Mathematical Marbling

Shufang Lu ■ *Zhejiang University*

Aubrey Jaffer ■ *Adnetik*

Xiaogang Jin ■ *Zhejiang University*

Hanli Zhao ■ *Wenzhou University*

Xiaoyang Mao ■ *University of Yamanashi*

Instead of using fluid dynamics to simulate marbling, the proposed method takes a mathematical approach with closed-form expressions. This method improves control, ease of implementation, parallelism, and speed, enabling real-time visual feedback and creation of vivid flowing animations. Users can start designs from a blank sheet, raster images, or videos.

Marbling creates stone-like or intricate abstract decorations from inks (or paint) floating on water or gel. It's a decorative art with several distinct traditions originating in Asia, perhaps as many as 1,000 years ago. It spread to Europe in the 16th century, where its primary application was producing book covers and endpapers. Mechanized bookbinding caused the decline of marbling in the West, but it has enjoyed a revival as a folk art since the 1970s. Although primarily used for decoration, marbling has security applications. Marbling ledger book edges makes missing pages apparent, and documents written over pale marbling are tamper-resistant.

Digital simulations based on complex physical models have been commonly used to create marbling images.[1,2] However, these methods produce blurry contours because the time-iterative-relaxation nature of the solver makes dissipation inevitable. The more marbling operations are applied, the blurrier the result is. So, these methods have difficulty producing publication-quality images because fine features will be lost. (For more on digital marbling methods, see the related sidebar.) This motivates us to find simple closed-form mathematical formulas to simulate marbling.

Here, we present deformation formulas for simulating marbling, while avoiding the computational cost of full fluid simulation. This approximation is rich enough to capture many phenomena, and it solves the dissipation problem and ensures the resulting images' sharp contours. Besides simplicity, using mathematical formulas provides advantages for control, speed, implementation ease, parallelism, and vector output. It enables the generation of beautiful designs with real-time visual feedback and progressive fluid-like illustration of marbling.

Our Method

Our mathematical treatment of marbling starts with the assumptions of incompressible and immiscible 2D fluid inks. Our tool function formulas are based on topological computer graphics,[3] which generates marbling designs with sharp contours and vector marbling output.

Tool Functions

Our method mainly supports five types of patterning tools, which we'll describe with forward-transform formulas. These tools are based on those frequently used for traditional marbling. (For a look at the traditional marbling process, see the related sidebar.) Proving that these tool functions are incompressible is easy.

A key feature of these marbling transforms is that the displacement parallel to a line depends only on the perpendicular distance from the line. Because of this, the backward transform is simply the forward transform with its displacement negated. So, our tool functions can be applied forward and in the inverse. Forward application

 Published by the IEEE Computer Society

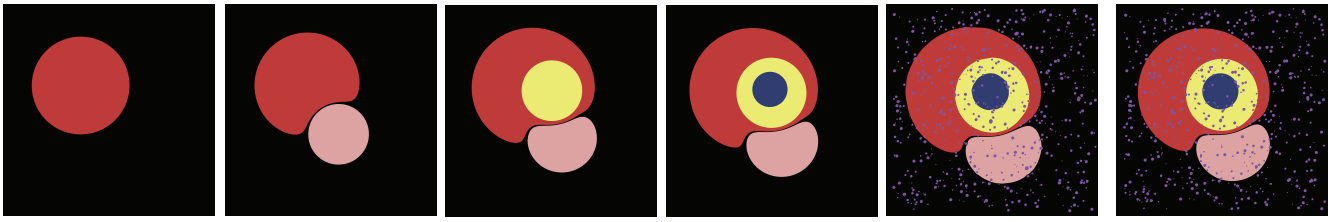

Figure 1. Placing ink drops (a) with deformation and (b) without deformation. With deformation, each drop will be distorted by all the subsequent drops. Without deformation, if a droplet overlaps with an existing one, the newer one overwrites the overlapped part.

generates vector-based output; inverse application generates raster output. In both cases, the closed-form expressions preserve the design's quality and sharpness. Although our method doesn't directly support a free-hand curve interaction tool, we can simulate its effect by combining existing tools.

To create patterns, our method uses the tool functions to deform the current pattern. As users work with the tools, our method implements raster image warping on the GPU to achieve real-time immersive feedback.

Specifically, to determine the color at a point $\mathbf{P}$ in the current pattern, our method first performs backward image-mapping to trace the trajectory for $\mathbf{P}'$ in the previous pattern. It then samples the color at point $\mathbf{P}'$ to point $\mathbf{P}$.

Ink drops. When an artisan (called a marbler) applies ink drops to the liquid substrate, those drops deform any previous drops. Also, placing a drop within a previous drop will force the previous one to spread.

Simulating this process faithfully with physics-based methods is difficult. But a mathematical treatment produces a simple, exact solution. The first drop forms a circular spot with area a. If a second drop with area b is put in the center of the first drop, the total covered area increases from a to $a + b$. Points at the center move from radius 0 to radius $\sqrt{b/\pi}$, and boundary points move from radius $\sqrt{a/\pi}$ to $\sqrt{(a+b)/\pi}$. So, this transformation is incompressible in that the area of ink regions that didn't receive the drop doesn't change.

Given $\mathbf{P}$ and an ink drop centered at $\mathbf{C}$ with radius r, if $|\mathbf{P}-\mathbf{C}| < r$, $\mathbf{P}$ is within the drop and takes its color. Otherwise, $\mathbf{P}$ is displaced radially from $\mathbf{C}$:

$$\mathbf{P}' = \mathbf{C} + (\mathbf{P}-\mathbf{C})\sqrt{1+\frac{r^2}{|\mathbf{P}-\mathbf{C}|^2}}\,.$$

According to the transform function we just gave, the last drop is rendered last and has a round outline. However, that drop distorts the second-to-last drop. Working backward through the ink droppings, each drop will be distorted by all the subsequent drops (see Figure 1a).

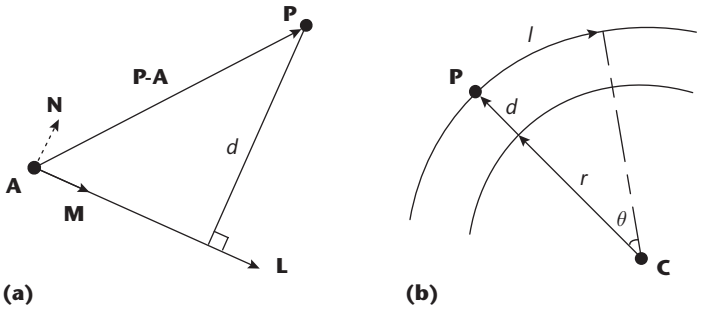

Figure 2. Tool functions for (a) single and (b) circular tine lines. Figures 3b through 3d show an example of a single tine-line pattern; Figure 4c shows a circular tine-line pattern.

Sprays of small droplets cause little distortion to underlying inks, yet require as much computation as if they were larger. So, we provide another function with no deformation. If a droplet overlaps with an existing one, the newer one overwrites the overlapped part (see Figure 1b).

Straight tine-line patterns. This function runs tine lines through a marbling in any direction. (A tine is a tooth of a virtual comb used to create marblings.) Consider the pattern transformation induced by drawing with a single tine straight from one side of the marbling tank to the other. Points near a tine's trajectory move in the direction of the tine's motion. The amount of motion in the tine's direction is (roughly) inversely proportional to the point's distance from the tine's trajectory.

This inversely proportional function provides many possibilities. The function we choose parameterizes both the maximum shift and the shift gradient's sharpness. Because the point's motion is perpendicular to the shortest distance between the point and the line, this displacement and its inverse are equally easy to calculate. So, each cohort of points at a distance from the tine line shifts by the same amount. On an infinite plane, these parallel shifts neither compress nor expand the inks.

Related Work in Marbling Simulation

Early research on marbling simulation favored numerical simulation. Physics-based simulations view marbling as a 2D computational fluid dynamics (CFD) problem and try to numerically solve complicated Navier-Stokes (NS) equations.[1] B. Tevfik Akgun developed a computer-aided paper-marbling tool for generating traditional Turkish art forms.[2] To obtain the effect of fluid fluctuations at different levels, Rüyam Acar and Pierre Boulanger introduced a multiscale fluid model and a sharp fluid boundary method to simulate marbling.[3] Although their method can model highly turbulent marbling effects, it doesn't deal with some traditional marbling patterns such as comb patterns. Acar proposed a level-set-driven method that provides a flexible environment to model a range of flows and artistic effects in 2D, and applied it to marbling.[4] All these methods were implemented on a CPU. They don't provide real-time feedback because they must solve the time-consuming physics equations.

Xiaogang Jin and his colleagues presented a novel digital marbling framework by solving NS equations on the GPU.[5] But it suffered from blurry ink interfaces due to energy dissipation. To combat the dissipation, Jiayi Xu and her colleagues employed complex high-order advection techniques.[6] However, this method might cause instability. To reduce dissipation while retaining stability, Hanli Zhao and his colleagues employed an accurate yet fast third-order unsplit semi-Lagragian constrained interpolation profile method.[7]

Marbling images generated using CFD models have three limitations. The first is the dissipation we just mentioned. Researchers often apply complicated computational mechanisms to combat this effect.[8] Dissipation can lead to blurring or mixing of colors, which doesn't conform to the sharp features of real-world marbling. Dissipation is inevitable in physics-based methods. Even higher-order advection techniques such as fifth-order B-splines can't eliminate it.[3]

The second limitation is speed. Although physics-based simulations have motivated many papers proposing fast solvers for various scenarios, generating megapixel-sized images with real-time feedback is still a challenge.

The third limitation is control. Prior methods usually have many physical parameters (for example, viscosity of liquids and force of manipulation) whose effects on marbling aren't obvious.

Corel Painter supports a powerful image-editing function that lets users interactively create several traditional marbling effects.[9] The program considers marbling as a distortion in which the user drags colors with a virtual comb and mixes them. These marbling effects are limited. Corel Painter supports only wavy path combing in horizontal and vertical directions; it can't reproduce swirling patterns. Moreover, the process is static, and users receive no progressive feedback.

Although you can simulate fluid-like turbulent motions by numerically solving the fluid equations, a closed-form solution would be desirable. Ken Perlin introduced a procedural turbulence noise function (called *Perlin noise*) to generate various 2D and 3D textures.[10] Owing to Perlin noise's efficiency and simplicity, it's frequently used instead of physics-based methods to simulate fluid-like animations. However, because Perlin noise functions have nonzero divergence, they can't simulate incompressible fluids. Karl Sims used linear superposition of flow primitives to modulate velocity vector fields and warp 2D

In Figure 2a, if **L** is a tine line with arbitrary slope, **N** is a unit vector perpendicular to **L**, **A** is a point on **L**, and **M** is the unit vector in the direction of **L**, the mapping for **P** is

$$\mathbf{P}' = \mathbf{P} + \frac{\alpha\lambda}{d+\lambda}\mathbf{M}, \tag{1}$$

where $d = \left|(\mathbf{P}-\mathbf{A})\cdot\mathbf{N}\right|$ is the displacement function representing the distance from **P** to **L**, and the scalars α and λ control the maximum shift and sharpness of the shift gradient.

For the input image in Figure 3a, Figures 3b through 3d show the results of the same tine-line operation for three values of (α, λ). Our method's results are similar to those of Jiayi Xu and her colleagues' method (see Figure 3e).[2]

Because the composition of homeomorphisms is a homeomorphism, we can compose the mapping of multiple parallel tine-line strokes into a single function. This composite function can comb in any direction (see Figure 4a). Users can modify it to form many traditional designs.

For evenly spaced multiple tines that move as a rigid assembly, we define a function representing the displacement from the set of parallel tines:

$$d' = s/2 - \left|\mathrm{fmod}(d,s) - s/2\right|,$$

where d is the distance from **P** to an arbitrary tine line and s is the spacing between tines. In this way, we compute the mapping function by replacing d in Equation 1 with d'. The new function produces similar results with less computational cost.

Wavy patterns. Curved tine trajectories contribute much to marbled images' beauty and charm.

images to create the visual effect of flow.[11] But neither of these methods is tailored for progressive, interactive marbling.

Vector graphics, which defines primitives geometrically, is an alternative representation of images. It can produce high-quality images because the vector graphics representation is resolution-independent.[12] Vector images are used for rendering high-quality surface details on 3D objects even at high zoom levels.[13,14] Lvdi Wang and his colleagues introduced an effective vector representation for solid textures and mapped them onto mesh surfaces in real time.[15] Ryoichi Ando and Reiji Tsuruno presented a framework for generating marbled images that can be exported into a vector graphics format.[16] This framework allows arbitrary mouse-controlled movement of a virtual tine. However, the underlying fluid motion requires solving complicated, time-consuming physics equations. In the main article, we focus on the more intuitive mathematical marbling while supporting the output of vector geometries and high-quality surface details for rendering on 3D objects.

References

1. J. Stam, "Stable Fluids," *Proc. Siggraph*, ACM, 1999, pp. 121–128.
2. B. Akgun, "The Digital Art of Marbled Paper," *Leonardo*, vol. 37, no. 1, 2004, pp. 49–52.
3. R. Acar and P. Boulanger, "Digital Marbling: A Multiscale Fluid Model," *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 4, 2006, pp. 600–614.
4. R. Acar, "Level Set Driven Flows," *ACM Trans. Graphics*, vol. 26, no. 4, 2007, article 15.
5. X. Jin, S. Chen, and X. Mao, "Computer-Generated Marbling Textures: A GPU-Based Design System," *IEEE Computer Graphics and Applications*, vol. 27, no. 2, 2007, pp. 78–84.
6. J. Xu, X. Mao, and X. Jin, "Nondissipative Marbling," *IEEE Computer Graphics and Applications*, vol. 28, no. 2, 2008, pp. 35–43.
7. H. Zhao et al., "AtelierM++: A Fast and Accurate Marbling System," *Multimedia Tools and Applications*, vol. 44, no. 2, 2009, pp. 187–203.
8. B. Kim et al., "Advections with Significantly Reduced Dissipation of Diffusion," *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 1, 2007, pp. 135–144.
9. R. Grossman, *Digital Painting Fundamentals with Corel Painter 11*, Course Technology PTR, 2009.
10. K. Perlin, "An Image Synthesizer," *Computer Graphics*, vol. 19, no. 3, 1985, pp. 287–296.
11. K. Sims, "Choreographed Image Flow," *J. Visualization and Computer Animation*, vol. 3, no. 1, 1992, pp. 31–43.
12. A. Orzan et al., "Diffusion Curves: A Vector Representation for Smooth-Shaded Images," *ACM Trans. Graphics*, vol. 27, no. 3, 2008, article 92.
13. D. Nehab and H. Hoppe, "Random-Access Rendering of General Vector Graphics," *ACM Trans. Graphics*, vol. 27, no. 5, 2008, article 135.
14. S. Jeschke, D. Cline, and P. Wonka, "Rendering Surface Details with Diffusion Curves," *ACM Trans. Graphics*, vol. 28, no. 5, 2009, article 117.
15. L. Wang et al., "Vector Solid Textures," *ACM Trans. Graphics*, vol. 29, no. 4, 2010, article 86.
16. R. Ando and R. Tsuruno, "Vector Graphics Depicting Marbling Flow," *Computers and Graphics*, vol. 35, no. 1, 2011, pp. 148–159.

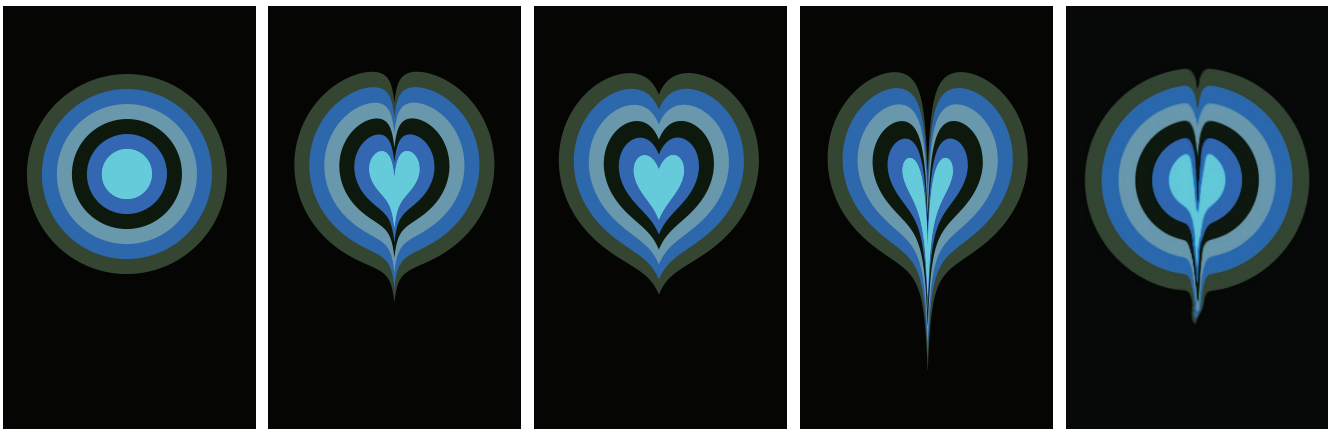

Figure 3. Tine-line pattern examples. (a) The original input. (b) The results of our method for (α, λ) = (80, 8), where α and λ are scalars controlling the maximum shift and the shift gradient's sharpness. (c) Our results for (80, 32). (d) Our results for (240, 8). (e) The tine-line geometry motion made by Jiayi Xu and her colleagues' physics-based method.[2] Our tine-line pattern function provides simulation effects similar to that method.

Traditional Marbling

The traditional marbling process consists of three steps.[1] First, the artisan (called a marbler) places background liquid in a tray and sprinkles or drops the inks onto the liquid surface with eyedroppers or brushes to create an initial design. The liquid layer must be thick enough to keep the inks floating on its surface.

Then, the marbler uses styluses, combs (also called rakes), and other tools to change the initial pattern. Arranging the comb's tines with different spacings creates different effects. As the marbler runs the tools back and forth across the tray, a complex design emerges. An intricate pattern usually requires several strokes.

After completing the pattern, the marbler gently applies a sheet of paper, fabric, or some other material onto the tray. The pattern created on the surface transfers to the contact material.

The patterning tools and how they're manipulated are crucial to producing impressive features. Many modern marblers search for new effects or techniques to heighten their expressivity. Unfortunately, this task isn't easy because marbling consists of several steps, and the marbler must start from scratch if he or she makes a mistake.

Reference

1. D. Maurer-Mathison, *The Ultimate Marbling Handbook: A Guide to Basic and Advanced Techniques for Marbling Paper and Fabric*, Watson-Guptill, 1999.

This function generates wavy patterns in any direction. In our model, the wavy path is a sinusoidal displacement (versus distance) applied after a (straight) comb operation. The results look like the deformation from a comb stroked along a wavy path (see Figure 4b).

Let $f(v) = A \cdot \sin(\omega v + \psi)$ be the displacement function. By interactively specifying the amplitude A, the wavelength ω, and the phase ψ, users can move a comb in various wavy paths. This operation maps $\mathbf{P}$ to $\mathbf{P}'$ in the direction at angle t:

$$\mathbf{P}' = \mathbf{P} + f\left(\mathbf{P} \cdot [\sin t, -\cos t]\right)[\cos t, \sin t]\,.$$

Circular patterns. This function moves tines in circular trajectories controlled by the radius r and center $\mathbf{C}$ of the swirl. Our method treats a circular motion similarly to a straight one. A circle is perpendicular to every radius ray at their intersection. For a given $\mathbf{C}$, motion along the arc containing $\mathbf{P}$ is inversely proportional to the minimum radial distance from $\mathbf{P}$ to the tine circle. The concentric circles, which are each rotated by different amounts, neither compress nor expand the inks. The scalars α and λ play the same role they do in the linear transformation.

As Figure 2b shows, this operation displaces points along arcs around $\mathbf{C}$. It maps $\mathbf{P}$ to

$$\mathbf{P}' = \mathbf{C} + (\mathbf{P} - \mathbf{C})\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}, \qquad (2)$$

where its angle subtended at $\mathbf{C}$ is $\theta = l/(|\mathbf{P} - \mathbf{C}|)$, the displacement arc's length is $l = \alpha\lambda/(d + \lambda)$, and $d = |\mathbf{P} - \mathbf{C}| - r|$.

The circular pattern's direction depends on whether θ is positive or negative. Positive θ generates a clockwise pattern, and vice versa. Figure 4c shows a circular tine-line pattern.

Vortex patterns. Vortices, which wind more as they near the center, are popular in marbling. We can obtain vortex patterns using the same mapping

(a) (b) (c)

Figure 4. Images made with our method. (a) A pattern made by a single virtual comb tine. (b) A wavy pattern. (c) A circular tine-line pattern. These computer-generated patterns are similar to traditional manually generated ones.

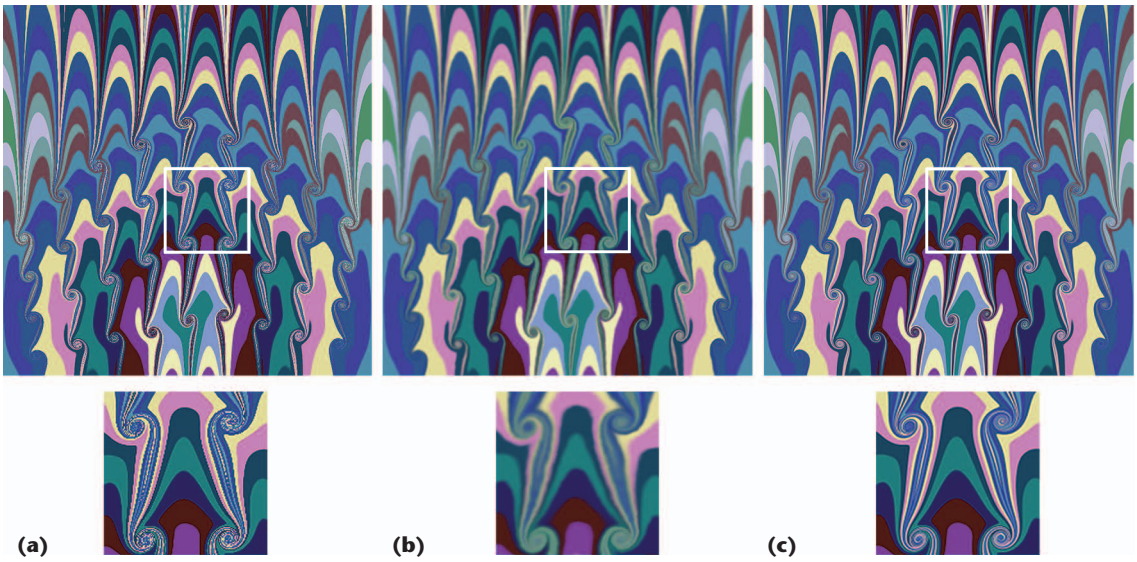

Figure 5. Vortex patterns' image quality. (a) Our method usually produces aliasing artifacts. (b) Updating an image and using the update for the next operation can cause blurring. (c) Our alternative updating technique produces sharp contours. The bottom images show details.

function as Equation 2, with one change. Here, we set $d = |\mathbf{P} - \mathbf{C}|$, where $\mathbf{C}$ denotes the vortex's center (see Figure 5).

Preserving Sharp Contours in Raster Images

Sharp contours between the inks are crucial characteristics of real-world marbling images. We employ two techniques to preserve sharp contours for marbling raster images.

Because our method is based on image deformation, it usually generates aliasing artifacts (see Figure 5a). Generally, antialiasing computes images at a higher resolution and then downsamples them. We employ 2 × 2 rotated-grid supersampling (RGSS) antialiasing because of its low cost and high quality. Our method also provides other common supersampling patterns such as 4 × 4 grid, Quincunx, 4 × 4 checker, and 8-rooks.[4]

If we update the image for each operation and use it as input for the next operation, contrast fading occurs, and the resulting image is blurred after some operations. For n operations, the color at $\mathbf{P}_n$ is

$$C(\mathbf{P}_n) \leftarrow C(\mathbf{P}_{n-1}) \leftarrow \ldots \leftarrow C(\mathbf{P}_1) \leftarrow C(\mathbf{P}_0),$$

where $\mathbf{P}_i (i = 0, 1, \ldots, n - 1)$ is the back-traced point of $\mathbf{P}_{i+1}$, i is the number of image deformation operations, and $\leftarrow$ represents that $C(\mathbf{P}_i)$ is obtained by sampling the color at point $\mathbf{P}_{i-1}$. Signal diffusion is inevitable unless we apply an ideal sinc filter. As n increases, the results will get increasingly blurry (see Figure 5b).

To solve this problem, we employ an alternative image-updating technique similar to that of Karl Sims.[5] For each point in the current image, we trace its mapping point in the ink-drop pattern directly and sample the color at that point to the current position:

$$C(\mathbf{P}_n) \leftarrow C(\mathbf{P}_n \rightarrow \mathbf{P}_{n-1} \rightarrow \ldots \rightarrow \mathbf{P}_1 \rightarrow \mathbf{P}_0).$$

Consequently, we can compute the composition operation in one shader; Figure 5c shows the improved results. This technique can also show an animation (evolution) of the marbling process. However, unlike the first technique, this one's performance decreases gradually as the number of operations increases. So, our method provides both techniques.

Vector Image Output

Our method supports compact resolution-independent vector output when users create the initial pattern from ink drops. We approximate each initial circular drop by an inscribed regular n-gon filled with the drop's color. We choose the value of n according to the drop's radius. We transform the initial drop's polygon points to new positions according to the composite formula derived from the marbling operations. Each point's displacement depends on the marbling operations. Two adjacent points in an initial drop might be far apart after the marbling, so undersampling artifacts might occur.

To reduce the artifacts, we employ adaptive refinement to keep the boundary smooth. As long as the distance between two transformed adjacent polygon points is larger than a user-specified threshold T, we insert a new sampling point in the

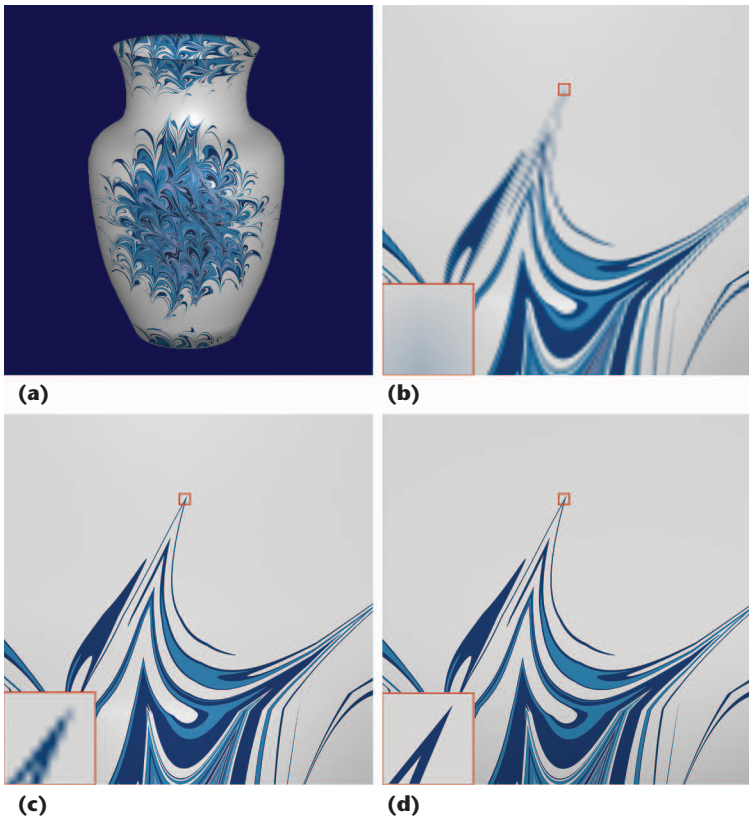

Figure 6. Rendering high-quality surface details on 3D objects. (a) We rendered this vase with a marbling pattern. (b) Traditional texture mapping with an 800 × 800 texture shows blurry edges in a close-up. (c) Even an 8,000 × 8,000 texture doesn't provide sharp edges for extreme close-ups. (d) Our surface-detail rendering guarantees crisp edges even for extreme close-ups.

```
tc ← T(x)
p ← tc * (w, h)
for all operations(f) do
  p ← f(p)
end for
for all inkdrops(col, r, c) do
  if ||p − c|| ≤ r then
    return col
  end if
end for
return bc
```

Figure 7. Vector marbling texture fetch. This algorithm lets our method render high-quality surface details on 3D objects. For an explanation of the symbols, see the main article.

middle of the arc subtended by the two points in the initial drop. Empirically, we can achieve pleasing results if we set T at 1 pixel. We can use a smaller threshold for higher quality or when magnifying the marbling image.

Rendering Surface Details on 3D Objects

Given a parameterized 3D object (such as the vase in Figure 6a), simply mapping the marbling image onto the 3D surface as a texture doesn't retain sharp texture features. Figure 6b demonstrates this effect with a texture resolution of 800 × 800. Even a high-resolution texture (8,000 × 8,000 pixels) can't faithfully reproduce the crisp boundaries for close-ups (see Figure 6c).

Our closed-form solution makes it possible to render high-quality surface details on 3D objects.[6,7] As the user designs a texture, our method records its creation history (including parameters for ink drops and operations). Then, it computes the texture for 3D objects on the fly using two steps.

First, it renders the 3D object using deferred shading to store per-pixel attributes (positions, normals, and texture coordinates) into local video memory (a G-buffer).

Second, it renders screen-aligned quadrilaterals by retrieving the per-pixel attributes stored in the G-buffer. To calculate the color of a pixel at position $\boldsymbol{x}$, the pixel shader must access these data structures:

- the buffer T used to store texture coordinates ($\boldsymbol{tc}$) in the first step,
- the patterning operations f,
- the ink drops ($\boldsymbol{col}$, r, $\boldsymbol{c}$) (these parameters represent the color, radius, and center),
- the background color $\boldsymbol{bc}$, and
- the marbling image's width w and height h.

Figure 7 shows the pseudocode; $\boldsymbol{p}$ is the point whose position is obtained by multiplying $\boldsymbol{tc}$ by (w, h).

We implement the rendering on the GPU to obtain high performance. Such a method keeps contours sharp at high zoom levels at interactive frame rates and overcomes the limited resolution while reducing texture memory usage.

For the antialias pass, we use a technique similar to that in the section "Preserving Sharp Contours in Raster Images." Figure 6d shows the results.

Using Our Method

Figure 8 shows an example of generating a digital marbling starting with a blank sheet. After making the ink-drop pattern (see Figure 8a), the user guided a comb along a straight line from right to left (see Figure 8b). The user then guided the comb back in the opposite direction, with the teeth passing in between where they passed before (see Figure 8c). The previous two steps were repeated vertically (see Figure 8d). Then, the user performed two horizontal combings in opposite directions (see Figure 8e). Finally, the user applied a wavy pattern function (see Figure 8f).

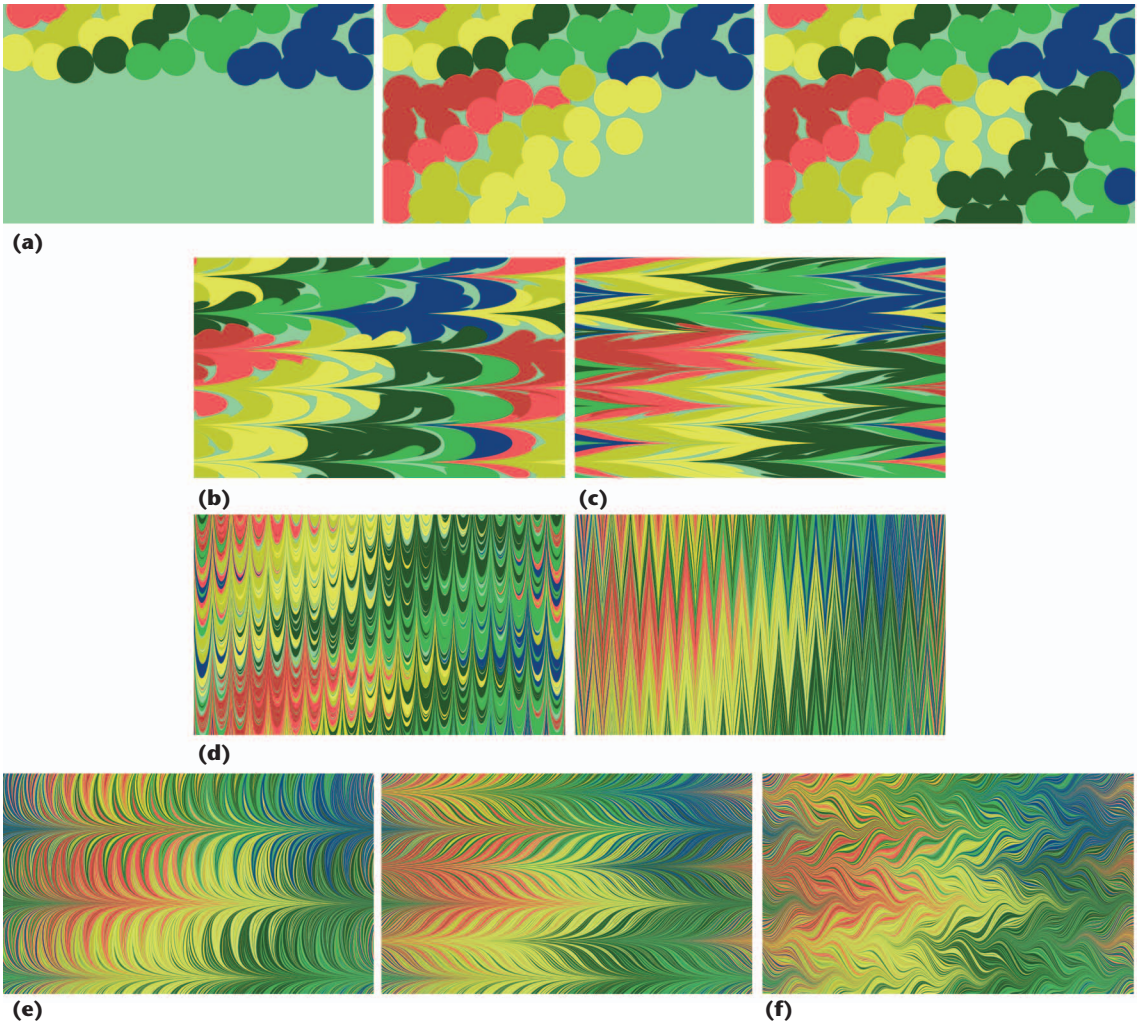

Figure 8. Generating digital marbling. (a) Making the initial ink-drop pattern. (b) Guiding the comb along a straight line from right to left. (c) Guiding the comb back in the opposite direction. (d) Repeating the previous two steps orthogonally. (e) Making two horizontal combings in opposite directions. (f) Applying a wavy pattern function.

Throughout this process, users can choose either technique we described in the section "Preserving Sharp Contours in Raster Images." They can create the resolution-independent vector images at an intermediate stage, with some delay. They can then zoom in on regions to see the fine detail at any scale.

Performance and Comparison

We implemented our method using High Level Shading Language shaders on the GPU. To investigate our method's performance, we ran it on a 1.83-GHz Intel Core 2 Duo E6320 CPU and an Nvidia GeForce 8800 GTS GPU.

Table 1 compares our method's performance with that of Xu and her colleagues' method. With a 2 × 2 RGSS filter, our method outperformed the other one by an order of magnitude. The other method's interactivity is limited to low resolutions owing to the

Table 1. The performance (in fps) of our method and Jiayi Xu and her colleagues' method[2] in the same environment.

Resolution (pixels)	Xu and her colleagues' method	Our method			
		No antialiasing	2 × 2 rotated-grid supersampling	4 × 4 checker	4 × 4 grid
512 × 512	30	1,560	732	250	239
1,024 × 512	15	1,265	375	125	120
1,024 × 768	10	850	246	83	80
1,024 × 1,024	12	660	189	64	61
1,280 × 1,024	7	548	156	55	53

(a) (b) (c)

Figure 9. A comparison of marbling images. (a) A real marbling pattern.[8] (b) The results with our method. (c) The results with Rüyam Acar's level-set method.[1] Our method can produce marbling effects comparable to real artwork. (Sources: Figure 9a, Diane Maurer-Mathison; Figure 9c, Rüyam Acar; used with permission.)

time-consuming physics simulation. Physics-based methods, including Rüyam Acar's CPU-implemented level-set method,[1] can't achieve real-time performance for high-resolution marbling images.

We also tested our method on a ThinkPad X61 laptop with an integrated Intel GMA X3100 graphics card. The method still achieved 13 fps when creating 1-Mpixel images using the 2 × 2 RGSS filter. So, our method not only lets users design images with real-time visual feedback, it also lets them experience the marbling process immersively because of its vivid fluid-like animations. Besides of our mathematical model's faster execution, it can produce vector images (see Figure 3b), whereas Xu and her colleagues' method (see Figure 3e) creates raster patterns.

Figure 9 compares a real marbling pattern with our results and those of Acar's physics-based method. The two computer-based methods generate visually similar patterns under the same manipulation steps.

Applications

We've applied our method to image editing and video-stream processing. Thanks to the high processing speed, we can process video streams in real time. Actually, every video frame is a marbling pattern, and the interesting transformation of the related video images could be used in filmmaking and advertising. For an example, see the supplementary video at http://youtu.be/ZgVblaKhC_4.

We've also applied our method to scene decoration (see Figure 10).

Our real-time design process running on stock hardware is engaging and easy to learn and use. Amateurs can produce beautiful designs in a few minutes. Artists can use our method to try different designs before they create works with real marbling materials.

We'll explore more mathematical functions such as free-hand tools to achieve a wider variety of effects. For the vector graphics output, the number of vertices grows quickly with the number of operations. We could address this issue by taking shape simplification into account. Alternatively, the current polygonal approximation to a deformed ink-drop boundary curve could be subdivided at the instant a tine crosses it. We also plan to investigate marbling solid textures by extending our focus to three dimensions, inspired by Lvdi Wang and his colleagues' research.[9]

Acknowledgments

We thank Diane Maurer-Mathison (www.dianemaurer.com) for permission to use the marbling texture in Figure 9a. Xiaogang Jin received support from the China 973 program (grant 2009CB320801), the National Natural Science Foundation of China—Microsoft Research Asia Joint Funding (grant 60970159), Zhejiang Provincial Natural Science Foundation of China (grant Z1110154), and the National Natural Science Foundation of China (grants 60933007 and 60833007). Hanli Zhao received support from the National Natural Science Foundation of China (grant 61100146) and Zhejiang Provincial Natural Science Foundation of China (grant Y1110004).

References

1. R. Acar, "Level Set Driven Flows," *ACM Trans. Graphics*, vol. 26, no. 4, 2007, article 15.
2. J. Xu, X. Mao, and X. Jin, "Nondissipative Marbling," *IEEE Computer Graphics and Applications*, vol. 28, no.

(a) (b) (c)

Figure 10. Applying our marbling to scene decorations. (a) A vase, book, and tablecloth. (b) A window design. (c) Wallpaper, upholstery, and carpeting. We rendered these 3D scenes with ray tracing in 3D Studio Max.

2, 2008, pp. 35–43.

3. A. Jaffer, "Ink Marbling," 2011; http://people.csail.mit.edu/jaffer/Marbling.
4. T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering*, 3rd ed., AK Peters, 2008.
5. K. Sims, "Choreographed Image Flow," *J. Visualization and Computer Animation*, vol. 3, no. 1, 1992, pp. 31–43.
6. D. Nehab and H. Hoppe, "Random-Access Rendering of General Vector Graphics," *ACM Trans. Graphics*, vol. 27, no. 5, 2008, article 135.
7. S. Jeschke, D. Cline, and P. Wonka, "Rendering Surface Details with Diffusion Curves," *ACM Trans. Graphics*, vol. 28, no. 5, 2009, article 117.
8. D. Maurer-Mathison, *The Ultimate Marbling Handbook: A Guide to Basic and Advanced Techniques for Marbling Paper and Fabric*, Watson-Guptill, 1999.
9. L. Wang et al., "Vector Solid Textures," *ACM Trans. Graphics*, vol. 29, no. 4, 2010, article 86.

Shufang Lu *is a PhD candidate at the State Key Laboratory of CAD & CG at Zhejiang University. Her research interests include marbling simulation, nonphotorealistic rendering, and image processing. Lu received her BSc in software engineering from Wuhan University. Contact her at lushufang@cad.zju.edu.cn.*

Aubrey Jaffer *is a mathematician creating analyses and visualizations at Adnetik. His research interests include convection, radiative transfer, numerical analysis, algebraic geometry, symbolic algebra, and space-filling curves. Jaffer has a BS in mathematics from MIT. Contact him at agj@alum.mit.edu.*

Xiaogang Jin *is a professor at the State Key Laboratory of CAD & CG at Zhejiang University. His research interests include implicit-surface computing, cloth animation, crowd and group animation, texture synthesis, and digital geometry processing. Jin has a PhD in applied mathematics from Zhejiang University. Contact him at jin@cad.zju.edu.cn.*

Hanli Zhao *is an associate professor at Wenzhou University's Intelligent Information Systems Institute. His research interests include nonphotorealistic rendering and general-purpose GPU computing. Zhao has a PhD in computer science from Zhejiang University. Contact him at hanlizhao@gmail.com.*

Xiaoyang Mao *is a professor at the University of Yamanashi. Her research interests include nonphotorealistic rendering, texture synthesis, perception, and affect-based rendering and visualization. Mao has a PhD in computer science from the University of Tokyo. Contact her at mao@yamanashi.ac.jp.*