사실적 연기 시뮬레이션을 위한 Semi-Lagrange 방법에서의 이류항 계산방법 개선

박 수 완[†] · 장 문 희[††] · 김 은 주[†††] · 유 관 우[††††]

요 약

자연 현상에서 나타나는 연기나 난류의 움직임을 사실적으로 시뮬레이션하기 위해서는 Navier-Stokes 방정식을 사용할 수 있다. 이 방정식을 이용한 구현은 방대한 연산량과 계산의 복잡성으로 인하여 실시간 시뮬레이션이 어렵다. 실시간 처리를 위해서는 Navier-Stokes 방정식의 근사 형태를 사용하는 것이 일반적이다. 유체 시뮬레이션의 이류(advect) 과정을 근사화하기 위해, Semi-Lagrangian 방법을 이용하면, 연기 시뮬레이션의 경우는 시간이 지남에 따라 밀도가 현저히 줄어들고, 소규모의 소용돌이(small-scale vorticity) 현상 등을 표현하기가 어렵다. 본 논문에서는 이 문제를 해결하기 위해 이류항(advection term)을 계산하는 새로운 수치해석 방법을 제안한다. 이 방법에서는 이류항의 값을 구할 때, 격자(grid) 중심의 현재 속도에 비례하는 임계영역을 격자 주변에 설정하고, 임계영역 내에 있는 격자들 중에서 현재 격자의 위치로 이류하는 속도를 가진 격자를 추적하여, 그 격자에서의 속도를 현재 격자의 이류속도 벡터로 사용한다. 이는 밀도와 소용돌이 현상의 수치적 소실을 줄여서, 사실성을 높이면서도, 실시간 처리가 가능하다. 본 논문에서는 GPU 구현을 통해 벡터 연산 등의 효율성을 높임으로써, 제안하는 방법의 실시간이 가능함을 보인다.

키워드 : 컴퓨터 그래픽스, 유체 애니메이션, 난류, Semi-Lagrange, 이류항

Improved Calculation of the Advection Term in the Semi-Lagrange Method for Realistic Smoke Simulation

Su Wan Park[†] · Mun Hee Chang[††] · Eun Ju Ki[†††] · Kwan Woo Ryu[††††]

ABSTRACT

In the field of computer graphics, Navier-Stokes equations would be used for realistic simulations of smokes and currents. However, implementations derived from these equations are hard to achieve for real-time simulations, mainly due to its massive and complex calculations. Thus, there have been various attempts to approximate these equations for real-time simulation of smokes and others. When the advection terms of the equations are approximated by the Semi-Lagrange methods, the fluid density can be rapidly reduced and small-scale vorticity phenomena are easy to be missed, mainly due to the numerical losses over time. In this paper, we propose an improved numerical method to approximately calculate the advection terms, and thus eliminate these problems. To calculate the advection terms, our method starts to set critical regions around the target grid points. Then, among the grid points in a specific critical region, we search for a grid point which will be advected to the target grid point, and use the velocity of this grid point as its advection vector. This method would reduce the numerical losses in the calculation of densities and vorticity phenomena, and finally can implement more realistic smoke simulations. We also improve the overall efficiency of vector calculations and related operations through GPU-based implementation techniques, and thus finally achieve the real-time simulation.

Key Words : Computer Graphics, Fluid Animation, Turbulence, Semi-Lagrange, Advection

1. 서 론

컴퓨터 그래픽스 분야에서는 특수효과에 사용하기 위해 연기, 물, 불과 같은 유체의 사실적인 움직임을 표현하는 기술에 관한 연구가 활발하다. 또한 계산유체역학(CFD, Computational Fluid Dynamics)분야에서는 이미 오래전부터 다양한 상황에서 유체의 흐름(유동)을 사실적으로 시뮬레이션 하기 위한 연구가 있어 왔다. 하지만 유동을 해석하기 위한 이론과 풀이기법들은 그래픽스 분야에 직접 적용하기는 곤란하다. 그래픽스 분야에서는 사실성 뿐만 아니라 계산에 소요되는 시간 또한 고려해야 할 중요한 요소인데 반해, 정확성과 정밀성을 중요시하는 계산유체역학 기법들은

※ 이 논문은 부분적으로 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(R05-2004-000-12641-0). 또한 이 논문은 부분적으로 교육인적자원부 2단계 두뇌한국(BK) 21사업의 지원을 받아 수행된 연구임.
† 준 회 원 : 경북대학교 컴퓨터공학과 박사과정(교신 저자)
†† 정 회 원 : 경북대학교 컴퓨터공학과 석사과정
††† 정 회 원 : 동명정보대학교 정보통신공학과 전임강사
†††† 정 회 원 : 경북대학교 컴퓨터공학과 교수
논문접수 : 2007년 1월 12일, 심사완료 : 2007년 7월 9일

계산량이 방대하여 처리하는 시간이 많이 소요되기 때문이다. 따라서 그래픽스 분야에서는 컴퓨터의 성능이 향상된 후부터 CFD의 이론을 그래픽스 분야에 적용할 수 있게 되었고, 약간의 물리적 사실성을 희생하더라도 시각적인 사실성을 유지하는 범위 내에서 효율성을 높이도록 하였다. 이렇듯 컴퓨터 그래픽스 분야에서는 유체의 사실적 시뮬레이션을 위해 유체역학에 기반을 둔 연구가 진행되어 왔다. Foster와 Metaxas는 3차원 유체의 움직임을 시뮬레이션하기 위해 Navier-Stokes 방정식을 유한 차분법(finite difference method)을 이용하여 계산하였다[1]. 하지만 이 방법은 명시적 솔버(explicit solver)를 사용하였기 때문에 상대적으로 긴 시간 간격에 대해서는 시스템이 불안정하여 깨어지는 현상이 발생하였고, 대화형이고 실시간이 되는 유체 시뮬레이터에는 사용할 수가 없었다.

Stam은 안정성이 있으며 긴 시간 간격으로 시뮬레이션 가능한 Semi-Lagrangian 방법을 제안하였다[2]. 이 방법은 Navier-Stokes 방정식을 풀 때 이류항(advection term)을 계산하는 부분을 근사화하는 암시적 방법(implicit method)이다. 이 방법은 격자의 이류항을 구할 때, 격자의 현재 속도를 가지고 역 추적해서 추적한 위치의 격자로부터 속도를 가져오는 것이다. 이 방법을 적용했던 많은 연구들이 좋은 결과를 보여주었다[3,4,5]. 하지만 Stam의 방식은 실제 유체와 비교해서 빨리 소멸하고 소규모의 와도(소용돌이)와 같은 물리량의 수치적 소실이 커서 연기의 섬세하고 역동적인 움직임을 다루는데 한계가 있었다.

이 문제는 Fedkiw의 논문[6]에서 개선이 이루어졌다. 이 논문은 계산량이 많은 점성 Navier-Stokes 방정식 대신 가스 표현에 적합한 비점성의 유체표현을 위한 Euler식을 사용함으로써 기존 방법들보다 좀 더 넓은 간격의 격자에서 빠르고 효율적으로 연기를 표현할 수 있는 수치적 방법을 제안하였다. 이는 성긴 격자에서 표현하기 어려운 작은 크기의 회전 특성을 모델링하기 위해 와도 제한(vorticity confinement)항을 추가함으로써 이전 Stam의 논문에서 발생한 문제점을 해결하였다. 이 방법은 Stam의 Semi-Lagrangian 방법으로 이류항을 계산할 때 와도 제한 성분을 추가함으로 수치적 소실을 보정하려는 것이다. 따라서 Semi-Lagrangian 방법 자체가 가지고 있는 수치적 소실의 문제를 해결하지는 못했다.

Hong[7]은 연기의 와도를 유지하기 위해 이류항을 구하는 과정에서 발생하는 오차를 보정해서 수치적 소실을 줄이고, 와도의 이류를 첨가해서 모델링했다. 이는 Stam의 Semi-Lagrangian 방법에다가 2차적인 정확도를 높이고 와도가 이류함으로써 기존의 방법들보다 훨씬 사실성 있는 애니메이션을 보였다. 그러나 이 방법 또한 Semi-Lagrangian 방법을 개선하기는 했으나 수치적 소실 문제를 해결하지는 못했다. Semi-Lagrangian 방법은 Euler 방법으로 모델링(유체를 격자 형태로 모델링)하지만 각 격자의 이류항의 값을 계산할 때는 Lagrange 방법(유체를 파티클로 모델링)인 것처럼 유선형(streamline)을 가정해서 구한다. 반면에, 역추적 방법을 통해서 이류항을 구할 때 현재 속도로 역추적하기 때문에 현재 속도가 0에 가까운 값을 가진 격자에서는 이류항의 값이 무조건 0이 되거나 물리량이 격자 자신의 값이 된다. 또한 현재 속도로 역추적해서 찾은 위치의 속도나 물리량이 0인 경우엔 소실이 더 커진다. 따라서 Hong의 방법으로 2차 정확도를 높일 지라도 다음과 같은 문제로 인한 소실은 피할 수가 없다. 이는 속도에서 뿐만 아니라 밀도 등의 다른 물리량의 이류에 더 많은 영향을 끼친다.

본 논문에서는 이 문제를 해결하기 위해 이류항을 계산할 때 수치적 소실을 줄이는 새로운 방법을 제안한다. 이류항의 값을 구할 때, 현재 격자에 임계 영역(critical region)을 설정한 후 임계영역 내에 있는 격자들 중에서 다음 단계에서 현재 격자의 위치로 오는 속도를 가진 격자의 위치를 추적하여, 그 격자의 속도를 이류 속도 벡터로 적용한다. 현재 격자의 속도를 이용해서 이류할 속도를 역추적하지 않기 때문에 제안한 방법은 밀도와 소용돌이(와도, vorticity) 현상의 수치적 소실을 줄여서 사실성을 높인다. 이 방법은 암시적 방법(implicit method)이어서 상대적으로 긴 시간 간격(time step)에서도 안정적이다.

본 논문은 개선된 이류항 계산법을 적용하여 연기를 시뮬레이션한다. 연기는 비점성 유체이므로 Euler 식을 사용하며 이류항을 계산하는 부분과 와도 제한항을 첨가하는 것을 제외하고는 기존의 Stam이 한 방식과 동일하게 구현한다. 즉, 비압축성 기체를 Navier-Stokes 수식기반으로 외부힘, 이류, 와도 제한힘 등을 적용하여 각 격자마다 반복적으로 수치연산을 수행한다. 이 과정들은 벡터 등의 반복 연산이 많으므로 GPU를 사용할 경우 속도가 향상될 수 있다[8]. 본 논문은 속도 개선을 위해서 GPU 상에서 구현한다. 이는 소규모의 와도가 많은 난류(turbulence) 시뮬레이션에서 보다 정확하고 사실성 있는 시뮬레이션이 가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 기본적인 유체 시뮬레이터 모델을 기술하고, 3장에서는 소실을 줄이는 이류항 계산을 기술한다. 4장에서는 실험 결과 및 분석을 기술하고, 5장에서는 결론 및 향후 연구 과제를 기술한다.

2. 기본적인 유체 시뮬레이터 모델

본 논문에서는 비압축성, 무점성의 기체를 가정하며, 이러한 기체의 시뮬레이션을 위해 Navier-Stokes 방정식을 사용한다. 격자의 중심에 속도와 밀도 값을 저장하며 격자 모델과 시뮬레이션 과정은 Stam의 논문[2]에서 사용한 방식을 따르며, 여기에 Fedkiw가 제안한 방법[6]으로 와도 제한을 추가하여 소용돌이를 표현한다.

2.1 Navier-Stokes 방정식

속도 벡터장을 $\mathbf{u}$ 로 나타낼 때, Navier-Stokes 방정식은 다음 식과 같다.

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

$$\nabla \cdot \mathbf{u} = 0 \tag{2}$$

여기서 $\mathbf{f}$는 중력이나 부력과 같은 외력이고, p는 속도장에서의 압력이며, ρ는 밀도이다. ν는 점성계수이고, 식 (2)는 비압축성일 때 유체의 질량이 보존되는 것을 의미한다. 무점성인 경우 Navier-Stokes 방정식은 확산항이 0인 무점성의 Euler 방정식이 되어 다음 식이 되며, 본 논문에서는 이 방정식을 사용한다.

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{\nabla p}{\rho} + \mathbf{f} \tag{3}$$

Navier-Stokes 방정식을 푸는 방법을 더 쉽게 하기 위해서 일반적인 유체 시뮬레이션 처리과정을 생각해 보면 다음과 같다[2].

$$P \circ F \circ D \circ A$$

여기서 A는 이류항을 구하는 과정, D는 확산항을 구하는 과정, F는 외부 힘을 구하는 과정, P는 정사형항을 구하는 과정이다. 이 과정들이 반복되면서 유체 움직임의 시뮬레이션을 한다. 본 논문에서는 무점성 기체를 구현하기 때문에 점성과 관련된 확산항은 고려하지 않는다.

2.2 와도 제한 (Vorticity Confinement)

하수구에서 물이 급격히 빠지거나 난류가 흘러갈 때, 유체가 물의 진행방향과 수직이 되는 방향으로 회전하는 것을 볼 수 있다. 이렇게 회전하는 유동을 와도(소용돌이, vorticity)라고 한다. 유체를 Euler 방법으로 성긴 격자에서 시뮬레이션하면 와도 중에서 소규모의 소용돌이는 소실이 되는데, 와도제한을 이용하면 작은 소용돌이 특성을 모델링 할 수 있다[6]. 와도 제한은 와도를 형성하는 힘을 생성한다. 연기의 움직임을 속도 $\mathbf{u}$로 표현하면, 와도 ω는 다음과 같이 계산된다[9].

$$\omega = \nabla \times \mathbf{u} \tag{4}$$

와도 ω는 속도장을 맴도는(spin) 회전의 중심축이다. 와도의 x, y 성분은 속도에 컬(cull)을 취해서 얻는다. 여기서 정규화된 벡터장 $\mathbf{N}$을 얻을 수 있다.

$$\mathbf{N} = \frac{\eta}{|\eta|} (\eta = \nabla |\omega|) \tag{5}$$

식 (4)와 식 (5)로부터 시간 간격마다 속도장을 회전하는데 필요한 힘의 방향과 크기를 결정할 수 있다. 와도제한 힘(confinement force)은 다음과 같이 계산할 수 있다.

$$\mathbf{f}_{conf} = \epsilon \mathbf{h}(\mathbf{N} \times w) \tag{6}$$

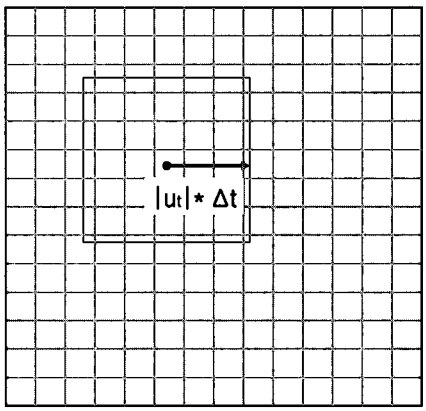
ϵ은 와도 제한 힘을 제어하는데 사용되는 상수 값이며, 식 (6)은 연기의 소용돌이치는 특징을 강조하는 힘을 제공한다.

2.3 GPU를 이용한 유체 시뮬레이션

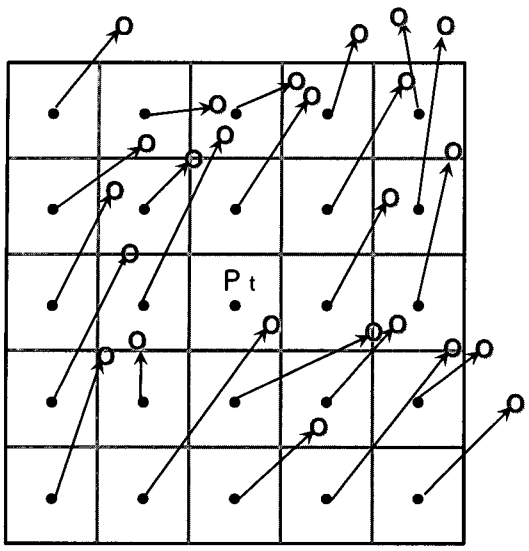
GPU는 그래픽 작업에 최적화 되어 있다. 따라서 물리기반 유체 시뮬레이션을 GPU로 구현하여 성능을 향상시키기 위해서는 그래픽 하드웨어 구조에 대한 정확한 이해가 필요하다. GPU 프로그래밍은 정점 프로그램(vertex program)과 단편 프로그램(fragment program)으로 나눈다. 본 논문은 격자 기반 모델링이기 때문에 단편 프로그램을 사용한다. GPU를 이용한 계산에서는 데이터를 텍스처(texture)로 표현한다. 이런 텍스처는 배열(array)로써 여겨질 수 있으며 배열의 인덱스 계산은 GPU 계산에서 텍스쳐 좌표 매핑에 해당한다. 현재 GPU에서 지원하는 텍스처는 유체 시뮬레이션할 때 필요한 기본 연산을 모두 지원하고 있다. 기본 자료구조형은 벡터이며 2~4개의 성분을 갖는다. 본 논문에서는 속도, 압력, 와도를 텍스처로 표현하여 연산하며, 메모리에 있는 데이터를 읽는 기본 연산은 텍스처 검색(texture lookup)을 이용한다. 계산결과의 피드백을 위해서는 OpenGL의 pbuffer와 ping-pong기법을 사용한다[10,11].

3. 소실을 줄이는 이류항 계산

현재 속도 $\mathbf{u}_t$를 이용하여 이류항을 구하는 Semi-Lagrangian 방법은 현재 속도가 0일 경우, 실제와는 달리 이류항이 0이 된다. 즉 역추적 방법을 통해서 이류항을 구할 때 현재 속도로 역추적하기 때문에 현재 속도가 0인 격자에서는 이류항의 속도가 무조건 0이 되거나 물리량이 격자 자신의 값이 된다. 또한 현재 속도로 역추적해서 찾은 위치의 속도나 물리량이 0인 경우엔 갑작스러운 값의 변화로 소실이 더 커진다. 이 문제를 해결하기 위해 본 논문에서는 현재 격자점 주변에 임계영역을 설정해서 임계영역 내에 있는 격자점 중에서 다음 단계에 현재 격자점으로 이류할 속도를 가진 격자를 찾아 그 격자의 속도를 이류항의 속도 벡터로 적용한다. 다음 단계에서 현재 격자점으로 이류하는 속도 $\mathbf{u}_t^*$는,



(그림 1) 임계영역



(그림 2) 각 격자의 물리량이 Δt시간 후에 이류에 의해 이동할 위치

먼저 임계 영역을 설정하고, 임계영역 내에서 이류할 속도가 있는 격자를 찾은 다음, 오차를 보정하여 이류할 속도를 계산하는 세 과정으로 나누어서 구한다.

첫 번째로 임계영역을 설정하는 과정은 이류항을 구하고자 하는 격자의 현재 속력에 비례하는 정사각형 모양의 임계영역을 (그림 1)과 같이 설정한다. 각 격자의 현재 속도의 크기 $|\mathbf{u}_t|$에 시간간격 Δt를 곱한 값을 l이라 두자.

$$l = |\mathbf{u}_t| \Delta t \tag{7}$$

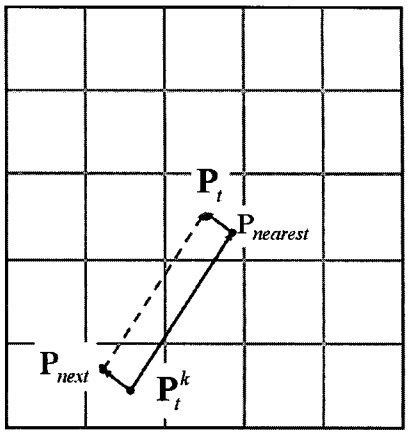
한 변의 길이가 $2l$이고 이류항을 구하고자 하는 격자의 중심점이 임계 영역의 중심에 오도록 정사각형의 임계영역을 설정한다(3차원의 경우엔 정육면체가 된다). 구하고자 하는 현재 격자의 속도 값이 작아서 주변을 찾는 범위가 좁아지거나 너무 커져서 검색 범위가 많아지는 것을 막기 위해서, 아래와 같이 l 값의 최대 최소값을 설정한다.

$$l = \alpha \Delta x \quad \text{if} \quad l < \alpha \Delta x \tag{8}$$

$$l = \beta \Delta x \quad \text{if} \quad l > \beta \Delta x \tag{9}$$

여기서 Δx는 한 격자의 간격이고, α와 β는 사용자가 지정하는 최소와 최대 임계영역을 위한 상수이다.

이렇게 임계영역의 범위를 지정한 후, 현재 격자로 이류할 속도를 가진 격자를 찾는 두 번째 과정으로 넘어간다. 두 번째 과정에서는 임계영역 내에 있는 격자들의 속도를 조사해서 현재 구하고자 하는 위치로 이류할 속도 $\mathbf{u}_t^*$을 갖고 있는 격자를 찾아낸다. (그림 2)는 임계 영역 내에 있는 각 격자들이 자신의 속도로 Δt시간 후에 각 격자의 물리량이 이류에 의해 이동할 위치들을 나타낸다.



(그림 3) 찾는 위치의 오차를 보정

현재 이류항을 구하고자 하는 격자의 가운데 위치를 $\mathbf{P}_t$라고 하고, 임계영역 내에 있는 i번째 격자의 중심점의 위치를 $\mathbf{P}_t^i$, i번째 격자의 속도를 $\mathbf{u}_t^i$이라고 두면, i번째 격자의 중심점에 있는 물리량이 Δt시간 후에 이류하여 도착하는 위치는 $\tilde{\mathbf{P}}_{t+1}^i$이 된다. 이는 아래와 같이 계산할 수 있다.

$$\tilde{\mathbf{P}}_{t+1}^i = \mathbf{P}_t^i + \mathbf{u}_t^i \Delta t \tag{10}$$

$\tilde{\mathbf{P}}_{t+1}^i$ 중에 현재 위치 $\mathbf{P}_t$와의 거리가 가장 가까운 점 $\mathbf{P}_{nearest}$을 찾는다(그림 3). 여기서 $\mathbf{P}_t^k$는 이류해서 $\mathbf{P}_{nearest}$에 도달할 속도를 가지고 있는 격자의 가운데 위치이다.

마지막 과정에서는 오차를 보정하여 이류할 속도를 계산한다. 각 격자의 속도가 모두 격자의 중간 위치에서 정의되어 있고, 이 속도를 기반으로 찾기 때문에 (그림 3)과 같이, 오차를 $\overrightarrow{\text{err}}$ 라고 두면, $\overrightarrow{\mathbf{P}_{nearest}\mathbf{P}_t}$ 만큼 오차가 발생한다.

$$\overrightarrow{\text{err}} = \overrightarrow{\mathbf{P}_{\neq arest}\mathbf{P}_t} \tag{11}$$

오차를 보정하기 위해서 찾은 격자의 위치 벡터인 $\mathbf{P}_t^k$ 에 $\overrightarrow{\text{err}}$ 를 더한다.

$$\mathbf{P}_{next} = \mathbf{P}_t^k + \overrightarrow{\text{err}} \tag{12}$$

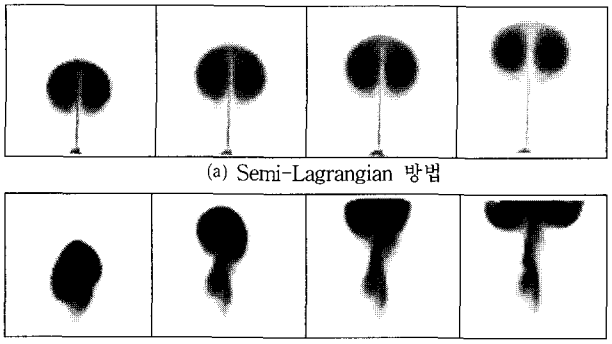
결과적으로 $\mathbf{P}_{next}$ 위치에 있는 속도가 이류항의 속도 $\mathbf{u}_t^*$이 된다. 제안한 방법은 주변속도를 이용해서 이류할 속도를 미리 예측하는 암시적 방법이므로 상대적으로 긴 시간간격에서도 안정적인 시뮬레이션이 가능하다.

4. 실험 결과 및 분석

본 논문에서 제시하는 방법을 사용하여 Pentium IV 3.0Hz, 1G 메모리, geforce 5700(128M) 그래픽카드 상에서

C++, Cg, OpenGL을 이용하여 실험하였다. 격자수는 64×64이며 밀폐된 곳에서 일정량의 기체를 쏘아 올렸을 때의 동작을 Stam의 Semi-Lagrangian 방법과 제안하는 방법을 비교하여 표현했다. 시간간격 Δt를 1, 1.5로 두고 와도제한 상수 ϵ을 변형하여 비교를 한 결과는 (그림 4, 5, 6, 7)에 제시되어 있다. 밀폐된 공간에 있는 연기의 밀도는 계속 유지되어야 하지만 시간이 지남에 따라 Stam의 Semi-Lagrangian 방법을 사용한 결과 밀도의 소실이 있음을 보인다[(그림 4-(a), 5-(a))]. 이러한 밀도의 소실은 와도제한 상수가 작을 때 더 크다. 반면에 같은 힘과 같은 밀도의 연기로 시작해서 제안한 방법을 사용한 결과 연기의 밀도가 거의 소실되지 않는 것을 알 수 있다[(그림 4-(b), 5-(b))]. Stam의 Semi-Lagrangian 방법과 제안한 방법 모두 큰 규모의 와도는 잘 나타난다. 하지만 Stam의 Semi-Lagrangian 방법일 경우 소규모의 와도는 급격히 소실되는 반면에 제안한 방법은 소규모의 와도가 잘 유지되기 때문에 역동적이고 섬세한 연기의 표현이 가능했다. 또한 시간 간격을 늘려도 기존의 방법과 같은 안정성을 보였다[(그림 6, 7)].

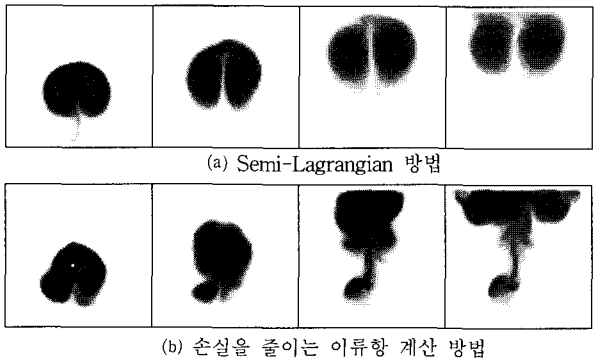
실험 결과 모든 시뮬레이션의 속도는 60 FPS 이상이었다. 제안한 방법의 연산 과정은 Stam의 Semi-Lagrangian 방식보다 다소 추가된 부분이 있지만 추가된 계산량은 거의 상수 값에 가깝다. 폭발 등의 큰 속도 값을 가지지 않는다면 한 격자의 속도가 만들어내는 임계 영역은 주변의 몇 격자를 포함하지 않을 것이고, 임계 영역의 최대 구간폭을 사용자가 결정할 수 있기 때문이다. 또한 GPU를 사용하기 때문에 연산의 효율성을 기할 수 있기 때문이다.



(a) Semi-Lagrangian 방법

(b) 손실을 줄이는 이류항 계산 방법

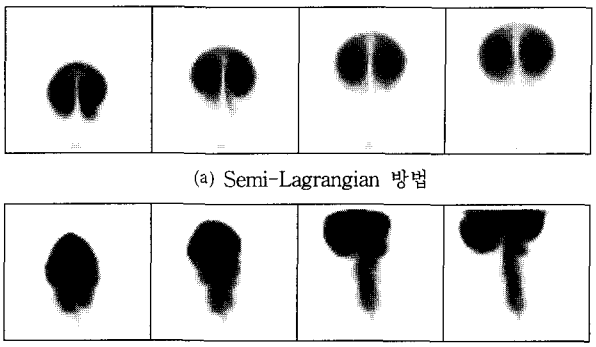
(그림 4) $\Delta t = 1$ $\epsilon = 0.10$ 일 때 시뮬레이션 결과



(a) Semi-Lagrangian 방법

(b) 손실을 줄이는 이류항 계산 방법

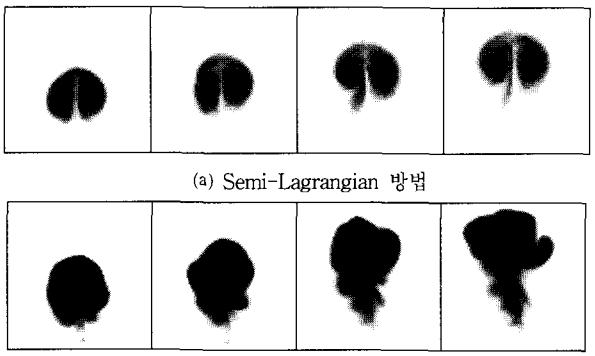
(그림 5) $\Delta t = 1$ $\epsilon = 0.155$ 일 때 시뮬레이션 결과



(a) Semi-Lagrangian 방법

(b) 손실을 줄이는 이류항 계산 방법

(그림 6) $\Delta t = 1.5$ $\epsilon = 0.10$ 일 때 시뮬레이션 결과



(a) Semi-Lagrangian 방법

(b) 손실을 줄이는 이류항 계산 방법

(그림 7) $\Delta t = 1.5$ $\epsilon = 0.155$ 일 때 시뮬레이션 결과

5. 결론 및 향후 연구 과제

본 논문에서는 연기의 움직임을 사실적으로 표현하기 위한 Semi-Lagrange 방법에서 이류항 계산의 개선을 통한 수치적 방법을 제안했다. 물리 기반 유체 시뮬레이션에서 많이 사용하는 비압축성 Navier-Stokes 방정식을 이용하여 구현할 경우 기존의 방법은 이류항 계산을 근사화 하는 과정에서 물리량의 수치적 손실로 인해 섬세하고 역동적인 연기의 동작을 표현할 수 없었다. 특히 기존의 방법은 시간이 지남에 따라 밀도와 소규모의 와도가 너무 빨리 사라지는 문제점이 있었다. 본 논문에서는 이 문제점을 개선하기 위해 이류항 계산을 세 단계로 나누어서 좀 더 정밀한 계산을 했다. 본 논문의 방법은 연기의 이류 과정에서 격자 주변의 임계영역을 설정해서 임계영역내에 있는 격자들 중 현재의 위치로 이류할 속도를 가진 격자를 찾아내어, 그 격자의 속도나 물리량을 이류항의 값으로 사용하는 새로운 방법을 제안했다. 그 결과 밀도의 경우에 소실량도 적으며 소규모의 와도가 보존되어 역동적인 연기를 세밀하게 표현할 수 있었다. 또 이 방법은 암시적 방법이어서 긴 시간 간격에서도 안정적인 시뮬레이션이 가능하였다. 또한 GPU를 이용하여 구현함으로써 다소 추가된 계산 부분이 있을지라도 종래와 같은 속도를 유지하였다.

본 논문에서 제안한 방법은 폭발과 같이 유속의 증폭이 심한 유체를 제외한 시뮬레이션에서는 격자수를 확장하더라

도 임계영역 내의 검색하는 격자의 수는 많지 않아서 격자 수를 확장하여 유체 시뮬레이터에 활용할 수 있다. 또한 본 논문은 특히 소규모의 와도를 역동적으로 표현할 수 있기 때문에 난류(turbulence) 시뮬레이션에도 유용하게 사용할 수 있을 것이다. 그러나 폭발과 같이 유속이 급격히 빨라지는 경우에는 임계영역의 확장으로 인한 속도 저하를 가져올 수 있다. 따라서 향후에는 이 부분에 관한 연구가 진행되어야 한다.

참 고 문 헌

[1] N. Foster and D. Metaxas, "Modeling the Motion of Hot, Turbulent Gas," ACM SIGGRAPH, pp.181-188, 1997.

[2] J. Stam, "Stable fluids", ACM SIGGRAPH 1999, pp.121-128, 1999.

[3] N. Foster and R. Fedkiw, "Practical Animation of Liquid", ACM SIGGRAPH, pp.23-30, 2001.

[4] A. Treuille, A. McNamara, Z. Popovic, and J. Stam, "Keyframe Control of Smoke Simulations", ACM SIGGRAPH, pp.716-723, 2003.

[5] M. Carlson, P. J. Mucha and G. Turk "Rigid Fluid: Animating the Interplay Between Rigid Bodies and Fluid", ACM SIGGRAPH, pp.377-384, 2004.

[6] R. Fedkiw, J. Stam, and H. W. Jensen, "Visual simulation of smoke", ACM SIGGRAPH, pp.15-22, 2001.

[7] J. Hong, J. Kim, "Animating smoke with dynamic balance", Computer Animation and Virtual Worlds 16, pp.405-414, 2005.

[8] R. Fernando(Ed.), *GPU Gems*, Addison Wesley, 2004.

[9] J. Steinhoff and D. Underhill, "Modification of the Euler equations for vorticity confinement", Application to the computation of interacting vortex rings. Physics of Fluids 6(8), pp.2738-2744, 1994.

[10] GPGPU : http://www.GPGPU.org

[11] M. Pharr(Ed.), *GPUGems 2*, Addison Wesley, 2005.

박 수 완

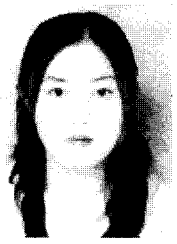
e-mail : jpsuwan@naver.com

1999년 경북대학교 유전공학과(학사)

2002년 경북대학교 컴퓨터공학과 (공학석사)

2002년~현재 경북대학교 대학원 컴퓨터공학과 박사과정

관심분야 : 컴퓨터 그래픽스, 알고리즘



장 문 희

e-mail : munhee3@daum.net

2002년 경북대학교 물리학과(학사)

2007년 경북대학교 컴퓨터공학과 (공학석사)

2007년~현재 도담시스템스 영상개발팀 근무

관심분야 : 그래픽스, 물리기반모델링, IG 등

김 은 주

e-mail : ejkim@tu.ac.kr

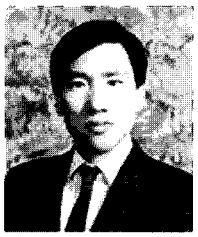
1984년 경북대학교 전자공학과(학사)

1986년 경북대학교 전자공학과(공학석사)

2003년 경북대학교 컴퓨터공학과 (공학박사)

2000년~현재 동명대학교 정보통신대학 정보통신공학과 전임강사

관심분야 : 컴퓨터 그래픽스, 멀티패러다임 프로그래밍, 병렬알고리즘 등



유 관 우

e-mail : kwryu@knu.ac.kr

1980년 경북대학교 전자공학과(학사)

1982년 한국과학기술원 전산학과(공학석사)

1990년 메릴랜드대학교 전산공학(공학박사)

1982년~현재 경북대학교 컴퓨터공학과 교수

관심분야 : 컴퓨터 그래픽스, 계산기하학, 계산 이론