한국게임학회 논문지
제6권 제3호 2006년 9월

3D게임에서 이동 장애물을 고려한 동적 경로 탐색 기법

권오익[0], 황보택근
경원대학교 소프트웨어대학
pookwon@gmail.com[0], tkwhangbo@kyungwon.ac.kr

A Dynamic Path-Finding Method Avoiding Moving Obstacles in 3D Game Environment

Oh-Ik Kwon[0], Teag-Keun Whangbo
College of Software, Kyungwon University

요 약

게임 인공지능 분야중 하나인 경로탐색은 좀더 사실적인 게임을 만들기 위한 중요한 요소이다. 경로 탐색 시스템은 한정된 자원을 소비해야만 하는 제약사항 때문에 때때로 단순하게 처리되어 사실적이지 못한 경로를 생성하였다. 기존 연구에서는 정적인 지형과 장애물들을 이용하여 자연스럽게 회피하는 경로생성에 집중하였다. 하지만, 게임 공간에서는 다양한 종류의 움직이는 장애물들이 존재한다. 따라서 이러한 움직이는 장애물을 자연스럽게 회피하는 경로를 생성하는 시스템이 필요하다.

본 논문에서는 네비게이션 메시(Navigation Mesh)로 공간을 표현하며 지형의 특성을 고려한 경로 탐색 방법을 적용하고, 움직이는 물체를 회피하기 위하여 지능적인 밀개와 끌개의 방법을 사용하여 경로 탐색을 수행한다. 제안된 시스템을 통하여 생성된 경로를 살펴보고 실제 게임에서의 활용성을 검증한다.

ABSTRACT

Path-finding, one of the traditional Game A.I. problems, becomes an important issue to make games more realistic. Due to the limited resources in the computer system, path-finding systems sometimes produce a simplified and unrealistic path. The most recent researches have been focused on the path-finding avoiding only static obstacles. Various moving obstacles are however deployed in real games, a method avoiding those obstacles and producing a smooth path is necessary.

In this paper, navigation mesh is used to represent 3D space and its topological characteristics are used for path-finding. Intellectual repulsor and attractor are also used to avoid moving obstacles and to find an optimal path. We have evaluated the path produced by the method proposed in this paper and verified its usability in real game.

Keyword : Path-Finding, Navigation Mesh, Obstacle Avoidance

본 연구는 제2단계 BK21사업에 의하여 지원 받았음

1. 서 론

게임에서의 경로탐색은 게임의 인공지능적인 요소 중 하나로 게임의 완성도와 밀접한 관련이 있다. 자신이 조작하는 캐릭터가 지능적으로 길을 찾아갈 수 있다면 완성도 높은 게임이 될 것이다. 일반적인 경로탐색과 다른 점은 2차원 지도에서의 경로탐색과 달리 3차원 공간에서 경로탐색을 수행해야하며, 실시간으로 많은 요소들이 변화된다는 점이 다르다. 예를 들어 캐릭터가 목표지점을 향하여 최적 경로를 따라서 이동하는 도중 움직이는 장애물에 의하여 방해를 받게 된다면 기존의 경로를 변경해야 되는 상황이 발생하게 된다.

기존의 게임에서의 경로 탐색에 관한 연구는 주로 정적인 상황에서 좀 더 자연스러운 경로를 찾는 방법에 주안점을 두고 있었다[1,2]. 하지만, 게임 환경은 동적인 객체들이 다수 존재하는 경우가 빈번하므로 단순한 경로 탐색은 실용성이 떨어진다.

본 논문에서는 대표적인 공간 표현 기법인 네비게이션 메시[3]를 바탕으로 열린 지형과 닫힌 지형을 구분하여 각각의 특성에 맞는 경로 탐색 방법을 적용한 경로 탐색 시스템을 제안하고자 한다. 열린 지형은 사방이 막혀있지 않는 공간으로 정의되며 이동에 방해되는 장애물이 비교적 적은 특징을 지닌다. 반면, 닫힌 지형은 사방이 막혀있는 건물 안과 같은 구조로 정의 되며, 좁은 외길이나 공간을 이동하는 포탈(Portal)의 존재가능성이 높은 복잡한 구조를 갖는 특징이 있다. 이러한 특성에 적합하도록 닫힌 지형에서는 오프라인에서 계산된 최적의 참조테이블을 사용한다. 참조테이블은 동적인 객체에 의한 경로 변경에 강인하지만, 테이블의 저장 공간의 낭비가 크다는 단점이 있다[4]. 열린 지형에서는 큰 단위의 지형을 탐색하기 때문에 속도를 우선시하는 RTA(Real-Time A*)[5,6]를 사용한다. 탐색 도중 만날 수 있는 동적 객체에 대한 처리 방법으로 기존의 밀개와 끌개(Repulsors and Attractors)의 방법[7]을 지능적으로 개선하여 사용한다.

본 논문의 구성은 다음과 같다. 2장 관련연구에서는 공간 표현기법인 네비게이션 메시와, 경로 탐색 방법으로 쓰인 RTA와 계층적 경로참조표 그리고 밀개와 끌개의 방법에 관하여 살펴본다. 3장 제안된 시스템에서는 열린 지형과 닫힌 지형을 구분하는 경로 탐색 시스템의 전체적인 개요와 개선된 밀개와 끌개의 방법에 관하여 설명한다. 4장에서는 제안된 시스템의 구현 결과와 생성된 경로의 효율성을 검증하며, 마지막 5장에서는 결론과 향후 추가적으로 필요한 요소에 대하여 논의 한다.

2. 관련연구

경로 탐색 시스템은 크게 두 가지 요소로 구분된다. 탐색할 공간을 표현하는 방법과 표현된 공간을 탐색하는 방법이 필요하다. 표현된 공간을 탐색하기 위해서 A*를 개선한 RTA(Real-Time A*)를 사용한다. RTA는 A*에서 탐색 노드의 수를 줄임으로써 수행 시간을 단축하는 알고리즘이다.

A* 이외에 탐색 방법으로 경로참조표[8]를 활용하는 방법이 있다. 이것은 실시간으로 공간을 탐색하는 것이 아니고 오프라인에서 이동 가능한 조합을 계산하여 최적의 경로참조표를 생성하는 방법으로, 실시간에서는 단지 표를 참조하는 것만으로 검색기능을 수행할 수 있다. 그러나 경로참조표의 높은 메모리 요구량 때문에 부가적인 메모리 최적화 방법이 필요하다. 이동될 수 있는 노드가 N개일 경우 N^2 갯수만큼 공간이 필요함으로 노드를 그룹별로 분리하는 계층형 경로참조표가 필요하다. 계층형 경로참조표[9]는 이동공간을 계층적으로 분리하고, 큰 그룹단위에서 경로를 찾은 후 작은 그룹단위에서 세부적인 경로를 탐색하는 방법으로 활용된다.

공간 표현 방법으로 네비게이션 메시를 사용한다. 네비게이션 메시는 크게 삼각형 기반과 사각형 기반으로 나눌 수 있는데 여기에서는 삼각형 기반을 사용한다. 삼각형 기반의 메시가 사각형 기반에 비하여 단순한 경로 탐색을 수행할 수 있다[10].

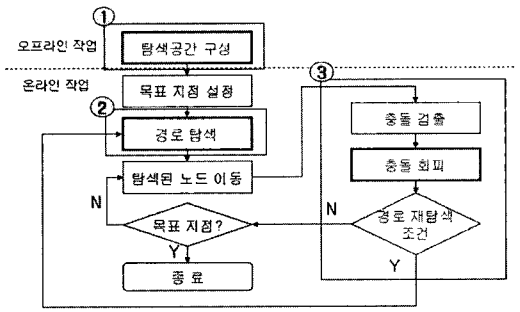
경로 탐색 분야에 또 다른 중요한 문제는 장애물과의 충돌처리 문제이다. 장애물은 벽과 같은 고정적인 객체와 이동 될 수 있는 동적 객체로 나눌 수 있다. 기존의 연구에서는 주로 정적인 장애물에 대한 회피를 다루었지만, 게임에서는 동적 객체가 빈번하게 존재하기 때문에 이에 대한 적절한 처리가 필요하다. 동적 객체를 처리하는 방법에는 그래프를 실시간으로 업데이트 하는 방법[11] 과 밀개와 끌개를 활용하는 방법이 있다. 본 논문에서는 밀개와 끌개를 확장하여 보다 지능적인 회피를 수행 할 수 있었다.

본 연구에서는 지형적인 특성을 고려한 탐색 방법과 동적 장애물을 지능적으로 회피 할 수 있는 시스템을 제안한다.

3. 제안한 동적 경로 탐색 기법

3.1 시스템 흐름

제안한 경로 탐색 시스템의 구성은 다음과 같다. 탐색 공간을 구성하고 구성된 공간을 탐색하는 단계 그리고 충돌 위험 객체가 존재 할 경우 회피 하는 충돌 회피 단계 등의 3단계로 나눌 수 있다. [그림1]은 경로 탐색 시스템의 흐름을 보여준다. 단계 1에서는 탐색 공간을 구성한다. 탐색 공간을 효율적으로 구성하기 위하여 본 논문에서는 열린 지형과 닫힌 지형의 특징을 구분하여 처리한다.



[그림1] 경로 탐색 시스템 흐름도

열린 지형에서는 네비게이션 메시를 기반으로 지형을 구성하며, 닫힌 지형에서는 네비게이션 메시를 기반으로 계층을 분할한 경로참조표를 생성한다. 단계 2에서는 실질적인 경로탐색을 수행한다. 열린 지형에서는 실시간에 적합한 RTA로 경로 탐색을 수행하며 닫힌 지형에서는 계층적 경로 참조표를 활용하여 경로를 탐색한다. 경로 탐색 과정 후 단계 3에서는 탐색된 경로를 이동한다. 현재의 캐릭터가 위치하고 있는 노드가 충돌 가능성이 있는 동적 객체에 근접해 있을 경우에 제안된 충돌 회피 방법을 적용하여 경로를 변경한다.

이 장에서는 제안한 경로 탐색 시스템에서 핵심적인 요소인, 열린 지형과 닫힌 지형을 구분하는 이유와 밀개와 끌개의 방법을 확장하여 동적 객체를 자연스럽게 회피하는 방법에 대하여 설명한다.

3.2 열린 지형과 닫힌 지형의 구분

게임에서 공간은 크게 두 가지 분류로 나눌 수 있다. 첫째, 사방이 뚫려 있는 열린 지형과, 둘째, 사방이 벽과 같은 장애물로 막혀있는 닫힌 지형으로 구분된다. 두 지형들은 여러 가지 특징들을 갖고 있는데 경로 탐색에서 지형적 특징을 고려하여 특성화 한다면 더욱 효율적인 경로탐색을 수행 할 수 있다. [그림2]는 (주)블리자드코리아에서 서비스하고 있는 온라인 RPG 장르의 "World of Warcraft" 게임의 스크린 샷이다. [그림2] (1)는 열린 지형의 예를 보여주기 위한 스크린 샷이고, [그림2] (b)는 건물 내부의 닫힌 지형을 보여주는 스크린 샷이다.

(a) 열린 지형 (b) 닫힌 지형

[그림2] 실제 게임에서의 지형

열린 지형은 일반적으로 건물 바깥의 외부 지형을 의미하며 다음과 같은 특징을 지닌다. [그림2] (a)에서와 같이 대체적으로 큰 지형을 표현하며 나무가 우거져 있는 숲과 같은 예외적인 지형을 제외한다면 복잡한 지형보다는 시야에 장애물 요소가 적은 단순한 지형 구조를 가진다. 따라서 본 시스템에서는 이러한 특징들을 잘 수용할 수 있는 각각의 방법을 통하여 경로 탐색 시스템의 효율성을 높이고자 한다.

반면 닫힌 지형은 건물 안의 방안이나 복도와 같은 지형 구조를 의미하며 다음과 같은 특징을 지닌다. 일반적으로 좁은 지형단위로 구분되며, [그림2] (b)와 같이 미로와 같은 복잡한 구조를 가질 확률이 높다. 복도와 같은 외길이나, 방과 방 사이를 연결하는 문, 엘리베이터와 같은 포탈(portal)도 다수 존재 한다. 여러 개의 방으로 나뉘어져 있는 경우라면 탐색 경로의 효율성도 고려해야 된다. 예를 들어 방 하나를 통과하여 이동하는 경로보다 두 개를 거쳐 가는 거리가 더 짧을 수 있다.

[그림3]에서는 열린 지형과 닫힌 지형에서의 구분된 탐색 방법을 보여준다. 닫힌 지형에서는 다음과 같은 방법을 사용한다. 경로 탐색을 위한 사전 탐색 공간 표현 기법으로 다각형의 대표 지점인 중점을 그래프의 노드로 구성하는 네

비게이션 메시를 사용한다. 이렇게 구성된 노드를 바탕으로 다익스트라 경로 탐색 방법[10]을 바탕으로 최소 비용 경로참조표를 생성한다. 이 경우 참조표의 메모리 사용량을 줄이기 위하여 제안된 층 분할 방법을 통하여 적절한 그룹으로 분리하는 작업 수행 후 계층적 경로참조표를 생성한다. 생성된 참조표는 닫힌 지형에서의 경로 탐색 방법으로 이용된다.

	열린 지형 (외부)	닫힌 지형 (내부)
공간 표현 기법	네비게이션 메시 (Navigation Mesh)	네비게이션 메시 (Navigation Mesh)
경로 탐색 기법	RTA (Real-Time A*)	계층적 경로참조표 (Hierarchical Lookup-Table)
동적 객체 회피	지능적 밀개와 끌개 (Intellectual Repulsor & Attractor)	지능적 밀개와 끌개 (Intellectual Repulsor & Attractor)

[그림3] 열린 지형과 닫힌 지형의 경로 탐색 방법

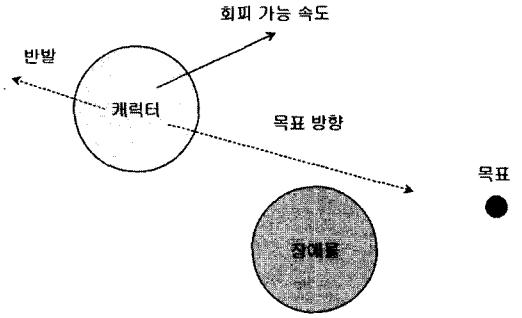
열린 지형에서는 다음과 같은 방법을 사용한다. 닫힌 지형에서와 마찬가지로 최적화된 네비게이션 메시를 기반으로 최적경로를 보장하기 보다는 속도를 우선시하는 RTA의 방법을 사용하여 목표지점까지의 경로 탐색을 수행한다. RTA를 사용하는 이유는 경로 이동 도중 동적 객체에 의한 경로 변경이 빈번할 수 있기 때문에 처음 탐색된 경로가 반드시 최적 경로임을 보장하지 않기 때문이다.

3.3 확장된 밀개와 끌개 방법

3.3.1 밀개와 끌개

좀 더 사실적인 시뮬레이션 행동을 얻기 위해서는 무엇을 피해야 하고, 무엇을 가까이 해야 하는지를 알고 있는 것이 도움이 된다. 밀개와 끌개는 이동 객체들의 충돌 회피 수단으로 사용되며 무리 짓기 행동의 시뮬레이션, 자동차 경주의 충돌 회피, 게임 환경에서 적을 추적하기와 같은 용도로 널리 쓰이는 개념이다[11].

움직이는 객체에 밀개의 힘을 적용하여 캐릭터가 피해야 할 객체 주위에 접근하게 되면 밀개로부터 멀어지는 힘을 받게 된다. [그림4] 에서 캐릭터가 목표 방향으로 이동하기 위해서는 목표 방향대로 이동해야 한다. 하지만 그 경우 장애물에 방해를 받기 때문에 회피 가능 속도로 이동 방향을 바꾸어야만 한다. 이 때 장애물과 캐릭터 사이의 거리에 따른 반발력을 적용하여 회피 가능 속도로 이동 방향을 변경한다.



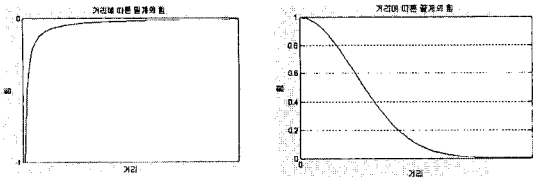
[그림4] 반발력을 이용한 충돌 회피

한편, 캐릭터에 영향을 주는 밀개는 하나 이상일 수 있기 때문에 힘들에 대한 결합 방법이 필요하다. 그리고 끌개의 영향력과 기존의 캐릭터의 영향력을 고려해야 한다. 다음의 수식 1은 뉴턴의 역학에 기반을 둔 밀개와 끌개의 결합 벡터를 구하는 수식이다.

$$a = \sum \frac{f}{m} \quad (m > 0)$$
$$v = v_0 + a \times dt \qquad (1)$$
$$p = p_0 + v$$

수식 1에서 a는 가속도이며, m은 객체의 질량이다. v_0는 현재 속도, v는 새로운 속도이며, p_0는 현재 위치, p는 새로운 위치이다. 가속도 a는 주위에 존재하는 밀개와 끌개의 힘(f)을 질량으로 나눈 값의 합이며, 각각의 밀개에 대한 가속도 값을 합하여 가속도 a를 구한다. 매 프레임 마다 원래의 속도 v_0에 가속도 a를 적용하여 새 속도 v를 구하고, 새 속도를 원래의 위치 p_0에 적용하여 현재 프레임에서의 위치 p를 구한다.

가속도 a를 결정할 때 이동 장애물과 캐릭터의 거리가 고려된다. [그림5]는 거리에 따른 밀개와 끌개의 힘을 보여준다. [그림5] (a)와 같이 밀개의 힘 f는 캐릭터와의 거리에 반비례한다. 거리가 가까워질수록 미는 힘의 크기는 커진다. 힘의 크기가 마이너스인 의미는 객체와 멀어지는 방향을 의미한다. 반면 끌개의 힘은 거리가 가까워질수록 끄는 힘이 커진다. [그림5]에서 x축은 거리를 의미하는 것으로 객체사이의 거리이기 때문에 0 이하가 될 수 없다.

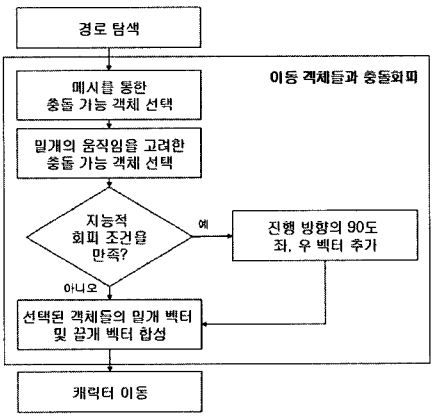


(a) 밀개의 힘 (b) 끌개의 힘

[그림5] 밀개 $f=-\frac{1}{d}$ 그래프와 끌개 $f=e^{-d^2}$ 그래프

3.3.2 밀개와 끌개의 확장

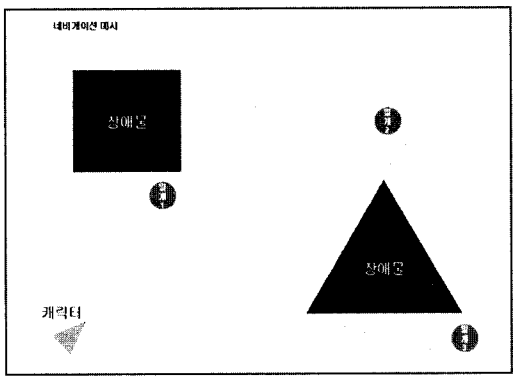
기존의 밀개와 끌개의 방법을 바탕으로 [그림6]과 같이 확장하여 적용한다. RTA 또는 경로 참조표를 통한 경로 탐색의 결과로 경로들의 정보를 활용하여 캐릭터의 이동을 최적화 한다. 이 과정은 충돌 가능한 이동 장애물들이 다수 존재 할 경우 모든 개체와의 충돌검사를 피하기 위함이다. 네비게이션 메시에 장애물 존재정보를 표시하여 캐릭터의 주변에 있는 장애물만 선택하게 되며, 선택된 장애물 중에 캐릭터가 탐색하게 될 방향에 영향을 미치지 않는 장애물을 선택에서 제외 하게 된다.



[그림6] 이동 객체들의 충돌회피 흐름도

영향을 미치는 경우에는 단순하게 밀개와 끌개의 적용 하는 것 보다 짧은 경로로 회피를 수행하기 위하여 방향을 고려한 충돌 가능성 조건을 검사한다. 조건을 만족 하는 경우에는 추가적인 벡터 적용하여 지능적인 회피를 수행한다. 다음은 단계별로 과정을 설명한다.

첫 번째 과정은 네비게이션 메시 정보에서 충돌 가능 장애물의 선택이다. 먼저 현재 캐릭터가 위치하고 있는 메시가 밀개의 영향력이 미치는 메시 인지 검사한다. 밀개가 위치하는, 공유 변을 갖고 있는 주변 메시에 밀개의 존재 여부를 표시하여 캐릭터의 주변에 밀개의 존재 여부를 판단한다. [그림7]에서는 밀개 1의 주변 메시에 캐릭터가 위치하게 되므로 밀개 1을 충돌 가능성이 있는 밀개로 선택한다.



[그림7] 밀개의 영향 범위

최종적으로 메시 범위와 장애물의 이동 방향을 고려하여 선택 한다. 밀개의 움직임 벡터가 캐릭터와 멀어지는 방향이라면 이전 과정에서 선택된 객체라 할지라도 캐릭터의 움직임에 영향을 주지 못하므로 제외한다.

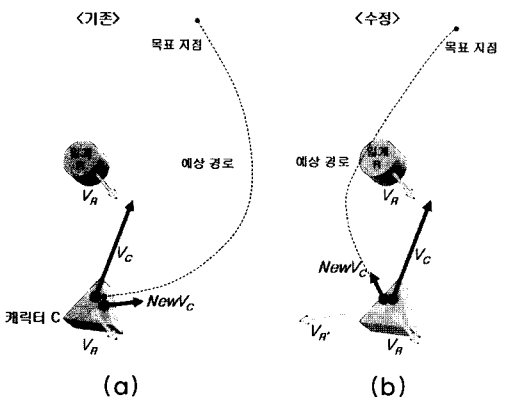
두 번째 과정으로 밀개를 효율적으로 회피 할 수 있는 반발력을 더 할 것인지 여부를 판단한다. 밀개와 끌개를 바로 적용하게 되면 캐릭터의 움직임 방향을 고려하지 않기 때문에 효율적이지 못한 회피가 발생 할 수 있다. 이러한 반발력을 추가하는 조건은 다음과 같다.

캐릭터의 전방에 밀개가 존재
캐릭터의 위치와 가까워지는 방향으로 이동
밀개의 반발력의 크기가 임계 범위

위의 조건을 검사하기 위하여 수식 2의 조건 1, 2, 3을 검사한다. 밀개가 캐릭터 전방에 존재하는지 검사하기 위하여 조건 1과 조건 2를 검사하여 밀개가 전방에 존재하는지 검사한다. 조건 1, 2를 만족하는 경우 밀개가 캐릭터의 위치와 가까워지는지 검사하기 위해 조건 3에서 벡터 V_C와 벡터 V_R이 이루고 있는 방향을 고려한다. 조건을 모두 만족하는 경우 V_C를 기준으로 벡터 V_R의 반대방향으로 90도인 벡터 V_C을 추가한다.

$$
\begin{aligned}
&P_i = i \text{ 인덱스의 객체 위치} \\
&V_i = \{\ i \quad \text{인덱스의 객체의 벡터, 방향, 크기}\ \} \\
&V_{ij} = P_i - P_j \\
&\text{조건1} : V_{RC} \bullet V_C > 0 \\
&\text{조건2} : V_{RC} \times V_C \quad (\text{ 부호가반대 }) \quad V_R \times V_C \\
&\text{조건3} : V_R \bullet V_C < 0
\end{aligned} \tag{2}
$$

[그림8]은 위의 조건을 검출 과정을 도식화 한 것이다. 작은 원에서는 짧은 거리이기 때문에 방향을 전환하기 어려운 거리이다. 큰 원은 임계 범위를 의미하며, 이 범위를 넘어서면 캐릭터에 영향력이 적기 때문에 고려하지 않아도 된다. [그림8] (a)는 위의 조건을 검출하여 적용한 결과를, [그림8] (b)는 기존의 방법을 그대로 사용 했을 경우를 보여준다. [그림8] (a)와 같이 새로운 이동 벡터 $NewV_C$는 밀개의 반발력에 의하여 그림에서 우측 방향을 갖는다. 하지만 밀개 R의 움직임 벡터의 방향을 고려한다면 [그림8] (b)와 같이 좌측으로 방향을 선회하여 회피경로를 좀 더 단순화 시키고 짧게 만들 수 있다.

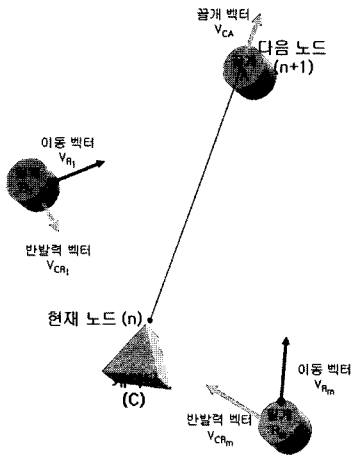


[그림8] 밀개의 방향을 고려한 캐릭터의 움직임

3.3.3 밀개와 끌개의 힘의 합성

여기에서는 이전 절에서 설명한 과정에 대하여 실제로 힘을 합성하는 방법 대하여 설명한다. 첫 번째 과정으로 사전에 선택에 의하여 충돌 가능성이 있는 장애물이 캐릭터에 미치는 힘을 구한다. [그림9]는 회피해야할 장애물 밀개 R_1과 R_m이 캐릭터의 다음노드 탐색에 영향을 미칠 때의 힘의 적용 방법을 도식화한 것 이다.

[그림9]에서 밀개의 이동 벡터 V_{Rm}은 밀개 m의 움직임 방향과 힘의 크기를 나타내며, 밀개의 다음위치를 결정하는데 사용된다. 밀개의 힘을 적용하기 위한 캐릭터에 적용 될



[그림9] 밀개와 끌개의 영향을 받는 캐릭터

반발력 벡터의 크기는 수식 3으로 계산한다.

$$
\begin{aligned}
PO_R(d) &= -\frac{1}{d+0.1}\ (d \geq 0) \\
PO_A(d) &= e^{-d^2}
\end{aligned} \tag{3}
$$

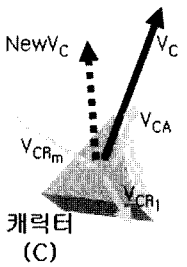
수식 3에서 d는 캐릭터 C와 밀개 객체 R의 유클리디안 거리이며 $PO_R(d)$은 밀개가 캐릭터에 미치는 힘을 계산한다. 거리에 따른 밀개의 힘은 거리가 가까워질수록 미는 힘이 커진다. 밀개와의 거리가 0에 가까워져 밀개의 힘이 무한대로 커지는 문제점을 회피하기 위하여 거리에 0.1을 더한다. $PO_A(d)$는 캐릭터와 끌개와의 거리 d에 해당하는 끌개의 힘을 구하는 함수이다.

캐릭터의 영향을 주는 밀개와 끌개의 힘 V_{CRm}, V_{CA}는 다음 수식 4로 구한다. 수식 4(a)는 m번 밀개의 위치P_{Rm} 에서 캐릭터의 위치P_C 로 향하는 방향 벡터를 정규화 하고, P_{Rm}과 P_C의 거리 값 d를 밀개의 힘을 경정하는 함수 $PO_R(d)$로 밀개의 힘의 크기를 얻은 후 정규화된 방향 벡터와의 곱으로 최종 캐릭터에 적용 시킬 벡터를 구한다. 수식 4(b)는 끌개에 관한 벡터를 구하기 위한 수식으로 밀개와 동일하게 적용한다.

$$
\begin{aligned}
&\text{(a)}\ V_{CR_m} = PO_R(d)\frac{P_{R_m} - P_C}{|P_{R_m} - P_C|} \\
&\text{(b)}\quad V_{CA} = PO_A(d)\frac{P_A - P_C}{|P_A - P_C|}
\end{aligned} \tag{4}
$$

두 번째 과정은 캐릭터의 이동 벡터와 합성이다. [그림10]

과 같이 캐릭터의 적용되는 밀개들의 반발력 벡터 V_{CRm}, V_{CR1} 과 끌개의 벡터 V_{CA} 그리고 캐릭터의 이동 벡터의 합을 구하여 $NewV_C$를 계산한다.



[그림10] 캐릭터에 적용되는 벡터의 합성

[그림10]에서 새로운 이동 벡터인 $NewV_C$를 구하기 위하여 수식 5를 사용한다. 캐릭터 주위의 밀개들이 앞서 소개한 좌우회피조건을 만족할 경우 캐릭터의 새로운 움직임 벡터($NewV_C$)는 다음과 같다. 현재 캐릭터의 이동 벡터인 V_C 와 추가적인 회피 벡터 $V_{C'}$, 밀개들의 이동벡터 합(ΣV_{CRi},V_{CA})으로 결정되며, 그렇지 않은 경우는 회피 벡터인 $V_{C'}$을 제외한 합을 통하여 구한다.

$$NewV_C = \begin{cases} \text{조건만족} : V_c + \sum_{i=1}^{m} V_{CR_i} + V_{CA} + V_{c'} \\ \text{그렇지 않음} : V_c + \sum_{i=1}^{m} V_{CR_i} + V_{CA} \end{cases} \quad (3)$$

새로운 움직임 벡터를 바탕으로 현재 프레임에서 한 캐릭터의 위치는 수식 6과 같이 구한다. s는 캐릭터의 속도이며, t는 한 프레임을 그리는데 걸리는 시간이다. P_C는 현재의 위치, $NewP_C$는 새로운 위치이다.

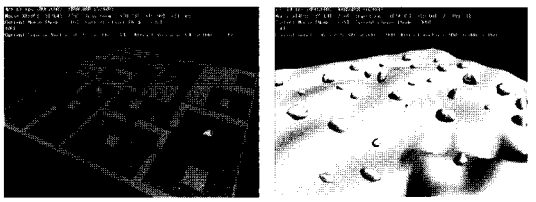
$$NewP_C = s\frac{NewV_C}{|NewV_C|}t + P_C \quad (6)$$

t 시간만큼 지난 다음 프레임에서의 캐릭터의 위치는 정규화된 캐릭터의 새로운 이동 벡터 $NewV_C$를 캐릭터의 속도 s 값과 이전 프레임을 그리는데 걸린 소요 시간 t를 곱하여 위치 변화량을 구한 후 기존 캐릭터의 위치값 P_C 에 더하여 얻는다.

4. 구현 및 결과 분석

제안된 경로 탐색 시스템의 구현 및 실험 환경은 다음과 같다. 윈도우XP 기반으로 Visual C++ .NET 2005와 DirectX SDK 9.0c 버전을 기반으로 구현하였으며, 하드웨어는 펜티엄4 3.0Ghz, 1G 메모리, Nvidia GeForce 6600에서 실험하였다.

실험에 사용된 두 가지의 지형은 각각 다음과 같다. [그림11]는 닫힌 지형이며, [그림12]은 열린 지형이다. 닫힌 지형에서는 볼록한 부분이 벽을 의미하여 탐색할 수 없는 부분을 의미한다. 열린 지형에서는 커다란 장애물이 다수 존재하며, 각각의 공간에 동적 장애물로 20여개가 골고루 분포되어 직선운동을 수행한다. 또한 탐색 경로 추적을 위하여 경로 탐색 중 1초에 10번씩 현재 위치에 대한 점을 찍는다.



[그림11] 실험에 사용된 닫힌 지형

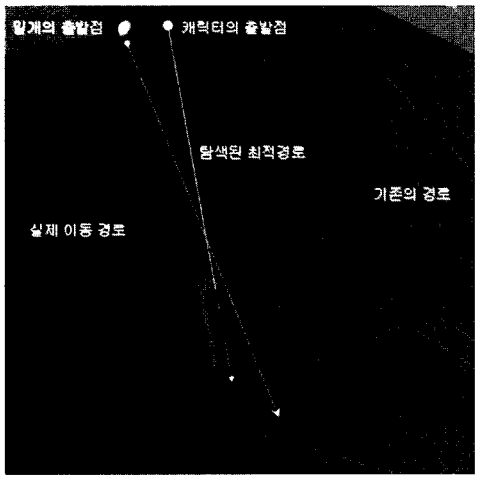
표 1에서는 실험에 사용된 열린 지형과 닫힌 지형의 정점과 메시 정보를 보여준다.

	지형	네비게이션 메시
닫힌 지형	정점 : 962 메시 : 1,489	정점 : 303 메시 : 158
열린 지형	정점 : 21,911 메시 : 40,730	정점 : 2,924 메시 : 5,597

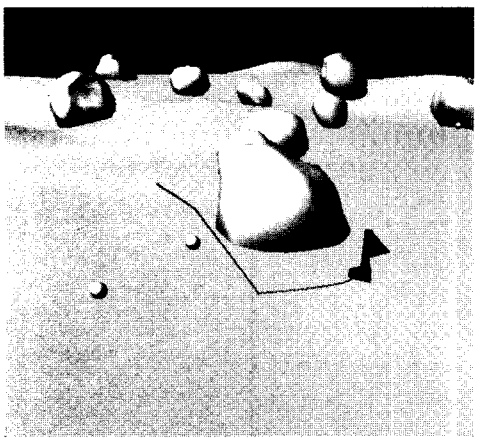
[표1] 지형과 네비게이션 메시 데이터 정보

각각의 환경에서 순수 경로탐색에 소모된 비용을 25회 측정해본 결과 다음과 같았다. 닫힌 지형에서는 평균 3.72ms 이였으며, 열린 지형에서는 45.64ms이다. 게임에서 초당 30 프레임을 그렸을 때 부드럽다고 가정할 때 1프레임을 그리는데 걸리는 시간은 33ms이다. 33ms이면 열린 지형에서는 단지 약 2 프레임을 그리는 시간을 소모하기 때문에 게임에 적용하는데 문제가 없는 연산량이다. 닫힌 지형이 더 빠른 연산을 보인 이유는 미리 계산된 계층적 경로참조표를 활용했기 때문이다.

밀개와 끌개를 확장한 결과는 다음과 같다. [그림12]과 같이 캐릭터의 출발점에서 목표지점까지 탐색된 최적경로는 직선이다. 하지만 탐색 도중 밀개와 충돌 가능성이 생기기 때문에 기존의 밀개와 끌개의 방법으로는 대략적으로 두꺼운 점선으로 표시된 경로를 탐색하게 된다. 하지만, 지능적인 적용을 통하여 밀개와 캐릭터의 방향을 고려하면 실제 이동 경로와 같은 경로를 그리며 탐색하게 된다. 열린 지형에서도 [그림13]와 같이 경로를 탐색 하는 것을 볼 수 있다.



[그림12] 확장된 밀개와 끌개의 적용



[그림12] 확장된 밀개와 끌개의 적용

위 결과에서 확인 할 수 있듯 밀개와 끌개를 확장한 방법은 경로의 곡선화와 같은 부가적인 방법을 쓰지 않아도 사실적인 경로 탐색을 수행하는 것을 볼 수 있다.

5. 결론 및 향후 연구 방향

본 논문에서는 네비게이션 메시를 바탕으로 밀개와 끌개의 방법으로 게임에서 움직이는 물체를 효과적으로 회피하기 위한 수단에 관한 연구를 수행하였다. 제안한 시스템으로 닫힌 지형에서는 계층적 경로참조표를 활용하여 탐색시 소모되는 연산량이 게임에 적용 가능한 것을 보았으며, 열린 지형에서는 방대한 데이터를 처리하기에 적합한 RTA를 사용하여 검증하였다. 또한, 동적 객체를 회피해야 할 상황이 왔을 때 객체의 방향을 고려하여 기존에 탐색된 경로를 최소한으로 수정하는 방향으로 회피 하도록 설정함으로써 좀 더 지능적이며 짧은 경로를 탐색을 수행하였다.

논문에서 제안한 경로 탐색 시스템을 확장하여 전략을 고려한 경로 탐색 방법, 수영이나 뛰어넘기와 같은 행위를 적용한 범용적인 시스템이 되도록 구현하고, 배포가 가능한 형태로 수정한다면 실제 게임에서도 활용이 용이한 시스템이라고 사료된다. 향후 로봇의 장애물 회피에 적용하는 연구와 경로탐색 시스템으로 얻어진 회피 경로에 대하여 최적의 경로임을 검증하는 방법에 관하여 연구가 필요하다.

참고문헌

[1] 김형일, 정동민, 김준태, 엄기현, 조형제, "내접원을 이용한 3D게임에서의 이동경로 곡선화", 한국정보과학회 학술발표대회논문집, 30권 2호, pp52-54, 2003.

[2] 전현주, 유견아, "그리드 기반 게임에서 Voronoi diagram을 이용한 곡선 경로찾기", 한국컴퓨터종합학술대회, Vol. 32, No. 1, pp604-606, 2005.

[3] Paul Tozour, "Building a Near-Optimal Navigation Mesh", AI Game Programming Wisdom, Charles River Media, 2002.

[4] William van der Sterren, "Path Look-up Tables - Small is Beautiful", AI Game Programming Wisdom 2, Charles River Media, 2003.

[5] 최은미, 김인철, "미지의 공간상의 실시간 최단 경로 탐색 알고리즘에 대한 분석", 한국정보처리학회 춘계학술대회 논문집 제12권 제1호, 2005.

[6] A. Felner, R. Stern, A. Ben-Yair, S. Kraus, N.

Netanyahu, "PHA*: Finding the Shortest Path with A* in An Unknown Physical Environment", Journal of Artificial Intelligence Research (JAIR) 21:631-670, 2004.

[7] John Olsen, "Attractors and Repulsors", Microsoft, Game Programming Gems, Charles River Media, 2004.

[8] Smith Surasmith, "Preprocessed Solution for Open Terrain Environments", AI Game Programming Wisdom, Charles River Media, 2002.

[9] Adi Botea, Martin Muller, "Near Optimal Hierarchical Path-Finding", Journal of Game Development, 2004.

[10] Paul Tozour, "Search Space Representations", AI Game Porgramming Wisdom 2, Charles River Media, 2003.

[11] 김태원, 조경은, 엄기현, "실시간 A* 길찾기와 동적 그래프 문제를 위한 계층적 그래프 모델", 한국게임학회 2004년 하계 총회 및 학술발표대회, pp149-155, 2004.

[12] O' Rourke Joseph, "Computational Geometry in C", Second Edition, Cambridge University Press, 1994.

[13] Geraint Johnson, "Avoiding Dynamic Obstacles and Hazards", AI Game Programming Wisdom, Charles River Media, 2003.

권 오 익

2004년 8월 경원대학교 전자계산학과(학사)
2004년 9월 ~ 현재 경원대학교 전자계산학과 석사과정
관심분야 : 3D 게임 엔진, 3D 경로 탐색

황 보 택 근

1983년 2월 : 고려대학교 공과대학 학사
1987년 8월 : CUNY 전산학과 석사
1995년 8월 : S.I.T. 전산학과 박사
1995년 ~ 1997년 : 삼성종합기술원 선임연구원
1997년 ~ 현재 : 경원대학교 소프트웨어대학 부교수
관심분야 : 3D 게임 엔진, 3D 경로 탐색, 3D 그래픽스

논문투고일 - 2006년 8월 1일
심사완료일 - 2006년 9월 6일