Journal of Digital Contents Society Vol. 9 No. 4 Dec. 2008(pp. 569-581)

윈도우 기반 네비게이션 메쉬에서 곡선 생성 기법

김형일*

요 약

본 논문에서는 윈도우 환경을 기반으로 하는 가상현실이나 게임에서 사용되는 네비게이션 메쉬(navigation mesh)에서 곡선을 생성하는 기법을 제안한다. 윈도우 환경을 기반으로 하는 가상현실이나 게임에서 주로 사용되는 지형 생성 방식에는 그리드(grid) 방식과 네비게이션 메쉬 방식이다. 그리드 방식은 지형을 생성할 때 격자를 이용함으로 자연스러운 지형을 생성하기 어렵다. 또한 정밀한 묘사를 위해서는 많은 수의 격자가 필요하여 시스템에 부담을 준다. 이러한 문제를 해결하기 위해 네비게이션 메쉬가 등장하였으며, 네비게이션 메쉬는 그리드 방식에 비해 시스템에 부담을 주지 않는다는 장점이 있다. 그러나 이러한 네비게이션 메쉬에서 캐릭터가 이동하게 되면 부자연스럽게 움직이게 되는 문제가 발생한다. 캐릭터의 부자연스러운 이동 문제를 해결하기 위해 본 논문에서는 구형 곡선을 이용한 곡선 생성 기법을 제안한다. 본 논문에서 제안한 구형 곡선 생성 기법을 윈도우 기반 가상현실이나 게임에 이용하면 캐릭터가 자연스럽게 이동할 수 있는 장점이 있다.

Generating Curved Lines in a Windows based Navigation Mesh

Hyungil Kim*

Abstract

In this paper, methods are proposed for creating a curved line in a navigation mesh, which is used in a windows based virtual reality or games. The main geographies used in a windows based virtual reality or games are created by using either grids or navigation meshes. When using grids, it is difficult to create a natural geography. Multiple grids are needed to ensure precise descriptions, which can result in system overload. A navigation mesh was developed to address this issue since the navigation mesh has the benefit of placing fewer loads on the system than a grid method. The drawback, however, is that the movements of character in navigation mesh are less natural. In this paper, a method for creating a curved line using round curves is proposed in order to resolve this issue. Using the methods for creating a curved line proposed in this paper in a windows based virtual reality or games will eliminate the unnatural or awkward movement of characters that arises in the use of a navigation mesh.

Keywords : Curved Line, Navigation Mesh, Path Finding, Virtual Reality

1. 서론

윈도우 기반 가상현실이나 게임에서는 현실세계와 같은 현실감을 중요하게 여기며, 이러한 현실감을 나타내기 위해 복잡한 표현 방법을 요구하는 경우가 많아 높은 하드웨어 성능을 요구한다[1][2]. 특히 현실감을 나타낼 때 어려운 부분은 지형이다. 지형은 가상현실이나 게임에서 가장 많은 영역을 차지하는 부분이기도 하고, 캐릭터가 이동하기 위한 기본 수단이기 때문에 많은 제약조건을 갖게 된다. 지형을 현실감 있게 표현하기 위해서는 지형을 세밀하게 묘사해야 하며, 세밀한 지형공간의 묘사는 시스템에 많은 부담을 줄뿐만 아니라, 이동경로 생성에도 많은 시간을 요구한다. 시스템의 부하를 줄이고 캐릭터의 이동경로 생성을 빠르게 표현하기 위해서는 지

※ 제일저자(First Author) : 김형일
접수일:2008년 10월 29일, 심사일:2008년 12월 02일
* 나사렛대학교 멀티미디어학과 교수
hkim@kornu.ac.kr
▣ 본 논문은 2008년도 나사렛대학교 학술비 지원에 의하여 연구되었음

형을 단순화하여 표현하는 것이 중요하다[2].

지형 표현 방법에는 그리드 방식과 네비게이션 메쉬 방식이 있다[3][4]. 그리드 방식은 사각형 격자를 이용하여 지형을 표현하므로 이동경로 탐색에 유용한 방법이다. 그러나 세밀한 지형을 묘사하기 위해서는 많은 수의 격자가 필요하게 되어 시스템에 많은 부하가 작용한다. 또한 그리드 방식을 이용하여 지형을 표현하였을 경우에 캐릭터의 이동경로는 부자연스러운 꺾은선으로 나타나는 단점이 있다[3]. 이와 같이 이동경로가 꺾은선으로 나타나는 이유는 이동경로를 빠르게 생성하기 위해서 지형에 표현된 이동경로를 사용하기 때문이다. 이러한 꺾은선은 캐릭터의 이동에 현실감을 떨어뜨리는 주요한 문제로 작용한다. 그리드 지형에서 이동경로를 자연스럽게 표현하기 위해 격자를 세밀하게 나누어, 꺾은선 부분에서 자연스러운 이동이 이루어질 수 있도록 처리하기도 한다. 그러나 이러한 격자 분할 방법은 여전히 격자를 기반으로 함으로써 완전한 곡선화는 이룰 수 없으며, 많은 격자를 생성하기 때문에 시스템에 부담을 주는 문제가 발생한다.

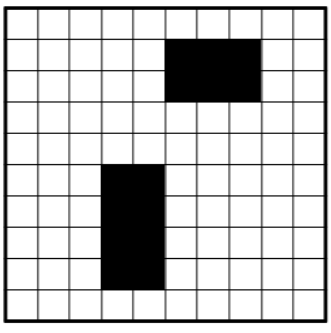
이러한 그리드 지형에서 발생하는 시스템 부하를 줄이기 위해 웨이포인트 방식을 사용하기도 한다. 캐릭터가 지형의 모든 영역을 이동할 수 있다면 시스템은 캐릭터가 이동하는 지역을 탐색할 때마다 지형 전체를 검색해야 하는 문제가 발생하기 때문에 이동경로 생성에 많은 시간이 소요된다. 이러한 시스템 부하 문제를 해결하기 위해 웨이포인트를 활용하기도 한다. 웨이포인트는 캐릭터가 이동할 수 있는 이동경로로써 지형에 설정하며, 캐릭터는 설정된 웨이포인트로만 이동할 수 있다. 이러한 웨이포인트로 캐릭터를 이동시키게 되면 지형 전체를 검색하지 않아도 된다는 장점이 있다. 웨이포인트를 사용할 때는 세밀한 지형을 먼저 생성시킨 후 이동경로를 디자인함으로써 이동경로를 쉽게 생성할 수 있으나, 캐릭터의 이동 자유도가 떨어지게 되고 게임 디자이너가 이동경로를 미리 설정해야 한다는 부가적인 작업이 요구된다.

이러한 문제점 해결을 위해 최근에는 네비게이션 메쉬를 사용하기도 한다. 네비게에션 메쉬는 삼각형 모양의 메쉬를 이용하여 지형을 만드는 방식이다[3]. 메쉬는 세 개의 점을 이용하여 형성되는 삼각형이지만, 크기나 모양에 제약을 주지 않으므로 지형을 표현할 때 그리드 방식에 비해 시스템 부담이 매우 적다. 또한 자연스러운 지형을 표현하면서도 이동경로의 공간 탐색시간을 축소시킨다. 그러나 네비게이션 메쉬에서도 캐릭터가 이동할 때 꺾은선이 발생한다.

네비게이션 메쉬에서 발생하는 캐릭터의 부자연스러운 이동경로 문제점을 해결하기 위해 본 논문에서는 구형 곡선을 생성하여 캐릭터의 이동경로를 자연스럽게 만드는 곡선 생성 기법을 제안한다. 본 논문에서 제안한 곡선 생성 기법을 윈도우 기반 가상현실이나 게임에서 캐릭터의 이동경로에 사용하면 캐릭터의 이동경로가 현실감 있게 처리되는 장점이 있다.

2. 지형 생성 기법

초기의 가상환경 또는 게임환경에서는 공간이나 지형을 일정한 크기의 격자 모양으로 분할하여 공간이나 지형을 표현하였다[3][4]. 그 이유는 일정한 크기의 격자로 공간을 분할하였을 때 가상환경이나 게임환경을 만드는 데 필요한 그래픽 작업이 줄어들고, 관리하는 것도 용이하기 때문이다. 특히 초창기 게임에서는 공간 데이터를 표현하는 많은 양의 픽셀 데이터를 모두 메모리에 저장할 수 없었다. 그러므로 공간을 일정한 크기의 격자로 분할하고 조그만 그림 데이터들을 중복하여 격자에 채움으로써 메모리 효율성을 높였다. 이와 같이 생성하려는 지형을 동일한 크기의 격자로 분할하여 표현하는 방식을 그리드 방식이라 한다. (그림 1)은 그리드 방식을 이용하여 표현한 지형의 예이다.



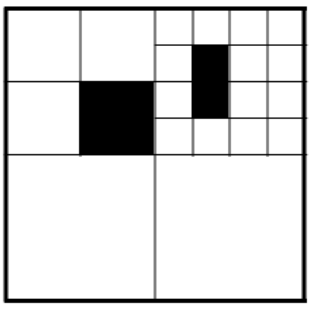
(그림 1) 그리드 방식

그리드 방식은 다음과 같은 장점을 갖는다. 그리드 방식은 공간이나 지형을 일정한 크기로 분할하므로 적용하기 간단하다. 또한 그리드 방식은 공간을 균일한 크기의 격자 모양으로 나누기 때문에 공간이나 지형의 위치에 대한 좌표 계산이 용이하다. 즉, 그리드 방식은 임의의 위치에 접근하기가 쉽다. 그리고 같은 넓이의 공간이나 지형에 대해 동일한 크기의 격자를 가지는 균일성으로 인해 시스템 구현 시에 데이터의 저장이 용이하며, 자동화된 지형 생성이 가능하다.

반면 그리드 방식은 다음과 같은 단점을 갖는다. 먼저 그리드 방식에서 객체를 정밀하게 묘사하기 위해서는 격자의 크기를 매우 작게 만들어야 한다. 격자의 크기 축소는 격자의 수를 증가시키므로 데이터의 크기를 증가시킨다. 그리고 격자의 크기가 축소되어 격자의 수가 증가되면 탐색 영역이 증가되고 탐색 속도가 저하된다. 그리드 방식은 평지 지형을 모델링할 때는 적합하지만 굴곡이 심한 지형의 경우 특징을 정확히 표현하기 힘들다. 그러므로 2차원 공간 표현에 유용하다.

그리드 방식은 사각형 격자를 이용하여 지형을 표현하고 캐릭터는 수직, 수평, 대각 방향으로 이동하기 때문에 이동경로가 꺾은선 형태로 나타나는 등 자연스러운 경로 표현이 힘들다. 그리드 방식에서 이동 경로를 자연스럽게 표현하기 위해서는 격자를 세밀하게 나누어 꺾은선 부분을 자연스럽게 처리하면 되지만, 격자의 증가는 탐색 공간을 증가시킨다는 문제를 발생시킨다.

이러한 문제점을 해결하기 위해 그리드 방식에 상세도를 적용하여 격자 공간을 표현하는 방법이 있다. 상세도는 공간을 자세히 표현해야 하는 부분과 그렇지 않은 부분으로 나누어 공간을 표현하는 방법이다. 그리드 방식에 상세도를 적용하면 상세히 표현할 부분은 작은 격자의 크기로 지형을 세분화하여 표현하고, 그렇지 않은 부분은 큰 격자를 활용하여 표현하는 방법이다. (그림 2)는 상세도를 적용한 그리드 방식이다. 상세도를 적용한 그리드 방식을 지형 생성에 적용하여도 3D를 표현하기 어려우며, 캐릭터의 자연스러운 이동을 생성하기가 어렵다는 문제점은 여전히 해결되지 않는다.



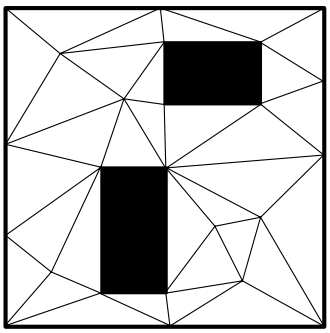
(그림 2) 상세도를 적용한 그리드 방식

게임이나 가상현실에서는 지형이나 객체를 표현할 때 많은 연산을 필요로 하기 때문에 고속의 하드웨어 장치를 필요로 한다. 그러나 일반적으로 게임이나 가상현실 사용자들은 일반 사용자들이 대다수이며, 고속의 장치를 소유한 일반 사용자들은 거의 없기 때문에 게임이나 가상현실 환경을 만들 때는 연산을 많이 필요로 하는 경우를 피하여 환경을 구축한다[5][6]. 이와 같은 문제로 게임이나 가상현실에서는 3차원 작업을 2차원으로 변환하여 작업을 수행하는 경우가 많다. 이때 3차원과 같이 표현하면서 2차원으로 연산을 수행할 수 있는 방법에 활용되는 대표적인 지형 생성 방법이 네비게이션 메쉬 방법이다[5].

네비게이션 메쉬에서는 삼각형 모양으로 이루어진 메쉬에 높이를 서로 다르게 하여 다리와 같은 3차원 지형지물도 쉽게 표현할 수 있으며, 이러한 네비게이션 메쉬를 게임이나 가상현실에서 활용하면 3차원 형태의 지형을 그대로 유지하면서 2차원 구조로 연산을 수행할 수 있는 장점이 있다[7][8][9]. 3차원 공간상에서 이동경로를 생성할 때 네비게이션 메쉬 방법에서는 연산량을 축소시키기 위해 3차원 공간을 2차원에 투영시킨 지형에서 이동경로를 생성한 후, 이동을 수행할 때는 높이가 적용된 3차원 형태의 네비게이션 메쉬에 적용한다[5][6].

네비게이션 메쉬 방식은 삼각형 모양의 메쉬를 이용하여 지형을 표현하는 방법이다. 메쉬는 세 개의 점을 이용하여 형성되는 삼각형이지만, 크기나 모양에 제약을 주지 않으므로 불규칙적인 형태를 갖는다. 삼각형 모양의 메쉬를 생성할 때는 지형에서 중요한 지점을 추출하여 활용한다. 이 방법은 다음과 같다. 먼저 네비게이션 메쉬 방법은 지형에서 산봉우리, 계곡, 평지와 같

은 특정한 의미를 갖는 위치점들을 추출한다. 다음으로 그 점들을 삼각형으로 연결하여 지형을 표현한다. 지형으로부터 의미 있는 위치점들을 추출하여 지형의 모습을 표현하므로 그리드 방법보다 사실적인 지형의 표현이 가능하다.



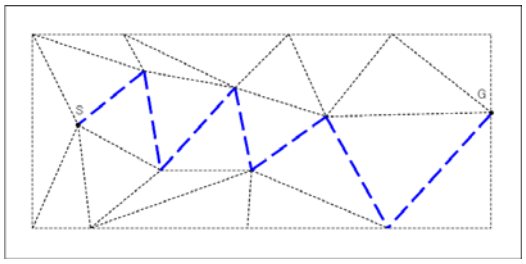
(그림 3) 네비게이션 메쉬 방식

네비게이션 메쉬 방법에서 삼각형을 생성할 때는 다음과 같은 규칙을 따른다. 먼저 인접한 두 삼각형은 두 개의 정점과 한 개의 변을 공유한다. 그리고 두 개의 삼각형이 동일한 면에 겹칠 수 없다. 마지막으로 전체 공간을 표현할 수 있는 최소한의 삼각형을 사용하여 지형을 표현한다. 마지막 규칙인 삼각형의 수는 지형의 상세도에 따라 다르다. (그림 3)은 네비게이션 메쉬 방식으로 지형을 표현한 예이다.

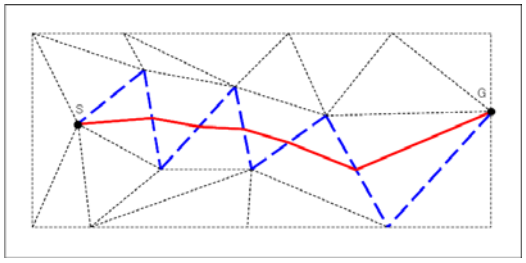
이러한 네비게이션 메쉬 방식은 다음과 같은 장점을 갖는다. 네비게이션 메쉬 방식은 그리드 방식에 비해 지형을 표현하는 데이터의 크기가 작으며, 굴곡이 심한 지형을 표현하는데 적합하다. 그리드 방식은 동일한 크기의 격자에 따라 지형이 분할되지만, 네비게이션 메쉬 방식은 지형적 특성에 따라 삼각형 메쉬의 크기가 조절되어 지형이 분할된다. 즉, 굴곡이 없는 평지는 몇 개의 큰 삼각형으로 표현되고 굴곡이 심한 지형은 작은 삼각형으로 표현된다. 이와 같은 특징은 지형의 상세도가 유사할 경우 그리드보다 네비게이션 메쉬 방식이 지형을 더 작은 데이터의 크기로 표현할 수 있다는 것을 의미한다. 또한 같은 데이터의 크기로 지형을 형성할 경우에는 네비게이션 메쉬 방식이 그리드 방식보다 굴곡이 심한 지형을 보다 상세하게 표현할 수 있다. 네비게이션 메쉬 방식은 탐색 속도를 개선하기 위해 최대한 지형을 단순하게 표현하기 때문에 그리드 방식에 비해 빠른 탐색 속도를 보장할 수 있다. 또한 굴곡이 있는 지형을 표현하는데 적합하므로 실내와 실외의 지형을 표현하는데 모두 적합하다.

네비게이션 메쉬에서 이동경로를 생성할 때는 일반적으로 메쉬를 구성하는 각 선분을 이용한다. (그림 4)에 네비게이션 메쉬에서 이동경로가 생성된 모습을 나타내었다. 그림에 나타난 얇은 점선으로 이루어진 삼각형이 지형을 만들어내는 요소인 메쉬이고, S는 시작지점이고 G는 종료지점이다. 굵은 점선은 시작지점에서 종료지점까지의 이동경로를 나타낸다. 그림에서 보는 바와 같이 캐릭터의 이동경로인 굵은 점선은 꺾은선으로 나타나 있기 때문에 캐릭터가 이동할 경우에 매우 부자연스럽다.

이와 같은 문제점을 해결하기 위해 네비게이션 메쉬에서는 삼각형 메쉬의 각 변의 중점을 이용한다. 각 변의 중점을 이용하여 캐릭터를 이동시키면 꺾은선을 완만하게 만들 수 있다. (그림 5)에서 굵은 점선은 초기 이동경로이고 굵은 실선은 변의 중점을 이용하여 재생성한 이동경로이다. 변의 중점을 이용하여 이동경로를 생성하면 초기 이동경로보다는 자연스럽지만 여전히 꺾은선은 존재한다.



(그림 4) 네비게이션 메쉬에서 이동경로



(그림 5) 네비게이션 메쉬에서 변의 중점을 이용한 이동경로

3. 곡선 생성 기법

곡선 생성 방법은 그래픽스 분야에서 주로 연구되어 왔다[10][11]. 가상현실이나 게임에서는 곡선 표현에 관한 연구가 미미하여 캐릭터의 자연스러운 이동경로를 생성하기 어려운 실정이다. 그래서 본 연구에서는 다양한 곡선 생성 방법에 대해 연구를 수행하여 가상현실이나 게임에 적합하게 사용할 수 있는 곡선 생성 방법을 제안한다.

그래픽스 분야에서 주로 사용하고 있는 곡선 생성 방법은 Chaikin 기법[12], Hermite 기법[13], Bezier 기법[14], B-Spline 기법[15][16], NURBS 기법[17][18] 등이다. 이러한 곡선 처리 기법들은 각자만의 곡선화 특성이 있기 때문에 모두 가상현실이나 게임에 적합하다고 하기는 어렵다. 왜냐하면 이러한 곡선 처리 기법들은 연산시간이 길기 때문에 장거리 이동경로를 생성할 경우에는 시스템에 부하로 작용하기 때문이다. 가상현실이나 게임에서는 정교한 곡선을 생성하는 것보다는 사람이 인지할 수 없을 정도의 곡선화만 이루게 하고 메모리의 부담을 줄여주는 것이 더욱 효과적인 알고리즘이다[4].

본 논문에서 활용한 곡선화 방법들도 네비게이션 메쉬를 기반으로 하기 때문에 환경은 3차원에서 표현되지만 연산을 수행할 때는 3차원 환경을 2차원으로 투영한 후에 경로를 생성하고 곡선화를 수행한다. 그리고 캐릭터가 이동할 때는 2차원에서 생성된 경로를 이용하여 높이가 주어진 3차원 네비게이션 메쉬에 적용되기 때문에 연산량을 줄이면서 경로 생성과 곡선화를 빠르게 이룰 수 있는 장점이 있다. 게임이나 가상현실에서 캐릭터의 이동경로에서 발생한 꺾은선을 곡선화시킬 때도 정교한 곡선 표현보다는 빠르게 곡선을 형성할 수 있는 기법이 더 우수한 기법이다. 이러한 관점에서 본 논문에서는 곡선의 정교한 표현보다는 곡선을 빠르게 생성할 수 있는 기법들만을 활용하여 캐릭터의 이동경로 곡선화에 적용하였다.

George Chaikin은 재귀적 분할(recursive subdivion) 방법을 기반으로 꺾은선이나 다각형의 모서리를 잘라내어 곡선을 만드는 알고리즘을 제안했다[12]. 재귀적 분할 방법은 반복적인 분할 작업을 통하여 초기의 꺾은선이나 다각형을 곡선처럼 표현하는 방법이다[19][20][21].

Chaikin 기법은 다음의 두 단계로 구성된다. 첫 번째 단계는 제어점들 사이의 각 선분에서 선분의 1/4 지점과 3/4 지점에 새로운 제어점을 생성한다. 두 번째 단계에서는 각 제어점에 가까운 새로 생성된 두 제어점을 연결한다. 새로 생성된 제어점들을 연결한 후에는 기존의 제어점과 새로 생성된 제어점 사이의 선을 제거한다. Chaikin 기법은 이 두 과정을 반복 수행하여 초기 선분을 곡선에 가깝게 만든다.

수식 1과 수식 2는 Chaikin 기법에서 새로운 제어점을 결정하는 수식이다. 수식에서 P는 제어점을 나타낸다. 그리고 제어점 P_i^n에서 i는 제어점 번호이고 n은 분할 횟수이다. 제어점 번호는 0부터 시작하며 분할 횟수의 초기값도 0이다. 수식 1은 P_i^n 지점과 P_{i+1}^n 지점 사이의 1/4 지점에 새로운 제어점을 결정하는 식이고, 수식 2는 P_i^n 지점과 P_{i+1}^n 지점 사이의 3/4 지점에 새로운 제어점을 결정하는 식이다.

$$P_{2i-1}^{n+1} = \frac{3}{4}P_i^n + \frac{1}{4}P_{i+1}^n \quad (1)$$

$$P_{2i}^{n+1} = \frac{1}{4}P_i^n + \frac{3}{4}P_{i+1}^n \quad (2)$$

수식 3, 수식 4, 수식 5는 1회 분할을 수행하여 생성된 제어점들이다. (그림 7)은 1회 분할을 수행하여 새로 생성된 제어점 P_0^1, P_1^1, P_2^1, P_3^1, P_4^1, P_5^1로 구성된 선이다. (그림 8)은 Chaikin 기법을 적용하여 2회 분할 후의 꺾은선이다. 초기의 상태에 비해 곡선 형태로 진화되었다.

$$P_0^1 = \frac{3}{4}P_0^0 + \frac{1}{4}P_1^0 \ ,$$

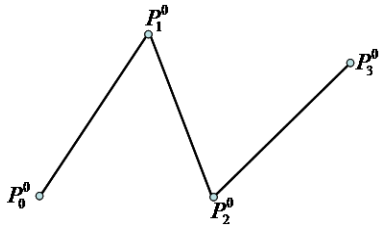
$$P_1^1 = \frac{1}{4}P_0^0 + \frac{3}{4}P_1^0 \quad (n=0,\ i=0) \quad (3)$$

$$P_2^1 = \frac{3}{4}P_1^0 + \frac{1}{4}P_2^0 ,$$

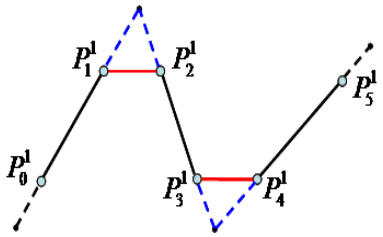
$$P_3^1 = \frac{1}{4}P_1^0 + \frac{3}{4}P_2^0 \quad (n=0,\ i=1) \quad (4)$$

$$P_4^1 = \frac{3}{4}P_2^0 + \frac{1}{4}P_3^0,$$
$$P_5^1 = \frac{1}{4}P_4^0 + \frac{3}{4}P_5^0 \quad (n=0,\ i=2) \qquad (5)$$

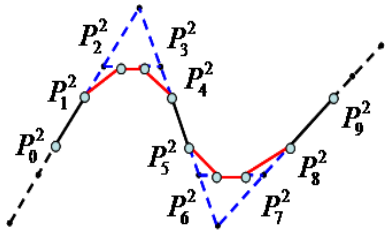
(그림 6)은 4개의 제어점 P_0^0, P_1^0, P_2^0, P_3^0으로 구성된 꺾은선이다. (그림 7)은 꺾은선에 Chaikin 기법을 적용하여 1회 분할한 결과이다.



(그림 6) 4개의 제어점으로 구성된 꺾은선

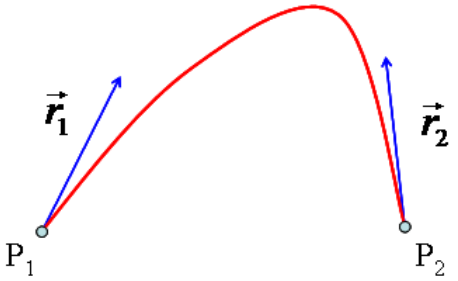


(그림 7) Chaikin 알고리즘을 이용하여 1회 분할 수행



(그림 8) 2회 분할 후의 꺾은선

Hermite 곡선은 스플라인(spline) 곡선의 일종으로 두 개의 제어점과 두 제어점에 대한 접선벡터를 이용하여 그리는 곡선이다[13][22][23]. (그림 9)는 Hermite 곡선의 예이다. (그림 9)에서 P_1과 P_2는 시작 제어점과 끝 제어점이고, $\overrightarrow{r_1}$은 P_1의 접선벡터, $\overrightarrow{r_2}$는 P_2의 접선벡터이다. Hermite 곡선은 제어점 P_1에서 $\overrightarrow{r_1}$ 접선벡터를 따라 그려지기 시작하며, 제어점 P_2에서 $\overrightarrow{r_2}$ 접선벡터를 따라 곡선이 완성된다.



(그림 9) Hermite 곡선의 예

식 6은 Hermite 곡선 함수를 매개변수 표현식으로 나타낸 것이다. 매개변수 표현식은 새로운 매개변수 t를 사용하여 x=f(t), y=g(t)의 형태로 나타난다. 기하형상을 표현할 때는 매개변수 t를 변화시켜 (x, y) 좌표를 계산할 수 있는 매개변수 표현식을 주로 사용한다. 기하형상 표현에서 매개변수 표현식을 사용하는 이유는 하나의 x값에 대응하는 서로 다른 두 개 이상의 y값도 표현할 수 있기 때문이다. 식 6에서 t는 매개변수, P_1과 P_2는 제어점, $\overrightarrow{r_1}$과 $\overrightarrow{r_2}$는 접선벡터이다. 곡선화 함수에서 매개변수의 다항식은 여러 가지 형태가 있을 수 있지만 주로 3차 다항식(cubic polynomial)을 사용한다. 3차보다 낮은 차수의 경우에 부드러운 곡선을 표현하기 어려우며, 3차보다 큰 경우에는 계산시간이 길어진다는 단점이 있기 때문이다.

$$\begin{aligned} P(t) = {} & (2t^3 - 3t^2 + 1)P_1 \\ & + (-2t^3 + 3t^2)P_2 \\ & + (t^3 - 2t^2 + t)\overrightarrow{r_1} \\ & + (t^3 - t^2)\overrightarrow{r_2} \end{aligned} \qquad (6)$$

Bezier 곡선은 n개의 제어점으로 부터 얻어지는 n-1차 곡선으로 프랑스의 Bezier에 의해 제안된 곡선화 기법으로 Bezier 곡선 함수 $B(t)$는 식 7과 같다[14][24][25][26][27].

$$B(t) = \sum_{i=0}^{n} J_{n,i}(t)P_i \quad (0 \le t \le 1) \qquad (7)$$

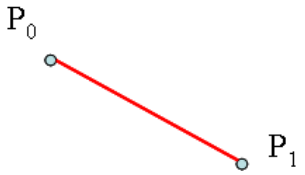
Bezier 곡선 함수에서 P_i는 Bezier 곡선의 제어점을 나타내고, n은 제어점의 수이다. $J_{n,i}(t)$는 식 8과 같다.

$$J_{n,i}(t) = {}_nC_i\, t^i (1-t)^{n-i} \tag{8}$$

Bezier 곡선에서 시작 제어점 P_0은 $t=0$에 대응하고, 마지막 제어점 P_n은 $t=1$에 대응한다. 그러므로 Bezier 곡선은 t 값이 0에서 1까지 변화하면서 Bezier 곡선 함수에 따라 그려지는 곡선을 의미한다. 이 때 Bezier 곡선의 특징은 시작 제어점과 마지막 제어점을 제외한 나머지 제어점을 지나지 않는다.

제어점이 P_0와 P_1으로 두 개인 경우 Bezier 곡선 함수를 적용하면 식 9와 같다. 이러한 경우를 선형 Bezier 곡선이라 한다. 선형 Bezier 곡선의 의미는 t 값을 0에서 1까지 변화시키며 두 제어점 사이의 선분을 t : (1-t)로 내분하는 점을 표현한 것이다. 그러므로 선형 Bezier 곡선 함수의 결과는 (그림 10)과 같이 단순히 두 점 사이에 직선을 그린다.

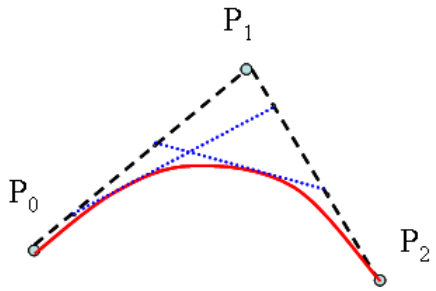
$$B(t) = (1-t)P_0 + tP_1 \tag{9}$$



(그림 10) 제어점이 2개인 경우

제어점이 P_0, P_1, P_2로 세 개인 경우 Bezier 곡선 함수를 적용하면 식 10과 같다. 이러한 경우를 2차 Bezier 곡선이라 한다. 2차 Bezier 곡선은 t와 (1-t)에 따라 P_0과 P_1을 내분하는 점과 P_1과 P_2를 내분하는 점을 연결한 선분을 다시 t와 (1-t)로 내분하는 점의 궤적을 따라 그린 곡선이다. (그림 11)은 제어점이 3개인 경우의 Bezier 곡선을 보인 것이다.

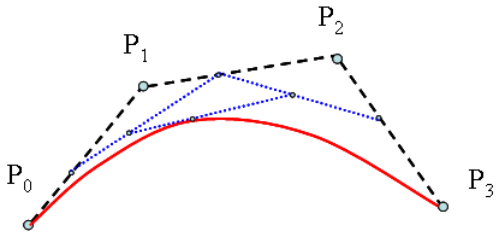
$$B(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2 \tag{10}$$



(그림 11) 제어점이 3개인 경우

제어점이 P_0, P_1, P_2, P_3로 네 개인 경우, Bezier 곡선 함수를 적용하면 식 11과 같다. 이러한 경우를 3차 Bezier 곡선(cubic bezier curves)이라 한다. 3차 Bezier 곡선의 경우도 2차 Bezier 곡선처럼 제어점을 분할하는 내분점이 발생한다. 그러나 3차 Bezier 곡선 함수는 내분점 사이의 선분을 다시 내분하여 선분을 생성한다. 즉, Bezier 곡선 함수는 제어점의 수가 증가할 경우에 제어점 사이의 내분점을 연결하는 선분도 재귀적으로 내분한다. 그리고 이 내분점을 따라 곡선을 출력한다. (그림 12)는 제어점이 4개인 경우의 Bezier 곡선을 보인 것이다.

$$\begin{aligned} B(t) = {} & (1-t)^3 P_0 + 3t(1-t)^2 P_1 \\ & + 3t^2(1-t)P_2 + t^3 P_3 \end{aligned} \tag{11}$$



(그림 12) 제어점이 4개인 경우

B-Spline 기법은 Bezier 곡선의 단점을 보완한 기법이다[15][16]. Bezier 곡선은 하나의 제어점을 변화시키면 전체 곡선이 변화한다는 단점이 있다. B-Spline 기법이 Bezier 기법보다 정교한 곡선을 생성하는 이유는 B-Spline 기법은 제어점과 관련된 각 구간마다 곡선화를 수행할 함수를 활용하기 때문이다. NURBS 기법은 B-Spline을 기반으로 각 제어점에 별도의 가중치를 부

여하여 곡선을 생성하는 방법이다[17][18]. B-Spline 기법에서는 곡선의 모양을 변화시키기 위해 각각의 제어점을 조절하지만, NURBS 기법에서는 각각의 제어점과 가중치를 이용하기 때문에 B-Spline 기법보다 자유롭고 다양한 표현이 가능해진다. B-Spline 기법은 Bezier 기법을 기반으로 제어점이 추가되는 구간마다 곡선화 함수를 계산하여 곡선을 그리는 기법이다. 그래서 B-Spline 기법은 각 곡선화 구간마다 곡선화 함수를 계산하므로 Bezier 기법보다 정교한 곡선을 표현할 수 있지만 연산량이 Bezier 기법보다 많다[16]. NURBS 기법은 B-Spline 기법을 기반으로 제어점에 가중치를 추가하는 기법이기 때문에 B-Spline 기법보다 자유롭고 정교한 곡선을 그릴 수 있지만 B-Spline 기법보다 많은 연산량을 갖는다[17]. 캐릭터의 이동경로에 대한 곡선화에서 정교한 곡선 생성은 큰 의미가 없다. 게임이나 가상현실에서 캐릭터가 이동할 때 시각적으로 자연스럽게 이동하는 것이 주요한 목표기 때문에 다른 기법들에 비해 연산량이 적으면서 곡선화를 빠르게 생성하는 기법이 게임이나 가상현실에 가장 적합한 기법이다. 그래서 본 논문에서 곡선을 정교하고 자유롭게 표현하지만 Bezier 기법에 비해 연산량이 많은 B-Spline 기법과 NURBS 기법은 본 연구에서 제외하였다.

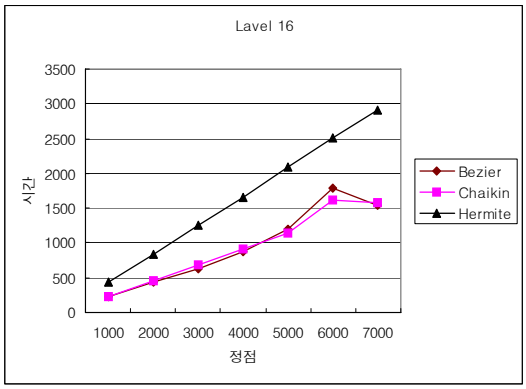
4. 실험결과

게임이나 가상환경에서 캐릭터의 이동경로는 여러 개의 꺾은선들로 이루어진 집합이다. 이러한 이동경로 집합에서 최소 단위는 직선이며, 두 직선이 교차하는 지점은 하나의 변곡점이 형성된다. 이 변곡점으로 인해 이동경로는 꺾은선이 발생하는 것이다.

실험 데이터는 크게 두 가지 방법을 이용하여 생성하였다. 실험 데이터를 생성하는 하나의 방법은 분할되는 직선의 수량에 따라 레벨을 정하여 데이터를 생성하는 방법이다. 레벨은 2의 지수를 활용하여 다양한 레벨을 형성하였다. 실험에 사용된 레벨은 16, 32, 64, 128, 256, 512, 1024이다. 실험 데이터를 생성하는 다른 하나의 방법은 이동경로의 곡선화를 위해 사용되는 정점의 수량 증가에 따라 데이터를 생성하는 방법이다. 정점의 수량이 많을수록 곡선화율이 높게 나타난다. 실험에 사용된 정점은 1000, 2000, 3000, 4000, 5000, 6000, 7000이다. 곡선화에 대한 성능 측정은 같은 조건에서 서로 다른 곡선화 기법을 동일한 실험 데이터에 적용하여 곡선화를 수행한 후, 곡선 생성에 걸리는 시간을 이용하였다. 각 실험에 대한 성능 측정 단위는 마이크로초(μs)이다. 실험에 사용된 곡선 생성 기법은 Chaikin 기법, Hermite 기법, Bezier 기법이다.

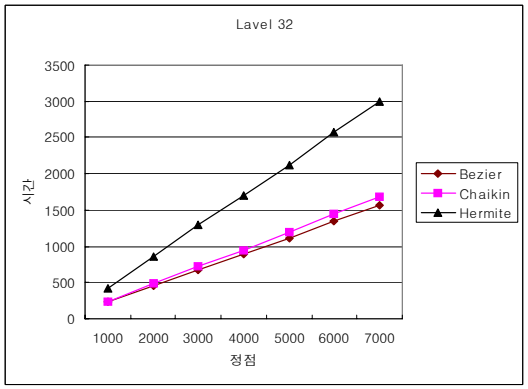
(그림 13)은 레벨 16에서의 실험결과이다. 정점의 수가 1000개일 때 Bezier 기법은 Chaikin 기법보다 3.6%의 성능 향상을 나타내었고, 정점이 2000개일 경우에는 3.1% 성능 향상을 나타내었으며, 정점이 3000개일 경우에는 8.2%의 성능 향상을 나타내었다. 정점이 1000개일 때 Bezier 기법은 Hermite 기법보다 102.2%의 성능 향상을 나타내었고, 정점이 2000개일 경우에는 87.3% 성능 향상을 나타내었으며, 정점이 3000개일 경우에는 99.2%의 성능 향상을 나타내었다. 레벨 16에서 Chaikin 기법에 대한 Bezier 기법의 평균 성능 향상률은 0.8%로 나타났으며, Hermite 기법에 대한 Bezier 기법의 평균 성능 향상률은 82.9%를 나타냈다. 모든 실험구간에서 Bezier 기법이 Hermite 기법 보다는 매우 우수하게 나타났고, 대다수의 실험구간에서 Bezier 기법이 Chaikin 기법보다는 비교적 우수하게 나타났다. 정점수가 1000개일 때 Chaikin 기법은 Hermite 기법보다 95.1%의 성능 향상을 나타내었고, 정점수가 2000개일 경우에는 81.6% 성능 향상을 나타내었으며, 정점수가 3000개일 경우에는 84.0%의 성능 향상을 나타내었다. 레벨 16에서 Hermite 기법에 대한 Chaikin 기법의 평균 성능 향상률은 76.9%를 나타내었다. 모든 실험구간에서 Chaikin 기법이 Hermite 기법보다 매우 우수함을 나타내었다. 이와 같은 결과가 발생한 이유는 Chaikin 기법이 제어점을 많이 발생시키는 단점은 있으나 구현 방법이 간단하여 쉽게 곡선을 생성할 수 있는 장점이 있기 때문이다. Hermite 기법은 제어점의 생성이 복잡하며 많은 제어점이 곡선 생성에 필요하게 되어 곡선화를 수행할 때 오랜 시간이 걸린다. Bezier 기법은 Hermite의 단점을 보완한 기법으로 Chaikin 기법이나 Hermite 기법에 비해 곡선을 생성할 때

필요한 제어점의 수량이 적고 곡선을 생성하기가 수월하다는 장점이 있다.



(그림 13) Level 16에서의 실험결과

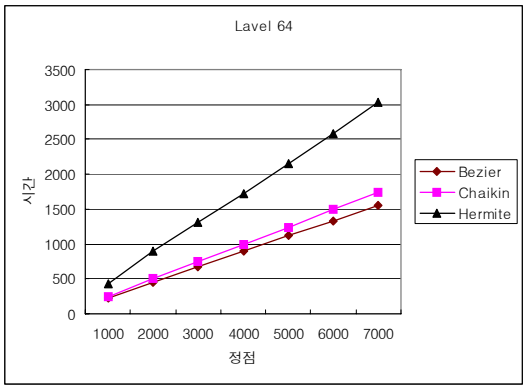
(그림 14)는 레벨 32에서의 실험결과이다. 정점수가 1000개일 때 Bezier 기법은 Chaikin 기법보다 6.1%의 성능 향상을 나타내었고, 정점수가 2000개일 경우에는 6.8% 성능 향상을 나타내었으며, 정점수가 3000개일 경우에는 7.0%의 성능 향상을 나타내었다. 정점수가 1000개일 때 Bezier 기법은 Hermite 기법보다 85.5%의 성능 향상을 나타내었고, 정점수가 2000개일 경우에는 90.0% 성능 향상을 나타내었으며, 정점수가 3000개일 경우에는 89.4%의 성능 향상을 나타내었다. 레벨 32에서 Chaikin 기법에 대한 Bezier 기법의 평균 성능 향상률은 7.0%로 나타났으며, Hermite 기법에 대한 Bezier 기법의 평균 성능 향상률은 89.8%를 나타냈다. 모든 실험구간에서 Bezier 기법이 Hermite 기법이나 Chaikin 기법보다는 우수하게 나타났다. 정점수가 1000개일 때 Chaikin 기법은 Hermite 기법보다 74.7%의 성능 향상을 나타내었고, 정점수가 2000개일 경우에는 77.8% 성능 향상을 나타내었으며, 정점수가 3000개일 경우에는 76.9%의 성능 향상을 나타내었다. 레벨 32에서 Hermite 기법에 대한 Chaikin 기법의 평균 성능 향상률은 77.7%를 나타내었다. 모든 실험구간에서 Chaikin 기법이 Hermite 기법보다 매우 우수한 성능을 나타냈다. 레벨 32에서 정점수가 증가할수록 Bezier 기법과 Chaikin 기법은 Hermite 기법에 비해 안정적인 상태로 완만한 연산시간 증가를 나타내었지만, Hermite 기법은 매우 빠르게 증가하는 결과를 나타냈다. 정점수가 증가할 경우 Hermite은 많은 수의 복잡한 제어점을 활용하기 때문에 다른 기법에 비해 곡선 생성에 많은 시간이 소요된다는 것을 본 실험결과에서 알 수 있었다.



(그림 14) Level 32에서의 실험결과

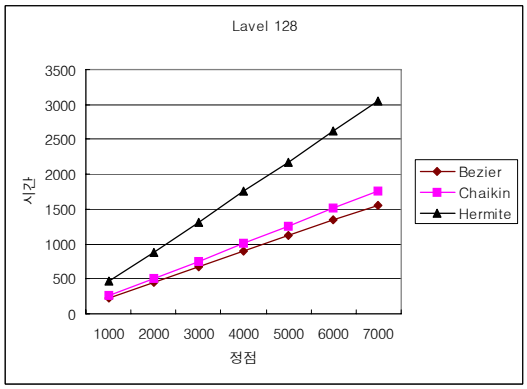
(그림 15)는 레벨 64에서의 실험결과이다. 정점수가 1000개일 때 Bezier 기법은 Chaikin 기법보다 11.0%의 성능 향상을 나타내었고, 정점수가 2000개일 경우에는 11.0% 성능 향상을 나타내었으며, 정점수가 3000개일 경우에는 11.0%의 성능 향상을 나타내었다. 정점수가 1000개일 때 Bezier 기법은 Hermite 기법보다 92.9%의 성능 향상을 나타내었고, 정점수가 2000개일 경우에는 100.0% 성능 향상을 나타내었으며, 정점수가 3000개일 경우에는 94.1%의 성능 향상을 나타내었다. 레벨 64에서 Chaikin 기법에 대한 Bezier 기법의 평균 성능 향상률은 11.3%로 나타났으며, Hermite 기법에 대한 Bezier 기법의 평균 성능 향상률은 94.5%를 나타냈다. 모든 실험구간에서 Bezier 기법이 Hermite 기법이나 Chaikin 기법보다는 우수하게 나타났다. 정점수가 1000개일 때 Chaikin 기법은 Hermite 기법보다 73.7%의 성능 향상을 나타내었고, 정점수가 2000개일 경우에는 80.0% 성능 향상을 나타내었으며, 정점수가 3000개일 경우에는 74.7%의 성능 향상을 나타내었다. 레벨 64에서 Hermite 기법에 대한 Chaikin 기법의 평균 성능 향상률은 74.2%를 나타내었다. 모든 실험구간에서 Chaikin 기법이 Hermite 기법보다 매우 우수한 성능을 나타냈다. 레벨 16과 레벨 32에 비해 레벨 64에서 Bezier 기법과 Hermite 기법이 Chaikin 기법에 비해 더

욱 우수한 성능 향상을 나타내었다. 이와 같은 결과로 Berzier 기법과 Hermite 기법은 레벨과 정점수가 증가할수록 우수한 성능을 나타낸다는 것을 확인할 수 있었다.



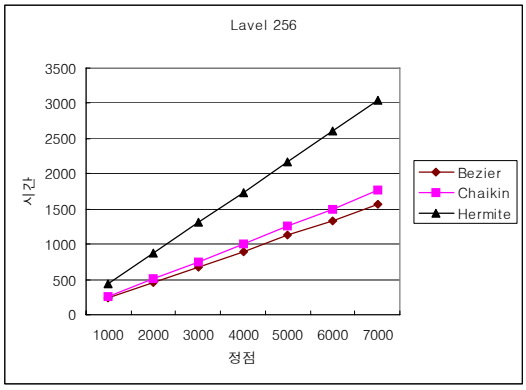
(그림 15) Level 64에서의 실험결과

(그림 16)은 레벨 128에서의 실험결과이다. 정점수가 1000개일 때 Bezier 기법은 Chaikin 기법보다 12.4%의 성능 향상을 나타내었고, 정점수가 2000개일 경우에는 11.1% 성능 향상을 나타내었으며, 정점수가 3000개일 경우에는 12.4%의 성능 향상을 나타내었다. 정점수가 4000개, 5000개, 6000개, 7000개로 증가함에 따른 각각 12.6%, 12.4%, 13.2%, 13.2%의 성능 향상은 나타내었다. 정점수가 1000개일 때 Bezier 기법은 Hermite 기법보다 108.0%의 성능 향상을 나타내었고, 정점수가 2000개일 경우에는 91.0% 성능 향상을 나타내었으며, 정점수가 3000개일 경우에는 94.5%의 성능 향상을 나타내었다. 정점수가 4000개, 5000개, 6000개, 7000개로 증가함에 따라 각각 95.7%, 93.9%, 95.3%, 96.2%의 성능 향상을 나타내었다. 레벨 128에서 Chaikin 기법에 대한 Bezier 기법의 평균 성능 향상률은 12.5%를 나타났으며, Hermite 기법에 대한 Bezier 기법의 평균 성능 향상률은 96.4%를 나타냈다. 모든 실험구간에서 Bezier 기법이 Hermite 기법이나 Chaikin 기법보다는 우수하게 나타났다. 정점수가 1000개일 때 Chaikin 기법은 Hermite 기법보다 73.7%의 성능 향상을 나타내었고, 정점수가 2000개일 경우에는 80.0% 성능 향상을 나타내었으며, 정점수가 3000개일 경우에는 74.7%의 성능 향상을 나타내었다. 레벨 128에서 Hermite 기법에 대한 Chaikin 기법의 평균 성능 향상률은 73.3%를 나타내었다. 모든 실험구간에서 Chaikin 기법이 Hermite 기법보다 매우 우수한 성능을 나타내었다.

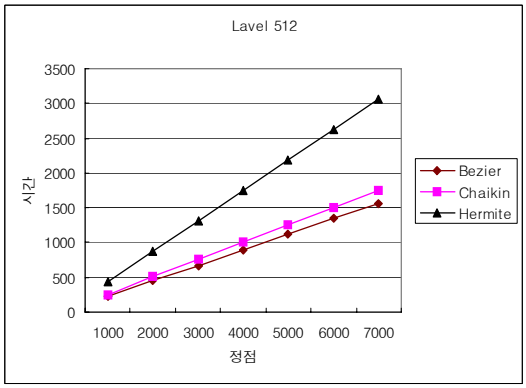


(그림 16) Level 128에서의 실험결과

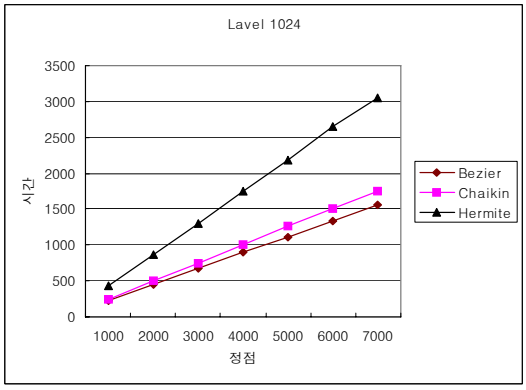
(그림 17), (그림 18), (그림 19)는 레벨 256, 레벨 512, 레벨 1024에서의 실험결과이다. 레벨 256에서 Chaikin 기법에 대한 Bezier 기법의 평균 성능 향상률은 11.9%로 나타났으며, Hermite 기법에 대한 Bezier 기법의 평균 성능 향상률은 93.8%를 나타냈다. 모든 실험구간에서 Bezier 기법이 Hermite 기법이나 Chaikin 기법보다는 우수하게 나타났다. 레벨 256에서 Hermite 기법에 대한 Chaikin 기법의 평균 성능 향상률은 73.0%를 나타냈다. 모든 실험구간에서 Chaikin 기법이 Hermite 기법보다 매우 우수한 성능을 나타냈다. 레벨 512와 레벨 1024에서 Chaikin 기법에 대한 Bezier 기법의 평균 성능 향상률은 각각 12.1%와 12.2%를 나타냈으며, Hermite 기법에 대한 Bezier 기법의 평균 성능 향상률은 각각 94.6%와 94.6%를 나타냈다. 모든 실험구간에서 Bezier 기법이 Hermite 기법이나 Chaikin 기법보다 우수한 성능을 나타냈다. 레벨 512와 레벨 1024에서 Hermite 기법에 대한 Chaikin 기법의 평균 성능 향상률은 각각 79.9%와 73.8%를 나타냈다. 모든 실험구간에서 Chaikin 기법이 Hermite 기법보다 매우 우수한 성능을 나타냈다.



(그림 17) Level 256에서의 실험결과

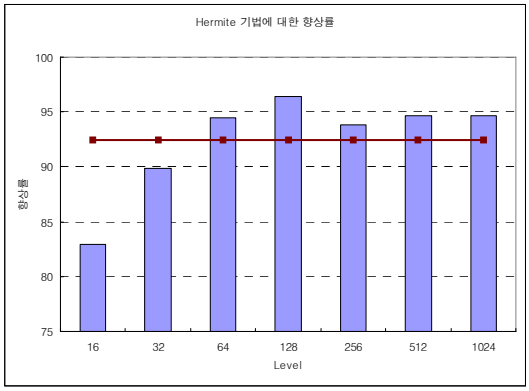


(그림 18) Level 512에서의 실험결과

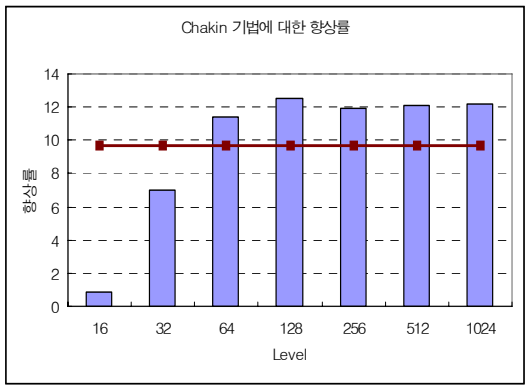


(그림 19) Level 1024에서의 실험결과

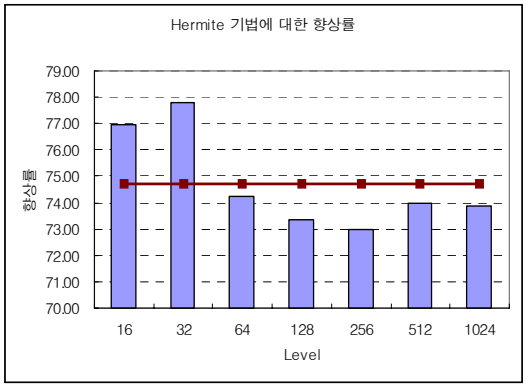
(그림 20)은 Hermite 기법에 대한 Bezier 기법의 평균 성능 향상률을 나타낸 그림으로 결과 그래프를 보면 레벨이 커질수록 평균 성능 향상이 매우 높게 나타난다는 것을 알 수 있으며, 평균 성능 향상률은 92.4%이다. (그림 21)은 Chaikin 기법에 대한 Bezier 기법의 평균 성능 향상률을 나타낸 그림으로 모든 구간에서 Bezier 기법이 Chaikin 기법보다 우수한 성능을 나타낸다. (그림 22)는 Hermite 기법에 대한 Chaikin 기법의 평균 성능 향상률을 나타낸 그림으로 결과 그래프를 보면 레벨이 커질수록 평균 성능 향상이 매우 높게 나타난다는 것을 확인할 수 있으며, 평균 성능 향상률은 74.73%를 나타냈다.



(그림 20) Hermite 기법에 대한 Bezier 기법의 평균 성능 향상률



(그림 21) Chaikin 기법에 대한 Bezier 기법의 평균 성능 향상률



(그림 22) Hermite 기법에 대한 Chaikin 기법의 평균 성능 향상률

5. 결론

본 논문에서는 윈도우 기반 가상현실이나 게임에서 사용되는 네비게이션 메쉬에서 곡선을 생성하는 기법을 제안한다. 가상현실이나 게임에서 주로 사용되는 지형 생성 방식은 그리드 방식과 네비게이션 메쉬 방식이다. 그리드 방식으로 지형을 생성할 때는 격자를 이용함으로 자연스러운 지형을 생성하기 어렵다. 이러한 문제를 해결하기 위해 네비게이션 메쉬가 등장하게 되었다. 그러나 네비게이션 메쉬에서 캐릭터가 이동할 때는 메쉬를 이용하기 때문에 부자연스러운 이동이 발생한다. 이러한 문제점을 해결하기 위해 본 논문에서는 곡선 생성 기법을 제안한다.

본 논문에서 제안한 곡선 생성기법 Hermite 기법, Chaikin 기법, Bezier 기법이다. 이 세 가지 기법들을 다양한 데이터에서 실험을 수행하였다. 레벨 16에서 Bezier 기법은 Hermite 기법보다 평균 82.92%의 성능 향상을 나타냈고, Chaikin 기법보다는 0.83%의 성능 향상을 나타냈다. 레벨 32와 레벨 64에서는 Bezier 기법이 Hermite 기법보다 평균 89.89%와 94.50%의 성능 향상을 나타냈고, Chaikin 기법보다는 평균 7.02%와 11.39%의 성능 향상을 나타냈다. Bezier 기법은 Hermite 기법보다 모든 실험 레벨에서 평균 92.40%의 성능 향상을 나타냈고, Chaikin 기법보다는 9.72%의 성능 향상을 나타냈다. Chaikin 기법은 Hernite 기법보다 모든 실험 레벨에서 평균 74.73%의 성능 향상을 나타냈다. 모든 실험구간에서 레벨이 낮은 경우에는 Bezier 기법이나 Chaikin 기법이 Hermite 기법보다 약간 우수한 성능을 나타내었지만, 레벨이 높아질수록 Bezier 기법과 Chaikin 기법은 매우 우수한 성능을 나타냈다.

실제적인 가상현실이나 게임에서는 본 연구에서 실험한 환경보다 더 높은 레벨과 정점이 발생된다는 것을 고려해 보면 효과적인 곡선화기법은 Bezier 기법이며, 본 논문에서 제안한 곡선 생성 기법은 윈도우 기반 가상현실이나 게임에서 캐릭터의 이동경로를 현실성 있게 이동시킬 수 있는 장점이 있다.

참 고 문 헌

[1] E. Angel, Interractive Computer 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 Proceedings of the IEEE International Conference on Advanced Robotics, 1997.

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

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

[5] M. DeLoura, Game Programming Gems, Charles River Media, 2000.

[6] D. Treglia, Game Programming Gems 3, Charles River Media, 2002.

[7] R. Graham, H. McCabe and S. Sheridan, "Pathfinding in Computer Games," the Journal of the Institute of Technology Blanchardstown, 2004.

[8] J. C. O'Neill, "Efficient Navigation Mesh Implementation," Journal of Game Development, 2004.

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

[10] S. Quinlan and O. Khatib, "Elastic Bands: Connecting Path Planning and Control," In Proceedings of the IEEE International Conference on Robotics and Automation, 1993.

[11] T. W. Sederberg, J. Zheng, D. Sewell and M. Sabin, “Non-uniform Recursive Subdivision Surface," In Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, 1998.

[12] G. Chaikin, “An algorithm for high speed curve generation,” Computer Graphics and Image Processing, 1974.
[13] J. Chi, Y. Zhang and C. Zhang, "Optimized Geometric Hermite Curve Based on Curve Length Minimization," In Proceedings of the 2008 IEEE 8th International Conference on Computer and Information Technology Workshops, 2008.
[14] K. K. Gorowara, "On Bezier Curves and Surfaces," In Proceedings of the IEEE National Aerospace and Electronics Conference, 1988.
[15] J. D. Foley, A. V. Dam, S. K. Feiner and J. F. Hughes, Computer Graphics - Principles and Practice(2nd ed.), Addison Wesley, 1996.
[16] H. Pottmann and G. Farin, "Developable rational Bezier and B-spline surfaces," Computer Aided Geometric Design, 1995.
[17] L. Piegl and W. Tiller, The NURBS Book(2nd ed.), Springer-Verlag, 1997.
[18] D. F. Roger, An Introduction to NURBS with Historical Perspective, Morgan Kaufmann Publishers, 2001.
[19] C. Conti and N. Dyn, “Blending Based Chaikin type Subdivision Schemes for Nets of Curves,” Mathematical Methods for Curves and Surfaces, 2004.
[20] N. Dyn and E. Farkhi, “Spline Subdivision Schemes for Convex Compact Sets,” Journal of Computational and Applied Mathematics, 2000.
[21] M. F. Hassan and N. A. Dodgson, “Reverse Subdivision,” Advances in Multiresolution for Geometric Modelling, 2005.
[22] D. Brunstein, G. Barequet and C. Gotsman, "Animating a Camera for Viewing a Planar Polygon," In Proceedings of the Vision, Modeling, and Visualization Conference 2003, 2003.
[23] D. Crocker, M. Goebel and D. Costa, "Interpolation of animal tracking data in a fluid environment," Journal of Experimental Biology, 2006.
[24] M. Ech and H. Hoppe, "Automatic Preconstruction of B-Spline Surfaces of Arbitrary Topological Type," Computer Graphics, 1996.
[25] D. Morera, P. Carvalho and L. Velho, "Geodesic B´ezier Curves: a Tool for Modeling on Triangulations, " Brazilian Symposium on Computer Graphics and Image Processing, 2007.
[26] A. Rišus, "approximation of a cubic bezier curve by circular arcs and vice versa," Information Technology and Control, 2006.
[27] E. Ülker and A. Arslan, "Curve fitting in terms N-Control points on 2D using fuzzy set and rough set theory," International Journal of Computer Science and Network Security, 2006.

김 형 일

2004년 :동국대학교 컴퓨터공학과 (공학박사)
1996년～1998년:(주)경기은행
2005년～2006년:동국대학교 컴퓨터공학과 IT교수 (정보통신부)
2007년～현 재:나사렛대학교 멀티미디어학과 교수
관심분야 : 지능형시스템, 추천시스템, 기계학습, 게임