JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 17, No. 12 : 2937~2943 Dec. 2013

방사기저함수(RBF) 기반 벡터 필드를 이용한 실시간 군집 시뮬레이션

성만규*

Real-time Flocking Simulation through RBF-based Vector Field

Mankyu Sung*

Dept. of Game&Mobile Contents, Keimyung University, Daegu, 704-701, Korea

요 약

본 논문은 방사기저함수(Radial Basis Function)를 이용한 실시간 군집 시뮬레이션 프레임웍을 제안한다. 제안된 프레임웍에서는 군집이 존재하는 환경을 격자 구조로 모델링 한 후, 격자 구조 각 셀에 하나의 방향 벡터를 할당한 벡터필드를 선형 방사기저함수를 이용하여 실시간으로 합성한다. 방사기저함수를 통한 벡터필드 생성 시, 마우스를 통한 제어라인(Control line)을 이용하며, 이 벡터필드 위에서 군집들은 벡터 필드 흐름에 따라, 자신의 움직임을 결정한다. 방해물과의 충돌회피는 반발벡터필드로 모델링하여, 기존의 벡터필드에 오버레이 하여 이용하고, 다른 캐릭터와의 충돌회피는 lattice-bin 알고리즘에 빠른 충돌회피를 수행한다.

ABSTRACT

This paper introduces a real-time flocking simulation framework through radial basis function(RBF). The proposed framework first divides the entire environment into a grid structure and then assign a vector per each cell. These vectors are automatically calculated by using RBF function, which is parameterized from user-input *control lines.* Once the construction of vector field is done, then, flocks determine their path by following the vector field flow. The collision with static obstacles are modeled as a repulsive vector field, which is ultimately over-layed on the existing vector field and the inter-individual collision is also handled through fast lattice-bin method.

키워드 : 군집시뮬레이션, 방사기저함수, 벡터필드

Key word : Flocking, Radial Basis Function, Vector Field

접수일자 : 2013. 08. 07 심사완료일자 : 2013. 09. 02 게재확정일자 : 2013. 09. 23

* **Corresponding Author** Mankyu Sung(E-mail:mksung@kmu.ac.kr, Tel:+82-53-580-6684)
Dept. of Game&Mobile Contents, Keimyung University, Daegu, 704-701, Korea

Open Access http://dx.doi.org/10.6109/jkiice.2013.17.12.2937 print ISSN: 2234-4772 online ISSN: 2288-4165

ⓒThis is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License(http://creativecommons.org/li-censes/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Copyright © The Korea Institute of Information and Communication Engineering.

Ⅰ. 서 론

군집시뮬레이션은 영화 혹은 애니메이션과 같은 콘텐츠의 스펙터클한 장면에 주로 쓰이는 기술로서, 해당 콘텐츠의 전반적인 흐름 상 가장 중요한 장면에 주로 사용되기 때문에, 콘텐츠 전반적인 기술적 수준을 나타내는 핵심기술이다(그림 1)[1]. 군집 시뮬레이션에서는 하나의 캐릭터가 아닌, 수백, 수천 명의 움직임을 동시에 디자이너의 의지대로 제어하여야 하며, 개별적인 움직임 보다 전체적인 움직임에 대한 제어가 더 중요하다[2]. 시간적 흐름에 따른 포메이션의 변화는 애니메이터 혹은 디자이너의 간략한 스케치를 이용하여 실시간으로 반영되어야 하고, 방해물과의 충돌 및 서로 간의 충돌회피 또한 자동으로 이루어져야 한다.

본 논문은 벡터 필드(Vector Field)를 이용한 새로운 군집 제어 프레임웍을 제안한다. 벡터 필드는 2차원, 혹은 3차원 배열로 이루어진 벡터의 집합으로서 군중이 존재하는 공간에 임베디드(embeded)되어, 군집의 흐름을 조절한다. 군집이 존재하는 공간을 격자 구조로 나누어, 격자마다 하나의 벡터를 할당한 후, 이 벡터는 디자이너가 공간상에 직접 입력한 제어라인(Control line)을 이용한 방저기함수를 통해 자동으로 계산된다. 고정적인 방해물의 경우 새로운 반발벡터 필드를 기존의 벡터 필드에 오버레이하여 충돌회피를 하며, 개체 간 충돌회피는 래티스-빈 방법을 통해 충돌체크의 횟수를 최소화하여 빠른 충돌회피를 가능케 한다. 제어라인을 이용한 방저기함수 기반 벡터 필드 디자인은, 실시간으로 벡터 필드를 변경 가능하므로, 실시간으로 디자이너의 의도대로 군집의 움직임을 제어할 수 있다.

그림 1. 영화에서의 군집(군중) 장면 [1]
Fig. 1 A Crowd scene in a movie[1]

Ⅱ. 관련 연구 동향

군집 혹은 군중 시뮬레이션은 크게, 흐름(flow) 기반 접근방법, 개체(Entity) 기반 접근방법 그리고 에이전트 기반 접근방법으로 나뉜다[3]. 흐름 기반 접근방법은, 전체적인 군집의 움직임을 유체(Fluid)를 움직임으로 모델링 하는 기법이다. 이 방법은 전체적인 군중의 흐름을 자연스럽게 표현할 수 있으나, 디자이너의 의도대로 흐름을 제어하기가 어렵다는 단점이 있다. 이 단점을 극복하기 위해 Chenney 등은 타일 기반으로 흐름을 제어 하는 기법을 개발하였다[4].

이 방법은 블록을 조립하듯이 공간상에 흐름 블록을 배치하여 군중의 흐름을 제어하는 방법이다. 본 논문에서 제안하는 프레임웍은 흐름 기반 군중 모델링 방법의 하나로, 군중의 흐름 벡터필드를 이용하여 조정하며, 벡터필드 디자인 위해 방저기 함수를 이용하여 제어성(Controllability)을 높인다. 개체 기반 접근방법은 개체들을 동일한 성질의 속성을 가진 개체(Entity)로 모델링 하는 방법으로 대표적으로 파티클 시스템이 있다. 물리학자 Helbing이 제안한 사회적 힘 모델(Social Force Model)은 이와 같은 파티클 시스템에 기반을 두어 개체의 움직임이 다양한 사회적, 심리적인 힘에 의해 영향을 받도록 모델링한 기법이다[5].

이 모델링을 통하면 좁은 문을 통해 통과하는 다양한 군중의 움직임 등을 효율적으로 시뮬레이션할 수 있다. 하지만 전체적인 포메이션 제어가 어려우며, 개체만의 독특한 성질을 부여할 수 없다. 에이전트 기반 접근방법은 하나하나의 개체를 자율적이고, 지적능력을 갖춘 에이전트로 가정하고, 각자 스스로 행동을 결정하는 방법을 말한다. 이 방법은 시뮬레이션 시, 한 명씩 차례대로 다음 행동을 미리 정해진 룰들을 통해 결정하게 된다. 이 방법은 개체의 다양성을 표현할 수 있지만, 개체 수가 늘어남에 따라 전체적인 시뮬레이션 속도도 기하급수적으로 늘어나는 단점이 있다. 이 세 가지 접근방법 외에, 다양한 디포메이션 방법을 군집시뮬레이션에 도입하여 군중의 포메이션을 제어하는 방법 또한 제안되었다 [6].

Ⅲ. 시뮬레이션 프레임웍

3.1. 방사기저함수(Radial Basis Function)

방저기함수는 복잡한 다변수 함수(Multivariate)함수를 단변량함수(univariate)의 선형 결합(linear combination)으로 근사하기 위한 하나의 보간 방법으로 주어진 실측 데이터를 이용하여 원함수를 추정하기 위해 많이 사용되는 기법이다 [7]. 즉, 원함수를 $g(x)$라고 하면, 이 함수를 방저기 함수 $s(x)$를 이용하여 근사한다. 이를 위해 새로운 함수 $p(x)$와 서로 다른 N개의 포인트들, $X=\{x_1,x_2,x_3,...x_N\} \subset R^d$에서 실제 취득한 실측값을 이용하여 추정한다.

방사기저함수 함수 $s(x)$ 는 다음 수식 (1)과 같은 형태를 취한다.

$$s(x) = p(x) + \sum_{i=0}^{N-1} \lambda_i \phi(|x - x_i|), \quad x \in R^d \quad (1)$$

이 수식에서 $p(x)$는 최대 k 차의 다항식으로, 계산의 편리성을 위해 생략 가능하며, λ_i 는 실수 값을 가지는 가중치, $|\bullet|$는 유클리디안 거리(Norm), $\phi()$는 베이시스 함수, $|x - x_i|$는 x와 x_i 사이의 거리를 나타낸다. 커널이라고도 불리는 베이시스 함수 $\phi()$는 양수 값을 갖는 거리 값 r을 입력받아, 함수의 가중치 값을 리턴 한다. 본 연구에서는 이 베이시스 함수로서 간단한 역 다중2차(Inverse multiquadrics) 베이시스함수를 이용 한다 [7].

$$\phi(r) = (1/\sqrt{r})^2 \quad (2)$$

본 연구에서는 이와 같은 방저기 함수를 2차원의 벡터 필드에 적용하여 군중의 움직임을 제어 하는 알고리즘을 제안한다.

3.2. 벡터 필드 디자인

군집 내 개체들의 움직임은 격자구조로 모델링된 2차원 벡터 필드를 이용해 제어된다. 격자구조의 각 셀에는 하나의 벡터를 할당하며, 벡터의 방향은 개체들이 움직이는 방향을 제어한다. 그림 2는 격자구조에 할당된 벡터필드의 일부분과 그 벡터필드 위에 놓인 개체를 나타낸다. 벡터 필드위에 놓인 개체들은 벡터에 지정된 벡터에 따라 본인의 움직임을 결정한다. 본 논문은 2차원 벡터 필드를 기준으로 설명 하지만, 제안된 모든 알고리즘은 3차원으로 모두 쉽게 확장 가능하다. 확장을 위해서는 2차원 격자는 3차원 박스로 확장되며, 2차원 벡터는 3차원 벡터로 설정하면 된다.

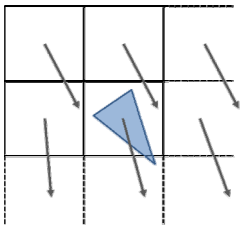

그림 2. 2차원 벡터필드와 개체
Fig. 2 2D Vector field and an individual moving entity

격자 관리를 편리하게 하기 위해 모든 격자는 정사각형의 모습을 갖는다고 가정한다. 격자의 길이(높이)를 w 라고 가정하고 개체가 존재하는 공간의 너비(x축)와 높이(y축)를 각각 폭(width)과 높이(height)로 가정하면, x축으로의 격자의 갯수 $n_x = width/w$, y축으로의 격자의 갯수는 $n_y = height/w$ 로 계산되며, 전체 격자의 갯수, 즉 벡터필드 내 벡터의 갯수는 $n_x \times n_y$ 이다. 이 벡터들은 인덱스의 접근 편의성을 위해 2차원 배열로 저장된다.

벡터 필드를 디자인하기 위해서 제안된 알고리즘은 정규 방사기저함수(Normalized Radial Basis Function Network)을 이용한다. 정규 방사기저 함수거지 함수는 거리 값을 정규화한 함으로서 상대적인 가중치값을 이용 하는 방사기저함수이다. 이를 위해 사용자로부터 마우스를 통한 제어라인(Control Line)을 입력받는다. 본 논문에서 제안하는 알고리즘에서는 제어 라인을 방사기저함수에서 필요한 실측 데이터로 활용한다. 즉, 마우스를 통해 입력받은 N개의 선분을 제어라인으로 활용하여, 공간상의 모든 벡터필드값을 제어라인과의 거리 값을 이용한 방사기저함수를 통해 자동으로 계산한다. 이를 수학적으로 표현하면, 벡터 필드 내 m개의 벡터 $V=\{v_{x1}, v_{x2},....v_{xm}\}$, $x_i \in R^2$ 라고 하고(x_i는 v_i가 존재하는 격자의 정 가운데 위치), 입력받은 제어라인 $f=(x_s,x_e)$는 시작 포인트 x_s와 끝점 x_e로 이루어져

있다고 선분으로 가정하자.

N개의 제어라인을 $f=\{f_1(x_s,x_e),..,f_N(x_s,x_e)\}$라고 하면, 수식 (2) 에서 나타낸 베이시스 함수를 이용하여, 벡터 v_i의 위치 x_i와 제어라인의 시작점 x_s간의 거리값 $r=|x_i-x_s|, x_i,x_s \in R^2$를 계산한다. 이 거리값을 수식 (2)에 대입하면 베이시스 함수 값을 구할 수 있다. 본 논문에서는 이 값을 정규화 하여 사용한 정규 선형 방사기저함수를 이용한다. 베이시스 함수를 정규화(Normalized) 한 후, 제어라인 벡터에 곱하여 새로운 벡터를 구한다. 즉, 벡터필드 안의 벡터 v_i 는 다음 수식 (3)에 의해 계산된다. 본 수식에서 $f(x)$는 사용자로부터 입력받은 제어라인을 의미한다.

$$v_i = v_i + \sum_{i=1}^{N}\left[\frac{\phi_i(r)}{S} f_i(x_e)\right] \tag{3}$$
$$S = \sum_{i=1}^{N}\phi_i(r)$$

수식 (3) 에서 S는 모든 가중치 값의 합을 나타내며 이를 각 가중치로 나누어 정규화 한다. 이 정규화된 가중치를 제어라인 벡터에 곱한 후, 각 벡터필드 값을 갱신한다.

공간 내 존재하는 방해물(Obstacle)에 대한 벡터 필드는 반발벡터(repulsive vector)로 모델링 하여 기존의 벡터필드에 더해진다. 방해물의 위치 및 크기는 바운딩 서클(Bounding Circle)로 모델링한다. 바운딩 서클은 방해물을 전부 포함하는 최소크기의 원을 의미한다. 개체들이 방해물에 접근했을 때, 미리 방해물을 인지하고 회피하여야 하므로, 바운딩 서클보다 k배 큰($k>1.0$) 원을 이용한다. 이 과정은 수식 (4)에 나타나 있다.

$$v_i = v_i + k\hat{r_i} \tag{4}$$
$$\hat{r_i} = \frac{x_i - c}{\| x_i - c \|}$$

수식 (4)의 파라미터 k는 얼마만큼 개체들이 미리 충돌을 인지하는지를 조절하게 된다. 격자로 모델링 된 벡터필드의 위치를 바운딩 서클과 비교하여, 격자의 일부가 이 서클 안에 들어오면, 반발벡터로 세팅한다. 서클의 중심위치를 c 라고 하고, 서클 안에 들어 있는 격자들의 중심점의 위치를 x_i 라고 한다면 반발벡터 $\hat{r_i}$ 는 수식 (4)에 의해 계산된다. 이 $\hat{r_i}$ 벡터는 기존의 벡터 값인 v_i 에 더해져 업데이트된다. 이때 기존의 v_i이 $\hat{r_i}$ 과 반대 방향을 가질 경우, 최종 값이 상쇄될 수 있다.

3.3. 개체 모델링

군집을 이루는 개별 개체들은 초기로 랜덤하게 설정된 위치를 시작으로 서로 간의 충돌을 회피하면서 벡터필드의 벡터에 따라 움직인다. 개체들을 모델링하기 위해 위치(p), 속도(v), 가속도(a)를 속성 값으로 부여하며, 상수 값으로 최대 속도 MAX_SPEED 와 최대 힘의 크기인 MAX_FORCE값을 설정한다. 위치, 속도 및 가속도 값의 데이터 타입은 모두 벡터이며, 최대속도 및 최대 힘은 스칼라 값을 가진다. 본 논문에서 제안하는 시뮬레이션 프레임웍에서 개체들의 움직임은 개체의 가속도 값을 조정함으로서 이루어진다. 가속도 값을 기반으로 하는 시뮬레이션은 부드러운 개체의 움직임을 얻을 수 있으며 시뮬레이션 절차가 단순화 할 수 있는 장점이 있다 [8]. 가속도와 힘과의 관계는 뉴턴의 제2법칙인 $F=ma$를 통해 나타낼 수 있다. 이 법칙에서 비중을 1로 가정하면, $F=a$로 되며 이를 기반으로 환경에서 발생하는 다양한 힘(Force)들을 가산(add)하여 개체의 가속도 값으로 설정한다. 제안된 프레임웍에서 개체의 움직임을 제어하기 위한 힘(F_i)은 개체들은 벡터필드를 따라 개체가 움직이는 1) 흐름 필드 힘(flow field force)와 개체 간의 충돌회피를 위한 2) 개체 간 충돌회피 힘(collision avoidance force)을 결합하여 사용한다. 즉, 개체 A의 가속도인 A_a를 수학적으로 표시하면 수식 (5)와 같다.

$$A_a = \alpha F_d + \beta F_c \tag{5}$$
$$F_d = (v_i - A_v)$$
$$F_c = (v_o - A_v)$$

수식 (5) 에서 F_d와 F_c 는 각각 흐름 필드 힘과 충돌회피 힘을 나타내며, α와 β는 이 두 힘의 가중치를 나타낸다. F_d은 수식 (3)과 (4)로부터 계산된 벡터 필드 값 v_i와 개체 A의 현재 속도인 A_v의 차이를 통해 계산되며, 개체 간 충돌회피를 위한 F_c는 주변 개체들의 속도를 반영하여 계산된 충돌 회피 벡터 v_o 와 현재 속도

인 A_v의 차이로 계산된다. 본 연구에서는 개체 간 충돌 방지에 가중치를 더 부여하기 위해 α를 1로, β를 2로 설정한다.

3.3.1. 흐름 필드 힘(F_d=flow field force)

흐름 필드 힘은 개체들을 벡터필드를 따라 움직이게 하기 위한 힘이다. 이 힘은 개체의 현재 위치인 p를 파라미터로 입력받아, 위치 p에 해당하는 벡터필드의 벡터값을 반환한다. 벡터필드내의 특정 위치상의 벡터를 구하기 위해서는 위치정보 p를 이용하여 2차원 배열상의 인덱스를 계산하여야 한다. 이를 위해 p의 각 축을 격자 크기인 w로 나누어 정수형으로 캐스팅 한 후 2차원 배열의 인덱스를 구한다. 계산된 인덱스 값을 이용하여 벡터 필드 상의 벡터 v_i 구한 후 수식 (5)를 이용하여 F_d를 계산한다.

3.3.2. 개체 간 충돌회피 힘(F_c=Collision Avoidance Force)

개체들 간의 충돌은, 정적인 방해물과의 충돌회피가 아닌 동적으로 움직이는 개체들 간의 충돌을 회피하여야 하므로, 매 프레임 충돌체크의 횟수를 줄이는 것이 가장 중요하다. 이를 위해 본 프레임웍에서는 래티스-빈 방법을 통해 충돌체크의 횟수를 줄이도록 하였다. 래티스-빈 방법은 공간을 여러 개의 격자 형태로 나눈 후 각 격자에 개체들의 포인터를 저장하기 위한 리스트를 할당한다. 이후, 실행 타임에 동적으로 움직이는 개체들의 움직임을 추적하여 특정 격자 안으로 개체들이 들어오면 해당 격자에 할당된 포인터 리스트에 개체의 포인터를 추가하고, 격자를 벗어나면 리스트 안의 개체 포인터를 삭제한다. 즉, 격자 안의 리스트에는 현재 격자 안에 머무르고 있는 개체들의 목록을 저장한다. 래티스-빈 방법은 특정 개체가 들어 있는 격자 내 리스트에 존재하는 개체들만을 대상으로 충돌검사를 수행함으로써 빠른 충돌 검사를 가능케 한다.

즉, 특정 격자 내 리스트에 포함되는 개체들의 목록 $A = \{A^1, A^2, \ldots A^m\}$라고 하고, 개체 A^i의 위치를 A_p^i라고 하면, 실제 개체 간 충돌검사는 개체 목록 A중에 A^i와의 거리가 r 미만인 새로운 개체들인 목록인 A^*를 대상으로 한다.

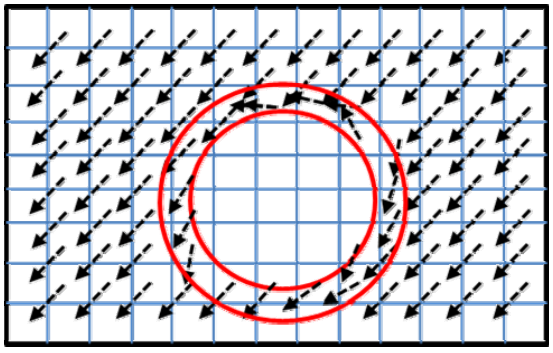

그림 3. 충돌회피를 위한 반발벡터 모델링
Fig. 3 Repulsion vector modeling for collision avoidance

거리 임계값r은 개체가 존재하는 전체 공간의 크기에 따라 달라지지만 본 연구에서는 25로 설정하였다. A^*을 수학적으로 표시하면 다음 수식 (6)과 같다.

$$A^* = \{A_i | r < \| A_p^i - A_p^k \|, i \neq k,\ 0 \leq k \leq m\} \tag{6}$$

만일 A^*가 공집합일 때에는 가까운 거리에 충돌을 일으킬 만한 개체가 존재 하지 않으므로 $F_d = 0$로 계산된다. 그렇지 않다면 특정 개체인 A^i의 F_c은 수식 (7)에 의해 계산되다.

$$\begin{aligned} d_{i,j} &= A_p^i - A_p^j, A^j \in A^* \\ s_i &= \sum_{j=1} (\widehat{d_{i,j}}) / \| d_{i,j} \| \\ v_o &= s_i / k \end{aligned} \tag{7}$$

수식 (7)에 나타나 있듯이 F_c를 계산하기 위해서 A^* 내 모든 다른 개체의 위치와의 차이 $d_{i,j}$를 구한다. 다음에 이를 정규화(normalized) 후에 벡터의 길이로 나누어, 평균을 내면 수식 (7)의 v_o 가 계산되며, 이 후에 현재 속도와의 차이를 구함으로써 최종적으로 개체 간 충돌회피 힘인 F_c를 계산할 수 있다.

계산된 F_d와 F_c 는 수식 (5)에서 나타나 있듯이 가중치 값과 곱해져 최종 힘을 구하며 이 값은 곧바로 가속도 값(A_a)로 설정된다. 만약 최종 힘의 크기가 상수로 지정된 MAX_FORCE와의 비교하여 만약, 이 힘이 그 값보다 크다면 MAX_FORCE 이상 나올 수 없도록

자동으로 조정된다.

Ⅳ. 실험 결과

제안된 프레임웍은, 자바(Java) 기반의 비주얼프로그래밍 언어인 프로세싱(http://processing.org)을 통해 구현되었다. 개체가 존재하는 전체 공간의 크기는 윈도우 크기에 맞추어 설정되었으며, 실행 속도를 파악하고자 윈도우 아래에 현재의 프레임 속도를 표시하였다. 그림 4은 제어라인(그림의 빨간색 선)를 기반한 방사기저함수를 통해 벡터 필드를 디자인한 것이다. 제어라인은 시뮬레이션이 시작하기 전, 혹은 실행타임에도 언제든지 추가할 수 있으므로, 실시간으로 개체의 움직임을 제어할 수 있다. 디자인은 마우스 드래깅을 이용한다 즉, 마우스를 통해 원하는 제어라인의 시작점을 클릭 한 후에, 드래깅하면 원하는 제어라인을 생성 할 수 있다.

그림 5는 200개의 개체를 벡터 필드를 통해 시뮬레이션하는 스크린 샷들이다. 개체들은 서로 간의 충돌을 회피하면서, 원으로 표현되어 있는 방해물을 자동으로 피하며, 벡터 필드를 따라 움직인다. 그림 5에서 나타나 있듯이 전체 벡터 필드는 9개의 제어 라인을 통해 디자인되었으며, 개체들이 스크린 밖을 나가면 자동으로 초기화하도록 설정하였다. 스크린 밑의 현재 프레임 속도는 59~60 이상 나오는 것을 확인하였다. 본 시뮬레이션 결과는 동영상 사이트 Youtube에서 확인할 수 있다.(http://youtu.be/SbbRJe6eDUA). 동영상을 보면 벡터의 크기가 0가 될 경우 필드 내에 벡터의 모습이 나타나지 않는다.

제안된 알고리즘의 가장 큰 장점은 높은 제어성에 있다. 사용자는 2-3개의 제어라인 만을 공간상에 설정하면, 알고리즘은 방저기 함수를 통해 전체 벡터 필드를 자동으로 생성한다. 표1은 동일한 벡터 필드를 수작업을 통해 생성 할 때와 제안된 알고리즘을 이용하여 생성했을 때의 시간을 비교 한 것이다. 테이블에서 나와 있듯이 100x100개의 벡터 필드를 생성 시 알고리즘을 이용한 자동 설정이 훨씬 적은 시간이 소요됨을 알 수 있다.

표 1. 수작업을 통한 벡터 필드와의 생성 시간 비교
Table. 1 Time comparison between automatic generated vector field against manually created vector field

	제안된 알고리즘을 통한 벡터 필드 생성	수작업을 통한 벡터필드 생성
생성시간	0.5sec	12min
벡터필드크기	100x100	100x100

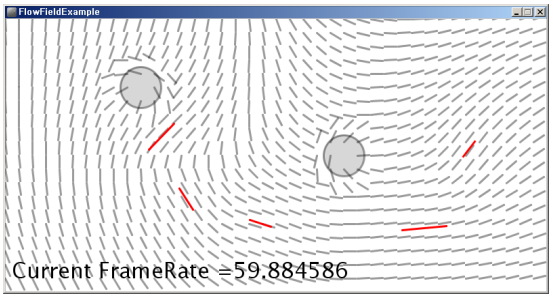

그림 4. 방사기저함수를 통한 벡터 필드 디자인
Fig. 4 Vector field design using radial-basis function

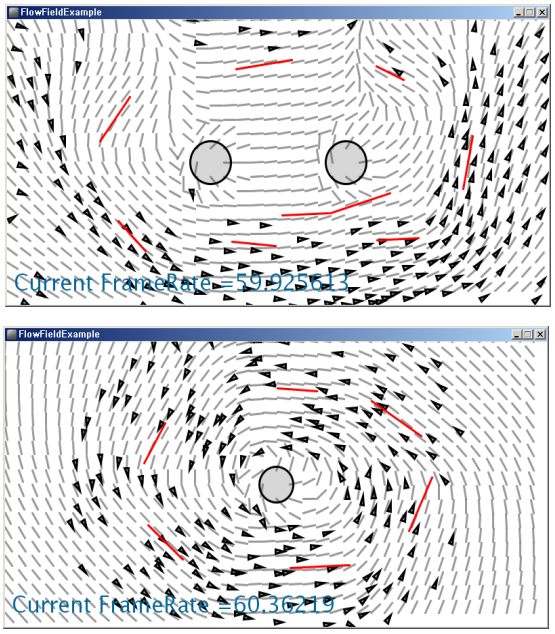

그림 5. 벡터 필드를 이용한 개체 군집 시뮬레이션 스크린 캡처
Fig. 5 An examples of crowd simulation using a vector field generated through radial-basis function.

V. 결 론

본 논문은 방사기저함수를 통하여, 공간에 임베디드 되어 있는 벡터 필드를 실시간으로 디자인하여, 군집을 실시간으로 제어하는 새로운 프레임웍을 제안하였다. 본 프레임웍에서는 방사기저함수를 통해 단순한 몇 개의 제어라인을 통해서도 벡터필드를 전체적으로 디자인할 수 있으므로 쉽게 군집의 움직임을 조정할 수 있는 장점이 있다. 또한, 공간 내 존재하는 방해물 및 개체간의 충돌 회피 또한 자동으로 수행되므로 자연스러운 부드러운 궤적을 얻을 수 있다. 추후에, 제안된 프레임웍을 플러그 인 형태로 바꾸어, 마야 등 실제 콘텐츠 제작 파이프라인에 결합하면 수많은 개체들이 필요한 군집 장면에 효과적인 제작 툴로 사용될 수 있으리라 기대한다.

또한, 실험결과 방해물이 존재하고 두 개체간의 충돌하는 힘이 클 때, 방해물과 어쩔 수 없이 충돌이 발생하는 경우가 발생하는 경우가 있음을 발견하였으나 추후 연구를 통해 바운딩 서클이 아니라 좀 더 세밀 하게 방해물을 포함 시킬 수 있는 바운딩 박스를 이용한 충돌 감지 연구를 지속 할 예정이다.

REFERENCES

[1] Massive Software, http://www.massivesoftware.com/film.html, 2009

[2] C. W. Reynolds, “Flocks, herds and schools: a distributed behavioral model,” *in ACM SIGGRAPH ’87 Conference Proceedings*, pp. 25 - 34, 1987.

[3] S. Zhou, D. Chen, W. Cai, L. Luo, M.Y.Hean Low, F. Tian, V. S. Tay, D. W. Szeong and B.D.Hamilton., “Crowd Modeling and Simulation Technologies,” *ACM Trans. on Modeling and Computer Simulation*, vol. 20, no. 4, 2010.

[4] S. Chenney, “Flow tiles”, *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on computer animation*, pp. 233 - 242, 2004.

[5] D. Helbing, I. Farkas, and T. Vicsek, “Simulating dynamical features of escape panic”. *Letters to Nature*, vol. 407, pp. 487 - 490, 2000.

[6] M. M. Kim, K. L. Hyun, J. M. Kim, J. H. Lee, “Synchronized Multi-Character Motion Editing”, *ACM Transactions on Graphcis (SIGGRAPH 2009)*, vol. 28, no. 3, 2009.

[7] M. D. Buhmann, *Radial Basis Functions: Theory and Implementations*, Cambridge Monographs on Applied and Computational Mathematics, 2009.

[8] C. W. Reynolds, “Steering behaviors for autonomous characters”, *Game developers conference*, 1999.

성만규(Mankyu Sung)

1993년 2월 충남대학교 전산학과 학사
2005년 12월 (미) 위스콘신대학 컴퓨터사이언스 석사
2005년 12월 (미) 위스콘신대학 컴퓨터사이언스 박사
2006년 2월 ~ 2012년 2월 한국전자통신연구원(ETRI) 선임연구원
2012년 3월 ~현재 계명대학교 게임모바일콘텐츠학과 조교수
※관심분야 : 컴퓨터 그래픽스, 컴퓨터 애니메이션, 컴퓨터 게임, HCI