컴퓨터 게임 환경에서 일반화 가시성 그래프를 이용한 경로찾기†

유견아*, 전현주**

Path-finding by using generalized visibility graphs in computer game environments

Kyeonah Yu, Hyunjoo Jeon

Abstract

In state-of-the-art games, characters can move in a goal-directed manner so that they can move to the goal position without colliding obstacles. Many path-finding methods have been proposed and implemented for these characters and most of them use the A* search algorithm. When the map is represented with a regular grid of squares or a navigation mesh, it often takes a long time for the A* to search the state space because the number of cells used in the grid or the mesh increases for higher resolution. Moreover the A* search on the grid often causes a zigzag effect, which is not optimal and realistic. In this paper we propose to use visibility graphs to improve the search time by reducing the search space and to find the optimal path. We also propose a method of taking into account the size of moving characters in the phase of planning to prevent them from colliding with obstacles as they move. Simulation results show that the proposed method performs better than the grid-based A* algorithm in terms of the search time and space and that the resulting paths are more realistic.

Key Words: Game, path-finding, visibility graph, grid-based A* algorithm, configuration space

† 본 연구는 2004학년도 덕성여자대학교 자연과학연구소 연구비 지원으로 이루어졌음.

* 덕성여자대학교 컴퓨터공학부 교수

** 덕성여자대학교 컴퓨터공학부 석사과정

1. 서론

최근 컴퓨터 게임에서는 각자의 목표를 갖고 움직이며 장애물과 부딪히지 않고 원하는 목표지점으로 이동할 수 있는 사람, 동물, 운송수단 등의 캐릭터 사용이 많아지고 있다. 캐릭터들이 스스로 길을 찾아가는 경로찾기(path-finding) 문제는 컴퓨터 게임에서 기초적인 구성요소 중의 하나라고 할 수 있다. 최근 컴퓨터 게임에서 가장 많이 쓰이는 경로찾기 알고리즘은 인공지능 분야에서 소개된 A* 탐색 알고리즘이다. 그리드 기반의 게임 환경에서 A* 알고리즘은 그리드 타일들을 노드로 하고 이웃하는 타일로의 이동을 링크로 하는 트리를 탐색하여 빠른 길을 찾아내는데 이 방법의 단점은 현실감을 위해 그리드를 세분화하면 노드 수가 증가하여 탐색공간의 크기가 커지고 탐색 시간이 증가할 뿐 아니라 구해진 경로가 지그재그 스타일로 캐릭터의 잦은 방향 전환이 필요하다는 것이다.

가시성 그래프(visibility graph, Vgraph)는 장애물의 볼록 정점을 노드로 하고 서로 보이는, 즉 장애물과 교차하지 않는 두 정점 사이의 연결선을 링크로 하는 그래프로 탐색공간의 크기를 줄일 수 있는 방안으로 여러 문헌[1,2,3]에서 등장하였다. Vgraph는 로보틱스 분야에서 로봇의 경로 계획을 위해 널리 이용되는 방법으로 최적의 답을 찾아낸다는 점 뿐 아니라 다양한 구현 알고리즘이 개발되어 있어 컴퓨터 게임에의 응용이 기대되고 있다. 그러나 장애물의 정점을 지나는 경로를 계획하기 때문에 부피가 있는 캐릭터의 처리를 어떻게 하는가 하는 점뿐만 아니라 Vgraph 장애물의 충돌 모델이 캐릭터가 회전하지 않는다는 가정을 해야 하는 점 등, 해결되지 않은 문제점들로 인하여 활발한 응용은 아직 안 되고 있는 실정이다 [1]. 본 논문에서는 가시성 그래프를 컴퓨터 게임에서의 경로 찾기에 응용하기 위해 이 문제점들을 해결하는 방안을 제시한다. 부피있는 캐릭터의 운동을 계획하기 위해 캐릭터를 점으로 환산했을 때 확장된 장애물을 나타내는 형상공간(configuration space, C-공간)을 소개한다. 움직이는 캐릭터의 회전운동을 고려하여 캐릭터를 디스크 형태의 모양으로 가정했을 때 장애물은 경계선을 디스크의 반지름만큼 확장하는 솔리드 오프세팅에 의해 구해진다. 이와 같이 변형된 게임 환경은 다각형 환경이 아닌 원의 호와 선분이 섞인 일반화 다각형(generalized polygon) 환경이므로 다각형 환경에서 제안된 Vgraph 알고리즘을 그대로 사용하지 못한다. 그러므로 기존의 Vgraph를 일반화 다각형으로 확장한 일반화 가시성 그래프 (GVgraph) 알고리즘을 정의하며 비다각형 환경에서도 시간복잡도가 나빠지지 않음을 보인다. GVgraph에 의한 탐색을 그리드 기반의 A* 알고리즘과 탐색 노드수, 탐색 시간 등에서 정량적으로 비교하고 생성된 경로들의 질적인 비교를 위해 다양한 형태로 시뮬레이션한다.

2. 관련 연구

A* 알고리즘은 여러 분야에서 다양한 방식으로 응용되고 있는 인공지능 분야의 대표적인 탐색 알고리즘이다. A* 알고리즘의 컴퓨터 게임 응용을 소개한 것은 [4]에서이며 그 이후 경로찾기를 필요로 하는 거의 대부분의 게임들에서 A*를 그대로 혹은 변형하여 사용하고 있다. A* 알고리즘은 시작지점에서 목표지점까지 이르는데 드는 비용을 최소화하는 경로를 찾는데 이를 위해 현재 위치에서 목표 위치까지 소요될 비용을 추정한 값(이를 휴리스틱이라고 하며 실제 소요될 값보다 작게 추정함을 원칙으로 한다.)을 실제 시작지점에서 현재 노드까지 소요된 비용에 합하여 노드의 값으로 이용한다. 컴퓨터 게임에서 그리드 기반 경로찾기를 위하여 사용되는 A* 알고리즘의 기본적인 절차는 다음과 같다.

1단계: 시작 노드를 open 리스트에 넣는다.

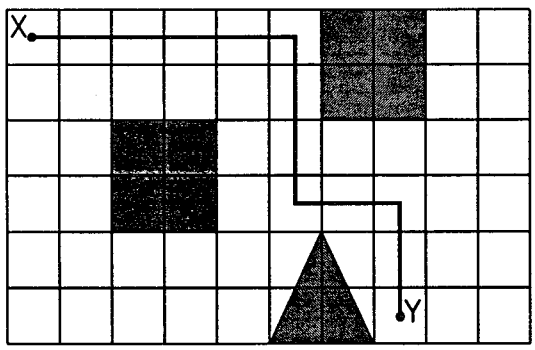
2단계: open 리스트에서 노드값이 가장 작은 노드를 선택한다.

3단계: 2단계에서 선택된 노드가 목표이면 '성공'을 리턴하고 끝낸다.

4단계: 아니면 2단계에서 선택된 노드의 이웃 노드들의 노드값을 계산하고 open 리스트에 넣는다. (이 과정에서는 이웃 노드로 이동 가능한지, 이미 갔던 노드는 아닌지 등, 고려해야 할 사항들이 있으나 자세한 내용은 생략한다.)

5단계: open 리스트에 노드가 없으면 '실패'를 리턴하고 끝내고 아니면 2단계로 되돌아간다.

자주 사용되는 휴리스틱으로는 두 노드 사이의 직선거리가 있는데 이를 적용하여 찾은 경로의 예가 <그림 1>에 있다. 이와 같은 단순한 거리 휴리스틱 이외에 다양한 정보를 수치화하여 노드 값으로 사용함으로서 게임의 특성에 맞는 경로를 찾는데 사용하기도 한다. 예를 들어, [3]에서는 경사도를 가중치로 두어 평지에 있는 경로를 선호할 수 있도록 하였으며 [2]에서는 각도 가중치를 두어 급격한 턴을 피하도록 유도하였다.



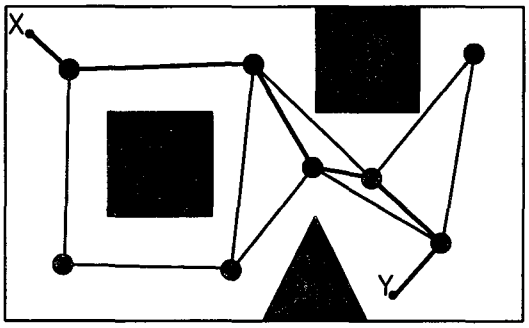
<그림 1> 그리드 기반에서 길찾기

그리드 기반의 게임 환경에서 A* 알고리즘을 이용하면 그리드 타일 수에 따라 탐색공간이 커지고 탐색 시간도 기하급수적으로 증가하는 단점이 있다. 뿐만 아니라 구해진 경로도 90도 턴을 수시로 요구하여 게임의 현실감을 떨어뜨리는 요인이 되고 있다. 이를 해결하기 위하여 연구들도 진행되고 있는데, 크게 A* 알고리즘의 성능 자체를 향상시키는데 중점을 둔 연구와 게임 공간을 나누는 방법을 달리하여 탐색공간의 크기를 줄이는 데에 중점을 둔 연구로 나누어 볼 수 있다.

제한된 메모리에서 A* 알고리즘을 실행하기 위해 반복적 깊이 증가 A*(iterative deepening A*, IDA*) 방법을 사용한다거나 open 리스트의 정렬을 효율적으로 하는 방법[4] 등이 전자에 해당한다. 또한 전체 그리드 타일을 모두 대상으로 하지 않고 일정 구역으로 나누어 구역별로 A*를 적용하고 다시 계층적으로 확대하는 방법도 연구되었으며 [5], 구해진 경로의 급격한 턴을 최소화하기 위해 각도 가중치를 휴리스틱에 반영하고 후처리과정을 통해 곡선화한다거나[2] 그리드 타일의 모양을 다양화하는 방법도 제안되었다[6].

탐색공간을 설정하는 연구로는 모서리 그래프(corner graph), 웨이포인트 그래프(waypoint graph), 네비게이션 메쉬(navigation mesh, NavMesh) 등을 예로 들 수 있다[7].

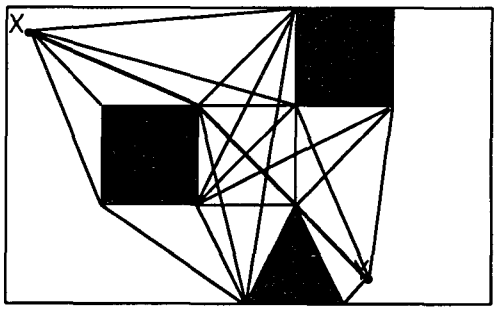
모서리그래프는 장애물의 모서리에 웨이포인트를 놓고 그 사이를 연결한 링크들로 이루어진 그래프이다. 즉, 웨이포인트를 노드로 하고 노드를 잇는 선분이 자유공간에 놓여있으면 링크를 추가한다. 웨이포인트 그래프는 모서리 그래프와 유사하나 웨이포인트를 장애물의 모서리에 두는 것이 아니라 모서리 사이의 임의의 중간지점에 두는 방식이다. 이 방식은 게임 설계자가 적당한 장소에 임의로 웨이포인트를 두기 때문에 특히 3차원 게임에서도 많이 사용되고 있다. 그러나 웨이포인트 그래프 방식은 <그림 2>와 같이 최적의 경로를 찾아내지는 못하며 설계자가 임의로 웨이포인트를 두기 때문에 동적으로 변하는 환경에서 사용하기 어렵다는 것이 단점이다.



<그림 2> 웨이포인트 그래프에서 경로찾기

NavMesh는 자유공간을 여러 개의 작은 볼록다각형으로 나누는 방식이다. 캐릭터는 볼록다각형 안에서는 자유롭게 움직일 수 있으며 이 볼록다각형들이 노드를 형성한다. 이웃한 다각형 사이의 경계가 링크가 된다. 이 방식의 장점으로는 여러 모양과 능력의 캐릭터에게 최상의 경로를 찾아 줄 수 있다는 것과 자동으로 NavMesh를 생성할 수 있다는 것이다. 하지만 복잡한 지형을 나타내기 위해서는 많은 수의 볼록다각형으로 맵을 표현해야 하므로 많은 저장 공간을 필요로 하고 A* 탐색 알고리즘의 상태공간이 커져서 탐색시간이 많이 걸린다는 것은 그리드 기반의 경우와 유사하다.

Vgraph는 컴퓨터 기하학 분야나 로보틱스 분야에서 널리 알려지고 많이 응용되는 그래프로서 시작점과 끝점, 장애물의 모서리가 노드를 형성하고 서로 보이는(visible) 노드 사이에 링크가 형성된다(그림 3). 즉, 노드 사이를 잇는 선분이 장애물과 만나지 않으면 링크가 된다. 앞에서 설명한 모서리그래프와 유사한 듯이 보이지만 모서리그래프는 설계자가 수동으로 웨이포인트를 설정하는 hand-placed 웨이포인트 방식이고 Vgraph는 자동으로 노드가 설정되고 그래프가 만들어지기 때문에 두 가지는 근본적으로 다르다고 할 수 있다. Vgraph는 A* 알고리즘과 결합하여 최상의 경로를 찾아낸다는 장점이 있으며 자동으로 그래프를 생성하므로 게임 환경이 동적으로 변하는 경우에도 일관적으로 적용할 수 있다는 장점이 있다. 하지만 노드간의 단순 연결 관계만을 고려하여 캐릭터가 이동하면서 장애물에 충돌할 위험성이 지적되고 있다[8].



<그림 3> Vgraph를 이용한 경로찾기

3. 일반화 가시성 그래프 (Generalized Visibility Graph, GVgraph)

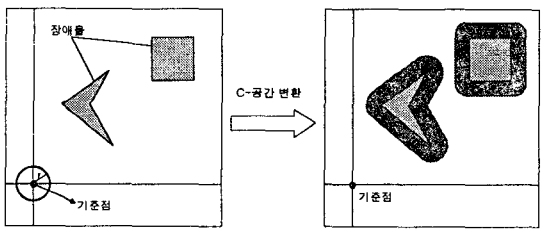
앞에서 소개한 바와 같이 가시성 그래프는 이미 컴퓨터 게임에서 경로를 찾기 위해 응용된 바 있다. 하지만 장애물들이 부피를 고려하지 않고 생성된 그래프 위에서 노드간에 연결된 링크를 따라 경로가 생기기 때문에 캐릭터가 이동하다 장애물의 모서리에 부딪히는 것이 단점으로 지적되었다. 이를 해결하기 위해 기존의 게임에서는 가시성 그래프의 정점을 임의로 장애물의 꼭지점에서 약간 떨어진 위치로 조정하여 가시성 그래프를 생성하였다[4][8]. 본 연구에서는 움직이는 캐릭터가 직선 이동뿐 아니라 회전 운동을 하는 경우에도 장애물과 부딪히는 것을 피하기 위해 캐릭터를 반경 r의 디스크로 모델링한 후 형상공간(C-공간) 문제로 변환하여 경로를 계획한다. C-공간은 로봇 운동계획에서 로봇을 점으로 치환할 때, 환경을 구성하는 장애물들을 그에 맞게 매핑한 공간을 말한다. 점의 운동을 계획하는 일은 부피가 있는 물체의 운동을 계획하는 것보다 명확하게 나타낼 수 있을 뿐 아니라 컴퓨터 기하학의 여러 알고리즘을 이용하여 체

계적인 접근을 할 수 있다는 장점이 있다[9].

3.1 C-공간 문제로의 변환

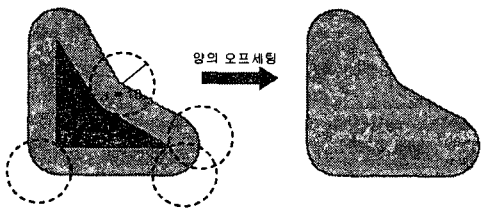
<그림 4>과 같이 원형 물체가 다각형 모양의 장애물들 사이를 움직이는 문제는 (그림 4 좌측) 점이 확장된 C-공간 장애물 사이를 움직이는 문제(그림 4 우측)로 변환하여 생각할 수 있다[10]. 다각형 환경이 선분과 원의 호를 포함하는 일반화 다각형 환경으로 확장되었음을 알 수 있다. 이와 같이 원래의 문제를 C-공간 문제로 변환하기 위해서는 다각형을 원형 장애물의 반지름만큼 확장하는 것이 필요하다. 이와 같은 확장 과정은 Minkowski 합과 차 연산의 특별한 경우로 정의되는 양의 솔리드 오프세팅(positive solid offsetting) 연산을 적용하면 된다. 원형의 반지름을 r이라고 하고 다각형을 set S로 표시하면 S의 오프셋은 식 (1)과 같이 정의되며 $\uparrow^*$을 양의 오프세팅 연산자라고 한다[11].

$$S \uparrow {}^{*}r = \{p : \exists q \in S, \|p - q\| \le r\}$$



<그림 4> C-공간 문제로의 변환

식 (1)과 같이 정의된 양의 솔리드 오프세팅은 다음과 같이 구현된다. 주어진 n각 다각형(n-sided polygon)을 시계반대 방향의 정점 [v_1,v_2, …, v_n]의 순서집합 (ordered list of vertices)으로 나타내면 연속된 정점의 쌍 [v_i, v_{i+1}]은 선분을 구성한다. n_{i-}와 n_{i+}를 각각 선분 [v_{i-1}, v_i]와 [v_i, v_{i+1}]에 대한 outward normal이라고 할 때, angle (n_{i-}, n_{i+})≥0 인 경우를 볼록 정점(convex vertex) 과 그렇지 않은 경우를 오목 정점(concave 또는 reflex vertex)라고 한다. 그림 3.2와 같이 볼록 정점의 경우에는 호를 형성하고 오목한 경우에는 정점으로 변환된다. 본 논문에서는 양의 솔리드 오프세팅에 의해 생성된 일반화 다각형을 표현하는 방식으로 <그림 5>에서 제시한 바와 같이 원래의 다각형의 정점을 호의 시작점, 중점, 끝점(v_{i-}, v_i, v_{i+})의 3점으로 구성한 순서 집합을 택한다[12]. 오목 정점의 경우는 v_{i-}-= v_{i+} 인 특수한 경우로 취급된다. 이와 같이 일반화 다각형을 표현하는 자료구조의 장점은 호와 선분 정점을 일관되게 표현할 수 있어 동질적인 데이터 표현을 제공한다는 것이다.

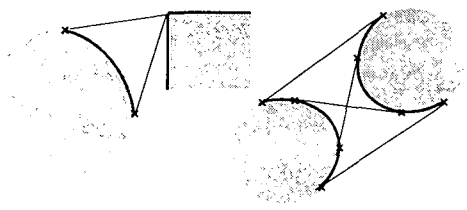


List of vertices	Ordered list of arc
(0,0)	(-10,0) (0,0) (0, -10)
(50,0)	(50,-10) (50,0) (8.32, 55.55)
(20,20)	(20.1,37.9),(20,20) (20.1,37.9)
(0,50)	(8.3, 55.6) (0,50) (-10, 50)
(0,0)	(-10,0) (0,0) (0, -10)

<그림 5> 솔리드 오프세팅

3.2 일반화 가시성 그래프 (GVgraph)

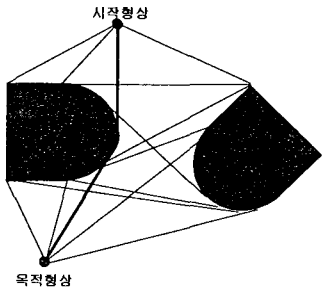
일반화 다각형은 원의 호와 다각형으로 이루어진 도형군이다. GVgraph는 C-공간 장애물이 일반화 다각형으로 이루어진 경우에 생성된다. 이때 장애물을 이루고 있는 선분의 끝점뿐만 아니라 호의 끝점 또한 정점(vertex)이라 한다. 일반화 다각형에서는 원호를 포함하고 있기 때문에 정점 사이의 링크를 생성하는 과정이 Vgraph의 경우와 달리 다음 3가지 경우를 고려해 주어야 한다: ① 정점과 정점 ② 호와 정점 ③ 호와 호.



(a) 정점과 호 (b) 호와 호
<그림 6> 가상점이 생기는 경우

먼저 ①의 경우일 때 Vgarph와 마찬가지로 노드에는 시작형상, 목적형상, 장애물의 정점을 포함한다. 그리고 정점을 연결한 선분 중에 링크가 장애물과 교차하지 않는 선분이 링크가 된다. 호의 경우, 선분과는 달리 호의 끝점들에 대해 다른 정점이나 호와의 가시(visible) 여부를 결정하는 것의 의미가 없다. 그러므로 ②,③의 경우에는 호의 가시성을 따져주기 위해 가상점(fictitious point)을 생성한다[13]. ②의 경우 노드로 정해진 정점과 호가 접하는 점이 가상점이다. <그림 6>(a)에서처럼 정점에서 하나의 호에 최대 2개의 가상점이 생성된다. ③의 경우에는 하나의 호에 대해 외접점과 내접점이 최대 2개씩 그러므로 총 4개의 가상점이 생길 수 있다(그림 6(b)). 즉, GVgraph의 노드에는 가상점이 추가되고 링크에는 가상점을 생성한 선분과 호 위에 추가된 두 개의 이웃한 가상점을 연결한 부분이 추가된다. 이렇게 생성된 GVgraph의 예는 <그림 7>과 같다. 일단 그래프가 생성되면, 그래프에 탐색 알고리즘들을 적용해 최단 경로를 찾을 수 있다. 탐색 알고리즘으로는 A* 알고리즘이 사용되며 과소추정된 휴리스틱으로 그래프의 최단 경로를 찾는 것을 보장한다.

Vgraph를 형성하기 위한 장애물들의 정점의 수를 n이라 할 때, 임의의 두 개의 정점을 연결한 선분이 교차하는지의 여부를 판별해야 한다. n개의 정점 중에서 두 개의 정점을 선택하는 경우의 수는 ${}_nC_2$ 이다. 장애물을 구성하는 선분의 수는 장애물 정점의 수와 같으므로 선택된 정점을 연결한 선분이 장애물을 구성하는 선분과 교차하는지를 판별하기 위해서는 $O(n^3)$의 시간 복잡도가 요구된다.



<그림 7> Generalized Visibility Graph의 예

GVgraph에서는 장애물의 정점 및 호의 수를 n이라고 할 때, 정점과 정점, 정점과 호, 호화 호의 조합의 수는 모두 ${}_nC_2$ 이다. 호를 포함하는 경우, 호 위에 가상점을 생성하여 다른 장애물의 선분과 교차하는지 판별해야한다. 한 조합에서 가능한 최대 가상점의 수는 4이므로 이를 연결한 선분이 장애물을 구성하는 선분과 교차하는지를 판별하는 데에는 Vgraph의 경우와 마찬가지로 $O(n^3)$의 시간 복잡도가 요구된다. 이렇게 생긴 가상점은 다른 조합을 생성하는 데에는 고려되지 않아도 되므로 전체 시간 복잡도는 증가하지 않는다.

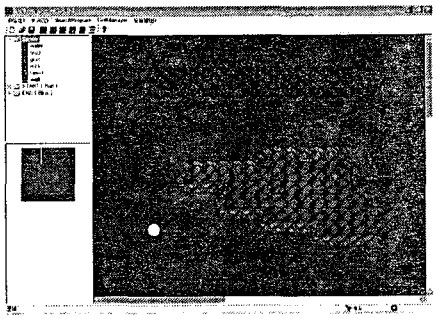
4. 구현 및 시뮬레이션

셀-기반 맵 에디터는 스타크래프트를 비롯한 많은 컴퓨터 게임에서 배경을 편집하는데 사용되고 있다. 이 에디터를 사용하면 화면 영역을 작은 정사각형의 세트, 즉 셀로 나누어 셀 단위로 지형을 구성하여 각 셀들이 A* 탐색의 상태 공간을 구성하게 된다. 이와 같이 셀단위로 경로를 탐색하면 그리드 기반 경로 찾기에서 나타났던 A* 알고리즘 적용의 문제점들이 그대로 나타나게 된다. 본 연구에서는 정사각형 셀들로 이루어진 128*128 크기의 영역에 16*16 사이즈의 도트로 이루어진 타일들을 정렬하여 지형을 구성한 셀 기반 게임 환

경에서 기존의 그리드 방식으로 A*를 적용해 보고 GVgraph를 이용하는 경우와의 차이를 비교하였다.

4.1 경로찾기 시뮬레이터의 구현

셀 기반 게임의 배경화면을 생성하기 위한 맵-에디터(그림 8)는 VC++을 이용해 Pentium Ⅳ에서 구현되었다. (맵-에디터는 스타크래프트와 같은 RPG(Role Playing Game)에서 게임 플레이어가 자신이 원하는 맵을 직접 구성할 수 있도록 사용자에게 제공되고 있다[14].) 3등분된 윈도우의 오른쪽 영역은 실제 사용자가 마우스를 이용해 제작한 맵의 모양을 보여주고, 왼쪽 상단은 현재 표현 가능한 지형들의 타일들을 표시한다. 왼쪽 하단은 전체적인 맵의 모양을 표시하는 미니-맵을 위치시키며 현재 표현되고 있는 부분은 사각형 틀 안에서 보여주고 있다. 사용자는 왼쪽 상단의 지형을 더블클릭 한 후, 오른쪽 윈도우에 드래그 함으로써 원하는 맵을 그릴 수 있다.



<그림 8> 맵-에디터의 구성

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 이루게 된다. 여기서 셀-테이블은 각 셀의 비트맵에 따른 고유 플래그를 저장하게 되고 플래그의 종류에 따라 이동가능 여부를 결정하게 된다. 본 연구에서는 forest, rock, wall, water, grass와 mud의 타일 중 forest, rock, wall, water와의 경계로 표시되는 영역을 장애물 영역으로 간주하고 이동할 수 없게 제작하였다.

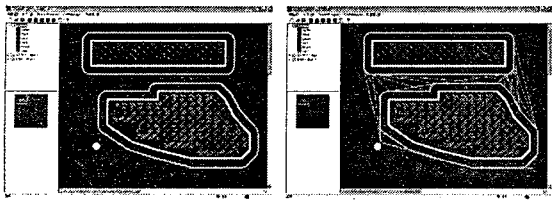
4.2 GVgraph의 구현

본 논문에서는 일반화 다각형의 동질적인 데이터 표현을 제공하기 위해 호와 정점을 도형의 코너(corner)라고 정의하고 다각형을 정점의 리스트로 표현하는 것과 마찬가지로 일반화 다각형을 코너 리스트로 표현한다. 이러한 자료구조로 표현되면 장애물이 n개의 코너로 이루어져 있는 경우, GVgraph는 ${}_nC_2$ 정점쌍에 대해 다음과 같은 방법으로 구현된다.

① **정점과 정점** : 선택된 모서리가 둘다 정점인 경우 리스트의 첫 번째 좌표를 연결하여 다른 선분과의 교차여부를 판별해 링크를 구한다. 이때 Vgraph에서는 고려하지 않았던 직선과 호의 교차여부를 판별하여야 한다. 이를 위해 [15]에서 소개된 일반화 다각형을 위해 수정된 plane-sweep 알고리즘을 이용하여 판별할 수 있다.

② **호와 정점** : 정점과 호의 접선을 이용해 가상점을 구할 수 있는데, 이를 그림 3.2 리스트의 마지막 모서리에 대하여 구현한다면 원의 중점이 (0, 50)이므로 원의 방정식 $x^2+(y-50)^2=10^2$이 된다. 그러면 정점을 지나고 중심에서 반지름인 10만큼의 거리를 갖는 직선의 식을 구하여 가상점의 후보를 구할 수 있다. 후보 가상점 중 호 위의 점은 노드에 추가된다.

③ **호와 호** : 호와 호의 접선을 구하기 위해서는 호에 대한 원의 방정식을 ②경우와 같이 구하고 두 원의 식에 모두 접하는 직선의 식을 구한다. 구해진 식을 이용해 접선을 구한 후 가상점의 후보를 구할 수 있다. ②의 경우에서와 같이 후보 가상점 중 호 위의 점이 노드에 추가된다.



(a) C-공간 장애물 (b) GVgraph

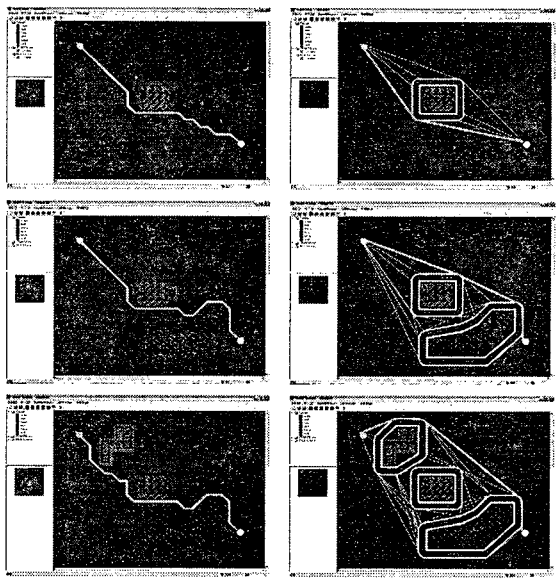
<그림 9> C-공간 장애물로의 변환과 생성된 GVgraph

<그림 9>는 <그림 8>의 게임 배경에 대해 위의 방법으로 생성된 GVgraph를 나타낸다. 배경에 있는 장애물들을 둘러싼, 선분과 호로 이루어진 일반화 다각형이 솔리드 오프세팅에 의해 계산된 C-공간 장애물이며, 이들 사이에 연결선들은 GVgraph의 링크를 형성한다. 이렇게 생성된 GVgraph에서 유클리드 거리를 휴리스틱값으로 갖는 A* 알고리즘을 이용해 최단경로를 탐색한다.

4.3 시뮬레이션 결과

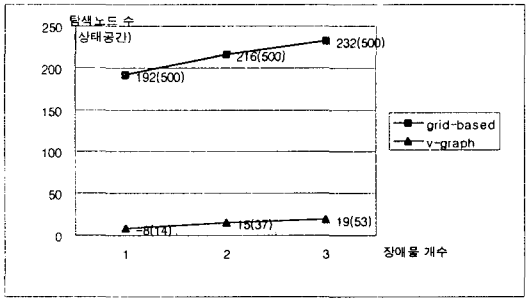
<그림 10>은 동일한 (시작점, 목적점) 쌍에 대해 장애물의 개수를 1씩 증가시켜 가며 각각 그리드 기반 A*를 적용한 경우와 일반화 가시화 그래프를 적용한 경우를 비교한 것이다. (b) GVgraph A*에서는 가느다란 선들이 GVgraph 링크들이며 그 중 굵게 표시된 선이 A* 알고리즘에 의해 찾아진 경로이다. 그리드 기반 A* 알고리즘에서는 상하좌우 운동 뿐 아니라 대각선 운동을 허용했음에도 불구하고 모든 경우에서 이동 경로의 꺾임이 많아 급격한 방향전환이 빈번하게 요구되고 전체적으로 자연스럽지 못함을 알 수 있다. 반면에 GVgraph에 의해 생성된 경로들은 가장 가까운 최단경로일 뿐 아니라 꺾임이 최소화되어 있음을 알 수 있다. 이와 같은 경로의 질적인 비교 이외에 탐색 노드수와 실행시간을 정량적으로 비교한 결과를 <그림 11>과 <그림 12>에서 보여준다.

<그림 11>에서는 장애물의 개수에 따라 두 가지 방식의 탐색 노드수를 비교하였으며 괄호안은 전체 상태 공간의 크기를 나타낸다. 그리드 기반 방식에서는 처음부터 일정한 크기의 그리드로 배경을 나누므로 장애물의 개수에 상관없이 500으로 일정한 반면 GVgraph 방식에서는 장애물의 개수가 늘어나면서 전체 상태공간의 크기도 늘어난다. 그렇지만 탐색 노드수는 그에 비례하여 증가하지는 않으므로 장애물의 개수가 증가함에 따라 두가지 방식에 의해 탐색되는 노드수는 점점 그 차이가 더 발어짐을 알 수 있다. <그림 12>의 실행시간 비교에서도 마찬가지로 장애물의 개수가 증가함에 따라 그 차이가 더 벌어진다. 전체적으로는 GVgraph를 이용하면 그리드 기반 A* 알고리즘을 이용하여 경로를 찾는 것에 비해 탐색 노드수는 6.6%에 불과하였으면 탐색 시간은 14.9배 빨라졌다.

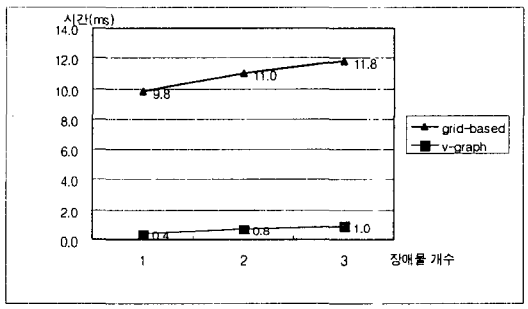


(a) 그리드 기반 A* (b) GVgraph A*

<그림 10> 그리드 기반 A* 알고리즘과 GVgraph에 의한 결과 비교



<그림 11> 탐색 노드수의 비교



<그림 12> 실행 시간의 비교

<표 1> 탐색 시간과 GVgraph 생성시간

그래프 종류	장애물 개수	탐색 시간(ms)	GVgraph 생성시간(ms)
그리드 기반	1	9.8	-
	2	11.0	-
	3	11.8	-
	평균	10.87	-
GVgraph	1	0.4	6.7
	2	0.8	17.7
	3	1.0	27.7
	평균	0.73	17.37

전체 실행 시간을 비교하기 위해서는 GVgraph를 생성하는데에 소요되는 시간을 고려해 주어야 한다. <표 1>에서 볼 수 있듯이 GVgraph를 생성하는데에 소요되는 시간은 탐색에 소요되는 시간에 비해 비교적 큰 17.37ms 정도이다. 하지만 GVgraph는 한번 생성되고 나면 다시 계산할 필요가 없기 때문에 현재로서는 큰 문제가 되지 않으나 게임 배경이 동적으로 변화하는 경우에는 문제가 될 수 있으므로 이런 경우를 위하여 GVgraph를 더욱 빠르게 생성하는 방법에 대한 연구나 그 밖의 동적인 환경에 대처하는 방법에 대한 연구가 필요하다.

5. 결론

본 논문에서는 로보틱스 분야에서 경로를 찾기 위한 알고리즘으로 알려진 가시성 그래프를 컴퓨터 게임에서 최적의 길을 찾는데 이용하는 방법을 제안하였다. 부피있는 캐릭터의 운동을 계획하기 위해 캐릭터를 점으로 환산하는 C-공간에서 표현하였다. 움직이는 캐릭터의 회전운동을 고려하여 캐릭터를 디스크 형태의 모양으로 가정했을 때 C-공간으로 변형된 게임 환경은 다각형 환경이 아닌 원의 호와 선분이 섞인 일반화 다각형이 되므로 기존의 Vgraph를 일반화 다각형으로 확장한 GVgraph 알고리즘을 정의하여 경로를 탐색하였다. GVgraph에 의한 찾아진 경로는 기존에 가장 많이 이용되는 그리드 기반의 A* 알고리즘에 의한 경로와 비교되었다. GVgraph에 의한 경로는 그리드 기반의 A* 알고리즘에 의한 경로에 비해 꺾임이 적고 자연스러웠으며 탐색 노드수와 탐색 시간 등에서 정량적으로 비교한 결과에서도 모든 비교 부문에서 우수하였다. 본 논문에서는 GVgraph를 생성하는데 걸리는 시간 복잡도를 기존의 Vgraph의 경우에 비해 나빠지지 않도록 알고리즘을 고안하였으나 Vgraph를 보다 효율적으로 생성하기 위해 제안된 알고리즘[16]들을 사용하여 더 효과적으로 GVgraph를 만들 수 있는가에 대한 연구가 계속되어야 한다. 이는 동적인 장애물이 존재하는 게임에서도 GVgraph를 효과적으로 사용할 수 있는 기초연구가 될 것이다.

참고문헌

[1] T. Young, "Expanded Geometry for Points-of-Visibility Pathfinding", Game Programming Gems 2, edited by M. Deloura, Charles Rive Media, 2001.

[2] M. Pinter, "Towards more realistic pathfinding", Game Developer Magazine, March, 2001.

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

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

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

[6] P. Yap, "Grid-based path-finding", Lecture notes in Artificial Intelligence, vol2338, pp44-55, 2002.

[7] AI Game Programming Wisdom 2, Edited by Steve Rabin, Charles Rive Media, 2004.

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

[9] Jean-Claude Latombe, Robot Motion Planning, Kluwer Academic Publishers, 1991.

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

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

[12] 유견아, 일반 다각형 환경에서 3차원 C-공간 장애물 구성에 관한 연구, 자연과학논문집, 덕성여대 자연과학연구소, pp85-97, Vol.8, 2002.

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

[14] http://blog.naver.com/harkon/, 최성용, 2001.

[15] 안진영, 유견아, 일반화 다각형을 위한 plane-sweep 알고리즘의 구현, 한국정보과학회 봄 학술발표논문집, pp.691-693, 2002.

[16] M. de Berg et. al., Computational Geometry: Algorithms and Appplications 2nd edition, Springer, pp 19-43, 1998.

주 작 성 자 : 유 견 아

논문투고일 : 2005. 06. 08

논문심사일 : 2005. 06. 10(1차), 2005. 06. 11(2차), 2005. 06. 22(3차)

심사판정일 : 2005. 06. 22

저자소개

유견아
1986 서울대학교 공과대학 제어계측공학과 학사
1988 서울대학교 공과대학 제어계측공학과 석사
1995 미국 USC Computer Science 박사
1996 ~ 현재 덕성여자대학교 컴퓨터공학부 교수
관심분야 : 인공지능, 로봇 알고리즘, 연산 기하학

전현주
2004 덕성여자대학교 지연과학대학 컴퓨터과학부 학사
2004 ~ 현재 덕성여자대학교 전산 및 정보통신대학원 석사과정
관심분야 : 인공지능, 게임 프로그래밍