Georgia, USA, 2006.

[3] A.Guttman, “R-trees: a dynamic index structure for spatial searching”, *In SIGMOD*, pp.47-57, Massachusetts, June 18-21, 1984.

[4] Gisli R. Hjaltason, Hanan Samet, “Distance Browsing in Spatial Databases,” *ACM Transactions on Database Systems*, Vol. 24, No. 2, pp.265-318, 1999.