NPC의 자연스러운 이동경로를 보장하는 효율적인 상태공간의 생성

(Efficient State Space Generation for Guaranteeing a Natural-Looking Path for NPCs)

유 견 아 †

(Kyeonah Yu)

요 약 컴퓨터 게임에서 NPC(non-player character)가 이동하는 자연스러운 경로를 찾기 위해서는 탐색을 위한 공간을 어떻게 표현할 것인가에 대한 연구가 어떤 탐색 방법을 사용할 것인가에 대한 연구 못지 않게 중요하다고 할 수 있다. 최근까지 게임 경로 찾기의 동향을 보면 경로 계획을 위한 탐색 방법으로는 A* 알고리즘이 단연 우위를 보이지만 A* 알고리즘을 적용하기 위한 상태 공간 표현 방식으로는 게임을 위해 만들어진 여러가지 표현 방법들이 사용되고 있다. 기존의 방법들은 탐색 공간의 크기가 너무 크거나, 최적의 경로를 찾지 못하거나, 경로가 자연스럽지 못하는 등의 단점 뿐 아니라 노드와 링크의 생성이 자동적이지 못하고 레벨 디자이너에 의존하는 것도 문제점으로 지적되고 있다. 본 논문에서는 경로가 자연스럽게 보이기 위한 성질을 정의하고 이를 충족하는 경로를 생성할 수 있도록 로보틱스 분야의 가시성그래프를 응용한 일반화 가시성그래프를 이용하여 상태공간을 표현할 것을 제안한다.

키워드 : 탐색, 경로계획, 상태 공간, 컴퓨터 게임, 가시성그래프

Abstract How to represent the search space is as important as which search algorithm to use for finding natural-looking paths for moving NPC (non-player character) in computer games. Recently, various state space representation methods which have been developed for computer games are being used while A* algorithm dominates as the preferred search algorithm. These representation methods show some drawbacks such as the size of state space is too large, there is no guarantee for optimality, the path found is not natural-looking, and the generation of nodes and links is not automatic by depending on a level designer. In this paper the requirements for natural-looking paths are introduced and to find paths satisfying these requirements, the use of the generalized visibility graphs which is the extended version of the visibility graph in Robotics is proposed.

Key words : search, path planning, state space, computer game, visibility graph

1. 서 론

최근 컴퓨터 게임에서 캐릭터들의 이동 계획을 위한 가장 많이 사용되는 방법은 단연 인공지능 분야의 A* 알고리즘이다. 응용 분야에 따라 D* 알고리즘[1]이나 IDA* 와 같이 변형된 A* 알고리즘[2]을 사용한다거나, 경로 찾기에 가장 많이 사용되는 휴리스틱인 유클리디언 거리에 고도나 가시여부 등을 혼용한 휴리스틱과 함께 사용한다거나[3], A* 알고리즘의 적용을 계층적으로 하는 등[4], 다양한 방법으로 탐색의 효율은 높이는 연구가 이루어져 왔다. 그러나 경로 찾기에 있어서 탐색방법의 결정만큼 중요한 이슈가 상태공간을 어떻게 표현하는 가에 관한 것이다[5,6]. 최근까지 게임 분야에서 많이 사용되고 있는 표현 방식으로는 균일 격자(regular grids), 웨이포인트 그래프(waypoint graph), 네비게이션 메쉬(navigation mesh) 등으로 게임의 효율성을 위해 개발 단계에서 레벨 디자이너가 직접 게임공간을 표현하는 경우가 많았다. 그러나 하드웨어의 발달로 자원의 제약이 약해진 지금은 보다 다양한 방법의 응용이 시도되고 있으며 로보틱스 분야의 가시성 그래프(visibility graph, Vgraph)도 그 중 하나이다[7,8]. 가시성 그래프를 포함한 기존의 방법들은 여러 가지 장단점을 가지고 있지만 공통적으로 생성된 경로가 직선 경로들

† 종신회원 : 덕성여자대학교 컴퓨터공학부 교수
kyeonah@duksung.ac.kr
논문접수 : 2006년 7월 21일
심사완료 : 2007년 1월 23일

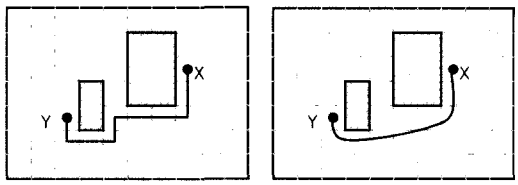
의 연결로 되어 있어 꺾임이 많고 자연스럽지 못하다는 단점이 있다. "캐릭터의 이동경로가 자연스럽다"를 정의하면 같은 배경에서 사람이라면 택했을 경로와 유사한 경로로 이동하는 것을 말한다[9]. 이와 같은 경로의 특징을 종합해 보면 첫째, 짧은 경로를 두고 크게 우회하지 않으며, 둘째 장애물들로부터 어느 정도의 간격(clearance)을 두고 움직이며, 셋째 턴(turn)을 할 때에는 완만하게(smooth) 한다는 것이다. 본 논문에서는 이와 같은 조건을 만족하는 경로를 보장하기 위하여 일반화 가시성 그래프를 이용하여 탐색공간을 표현하는 것을 제안한다.

일반화 가시성 그래프는 확장된 장애물 위에서 생성된 가시성 그래프이다. 확장된 장애물은 다각형 장애물의 경계로부터 일정한 거리 r만큼 오프세팅하여 얻어지는데 이렇게 구해진 도형은 선분과 원의 호로 이루어진 일반화 다각형(generalized polygon)이 되며 이 위에서 생성된 가시성 그래프를 일반화 가시성 그래프(generalized visibility graph, GVgraph)라고 한다[10]. GVgraph를 이용하여 생성된 경로는 다음과 같은 특징을 갖게 된다. 첫째, 최단 거리 경로를 제공하는 것이 증명된 Vgraph에서의 경로에 근사한 최적 경로(near optimal)를 찾는다[11]. 둘째, 확장된 장애물 위에 경로가 생성되므로 생성된 경로는 적어도 장애물로부터 r 만큼 떨어져 있다. 셋째, 링크의 연결 부분이 원의 호로 되어 있어 급격한 턴이 없어진다. 결론적으로 게임공간을 GVgraph를 이용해 표현하면 찾아진 경로는 자연스러운 경로가 갖추어야 할 항목을 모두 만족시킨다는 것이다. 본 논문에서는 게임 공간을GVgraph를 이용하여 표현하는 방법을 설명하고 이를 경로 계획에 효과적으로 이용하기 위한 빠른 알고리즘을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 게임공간 표현에 대한 여러 가지 관련연구에 대해 살펴보며, 3장에서는 일반화 가시성 그래프로 게임 공간을 표현하는 과정에 대하여, 4장에서는 상태공간을 정의하고 A*를 이용하여 효과적으로 경로를 계획하는 방법에 대하여 설명한다. 5장에서는 결과를 분석하고 실제 게임 환경에 적용하여 실행한 실험 결과를 제시하고 마지막으로 6장에서 결론을 도출한다.

2. 관련 연구

균일 격자로 게임 공간을 표현하는 것은 가장 간단한 방식이다. 그러나 A* 알고리즘을 적용하기 위해 각 셀을 그래프의 노드로, 이웃한 노드들을 자식 노드로 간주하기 때문에 탐색공간의 크기가 크며 구해진 경로는 꺾임이 많아 어색하게 보이는 단점이 있다. 그러므로 탐색을 계층적으로 적용하여 탐색의 효율을 높이기도 하고



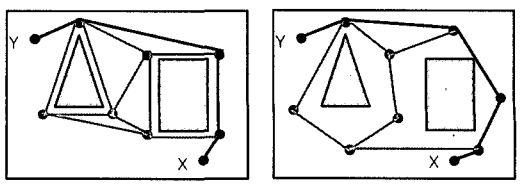
(a) (b)

그림 1 균일 격자 - 후처리 과정을 통한 경로의 완성

[4] 찾아진 경로의 지그재그 효과를 없애기 위해 잡아당겨 펴기(string pulling) 및 곡선화(smoothing) 등의 후처리 과정(post processing)을 통해 문제를 해결하는 추가적인 노력이 필요하다[12,13]. 그림 1(a)는 상하좌우의 운동을 허용했을 때 A*를 적용한 결과이고 (b)는 이에 대해 후처리 과정을 통하여 곡선화 된 결과이다.

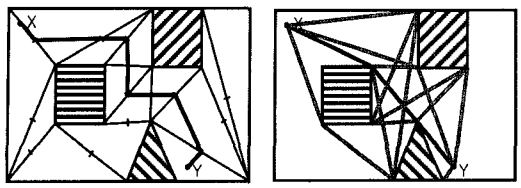
웨이포인트 그래프 방식은 레벨 디자이너가 임의의 장소에 노드를 두고 그 사이의 링크를 결정하는 방식으로 탐색공간의 크기는 비교적 작지만 최적의 경로를 찾는 것을 보장하지 못하며 무엇보다도 레벨 디자이너 개인에 의해 게임 공간이 표현됨으로써 여러 단계의 개발자들의 조율이 문제가 된다. 노드를 장애물의 모서리에 두는 모서리 그래프 예와 장애물 사이에 두는 일반 웨이포인트 그래프 예를 그림 2(a), (b)에서 각각 보여준다. 두 가지 방식 모두 그래프가 만들어진 이후, 경로찾기를 위해 출발점과 목표점(X, Y)을 그래프에 삽입하는 과정이 문제가 된다. X, Y와의 가능한 링크를 모두 찾는 것은 계산적으로 복잡하므로 가장 가까운 노드에 연결하여 탐색을 시작하는데 이로 인해 찾아진 경로는 최적이 아니게 되며 균일 격자 표현에서와 마찬가지의 후처리가 필요하다.

네비게이션 메쉬는 작은 다각형 조각으로 게임 환경을 표현하는 방식으로 3차원 게임 공간 등, 다양한 환경을 표현할 수 있는 장점이 있으나 이들은 경로찾기에 직접적으로 이용되지 않는 지형에 관한 데이타로 탐색공간의 크기가 크면서도 A* 알고리즘을 이용한 경로찾기에 적합하지 않다. 또한 자동 생성 방법들이 제안되고 있으나 아직은 레벨 디자이너에 의해 생성되고 있으며 생성된 경로 역시 꺾임이 많아 부자연스럽다(그림 3(a)).



(a) (b)

그림 2 웨이포인트 그래프 방식에 의한 경로



(a) (b)

그림 3 삼각형 기반 네비게이션 메쉬와 가시성 그래프

그러므로 네비게이션 메쉬에 의해 표현된 게임을 경로 찾기에 적합하도록 수정하는 방법들이 제안되고 있다[3].

기존의 게임 공간 표현 방식들의 문제점들을 해결하기 위하여 새롭게 제안되고 있는 방식이 로보틱스 분야의 가시성 그래프이다[7]. 가시성 그래프는 장애물들을 다각형으로 모델링하고 다각형의 꼭지점들을 그래프의 노드로 하고 서로 보이는(visible) 노드들끼리 연결하여 그래프의 링크로 한다(그림 3(b)). 즉, 가시성 그래프는 결국 맵에서 경로찾기에 필요한 연결 정보만을 뽑아 내서 효율적이고 자동적으로 공간을 표현한 것이 가장 큰 장점이라고 할 수 있다. 뿐만 아니라 이렇게 구해진 경로는 최적이라는 것이 증명되어 있다. 하지만 가장 큰 취약점은 경로가 장애물의 변 혹은 꼭지점 위에 생성되는 wall-hugging 문제이다. 본 논문에서는 가시성 그래프의 장점을 유지하면서도 wall-hugging문제를 해결할 수 있도록 일반화 가시성 그래프를 이용하여 게임 공간을 표현할 것을 제안한다.

3. GVgraph를 이용한 게임 공간의 표현

앞 장에서 지적한 가시성 그래프의 단점을 보완하기 위하여 우선 장애물의 변들을 r만큼 오프세팅하여 장애물을 확대한다. 여기서 r을 캐릭터를 둘러싸는 원의 반지름으로 하면 움직이는 캐릭터가 장애물과 부딪히지 않는 경로를 찾을 수 있을 뿐 아니라 r을 더 크게 하면 캐릭터가 장애물로부터 여유공간(clearance)을 가지고 움직이는 것처럼 보이게 할 수 있다. 또한 캐릭터를 원으로 모델링 하면 방향성을 고려하지 않아도 되므로 장애물의 r만큼 부풀리는 확장 과정이 용이해진다.

3.1 장애물의 확장

장애물의 경계선 확장은 민코스키 합(Minkowski sum)의 한가지 구현 방법인 양의 솔리드-오프세팅(positive solid-offsetting)에 의해 이루어진다. S를 다각형 장애물을 표현하는 집합이라고 하면, r의 솔리드-오프세팅은 다음과 같이 정의된다: $\{p : \exists q \in S, \|p-q\| \le r\}$ [14]. 이와 같은 정의에 따라 솔리드-오프세팅을 구현하기 위하여는 장애물을 n개의 꼭지점으로 시계 반대 방향의 리스트 $[v_0, v_1, v_2, \cdots, v_{n-1}, v_n]$($v_0=v_n$)로 표현하고 다음과 같이 볼록(convex)과 오목(concave) 꼭지점의 경우로 나누어 처리한다. n_{i-}와 n_{i+}를 선분 $[v_{i-1}, v_i]$와 $[v_i, v_{i+1}]$ 각각의 외법선(outward normal)이라고 할 때, $\theta = angle(n_{i-}, n_{i+})$가 양이면, v_i는 볼록이고 θ가 음이면 v_i는 오목이다. v_i가 볼록 꼭지점인 경우에 r만큼 양의 오프세팅을 하면 v_i를 원의 중심으로 하고 v_i로부터 각각 n_{i-}와 n_{i+} 방향으로 r만큼 뻗어 나간 두 점을 종점(endpoints)으로 하는 호로 변환된다. v_i가 오목꼭지점인 경우에는 v_i는 n_{i-}와 n_{i+}의 중선 방향으로 $r/sin(\theta/2)$만큼 뻗어 나간 점으로 종점으로 하는 호로 변환된다. 그림 4에서 v_1과 v_3는 볼록 꼭지점, v_2는 오목 꼭지점의 예이다. 다각형 장애물을 솔리드-오프세팅 하면 결과는 지역적 비볼록 일반화 다각형(locally non-convex generalized polygon)이 된다[11]. 즉, 모든 일반화 다각형의 오목 부분들은 두 개의 선분이 교차함으로써 얻어지는 교차점들로만 구성된다.

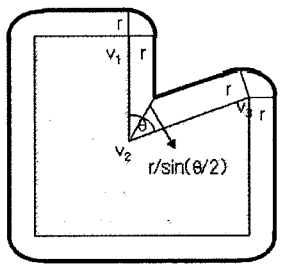


그림 4 솔리드-오프세팅 연산 결과 : 일반화 다각형

3.2 일반화 가시성 그래프(GVgraph)의 생성

3.1절에서 솔리드-오프세팅에 의한 결과는 일반화 다각형으로 확장된 장애물의 집합이다. 일반화 다각형 위에 가시성 그래프를 생성하면 링크가 선분과 원의 호로 이루어지게 되는데 이를 일반화 가시성 그래프라고 한다[10]. GVgraph를 생성하기 위하여는 일반화 다각형 사이의 가시성 관계(visibility relation)가 정의 되어야 한다. 즉, 다각형의 꼭지점 사이에서 정의되어 있는 가시성 관계를 꼭지점과 원의 호, 원의 호와 원의 호 사이로 확장해야 한다. 솔리드-오프세팅에 의해 확장된 장애물들은 볼록 부분은 원의 호이고 오목 부분은 오목 꼭지점이라는 사실에 주목하면 꼭지점과 원의 호 사이의 가시성은 고려하지 않아도 됨을 알 수 있다. 왜냐하면 오목 꼭지점으로부터의 링크는 절대 최단 거리 경로를 구성하지 않기 때문이다[11]. 두 개의 호 사이의 가시성 관계는 다음과 같이 정의된다: 점 p_1, p_2를 각 호 위의 점이라고 하고 선분 $[p_1, p_2]$가 두 호에 모두 접할 때, "두 호는 서로 가시성 있다(visible)"라고 하며 이 때 점 p_1, p_2를 가상점(fictitious vertices)이라고 한다[10]. 그

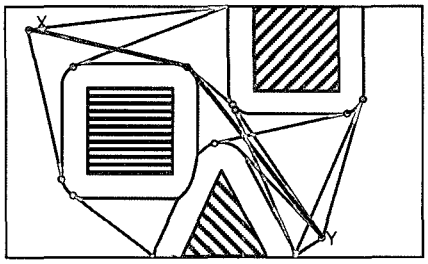


그림 5 GVgraph로 표현한 탐색 그래프

러면 탐색 그래프의 노드는 호의 두 종점들과 가상점들이 되며 링크는 장애물의 경계선(boundaries)과 가상점을 잇는 가시 링크(visible link)가 된다. 그림 5에서 흰색 노드들이 이와 같이 생성된 가상점들이다.

다각형 환경의 꼭지점의 수가 n이라고 할 때, 두 호를 잇는 접선은 최대 4개까지 가능하므로 호 위에 생성될 수 있는 가상점의 수는 O(n)이고 탐색그래프의 노드 수는 $O(n^2)$이 된다. 그러나 가상점의 생성은 링크와 동반하므로 탐색 그래프의 링크의 수는 $O(n^4)$이 아니라 $O(n^2)$이다[11].

4. A* 알고리즘에 의한 경로찾기

A* 알고리즘을 이용하여 최단 경로를 찾기 위해서는 우선 문제를 상태공간 표현(state space representation)으로 정의한다. 상태공간은 일반적으로 [S, G, N, A]의 4 가지 요소로 된 집합으로 표시한다[15].

S: 초기 상태, 자유 공간에 있는 출발점

G: 목표 상태, 자유 공간에 있는 도달하고자 하는 점

N: 상태의 집합, GVgraph로 표현된 탐색 그래프의 노드 및 S와 G

A: 링크의 집합, GVgraph로 표현된 탐색 그래프의 링크 및 S와 G로부터 GVgraph로 연결된 가시 링크(visible links)

정적인 장애물에 대해 GVgraph는 한번 생성되면 지속적으로 이용할 수 있지만 S와 G가 정해지면 집합 A를 생성해야 한다. 이를 위해서는 S와 G로부터의 다른 노드로의 연결선이 기존의 탐색 그래프와 일일이 교차여부를 확인하여야 하므로 시간이 많이 걸리는 것이 단점이다. 본 논문에서는 이 과정을 효과적으로 처리하여 경로 계획에 걸리는 시간을 단축하는 동시에 근사 최단거리의 경로를 구할 수 있는 알고리즘을 소개한다.

4.1 S와 G로부터의 가시 링크 생성

출발점 S와 목표점 G가 주어지고 S로부터 G에 이르는 경로를 찾는 부분이 게임 실행 동안 반복적으로 일어나는 부분이다. 그러므로 S와 G를 빠르게 탐색 그래프에 포함시키는 일은 전체 성능에 가장 크게 영향을 미치는 부분이라고 할 수 있다. S와 G로부터 가시 접선은 O(n)이므로 각 접선에 대해 장애물과 교차 여부를 확인하는 일은 $O(n^2)$이 걸리게 되는데 이 실행 시간을 회전 plane sweep(Rotational Plane Sweep, RPS) 알고리즘을 이용하여 O(nlogn)으로 향상시키고자 한다. RPS알고리즘은 Vgraph를 효율적으로 생성하기 위해 D.T. Lee[16]에 의해 제안된 방식으로 본 장에서는 RPS 알고리즘을 설명하고 일반화 다각형 환경을 위해 수정된 RPS알고리즘을 소개한다.

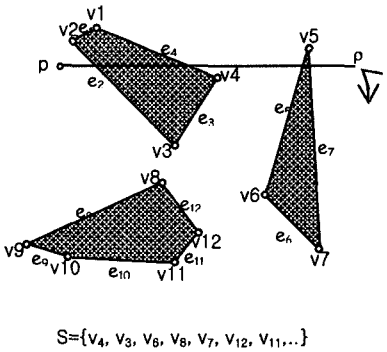
4.1.1 회전 plans-sweep(RPS) 알고리즘[17]

RPS 알고리즘의 핵심 아이디어는 교차 테스트를 순환적(cyclic) 순서로 수행하면서 얻어진 가시성 정보를 다음 순서에서의 가시성을 결정할 때 이용하는 것이다. 이는 연산 기하학 분야의 널리 알려진sweep-line 알고리즘과 비슷한 패러다임으로 차이점은 sweep-line을 수직-수평으로 이동하는 것이 아니라 임의의 점을 중심으로 반직선 ρ를 (0, 2π]로 회전시키면서 스위핑한다는 것이다.

탐색 그래프에 추가하고자 하는 노드(이 문제에서는 S와 G를 나타내는데 p로 대표하여 표기한다.)와 장애물 집합 O를 바탕으로 가시성을 판별하는 루틴은 두 단계의 초기화 과정을 거치는데 첫째로 p와 O를 입력 받은 후 p에 대한 초기 반직선(half-line) ρ를 기준으로 O에 포함된 장애물의 꼭지점을 시계방향으로 정렬하여 S라고 한다. 두 번째 과정으로 양의 방향 x축과 평행한 초기 half-line ρ와 교차하는 장애물의 선분이 있으면 교차 순서대로 이진탐색트리에 저장한다. 이러한 초기화 과정을 거친 후 S의 정렬된 순서대로 이진탐색트리를 이용하여 가시성을 판별한다. v_i에 대한 가시성 여부의 판별은 이진탐색트리의 가장 왼쪽(leftmost) 노드와의 교차 여부만 확인해 주면 되므로 O(logn)의 시간이 소요된다. 판별이 끝나면 현재의 장애물의 꼭지점을 기준으로 시계방향(CW) 즉, 교차여부를 판별해 나가는 방향에 있는 장애물의 선분은 이진탐색트리에 저장하고 반시계방향(CCW) 즉, 교차판별 영역에서 벗어난 선분은 이진탐색트리에서 삭제한다. 이러한 과정을 정렬된 노드의 개수만큼 수행하여 p에 대하여 가시성을 갖는 모든 노드 집합을 반환한다. 그림 6은 예를 이용하여 이 과정을 설명하고 있다. 이 알고리즘의 시간복잡도(time complexity)를 분석해 보면, 초기에 정렬 리스트와 이진탐색트리를 생성하는 데에 O(nlogn), 삭제 및 삽입 연산에 의해 트리의 상태를 갱신하는 데에 O(logn), 교차 여부를 판별하는 데에 O(logn)으로 전체 O(nlogn)의 시간 복잡도가 된다.

4.1.2 일반화 다각형 환경에서의 RPS 알고리즘

RPS 알고리즘을 일반화 다각형 환경으로 확장하기



S={v_4, v_3, v_6, v_8, v_7, v_{12}, v_{11},..}

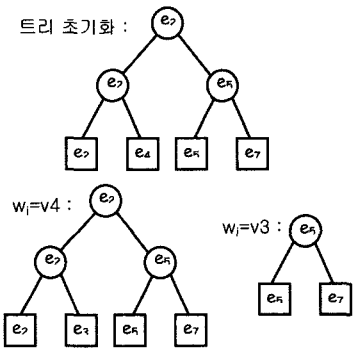


그림 6 이진탐색트리의 초기화 및 삽입, 삭제 연산

위해서는 초기화 과정에 가상점을 생성하는 단계를 추가하여 세 단계로 늘려야 하며 점 p로부터 가시성 여부를 판단하기 위해 교차 여부를 테스트 하는 알고리즘도 호를 포함하여 확장하여야 한다. 첫 번째 초기화 단계로 p에 대하여 장애물의 호 위에 가상점을 생성하며 이렇게 생성된 가상점을 O의 꼭지점 집합에 포함시킨다. 원래 RPS 알고리즘에서 다각형 장애물의 꼭지점들을 초기 단계에 정렬하는 이유는 꼭지점들이 이진탐색트리의 상태 변화를 위한 크리티컬 이벤트(critical event)들이기 때문이다. 반직선 ρ가 시계방향으로 스위핑하면서 만나는 선분이 꼭지점을 기점으로 달라지는데 이때 이진탐색트리를 갱신하여야 한다. 점 p에서 가상점을 생성하는 과정은 3.2절에서 가시 링크를 생성하는 방법과 동일하다. 그림 7에서 검은 점들은 기존의 종점들이고 흰 점들이 노드 p에 대하여 생성된 가상점이다. 가상점에 의해 하나의 호가 두 개의 세그먼트 e_3와 e_4로 나뉘었음을 볼 수 있다. 가상점들과 기존의 종점들이 집합 S의 원소가 되면 첫 번째 단계가 끝나며 다음 두 단계로는 기존의 RPS 알고리즘의 초기화 단계를 차례로 실행한다. 이와 같은 초기화 단계를 거쳐 진행되는 RPS알고리즘을 정리하면 그림 8의 흐름도와 같다.

초기화에 의해 정렬된 리스트와 반직선 ρ와의 초기 교차 상태를 나타내는 이진탐색트리로부터 시작하여 점

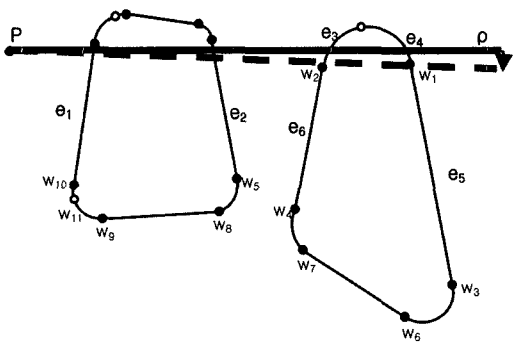


그림 7 일반화 다각형 환경에서의 RPS알고리즘 진행

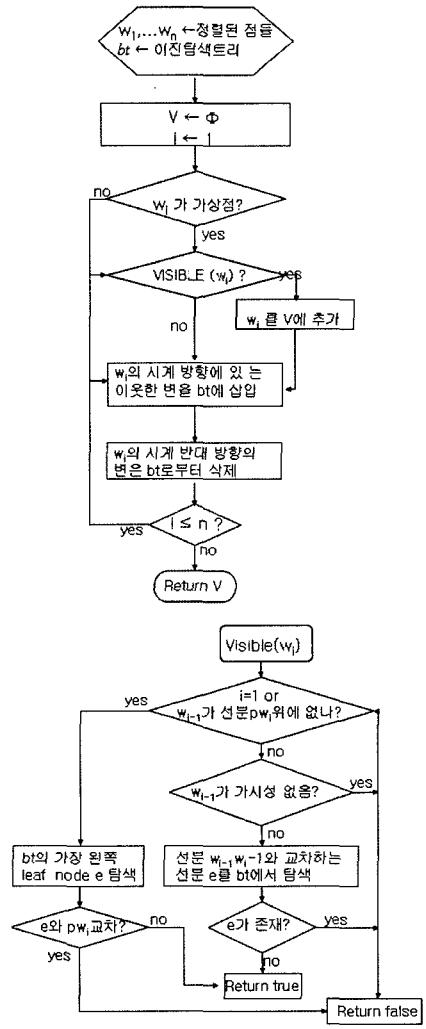


그림 8 일반화 다각형 환경에서의 RPS알고리즘 진행

p와 가시성이 있는 점들의 집합 V를 리턴하게 된다. 이 흐름도에서 VISIBLE(w_i)은 점 p와 w_i의 가시여부를 판

단하여 참 혹은 거짓을 리턴하는 부함수이다. VISIBLE (w_i)도 선분과 선분의 교차 여부를 확인하는 기존의 알고리즘을 선분과 호와의 교차 여부를 확인할 수 있도록 확장해야 한다. 이를 위하여 [10]에서 제안한 호로 확장된 plane-sweep 알고리즘을 사용하였다. 호를 포함한 환경에서도 점 p에서 호에 대한 가상점은 최대 2개까지 생성되므로 크리티컬 이벤트는 O(n)이고 알고리즘의 나머지 과정은 동일하게 진행되므로 전체 시간 복잡도는 변함없이 O(nlogn)이 된다.

4.2 A*에 의한 경로 탐색

상태 공간이 만들어지고 나면 경로찾기의 마지막 단계로 A* 알고리즘을 이용하여 최단 경로를 찾는다. 여기서 휴리스틱으로는 목표점까지의 직선 거리를 나타내는 유클리디언 거리를 사용한다.

정리 1. GVgraph에서의 탐색을 위한 유클리디언 거리 휴리스틱은 허용 가능(admissible)하다.

정리 1에 대한 증명 $p=(v_0,v_1,\cdots v_k)$를 v_0 = u로부터 v_k=g까지의 경로라고 하자. h(u)를 목표 노드까지의 직선거리라고 했으므로 $h(u) \le h(v)+w(u,v)$이다. 여기서 w(u,v)는 u에서 v로의 실제 이동에 드는 비용이며 GVgraph에서는 선분 혹은 원의 호가 된다. 그러면,

$$
\begin{aligned}
w(p) &= \sum_{i=0}^{k-1} w(v_i, v_{i+1}) \\
&\ge \sum_{i=0}^{k-1} (h(v_i) - h(v_{i+1})) \\
&= h(v_0) - h(v_k) \\
&= h(u) - h(g) = h(u) \\
&\therefore h(u) \le w(p)
\end{aligned}
$$

이 되어 임의의 노드 u에서의 휴리스틱 값(h(u))은 실제 목표에 이르는 비용(w(p))보다 적게 추정된다. □

그러므로 휴리스틱의 허용 가능 성질에 의해 경로가 존재하면 최단 거리 경로를 찾고 리턴하며 그렇지 않으면 '실패'를 리턴한다.

정리 2. 탐색에 걸리는 시간 복잡도는 GVgraph의 노드 수가 $O(n^2)$일 때 $O(n^2)$이다.

정리 2에 대한 증명 정리 1의 증명에서 $h(u) \le h(v) + w(u,v)$의 의미는 이 휴리스틱에 의하여 탐색을 하는 중에 한번 찾은 노드를 더 짧은 경로로 다시 발견 할 수 없다는 단조성(monotonicity)을 의미한다. 그러므로 최악의 경우에도 탐색하는 노드의 수는 전체 상태 수보다 적다. 그러므로 탐색에 소요되는 시간 복잡도는 전체 상태수의 복잡도인 $O(n^2)$와 동일하다. □

휴리스틱이 허용 가능함을 증명할 수만 있다면 더욱 다양한 휴리스틱과 함께 A* 알고리즘을 적용하여 찾아지는 경로의 질을 조정할 수 있다. 예로써, [3]에서는 고도(height)에 관한 정보를 거리와 함께 사용한 예를 보였다.

5. 실험 및 결과

게임 공간을 효과적으로 표현하기 위해 제안된 방식의 실험 결과를 3가지로 정리하였다. 첫 번째는 GVgraph 방식과 기존의 다른 게임공간 표현 방식과 비교 결과를 나타내었으며 두 번째로는 다각형 장애물을 부풀리는 정도(허용하는 여유공간)를 너무 크게 하여 통로가 사라지는 경우를 해결 방안과 함께 보여 주었다. 마지막으로는 정적 장애물에 대한 GVgraph 생성 후, 출발점과 목표점을 삽입하고자 본 고에서 제안한 알고리즘이 전체 GVgraph를 생성하는 것에 비해 얼마나 효율적인가에 대한 실험 결과를 보여주었다.

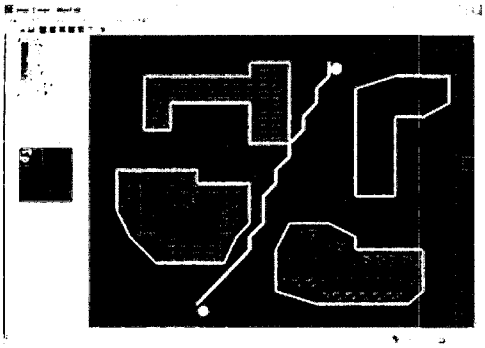
5.1 탐색공간 표현 방식별 A* 알고리즘 실행 결과

탐색 공간의 선택에 있어서 고려되는 사항으로는 탐색 속도, 탐색공간의 크기, 경로의 최적여부(optimality), 동적 환경에 대응 가능여부, 경로의 질(quality) 등이 있다[5]. 기존에 많이 사용되는 게임 공간 표현 방식인 균일격자(RG), 웨이포인트 그래프(WG), 네비게이션 메쉬(NM)와 최근 사용되기 시작한 가시성그래프(VG), 본 논문에서 제안한 일반화 가시성 그래프(GVG)에 A* 탐색 알고리즘을 적용한 결과를 위에서 언급한 고려 사항들에 대하여 표 1에 비교하였다. 노드와 링크의 수가 많아지면 탐색 시간이 오래 걸리므로 탐색시간은 탐색 공간의 크기로 대표하였으며 최적성은 최단 거리 경로를 의미하였다. 동적 환경에 대응 가능여부 탐색공간을 생성하는 방식이 자동으로 가능한가에 대한 여부이다. 경로의 질을 결정하는 요인으로는 부드러운 턴(smooth turns)을 하는가와 장애물로부터의 여유공간을 두고 움직이는 가이다. 이 결과를 뒷받침하기 위하여 RG, VG, GVG에 대한 시뮬레이션 결과를 그림 9에 보여준다.

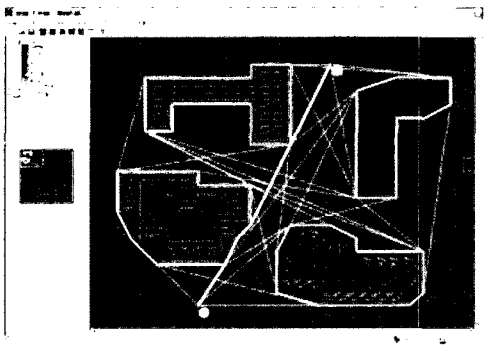
어느 정도의 현실감을 위해서는 균일격자나 네비게이션 메쉬의 경우, 탐색공간의 크기가 매우 커지며 작은 격자나 다각형들로 인해 발견된 경로가 지그재그 형으로 꺾임이 많아 자연스럽지 못하다. 웨이포인트 그래프의 경우에는 레벨 디자이너가 중요하다고 판단하는 장소에만 웨이포인트를 설정하므로 탐색공간은 작다. 하지

표 1 여러 탐색공간 표현 방식의 비교

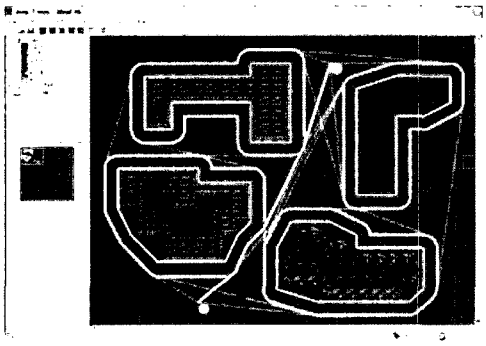
	RG	WG	NM	VG	GVG
size	large	small	large	small	small
Optimal?	no	no	no	yes	near
Dynamic?	no	no	yes	yes	yes
turns	sharp	sharp	sharp	sharp	smooth
clearance	yes	yes	yes	no	yes



(a) 균일격자 표현 방식에서 찾은 경로



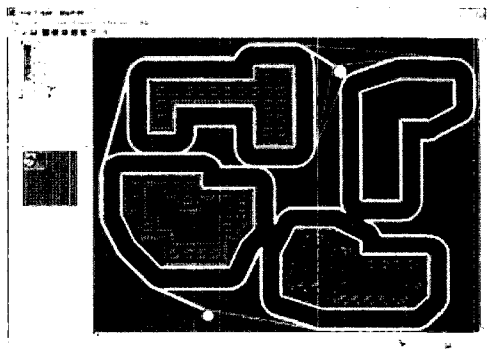
(b) Vgraph 표현 방식에서 찾은 경로



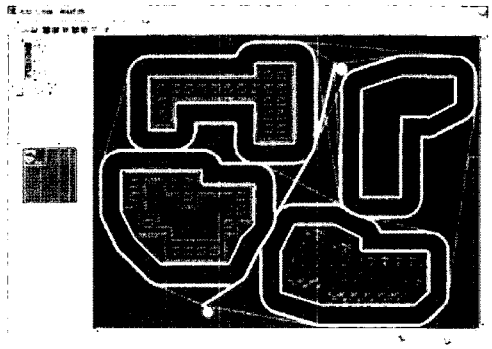
(c) GVgraph 표현 방식에서 찾은 경로

그림 9 여러 탐색공간 방법에 A*를 적용한 결과

만 최적의 경로를 찾지 못하며 사람이 관여해야 하므로 게임 환경의 조그만 변화에도 자동으로 대응할 수 없는 것이 큰 단점이다. 네비게이션 메쉬의 경우에도 작은 다각형들을 저장하는 데에 많은 메모리 공간이 소요되며 자동으로 공간을 나누는 데에 설계자의 도움이 필요하다. Vgraph는 탐색공간도 적고 자동 생성이 가능하며 최적 경로를 찾는 장점들이 있지만 가장 큰 단점은 경로의 질에 있다. 장애물의 벽을 따라가는 경로와 턴이 부드럽지 않다는 것이 그 이유이다. 본 논문에서 제안한



(a) 허용 여유공간이 커짐에 따라 우회 경로 생성



(b) Voronoi 링크를 이용하여 근사 최단 경로 생성

그림 10 우회 경로와 Voronoi 링크에 의한 근사최적 경로

방식인 GVgraph는 표 1에서 보는 바와 같이 기존의 Vgraph의 장점은 최대한 유지하며 단점을 보완하였음을 알 수 있다.

5.2 허용 여유공간에 따른 경로의 변화 및 해결

본 논문에서는 허용 여유공간을 솔리드 오프세팅 연산에 의해 자동으로 처리하므로 허용 여유공간을 인자화할 수 있다. 허용 여유공간이 커지면 경로의 안정성은 높아지는 장점이 있으나 그림 10(a)와 같이 원래 자유공간이던 지역이 장애물로 분류되어 멀리 우회하는 경로를 찾는 결과를 낳기도 한다. Voronoi 링크를 이용하면 부풀려진 장애물로 인해 자유공간이 막히는 경우, 경로가 우회하는 것을 막을 수 있다. Voronoi링크는 두 장애물에 이르는 거리가 같은 점들의 집합, 다시 말하면 두 장애물로부터 가장 멀리 형성되는 경로이다. 두 장애물 사이의 최소 거리가 움직이는 캐릭터의 지름보다 크다면 우회하는 경로를 택하지 않고 Voronoi 링크를 생성, 이용함으로써 경로가 근사 최적(near optimal)을 유지하도록 할 수 있다(그림 10(b)).

5.3 제안된 GVgraph 생성 알고리즘의 실행시간 개선에 관한 결과

제안과 같이 경로찾기 할 때마다 매번 전체 맵을 계

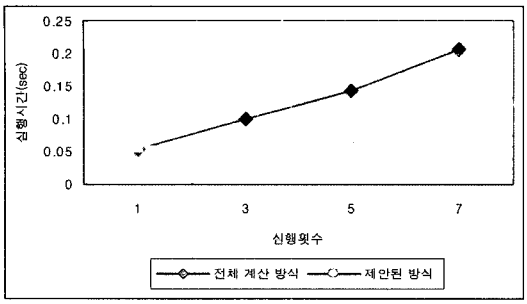


그림 11 전체 GVgraph 생성 방식 대비 제안된 방식의 실행 시간 비교

산하지 않고 주어진 출발점과 목표점을 효율적인 알고리즘에 의해 삽입하는 방법의 이론적인 우월성은 4장에서 시간복잡도에 대한 분석으로 확인한 바 있다. 이에 대해 게임을 진행하면서 실제로 얻을 수 있는 정량적인 효과를 시뮬레이션을 통해 확인하였다. 시뮬레이션 방법은 한 게임 환경에 대해 NPC가 이동할 필요가 있을 때마다 기존의 방식대로 GVgraph를 생성하여 경로찾기 하는 경우와 정적인 장애물에 대해 일단 맵을 만들어 놓고 NPC가 이동할 때마다 출발점과 목표점을 삽입하며 경로찾기 하는 경우의 실행 시간을 비교하였다. 그림 11의 결과를 보면 1회 실행 시에는 출발점, 목표점을 포함하여 GVgraph를 생성하는 것이 약간 더 효율적인 것을 볼 수 있는데 이는 제안된 방식이 전체 생성 과정을 둘로 나누어 실행하기 때문에 생긴 오버헤드에 기인한 것이다. 하지만 실행 횟수가 증가함에 따라 실행시간의 개선도가 점점 좋아지는 것을 알 수 있다. 보통 컴퓨터 게임에서 NPC가 동일한 환경에서 다양한 위치 이동을 하게 되므로 대부분의 경우에 제안된 방식의 성능이 좋을 것임을 예상할 수 있다.

6. 결 론

경로찾기에서 상태공간을 어떻게 표현할 것인가를 결정하는 일은 탐색 방법을 선택하는 일만큼 중요하다. 본 논문에서는 로보틱스 분야의 잘 알려진 경로 찾기 방식 중 하나인 Vgraph와 유사하게 최소의 탐색공간을 생성하는 GVgraph로 표현할 것을 제안하였다. Vgraph를 이용하면 최소의 탐색공간으로 빠른 탐색을 보장하지만 찾아진 경로가 장애물의 벽을 따라 간다거나 급격한 턴이 있는 등의 단점이 있다. GVgraph는 부풀린 장애물에 Vgraph를 생성하도록 수정한 것인데 이를 이용해 탐색공간을 표현하면 기존의 장점을 최대한 유지하면서 단점을 해결할 수 있다는 것이 본 논문의 핵심 내용이었다. GVgraph를 생성하는 과정과 함께 이를 이용해 효율적으로 경로찾기를 할 수 있는 알고리즘도 소개하였다. 실험결과를 통해 GVgraph는 작은 탐색공간을 생성하여 빠른 탐색을 하면서도 이를 이용해 찾은 경로는 근사 최적인 동시에 장애물에서 필요한 만큼 떨어져서 급격한 턴 없이 생성되어 이동 과정이 자연스럽게 보이는 것을 확인할 수 있었다.

GVgraph를 효과적으로 생성하는 일은 탐색 시간을 개선하기 위한 필수 전제 조건이다. 본 논문에서는 Vgraph를 생성하는 RPS 알고리즘을 일반화 다각형 환경으로 확장하는 방법을 소개하였는데 RPS 알고리즘보다 시간복잡도 면에서 더 효율적인 방식이 여러 가지가 있다 [18]. 대부분 종점을 특정한 방식으로 정렬한다거나 분할하는 방법으로 성능을 향상시키는데 원의 호를 포함하게 되면 이 방법들이 바로 적용되는 것이 불가능하거나 어렵다. 그러므로 GVgraph를 효율적으로 생성하기 위한 새로운 방법을 연구하거나 기존의 Vgraph방법을 확장하는 방법에 대한 연구가 필요하다. 또한 허용 여유공간을 인자화 하고 인자의 변화에 따라 막히는 통로에 대한 처리를 자동화 하기 위해서는 [9]에서 소개한 VV-complex를 접목시키는 것이 한가지 방법이다. VV-complex는 Voronoi 다이어그램과 Vgraph를 합쳐 만든 구조이며 5장에서 이미 Voronoi링크를 이용해 막힌 통로를 이용하는 것을 보였었는데 이를 확장한 개념이라고 할 수 있다. VV-complex를 이용하면 필요한 여유공간이 확보되지 못하는 영역에 가능한 최대의 여유공간을 보장하는 경로를 찾을 수 있을 것이며 앞으로 VV-complex를 게임 공간을 표현하는 데에 적용하기 위한 연구가 필요하다.

참 고 문 헌

[1] Stentz, A., "The Focused D* Algorithm for Real-Time Replanning," in Proceedings of IJCAI'95, pp. 1652-1659, 1995.

[2] Yap, P., "Grid-based Path-finding," Lecture notes in Artificial Intelligence, Vol.2338, pp.44-55, 2002.

[3] Jung, D., Kim, H., Kim, J., Um K., Cho, H., "Efficient Path Finding in 3D Games by using Visibility Tests with the A* Algorithm," In: Proceedings of the International Conference Artificial Intelligence and Soft Computing, Spain, 2004.

[4] Botea, A., Muller, M., Schaeffer, J., "Near optimal hierarchical path-finding. Journal of game development," Vol.1(1), pp.1-22, 2004.

[5] Tozour, P., "Search Space Representations," In: Rabin, S. (eds.): AI Game Programming Wisdom 2. Charles Rive Media, pp.85-102, 2004.

[6] Stout, B., "Smart moves: Intelligent path-finding," Game developer magazine October, pp.28-35, 1996.

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

[8] Young, T., "Expanded Geometry for Points-of-Visibility Pathfinding," In: Deloura, M. (eds.): Game Programming Gems 2, Charles Rive Media, 317-323, 2001.
[9] Wein, R., Van der Berg, J.P., Halperin, D., "The Visibility-Voronoi Complex and Its Applications," In: Proc. European Workshop on Computational Geometry, pp.151-154, 2005.
[10] Laumond, J. P., "Obstacle growing in a nonpolygonal world," Inform. Proc. Letter, Vol.25(1), pp.41-50, 1987.
[11] Latombe, J. C., Robot Motion Planning, Kluwer Academic Publishers, 1991.
[12] Pinter, M., "Towards more realistic pathfinding," Game Developer Magazine April, 2001.
[13] Rabin, S., "A* speed optimizations and A* Aesthetic Optimizations," In: Deloura, M. (eds.): Game Programming Gems. Charles Rive Media, pp.264-287, 2000.
[14] Rossignac, J. and Requicha, A. G., "Offsetting operations in solid modeling," Computer Aided Geometric Design, Vol.3, pp.129-148, 1986.
[15] Luger, G. and Stubblefield, W: Artificial Intelligence:Structures and Strategies for Complex Problem Solving. 3rd edn. Addison Wesley Longman Inc., 1998.
[16] Lee, D.T., Proximity and Reachability in the Plane, Ph.D. Dissertation, University of Illinoi, Urbana, IL, 1978.
[17] de Berg, M., van Kreveld, M., Overmas, M., Schwarzkorf, O., Computational Geometry-Algorithms and Applications. 2nd edn. Springer-Verlag, Berlin Heidelberg New York, 2000.
[18] Kitzinger, J., The Visibility Graph Among Polygonal Obstacles: a Comparison of Algorithms, M.S. Dissertation, University of New Mexico, 2003.

유 견 아
1986년 서울대학교 공과대학 제어계측공학과 학사. 1988년 서울대학교 공과대학 제어계측공학과 석사. 1995년 미국 USC Computer Science 박사. 1996년~현재 덕성여자대학교 컴퓨터공학부 교수. 관심분야는 인공지능, 기계학습, 로봇 알고리즘, 연산 기하학