졸업논문청구논문

# 전산 유체역학과 매개변수 곡선을 이용한 실시간 마블링 아트 시뮬레이션

# Real-Time Ebru Art Simulation Based on Particle-Based CFD with Parametric Curves

**최 재 열** (崔 在 烈 **Choi, Jaeyeol)**

20119

**과학영재학교 경기과학고등학교**

**2023**

# 전산 유체역학과 매개변수 곡선을 이용한 실시간 마블링 아트 시뮬레이션

# Real-Time Ebru Art Simulation Based on Particle-Based CFD with Parametric Curves

[논문제출 전 체크리스트]

1. 이 논문은 내가 직접 연구하고 작성한 것이다. ☑
2. 인용한 모든 자료(책, 논문, 인터넷자료 등)의 인용표시를 바르게 하였다. ☑
3. 인용한 자료의 표현이나 내용을 왜곡하지 않았다. ☑
4. 정확한 출처제시 없이 다른 사람의 글이나 아이디어를 가져오지 않았다. ☑
5. 논문 작성 중 도표나 데이터를 조작(위조 혹은 변조)하지 않았다. ☑
6. 다른 친구와 같은 내용의 논문을 제출하지 않았다. ☑

# Real-Time Ebru Art Simulation Based on Particle-Based CFD with Parametric Curves

Advisor : Teacher Choi, Moonseong

by

20119 **Choi, Jaeyeol**

**Gyeonggi Science High School for the gifted**

A thesis submitted to the Gyeonggi Science High School in partial fulfillment of the requirements for the graduation. The study was conducted in accordance with Code of Research Ethics.*

2022. 07. 13

**Approved by**
**Teacher Choi, Moonseong**
**[Thesis Advisor]**

*Declaration of Ethical Conduct in Research: I, as a graduate student of GSHS, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

# 전산 유체역학과 매개변수 곡선을 이용한 실시간 마블링 아트 시뮬레이션

최 재 열

위 논문은 과학영재학교 경기과학고등학교 졸업논문으로
졸업논문심사위원회에서 심사 통과하였음.

2022년 07월 13일

심사위원장　정 태 선　(인)

심사위원　김 보 련　(인)

심사위원　최 문 성　(인)

# Real-Time Ebru Art Simulation Based on Particle-Based CFD with Parametric Curves

## Abstract

The research suggests a fluid interface that could be run in real-time based on Smoothed Particle Hydrodynamics to implement an Ebru art simulation. Besides, it represents a palette layer separated from the fluid interface which was able to draw smooth edges and unique patterns in Ebru art using Bezier curves. Conducted the research, it has shown that main manipulations of Ebru art, stroke and drop, could be reproduced in the simulation.

# 전산 유체역학과 매개변수 곡선을 이용한 실시간 마블링 아트 시뮬레이션

## 초 록

본 연구는 마블링 아트 시뮬레이션을 구현하기 위하여 입자를 기반으로 한 전산 유체역학 모델을 활용하여 실시간으로 외부 조작과 상호작용하는 유체 인터페이스를 제안하였다. 또한, 유체 인터페이스와 별개로 물감 레이어를 제시하고 베지어 곡선을 활용해 부드러운 테두리를 완성하여 마블링 아트에서 보이는 물감의 무늬를 재현할 수 있었다. 마블링 아트에서 주로 사용되는 두 가지 물리적 조작을 시뮬레이션에 반영할 수 있는 방법을 모색하고 실제로 원활하게 작동함을 보였다.

# 목차

# 사용된 그림 목록

# I. 서론

## I.1 연구 동기

마블링 아트란, 얕은 물로 채워진 수조에 물 위에 뜨는 지용성 잉크를 떨어트려서 만드는 예술 작품이다. 표면을 긋고 도중에 다른 잉크를 새로 떨어트리면서 그림을 변형하고, 결과적으로 복잡하고 화려한 무늬를 종이에 찍어내게 된다. 오늘날 디지털 페인팅이 점차 상용화되면서 수채화, 유화 등 다양한 기법으로 만들어지는 작품을 시뮬레이션을 통해 간편하고 사실적으로 표현할 수 있게 되었다. 하지만 마블링 아트에 관련된 시뮬레이션은 아직 표현력이나 조작에 있어서 한계를 보이는 점들이 분명히 있었고, 입자 기반 유체역학을 기반으로 하는 보다 개선된 시뮬레이션을 제시해 보고자 한다.

## I.2 연구 목적

본 연구에서는 유체역학 시뮬레이션과 매개변수 곡선을 기반으로 실시간 상호작용과 날카로운 곡선의 표현을 동시에 만족할 수 있는 마블링 아트 프로세스를 제안하고자 한다. 특히, 외부에서 일어나는 물리적인 상호작용을 적용하기 간편한 입자법(Particle Method)의 일종인 SPH(Smoothed Particle Hydrodynamics)를 통해 외부 조작과 유체 시뮬레이션을 융합할 것이다. 유체의 표면 상에서 마블링 잉크 방울의 변형 과정을 테두리를 이루는 점들의 움직임과 베지어 곡선으로 표현하여 마블링 아트를 유체 시뮬레이션에 덧붙여서 자유자재로 시뮬레이션 가능한 형태로 바꿀 것이다.

# II. 이론적 배경

## II.1 SPH 유체

[1]에서 제시하는 SPH(Smoothed Particle Hydrodynamics)는 연속한 유체 흐름을 작은 입자들의 움직임으로 대응하여 근사하는 방법으로, 각 입자에 적용되는 힘과 속도를 독립적으로 계산하여 다음 프레임을 생성한다. 입자계는 이산적이므로 연속적인 실제 현상과 시뮬레이션을 이어줄 법칙이 필요한데, SPH에서는 간단한 항등식을 통해 이 문제를 해결한다. 유클리드 좌표계 내의 공간 $V$에서 정의된 스칼라 함수 $f(\mathbf{r})$에 대해 다음 식이 성립한다.

$$f(\mathbf{r}) = \int_V f(\mathbf{r}')\delta(\mathbf{r}-\mathbf{r}')d\mathbf{r}' \tag{1}$$

이때 $\delta(\mathbf{r})$은 디랙 델타 함수이다. SPH에서는 연속한 유체의 흐름을 입자계로 정밀하게 이산화하기 위한 방법으로 커널 함수 $W(\mathbf{r},h)$를 사용한다. 이는 $\delta(\mathbf{r})$을 대체하는 함수로, $\lim_{h\to 0} W(\mathbf{r},h) = \delta(\mathbf{r})$을 만족하는 방사형 함수이다. 앞서 언급한 스칼라 함수에 대한 항등식을 커널을 사용하여 모든 입자에 대한 합으로 구할 수 있다. 이때, 각 입자의 질량은 $m_i$이며 해당 위치 $\mathbf{r}_i$에서 유체의 밀도를 $\rho_i$로 정의한다.

$$f(\mathbf{r}) = \sum_i \frac{m_i}{\rho_i} f(\mathbf{r}_i) W(\mathbf{r}-\mathbf{r}_i,h) \tag{2}$$

이제 임의의 위치에서 연속함수의 값을 근사할 수 있는 공식을 얻었으므로 이를 활용하여 다음과 같이 스칼라장의 기울기를 구할 수 있다. 벡터장 발산의 근사식과도 매우 유사한 형태이다.

$$\nabla f(\mathbf{r}) = \sum_i m_i \frac{f(\mathbf{r}_i) - f(\mathbf{r})}{\rho_i} \nabla W(\mathbf{r}-\mathbf{r}_i,h) \tag{3}$$

이제 위의 원리를 바탕으로 유체 입자의 운동을 해결할 수 있다. 비압축성 및 비점성 유체의 운동방정식은 오일러 유체방정식으로 설명할 수 있다. 이 식을 이용하여 각 유체 입자의 속도를 계산할 수 있다. $P$는 압력, $\mathbf{f}$는 외력을 나타낸다.

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\frac{1}{\rho}\nabla P + \frac{1}{\rho}\mathbf{f} \tag{4}$$

시뮬레이션에서 가장 먼저 이루어지는 것은 밀도와 압력의 계산이다. 밀도는 제시된 식을 통해 간단하게 계산할 수 있고, 압력은 유체의 밀도에 비례한다는 가정을 사용하여 보편적으로 다음과 같이 구한다.

$$\rho(\mathbf{r}) = \sum_i m_i W(\mathbf{r} - \mathbf{r}_i, h), P(\mathbf{r}) = k(\rho(\mathbf{r}) - \rho_0) \tag{5}$$

이제 유체 내부에서 유도되는 힘을 계산할 차례이다. 같은 항을 유도하는 데에도 다양한 접근이 있으므로 그 중에 계산 속도와 계산 결과의 안정성 사이 절충안을 사용해야 한다. [1]에서는 $\nabla P$를 $\rho$로 나누는 대신 한 번의 계산으로 구할 수 있는 방법을 제시하였다.

$$\nabla(\frac{P}{\rho}) = -P\frac{\nabla\rho}{\rho^2} + \frac{\nabla P}{\rho} \tag{6}$$

$$\frac{\nabla P}{\rho} = \nabla(\frac{P}{\rho}) + P\frac{\nabla\rho}{\rho^2} = \sum_i m_i(\frac{P(\mathbf{r})}{\rho(\mathbf{r})^2} + \frac{P(\mathbf{r}_i)}{\rho(\mathbf{r}_i)^2})\nabla W(\mathbf{r} - \mathbf{r}_i, h) \tag{7}$$

## II.2 마블링 무늬의 수학적 모델

마블링 아트에서 이루어지는 기본적인 조작은 바로 선(stroke)이다. 예리한 물체로 유체의 표면에 자유롭게 선을 긋는 방식으로 진행된다. 이 상호작용을 시뮬레이션에 구현하려면 인위적인 조작이 유체의 움직임이 어떤 영향을 주는지 모델링할 필요가 있다. [2]에서는 이 현상을 구현하기 위해 직접 벡터장을 변환하는 함수를 제시하였다. 접촉점을 원점으로 하는 극좌표계에서 물리적인 접촉에 의해 발생하는 변화를 벡터장 $\mathbf{F}$로 정의했을 때, 이는 비압축성 유체에서 다음과 같은 조건을 만족한다.

$$\nabla \cdot \mathbf{F}(r,\theta) = \frac{1}{r}\frac{\partial \mathbf{F}_r}{\partial r} + \frac{1}{r}\frac{\partial \mathbf{F}_\theta}{\partial \theta} = 0, \|\lim_{r\to\infty} \mathbf{F}(r,\theta)\| = 0 \tag{8}$$

접촉점에서 물체를 움직이는 속도가 원점에서 $\mathbf{F}$의 크기를 결정할 것이다. 접촉점이 x축의 양의 방향으로 속력 $U$로 움직이고 있다고 생각하면 $\mathbf{F}$의 직교분해 $\mathbf{F}_r, \mathbf{F}_\theta$는 원점에서 다음 조건을 만족하게 된다.

$$\mathbf{F}_r(0,\theta) = U\cos\theta, \mathbf{F}_\theta(0,\theta) = -U\sin\theta \tag{9}$$

[2]에서는 해당 조건들을 모두 만족하는 해를 다음과 같이 삼각함수와 지수함수의 조합으로 제시하였다. 이때 $L$은 지수의 차원을 상수로 맞추어 주기 위한 상수이다.

$$\mathbf{F}_r(r,\theta) = U\cos\theta\exp\frac{-r}{L}, \mathbf{F}_\theta(r,\theta) = [\frac{r}{L}-1]U\sin\theta\exp\frac{-r}{L} \tag{10}$$

$F$를 접촉점을 원점으로 하는 카테시안 좌표로 되돌릴 수 있다.

$$let r = \sqrt{x^2+y^2}, \mathbf{F}_x = U\frac{rL-y^2}{rL\exp\frac{r}{L}}, \mathbf{F}_y = U\frac{xy}{rL\exp\frac{r}{L}} \tag{11}$$

이제 해당 함수를 변형하여, 한 프레임 동안 접촉점이 **B**에서 **E**로 이동하였을 때 임의의 좌표 **P**에 대한 시간 $\Delta t$초 이후의 mapping **Q**를 다음과 같이 일반화할 수 있다.

$$\mathbf{Q}(\mathbf{P},\mathbf{B},\mathbf{E},L) = \mathbf{P} + \begin{pmatrix}\mathbf{n}_x & -\mathbf{n}_y \\ \mathbf{n}_y & \mathbf{n}_x\end{pmatrix} \cdot \mathbf{F}(x, y, \frac{\|\mathbf{v}\|}{\Delta t}, L)\cdot\Delta t \tag{12}$$

$$\mathbf{v} = \mathbf{E}-\mathbf{B}, \mathbf{n} = \frac{\mathbf{v}}{\|\mathbf{v}\|}, x = \mathbf{n}\cdot(\mathbf{P}-\mathbf{B}), y = \|\mathbf{n}\times(\mathbf{P}-\mathbf{B})\| \tag{13}$$

## II.3 베지어 곡선

베지어 곡선은 컴퓨터 그래픽스 및 응용 분야에서 사용되는 매개변수 곡선(parametric curve)의 일종이다. 양 끝점과 몇 개의 제어점으로 정의할 수 있고, 이들을 매개변수 $t$에 대한 다항식으로 정의하여 두 점을 부드러운 곡선으로 이어주게 된다. 일반적으로 3차 다항식을 쓰며, 점 $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$이 순서대로 주어질 때 $\mathbf{P}_0$와 $\mathbf{P}_3$을 잇는 다항식을 다음과 같이 정의할 수 있다.

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3, 0\le t\le 1 \tag{14}$$

베지어 곡선의 $t$에 대한 도함수와 이계도함수는 다음과 같다.

$$\mathbf{B}'(t) = 3(1-t)^2(\mathbf{P}_1-\mathbf{P}_0) + 6(1-t)t(\mathbf{P}_2-\mathbf{P}_1) + 3t^2(\mathbf{P}_3-\mathbf{P}_2) \tag{15}$$

$$\mathbf{B}''(t) = 6(1-t)(\mathbf{P}_2-2\mathbf{P}_1+\mathbf{P}_0) + 6t(\mathbf{P}_3-2\mathbf{P}_2+\mathbf{P}_1) \tag{16}$$

# III. 제안하는 시스템

## III.1 마블링 아트를 위한 유체 구현

본 연구에서는 물감의 표현 및 물리적인 조작과 관련된 모든 것이 이루어지는 유체의 인터페이스를 SPH의 원리를 적용하여 구현하였다. 시뮬레이션에는 총 $N$개의 유체 입자가 존재하며, 매 프레임마다 가상의 시간 간격 $\Delta t = t_n - t_{n-1}$을 두고 모든 입자를 동시에 갱신한다. 각 입자 $p$에는 입자의 2차원 직교좌표 기준 위치 $\mathbf{x}_p$와 속도 $\mathbf{v}_p$, 밀도 $\rho_p$와 압력 $P_p$가 할당된다. 이들은 모두 시간 $t$에 대해 변화하는 값이다.

SPH 시뮬레이션의 가장 큰 문제점은 각 위치에서 모든 입자에 대한 합을 계산할 때 $O(N^2)$이라는 시간복잡도를 소모하기 때문에 입자 수를 늘리기 곤란하다는 것이다. 본 연구에서는 이 문제를 합을 계산할 때 해당 위치에서 일정 반경 내에 분포하는 입자들만 계산에 사용하는 방법으로 해결하였다. 적절하게 정해진 반경 내의 입자 수가 N에 비해 충분히 작을 때 시뮬레이션은 밀도, 압력, 및 속도를 계산하는 전 과정에서 $O(N)$의 시간복잡도를 갖게 된다. 본 연구에서는 값을 계산하기 전에, 가능한 모든 입자 쌍에 대해 입자 간의 거리가 일정값 이하인지 검사한 후 해당 쌍을 저장한다. 이 과정은 매번 반복이 시작될 때 한 번씩만 실행되며, $O(N^2)$의 시간복잡도를 가지지만 그 이후로 일어나는 모든 계산에서 연산 수 및 함수 호출 횟수를 효과적으로 줄일 수 있다.

2.1에서 언급한 커널 함수 $W(\mathbf{r},h)$는 다양하게 정할 수 있는데 본 연구에서는 이 중 [3]에서 제시한 cubic spline kernel 함수를 사용하였다. 이 함수는 원점에서 떨어진 거리 $\|\mathbf{r}\|$이 $2h$ 이상일때 값이 0인 특성을 가지고 다항함수로 표현이 가능하기 때문에 계산이 용이하다. 제시된 커널 함수와 기울기를 다음과 같이 나타낼 수 있다. 이때, $x = \|\mathbf{r}\|/h$로 생각한다.

$$W(\mathbf{r},h) = \frac{1}{\pi h^3}\begin{cases} 1-\frac{3}{2}x^2+\frac{3}{4}x^3, & 0 \le x \le 1 \\ \frac{1}{4}(2-x)^3, & 1 \le x \le 2 \\ 0, & x \ge 2 \end{cases} \tag{17}$$

$$\nabla W(\mathbf{r},h) = \frac{\mathbf{r}}{\|\mathbf{r}\|}\frac{1}{\pi h^4}\begin{cases}\frac{9}{4}x^2 - 3x, & 0 \le x \le 1\\ -\frac{3}{4}(2-x)^2, & 1 \le x \le 2\\ 0, & x \ge 2\end{cases} \tag{18}$$

다음은 유체 인터페이스의 입자들을 SPH의 원리에 기반하여 갱신하는 알고리즘이다.

**Algorithm 1** Fluid Simulation Based on SPH

```
PAIRS, PARTICLES                                   ▷ list of pairs, particles
procedure ADDPAIRS
    clear PAIRS
    for p in PARTICLES do
        for q ≠ p in PARTICLES do
            d ← ‖x_p − x_q‖
            if d < 2h then add (p,q) to PAIRS
end procedure
procedure CALCULATE
    for (p,q) in PAIRS do
        Δρ ← W(x_p − x_q, h)                       ▷ mass of each particle is fixed to 1
        ρ_p ← ρ_p + Δρ
        ρ_q ← ρ_q + Δρ
    for p in PARTICLES do
        P_p ← k(ρ_p − ρ_0)                         ▷ Constrain the values between 0 and P_max
end procedure
procedure SPH
    for (p,q) in PAIRS do
        P_pq ← P_p/ρ_p^2 + P_q/ρ_q^2
        a ← −P · ∇W(x_p − x_q, h)                  ▷ acceleration
        Δv ← aΔt
        v_p ← v_p + Δv
        v_q ← v_q − Δv
end procedure
procedure MOVE
    for p in PARTICLES do
        x_p ← x_p + v_pΔt
end procedure
```

하지만 이렇게 구현된 시뮬레이션에서는 유체 입자들이 쉬지 않고 공간 상에서 흐르는 현상을 관찰할 수 있었고, 점성에 의한 효과를 고려하지 않은 것이 그 원인이라고 추정하

였다. 이렇게 되면 실제 결과물을 만드는 과정에서 물감이 형태를 유지하지 못할 것이고 이는 연구의 본래 목적을 훼손하게 된다. 본 연구에서는 이 문제를 해결하기 위하여 유체에 점성을 추가할 수 있는 방법을 모색했고, [4]에서 제시한 점성의 의도를 살리면서 빠르고 안정적인 구현이 가능한 XSPH를 사용하였다. XSPH에서는 기존 계산에서 주변 입자의 흐름과 속도를 비슷하게 조정하는 과정을 추가하여 보다 정적인 유체 인터페이스를 표현한다. 해당 과정을 표현하는 수식은 다음과 같다.

$$\widehat{\mathbf{v}}_i = \mathbf{v}_i - \varepsilon \sum_j \frac{m_j}{\bar{\rho}_{ij}} (\mathbf{v}_i - \mathbf{v}_j) W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{19}$$

$\bar{\rho}_{ij}$는 $i$번째 입자와 $j$번째 입자 간의 밀도의 평균이며, 조화평균을 사용한다.

$$\frac{1}{\bar{\rho}_{ij}} = \frac{1}{2}\left(\frac{1}{\rho_i} + \frac{1}{\rho_j}\right) \tag{20}$$

이 과정을 통해 유체의 속도장을 부드럽게 만들어주어 점성 효과를 인위적으로 형성해 준다. 이렇게 하면 유체에 물리적인 변화가 이루어져도 시간이 지났을 때 안정한 상태로 수렴하게 된다. XSPH에 기반하여 수정된 갱신 알고리즘은 다음과 같다. 기존 SPH에서 사용하던 연산을 입자에 먼저 적용하고, 만들어진 속도장을 바탕으로 점성 효과를 계산하여 추가하는 방식이다.

**Algorithm 2** Velocity Modification Based on XSPH

```
    procedure XSPH
2:      for (p,q) in PAIRS do
            Δv ← v_q − v_p
4:          Δv ← (1/2)(1/ρ_p + 1/ρ_q) W(x_p − x_q, h) Δv
            v_p ← v_p + Δv
6:          v_q ← v_q − Δv
    end procedure
8:  procedure MOVE
        for p in PARTICLES do
10:         x_p ← x_p + v_p Δt
    end procedure
```

이때, $\varepsilon$은 유체의 움직임에 점성이 얼마나 영향을 미치는지 결정하는 0에서 1 사이의 계수이다. 다음 그림은 유체 인터페이스가 만들어지고 실행되는 과정을 앞서 제시한 의사코드를 토대로 나타내는 다이어그램이며, 다음과 같이 만들어지는 결과물을 시각화해 보았다.

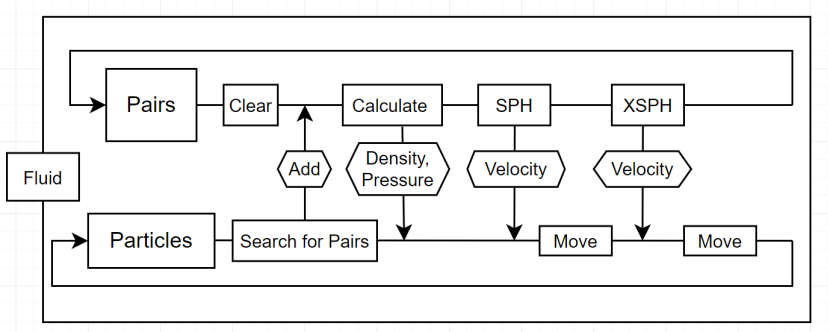



**그림 1.** 유체 시뮬레이션의 전 과정을 나타내는 다이어그램. 적절한 입자들의 쌍을 찾아서 저장한 뒤, 압력과 밀도를 계산한 후 SPH를 적용하여 다음 프레임에서 유체의 속도를 얻고, XSPH를 통해 조정 과정을 한 번 거쳐준다.

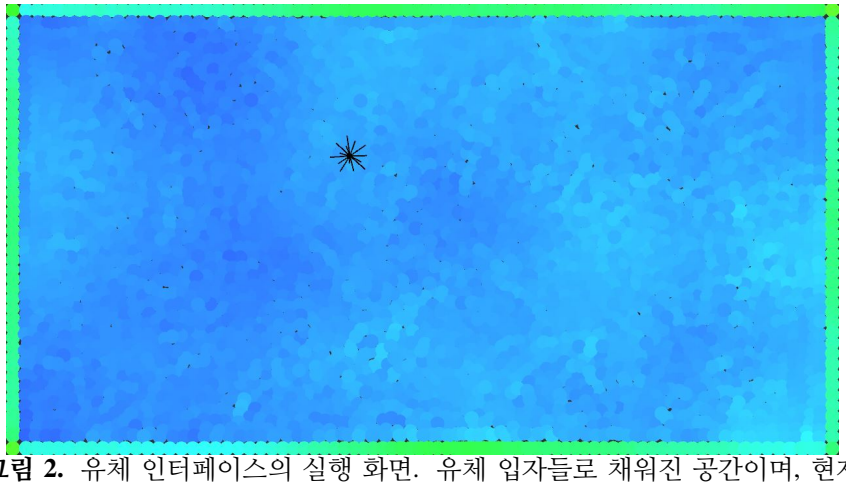


**그림 2.** 유체 인터페이스의 실행 화면. 유체 입자들로 채워진 공간이며, 현재 입자의 색은 각각의 압력을 나타낸다. 검은 선은 하나의 입자에서 쌍이 어떻게 형성되는지 나타낸 것이다.

## III.2 색의 경계에 대한 베지어 곡선

앞서 구현한 유체 인터페이스로는 마블링 아트를 제대로 재현해 내는 데 한계가 있다. 입자로 표현하는 유체 시뮬레이션 특성상 각 입자들은 마블링 아트에서 보이는 양상과 다르게 세밀하고 뚜렷한 무늬 표현이 불가능하고, 물리적 조작을 반복하면 흩어져 나가게 된다. 대신에 본 연구에서는 얇은 유체 위에 떨어진 잉크는 표면의 입자들과 섞여서 흐름을 차지하는 것이 아니라 표면 위에서 미끄러지며 아래에 있는 유체의 흐름을 따라간다고 생각한다. 서로 다른 물감 방울들이 위에서 겹칠 때 이들을 모두 유체 입자들로 취급하면 처리하기 매우 복잡한 상황이 되고, 크고 작은 오류를 범할 것이다. 그러므로 유체 입자의 움직임과 육안으로 확인할 수 있는 물감의 형태 변형은 다르게 보아야 한다고 판단했고, 본 연구에서는 물감이 이동하는 영역과 유체의 영역을 다른 두 레이어로 분리하여 생각하였다.

마블링 아트에서 요구하는 사항 중 하나는 물감 방울을 렌더링했을때 형태가 번지지 않고 깔끔하게 이어지는 곡선이다. 이를 만족시키려면 물감 방울의 테두리가 되는 점들을 정의하고 곡선으로 이어 주어야 한다. 불필요한 연산을 줄이기 위하여 본 연구에서는 물감 내부의 입자까지 일일이 구현하는 대신, 물감 방울에서 하나의 테두리를 구성하는 점열을 각각 '색의 경계'로 정의하고 레이어에서는 색의 경계만 다룰 수 있도록 하여 훨씬 효율적인 전처리를 설계했다. 유체 인터페이스의 입자 움직임이 물감 레이어(palette)에서 색의 경계의 움직임에 실시간으로 영향을 줄 수 있도록 설계하였다.

물감과 유체의 상호작용을 다루기 이전에, 본 연구에서는 색의 경계를 하나의 부드러운 곡선으로 렌더링하는 방법을 제시한다. 기존의 베지어 곡선은 곡선이 지나지 않는 부수적인 점인 두 제어점을 필요로 한다. 하지만 현재 문제에서는 제어점 대신에, 양 끝점과 함께 인접한 곡선이 서로 자연스럽게 이어져야 한다는 조건이 주어졌다. 그러므로 두 점 $\mathbf{P}_n, \mathbf{P}_{n+1}$을 양 끝점으로 하는 곡선 $\mathbf{B}_n$를 구성하는 두 제어점을 미지수 $\alpha_n, \beta_n$를 통해 $\mathbf{B}_n(t) = (1-t)^3\mathbf{P}_n + 3(1-t)^2 t\alpha_n + 3(1-t)t^2\beta_n + t^3\mathbf{P}_{n+1}$로 표현하고 서로 만나는 두 곡선의 매개변수

$t$에 대한 도함수와 이계도함수가 같다는 식을 사용하여 해를 구한다.

$\alpha_n, \beta_n$에 대한 연립방정식을 세우는 과정은 다음과 같다. 색의 경계는 $\mathbf{P}_0, \mathbf{P}_1, \cdots, \mathbf{P}_N$으로 주어져 있다고 가정한다. 인접한 두 곡선 $\mathbf{B}_n$와 $\mathbf{B}_{n+1}$이 만나는 점에서 $t$에 대한 도함수와 이계도함수의 값이 같다는 것을 방정식으로 나타낼 수 있다.

$$\mathbf{B}'_{n-1}(1) = \mathbf{B}'_n(0) \tag{21}$$

$$3(\beta_{n-1} - \mathbf{P}_n) = 3(\alpha_n - \mathbf{P}_n), \alpha_n + \beta_{n-1} = 2\mathbf{P}_n \tag{22}$$

$$\mathbf{B}''_{n-1}(1) = \mathbf{B}''_n(0) \tag{23}$$

$$6(\mathbf{P}_n - 2\beta_{n-1} + \alpha_{n-1}) = 6(\beta_n - 2\alpha_n + \mathbf{P}_n), 2\beta_{n-1} - \alpha_{n-1} = 2\alpha_n - \beta_n \tag{24}$$

$\mathfrak{R}_2$ 형태의 벡터 $\alpha_n (n = 0, 1, \cdots, N-1)$를 먼저 구하면, 식에 대입하여 벡터 $\beta_n$도 구할 수 있다. 하지만, 미지수의 수가 만들어진 방정식보다 많다는 것을 알 수 있고 이를 보완하기 위해 본 연구에서는 경계조건을 다음과 같이 추가하였다.

$$\mathbf{B}''_{N-1}(1) = \mathbf{B}''(0) = 0 \tag{25}$$

$$\beta_0 = 2\alpha_0 - \mathbf{P}_0, 2\beta_{N-1} = \alpha_{N-1} + \mathbf{P}_N \tag{26}$$

이렇게 주어진 모든 조건을 $\alpha_n$와 $\mathbf{P}_n$만이 남도록 정리하면 선형 연립방정식이 만들어지고, 이를 행렬로 표현하면 다음과 같이 나타낼 수 있다.

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 4 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 4 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & 4 & 1 \\ 0 & \cdots & 0 & 0 & 0 & 2 & 7 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{N-3} \\ \alpha_{N-2} \\ \alpha_{N-1} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_0 + 2\mathbf{P}_1 \\ 4\mathbf{P}_1 + 2\mathbf{P}_2 \\ 4\mathbf{P}_2 + 2\mathbf{P}_3 \\ \vdots \\ 4\mathbf{P}_{N-3} + 2\mathbf{P}_{N-2} \\ 4\mathbf{P}_{N-2} + 2\mathbf{P}_{N-1} \\ 8\mathbf{P}_{N-1} + \mathbf{P}_N \end{bmatrix}$$

위 행렬은 삼대각(Tridiagonal) 행렬 형태를 만족하므로 [5]에서 제시하는 알고리즘을 사용하여 간단한 소거법으로 $O(N)$의 시간복잡도에 해결할 수 있다. 본 연구는 시뮬레이션 도중

점의 추가와 삭제에 유연하게 대응하고 해당 알고리즘을 편리하게 사용하기 위하여 System 이라는 객체를 정의하여 사용하였다. Palette의 점 하나가 System 객체에 대응되며, 색의 경계를 이루는 System 객체들을 순차적으로 계산에 사용하여 해를 구하게 된다. System 객체는 대응하는 점 $\mathbf{P}$, 계산 도중 발생하는 값들을 저장하기 위한 변수 $\mathbf{r}, \mathbf{R}, Z$, 해를 저장하는 $\alpha, \beta$로 이루어져 있다. 다음은 앞서 제시한 방정식의 해를 Thomas' Algorithm을 적용하여 해결하는 과정을 나타낸 것이다.

**Algorithm 3** Applying Thomas' Algorithm with System Objects

A Border is given as $\mathbf{P}_0, \mathbf{P}_1, \cdots, \mathbf{P}_N$ ▷ A system holds $\mathbf{P}, \mathbf{r}, \mathbf{R}, Z, \alpha, \beta$
**function** SET($n$) ▷ $n = 0, 1, 2, \cdots, N-1$
  **if** $n = 0$ **then**
    $\mathbf{r}_n \leftarrow \mathbf{P}_n + 2\mathbf{P}_{n+1}$
    $\mathbf{R}_n \leftarrow 0.5\mathbf{r}_n$
    $Z_n \leftarrow 0.5$
  **else if** $0 < n < N-1$ **then**
    $\mathbf{r}_n \leftarrow 4\mathbf{P}_n + 2\mathbf{P}_{n+1}$
    $\mathbf{R}_n \leftarrow \frac{1}{4-Z_{n-1}}(\mathbf{r}_n - \mathbf{R}_{n-1})$
    $Z_n \leftarrow \frac{1}{4-Z_{n-1}}$
  **else if** $n = N-1$ **then**
    $\mathbf{r}_n \leftarrow 8\mathbf{P}_n + \mathbf{P}_{n+1}$
    $\mathbf{R}_n \leftarrow \frac{1}{7-2Z_{n-1}}(\mathbf{r}_n - 2\mathbf{R}_{n-1})$
**end function**
**function** SOLVE($n$)
  **if** $n = N-1$ **then**
    $\alpha_n \leftarrow \mathbf{R}_n$
    $\beta_n \leftarrow 0.5(\alpha_n + \mathbf{P}_{n+1})$
  **else**
    $\alpha_n \leftarrow \mathbf{R}_n - Z_n\alpha_{n+1}$
    $\beta_n \leftarrow 2\mathbf{P}_{n+1} - \alpha_n$
**end function**
**procedure** CURVE ▷ Get formula of curves $\mathbf{B}_n$
  **for** $n = 0, 1, \cdots, N-1$ **do**
    SET($n$)
  **for** $n = N-1, \cdots, 1, 0$ **do**
    SOLVE($n$)
**end procedure**

적절한 곡선 $\mathbf{B}_n$을 찾아내면 구간 $[0,1]$에서 적절한 $m$개의 실수 $t_1 < t_2 < \cdots < t_m$를 골라 $t$에 대입하여 점열 $\mathbf{B}_{ni} = \mathbf{B}_n(t_i)$을 얻을 수 있고, 해당 점열을 선분으로 이어주면 곡선의 모양이 만들어진다. 이렇게 이어서 만들어진 하나의 큰 다각형이 색칠되는 영역이 된다. 물감 레이어 내의 모든 경계에 대해 이 과정이 끝났으면 경계가 생성된 순서대로 내부를 칠하면 된다. 나중에 떨어진 물감이 먼저 떨어진 물감 층을 덮어야 한다.

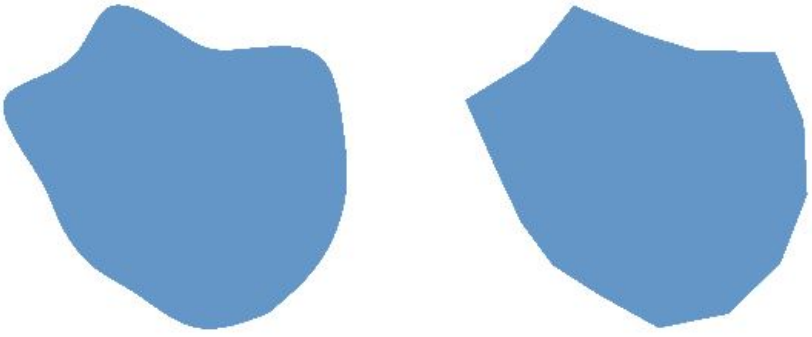


**그림 3.** 무작위로 생성된 다각형과(오른쪽), 앞서 제시안 방법을 사용하여 매끄럽게 보정한 다각형(왼쪽).

## III.3 유체의 물리적 조작 및 변형

본 연구에서는 크게 두 가지, 스트로크(stroke)와 드롭(drop)을 시뮬레이션의 주된 조작으로 다루게 된다. stroke란 물감을 떨어트린 유체 표면에 얇은 막대로 곡선 형태의 획을 그어 무늬를 생성하는 과정이다. drop은 새로운 물감 방울을 지정한 위치에 떨어트리는 과정으로, 이때 그 지점을 중심으로 원형 무늬가 생기게 된다.

일반적인 방법으로는 stroke에서 요구하는 물리적인 상호작용을 구현하기 까다롭다. 얇은 막대로 유체의 표면에 가하는 stroke는 단순히 특정 입자를 잡고 움직이는 것으로 해결되는것이 아니기 때문이다. 본 연구에서는 막대가 유체와 상호작용하는 과정을 직접 모델링하지 않고, [3]에서 제시한 실제 stroke에서 일어나는 변환과 유사한 벡터장을 전체 변환에

추가하는 행위로 표현하였다.

현 문제에서 구현해야 하는 것은 마우스를 클릭한 후 자유롭게 이동하면서 유체에 변화를 주는 것이다. 시뮬레이션에서 반드시 만족해야 할 것은 인터페이스가 끊어지지 않는 자연스러운 변화가 일어나야 한다는 것과, 변형 이후 마우스가 지나간 경로 부근의 영역 이외에는 형태에 큰 변화가 일어나지 않아야 한다는 점이다.

본 연구에서는 매 반복마다 마우스 포인터의 움직임을 계산하여 이를 점진적인 변화에 반영하는 방법을 제시한다. 앞서 언급한 벡터장을 계산한 뒤, 이를 연속적인 흐름이 이루어져야 하는 유체 입자들을 가속시키는 데 적용하고, 동시에 Palette의 점들을 직접 이동시키는 데에도 사용한다. 빠른 마우스의 움직임에 안정적인 변환을 유지하기 위해 한 프레임 동안 마우스의 위치 변화를 여러 단계로 쪼개어서 변환을 적용하였다. 다음은 마우스가 움직이는 동안 stroke가 표현되는 과정을 설명하는 알고리즘이다.

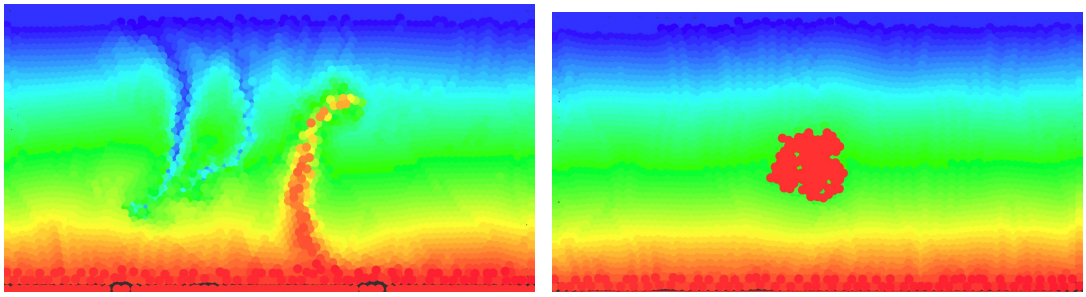
**Algorithm 4** Implementation of Stroke

Stroke step size $D$, list PARTICLES, POINTS
**function** DISPLACEMENT(x, y, U, L) ▷ using the formula
  $r \leftarrow \sqrt{x^2+y^2}$
  $\mathbf{d} \leftarrow (rL-y^2, xy)$
  $\mathbf{d} \leftarrow \frac{U}{rL\exp(r/L)}\mathbf{d}$
  **return d**
**end function**
**procedure** STROKE($\mathbf{O}_{old}, \mathbf{O}_{new}$) ▷ **O** is the position of the pointer
  $\mathbf{O} \leftarrow \mathbf{O}_{old}$
  $\mathbf{o} \leftarrow \mathbf{O}_{new} - \mathbf{O}_{old}$
  $\mathbf{n} = (n_x, n_y) \leftarrow \frac{\mathbf{o}}{\|\mathbf{o}\|}$
  $\mathbf{s}_x \leftarrow (n_x, -n_y)$
  $\mathbf{s}_y \leftarrow (n_y, n_x)$
  **for** $i = 0, 1, \cdots, D-1$ **do** ▷ divide total stroke in $D$ steps
    $\mathbf{O} \leftarrow \mathbf{O} + \frac{1}{D}\mathbf{o}$
    **for** p in PARTICLES **do**
      $\mathbf{c} \leftarrow \mathbf{x}_p - \mathbf{O}$
      $\mathbf{d} \leftarrow$ DISPLACEMENT$(\mathbf{n}\cdot\mathbf{c}, \|\mathbf{n}\times\mathbf{c}\|, k_1\frac{\|\mathbf{o}\|}{\Delta t}, L_1)\cdot\frac{1}{D}$
      $\mathbf{v}_p \leftarrow \mathbf{v}_p + (\mathbf{s}_x\cdot\mathbf{d}, \mathbf{s}_y\cdot\mathbf{d})$

**Algorithm 5** Implementation of Stroke

**for P** in POINTS **do** ▷ update particles' position
  $\mathbf{c} \leftarrow \mathbf{P} - \mathbf{O}$
  $\mathbf{d} \leftarrow \text{DISPLACEMENT}(\mathbf{n}\cdot\mathbf{c}, \|\mathbf{n}\times\mathbf{c}\|, k_2\frac{\|\mathbf{o}\|}{\Delta t}, L_2)\cdot\frac{\Delta t}{D}$ ▷ evaluate
  $\mathbf{P} \leftarrow \mathbf{P} + (\mathbf{s}_x\cdot\mathbf{d}, \mathbf{s}_y\cdot\mathbf{d})$
**end procedure**

또한 drop, 즉 물감을 떨어트렸을 때 일어나는 현상을 구현해야 한다. 이에 대해서는 이미 퍼져 있는 방울은 사방으로 밀어내고, 자기 자신은 넓게 퍼지려는 현상이 관찰되었다. 본 연구에서는 새로운 위치에 물감을 떨어트렸을 때, 해당 위치 근처에 일정 시간 동안 새로운 유체 입자를 반복적으로 생성하고, 그와 동시에 경계에 머물러 있는 입자를 제거하는 메커니즘을 설계했다.



**(a)** **(b)**

**그림 4.** (a)은 stroke, (b)는 drop 과정 이후 유체 인터페이스에 생긴 변화를 나타낸 것이다. 입자의 색은 초기위치를 나타내며, 변화를 육안으로 설명하기 위한 것이다.

## III.4 유체와 물감의 상호작용

색의 경계의 움직임은 유체 인터페이스의 흐름에 영향을 받아야 한다. 본 연구에서는 색의 경계가 실시간으로 유체에 떠다니는 느낌을 부여하기 위한 방법을 제시한다. Palette의 색의 경계를 이루는 점들도 유체 입자와 비슷하게 각자의 고유한 속도와 위치를 갖는다. 앞서 사용한 방식과 비슷하게 경계 입자 근방(거리 $h'$ 이내)에 있는 유체 입자들을 찾고, 이들의 평균 속도를 해당 입자의 속도로 설정한다. Palette의 입자 수를 $M$, 유체 입자 수를 $N$이라고

하면 색의 경계의 속도를 계산하는 전체 과정은 $O(MN)$의 시간복잡도를 갖는다.

**Algorithm 6** Interaction between Fluid Particles and Palette Points

```
1: procedure MOVE                          ▷ make palette points to follow fluid movement
2:     for P in POINTS do
3:         reset count, v
4:         for p in PARTICLES do
5:             d ← ||P − x_p||
6:             if d < h' then              ▷ evaluate average velocity near the palette point
7:                 count ← count + 1
8:                 v ← v + v_p
9:         v = v · 1/count
10:        P = P + vΔt                     ▷ update the point
11: end procedure
```

이렇게 계산한 평균을 해당 점의 속도로 사용하면 색의 경계가 물리적 조작에 이어서 실시간으로 부드럽게 움직이며 비로소 유체 인터페이스를 따라서 움직이는 물감 방울을 표현할 수 있게 된다.

색의 경계의 각 점끼리는 렌더링을 제외하고 서로 영향을 주지 않는다. 단, 경계에서 인접한 점 사이의 거리가 특정 길이 이상일 경우 복잡하게 변하는 모양의 처리가 힘들어지고 외부 조작에 제대로 대응하지 못하므로 해당 위치의 베지어 곡선 $\mathbf{B}(t)$에서 $t = 1/2$ 지점에 새로운 점을 추가하여 균형을 맞추게 된다. 반대로 두 인접한 점이 너무 가까워지게 되면 해당 위치에서 곡선이 매끄럽게 나오지 않을 수 있기 때문에 제거가 일어나기도 한다.

새로운 drop이 일어나면 해당 위치를 중심으로 작은 반경을 갖는 정다각형의 색의 경계를 생성한다. 3.2에서 제시한 메커니즘에 따라 원형의 자국이 생기고, drop이 일어난 직후에는 해당 위치에 추가한 유체 입자들의 영향으로 물감 방울의 크기가 서서히 커질 것이다.

# IV. 결과

다음 그림은 본 연구에서 제작한 마블링 아트 시뮬레이션이 이루어지는 과정을 요약한 다이어그램이다.

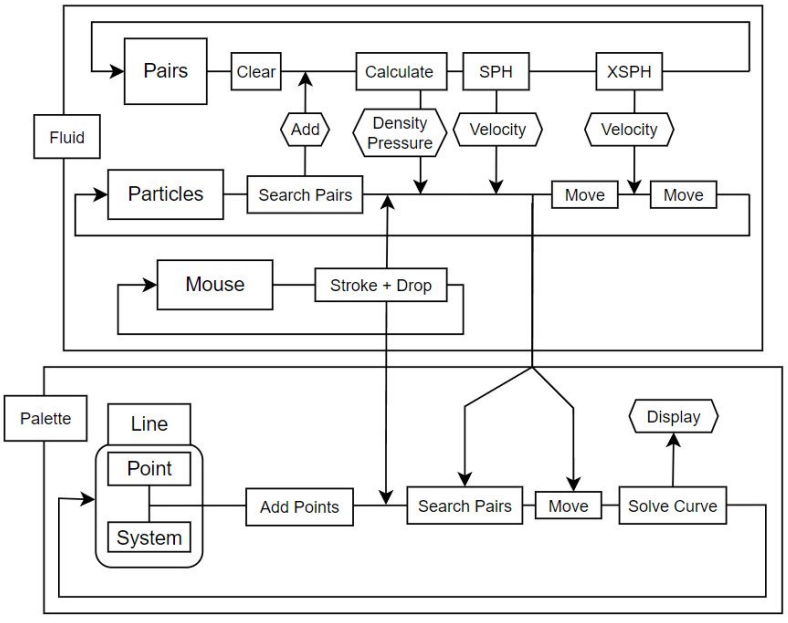
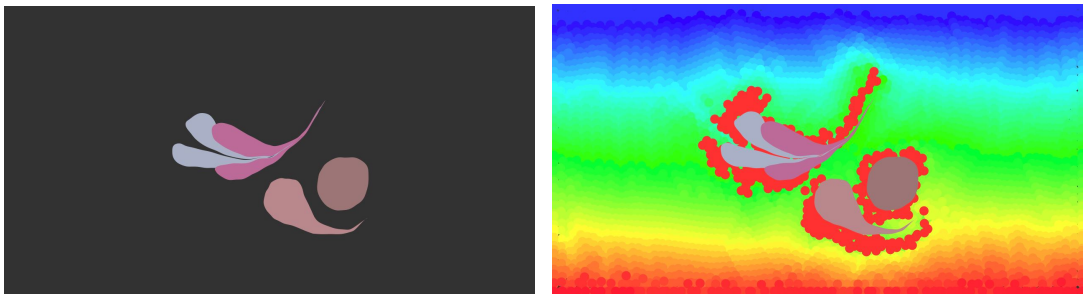


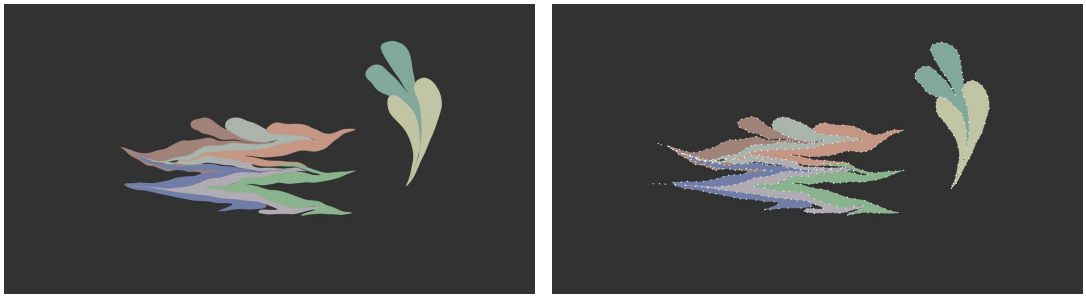

**그림 5.** 전체 과정을 나타내는 다이어그램. 앞서 유체(Fluid)의 역할에 물감 레이어(Palette)의 기능이 추가되었고, 각각 실시간으로 stroke과 drop 신호를 처리하도록 만들었다.

다음 여러 차례 그림은 실제 시뮬레이션을 통해 확인할 수 있는 결과물이다. 본 연구에서는 먼저 마우스에서 조작을 가했을 때 유체 인터페이스의 움직임과 함께 물감 레이어에 그려지는 형태의 변화가 확실히 드러나는지, 또한 물감 레이어에서 물감 방울의 형태를 여러 차례 변형했을 때 정상적으로 점들이 추가되고 매끄러운 곡선으로 잘 이어지는지 확인하였다. 또한, 여러 방울의 물감을 떨어트렸을 때 이들이 서로 뒤섞이려 하지 않고 조작의 방향을 나란히 따라갈 수 있는지 직접 시뮬레이션을 통해 확인하였다.



**(a)** **(b)**

**그림 6.** (a)는 시뮬레이션의 결과를, (b)는 유체 인터페이스의 움직임과 물감 레이어 사이의 상호작용으로 인한 변화를 나타낸다. 붉은 색의 입자들은 drop 과정으로 생성된 새 입자들이다.



**(a)** **(b)**

**그림 7.** (a)는 시뮬레이션의 결과를, (b)는 복잡한 조작 아래에서 색의 경계를 이루는 점들이 어떻게 추가되고, 곡선으로 이어졌는지를 나타낸다.

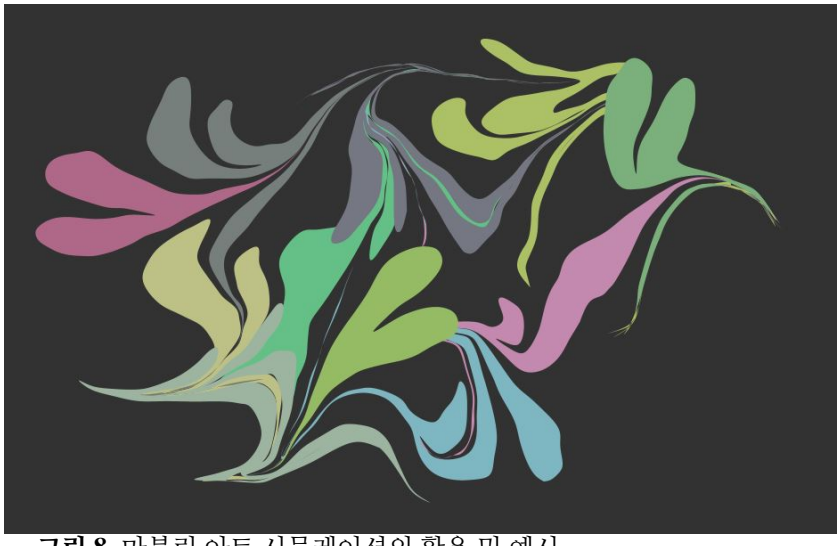


**그림 8.** 마블링 아트 시뮬레이션의 활용 및 예시.

# V. 결론

본 연구의 목표는 실시간으로 상호작용 가능한 마블링 아트 시뮬레이션을 제시하는 것이었고, 이를 크게 네 단계에 걸쳐 달성하였다. 먼저, 크게 시뮬레이션의 물리 엔진을 담당하는 유체 인터페이스를 SPH를 통해 구현하였고, 마블링 아트를 효과적으로 재현할 수 있도록 XSPH를 통해 보완하였다. 유체 인터페이스에 덧대 실제로 결과물이 보이는 모습을 결정하는 레이어인 palette를 정의하였고, 물감의 형태를 결정하는 테두리, 즉 색의 경계를 점열로 나타낸 후 베지어 곡선과 Thomas' Algorithm을 통해 매끄러운 곡선으로 변환하였다. 이는 단순히 경계를 부드럽게 뭉갠 형태가 아니라, 여러 조작으로 만들어지는 날카로운 부분까지 제대로 표현할 수 있다는 것을 확인하였다. 또한, 본 연구에서는 유체 인터페이스 및 Palette에 각각 외부에서 Stroke와 Drop을 가했을 때 어떤 변화가 일어나는지 제시하였고, 마지막으로 색의 경계의 움직임을 유체 인터페이스가 실시간으로 제어할 수 있도록 하는 상호작용을 구현하여 실제와 비슷한 결과가 나왔음을 알 수 있었다.

# 참조 문헌


[1] Cossins, Peter J. "Smoothed particle hydrodynamics." arXiv preprint arXiv:1007.1245 (2010).

[2] Jaffer, Aubrey. "Oseen Flow in Paint Marbling." arXiv preprint arXiv:1702.02106 (2017).

[3] Monaghan, Joe J. "Smoothed particle hydrodynamics." Annual review of astronomy and astrophysics 30 (1992): 543-574.

[4] Monaghan, Joseph J. "SPH compressible turbulence." Monthly Notices of the Royal Astronomical Society 335.3 (2002): 843-852.

[5] Lee, W. T. "Tridiagonal matrices: Thomas algorithm." MS6021, Scientific Computation, University of Limerick (2011).

[6] Mortenson, Michael E. Mathematics for computer graphics applications. Industrial Press Inc., (1999).

# 감사의 글

한 학기 동안 해당 졸업논문 작성을 지도 및 격려해 주신 최문성 선생님께 정말 감사드립니다.

# 연구활동

- 2020학년도 교내 기초 R&E "2차원 볼록 다각형 폐곡선 내부에서 효율적인 점광원 배치" 연구
- 2021학년도 교내 심화 R&E "강화학습을 통한 탄성체의 변형 및 이동 연속 제어" 연구