기계학습 기반 유체 시뮬레이션의 비말 검출 알고리즘

김재형 [1], 성수경 [1], 신병석 [1*]
[1] 인하대학교 전기컴퓨터공학과

wogudkim@gmail.com, rebirth87@inha.edu, bsshin@inha.ac.kr

Splash Detection Algorithm for Machine Learning-based Fluid Simulation

Jae-Hyeong Kim[1], Su-Kyung Sung[1], Byeong-Seok Shin[1*]
[1]Dept. of Electrical and Computer Engineering, Inha University

요 약

인공지능 기술의 발전에 따라 유체 시뮬레이션 분야에서는 복잡한 액체의 흐름을 모사하기 위해 기계학습 기술이 많이 활용되고 있다. 이러한 시뮬레이션에서 성능 향상의 가장 중요한 요소는 학습 데이터다. 이 논문에서는 기계학습 기반 유체 시뮬레이션의 학습 데이터 생성 단계 중 기존의 방법보다 효율적으로 비말(splash) 탐색하는 방법을 제안한다. 기존 방법에서는 CPU 환경에서 큐(queue)를 이용하는 너비우선탐색(breadth first search) 기법을 사용하기 때문에 처리속도가 느리다. 반면에 제안하는 기법에서는 배열로 되어 있는 해시 테이블(hash table)을 이용해 충돌 문제를 해결해 GPU 환경에서 비말을 신속하게 검출하도록 하기 때문에 빠른 학습 데이터 생성이 가능하도록 했다. 이 알고리즘의 유효성을 확인하기 위하여 정확성과 수행시간을 확인하였다.

1. 서론

최근에 유체 시뮬레이션 분야에서는 복잡한 액체의 흐름을 모사하기 위해 기계 학습 기술을 많이 사용한다. 특히 비말 현상의 모델링은 게임 및 영화에서 특수 효과를 만드는 등 다양한 분야에서 중요한 역할을 한다. 단순히 물결만 표시하는 것보다 급격한 유체의 이동에 따른 비말을 표시함으로써 현실감 있는 유체의 움직임을 표현할 수 있다.

기존의 비말 현상을 모델링하는 방법들은 계산 비용이 높아 처리 속도가 느리며, 복잡한 환경에서 현실적인 모사가 어렵다. 그러나 최근 심층신경망을 이용하여 비말을 예측하기 위한 효율적이고 정확한 모델을 개발하는 것이 가능하다.

이러한 모델링을 위해서는 많은 양의 학습 데이터가 필요하기 때문에 빠른 속도로 학습 데이터를 생성해야 한다. 따라서 본 논문에서는 비말 현상을 판단하는 알고리즘에 대한 연구를 수행하며, 기존보다 더욱 효율적인 알고리즘을 제안한다.

2. 관련 연구

기존의 유체 역학 시뮬레이션 방법론 중 대표적인 SPH (Smoothed Particle Hydrodynamics) [1]는 유체 내부의 속도, 밀도의 물리적인 양을 이산적인 입자(particle)로 모사하여 시뮬레이션을 수행한다. 그러나 SPH 는 순간적인 불안정성과 부적절한 양자화의 한계로 인해 높은 해상도와 그에 대한 정확한 모사가 어렵다. J. U. Brackbill et al. (1988) [1]이 제안한 FLIP (Fluid Implicit Particle)은 SPH 의 입자 기반 접근 방식을 사용하면서도, 유체 역학의 연속성을 보장하는 Particle-In-Grid 방식을 사용하여 더 정확한 모사가 가능하다.

Kiwon Um et al. (2018) [2]이 제안한 MLFLIP (Machine Learning FLIP)은 유체의 비말 현상을 모델링하기 위한 기계학습 기반의 유체 시뮬레이션 방법론이다. MLFLIP 은 FLIP 기반의 유체 시뮬레이션에서 비말 현상에 대한 속도 변화를 학습하여 기존의 물리 시뮬레이션에서는 고려하지 못하는 부분까지 예측이 가능해 물리적 모델링의 한계를 극복할 수 있다. MLFLIP 은 기계학습 기반이기 때문에 비말 현상에 대한 FLIP 시뮬레이션 데이터를 필요로 하는데 CPU 환경에서 BFS(Breadth-First Search) 알고리즘을 활용하여 직렬 연산으로 비말 현상을 판단하기 때문에 해상도가 높아지거나 데이터 수가 늘어날수록 효율이 떨어진다. MLFLIP 이 학습하기 위해서는 충분한 학습 데이터를 생성해야 하기 때문에 본 논문에서는 높은 해상도에서도 보다 효율적으로 학습 데이터를 생성할 수 있는 비말 탐색 알고리즘을 제안한다.

3. 비말 탐색 알고리즘

비말을 판단하는 기준은 입자가 들어있는 격자의 연속된 개수다. 기존 방법은 CPU 환경에서 BFS 알고리즘으로 비말 탐색을 하기 때문에 격자의 크기가 $N \times N$ 일 때 시간 복잡도는 $O(N^2)$이다. GPU 환경에서 해당 알고리즘을 수행하게 되면 d 개의 스레드에서 동시에 연산하기 때문에 시간 복잡도는 $O(N^2/d)$가 되는데 데이터 간 충돌이 발생하고 그래프의 구조에 따라 스레드 당 처리하는 큐의 크기 달라 부하불균형이 발생할 수 있다. [4]

우리의 알고리즘은 이러한 문제를 해결하기 위해 해시 테이블을 이용하여 최솟값을 저장하는 방식을 활용하기 때문에 GPU 환경에서 효율적인 탐색이 가능하다. 또한, 시뮬레이션의 해상도가 낮은 경우 그래프의 크기가 작아 병렬화로 인해 발생하는 오버헤드 비용으로 인해 직렬 연산보다 느릴 수 있어 두 연산 환경에서 모두 활용할 수 있도록 알고리즘을 구성했다.

```
1:  procedure detectSplashParticle():
2:      Initialize Grid to -1
3:      for each particle i:
4:          base ← Grid coordinate include X_i
5:          G_base ← min(X_i, G_base)
6:      end for
7:      setGroupNum()
8:      Update Hash Table
9:      Update Grid with Hash
10:     for each particle i:
11:         if number of G include X_i < 5:
12:             T_i = Splash
13:     end for
14: end procedure
```

<알고리즘 1> 비말 탐색

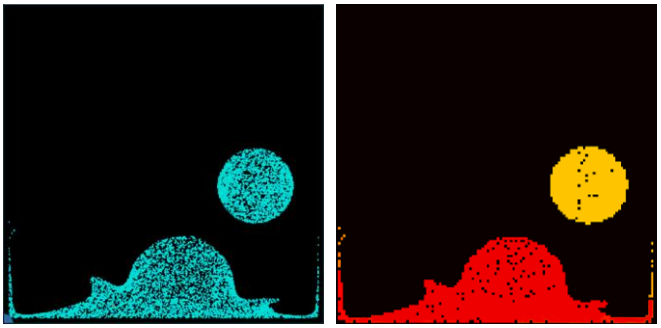
비말 현상을 탐색하는 방법은 알고리즘 1 에 자세히 나와있다. 첫번째로, 2 차원 배열 G 에 대해서 입자 X_i 가 있는 격자에 X_i 중 가장 작은 값을 저장한다. (2 ~ 6 번 줄). 다음으로 연속적인 입자 X_i 가 있는 격자에 같은 번호를 부여한다(7 번 줄).

```
1:  procedure setGroupNum():
2:      for each Grid i, j include particle:
3:      // i', j' is Up, Down, Left, Right
4:          G_{i,j} ← min(G_{i,j}, G_{i',j'})
5:          if G_{i',j'} is not G_{i,j}:
6:              if Hash(G_{i',j'}) is first update:
7:                  Hash(G_{i',j'}) ← G_{i,j}
8:              else:
9:                  Update Hash Table
10:         G_{i',j'} ← Hash(G_{i',j'})
11:     end for
12: end procedure
```

<알고리즘 2> 그룹번호 저장

번호를 부여하는 과정은 알고리즘 2 에 자세하게 기술되어 있다. 먼저, $G_{i,j}$ 와 해당 좌표의 상, 하, 좌, 우를 비교하여 그룹의 최솟값을 $G_{i,j}$ 에 저장한다(2 ~ 4 번 줄). 두 번째로 상, 하, 좌, 우 좌표에 대해서 해시 테이블에 처음 등록되는 값이라면 $G_{i,j}$ 값을 저장해주고, 이미 등록되어 있는 값이라면 더 작은 값으로 업데이트해준다 (5 ~ 9 번 줄). 다음으로 해시테이블에 저장된 값으로 $G_{i',j'}$을 업데이트해준다 (10 번 줄).

이후 해시테이블을 한 번 더 업데이트해서 격자에 다시 저장해준다(알고리즘 1 의 8 ~ 9 번 줄).

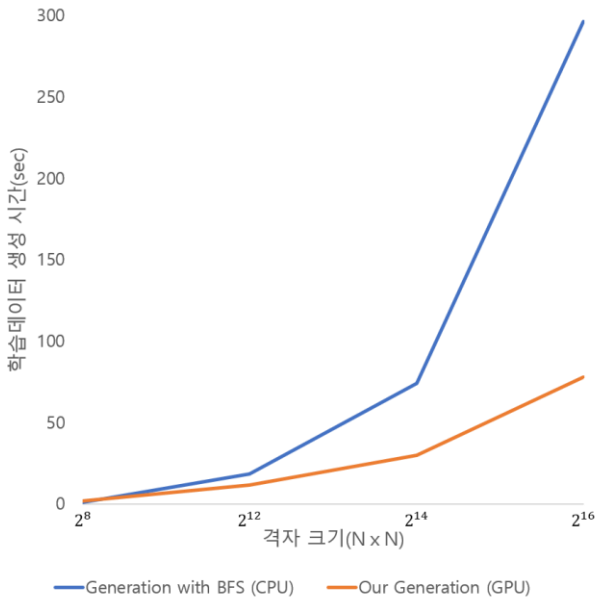


(그림 1) 시뮬레이션에 대한 알고리즘 적용 결과. (a) 입자 단위로 표현된 시뮬레이션의 한 장면. (b) 격자 단위로 그룹번호에 따라 색을 구분한 장면.

해당 단계까지 진행하면 그림 1(a)에 대하여 그림 1(b)처럼 입자 X_i가 들어있는 격자를 그룹 별로 구분이 된다. 마지막으로 입자 X_i가 들어있는 격자의 그룹의 크기를 확인하여 5 미만이라면 입자의 타입 T_i을 비말로 저장해준다. (알고리즘 1 의 10 ~ 13 번 줄)

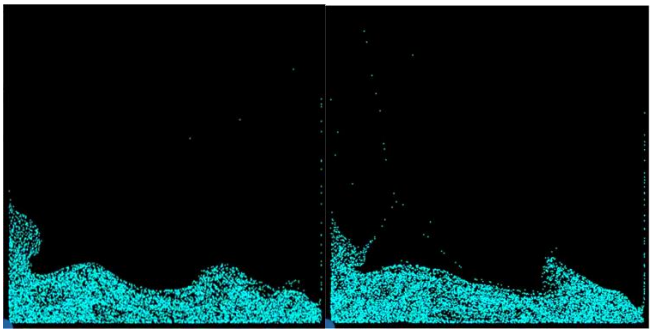
4. 실험 결과

본 논문에서의 실험은 유체의 입자의 개수는 9000 개, 격자의 크기는 128×128 로 600 frame 동안의 시뮬레이션으로 설정하였고, CPU 는 Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz, GPU 는 GeForce RTX 3090 를 사용하였다.



(그림 2) 학습데이터 생성시간 비교

Kiwon Um et al. (2018) [2]가 제안했던 BFS 방식의 큐의 탐색 시간이 본 논문에서 제안한 해시 테이블의 탐색 시간보다 오래 걸려 그림 2 와 같은 차이가 발생하였다. MLFLIP 은 기계학습 기반으로 성능을 향상시키기 위해서 더 많은 학습 데이터를 필요로 하기 때문에 학습 데이터 수를 늘릴수록 학습 데이터 생성 수행시간의 차이는 더 많이 차이가 발생하게 된다.



(그림 3) 시뮬레이션 결과 비교. (a)기존 물리 시뮬레이션 결과. (b) 기계학습 기반의 시뮬레이션 결과.

비말 탐색 알고리즘을 사용하여 비말 부분인 입자를 구분하고 해당 입자의 속도변화에 대한 학습 데이터를 생성한다. 생성한 학습 데이터로 MLFLIP 을 학습하면 새로운 시뮬레이션을 만들 때 학습한 모델이 입자의 비말 현상을 판단하고 속도변화를 적용해주면 그림 3(a)과 그림 3(b)의 차이처럼 보다 현실적인 유체의 흐름을 얻을 수 있다. 본 논문의 비말 탐색 알고리즘을 이용했을 때 기존의 MLFLIP 과 같은 결과를 도출하였으며 이를 통해 해당 알고리즘의 정확성을 확인했다.

5. 결론

본 논문은 비말 현상에 대한 기계학습 기반 유체 시뮬레이션에서 학습 데이터 생성 단계 중 비말 탐색에 대하여 기존의 알고리즘보다 효율적으로 탐색하는 방법을 제안한다. 기존 CPU 환경의 BFS 방식의 탐색 알고리즘은 GPU 환경에서 수행 시 문제가 발생한다. 본 논문은 GPU 환경에서 탐색할 수 있도록 해시 테이블을 이용하여 비말 탐색 알고리즘을 적용하였다. 그 결과, 기존의 방식보다 빠른 수행시간으로 학습 데이터 생성을 하였고, 높은 해상도의 학습 데이터를 만들 때 그 차이가 크게 발생하였다.

추가적으로, 큐가 아닌 배열 형태의 해시테이블을 이용하기 때문에 CPU 와 GPU 환경에 상관없이 활용이 가능하여 범용성과 확장성 부분을 기대할 수 있다.

사사

본 논문은 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No. RS-2022-00155915, 인공지능융합혁신인재양성(인하대학교)).

참고문헌

[1] R. A. Gingold, J. J. Monaghan “Smoothed Particle Hydrodynamics: theory and application to non-spherical stars” Monthly Notices of the Royal Astronomical Society, vol. 181, no.3, pp. 375~389, 1977

[2] J. U. Brackbill, D. B. Kothe, H. M. Ruppel “Flip: A low-dissipation, particle-in-cell method for fluid flow” Computer Physics Communications, vol. 48, no. 1, pp. 25~28, 1988

[3] Kiwon Um, Xiangyu Hu, Nils Thuerey “Liquid Splash Modeling with Neural Networks” Computer Graphcis Forum, vol. 37, no. 8, pp. 171~182, December 2018

[4] Aydin Buluc, Kamesh Madduri “Parallel breadth-first search on distributed memory systems” SC ’11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Washington, Seattle, November 2011, pp. 1~12