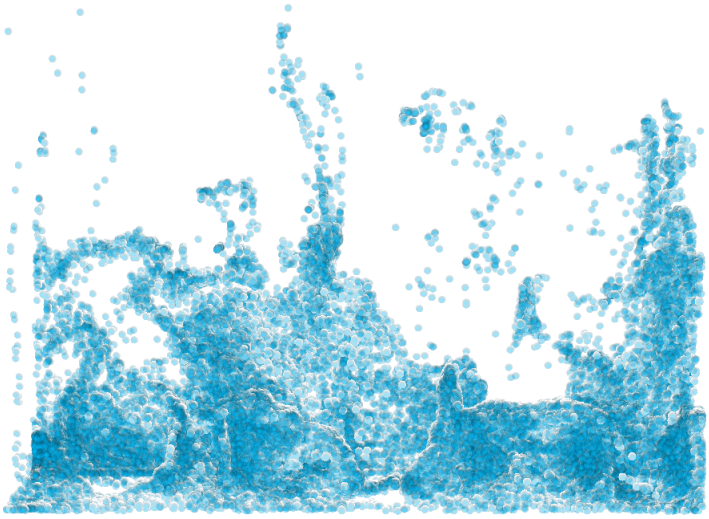
Real-Time Fluid Simulation

장동수
2020-23657, ECE



1. Abstract

애니메이션이나 게임에서 액체나 기체의 사실적인 움직임을 표현하기 위해서 실시간 유체 시뮬레이션은 중요하다. 본 프로젝트에서는 유체 시뮬레이션에 대한 전반적인 방법에 대해 간략하게 알아보고 Semi-Lagrangian 방법을 사용하여 고체벽으로 둘러쌓인 유체를 시뮬레이션을 한다. 그리고, screen space fluid rendering 방법을 통해 사실적이고 효율적인 실시간 렌더링을 구현한다.

구체적으로 본 레포트에서 저자의 기여는 1)Fluid simulation 전반 대한 공부, 2) Github[1] 의 코드를 사용하여 fluid simulation을 구현, 3)기존 코드 병렬화를 통해 속도 가속 구현, 4)Screen space rendering 구현으로 요약된다.

2. Fluid Simulation

유체 시뮬레이션을 하는 방식은 크게 2가지로 나뉘는데, 관점에 따라 Lagrangian 접근법과 Eulerian 접근법으로 분리된다. Lagrangian 접근법에서는 유체를 입자의 집합으로 나타내고 각 입자의 위치와 속력을 추적한다. 반면에, Eulerian 접근법에서는 임의의 시간과 위치에 대한 유체의 지나가는 속도 $u(x,t)$ 벡터장을 구한다. Eulerian 접근법은 그리드를 통해 위치에 대한 그래디언트 값을 계산하기 쉽기 때문에, 본 프로젝트에서는 Eulerian 접근법에 기반하고 Lagrangian 방법을 차용하는 Particles in cell (PIC) 방법을 채택하여 시뮬레이션을 수행한다.

시뮬레이션시 시간과 공간을 불연속적으로 이산화한다. 시간의 경우, 초기 상태에서 $\triangle\, t$ 초 단위로 끊어서 다음 프레임의 유체의 상태 구한다. Eulerian 접근법에서는 공간을 grid로 나누어 다음 상태에 대해 갱신한다.

압축 불가능하고 점성이 없는 유체의 속도는 Navier Stokes 방정식에서 점성항을 제외한 다음과 같은 Euler Equation이라는 편미분방정식을 따른다.

$$\frac{\partial u}{\partial t} + u \cdot \nabla u + \frac{1}{\rho}\nabla p = g \tag{1}$$

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

위 방정식에서 u, t, ρ, p, g 는 각각 유체의 속도, 시간, 유체의 밀도, 유체의 압력, 중력을 의미한다. (1)은 Momentum Equation, (2)는 Incompressibility condition이라고 하며, 각각 운동방정식과 비압축성을 의미한다.Euler 방정식의 해석적 해는 아직 난제로 남아있다. 그래서 유체시뮬레이션 시, splitting이라는 방법을 이용하여 방정식을 3개의 방정식으로 나눈 후, 차례대로 수치적으로 근사하여 방정식을 푼다. Splitting을 적용하면 Euler equation은 다음과 같은 방정식들로 나뉜다.

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = 0 \tag{3}$$

$$\frac{\partial u}{\partial t} = g \tag{4}$$

$$\frac{\partial u}{\partial t} + \frac{1}{\rho}\nabla p = 0 \tag{5}$$

(3), (4), (5) 방정식은 각각 advection, body force (gravity), pressure equation을 의미한다. (4) 와 (5)의 방정식은 그리드로 공간을 나누어 수치적으로 방정식을 근사하고, (3)의 방정식은 다음에 나올 입자들을 이용해서 적용한다.

다음은 PIC 방법에 관한 개괄적인 설명이다. 이 방법은 입자 (marker particles) 들의 집합을 이용해 shape을 나타낸다. 이 입자들은 위치와 속도를 가지고 있고, 그리드에 (interpolation을 통하여) 위치와 속도 정보를 넘겨준다. 그리드에서 위에서 구한 Euler equation을 풀어 다음 time step 의 유체와 속도에 대한 정보를 구한다. 그 후 그리드에서 다시 입자에게 속도를 전달하고 입자들의 위치와 속도를 갱신한다.

정리하면 전체 알고리즘은 다음과 같다.

```
Initialize particles with initial state
Until end of simulation:
    Move velocities and positions from particles to grid다
    Solve body force equation (gravity)
    Solve pressure equation
    Move velocities from grid to particles
    Advect particles according to particle velocities
```

위 알고리즘의 각 단계에서는 grid나 입자의 방정식을 풀 때, grid 끼리나 입자끼리의 dependency는 없다. 그러므로 병렬화가 되어도 문제가 없기 때문에, base code에서는 구현 안된 병렬화를 저자가 openMP를 이용하여 적용하였다. 이로 인해 훨씬 많은 입자들로도 딜레이가 없이 렌더링이 잘 되었다.

3. Screen Space Rendering

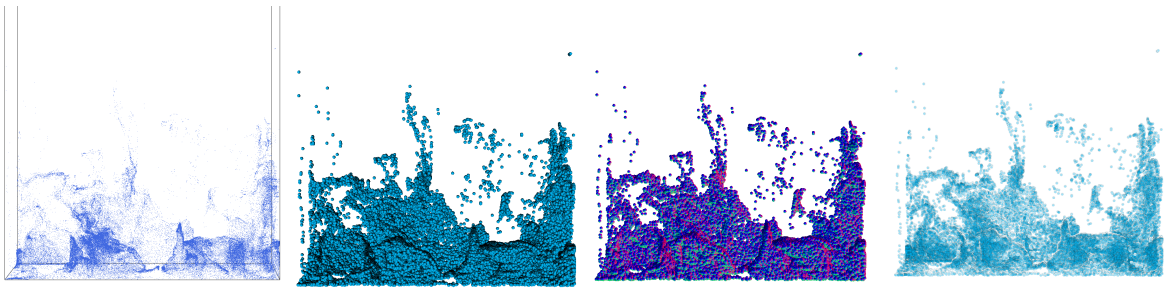


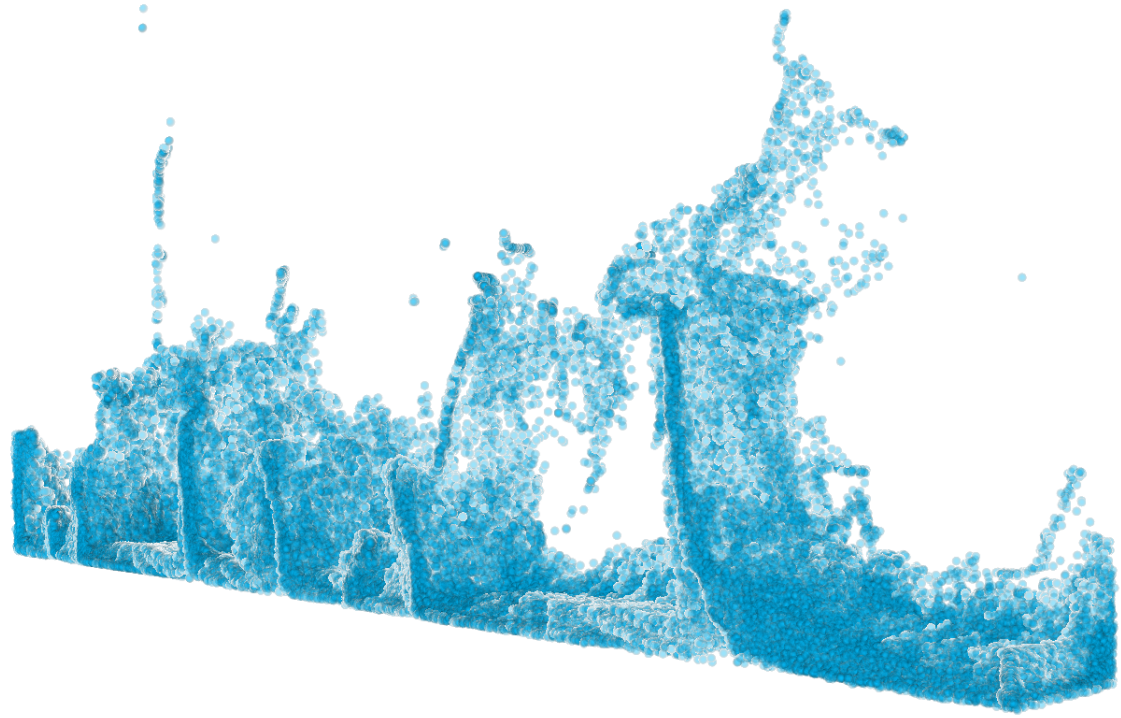
Figure 2. Screen Space Rendering. 왼쪽에서 오른쪽 순으로 입자 이미지, screen space rendering을 이용한 입자를 구 이미지, 구의 normal 이미지, 최종 결과물 이미지

2장의 Fluid simulation을 통해 particle들의 위치를 구하면 Figure 2의 가장 왼쪽과 같이 입자들의 집합으로 렌더링이 된다. 하지만, 입자들은 부피감이 없고 투명하지 않아 실제 액체와 유사하지 않다. 이 장에서는 [4]의 screen space rendering을 참고하여 부피감과 투명도를 넣어서 Figure 2의 가장 오른쪽과 같이 액체와 같게 만드는 방법에 대해 설명한다.

부피감을 넣는 가장 손쉬운 방법은 각 입자마다 입자의 위치를 중심으로 반지름이 일정한 구를 렌더링하는 것이다. 하지만, 이 방법은 입자의 수가 많아졌을 때 렌더링시 소요되는 시간이 크기에 매우 비효율적이다. Screen space rendering 방식은 구를 평면에 projection하면 원이라는 아주 쉬운 기하학적 성질을 이용하여 효율적으로 구를 렌더링한다.

Fragment shader 에서 각 입자의 위치를 vertex shader로 부터 받을 때, `gl_PointCoord`를 이용하면 [0, 1] x [0, 1]의 texture 정보 까지 같이 정보를 받을 수 있다. 이 때, `gl_PointCoord`가 원점을 중심으로 하는 원의 반지름 안에 있을 때만 렌더링하면 Figure 2의 왼쪽에서 2번째 사진과 같이 효율적으로 각 입자의 위치를 중심으로 하는 구를 렌더링 할 수 있다. 구를 렌더링 할 수 있으면 Figure 2 의 왼쪽에서 세번째 사진과 같이 normal 도 쉽게 구할 수 있다. normal을 알면 빛을 이용해서 diffuse shading을 입힐 수 있고, transparency를 추가하면 Figure 4의 가장 오른쪽 사진과 같이 우리가 원하는 최종 결과물이 나타난다.

실시간 유체 렌더링은 첨부된 동영상 2개에서 확인 할 수 있다. 두 동영상은 입자의 개수가 다른데, 입자의 개수가 많을 수록 딜레이는 심하지만 더 실사적인 유체 시뮬레이션이 되는 것을 확인할 수 있었다.



4. Conclusion

본 프로젝트를 통해, 실시간 유체 시뮬레이션을 살펴보고 구현하였다. 유체 시뮬레이션의 핵심은 Navier-stokes equation을 푸는 것이다. 하지만 이 방정식 하나를 푸는 것 자체가 어렵기 때문에 여러가지 테크닉을 써서 근사를 한다. 실시간 유체 시뮬레이션을 위해서는 효율적인 구현이 요구되어 screen space rendering을 이용하였다.

위에서 살펴보았듯이 분량 관계상 생략을 많이 해도 fluid simulation은 매우 복잡하다. 특히 레포트에 언급 안된 부분 중 수치적인 오차를 줄이기 위한 방법들이 많았는데 그 방법도 매우 복잡하다. Floating error를 잡기 위한 여러가지 테크닉도 있었는데, 이 역시 매우 복잡했다. 그래픽스 분야는 예쁘지만 어려운 것 같다.

5. References

1. Alex Sommer, 3D Real-Time Fluid Simulation, Master Project, 2018, https://github.com/a1ex90/Fluids3D
2. Robert Bridson, Fluid Simulation for Computer Graphics, Second Edition. A K Peters / CRC Press, 2015.
3. 장필식, Semi-Lagrangian과 PIC 방법을 사용한 그리드 기반 유체 애니메이션의 구현과 비교, 서울대학교 졸업논문, 2019, https://github.com/lasagnaphil/fluid-sim
4. Simon Green, Screen space fluid rendering for games, Game developers converence (GDC), 2010, http://developer.download.nvidia.com/presentations/2010/gdc/Direct3D_Effects.pdf