다중 에이전트에 기반한 집단행동의 기술 동향

이재문* · 엄종석**

1. 서 론

집단행동은 다수의 에이전트들이 자율적으로 움직일 때 나타나는 행동을 말한다. 이러한 집단행동은 영화나 게임 등에서 장면의 사실성을 증대시키거나 현실적으로 제작하기 어려운 장면을 시뮬레이션 하는데 사용된다. 집단행동의 가장 단순한 형태가 파티클 시스템이다[1,12,14,15]. 파트클 시스템은 영화나 게임에서 불, 비, 연기, 폭발 등의 효과를 나타내는데 사용된다. 파티클 시스템보다 진화한 집단행동이 무리짓기이다[2,3,7,9,11,13]. 이는 무리짓기에 속한 에이전트들이 상호 작용을 하면서 집단행동을 하기 때문에 보다 진화한 것이다. 무리짓기는 새 및 메뚜기 떼들, 수족관 속의 물고기들, 전투에서 대규모 군인들의 시뮬레이션에 사용된다. 군중 애니메이션은 보다 진화한 집단행동으로 군중속의 에이전트는 사람과 유사한 지능을 갖고 집단행동을 하여야 한다[5,6,8,10].

첨단 디지털 영상 제작의 기술은 1980년대 초 영화 'Star Trek II: The Wrath of Kahn'에서 도입된 이래로 헐리웃 영화를 중심으로 컴퓨터를 이용한 특수효과가 많이 사용되면서 큰 발전을 이루게 되었으며, 1990년대부터 컴퓨터 애니메이션 분야에서 큰 성장을 이루었다. 컴퓨터 그래픽스를 이용하여 불이나 연기, 폭발, 액체의 분무, 눈발, 비행기의 연기 흔적과 같은 특수효과를 실현하는 것이 파티클 시스템이다. 파티클 시스템은 여러 개의 개별적인 파티클들의 집합이다[1,15]. 각각의 파티클은 개별적인 특성을 갖고 있어 서로 간에 독립적으로 행동하지만, 또한 각각은 공통적인 특성을 공유하고 있기에 독립적으로 움직이더라도 전체적으로 볼 때 하나의 공통된 효과를 나타낸다.

무리짓기란 파티클 시스템과 아주 유사하다. 1980년대 후반에 최첨단 컴퓨터 그래픽스와 애니메이션을 사용한 BBC의 Horizon 다큐멘터리 방송에서 새들이 무리지어 나는 행동을 기가 막히게 표현한 적이 있다. 여기서 무리지어 다니는 새들을 에이전트(boid)라고 불렀고, 집단행동(무리짓기)의 단순한 규칙을 '조종 행동(steering behavior)'이라고 불렀다. 이 행동을 프로그램한 사람은 Craig Reynolds[3,11]였는데, 그 이후 Reynolds는 조종 행동에 대한 다양한 연구를 발표 하였다. 그러나 이러한 집단행동이 쉽게 게임 등에 사용되지 못하였는데 그 이유는 계산 량이 너무 많아 시스

※ 교신저자(Corresponding Author) : 이재문, 주소 : 서울시 성북구 삼선동(136-792), 전화 : 02)760-4135, FAX : 02)760-4488, E-mail : jmlee@hansung.ac.kr
* 한성대학교 멀티미디어공학과 교수
** 한성대학교 멀티미디어공학과 교수
(E-mail : jsum@hansung.ac.kr)
※ 본 연구는 2008 한성대학교 교내 연구비 지원 과제임

템 성능이 저하되기 때문이다. 이후 많은 게임에서 집단행동은 부분적으로 도입되었고, 게임 외 인공 수족관, 영화 등에서 컴퓨터 그래픽의 한 영역으로 사용되어 왔다.

파티클 시스템과 무리짓기를 구별하는 요소는 에이전트간 상호작용의 정도라 할 수 있다. 이러한 관점에서 볼 때 군중 애니메이션은 군중 애니메이션의 집단에 속한 에이전트가 보다 지능화되는 경우를 의미한다. [5]에서는 도시속 보행자들을 집단으로 간주하였고 이들의 자율적인 행동을 시뮬레이션 하였다. 이 경우 보행자 에이전트는 무리짓기에서 볼 수 있는 새 또는 물고기보다 훨씬 높은 수준의 지능을 갖는 상태에서 자율 행동을 한다. [6]에서는 심리학적 이론을 바탕으로 보다 자연스러운 자율 행동을 하는 보행자 에이전트에 대하여 연구하였으며, [8]에서는 새로운 빌딩 등 대중이 모이는 공공장소에서 돌발 상황이 벌어지는 상황을 시뮬레이션하는 연구를 하였다. 이러한 군중 애니메이션은 현실성을 증대시키기 위하여 일반 사람의 지능에 준하는 에이전트의 지능을 요구한다.

본 논문은 집단행동을 크게 파티클 시스템, 무리짓기 및 군중 애니메이션으로 분류하고 각각의 기본적인 기술이 무엇인 파악하고, 이를 기반으로 최근 연구들을 소개한다. 2장에서 집단행동의 기본적인 기술을 소개하며, 3장에서 최근 연구 동향을 소개하고, 4장에서 결론을 논한다.

2. 집단행동의 기본 기술

2.1 집단행동의 특성

집단행동으로 앞에서 소개한 파티클 시스템, 무리짓기 및 군중 애니메이션은 각각을 구별하는 특성은 있으나 이들 간 명확히 구분 짓기도 쉽지 않다. 시대적, 기술적 관점에서 보면 초기 단계에 파티클 시스템이 도입되었고, 이를 보다 발전시킨 것이 무리짓기라 할 수 있다. 또한 무리짓기에 보다 지능적 에이전트를 적용시킨 것이 군중 애니메이션이다. 표 1은 이러한 세 가지 집단행동에 대한 특성을 요약한 것이다. 파티클 시스템은 그 특성상 수많은 에이전트를 포함한다. 수많은 에이전트를 포함하더라도 이들은 서로 간 상호 작용도 없을 뿐만 아니라 지능도 매우 낮으므로 이를 실행하는 비용은 크지 않다. 무리짓기의 경우 대체로 1,000개 이하의 에이전트로 구성한다. 대규모 무리짓기에 대한 연구도 많이 일어나고 있다. [11]에서는 병렬처리를 사용하여 15,000 에이전트들에 대하여 초당 60프레임을 실행하는 연구를 하였으며, [13]에서는 대표 보이드를 정의하여 효율적인 무리짓기에 대한 연구가 있었으며, [16]에서는 주변의 영향을 주는 에이전트들이 자주 변경되지 않는다는 사실을 이용하여 동일한 에이전트 수에 대하여 40%이상의 초당 프레임수를 개선시켰다. 군중 애니메이션은 100개 이하의 소수 에이전트를 적용하고 있다. 이는 에이전트 자체의 지능도가 높음으로 자율 행동을 위한 계산이 많기 때문이다[15].

표 1. 집단행동별 특성

집단행동	에이전트의 수	상호 작용	지능화
파티클 시스템	대규모 (10,000개 이하)	전혀 없음	없음
무리짓기	중규모 (1,000개 이하)	제한적 주변 에이전트	제한적
군중 애니메이션	소규모 (100개 이하)	다양한 주변 에이전트	다양함

2.2 파티클 시스템

파티클 시스템은 불, 연기, 구름 등 불규칙적인

형태를 가지는 움직이는 에이전트를 관리하는 것이다. 파티클 시스템은 자동적으로 움직이는 수많은 파티클(입자)을 포함한다. 하나의 파티클은 생성되고, 집단행동을 하고 그리고 최종적으로 사라진다. 파티클은 살아 있는 동안 파티클 색상, 불투명성, 위치, 및 속도 등 그 자체의 상태를 변화시키는 행동을 한다[1].

파티클의 생성은 대체로 엄격한 확률론적 절차를 따른다. 즉, 매 프레임 파티클의 개수는 프로그래머가 설정한 분포에 따라 결정된다. 하나의 파티클은 대체로 위치, 속도, 모양, 색 및 생명주기를 갖는데 생성 시에 이러한 값들이 주어진다. 파티클의 종료는 매 프래임마다 감소하는 생명주기가 영이 되면 파티클 시스템에서 제거된다. 파티클의 애니메이션은 파티클이 살아 있는 동안 매 프레임마다 새로운 위치 및 속도를 계산하게 된다. 파티클 시스템의 특징은 비록 파티클이 주변의 영향을 받아서 위치 및 속도가 변경되나 다른 파티클과는 일제의 상호작용이 없다는 것이다[1,15]. 이러한 개념 때문에 하나의 파티클 시스템 내에 수천, 수만개의 파티클이 존재할 수 있다.

비록 파티클 시스템에서 에이전트의 집단행동의 계산이 다른 에이전트의 영향없이 계산될지라도 파티클 시스템 내에 에이전트의 수가 많기 때문에 가속화 기법이 필요하다. 가장 일반적인 가속화 기법이 메모리의 효율적인 사용이다. 앞에서 언급하였듯이 에이전트들은 임의의 시간을 보낸 후 소멸하게 되는데 이것은 또 다른 에이전트의 생성을 의미한다. 에이전트의 수가 증가함에 따라 매 프레임 생성/소멸하는 에이전트의 수가 증가하게 되고 이는 수많은 메모리 의 동적 할당을 요구하게 된다. 가속을 위한 가장 단순한 방법이 메모리 관리를 별도로 하여 동적 할당을 최소화하는 것이다. 또 다른 가속화 기법은 LOD(Level Of Detail) 준거를 사용하는 것이다. 즉 멀리서 보여지는 에이전트의 계산은 간략화 하거나 렌더링을 최소화 하므로써 가속화 할 수 있다. 기타 다양한 가속화 방법이 응용 환경에 따라 적용될 수 있다. [12]에서는 컴퓨터 그래픽스 기술 가운데 표현하기 가장 어렵다는 맥주 거품이나 물 같은 액체가 실제처럼 흐르는 생동감 있는 '유체 시뮬레이션' 기술을 개발하였다. 이 기술은 가스를 포함하고 있는 액체의 거품생성이나 물의 흐름을 실제와 거의 동일하게 재현할 수 있기 때문에 앞으로 상용화될 기술은 게임, 영화나 광고 분야에서 많이 쓰일 것으로 보인다. 또한 파도나 폭풍, 연기, 먼지 등 실제로 촬영하기 어려운 다양한 자연현상을 표현하는 데도 활용될 수 있다[12]. 또한 [14]에서는 물속에서 거품이 발생하는 시뮬레이션에 대한 연구를 하였다.

2.3 무리짓기

무리짓기란 많은 자연에서 관찰되는 현상으로 떼를 지어 다니는 새들, 어군속의 물고기들, 목장에서 양들, 음식물을 찾아다니는 개미들 등의 움직임이 모두 무리짓기에 속한다. 이러한 동물들의 집단은 무리속의 각각이 지속적으로 그들의 방향과 형태를 바꿀지라도 전체적으로 하나의 응집된 형태를 가지는 특성을 보인다. 무리짓기의 가장 주목할 만한 특징은 각각의 에이전트는 무리에 대한 어떠한 정보도 가지지 않는다는 점이다. 각각의 에이전트는 매 순간마다 자신의 주변을 다시 평가할 뿐, 무리에 대한 정보는 전혀 가지고 있지 않다. 무리짓기가 파티클과 다른 점은 하나의 움직이는 에이전트가 다음 움직임을 고려할 때 파티클은 자신과 주변의 영향만 고려하면 되지만 무리짓기는 다른 에이전트도 고려하여야 한다는 것이다. [3]에서는 무리짓기에서 자동 에이전트의 움

직임을 다음과 같이 액션 선택층, 조종층 및 이동층으로 분리 하였다.

- 액션 선택: 목표를 선택하고, 선택된 목표를 성취하기 위하여 어떻게 하여야 하는 것인지를 계획하는 것이다.

- 조종층: 선택된 목표를 성취하기 위하여 정해진 계획에 따른 궤적을 계산하는 것이다. 이층은 이러한 궤적에 따라 움직일 수 있도록 어디로 가야하고 얼마나 빠르게 움직여야 하는 지를 나타내는 조종력을 계산한다. 이 조종력의 계산은 장애물과 같은 주변 환경적인 요인뿐만 아니라 다른 에이전트들의 영향도 고려하여야 한다.

- 이동층: 이동층은 조종층에 의하여 계산된 조종력을 사용하여 해당 에이전트의 위치, 방향 및 속도 등을 새롭게 계산하는 층이다.

세 가지 층에서 자동적 움직임을 위하여 가장 많은 계산을 필요로 하는 층이 조종층이다. 조종층에서의 조종력 계산은 많은 경우 응용 환경에 따라 약간의 차이가 있다. 본 논문에서는 이러한 조종력 계산에 대한 두 가지 방법을 소개한다.

2.3.1 Reynolds의 조종력 계산

[3,11]에서는 개별적인 조종 행동과 집단적인 조종 행동 두 가지를 소개하였다. 개별적인 조종 행동의 가장 큰 특징은 조종력을 계산함에 있어 다른 에이전트의 영향을 고려하지 않는다는 것이다. 개별적인 조종 행동으로 찾기, 달아나기, 도착하기, 추격하기, 도피하기, 배회하기, 장애물 피하기, 벽 피하기, 끼워 넣기, 숨기기, 경로 따라가기 및 오프셋 추격하기 등이 있다[3]. 하나의 에이전트는 액션 선택에서 선택된 계획을 실행하면서 하나의 개별적인 조종행동만 계산하는 것이 아니라 여러 개의 개별적인 조종 행동을 계산하여야 하는데 이때 조심해야 하는 것은 우선순위가 명확하여야 한다. 예를 들어 "달아나기"를 수행중인 에이전트가 벽을 만나는 경우 "달아나기"보다 "벽 피하기"가 우선적으로 고려되어야 한다.

집단적인 조종 행동은 게임 세계내의 일부 또는 전 에이전트의 존재를 고려하여 조종의 힘을 계산하는 것이다. [3]에서는 이러한 집단적인 조종 행동은 결합(cohesion), 분리(separation) 및 정렬(alignment)의 조합으로 구성될 수 있다고 하였다. 분리는 다른 동료와 일정 거리를 유지하는 능력을 말하는데 이것은 다른 동료와 너무 가까이 가지 않음으로써 충돌을 피할 수 있도록 하는 행동이다. 응집은 주위 동료들과 그룹을 형성하고자 하는 행동이다. 응집은 주위의 동료들에 대한 위치 평균을 구함으로써 계산한다. 정렬은 주위의 동료들과 같은 방향을 유지하려는 행동이다. 이것은 주위 동료들의 평균 방향을 구함으로써 계산한다. 임의의 에이전트 b_i의 새로운 방향의 결정에 영향을 주는 에이전트의 집합을 V_i라 하자. 이때 [3]에 의하여 제시된 에이전트 b_i의 조종력 $d_{b_i}(t+\tau)$는 다음과 같이 계산된다.

$$d^s_{b_i}(t+\tau) = -\sum_{\forall b_j \subseteq V_i} \frac{b_j.pos - b_i.pos}{|b_j.pos - b_i.pos|^q} \quad // \text{분리 힘}$$

$$d^c_{b_i}(t+\tau) = \sum_{\forall b_j \subseteq V_i} \frac{b_j.pos}{\|V_i\|} - b_i.pos \quad // \text{결합 힘}$$

$$d^a_{b_i}(t+\tau) = \sum_{\forall b_j \subseteq V_i} \frac{b_j.dir}{\|V_i\|} \quad // \text{정렬 힘}$$

$$d_{b_i}(t+\tau) = C_s \times d^s_{b_i}(t+\tau) + C_c \times d^c_{b_i}(t+\tau) + C_a \times d^a_{b_i}(t+\tau) \quad // \text{최종 힘}$$

여기서 $b_i.pos$는 b_i의 현재 위치 벡터이고, $b_i.dir$는 b_i의 현재 방향 단위 벡터이다. $|v|$는 벡터 v의 크기를 나타내며, $\|S\|$는 집합 S의 요소 수를 표시한다. $d^s_{b_i}(t+\tau)$는 분리의 힘을 계산하는 것이다. $d^s_{b_i}(t+\tau)$에서 앞의 음수 부호는 이 힘은 반발력으

로 나타나야 하기 때문이다. 이 식에서 q는 적용 분야에 따라 0에서 3까지 다양하게 적용된다. 이것의 의미는 이 힘은 두 에이전트(b_j, b_i)간 거리에 반비례하여 그 힘이 계산된다는 의미로 가까이 있는 에이전트에 대하여 더 많은 효과를 원하는 경우 더 큰 q값을 적용하면 된다. $d_{b_i}^{c}(t+\tau)$는 에이전트 b_i가 주변 에이전트의 위치 중심으로 움직이도록 하는 힘이다. $d_{b_i}^{a}(t+\tau)$는 주변 에이전트들의 평균 방향과 같은 방향으로 움직이도록 하는 힘이다. 따라서 가능한 주변 에이전트들과 같은 방향으로 움직일 것이다. 최종적으로 $d_{b_i}(t+\tau)$는 가중치 C_s, C_c 및 C_a를 정하여 이들의 힘의 합으로 계산된다. 가중치 C_s, C_c 및 C_a는 응용 환경에 따라 달라질 것이며 다양한 실험을 통하여 원하는 가중치를 찾아야 한다.

2.3.2 Couzin의 조종력 계산

[4]에서 Couzin은 생물학적인 모델을 기반으로 에이전트들에 대한 집단행동을 연구하였다. [4]에서의 연구의 목적은 무리짓기에 적용되는 힘의 범위를 제어함으로써 무리짓기의 행동을 제어할 수 있음을 보였다. 이 논문에서는 무리속의 에이전트에게 적용되는 2가지 규칙이 제시하였는데, 첫째는 "각 에이전트는 항상 그들 사이의 최소한의 거리를 유지하려 한다"라는 것이다. 이 규칙은 가장 우선순위가 높은 규칙이고, 자연 환경에서 나타나는 행동과 일치한다고 주장하였다. 둘째는 "첫번째 규칙을 위배하지 않는 범위에서 에이전트들을 서로 끌어당기려 하고, 또한 서로 간 정렬하려 한다"라는 것이다.

[4]에서는 자동으로 움직이는 에이전트 각각은 3차원 공간에서 위치와 방향 벡터를 갖는다고 하였고, 조종의 힘은 매 프레임마다 계산되어 위치와 방향 벡터를 갱신하는 것으로 하였다. [4]에서는 [3]과 달리 해당 에이전트에 영향을 주는 에이전트의 집합에 대하여 이들을 세 가지 영역으로 구분하여 조종의 힘을 계산하였다.

그림 1은 [4]에서 제시한 세 가지 영역에 대한 표시이다. 해당 에이전트와 가장 가까이 있는 영역을 반발작용 영역(*zor*)이라 하고, 그 외의 영역을 적응작용 영역(*zoo*) 및 유인작용 영역(*zoa*)로 나누었다. 이때 앞에서와 같이 임의의 에이전트 b_i의 새로운 방향의 결정에 영향을 주는 에이전트의 집합을 V_i라 하면, 반발 작용 영역에서 에이전트 b_i의 조종력 $d_{b_i}^{r}(t+\gamma)$는 다음과 같이 계산된다.

$$d_{b_i}^{r}(t+\gamma) = -\sum_{j \neq i}^{n_r} \frac{b_j.pos - b_i.pos}{|b_j.pos - b_i.pos|^2}$$

상기 식에서 음수는 반발력임을 나타내고, 이 힘은 두 에이전트간 공간적 거리($b_j.pos - b_i.pos$)에 대하여 거리에 반비례하는 힘을 생성한다. 따라서 매우 가까이 있는 경우 강력한 반발 힘이 만들어 질 것이다. 이 힘은 제1규칙을 만족하는 힘이 된다. 만약 *zor*에 어떠한 에이전트가 존재하지 않는다면 *zoa*와 *zoo*에 있는 에이전트들에 의하여 이 힘을 계산한다. 이 영역에 대하여 보이지 않는 영역(*blind volume*)을 설정하여 하나의 에

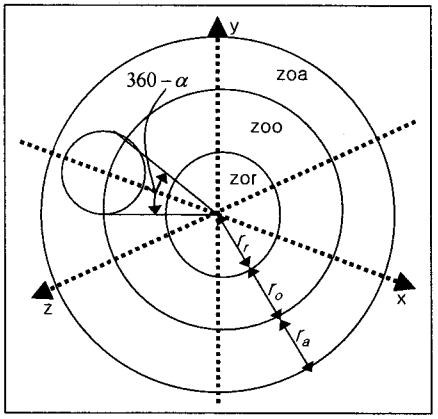


그림 1. 에이전트에 영향을 주는 세 가지 영역

이전트가 이 영역에서 볼 수 있는 각도를 설정하여 그 각도를 벗어난 에이전트는 이 힘의 계산에서 배제 하였다. 보이지 않는 영역에 있는 에이전트들을 제외한 상태에서 n_a, n_o를 각각 *zoa*, *zoo*에 있는 에이전트들의 수라고 할 때 각 영역별 힘은 다음과 같이 계산하였다.

$$d_{b_i}^o(t+\gamma)=\sum_{j=1}^{n_o}\frac{b_j.dir}{|b_j.dir|},$$

$$d_{b_i}^a(t+\gamma)=\sum_{j\neq i}^{n_a}\frac{b_j.pos-b_i.pos}{|b_j.pos-b_i.pos|^2}$$

$d_{b_i}^o(t+\gamma)$는 *zoo*에 의하여 생성되는 힘이고, $d_{b_i}^a(t+\gamma)$는 *zoa*에 의하여 생성되는 힘이다. $d_{b_i}^o(t+\gamma)$에서 $\frac{b_j.dir}{|b_j.dir|}$는 각 에이전트의 방향에 대한 단위 벡터를 의미하고, 이들을 전부 합한다는 것은 주변에 있는 에이전트들의 평균 방향으로 이동하려는 것과 같다. 이것은 제2규칙을 만족하는 힘이 된다. $d_{b_i}^a(t+\gamma)$는 정확히 $d_{b_i}^o(t+\gamma)$과 반대되는 힘이다. 즉 멀리 떨어져 있는 에이전트들에 대해서는 이들과 인접하려는 힘이 필요한데 이것이 그 힘이다. [4]에서는 에이전트 b_i의 새로운 방향을 결정하는 세 가지 힘에 대하여 최종적으로 다음과 같이 계산 하였다.

$$d_{b_i}(t+\gamma)=\begin{cases} d_{b_i}^r(t+\gamma) & // \ n_r\neq 0 \\ \frac{1}{2}[d_{b_i}^o(t+\gamma)+d_{b_i}^a(t+\gamma)] & // \ n_r=0,\ n_o\neq 0 \\ & // \ n_a\neq 0 \\ d_{b_i}^o(t+\gamma) & // \ n_r=0,\ n_a=0 \\ d_{b_i}^a(t+\gamma) & // \ n_r=0,\ n_o=0 \end{cases}$$

이것은 [3]이 제안한 분리, 정렬, 결합의 힘과 비교하여 보면 $d_{b_i}^r(t+\gamma)$는 분리의 힘과 일치하며, $d_{b_i}^o(t+\gamma)$는 정렬의 힘과 일치하고, $d_{b_i}^a(t+\gamma)$는 결합의 힘과 일치한다. [4]에서는 [3]과 달리 분리, 정렬, 결합의 힘을 그들의 영역에 따라 다른 가중치를 적용하였다는 것 외에 전체적으로는 매우 유사함을 알 수 있다.

2.3.3 이웃 에이전트 찾는 알고리즘

집단적인 조종 행동은 게임 세계내의 일부 또는 전 에이전트의 존재를 고려하여 조종의 힘을 계산하는 것이다. 필요에 따라서는 전체 에이전트를 고려하여 조종의 힘을 계산할 수 있으나, 실행 속도 및 무리짓기의 행동을 제어하기 위하여 대부분의 경우 에이전트의 일부분만 영향을 끼치는 대상으로 고려한다. 전체 중 일부분의 에이전트만 고려할 때 크게 두가지 방법이 사용된다. 첫 번째 방법은 해당 에이전트를 중심으로 일정한 반경 내에 존재하는 모든 에이전트를 영향을 끼치는 에이전트로 고려하는 것이다. 또 다른 방법은 해당 에이전트를 중심으로 가장 가까이 있는 k개의 에이전트를 영향을 끼치는 에이전트로 고려하는 것이다. 이 경우에는 전자와 달리 해당 에이전트 주변에 있는 에이전트들의 밀도에 따라 반경이 가변적으로 된다. 두 가지 방법은 각각 장단점이 있다. 예를 들어 전자의 경우 항상 일정한 반경 내에만 고려하므로 영향을 끼치는 에이전트의 수가 가변적이다. 극단적으로 영향을 끼치는 에이전트가 에이전트들의 전부가 될 수도 있고, 반대로 한 개도 없을 수 있다. 이것은 조종력을 계산하는 속도나 무리짓기 행동 제어에 한계가 있을 수 있다. 후자의 경우 항상 k개의 에이전트가 대상이 되므로 조종력을 계산하는 관점에서는 일정한 계산 속도나 k 값을 변경하므로써 무리짓기의 행동을 제어하는 장점은 있으나 k개의 이웃을 찾는 비용이 적지 않다. 따라서 무리짓기의 활용 영역에 따라서 이러한 방법중의 하나를 선택하여야 한다. 본 논문에서는 후자를 대상으로 k개의 이웃을 찾는 방법을 설명한다.

상기 알고리즘 *Steering*에서 입력은 에이전트

```
Algorithm Steering: Inputs: Flocking, k, Outputs: None
01: foreach agent in Flocking{
02:   kNN={φ};
03:   foreach neighbor in Flocking{
04:     if( ‖kNN‖ <k ||
            |agent.pos-neighbor.pos|< kNN[0])
05:       kNN.AddAgent(agent, neighbor, k);
06:   }
07:   agent.steerForce= ComputeSteerForce
      (agent, kNN);
08: }
```

그림 2. 조종력 계산 알고리즘

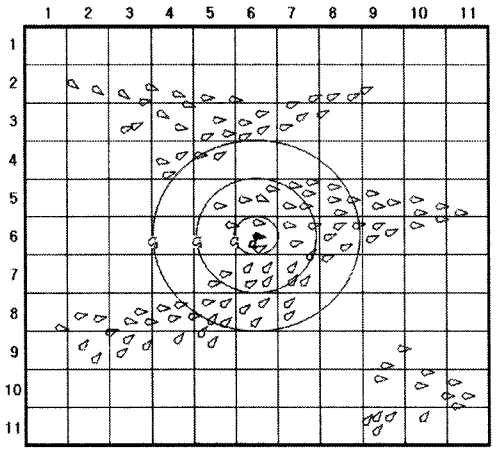


그림 3. 공간분할의 예

의 집합인 *Flocking*과 이웃의 개수인 k이다. 여기서 집합 *kNN*은 최대 k개의 에이전트를 저장하는 자료구조로 저장순서는 *agent*와 거리가 먼 순서로 저장한다. 따라서 *kNN*[0]에는 항상 가장 먼 거리에 있는 에이전트가 저장되어 있다. 함수 *ComputeSteerForce*는 주어진 에이전트에 대하여 조종력을 계산하는 함수이다. 상기 알고리즘은 *kNN.AddAgent*가 $O(1)$에 실행될 수 있다고 하더라도 $O(n^2)$의 시간 복잡도를 갖는다. 여기서 n은 무리짓기에서 에이전트의 전체 수이다.

공간분할 알고리즘은 무리짓기에서 k개의 가장 가까운 이웃, *kNN*(*k-nearest neighbors*)을 효율적으로 찾기 위하여 [3]에서 제안되었다. 인덱스, 트리 등 보조 데이터가 없는 상태에서 n개의 데이터에 대하여 모든 *kNN*을 찾는 것은 $O(n^2)$의 비용이 필요하다. 공간분할 방법은 하나의 에이전트에 대해서만 *kNN*을 찾는 것이 아니라 모든 에이전트에 대하여 *kNN*을 찾아야 한다는 점에 착안하여 *kNN*을 찾기 전에 모든 에이전트들에 대하여 에이전트들의 위치를 근사적으로 군집화 시키는 것이다.

예를 들어 그림 3과 같이 2D에서 게임의 세계를 그리드로 나눈 후, 전처리 작업으로 각 에이전트를 에이전트의 위치를 포함하는 셀에 저장한다. 그림 3에서 셀(2,2)에는 두 개의 에이전트가 저장되어 있다. 그런 후 임의의 지역에 존재하는 모든 에이전트를 찾는 것은 단순히 그 지역에 매핑되는 그리드의 셀만 조사하면 된다. 이러한 공간 분할된 그리드를 이용하여 *agent*에 대한 *kNN*을 찾는 것은 가장 먼저 *agent*의 위치에 매핑되는 그리드의 셀, 즉 C_1에 포함되는 셀에 저장된 에이전트를 찾고 *agent*와 근거리에 있는 셀의 순서로 C_2, C_3, ..., C_m에 포함된 셀들에 저장된 에이전트를 찾으면서 처음으로 k개를 만족하는 C_m에서 멈춘다. 이렇게 하면 경우 모든 에이전트를 탐색하는 대신에 $k'(\geq k)$개의 에이전트만 탐색하면 된다. 이 방법은 평균적으로 $O(kn)$의 시간 복잡도를 갖는다. 3D에서는 그리드 대신 3차원 배열을 사용한다. 그림 4은 2차원 공간분할 무리짓기에 대한 의사 코드이다.

알고리즘 *SteeringPS*와 *Steering*의 입력과 출력은 동일하다. 그림 4에서 *Grid*는 공간 분할된 에이전트를 저장하는 2차원 배열이다. 라인 01~08사이의 내용은 임의의 에이전트가 위치가 변경

```
Algorithm SteeringPS: Inputs: Flocking, k Outputs: None
01: foreach agent in Flocking{
02:   if(IsChangedGrid(agent.pos,agent.oldPos)==true){
03:       (x,y)= GetGridCoordinate (agent.oldPos)
04:       Grid(x, y).DeleteAgent(agent);
05:       (x,y)= GetGridCoordinate(agent.pos)
06:       Grid(x, y).AddAgent(agent)
07:   }
08: }
09: foreach boid in Flocking{
10:   kNN= getKnn(boid, k, Grid)
11:   agent.steerForce= ComputeSteerForce (boid, kNN)
12: }
13: // 이동행동: 각 에이전트에 대한 새로운 위치 계산
```

그림 4. 공간분할에 기반한 조종력 계산 알고리즘

되어 공간 분할 *Grid*에 재배치하는 과정이다. 즉, 함수 *IsChangedGrid*가 '참'을 리턴하는 경우 기존의 셀에서 이 에이전트를 제거하고, 새로운 셀에 이 에이전트를 삽입하여야 한다. 라인 09~12 사이는 함수 *getKnn*을 이용하여 k개의 가까운 이웃을 찾고 *ComputeSteerForce*을 이용하여 조종력을 계산한다. 함수 *getKnn*에서 효율적으로 k개의 가까운 이웃을 찾기 위하여 공간 분할 정보인 *Grid*를 이용한다. 이 알고리즘은 라인 01~08 사이가 $O(n)$이라는 가정하에 $O(kn)$의 시간 복잡도라고 할수 있다. 어쨌든 이 알고리즘은 실제 구현에서 공간 분할의 세분화에 대한 최적화를 고려하여야 한다. 예를 들어, 공간 분할의 세분화를 매우 크게 하는 경우(셀의 수가 매우 많은 경우) 라인 01~08사이에서 함수 *IsChangedGrid*에서 '참'을 리턴하는 경우가 많아 빈번한 삭제/삽입이 발생하여 속도가 느려질 수 있다. 반대로 분할을 적게하는 경우 셀의 수가 적어질 것이고 이러한 경우 함수 *getKnn*의 비용이 증가하게 된다.

2.4 군중 애니메이션

컴퓨터 애니메이션에서 자율 행동이란 에이전트 자체의 지각적인 능력을 갖춘 에이전트의 움직임을 말한다. 보통 자율 행동을 하는 에이전트는 파티클 시스템이나 무리짓기의 에이전트들보다 훨씬 적은 환경에서 시뮬레이션된다. 자율 행동은 보통 동물과 같이 관절로 연결된 객체에서 일어난다. 공중에 날아다니는 비행기, 전투에서의 탱크, 도시 환경에서의 보행자등의 시뮬레이션이 여기에 속한다. 이러한 자율 행동을 하는 에이전트는 주변 환경에 대한 많은 지식을 가져야 한다. 따라서 다른 에이전트 및 주변 환경에 대한 지식을 쌓기 위하여 감지, 기억 및 지각 능력이 있어야 한다. 또한 정적인 행동 규칙을 구성하는 내부 상태를 유지하여야 한다. 통상적으로 내부 상태는 습관, 흥미의 식별 및 감정의 상태를 관리한다. 자율 행동은 감지 기능에 의하여 얻어지는 외부 정보와 감정 등의 상태를 표시하는 내부 정보를 종합하여 자율 행동을 실행한다. 자율 행동 실행은 여러 개의 이동 행동의 순서화된 목록으로 분할된다.

군중 시뮬레이션이 자율 행동의 대표적인 예이다. 군중의 개개인은 군중의 구성원이므로 군중 개개인을 위한 특정 행동이 있다. 이러한 특정 행동중의 하나가 다가오는 사람과의 충돌을 회피하는 충돌 회피이다. [6]에서는 심리학적 이론에 기초하여 충돌에 대하여 충돌 예측, 충돌 회피 및 충돌 회피 후 다시 일상으로 돌아가는 세 가지 범위로 나누었고, 충돌 형태도 정면충돌, 후면충돌 및 측면충돌로 분리하여 연구하였다. [8]에서는 건물 내에서 비상사태가 발생하는 경우 비상구에 대한 시뮬레이션을 함으로써 건물의 설계 시 수용인원에 따른 최적의 비상구를 설계할 수 있도

록 하였다.

3. 집단행동 기술 동향

3.1 무리짓기 행동제어

[4]는 앞에서 언급하였듯이 조종의 힘을 계산할 때 해당 에이전트의 위치를 중심으로 영향을 미치는 이웃 에이전트들을 세 가지 영영으로 나누어 각각 다른 조종의 힘을 구하여 이들을 합하였다. 이렇게 세 영역으로 나누어 조종의 힘을 계산하는 이유는 영역의 크기를 조절하므로써 나타나는 다양한 무리짓기의 행동을 관찰하기 위함이다. 즉, 이러한 영역의 크기를 조절하므로써 가장 자연과 가까운 무리짓기의 형태를 찾을 수 있기 때문이다. 이 논문에서는 그룹 분극화(Group Polarization: p_{group})과 각 모멘텀(Angular Momentum: m_{group})을 정의하였고 이들을 $\Delta r_o(=r_o-r_r)$과 $\Delta r_a(=r_a-r_o)$의 크기에 따라 그룹 분극화와 각 모멘텀이 달라지는 것을 보였으며, 이에 따라 나타나는 무리의 전체 행동도 다르게 나타남을 그림 5와 같이 보였다.

이 본문에서 모든 측정 단위의 기본 길이는 에이전트의 길이를 1로 스케일 하였다. 그림 1에서 r_r은 1이고, r_o, r_a는 0~15까지 변화한 경우 그림

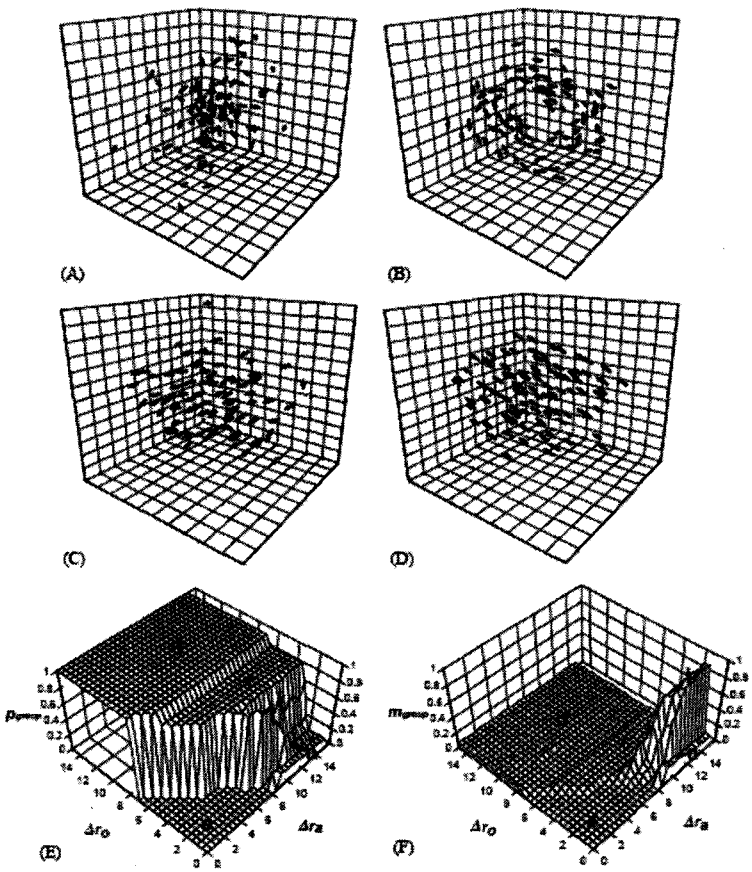


그림 5. Δr_o과 Δr_a에 따른 무리짓기의 다양한 형태

5는 Δr_o과 Δr_a에 따른 다양한 결과를 보이고 있다. 그림 5에서 에이전트의 수는 100이고, α는 270도이다. (E)와 (F)는 Δr_o와 Δr_a의 변화에 따른 p_{group}와 m_{group}의 변화를 나타내고 (E)와 (F)에서의 a, b, c, d는 각각 집단행동 (A), (B), (C), (D)에 해당한다. 즉, Δr_o가 작고, Δr_a가 비교적 큰 경우 p_{group}와 m_{group}값이 매우 작고 이 경우 (A)와 같이 벌레들이 윙윙거리는 집단행동을 보인다. 또한 이 상태에서 Δr_o를 조금 증가시키면 p_{group}는 매우 작으나, m_{group}은 매우 큰 값으로 나타나게 되는데 이 경우가 집단행동은 (B)와 같은 토러스로 나타난다. 이와 같이 이 논문에서는 Δr_o와 Δr_a의 값을 변화시켜 적절한 형태의 무리를 유지할 수 있다.

이 논문의 다른 기여는 무리짓기의 전체 형태(예를 들어 새떼들에 있어서 V자 또는 다이아몬드 형태)가 변할 때 이전의 형태가 새로운 형태에 영향을 준다는 사실을 발견하였다. 즉 동일한 파라미터를 설정하여도 이 파라미타를 설정하기 직전의 무리짓기 형태에 따라 새로운 무리짓기 형태가 정해진다는 사실이다. 이 논문에서는 이러한 현상이 자연에서도 발생하는 것인지는 알수 없다고 하였다.

3.2 무리짓기 병렬처리

이론적으로 무리짓기는 파티클 시스템에서 에이전트간 상호 작용을 하는 모델이다. 따라서 무리짓기는 앞서 설명하였듯이 계산 비용이 매우 많기 때문에 에이전트의 수를 많게 할 수 없다. 그러나 영화나 게임 등에서 현실감을 증대시키기 위하여 이러한 집단행동의 에이전트 수를 증가를 지속적으로 요구한다. [11]에서는 이러한 대규모 무리짓기를 해결하는 방법으로 병렬처리 기술을 사용항 수 있음을 보였다. 이전에도 다양한 무리

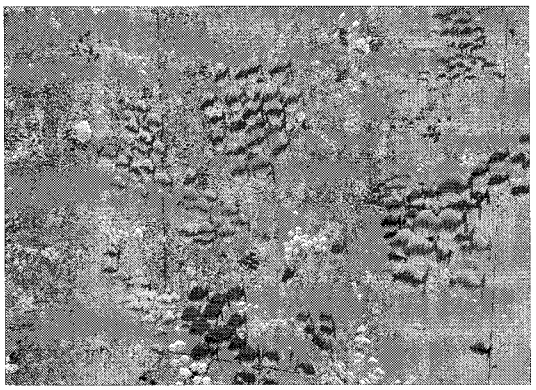


그림 6. PS3에서의 무리짓기 시연 장면

짓기에 대한 다양한 병렬처리가 연구되어 왔다. [2]는 50개의 Transputer를 사용하여 100개의 에이전트들이 무리짓는 것을 보였다. [10]에서는 GPGPU(General Purpose Graphic Processing Unit)를 사용하여 5,000에이전트들에 대해서는 초당 100프레임, 10,000에이전트들에 대해서는 초당 35프레임을 시뮬레이션할 수 있음을 보였다. [11]은 소니사에서 만들어진 PLAYSTATION3에서 2D 환경에서 렌더링을 포함하여 15,000 에이전트에 대하여 초당 60프레임을 시뮬레이션 할 수 있었음을 보였다. 3D 환경에서는 10,000 에이전트에 대하여 초당 60 프레임을 시뮬레이션 하였음을 발표하였다. [11]에서는 이 시뮬레이션을 PSCrowd라 하였는데 이 시뮬레이션에서도 3D 공간 분할 방법을 사용하였다. 그림 6은 [11]에서 10,000마리의 물고기에 대하여 초당 60 프레임을 시뮬레이션하는 시연 장면이다.

3.3 보행자 시뮬레이션

인간 행동에 대한 애니메이션은 컴퓨터 그래픽스 분야의 오래된 과제이며, 아주 중요하다. [5]에서는 도시 환경에서 실제 보행자의 행동을 시뮬레이션 하였다. 인간 행동에 대한 애니메이션은 크

게 자연적인 이동과 이동 계획으로 나누어지는데 [5]는 후자인 이동 계획에 중점을 두고 연구하였다. 여기서 보행자의 모델을 이동, 지각, 행동 및 인식력으로 구성하였다. 이를 위해서 계층적 데이터 구조를 제시하였다.

그림 7은 [5]에서 제안한 환경 모델에 대한 계층적 구조이다. 이 모델은 크게 가상 세계의 각 영역(그림 7에서 A, B, C 및 D)들의 연결하는 위상 맵(Topological Map), 지각적인 질의에 관계되는 정보를 포함하는 지각 맵(Preception Map) 그리고 이동에 대한 이동 계획을 가능하게 하는 패스 맵(Path Map)으로 구성한다. 위상 맵은 그래프로 구성되는데 노드는 가상 세계의 각 영역을 표시하며, 에지는 각 영역 사이의 이동 가능성을 표시한다. 지각 맵은 영역 내 다양한 움직이지 않는 개체들을 표시하는 지역 그리드 맵과 보행자와 같이 움직이는 개체들을 표시하는 전역 그리드 맵으로 구성한다. 패스 맵은 큰 영역에 대한 이동 계획을 위한 쿼드트리와 짧은 영역 이동 계획을 위한 그리드 맵을 구성한다. 쿼드트리의 각 노드는 이 노드에 의하여 표시되는 영역의 위치, 점유 형태, 이웃 노드들에 대한 포인터, 이동 계획을 위한 정보, 보행자들에 의하여 점유된 정보인 밀집

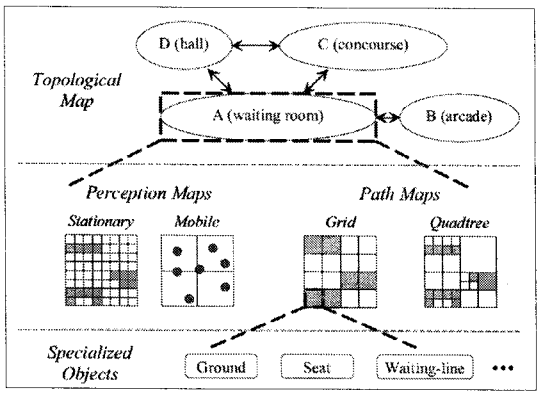


그림 7. 계층적 환경 모델

요소를 가지고 있다.

계층적 환경 모델을 이용하는 자율적 보행자 모델은 이동제어, 지각, 행동 및 인지 모듈로 나뉘어 개발되었다. 이동제어 모듈은 에이전트의 기본적인 움직임을 제어하는 모듈로 *DI-Guy*라는 상업적 모듈의 사용하였고, 지각 모듈은 주변 환경을 지각하는 것으로 그라운드 높이, 정적 주변 객체 및 이동 주변 개체를 앞에서 언급한 계층적 환경 모델을 이용하여 구별한다. 행동 제어 모듈은 주변에 있는 정적 객체와 이동 개체를 인식하여 그 상황에 맞는 행동을 제어한다. 인지 모듈은 목표를 정하고, 목표에 따라 이동 계획을 세우거나 세워진 이동 계획을 수정하는 모듈이다. [5]에서는 실험을 위하여 뉴욕시의 펜실베니아 역을 가상 세계로 택하였고, 이 역을 43개의 영역으로 나누어 위상 맵을 구성하였다. 그리고 실험을 위하여 통근자, 여행자, 행위자, 경찰 및 군인의 5종류의 보행자가 존재하는 것으로 하였다. [5]에서는 2.8GHZ Xeon CPU와 1GB 메모리로 구성된 PC에서 1,400 보행자에 대하여 순수 시뮬레이션만 초당 30 프레임이 가능하다고 하였다. 또한 실험 대상 지역인 펜실베니아 역의 복잡함(그래픽 관점에서) 때문에 시뮬에이션과 렌더링을 포함하는 경우 500 보행자에 대하여 초당 3.8 프레임의 시뮬레이션이 가능하다고 하였다.

4. 결 론

본 논문은 컴퓨터 애니메이션에서 가장 활발히 연구되고 있는 분야 중의 하나인 집단행동에 대한 기술 동향을 소개 하였다. 기술 동향은 초기에 파티클 시스템으로 시작하여 점차적으로 집단내에 존재 하는 에이전트의 지능이 고도화되는 방향으로 발전되고 있음을 보였다. 이러한 에이전트 지

능의 고도화하는 보다 현실감 있는 애니메이션을 제공하는 반면 기술 관점에서는 보다 복잡하고, 지능적인 기술이 요구된다.

파티클 시스템의 경우 기술적으로 많은 발전을 이루었으며, 영화 및 게임 등에서 보조적인 효과로 활발히 사용되고 있다. 최근 이들에 대한 다양한 기능을 갖는 API들이 독립적으로 개발되어 사용되고 있다. 무리짓기의 경우 기본적인 기술들은 상당 부분 명확히 정립된 단계이다. 간단하고 단순한 무리짓기의 경우 이러한 기본적인 기술을 사용하여 많이 사용되고 있으나 보다 정교한 무리짓기의 경우 적용 영역에 따라 별도의 연구가 필요하다. 또한 대규모 무리짓기를 위한 알고리즘 개발 및 병렬처리 등도 주요 연구 과제이다. 군중 애니메이션의 경우 아직도 초보 단계라 판단된다. 따라서 아직도 군중 애니메이션에 대한 기술이 명확히 정립되지 않은 채 많은 연구가 진행되고 있는 실정이다. 파티클 시스템이나 무리짓기의 경우 영화 및 게임 등의 엔터테이먼트 분야에 활용되고 있는 반면, 군중 애니메이션은 돌발 상황에 대한 시뮬레이션, 로봇제어에 대한 활용 등 실 생활에 적용되는 분야이다. 따라서 보다 현실적이고 정교한 기술이 필요한 부분이므로 향후 가장 많은 연구가 필요한 분야이다.

참 고 문 헌

[1] W. T. Reeves, "Particle systems: a technique for modeling a class of fuzzy objects," In Proceedings of SIGGRAPH '83. Computer Graphics, 1983, pp. 359-376.

[2] H. Lorek & M. White: "Parallel bird flocking simulation" Parallel Processing for Graphics and Scientific Visualization, 1993.

[3] Reynolds, C. W., "Steering Behaviors For Autonomous Characters," in Proc. of the 1999 Game Developers' Conference, 1999, pp. 763-782.

[4] Iain D. Couzin, Jens Krause, Richard James, Graeme D. Ruxton and Nigel R. Franks, Collective Memory and Spatial Sorting in Animal Groups, J. theory Biol., 2002, pp. 1-11.

[5] W. Shao and D. Terzopoulos, "Autonomous pedestrians," Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 19-28, 2005.

[6] S. Rymill and N. Dodgson, "A Psychologically-Based Simulation of Human Behavior," Theory and Practice of Computer Graphics 2005, pp. 25-42.

[7] Iaim D. Couzin, Jens Krause, Nigel R. Franks and Simon A. Levin, Effective leadership and decision making in animal groups on the move, NATURE Vol. 433, 2005, pp. 513-516.

[8] Pan Xiaoshan, Han C S, Law K H. A multi-agent-based simulation framework for the study of human and social behavior in egress analysis, In: Proceedings of the ASCE International Conference on Computing in Civil Engineering. American Society of Civil Engineers., 2005: 969-980.

[9] Mat Buckland, Programming Game AI by Example, Wordware Publications, 2005.

[10] N. Courty, Nicolas ; S. R. Musse, "Simulation of Large Crowds in Emergency Situations Including Gaseous Phenomena". In: IEEE CGI 2005 - Computer Graphics International, 2005, Stony Brook, 2005.

[11] Reynolds, C., "Big Fast Crowds on PS3," In Proceedings of Sandbox (an ACM Video Games Symposium), Boston, Massachusetts, July 2006.

[12] CLEARY, P. W., PYO, S. H., PRAKASH, M., AND KOO, B. K. Bubbling and frothing liquids. ACM Trans. Graph. (SIGGRAPH Proc.) 26, 3, 2007, pp. 971-976.

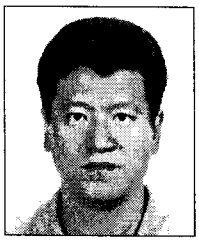
[13] 이재문, "대표 보이드를 이용한 대규모 무리의

효율적인 무리짓기," 한국게임학회 논문지, 제8권 제3호, 87-96, 2008

[14] Jeong-Mo Hong, Ho-Young Lee, Jong-Chul Yoon and Chang-Hun Kim, Bubbles Alive, ACM Transactions on Graphics (In Proceedings of ACM SIGGRAPH 2008), Vol. 27, Issue 3, 2008. pp. 481-484.

[15] Rick Parent, Computer Animation Algorithms & Techniques, Elsevier, 2008.

[16] 이재문, "이전 k 개의 가장 가까운 이웃을 이용한 무리짓기에 대한 공간분할 방법의 개선," 한국게임학회 논문지, 제9권 제2호, 2009.



이 재 문

- 1986년 한양대학교 전자공학과 (공학사)
- 1992년 한국과학기술원 전기및전자공학과 (공학박사)
- 1992년~1994년 한국전기통신공사
- 2002년~2003년 호주 시드니대학교 교환교수
- 1994년~현재 한성대학교 멀티미디어공학과 교수
- 관심분야 : 게임, 데이터베이스, 기계학습

엄 종 석

- 1982년 연세대학교 응용통계학과(학사,석사)
- 1985년 한국해양연구소
- 1986년~1991년 오하이오주립대학(Ph.D in Statistics)
- 1992년~현재 한성대학교 멀티미디어공학과 교수
- 관심분야 : 컴퓨터비젼, 데이터마이닝, 게임