



유체역학 엔진 개발기

게임에서 물리 기반의 유체를 구현하기 위한 고군분투 삽질

넥슨 코리아

옥찬호

발표자 소개

옥찬호 (Chris Ohk)

- Nexon Korea Game Programmer
- Microsoft VSDT MVP
(Visual Studio and Development Technologies)
- 페이스북 그룹 C++ Korea 대표
- IT 전문서 집필 및 번역 다수
 - 게임샐러드로 코드 한 줄 없이 게임 만들기 (2013)
 - 유니티 Shader와 Effect 제작 (2014)
 - 2D 게임 프로그래밍 (2014)
 - 러스트 핵심 노트 (2017)
 - 모던 C++ 입문 (2017)





2012년

NDC 서포터즈

2015년

넥슨 입사

2018년

NDC 발표

2021년

?

안내

이 발표는 개인 연구 프로젝트입니다.

개발하면서 겪었던 시행착오들과 고민들 위주로 설명하기에 시간 관계상
유체 시뮬레이션을 구현하는 방법에 대한 자세한 내용은 다루지 않습니다.
또한 수학과 관련된 내용도 최대한 다루지 않고 코드 위주로 다룹니다.
(차후에 Bonus 슬라이드나 다른 발표 자료로 다루려고 합니다.)

목차

1. 들어가며
2. 유체역학이란?
3. 유체역학 엔진 개발기
4. 마치며

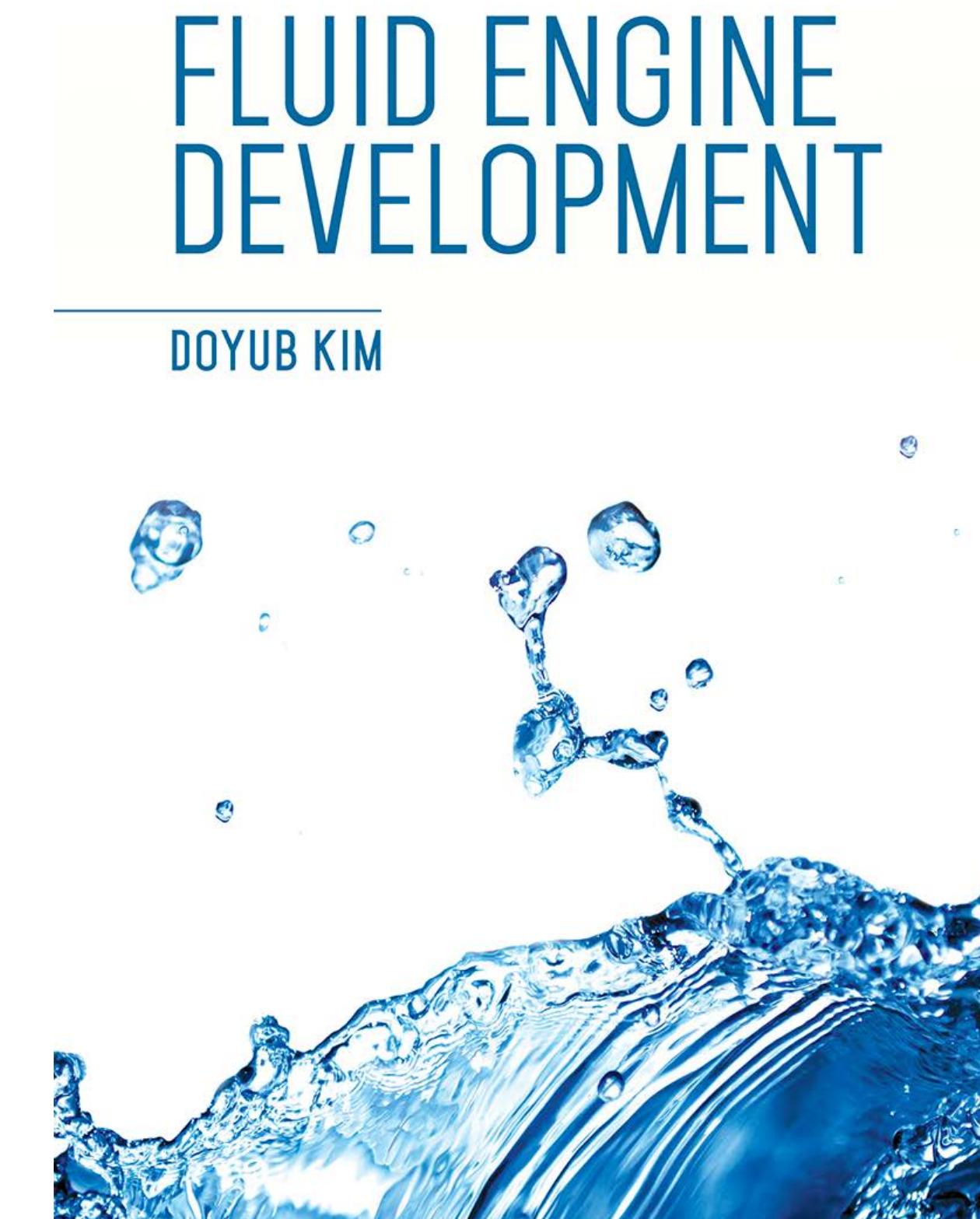
1. 들어가며

- 감사의 말
- 유체역학 엔진을 만들게 된 계기
- 유체역학을 활용해 만든 게임, 애니메이션

감사의 말

- “Fluid Engine Development”의 저자, 김도엽 님
(<http://fluidenginedevelopment.org/>)
 - CubbyFlow 프로젝트의 기반이 된 Jet Framework
(<https://github.com/doyubkim/fluid-engine-dev>)
 - 데모 동영상
 - <https://www.youtube.com/watch?v=EKrqvBaLMyA>
 - <https://www.youtube.com/watch?v=SqyrddEASVQ>

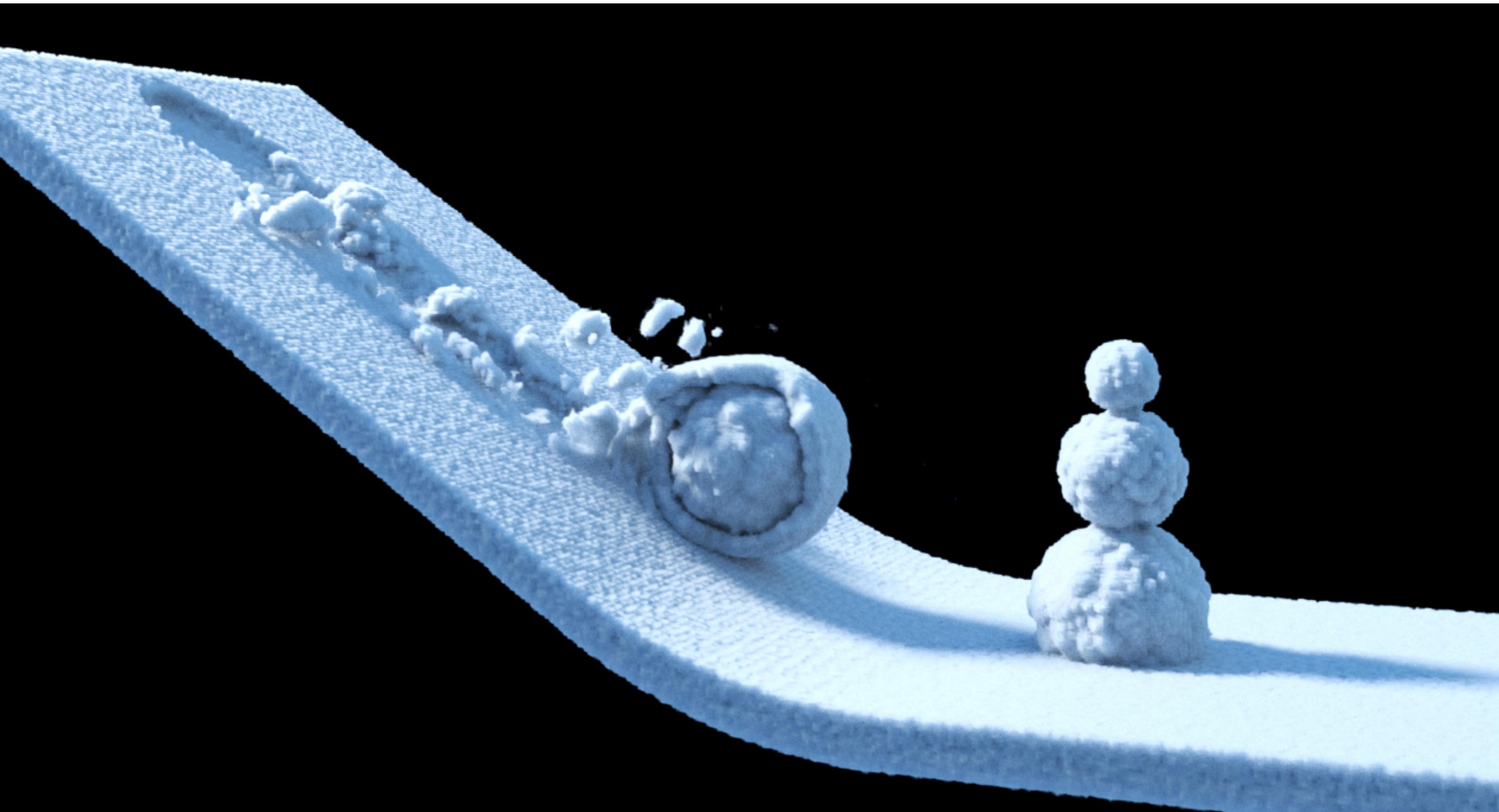
CRC CRC Press
Taylor & Francis Group



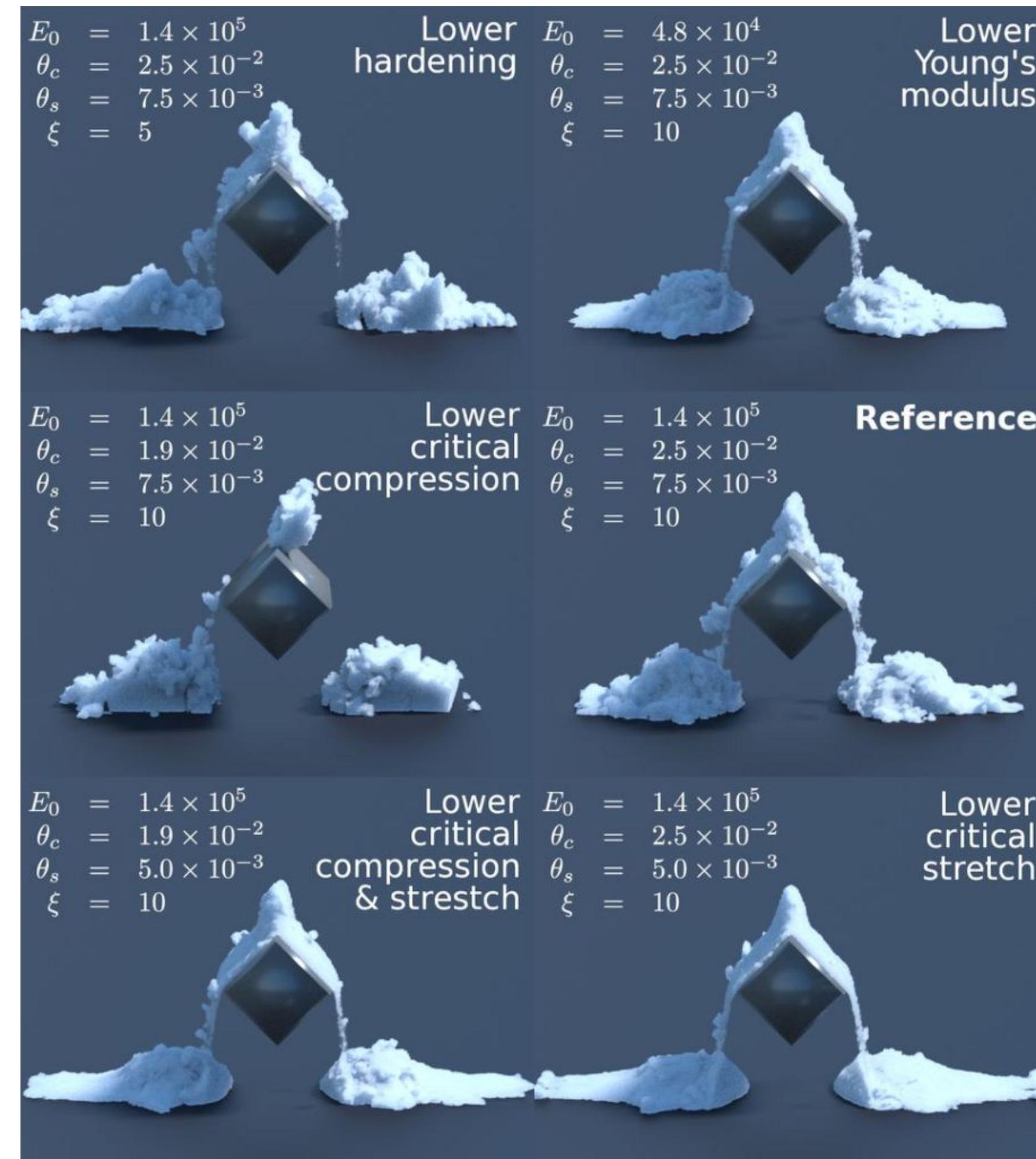
감사의 말

- CubbyFlow 프로젝트를 같이 만들어 나가는
 - 서울대학교 김동민 학생
(<https://github.com/dmk98>)
 - 한양대학교 김영중 학생
(<https://github.com/revsic>)
 - Satrec Intiative 전승현 연구원
(<https://github.com/FuZer>)
- 프로젝트에 기여해주시는 모든 분들

유체역학 엔진을 만들게 된 계기



유체역학 엔진을 만들게 된 계기



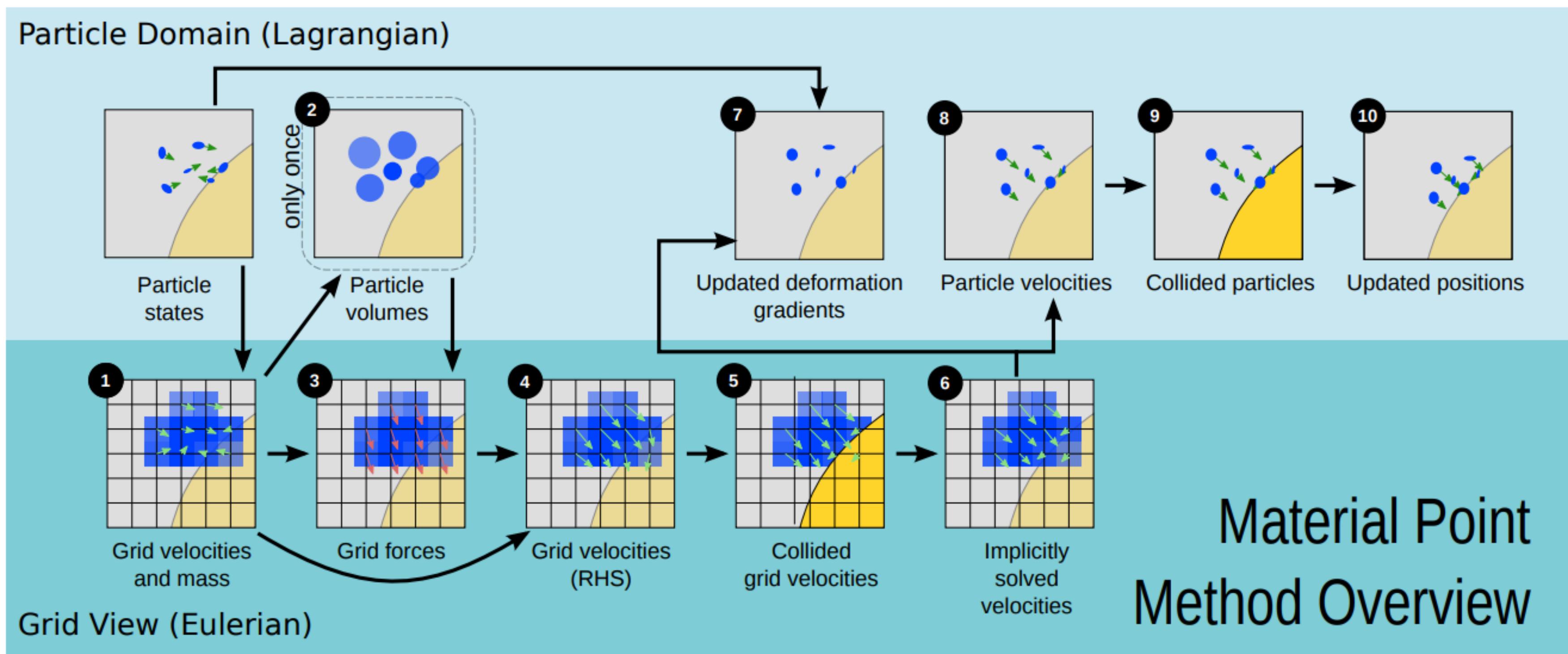
유체역학 엔진을 만들게 된 계기



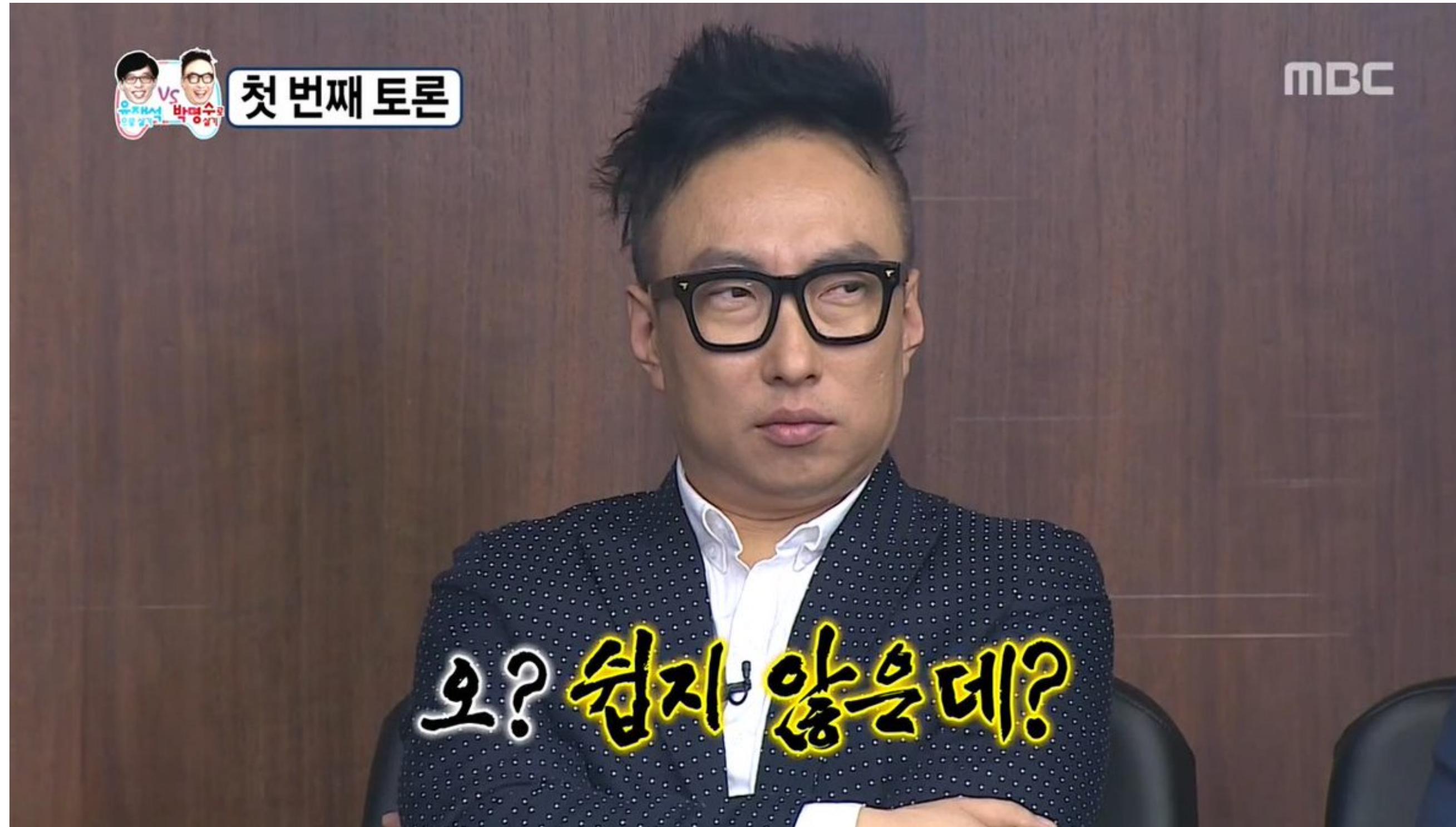
유체역학 엔진을 만들게 된 계기



유체역학 엔진을 만들게 된 계기



유체역학 엔진을 만들게 된 계기



유체역학 엔진을 만들게 된 계기

For integration, we need to compute $\frac{\partial \Psi}{\partial \mathbf{F}_E}$ and $\frac{\partial^2 \Psi}{\partial \mathbf{F}_E \partial \mathbf{F}_E} : \delta \mathcal{D}$. In this section, we will omit the subscripts E .

$$\begin{aligned}\Psi &= \mu \|\mathbf{F} - \mathbf{R}\|_F^2 + \frac{\lambda}{2}(J-1)^2 \\ \delta\Psi &= \delta \left(\mu \|\mathbf{F} - \mathbf{R}\|_F^2 + \frac{\lambda}{2}(J-1)^2 \right) \\ &= \mu \delta(\|\mathbf{F} - \mathbf{R}\|_F^2) + \lambda(J-1)\delta J \\ &= \mu \delta(\text{tr}(\mathbf{F}^T \mathbf{F})) - 2\mu \delta(\text{tr}(\mathbf{R}^T \mathbf{F})) + \mu \delta(\text{tr}(\mathbf{R}^T \mathbf{R})) + \lambda(J-1)\delta J \\ &= 2\mu \mathbf{F} : \delta \mathbf{F} - 2\mu \delta(\text{tr}(\mathbf{S})) + \lambda(J-1)J \mathbf{F}^{-T} : \delta \mathbf{F} \\ &= 2\mu \mathbf{F} : \delta \mathbf{F} - 2\mu \text{tr}(\delta \mathbf{S}) + \lambda(J-1)J \mathbf{F}^{-T} : \delta \mathbf{F} \\ \mathbf{F} &= \mathbf{R}\mathbf{S} \\ \delta \mathbf{F} &= \delta \mathbf{R}\mathbf{S} + \mathbf{R}\delta \mathbf{S} \\ \text{tr}(\delta \mathbf{S}) &= \text{tr}(\mathbf{R}^T \delta \mathbf{F}) - \text{tr}(\mathbf{R}^T \delta \mathbf{R}\mathbf{S}) \\ &= \text{tr}(\mathbf{R}^T \delta \mathbf{F}) - (\mathbf{R}^T \delta \mathbf{R}) : \mathbf{S} \\ &= \text{tr}(\mathbf{R}^T \delta \mathbf{F}) \\ &= \mathbf{R} : \delta \mathbf{F}\end{aligned}$$

Note that since $\mathbf{R}^T \mathbf{R} = \mathbf{I}$, $\mathbf{R}^T \delta \mathbf{R}$ must be skew-symmetric. Since \mathbf{S} is symmetric, $(\mathbf{R}^T \delta \mathbf{R}) : \mathbf{S} = 0$. Finally,

$$\begin{aligned}\delta\Psi &= 2\mu \mathbf{F} : \delta \mathbf{F} - 2\mu \text{tr}(\delta \mathbf{S}) + \lambda(J-1)J \mathbf{F}^{-T} : \delta \mathbf{F} \\ &= 2\mu \mathbf{F} : \delta \mathbf{F} - 2\mu \mathbf{R} : \delta \mathbf{F} + \lambda(J-1)J \mathbf{F}^{-T} : \delta \mathbf{F} \\ \frac{\partial \Psi}{\partial \mathbf{F}} : \delta \mathbf{F} &= (2\mu \mathbf{F} - 2\mu \mathbf{R} + \lambda(J-1)J \mathbf{F}^{-T}) : \delta \mathbf{F} \\ \frac{\partial \Psi}{\partial \mathbf{F}_E} &= 2\mu(\mathbf{F}_E - \mathbf{R}_E) + \lambda(J_E - 1)J_E \mathbf{F}_E^{-T}\end{aligned}$$

유체역학 엔진을 만들게 된 계기



유체역학 엔진을 만들게 된 계기



CRC
Taylor & Francis Group

FLUID ENGINE DEVELOPMENT

DOYUB KIM



유체역학 엔진을 만들게 된 계기



우-야 마음에 들어

목표

(~~퇴근하고 집에서 심심하니까~~)

게임에서 유체역학을 쓸 수 있게 해보자

유체역학을 활용해 만든 게임, 애니메이션

- 게임
 - Where's My Water? (2011)
 - Sprinkle (2011)
 - Watee (2011)
 - Splash !!! (2012)
- 애니메이션
 - 겨울왕국 (2013)
 - 모아나 (2016)

Sprinkle



<https://www.youtube.com/watch?v=7hYNpwouOZQ>

겨울왕국 (Frozen)



<https://www.youtube.com/watch?v=moSFlvxnbgk>

2. 유체역학이란?

- 유체, 그리고 유체역학
- 간단한 유체 시뮬레이터 구현해 보기
- 유체역학을 이해하기 위한 몇 가지 수학 용어 (Bonus*)
- 유체역학의 핵심, 나비에-스톡스 방정식
- 유체역학을 바라보는 관점, 입자 방식과 격자 방식

유체

고체에 비해 형상이 일정하지 않아 변형이 쉽고
자유로이 흐를 수 있는 액체와 기체와 플라즈마를 총칭하는 말

- 변형이 쉬움
- 흐르는 성질이 있음
- 모양이 정해져 있지 않음
- 점성이 있음
 - 점성 (Viscosity) : 유체의 서로 붙어 있는 부분이 떨어지지 않으려는 성질
- 압축성이 있음
 - 압축성 (Compressible) : 압력이나 온도가 변하면 밀도가 변한다는 것

유체

고체에 비해 형상이 일정하지 않아 변형이 쉽고
자유로이 흐를 수 있는 액체와 기체와 플라즈마를 총칭하는 말



물 (Water)



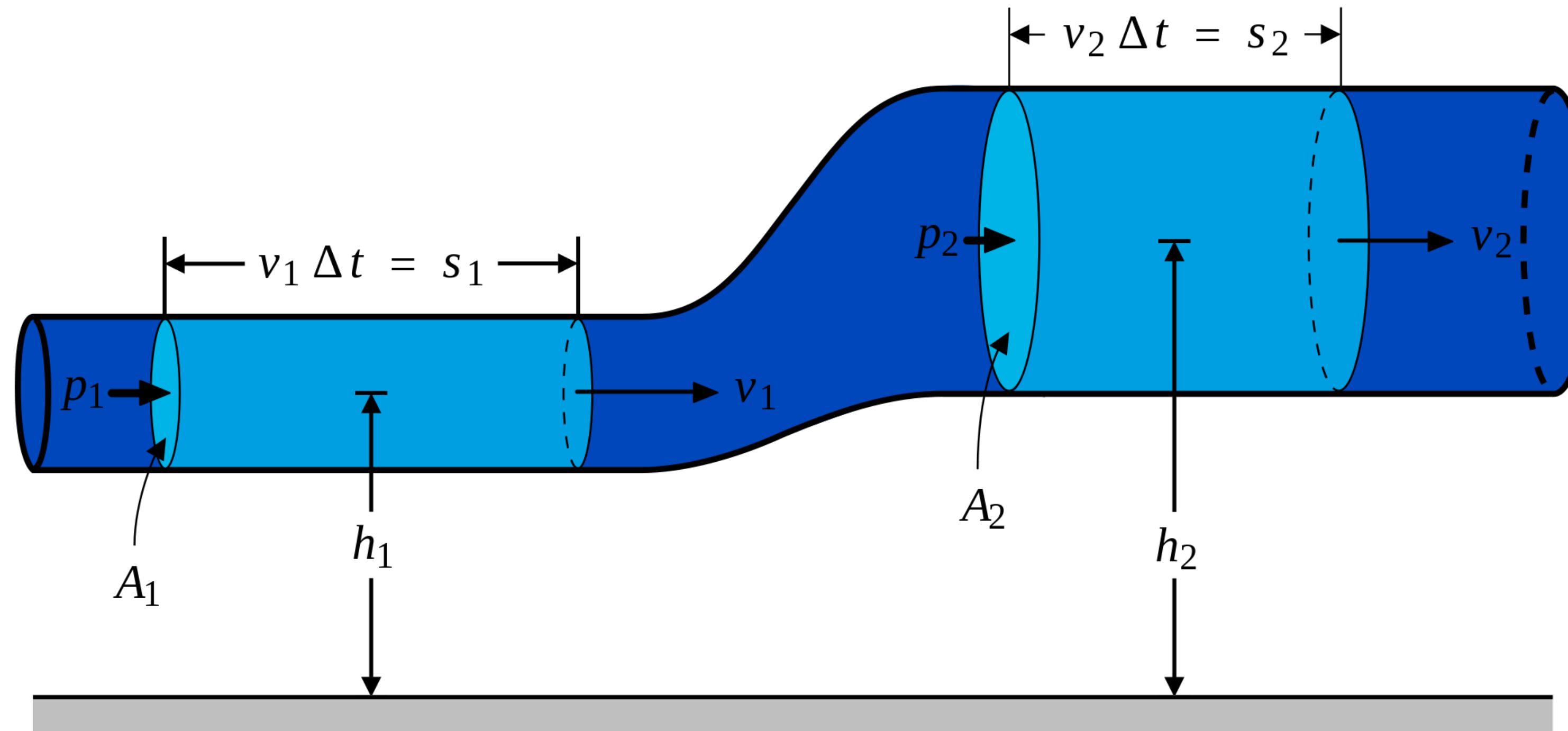
불 (Fire)



연기 (Smoke)

유체역학

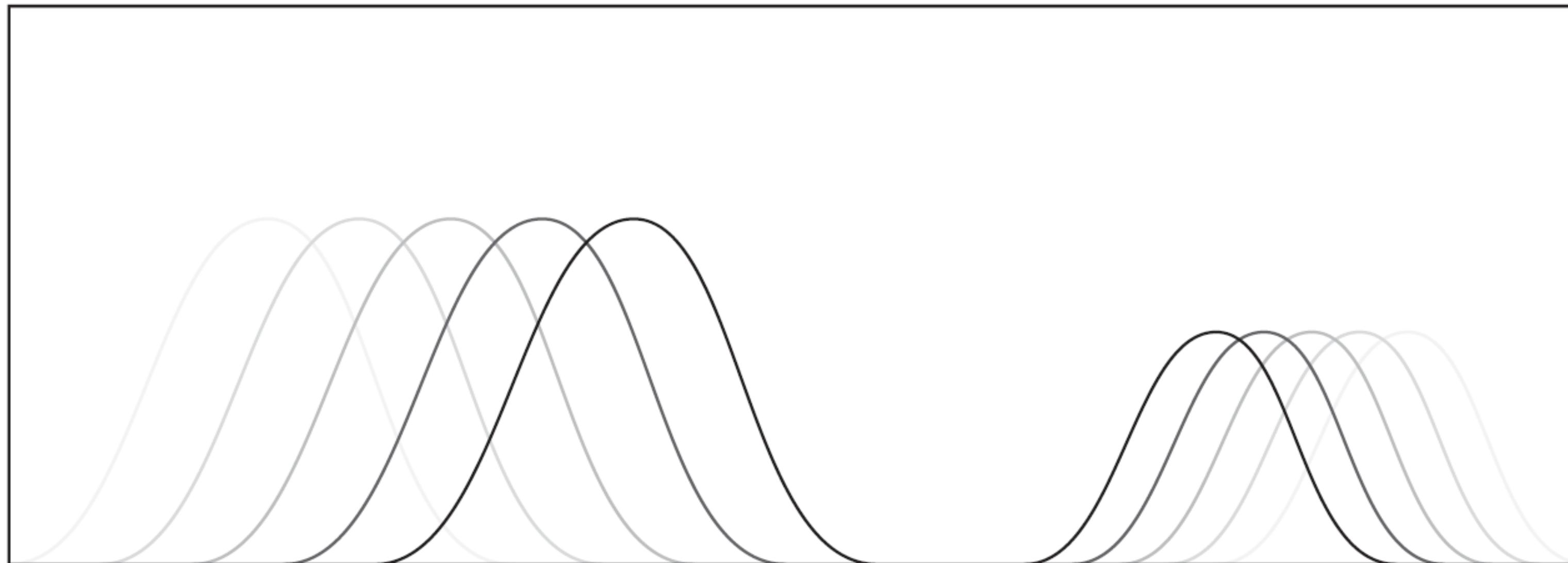
유체(액체와 기체)의 운동에 대해서 연구하는 학문



간단한 유체 시뮬레이터 구현해 보기

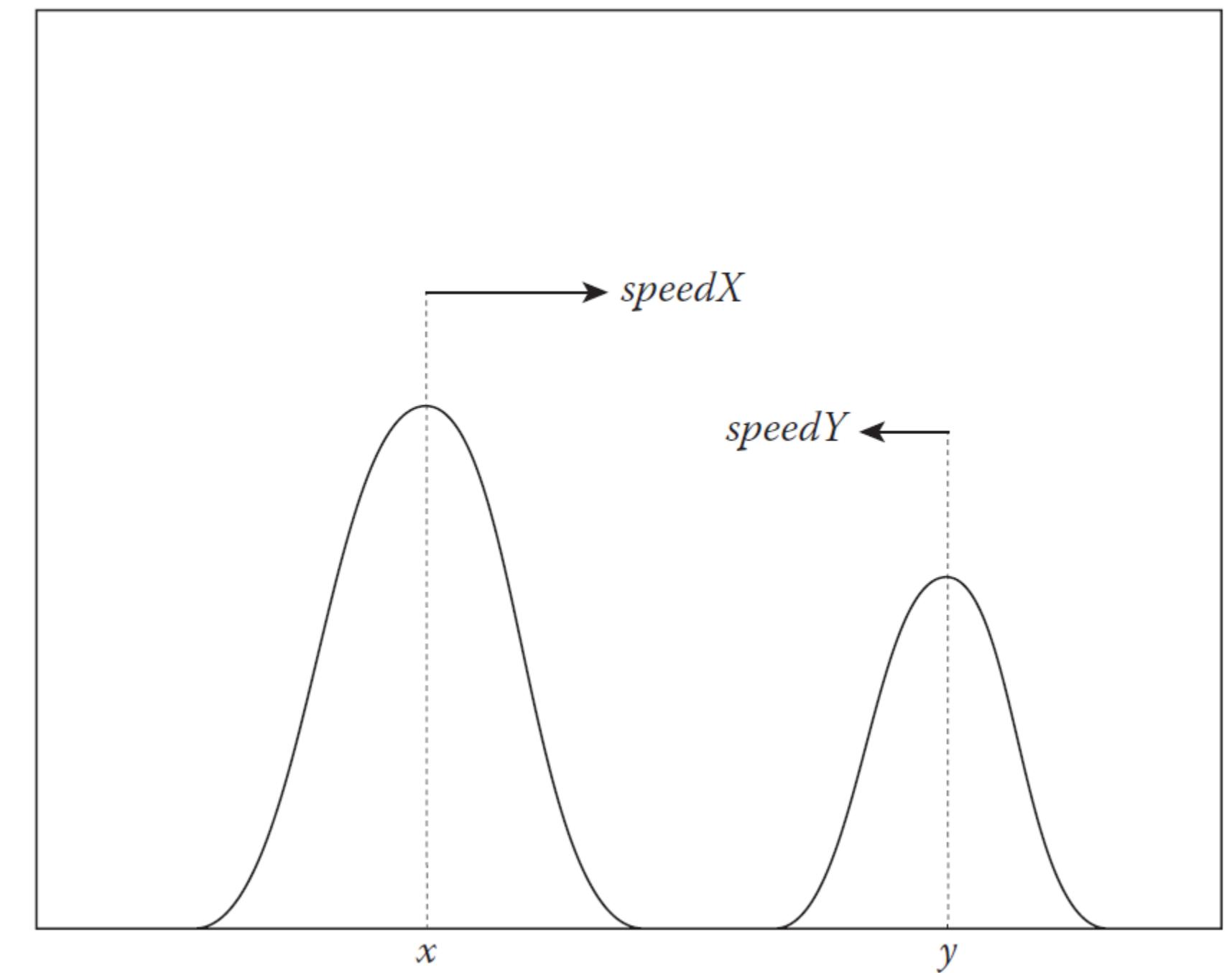
1차원 물결 시뮬레이션

(예제 코드를 제공해주신 김도엽님께 감사드립니다.)



1단계 : 상태 정의하기

```
int main() {  
    double x = 0.0;  
    double y = 1.0;  
    double speedX = 1.0;  
    double speedY = -0.5;  
  
    return 0;  
}
```



2단계 : 움직임 계산하기

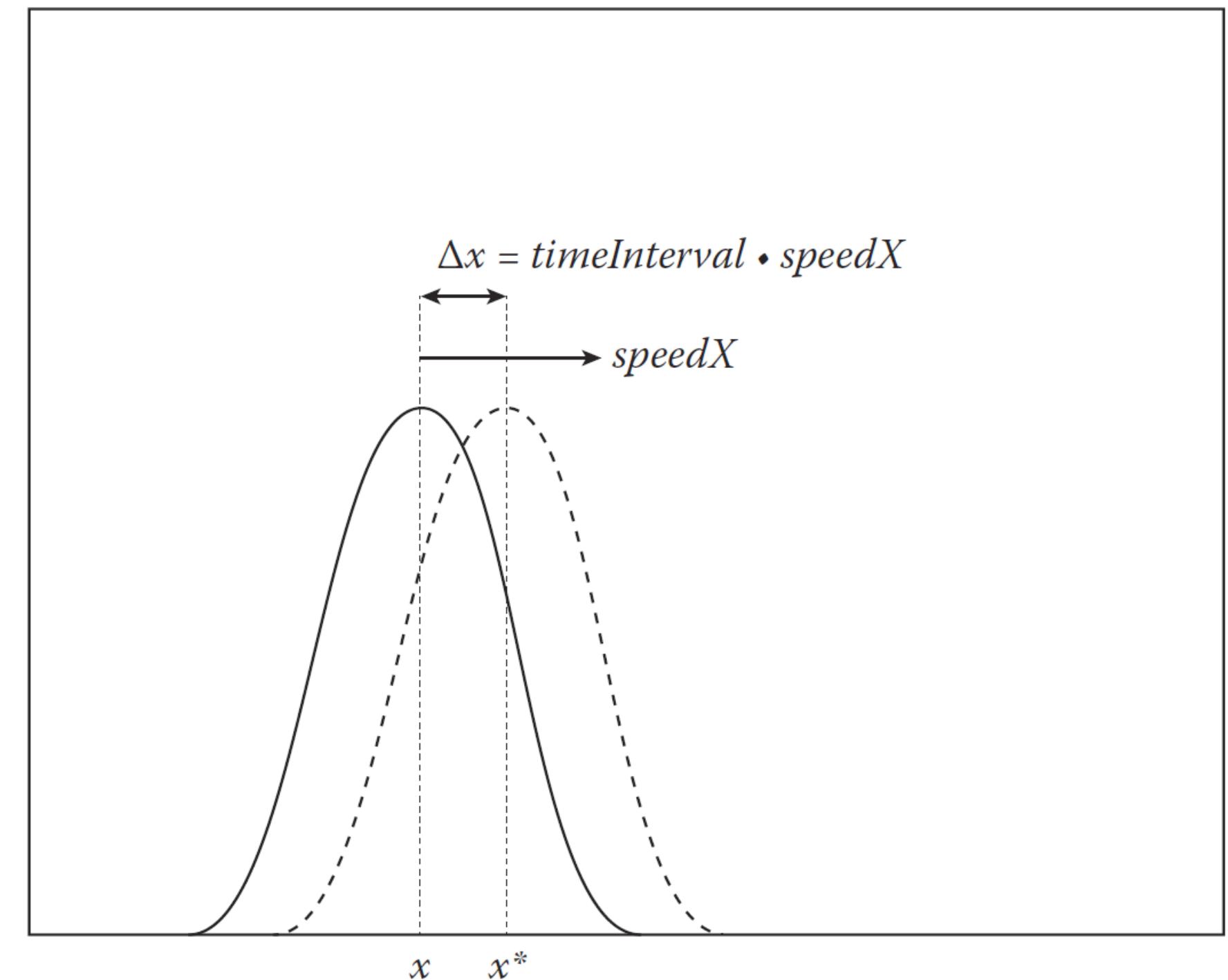
```
int main() {
    double x = 0.0;
    double y = 1.0;
    double speedX = 1.0;
    double speedY = -0.5;

    const int fps = 100;
    const double timeInterval = 1.0 / fps;

    for (int i = 0; i < 1000; ++i) {
        // Update waves
    }
    return 0;
}
```

2단계 : 움직임 계산하기

```
void UpdateWave(  
    const double timeInterval,  
    double* x, double* speed) {  
    (*x) += timeInterval * (*speed);  
}
```



3단계 : 충돌 처리

```
void UpdateWave(  
    const double timeInterval,  
    double* x, double* speed) {  
    (*x) += timeInterval * (*speed);  
  
    // Boundary reflection  
    if ((*x) > 1.0) {  
        (*speed) *= -1.0;  
        (*x) = 1.0 + timeInterval * (*speed);  
    } else if ((*x) < 0.0) {  
        (*speed) *= -1.0;  
        (*x) = timeInterval * (*speed);  
    }  
}
```

3단계 : 충돌 처리

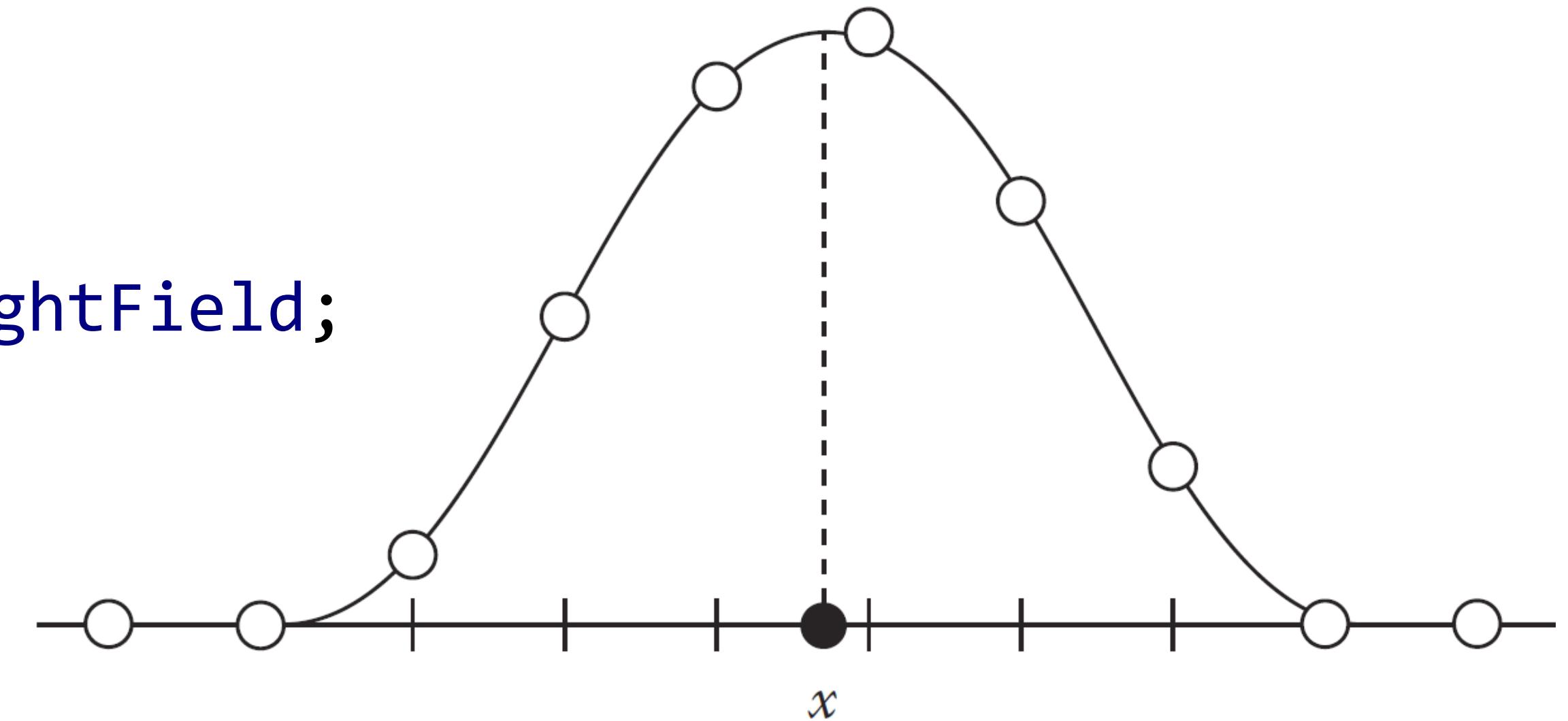
```
int main() {
    ...

    for (int i = 0; i < 1000; ++i) {
        // Update waves
        UpdateWave(timeInterval, &x, &speedX);
        UpdateWave(timeInterval, &y, &speedY);

    }
    return 0;
}
```

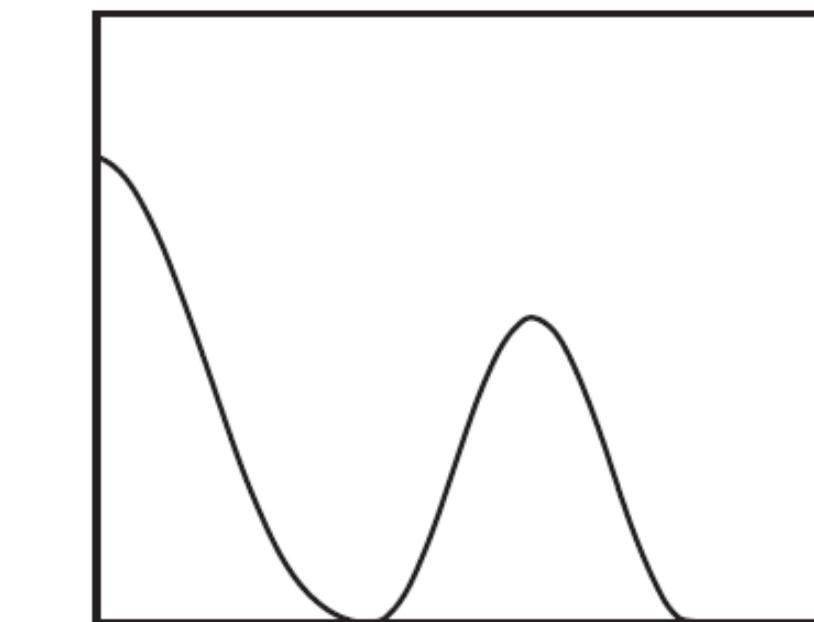
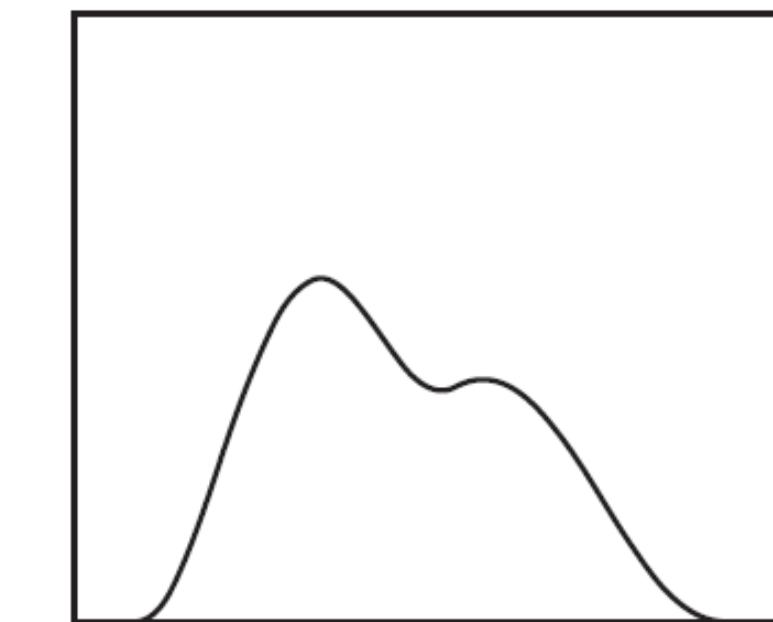
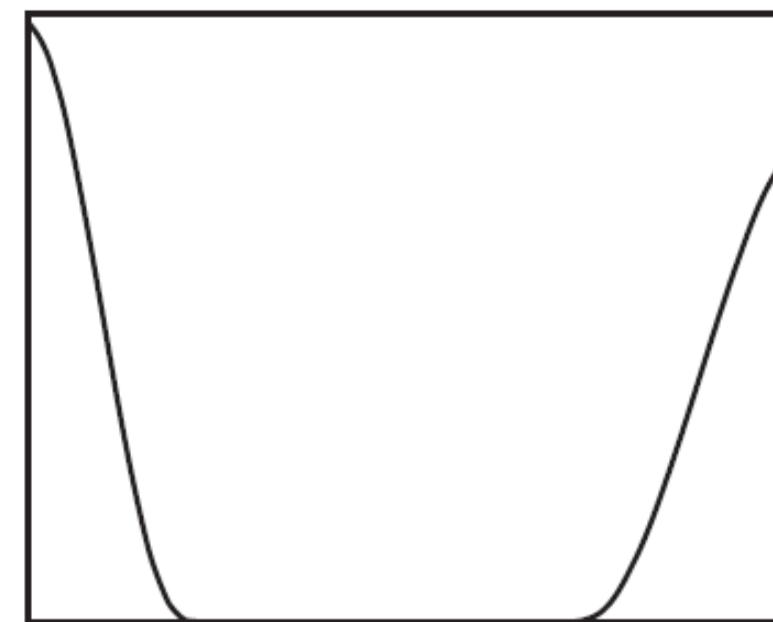
4단계 : 시각화

```
const size_t BUFFER_SIZE = 80;  
  
int main() {  
    const double waveLengthX = 0.8;  
    const double waveLengthY = 1.2;  
  
    const double maxHeightX = 0.5;  
    const double maxHeightY = 0.4;  
  
    std::array<double, BUFFER_SIZE> heightField;  
  
    ...  
}
```



4단계 : 시각화

```
void AccumulateWaveToHeightField(  
    const double x,  
    const double waveLength,  
    const double maxHeight,  
    std::array<double, BUFFER_SIZE>* heightField) {  
    const double quarterWaveLength = 0.25 * waveLength;  
    const int start = static_cast<int>((x - quarterWaveLength) * BUFFER_SIZE);  
    const int end = static_cast<int>((x + quarterWaveLength) * BUFFER_SIZE);
```



4단계 : 시각화

```
for (int i = start; i < end; ++i) {
    int iNew = i;
    if (i < 0) {
        iNew = -i - 1;
    } else if (i >= static_cast<int>(BUFFER_SIZE)) {
        iNew = 2 * BUFFER_SIZE - i - 1;
    }

    const double distance = fabs((i + 0.5) / BUFFER_SIZE - x);
    const double height = maxHeight * 0.5 *
        (cos(std::min(distance * M_PI / quarterWaveLength, M_PI)) + 1.0);
    (*heightField)[iNew] += height;
}
```

소스 코드 & 결과

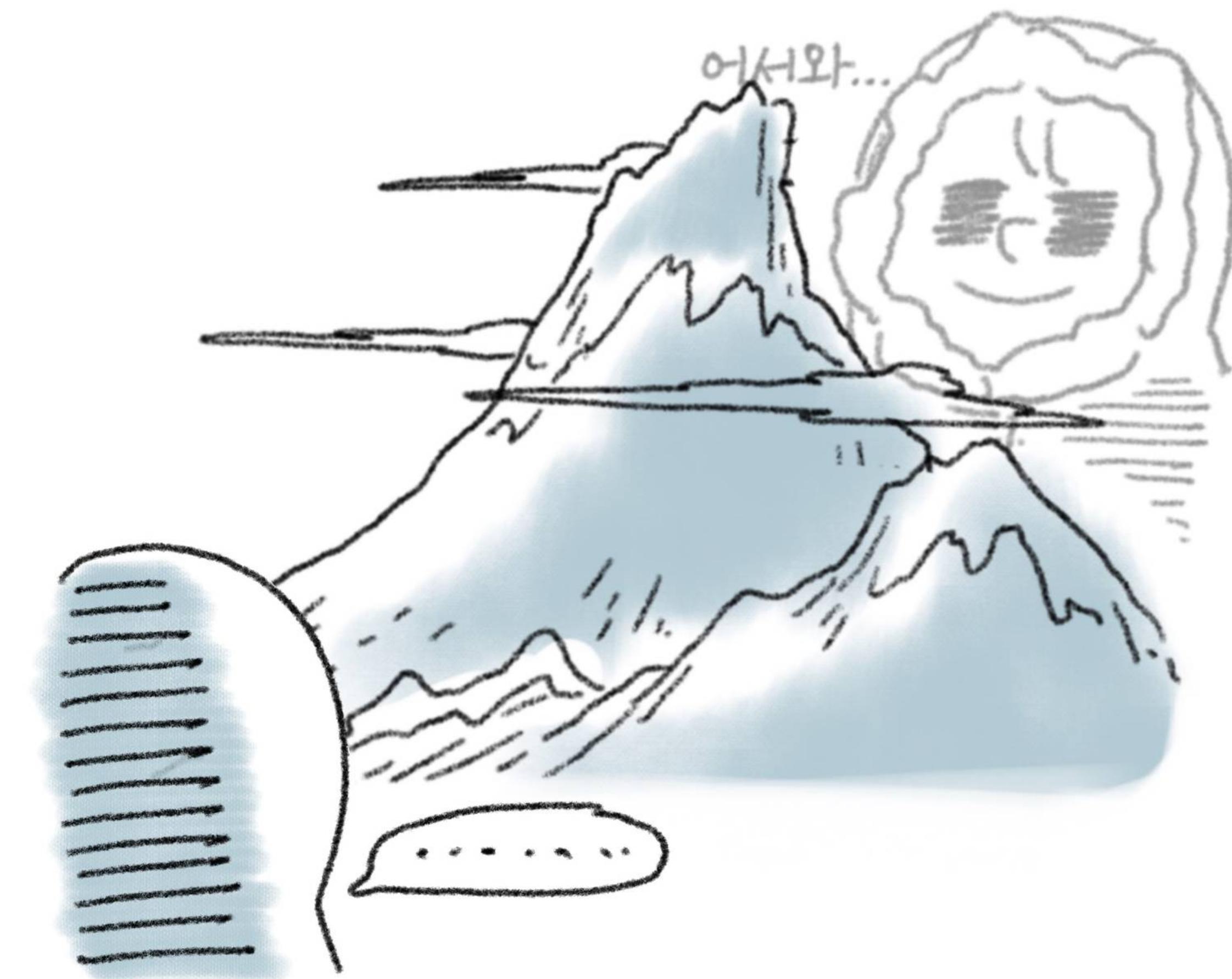
<https://ideone.com/46tFxu>

(예제 코드를 제공해주신 김도엽님께 감사드립니다.)

참 쉽죠?



◦ [제부터 오를 산은
에베레스트란다...]"

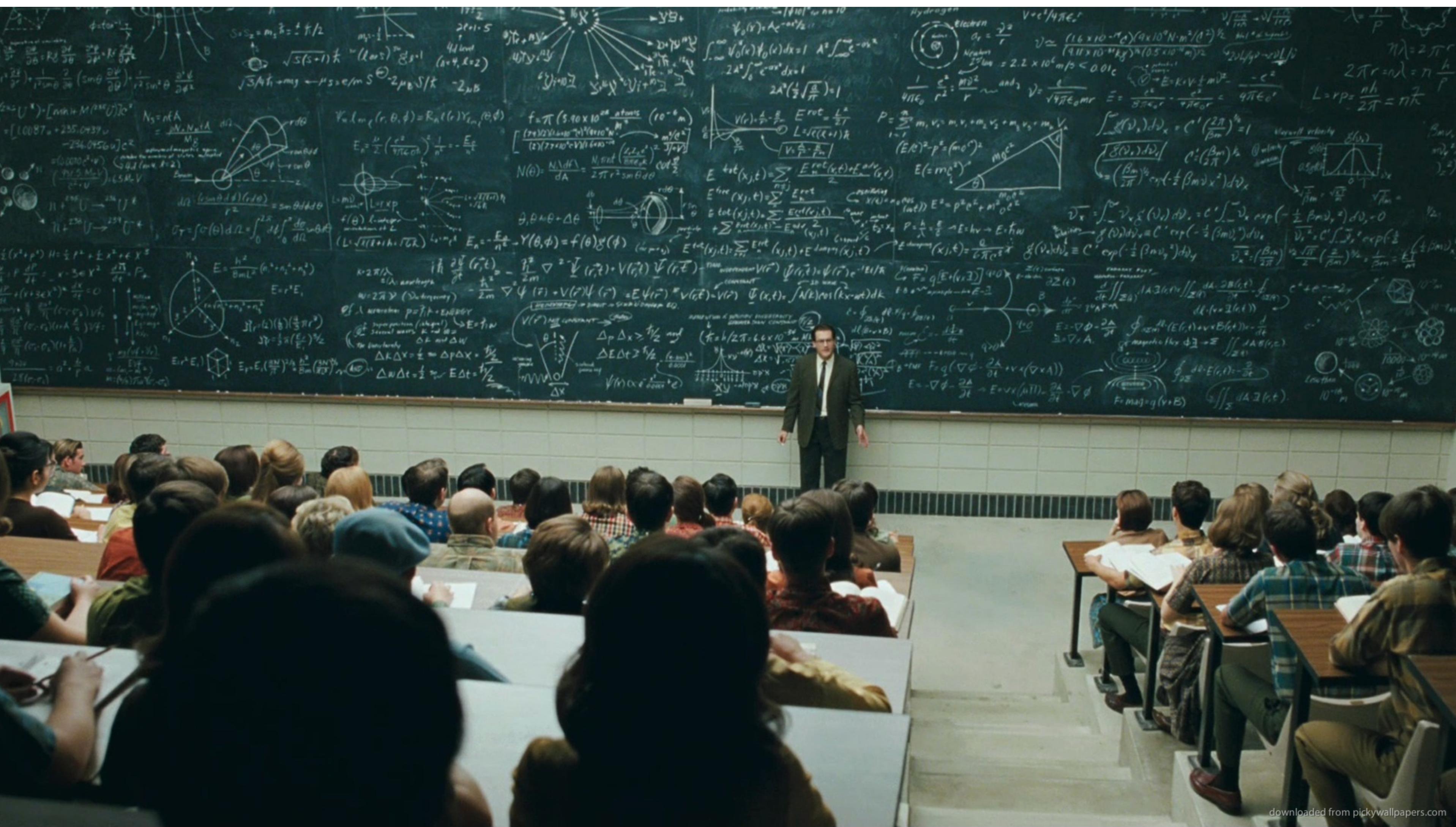


유체역학의 핵심, 나비에-스톡스 방정식

(Navier–Stokes equations for incompressible flow)

$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla \cdot \mathbf{v})\mathbf{v} = \mathbf{f} - \frac{\nabla p}{\rho} + \mu \nabla^2 \mathbf{v}$$

$$\nabla \cdot \mathbf{v} = 0$$

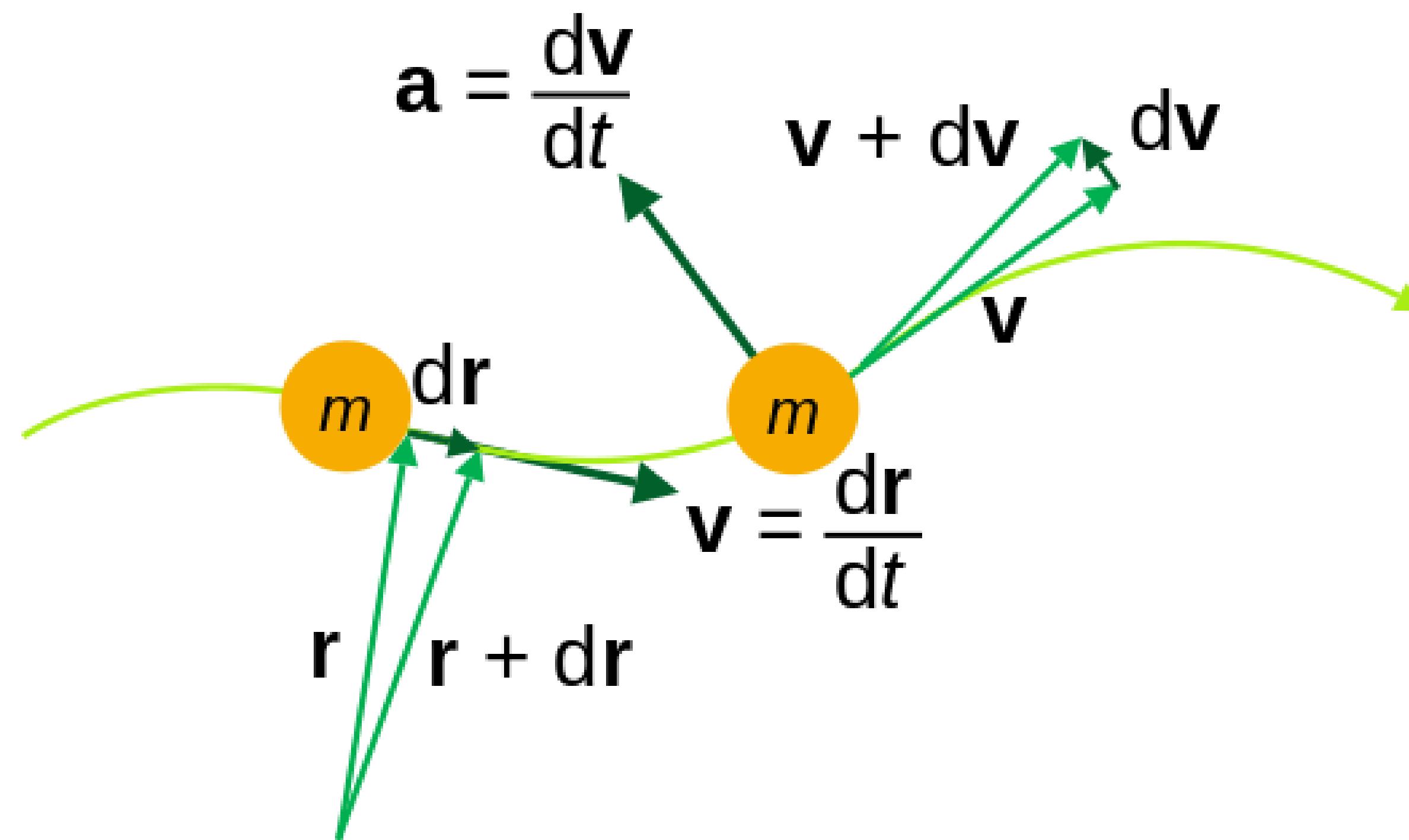


downloaded from pickywallpapers.com

$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla \cdot \mathbf{v})\mathbf{v} = f - \frac{\nabla p}{\rho} + \mu \nabla^2 \mathbf{v}$$

$$\boxed{\frac{\partial \mathbf{v}}{\partial t} + (\nabla \cdot \mathbf{v})\mathbf{v}} = f - \frac{\nabla p}{\rho} + \mu \nabla^2 \mathbf{v}$$

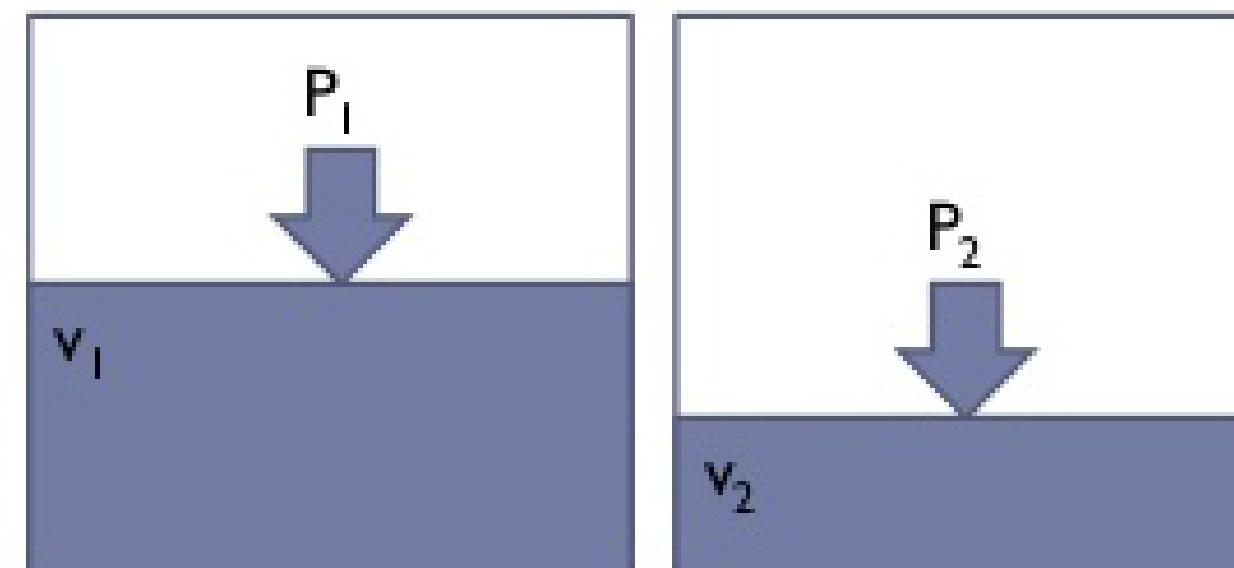
가속도
(Acceleration)



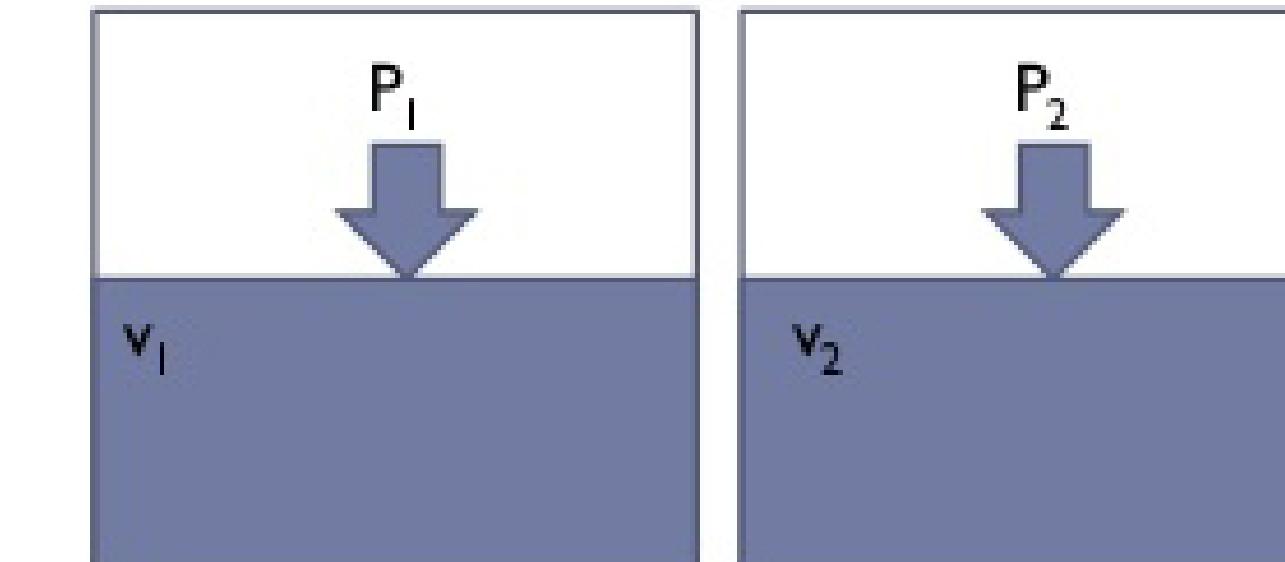
$$\frac{\partial \mathbf{v}}{\partial t} + \boxed{(\nabla \cdot \mathbf{v})\mathbf{v}} = f - \frac{\nabla p}{\rho} + \mu \nabla^2 \mathbf{v}$$

압축성
(Compressibility)

Compressible and Incompressible flows



Compressible fluid



Incompressible fluid

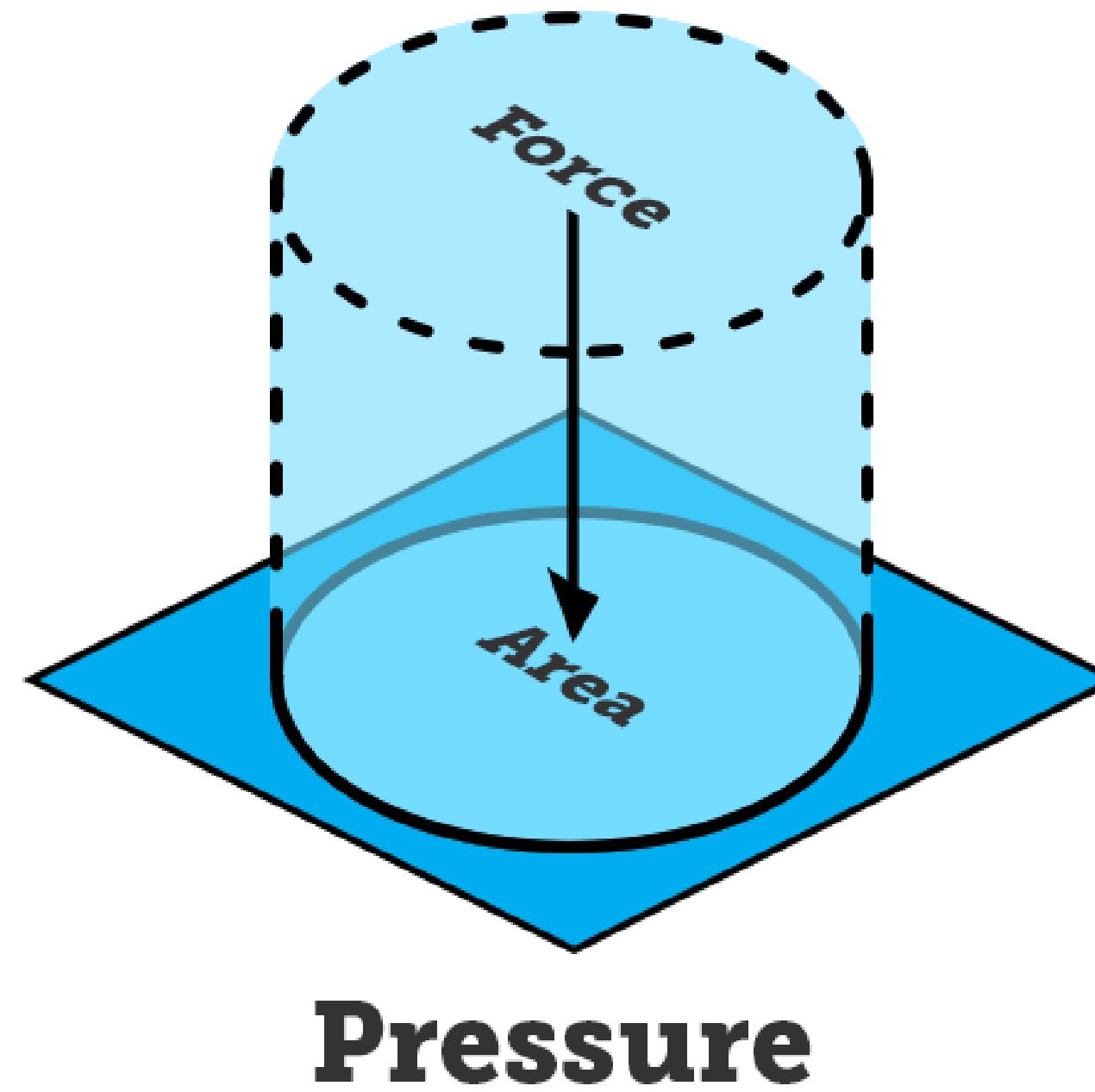
$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla \cdot \mathbf{v})\mathbf{v} = \boxed{f} - \frac{\nabla p}{\rho} + \mu \nabla^2 \mathbf{v}$$

외력
(External Force)



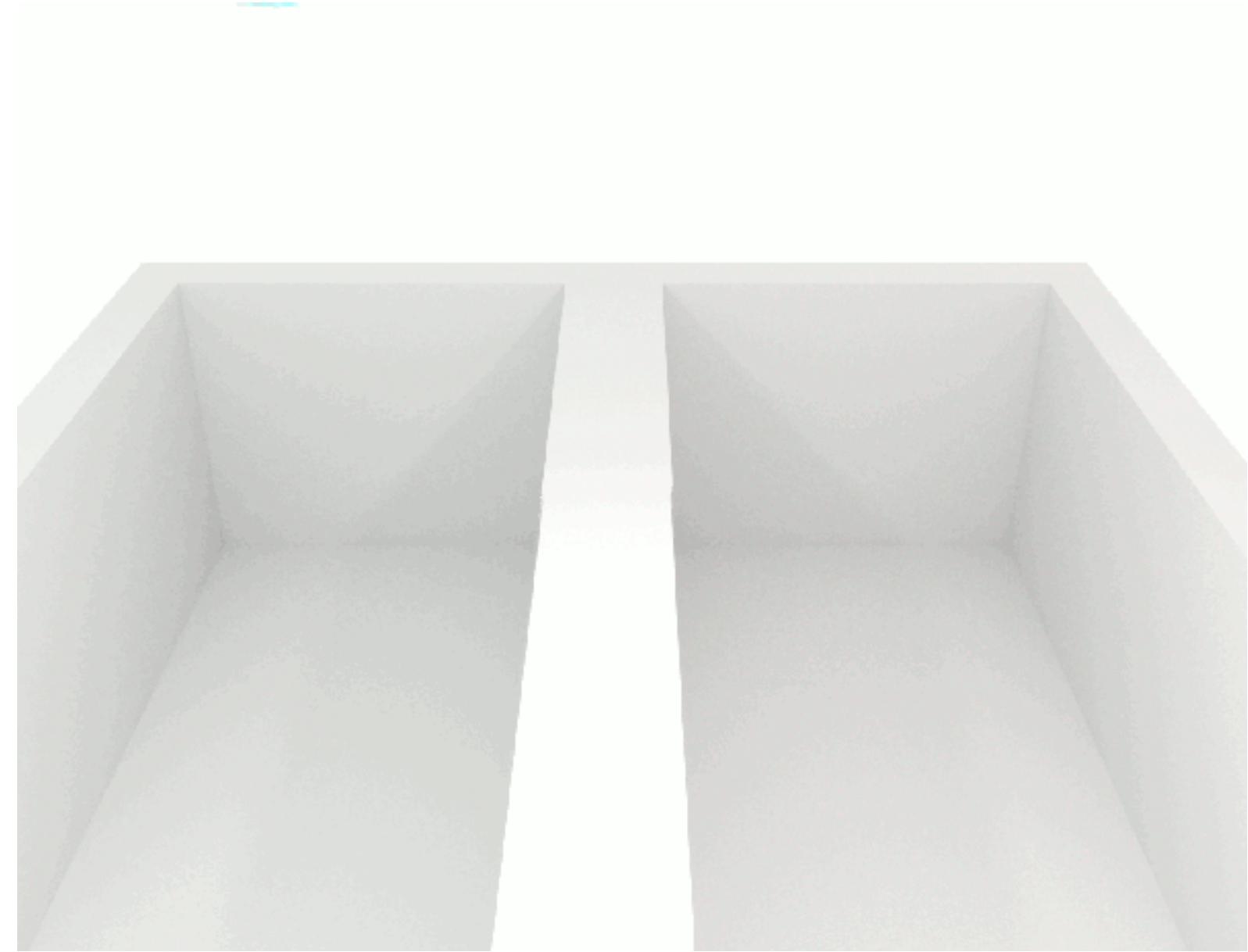
$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla \cdot \mathbf{v})\mathbf{v} = \mathbf{f} - \boxed{\frac{\nabla p}{\rho}} + \mu \nabla^2 \mathbf{v}$$

압력
(Pressure)



$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla \cdot \mathbf{v})\mathbf{v} = \mathbf{f} - \frac{\nabla p}{\rho} + \boxed{\mu \nabla^2 \mathbf{v}}$$

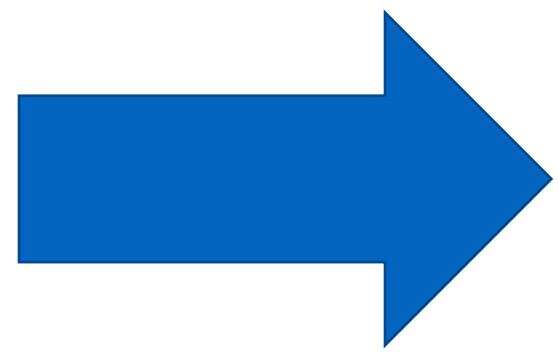
점성
(Viscosity)



$$\frac{\partial \mathbf{v}}{\partial t} + (\cancel{\nabla \cdot \mathbf{v}}) \mathbf{v} = f - \frac{\nabla p}{\rho} + \mu \nabla^2 \mathbf{v}$$

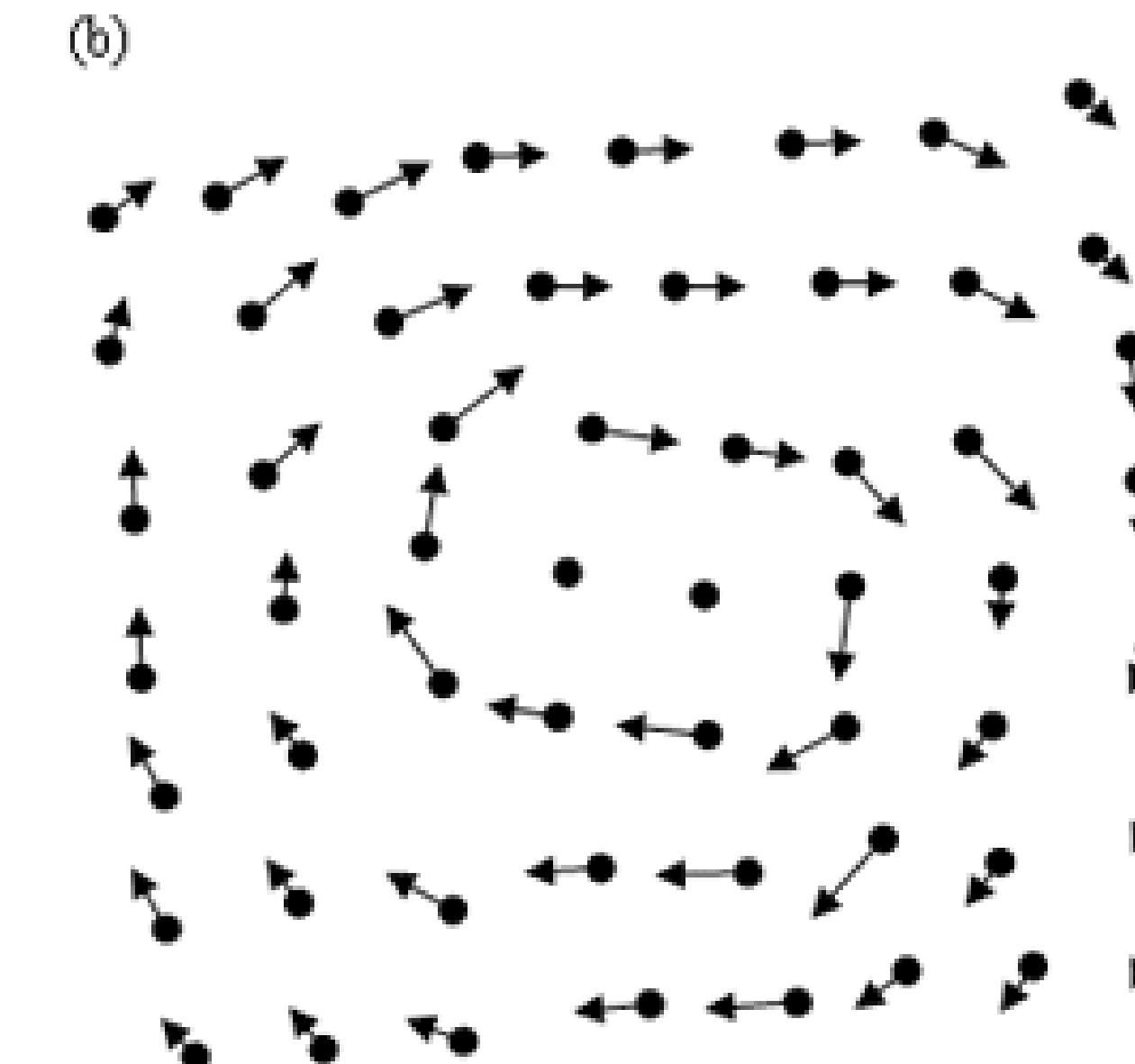
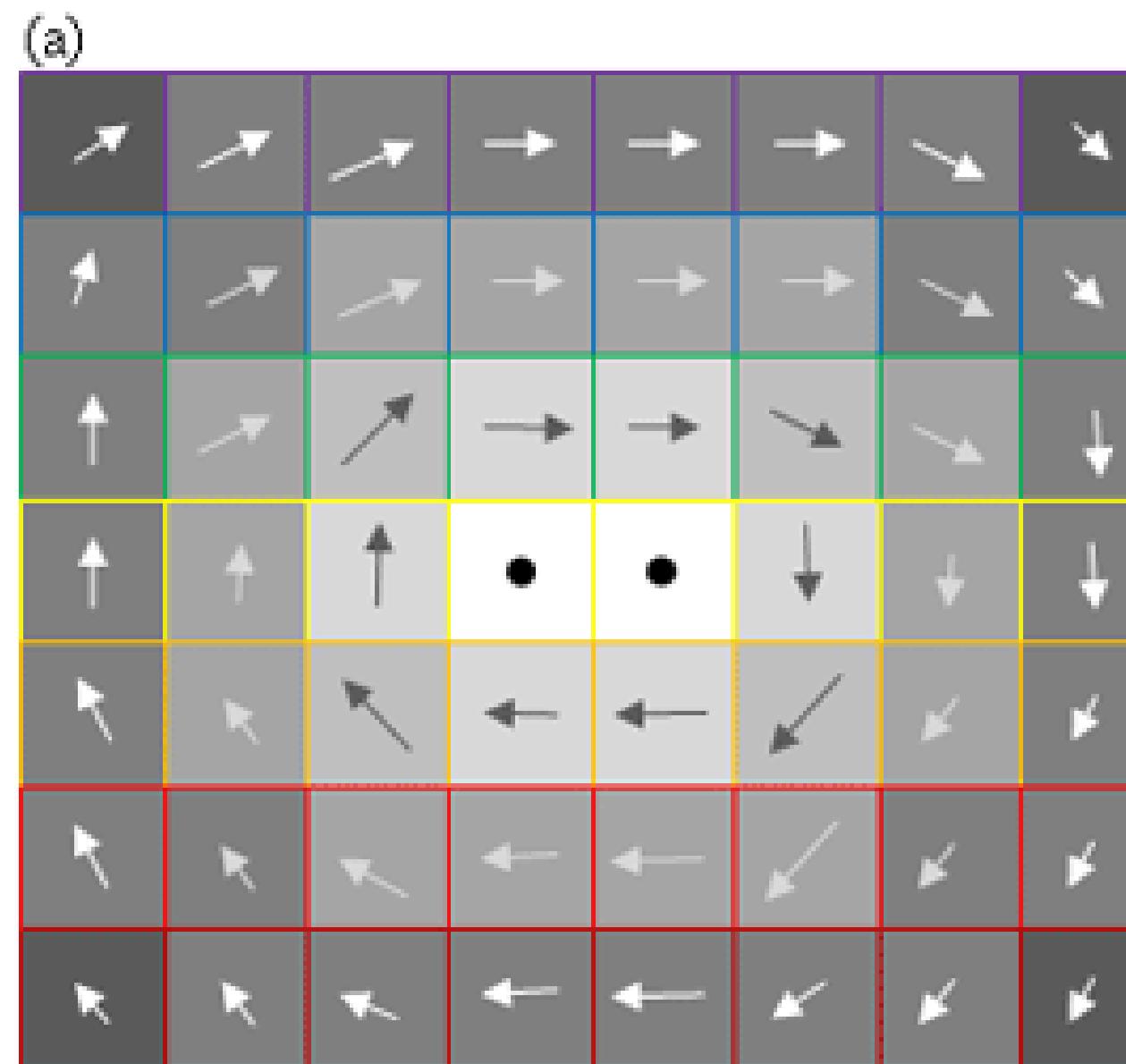
$\nabla \cdot \mathbf{v} \neq 0$: 압축성 (Compressible)

$\nabla \cdot \mathbf{v} = 0$: 비압축성 (Incompressible)



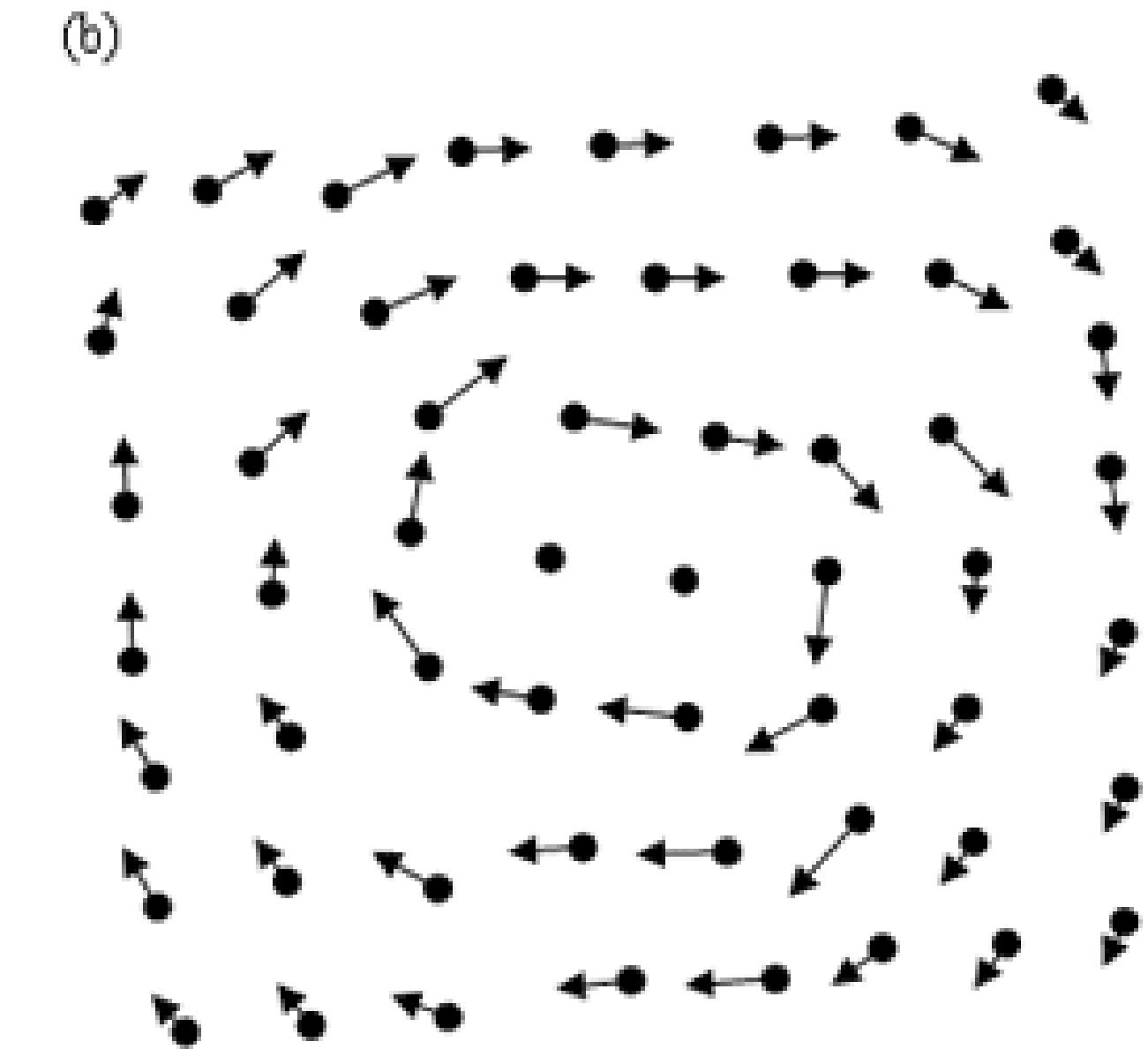
전산 유체에서는 보통 비압축성 유체를 고려

유체역학을 바라보는 관점, 입자 방식과 격자 방식 (Particle-based vs Grid-based)



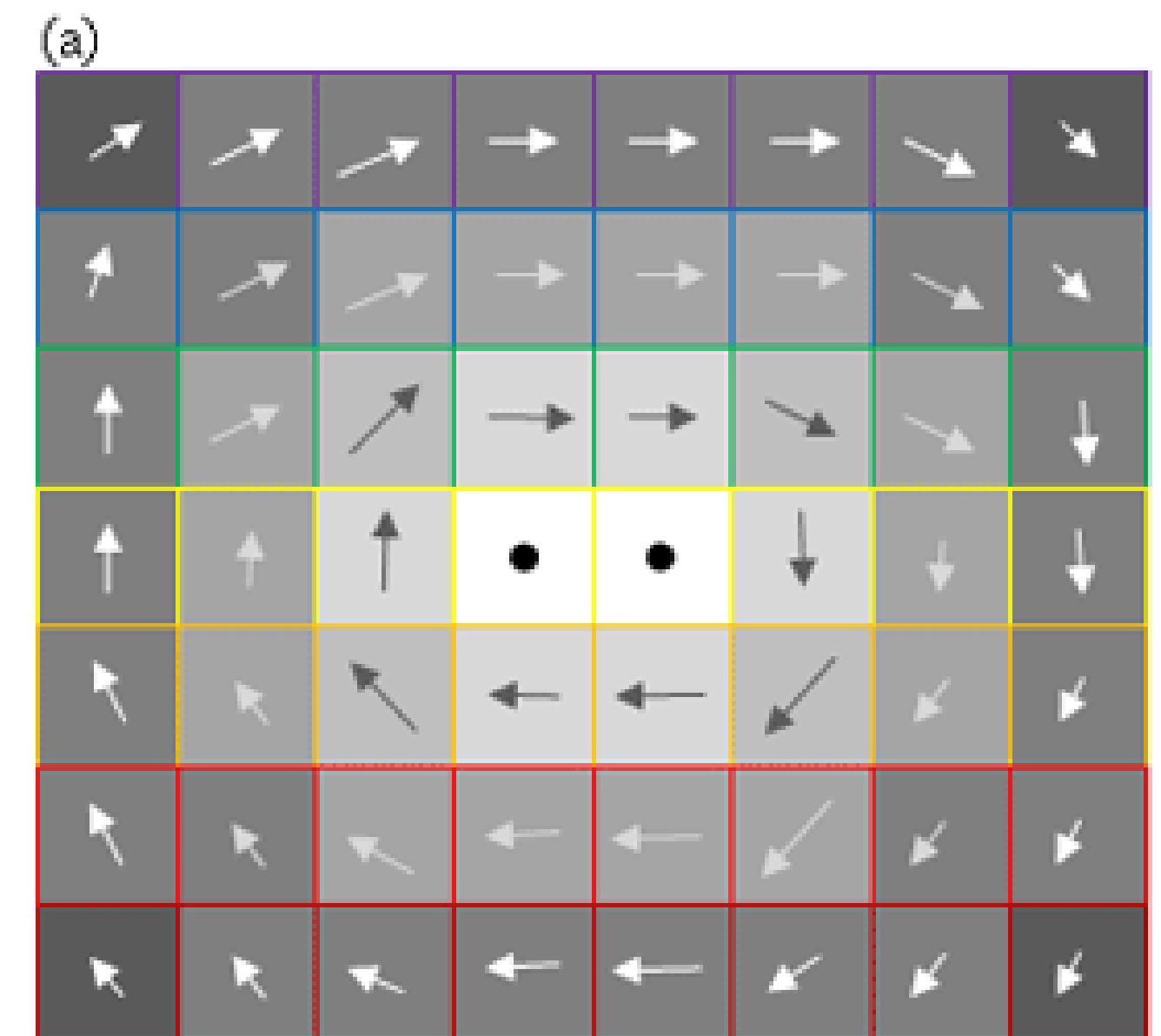
입자 방식 (Particle-based)

- 라그랑주 관점(Lagrangian View)
 - 입자 하나하나에 촛점을 맞춤
 - 각 입자를 따라가면서 물리량을 나타내는 기술법
(위치, 속도, 가속도 등)
 - 시간이 지난 후의 위치를 매팅 함수를 이용해 표기



격자 방식 (Grid-based)

- 오일러 관점 (Eulerian View)
 - 관찰할 지점을 고정 (점의 위치가 절대로 변하지 않음)
 - 점을 지나는 유체의 물리량(속도 변화율)을 표현
 - 유체가 포함된 영역(격자)의 각 점에 속성 값을 저장
 - 속도 : 화살표
 - 밀도 : 격자의 색깔
 - 압력 : 화살표의 색깔
 - 온도 : 격자의 외곽선 색깔



3. 유체역학 엔진 개발기

- 엔진 개발 과정
- 겪었던 시행착오들, 그리고 해결 과정
- 현재까지의 결과
- 앞으로 개선할 부분, 그리고 고민

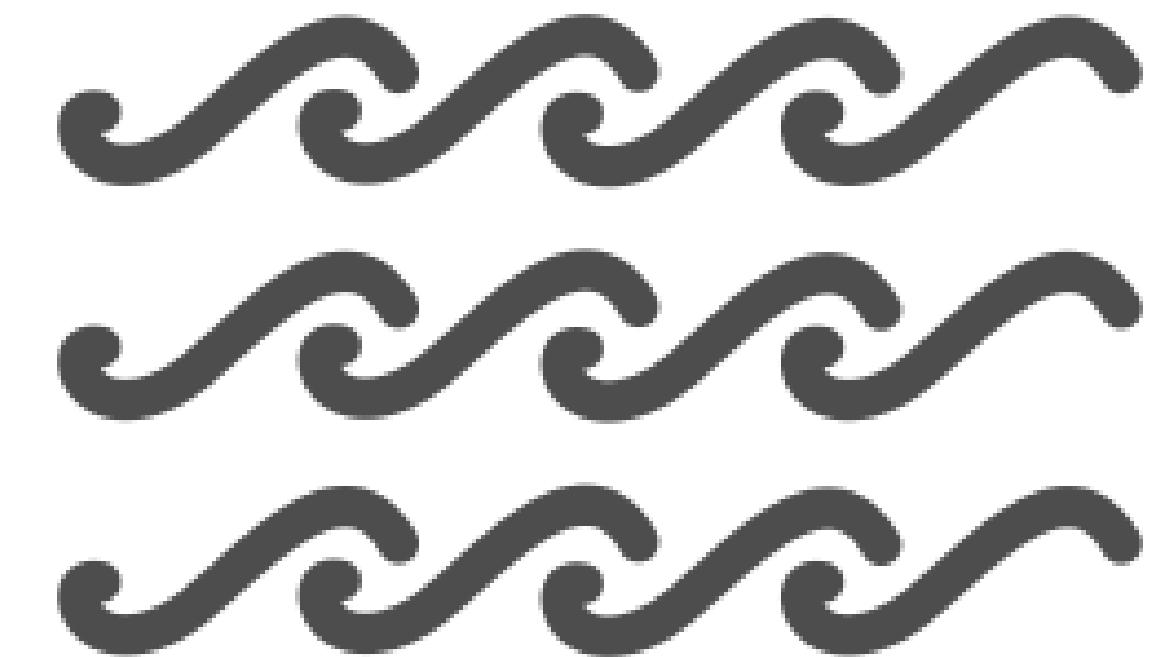
3.1. 엔진 개발 과정

- 유체역학 엔진 : CubbyFlow
- 어떻게 시뮬레이션하는가?
- 입자, 그리고 격자의 구현 (Bonus*)

CubbyFlow

Voxel-based fluid simulation engine for computer games

- <https://github.com/utilForever/CubbyFlow>
- Jet Framework를 기반으로 구현한 유체 엔진
- 라이선스 : MIT
- 작성 언어 : C++17
- 지원 컴파일러 : GCC, Clang, MSVC
- 지원 OS
 - macOS 10.12.6 이상, Ubuntu 17.04 이상
 - Windows, Windows Subsystem for Linux



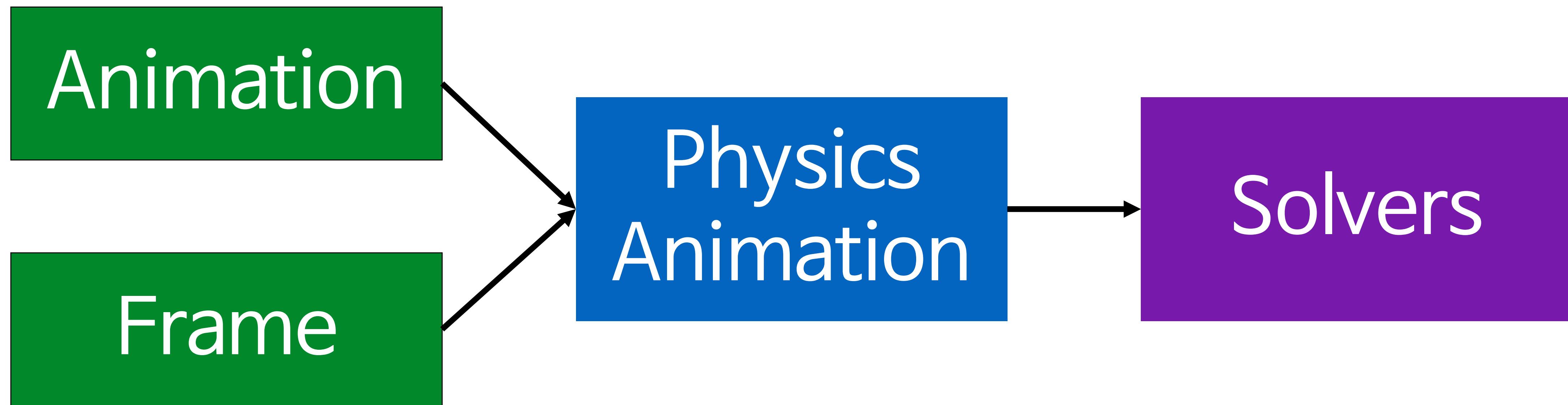
CubbyFlow의 주요 기능

- Basic math and geometry operations and data structures
- Spatial query accelerators
- SPH and PCISPH fluid simulators
- Stable fluids-based smoke simulator
- Level set-based liquid simulator
- PIC, FLIP, and APIC fluid simulators
- Upwind, ENO, and FMM level set solvers
- Jacobi, Gauss-Seidel, SOR, MG, CG, ICCG, and MGPCG linear system solvers
- Spherical, SPH, Zhu & Bridson, and Anisotropic kernel for points-to-surface converter
- Converters between signed distance function and triangular mesh
- C++ and Python API

CubbyFlow의 구조

- 수학 관련 클래스 : Array, Matrix, Vector, BLAS, CG, PDE, Quaternion, SVD
- 기하 관련 클래스 (1) : Box, Cylinder, Plane, Sphere, Triangle, TriangleMesh
- 기하 관련 클래스 (2) : BVH, KdTree, Octree, Quadtree, Transform, Ray
- 애니메이션 관련 클래스 : Animation, Frame, PhysicsAnimation
- 충돌 관련 클래스 : BoundingBox, Collider, ColliderSet, RigidbodyCollider
- 장(Field) 관련 클래스 : Field, ScalarField, VectorField, ConstantField, CustomField
- 입자 관련 클래스 : ParticleSystemData, ParticleSystemSolver, SPHSolver, PCISPHSolver
- 격자 관련 클래스 (1) : GridSystemData, GridFluidSolver, GridSmokeSolver
- 격자 관련 클래스 (2) : (ENO/FMM/Iterative/Upwind)LevelSetSolver
- 하이브리드 관련 클래스 : APICSolver, FLIPSolver, PICsolver
- 유틸 클래스 : Factory, Functor, MG, Parallel, Sampler, Serialization, Timer

어떻게 시뮬레이션하는가?



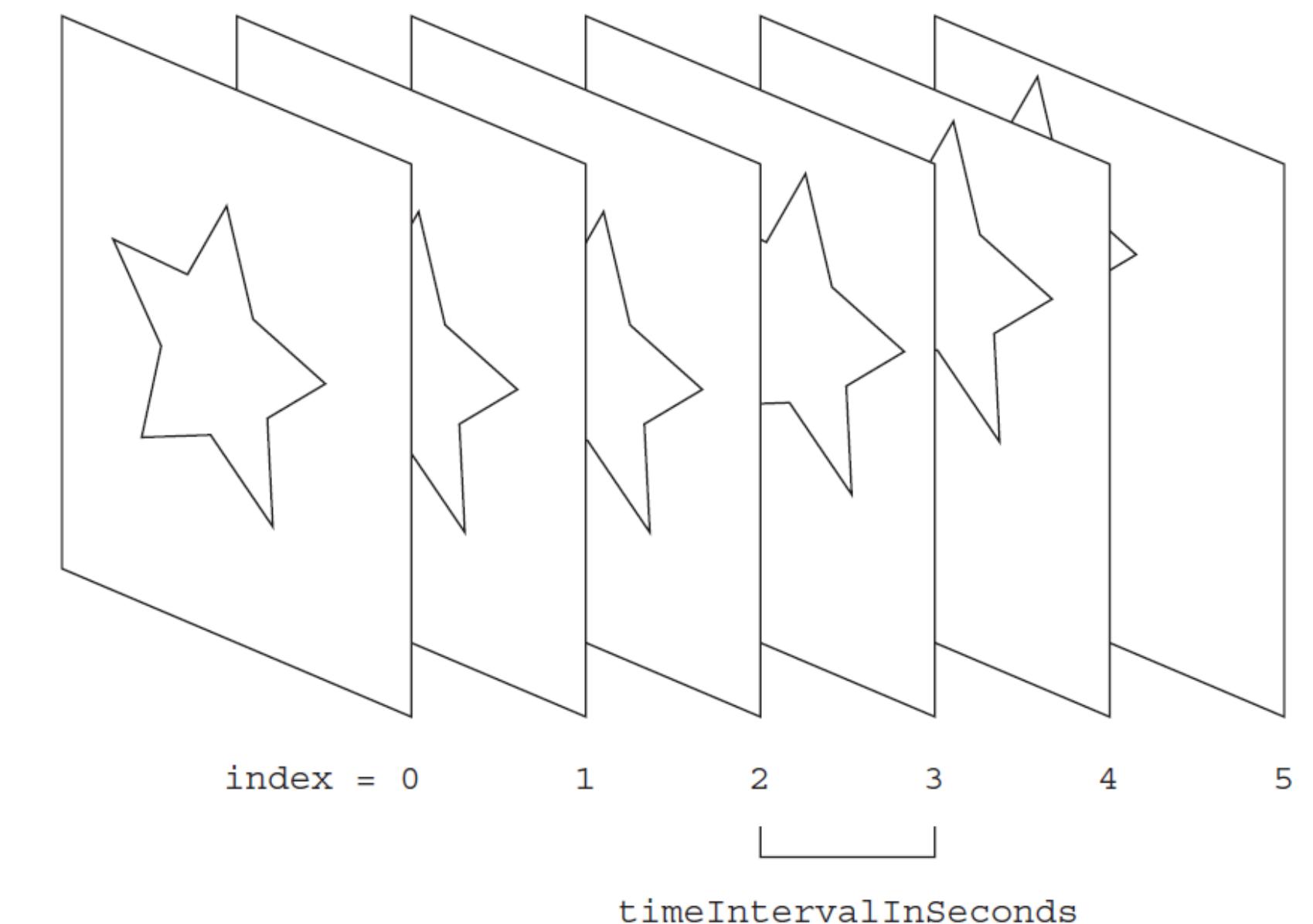
프레임 (Frame)

```
struct Frame final {
    int index = 0;
    double timeIntervalInSeconds = 1.0 / 60.0;

    double TimeInSeconds() const {
        return index * timeIntervalInSeconds;
    }

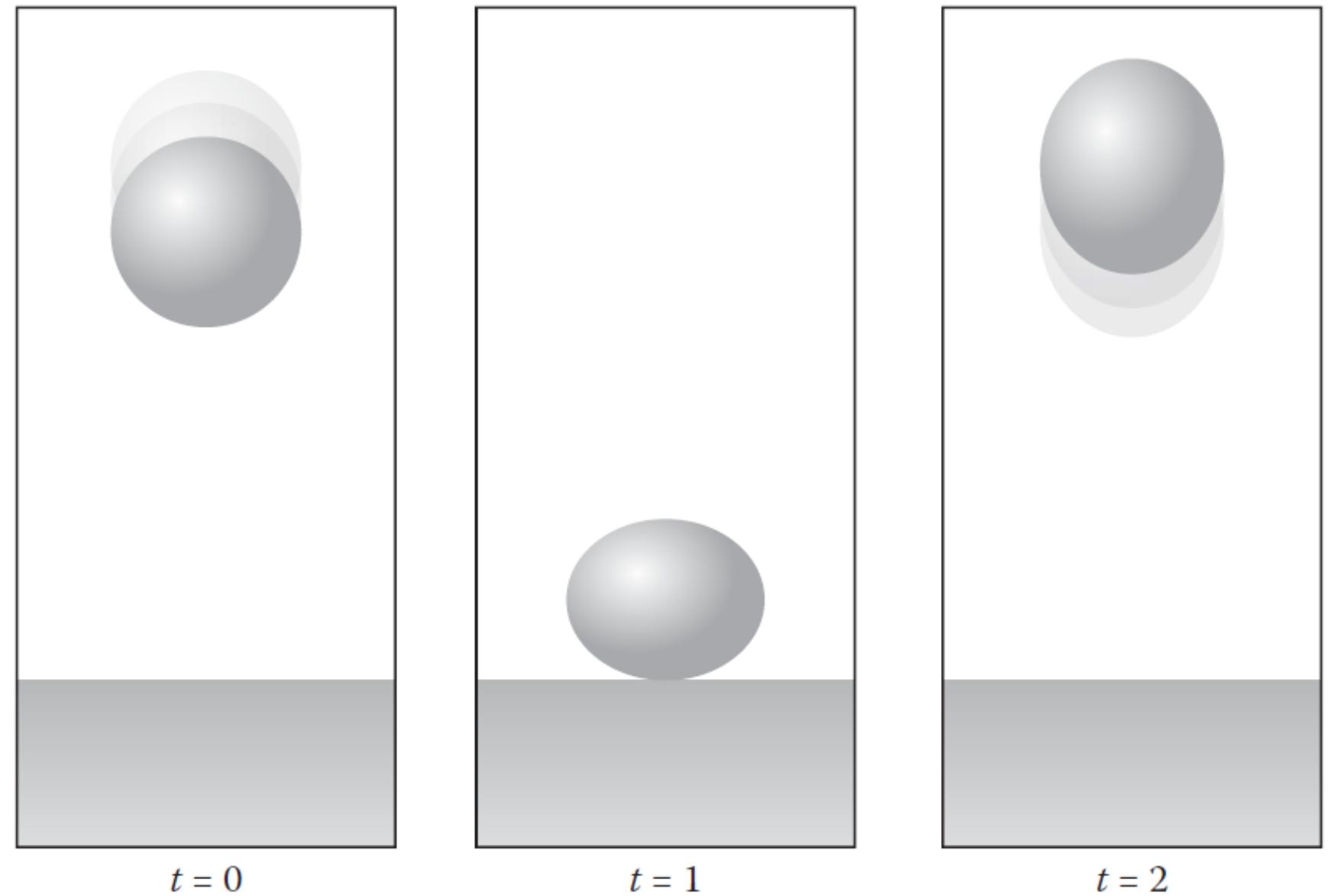
    void Advance() { ++index; }

    void Advance(int delta) { index += delta; }
};
```



애니메이션 (Animation)

```
class Animation {  
public:  
    void Update(const Frame& frame) {  
        // Some pre-processing here..  
  
        OnUpdate(frame);  
  
        // Some post-processing here..  
    }  
  
protected:  
    virtual void OnUpdate(const Frame& frame) = 0;  
};
```



물리 애니메이션 (Physics Animation)

```
class PhysicsAnimation : public Animation {  
    ...  
  
protected:  
    virtual void OnAdvancedTimeStep(double timeIntervalInSeconds) = 0;  
  
private:  
    Frame m_currentFrame;  
  
    void OnUpdate(const Frame& frame) final;  
  
    void AdvancedTimeStep(double timeIntervalInSeconds);  
};
```

물리 애니메이션 (Physics Animation)

```
void PhysicsAnimation::OnUpdate(const Frame& frame) {
    if (frame.index > m_currentFrame.index) {
        if (m_currentFrame.index < 0) {
            Initialize();
        }

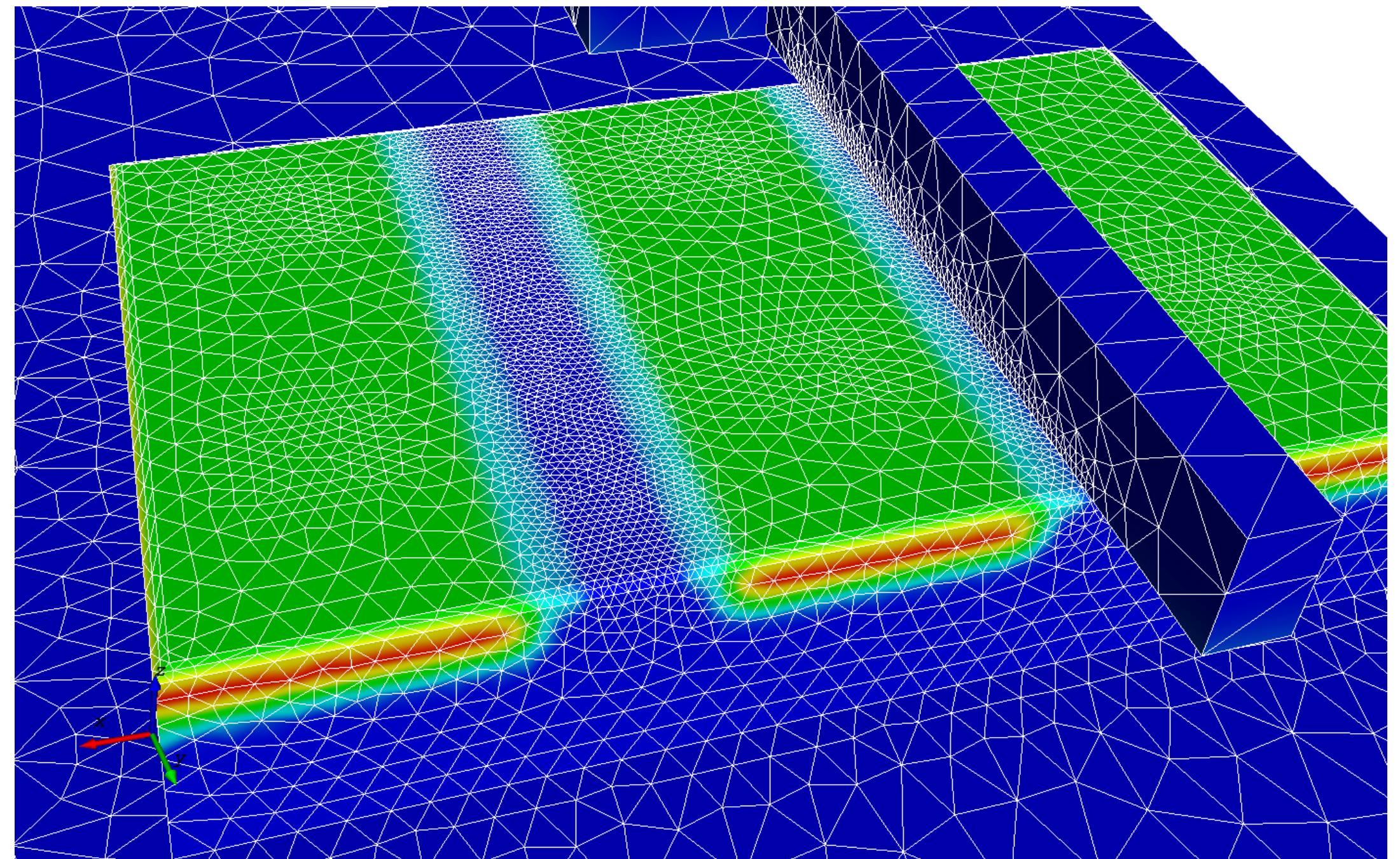
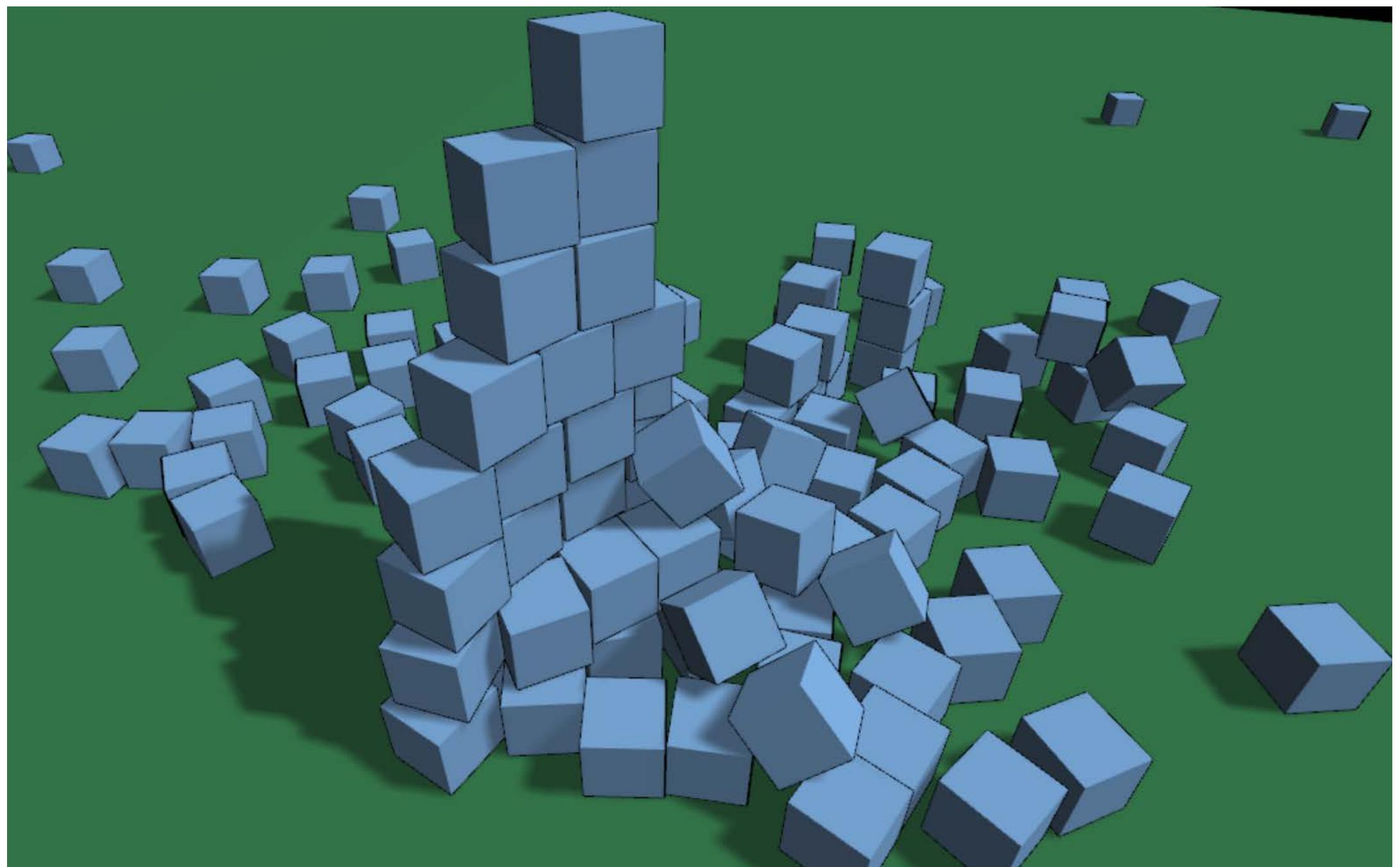
        int32_t numberOfFrames = frame.index - m_currentFrame.index;

        for (int32_t i = 0; i < numberOfFrames; ++i) {
            AdvanceTimeStep(frame.timeIntervalInSeconds);
        }
    }

    m_currentFrame = frame;
}
```

Solver

모델을 선택해 실제로 시뮬레이션 계산을 하는 부분



우리가 Solver에서 구현해야 할 것

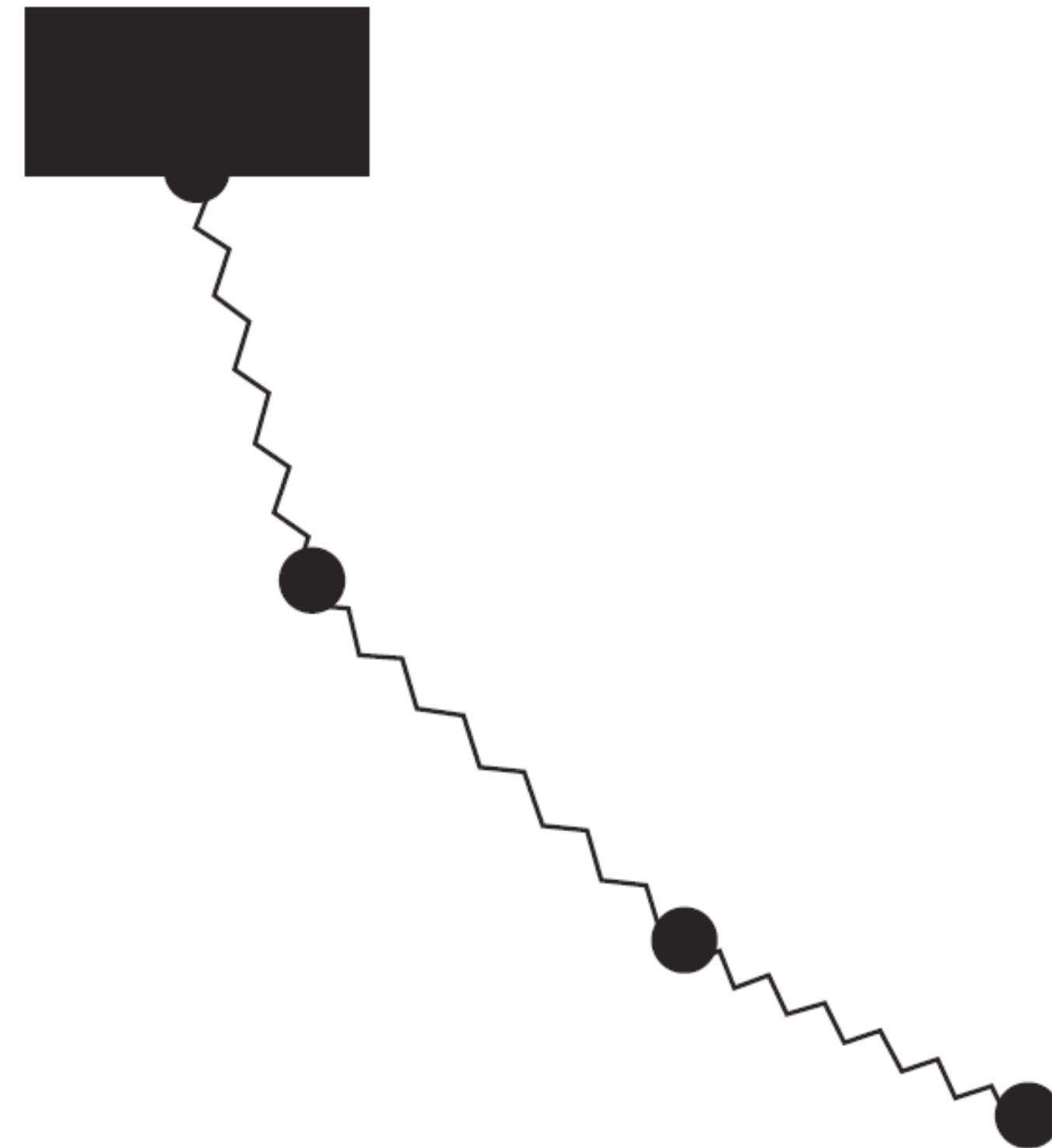
1. 물리적 상태를 어떻게 나타낼 것인가?
2. 힘을 어떻게 계산할 것인가?
3. 움직임을 어떻게 계산할 것인가?
4. 제약 조건과 장애물은 어떻게 적용할 것인가?

뭐라고 하는지 모르겠다.
그냥 **×**나 가만있어야겠다.



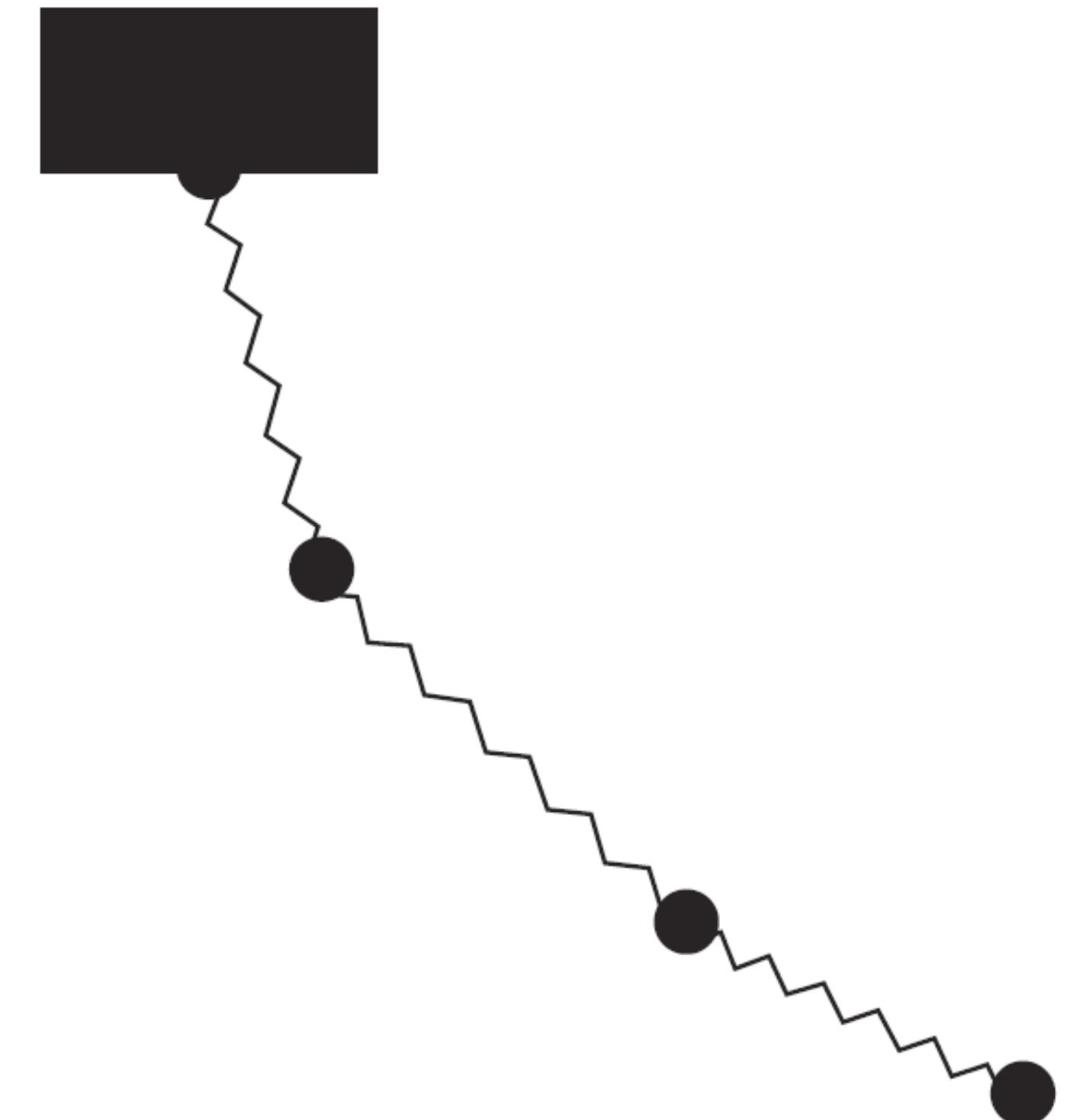
예제로 살펴봅시다. (Mass-spring System Solver)

(예제 코드를 제공해주신 김도엽님께 감사드립니다.)



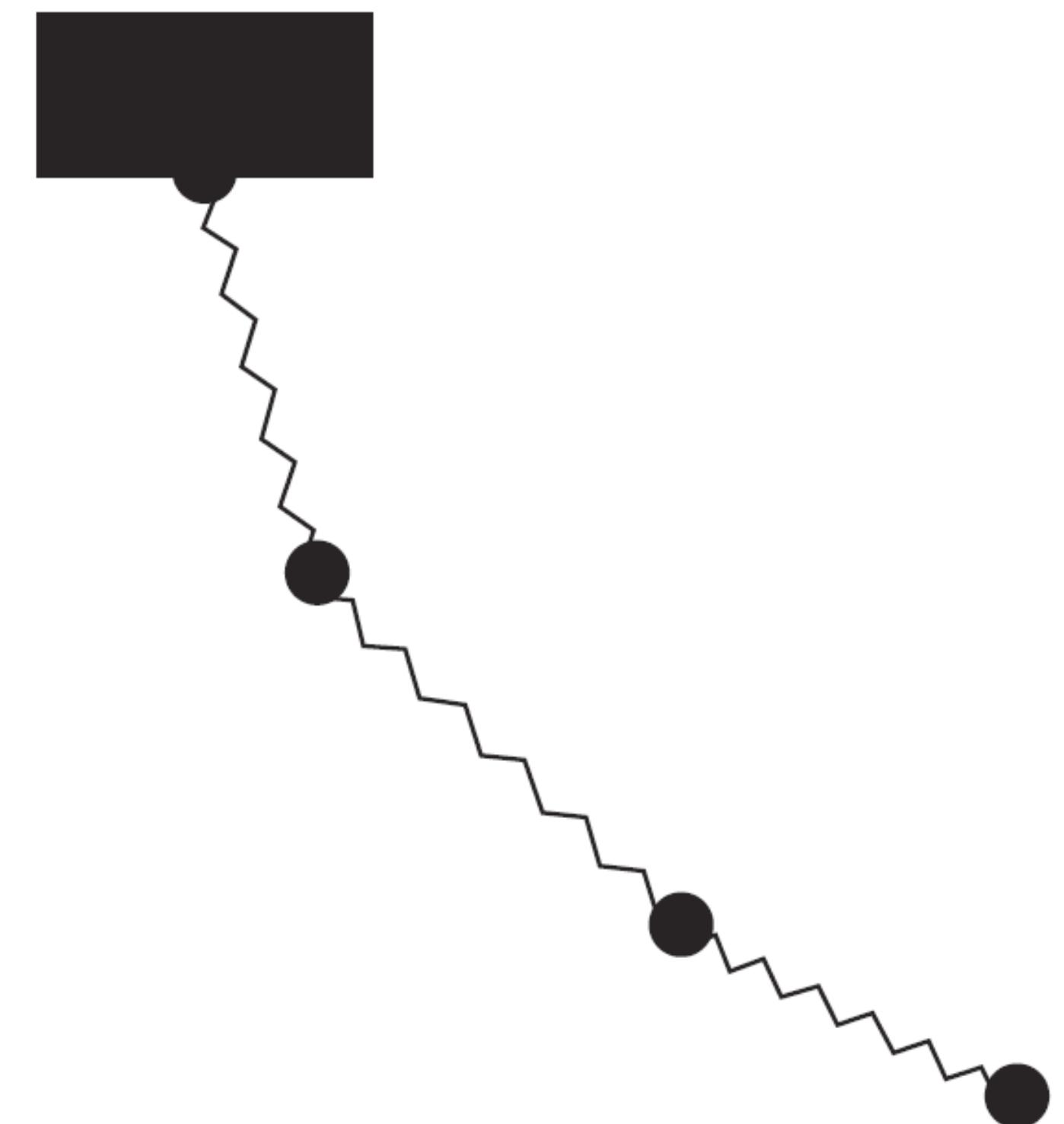
시뮬레이션 상태

```
class SimpleMassSpringAnimation : public PhysicsAnimation {  
public:  
    struct Edge {  
        size_t first;  
        size_t second;  
    };  
  
    std::vector<Vector3D> positions;  
    std::vector<Vector3D> velocities;  
    std::vector<Vector3D> forces;  
    std::vector<Edge> edges;  
};
```



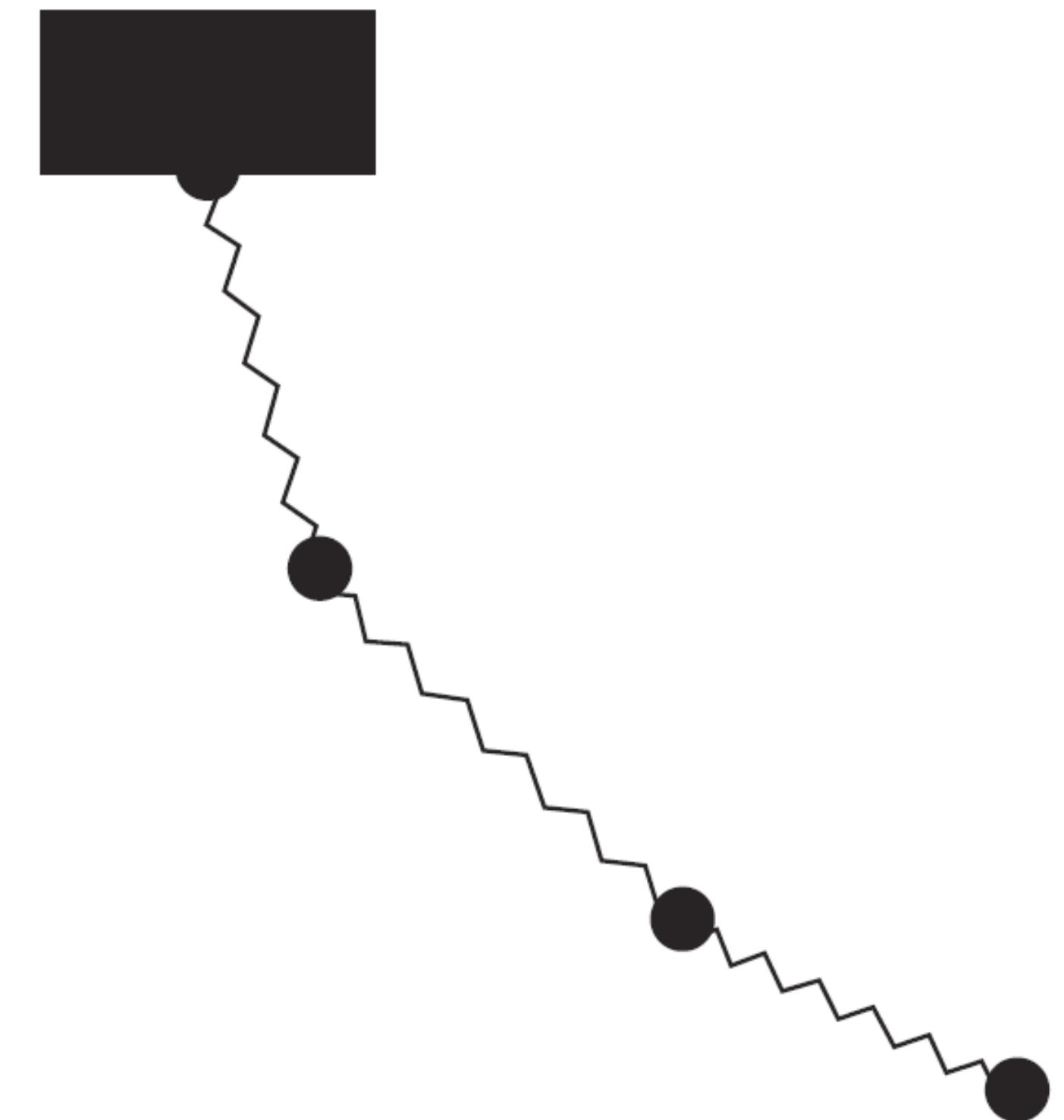
시뮬레이션 상태

```
void MakeChain(size_t numberOfPoints) {  
    if (numberOfPoints == 0) {  
        return;  
    }  
  
    size_t numberOfEdges = numberOfPoints - 1;  
  
    positions.resize(numberOfPoints);  
    velocities.resize(numberOfPoints);  
    forces.resize(numberOfPoints);  
    edges.resize(numberOfEdges);
```



시뮬레이션 상태

```
for (size_t i = 0; i < numberOfPoints; ++i) {  
    positions[i].x = -static_cast<double>(i);  
}  
  
for (size_t i = 0; i < numberOfEdges; ++i) {  
    edges[i] = Edge{ i, i + 1 };  
}  
}
```



힘과 움직임

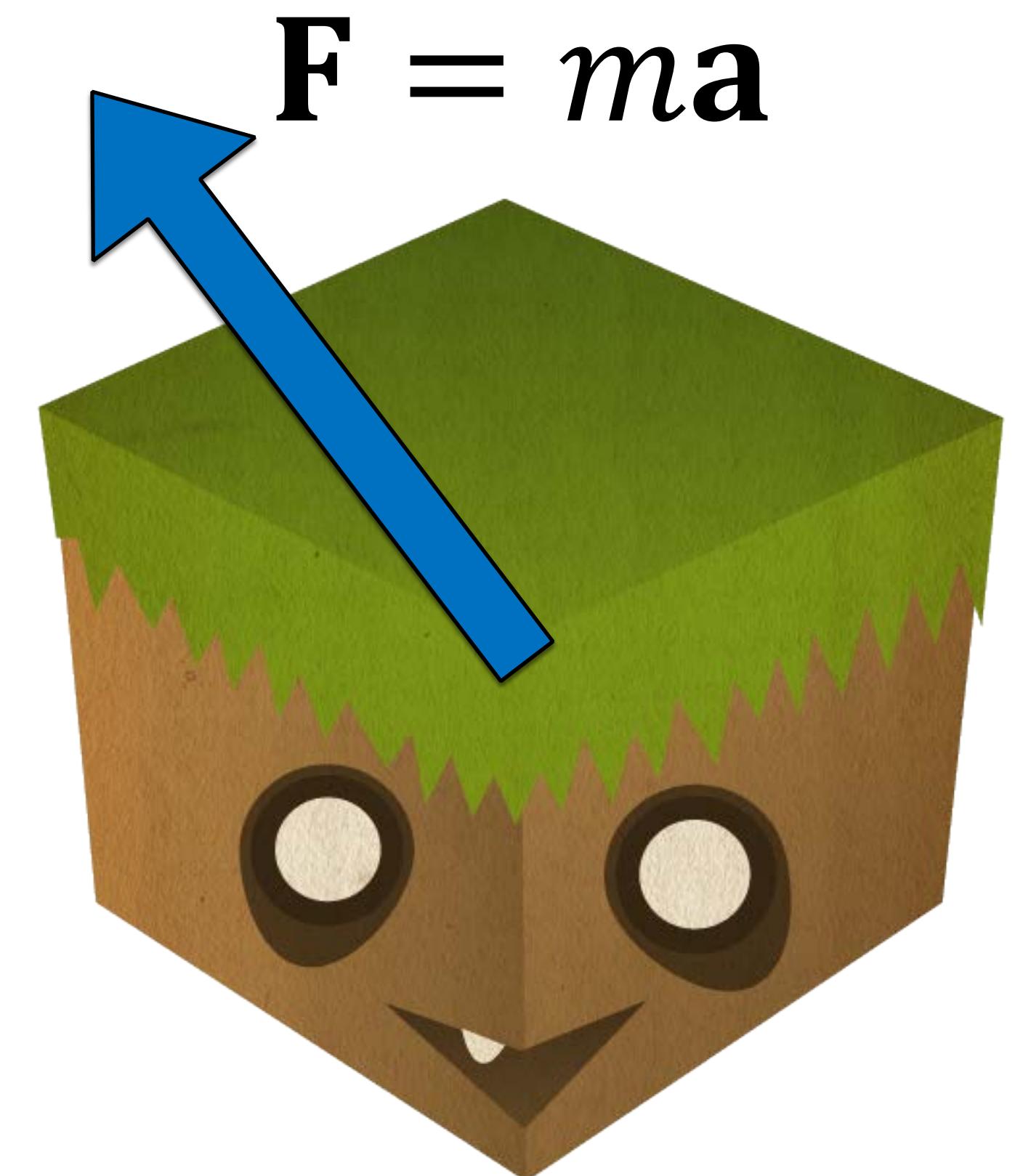
매번네
모나동
그란원
만나와
서지겨
우셨죠?



대신귀
여운마
크아이
콘을드
리겠습
니다

뉴턴 운동 제 2법칙

```
double mass = 1.0;  
  
void OnAdvanceTimeStep(double timeIntervalInSeconds) override {  
    size_t numberOfPoints = positions.size();  
  
    // Compute forces  
    for (size_t i = 0; i < numberOfPoints; ++i) {  
        forces[i] = ...  
    }  
}
```

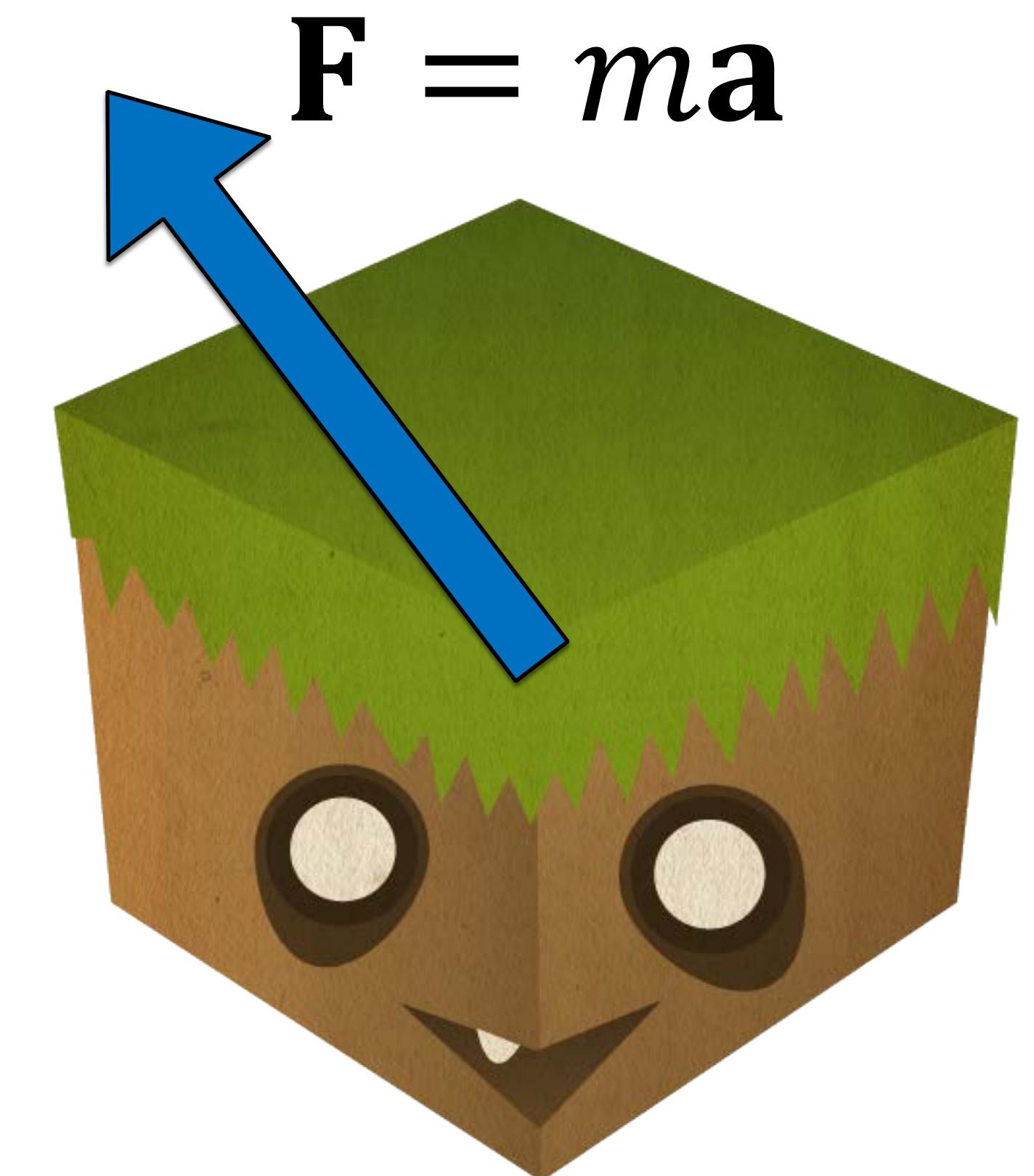


뉴턴 운동 제 2법칙

```
// Update states
for (size_t i = 0; i < numberOfPoints; ++i) {
    // Compute new states
    Vector3D newAcceleration = forces[i] / mass;
    Vector3D newVelocity = ...
    Vector3D newPosition = ...

    // Update states
    velocities[i] = newVelocity;
    positions[i] = newPosition;
}

// Apply constraints
...
}
```



중력



중력

```
Vector3D gravity = Vector3D(0.0, -9.8, 0.0);

void OnAdvanceTimeStep(double timeIntervalInSeconds) override {
    size_t numberOfPoints = positions.size();

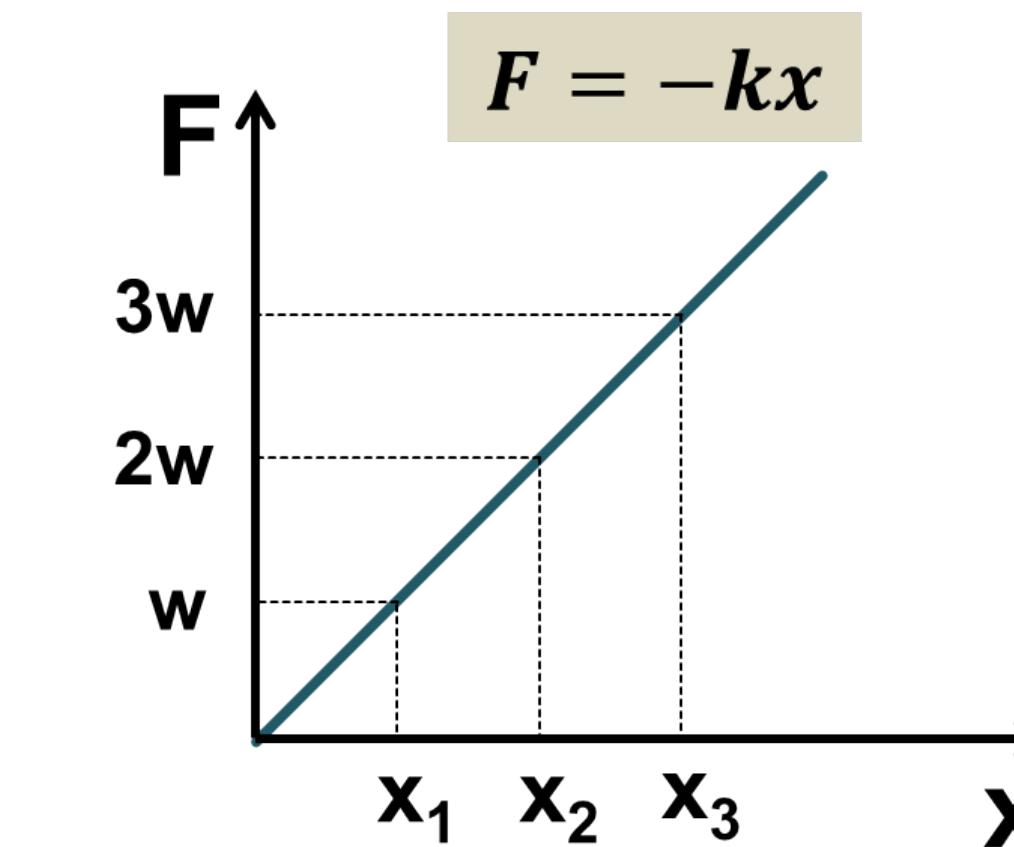
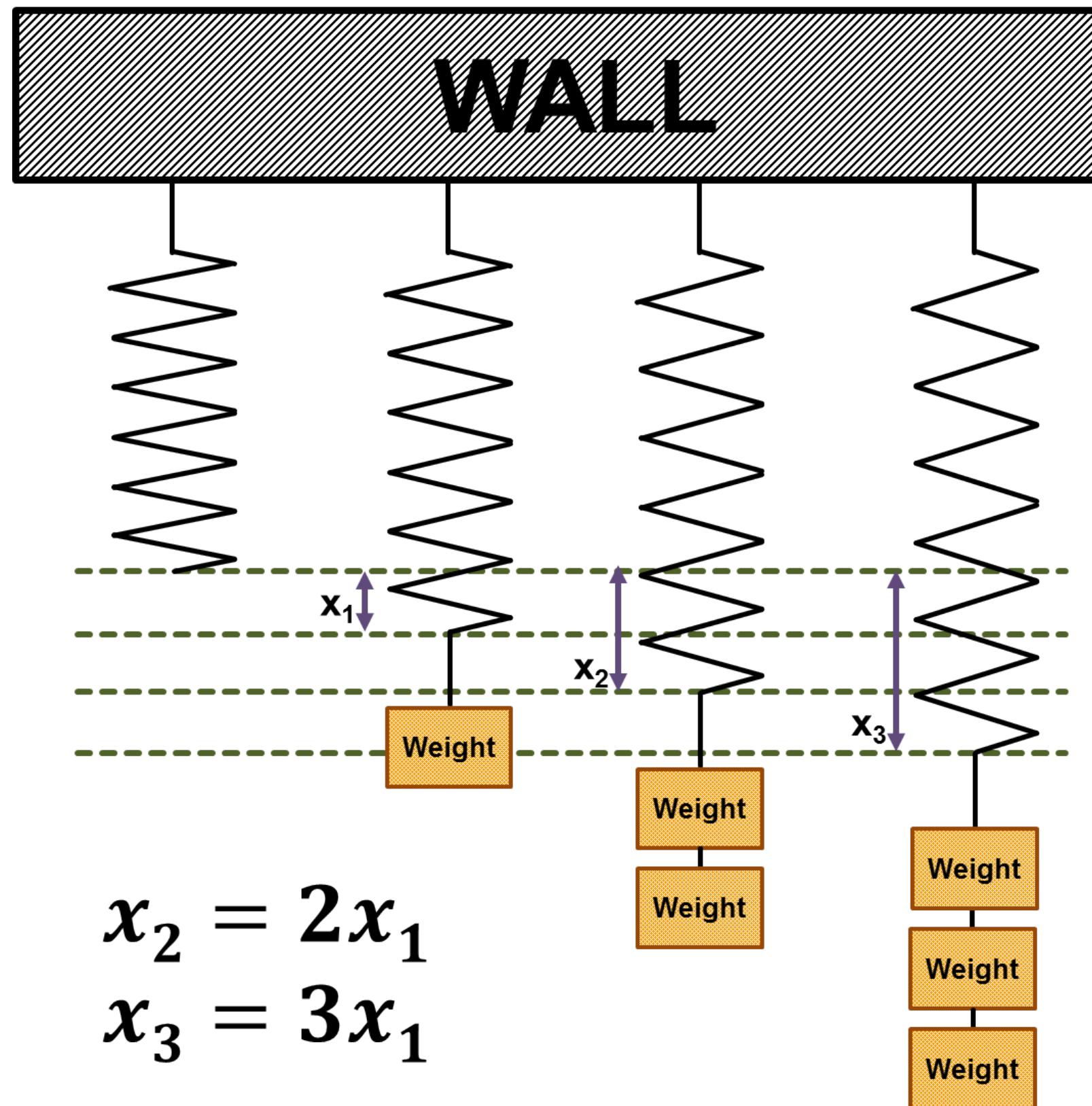
    // Compute forces
    for (size_t i = 0; i < numberOfPoints; ++i) {
        // Gravity force
        forces[i] = mass * gravity;
    }

    ...
}
```

$$\mathbf{F}_g = m\mathbf{g}$$

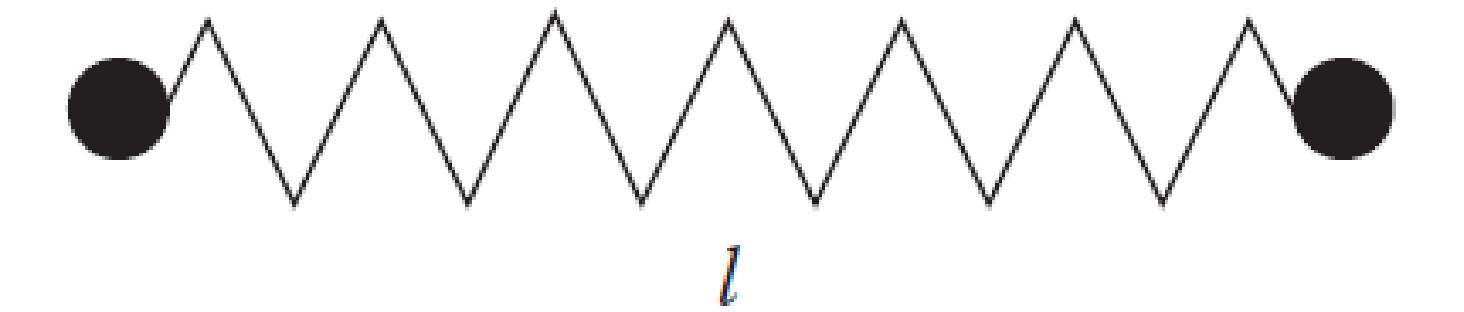
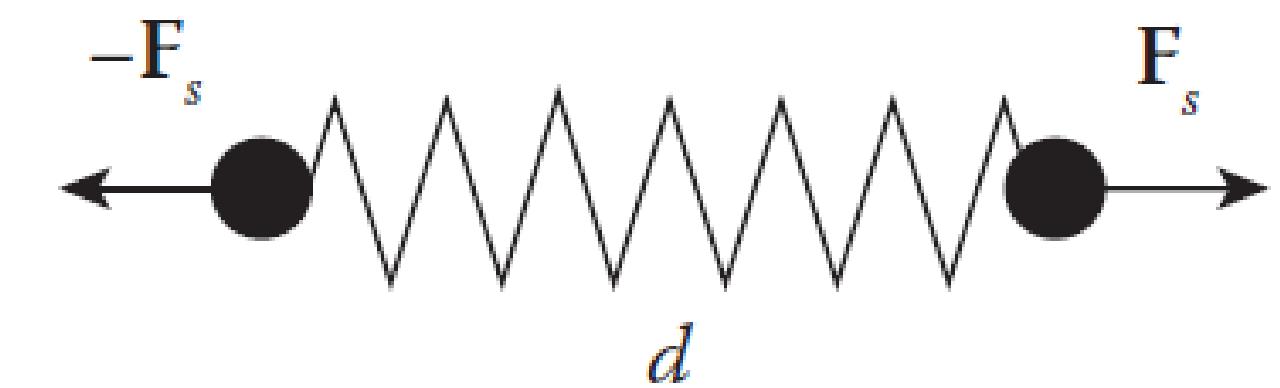


후크의 법칙 (Hooke's Law)



후크의 법칙 (Hooke's Law)

```
for (size_t i = 0; i < numberOfEdges; ++i) {  
    size_t pointIndex0 = edges[i].first;  
    size_t pointIndex1 = edges[i].second;  
    // Compute spring force  
    Vector3D pos0 = positions[pointIndex0];  
    Vector3D pos1 = positions[pointIndex1];  
    Vector3D r = pos0 - pos1;  
    double distance = r.Length();  
    if (distance > 0.0) {  
        Vector3D force = -stiffness * (distance - restLength) * r.Normalized();  
        forces[pointIndex0] += force;  
        forces[pointIndex1] -= force;  
    }  
}
```

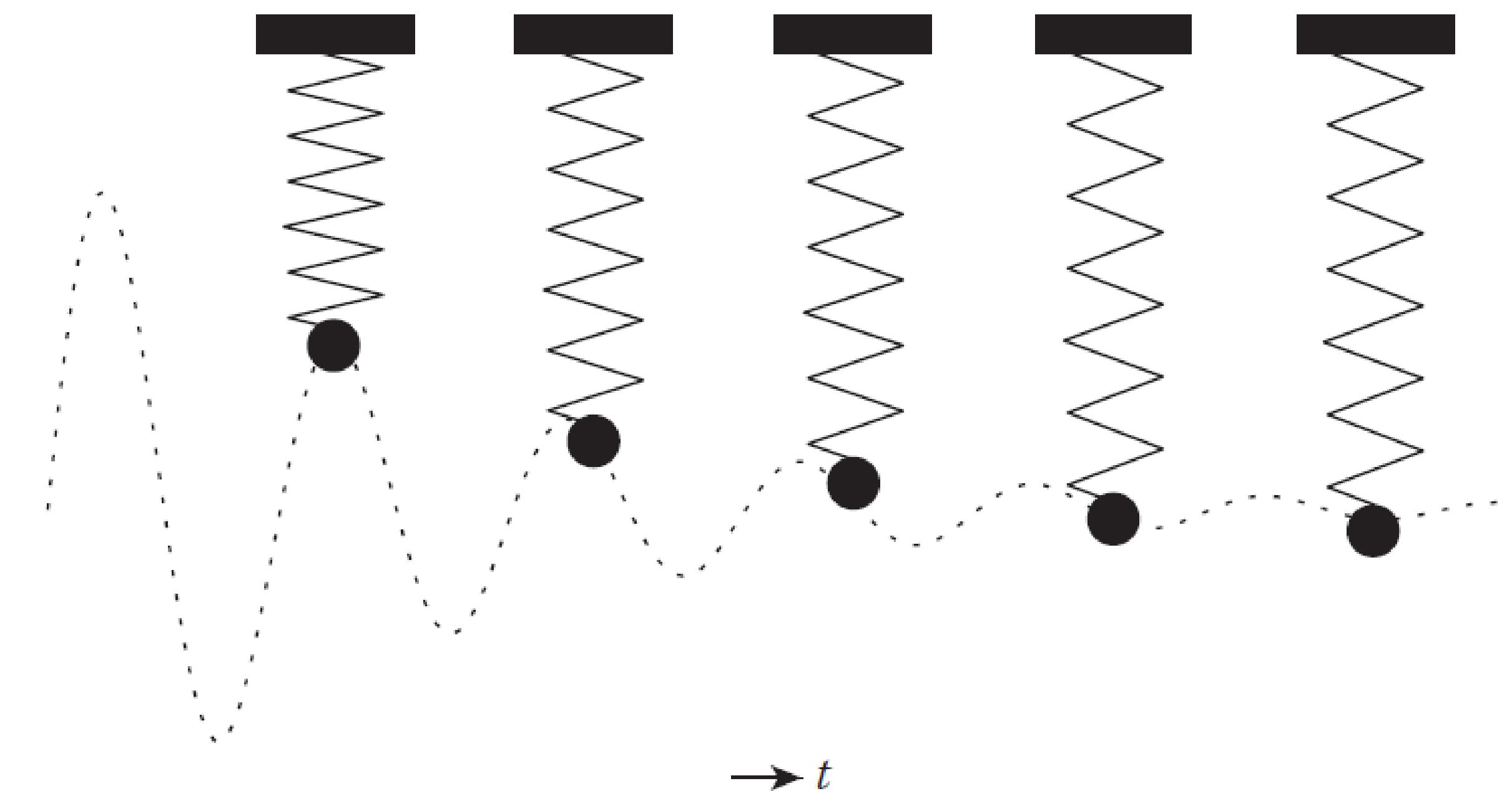


$$\mathbf{F}_{s0} = -k(d - l)\mathbf{r}_{10}$$

$$\mathbf{F}_{s1} = k(d - l)\mathbf{r}_{01}$$



감쇠



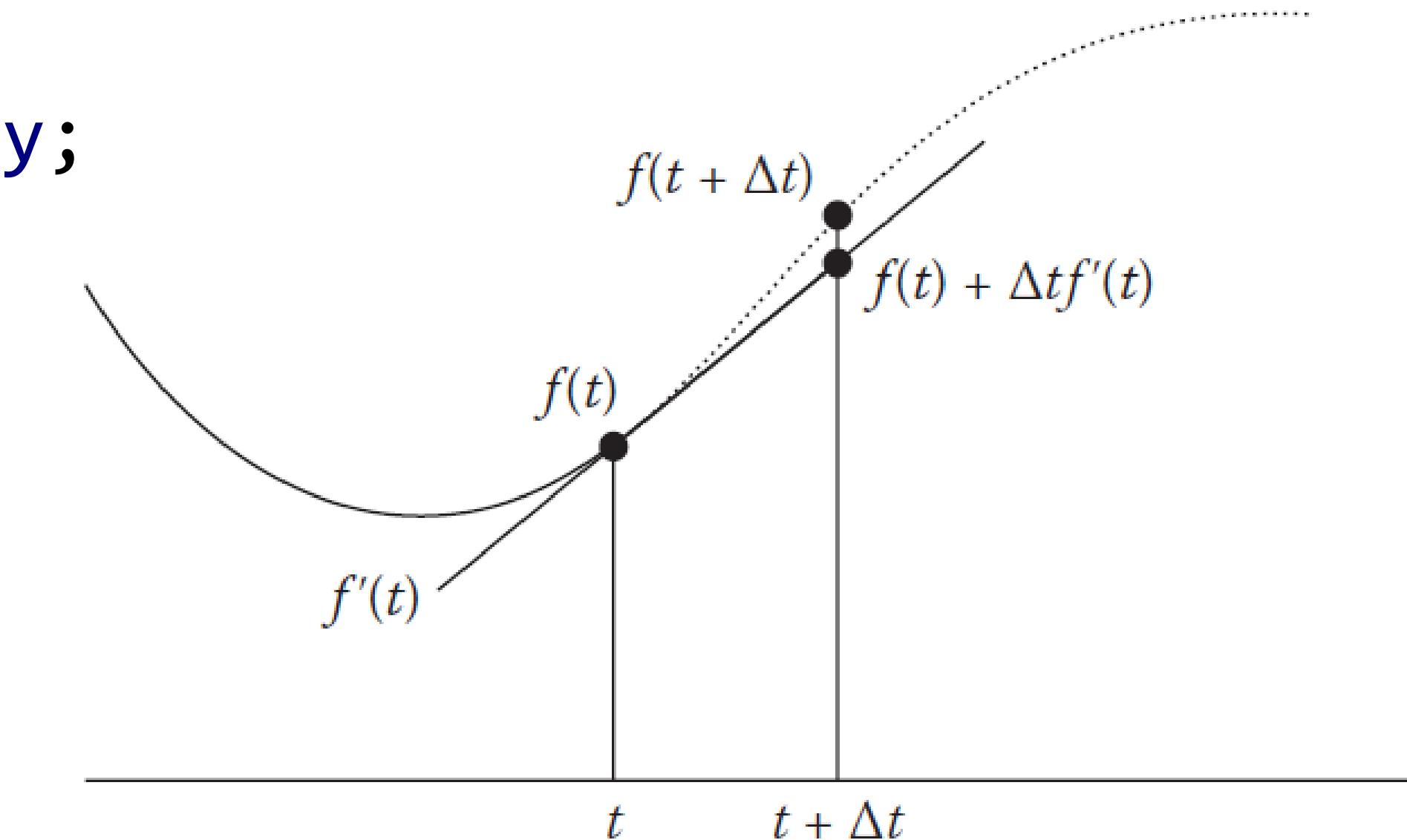
그만 알아보도록 하자



시간 통합

```
void OnAdvanceTimeStep(double timeIntervalInSeconds) override {
    // Update states
    for (size_t i = 0; i < numberOfPoints; ++i) {
        // Compute new states
        Vector3D newAcceleration = forces[i] / mass;
        Vector3D newVelocity = velocities[i] +
            timeIntervalInSeconds * newAcceleration;
        Vector3D newPosition = positions[i] +
            timeIntervalInSeconds * newVelocity;

        // Update states
        velocities[i] = newVelocity;
        positions[i] = newPosition;
    }
}
```



제약 조건

```
struct Constraint {  
    size_t pointIndex;  
    Vector3D fixedPosition;  
    Vector3D fixedVelocity;  
};  
std::vector<Constraint> constraints;  
  
void OnAdvanceTimeStep(double timeIntervalInSeconds) override {  
    // Apply constraints  
    for (size_t i = 0; i < constraints.size(); ++i) {  
        size_t pointIndex = constraints[i].pointIndex;  
        positions[pointIndex] = constraints[i].fixedPosition;  
        velocities[pointIndex] = constraints[i].fixedVelocity;  
    }  
}
```

장애물

```
double floorPositionY = -7.0;
double restitutionCoefficient = 0.3;
void OnAdvanceTimeStep(double timeIntervalInSeconds) override {
    // Update states
    for (size_t i = 0; i < numberOfPoints; ++i) {
        // Collision
        if (newPosition.y < floorPositionY) {
            newPosition.y = floorPositionY;
            if (newVelocity.y < 0.0) {
                newVelocity.y *= -restitutionCoefficient;
                newPosition.y += timeIntervalInSeconds * newVelocity.y;
            }
        }
    }
}
```

전체 소스 코드

<https://ideone.com/zafLCp>

(예제 코드를 제공해주신 김도엽님께 감사드립니다.)

유체 Solver도 마찬가지!

1. 물리적 상태를 어떻게 나타낼 것인가?

```
BoundingBox3D domain(Vector3D(), Vector3D(3, 2, 1.5));  
const double lz = domain.GetDepth();
```

```
// Build solver  
auto solver = PCISPHSolver3::GetBuilder()  
    .WithTargetDensity(1000.0)  
    .WithTargetSpacing(targetSpacing)  
    .MakeShared();
```

```
solver->SetPseudoViscosityCoefficient(0.0);  
solver->SetTimeStepLimitScale(10.0);
```

유체 Solver도 마찬가지!

1. 물리적 상태를 어떻게 나타낼 것인가?

```
// Build emitter
BoundingBox3D sourceBound(domain);
sourceBound.Expand(-targetSpacing);

const auto box1 = Box3::GetBuilder()
    .WithLowerCorner({0, 0, 0})
    .WithUpperCorner({0.5 + 0.001, 0.75 + 0.001, 0.75 * lz + 0.001})
    .MakeShared();
const auto box2 = Box3::GetBuilder()
    .WithLowerCorner({2.5 - 0.001, 0, 0.25 * lz - 0.001})
    .WithUpperCorner({3.5 + 0.001, 0.75 + 0.001, 1.5 * lz + 0.001})
    .MakeShared();
```

유체 Solver도 마찬가지!

1. 물리적 상태를 어떻게 나타낼 것인가?

```
const auto boxSet = ImplicitSurfaceSet3::GetBuilder()  
    .WithExplicitSurfaces({box1, box2})  
    .MakeShared();
```

```
const auto emitter = VolumeParticleEmitter3::GetBuilder()  
    .WithSurface(boxSet)  
    .WithMaxRegion(sourceBound)  
    .WithSpacing(targetSpacing)  
    .MakeShared();
```

```
solver->SetEmitter(emitter);
```

유체 Solver도 마찬가지!

2. 힘을 어떻게 계산할 것인가?

```
void ParticleSystemSolver3::OnAdvanceTimeStep(double timeStepInSeconds) {  
    BeginAdvanceTimeStep(timeStepInSeconds);  
  
    AccumulateForces(timeStepInSeconds);  
    TimeIntegration(timeStepInSeconds);  
    ResolveCollision();  
  
    EndAdvanceTimeStep(timeStepInSeconds);  
}
```

유체 Solver도 마찬가지!

2. 힘을 어떻게 계산할 것인가?

```
void ParticleSystemSolver3::AccumulateForces(double timeStepInSeconds) {  
    UNUSED_VARIABLE(timeStepInSeconds);  
  
    // Add external forces  
    AccumulateExternalForces();  
}
```

유체 Solver도 마찬가지!

2. 힘을 어떻게 계산할 것인가?

```
void ParticleSystemSolver3::AccumulateExternalForces() {
    ...
    ParallelFor(ZERO_SIZE, n, [&](size_t i) {
        // Gravity
        Vector3D force = mass * m_gravity;
        // Wind forces
        Vector3D relativeVel = velocities[i] - m_wind->Sample(positions[i]);
        force += -m_dragCoefficient * relativeVel;
        forces[i] += force;
    });
}
```

유체 Solver도 마찬가지!

3. 움직임을 어떻게 계산할 것인가?

```
void ParticleSystemSolver3::TimeIntegration(double timeStepInSeconds) {  
    ...  
    ParallelFor(ZERO_SIZE, n, [&](size_t i) {  
        // Integrate velocity first  
        Vector3D& newVelocity = m_newVelocities[i];  
        newVelocity = velocities[i] + timeStepInSeconds * forces[i] / mass;  
        // Integrate position.  
        Vector3D& newPosition = m_newPositions[i];  
        newPosition = positions[i] + timeStepInSeconds * newVelocity;  
    } );  
}
```

유체 Solver도 마찬가지!

4. 제약 조건과 장애물은 어떻게 적용할 것인가?

```
// Build collider
const auto box = Box3::GetBuilder()
    .WithIsNormalFlipped(true)
    .WithBoundingBox(domain)
    .MakeShared();

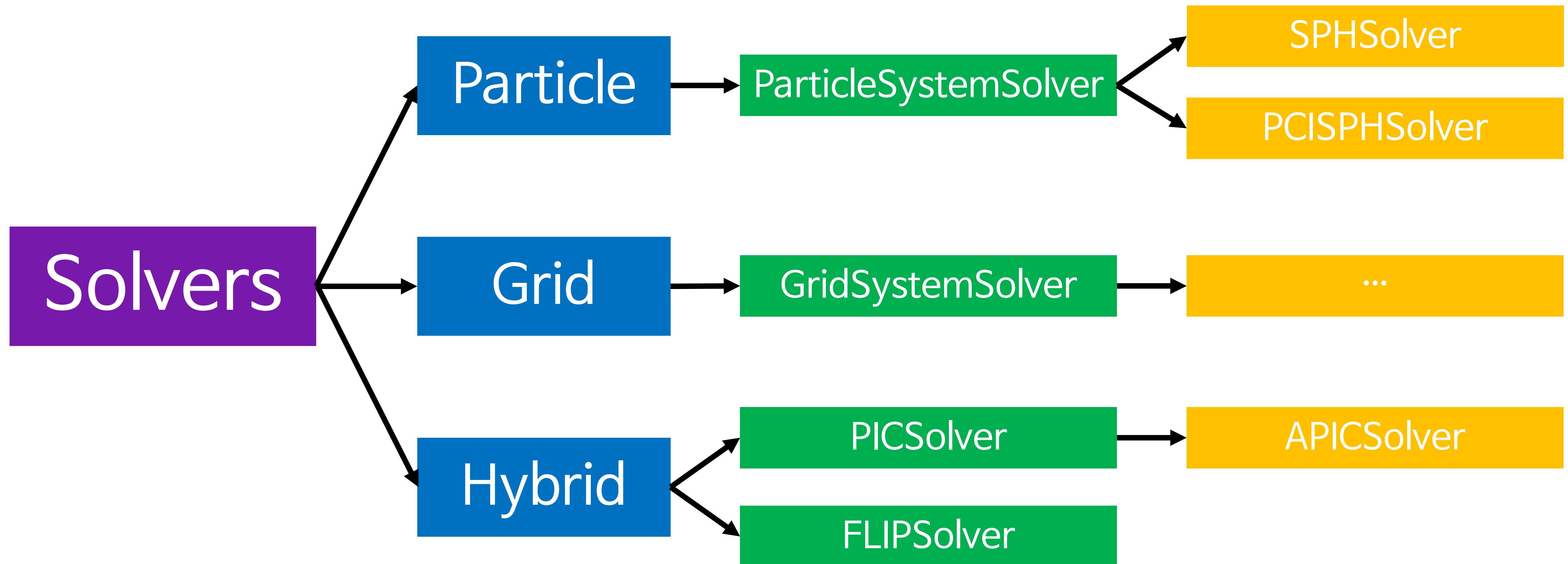
const auto collider = RigidBodyCollider3::GetBuilder()
    .WithSurface(box)
    .MakeShared();
```

유체 Solver도 마찬가지!

4. 제약 조건과 장애물은 어떻게 적용할 것인가?

```
void ParticleSystemSolver3::ResolveCollision(  
    ArrayAccessor1<Vector3D> newPositions,  
    ArrayAccessor1<Vector3D> newVelocities) {  
    if (m_collider != nullptr) {  
        ...  
        ParallelFor(ZERO_SIZE, numberOfParticles, [&](size_t i) {  
            m_collider->ResolveCollision(radius, m_restitutionCoefficient,  
                &newPositions[i], &newVelocities[i]);  
        });  
    }  
}
```

Solver 종류



3.2. 겪었던 시행착오들, 그리고 해결 과정

- 크로스 플랫폼 지원 문제
- 직렬화 문제 (Bonus*)
- 경쟁 상태 (Race Condition) 문제
- 부정확한 결과 문제 (Bonus*)
- 계산 속도 문제
- API 지원 문제

크로스 플랫폼 지원 문제

- 처음에는 Visual Studio 2017에서 개발
- 만들다 보니, 크로스 플랫폼을 지원해달라는 요청이 발생함
 - CMake 프로젝트 시스템을 사용해 보자!
- 그렇다면 VS 2017을 벗어나야겠네! (.sln, .vcxproj 의존)
- 하지만 VS 2017, 좋은데… 정말 좋은데…
 - 강력한 디버깅 도구
 - 다양한 패키지 (Visual Assist X, Resharper C++, CppDepend…)
- VS 2017에서 CMake를 사용할 수 있으면 얼마나 좋을까?



Visual C++ Team Blog

C++ tutorials, C and C++ news, and information about the C++ IDE Visual Studio from the Microsoft C++ team.

CMake support in Visual Studio



October 5, 2016 by [Marian Luparu \[MSFT\]](#) // [59 Comments](#)



⌚ Updated on **October 5, 2017** with the latest functionality included with **Visual Studio 2017 15.4**

[Visual Studio 2017](#) introduces built-in support for handling CMake projects. This makes it a lot simpler to develop C++ projects built with CMake without the need to generate VS projects and solutions from the command line. This post gives you an overview of the CMake support, how to easily get started and stay productive in Visual Studio.

<https://blogs.msdn.microsoft.com/vcblog/2016/10/05/cmake-support-in-visual-studio/>

utilForever / CubbyFlow

Code Issues 27 Pull requests 0 Projects 12 Wiki Insights Settings

Support CMake build system #209

Closed utilForever opened this issue on 4 Dec 2017 · 2 comments

utilForever commented on 4 Dec 2017 • edited

Owner +

- CubbyFlow
- cnpy
- googletest
- obj
- pystring
- winix
- UnitTests
- ManualTests
- HelloFluidSim
- HybridLiquidSim
- LevelSetLiquidSim
- Obj2Sdf
- Particles2Obj
- Particles2Xml
- SmokeSim
- SPHSim

utilForever added New Feature To-do labels on 4 Dec 2017

Assignees
utilForever

Labels
Completed
New Feature

Projects
1 closed project (show)

Milestone
Ver 0.64

Notifications
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Title Bar:** CubbyFlow - Microsoft Visual Studio
- Menu Bar:** File, Edit, View, VAssistX, Project, CMake, Build, Incredibuild, Debug, Team, Nsight, Tools, Architecture, Test, CppDepend, ReSharper, Analyze, Window, Help
- Toolbar:** Standard icons for file operations like Open, Save, Print, etc.
- Solution Explorer:** Shows the project structure under "CubbyFlow".
 - Libraries
 - Medias
 - Resources
 - Sources
 - API
 - Core
 - Animation
 - Animation.cpp
 - Frame.cpp
 - PhysicsAnimation.cpp
 - Collider
 - Emitter
 - FDM
 - Field
 - Flatbuffers
 - Geometry
 - Grid
 - MarchingCubes
 - Particle
 - PointGenerator
 - PointsToImplicit
 - Searcher
 - SemiLagrangian
 - Solver
 - Advection
 - FDM
 - Grid
 - Hybrid
 - LevelSet
 - Particle
 - PCISPH
 - SPH
 - ParticleSystemSolver2.cpp
 - ParticleSystemSolver3.cpp
 - SPH
 - Surface
 - Transform
 - Utils
- Current Document:** CMakeLists.txt
- Code Editor:** Displays the CMakeLists.txt file content.

```
1 # CMake version
2 cmake_minimum_required(VERSION 3.8.2 FATAL_ERROR)
3
4 # Include cmake modules
5 list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_SOURCE_DIR}/Builds/CMake")
6
7 # Declare project
8 project(CubbyFlow)
9
10 # Set output directories
11 set(DEFAULT_CMAKE_LIBRARY_OUTPUT_DIRECTORY ${CMAKE_LIBRARY_OUTPUT_DIRECTORY})
12 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_BINARY_DIR}/bin)
13 set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${PROJECT_BINARY_DIR}/lib)
14 set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${PROJECT_BINARY_DIR}/lib)
15
16 # Set enable output of compile commands during generation
17 set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
18
19 # Includes
20 include_directories(Includes)
21 include_directories(Libraries)
22 include_directories(Libraries/obj)
23 include_directories(Libraries/pybind11/include)
24 include_directories(Libraries/googletest/googletest/include)
25 if(CMAKE_CXX_COMPILER_ID MATCHES "MSVC")
26   include_directories(Libraries/winix)
27 endif()
28
29 # Find TBB
30 include(Builds/CMake/FindTBB.cmake)
31
32 # Tasking system options
33 include(Builds/CMake/TaskingSystemOptions.cmake)
34
35 # Compile options
36 include(Builds/CMake/CompileOptions.cmake)
37
```



크로스 플랫폼 지원 문제

- 처음에는 CMake 문법이 생소하게 느껴집니다.
- 다음 사이트를 참고하면 많은 도움이 됩니다.
 - <https://blogs.msdn.microsoft.com/vcblog/2016/10/05/cmake-support-in-visual-studio/>
 - <https://cmake.org/cmake/help/v3.11/>
 - <https://github.com/onqtam/awesome-cmake>
- 플랫폼마다 사용하는 컴파일러가 다릅니다.
따라서 조건문으로 처리해 서로 다른 옵션을 사용해야 합니다.
- (필수) 플랫폼마다 제대로 빌드되는지 테스트해야 합니다.
- 잘 만들어두면 손쉽게 크로스 플랫폼을 지원할 수 있습니다.

컴파일러에 따라 다른 옵션 사용

```
# Set warnings as errors flag
option(CUBBYFLOW_WARNINGS_AS_ERRORS ON)
if(CUBBYFLOW_WARNINGS_AS_ERRORS)
    if(CMAKE_CXX_COMPILER_ID MATCHES "MSVC")
        set(WARN_AS_ERROR_FLAGS "/WX")
    else()
        set(WARN_AS_ERROR_FLAGS "-Werror")
    endif()
endif()
```

경쟁 상태 (Race Condition) 문제

- CubbyFlow는 크로스 플랫폼 지원 프로젝트 (Windows, Linux, macOS 등)
- 평소에는 Windows와 Linux 플랫폼에서 테스트를 진행
 - UnitTests : 기능별 단위 테스트
 - ManualTests : 유체 시뮬레이션 테스트
 - MemPerfTests : 메모리 사용량 테스트
 - TimePerfTests : 시간 측정 테스트
- 별다른 문제가 없어 평화로운 나날을 보내고 있었습니다.
적어도 아는 동생이 집에 찾아오기 전까지는...

너 맥북 있어?

네, 이번에 샀어요 ㅎㅎ

오후 5:17

… 뭔데요?

오후 5:17

네. 치킨 사주실거죠? ^^

오후 5:17

1 오후 5:17

잘됐다! 나 뭐 하나만 부탁해도 되니?

1 오후 5:17

프로그램 하나만 빌드 좀 해보자.

* 각색한 내용입니다.

잠시 후...

“형, 이거 안돼는데요?”

```
[-----] 4 tests from APICSolver3Tests
[ RUN      ] APICSolver3Tests.WaterDrop
[      OK  ] APICSolver3Tests.WaterDrop (33582 ms)
[ RUN      ] APICSolver3Tests.DamBreakingWithCollider
ManualTests(4353,0x700008db7000) malloc: * error for
object 0x7ff9a84197b0: double free
* set a breakpoint in malloc_error_break to debug
[1] 4353 abort      ./ManualTests
```



**“잉, 그럴리가 없는데…
다시 한번 해볼래?”**

```
[-----] Global test environment tear-down
[=====] 149 tests from 58 test cases ran.
(1274768 ms total)
[ PASSED ] 149 tests.
```



잠시 후...

“형, 이거 또 안돼는데요?”

```
[-----] 4 tests from FLIPSolver3Tests
[ RUN      ] FLIPSolver3Tests.WaterDrop
[      OK  ] FLIPSolver3Tests.WaterDrop (32374 ms)
[ RUN      ] FLIPSolver3Tests.DamBreakingWithCollider
ManualTests(2837,0x70000209b000) malloc: * error for
object 0x7ff84f900038: incorrect checksum for freed
object - object was probably modified after being
freed.
* set a breakpoint in malloc_error_break to debug
[1] 2837 abort      ./ManualTests
```



현상

- Windows, Linux에서는 안생기는 오류가 **macOS**에서만 생깁니다.
- 항상 발생하지는 않고 **“간헐적”**으로 발생합니다.
세상에서 가장 싫어하는 말 중 하나 -_-
- 오류 메시지의 종류가 **여러가지**입니다.
- 이 문제를 해결하려면 맥북이 반드시 필요합니다.
- 어, 근데 난 맥북이 없는데…?

동생에게 다시 부탁해봅시다.

... 집에서 여기까지 너무 먼데;;

오후 5:17

어, 정말요?

오후 5:17

피자도 사주시면… 흥

오후 5:17

1 오후 5:17

너 주말마다 우리집 오면 안돼냐?

1 오후 5:17

대신 올 때마다 치킨사줄게!!!

1 오후 5:17

응, 제발… ㅠ

* 각색한 내용입니다.

**그렇게 주말에 다시 만났습니다.
이제 디버깅을 해봅시다.**

```
§ CubbyFlow::ImplicitSurfaceSet3::BuildBVH() const ImplicitSurfaceSet3.cpp:179
§ CubbyFlow::ImplicitSurfaceSet3::ClosestDistanceLocal(CubbyFlow::Vector<double, 3ul> const&) const ImplicitSurfaceSet3.cpp:83
§ CubbyFlow::Surface3::ClosestDistance(CubbyFlow::Vector<double, 3ul> const&) const Surface3.cpp:47
§ CubbyFlow::Collider3::GetClosestPoint(std::__1::shared_ptr<CubbyFlow::Surface3> const&, CubbyFlow::Vector<double, 3ul> const&, CubbyFlow::Collider3::ColliderQueryResult*) const Collider3.cpp:91
§ CubbyFlow::Collider3::ResolveCollision(double, double, CubbyFlow::Vector<double, 3ul>*, CubbyFlow::Vector<double, 3ul>*) Collider3.cpp:27
§ CubbyFlow::PICSolver3::MoveParticles(double)::$__5::operator()(unsigned long) const PICSolver3.cpp:297
§ void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::PICSolver3::MoveParticles(double)::$__5>(unsigned long, unsigned long, CubbyFlow::PICSolver3::MoveParticles(double)::$__5 const&, CubbyFlow::E
§ void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::F
§ void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::F
§ void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::F
§ _pthread_body 0x00007fff7d3236c1
§ _pthread_start 0x00007fff7d32356d
§ thread_start 0x00007fff7d322c5d
```

```
§ CubbyFlow::ImplicitSurfaceSet3::BuildBVH() const ImplicitSurfaceSet3.cpp:179
§ CubbyFlow::ImplicitSurfaceSet3::ClosestDistanceLocal(CubbyFlow::Vector<double, 3ul> const&) const ImplicitSurfaceSet3.cpp:83
§ CubbyFlow::Surface3::ClosestDistance(CubbyFlow::Vector<double, 3ul> const&) const Surface3.cpp:47
§ CubbyFlow::Collider3::GetClosestPoint(std::__1::shared_ptr<CubbyFlow::Surface3> const&, CubbyFlow::Vector<double, 3ul> const&, CubbyFlow::Collider3::ColliderQueryResult*) const Collider3.cpp:91
§ CubbyFlow::Collider3::ResolveCollision(double, double, CubbyFlow::Vector<double, 3ul>*, CubbyFlow::Vector<double, 3ul>*) Collider3.cpp:27
§ CubbyFlow::PICSolver3::MoveParticles(double)::$__5::operator()(unsigned long) const PICSolver3.cpp:297
§ void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::PICSolver3::MoveParticles(double)::$__5>(unsigned long, unsigned long, CubbyFlow::PICSolver3::MoveParticles(double)::$__5 const&, CubbyFlow::E
§ void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::F
§ void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::F
§ void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::F
§ _pthread_body 0x00007fff7d3236c1
§ _pthread_start 0x00007fff7d32356d
§ thread_start 0x00007fff7d322c5d
```

멀티스레드 문제

문제가 발생한 곳

```
§ CubbyFlow::ImplicitSurfaceSet3::BuildBVH() const ImplicitSurfaceSet3.cpp:179
§ CubbyFlow::ImplicitSurfaceSet3::ClosestDistanceLocal(CubbyFlow::Vector<double, 3ul> const&) const ImplicitSurfaceSet3.cpp:83
§ CubbyFlow::Surface3::ClosestDistance(CubbyFlow::Vector<double, 3ul> const&) const Surface3.cpp:47
§ CubbyFlow::Collider3::GetClosestPoint(std::__1::shared_ptr<CubbyFlow::Surface3> const&, CubbyFlow::Vector<double, 3ul> const&, CubbyFlow::Collider3::ColliderQueryResult*) const Collider3.cpp:91
§ CubbyFlow::Collider3::ResolveCollision(double, double, CubbyFlow::Vector<double, 3ul>*, CubbyFlow::Vector<double, 3ul>*) Collider3.cpp:27
§ CubbyFlow::PICSolver3::MoveParticles(double)::$__5::operator()(unsigned long) const PICSolver3.cpp:297
§ void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::PICSolver3::MoveParticles(double)::$__5>(unsigned long, unsigned long, CubbyFlow::PICSolver3::MoveParticles(double)::$__5 const&, CubbyFlow::E
§ void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::F
§ void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::F
§ void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void CubbyFlow::ParallelFor<unsigned long, CubbyFlow::F
§ _pthread_body 0x00007fff7d3236c1
§ _pthread_start 0x00007fff7d32356d
§ thread_start 0x00007fff7d322c5d
```

멀티스레드 문제

```
// Build collider
auto cyl1 = Cylinder3::Builder()
    .WithCenter({ 1, 0.375, 0.375 })
    .WithRadius(0.1).WithHeight(0.75)
    .MakeShared();
auto cyl2 = Cylinder3::Builder()
    .WithCenter({ 1.5, 0.375, 0.75 })
    .WithRadius(0.1).WithHeight(0.75)
    .MakeShared();
auto cyl3 = Cylinder3::Builder()
    .WithCenter({ 2, 0.375, 1.125 })
    .WithRadius(0.1).WithHeight(0.75)
    .MakeShared();
```

```
auto cylSet = ImplicitSurfaceSet3::Builder()
    .WithExplicitSurfaces({ cyl1, cyl2, cyl3 })
    .MakeShared();

auto collider = RigidBodyCollider3::Builder()
    .WithSurface(cylSet)
    .MakeShared();
```

```
void PICSolver3::MoveParticles(double timeIntervalInSeconds)
{
    ...
    Collider3Ptr col = GetCollider();
    if (col != nullptr)
    {
        ParallelFor(ZERO_SIZE, numberOfParticles, [&](size_t i)
        {
            col->ResolveCollision(0.0, 0.0, &positions[i], &velocities[i]);
        });
    }
}
```

```
double ImplicitSurfaceSet3::ClosestDistanceLocal(const Vector3D& otherPoint) const
{
    BuildBVH();

    const auto distanceFunc = [] (const Surface3Ptr& surface, const Vector3D& pt)
    {
        return surface->ClosestDistance(pt);
    };

    const auto queryResult = m_bvh.GetNearestNeighbor(otherPoint, distanceFunc);
    return queryResult.distance;
}
```

```
void SurfaceSet3::BuildBVH() const
{
    if (m_bvhInvalidated)
    {
        std::vector<BoundingBox3D> bounds(m_surfaces.size());
        for (size_t i = 0; i < m_surfaces.size(); ++i)
        {
            bounds[i] = m_surfaces[i]->BoundingBox();
        }

        m_bvh.Build(m_surfaces, bounds);
        m_bvhInvalidated = false;
    }
}
```

```
template <typename T>
size_t BVH3<T>::Build(size_t nodeIndex, size_t* itemIndices,
    size_t nItems, size_t currentDepth)
{
    ...
    // recursively Initialize children m_nodes
    size_t d0 = Build(nodeIndex + 1, itemIndices,
        midPoint, currentDepth + 1);
    m_nodes[nodeIndex].InitInternal(axis, m_nodes.size(), nodeBound);
    size_t d1 = Build(m_nodes[nodeIndex].child, itemIndices + midPoint,
        nItems - midPoint, currentDepth + 1);

    return std::max(d0, d1);
}
```

분석

- PICSolver::MoveParticles()에서 일정 입자 개수마다 스레드를 생성
- 각 스레드는 Collider::ResolveCollision()을 호출
- Collider::ResolveCollision()은 Collider::GetClosestPoint()를 호출
- Collider::GetClosestPoint()은 Surface::ClosestDistance()을 호출
- Surface::ClosestDistance()는 ImplicitSurfaceSet::ClosestDistanceLocal()을 호출
- ImplicitSurfaceSet::ClosestDistanceLocal()에서 BuildBVH()를 호출
- ImplicitSurfaceSet::BuildBVH()에서 m_bvh.Build()를 호출
- BVH::Build()에서 Build()를 재귀 호출
- 재귀 호출 과정에서 스레드 실행 순서에 따라 힘이 깨질 수 있음 (!!)

해결

- ImplicitSurfaceSet에서 BuildBVH()를 호출하면 안된다. (실행 순서 보장 X)
- BuildBVH()를 호출하는 로직을 스레드 바깥으로 옮기자. → Collider::Update()

```
void Collider3::Update(double currentTimeInSeconds, double timeIntervalInSeconds) {
    m_surface->UpdateQueryEngine();

    if (m_onUpdateCallback) {
        m_onUpdateCallback(this, currentTimeInSeconds, timeIntervalInSeconds);
    }
}

void ImplicitSurfaceSet3::UpdateQueryEngine() { BuildBVH(); }
```

```
[-----] Global test environment tear-down
[=====] 149 tests from 58 test cases ran.
(1274768 ms total)
[ PASSED ] 149 tests.
```

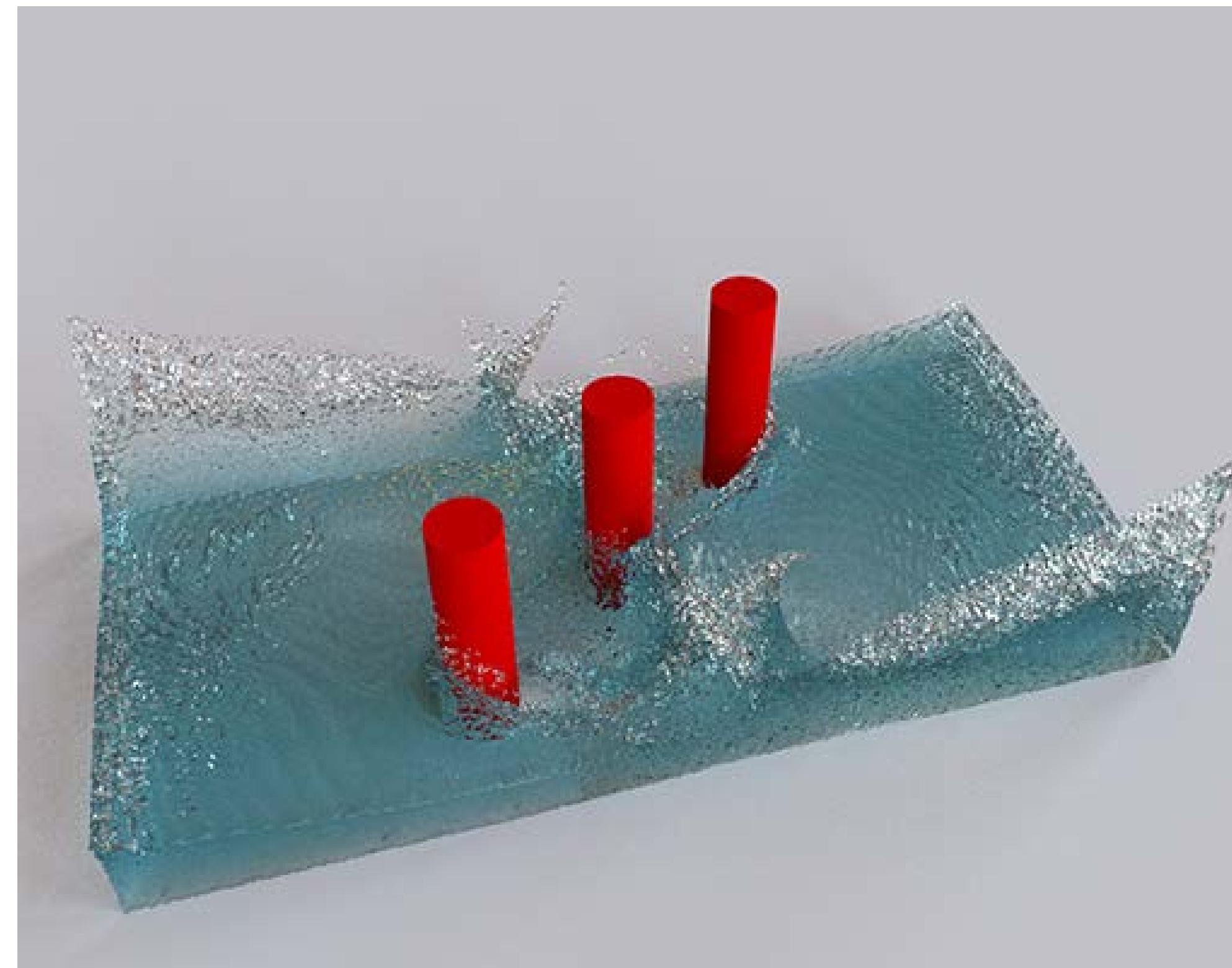


계산 속도 문제

- 이런 저런 큰 문제는 다 해결한 거 같습니다.
- 문제는 시뮬레이션 실행 속도가…

계산 속도 문제

- DamBreakingWithCollider 예제를 시뮬레이션한다고 가정해봅시다.



계산 속도 문제

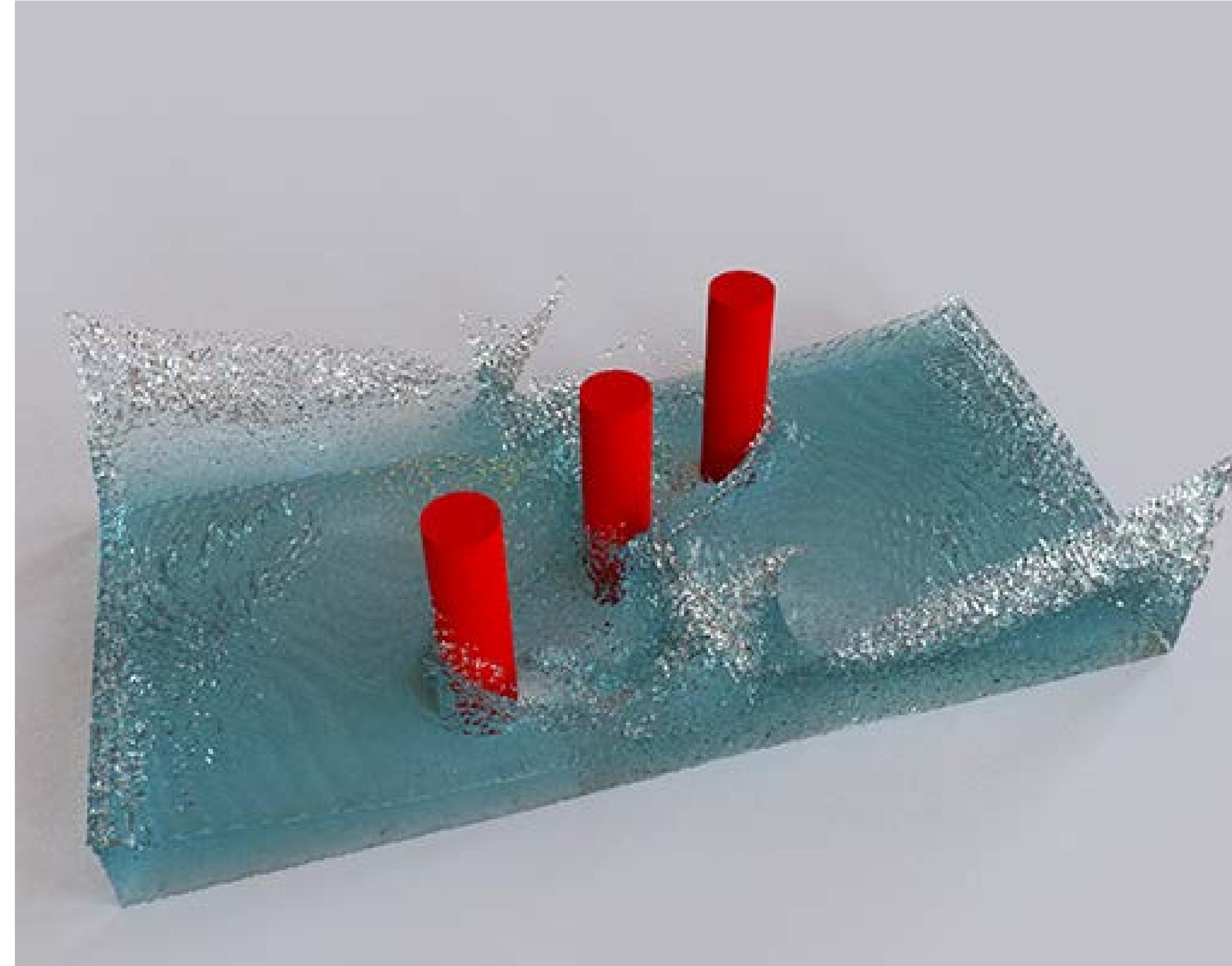
- 전 예제를 시뮬레이션했을 때 40프레임을 계산하는데 걸리는 시간이
 - Grid-based (Level-set) : 약 2~3시간 (…?)
 - Hybrid-based (PIC, FLIP, APIC) : 약 2~3시간 (…?)
 - Particle-based (PCISPH) : 약 2일 (?????????????)

어, 그러니까…

게임은 1초에 60프레임이 나와야 하는데



시뮬레이션은 3일에 60프레임이라구요?





계산 속도 문제

- 아, 물론 제 컴퓨터 성능이 그리 좋지 못합니다. (2015년 9월 구입;;)
- 하지만 느려도 너무 느린 거 같습니다.
- 어떻게 개선할 수 없을까요?

CPU 병렬 프로그래밍

- CubbyFlow는 C++11의 std::thread를 사용하고 있습니다.
- 하지만 CPU 병렬 프로그래밍을 할 수 있다면,
비약적인 성능 향상을 이뤄낼 수 있지 않을까 생각했습니다.
- 그렇다면, 우리가 사용할 수 있는 라이브러리는 무엇이 있을까요?
 - Intel TBB (<https://www.threadingbuildingblocks.org/>)
(2017년에 Apache 2.0 라이선스로 전환)
 - OpenMP (<http://www.openmp.org/>)
(컴파일러마다 다르지만 대부분 라이선스 문제 없음)

CPU 병렬 프로그래밍

- 그래서 적용해 보았습니다.
 - Cmake 빌드 시스템에 병렬 프로그래밍 라이브러리를 선택할 수 있는 컴파일 옵션 추가
 - Intel TBB 관련 설정을 위한 컴파일 옵션 추가
 - Parallel 관련 함수에 라이브러리에 따른 전처리 및 로직 추가

```

template <typename TASK>
inline void Schedule(TASK&& fn) {
#ifndef CUBBYFLOW_TASKING_TBB
    struct LocalTBBTask : public tbb::task {
        TASK func;
        LocalTBBTask(TASK&& f) : func(std::forward<TASK>(f)) { }
        tbb::task* execute() override {
            func();
            return nullptr;
        }
    };
    auto* tbbNode = new(tbb::task::allocate_root())
        LocalTBBTask(std::forward<TASK>(fn));
    tbb::task::enqueue(*tbbNode);
#endif
#ifndef CUBBYFLOW_TASKING_CPP11THREAD
    std::thread thread(fn);
    thread.detach();
#endif // OpenMP or Serial -> Synchronous!
    fn();
#endif
}

```

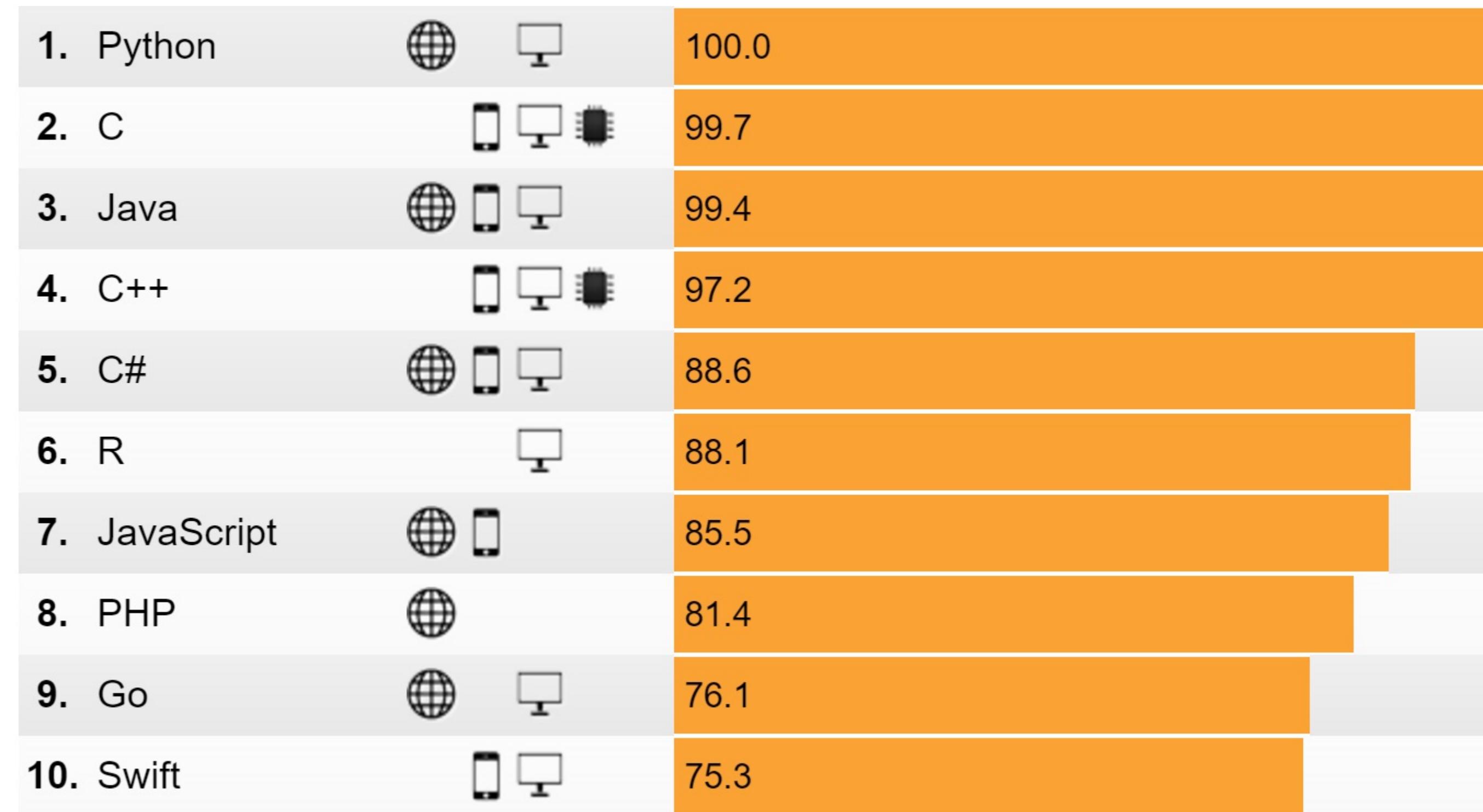
CPU 병렬 프로그래밍

- 개선 결과 : 2×10 코어에서 SmokeSim 예제를 250프레임 계산
 - C++11 Thread : 10분 14초
 - Intel TBB : 3분 58초
 - OpenMP (Static Scheduling) : 4분 9초
 - OpenMP (Dynamic Scheduling) : 3분 54초
 - Serial : 10분 7초

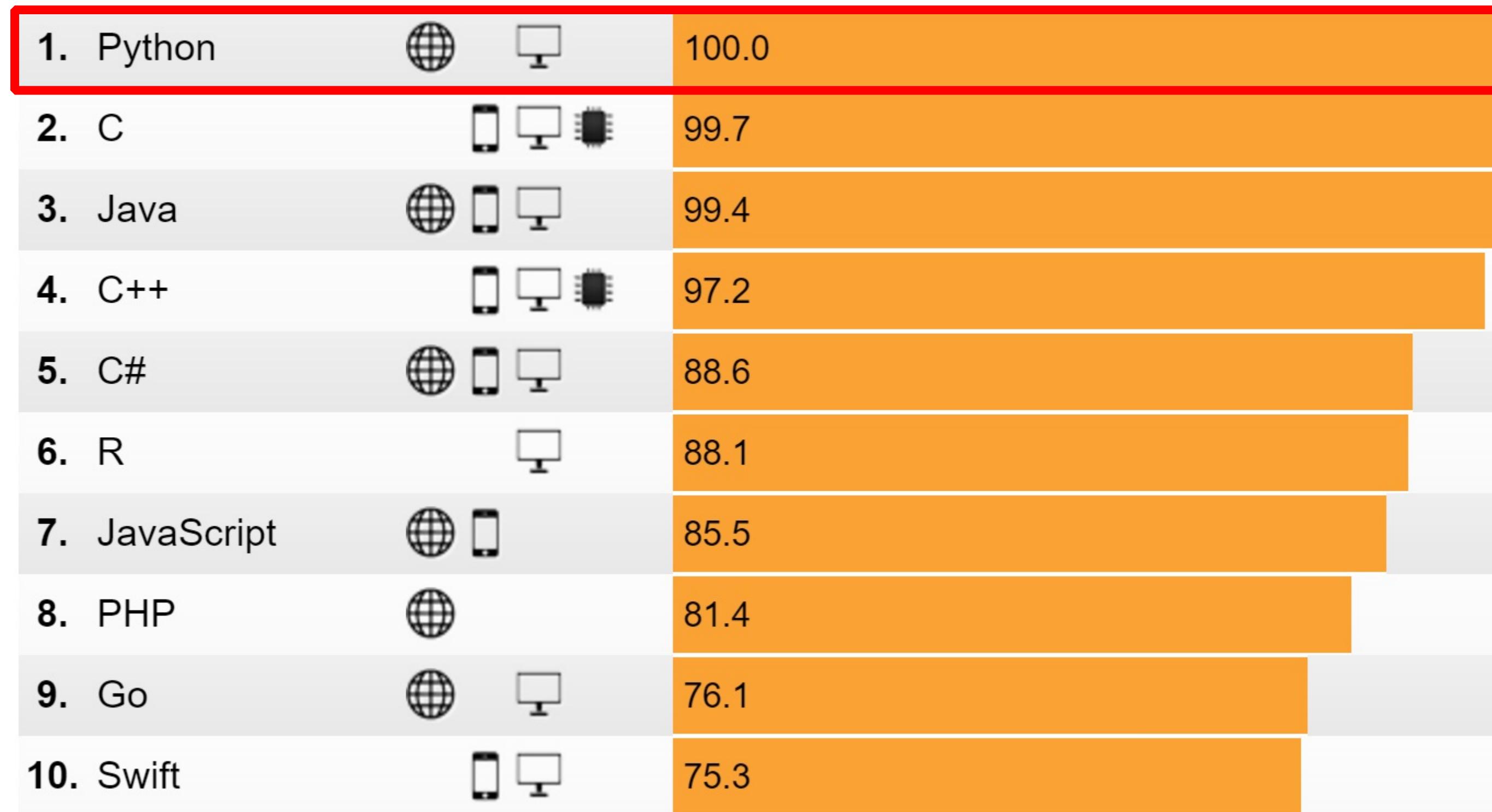


API 지원 문제

- C++이 아닌 다른 언어로 CubbyFlow를 사용할 수 있으면 좋겠습니다.
- 사람들이 많이 사용하는 언어를 먼저 지원해 봅시다.



Programming language rankings and image by [IEEE Spectrum](#)



Programming language rankings and image by [IEEE Spectrum](#)

API 지원 문제

- API 설계 방침
 - C++ 코드를 최대한 재사용할 수 있으면 좋겠습니다.
(프로그래밍 언어마다 다시 만드는 작업은 너무 무겁습니다.)
 - 확장 가능성을 고려해야 합니다.
(그렇지 않으면 새로운 클래스가 추가될 때마다 바인딩하기 어렵습니다.)

API 지원 문제

- API 설계 방침을 만족하면서 Python API를 만들 수 있을까…?
- 우리의 구세주 : pybind11 (<https://github.com/pybind/pybind11>)
 - C++ 코드를 기반으로 Python 모듈을 생성해주는 라이브러리
 - 열거체, 단일/다중 상속, 순수 가상 함수 등 C++ 핵심 기능들을 지원
 - STL 자료 구조, 반복자, 스마트 포인터 등 표준 라이브러리도 지원
 - Python 2, 3 및 PyPy 지원

Animation 클래스 예제 (C++)

```
class Animation
{
public:
    Animation();

    virtual ~Animation();

    void Update(const Frame& frame);

protected:
    virtual void OnUpdate(const Frame& frame) = 0;
};
```

Animation 클래스 예제 (pybind11)

```
#include <API/Python/Animation/Animation.h>
#include <Core/Animation/Animation.h>

#include <pybind11/pybind11.h>

using namespace CubbyFlow;

class PyAnimation : public Animation {
public:
    using Animation::Animation;

    void OnUpdate(const Frame& frame) override {
        PYBIND11_OVERLOAD_PURE(void, Animation, OnUpdate, frame);
    }
};
```

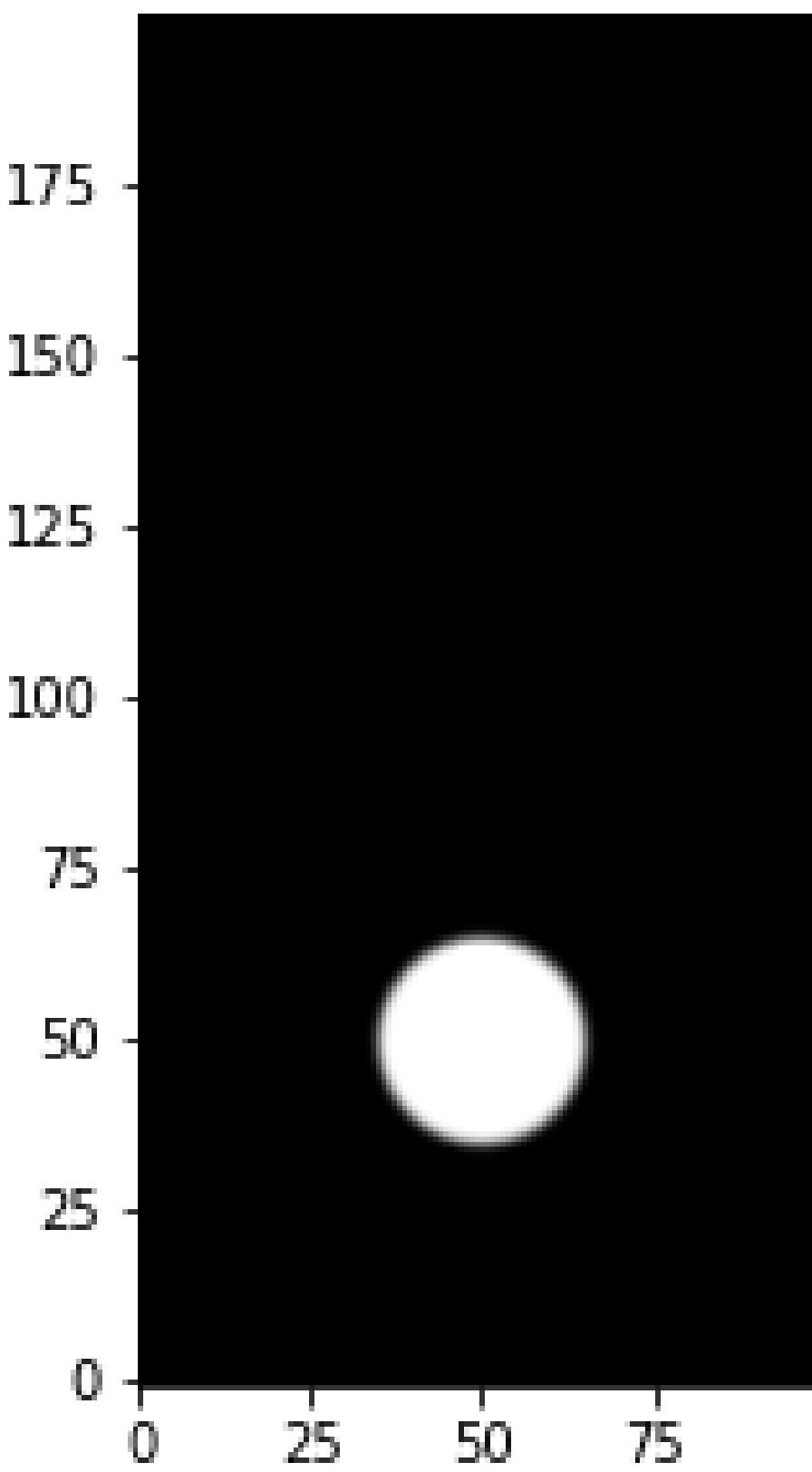
Animation 클래스 예제 (pybind11)

```
void AddAnimation(pybind11::module& m) {
    pybind11::class_<Animation, PyAnimation, AnimationPtr>(
        m, "Animation",
        R"pbdoc(
            Abstract base class for animation-related class.
        )pbdoc")
    .def(pybind11::init<>())
    .def("Update", &Animation::Update,
        R"pbdoc(
            Updates animation state for given `frame`.
        )pbdoc",
        pybind11::arg("frame"));
}
```

Python API 예제 코드

<https://ideone.com/v0twlv>

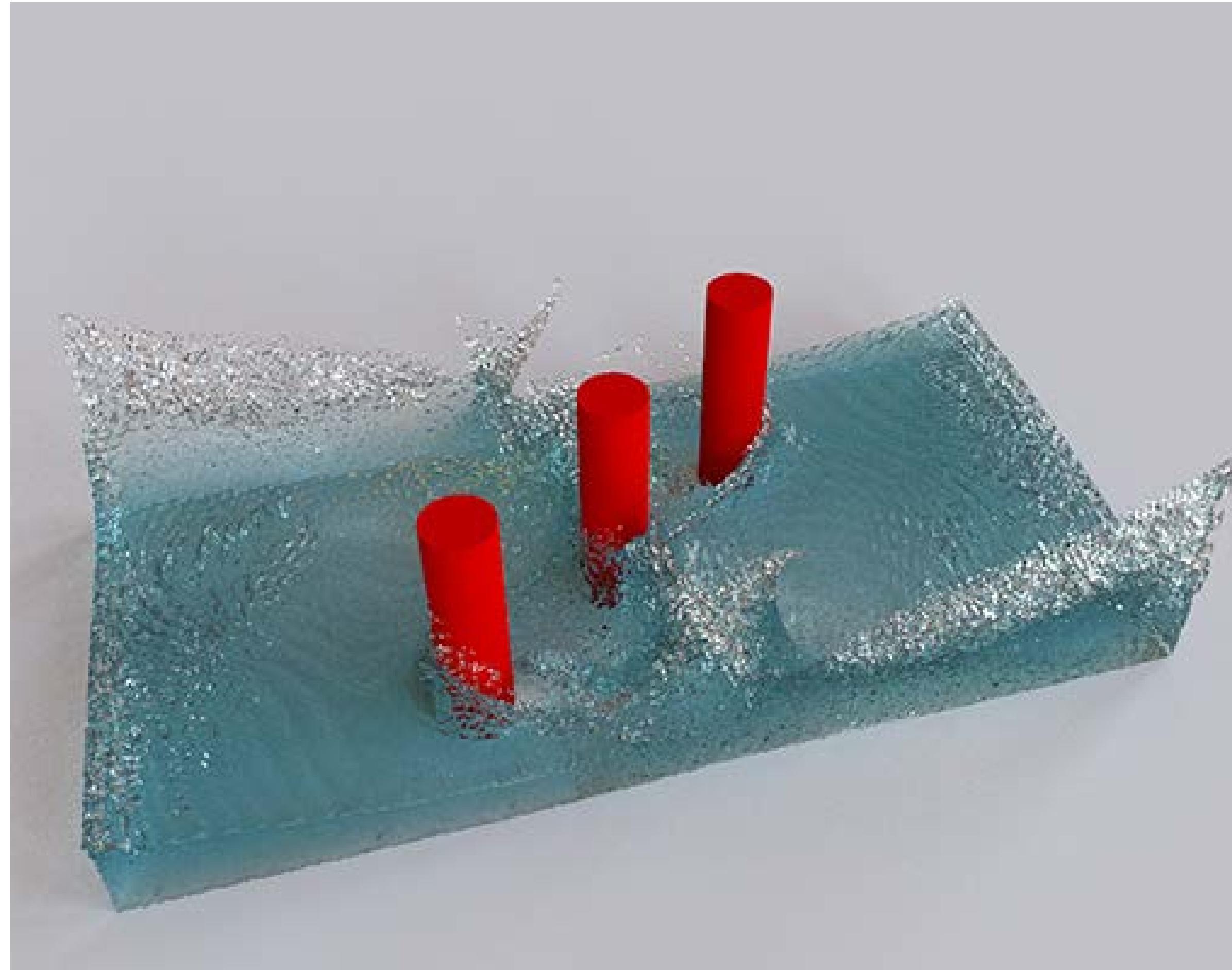
결과



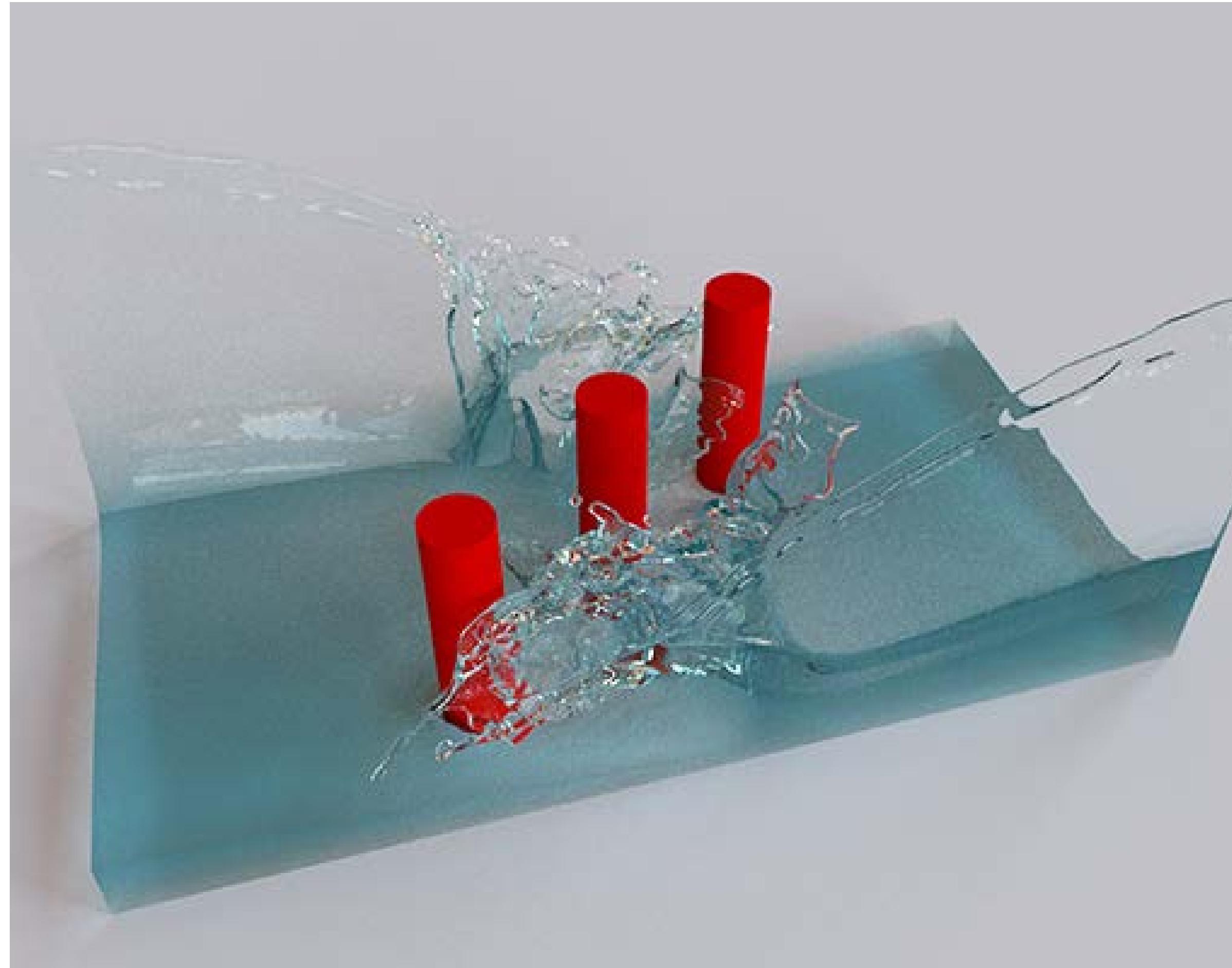
3.3. 현재까지의 결과

- Dam-breaking : PCISPH, LevelSet, PIC, FLIP, APIC
- Bunny-drop : High/Low viscosity
- Smoke : monotonic linear/cubic semi-Lagrangian
- Point-to-surface : Spherical, SPH blobby,
Zhu and Bridson's method, Anisotropic kernel

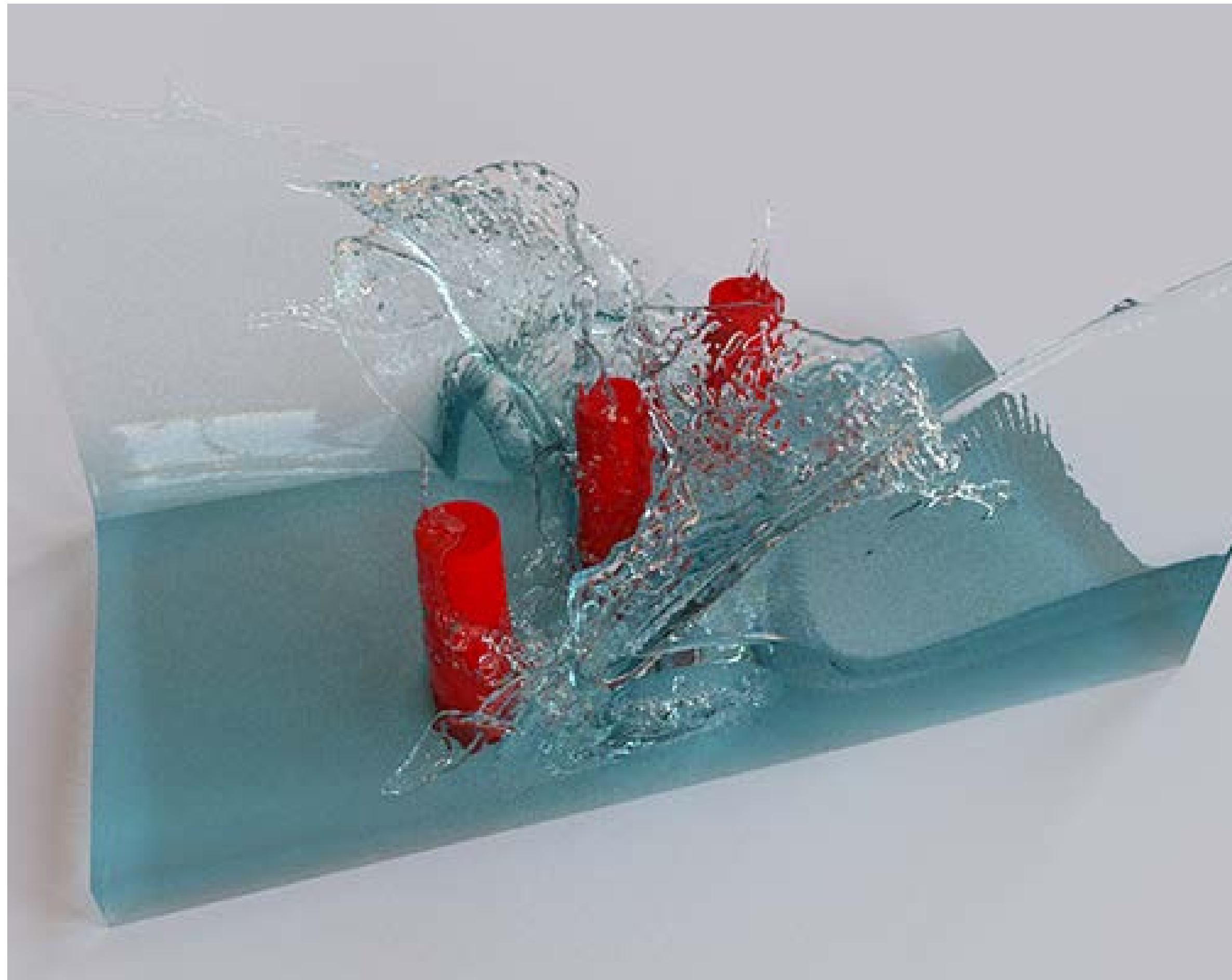
Dam-breaking simulation with PCISPH solver



Dam-breaking simulation with Level-Set solver



Dam-breaking simulation with PIC solver



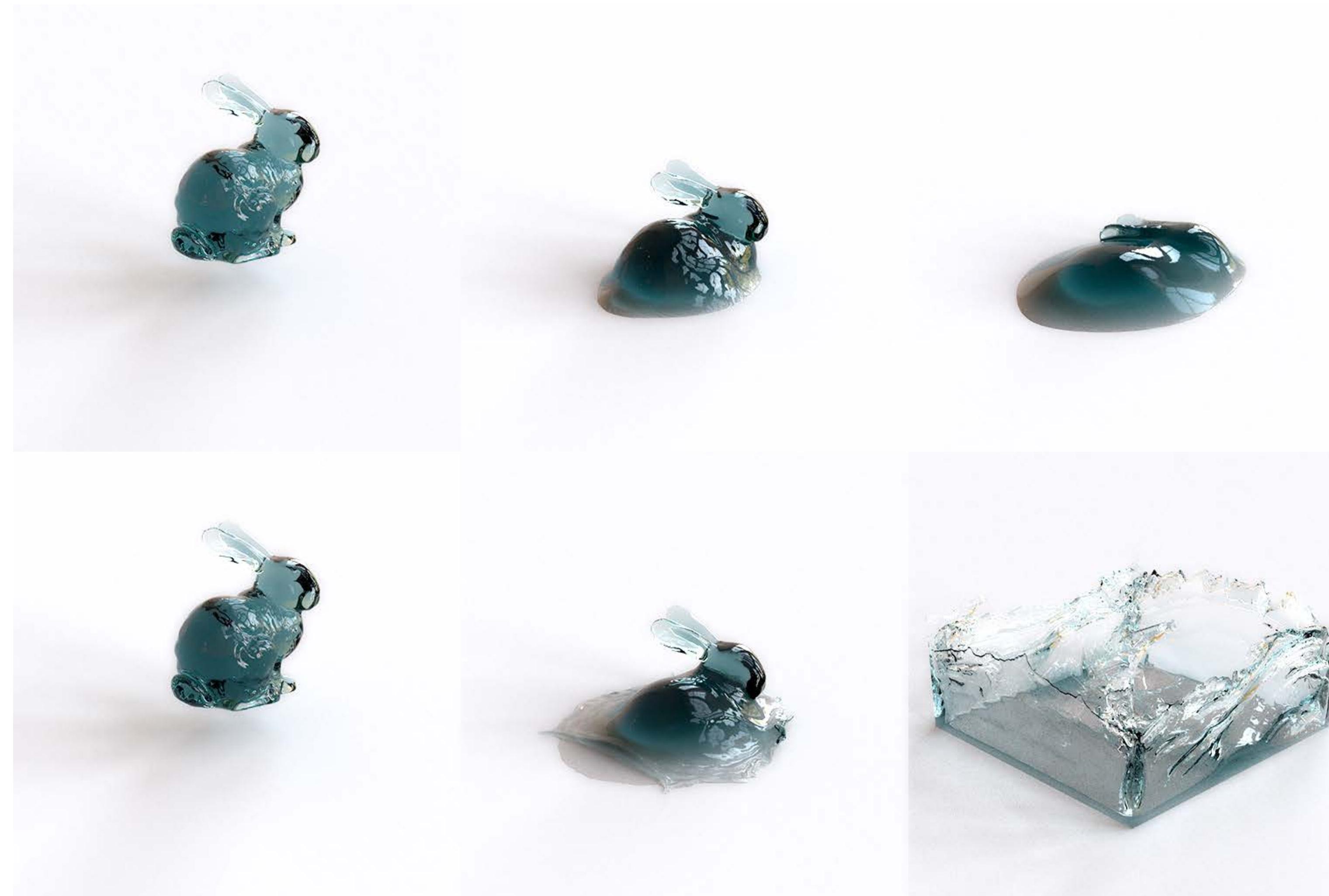
Dam-breaking simulation with FLIP solver



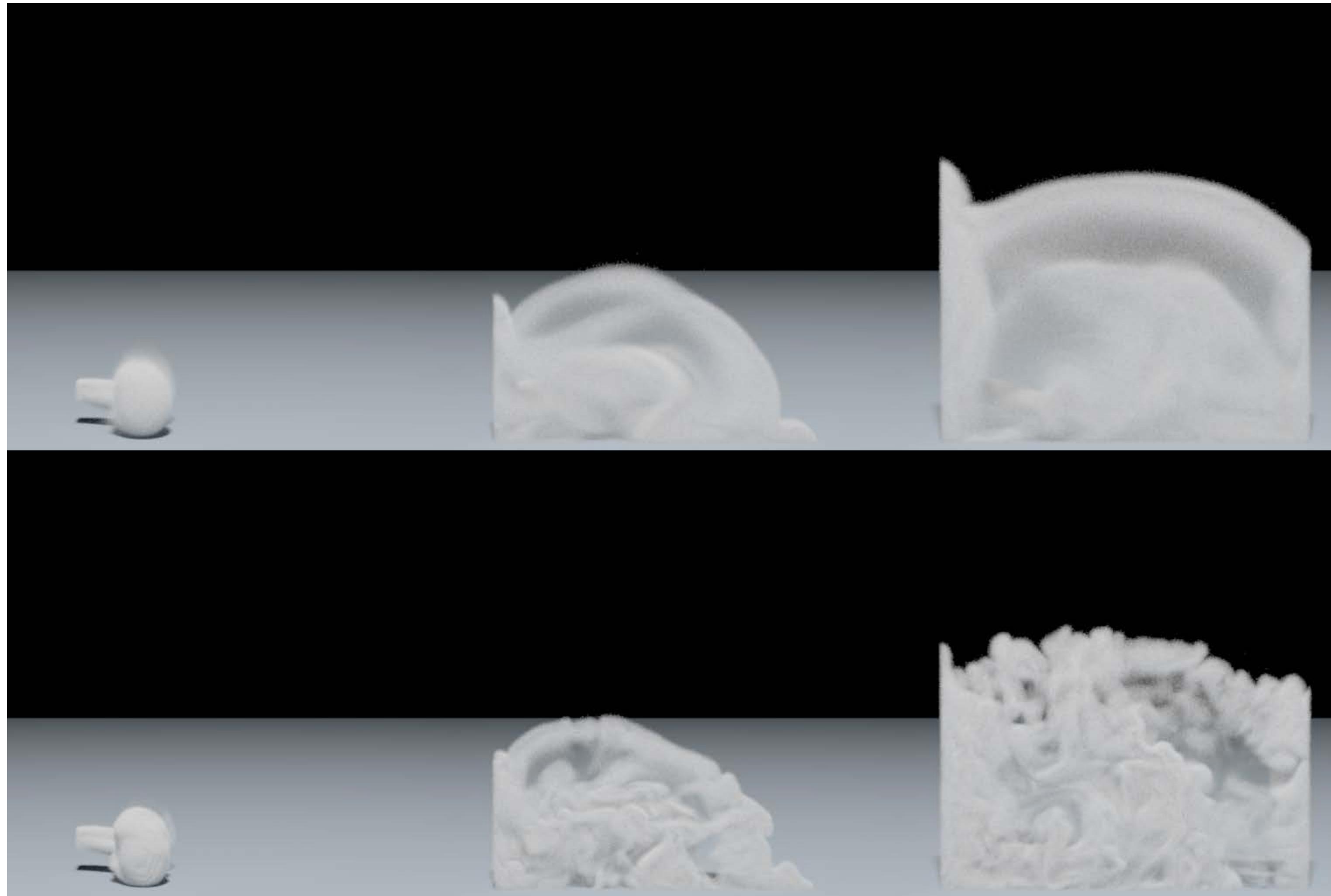
Dam-breaking simulation with APIC solver



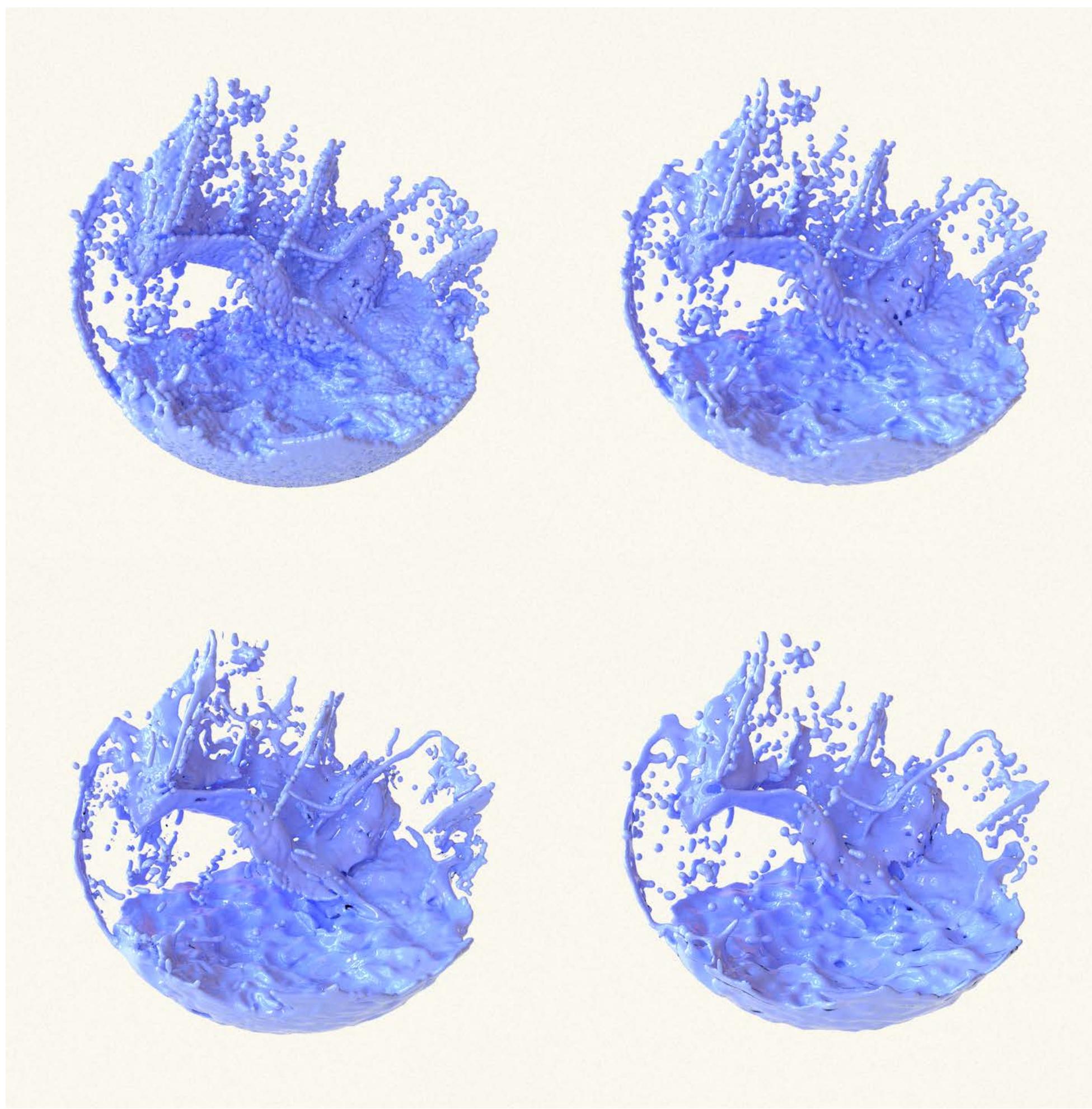
High and low viscosity simulations with Level-Set solver



Smoke simulation using monotonic linear/cubic semi-Lagrangian



Point-to-surface converter



3.4. 앞으로 개선할 부분, 그리고 고민

- 시뮬레이션 구현 문제
- 계산 속도 문제
- 렌더링 문제
- API 지원 문제

시뮬레이션 구현 문제

- 지금까지 구현한 방식
 - SPH, PCISPH solver
 - Upwind/ENO/FMM Level set solver
 - Level set-based liquid solver
 - Stable fluids-based smoke solver
 - PIC/FLIP/APIC solver
- 아직 구현하지 못한 방식
 - MPS/MPM solver, FAB solver
 - IVOCK fluid solver, Multiphase fluid solver

계산 속도 문제

- CPU 병렬을 통해 비약적인 성능 향상을 이뤄냈습니다.
- 하지만 게임에서 사용하기엔 아직 갈 길이 너무나 멍니다.
- 계산 성능을 향상시키기 위해 앞으로 해야 할 작업
 - GPU 병렬 프로그래밍 : CUDA, OpenCL
 - 머신러닝 기반 실시간 유체 시뮬레이션
 - ...

렌더링 문제

- 현재 CubbyFlow는 시뮬레이션만 수행하는 라이브러리입니다.
- 예제는 Mitsuba Renderer를 사용해 렌더링한 결과입니다.
- 실시간 시뮬레이션을 위해선 렌더러와 결합하는 작업이 필요합니다.
 - DirectX 11
 - DirectX 12
 - OpenGL
 - Vulkan
 - ...

API 지원 문제

- Python API는 pybind11 덕분에 쉽게 작업했습니다.
- 문제는 다른 언어의 API 지원입니다.
 - C# : C++/CLI, C++/CX로 해결 가능
 - Go : C 함수로 래핑해 cgo를 사용하면 가능, 하지만 너무나 느린 속도
 - Rust : C 함수로 래핑하면 가능
 - Java : JNI로 가능
 - JavaScript : WebAssembly…?
 - ...

4. 마치며

- 최신 연구 동향
- 결론

최신 연구 동향

유체의 계산을 가속화하기 위해 딥러닝(CNN, GAN)을 활용하는 추세입니다.

- Accelerating Eulerian Fluid Simulation With Convolutional Networks
(<https://arxiv.org/pdf/1607.03597.pdf>)
- Data-driven Fluid Simulations using Regression Forests
(<https://graphics.ethz.ch/~sobarbar/papers/Lad15/DatadrivenFluids.pdf>)
- Data-Driven Synthesis of Smoke Flows with CNN-based Feature Descriptors
(<https://arxiv.org/pdf/1705.01425.pdf>)
- Physics Forests: Real-time Fluid Simulation using Machine Learning
(<https://www.youtube.com/watch?v=55rsJI11FOA>)
- tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow
(<https://arxiv.org/pdf/1801.09710.pdf>)

결론

- 유체역학 엔진을 만들기는 쉽지 않습니다. 삽질은 필수입니다.
- CPU/GPU 병렬 프로그래밍을 통해 속도를 비약적으로 향상시킬 수 있습니다.
- 아직 게임에서 실시간으로 유체역학을 사용하기엔 무리라고 생각합니다.
- 하지만 머신러닝을 활용해 사용 가능성을 연구하는 중입니다.
 - 멀지 않은 미래에 가능하지 않을까 생각합니다. (돈과 시간이 문제일 뿐…)
- 프로젝트의 발전 가능성은 여전히 많습니다.
 - 건의 사항이나 협업은 언제나 환영합니다.
 - 저는 오픈 소스의 힘을 믿습니다.

감사합니다.

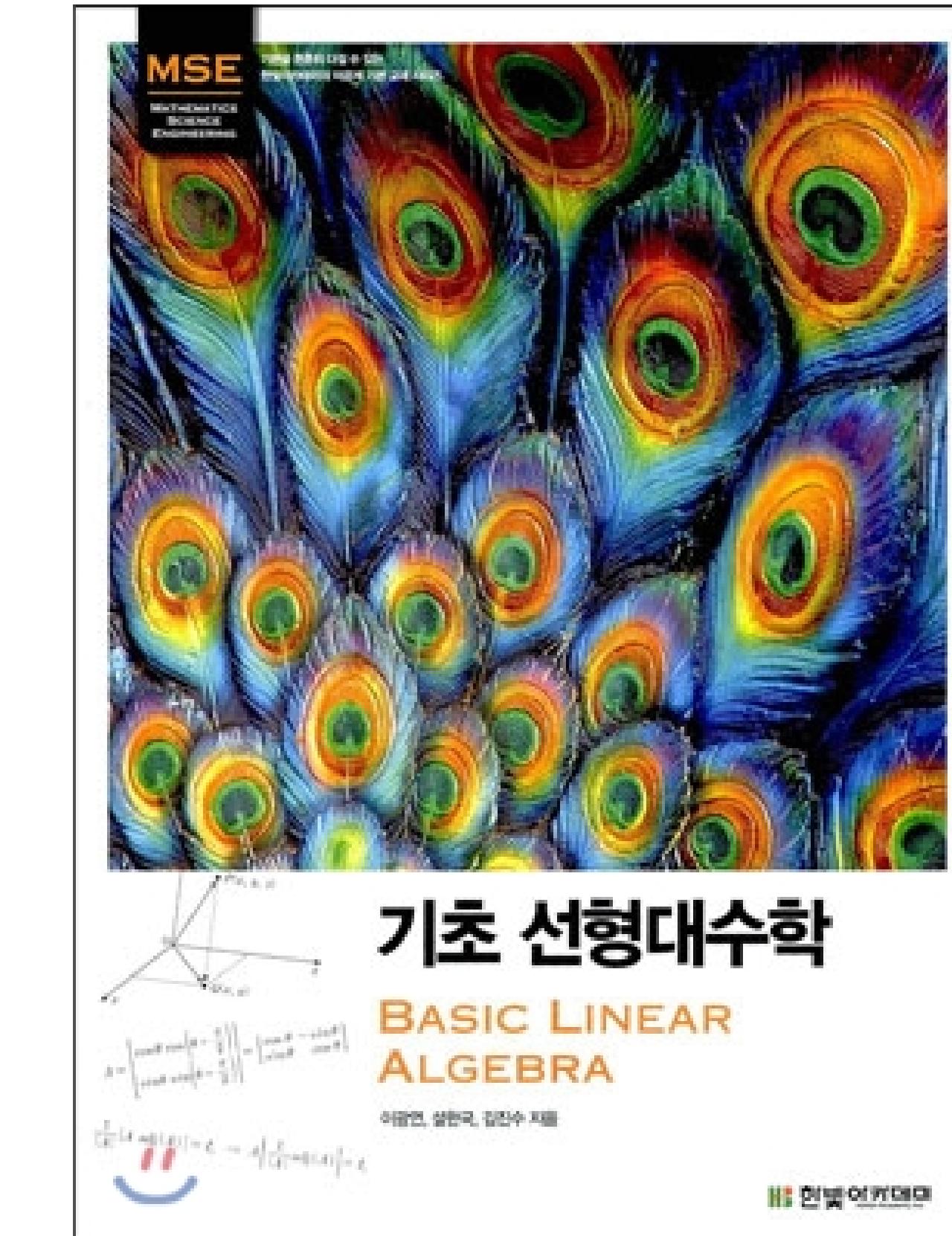
Bonus) 다루지 못했던 내용

- 유체역학을 이해하기 위한 몇 가지 수학 용어

유체역학을 이해하기 위한 몇 가지 수학 용어

- 행렬(Matrix) & 벡터(Vector)
- 선형 방정식(Linear Equation)
- 장(Field)
- 편미분(Partial Derivative)
- 기울기(Gradient)
- 발산(Divergence)
- 회전(Curl)
- 라플라시안(Laplacian)

도움이 될만한 책

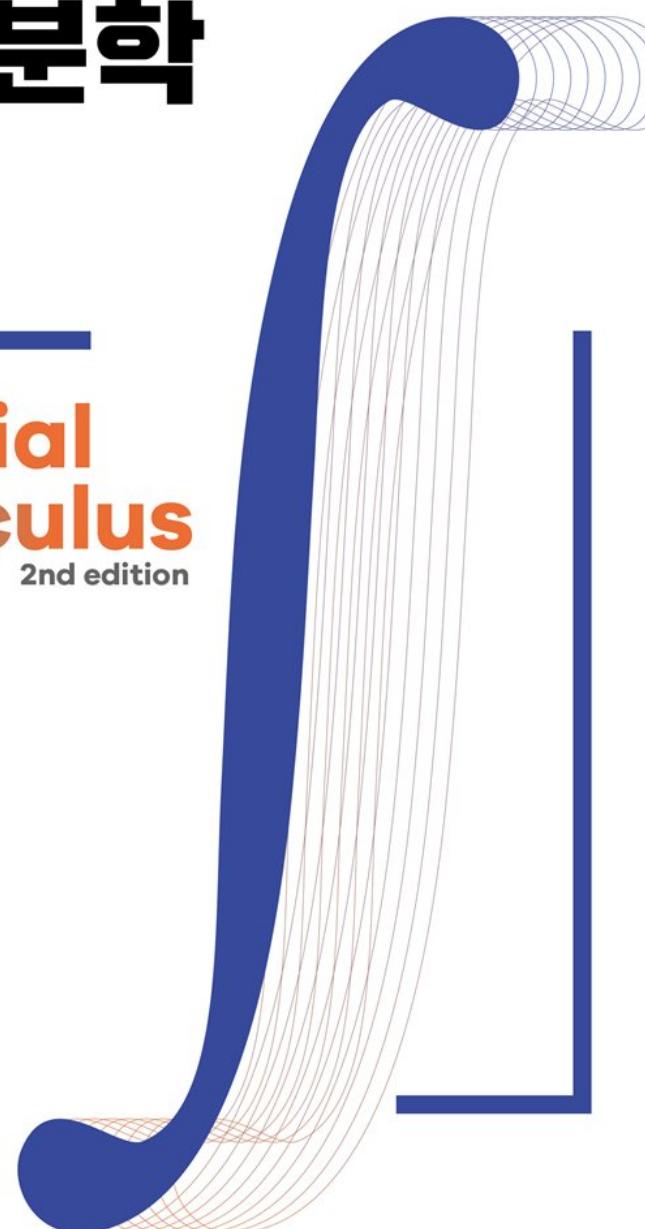


도움이 될만한 책

미분적분학
에센스

Essential
Calculus
2nd edition

James Stewart 지음
한빛수학교재연구소 옮김

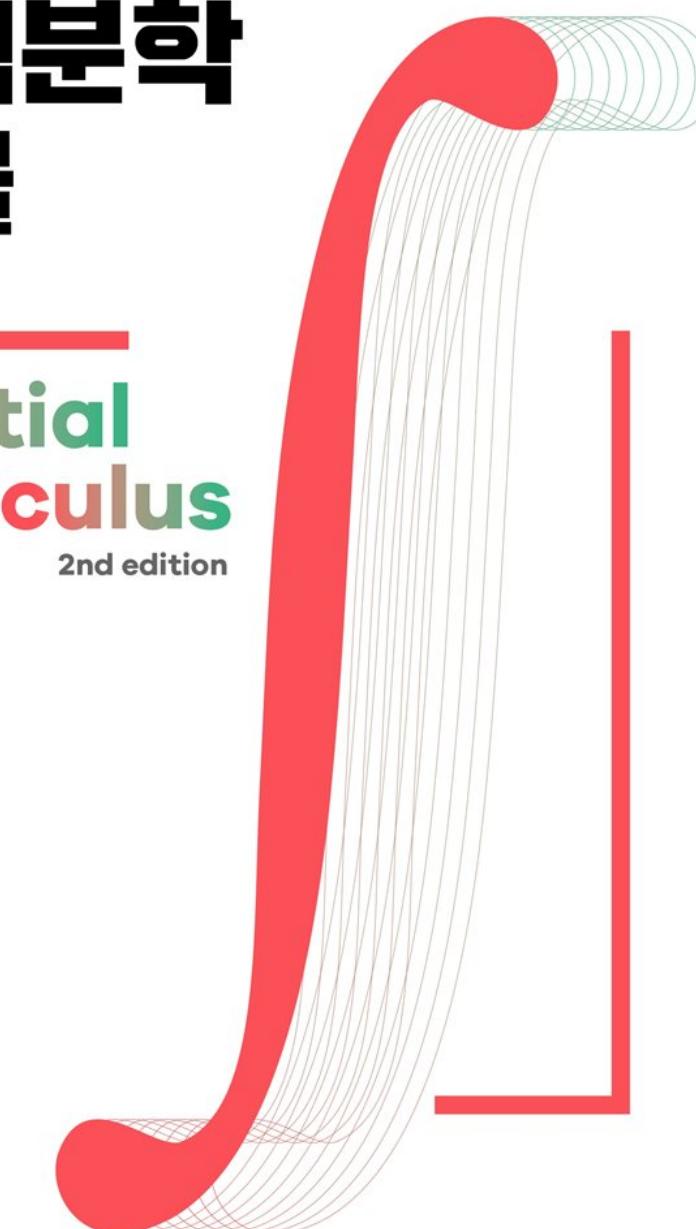


한빛아카데미

미분적분학
바이블

Essential
Calculus
2nd edition

James Stewart 지음
한빛수학교재연구소 옮김



한빛아카데미

행렬 (Matrix) & 벡터 (Vector)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

행렬
(Matrix)

$$\vec{a} = (a_x, a_y, a_z)$$

벡터
(Vector)

행렬 (Matrix) & 벡터 (Vector)

역행렬

단위행렬

내적

외적

...



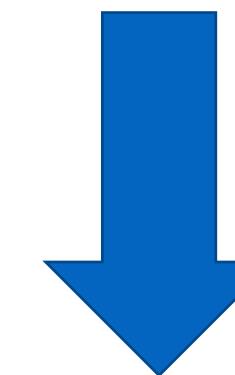
더 이상의
자세한 설명은
생략한다.

선형 방정식 (Linear Equation)

$$\begin{aligned}2x - y &= 3 \\-x + 2y &= 6\end{aligned}$$

선형 방정식 (Linear Equation)

$$\begin{aligned}2x - y &= 3 \\-x + 2y &= 6\end{aligned}$$



행렬, 벡터로 변환

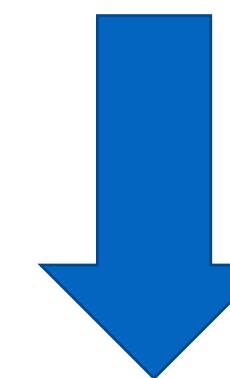
$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

선형 방정식 (Linear Equation)

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

선형 방정식 (Linear Equation)

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

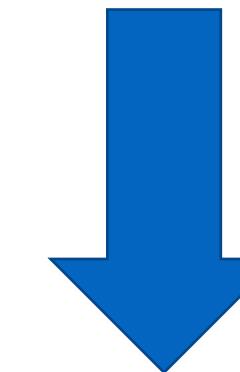


역행렬

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ 6 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

선형 방정식 (Linear Equation)

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$



일반화

$$Ax = b$$

선형 방정식을 푸는 방법

- 직접적인 방법 (Direct Method)
 - 가우스-조던 소거법 (Gauss-Jordan Elimination)
- 간접적인 방법 (Indirect Method)
 - 야코비 방법 (Jacobi Method)
 - 가우스-사이델 방법 (Gauss-Seidel Method)
 - 경사 하강법 (Gradient Descent Method)
 - 컬레(공역) 기울기법 (Conjugate Gradient Method)
 - 전처리된 컬레(공역) 기울기법 (Preconditioned Conjugate Gradient Method)

선형 방정식을 푸는 방법

- 직접적인 방법 (Direct Method)
 - 가우스-조던 소거법 (Gauss-Jordan Elimination)
 - 간접적인 방법 (Indirect Method)
 - 야코비 방법 (Jacobi Method)
 - 가우스-사이델 방법 (Gauss-Seidel Method)
 - 경사 하강법 (Gradient Descent Method)
 - 컬레(공역) 기울기법 (Conjugate Gradient Method)
 - 전처리된 컬레(공역) 기울기법 (Preconditioned Conjugate Gradient Method)
- $O(N^3)$

선형 방정식을 푸는 방법

- 직접적인 방법 (Direct Method)
 - 가우스-조던 소거법 (Gauss-Jordan Elimination)
- 간접적인 방법 (Indirect Method)
 - 야코비 방법 (Jacobi Method)
 - 가우스-사이델 방법 (Gauss-Seidel Method)
- 경사 하강법 (Gradient Descent Method)
- 컬레(공역) 기울기법 (Conjugate Gradient Method)
- 전처리된 컬레(공역) 기울기법 (Preconditioned Conjugate Gradient Method)

$O(N^3)$

$O(N^2)$

선형 방정식을 푸는 방법

- 직접적인 방법 (Direct Method)
 - 가우스-조던 소거법 (Gauss-Jordan Elimination)
- 간접적인 방법 (Indirect Method)
 - 야코비 방법 (Jacobi Method)
 - 가우스-사이델 방법 (Gauss-Seidel Method)
- 경사 하강법 (Gradient Descent Method)
- 컬레(공역) 기울기법 (Conjugate Gradient Method)
- 전처리된 컬레(공역) 기울기법 (Preconditioned Conjugate Gradient Method)

$O(N^3)$

$O(N^2)$

경사 하강법 (Gradient Descent)

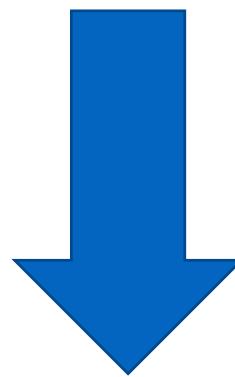
함수의 기울기(경사)를 구하여 기울기가 높은 쪽으로
계속 이동시켜서 극값에 이를 때까지 반복시키는 최적화 알고리즘

$$Ax = b$$

경사 하강법 (Gradient Descent)

함수의 기울기(경사)를 구하여 기울기가 높은 쪽으로
계속 이동시켜서 극값에 이를 때까지 반복시키는 최적화 알고리즘

$$Ax = b$$



$$F(x) = |Ax - b|^2$$

경사 하강법 (Gradient Descent)

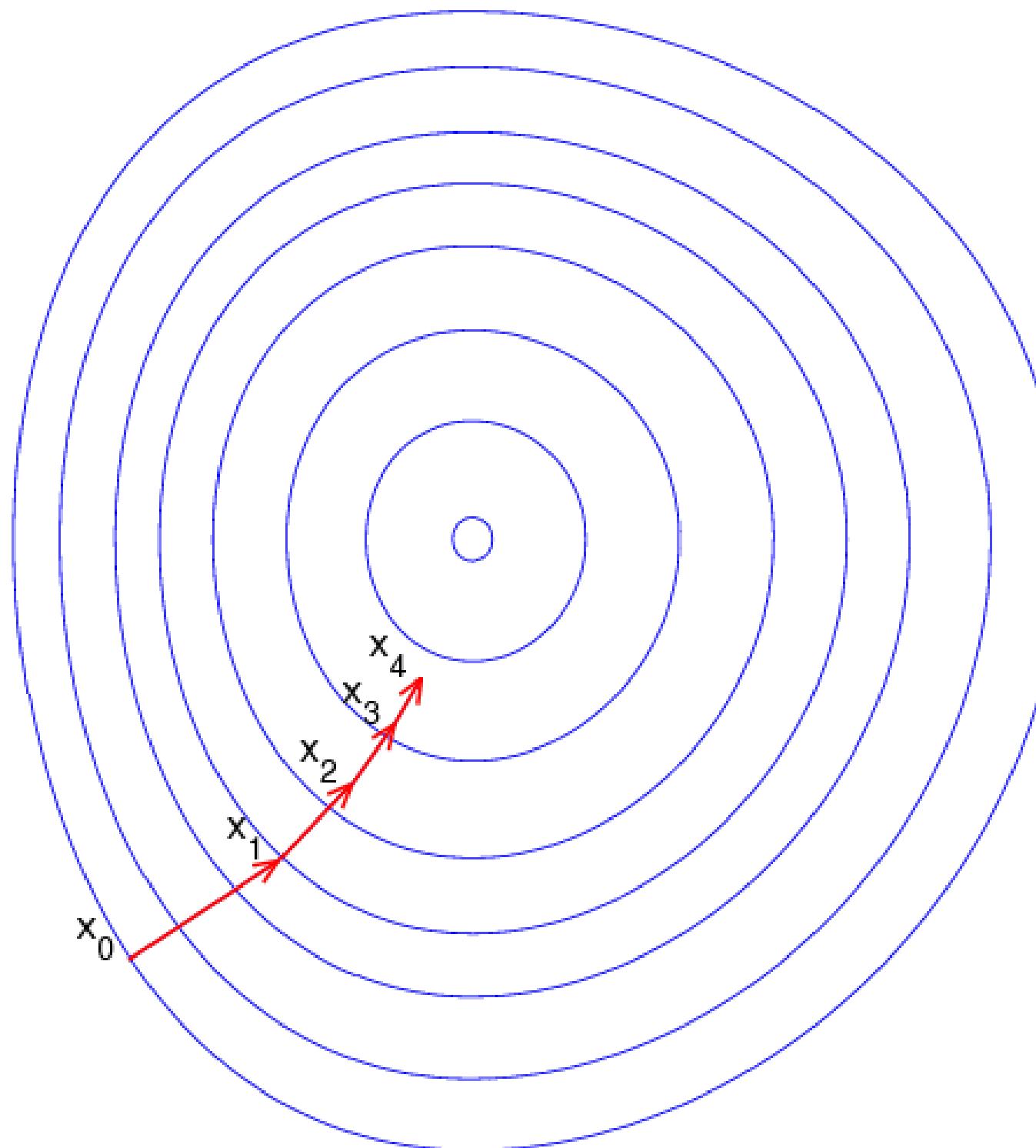
함수의 기울기(경사)를 구하여 기울기가 높은 쪽으로
계속 이동시켜서 극값에 이를 때까지 반복시키는 최적화 알고리즘

$$F(\mathbf{x}) = |\mathbf{Ax} - \mathbf{b}|^2$$

1. 시작점 \mathbf{x}_0 를 정한다.
2. 현재 \mathbf{x}_i 가 주어졌을 때,
 - a) 만약 $F(\mathbf{x}_i)$ 가 오차범위(예, 0.001) 내에 있으면, 해는 \mathbf{x}_i 다.
 - b) 아니라면, 다음 수식을 통해 \mathbf{x}_{i+1} 을 구한다.
$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma_i \nabla f(\mathbf{x}_i)$$
3. 2를 반복한다.

경사 하강법 (Gradient Descent)

함수의 기울기(경사)를 구하여 기울기가 높은 쪽으로
계속 이동시켜서 극값에 이를 때까지 반복시키는 최적화 알고리즘

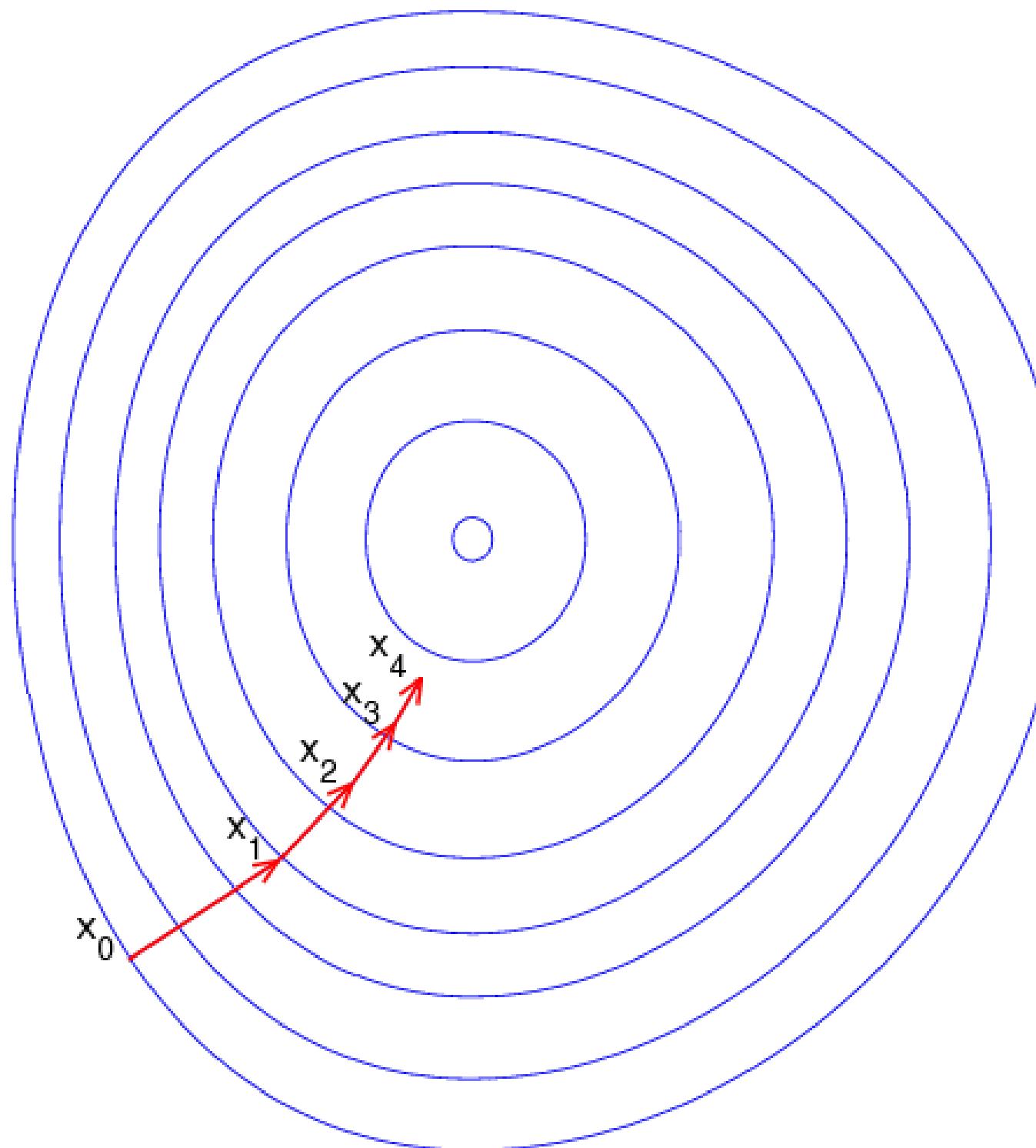


$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma_i \nabla f(\mathbf{x}_i)$$

γ_i : 이동할 거리를 조절하는 매개변수

경사 하강법 (Gradient Descent)

함수의 기울기(경사)를 구하여 기울기가 높은 쪽으로
계속 이동시켜서 극값에 이를 때까지 반복시키는 최적화 알고리즘



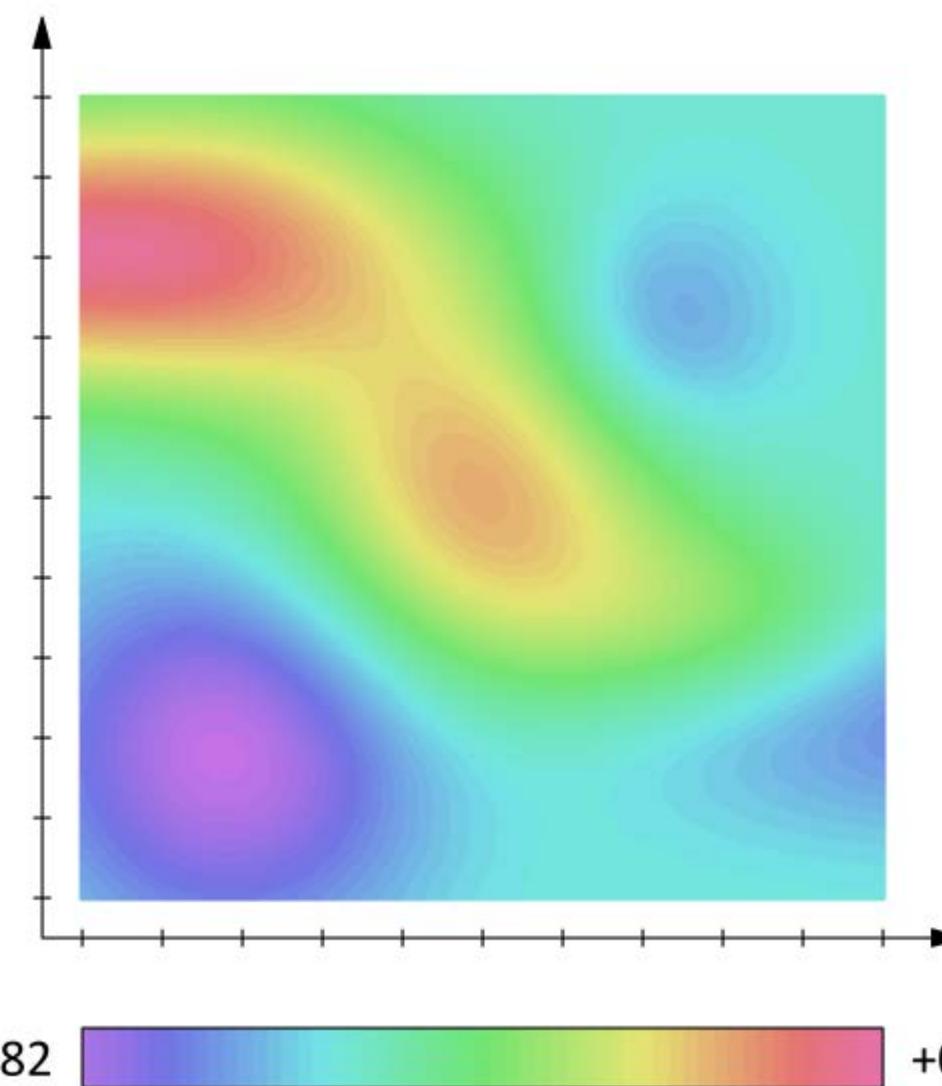
경사 하강법의 문제점 : 수렴 속도가 느림
→ 컬레(공역) 기울기 법을 사용!

선형 방정식을 푸는 방법

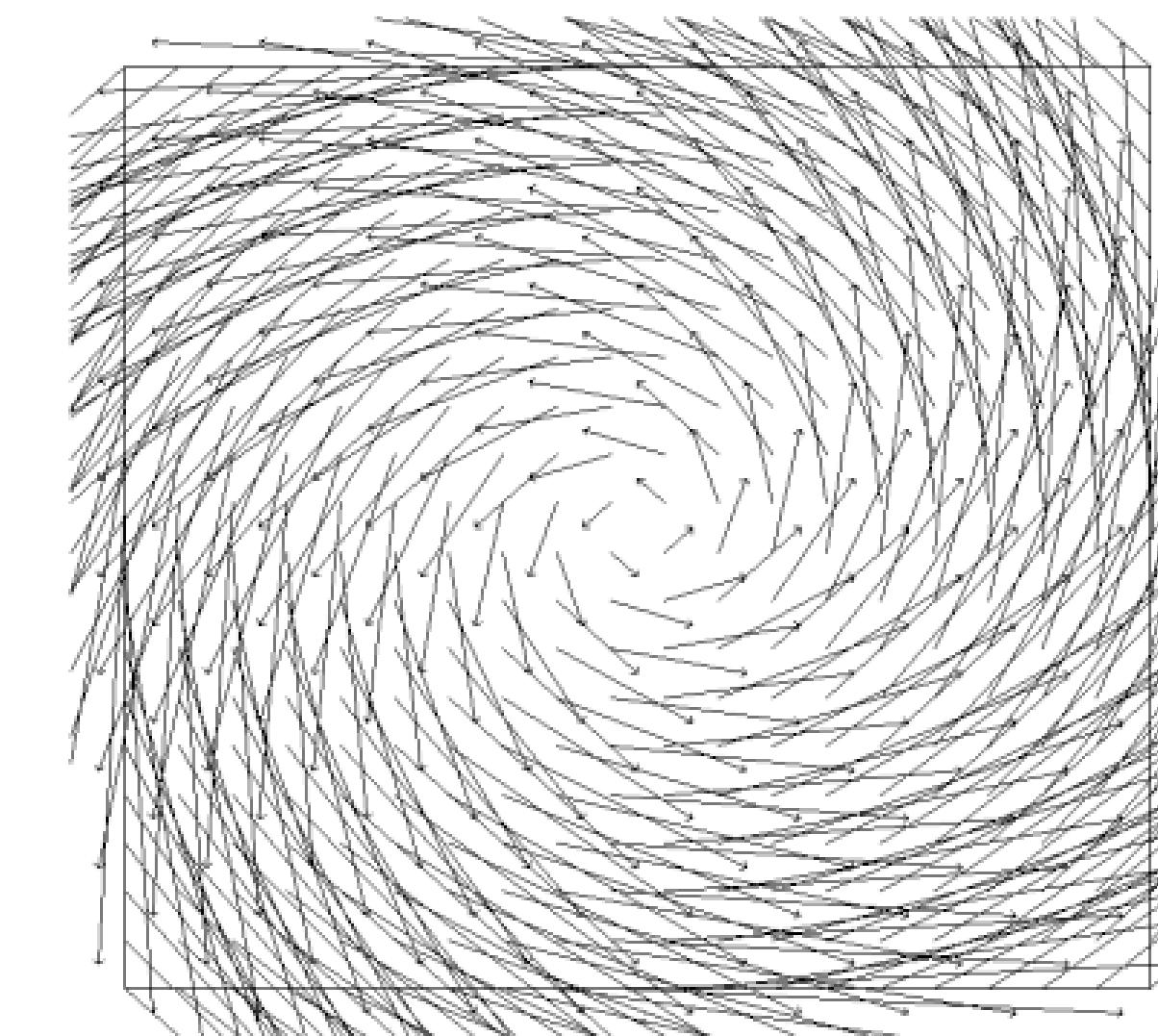
- 직접적인 방법 (Direct Method)
 - 가우스-조던 소거법 (Gauss-Jordan Elimination) $O(N^3)$
- 간접적인 방법 (Indirect Method)
 - 야코비 방법 (Jacobi Method)
 - 가우스-사이델 방법 (Gauss-Seidel Method) $O(N^2)$
- 경사 하강법 (Gradient Descent Method)
- 컬레(공역) 기울기법 (Conjugate Gradient Method)
- 전처리된 컬레(공역) 기울기법 (Preconditioned Conjugate Gradient Method)
 $O(N)$

장 (Field)

유클리드 공간의 각 점에 어떤 값을 대응시키는 것



스칼라장
(Scalar Field)



벡터장
(Vector Field)

편미분 (Partial Derivative)

다변수 함수의 특정 변수를 제외한 나머지 변수를
상수로 생각하여 미분하는 것

$$z = f(x, y) = x^2 + xy + y^2$$

x 에 대해 편미분
 y 는 상수 취급

$$\frac{\partial z}{\partial x} = 2x + y$$

y 에 대해 편미분
 x 는 상수 취급

$$\frac{\partial z}{\partial y} = 2y + x$$

기울기 (Gradient)

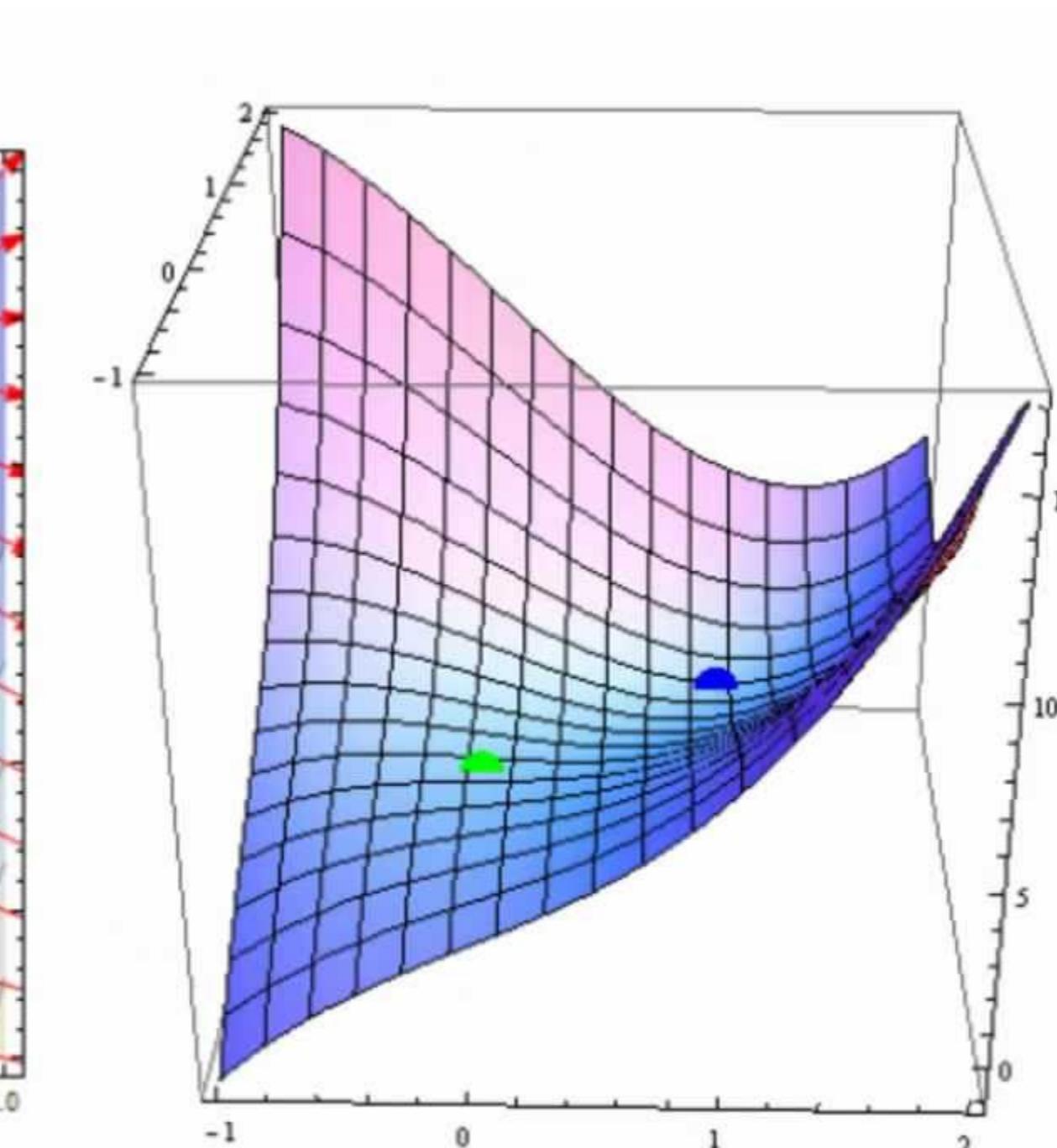
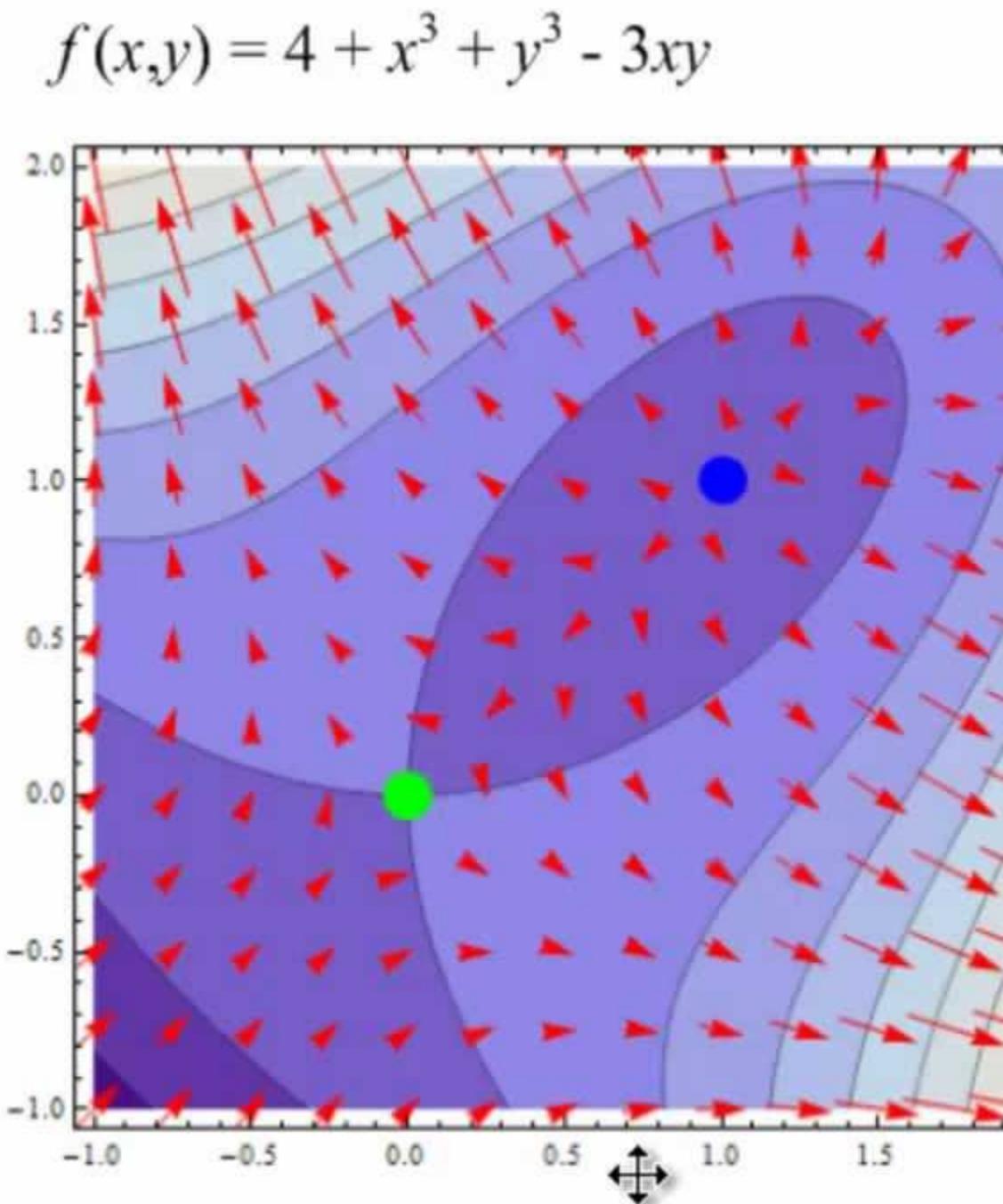
스칼라장의 최대의 증가율을 나타내는 벡터장

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x}(\mathbf{x}), \frac{\partial f}{\partial y}(\mathbf{x}), \frac{\partial f}{\partial z}(\mathbf{x}) \right)$$

기울기 (Gradient)

화살표의 방향 = 증가율이 최대가 되는 방향

화살표의 크기 = 증가율이 최대일 때의 증가율의 크기



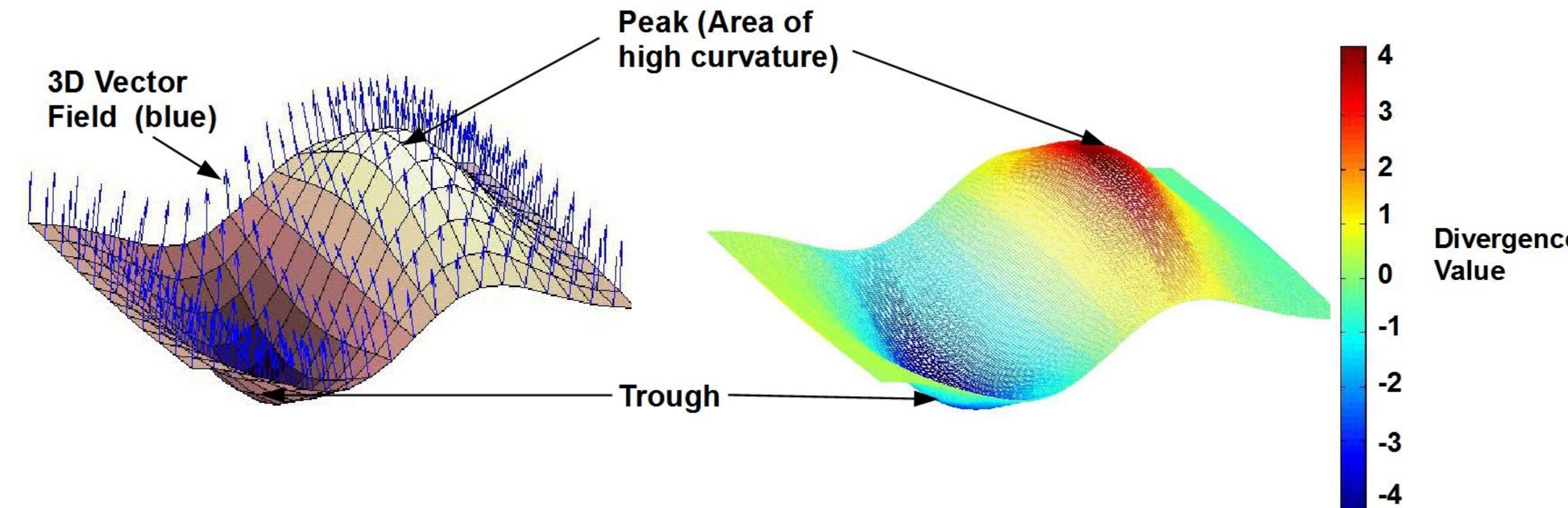
발산 (Divergence)

벡터장이 정의된 공간의 한 점에서의 장이 퍼져 나오는지,
아니면 모여서 없어지는지의 정도를 측정하는 연산자

$$\nabla \cdot \mathbf{F}(\mathbf{x}) = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}$$

발산 (Divergence)

벡터장이 정의된 공간의 한 점에서의 장이 퍼져 나오는지,
아니면 모여서 없어지는지의 정도를 측정하는 연산자



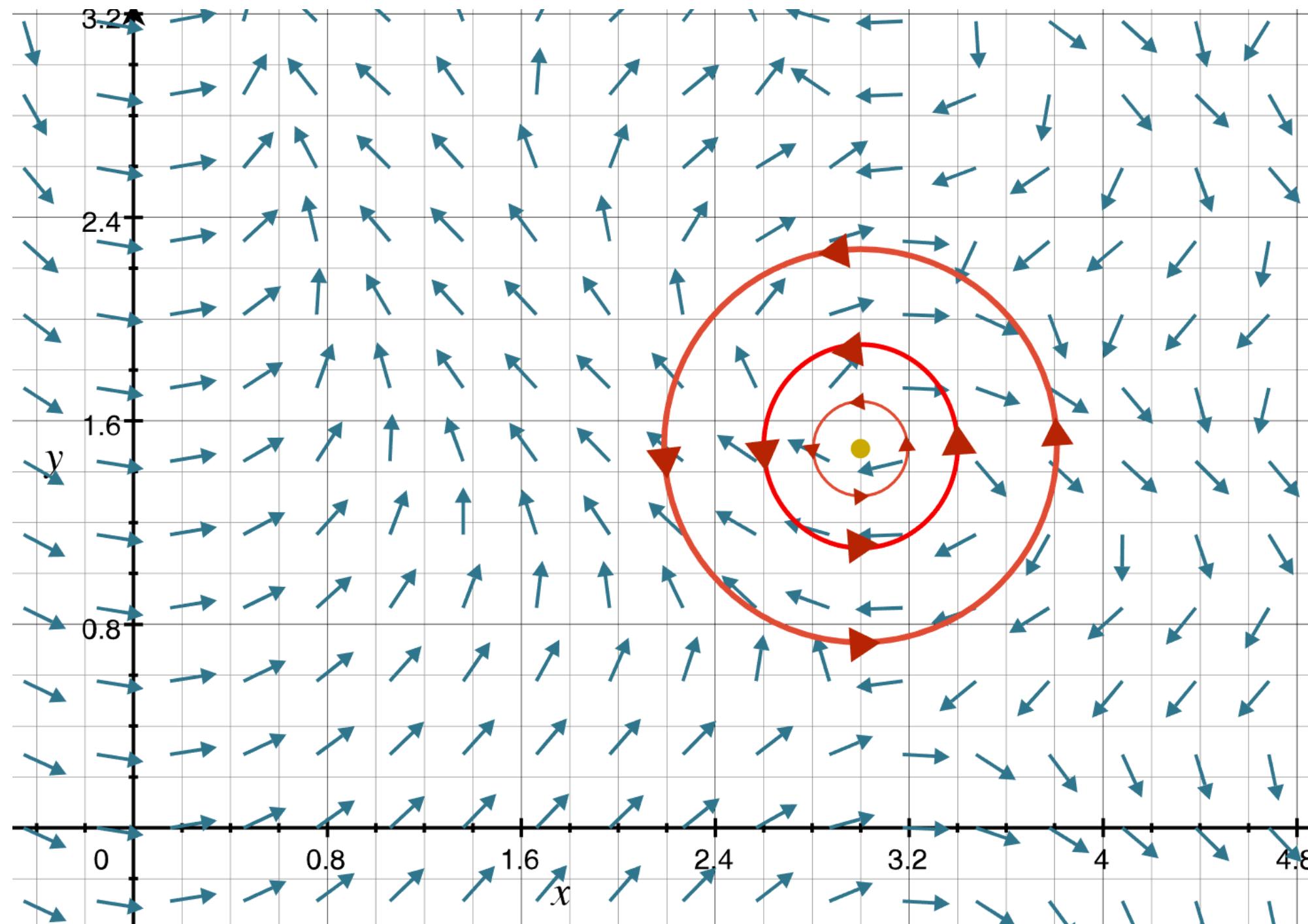
회전 (Curl)

벡터장 내에서 임의의 한 점의 매우 작은 공간이
주변의 벡터로 인해 발생하는 회전 정도를 측정하는 연산자

$$\nabla \times \mathbf{F}(\mathbf{x}) = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right) \times \mathbf{F}(\mathbf{x})$$

회전 (Curl)

벡터장 내에서 임의의 한 점의 매우 작은 공간이 주변의 벡터로 인해 발생하는 회전 정도를 측정하는 연산자



라플라시안 (Laplacian)

주어진 위치에서의 스칼라장 값이 근처에 있는
스칼라장 값의 평균과 얼마나 다른지를 측정하는 연산자

$$\nabla^2 f(\mathbf{x}) = \frac{\partial^2 f(\mathbf{x})}{\partial x^2} + \frac{\partial^2 f(\mathbf{x})}{\partial y^2} + \frac{\partial^2 f(\mathbf{x})}{\partial z^2}$$

라플라시안 (Laplacian)

주어진 위치에서의 스칼라장 값이 근처에 있는
스칼라장 값의 평균과 얼마나 다른지를 측정하는 연산자

