한국게임학회 논문지
제4권 제3호 2004년 9월

실시간 A* 길 찾기와 동적 그래프 문제를 위한 계층적 그래프 구조와 연산자

김태원* , 조경은** , 엄기현***
동국대학교 정보산업대학 컴퓨터멀티미디어공학과 멀티미디어공학전공

A Hierarchical Graph Structure and Operations for Real-time A* Path finding and Dynamic Graph Problem

Taewon Kim*, Kyungeun Cho**, Kyhyun Um***
Dept. of Computer Multimedia Engineering, Dongguk University

요 약

RPG, 전략 시뮬레이션등의 2D/3D 게임에서 동적으로 변화하는 장애물이나 지형 정보 등을 관리하기에는 대체로 동적 그래프가 적합하다. 이 논문에서는 빠르게 길 찾기를 수행하고 동적으로 변경할 수 있는 고정 레벨의 계층적 그래프 모델을 제안한다. 공간 분류나 공간 모델을 이용해 그래프를 분할하여 계층적 그래프를 구성하고, 동적 그래프의 연산자들을 제시하여 계층적 그래프 모델에서의 실시간 A* 길 찾기 방법을 실험하였다. 본 논문에서 제안한 모델이 동적 장애물이나 동적 구조를 가지는 게임 환경에서 빠르게 길 찾기를 수행하기에 적합한 그래프 모델임을 실험을 통해 입증하였다.

Abstract

A dynamic graph is suitable for representing and managing dynamic changable obstacles or terrain information in 2D/3D games such as RPG and Strategy Simulation Games. We propose a dynamic hierarchical graph model with fixed level to perform a quick A* path finding. We divide a graph into subgraphs by using space classification and space model, and construct a hierarchical graph. And then we perform a quick path finding on the graph by using our dynamic graph operators. With our experiments we show that this graph model has efficient properties for finding path in a dynamic game environment.

Key words : A* 길찾기 알고리즘, Path finding, 동적 장애물, 동적 그래프

1. 서론

RPG, 전략 시뮬레이션 등의 2D/3D 게임에서는 장애물이나 지형 등이 변경되는 경우가 빈번하다. 예를 들어, 게임에서 2개의 문을 가진 커다란 성을 대포로 공격해서 외벽에 구멍을 내는 경우를 생각해 보자. 그러면 기본적인 2개의 문외에도 1개의 구멍으로 성으로 들어갈 수 있다. 그 반대로 마법사가 성의 성문을 마법으로 외벽으로 바꿀 수 있다. 그러면 이제 성으로 들어갈 수 있는 길은 1개 밖에 존재하지 않는다. 이와 같이 게임에서 동적으로 변하는 장애물이나 지형 정보등을 반영하고 관리하기에는 대체로 동적 그래프가 적합하다. 말하자면, 동적 그래프에 정점(vertex)과 에지(edge)를 실시간에 삽입하고 삭제하거나, 정점의 위치와 에지의 연결을 바꾸어 관리할 수 있다. 또한, 그러한 동적 그래프에 대해 길 찾기 연산도 필요하다. 대체로, A* 길 찾기 알고리즘을 많이 사용한다. 이 알고리즘은 가장 빠른

길 찾기 알고리즘이지만 거대하고 복잡한 그래프에서 실시간으 로 길 찾기를 하기에는 몇 가지 약점이 존재한다. 먼저 시작점과 목표점이 연결되어 있지 않으면 모든 노드 를 탐색하며, 매우 많은 메모리와 시간을 소모한다. 한 프레임에 많은 개체들이 길 찾기를 하는데 대부 분의 경우 한 프레임에 시작점에서 목표점까지 이동하지 못한다 . 따라서 실시간에 객체의 속력 이상의 거리를 탐 색하는 것은 CPU와 메모리를 불필요하게 낭비할 수 있다. 이러한 문제점을 해결하고 빠르게 길 찾기를 수행하기 위해서 계층적 그래프를 구성한다[1]. 계층 적 그래프를 동적으로 변경하면 계층적 그래프의 계층 구조 와 연결 정보도 변경되어야 한다. 본 연구에서는, 이러한 문제점를 효율적으로 해결하기 위해, 그래프를 분할하고 각 분할된 그래프를 이용해 빠르게 길 찾기를 수행하며 동적 그래프 문제를 해결할 수 있는 계층적 그래프 모델을 제안한다. 이를 위 한, 본 논문의 구성은 다음과 같다. 2장에서 논문의 주 제와 관련된 연구들을 살펴보고, 3장에서는 계층적 그래프 구성방법을 설명하고, 4장에서는 동적 그래프 의 연산 알고리즘을 소개한다. 5장에서는 속도 향상 을 개선하기 위해 동적 그래프의 단순화된 연산 알고 리즘을 설명한다. 6장에서는 본 논문에서 제안한 동적 그래프 문제를 위한 계층적 그래프 모델에서의 실시간 A* 길 찾기 방법을 설명한다. 7장에서는 실험 방법 및 분석 내용을 기술하고, 마지막에 본 연구를 통해 얻은 연구 결과를 종합하고 앞으로 필요한 연구들을 제안한다.

2. 관련 연구

A* 기법들중 거대한 그래프에서 효과적으로 길을 결정하기 위해 계층적 길 찾기(hierarchical pathfinding) 방법이 사용된다[2] 이것은 검색 대상 공간을 계층적으로 분할하고, 먼저 큰 범위 공간에서 경로를 찾은 후 그보다 하위 범위 공간들에서 개별적으로 경로를 찾는 방식이다[3]. 계층적 길 찾기 방식은 게임분야뿐만 아니라 지능 교통 정보 시스템에서도 최단거리를 빨리 구하기 위하여 사용한다[4,5,6,7]. 또한, 주어진 그래프를 임의의 Partition으로 분할하고 다른 Partition과 연결되는 정점을 선택해 Super Graph를 구성해 시작 정점에서 목표 정점까지 가장 짧은 경로를 유도해내는 계층적 길 찾기를 구현하기도 한다. 계층적 그래프에서의 동적 그래프 문제를 해결하기 위해서 정점들 간의 상호연결성을 빠르게 검사하는 것이 중요하다. Henzinger는 동적 그래프의 정점들 간의 상호연결성을 검사하는 알고리즘을 제시했다 [8,9].

길 찾기 문제에서 취급되는 또 다른 중요한 문제는 객체의 이동 과정 중의 충돌 검사이다. 이 때 충돌 대상이 되는 장애물은 정적 장애물과 동적 장애물로 나눌 수가 있다. 특히 동적 장애물들을 그래프에 표현하고 동적 장애물 처리를 위한 네비게이션 메시 기술들이 개발되었다[1].

본 연구에서는 효과적인 길 찾기를 위한 동적 장애물과 동적 구조를 빠르게 반영할 수 있는 계층적, 동적 그래프 모델을 제안한다.

3. 계층적 그래프

계층적 그래프는 여러 계층의 그래프의 집합이다. 주어진 그래프를 원천 그래프라고 부르기로 정하며, 계층적 그래프의 최하층이 된다. 그 그래프를 여러 개의 부분그래프(subgraph)로 나눈 후, 부분그래프를 하나의 정점으로 표현하여 그 정점들만의 그래프로 구성하여 상위층으로 삼은 것이다. 이를 계층적 pivot 그래프라고 부른다. 최상위층은 루트라는 하나의 정점으로 구성한다. 계층은 2개 이상일 수 있다. 이 장에서는 계층적 그래프를 구성하는 계층적 pivot 그래프의 생성 방법을 설명한다.

3.1 원천 그래프 생성과 영역 분할

이 절에서는 계층적 그래프를 구축하기 위한 첫 단계로 영역 분할과 인덱스 부여 방법을 설명한다.

한 게임 공간은 네비게이션 메시로 표현한다. 이 네비게이션 메시에서 에지의 중점을 하나의 정점으로 생성하고 네비게이션 메시의 각각의 면에 인접한 에지를 서로 연결하여 원천 그래프를 생성한다. 이것은 무방향 그래프이다. [그림 1]은 2D 게임의 네비게이션 메시로부터 원천 그래프 G를 생성한 예이다.

개발자가 원하는 형태로 계층적 그래프를 구축하기 위해서는 먼저, 원천 그래프를 분할하여 영역들로 나눈다. 이 분할영역은 원천 그래프 G의 임의의 정점 집합으로, 보통 객체의 경계 볼륨이나 게임의 공간 분류 등 그래프 G의 서로 강하게 결합하는 정점 집합을 의미한다. 각 분할영역에 대해 일련 번호로 인덱스를 부여한다. 예로 보인 [그림 2]에서

나타나는 사각형의 번호가 분할영역별 인덱스이다.

각 분할영역내의 원 번호는 분할영역 인덱스와 같은 것이며, 각 분할영역에 속한 원천 그래프 의 부분 그래프를 구성하는 정점들이다. 경계선 상 에 있는 정점에는 분할영역 인덱스가 작은 순으로 순차 적으로 부여한다.

3.2 분할영역의 부분 그래프의 구분

인덱스가 부여된 분할영역이 결정되면, 각 분할영역에 서로 연결되지 않는 부분 그래프가 2개이상 있을 때 진행된다. 서로 연결되지 않는 부분그래프를 구별할 수 있도록 서로 다른 인덱스 번호가 부여한다.

3.3 계층적 Pivot 그래프 구성

다음으로 원천 그래프에 대한 상위계층의 계층적 pivot 그래프를 구성한다. 계층적 pivot 그래프는 pivot 정점들로 구성되는 그래프이다. pivot 정점은 한 부분 그래프의 대 표점이다. pivot 정점을 결정하는 방법으로는, 각 부분 그래프에 대한 AABB (Axis-Align Bounding Box) 또는 MBR (Minimum Bound ing Rectangle) 영역을 구성하고, 이에 속한 정점의 최대, 최소 좌표를 이용해 그 영역 의 평균 위치를 정하여 pivot 정점으로 정의한다. 이렇게 결정된 pivot 정점들로 계층적 pivot 그래프를 만든다.

[그림 4]는 [그림 3]으로부터 MBR을 이용하여 계층적 pivot 그래프를 구성한 예이다. 점선의 에지들로 구성된 것이 계층적 pivot 그래프이다. 사각형들은 MBR이다. [그림 4]에 표시된 중앙의 큰 정점은 계층적 pivot 그래프에 대한 상위계층의 pivot 정점으로 루트가 된다.

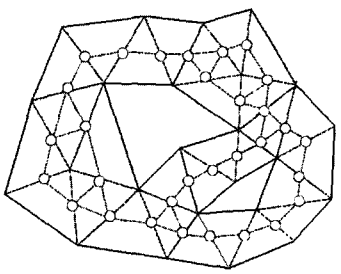

그림 1. 네비게이션 메시와 원천 그래프

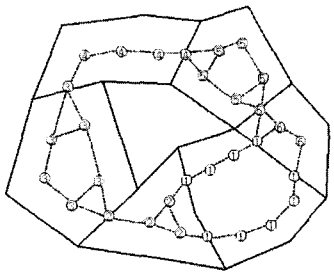

그림 2. 분할영역에의 부분그래프와 인덱스 부여

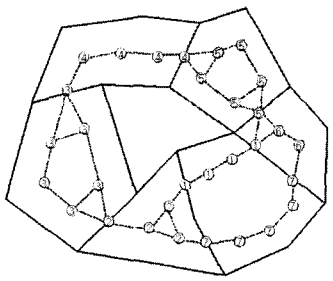

그림 3. 구분된 부분 그래프의 원천그래프

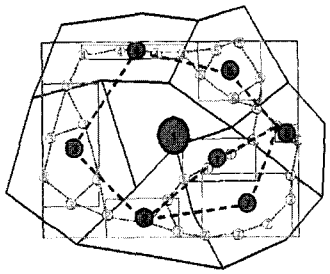

그림 4. 계층적 pivot 그래프의 구성 예

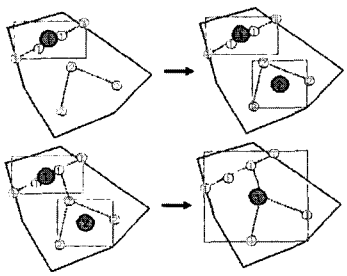

그림 5. 계층적 pivot 그래프에 대한 Insert(v, G)와 insert(e, G) 연산의 활용 예

4. 계층적 그래프의 동적 관리

계층적 그래프에 대해서 실시간에 정점과 에지를 삽입하고 삭제할 수 있다. 그리고 정점의 위치와 에지의 연결을 바꿀 수도 있다. 동적 그래프 연산에는 정점의 삽입과 삭제, 에지의 삽입과 삭제 등 4개의 단위 연산을 고려할 수 있다. 이 연산들로 계층적 그래프를 동적으로 변경하면 연쇄적으로 계층적 그래프의 계층 구조와 연결 정보가 변경될 수 있다. 이 장에서는 제안하는 계층적 그래프에 대한 동적 관리 연산을 개념 중심으로 설명한다.

4.1 일반적인 동적 그래프 관리 연산

계층적 그래프에 동적 그래프 연산을 적용하는 방법으로 다음 4개의 단위 연산 을 고려할 수 있다.

Insert(v, G) : 원천 그래프 G에 정점 v를 추가한다.

Insert(e, G) : 원천 그래프 G에 에지 e를 추가한다.

Delete(v, G) : 원천 그래프 G에서 정점 v를 삭제한다.

Delete(e, G) : 원천 그래프 G에서 에지 e를 삭제한다.

Insert(v, G)를 수행하면 원천 그래프 G에 정점을 추가하고 Insert(e, G)를 이용하여 필요한 에지를 추가한다. Insert(e, G)를 수행하여 같은 분할 영역 내에 있는 두 개의 부분 그래프가 연결되면 계층적 pivot 그래프도 수정한다. Delete(v, G)를 수행하면 정점을 삭제하고 그 정점에 연결된 에지를 Delete(e, G)로 삭제한다. Delete(e, G)를 수행하여 부분 그래프가 분리되면 계층적 pivot 그래프도 수정한다. [그림 5]는 하나의 에지를 삽입하여 두 개의 부분 그래프가 하나로 연결되는 경우를 예로 보인 것이다.

게임 공간에 하나의 객체를 추가하거나 제거하면서 연관된 정점이나 에지를 필요한 만큼의 단위 연산들을 이용하여 관리한다. 관리가 그 만큼 복잡해진다. 또한, 계층적 pivot 그래프를 수정할 때, pivot 정점을 고정시키면 갱신 속도는 빠르지만, 정확도가 떨어질 수 있다. pivot 정점을 동적으로 수정하는 것은 O(h) (h는 계층 단계)로 비교적 속도가 빠르지만 동적 그래프 연산마다 계속 점진적으로 계층 구조를 갱신해야 한다.

4.2 계층적 pivot 그래프의 관리 연산

상호 연결성을 검사하기 위해 사용한 연결 그래프 분할 알고리즘의 복잡도는 O(kn2)이므로 실시간에 계산하기에 부담이 너무 크다. 만약 동적 그래프 연산의 결과가 상호 연결성을 명시적으로 알 수 있다면, 연결 그래프 분할 알고리즘을 사용할 필요가 없고, 수행 속도는 훨씬 더 빨라질 수 있다. 이 장에서 제안하는 최적화된 동적 그래프 연산은 명시적으로 알 수 있다. 최적화된 동적 그래프 연산은 다음과 같이 정의한다.

Update(v, G) : 그래프 G에 속한 정점 v의 위치를 수정한다.

Update(e, G) : 그래프 G에 속한 에지 e를 삭제/삽입한다.

Construct(e, v, G) : 그래프 G에 속한 정점 v와 에지 e를 이용해 연결 그래프를 만든다.

Destruct(e, v, G) : 그래프 G에 속한 정점 v와 에지 e를 이용해 연결그래프를 만든다.

Connect(e, C1, C2, G) : 그래프 G에 속한 연결 그래프 C1과 C2를 연결한다.

Disconnect(e, C1, C2, G) : 그래프 G에 속한 연결 그래프 C1과 C2를 분리한다.

4.3 최적화된 동적 pivot 그래프 연산

Update(v, G)를 수행하면 v의 위치를 수정하고, pivot을 수정한다. Update(e, G)를 수행하면 v에 연결된 부분 e를 수정한다. 상호 연결성에 영향력을 주지 않는다고 가정했기 때문에 별도의 연결성 검사는 하지 않는다 [그림 6].

Construct(e, v, G)를 정점을 추가하고 에지를 추가해서 연결 그래프를 만든다. 1개의 연결 그래프로 가정하기 때문에 상호 연결성 검사가 필요 없다. Destruct(e, v, G)를 수행하면 연결 그래프의 모든 정점과 모든 에지를 삭제한다 [그림 7].

Connect(e, C1, C2, G)를 수행하면 pivot 그래프를 서로 연결하고 pivot 그래프의 에지에 하위 그래프의 에지를 추가한다. Disconnect(e, C1, C2, G)를 수행하면 pivot 그래프를 서로 연결하고 상위 그래프의 에지에서 pivot 그래프의 에지를 삭제한다 [그림 8].

4.4 최적화된 동적 영역 그래프 연산

Update(v, G)를 수행하면 V의 위치를 수정하고, 영역 내부라면 영역을 수정하지 않고 영역 외부라면 영역을 수정한다.Update(e, G)를 수행하면 V에 연결된 부분 E를 수정한다. 상호 연결성에 영향력을 주지 않는다고 가정했기 때문

에 별도의 연결성 검사는 하지 않는다 [그림 9].

Construct(e, V, G)를 수행하면 정점을 추가하고 에지를 추가해서 연결 그래프를 만든다.Destruct(E, V, G)를 수행하면 연결 그래프의 모든 정점과 모든 에지를 삭제한다 [그림 10].

Connect(e, C1, C2, G)를 수행하면 영역 그래프 C1, C2를 서로 연결하고 영역 그래프의 에지 (C1, C2)에 그래프의 에지 E를 추가한다. 영역 그래프의 상위 영역이 있고 상위 영역의 내부라면 상위 영역을 수정하지 않고 상위 영역의 외부라면 상위 영역을 수정한다.

Disconnect(e, C1, C2, G)를 수행하면 상위 그래프를 서로 연결하고 상위 그래프의 에지에서 하위 그래프의 에지를 삭제한다. 영역 그래프의 상위 영역이 있고 상위 영역의 내부라면 상위 영역을 수정하지 않고 상위 영역의 외부라면 상위 영역을 수정한다 [그림 11].

4.5 느슨한 계층적 영역 그래프

느슨한 영역을 사용하면 꽉 죄인 영역을 사용할 때보다 정확도가 떨어지지만 영역의 크기 계산을 보다 최소화할 수 있다 [그림 12]

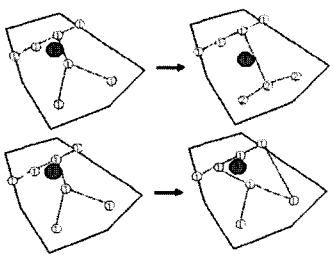

그림 6. Update 연산

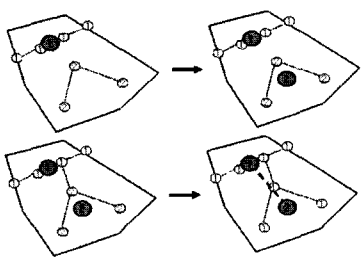

그림 7. Connect + Construct로 구현한 삽입 연산

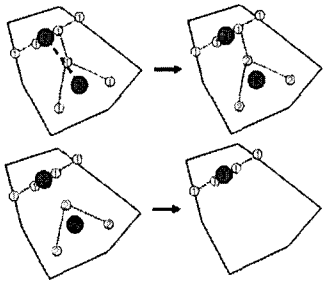

그림 8. Disconnect + Destruct로 구현한 삭제 연산

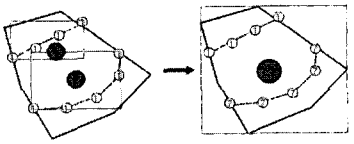

그림 9. 느슨한 영역 그래프

5. 계층적 A* 길 찾기

A* 길 찾기 알고리즘은 가장 빠른 길 찾기 알고리즘이지만 거대하고 복잡한 그래프에서 실시간 길 찾기를 하기에는 몇 가지 약점이 존재한다. 먼저 한 프레임에 많은 개체들이 길 찾기를 할 수 있다. 대부분의 경우 한 프레임에 시작점에서 목표점까지 이동하지 못한다. 따라서 실시간에 객체의 속력 이상의 거리를 탐색하는 것은 CPU를 불필요하게 낭비할 수 있다.

5.1 연결성 검사

시작점과 목표점이 연결되어 있지 않으면 A* 길 찾기 알고리즘은 모든 노드를 탐색하며 매우 많은 메모리와 시간을 소모한다. 이러한 것을 방지하기 위해 A* 길 찾기를 수행하기 전에 시작점과 목표점이 연결되어 있는지 검사해야 한다. 시작점과 목표점의 분할영역 인덱스를 구한다.

만약 분할영역 인덱스를 존재한다면, 다시 분할영역의 분할영역을 구한다. 이 과정을 반복해 분할영역이 0이 되기 전까지 이 과정을 반복한다. 분할영역 인덱스가 서로 같으면, 연결되었고, 서로 다르면 연결되지 않는다. 연결성 검사

의 시간 복잡도는 O(h)(h는 계층 단계)이다. [그림 13]에서 시작점 8과 목표점 9의 분할영역 인덱스는 각각 7과 4이다. 7과 4의 분할영역인덱스는 10으로 결국 8과 9는 서로 연결되어 있다.

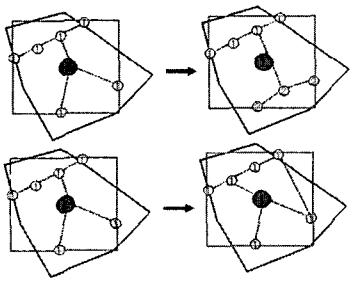

그림 10. Update 연산

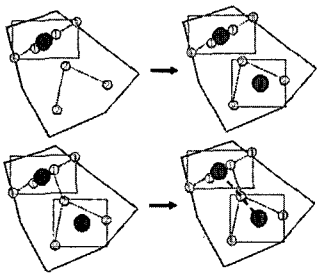

그림 11. Connect + Construct로 구현한 삽입 연산

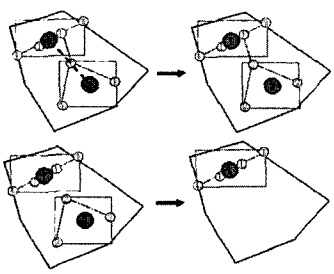

그림 12. Disconnect + Destruct로 구현한 삭제 연산

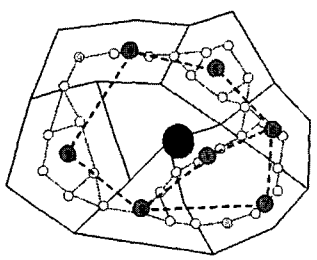

그림 13. 계층적인 길찾기 예제

5.2 거시적인 길 찾기

시작점과 목표점이 연결되어 있다면 이제 본격적인 길 찾기 알고리즘을 수행한다. 거대하고 복잡한 그래프는 A* 길 찾기 알고리즘을 수행할 때 매우 오랜 시간이 걸릴 수 있으므로 원래의 그래프를 단순화시킨 pivot 그래프를 사용해서 길을 찾아야 한다. pivot 그래프에 대해 A* 길 찾기 알고리즘으로 경로를 찾은 뒤, 시작점과 목표점을 제거한다. 그리고 원래 그래프의 목표점을 추가한다.

시작점 8과 목표점 9의 분할영역 인덱스는 각각 7과 4이다. 따라서 pivot 정점 7번과 4번을 시작점과 목표점으로 pivot 그래프에 대해서 A* 길 찾기 알고리즘을 수행한다. A* 길 찾기 알고리즘을 이용해서 pivot 정점 7번과 4번 사이의 경로를 찾은 pivot 그래프 경로가 {7, 6, 5, 4 }라고 가정해 보자. 이 결과에서 시작점과 목표점을 각각 제거한 뒤, 목표점 9를 추가하면 경로는 {6, 5, 9}가 된다. 만약 시작점과 목표점이 같은 연결그래프 내에 있다면 시작점과 목표점의 클러스터 인덱스는 같다. pivot그래프에 대해서 A* 길 찾기 알고리즘을 이용해서 길을 찾으면, pivot 그래프 경로는 시작점과 목표점의 클러스터 인덱스만 쌍으로 존재할 것이다. pivot 그래프 경로에서 시작점과 목표점을 제거하고 원래 그래프의 목표점을 삽입하면 경로는 목표점만 존재하게 된다.

5.3 미시적인 길 찾기 및 이동

미시적인 길 찾기는 거시적인 길 찾기에서 얻은 경로를 이용해 객체가 실제 이동할 경로를 찾는 과정이다. 미시적인 길 찾기에서 사용되는 A* 길 찾기 알고리즘은 pivot정점을 목표점으로 가지는 길 찾기 알고리즘과 실제 정점을 목표점으로 가지는 길 찾기 알고리즘 2가지가 존재한다.

시작점과 목표점이 서로 다른 클러스터에 존재한다면, 거시적인 길 찾기에서 구해진 경로는 pivot정점과 실제 목표점으로 구성되어 있을 것이다. 시작점과 목표점이 서로 같은 클러스터에 존재한다면, 거시적인 길 찾기에서 구해진 경로는 실제 목표점만 존재한다.

pivot 그래프는 실제 그래프를 단순화시킨 것이기 때문에 pivot 그래프와 실제 그래프는 연결되어 있지 않다. 하지만 실제 그래프에 pivot 정점을 클러스터 인덱스로 가진 정점은 존재한다. 그래서 pivot 정점을 목표로 하는 A* 길 찾기 알고리즘은 pivot정점까지의 직선거리를 휴리스틱으로 이용하며, 목표 pivot정점을 클러스터 인덱스로 가진 임의의

정점을 만나면 종료된다. 실제 정점을 목표로 하는 A* 길 찾기 알고리즘은 일반적인 A* 알고리즘과 같다.

시작점 8과 목표점 9의 거시적인 길 찾기 결과는 {6, 5, 9}라고 가정하자. 먼저 A* 길 찾기 알고리즘으로 시작점 8번에서 pivot 정점 6번까지의 직선거리를 휴리스틱으로 이용해 탐색한다. 실제 그래프의 정점 8번과 pivot 그래프 5번은 실제 연결되어 있지 않으므로 시작점 8번과 pivot 정점 6번까지의 경로는 실제 존재하지 않는다. 하지만 시작점과 pivot 정점 6을 클러스터 인덱스로 가진 정점과 서로 연결되어 있다. A* 길 찾기를 하면서 pivot 정점 6을 클러스터 인덱스로 가지는 실제 그래프의 정점을 만나면, 이제 탐색 목표는 pivot 정점 5가 된다. 마찬가지로 pivot 정점 5를 클러스터 인덱스로 가지는 실제 정점을 만나면 탐색 목표는 9가 된다. 목표점 9는 실제 그래프의 정점이고, 보통의 A* 길 찾기 알고리즘을 이용하면 된다.

거시적인 길 찾기에서 찾아진 경로는 연결 그래프끼리 서로 직접 연결된 경로이기 때문에 거시적인 길 찾기에서 찾아진 pivot정점을 가진 연결 그래프외의 다른 연결 그래프의 정점들은 탐색할 필요가 없다.

6. 실험 및 분석

실험 프로그램은 계층적 pivot 그래프에서의 A* 길 찾기 알고리즘을 구현했다. [그림 14]는 11개의 정점을 4개의 분할영역을 이용해 6개의 연결 그래프로 분할하고 만들어진 6개의 Pivot 정점을 이용하여 A* 길 찾기 알고리즘을 수행한 결과이다.

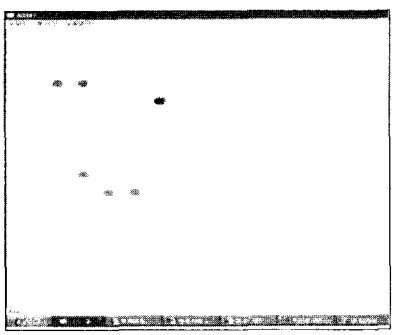

그림 14. 실험 프로그램

이 실험은 pivot(상위 그래프)정점을 이용한 미시적인 길 찾기 알고리즘을 구현한 것이다. 실험 목적은 pivot(상위 그래프)정점을 이용해 본 논문에서 제시한 미시적인 길찾기 알고리즘을 검증하는 것이다. 분할영역을 이용해 pivot그래프(상위 그래프)를 구축하고, pivot그래프(상위 그래프)의 pivot(상위 그래프)정점까지의 직선거리를 A* 길 찾기 알고리즘의 휴리스틱으로 이용하였다.

계층적 그래프를 생성하고 사용하기 위한 공간, 시간 복잡도 분석 내용을 기술한다. 일반 그래프와 계층적 피봇 그래프를 사용할 때의 공간 복잡도를 비교하면 표1과 같다. 일반 그래프를 사용하면 위치와 에지 정보가 필요하며, 계층적 피봇 그래프를 사용하면 추가적으로 클러스터 인덱스, 피봇 위치, 피봇 에지, 피봇 소속 정점 리스트가 필요하다. 그래서, 계층적 피봇 그래프를 사용하면 일반 그래프를 사용할 때보다 정점 인덱스를 short로 사용할 경우 약 40% 이상, int로 사용할 경우 약 66% 이상의 메모리 점유률이 증가한다는 단점이 있다. 또한, 정점(n개) 대비 피봇의 비율이 10%부터 30%로 정할 때 50%부터 70%까지 메모리 사용이 변화할 수 있다 [표2].

그렇지만, 계층적 피봇 그래프를 사용하면, 메모리 점유율이 거의 2배에 가깝게 증가하지만, 실제 정점과 피봇의 비율이 높을수록 A* 알고리즘에서 사용하게 되는 메모리 할당량은 감소한다. 실제로 A*알고리즘을 구현하기 위해서 탐색하는 각 노드는 부모 노드를 가리키는 포인터, 노드까지의 비용, 총 비용, 열린 목록에 들어 있는 지의 여부, 닫힌 목록에 있는 지의 여부에 관한 정보를 포함하며, 최소 20 Byte이다. 결과적으로 계층적 피봇 그래프를 사용함으로써 총 메모리 사용율을 약 20% 줄일 수 있다[표3].

다음은 계층적 피봇 그래프 생성과정의 성능을 분석한 결과이다. 그래프 생성, 공간 분류, 연결성 분할, 계층적 피봇그래프 생성, 동적 그래프 연산에 드는 비용을 나타낸다 [표4]. 표5는 동적 그래프 연산과 최적화된 동적 그래프 연산을 성능 비교하였다. 표에서 볼 수 있듯이 삭제 연산의 속도가 향상됨을 볼 수 있다.

7. 결론 및 향후 과제

계층적 그래프 모델은 그래프를 단순화시켜 빠르게 A* 길 찾기를 수행할 수 있다. 이 논문에서 계층적 그래프 모델과

임의의 클러스터를 이용해 모델을 구성하는 알고리즘, 그리고 동적으로 그래프를 변경하는 연산과 알고리즘에 대해서 발표했다. 또 제시된 4가지 그래프 표현 모델을 이용해 각각의 상황에 적합한 계층적 그래프 모델을 선택할 수 있다.

그래프 종류	그래프 요소	크기	
		2차원	3차원
일반 그래프	위치	8Byte*n	12Byte*n
	에지	4Byte*n	4Byte*n
계층적인 그래프	위치	8Byte*n	12Byte*n
	에지	4Byte*n	4Byte*n
	클러스터 인덱스	4Byte*n	4Byte*n
	피봇 위치	8Byte*m	12Byte*m
	피봇 에지	4Byte*m	4Byte*m
	피봇 소속 정점 리스트	4Byte*n	4Byte*n

표 1. 계층적 피봇 그래프 생성과정 공간 복잡도

정점 인덱스	정점과 피봇의 비율	2차원
short (2Byte)	10%	약 50% 이상 증가
	20%	약 60% 이상 증가
	30%	약 70% 이상 증가
int (4Byte)	10%	약 76% 이상 증가
	20%	약 86% 이상 증가
	30%	약 96% 이상 증가

표 2. 계층적 피봇 그래프에서의 정점(n개)과 피봇의 비율에 따른 메모리 사용 변화

	정점과 피봇의 비율	그래프 크기	노드 할당 크기	합계
일반적인 A* 길찾기	-	100%	100%	200%
계층적인 A* 길찾기	10%	150%	10%	160%
	20%	160%	5%	165%
	30%	170%	3.3%	173.3%

표 3. 피봇의 비율에 따른 계층적 A*알고리즘과 일반적인 그래프의 메모리 사용비교

과정	세부기능	시간복잡도
그래프 생성	그래프 데이터 로드	O(n)
	분할영역 인덱스 초기화	O(n)
공간 분류	분할영역 로드	O(m)
	분할영역 인덱스 할당	O(n)
연결성 분할	BFS 탐색	$O(n^2)$
	탐색된 정점 제거	O(n)
	연결되지 않은 정점의 수만큼 BFS 탐색	$O(kn^2)$
계층적 피봇 그래프 생성	연결 그래프의 Pivot 정점의 위치 계산	O(n)
동적 그래프 연산	연결 그래프의 인접 정점의 클러스터 인덱스 조사	O(n)

표 4. 계층적 피봇 그래프 생성과정 성능분석표

연산자	세부기능	시간복잡도
동적 그래프 연산	삽입	O(n)
	삭제 (연결성 분할 포함)	$O(kn^2)$
최적화된 동적 그래프 연산	수정	O(1)
	생성	O(n)
	파괴	O(n)
	연결	O(1)
	분리	O(1)
	삽입(생성 +연결)	O(n)
	삭제(분리 + 파괴)	O(n)

표 5. 동적 그래프 연산과 최적화된 동적 그래프 연산 성능 비교표

고정/동적 Pivot 그래프 모델은 간단한 길 찾기에 유리하고 구조가 간단하다. 영역 그래프 모델은 다중 계층에 유리하고 상대적으로 동적 그래프 연산을 할 수 있다. 느슨한 영역 그래프 모델은 정확도를 포기하는 대신에 가장 빠르다. 동적인 계층적인 그래프를 통해 계층적인 A* 길 찾기, 계층적인 그래프 모델 구성, A* 길 찾기의 미적 최적화가 가능하다.

참고문헌

[1] Paul Tozour, "Influence map method", Game Programming Gems 2

[2] Stout, Bryan W., Smart Moves: Intelligent Path-Finding, Game Developer

[3] Ning Jing, Yun-wu Huang, Elke A. Rundensteiner "Route Guidance Support in Intelligent Transportation System: An Encoded Path View Approach*" , University of Michigan Technical Report, 1995

[4] Ning Jing, Yun-wu Huang, Elke A. Rundensteiner "Route Guidance Support in Intelligent Transportation System: An Encoded Path View Approach*" , University of Michigan Technical Report, 1995

[5] Ning Jing, Yun-wu Huang, Elke A. Rundensteiner "Hierarchical Optimization of Optimal Path finding for transportation Applications", Changsha Institute of Technology, 1996

[6] Ning Jing, Yun-wu Huang, Elke A. Rundensteiner "Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation", 1998

[7] Kihong, Kim et al "A Partitioning Scheme for Hierarchical Path Finding Robust to Link Cost Update", School of Electronical Engineering, Seoul National University, Korea, 1998

[8] M.R. Henzinger "Fully Dynamic Biconnectivity in Graph", Algorithmica, 1992

[9] Valerie King, "Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraph", 1999

[10] Tomas, Leiserson et al, "Introduction to Algorithms", MIT Press, 1990

[11] Algorithms and Theory of Computation Handbook, CRC Press LLC, 1999

조 경 은

동국대학교 정보산업대학 컴퓨터멀티미디어공학과
1989.3 ~ 1993.2 동국대학교, 전자계산학과(공학사)
1993.3 ~ 1995.2 동국대학교, 컴퓨터공학과 대학원(공학석사)
1995.3 ~ 2001.8 동국대학교, 컴퓨터공학과 대학원(공학박사)
2001.8 ~ 2002.2 동국대학교, 산업기술연구소 연구원
2002.3 ~ 2003.2 안양대학교, 디지털미디어학부 강의전담교수
2003.3 ~ 2003.8 영산대학교, 멀티미디어공학부 게임공학과 전임강사
2003.9 ~ 현재 동국대학교, 정보산업대학 컴퓨터멀티디어공학과 전임강사
관심분야 : 컴퓨터 게임 알고리즘, 게임 인공지능, 컴퓨터 그래픽스, 멀티미디어 정보처리

엄 기 현

동국대학교 정보산업대학 컴퓨터멀티미디어공학과
1978.3 ~ 현재 동국대학교 컴퓨터멀티미디어 공학과 정교수
2001.3 ~ 2003.2 동국대학교 정보산업대학 학장
1995.3 ~ 1999.2 동국대학교 정보관리처 처장
1997.1 ~ 2002.12 한국 정보과학회 이사
2001.1 ~ 2002.12 한국 정보과학회 논문지 편집위원회 부위원장(데이타베이스 논문지 담당)
1998.8 ~ 2000.7 한국 정보과학회 데이타베이스연구회 운영위원장
1997.9 ~ 1999.8 한국 정보전산기관협의회 상임이사
1998.12 ~ 2001.12 한국 멀티미디어학회 부회장
1999.4 ~ 현재 Int. Conf. on Database Systems for Advanced Applications Steering Committee 위원
2004.1 ~ 현재 한국 게임학회 부회장
관심분야 : 멀티미디어 데이터베이스 & 응용, 게임 시스템 디자인과 게임 데이터베이스 응용

김 태 원

동국대학교 정보산업대학 멀티미디어공학과
2000.3 ~ 현재 동국대학교, 멀티미디어공학과 재학중
관심분야 : 컴퓨터 게임 알고리즘, 게임 인공지능