# Interactive Paper Marbling

NICK WALKER

CS354 Fall 2016

## 1 Introduction

Marbling is a traditional art practice in which acrylic inks are floated on a liquid surface, then transferred to a sheet of paper. The resulting patterns can resemble marble, but common outputs range from intricate spiraling patters to spattered, space-like designs.

Several works have modeled marbling as a Navier-Stokes fluid simulation, however, this is computationally demanding and dissipative. The resulting images are chronically blurry and fail to capture the well defined boundaries between colored areas that are characteristic of marbled patterns. In contrast, a paper from Lu et. al. describes a closed form approximate solution for the position of vertices along the boundaries of ink drops under various marbling operations [1].

In the Lu work, compositions are completely described by a sequence of marbling operations. We provide the equations that define the deformations created by each operation, describe how these operations can be rendered, and present our interactive implementation.

## 2 Operations

Each marbling operation defines a force field over the marbling surface. That is, each operation $\boldsymbol{F}$ accepts an coordinate on the marbling surface and returns a displacement vector.

$$\boldsymbol{F} : \mathbb{R}^2 \to \mathbb{R}^2$$

Operations may have parameters that control their placement of effect. We will notate these additional parameters as variables upon which the operation is conditioned. For example, here is the translation operation $\boldsymbol{T}$, given real scalar parameter $s$, which translates all points up along the y-axis by $s$.

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\boldsymbol{T}(\boldsymbol{p}|s) = \begin{bmatrix} 0 \\ s \end{bmatrix}$$

### 2.1 Ink Drop

The ink drop operation places a colored circle of radius $r$ at point $\boldsymbol{c}$. It deforms other points according to the field

$$r \in \mathbb{R}^+$$

$$\boldsymbol{O}(\boldsymbol{p}|\boldsymbol{c}, r) = -\boldsymbol{p} + \left( \boldsymbol{c} - (\boldsymbol{p} - \boldsymbol{c}) \sqrt{1 + \frac{r^2}{\|\boldsymbol{p} - \boldsymbol{c}\|^2}} \right)$$

which has the property of preserving the area of surrounding ink drops.

In addition to the field it applies, ink drop is the only operation that adds geometry to the simulation. The points it creates are subject to deformation by subsequent operations.

### 2.2 Tine Line

The line line operation simulates dragging a pointed instrument through the marbling surface. Its principle parameters are $\boldsymbol{c}$, a point on the line that will be combed through, and $\boldsymbol{d}$, the direction in which to comb. The strength of the force field is inversely proportional to the distance between $\boldsymbol{p}$ and $cb$. Parameters $\alpha$ and $\lambda$ control the magnitude and sharpness of the displacement falloff.

$$\alpha, \lambda \in \mathbb{R}^+$$

$$\boldsymbol{L}(\boldsymbol{p}|\boldsymbol{c}, \boldsymbol{d}, \alpha, \lambda) = \frac{\alpha\lambda}{d + \lambda} \hat{\boldsymbol{d}}$$

$$d = \|(\boldsymbol{p} - \boldsymbol{c})^\top \hat{\boldsymbol{c}}\|$$

## 2.3 Wavy Comb

The wavy comb operator simulates moving a comb of tines through the marbling surface in a sinusoidal pattern. $\theta$ controls the angle of the x-axis on which the curve sits, $\omega$ controls the wavelength and $A$ controls the amplitude.

$$\boldsymbol{W}(\boldsymbol{p}|\theta, A, \omega) = f\left(\boldsymbol{p}^\top \begin{bmatrix} \sin(\theta) \\ -\cos(\theta) \end{bmatrix}\right) \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}$$
$$f(v) = A\sin(\omega v)$$

## 2.4 Tine Circle

The tine circle operator simulates moving a tine in a circle with radius $r$ and center $\boldsymbol{c}$.

$$\boldsymbol{O}(\boldsymbol{p}|\boldsymbol{c}, r) = -\boldsymbol{p} + \left(\boldsymbol{c} - (\boldsymbol{p} - \boldsymbol{c})^\top \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}\right)$$
$$l = \frac{\alpha\lambda}{d + \lambda}$$
$$\theta = \frac{l}{\|\boldsymbol{p} - \boldsymbol{c}\|}$$
$$d = |\|\boldsymbol{p} - \boldsymbol{c}\| - r|$$

## 2.5 Vortex

The vortex operation simulates a tine moving in circles towards a point. It is identical to the tine circle operator except that the distance term no longer falls off with the radius.

$$d = \|\boldsymbol{p} - \boldsymbol{c}\|$$

# 3 Rendering

A sequence of operations fully describes a marbling composition. In order to render the composition, we evaluate each operation in sequence, transforming all existing points according to the operation displacement field, then creating new geometry (in the case of the ink drop operator). Drops are modeled by polygons that have a fixed density $d$ of points per unit arc-length, with $d$ typically set such that many points fall within a single pixel. The resulting paths are rasterized by drawing the them in the order in which they were deposited.

# 4 Implementation

We implemented the marbling simulation in Typescript–a language which compiles to Javascript–and leveraged the capable Canvas 2D graphics API. [1] While the operations work on all modern browsers, we have only tested our UI on the latest versions of Google Chrome.

**Algorithm 1** Render

```
Input: list of operations O, r geometry resolution
Output: bitmap image
procedure RENDER(O, r)
    drops ← []
    for o in O do
        for drop in drops do
            for p in drop.points do
                p ← p + o.displacementAtPoint(p)
            end for
        end for
        if o instance of InkDropOperation then
            drops.append o.createGeometry(r))
        end if
    end for
    image ← baseColor
    for drop in drops do
        image ← draw drop on top
    end for
end procedure
```

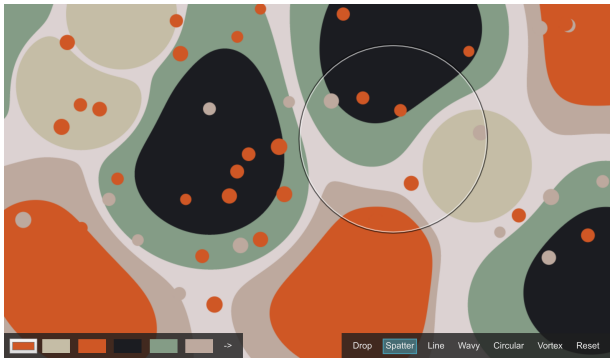


Figure 1: The interface of our application. A color panel on the bottom left provides several sets of predefined colors. The tool pane on the right provides access to the various operations.

Our tool lets users

- apply ink marbling operations with dynamic control over parameters
- preview displacement fields interactively
- view operation results almost instantly
- save compositions as PNG images

[1] All source code for our application is available at github.com/nickswalker/paper-marbling-js

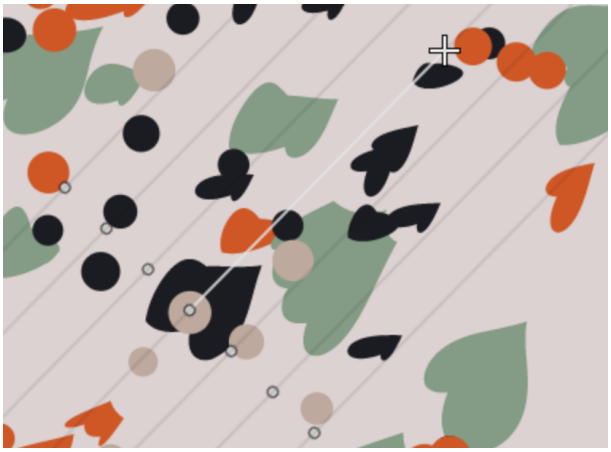

Figure 2: The cursor for the line tine operation as the user drags out the direction vector. The scroll wheel allows the user to control the number and spacing of the tines.

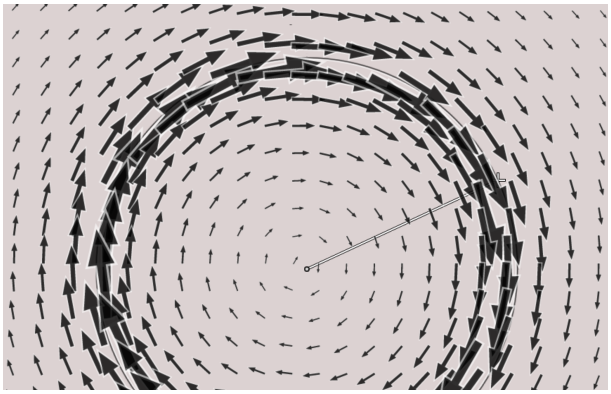

Figure 3: The displacement field preview as a user drags out the radius of a circular tine operation.

- script all operations using Javascript with an in-application script editor

## 4.1 Interactive controls

Users can control operations by clicking and dragging. Most operations accept the c parameter as their origin. We map this parameter to mouse-down coordinates on our canvas. Operations that require a direction allow the user to drag while the left mouse-button is held to create a direction vector. The length of this vector is mapped to strength related parameters of the tool. Some parameters, which the user may wish to set and persist across multiple applications of the operation, are mapped to the scroll wheel, and shift-scrolling sometimes controls a secondary parameter. We designed a flexible cursor rendering system which lets users see how their adjustments and movements are affecting the tool.

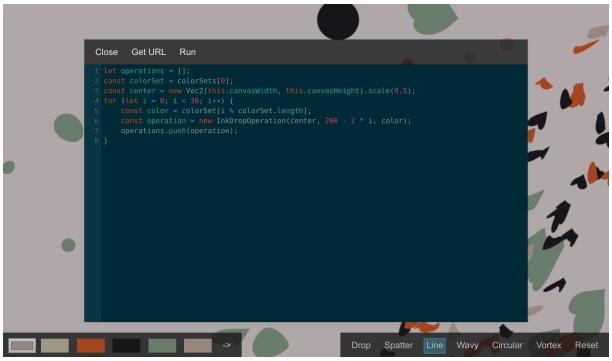


Figure 4: The script editor open above our application's main interaface. User's have access to all of the capabilities of the Javascript standard library as well as access to the marbling operation API.

A full preview of the displacement field that would be applied with the current tool parameterization can also be rendered by pressing F.

## 4.2 Script Editor

Taking advantage of Javascript's dynamic nature, we provide the same object-oriented operation API built for our use to users. Scripts can be shared as URLs, enabling easy sharing of precise compositions. The editor's syntax highlighting is provided by the popular open source CodeMirror library.

## 4.3 Limitations

Although the rendering approach we used allows for arbitrary precision, it has expense proportional to ink-drop point density. As more drops are added, the rendering process slows significantly, and the user will notice a delay between releasing the mouse and seeing the results of their operation. A raster rendering approach, in which operations are applied to pixels and not points along ink-drop boundaries would enable GPU acceleration and frame rates sufficient to provide intra-operation animation.

Because of the single threaded nature of Javascript virtual machines, long rendering times impact UI responsiveness. Support for background threads is becoming more widespread, however it brings additional implementation complexity.

A fundamental limitation of this marbling approach is its simplicity. As opposed to more sophisticated fluid simulation approaches, it cannot reproduce highly dynamic behavior seen in some types of marbling.

Figure 5: A pattern created with a script that placed concentric circles, then ran circular tines emanating from the left and right of the canvas.

Figure 6: A mix of randomly placed drops and vortices.

Figure 7: Alternating horizontal and vertical tines create familiar floral patterns

## 5 Conclusion and Future Work

The provided tools enable a range of traditional marbling patterns, but they cannot easily express freeform tine patterns. Future work should explore enabling freehand operations. Lu and collaborators have recently published work extending the 2D marbling operations to 3D. An extension of this project may attempt to implement this work.

## References

[1] Shufang Lu, A. Jaffer, Xiaogang Jin, Hanli Zhao, and Xiaoyang Mao. Mathematical marbling. *IEEE Computer Graphics and Applications*, 32(6):26–35, 2012.