내접원을 이용한 3D게임에서의 이동경로 곡선화

김형일°, 정동민, 김준태, 엄기현, 조형제
동국대학교 컴퓨터공학과
{hikim°, mjung, jkim, khum, chohj}@dgu.edu

Generating Curved Path in 3D games By Using Inscribed Circle

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

요 약

본 논문에서는 3D게임에서 삼각형 내접원을 이용한 이동경로의 곡선화를 제안한다. 일반적으로 게임에서는 맵(map)을 형성하기 위해 그리드(grid) 방식을 채택한다. 그러나 그리드 방식은 지형을 형성할 때 격자를 이용하므로 자연스러운 맵을 형성하기 어려우며, 세밀한 묘사를 위해서는 많은 수의 격자를 이용하여야함으로 경로탐색 시 연산시간과 메모리에 부담을 준다. 이러한 문제점을 해결하기 위해서 웨이포인트(waypoint)를 이용하여 이동경로를 설정하기도 하지만, 이러한 방법은 게임디자이너가 웨이포인트를 미리 설정하여야한다는 단점이 뒤따른다. 근래에는 이러한 문제점 해결을 위해 네비게이션 메쉬(navigation mesh)를 이용하기도 한다. 네비게이션 메쉬를 이용하면 맵을 간소화할 수 있고 빠른 자동 경로탐색을 이룰 수 있다. 그러나 이러한 네비게이션 메쉬에서 A* 알고리즘을 적용하여도 이동경로는 꺾은선으로 나타나게 되어 현실감을 떨어뜨리는 원인으로 작용된다. 이러한 이동경로의 곡선화 문제점을 해결하기 위해 본 논문에서는 삼각형 내접원을 이용하여 이동경로의 곡선화를 이루었다.

1. 서 론

3D게임이나 가상현실에서는 현실감을 중요하게 여기고 이러한 현실감의 묘사는 복잡한 알고리즘과 하드웨어의 성능을 요구한다[3][7]. 세밀한 지형 공간의 묘사는 메모리에 부담뿐만이 아니라 이동경로를 탐색하는데도 많은 시간을 요구하게 된다. 메모리의 요구를 줄이고 게임 진행속도를 떨어뜨리지 않기 위해서는 지형 공간 표현을 단순화하는 것이 중요한 문제이다.

지형의 공간 표현 방법에는 그리드 방법이 대표적이다. 그리드 방법은 사각형 격자를 이용하여 지형을 표현하므로 이동경로탐색에 유용한 방법이다. 그러나 그리드 방법을 이용하여 맵을 표현하였을 경우에 이동 경로는 꺾은선을 나타내게 되므로 자연스러운 경로 표현에 장애가 된다[3].

꺾은선으로 나타나는 이동경로를 현실에 가깝게 곡선화시키기 위해서 많은 연구가 진행되어 왔다. 그리드 방법에서 이동경로를 자연스럽게 표현하기 위해서 격자를 세밀하게 나누어 꺾은선 부분을 자연스러운 이동이 이루어질 수 있도록 처리하기도 한다[9]. 그러나 여전히 격자를 기반으로 이용함으로써 완전한 곡선은 이룰 수가 없으며, 격자 분할 방법은 메모리에 부담을 주는 문제로 작용하므로 이용가치가 높다고 말하기는 힘든 실정이다.

이러한 그리드 방식의 문제점을 해결하기 위해 웨이포인트 방식을 사용하기도 한다. 웨이포인트를 사용할 때는 세밀한 지형을 먼저 형성시킨 후 이동경로를 디자인함으로써 자연스러운 이동경로를 쉽게 생성할 수 있으나, 제한적인 이동만 가능하게 되는 단점과 웨이포인트를 미리 설정해야하는 단점이 있다.

이러한 문제점 해결을 위해 근래에는 네비게이션 메쉬를 사용하기도 한다. 네비게이션 메쉬는 삼각형 모양의 메쉬를 이용하여 지형을 만들게 된다[3]. 메쉬는 세 개의 점을 이용하여 형성되는 삼각도형이지만, 크기나 모양에 제약을 주지 않으므로 이동경로의 공간 탐색시간을 축소시킬 수 있다. 그러나 네비게이션 메쉬에서도 이동경로의 꺾은선은 존재되며, 이러한 이동경로의 꺾은선은 3D게임에서 현실감을 떨어뜨리는 원인으로 작용한다.

이러한 곡선화 문제를 해결하기 위해 본 논문에서는 네비게이션 메쉬의 표현 형태를 이용하여 곡선화에 접근하였다. 맵을 형성하는 메쉬들은 삼각형을 이루며 이동경로의 꺾은선들도 삼각형을 이룬다. 이때 발생되는 삼각형에 내접원을 생성하여 이용하면 꺾은선이 나타나는 이동구간의 곡선처리를 간단하게 처리할 수 있다. 본 논문에서는 3D게임에서 꺾은선을 발생시키는 이동구간을 곡선화 하는 방법을 제안한다.

2. 관련연구

3D게임에서 현실감을 유지시키기 위한 이동경로를 이루기 위해 로보틱스에서의 path finding이나 그래픽스에서 사용되는 곡선처리 기법을 이용하고 있다[6][8]. 그러나 이러한 기법들은 게임에 접목하기엔 어려운 문제들을 가지고 있다.

2.1 Chaikin 알고리즘

Chaikin 알고리즘은 곡선처리에 이용되는 알고리즘이다[10]. 꺾은선으로 이루어진 두 개의 선분 $\overline{AB}$, $\overline{BC}$가 둔각을 이루는 하나의 선으로 연결되었다고 가정하고 각 선분의 양 끝점을 P_i와 P_{i+1}이라고 하면 Chaikin 알고리즘은 각 선분에 두 개의 control point를 생성한다. 한 개의 선분에는 두 개의 control point가 존재하고 두 개의 선분이 하나의 선을 이루어 곡선화를 이룰 경우에는 두 선분에서 인접한 control point를 연결하여 중으로써 두 개의 선분이 세 개의 선분으로 이루어진 선을 유도한다. 이러한 과정을 반복 수행함으로써 꺾은선을 곡선화할 수 있는 방법을 제공하는 것이 Chaikin 알고리즘이다. Chainkin 알고리즘에서 control point를 생성하는 방법은 시작지점(P_i)에서 가까운 control point를 $Q_i = \frac{3}{4}4P_i + \frac{1}{4}P_{i+1}$을 이용하여 계산하고 끝점($P_{i+1}$)에 가까운 control point를 $R_i = \frac{1}{4}P_i + \frac{3}{4}P_{i+1}$ 을 이용하여 계산하게 된다.

Chainkin 알고리즘은 부드러운 곡선을 자연스럽게 만들 수는 있으나 꺾은선을 곡선화하기 위해서는 많은 control point를 계

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

산해 내는 것을 반복 수행함으로 메모리에 부담을 주게 된다.

게임에서는 정교한 곡선을 생성하는 것보다는 사람이 인지할 수 없을 정도의 곡선화만 이루게 하고 메모리의 부담을 줄여주는 것이 더욱 효과적인 알고리즘이다[7].

2.2 Hermit 곡선과 Bezier 곡선

Hermit 곡선은 법선벡터를 이용하여 곡선을 만드는 알고리즘이다[1]. 예를 들어 꺾은선을 이루는 A, B, C, D라는 네 개의 점이 있을 경우 Hermit 곡선을 적용하게 되면 시작지점 A에서 끝점 D까지 곡선을 만들게 되는데, 이때 곡선을 형성시키는 작용점은 컨트롤 벡터 $\overrightarrow{AB}$와 $\overrightarrow{CD}$이다. 시작지점 A에서 끝점 D까지 곡선을 형성할 때 벡터 $\overrightarrow{AB}$와 벡터 $\overrightarrow{CD}$는 곡선화에 대한 작용을 하고 곡선과 벡터 $\overrightarrow{AB}$와 벡터 $\overrightarrow{CD}$는 접점을 형성한다. Hermit 곡선은 사용자 관점에 적합한 곡선을 형성할 수 없다는 단점을 내포하고 있어서 Bezier 곡선이 나오게 되었다.

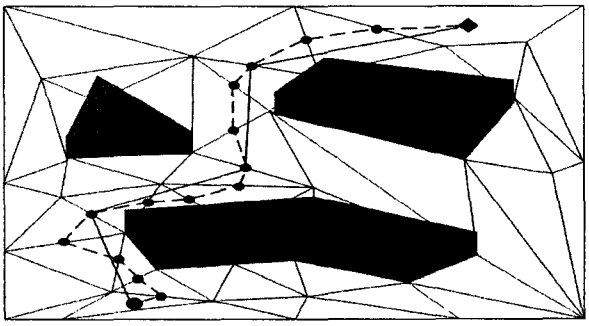
Bezier 곡선은 control point를 사용하여 곡선화를 이루게 된다[1]. Hermit 곡선에 활용한 예제를 Bezier 곡선에 적용하게 되면 B와 C가 곡선에 control point로 작용하여 곡선을 이루게 되고, 작용방식은 시작지점 A에서 끝점 D까지 곡선을 형성할 때 B와 C는 곡선을 자신의 방향으로 유도하는 역할을 한다. 이러한 control point를 적용함으로써 Bezier 곡선은 Hermit 곡선보다 자연스러운 곡선을 생성하게 되고 곡선의 자유로운 표현을 가능하게 하였다. 그러나 Bezier 곡선은 하나의 control point 변화가 곡선 전체에 영향을 미치는 단점을 내포하고 있다.

Hermit 곡선과 Bezier 곡선은 곡선을 자연스럽게 처리하는 것에는 적합할 수 있지만, 수행 시간과 control point의 영향을 고려한다면 3D게임에서는 적합한 곡선화 알고리즘이라 할 수 없다.

3. 이동경로의 곡선화

3.1 A*와 가시성판단을 적용한 최단경로 생성

네비게이션 메쉬를 이용한 상태 공간에서 A*를 적용하기란 쉬운 일이 아니다. 네비게이션 메쉬는 표현의 특성상 상태 공간을 축소시킬 수 있으나 표현의 단순화로 인해 A*를 효과적으로 적용하기 어려운 상태 공간이다. 이러한 A* 적용문제를 해결하기 위해 본 연구에서는 가시성 판단(Line of Sight) 이용하여 네비게이션 메쉬에서 이동경로의 최단거리를 생성하였다.



[그림 1] 최단경로를 위한 가시성 판단

가시성 판단은 현재의 웨이포인트(WP_i)와 이후의 웨이포인트(WP_{i+1})에 대한 가시성 판단을 연속적으로 거치게 되고 최대로 갈 수 있는 웨이포인트까지 이동경로를 재설정하게 된다. 가시성 판단을 적용함으로써 불필요한 메쉬를 거치지 않을 수 있고 초기의 A*의 설정을 최단거리로 재조정할 수 있게 된다. [그림 1]은 가시성 판단을 적용한 최단경로에 대한 그림이다.

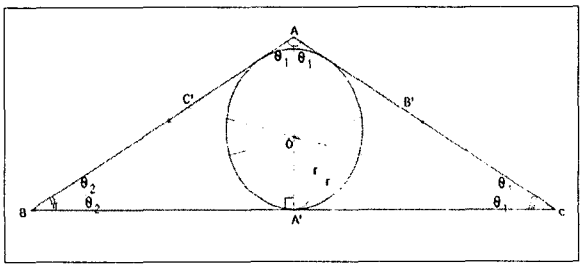
그러나 가시성 판단을 적용한 이동경로는 네비게이션 메쉬에서 최단거리로 사용될 수 있으나 꺾은선이 발생되어 자연스러움에 영향을 미친다.

3.2 곡선화를 위한 삼각형 내접원

내심, 외심, 무게중심, 수심, 방심을 삼각형의 오심이라한다. 이중에 내심은 삼각형의 내부에서 그릴 수 있는 최대의 원의 중심점(O)이다. 이때 생성된 원을 내접원이라 한다.

삼각형의 내심과 내접원에 대해서 [그림 2]에 나타냈었다. 내접원을 생성시키기 위해서는 삼각형의 세 내각을 이등분하는 이등분선을 대변에 그어준다. 세 이등분선 $\overline{AA'}$, $\overline{BB'}$, $\overline{CC'}$는 한 점(O)에서 만나게 되는데 이점을 내심(O)이라한다. 내심(O)을 통과하는 등변($\overline{AB}$, $\overline{AC}$)의 법선벡터와 끼지각(∠BAC) 이등분선($\overline{AA'}$)이 교차되는 지점(O)에서 밑변($\overline{BC}$)에 수선을 내리면 반지름(r)을 구할 수 있다. 이때 내심을 통과하는 법선벡터를 이용하여 접점을 구할 수 있다.

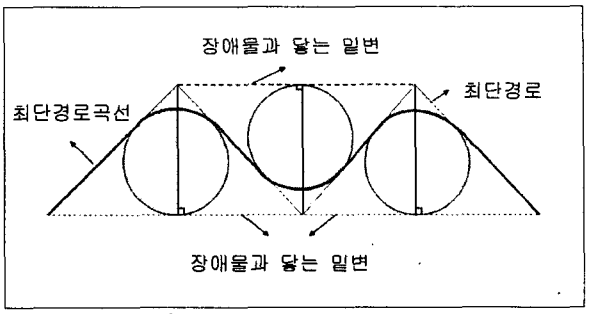
본 논문에서는 이동경로를 이등변삼각형으로 구성하여 내심, 내심원, 접점, 등을 구하게 된다.



[그림 2] 삼각형의 내심과 내접원

이등변삼각형의 높이($\overline{AA'}$)는 $\overline{AA'} = Rcos(\frac{1}{2}\theta)$이고 밑변의 길이($\overline{BC}$)는 $\overline{BC} = 2Rsin(\frac{1}{2}\theta)$이다.

메쉬로 이루어진 공간 상태에서 이동경로를 생성하게 되면 꺾은선 형태가 되며, 꺾은선의 변이점을 기준으로 이동경로를 분할하게 되면 이동경로에 대한 선분 집합을 얻을 수 있다. 이러한 분할된 선분들은 장애물과 같은 지형지물이 근접하여 있을 때 분할하게 되며, 분할된 구간에서는 꺾은선을 발생시키는 이동경로가 형성된다. 이러한 부자연스러운 이동경로를 이등변삼각형과 내접원을 이용하여 곡선화를 이룰 수 있다.



[그림 3] 최단경로의 곡선화

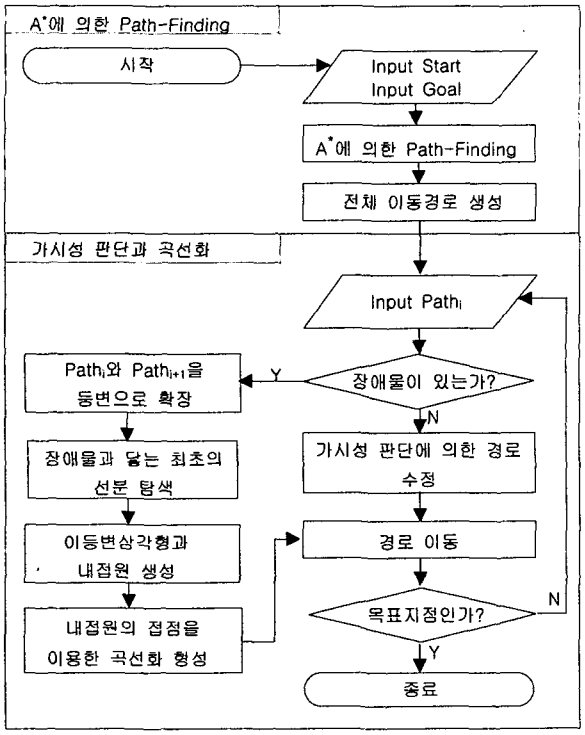
이러한 분할된 선분에 대해서 곡선화된 최단거리를 추출하기 위해서는 이등변삼각형의 밑변을 결정해야한다. 본 논문에서는 이등변삼각형의 밑변을 구하기위해서 분할된 선분의 중심에서 이등변삼각형을 이루며 확장하게 되고 확장된 선분은 장애물을

만날 경우 이등변삼각형을 완성하게 된다.

이동경로의 곡선화가 이루어지기 전에 가시성 판단이 먼저 수행되므로 장애물이 없을 경우에는 이동경로가 직선을 이루므로 이등변삼각형을 구성할 필요가 없다. [그림 3]은 최단경로가 결정된 후 최단경로의 곡선화가 이루어지는 방법을 도식화한 것이다.

3.3 내접원을 이용한 이동경로의 곡선화

[그림 4]는 내접원을 이용한 이동경로 곡선화에 대한 흐름도이다.

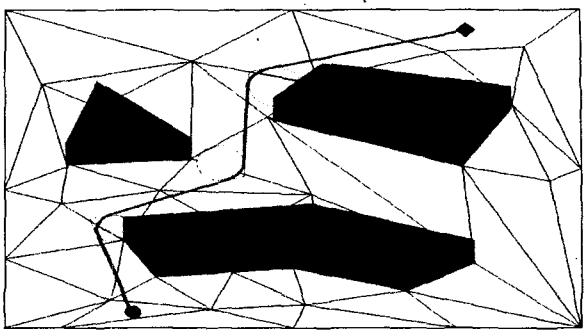


[그림 4] 내접원을 이용한 이동경로의 곡선화 흐름도

Step_1에서는 시작지점과 목표지점이 입력되면 A*에 의해 Path-Finding을 수행하고, 전체 $Path_i$(i=1,..i,..n)을 생성한다. 이렇게 생성된 이동경로는 가시성 판단과 곡선화가 이루어진 경로가 아니므로 꺾은선으로 이루어지게 된다. Step_2에서는 A*에서 생성된 이동경로를 보정하여 최단경로를 찾을 수 있도록 하고, 꺾은선으로 이루어진 부분을 곡선처리하기 위해 내접원을 생성하는 과정을 수행한다.

Step_1의 수행으로 전체 이동경로는 꺾은선들을 기준으로 분할된 이동경로 선분들로 나타낼 수 있다. 가시성 판단 모듈에 분할된 이동경로 선분($path_i$)이 입력되면 해당 경로의 분할 시작구간에 장애물이 있는지 판단하고, 장애물이 있을 경우 다음 이동경로 선분($path_{i+1}$)과 꺾은선을 형성하므로 곡선화 모듈에 의해 두 선분($path_i$, $path_{i+1}$)은 이등변삼각형을 이룰 수 있는 등변으로 확장된다. 그리고 경로를 등변으로 확장한 후 장애물과 닿는 최초의 밑변을 탐색하여 이등변삼각형을 형성한다. 형성된 이등변삼각형에서 내심을 구하고 내접원을 생성한다. 생성된 내접원과 등변에 닿는 접점을 이용하여 곡선화 시작점과 끝점을 생성하고, 곡선화 시작점부터 끝점까지는 내접원을 이용하여 이동한다.

[그림 5]는 네비게이션 메쉬 상태에서 이동경로에 곡선화를 적용했을 때는 그림이다. 점선은 이등변삼각형을 생성하기 위해 장애물에 닿는 최초의 밑변을 나타낸 것이다.



[그림 5] 네비게이션 메쉬에서의 최단경로 곡선화

얇은 실선은 A* 알고리즘 적용 후 가시성 판단을 수행하여 얻어진 최단경로이고 굵은 실선은 이등변삼각형의 내접원을 이용하여 곡선화된 최단경로이다.

4. 결론

본 논문에서는 3D게임에서 최단 이동경로 곡선화를 위해 가시성 판단을 이용하여 불필요한 메쉬의 이동을 회피하고 이등변삼각형의 내접원을 이용하여 이동경로의 곡선화를 이룰 수 있었다.

본 논문에서 제안한 이등변삼각형의 내접원을 이용한 이동경로 곡선화 방법은 3D게임이나 가상현실에서 이동경로 표현 시 현실성을 가미할 수 있고, 간단한 계산으로 곡선화를 이룰 수 있어서 효과적으로 적용할 수 있다.

향후 연구 과제로는 Visibility Graph를 이용하여 최단경로를 빠르게 탐색하는 방법과 이동경로 곡선화 수행시간을 단축할 수 있도록 접점과 중심점 빠르게 찾을 수 있는 방법에 관한 것이다.

참고 문헌

[1] E. Angel, *Interractive Comper Graphics*, Addison Wesley, 2000.

[2]H. Choset, I. Konukseven and A. Rizzi, "Sensor Based Planning: A Control Law for Generating the Generalized Voronoi Graph", *In Proc. IEEE Int. Advanced Robotics*, 1996.

[3] M. Deloura, *Game Programming Gems*, Charles River Media, 2000.

[4]M. Ech and H. Hoppe, "Automatic Peconstruction of B-Spline Surfaces of Arbitrary Topological Type", *Computer Graphics*, 1996.

[5]K. K. Gorowara, "On Bezier Curves and Surfaces", *Proceedings of the IEEE National*, 1988.

[6] S. Quinlan and O. Khatib, "Elastic Bands: Connecting Path Planning and Control", *IEEE International Conference on Robotics and Automation*, 1993.

[7] S. Rabin, *AI Game Programming Wisdom*, Charles River Media, 2002.

[8] T. W. Sederberg, J. Zheng, D. Sewell and M. Sabin,"Non-uniform Subdivision Surface", *Computer Graphics Annual Conference Series*, 1998.

[9] 김건형, 정광호, "개선된 A* 알고리즘의 적용", *한국게임학회 학술발표논문집*, 2003.

[10] Http://graphics.cs.ucdavis.edu/CAGDNotes/Refinernet/Refinernent.html