가시성그래프에 의해 최소 여유공간을 보장하는 길찾기

전현주° 유견아
덕성여자대학교 컴퓨터공학부
{hzoo°, kyeonah}@duksung.ac.kr

Finding a path with the minimum clearance by using Visibility graph

Hyunjoo Jeon° Kyeonah Yu
Dept. of computer science, Duksung Women's University

요 약

최근 게임에서 자주 등장하기 시작한 랜덤 지형 맵 생성기법으로 인해 단순한 경로 찾기가 아닌 지형분석을 통한 복잡한 경로 찾기 문제가 많은 관심을 받고 있다. 이에 로보틱스 분야에서 경로 찾기에 이용되는 가시성그래프(Visibility Graph, Vgraph)가 지형분석과 경로 찾기를 동시에 해결할 수 있는 방법으로 제안되고 있다. Vgraph를 이용하면 지형의 로드맵을 효과적으로 생성할 수 있을 뿐 아니라 A* 알고리즘과 결합하여 최적의 경로를 찾는 것을 보장하는 장점이 있다. 그러나 Vgraph에 의해 구해진 경로는 장애물의 정점에서 정점으로 이동하기 때문에 항상 장애물의 모서리를 향해 움직이며 벽에 붙어가는 듯이 보여 부자연스러운 것이 단점이다. 본 논문에서는 설계자가 요구하는 여유공간 c만큼 장애물을 확장하여 확장된 장애물에 대해 가시성그래프를 생성함으로써, Vgraph의 장점은 유지하며 단점을 보완할 수 있는 방법에 대해 제안한다.

1. 서 론

최근 게임을 더욱 흥미롭게 만들기 위해 사용자가 직접 지형을 변화시키는 랜덤 지형 맵 생성 기법을 제공하는 게임이 많아지고 있다 [1]. 이는 개발자가 과거처럼 지형을 미리 분석해 적절한 웨이포인트들을 설정하고 A* 알고리즘을 적용하여 길찾기를 하던 방식은 더 이상 사용하기 힘들어졌다는 것을 의미한다. 여러 게임들에서 자체적으로 지형 분석 능력을 갖도록 하고 있는데 이중, Force21나 Sims에서 가시성그래프(Vgraph)를 이용하여 지형의 로드맵을 자동으로 생성하고 A* 알고리즘을 이용해 길찾기를 하는 방법을 사용하고 있다. Vgraph는 장애물의 정점들 사이의 가시성 여부를 판단하여 가시성이 있는 정점들을 연결하여 생성된 선분들과 장애물의 경계선들을 그래프의 링크로 하며 장애물의 정점들을 그래프의 노드로 하여 구성된 무방향성 그래프이다. Vgraph 자체가 로드맵의 역할을 하여 A* 알고리즘을 이용하면 최단거리의 경로를 구할 수 있음이 증명되어 있다[2]. 그러나 Vgraph의 생성과정에서 알 수 있듯이 Vgraph이 의한 길을 따라가는 것은 장애물의 한 정점에서 다른 장애물의 정점으로 이동하거나 장애물의 경계선을 따라가는 것으로 게임에서 캐릭터가 움직이는 방식으로는 적합하지 않은 것이 사실이다.

여유공간 0의 경로를 생성하는 것이 Vgraph라면 최대 여유공간의 경로를 생성하는 것은 Voronoi 다이어그램을 이용한 방법이다. Voronoi 다이어그램은 장애물에 이르는 거리가 같은 점들로 이루어져 장애물에서 멀리 떨어진 자유공간에 경로를 생성하도록 해준다. 이와 같이 생성된 경로는 장애물로부터 멀리 떨어진 경로를 생성하여 안정감이 있는 반면 최단 경로보다 상당히 우회하게 된다.

본 논문에서는 최단 경로를 제공하는 Vgraph 방식을 이용하되 장애물의 경계선을 따라 가는 단점을 해결하기 위해 장애물의 경계를 원하는 여유공간 c만큼 부풀린 후 Vgraph를 생성하여 길찾기 하는 것을 제안한다. 이와 같이 장애물을 부풀리면 기존의 자유공간이 장애물의 확장된 부분에 막혀 경로가 우회해야 하는 경우가 발생하는데 이를 해결하기 위해 부풀린 장애물이 겹치는 부분에서는 Voronoi 다이어그램의 성질을 이용하여 링크를 추가함으로써 처리하는 방법을 소개한다.

2. 가시성그래프를 이용한 길찾기

가시성그래프를 이용한 길찾기 과정은 크게 두 단계로 나누어 볼 수 있다. 첫 번째는 로봇의 시작점과 목표점, 그리고 장애물로부터 Vgraph를 생성하는 단계이고 두 번째는 생성된 Vgraph를 이용하여 최단거리를 탐색하는 단계이다.

Vgraph는 장애물의 꼭지점들을 노드로 하고 꼭지점들을 연결한 선분중에 장애물과 겹치지 않는 선분을 링크로 하는 무방향 그래프로 정의된다. Vgraph를 생성하는 첫 번째 단계에서는 노드와 링크를 생성한다. Vgraph의 노드는 시작점과 목표점, 그리고 장애물의 정점들로 이루어진다. 이렇게 생성된 노드 중 중 임의의 두 개의 노드를 연결한 선분과 장애물이 교차하지 않는다면, 즉 두 노드가 서로 보인다면(visible), 이는 Vgraph의 링크가 된다. 그림 1은 이렇게 생성된 Vgraph이다. 이들 링크 중에 지지선(supporting lines)이 장애물을 뚫고 지나가는 링크들은(그림 1에서 점선으로 표기된 링크들) 최단 경로의 일부가 절대 되지 않기 때문에 경로 찾기 문제에서는 이들을 생략하고 연결하는 장애물에 대해 모두 bitangent한 링크만으로 이루어진 축소(Reduced) Vgraph를 생성하기도 한다[3].

최단 경로를 찾기 위한 두 번째 단계에서는 여러 가지 탐색 알고리즘을 사용할 수 있는데, 휴리스틱값으로 노드간의 직선거리를 취하는 A* 알고리즘의 경우, 최단경로를 보장할 뿐 아니라 시간복잡도도 장애물 정점의 수에 대하여 선형이므로 게임이나 로보틱스 분야에서 가장 많이 사용되고 있다.

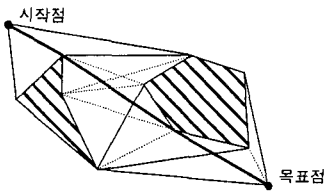


그림 1 Visibility Graph의 예

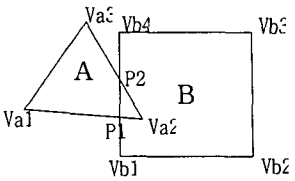


그림 2 두 다각형을 합집합하는 알고리즘

3. 최소 여유공간 c를 보장하는 길찾기

가시성그래프를 이용하여 경로를 구하되 원하는 여유공간을 보장하도록 하기 위한 알고리즘은 다음과 같은 단계를 따른다:

1단계: 장애물의 경계선을 원하는 여유공간 c만큼 확장
2단계: 확장된 장애물의 합집합
3단계: 2단계 결과에 대해 가시성그래프 생성
4단계: 확장된 장애물이 교차되는 부분의 여유공간이 c 이상인 부분에 대해 중선을 생성하여 링크로 추가
5단계: A* 알고리즘을 이용하여 최단경로 구함.

3.1 장애물의 확장

다각형 장애물을 c만큼 확장하는 것은 *Minkowski* 합과 차 연산의 특별한 경우로 정의되는 양의 솔리드 오프세팅(positive solid offsetting) 연산을 이용하면 된다. 다각형을 *set S*로 표시하면 *S*의 오프셋 *ES*는 다음과 같이 정의된다: $ES=\{p: \exists q\in S, \|p-q\|\le c\}$[4].

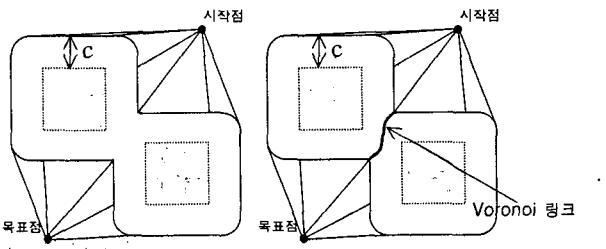
위 식과 같이 정의된 양의 솔리드 오프세팅을 구현하기 위해 주어진 *n*각 다각형(n-sided polygon)을 시계반대 방향의 정점 $[v_1, v_2, \cdots, v_n]$의 순서집합 (ordered list of vertices)으로 나타내고 n_i-와 n_i+를 각각 선분 $[v_{i-1}, v_i]$와 $[v_i, v_{i+1}]$에 대한 outward normal이라고 하자. θ= *angle* (n_i-, n_i+)의 부호에 따라 볼록 정점(convex vertex)이면 v_i를 중점으로 하고 n_i-와 n_i+의 방향으로 각각 *c*만큼 이동한 점을 끝점으로 하는 호가 형성되고, 오목 정점(concave or reflex vertex)이면 n_i-, n_i+의 중선 방향으로 $c/\sin(\theta/2)$만큼 이동한 정점으로 변환된다.

3.2 확장된 장애물의 합집합

확장된 장애물은 일반화다각형이지만 다각형의 합집합을 구하는 알고리즘과 동일하므로 다각형에 대해 설명한다. 다각형을 정점들의 순서리스트로 표현하고 그 정점들이 이루는 선분들에 대해 sweep-line 알고리즘[5]을 이용하여 교차점들을 모두 찾아낸다. 교차점들을 순서리스트에 적절히 삽입하고 새로운 도형을 트레이싱 해낸다. 이 과정을 예를 들어 설명하면 그림 2에서 다각형 A와 B를 각각 $\{V_{a1}, V_{a2}, V_{a3}, V_{a1}\}$와 $\{V_{b1}, V_{b2}, V_{b3}, V_{b4}, V_{b1}\}$의 순서리스트라고 하자. A와 B가 P_1과 P_2에서 교차한다고 하면 도형 A의 V_{a1}에서 트레이싱을 시작하여 P_1을 만나면 도형 B의 P_1을 찾아 그로부터 트레이싱을 계속하고 P_2에 이르면 다시 도형 A에서 P_2를 찾아 V_{a1}을 만나면 이 과정을 종료하게 된다. 이 결과는 다음과 같은 순서리스트가 된다. $\{V_{a1}, P_1, V_{b1}, V_{b2}, V_{b3}, V_{b4}, P_2, V_{a3}, V_{a1}\}$ 호와 호, 호와 일반 선분의 교차점도 마찬가지로 처리하면 되므로 일반화 다각형의 합집합에도 적용할 수 있다.

3.3 가시성그래프의 생성

확장된 장애물들은 호를 포함하고 있기 때문에 [6]에서 언급된 바와 같이 호와 호, 호와 정점에 대한 가시여부(visibility)를 결정해야 한다. 두 경우에 호의 가시성을 따져주기 위해 가상점(*fictitious point*)을 생성한다. 호와 정점의 경우 노드로 정해진 정점과 호가 접하는 점이 가상점이다. 정점에서 하나의 호에 최대 2개의 가상점이 생성된다. 호와 호의 경우에는 하나의 호에 대해 외접점과 내접점이 최대 2개씩 그러므로 총 4개의 가상점이 생길 수 있다. Vgraph의 노드에는 가상점이 추가되고 링크에는 가상점을 생성한 선분과 호 위에 추가된 두 개의 이웃한 가상점을 연결한 부분이 추가된다. 이렇게 생성된 Vgraph의 예는 그림 3 (a)와 같다. 일반화 다각형에 대해 가시성그래프를 실제 구현한 사례는 [7]에서 소개된 바가 있다.



(a) Vgraph의 예 (b) (a)에 Voronoi 링크를 추가

그림 3 확장된 장애물에 대한 Vgraph와 추가된 링크

3.4 가시성그래프 링크의 추가

솔리드 오프세팅에 의해 장애물들이 요구된 여유공간 만큼 확장되면 자유공간이 축소되고, 최단경로를 제공하던 자유공간이 막히면(예로 그림 3(a) 두 장애물 사이) 경로가 크게 우회하여 생성되게 된다. 확장된 장애물에 의해 없어지는 자유공간은 크게 장애물의 확장된 부분이 다른 장애물 자체와 겹치는 경우와 확장된 부분끼리 겹치는 두 가지 경우로 볼 수 있다. 그림 3의 경우가 두 번째에 해당한다. 한 장애물의 확장된 부분이 다른 장애물과 겹친다는 것은 그 사이의 자유공간이 설계자가 요구하는 여유공간을 보장하지 못한다는 의미이므로 겹치는 부분을 그대로 두어 장애물로 취급하고 경로가 생기지 않도록 한다. 그러나 두 번째 경우와 같이 장애물의 확장된 부분끼리 겹쳐서 길이 막힌 경우에는 장애물 사이가 최소 여유공간 보다는 넓다는 의미이므로 최단경로를 찾기 위해서는 이 공간을 이용할 수 있도록 해 주어야한다.

확장된 장애물이 겹친 부분 중, 최소 여유공간을 보장하는 공간에 대해서는 양쪽 장애물에서 최대한 많이 떨어진 링크를 생성한다. 장애물들로부터 최대한의 여유공간을 제공하는 경로를 위해 서론에서 언급한 Voronoi 링크를 생성한다. Voronoi 링크는

장애물의 기하학적 요소에 따라 다음 3가지 경우로 분류된다: 1. 정점과 정점사이는 두 점의 이등분점, 2. 변과 변 사이는 선분과 선분의 이등분선, 3. 변과 정점 사이는 선분과 점에 이르는 거리가 같은 궤적을 그리는 포물선. 이에 따라 그림 3(a)에 Voronoi 링크를 추가하면 그림 3(b)와 같다. 링크를 추가하기 전에는 확장된 장애물이 겹치는 부분으로 처리되어 장애물 사이에 경로를 생성할 수 없었으나 Voronoi 링크가 추가됨에 따라 그 링크를 포함한 경로를 생성할 수 있게 되었다.

3.5 A* 탐색에 의한 경로찾기

A*탐색을 위한 상태공간 [N,A,S,G]는 다음과 같이 정의된다.

N : 가시성그래프 노드의 집합
A : 가시성그래프 링크 및 추가된 Voronoi 링크
S : 시작점
G : 목표점

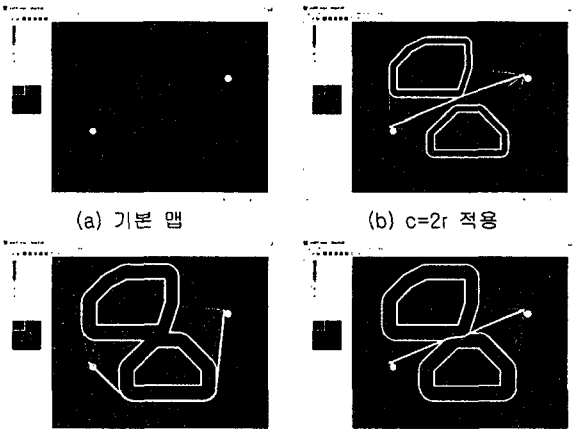
S에서 G에 이르는 경로를 찾는 것이 A* 탐색의 목표이다. 평가함수를 유클리디언 거리의 합으로 하고 휴리스틱으로 현재 노드에서 목표 노드까지의 직선거리를 취하면 과소평가된 휴리스틱 값으로 최적의 답을 찾아낼 수 있다. 이때 평가함수를 장애물의 고도 등과 같은 지형의 특성을 반영하여 A* 탐색을 하면 다른 척도에 대한 최적 경로를 찾을 수도 있다.

4. 시뮬레이션 결과

본 논문에서 제안한 여유공간을 적용한 가시성그래프의 생성과 이에 Voronoi 다이어그램을 확장 적용한 그래프를 생성하여 유클리디언 거리를 평가함수로 하는 A* 알고리즘을 적용한 결과를 비교하였다. 셀 기반 지형을 생성하기 위한 맵-에디터는 VC++를 이용해 제작하였다. 제작된 맵은 위치정보와 각 위치에 따른 셀의 정보를 2차원 배열의 맵 데이터로 저장하게 된다. 이렇게 저장된 맵 데이터의 값에 따라 셀 테이블의 데이터들을 호출하여 게임을 이룬다. 여기서 셀 테이블은 각 셀의 비트맵에 따른 고유 플래그를 저장하게 되고 플래그의 종류에 따라 이동가능 여부를 결정하게 된다.

그림 4(a)는 맵 에디터에 의해 생성된 게임 환경이다. 조그만 두 개의 원은 캐릭터의 출발점과 목표점을 나타내고 반경은 r이라고 하자. 레벨 설계자가 캐릭터가 움직이는데 필요한 여유공간 c를 움직이는 캐릭터의 반지름 r의 2배, 즉 2r로 설정한다면 그림 4(b)와 같이 원래 장애물들은 2r만큼 경계선이 확장되어 그 위에 Vgraph가 생성된다. A* 탐색 알고리즘으로 찾아낸 경로가 장애물 사이에 굵은 하늘색 선으로 표시된 것이다.

움직이는 캐릭터가 장애물 사이의 이동하는 공간에 대해 안정성을 더 확보하기 위해 여유공간 c를 4r로 지정한 경우의 결과는 그림 4(c)와 같다. 이 경우, 두 장애물의 확장된 공간이 겹쳐서 c가 2r일때 자유공간으로 남아 있던 장애물 사이 공간이 막힌 모습을 볼 수 있다. 이렇게 확장된 장애물에 의해 사라진 링크로 인하여 최단거리를 확보하지 못하고 그림 4(c)와 같이 아래로 우회하는 경로로 이동하게 된다. 같은 조건에서 그림 4(c)의 문제를 해결하기 위해 오버래핑이 일어나는 부분을 탐색하여 Voronoi 다이어그램을 적용한 결과가 그림 4(d)와 같다. 본 논문에서 제안하는 바와 같이 오버래핑 영역을 링크로 활용할 수 있어 자유공간이 축소되더라도 설계자가 요구하는 여유공간만 확보되어 있으면 경로를 우회하지 않는 방법으로 목표점을 찾아갈 수 있음을 확인할 수 있다.



(a) 기본 맵 (b) c=2r 적용

(c) c=4r 적용 (d) 그림(c)에 Voronoi 링크 추가

그림 4 결과 비교

5. 결론

본 연구에서는 안전하고 자연스러우며, 최단거리를 보장하는 경로를 찾기 위해 필요한 여유공간을 설정할 수 있도록 하여 장애물을 확장하고 확장된 장애물위에서 Vgraph를 생성하여 경로를 찾도록 하였다. 이때, 확장된 장애물끼리 오버래핑이 일어나 막힌 영역에 Voronoi 다이어그램을 적용하여 링크를 추가함으로서 멀리 우회하는 경로를 찾을 가능성을 없앴다. 제안된 방식으로 시뮬레이션한 결과, 찾아진 경로는 가장 가까운 루트를 이용하는 동시에 장애물로부터 일정 간격이상 떨어진 곳에 생성되어 자연스럽고 안정감 있게 보임을 알 수 있었다.본 논문에서는 이와 같은 성질을 가진 경로 생성 자체에만 중점을 두어, 시간복잡도 등의 연산의 효율성에 대한 고려는 하지 않았다. 실제 게임에 활용되기 위해서는 실시간으로 활용될 수 있는 정도의 효율적인 알고리즘으로 개선되어야 한다.

참고문헌

[1] S. Woodcock, "Game AI: The state of the industry", Game Developer Magazine, August, 2000.

[2] J.C. Latombe, Robot Motion Planning 4장, Kluwer Academic Publishers, 1991.

[3] R. Wein, J.P.van der Berg, and D. Halperin, "The Visibility-Voronoi Complex and Its Applications" Proc. European Workshop on Computational Geometry pp. 151-154, 2005.

[4] J. Rossignac and A.G. Requicha, "Offsetting operations in solid modelling", Computer Aided Geometric Design, vol. 3, pp129-148, 1986.

[5] J. O'Rourke, Computational Geometry in C, Cambridge Press, 1998.

[6] J.P. Laumond, "Obstacle growing in a nonpolygonal world", Inform. Proc. Letter, vol. 25(1), pp41-50, 1987.

[7] 유견아, 전현주, "컴퓨터 게임환경에서 일반화 가시성그래프를 이용한 경로찾기", 한국시뮬레이션학회 논문지 14(3), 2005.