AERO SIMULATOR

202128683 이지수
202128682 최윤재

미디어 라이트 형제

CONTENTS

/

01

[Introduction]

Purpose

- 브라우저 기반,
  실시간 유체 시뮬레이터 구현

- 사용자가 조작 가능한 인터페이스를
  통해 유체 흐름 시각화

- 물체와 상호작용에 따른 항공역학적
  수치 계산값 도출

Property

- Euler 방식의 유체 시뮬레이션

- 다양한 상황에서 유체 동작을
  실시간으로 시뮬레이션

  Ex) Wind-Tunnel, Streamline, Pressure

- 상호작용 가능한 매개변수 조정 기능

02

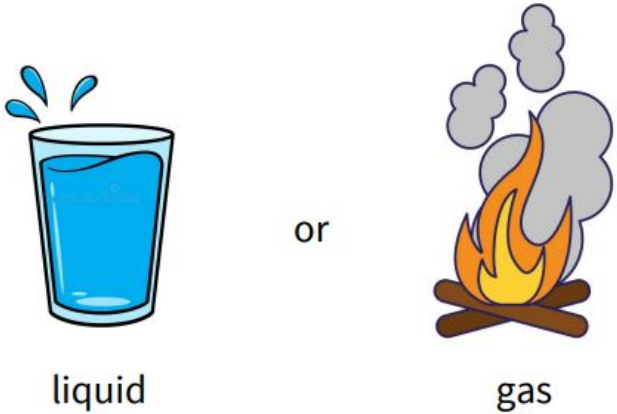
[Background]

Fluid

- **유체의 종류**:
  - 액체와 기체

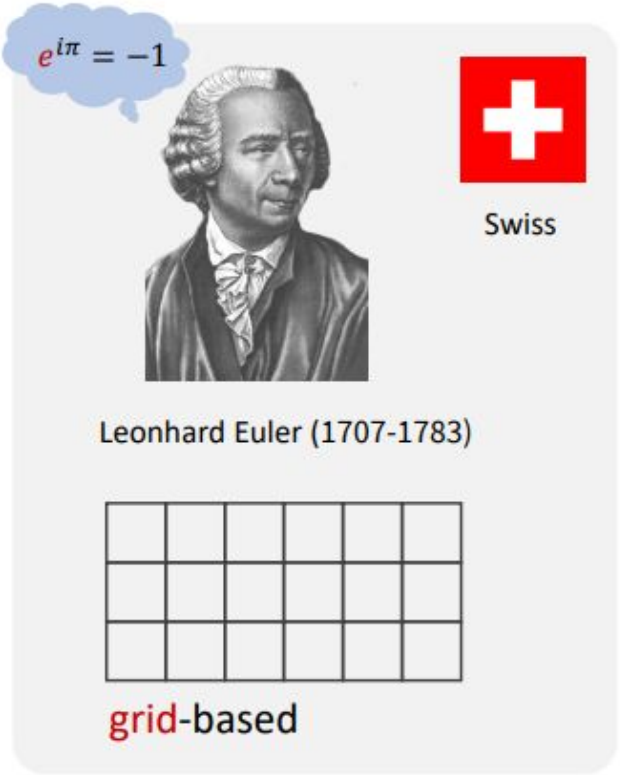
- **비압축성 (Incompressible)**:
  - 압축성을 무시하여 계산 단순화

- **비점성 (Inviscid)**:
  - 점성을 0으로 가정하여 계산 단순화



Simulation Method

- **Euler 방법(Grid-based)**:
  - 입자가 아닌 격자(Grid) 기반으로 유체 흐름 계산
  - 고정된 격자 위에서 값을 계산하여, 수학적 모델링과 계산에 용이
  - Lagrange의 입자(Particle) 기반의 grid-free가능성도 존재

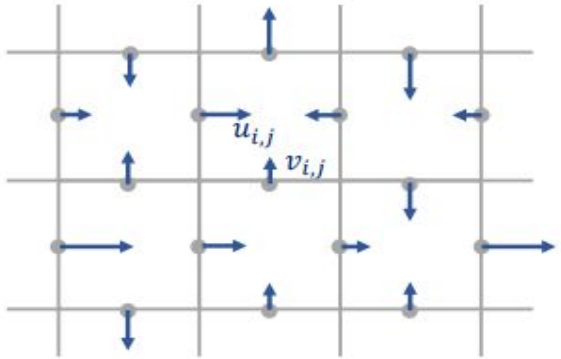


03

[Methodology]

Main-Steps

- **2차원 벡터 & Grid 기반 속도 필드**:
  1. 2차원 속도 벡터 u, v 사용,
     그리드 간격 h
  2. 각 격자점에서 속도값을
     계산하여 저장



staggered grid

- **시뮬레이션 주요 단계**:
  1. 중력 효과 적용(Gravity)
  2. 유체의 비압축성화(Incompress)
  3. 속도장의 이동(Advection)

1. Gravity

중력(Gravity) 적용:

- Gravity g: −9.81 $m/s2$
- Timestep Δt: (eg. 1/30s)

for all i, j

$$v_{i,j} \leftarrow v_{i,j} + \Delta t \cdot g$$

2. Incompress & Divergence

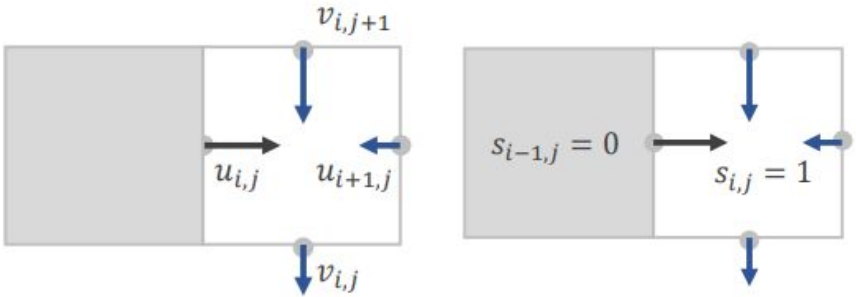
d: 발산, 해당 격자점에서 얼만큼의 유체가 입출하는지 측정

U_i+1,j - U_i,j, V_i,j+1 - V_i,j

s: 해당 격자점과 인접한 4개의 셀의 가중합

S_i-1,j=0, S_i,j=1

$$d \leftarrow u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j}$$



2. Cont’d

- Gauss-Seidel 방법을 사용
- U_i,j <- U_i,j+d*S_i-1,j/S~

for n iterations
  for all i, j
    $d \leftarrow u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j}$
    $s \leftarrow s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}$

$$
\begin{aligned}
u_{i,j} &\leftarrow u_{i,j} + d\, s_{i-1,j}/s \\
u_{i+1,j} &\leftarrow u_{i+1,j} - d\, s_{i+1,j}/s \\
v_{i,j} &\leftarrow v_{i,j} + d\, s_{i,j-1}/s \\
v_{i,j+1} &\leftarrow v_{i,j+1} - d\, s_{i,j+1}/s
\end{aligned}
$$

- Pressure

P_i, j: 해당 셀에서의 입력 값

ρ: 유체의 밀도

h: 격자 간격

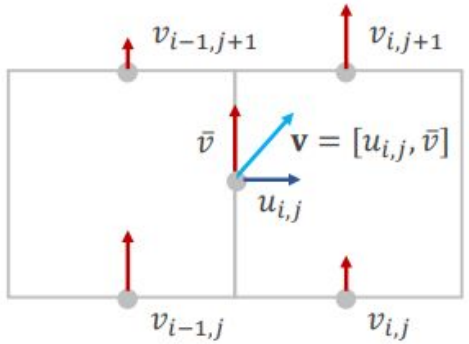
Δt: 시간 간격.

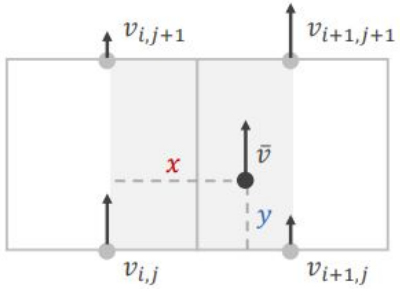
for all i, j
  $p_{i,j} \leftarrow 0$

for n iterations
  for all i, j
    $d \leftarrow u_{i+1,j} - u_{i,j} + v_{i,j+1} - v_{i,j}$
    $s \leftarrow s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}$
    ...
    $p_{i,j} \leftarrow p_{i,j} + \frac{d}{s} \cdot \frac{\rho\, h}{\Delta t}$

3. Advection

- Advection: 유체의 물리적 속성(속도, 밀도 등)을 이동시키는 과정
- Semi-Lagrangian: 유체 입자가 이동한 이전 위치를 추적하여 값을 업데이트하는 방식
- 유체의 부드러운 흐름을 표현하기 위해 선형 보간법 사용.





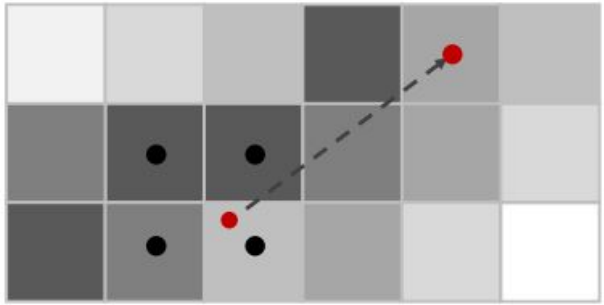
$w_{00} = 1 - x/h \quad w_{01} = x/h$

$w_{10} = 1 - y/h \quad w_{11} = y/h$

$$\bar{v} = w_{00}w_{10}v_{i,j} + w_{01}w_{10}v_{i+1,j} + w_{01}w_{11}v_{i,j+1} + w_{00}w_{11}v_{i+1,j+1}$$

- Smoke

- 연기의 밀도를 각 격자의 중심에 저장
- 연기의 이동 계산(속도 필드와 동일한 방식)
- 밀도 값은 회색 농도로 표현되어 시각적으로 흐름을 확인 가능.

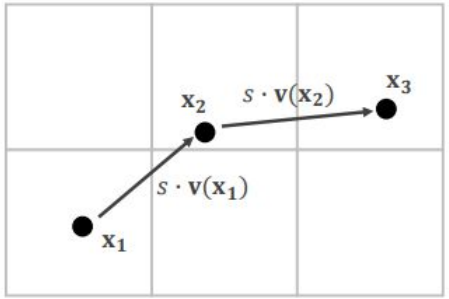


- Streamline

- Streamline은 유체의 흐름 방향과 속도의 시각적 표현
- 특정 시작점 X1, V(X1), X2 (S: step size)계산 후, 반복적으로 새로운 위치를 계산하여 그림

$\mathbf{x} \leftarrow$ start position
$s \leftarrow$ step size

for n steps
  $\mathbf{v} \leftarrow \text{sampleV}(\mathbf{x})$
  $\mathbf{x} \leftarrow \mathbf{x} + s\mathbf{v}$



04

[Implementation]

Simulation

```
integrate(dt, gravity) {
    var n = this.numY;
    for (var i = 1; i < this.numX; i++) {
        for (var j = 1; j < this.numY-1; j++) {
            if (this.s[i*n + j] != 0.0 && this.s[i*n + j-1] != 0.0)
                this.v[i*n + j] += gravity * dt;
        }
    }
}
```

- Integrate(dt, gravity) :

fluid의 속도와 중력 효과 계산

-SolveIncompressibility(numIters,dt)

: density Field 계산, 속도 벡터 조정

```
solveIncompressibility(numIters, dt) {

    var n = this.numY;
    var cp = this.density * this.h / dt;

    for (var iter = 0; iter < numIters; iter++) {

        for (var i = 1; i < this.numX-1; i++) {
            for (var j = 1; j < this.numY-1; j++) {

                if (this.s[i*n + j] == 0.0)
                    continue;

                var s = this.s[i*n + j];
                var sx0 = this.s[(i-1)*n + j];
                var sx1 = this.s[(i+1)*n + j];
                var sy0 = this.s[i*n + j-1];
                var sy1 = this.s[i*n + j+1];
                var s = sx0 + sx1 + sy0 + sy1;
                if (s == 0.0)
                    continue;

                var div = this.u[(i+1)*n + j] - this.u[i*n + j] +
                    this.v[i*n + j+1] - this.v[i*n + j];

                var p = -div / s;
                p *= scene.overRelaxation;
                this.p[i*n + j] += cp * p;

                this.u[i*n + j] -= sx0 * p;
                this.u[(i+1)*n + j] += sx1 * p;
                this.v[i*n + j] -= sy0 * p;
                this.v[i*n + j+1] += sy1 * p;
            }
        }
    }
}
```

Simulation

- SampleField(x,y,field) :

(x,y) 에서 특정 필드 U,V,S FIELD 계산

```
sampleField(x, y, field) {
    var n = this.numY;
    var h = this.h;
    var h1 = 1.0 / h;
    var h2 = 0.5 * h;

    x = Math.max(Math.min(x, this.numX * h), h);
    y = Math.max(Math.min(y, this.numY * h), h);

    var dx = 0.0;
    var dy = 0.0;

    var f;

    switch (field) {
        case U_FIELD: f = this.u; dy = h2; break;
        case V_FIELD: f = this.v; dx = h2; break;
        case S_FIELD: f = this.m; dx = h2; dy = h2; break

    }

    var x0 = Math.min(Math.floor((x-dx)*h1), this.numX-1);
    var tx = ((x-dx) - x0*h) * h1;
    var x1 = Math.min(x0 + 1, this.numX-1);

    var y0 = Math.min(Math.floor((y-dy)*h1), this.numY-1);
    var ty = ((y-dy) - y0*h) * h1;
    var y1 = Math.min(y0 + 1, this.numY-1);

    var sx = 1.0 - tx;
    var sy = 1.0 - ty;

    var val = sx*sy * f[x0*n + y0] +
        tx*sy * f[x1*n + y0] +
        tx*ty * f[x1*n + y1] +
        sx*ty * f[x0*n + y1];

    return val;
}
```

Simulation

```
advectVel(dt) {

    this.newU.set(this.u);
    this.newV.set(this.v);

    var n = this.numY;
    var h = this.h;
    var h2 = 0.5 * h;
    var xScale = 1.0; // x 방향으로 늘려주는 비율
    var yScale = 1.0; // y 방향 비율

    for (var i = 1; i < this.numX; i++) {
        for (var j = 1; j < this.numY; j++) {

            cnt++;

            // u component
            if (this.s[i*n + j] != 0.0 && this.s[(i-1)*n + j] != 0.0 && j < this.numY - 1) {
                var x = i * h * xScale;
                var y = j * h * yScale + h2;
                var u = this.u[i*n + j];
                var v = this.sampleField(x,y, V_FIELD);
                x = x - dt*u;
                y = y - dt*v;
                u = this.sampleField(x,y, U_FIELD);
                this.newU[i*n + j] = u;
            }
            // v component
            if (this.s[i*n + j] != 0.0 && this.s[i*n + j-1] != 0.0 && i < this.numX - 1) {
                var x = i * h * xScale + h2;
                var y = j * h * yScale;
                var u = this.sampleField(x,y, U_FIELD);
                var v = this.v[i*n + j];
                x = x - dt*u;
                y = y - dt*v;
                v = this.sampleField(x,y, V_FIELD);
                this.newV[i*n + j] = v;
            }
        }
    }

    this.u.set(this.newU);
    this.v.set(this.newV);
}
```

```
advectSmoke(dt) {

    this.newM.set(this.m);

    var n = this.numY;
    var h = this.h;
    var h2 = 0.5 * h;
    var xScale = 1.0; // x 방향 크기 비율
    var yScale = 1.0; // y 방향 크기 비율


    for (var i = 1; i < this.numX-1; i++) {
        for (var j = 1; j < this.numY-1; j++) {

            if (this.s[i*n + j] != 0.0) {
                var u = (this.u[i*n + j] + this.u[(i+1)*n + j]) * 0.5;
                var v = (this.v[i*n + j] + this.v[i*n + j+1]) * 0.5;
                var x = i*h*xScale + h2 - dt*u;
                var y = j*h*yScale + h2 - dt*v;

                this.newM[i*n + j] = this.sampleField(x,y, S_FIELD);
            }
        }
    }
    this.m.set(this.newM);
}
```

Draw

- **calculateForces()** : 양력과 항력 계산

```
function calculateForces() {
    var f = scene.fluid; // 유체 객체
    var n = f.numY; // 그리드의 세로 길이
    var lift = 0; // 양력 초기화
    var drag = 0; // 항력 초기화

    // 장애물의 중심과 반지름
    var cx = scene.obstacleX * f.numX; // 장애물 중심 X
    var cy = scene.obstacleY * f.numY; // 장애물 중심 Y
    var rx = scene.obstacleRadiusX * f.numX; // 타원의 X축 반지름
    var ry = scene.obstacleRadiusY * f.numY; // 타원의 Y축 반지름

    // 장애물 주위 셀들 순회
    for (var i = 1; i < f.numX - 1; i++) {
        for (var j = 1; j < f.numY - 1; j++) {
            // 장애물 표면만 고려 (타원 방정식으로 판단)
            var x = (i - cx) / rx; // 타원 기준으로 x 좌표 정규화
            var y = (j - cy) / ry; // 타원 기준으로 y 좌표 정규화
            var isOnSurface = (x * x + y * y) <= 1.0 && f.s[i * n + j] === 0.0;

            if (isOnSurface) {
            // 이웃 셀의 압력
                var px0 = f.p[(i - 1) * n + j]; // 좌측
                var px1 = f.p[(i + 1) * n + j]; // 우측
                var py0 = f.p[i * n + (j - 1)]; // 아래
                var py1 = f.p[i * n + (j + 1)]; // 위

                // x축과 y축 성분의 힘 계산
                var fx = (px1 - px0) * f.h; // x축 압력 차이
                var fy = (py1 - py0) * f.h; // y축 압력 차이

                // 항력과 양력에 합산
                drag += Math.abs(fx);
                lift += fy;
            }
        }
    }
```

Draw

```
function setObstacle(x, y, reset) {

    var vx = 0.0;
    var vy = 0.0;

    if (!reset) {
        vx = (x - scene.obstacleX) / scene.dt;
        vy = (y - scene.obstacleY) / scene.dt;
    }

    scene.obstacleX = x;
    scene.obstacleY = y;
    var rx = scene.obstacleRadiusX; // X축 반지름
    var ry = scene.obstacleRadiusY; // Y축 반지름
    var f = scene.fluid;
    var n = f.numY;
    var cd = Math.sqrt(2) * f.h;

    console.log("Setting Obstacle at:", x, y, "with radii X:", rx, " Y:", ry);
```

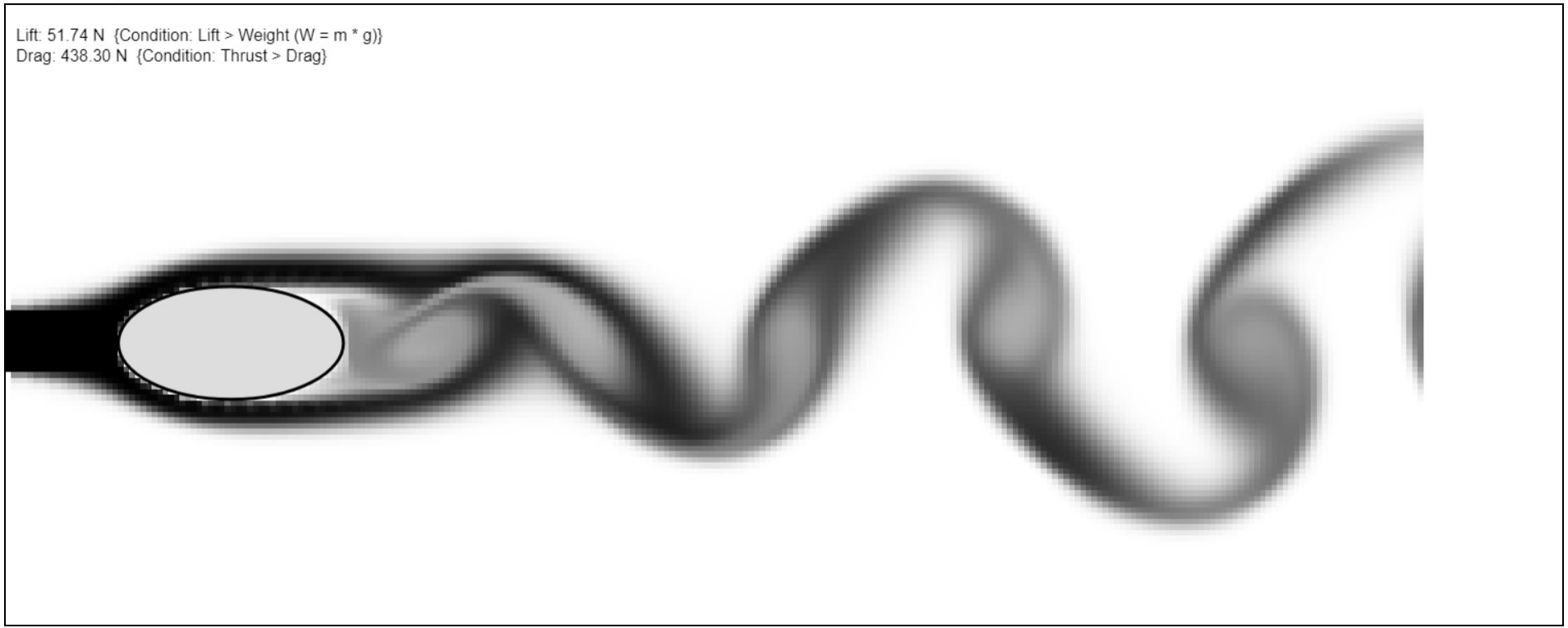
```
    for (var i = 1; i < f.numX-2; i++) {
        for (var j = 1; j < f.numY-2; j++) {

            f.s[i*n + j] = 1.0;

            dx = (i + 0.5) * f.h - x;
            dy = (j + 0.5) * f.h - y;

            if ((dx * dx) / (rx * rx) + (dy * dy) / (ry * ry) < 1) {
                f.s[i*n + j] = 0.0;
                f.u[i*n + j] = vx;
                f.u[(i+1)*n + j] = vx;
                f.v[i*n + j] = vy;
                f.v[i*n + j+1] = vy;
            }
        }
    }

    scene.showObstacle = true;
}
```

05

[Results & Analysis]

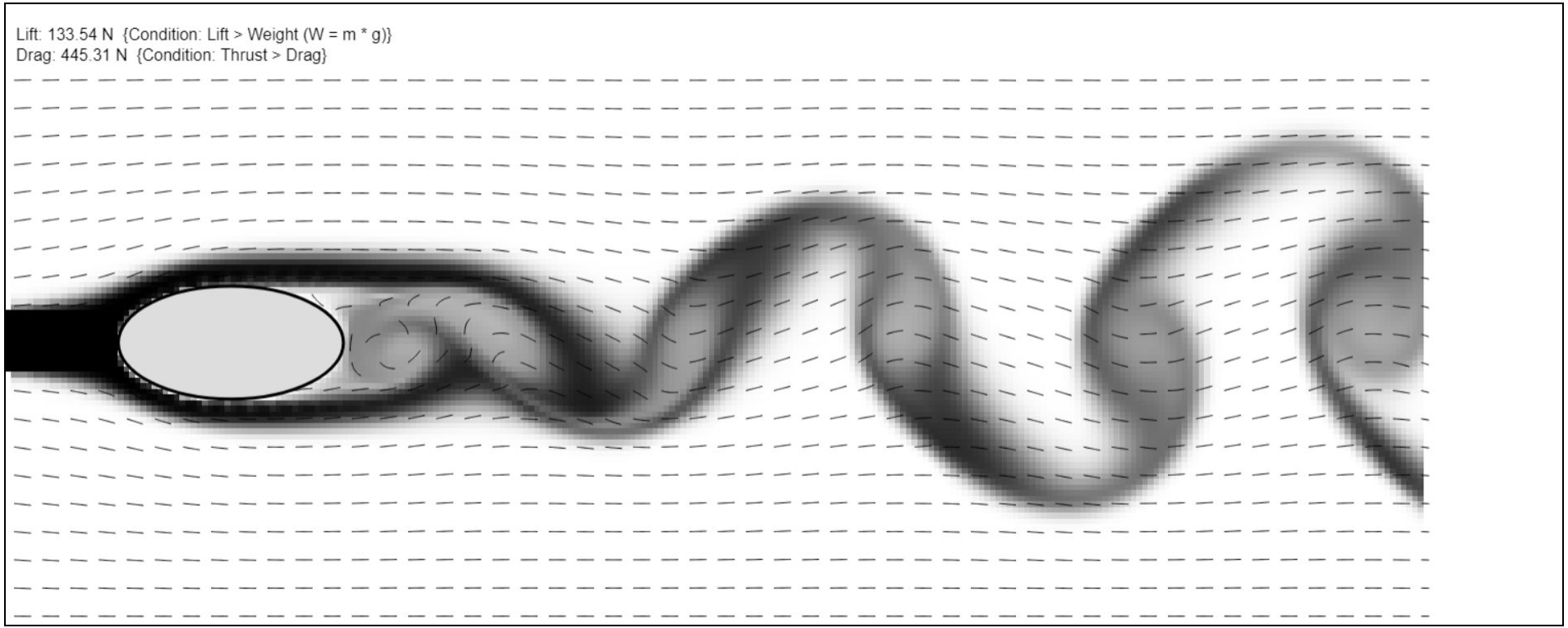
Wind Tunnel - Smoke

Wind Tunnel ☐ Streamlines ☐ Pressure ☑ Overrelax

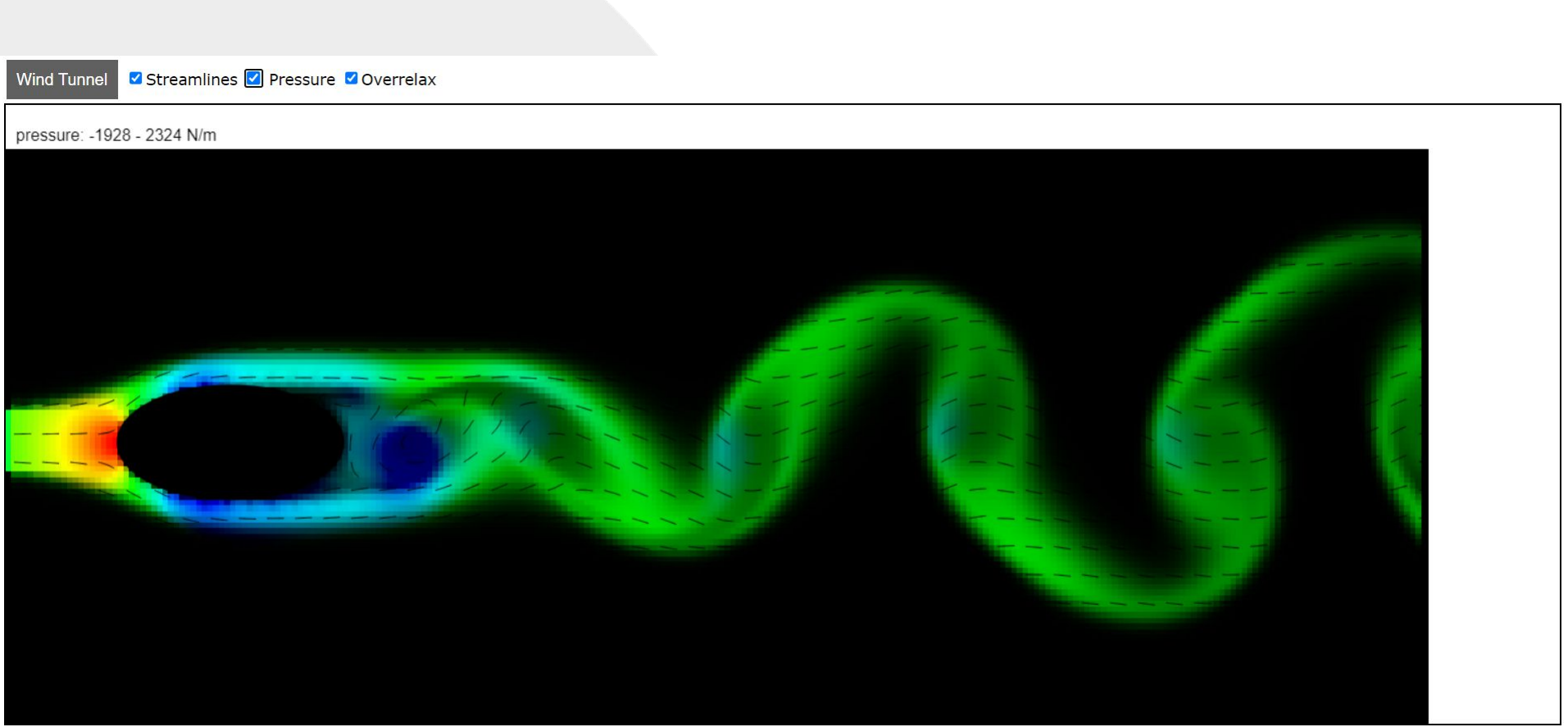


Streamline

Wind Tunnel ☑ Streamlines ☐ Pressure ☑ Overrelax



Pressure


Wind Tunnel
Streamlines
Pressure
Overrelax
pressure: -1928 - 2324 N/m

06

[Challenges]

Challenges

- **3D 확장성**

  : Grid를 3차원으로 확장하여 입체적 효과

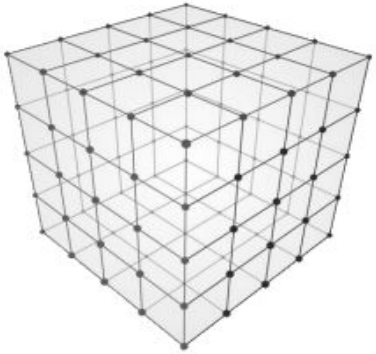
- **Obstacle**

  : 다양한 Object 와 동적 경계조건 구현 가능성

- **Fluid 점성**

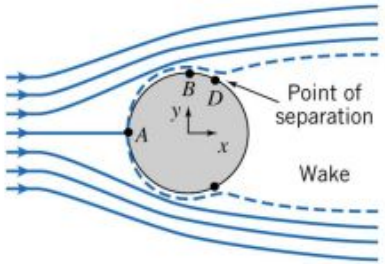
  : 유체의 내부 마찰력(점성)을 표현하여 현실적인 유체 표현

2D => 3D



Plane Object

Viscous Flow



[감사합니다.]