동적 3D 게임 환경에서의 실시간 경로탐색

권오익[1], 황보택근[2]
경원대학교 소프트웨어대학
525252@gmail.com[1], tkwhangbo@kyungwon.ac.kr[2]

Real-Time Path Finding on Dynamic 3D Game Environment

Oh Ik Kwon[1], Taeg Keun Whangbo[2]
College of Software, Kyungwon University[1][2]

요 약

한정된 자원을 사용할 수 있는 게임 AI 분야에서는 시스템 자원을 적절하게 활용하여 현실감을 극대화 시키려는 노력이 중요한 이슈이며, 3D 게임에서 캐릭터들의 자연스러운 경로 탐색은 현실성을 높이는 중요한 척도 중 하나이다. 기존 연구에서는 주로 정적인 지형, 객체들을 적절하게 회피하는 경로에대한 연구가 많이 진행되었다. 그러나 최근 널리 이용되고 있는 다중사용자가 접속하는 온라인 RPG 게임에서는 기존 방법을 그대로 적용하기에 많은 연산량이 필요한 문제점이 있다. 본 논문에서는 네비게이션 메시(Navigation Mesh) 기반으로 최적화된 A*, 그리고 밀개(Repulsors)의 방법을 통하여 동적인 환경에서 자연스러운 경로탐색을 수행하며 3D 게임에 적용 가능한 연산량을 충족하는 경로탐색 시스템을 제안하였다.

Keyword : 3D Game AI, Path-Finding, Navigation Mesh, Collision Avoidance

1. 서론

게임 AI 분야 중 하나인 경로탐색 시스템은 전통적으로 게임뿐만 아니라 자동차의 네비게이션 시스템, 각종 시뮬레이션 시스템에 다양하게 활용되고 있는 분야중 하나이다[1]. 국내의 경로탐색 시스템은 게임 분야에 적용될 기술을 연구하기 보다는 자동차의 네비게이션 시스템과 같은 다른 분야에 적용할 수 있는 기술개발이 활발하게 진행되었다.

기존의 연구들은 주로 정적인 지형들, 예를 들면 웅덩이, 장애물, 도달할 수 없는 지형들을 피하면서 경로 탐색하는 방법에 대한 연구가 활발하였다[2]. 하지만 전략게임이나 RPG 게임과 같이 움직임이 많은 객체들이 게임상에 존재하는 동적 환경의 경우에는 기존의 탐색방법을 사용하기 힘들다. 따라서 동적인 환경이 존재하는 게임에서 사용될 수 있는 적은 연산량을 가지면서 자연스러운 경로탐색 시스템이 필요하다.

본 논문에서는 네비게이션 메시를 사용하고 부가적으로 계층적인 개념을 추가하여 경로탐색에 사용한다. 네비게이션 메시로 표현된 공간을 효율적으로 탐색하는 방법으로 A*[4]를 사용한다. 탐색된 경로의 변경이 빈번한 경우 기존의 A*는 효율성이 떨어진다. 최적 경로를 찾기보다는 좀더 빠른 접근이 필요하며 자연스러운 탐색을 위해서 점진적 A*의 사용이 필요하다.

2 장 관련연구에서는 기존의 네비게이션 메시, A*에서 본 논문에 사용된 기법을 설명하고, 3 장에는 제안된 시스템에 사용하는 계층적 네비게이션 메시와 점진적 A*, 밀개와 끌개의 방법을 설명한다. 4 장에는 제안된 시스템을 통하여 생성된 경로를 보여주며, 마지막장에서는 결론 및 향후 연구 방향에 대하여 기술한다.

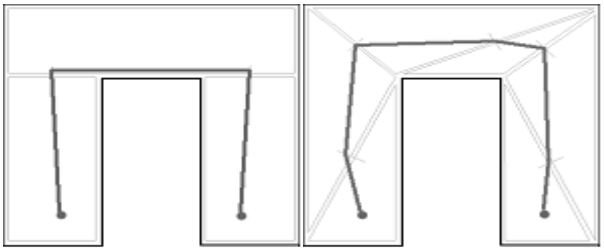
2. 관련연구

2-1 네비게이션 메시(Navigation Mesh)

검색공간을 표현하는 방법은 여러 가지가 있다. 서론에서 소개한 다양한 기법들중 가장 강력한 방법으로 알려진 네비게이션 메시는 3 차원 게임 환경에서 “탐색을 할 수 있는 공간”으로 정의된다. 이러한 네비게이션 메시는 평면을 기술하는 볼록 다각형의 집합이며, 게임내의 캐릭터들이 게임 세계안에서 네비게이션과 경로탐색을 위해 사용하는 간단하면서 직관적인 “평면도”이다. 네비게이션 메시는 이동 능력에서 서로 다른 캐릭터들에 대해 최적의 경로를 비교적 신뢰성있게 찾아낼 수 있다[5]. 반면, 많은 수의 다각형들을 저장해야 하며, 복잡한 환경에서는 연산량이 급격히 증가하는 문제가 발생한다[3]. 네비게이션 메시의 다각형 안에서는 어떠한 것에도 부딪히지 않고 마음대로 돌아다닐 수 있으며 각 다각형은 검색공간 그래프의 한 노드가 되고, 인접한 다각형들 사이의 연결이 그래프의 간선이 된다.

네비게이션의 종류는 N 변 다각형을 메시로 사용하는 경우와 삼각형을 메시로 사용하는 경우로 나눌 수 있다. N 변 다각형 기반에서는 팔각형태의 방하나를 팔각형으로 표시 할 수 있지만, 삼각형 기반은 최소한 6 개의 삼각형이 필요하므로 메모리를 적게 차지한다. 하지만 N 변 다각형 기반의 메시는 벽에 가까이 붙어서 탐색 할 가능성이 높다는 단점을 가지며, 이러한 문제를 교정하기 위해서는 추가적인 작업이 필요하다. 삼각형 기반의 메시의 경우에는 벽에 붙어 탐색 할 수 있는 가능성을 피할 수 있다. 또한 정점 풀링(Pooling)이 용이하며 다각형기반의 방식과 정점의 개수를 비슷하게 유지 할 수 있는 장점을 가진다[3].

그림 1 (a) , 그림 1(b)를 비교해보면 삼각형 기반의 메시를 사용하는 방법에서는 벽에 가까이 붙는 문제를 자동적으로 해결해준다. 본 논문에서는 삼각형 기반의 네비게이션 메시를 사용하였다.



(a) 다각형 기반 (b) 삼각형 기반

그림 1 네비게이션 메시의 종류

2-2 A* Search

A* 알고리즘은 맵 상의 두 지점들 사이의 경로(Path)를 찾는 경로탐색 알고리즘 중 하나이다. A*는 두 지점을 잇는 여러 가지 경로가 존재할 경우 최단경로를 찾는다는 점과 가장 짧은 시간을 소비하여 찾는다는 장점을 지니고 있다[6]. A*의 경우 목적지까지 경로에 대한 비용을 추정하게 되는데 이 추정에는 휴리스틱(Heuristic)을 사용한다. A*는 수식 (1)과 같이 표현된다.

$$f(n) = g(n) + h(n) \qquad (1)$$

노드 n 에 해당하는 비용(f)은 시작점에서 노드 n 까지의 실제 비용 g 와 노드 n 에서 목표지점까지의 휴리스틱 추정치 값(h)의 합으로 계산된다. A*에서는 열린 목록(Open list)과 닫힌 목록(Close list)을 관리한다. 열린 목록은 아직 조사 하지 않는 상태들을 담은 것이며, 닫힌 목록은 이미 조사한 상태들을 담은 목록이다.

A*의 성능은 휴리스틱 값(h)을 어떻게 추정하느냐에 따라 결정된다. 휴리스틱 값을 과대평가하지 않고 적절한 값으로 추정해야 A*가 가장 최적의 경로를 찾아 준다. 적절하게 추정되지 않는 휴리스틱 값 때문에 쓸모없는 노드 탐색으로 인하여 소비되는 비용이 증가 될 수 있다. 하지만 최적의 경로보다 검색 속도가 우선한다면 비용을 과대평가해서 검색 속도를 높여주는 방법도 있다[7].

3. 제안 시스템

그림 2 에서 전체 시스템의 흐름을 보여준다.

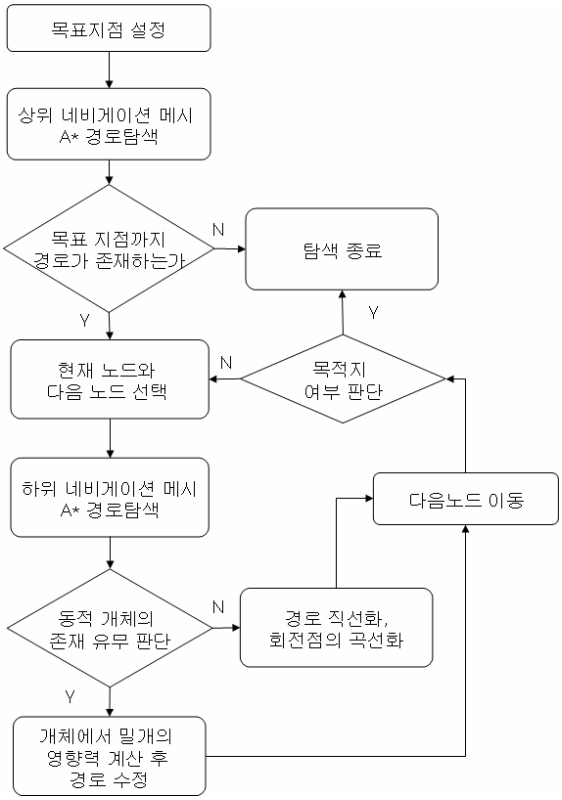


그림 2 제안 시스템 흐름도

실제 게임에서 일어나는 상황으로 예를 들면, 사용자가 조종하는 캐릭터를 목표지점으로 이동시키기 위해 명령을 내린다. 경로탐색 시스템은 목표지점의 좌표를 입력받아 저장한다. 이미 오프라인으로 구축되어있는 계층적 네비게이션 메시에서 상위 레벨의 메시를 통하여 목표지점까지 A*를 수행한다. 게임화면 시야에는 보이지만 장애물에 의하여 사방이 막힌 경우에는 목표지점까지 도달 할 수 없게 된다. 이 경우 하위 네비게이션 메시까지 연산 과정을 거치지 않고 바로 탐색이 종료된다. 그렇지않을 경우에는 현재노드와 다음 노드를 선택하고, 범위안에 움직이는 객체가 있는지 검사한다. 객체가 존재하지 않는 경우에는 바로 하위 네비게이션 메시를 통하여 A*를 수행하여 경로를 탐색하고, 자연스러운 움직임을 위해 탐색 경로를 직선화시키고, 회전점을 부드럽게 만든다. 객체가 존재하는 경우는 하위 네비게이션 메시를 통하여 점진적 A*를 수행한다. 이 경우는 움직이는 객체의 밀개의 힘에 의해 경로가 수정 될 가능성이 높기 때문에 자세한 A*의 연산을 제한시킨다. 상위 네비게이션 메시 상에서 선택된 다음 노드까지 이동했다면 다음 노드를 현재 노드로 변경하고 다음 노드에 대한 탐색 과정을 반복한다.

3-1 계층적 네비게이션 메시(Hierarchical Navigation Mesh)

앞서 기술한 것처럼 네비게이션 메시의 구성에 따라 경로탐색 시스템의 성능에 큰 영향을 미친다. 검색 공간의 표현이 게임 세계의 지형과 잘 맞지 않는 경우, 경로탐색 시스템의 실제 움직임을 화면에 자연스럽게 보이게 하는 것은 매우 힘들다[5]. 탐색공간을 너무 복잡하게 표현하면 쓸모없는 노드들의 탐색으로 인하여 성능이 떨어지는 문제점이 있다. 반대로 공간을 과도하게 줄이게 되면 부자연스러운 경로를 탐색하게 될 가능성이 높아지기 때문에 적절한 조합이 필요하다.

계층적 네비게이션 메시는 지형을 바탕으로 생성되며, 지형에서 필요 없는 노드들을 합친 메시들로 이루어진 하위계층과 필수적인 메시들로만 구성되어있는 상위계층으로 나눠진다. 그림 3 은 계층적 네비게이션의 예를 보여준다.

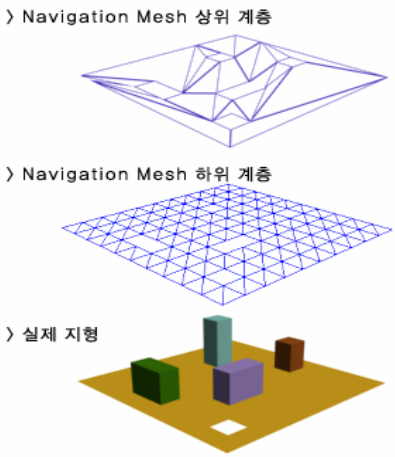


그림 3 계층적 네비게이션 메시의 구성

게임에서의 탐색 경로는 동적인 물체에 의하여 변경될 경우가 매우 빈번하다. 그렇기 때문에 자세한 경로 탐색 보다는 계층적 네비게이션 메시의 상위층을 통하여 목적지까지 도달 가능성을 판단하고, 가능하다면 탐색된 노드별로 하위층의 네비

게이션 메시를 탐색하여 실제 경로를 생성하는 계층적 네비게이션 메시의 사용이 더욱 효율적이다.

계층적 네비게이션의 구성에서 계층의 결정은 두 단계가 적합하다. 상위 계층은 성긴 삼각형으로 구성되며 하위 계층은 조밀한 삼각형으로 메시들이 구성된다. 여러개의 다양한 계층을 사용하지 않는 이유는 계층이 늘어남에 따라 메모리의 사용량이 크게 증가하기 때문이다.

메시의 생성은 게임 지형 디자이너가 수동으로 생성하는 방법도 있겠지만, 게임의 방대한 맵을 수동으로 제작하기란 시간적인 비용의 소비가 매우 크기 때문에 유용하지 않다. 따라서 지형에 쓰이는 삼각형 메시를 적절하게 통합하여 네비게이션 메시로 생성하는 방법이 필요하다. 하지만 이 방법을 연구하는 것도 커다란 주제이기 때문에 본 논문에서는 Paul[5]이 제안한 시스템을 적용하였다. 이 시스템의 핵심은 지형 데이터에서 “걸을 수 있는” 평면을 법선 정보를 이용하여 법선이 임계값 안에서 위로 향하고 있는 평면을 선택한 다음, Hertel-Mehlhorn 알고리즘[9]을 사용하여 볼록한 다각형을 찾는다. 그 후 인접한 노드를 합쳐 다각형을 구성한 다음 삼각형으로 나누어 하위 계층을 구성하는 메시를 생성한다. 생성된 하위 계층의 메시에 “3→2 합치기” 방법을 사용하고 일정 크기 이하의 작은 삼각형 메시를 합친다. 이 방법을 통하여 상위 계층을 구성하는 메시를 생성한다.

3-2 점진적 A*

관련연구에서 설명한 A*는 실시간 게임 환경에 그대로 적용하기에는 적합하지 않다. 많은양의 메모리와 연산이 필요하며, 게임내의 지형이 수천이상의 열린 목록과 닫힌 목록을 요구하는 정도의 큰 지형이라면 부가적인 최적화가 필요하다. 따라서 A*를 적용할 때 모든 세부 경로를 한 번에 탐색하지 않고 분할 탐색하는 방법이 필요하다. 사용자가 이동 명령을 내렸을 경우 한꺼번에 많은 A* 연산의 과정이 반복되어 일시 멈춤 현상이 발생 할 수 있기 때문이다. 그러므로 한꺼번에 연산을 수행하기 보다는 프레임별로 나누어서 연산을 수행하는 것이 화면에 보이는 측면에서 더욱 자연스러운 결과를 보여 줄 수 있다. 크게 A*를 최적화하기 위하여 두 가지의 방법을 사용한다. 우선 A* 프로세스를 한꺼번에 계산하지 않고 분할 계산하는 방법과 목록들의 탐색에 따른 비용절감을 위한 최적화 방법이다.

점진적인 A*를 적용하기 위하여 첫 프레임을 화면에 표시하기 전에 상위 네비게이션 계층에서 과대평가된 휴리스틱을 사용하여 빠르게 탐색한 다음 캐릭터를 출발시킨다. 다음 프레임에서 선택된 상위 계층 노드에 포함된 하위 네비게이션 계층의 노드들을 바탕으로 A*를 사용한다. 선택된 하위 계층의 노드들이 복잡할 경우에도 집중된 A* 연산으로 인하여 부자연스러운 움직임이 발생할 수 있다. 이런 경우를 방지하기 위해서 한 프레임 당 A* 연산 프로세스 횟수를 제한하는 방법을 사용한다. 앞서 소개한 최적화 방법으로는 최적 경로는 보장하지 못하지만 A* 연산으로 발생할 수 있는 멈춤 현상을 제거 할 수 있다.

최적화된 A*를 적용하기 위해서 메모리 최적화를 수행한다. A*의 경로탐색 과정을 보면 열린 목록과 닫힌 목록은 빈번하게 참조 되는 항목이다. 이러한 목록들의 자료구조가 중요한데, 해시 테이블(Hash table)을 사용하여 빠르게 참조 한다. 또한 목록 참조 횟수를 줄이는 방법도 유용하다. 이 방법은 각 네비게이션 메시 노드안에 플러그 배열을 추가하여 정보를 저장하는 방법을 쓴다[10]. 플러그 배열추가로 목록이 열린 목록에 있는지 닫힌 목록에 있는지를 미리 판단하여 목록 참조 횟수를 줄일 수 있다. 하지만 메모리 요구량이 노드당 1 바이트 늘어나게 되는 단점이 존재한다.

3-3 밀개(Repulsors)

기존의 A*, 네비게이션 메시가 사용된 논문은 주로 정적인 환경을 중심으로 연구되어왔다. 하지만 이러한 방법을 그대로 동적인 물체가 존재하는 게임에 적용하기란 쉽지 않기 때문에 다른 추가적인 방법이 필요하다. 동적인 환경에서 기존의 방법을 활용하기 위해서는 움직이는 객체 정보를 참고하여 캐릭터의 탐색된 경로에 영향을 주는 방법이 필요하다. 여기에서 끌개와 밀개의 방법 중 밀

개의 방법을 사용한다. 이 방법은 객체들이 무엇을 가까이 해야 하고 무엇을 피해야 하는지 알게 함으로 좀 더 사실적인 행동을 만들어내는데 도움을 주는 방법이다[8].

$$a = \sum f \cdots (a)$$
$$v = v_0 + a \cdot dt \cdots (b) \quad (2)$$
$$p = p_0 + v \cdots (c)$$

여기에서 f는 밀개의 힘, a는 가속도, v_0는 현재 캐릭터의 속도, v는 새로운 속도, dt는 시간변화량, p는 새로운 위치, p_0는 현재 위치이다. 밀개의 힘을 적용하기 위해 하위 네비게이션 메시에서 수식 2(a)를 통해 탐색범위 안에 있는 모든 반발력(f)의 총 합(가속도)을 구한다. 캐릭터의 현재 속도에 구한 가속도를 적용하여 수식 2(b)로 새로운 속도(v)를 구한다. 수식 2(c)을 통하여 새 속도를 캐릭터에 적용하여 새 위치(p)를 구한다. 기존 연구에서는 끌개의 영향을 미치는 힘(속도)의 합을 구할때 방향에 대한고려가 없었다.

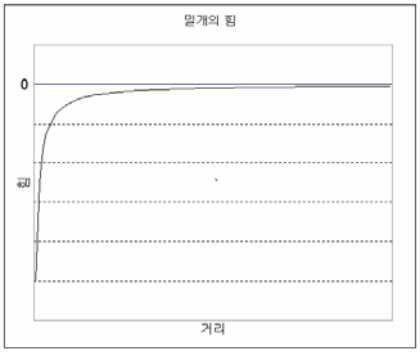


그림 4 거리에 따른 밀개의 힘

본 시스템에서는 방향을 고려하게 되는데 연산량의 감소를 위해 8 방위(0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°)만 고려한다. 또한 구현코드 최적화를 위해 삼각함수의 각각의 sin, cos 값을 미리 계산하여 배열에 저장한다. 그림 2 는 $y = -x^{-1}$ 그래프를 오른쪽으로 0.01 만큼 이동시킨 $y = -\frac{1}{x + 0.01}$ 그래프이다. 오른쪽으로 약간 이동 시킨 이유는 거리가 0 이 되는 경우 힘이 무한대로 강해질 우려가 있기 때문이다.

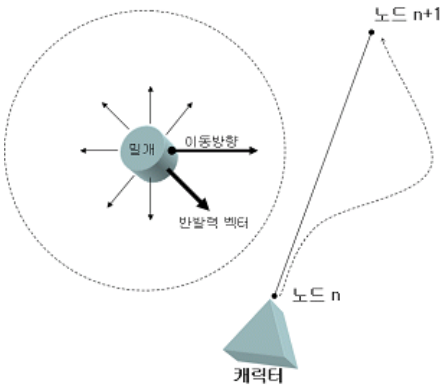


그림 5 캐릭터와 밀개의 상호작용

그림 5 에서 캐릭터는 노드 n 에서 노드 n+1 까지 이동 중이다. 화면에서 우측방면으로 이동하고 있는 밀개를 내장한 객체는 영향 범위를 가지는데 그림 5 에서 원형 점선으로 나타나있다. 캐릭터가 이동하고 움직이는 객체의 영향력 안에 들게 되면, 반발력 벡터의 방향을 결정하기 위해 캐릭터와 객체사이의 최단거리 직선과 미리 정의되어있는 물체의 방향 중 가장 가까운 방향을 선택한다. 그 후 그림 4 에 의해 거리에 따른 힘을 구하여 최종적으로 캐릭터에 적용될 가속도를 구한다. 캐릭터에서 파생되는 점선은 밀개에 의한 반발력이 적용된 실제 움직임 곡선이다.

4. 구현 및 분석

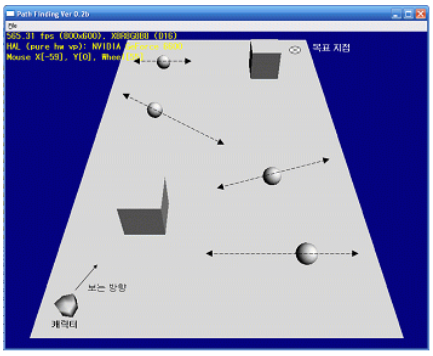
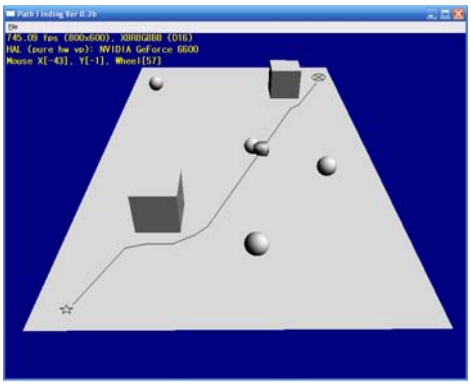


그림 6 경로탐색 시스템 구현결과

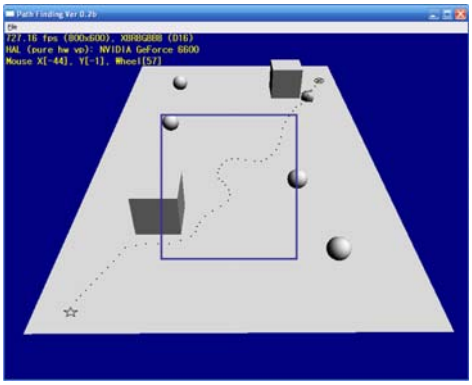
본 논문에 제시된 경로탐색 시스템을 실제 3D

게임에서의 적용 가능성을 실험하기 위해서 간단한 지형과 장애물, 그리고 움직이는 객체들을 구성하여 구현해보았다.

그림 6 에서 평평한 지형데이터에 2 개의 고정 장애물 그리고 4 개의 동적 객체가 배치되어있는 것을 볼 수 있다. 동적 객체들은 부근의 표시된 화살표 방향으로 각각 다른 일정한 속력으로 움직인다. 이미 상용화되어 있는 게임을 살펴보면, 하나의 방에서 캐릭터의 이동에 방해가 될 만한 이동성이 있는 객체는 많은 수가 아니다.



(a) 동적 객체를 무시한 탐색



(b) 동적 객체를 적용한 탐색

그림 7 구현 결과

그림 7(b)는 본 논문에서 제시한 시스템을 통하여 시작지점에서 목표지점까지 경로를 탐색한 결과이다. 그림 7(a)는 중간에 충돌이 생기는 반면, 그림 7(b)의 네모 박스에는 움직이는 객체의 영향을 받아 회피하며 탐색했던 경로를 보여준다.

5. 결론 및 향후 과제

본 논문에서 제시한 시스템은 계층적 네비게이션 메시와 최적화된 A* 그리고 밀개를 이용하여 동적인 3D 게임 환경에서 움직이는 객체를 회피하면서 자연스러운 경로로 탐색함을 보여주었다. 제시된 경로탐색 시스템은 동적인 환경에서 특히 적합하다. 밀개를 통한 자연스러운 경로탐색을 위하여 최적 경로를 찾는 A*를 수행하지 않고 대강의 경로를 빠르게 탐색하고 탐색도중에 밀개의 영향력을 계산하는데 초점을 맞추어 수행하였다. A*의 최적화는 많은 연구가 되었으므로 비약적인 개선은 쉽지 않은 문제이다. 그러나 여전히 탐색 공간을 효율적으로 자동 구성하는 연구는 필요하며, 여러 집단을 이루어 이동하는 동적 게임 환경에서 끌개와 밀개를 적용할 수 있는 방법에 대한 연구가 필요하다.

참고문헌

[1] Ning Jing, Tun-wu Huang, Eake A. Rundensteiner “Route Guidance Support in Intelligent Transportation System: An Encoded Path View Approache”, University of Michigan Techical Report, 1995

[2] 정동민, 김형일, 김준태, 엄기현, 조형제, “가시성 그래프와 A* 알고리즘을 이용한 3D game 에서의 효율적인 경로 탐색”, 2004

[3] Paul Tozour, “Search Space Representations”, AI Game Porgramming Wisdom 2, 2003

[4] Stout, Bryan, “The Basic of A* for Path Planning”, Game Progamming Gems, Charles River Media, 2000

[5] Paul Tozour, “Building a Near-Optimal Navigation Mesh”, AI Game Programming Wisdom, 2002

[6] Stout, Bryan, “Smart Moves: Intelligent Path-Finding”, Game Developer, 1996

[7] Rabin, Steve, “A* Speed Optimizations”, Game Programming Gems, Charles River Media, 2000

[8] John Olsen, “Attractors and Repulsors”, Microsoft, 2004

[9] O'Rourke Joseph, “Computational Geometry in C”, Second Edition, Cambridge University Press, 1994

[10] Daniel Higgins, “How to Achieve Lightning Fast A*”, AI Game Programming Wisdom, 2002