Journal of Korea Game Society 2014 Oct; 14(5): 35-42
http://dx.doi.org/10.7583/JKGS.2014.14.5.35

게임 콘텐츠, 산업, 마케팅

대규모 온라인 FPS 게임을 위한 효율적인 캐릭터 방향 갱신 기법

임종민, 이동우, 김영식
한국산업기술대학교 게임공학과
{jjonga8, ldwnim}@naver.com, kys@kpu.ac.kr

An Efficient Method to Update Character Moving Directions for Massively Multi-player Online FPS Games

Jong-Min Lim, Dong-Woo Lee, Youngsik Kim
Dept. of Game and Multimedia Engineering, Korea Polytechnic University

요 약

최근 First Person Shooter (FPS) 게임 시장에서 '플래닛 사이드 2' 를 비롯한 대규모 온라인 FPS (MMOFPS: Massively Multi-player Online FPS) 장르가 주목받고 있다. 수백 또는 수천 명이 접속하는 게임 서버에서 네트워크 부하를 경감시키기 위해서 널리 사용되는 방법 중 하나는 데드레커닝이다. 본 논문에서는 MMOFPS 게임을 위한 데드레커닝 구현 시 주요 캐릭터 상태 갱신 변수 중 하나인 방향에 대하여 효율적으로 허용 임계각을 계산하는 수학적 방법을 제안한다. 제안하는 방법은 게임사용자들을 대상으로 하는 수행 실험을 통해 움직임 오차를 최소화하며 자유스러운 방향 갱신에 대한 효율성을 검증 하였다.

ABSTRACT

In the market of First Person Shooter (FPS) games, Massively Multi-player Online FPS games (MMOFPS) like 'PlanetSide 2' have been popular recently. Dead reckoning has been widely used in order to mitigate the network traffic overload for the game server with hundreds or thousands of people. This paper proposes the efficient analytical method to calculate the tolerable threshold angle of moving direction, which is one of the most important factors for character status updating when dead reckoning is used in MMOFPS games. The experimental results with game testers shows that the proposed method minimizes the position error for character moving and provides natural direction updates of characters.

Keywords : Massively Multi-player Online First Person Shooter(MMOFPS: 대규모 온라인 FPS), Dead Reckoning(데드레커닝), Moving Direction Updating(방향 갱신)

Received: Aug. 07, 2014 Accepted: Sept. 11, 2014
Corresponding Author: Youngsik Kim (Korea Polytechnic Univ.)
E-mail: kys@kpu.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

최근 동시접속이 2000명까지 가능한 대규모 온라인 FPS (MMOFPS)게임 '플래닛사이드'2 가 국내에 출시하였다. 이 게임은 피시방 게임 이용 순위 28위, FPS 부문 6위를 기록하여 주목받고 있다[1]. MMOFPS는 MMORPG (Massively Multi-player Online Role Playing Game)와 FPS 성격을 동시에 갖춘 장르로서 몇 백, 몇 천 명의 사람들이 동시 접속하여 FPS 게임을 즐기는 것이 특징이다. 이전에 red5studio가 FireFall을 출시하면서 MMOFPS라는 새로운 장르의 게임을 개척하였고 이후 차세대 FPS 게임 장르로 주목받고 있다[2,3,4,5,6].

[Fig. 1] Screen shot of 'Planet Side2'

대규모 온라인 게임에서 하나의 서버는 수백 또는 수천 명의 클라이언트들을 처리하는데 클라이언트들이 객체의 상태 갱신 패킷을 자주 보내게 된다면 서버에 과다한 네트워크 부하가 걸리게 된다. 따라서 클라이언트들과 서버는 일정한 상태 갱신 주기에 맞춰 객체의 상태 갱신 패킷을 상대에게 보내며 주기 사이에는 이전에 받았던 정보들로 상대방의 위치를 예측하고 임계 값 이상으로 상태가 변했을 시에만 상태 갱신 패킷을 보낸다.

FPS게임은 상대 캐릭터의 움직임에 대하여 즉각적으로 반응하여야 하므로 다른 게임들보다 상대 캐릭터에 대한 묘사가 더 세밀하게 이루어져야 한다. 이를 위해서는 임계값을 낮추고 갱신 주기를 짧게 두어 캐릭터의 상태를 자주 갱신하여야 한다. 하지만 대규모 온라인 FPS 게임과 같은 경우, 짧은 주기로 갱신하면 패킷의 양이 늘어나 서버의 부하가 커지게 된다. 따라서 부하를 줄이면서 세밀한 표현도 가능한 효율적인 상태 갱신이 필요하다. 이에 본 논문에서는 온라인 FPS 게임을 위한 데드레커닝 구현 시 캐릭터 상태 갱신 주요 변수들을 살펴보고 변수 중 하나인 방향에 관하여 효율적인 갱신을 하기 위한 수학적 방법을 제안하고 게임사용자들을 대상으로 실험하여 그 효율성을 검증한다.

2. 연구 배경

2.1 데드레커닝

게임 개발에서의 데드 레커닝은 시간 지연으로 인하여 사용자들끼리의 화면 불일치로 생겨난 오차가 게임을 진행하기 힘들 정도로 발전하는 상황을 말하는 아주 좁은 뜻의 단어였지만 지금은 그것을 없애기 위한 동기화 기법 및 예측 알고리즘을 뜻한다[7]. 시간지연이란 패킷이 클라이언트에서 서버를 거쳐서 다시 클라이언트까지 오는데 걸리는 시간이다. 시간지연으로 인하여 사용자들은 각각 다른 화면을 보게 되며 이는 게임의 공평성을 흩트리게 된다. 개발자는 사용자들이 시간지연으로 인한 오차를 못 느끼게 하고 게임의 공평성도 유지하기 위하여 데드레커닝을 사용한다. 대규모 온라인 게임에서 하나의 서버는 수백 또는 수천 명의 클라이언트들을 처리하는데 클라이언트들이 객체의 상태 갱신 패킷을 자주 보내게 된다면 서버에 과다한 네트워크 부하가 걸리게 된다. 따라서 대규모 온라인 게임의 경우, 시간지연해결을 포함하여 패킷 양을 줄이기 위해 쓰기도 한다[8].

상태를 갱신하는 데에는 세 가지 방법이 있다. 첫 번째는 클라이언트와 서버가 서로 상태 변수들을 주기적으로 전송하는 방법이며 두 번째는 상태

가 일정값 이상으로 변하였을 때만 상태 갱신 패킷을 전송하는 방법이다. 세 번째는 일정한 주기로 갱신 패킷을 전송하되 즉각적인 반응을 위해 상태가 일정값 이상으로 변했을 때도 보내는 방법이다. 세 방법 모두 갱신 패킷을 못 받았을 경우 마지막으로 받은 상태 정보들을 토대로 예측 알고리즘에 따라 게임을 진행한다[9,10].

FPS의 경우 정확한 위치와 즉각적인 반응이 중요하므로 세 번째 방법이 적합하며 본 논문은 세 번째 방법을 기반으로 제안한다.

2.2 위치예측

클라이언트가 상대방 캐릭터 위치를 예측하는 방법 중 하나는 마지막으로 수신한 위치, 속도, 가속도 정보를 [Table 1]과 같은 물리공식에 대입하는 것이다.

[Table 1] Position Estimation Equations[11]

	X: Position V: Velocity a: Acceleration t: time t0: initial time
Position	X(t) = X(t0)
Velocity	X(t) = X(t0) + V(t0) * (t - t0) V(t) = (X(t) - X(t0)) / (t - t0)
Acceleration	X(t) = X(t0) + V(t0) * (t - t0) + 1/2*a(t0)*(t-t0)^2 a(t) = (V(t) - V(t0)) / (t - t0)

[Table 1]의 첫 번째 식은 새로운 위치 예측에 기존의 위치정보만을 이용하는 것이며 두 번째 식은 위치와 속도를, 세 번째 식은 위치, 속도, 가속도를 이용하는 것이다[11,12]. FPS게임에서는 사용자가 일정한 속도로 걷거나 뛴다. 또한 방향키를 떼었을 때도 바로 멈춘다. 이렇게 등속도로 움직이므로 가속도가 필요 없어 두 번째 식을 이용하는 것이 좋다.

3. 캐릭터 방향 상태 갱신

3.1 온라인 FPS 게임 상태 갱신 주요변수

FPS 게임에서 클라이언트가 상대방의 위치를 예측하는 데에 필요한 주요 변수는 동작, 방향, 방향키 3가지다. 동작은 걷기, 뛰기, 앉아서 걷기 등 일련의 행동 상태를 의미한다. 방향은 상대방 캐릭터가 보고 있는 시선 방향 벡터를 의미하며 사격 시에는 총알 궤적의 방향이 되기도 한다. 방향키는 상대 사용자가 누른 방향키로서 ↑↓(상,하) 키는 전진과 후진, ←,→(좌,우) 키는 좌우 이동 여부를 의미한다. 클라이언트는 상대방의 동작을 통해 이동 속도를, 방향과 방향키를 통해 이동 방향을, 방향키를 통하여 이동 여부를 유추해 낼 수 있으며 여기에 마지막으로 수신한 위치를 더하여 상대방의 위치를 예측한다. 따라서 사용자의 조작으로 이 변수들의 값이 바뀐다면 클라이언트는 이 변수들에 대한 상태 갱신 패킷을 서버에 보내야 한다.

동작은 걷기, 혹은 뛰기를 하는 경우가 대부분이며 앉아서 걷거나 점프하여 가는 경우는 가끔 일어난다. 상대적으로 방향키, 방향에 비해 동작 상태의 변화는 적다. 사용자는 방향키를 조작하여 캐릭터의 이동방향을 결정할 수도 있는데 이때 사용자의 방향은 바뀌지 않는다. 사용자가 ↑ 버튼을 눌러 전진하는 이동방향을 0^o라 두고 그것을 기준으로 시계방향을 +방향으로 정한다면, 방향키 입력에 따른 이동방향은 아래 [Table 2]와 같다.

[Table 2] Direction with Arrow Keys

Arrow Keys	Direction
↑	0^o
↑,→	45^o
→	90^o
→,↓	135^o
↓	180^o
↓←	225^o
←	270^o
↑←	315^o

방향키로는 45^o 간격으로만 이동방향을 결정할 수 있다. 이동방향을 세밀하게 정할 수가 없으므로 사용자들이 전진과 후진의 여부를 결정하는 용도로 많이 쓰므로 방향키의 상태 변화는 적다고 볼 수 있다. 그에 비해 방향은 아주 빈번하게 변화한다. 사용자는 적들의 출현을 대비하여 주변을 살피기도 하고 이동 중에는 마우스를 통해 캐릭터를 회전시켜 이동방향을 결정한다. 적이 출현하였을 때에는 적을 맞추기 위해 시선 방향을 회전시켜 사격방향을 정한다. 방향키와 동작 상태에 비해 사용자의 조작 빈도가 높다. 따라서 상태 갱신이 빈번하게 일어나며 갱신 패킷 량에도 큰 영향을 미친다.

3.2 임계각과 위치 오차

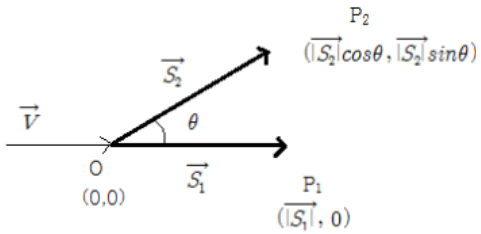
캐릭터가 회전을 할 때마다 서버에 방향 갱신 패킷을 보낸다면 회전 빈도가 잦은 FPS게임에서 패킷량이 많이 늘어나게 되며 서버에 부하가 커지게 된다. 따라서 캐릭터가 일정한 각도 이상으로 회전할 때 방향 갱신 패킷을 보내야 한다. 이 각도를 임계각이라고 하자. FPS 게임에서는 캐릭터의 x축과 z축에 대한 회전이 잘 일어나지 않으므로 이 논문에서는 y축에 대한 임계각만을 다룰 것이다.

임계각을 설정하게 되면 캐릭터가 이동 중 임계각보다 적은 각도로 회전하였을 경우 갱신패킷을 보내지 않는다. 위 [Table 1]에서 언급한 두 번째 식을 이용해 예측한다면 실제 위치와 예측하는 위치가 차이가 나게 된다. 임계각을 작게 설정하면 오차가 줄어들지만 자주 갱신하게 되어 패킷량이 늘어나게 된다. 반대로 크게 설정하면 오차가 커지게 된다. 따라서 패킷량을 최대한 줄이면서 사용자가 오차를 인식하지 않는 적당한 임계각을 찾는 것이 중요하다.

임계각보다 미세하게 작아 방향이 갱신되지 않고 오차가 나는 최대 각도 크기를 최대 오차각이라고 하자. 이 때 최대 오차각에 대한 정의는 다음과 같다.

$$Max\, x \;where\; x < \theta_{threshold} \text{ and } \lim_{x \to \theta} x = \theta_{threshold} \qquad \text{(eq. 1)}$$

어떤 캐릭터가 일정한 속도로 달리고 있을 때 한 주기 동안 실제 캐릭터 위치와 클라이어트들이 예측하는 캐릭터 위치의 최대 위치 차이는 다음과 같다.



[Fig. 2] Maximum Value between Real Position and Estimated Position for Character

θ는 최대 오차각이며 점 O는 주기가 시작한 시점에서의 캐릭터의 위치이다. 캐릭터는 이전 주기에서 $\overrightarrow{V}$ 방향으로 달려오고 있었다. 이 지점에서 캐릭터는 최대 오차각 θ 만큼 회전하여 달린다. P_2는 주기가 끝날 때 즈음의 캐릭터 위치이고 P_1은 예측 위치이다. S_1과 S_2는 각각 P_1, P_2에 대한 위치 벡터이다. 점 O를 (0,0)로 두었을 때 P_1과 P_2의 좌표는 각각 $(|\overrightarrow{S_1}|, 0)$, $(|\overrightarrow{S_2}|cos\theta, |\overrightarrow{S_2}|sin\theta)$ 이다. 속도 v로 한 주기 t동안 캐릭터가 움직일 수 있는 거리 s = v * t이므로 한 주기 동안 캐릭터가 움직이는 거리 s 는 (eq. 2)와 같다.

$$s = |\overrightarrow{S_1}| = |\overrightarrow{S_2}| \qquad \text{(eq. 2)}$$

최대 위치 오차 d는 (eq. 3)과 같다.

$$\begin{aligned} d &= \sqrt{(|\overrightarrow{S_2}|\cos\theta - |\overrightarrow{S_1}|)^2 + (|\overrightarrow{S_2}|\sin\theta)^2} \\ &= \sqrt{s^2\cos^2\theta - 2s^2\cos\theta + s^2 + s^2\sin^2\theta} \\ &= \sqrt{2}\, s \sqrt{1-\cos\theta} \\ &= \sqrt{2}\, vt \sqrt{1-\cos\theta} \end{aligned} \qquad \text{(eq. 3)}$$

상태 갱신 주기 t가 길수록, θ가 커질수록 최대 위치 오차가 커진다. 따라서 임계각이 커짐에 따라 최대오차각도 커지고 그에 따라 최대 위치 오차도 커지게 된다.

4. 수치적 방법 제안

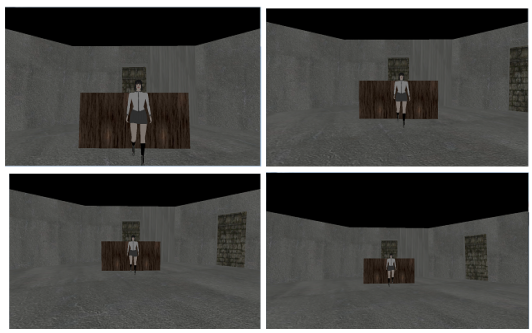
4.1 실험을 통한 허용오차 계산

실제 캐릭터의 위치와 예측하는 캐릭터의 위치 차이가 크면 충돌체크 시 불합리한 결과가 나올 수 있으며 이에 따라 게임의 공평성이 무너진다. 위치 차이는 사용자가 인식하지 못 할 만큼 나야 한다. 사용자가 위치 차이를 인식하기 시작하는 경계부분의 오차를 허용 위치 오차라고 하자. 본 논문에서는 허용위치 오차는 개인마다 다르므로 실제 게임 사용자들을 대상으로 허용 위치 오차 실험을 하였다. Y축은 중력과도 관계가 있어 실험변수가 늘어난다. 이 실험에서 다루기에는 실험범위가 크므로 Y축에 관해선 다루지 않는다.

실험을 위한 온라인 FPS 캐릭터는 [Fig. 3]과 같이 제작하였다. 캐릭터의 실제 위치와 예측 위치가 오차가 나는 것을 사용자들이 인식하는지 알아보기 위해 DirectX11으로 3D 여성캐릭터 애니메이션과 밀폐된 맵을 만들었으며 캐릭터 뒤에는 비교가 쉽게 일정한 거리에 1m 크기의 상자 2개를 놓는다. 실험지원자에게는 일정한 간격으로 떨어진 두 캐릭터를 일정한 짧은 주기로 번갈아가면서 보여준다. 두 캐릭터는 같은 종류의 메쉬를 가지고 있고 애니메이션 동작도 똑같으며 각각의 위치에서 제자리 걷기를 하는 상황이다. 실험지원자들에게는 한 캐릭터라고 말해준 후 카메라의 위치와 캐릭터들의 간격을 조정하면서 캐릭터의 움직임이 자연스러운지 물어본다.

[Fig. 3] FPS Characters used in Experiments

실험방법은 다음과 같다. 남녀 20명을 대상으로 2014년 7월 7일부터 7월 9일까지(3일간) 실험을 실시하였다. 실험지원자들은 20대~30대 후반의 대학생, 직장인들로서 실험참여자들의 평균연령은 26세였다. 실험지원자들의 구성은 남자가 15명이었으며 여자는 5명이었다. 실험은 노트북에서 가상환경을 보여주면서 하였다. 노트북의 사양은 intel core i7 2630QM CPU 와 을 775MHZ CLOCK 성능을 가진 NVIDIA GeforceGTX560 GPU를 가지고 실험하였다. 어플리케이션은 평균적으로 1초에 이미지들을 250번 영사한다.(250fps) 실험에서는 주기를 0.1초라고 두었으며 캐릭터들 간의 간격은 조금만 차이나도 확연하게 느껴지므로 0.5cm부터 3.0cm까지 0.5cm 간격으로 보여주었다. 카메라는 캐릭터 위치로부터 1m부터 4m까지 1m 간격의 조금 큰 단위로 떨어져서 보여주었다. 간격은 둘 다 실물크기를 가상하였을 때의 거리이다.



[Fig. 4] Distance between Character and Camera 1m (upper left), 2m (upper right), 3m (lower left), 4m (lower right)

각 조건에 대한 캐릭터의 움직임이 자연스러운지 실험 지원자들에게 물었으며 그에 대하여 상당히 자연스럽게 느낀다면 5점, 상당히 부자연스럽게 느낀다면 1점을 적도록 하였다. 그 외의 점수들은 [Table 3]과 같다.

[Table 3] Scores of Natural Feeling for Character Moving

Natural Feeling	Scores
Very Natural	5
Natural	4
Normal	3
Unnatural	2
Very Unnatural	1

각 조건에 대하여 20명에게 점수를 매기게 하였고 실험지원자들이 체크한 점수들을 합산하여 낸 평균은 아래와 같다.

[Table 4] Average Scores for Experimental Tolerable Position Errors

A \ B	0.5cm	1.0cm	1.5cm	2.0cm	2.5cm	3.0cm
1m	1.75	1.35	1.10	1.05	1.00	1.00
2m	2.70	2.25	2.10	1.55	1.45	1.00
3m	4.05	3.50	3.15	3.05	2.45	2.10
4m	4.20	3.90	3.75	3.45	3.00	2.70

A : Distance(m) between Character and Camera
B : Distance(cm) between Two Characters

'Normal' 의견인 3점을 기준으로 평균이 3점미만이면 실험지원자들이 대체로 부자연스럽다고 느꼈다고 할 수 있으며 이는 위치 오차에 대하여 인식하고 있다고 할 수 있다. 3점 이상인 경우는 대체로 자연스럽다고 느꼈다고 할 수 있으며 이는 위치 오차에 대하여 인식을 하지 못한 것이라고 할 수 있다. 카메라와 캐릭터들 간 간격이 4m일 경우, 오차가 3.0cm 일 때 평균이 2.7로 실험지원자들이 대체로 위치 오차를 인식하고 있지만 바로 다음 간격인 2.5cm일 때 평균이 3.0으로 위치오차에 대하여 인식을 하지 못하는 경계가 되므로 3.0이 허용 위치오차라고 불 수 있다. 간격이 3m일 경우 2.0cm가 허용 위치라고 할 수 있으며 2m와 1m의 경우 알 수 없다.

4.2 허용 임계각을 계산하는 효율적인 수치적 방법 제안

허용 위치 오차가 정해진다면 이를 통해 효율적인 임계각을 예상할 수 있다. 위의 식 (eq. 3)에서 최대 위치오차 d를 허용위치오차로 둔다면 효율적인 임계각 θ는 (eq. 4)와 같은 수치적 방법으로 계산할 수 있다.

$$\theta = \cos^{-1}(1 - 1/2(d/vt)^2) \qquad \text{(eq. 4)}$$

보통의 인간은 달릴 때 평균적으로 시속 20km/h로 달린다. 이것을 초 단위로 바꾸면 5.5m/s이 된다. 속도 v를 5.5m/s로 두고 주기 t를 0.05초(50ms)로 두었을 때, 허용위치오차에 따른 효율적인 임계각은 아래와 같다.

[Table 5] Threshold Angles for Tolerable Position Errors

Tolerable Position Error (cm)	Threshold Angle (°)
2.0	4~5
2.5	5~6
3.0	6~7
3.5	7~8
4.0	8~9

거리가 멀어질수록 허용 위치오차는 늘어나며 그에 따라 임계각도 크게 늘릴 수 있다. 이는 패킷량을 줄일 수 있음을 의미한다. 따라서 캐릭터들 간의 거리에 따라서 임계각을 조절한다면 패킷량을 줄이면서 위치 오차를 줄일 수 잇는 효율적인 방향 갱신을 설정할 수 있다.

허용 위치오차는 캐릭터의 크기, 맵의 특성, 조명 등에 따라 차이가 난다. 따라서 개발자는 허용

위치오차 실험을 통해 그 게임에 맞는 효율적인 임계각을 구해야 한다.

5. 결론 및 향후 연구과제

본 논문에서는 대규모 온라인 FPS (MMOFPS) 게임에서 데드레커닝 구현 시 갱신하는 주요 변수들을 살펴보았다. 그 중 방향은 빈번하게 갱신이 일어나는 변수이며 효율적으로 갱신할 경우 서버에 부하를 줄일 수 있다. 이에 따라 임계각 설정이 필요하며 설정에 따른 예측 위치와 실제 위치와의 위치오차, 속도, 갱신 주기와의 상관관계를 살펴보았다. 이에 근거하여 효율적인 방향 갱신 방법도 제안하였다.

데드레커닝 구현 시 시간 지연뿐만 아니라 다양한 변수들을 고려해야 하며 허용 위치 오차의 기준은 개개인마다 다른 상대적인 개념이므로 이 연구내용이 절대적으로 쓰이기는 어렵다.

향후 연구 과제는 보다 더 다양한 변수들을 고려하여 신뢰도를 높이면서 임계각 설정에 따른 평균 패킷 갱신율의 변화를 살펴보는 것이며 제안한 것을 실제 게임에 적용하여 효율성 증진을 실증하는 것이다.

REFERENCES

[1] “Daum PlanetSide2, MMOFPS Myth”, Internet News Herald Economics, 2014.7.7)

[2] “Massively multiplayer online game”, wikipedia, http://www.wikipedia.org

[3] FireFall, http://www.firefall.co.kr

[4] Daum PlanetSide2, http://ps2.daum.net

[5] Sang-Jung Kim, “A Study on the Correlation about Creating Component of Pleasure and Satisfaction on FPS Game,” Journal of Korea Game Society, Vol. 9, No. 6, pp.45~56, 2009

[6] Ji-Hun Lee, “A Study on Factors to Affect Reuse Intention and Conversion Intention by Evaluation after Game Use - with Two FPS Games as a Main Consideration -,” Journal of Korea Game Society, Vol. 9, No. 6, pp.69~78, 2009

[7] Seong-Rak Kim, Nam-Kyun Yun, Yong-Wan Koo, “Design and Implementation of Dead Reckoning Algorithm for Network Game,” Korea Information Processing Society Vol 7 No. 8, 2005.

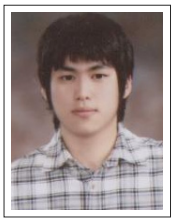
[8] Kyung-Chul Kim, Online Game Server, EGO, pp141-158 , 2012.

[9] Eric Lengyel, “Believable Dead Reckoning for Networked Games,” Game Engine Gems2 pp307-327, A K Peters, 2011.

[10] Kwang-Hyun Shim, Jong-sung Kim, “A Study On Performance Analysis and Improvement of Dead-Reckoning Algorithm in Networked Virtual Environment,” Korea ETRI, 2001.

[11] Hyungsub Shim, Gyuhwan Oh, “Network Latency Compensation for Low Speed Projectile in First Person Shooting Game,” Korea Society for Computer Game No15, 2008.12.

[12] Wladimir Palant, Carsten Griwodz, Pål Halvorsen, “Evaluating dead reckoning variations with a multi-player,” University of Oslo, 2006



임 종 민 (Lim, JongMin)

2007년 3월-현재 한국산업기술대학교 게임공학과 전공 재학 중

관심분야 : 게임 엔진 설계, 게임 사운드 프로그래밍

이 동 우 (Lee, DongWoo)

2007년 3월-현재 한국산업기술대학교 게임공학과 전공 재학 중

관심분야 : 게임서버 프로그래밍, 게임 기획

김 영 식 (Youngsik Kim)

1993년 연세대학교 컴퓨터과학과 학사
1995년 연세대학교 컴퓨터과학과 석사
1999년 연세대학교 컴퓨터과학과 박사
1999년-2005년 삼성전자 System LSI 책임연구원
2013년 University of Pittsburgh 방문교수
2005년-현재 한국산업기술대학교 부교수

관심분야 : 게임기구조, 컴퓨터구조, 3차원 그래픽가속기, 임베디드 시스템 등