네트워크 게임을 위한 데드 러커닝 알고리즘의 설계 및 구현

김 성 락[†] · 윤 남 균[†] · 구 용 완[††]

요 약

네트워크 게임은 네트워크상의 분산된 사용자들이 경쟁과 협력을 통하여 결과를 만들어 가는 것으로서 공동작업의 한 형태로 볼 수 있다. 본 논문에서는 그룹 통신 플랫폼을 사용하여 네트워크 게임을 수행하기 위한 요구사항을 정리하고 이러한 요구사항을 수용하는 데드 러커닝의 설계와 구현에 관하여 기술한다. 이러한 노력은 게임 개발의 생산성을 높여주고, 사용자 간 대화기능 등을 활용하여 게임의 질을 높일 수 있다. 또한 다양한 망과 단말의 능력 또한 게임 제공자, 게임 그룹 생성자의 의견에 따라서 유연한 형태로 다양한 결과를 제공할 수 있다.

Design and Implementation of Dead Reckoning Algorithm for Network Game

Seong-Rak Kim[†] · Nam-Kyun Yun[†] · Yong-Wan Koo[††]

ABSTRACT

Network games can be regarded as a kind of group work to make target results through competition and cooperation. This paper summarizes the requirements for a group communication platform to support multi-user network games and describes the design and implementation principles of such a dead reckoning. This approach enhances the productivity of network game development by separating the development phase of a game from making it networked. Under the this paper, flexible enough to provide multimedia games networked by various forms of architectures.

1. 서 론

인간과 인간의 대결로 벌어지는 네트워크 게임은, 새로운 형태의 사이버 문화를 형성하면서 거대한 문화적인 조류로 다가오고 있다. 더욱이 최근에는 클라이언트의 동기화, 서버의 분산을 위한 작업부하(work-load) 분배, 그래픽 효과(effect) 및 기술, 네트워크 관련 기술 등 고속 인터넷 시대에 걸맞는 통신환경의 발달로, 멀티미디어 데이터의 고속통신이 더욱 용이하게 지원됨으로써, 네트워크 게임은 보다 큰 폭으로 성장하고 있다[1]. 네트워크 게임은 네트워크 상에 분산되어 있는 서로 다른 사용자들이 상호간의 경쟁과 협력을 통하여 결과를 만들어 가는 특징을 지닌다. 네트워크 게임의 이러한 특징은, 게임을 그룹웨어(groupware) 또는 컴퓨터지원 공동작업(CSCW : Computer Supported Cooperative Work)의 한 형태로 볼 수가 있음을 의미한다. 이러한 관점은 다자간 네트워크 게임의 개발과 서비스에 근본적인 변화를 가능하게 해준다[2].

네트워크 게임 제작을 위한 여러 가지 기반 기술의 가장 핵심적인 내용은, 현재 느린 대역폭을 가진 네트워크 선로에서 사용자가 네트워크에 접속되어 있다는

† 정 회 원 : 오산대학 정보관리과 교수
†† 종신회원 : 수원대학교 전자계산학과 교수
논문접수 : 2000년 2월 10일, 심사완료 : 2000년 6월 30일

느낌이 들지 않을 정도의 빠른 성능과 무한대에 가까울 정도의 사용자 접속을 수용하는 방안으로 기술 개발을 하는 것이다. 또한, 어떠한 외부 돌발 변수, 즉 접속이 잠시 끊어지거나, 지연이나 충돌 문제와 같은 어떠한 네트워크 상황에서도 이를 효율적으로 제어하여 사용자가 전혀 느끼지 못할 정도로 제작해야만 한다. 그러나 현재의 네트워크 대역폭은 실시간으로 움직이는 서로 간의 게임 데이터를 전송하기는 불가능하다. 따라서 각 클라이언트는 이러한 메시지 송수신을 최소화하고 극도로 제한된 메시지 패킷을 가지고 예측하는 알고리즘이 필요하게 된다. 이러한 네트워크 환경에서 객체들을 동기화 시키고 제한된 메시지 패킷을 가지고 이를 예측하는 알고리즘으로 현재까지는 데드 러커닝(dead reckoning)이 유일한 방법이다.

원래 데드 러커닝은, 각 클라이언트들이 제각기의 예측 알고리즘을 가지고 모든 클라이언트들의 상황을 계산하는 도중, 오차가 게임을 진행하기 힘들 정도로 발전하는 상황을 말하는 아주 좁은 뜻의 단어이지만, 현재는 동기화·예측 알고리즘을 포함하는 모든 알고리즘을 뜻하는 말로 그 의미가 확대된 상황이다. 따라서, 본 논문에서는 데드 러커닝을 후자의 의미로 사용한다.

논문의 구성은 1장 서론에 이어, 2장에서는 네트워크 게임의 기본 개념 및 구성요소를 기술하고, 3장에서 효율적인 데드 러커닝 검출 및 제거 알고리즘과 이를 위한 각 클라이언트의 동기화 방법, 그리고 각 클라이언트에서 접속된 다른 모든 클라이언트의 상태를 예측 할 수 있는 알고리즘을 제시한다. 4장에서는 본 논문의 데드 러커닝 알고리즘을 기반으로 제작된 3차원 채팅 비비에스(BBS)의 구현 화면과 함께, 최대 사용자수·동시 접속 시도 사용자수·이동 메시지(move message)를 중심으로 하는 시뮬레이션 데이터 값을 기술한다.

2. 네트워크 게임(Network game)

2.1 네트워크 게임의 분류

최근의 게임 소프트웨어들은 소형 컴퓨터의 급속한 발달로 인해 음향효과나 정교한 그래픽 처리를 이용한 실제와 유사한 환경 하에서 게임을 즐길 수 있도록 제작되어진 것들이 많이 있다. 또한, 정보 통신 기술의 발달로 기존의 한두 사람에 의한 게임 진행 방식을 탈피하여 여러 명의 사용자들이 동시에 한 게임을 진행하는 다중 사용자용 게임이 많이 추천되고 있다[3]. 이러한 네트워크 게임은, 이미 설계되어 있는 시나리오에 따라 게임이 진행되는 것이 아니라 게임에 참가한 사람들의 각각의 상상력을 통해 내용이 전개되는 것이다. 그러므로 배경 스토리를 바탕으로 한 기본적인 가상 세계 구축만 게임 기획자가 하는 일이며, 그 속에서 경험하는 것들은 모두 사용자들의 행동에 따라 결정되므로, 끊임없는 변화와 발전이 네트워크 게임을 이끌어 가는 가장 중요한 요소이자 특징이다. 네트워크 게임을 분류하면 다음과 같다.

2.1.1 진행 형태에 의한 분류

현재 네트워크 게임은 대개 10명 내외의 사용자들이 게임을 하는 경우인 IPX를 기반으로 하는 게임과 수천의 사용자들이 한 장소에서 같이 게임을 하는 방식인 TCP/IP를 이용하는 경우가 있다. TCP/IP를 기반으로 하는 게임은 모두 인터넷을 기반으로 동작되며, IPX를 기반으로 하는 게임은 특별한 서버 없이 로컬(local) 네트워크 상에서 자신들끼리 서버를 만들고 이 안에서 게임이 진행되는 형태이다. 현재 대부분의 네트워크 게임은 이 두 가지 형태를 모두 지원하고 있다.

① IPX 기반

IPX를 기반으로 하는 게임은 지역 네트워크 내에서만 상대편이 보이게 되며, 게임을 하는 사람 중에 한 사람이 서버로 동작하고, 다른 사람들은 이 서버에 의해 게임을 진행 받게 된다. 겨우 8명 내지 많으면 10명을 넘지 않는 상황이기 때문에, 네트워크 성능이나 기타 사용자들 간에 물리적인 위치에 영향을 받지 않으며, 한 개의 서버에 걸리는 노드가 매우 작기 때문에 개발에 대한 어려움은 그리 크지 않다.

② TCP/IP 기반

TCP/IP를 기반으로 하는 게임은 수백 혹은 수천의 사람이 하나의 게임 서버에 연결하여 이를 통해 서로 대화하며 경쟁하는 형태이다. 이러한 형태의 게임 중에서 가장 근간이 될 수 있는 것이 텍스트를 기반으로 하는 머드(MUD : Multi-User Dungeon)이다.

TCP/IP를 기반으로 하는 게임은 대개 한 개의 거대한 서버를 사용하기 때문에 서버의 성능에 의존하게 되며, 이런 특성은 게임에 있어 멀리 떨어진 사용자들을 고려해야 하기 때문에 네트워크 기술이 중요하다.

2.1.2 게임 진행자 수에 의한 분류

① 일 대 일(head to head) 방식

두 사람이 각각 컴퓨터를 연결하여 게임을 진행하는 방식이며, 보통 peer to peer 형태의 네트워크 형태를 가지고 있다. 두 사람만이 게임을 진행하기 때문에 대개 한 사람이 접속 가능한 호스트를 만들고, 다른 사람이 이 호스트의 주소와 포트를 알아내서 접속을 하게된다.

② 다중 사용자(multi-player) 방식

다수의 사용자가 게임을 진행하는 방식이며, 다수의 컴퓨터가 연결되어야 하므로, 서로간의 메시지 교환이나 전송을 제어하기 위한 중계기 역할 기능을 하는 서버가 반드시 필요해 진다.

2.1.3 게임 진행 방식에 의한 분류

① 라운드 베이스드(round based) 방식

반환(turn) 형식(즉 바둑이나 장기처럼 한 사람이 행동을 취한 후 다른 사람이 다음 행동을 취하는 형식)을 기본으로 하는 방식으로, 구현이 상당히 간단하며 대규모의 네트워크 대역폭과 신뢰도를 요구하지 않아 텍스트 기반 네트워크 게임(MUD)에 사용된다.

② 액션 베이스드(action based) 방식

실시간(real time)으로 진행하는 방식(검도나 태권도처럼 서로가 동시에 행동을 취하는 방식)으로, 현재 대부분의 네트워크 게임에서 사용하는 방식이며, 구현이 복잡하고 제작이 어렵다. 또한 빠른 전송 속도, 고 기능의 서버를 요구하며, 현재 네트워크 대역폭으로는 정확한 구현은 불가능하기 때문에, 게임 진행에 무시할 수 있는 정도의 오차는 용인한다.

2.1.4 서버의 종류에 의한 분류

① 피어 서버(peer server) 방식

각각의 클라이언트들은 클라이언트 기능과 서버 기능을 동시에 갖는다. 즉 소수의 사용자중 서버가 될 사용자를 정하고, 그 서버를 중심으로 게임을 진행하는 방식이며 8명 이하의 적은 수의 사용자가 게임을 하기에 적당하고, 속도가 빠르다.

② 원격 서버(remote server) 방식

수많은 사용자가 동시에 접속하여 게임을 진행하는 방식으로 오버 헤드가 크고, 메시지 교환이나 패킷 전송량이 엄청나다. 서버에 걸리는 부하가 크기 때문에, 전용 서버 역할을 하는 컴퓨터를 따로 두게되고, 서버는 사용자들간의 메시지 전달과 조정에만 관여하며 게임 진행에는 일체 관여하지 않는다.

2.2 메시지(message)

본 논문에서의 메시지란 클라이언트/서버 통신상에서 네트워크 선로를 통해 주고받는 통신의 규약이다. 따라서, 동기화 및 데드 러커닝의 구현 알고리즘의 핵심은 이러한 메시지를 최대한 줄여서 네트워크 부하에 상관없이 게임을 진행하게 하는 것이다.

2.2.1 메시지(message)의 종류

① 일반 메시지(common message)

통상 사용자가 이동을 하거나 대화를 할 때 이를 알리기 위한 실제 데이터가 들어간 메시지이다. 예를 들어 채팅(chatting)을 위한 텍스트(text) 메시지, 이동(move) 메시지 같은 부류가 여기에 속한다.

② 시스템 메시지(system message)

각 클라이언트간, 혹은 서버와 클라이언트간에 효율적인 데이터 전송과 네트워크 상황을 제어하기 위해 보내는 내부 메시지이다. 예를 들어 오랫동안 반응이 없는 클라이언트의 접속 여부를 묻는 메시지 같은 부류가 여기에 속한다.

〈표 1〉 일반 메시지와 시스템 메시지의 특성

특성 \ 종류	일반 메시지	시스템 메시지
사용 빈도	높음	낮음
메시지 길이	길고 가변적임	짧고 고정적임
생략 가능 여부	가능	불가능
예측 가능 여부	가능	불가능

<표 1>과 같이 일반 메시지는 시스템 메시지에 비해 클라이언트/서버 통신상에서 많은 부하를 차지하고 있고 메시지를 놓쳐도 치명적인 에러가 없으며, 다른 클라이언트에서 예측이 가능하다. 그러므로, 동기화 및 데드 러커닝 알고리즘의 구현은 일반 메시지를 중심으로 이루어져야 함을 알 수 있다. 여기서 예측이란 정확하게 상황을 맞추는 것이 아닌 한번 이동을 하면 계속 이동을 하고 있으며, 한번 대화를 하면 계속 대화를 할 확률이 높다는 것에 착안한 확률적인 예측이다.

반면 시스템 메시지는 언제 사용자가 접속이 끊길지, 혹은 언제 네트워크 트래픽(traffic)이 걸려 메시지가 오는 속도가 불규칙하게 될지 예측하기가 불가능하므로 시스템 메시지를 중심으로 동기화 및 데드 러커닝 알고리즘을 구현하기는 부적합하다.

2.2.2 메시지 통신

① 동기화 통신(synchronous communication)

바둑이나 장기처럼 라운드 베이스드(rounded based) 운영 방식이며, 서버는 로그인한 사용자들의 순번을 정해주고, 사용자가 행동을 결정 할 때까지 다음 사용자는 기다린다. 이 방식은 많은 사람들이 게임을 하더라도 한번에 메시지를 주고받는 사용자는 하나 뿐이므로 메시지 통신 부하가 적고 서버의 부담이 적다.

② 비동기화 통신(asynchronous communication)

실시간(real time)을 요구하는 게임의 경우에는 몇 사람이 동시에 게임을 할지 알 수 없고, 메시지를 전달하고 그에 따른 응답이 올 때까지 기다린다면, 빠르고 박진감이 있는 게임을 즐길 수 없다. 물론, 이 경우에도 메시지를 보내고 나서 일정 시간 동안 응답 메시지를 기다리지만, 동기화 통신 방식처럼 무작정 기다리지 않는다.

비동기로 움직이는 각 클라이언트들은 각기 서로 다른 데이터를 가지게 된다. 따라서, 비동기화 통신의 경우, 어느 정도 게임을 진행하는데 지장이 없는 오류, 즉 실제 어떤 물건의 정확한 위치는 X는 10.0 Y는 12.1인데 서버나 사용자는 X는 10.2 Y는 11.9 정도로 알고 있을 정도의 오차는 무시하고, 이러한 오류가 게임에 지장을 줄 정도로 생기는지 검출하고, 만일, 오류가 게임에 지장을 줄 정도로 생겼다면, 이를 해결해야 할 알고리즘이 필요하다.

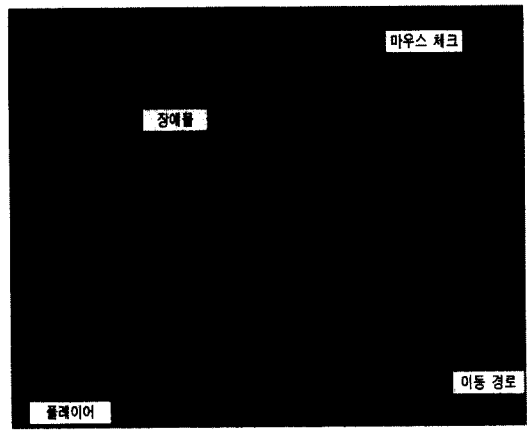
또한, 비동기화 통신에서 모든 클라이언트는 서버와 실시간으로 아무 때나 데이터를 주고받는다. 따라서, 각 클라이언트는 이러한 메시지를 보관 할 수 있는 메시지 큐와 서버와 접속되어 있는 다른 클라이언트의 정보를 표현하기 위해 타 클라이언트 예측 데이터 버퍼를 가지고있다. 물론 동기화 통신에서도 이러한 버퍼와 큐를 가질 수 있지만, 비동기화 통신에서 이러한 버퍼와 큐는 프로세서(processer)나 쓰레드(thread)로 제어가 되야 한다. 비동기화 통신에서 메시지는 서버와 항상 실시간으로 주고받기 때문에 이러한 버퍼와 큐가 프로세서나 쓰레드로 제어되지 않으면 랜더링 시간(rendering time)은 블록킹(blocking)이 되기 때문에 메시지를 받을 수가 없게 되기 때문이다.

2.3 사용자 인터페이스

데드 러커닝을 구현하기 위해서는 다른 사용자의 상태를 예측하는 알고리즘을 구현해야 한다. 예측 알고리즘은 사용자의 입력 인터페이스(input interface)에 따라 알고리즘의 구성이 달라지게 된다.

2.3.1 마우스 체크 방식

(그림 1)과 같이 마우스 버튼으로 미리 갈 곳을 포인트하고, 그 과정은 컴퓨터가 알아서 길 찾기 알고리즘으로 찾아가는 방식이다. 이 방식을 사용하면 네트워크 패킷의 양이 줄고, 서로 간의 길 찾기 예측 알고리즘으로 상대방의 위치를 제어 할 수 있다. 즉, 움직일 때마다 메시지를 보내지 않고, 이동할 목표점만 서버에게 보내주고 서버 역시 다른 모든 클라이언트에게 목표점만 보내주면 다른 클라이언트들은 이동한 사용자의 위치를 예측 할 수가 있어진다. 현재 블리자드(Blizard)사가 많이 쓰고 있는 입력 인터페이스이다.



(그림 1) 마우스 체크 인터페이스

2.3.2 입력 예측 방식

이 방식은 사용자들이 한번 앞으로 가면, 계속 앞으로 갈 확률이 높고, 장애물이 있을 경우, 그를 피하는 패턴 역시 상당히 일정하기 때문에 그를 이용하여 사용자들의 위치를 예측하는 경우이다. 즉, 한번 이동을 시

작 할 때 메시지를 보내고, 다시 이동을 멈추거나, 다른 방향으로 바꾸었을 때만 메시지를 보내는 방식이다.

2.4 지연과 충돌

지연과 충돌은 전체 데드 러커닝의 알고리즘에서 예외처리에 해당되는 부분이며 이러한 지연과 충돌은 너무나 빈번히 일어나기 때문에 확실하고 안정적인 처리만이 전체 데드 러커닝 알고리즘의 성능을 좌우하게 된다.

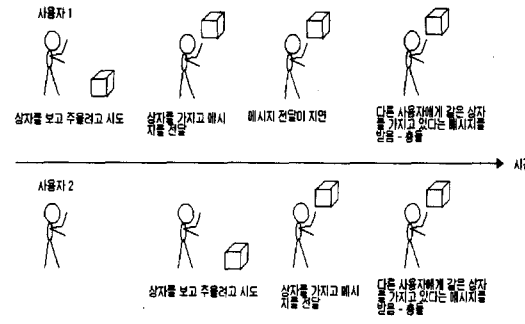
2.4.1 지 연

네트워크를 통해서 온 데이터는 결국 전송 시간만큼의 과거 데이터가 된다. 만일 이러한 지연이 0.1 밀리 세컨드 이하라면 어느 정도 허용할 만 하지만, 대부분의 경우 수십 밀리 세컨드이며, 어떤 경우는 1초를 넘어가는 상황까지 발생한다. 더구나, 운영체제내의 스케쥴링 정책 때문에 비록 메시지를 빨리 보냈지만 다른 메시지가 보내진 후 메시지가 보내지는 상황까지 벌어진다. 이러한 지연은 메시지 통신마다 반드시 일어나는 예외처리 상황이며, 예측 알고리즘을 어렵게 만드는 원인중의 하나이다. 따라서, 메시지를 받는 다른 클라이언트는 그 시간을 참조하여 너무 오래된 시간은 **버리고, 구현 가능한 시간의 데이터만 갱신하게 된다.**

처음 동기화 과정에서 보내는 호출 메시지(ping message)와 이에 대한 응답 메시지의 시간을 산출한 후 이에 따른 10회 정도의 평균값을 현재의 클라이언트 시간과 더해 서버에게 보내면 서버는 다시 모든 클라이언트에게 메시지를 전송하게 된다.

2.4.2 충 돌

(그림 2)에서처럼 한 사용자가 상자를 집게 되면, 자신의 환경 내에 상자의 상태가 변경되면서 동시에 다른 사용자들에게 상자를 집겠다는 메시지를 보낼 것이다. 일반적으로 다른 모든 사용자들의 확인 메시지까지 받아야 상자를 주울 수 있겠지만, 이러기엔 네트워크 대역폭이 너무 느리다. 그런데, 다른 사용자가 이러한 메시지를 받기 전에 상자를 줍고, 지연이 어느 정도 일어나면 (그림 2)와 같은 충돌은 충분히 일어 날 수 있는 일이며, 상자를 줍는 문제가 아닌, 더 빈번한 접촉, 예를 들어 칼 싸움이나 총 싸움의 경우에는 더욱 더 빈번히 발생한다.

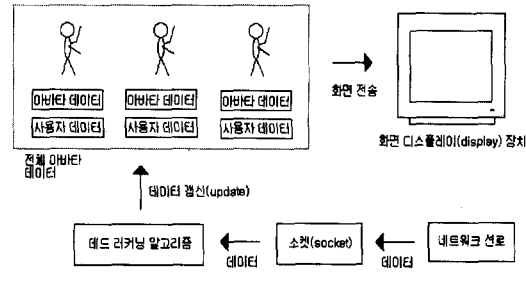


(그림 2) 네트워크 게임에서의 충돌

3. 데드 러커닝(dead reckoning) 구현

3.1 데이터 구조(data structure)

네트워크 게임의 화면을 구성하기 위해서는 현재 접속 되어있는 다른 사용자의 모습을 표현해야 하고, 어떤 행동을 취하고 어디에 위치에 있는지 알려 주어야 한다. 이러한 표현은 다른 사용자들의 상태이기 때문에 네트워크를 통해 그 데이터를 받아야만 비로소 화면에 표현 할 수가 있다. 데드 러커닝 알고리즘 중 사용자가 어떤 행동을 취하는지를 나타내는 데이터가 바로 사용자 데이터(user data)이고, 사용자의 모습을 나타내는 데이터가 아바타 데이터(avatar data)[1]이다. (그림 3)은 네트워크 게임에서 데이터의 흐름을, 데드 러커닝 알고리즘과 아바타 데이터와 사용자 데이터의 관계로 나타낸 것이다.



(그림 3) 네트워크 게임에서 데이터의 흐름

앞서 데드 러커닝 알고리즘의 핵심은 다른 사용자에게 오는 메시지(message)를 최소화하여 네트워크 트래

1) 힌두 신화에서 유래된 단어로 분신, 화신이란 뜻으로 울티마(Ultima)라는 게임에서 사용되어진 이후 네트워크 상에서 자신의 이미지 데이터(image data)를 통칭하는 말로 고유명사화 되었다.

픽(network traffic)을 해결하는 것이라 했다. 따라서, 데드 러커닝 알고리즘을 구현하는 일은 위의 두 데이터를 제어하는 것이 핵심이다. 즉, 최소한으로 오는 메시지를 중심으로 다른 사용자의 상태를 예측하여 아바타 데이터와 유저 데이터에 그 예측한 데이터를 갱신(update)하고, 만일 그 예측한 데이터 값이 게임에 지장을 줄 정도로 심각하다면 그 사용자의 아바타 데이터와 유저 데이터를 재전송 하도록 서버에게 요구하는 것이다.

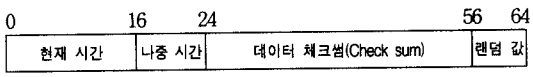
3.1.1 아바타 데이터(avatar data)

아바타 데이터는 사용자의 모습을 정의한 데이터이며, 구조를 정의하면 (그림 4)와 같다. AVATAR_ID는 64비트(bit) 길이의 숫자이다. 이 AVATAR_ID의 원리는 Active-X나 OLE 컨트롤의 GUID와 그 원리와 동작하는 순서가 같으며, (그림 5)는 AVATAR_ID의 비트(bit) 구조를 표현한 것이며, 사용자가 특수한 편집기를 통해 아바타 이미지를 만들면 생성된다.

처음 16비트는 현재 생성한 순간의 시간을, 다음 8비트는 랜덤한 시간이 지난 후 그 시간 을 8비트로 압축해서 표현하고, 다음 32비트는 아바타 데이터의 각종 데이터를 체크 썸(check sum)한 값을 표현한다. 그리고 그 이후의 8비트는 랜덤 값을 갖는다.

```
typedef struct _avtardata
{
  UINT          AVTAR_ID;
  CSIZE         AVATAR_SIZE;
  C3DPolygon    *pAVATAR_POLYGON_DATA;
  C3DTexture    *pAVATAR_TEXTURE_DATA;
  CMotion       *pAVATAR_MOTION_DATA;
  BOOL          isFree;
} AVATARDATA;
```

(그림 4) 아바타 데이터의 구조



(그림 5) AVATAR_ID의 비트(bit) 구조

AVATAR_ID의 가장 큰 용도는 하드디스크의 캐쉬(cache) 검색이다. 네트워크 상에서 새로운 사용자가 등장하면, 각 클라이언트는 현재 새로운 사용자의 아바타 데이터가 하드디스크 안의 캐쉬에 있는지 검색한다. 만일 새로운 사용자가 이전에 만난 적이 있는 사용자라면, 하드디스크 안에 아바타 데이터가 남아 있으므로 하드디스크 안에 있는 아바타 데이터를 기반으로 현재 화면을 재구성하게 된다. 만일 그렇지 않고 하드디스크 안에 아바타 데이터가 있지 않으면, 서버에게 새로운 사용자의 아바타 데이터를 요구하고 네트워크 상에서 전송을 받은 후 화면을 재구성하게 된다. 이러한 아바타 데이터를 검색하는데 키 데이터가 되는 것이 바로 AVATAR_ID이며, 처음 새로운 사용자가 등장하게 되면 서버는 모든 각 클라이언트에게 이 AVATAR_ID만 전송한다. (그림 6)은 본 논문에서 제안하는 AVATAR_ID를 이용한 하드디스크의 캐쉬 검색 알고리즘이다.

AVATAR_SIZE는 물질적인 아바타 이미지의 크기이며, 나머지 pAVAT-AR_POLYGON _DATA는 3차원 데이터, pAVATAR_TEXTURE_DATA는 텍스쳐 맵핑 소스(texture mapping source)용 데이터, pAVAT-AR_MOTIO-N_DATA는 각각의 행동 데이터이다. 마지막으로 isFree는 공개인지 아닌지를 나타내는 플래그이다. 만일 이 플래그가 참이라면, 다른 사용자가 이 아바타 이미지를 사용할 수 있게 하는 것이고 그렇지 않다면, 아바타 이미지를 만든 사람만 사용할 수 있게 된다. 공개 여부는 아바타 편집기에서 지정해주며 이러한 부분은 본 논문의 범위를 벗어나기 때문에 생략한다.

```
AVATARDATA* GetAvatarData(AVATAR_ID id)
{
   LoadAvatarfromCache(id) // 캐시에서 읽어 온다.
}
void ReceiveFromServer(AVATAR_ID id)
{
   AVATARDATA* data = GetAvatarData();

   if( data == NULL) SendServer(id);
   // 만일 하드디스크나 메모리에 아바타 데이터가 없으
      면 서버에게
   // 데이터 요청을 한다.
   else DrawAvatar( data );
}
void ReceiveFromServer(AVATARDATA data)
{
   DrawAvatar( data );
   SaveAvatartoCache( data );
   // 처음 그리는 아바타는 캐시에 저장한다.
}
```

(그림 6) AVATAR_ID를 이용한 캐쉬 검색 알고리즘

3.1.2 사용자 데이터(user data)

아바타 데이터가 사용자의 모습을 보여 주기 위한

데이터이기 때문에 상수(const)의 성격을 가지고 있다면, 사용자 데이터는 사용자의 현재 상태나 위치를 표현하기 때문에 변수(value)의 성격을 가지고 있다.

(그림 7)의 사용자 데이터 구조에서, nu_gesture_flag는 아바타 데이터의 pAVATAR_ MOTION_ DATA의 값과 같은 값을 가진다. nu_chprivate는 사용자의 개인 신상을 공개 할 것인지 안 할 것인지를 선택하는 플래그이고 nu_chghost는 유령모드인지 아닌지를 판단하는 플래그이다. 만일 유령모드로 세팅되어 있다면, 다른 사용자는 자신을 볼 수가 없게 된다. nu_name은 현재 사용자의 실제 이름, nu_nick은 사용자의 아이디, nu_sex는 성별을 나타낸다. 이상이 신상을 기록한 간단한 데이터들이며, nu_command는 현재 어느 명령을 사용자가 내렸는지 설정하는 작은 버퍼이며, 대화를 했을 경우는 대화 내용이 nu_command에 들어가게 된다. nu_x, nu_y는 위치를 나타내는 중요한 데이터들로, 예측 알고리즘에서 각 사용자 데이터들의 위치를 계산해 이곳에 값을 집어넣는다.

```
typedef struct _userdata
 {
      int     nu_gesture_flag;
      BOOL    nu_chprivate;
      BOOL    nu_chghost;
      char    nu_name[MAX_NAME_SIZE];
      char    nu_nick[MAX_NICK_SIZE];
      char    nu_command[MAX_COMMAND_SIZE];
      int     nu_sex;

      float   nu_x;
      float   nu_y;
      float   nu_directionH;
      float   nu_directionV;

 } USERDATA;
```

(그림 7) 사용자 데이터의 구조

3.2 데드 러커닝 알고리즘

(그림 8)과 (그림 9)는 본 논문에서 제안하는 알고리즘으로, (그림 8)은 네트워크 게임의 메인(main) 부분에서 데드 러커닝 알고리즘을 호출하는 내용이며, (그림 9)는 실질적인 데드 러커닝 구현 알고리즘이다.

USERDATA는 앞에서 기술한 사용자 데이터이고, Threshold는 어느 정도의 오차를 허용 할 것 인지의 오차 값이다. 이 오차 값을 크게 잡으면 네트워크 패킷 전송 수가 줄어들어 속도는 개선되지만, 부정확한 사용자 데이터 값을 가지게 된다.

Loop() 함수는 네트워크 게임의 메인 부분에서 유휴시간(idle time)에 호출되는 함수이며, 클라이언트의 성능이 높을수록 자주 호출되는 함수이다. 따라서 클라이언트의 성능이 높으면 이 함수가 자주 호출되어 더욱 더 정확한 사용자 데이터 값을 가지게 된다.

현재 나 자신을 **I**, 내가 사용하고 있는 컴퓨터를 **A**, 다른 모든 사용자들을 **B**라고 정의하자. Loop()함수 내의 ProcessUser() 함수가 바로 예측 함수이다. 즉, **A**가 **B**의 모든 상황들을 예측하게 해주는 함수이다. ProcessUser() 함수를 호출하고 나서, **I**의 입력을 받는데 **I**의 입력한 결과 값과 **B**가 사용했을 **A**의 사용자 데이터 값의 차를 비교해서 Threshold보다 크면, 서버에게 데이터를 보내며, 서버는 **B**에게 **A**의 사용자 데이터를 다시 받아서, 다시 갱신 하도록 명령하는데, 그 기능을 수행하는 함수가 바로 ReceiveFromServer()이다.

```
void main()
 { while(1)
     Loop();          // 알고리즘 4 실행..    }
```

(그림 8) 메인 부분에서의 데드 러커닝 알고리즘 호출

```
USERDATA   PDU[MAX_USER];   // 각 유저들의 데이터
USERDATA   HERO;            // 본인의 데이터
UINT       Threshold = DEFINED_THRESHOLD
// 무시할수 있는 오류 값.
void Loop()
{
  int i;
  for(i = 0; i<MAX_USER; i++)
        ProcessUser( PDU[i]);
  // 모든 유저들의 위치 상황을 예측한다.
  UNIT user = InputKey();          // 사용자에게 키 입력을
                                      받는다.
                                   // Non-blocking 함수
  if( ProcessUser( HERO ) - user > Threshold )
      SendtoServer(HERO);
  // 모든 유저들의 위치 상황을 한 알고리즘으로 본인도 예측해
     본다.
  // 만일 오차가 심각한 수준이면 서버에게 데이터를 보낸다.
  for(i = 0; i<MAX_USER; i++)
        DrawUser( PDU[i]);         // 모든 유저들을 그린다.
  DrawUser(HERO);                  // 본인을 그린다.
}
void ReceiveFromServer(USERDATA data)
{
  InitProcessUser(data);
  // 오차가 심한 유저 데이터를 재 설정한다.
}
```

(그림 9) 데드 러커닝의 구현 알고리즘

3.3 동기화(synchronize) 알고리즘

동기화의 구현 알고리즘을 실제 데드 러커닝에 적용하려면 두 가지 상황을 고려해야 한다. 첫번째는, 후진입이 불가능한 경우인데, 한번 게임을 시작하면, 다른 사용자는 게임을 하는 중간에 진입을 할 수가 없는 경우이다. 대개 피어 서버 방식(peer server)의 네트워크 게임에서 사용되어지며, 서버기능을 담당하게 된 클라이언트가 각 모든 클라이언트에게 호출 메시지(ping message)를 보내서 응답한 시간을 참조하여 그 평균값을 기초로 시간을 맞추는 작업을 하게 되는 경우이다. 두번째는, 후진입이 가능한 경우로 게임 진행 중에도 다른 사용자가 게임에 진입 할 수 있는 경우이다. 본 논문에서 사용하는 시뮬레이션은 3차원 채팅 비비에스(BBS)로, 현재 사용자가 있음에도 또 다른 사용자가 접속하여 같이 채팅이라던지, 게시판 서비스나 자료실 서비스를 받을 수 있어야하므로 후진입이 가능하도록 동기화를 구현한다. 동기화의 구현 단계는 아래와 같다.

1. 클라이언트는 서버에게 호출 메시지(ping message)를 보낸다.
2. 서버는 클라이언트에게 응답 메시지를 보낸다.
3. 클라이언트는 서버에게서 응답 메시지를 받은 후 시간을 계산하고 1, 2의 과정을 10번 되풀이 하여 그 평균값을 구한다. 그 평균 값을 t라고 한다.
4. 서버는 클라이언트에게 현재 시간을 보낸다. 그 현재 시간을 s라고 하고, 그 메시지가 온 순간의 클라이언트 시간을 c라고 한다.
5. 현재 클라이언트의 시간을 T라고 하면, 현재 클라이언트 상에서 서버의 시간을 구할 수 있는 공식은 'S = T - (c - (s + t))'이다. 공식을 살펴보면 s + t는 서버의 시간과 평균 네트워크 지연 값의 합이다. 따라서 클라이언트에서는 s + t가 현재 서버의 시간임을 알 수가 있다. 여기에 현재 클라이언트 시간인 c에서 s + t를 뺀다면 서버 시간과 클라이언트 시간과의 시간차를 알 수가 있으며, 임의적인 클라이언트 시간인 T에서 위의 계산된 값을 뺀다면 그에 해당되는 임의적인 서버 시간을 구할 수가 있다. 이제, (c - (s + t))를 동기화 상수 A로 정의하면 최종 공식은 아래와 같다.

$$S = T - A$$

6. 모든 메시지 처리 및 시간 계산은 이 서버 시간을 중심으로 처리가 되어야 시간을 맞추는 동기화가 가능하며 동기화 상수 A를 구하는 알고리즘은 (그림 11)이다.

```
static int     nMessageCount = 0;
static TIME nTime[10], nFirstTime[10];
static TIME nServerTime[10], nClientTime[10];
static BOOL isBlocking = TRUE;
void ReceiveMessage()     //based Thread
{
 ReceveFromServer(&nServerTime[nMessageCount]);
 //서버에게서 서버가 메시지를 보낸 순간의 시간도 받는다.
 nClientTime[nMessageCount] = GetTimetick();
 nTime[nMessageCount] = nClientTime[nMessageCount] -
                                    nFirstTime[nMessageCount];
 nMessageCount++;
 if(nMessageCount > 10) isBlocking = FALSE;
}
void SendPingMessage()       //based Thread
{
  nFirstTime[nMessageCount] = GetTimetick();
  SendServer(PING_MESSAGE);
}
int Get()
{
 int i, t, c, s, s1 = 0, s2 = 0, s3 = 0, A = 0;
 for(i=0; i<10; i++)     SendPingMessage();
 Blocking( isBlocking );
 // isBlocking 값이 FALSE 일 때 까지 블록킹한다.
 for(i=0; i<10; i++) s1 += nTime[i];
 t = s1/ 10;        // 평균 네트워크 지연 값을 구한다.
 for(i=0; i<10; i++) A += nClientTime[i]-(nServerTime[i]+t);
 return (A/10);    // 동기화 상수를 구함.
}
```

(그림 11) 동기화 상수를 구하는 알고리즘

4. 네트워크 게임의 구현 및 성능평가

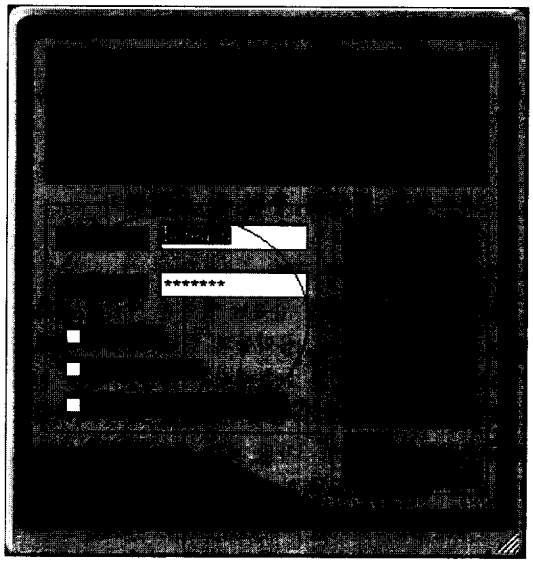
4.1 네트워크 게임의 구현

게임을 제작하는 일은 엄청난 인력과 시간을 필요로 한다. 그래픽, 시나리오, 음악등의 연출 작업이 필요하기 때문이다. 따라서 본 논문에서는 제안한 데드 러커닝 알고리즘을 구현하기 위해서, 3차원 채팅 비비에스를 구현하였다. 이러한 3차원 채팅 비비에스의 구현은, 네트워크 게임제작을 위한 그래픽, 시나리오, 음악 등의 작업이 이루어지면, 본 알고리즘을 언제든지 탑재할 수 있음을 의미한다. 3장의 데드 러커닝 알고리즘에 따라 구현된 모델은 TCP/IP 기반이며, 다중 사용

자, 액션 베이스드, 원격 서버 기반 모델이다. 서버는 Unix 기반이며, 클라이언트는 Windows 95/98을 운영체제로 사용하고, 소켓(socket) 프로그래밍을 이용하여 네트워크 통신을 하였다. 서버는 Unix 기반의 GCC로 제작되었고, 클라이언트는 Visual C++ 6.0, MFC - DirectX 6.0 - IM Mode로 개발하였다.

3차원 채팅 비비에스는 현재 3가지의 기능을 가지고 있는데 첫 번째는 다른 사용자와의 채팅 기능, 두 번째는 게시판 열람 및 작성 기능, 세 번째는 파일 다운로드(download) 및 업로드(upload) 기능을 가지고 있다. 또한, 전자 메일(E-mail) 기능은 아직 지원하고 있지 않지만, 자체 내의 메일 기능은 지원하고 있다. 데드 러커닝 알고리즘의 성능을 구현하는 부분은 아바타가 표시되고, 상대방의 메시지에 따라 아바타가 이동하는 부분이다. 따라서, 시뮬레이션 상에서 상대방 아바타가 자연스럽게 이동하고, 대화하는 부분에서 어느 정도의 속도가 나오는지가 핵심이다.

(그림 12)는 처음 사용자가 제일 처음 시뮬레이션을 구동 시켰을 때 나오는 화면이다. 먼저 ①은 공지 사항을 나타내는 부분이다. 이 부분은 서버에게 메시지를 받아서 표시하는 부분이다. ②는 사용자의 입력 부분이며 아이디, 비밀 번호 등을 입력받는 부분이며, ③은 현재 자신의 아바타 모습을 바꾸어주는 인터페이스(interface)이다



(그림 12) 초기 접속 화면

(그림 13)은 데드 러커닝을 이용하여 상대방이 움직이는 모습이다. 움직이는 상대방은 인사말도 건네고 있다. 그림을 보면 점과 선이 있는데 상대방이 걸어온 길(path)을 표시하고 있다. 메시지를 받아서 상대방 아바타를 표시한 부분은 점으로 표시되어 있고, 그 외의 선 부분은 메시지를 받지 않고 상대방의 움직임을 예상하여 표시한 부분이다. 자세히 보면 방향을 바꾸었을 때만 메시지를 받고 그렇지 않을 경우는 메시지를 받지 않고 임의적으로 표시했음을 볼 수가 있다.

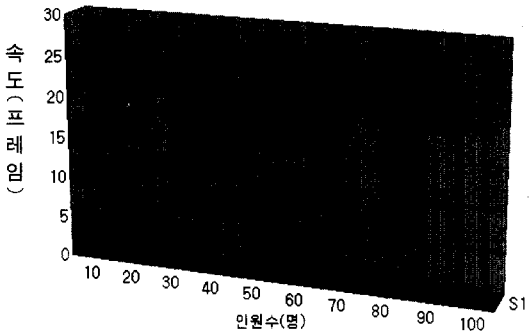
(그림 13) 데드 러커닝을 적용한 아바타의 표현

4.2 성능 평가

4.2.1 최대 사용자 수에 따른 성능 평가

이 평가는 알고리즘 자체의 성능보다는 서버의 성능에 따라 그 결과 값이 좌우되지만, 네트워크 게임 엔진의 성능 평가를 할 때, 중요한 항목이므로 테스트를 했다. 결과는 (그림 14)와 같이 30명 일 때 사용 인원 대비 최적의 성능을 발휘 하지만, 그 이상으로 인원이 늘어날수록 기하 급수적으로 성능이 저하됨을 알 수가 있다. 최대 사용자 수에 따른 성능 평가의 기준인 프레임은 1초에 화면이 몇 번 갱신되는지 나타낸다. 사용자 수의 세팅(setting)은 서버의 코드를 직접 수정하였으며, 결과 값은 클라이언트의 프로그램 내의 Loop()

함수 내에 속도를 측정하는 코드를 삽입하여 (그림 13)처럼 좌측 상단에 표시 되도록 하였다. 따라서, 이 결과 값은 시뮬레이션을 구동시키면 항상 볼 수 있다.



인 원 수(명)	10	20	30	40	50	60	70	80	90	100
속도(프레임)	28	24	22	16	9	4	2	1	1	1

(그림 14) 최대 사용자 수에 의한 평가 결과

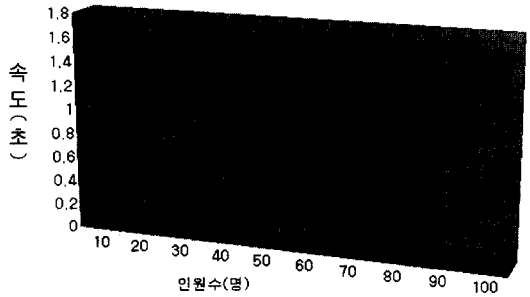
4.2.2 동시 접속 시도 사용자 수에 따른 성능 평가

사용자들이 접속시 얼마나 빨리 접속되어 동기화가 이루어지는가를 알아 볼 수 있는 평가로, 동시 접속과 비슷한 환경을 구현하기 위해, 서버에게 접속 요청 메시지만 보내는 독립된 작은 프로그램을 제작하였다. 접속 요청을 받은 서버는 각각의 클라이언트를 동기화하기 위한 메시지를 주고받으며, 각 클라이언트의 상태를 초기화한다. 결과 값은 (그림 15)처럼 클라이언트 프로그램에서 접속 메시지를 보낸 시간과 받은 시간의 차를 Visual C++의 Output창에 표시한다. 성능 평가의 결과는 (그림 16)과 같다.

```
    static char String[512];
    wsprintf( String,"%WhisperTo: [%s]",command->c_talkbuff);
}

void OnMessageConnectEvent(LPARAM lParam)       // CASE : 02
{
    t_ntrUnit *Unit = (t_ntrUnit *)lParam;
    AfxGetDirt()->Send(Unit);

    time = timeGetTime();
}

void OnMessageEnterEvent(LPARAM lParam)         // CASE : 04
{
    t_ntrUnit *Unit = (t_ntrUnit *)lParam;

    TRACE( "\n >>> %f ", (timeGetTime()-time) / 1000.0f );

    if( strcmp( Unit->nu_nick, AfxGetGenesis()->who_am_i) == NULL)  return;

    AfxGetDirt()->Enter(Unit);
    AfxGetGenesis()->m_NFCUnds.m_pUs
    //////
    //
    if(AfxGetDirt()->isReady == FALS
    if(AfxGetDirt()->a_Mode != DIRTM

    AfxGetDirt()->Render(\
```

>>> 0.02

(그림 15) 동시 접속 시도 사용자 수에 의한 성능 평가의 테스트 코드

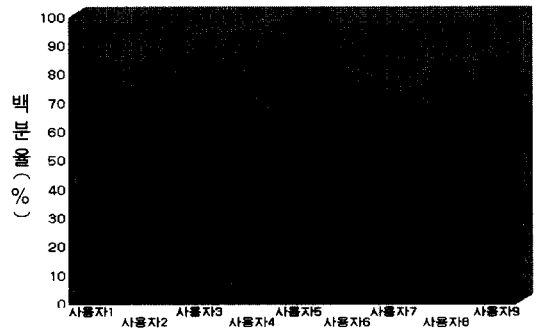


인원수(명)	10	20	30	40	50	60	70	80	90	100
속 도(초)	0.01	0.01	0.02	0.09	0.2	0.5	0.8	1	1.3	1.61

(그림 16) 동시 접속 시도 사용자 수에 의한 성능 평가

4.2.3 이동 메시지에 따른 성능 평가

일반 메시지 중 가장 부하가 큰 이동 메시지를 중심으로 한 성능 평가이다. 성능 평가 방법은 여러 명의 사용자를 두고, 데드 러커닝 알고리즘이 얼마나 신뢰도가 있는지 알아보았다. 즉, (그림 9)를 보면 중간에 if문이 있는데, 이 if문이 얼마나 실행되는지를 테스트하는 것이다. Loop()이 실행된 회수가 100퍼센트라고 하면, 결과 값은 if문이 실행되지 않을 경우의 확률이며, 이 값은 바로 메시지를 받지 않은 상태의 클라이언트들의 적중률을 나타낸다. 결과 값의 출력은 동시 접속 시도 사용자 수에 의한 성능 평가와 같은 Visual C++의 Tracer 기능을 이용했다. 이동 메시지에 따른 성능평가의 결과는 (그림 17)과 같다.



(그림 17) 이동 메시지에 의한 성능 평가

5. 결론 및 향후 연구 과제

본 논문은 네트워크 게임 제작의 핵심이라고 볼 수 있는 데드 러커닝 알고리즘의 구현을 제안하였다. 느

린 네트워크 대역폭의 한계를 벗어나서, 사용자가 마치 로컬(local)에서 게임을 진행하는 착각을 일으킬 정도로 구현하게 하는 것이 본 알고리즘의 목적이다. 네트워크 게임을 진행하기 위해서는 사용자간에 엄청난 양의 네트워크 통신이 필요하지만, 현재의 네트워크 대역폭은 필요량을 감당하기가 불가능하다. 따라서, 네트워크 통신을 최소화하고 압축되고 생략되어 있는 메시지를 기반으로 현재 상태를 예측하고, 실패했을 시에는 현재 상태를 복구하는 데드 러커닝 알고리즘이 필요하다. 본 논문에서는 이동 메시지를 중심으로 데드 러커닝 알고리즘을 구현하였고, 성능 평가도 그 부분에 주안점을 맞추었다.

따라서, 본 논문에서 제안한 알고리즘의 실질적인 성능평가 부분인 이동 메시지 적중률이 사용자에 따라 차이는 있지만, 대략 80퍼센트를 상회함을 알 수 있다. 이는 다른 네트워크 게임들이 자료 공개를 하고 있지 않아 비교를 할 수는 없지만, 상당히 만족할 만한 수치로 사료된다. 최대 사용자수 및 동시 접속 시도 사용자수의 평가 부분은 서버와 네트워크 선로의 영향을 더 받으며, 데드 러커닝 알고리즘 뿐만이 아니라, 그래픽 엔진의 성능까지 반영되므로, 정확한 평가를 내리기는 힘든 부분이며, 구현 환경은 서버는 Axil기종이고, 클라이언트는 펜티엄 II-400, 그래픽 카드는 부두밴시이다.

향후 연구 과제는 보다 더, 다양한 환경에서 성능 테스트를 해서, 본 논문에서 제안한 데드 러커닝 알고리즘의 신뢰도와 성능을 높이는 것이며, 그래픽, 시나리오, 음악 작업을 마치고, 실제 게임에 적용하는 것이다.

참 고 문 헌

[1] 이향선, "새로운 사이버 문화, 네트워크 게임", 프로그램 세계, 신영미디어, 1998. 8.

[2] Khoshafian, S. and M. Buckiewicz, Introduction to Groupware, Workflow, and Workgroup Computing, John Wiley & Sons, Inc. 1995.

[3] "멀티미디어를 지원하는 다중 사용자용 게임 엔진 개발", 정보 통신부 산·학·연 공동기술개발 사업 최종 연구 보고서, 1997.

[4] 장석원, "가상공간 충격, 머드 제작", 마이크로소프트웨어, 정보시대, 1995. 3.

[5] 한선영, "PC 네트워크에서의 파일전송 프로토콜의 설계 및 구현에 관한 연구", 정보과학회 논문지, 1998.

김 성 락

e-mail : ksr01@mail.osan-c.ac.kr

1984년 울산대학교 전자계산학과 졸업(학사)

1989년 한양대학교 산업대학원 전자계산학전공(석사)

2000년 수원대학교 대학원 전자계산학과 박사과정 수료

1995년~현재 오산대학 정보관리과 조교수

관심분야 : 분산운영체제, 네트워크보안, 인터넷응용, 전자상거래 등

윤 남 균

e-mail : nkyun@cs.suwon.ac.kr

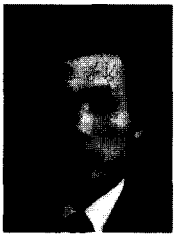
1992년 수원대학교 전자계산학과 졸업(학사)

1996년 수원대학교 대학원 전자계산학과 졸업(석사)

2000년 수원대학교 대학원 전자계산학과 박사과정 수료

1997년~현재 오산대학 정보관리과 겸임 전임강사

관심분야 : 분산운영체제, 멀티미디어, 정보통신, 인터넷응용, 전자상거래 등



구 용 완

e-mail : ywkoo@cs.suwon.ac.kr

1976년 중앙대학교 전자계산학과 졸업(학사)

1982년 중앙대학교 대학원 전자계산학과 석사과정 졸업(석사)

1988년 중앙대학교 대학원 전자계산학과 박사과정 졸업(박사)

1983년~현재 수원대학교 자연과학대학 전자계산학과 정교수, 수원대학교대학원(일반, 교육, 산업경영) 전자계산학과 주임교수, 동 대학교 전자계산소 소장

관심분야 : Operating System을 근간으로한 Distributed System 및 System Software Open System, Multimedia, Real-Time System, Computer Network, Distributed Data Base, Software Engineering, 인터넷응용, 전자상거래 등