# 모바일 환경에서의 빌보드 기법을 통한 실시간 유체 시뮬레이션 렌더링

## Realtime Fluid Simulation and Rendering Using Billboard method on Mobile Environment

우상혁, 조미리나, 박동규
창원대학교

Woo Sang-Hyuk, Jo Mirina, Park Dong-Gyu
Changwon National University

**요약**

본 논문에서는 안정적 유체 애니메이션 기법을 이용한 불과 연기 애니메이션에 대해서 기술한다. 휴대전화의 연산 성능이 발전하면서 3D 콘텐츠와 고급 어플리케이션의 구현이 가능하게 되었다. 이러한 배경을 활용하여 지금까지 PC 기반에서 주로 개발되었던 유체 시뮬레이션을 모바일 장치에서 구현하였다. 안정적 유체 애니메이션 기법을 사용한 불 시뮬레이션은 실시간 렌더링을 목표로 하였으며 외부환경과 불 시뮬레이션 사이의 상호작용을 구현하였다. 그리고 모바일 3D 게임 콘텐츠에 불 애니메이션을 결합하여 게임의 객체 혹은 배경으로 사용 가능하도록 실용성을 더하였다.

**Abstract**

This paper presents a fire and smoke animation system using stable fluid animation techniques. Stable and fast fluid simulation methods are developed in PC and console games, but fluid simulation and interactive fluid models still have many problems. We studied and implemented physics-based models for fluids like fire and smoke effects using mobile 3D system. The mobile platform of our system is WIPI, which are the standard mobile platform in Korea, also we adopted NF3D API for our 3D programming API.

## I. 서 론

최근 컴퓨터 하드웨어와 모바일 디바이스의 성능이 급격히 발전함에 따라 고급 렌더링 기술이 대중화되고 게임과 같은 실시간 렌더링 애플리케이션 분야의 연구가 활성화 되었다. 하드웨어의 발전으로 렌더링 작업에 집중되었던 계산 작업은 다른 작업에 사용할 수 있는 여유를 가져다주었으며 이에 따라 물리기반 애니메이션이 폭넓게 활용되게 되었다.

물리기반 모델링은 컴퓨터 그래픽스의 주요 연구 분야 가운데 하나로 이미 오래전에 자리 잡았으며 미분 방정식으로 표현된 운동 방정식을 수치적분을 이용하여 객체의 새로운 상태를 지속적으로 계산해 내는 방식이 일반적이다. 이러한 물리기발 모델링은 입자의 운동, 강체 시뮬레이션, 비정형 물체의 운동, 유체의 움직임을 표현하기 위하여 폭넓게 연구되어 활용되고 있다.

본 연구에서는 Stam이 제안한 Stable Fluid 기법과 모바일 환경에서의 NF3D API를 사용한 안정적인 불 애니메이션 기법에 대하여 다루고자 한다. 최근 실시간 유체 시뮬레이션 기법에 대한 연구 동향을 알아보고 이를 바탕으로 구현한 내용에 대하여 논하고자 한다.

## II. 기존 연구

유체의 움직임을 표현하기 위해 전통적으로 사용한 방법은 나비에(Navier)가 1823년 비압축성 점성유체에 관한 운동 방정식으로 내어 놓은 나비에-스토크스(Navier-Stokes) 방정식이다. 나비에-스토크스 방정식은 점성을 가지지 않은 비압축성 유체의 움직임을 기술하는 방정식으로 유체내의 속도장(velocity field)가 u이고, 밀도가 $\rho$, 압력이 $p$, 점성계수가 $\nu$, 외부 힘이 f라고 할 때 다음과 같이 표현되는 방정식이다[7].

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f} \quad (1)$$
$$\nabla \cdot \mathbf{u} = 0$$

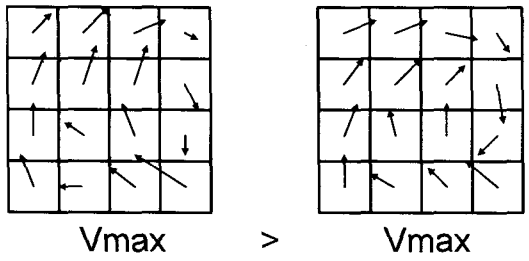
컴퓨터 그래픽스 분야의 유체 시뮬레이션이 본격적으로 시작된 시점은 1997년 Foster의 논문[1]부터이다. Foster 방법은 나비에-스토크스(Navier-Stokes) 방정식을 이용하는데 이 방정식에서 수식(1)과 같이 유체의 속도장 u를 (ux, uy, uz) 성분으로 각각 분해하여 나비에-스토크스 방정식을 각각에 대하여 이산화 하여 수치 적분을 수행하는 방법이다. 이 방법은 안정적인 시뮬레이션을 위하여 매우 짧은 시간 간격을

사용해야 했기 때문에 계산효율이 낮고 불안정성 문제가 발생하는 단점이 있다.

Foster 방법의 단점을 극복하고 나비에-스토크스 방정식을 이용한 유체 시뮬레이션을 실용적으로 수행할 수 있는 방법은 J. Stam의 Stable Fluids[2] 기법이다.

### 2.1 Stam의 Stable Fluid 기법

이 기법은 안정적인 나비에-스토크스 방정식 수치 적분(근사)방법으로 준(準) 라그랑지안(semi-Lagrangian) 기법을 사용하였는데, 그는 이 논문에서 나비에-스토크스 방정식을 구성하는 항들의 의미를 분석하여, 이류(移流)항을 semi-Lagrangian 기법으로 근사하였다. Stam의 방법은 유체의 흐름을 안정적으로 시뮬레이션 하는 기법으로 [그림 1]과 같이 이류 이후의 속도장의 최대치가 이류 이전 속도장의 최대치보다 항상 작도록 하여 이류시의 발산 문제를 해결하였다.




▶▶ 그림 1. Stam의 안정적인 시뮬레이션에서 사용된 최대 속도 문제

Stam 기법의 문제점으로는 유체내의 소용돌이를 제대로 표현하지 못하고 너무 빨리 안정화되는 결과로 인하여 소용돌이치는 유체의 형태를 표현하기가 어렵다. 따라서 소용돌이 유폐(Vorticity Confinement)를 추가적으로 다루어 사실성을 높이려는 연구가 이어졌다.

Stam의 연구에 이어 Fedkiw의 소용돌이 유폐(Vorticity Confinement) 기법 [5]은 유체 내의 소용돌이도를 찾아내고 이를 유지하는 추가의 힘을 가하는 방법이다. 이 방법은 유체의 소용돌이를 증폭하는 효과가 있으며, Stam의 기법에 비해 더욱 사실적인 유체 동작을 생성한다.

### 2.2 오일러 방식과 라그랑주 방식의 차이

유체 시뮬레이션의 두 가지 큰 분류로는 그리드 기반 방법인 오일러 방식과 파티클에 기반을 둔 라그랑주 방식이 있다. 오일러 방식은 고정된 계산 노드를 가지는 것이 특징이며, 고정된 공간 내에서 유체가 움직인다. 그리고 고정된 계산 노드는 해당 위치에서 유체가 가질 속도를 저장하고 있다. 라그랑주 방식은 계산노드가 유체를 따라 이동하며, 따라서 유체를 입자의 집합으로 표현한다. Stam의 [2] 기법과 Fedkiw의 [5] 기법은 모두 오일러 방식에 해당된다. 그런데 Stam의 방식에 semi-Lagrangian 기법이 포함되어 있다고 하는 이유는 이류(移流)항의 계산에서 Lagrangian 입자처럼 계산 노드를 다루기 때문이다. 그 외의 부분에서 Stam의 방식은 오일러 기법이며, 이류항의 계산도 고정된 그리드 각각에 대해 수행된다. 따라서 Stam의 [2] 기법은 오일러 기법이 가진 단점을 모두 가진다. Fedkiw의 [5] 논문 기법 역시 소용돌이를 증폭한다는 점을 제외하고는 이전의 오일러 기법과 동일하므로, 이 방법 역시 오일러 기법이 가진 단점을 그대로 가진다.

라그랑주 기법을 이용한 유체 시뮬레이션 기법 중에서 대표적인 논문으로는 Desbrun의 1996년 논문 [10], Hadap의 2001년 논문 [11], Müller의 2003년 논문 [12]이 있다. 이 방법은 지나치게 많은 메모리를 사용하는 Eulerian 기법의 단점을 피할 수 있다는 장점이 있으나, 이웃 입자 찾기에 많은 시간을 소비해야 하며, 비압축성을 보장하기 위해 복잡한 코드를 작성해야 한다.

### 2.3 유체 시뮬레이션의 문제점

유체와 강체의 상호작용을 다룬 Carlson의 연구에서[13] 계산 시간은 시스템 성능 Pentium4 2GHz CPU/1GB RAM을 가진 컴퓨터에서 64×64×64 그리드를 사용할 경우 1초 시뮬레이션에 필요한 시간은 약 3시간이 필요하며, 94%의 연산시간이 유체와 레벨셋 방정식을 푸는데 소모되었다. 유체 시뮬레이션에서 물리적 사실성에 중점을 두게 되면 연산량이 증가하는 문제점이 있다.

## III. 불 애니메이션을 위한 방법

### 3.1 불 애니메이션의 기초

불을 생성하기 위한 요소로는 연료, 열, 산소, 비활성가스등의 성분이 필요하다. 만일 연료가 매우 뜨겁고 충분한 산소가 공급될 경우 연료는 산소와 결합하여 불을 만들 것이다. 이 불은 열과 빛 그리고 재를 생성하여 연료가 모두 소모될 때까지 타게 될 것이다.

이를 의사코드로 나타내면 다음과 같이 표현할 수 있을 것이다. 산소와 연료 그리고 열이 각각 oxygen[], fuel[], heat[] 배열에 저장되어 있을 경우 이 값을 사용하여 다음과 같은 코드를 만들 수 있다.

```
procedure burn
for every pixel i
{
   O = oxygen[i];
   F = fuel[i];
   H = heat[i];
   reaction_rate=(O*F*H-energy_barrier)*rate_constant
   if (reaction_rate <0) reaction_rate = 0
   if (reaction_rate >maxrate) reaction_rate = maxrate

   oxygen[i] -= reaction_rate
   fuel[i] -= reaction_rate
   temp[i] += reaction_rate * Exothermicness
   if (oxygen[i] <0) oxygen[i] = 0
   if (fuel[i0fuel[i] = 0
}
end of procedure
```

연소가 일어나게 되면 이로 인하여 대류현상이 발생하게 된다. 연소가 일어난 지역의 열은 주변의 열에 비하여 매우 높기 때문에 주변 온도가 낮을 경우 열이 높은 곳의 온도는 상승하게 되고 이로 인하여 속도장의 변화가 일어나게 된다. 이러한 속도장의 변화는 다음과 같은 의사코드로 표현할 수 있다. 온도와 대류상수의 곱은 상승기류를 일으키며 이러한 상승기류는 수직 속도장에 더해지게 된다.

이때 약간의 수직 제동력을 추가하여 안정적인 시뮬레이션이 되도록 조절할 필요가 있다.

```
procedure setForces
   for every pixel i
   {
      velocity_vert[i] += temp[i] * Convectivness
      velocity_vert[i] *= velocity_damping
      velocity_horiz[i] *= velocity_damping
   }
end of procedure
```

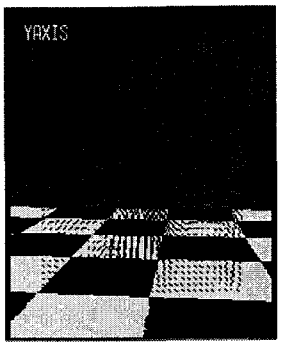
불의 색상은 매질의 온도에 의하여 결정된다. 즉 높은 온도의 불은 높은 에너지를 발생시키며, 저온의 가스는 적외선 근처의 빛의 긴 파장을 가지고 온도가 높을수록 파장은 짧아진다.

# IV. 모바일 환경의 불 애니메이션

## 4.1 모바일 플랫폼

불 시뮬레이션 결과를 모바일 환경에서 렌더링하기 위하여 WIPI 에뮬레이터와 NF3D SDK를 사용하였다. NF3D는 3D 어플리케이션 개발 환경이며, 모바일 3D 표준 API인 OpenGL ES 스펙을 기반으로 한다.

속도장, 온도장의 데이터를 저장하기 위해서는 배열이 필요하다. 그리드의 데이터를 저장하는 배열을 만들기 위해서 핸들기반으로 메모리를 사용하였다. 현재 불이 애니메이션 되는 화면 버퍼의 크기는 128*128 개의 셀로 이루어진 그리드 기반이다.

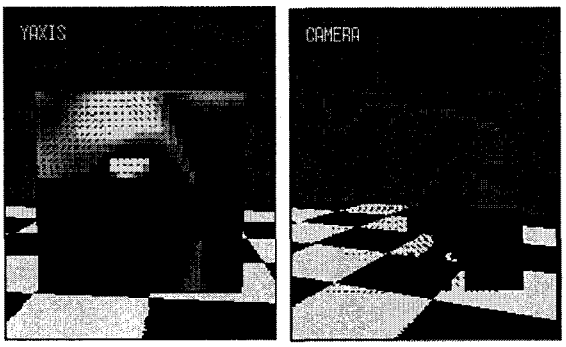



▶▶ 그림 2. 속도장을 그린 화면

## 4.2 빌보드상의 시뮬레이션

이미지 기반의 애니메이션을 구현하기 위해서 빌보드 기법을 사용하였다. 빌보드 기법이란 이미지를 항상 카메라 시점을 바라보도록 만들어 항상 같은 면을 관찰자에게 보여주는 방법이다. [그림 2]는 빌보드에 속도장을 표시한 결과이다. 그리드를 구성하는 셀의 개수는 조정가능하며 셀이 많을수록 렌더링 품질은 향상되며, 연산 비용은 증가한다. 이 경우에는 빌보드가 Y축을 기준으로 카메라의 이동이나 회전에 따라 항상 카메라에 정면이 되도록 회전한다. 속도장은 사용자가 입력한 외부 힘에 따라 변화시킬 수 있고, 변화되는 속도장의 크기와 방향을 빨간색 선의 패턴으로 알 수 있다. [그림 3]은 속도장을 따라 유체가 이동하는 모습을 그린 것이다.

빌보드에 불 시뮬레이션을 출력하기 위해서 첫 번째로 오프스크린 버퍼에 시뮬레이션에 따른 온도장의 변화를 나타낸다. 오프스크린 버퍼를 이미지 파일로 저장한 후 저장된 파일을 불러와서 빌보드에 출력이 가능하다.

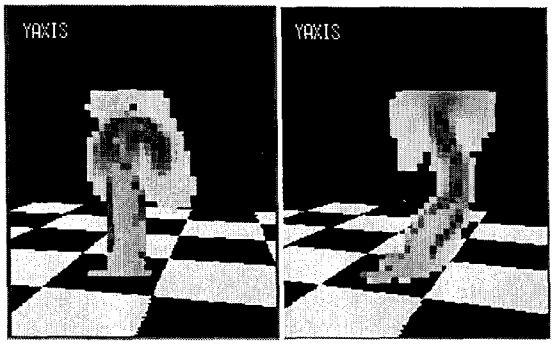



▶▶ 그림 3. 속도장에 따른 유체의 이동

Stam의 기법으로 유체 시뮬레이션 코드를 작성하면, 연기 시뮬레이션이 가능하다[7]. 이 시뮬레이션 프로그램을 응용하여 불 애니메이션을 만들기 위해서는 3.1절에서 언급한 내용과 불 시뮬레이션에 필요한 기능이 추가되어야 한다. Stam의 방법에서 밀도장을 온도장의 개념으로 바꾸어 사용했다. 그리

드의 하단부에 불을 만들고자 하는 너비만큼의 셀들에 높은 온도를 적용한다. 높은 셀의 온도는 이웃 셀로 전달되며 유체의 흐름이 생긴다. 불의 특징은 연기와 다르게 공기 중으로 떠다니지 않는다. 이러한 특징을 시뮬레이션하기 위해 냉각 인자를 추가하였다. 냉각 인자의 영향으로 이웃의 그리드 셀을 거치면서 처음 높은 온도를 적용한 셀의 온도보다는 낮은 온도가 전달된다. 온도가 낮아지면서 연기처럼 확산되는 것을 막고 불의 형태를 유지할 수 있다.

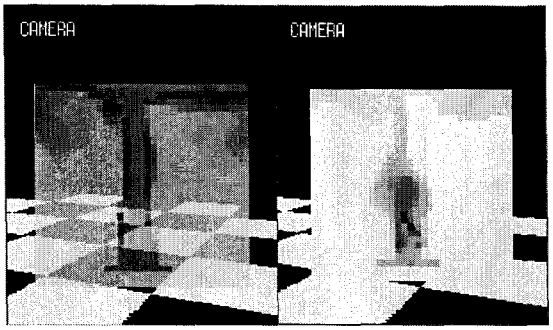
부양력을 적용하여 불과 연기는 아래에서 위로 올라가는 현상을 만들었다. 불은 아래에서 위로 향하는 힘이 지속적으로 가해지게 되며 온도가 낮은 연기는 부양력을 적용하지 않아서 옆으로 퍼지거나 하강한다.




▶▶ 그림 4. 외부 힘에 의해 움직이는 불과 연기

### 4.3 알파브렌딩기법

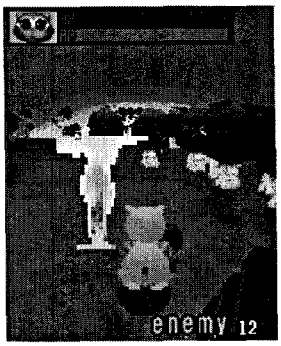
시뮬레이션 결과를 렌더링 할 때 빌보드가 3D 공간에서의 배경과 후면의 물체를 가리지 않기 위해서 투명한 속성을 가져야 한다. NF3D에서는 스프라이트의 속성을 설정하는 함수를 통해서 특정 색상을 투명으로 만들 수 있다. 빌보드 전체적인 투명도는 0부터 100까지 파라미터를 통하여 조절이 가능하다. [그림 5]는 빌보드에 다른 투명도를 설정했을 때의 모습을 보여주고 있다.




▶▶ 그림 5. 빌보드에 그려진 불 시뮬레이션과 각각 다르게 설정된 빌보드 투명도

### 4.4 불의 색상

온도에 의해 불의 색상을 결정하였다. 온도는 0부터 1사이의 값으로 되어 있으며, 255단계로 맵핑하였다. RGB 색상을 미리 설정해두고, 온도의 임계치에 따라 연기와 불을 나누어 그려주었다. 불의 색상을 결정하는 함수에서 미리 할당된 색상 사이의 값들을 선형 보간하여 그러데이션 효과를 구현하였다. 시뮬레이션 되는 불의 색상은 온도가 높은 곳에서 낮은 곳으로 갈수록 흰색, 노란색, 빨강색으로 변화된다. [그림 6]은 본 논문에서 제안한 빌보드 환경에서 구현한 불과 연기의 시뮬레이션 결과이며, [그림 7]은 빌보드에 50%의 투명도를 주어서 시뮬레이션한 결과이다




▶▶ 그림 6. 게임 콘텐츠에 적용한 불 시뮬레이션


▶▶ 그림 7. 빌보드 투명도 50% 설정

### 4.5 게임 콘텐츠와 결합

NF3D 기반으로 구현된 모바일 3D 아케이드 게임 콘텐츠 [14]에 불 시뮬레이션을 적용하였다. 4.2절에서 언급한 빌보드 기법을 사용하여 불 애니메이션을 게임 배경의 일부로 배치하였다. 사용자가 키패드 조작으로 직접 빌보드 갱신율을 조정할 수 있게 구현했다.

## V. 결론 및 향후 연구과제

지금까지 불과 연기와 같은 유체 시뮬레이션의 방법들을 살펴보고, 안정적 유체 애니메이션 기법을 통한 모바일 환경에서의 불 시뮬레이션의 구현 결과를 나타냈다. 실시간 렌더링을 목표로 구현하였으며 게임 콘텐츠에 적용했다.

향후 게임의 등장인물과 같은 객체와의 상호작용 기능이 더해지면 게임 콘텐츠에서의 불 애니메이션의 가치가 커질 것으로 기대된다. 예를 들어, 주인공이 불 옆으로 뛰어가거나 무기를 휘둘렀을 때 생성된 기류가 불 시뮬레이션에 영향을 주는 것이다. 게임 객체와 불 시뮬레이션을 그려주는 빌보드와의 거리가 멀어서 상호작용이 일어나지 않는 경우도 있다. 이런 경우에 게임 속도에 영향을 줄이는 방법은 불 시뮬레이션 버

퍼를 여러 개의 이미지 파일로 저장하여 애니메이션 하는 것이다. 미리 저장된 이미지 파일을 이용하여 애니메이션 하는 동안에는 시뮬레이션의 갱신율을 낮추거나 시뮬레이션을 정지시킬 수 있다. 그리고 불 시뮬레이션 버퍼의 갱신율을 빌보드와 카메라의 거리에 따라 변화 가능하도록 구현 예정이다. 불 시뮬레이션의 연산으로 인한 게임의 속도 저하를 최소화하기 위한 방법이다. 빌보드와 카메라 사이의 거리가 멀어지거나 혹은 화면에 그려지지 않는 상태에서는 시뮬레이션 속도와 빌보드의 갱신속도를 늦출 수 있다.

많은 데이터를 처리해야 하는 그리드 방식의 유체 시뮬레이션 특성과 모바일 장치의 성능문제로 속도와 렌더링 품질에서 미흡한 점이 있다. 앞으로는 속도와 렌더링 품질에서 모바일에 최적화된 방법을 찾는 연구가 필요하다.

## ▌참 고 문 헌▐


[1] N. Foster and D. Metaxas. Modeling the Motion of a Hot, Turbulent Gas. In Proc. of SIGGRAPH 2001, pp.23-30, 2001.

[2] J. Stam. Stable Fluids. In Proc. of SIGGRAPH 99, pp.121-128, 1999.

[3] Mark Jason Harris. Real-Time Cloud Simulation and Rendering. Ph.D Dissertation, Chapel Hill, 2003.

[4] B. E Feldman, J. F. O'Brien, B. M. Klingner and T. G. Goktekin. Fluids in Deforming Meshes, In Proc. of Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2005, pp.255-259, 2005.

[5] R. Fedkiw, J. Stam, and H. Jensen. Visual Simulation of Smoke. In Proc. of ACM SIGGRAPH 2001, pp.15-22, 2001.

[6] Zeki Melek, John Keyser. Interactive Simulation of Fire. 10th Pacific Conference on Computer Graphics and Applications(PG'02), 2002.

[7] J. Stam. Real-Time Fluid Dynamics for Games. In Proc. of the Game Developer Conf. Mar. 2003.

[8] J. Stam. Interacting with Smoke and Fire in Real-Time. Communications of the ACM, Volume 43, Issue 7, pp.76-83, 2000.

[9] J. Stam. A Simple Fluid Solver based on the FFT, Journal of Graphics Tools Volume 6, Number 2, pp. 43-52, 2001.

[10] Desbrun, M., and Cani, Marie-Paule. Smoothed particles: A new paradigm for animating highly deformable bodies. Eurographics Workshop on Computer Animation and Simulation (EGCAS), pp.61 - 76. 1996.

[11] S. Hadap, and N. Magnenat-Thalmann, Modelling Dynamic Hair as a Continuum. Computer Graphics Forum, Vol.20, No.3, pp.329-338, 2001.

[12] M. Müller, D. Charypar, M. Gross, Particle-Based Fluid Simulation for Interactive Applications, in Proceedings of ACM SIGGRAPH Symposium on Computer Animation (SCA) 2003, pp.154-159, 2003.

[13] Mark Carlson, Peter J. Mucha, Greg Turk. Rigid Fluid: Animating the Interplay Between Rigid Bodies and Fluid, Proc. of SIGGRAPH 2004, pp.377-384, 2004.

[14] 우상혁, 박보영, 조미리나, 박동규. 모바일 환경의 3D 아케이드 게임 구현, 한국멀티미디어학회 2006년도 춘계학술 발표대회논문집, 제9권 제1호, pp.190-193, 2006.