한국산학기술학회논문지
Vol. 11, No. 12 pp. 5083-5088, 2010

NPC 인공 지능을 위한 무리짓기 구현

유현지[1], 이면재[1], 김경남[2]
[1]백석대학교 정보통신학부
[2]중앙대학교 첨단영상대학원

Flocking Implementation for NPC AI

Hyun-ji yoo[1], Myoun-Jae Lee[1] and Kyoung-Nam Kim[2]
[1]**Department of Information & Communication, Baekseok University**
[2]**GSAIM, Chung-Ang University**

요 약 무리를 형성하는 NPC들의 인공 지능을 실세계의 무리짓기와 유사하게 구현하는 것은 게임의 재미를 증가시키는 요인이 될 수 있다. 이를 위하여, 본 논문에서는 분석된 실세계에서의 물고기 무리짓기의 행동 패턴을 설계하고 오우거 엔진을 이용하여 구현한다. 구현된 무리짓기의 효용성을 판단하기 위하여 실세계의 물고기 떼의 행동 패턴과 비교한다. 비교 결과, 구현된 물고기 떼의 행동 패턴과 실세계의 행동 패턴은 비슷함을 보인다.

Abstract An implementation of NPC AI(artifical intelligence) is similar with real world's flocking can increase fun factor of game. To this end, we design fish flocking patten of analyzed real world, implement using Ogre engine in this paper. To determine the usefulness of implemented fish flocking, we compare fish flocking in real world with implemented fish flocking. Implemented behavioral patterns of fish flocking show similar behavioral patterns of fish flocking in real world.

Key Words : Artificial intelligence(AI), NPC flocking, NPC AI

1. 서론

우리가 살고 있는 현 세상에서는 다양한 동물들이 무리를 지어 행동한다. 새 무리들이 일정한 패턴을 만들어 목적지까지 날아가거나, 물고기들이 떼를 지어 다니거나, 개미들이 일렬로 맞추어 다니거나 벌들이 무리를 지어 공격하는 것 등의 여러 동물들이 떼를 지어 생활하고 있다. 이러한 실세계의 무리짓기(Flocking)에 대한 연구는 생물학적 관점으로 진행되고 있다. 새가 있을 경우와 없을 경우에 열대 숲에서 병정 개미들이 먹이 포획 양을 측정하거나[1], 병정 개미들이 먹이를 먹을 수 있는 세 개의 경로를 만들어 놓은 후 각 병정 개미들의 움직임을 살펴보거나[2], 먹을 것을 제공하거나 물고기의 찢겨진 조각을 어항에 던져서 물고기떼에게 공포감을 조성하는 경우 등에서 물고기들의 무리짓기 형태를 분석하거나[3], 무리들 가운데 정보를 전달하는 방법[4]에 관한 연구 등이 있다. 그림 1은 병정 개미와 물고기 무리짓기를 보여주고 있다.

(a) 병정 개미 (b) 물고기

[그림 1] 무리 짓기

게임에서 무리짓기는 그룹을 형성하는 NPC(Non Player Character)의 행동에 주로 사용된다. 대부분의 게임에서 NPC들은 이동 시스템과 애니메이션 재생 시스템을 통하여 조정된다[5]. 즉 무리를 형성하여 일정한 패턴

*교신저자 : 김경남(hsfruit@lycos.co.kr)
접수일 10년 10월 15일 수정일 10년 10월 28일 게재확정일 10년 12월 17일

으로 캐릭터를 공격하거나, 도망갈 때 사용된다. 또한 일정한 패턴으로 무리를 지어 특정 목표에 난입하거나 대열을 맞추어서 이동하는데 사용된다. 이러한 방법들은 무리에 속한 모든 NPC에 적용된다. 무리에 적용되는 AI(Artifical Intelligence) 알고리즘 같은 것이 단순할 경우 움직임의 패턴이 단순하므로 게임의 재미가 반감된다. 결과적으로, 게이머들은 NPC들의 단순한 이동 때문에 쉽게 게임을 정복할 수 있으며, 게임의 재미 또한 감소될 수 있다.

본 논문은 이를 개선하기 위하여 생물학적인 무리짓기 이론을 게임 NPC 인공 지능에 접목하기 위한 것이다. 이를 위하여 물고기 무리짓기 연구[3]를 설계하고, 오우거(Ogre) 엔진을 이용하여 구현한다. 그리고, 구현된 무리짓기와 실세계의 무리짓기의 패턴을 비교하여 실세계의 무리짓기 형태를 NPC들의 인공지능에 적용될 수 있음을 보여준다.

본 논문의 구성은 다음과 같다. 2장에서 물고기 무리짓기 연구[3]와 오우거 엔진을 간략하게 소개한다. 3장에서 물고기 무리짓기를 설계하고, 구현한다. 그리고 4장에서 결론 및 추후 연구 방향을 기술한다.

2. 관련 연구

2.1 오우거 엔진

오우거 엔진은 3D 그래픽 엔진으로 영국의 Stave ' sinbad' Streeting라는 소프트웨어 엔지니어가 2001년 개발하고, 2010년 4월에 1.7 버전까지 배포되었다.

오우거 엔진은 C++기반의 객체지향 인터페이스 방식의 설계로 간결한 클래스와 인터페이스를 제공하고, Direct 3D와 OpenGL을 동시에 지원한다.

오우거 3D 그래픽 엔진의 구조는 Root를 기준으로 장면관리자(Scene Manager), 랜더링(Rendering), 자원관리자(Resource Manager)의 3부분으로 나뉜다.

장면관리자는 화면에 보여지는 객체, 카메라, 광원, 평면과 같이 장면을 보여주기 위해 필요한 것들을 모두 관리한다. 자원관리자는 이미지, 객체에 입힐 메쉬, 하늘을 꾸미기 위한 메쉬, 음악과 같이 화면을 꾸며주는 자원들을 관리한다. 렌더링은 장면관리자와 자원관리자에 의해 설정된 객체의 내용을 화면에 출력해 주는 일을 한다.

2.2 무리짓기 이론

무리짓기는 1987년 Craig Reynolds가 발표한 논문에서 처음 소개된 기법으로 물고기 떼, 새 떼와 같이 무리의 집단 행동을 보이도록 유도하는 규칙이다[6,7].

Reynolds가 제안한 무리짓기 행동에는 충돌회피와 응집이 있다. 충돌 회피는 개체간 충돌을 피하기 위한 최소 거리를 유지하기 위한 규칙으로 충돌 회피 영역(δ)을 항상 모든 개체들이 최소로 유지해야 한다. 응집은 개체들이 모두 동일한 방향을 향하거나 모일 수 있는 규칙으로 상호 작용 영역(ρ)안에 있는 개체들간에 서로 방향과 위치 정보를 교환한다.

식(1)[4]은 충돌 회피를 위한 것으로 t시간에 개체 i의 충돌 회피 영역에 개체 j가 포함된 경우에 개체 i의 이동 방향을 나타낸다. $c_i(t)$와 $v_i(t)$는 각각 t시간의 개체 i의 위치 벡터와 방향 벡터를 각각 의미하며 d_i는 개체가 이동하려는 이동 방향을 나타낸다. 개체 i의 회피 영역에 있는 개체 j에 대하여 개체 i와의 거리에 대한 개체 j와의 벡터 요소의 차이를 모두 합한 값의 음의 방향, 즉 반대 방향으로 이동함을 표현한다.

$$d_i(t+\triangle t) = -\sum_{j \neq i} \frac{c_j(t) - c_i(t)}{|c_j(t) - c_i(t)|} \quad (1)$$

응집은 개체 i의 상호 작용 영역 ρ 에 개체 j가 위치했을 때 같은 방향으로 모이게 하는 것으로 식(2)는 이를 나타낸다. 첫 번째 항은 위치 벡터를 나타내고 두 번째 항은 방향 벡터를 나타낸다.

$$d_i(t+\triangle t) = 1/2[\sum_{j \neq i} \frac{c_j(t) - c_i(t)}{|c_j(t) - c_{i(t)}|} + \sum_{j=1} \frac{v_j(t)}{|v_i(t)|}] \quad (2)$$

충돌 회피 영역 δ 와 상호 작용 영역 ρ 에 있지 않는 개체들은 각 개체가 진행하는 방향대로 이동한다. 식(3)은 이동하려는 방향 벡터가 이전 방향 벡터 값을 그대로 반영함을 보여준다.

$$d_i(t+\triangle t) = v_i(t) \quad (3)$$

식(4)는 식(1)과 식(2)와 식(3)의 경우중에서 개체의 방향 벡터를 구하고 새로운 위치 벡터를 구하는 것을 보여준다. 첫 번째 항은 위치 벡터를 나타내고 2번째 항은 식(1), 식(2), 그리고 식(3)을 이용하여 계산된 방향 벡터와 일정한 시간($\triangle t$), 그리고 이동 속도인 s를 곱한 것이다.

$$c_i(t+\triangle t) = c_i(t) + v_i(t+\triangle t)\triangle t^* s \quad (4)$$

3. 무리짓기 설계및 구현

3.1 물고기 무리짓기의 설계

표 1은 2장에서 언급된 실세계의 물고기 무리짓기 형태[3]과 기존 연구들[8,9]를 바탕으로 물고기 무리짓기를 의사 코드(pseudo code)로 설계한 것이다. i는 현재 위치 벡터와 방향 벡터를 계산하기 위한 물고기 번호이고, j는 참조되는 물고기 번호이다. N은 총 물고기 마리수이다. 단계 (5)에서는 물고기 i의 충돌 회피 영역안에 물고기 j가 있는지 판단하여 이 값이 참인 경우에, 단계 (6)에서 식(1)을 이용하여 방향 벡터를 구한다. 단계 (7)에서는 물고기의 상호 작용 영역안에 물고기 j가 있는지 판단하여 이 값이 참인 경우에, 단계 (8)에서 식(2)을 이용하여 방향 벡터를 구한다. 단계 (9)에서는 단계 (5)와 단계 (7)에서의 이 두 조건을 만족하지 못한 물고기의 경우 단계 (10)에서 식(3)을 이용하여 위치 벡터를 구한다. 단계 (11)에서는 식(4)을 이용하여 물고기 i의 새로운 위치 벡터를 구한다.

[표 1] 물고기 무리짓기 설계

```
(1) for(i=1;i<=N;i++)
(2)   generate position vector, direction vector
(3)   for(i=1;i<=N;i++)
(4)     for(j=1;j<=N;j++){
(5)       if  j in avoidance range of fish(i) collision
(6)          compute di using equation(1)
(7)       else if j in interaction range of fish(i)
(8)          compute di using equation(2)
(9)       else
(10)        compute di using equation(3)
(11)    compute position vector(equation(4))
(12) }
```

3.2 물고기 무리짓기 구현

표 2는 FrameListener 클래스를 상속받은 MainListener의 일부를 보여준다. FrameListener 클래스는 프레임 단위로 처리해야 할 일들을 기술하는 클래스로 이 클래스를 상속한 클래스를 오우거 엔진에 등록하면 자동적으로 프레임이 시작되기 전에는 frameStarted() 함수를 호출하고 렌더링이 된 후에는 frameEnded() 함수를 호출한다. 루트 객체(단계 (1))와 윈도우(단계 (2))와 장면관리자(단계 (3)), 카메라(단계 (4))를 각각 생성한다. 이후에 카메라의 위치(단계 (5))와 시점(단계 (6))을 설정하고 단계 (4)에서 생성된 카메라를 갖는 뷰포트를 만든다(단계 (7)). 그리고 주변광원을 설정한다(단계 (8)). 단계 (9)부터 단계 (13)까지는 물고기 개체를 만드는 과정이다. 단계 (10)에서는 물고기 메쉬(f.mesh)를 로드하여 엔티티를 생성하고 단계 (11)에서는 위치 벡터를 생성하기 위하여 어항의 크기인 200보다 작은 숫자를 랜덤하게 생성하는 과정이다. 현재 구현되는 프로그램은 x, z 평면에 물고기가 그려진다. 단계 (12) 에서는 장면 노드인 fish를 생성하고 이 노드의 위치를 단계 (11)에서 생성된 좌표로 설정한다. 단계 (13)에서는 단계 (10)에서 생성된 fishEnt 엔티티를 장면 노드인 fish에 연결시킨다.

[표 2] MainListener 클래스 생성자

```
class MainListener : public FrameListener{
 public:
 MainListener(){
(1)    mRoot=new Root();
(2)    mWindow=mRoot->getAutoCreatedWindow();
(3)    mSceneMgr=mRoot->createSceneManager();
(4)    mCamera = mSceneMgr->createCamera("cam");
(5)    mCamera->setPosition();
(6)    mCamera->lookAt();
(7)    mViewport =
     mWindow->addViewport(mCamera);
(8)    mSceneMgr->setAmbientLight();
(9)    for(i=0; i<N<i++)){
(10)    fishEnt[i]=mSceneMgr->createEntity(n,"f.mesh");
(11)    x= rand()%200, z= rand()%200;
(12)
      fish[i]=mSceneMgr->getRootSceneNode()->create
        ChildSceneNode(String(name)",Vector3(x,0,z);
(13)    fish[i]->attachObject(fishEn[i]);
  }
 }
}
```

표 3은 MainListener 클래스의 framestarted() 메소드의 일부분을 보여주는데 이는 [표 1]의 단계 (3)부터 단계 (11)까지의 과정을 구현한 것이다. 단계 (10)은 물고기의 새로운 위치를 설정하는 부분이다. 해당 프레임이 렌더링된 후에 frameEnded() 메소드에서는 별도의 일을 처리하지 않고 항상 true를 반환한다.

[표 3] Main Listener 클래스의 frameStarted()

```
(1) bool frameStarted(const FrameEvent &evt){
(2)   for(i=0; i<N;i++)
(3)    for(j=0;j<N;j++) {
(4)      if (i==j) continue;
(5)      if (collision_range(i,j))
(6)         d[i]=equation(1);
(7)      else if (interaction_range(i,j))
(8)         d[i]=equation(2);
(9)      else d[i]=equation(3);
(10)      fishNode[i]->setPosition(fish[i]->getPosition()
(11)                                    +d[i]*t*s);
(12)    }
(13) }
(14)
(15)  bool frameEnded(const FrameEvent &evt){
(16)     return true;
(17)  }
```

표 4는 초기화 과정과 렌더링을 기술하는 game 클래스의 일부를 보여준다. 단계 (2)에서는 루트 객체인 mRoot를 생성하고 단계 (4)에서는 윈도우를 생성하고 단계 (6)에서는 리소스 그룹 관리자를 초기화한다. 단계 (8)에서는 mMainListener 객체를 생성하고, mRoot 객체의 프레임 리스너로 mMainListener를 등록한다(단계 (9)). go() 함수에서는 startRendering()함수를 호출하여 렌더링을 지속한다(단계 (13)).

[표 4] game 클래스

```
class game {
(1)    game() {
(2)      mRoot =  new Root("plugins.cfg","ogre.cfg",
(3)                                  "log.txt");
(4)      mWindow = mRoot->initialise(true, "init");
(5)
(6)    //리소스 그룹 관리자 초기화
(7)
(8)      mMainListener = new MainListener(mRoot);
(9)      mRoot->addFrameListener(mMainListener);
(10) }
(11)
(12) void go() {
(13)    mRoot->startRendering();
(14) }
```

표 5는 main.cpp의 일부를 보여준다. 게임 클래스 객체인 app를 생성하고(단계 (1)), 단계 (3)에서 게임 루프에 진입한다. 그리고, 예외가 발생한 경우에 예외처리를 수행한다(단계 (4)).

[표 5] main() 함수

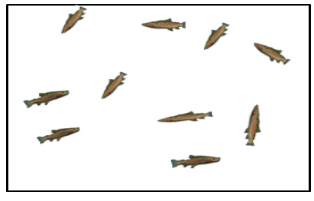
```
int main(int argc, char *argv[])    {
(1)    game app;
(2)    try {
(3)       app.go();
(4)    } catch( Ogre::Exception& e ) {
(5)          MessageBox( NULL, "exception has
    occurred!");
(6)    }
(7)    return 0;
}
```

3.3 성능 평가

물고기 무리짓기를 구현하기 하기 위해 사용된 오우거 SDK 버전은 1.4.6이고 컴파일러는 Visual C++ express 2005이다. 실험에 사용된 컴퓨터 CPU는 Intel core E7200으로 2.53GHz이고 RAM은 2GB, 그래픽 메모리는 512MB이고 운영체제는 Microsoft XP Pro이다.

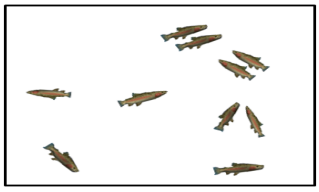
실험에 사용된 조건은 D.J. HOARE et al.[3]에서의 무리짓기 환경과 동일하다. 구현시 사용된 물고기의 수는 10마리, 어항의 크기는 정사각형 크기로 가로 200, 세로 200의 크기를 갖는다. 응답 대기 시간 $\triangle t$를 0.1로 설정하고 물고기의 이동 속도 s는 1.25, 물고기의 크기는 4cm로 설정한다. 충돌 회피 영역 δ 은 1BL(Body Length)로 설정하는데, 1BL은 4cm의 값을 갖는다. 상호 작용 영역 ρ 는 2BL로 설정한다. 물고기의 초기 위치와 방향은 랜덤 함수를 이용하여 생성된다.

그림 2는 시뮬레이션 시작시의 화면으로 물고기의 위치와, 방향이 모두 랜덤하게 주어진 상태로 응집과 정렬이 되어 있지 않음을 확인 할 수 있다.



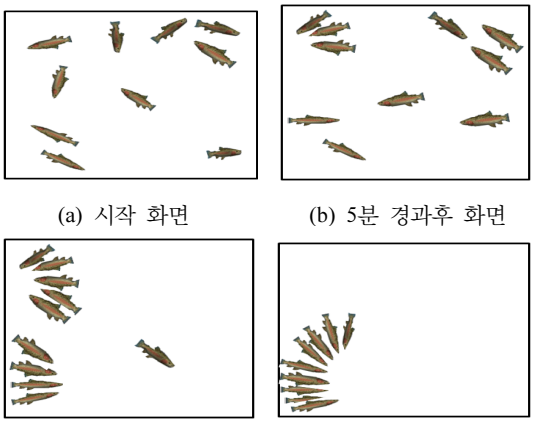
[그림 2] 시작 화면

그림 3은 부분 정렬 및 부분 응집 상태를 보여주는 것으로 오른쪽 상단 부분에 물고기들이 2마리씩 무리를 짓거나 머리 부분이 동일한 지점으로 정렬되는 모습을 볼 수 있다.



[그림 3] 부분 정렬및 응집

그림 4는 물고기 10마리가 무리를 형성하는 과정을 4단계로 나누어 놓은 것이다. 그림 4(a)는 시작 후 30초가 경과된 상태의 화면으로 혼자 움직이는 물고기들이 많다. 그림 4(b)는 시뮬레이션 실행 후 5분 후의 화면으로 2~3마리들이 무리를 짓거나 정렬하고 있다. 그림 4(c)는 10분 후의 화면으로 10마리의 물고기가 일정한 크기의 무리를 형성하고 있다. 이후부터는 물고기들이 일정 크기의 무리를 형성하여서 보다 큰 무리를 형성하는데에 많은 시간이 소요되었다. 그림 4(d)는 물고기 10마리가 모두 무리를 형성했을 때의 모습이다.

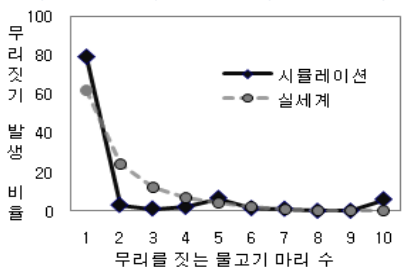


(a) 시작 화면 (b) 5분 경과후 화면

(c) 10분 경과후 상태 (d) 40분 경과 상태후

[그림 4] 무리짓기 상태 변화

그림 5는 실세계의 무리짓기 실험[3]과 오우거 엔진을 이용하여 무리짓기를 구현한 비교 결과이다. 두 실험 모두 30초 간격으로 60분간 관찰하며 실험 환경은 그림 3과 그림 4에서와 동일하게 설정한다. 가로 축은 무리짓기를 형성하는 물고기들의 마리 수이고, 세로 축은 각 마리로 구성되는 무리짓기 횟수를 모두 합한 값에 대해 해당 마리로 무리를 짓는 횟수를 백분율로 표현한 것이다. 시뮬레이션의 경우 전체 실험 시간 동안 1마리로 무리짓는 비율이 전체 무리짓기 비율의 약 80%를 차지한다. 두 경우 모두 60% 이상의 시간을 무리를 짓지 않고 1마리씩 독자적으로 움직이고 있다. 두 경우 모두 시간이 경과될수록 물고기들의 응집과 정렬로 인해 물고기 무리의 크기가 커짐을 알 수 있다. 그러나, 시뮬레이션의 경우에서 10마리 모두 무리로 되는 형성되는 비율은 약 43분 정도에 10마리가 모두 응집되어서 6마리에서 9마리가 무리로 형성되는 비율에 비해 상대적으로 높고, 실세계에서 10마리가 모두 무리로 형성되는 비율에 비해 높다. 이러한 이유는 시뮬레이션의 경우 실세계에서와 어항의 크기와 유사한 화면의 크기 설정과 벽면에 부딪혔을 경우에 부자연스럽게 구현되는 것등의 한계를 가지고 있기 때문이라고 판단된다. 이 경우를 제외하면 실세계에서의 무리짓기와 시뮬레이션의 무리짓기 형태는 유사함을 보인다. 이와 같이 실세계와 시뮬레이션의 무리짓기 비율은 대부분의 경우에 무리짓기 변화 정도가 비슷하며 변화 형태 또한 유사하다. 이는 구현된 무리짓기를 게임의 무리짓기에 응용하게 되는 경우 실세계에서와 비슷한 형태의 NPC 무리짓기를 게이머에게 제공할 수 있음을 보여준다.



[그림 5] 시뮬레이션과 실세계의 무리짓기 비교

4. 결론 및 향후 연구 방향

본 논문에서는 물고기의 무리짓기 행동을 설계하고 오우거 엔진을 이용하여 구현하였다. 그리고 구현된 결과를 실세계의 무리짓기와 비교하였다. 실험 결과, 실세계의 무리짓기와 시뮬레이션의 무리짓기는 유사함을 보였다. 본 논문의 무리짓기 알고리즘을 게임의 NPC 무리짓기에 적용하는 경우 게이머들에게 실세계와 비슷한 무리짓기 환경을 제공하여 게임의 몰입을 증가시킬 수 있을 것으로 판단된다.

추후에는 BL 크기와 어항 크기, 그리고 물고기 수 등의 다양한 조건에서 구현된 무리짓기 알고리즘에 대한

성능을 평가할 예정이다.

참고문헌

[1] PETER H.WERGE, MARTIN WIKESKI, et al., "ANTIBIRDS PARASITIZE, FORAGING ARMY ANTS", Ecology, 86(3), pp.555-559, 2005.

[2] I.D.Couzin and N.R.Franks,"Self-Organized lane formation and optimized traffic flow in army ants", Proceedings of the Royal Society of London, Series B. 270: 139-146, 2003.

[3] D.J.HOARE, I.D. COUZIN, J.-G.J. GODIN& J. KRAUSE, "Context-dependent group size choice in fish", Elsevier Ltd. ANIMAL BEHAVIOUR, pp.155-164, 2004.

[4] Iain D. Couzin1,2, Jens Krause, et al., “Effective leadership and decision-making in animal groups on the move”, Nature 433, pp.513-516, February 2005.

[5] Alt, G., and King, K, "Intelligent Movement Animation for NPCs", AI Game Programming Wisdom 2, Charles River Media, 2003.

[6] Steven Woodcock, “플로킹: 집단 행동을 흉내내는 간단한 기법”, Game Programming Gems 1, pp.401-415, 2001.

[7] Steven Woodcock, “먹고 먹히는 플로킹: 포식자와 먹이”, Game Programming Gems 2,이 pp.423-430, 2002.

[8] 유현지, 이면재, “Ogre 엔진을 이용한 물고기 떼 시뮬레이션”, 한국산학기술학회추계학술발표논문집, 제10권, 제2호, pp782-784, 2009.12.

[9] 유현지, 박종호, 이면재, “물고기 무리짓기 구현”, 한국산학기술학회 춘계학술발표논문집 제11권, 제1호, pp1010-1013, 2010.5.

유 현 지(Hyun-Ji Yoo) **[준회원]**

- 2011년 2월 : 백석대학교 정보통신학부 멀티미디어 전공 졸업 예정

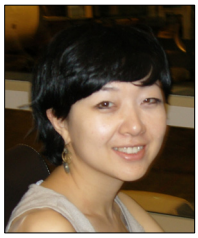
<관심분야>
게임 인공지능, 안드로이드 프로그래밍

이 면 재(Myoun-Jae Lee) **[종신회원]**

- 1992년 2월 : 홍익대학교 전자계산학과 졸업 (이학사)
- 1994년 2월 : 홍익대학교 전자계산학과 대학원 졸업(이학 석사)
- 2009년 3월 ~ 현재 : 백석대학교 정보통신학부 교수

<관심분야>
게임 인공지능, 게임 엔진

김 경 남(Kyoung-Nam Kim) **[정회원]**



- 1994년 2월 : 홍익대학교 미술대학 회화과 졸업(학사)
- 1997년 2월 : 홍익대학교 미술대학 회화과 졸업(석사)
- 2008년 ~ 현재 : 중앙대학교 첨단영상대학원 예술공학 박사과정 수료

<관심분야>
미디어 아트, 게임 아트