가시성 그래프와 A* 알고리즘을 이용한 3D game에서의 효율적인 경로 탐색.

정동민, 김형일, 김준태, 엄기현, 조형제
동국대학교 컴퓨터공학과
e-mail:{mjung, hikim, jkim, khum, chohj}@dgu.edu

Efficient path finding in 3D game by using Visibility Graph and A* Algorithm.

Dongmin Jung, Hyoungil Kim,
Juntae Kim, Kyhyun Um, Hyungje Cho
Dept. of Computer Engineering, Dongguk University

요 약

본 논문에서는 Navigation Mesh로 이루어진 3D 게임에서 가시성 그래프(Visibility Graph)와 A* 알고리즘을 혼용한 효율적인 경로 탐색 방법을 제안한다. Navigation Mesh로 지형을 생성할 때 이동에 꼭 필요한 Mesh로만 최대한 단순하게 지형을 구성하는 경우에는 경로 탐색을 위하여 A* 알고리즘을 적용할 수 있으나, 일반적으로 세밀하게 구성된 Navigation Mesh에서 A* 알고리즘을 적용할 경우 탐색할 공간이 많아지기 때문에 경로 탐색이 매우 비효율적이 된다. 세밀하게 구성된 Navigation Mesh에서도 효율적인 탐색을 하기 위해서 본 논문에서는 가시성 그래프를 이용하여 탐색 공간을 줄이는 방법을 사용하였다. 장애물들의 정점을 찾아 반드시 통과하여야 하는 mesh 들을 선정하고 A*의 휴리스틱 함수를 이 mesh들을 지나가는 거리로 정의함으로써 기본적인 A* 알고리즘을 수행하는 것보다 탐색을 위하여 방문하는 mesh들의 수를 현저히 줄일 수 있었다.

1. 서론

3D 게임에서 현실적인 캐릭터 이동과 효율적인 경로 계획을 지원하기 위해서는 복잡한 알고리즘과 하드웨어 성능이 요구된다. 현실적인 캐릭터 이동을 위해서는 세밀하게 지형을 표현해야 하는데, 세밀한 지형 공간의 묘사는 메모리나 하드웨어에 상당한 부담을 주기 때문에 경로 탐색 시 게임 진행 속도를 저하하는 원인이 된다. 그래서 경로 탐색 속도를 향상시키기 위해 Navigation Mesh와 같은 방법[6]으로 최대한 지형 공간을 단순화시킨다.

Navigation Mesh는 3D 상태 공간을 2D 평면 공간으로 표현함으로써 3D 환경에서 효율적인 이동과 탐색이 이루어질 수 있도록 보장하며, 또한 탐색 속도 개선을 위해 캐릭터의 이동에 필요한 정보만으로 최대한 지형을 단순화시키기 때문에 빠른 탐색 속도를 보장할 수 있다. 하지만 지형을 단순화시켜 구성하기 때문에 캐릭터의 이동이 단순해져 비현실적일 수밖에 없다. 그래서 현실적인 캐릭터 이동을 위해 세밀하게 Navigation Mesh를 생성하더라도 기존의 빠른 탐색 속도를 보장할 수 있는 방식이 필요하다.

그래서 본 논문에서는 가시성 그래프와 A* 알고리즘을 혼용하는 방법을 통해 세밀한 Navigation Mesh로 이루어진 3D 게임에서 보다 효율적인 경로 탐색이 이루어지도록 제안하였다.

2. 가시성 그래프

가시성 그래프는 장애물이 존재하는 공간에서 로봇의 이동 경로를 결정하는 로봇 동작 계획에 사용되는 알고리즘[2]이다. 이러한 가시성 그래프는 다음과 같이 정의 내린다.

표 1. 가시성 그래프의 정의

Visibility Graph VG = (V, E)
V : 장애물의 정점(Vertex) E : 가시적(Visible)인 노드의 쌍 Weight of an edge(u, v) : u와 v의 기하학적인 거리

본 연구는 한국과학재단 특정기초연구(과제번호: R01-2002-000 -00298-0) 지원에 의해 수행되었음.

표 1의 가시성 그래프의 정의를 이용하면 그림 1과 같이 Navigation Mesh 상에서 가시성 그래프를 생성할 수 있다.

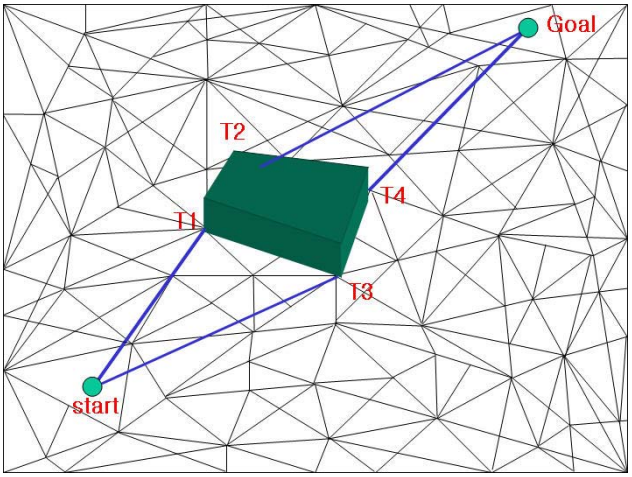

그림 1 가시성 그래프(Visibility Graph)

가시성 그래프를 생성하기 위해서 노드와 링크를 생성해야 한다. 노드는 장애물과 목표점, 장애물의 꼭지점으로 구성되어 있으며, 두 노드를 연결했을 때 생성되는 선분과 장애물이 교차하지 않는다면 가시성 그래프가 생성된 것이다. T1, T2, T3, T4들은 장애물의 정점을 의미하며, 시작 지점과 목표 지점 그리고 각각의 정점들을 연결하면 가시성 그래프가 생성된다. 이렇게 생성된 가시적인 연결 노드들에 대하여 유클리드 거리를 휴리스틱으로 사용하는 A* 알고리즘을 적용하여 최단 거리를 찾을 수 있다. 또한, 가시성 그래프의 가시적인 영역 안에서는 장애물이 없다는 것을 의미하기 때문에 계산량을 줄일 수 있다.

3. Navigation Mesh 상에서의 가시성 그래프

Navigation Mesh로 이루어진 3D 게임에서는 3D 상태 공간을 높이 정보를 제거한 2D 평면 공간으로 표현한다. 그래서 장애물의 정점들을 기반으로 이동 경로를 생성하는 가시성 그래프의 경우, 언덕과 같은 높이 정보를 가진 지형에서는 그림 2의 VG처럼 지형을 관통해 가시선이 생성되는 문제점이 발생한다.

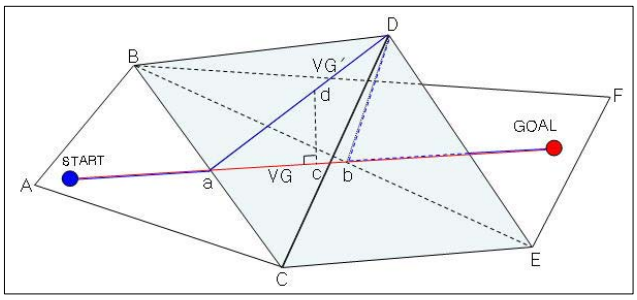

그림 2 Navigation Mesh 상에서의 가시성 그래프

이 가시선 VG에 높이 정보를 포함시켜 올바른 가시선을 생성하기 위해서는 우선 Y 좌표를 고려하지 않고 X와 Z의 좌표만을 이용하여 가시선 VG를 생성한다. 이 생성된 VG에 대하여, 가시선 VG가 경유하는 Mesh 평면과 교차하는지 검사한다. 만약 가시선 VG와 Mesh가 교차하게 되면, 가시선 VG와 교차하는 Mesh의 공유변에 대하여 직선의 방정식을 사용하여 가시선 VG와 $\overline{BC}$가 만나는 점 a를 추출할 수 있고, 점 a에서 가시선 VG가 $\overline{BD}$ 나 $\overline{CD}$ 를 거쳐 갈 경우에는 앞에서 언급한 방식으로 가시선과 선분의 교차점을 추출하면 "Start-a-D-b-Goal"로 이어지는 3D 가시성 그래프 VG'을 생성할 수 있다. 아래의 그림 3은 언덕과 같은 지형을 관통하지 않는 3D 가시성 그래프를 생성한 데모 프로그램이다.

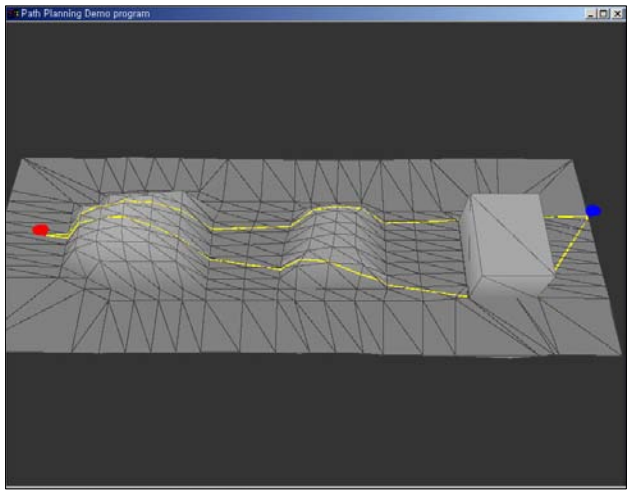

그림 3 가시성 그래프 생성 데모 화면

4. 가시성 그래프를 이용한 A* 알고리즘

휴리스틱 탐색(Heuristic Search)은 각각의 상태에서 해집합에 이르는 과정의 난의도에 대한 평가를 이용하는 탐색 과정이다. 휴리스틱 탐색은 탐색 속도 증가와 탐색 공간 축소를 해결하기 위해 제시된 방법으로 A* 알고리즘에서는 다음과 같이 정의내리고 있다.

표 2 A* 알고리즘

f(n) = g(n) + h(n)

- f(n) : 초기 Mesh S(Start)에서 출발하여 Mesh N을 거쳐 목표(Goal)에 이르는 비용의 추정값.
- g(n) : 초기 Mesh S에서 Mesh N에 이르는 비용.
- h(n) : Mesh N에서 목표에 이르는 비용의 추정값.

A*알고리즘에서는 초기 Mesh로부터 인접한 각 Mesh에 이르는 비용 g(n)과 현재 Mesh에서 목표에 이르는 추정 비용 h(n)을 더한 총 경로 비용 f(n)을 계산하여, f(n)이 가장 최소인 상태를 찾아 현재 지

점에서 목표 지점까지의 이동 경로를 찾는다. h(n)은 어떤 Mesh를 선택할 것인지에 대해 영향을 미치는 경로 비용 추정값으로 h(n)을 크게 하면 목표에 더 가까운 Mesh를 선택할 수 있다. 이러한 추정 비용 h(n)을 과소평가하면 검색하는 Mesh의 형태는 타원형태가 되며, 과대평가하게 되면 시작과 목표를 양 끝 모서리로 하는 마름모나 육각형 모형이 된다. 추정 비용 h(n)의 과대평가는 검색 속도를 떨어뜨리기는 하지만, 시작 지점과 목표 지점 사이에 비교적 단순한 모양의 장애물의 존재하는 경우에는 추정 비용을 과대평가하는 쪽이 더 빠른 결과를 얻을 수 있다. 가시성 그래프와 A* 알고리즘을 혼용한 방식은 이 추정 비용을 과대평가하여 빠른 검색을 얻는 방식이다. 그림 1을 예로 설명하면, 시작 지점(Start)에서 목표지점 까지 가시성 판단을 통해 장애물이 존재하는지 먼저 검사한다. 만약 장애물이 존재할 경우에는 가시적인 장애물의 정점들(T1, T2)까지의 거리와 각각의 장애물 정점(T1, T2)에서 목표지점(Goal)까지의 거리를 계산하여 경로 비용이 최소가 되는 정점(T1)을 찾는다. 이 정점을 부 목표지점으로 삼아 A* 알고리즘을 이용하여 경로를 탐색하고, 정점(T1)에서 다시 위의 과정을 반복하면 탐색 공간을 줄여 탐색 속도를 빠르게 할 수 있다.

표 3 가시성 그래프를 적용한 A* 알고리즘 휴리스틱

s : state (Mesh)
d(a, b) : A, B 사이의 유클리드 거리
t1, t2, ..., tn : S(start)에서 visible한 장애물의 정점들이라고 할 때,

f(s) = g(s) + h(s,G) (G: Goal)
h(s,G) = d(s, G)
= min[d(s, t1) + d(t1, G),
d(s, t2) + d(t2, G),
...
d(s, tn) + d(tn, G),]

아래의 그림 4 순서도에는 가시성 그래프와 A* 알고리즘을 혼용하는 방식을 설명하였다. 시작점과 목표점이 입력이 되면 시작 지점에서 목표 지점까지 가시성 판단을 통해 장애물이 존재하는지 검사하게 된다. 장애물이 없으면 목표지점까지 A*를 이용하여 이동하게 되고, 장애물이 존재하게 되면 시작 지점에서 장애물의 정점들까지 거리를 계산하여 최단 거리에 있는 장애물의 정점을 추출하게 된다.

이 추출된 좌표가 부 목표지점이 되어 A* 알고리즘을 이용하여 부 목표 지점까지 이동하게 되고, 추출된 부 목표지점은 다시 시작 지점이 되어 재귀적으로 부 목표지점을 찾게 된다.

가시성 그래프와 A* 알고리즘을 혼용한 방식에서는 시작 지점과 부 목표 지점까지 장애물의 존재하지 않기 때문에 추정 비용 h(n)을 과대평가하여 빠른 탐색을 할 수 있다.

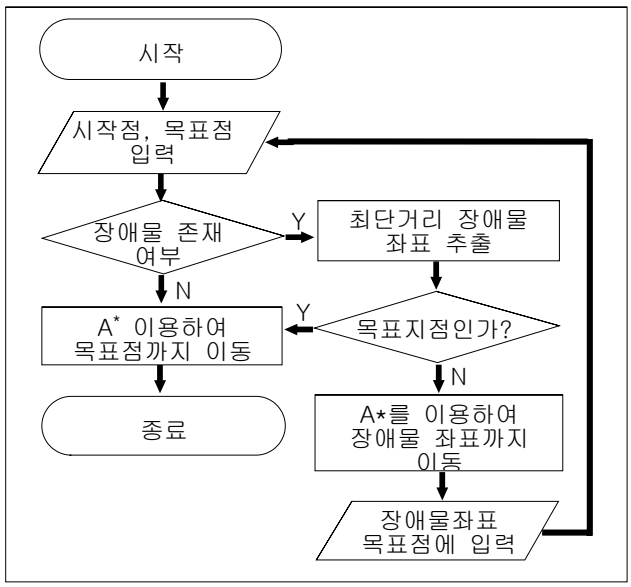

그림 4 가시성 그래프와 A* 알고리즘 혼용 방식 순서도

그림 5를 보면 장애물 정점(T2, T4, T5, T7)이 목표지점까지의 부 목표 지점이 되고, 각각의 부 목표 지점 사이에 A* 알고리즘을 적용하여 불필요한 탐색을 줄여 효율적으로 탐색을 할 수가 있다.

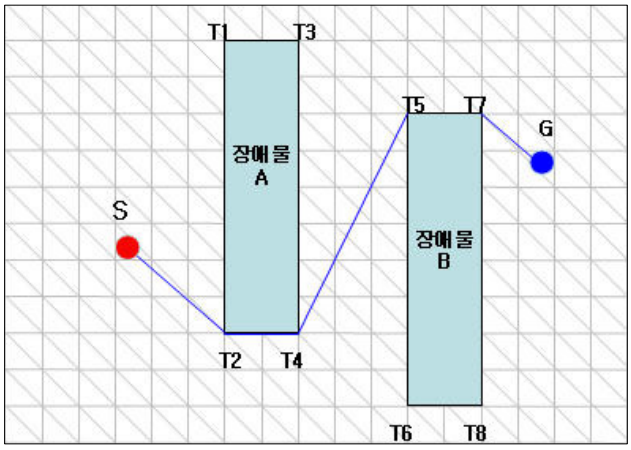

그림 5 가시성 그래프와 A* 알고리즘 혼용 방식

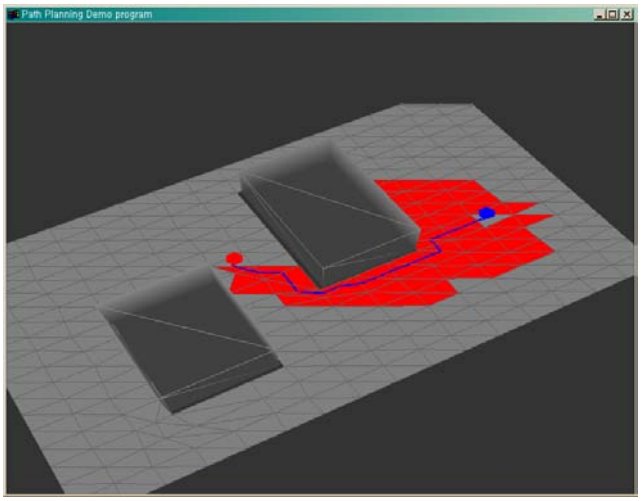

그림 6 A* 알고리즘만을 적용한 데모 프로그램

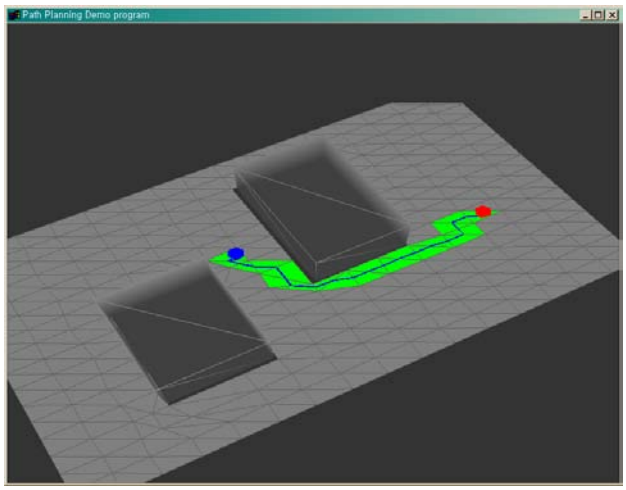

그림 7 가시성 그래프와 A* 알고리즘을 혼용한 데모프로그램

5. 결론

본 논문에서는 3D 게임에서 탐색 시간을 최소화하면서 효율적인 경로 탐색을 위해 Navigation Mesh 상에서 가시성 그래프와 A* 알고리즘을 혼용하여 적용할 수 있는 방법을 제안하였다

세밀하게 구성된 Navigation Mesh에서 A*알고리즘을 적용하면 탐색 공간이 많아지기 때문에 탐색 속도가 저하된다. 하지만 가시성 그래프를 이용하여 장애물의 정점을 찾아 반드시 찾아야 하는 Mesh들을 선정하고 A*의 휴리스틱 함수를 Mesh들을 경유하는 거리로 정의함으로써 A* 알고리즘만을 적용했을 때보다 탐색 공간이 현저히 줄어드는 것을 알 수 있었다.

참고문헌

[1] Helmut Alt, Michal Godau and Sue Whitesides, "Universal 3-dimensional visibility representations for graphs", Computational Geometry: Theory and Applications, v.9 n.1-2, pp111-125, Jan. 1998. 3

[2] M.de Berg, M.van Kreveld, M.Overmars and O.Schwarzkopf, *Computational Geometry Algorithms and Applications,* Springer, pp305-314,1997.

[3] Frédo Durand, George Drettakis and Claude Puech, "The 3D visibility complex", ACM Trans. Graph. 21(2): pp 176-206, 2002.

[4] Ghosh, SK and DM Mount, "An output sensitive algorithm for computing visibility graphs", Proc. 28lh Syrup. on Foundations of Computer Science, Los Angeles, 1987, 11-19.

[5] Mark H. Overmars and E. Welzl, “New methods for computing visibility graphs”, Proceedings of thefourth annual symposium on Computational geometry, pp164-171, June 06-08, 1988

[6] M. Pocchiola and G. Vegter, “Computing the visibility graph via pseudo-triangulations.”, In: Proc. 11th Annu. ACM Sympos. Comput. Geom., pp 248-257, June 1995.

[7] Greg Snook, "Simplified 3D Movement and Pathfinding Using Navigation Meshes", *Game Programming Gems*, Charles River Media, 2000.

[8] S. K. Wismath, Computingthe, "Full Visibility Graph of a Set of Line Segments", *Information Processing Letters*, 42, pp. 257-261, 1992.7