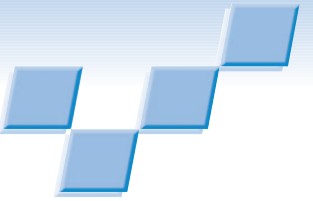
Computer-Generated Marbling Textures: A GPU-Based Design System



Xiaogang Jin and Shaochun Chen
Zhejiang University

Xiaoyang Mao
University of Yamanashi

A computer system for interactively creating marbling textures is built on the physical model of the traditional marbling process. The approach generates marbling designs as the result of color advection in the 2D flow fields obtained by numerically solving the Navier-Stokes equations on the GPU with a multigrid solver.

Marbling is a traditional art that people all over the world have loved for centuries. Started in either Turkey or Persia in the 12th century for decorative purposes, the craft migrated to Europe in the 16th century, where it became an essential part of bookbinding. Marbled papers' intricate patterns decorate the inside covers of fine books; cover the folds, strings, and glue marks of the bindings; and serve as an aesthetic transition from the dark leather covers to the white pages inside. Used as book edges, they prevent erasure and forgery; any pages removed from the book would interrupt patterns. Today, more practitioners of this beautiful art exist than ever before in history—we can see marbling designs on picture frames, note cards, collages, origamis, lampshades, and many other items in our daily lives.

The marbling process is characterized by the manipulation of floating pigment, and the transfer of the resulting design onto paper or some other material. It's not as trivial a task as it seems. With traditional marbling, the design process and the resulting design quality are largely constrained by various physical conditions—such as pigment floatability, liquid viscosity, or even environmental humidity. The resulting designs aren't directly useable for seamless tiling and the design size is constrained by the size of the trays used to hold the liquid. If the resulting design is not satisfactory, a marbler has to skim all the liquid from the tray and restart the designing process from scratch. This article therefore provides an exploration into the marbling process and shows how we can algorithmically simulate this traditional art. We then present an interactive computer system for creating various marbling textures.

Traditional marbling process and our approach

A traditional marbling process consists of the following three steps:[1]

1. After preparing a shallow tray, marblers add paint colors, which disperse onto the thickened liquid's surface to create a basic design. Eyedroppers, pipettes, and bamboo brushes act as color applicators. The liquid should be thick enough to keep color paints floating on its surface.
2. Marblers use sticks, combs, or other tools to make patterns. As they wield the combs back and forth across the tray, the drops of paint are pulled into elongated shapes. An intricate pattern usually requires running through the color several times. Combs of differing spacing will produce a different effect, as will the manner of drawing the comb down the tray.
3. Once the pattern is complete, marblers carefully lower a sheet of paper (or fabric, wood, leather, and so on) onto the liquid's surface to absorb the floating colors, and then lift it off, rinse it, and dry it. That particular design can never again be created.

Although the last step contributes to the quality of the final design appearing on the paper or fabric, the pattern itself is mainly determined during the first two steps. Based on such an observation, we currently focus on the first two steps in our simulation. Our designing system offers many contributions. First, we relieve marblers from physical constraints, letting them control the liquid's property by changing the viscosity coefficient, designing their own patterning tools, and providing the ability to undo or redo each step of the design process to exploit the best results through trial and error. We treat different colored inks as independent layers to prevent color mixing, a desirable feature of real-world marbling.

Next, a physically based marbling simulation process—that is, a 2D fluid dynamics simulation—

 Published by the IEEE Computer Society

enables realistic marbling designs. Our system provides intricate designs through color advection in the 2D flow field obtained by numerically solving the Navier-Stokes equations, mainly based on the stable fluids method.[2,3]

Due to the large amount of parallelism in graphics hardware, modern GPUs demonstrate significantly higher performance than CPUs in most numerical applications. We further exploit these advantages by employing a multigrid solver on the GPU for solving the Navier-Stokes equations more efficiently. With our system, users can get a real-time system response to their operations, in the same way as they work with real-world marbling.

Finally, our computer marbling system makes it possible to integrate the design process into modern CAD/CAM systems. Procedurally generated marbling textures can be directly used for rendering objects and scenes decorated with marbling designs.

GPU-based marbling simulation

During the marbling process, either applying colors to the liquid or moving a patterning tool causes the liquid to flow. As a result, the fluid carries colors to form various designs. Therefore, depending on the velocity field of the liquid resulting from user operations, we can generate the marbling design as the result of color advection in the fluid. Several papers address the development of efficient fluid solvers tailored for computer graphics.[2-4] Since the colors actually float on the liquid surface, we use a 2D fluid dynamics simulation based on Stam's stable solver.[3] However, we have implemented our simulation on the GPU instead of the CPU for achieving the best time efficiency.

Incompressible Navier-Stokes

In the fluid dynamics field, the incompressible fluid's motion at any given time is governed by its velocity field $\mathbf{u}$, which satisfies the following Navier-Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{F} \quad (1)$$

$$\nabla\cdot\mathbf{u} = 0 \quad (2)$$

where ρ denotes the fluid density, ν the kinetic viscosity, p the local pressure, and $\mathbf{F}$ represents any external forces that act on the fluid. The four terms on the right-hand side of Equation 1 are accelerations, while Equation 2, the continuity equation, enforces the incompressibility assumption by ensuring that the fluid always has zero divergence. We can only solve analytically the Navier-Stokes equations for a few simple physical configurations. However, while engineering problems require an accurate result for accuracy and safety purposes, we are more interested in the visual effects in our application. Thus, an incremental numerical solver suits our needs.

To solve the equations numerically, we discretize the computational domain—which corresponds to the area of the tray—to a grid of $m \times n$ cells. Both the velocity and local pressure are defined at the center of each cell. To use all four channels in the floating point units, the computational domain is then divided into four quadrants, which are layered into the (x, y, z, w) channels of a single texture with half the size in each dimension. The key to the solution is to take steps in time and update the velocity field at each time step via four computations, as described later.

Advection. Intuitively, advection is the process by which a fluid's velocity transports itself and other particles along with the flow. The nonlinear term $-(\mathbf{u}\cdot\nabla)\mathbf{u}$ in Equation 1 stands for this self-advection of the velocity field. To determine the velocity at a point $\mathbf{x}$ at the new time $t + \delta t$, we trace the trajectory from it back in time for its former position $\mathbf{x}'$ at time t, and copy the velocity at point $\mathbf{x}'$ to point $\mathbf{x}$. With this method, no

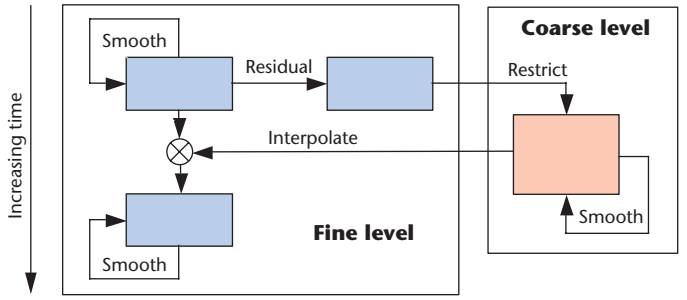
Related Work

In recent years, there has been increasing interest in nonphotorealistic rendering techniques that aim to simulate some particular artistic styles or media. We can classify existing techniques into roughly three distinct approaches. The first approach works directly from meshes or other 3D geometries. The second approach takes source images or photographs as input and converts them into renderings with a painterly look. Lastly, interactive systems, such as computer-generated watercolor,[1] depend on user-defined input. Different approaches can also be combined together, to leverage the strengths of each.

Mao's AtelierM system was the first published work on a featured digital marbling system.[2] It's built on the physical model of the traditional marbling process. The approach models marbling's patterning process as a 2D computational fluid dynamic problem, which is then solved numerically. As an interactive system, however, AtelierM cannot afford a real-time response. The pattern manipulation process is approximately modeled as a 2D steady flow. Users must design the full path of one particular operation beforehand, wait for a few seconds for the resulting static pattern, and then design another. Akgun as well as Acar and Boulanger have used similar methods, the latter introducing an improved multiscale fluid model, but still with low time efficiency.[3,4] By adopting the idea of Harris,[5] we extended Mao's work to perform the fluid dynamics simulation on the GPU and implemented a real-time designing system. Our system provides a virtual environment that allows users to experience the process of moving tools in liquid and see the patterns change gradually. It's also possible for users to make several operations simultaneously and watch the interactions among them.

References

1. C.J. Curtis et al., "Computer-Generated Watercolor," *Proc. Siggraph*, ACM Press, 1997, pp. 421-430.
2. X. Mao, T. Suzuki, and A. Imamiya, "AtelierM: A Physically Based Interactive System for Creating Traditional Marbling Textures," *Proc. 1st Int'l Conf. Computer Graphics and Interactive Techniques in Australasia and South East Asia*, ACM Press, 2003, pp. 79-86.
3. B.T. Akgun, "The Digital Art of Marbled Paper," *Leonardo*, vol. 37, no. 1, 2004, pp. 49-51.
4. 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.
5. M.J. Harris, "Fast Fluid Dynamics Simulation on the GPU," *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, Addison-Wesley, 2004, pp. 637-665.



1 A multigrid V-cycle.

matter how large δt, the velocity at the new time step obviously does not exceed the maximum value of the velocity field at the previous time step. This makes the solver always stable. This technique is known as the *method of characteristics*.[2]

Diffusion. We obtain the diffusion term $\nu\nabla^2\mathbf{u}$ from Equation 1 by solving the viscous diffusion equation $\partial\mathbf{u}/\partial t = \nu\nabla^2\mathbf{u}$ using an implicit formulation:

$$(\mathbf{I} - \nu\delta t\nabla^2)\mathbf{u}(\mathbf{x}, t+\delta t) = \mathbf{u}(\mathbf{x}, t) \quad (3)$$

where $\mathbf{I}$ is the identity matrix. This is a Poisson equation for velocity. Like the implicit method for computing advection, this formulation is stable for arbitrary time steps and viscosities.

Force application. The force application step is as simple as adding the external forces directly to the current velocity. The forces are in the tangential directions of the paths and their magnitudes are a given constant specified by users. One special case is creating the initial pattern: when applying colors to the liquid, a previously dropped spot can be pushed aside by the subsequently dropped spots, changing its shape. Also, a spot can be dropped right within an existing spot and force it to spread to the surroundings. The initial pattern is created as procedural textures in Mao's AtelierM system,[1] while ours uses physical simulation for that step, too. To produce the effect that color spots push each other, we assign an external force of constant magnitude (1 in our current implementation) to each particle of the currently dropped spot, which points outward from its center.

Projection. After the previous three computations, the result is a new divergent velocity field $\mathbf{w}$. We end each time step with a projection operation that projects $\mathbf{w}$ onto its divergence-free component $\mathbf{u}$. As stated by the Helmholtz-Hodge Decomposition Theorem,[5] we can correct the divergence of the velocity by subtracting the gradient of pressure field $\mathbf{w} = \mathbf{u} + \nabla p$. Applying the divergence operator to both sides of this equation yields another Poisson equation:

$$\nabla^2 p = \nabla \cdot \mathbf{w} \quad (4)$$

which is a Poisson equation for pressure. We can solve both Equations 3 and 4 using a V-cycle multigrid scheme, which we'll describe later. After we arrive at the pressure p, we can use it to compute the divergence-free field $\mathbf{u}$, which is the final result of a single time step.

For the initial conditions, the velocities and pressures are set to zero at all cells. When zero Dirichlet boundary conditions are used, velocity goes to zero at the boundaries, while the rate of change of pressure in the direction normal to the boundary is set to zero. This is a natural simulation to the real marbling. Since we aim to create seamless marbling textures, we have eliminated the boundaries by identifying the opposite sides of the computational domain instead, so that the velocity and pressure at the boundaries of the domain are equal to those at the opposite side. That is, the fluid flow out of the tray enters the tray again from the other side of the tray.

Multigrid solver for Poisson equations. We can discretize the Laplacian operator in Equations 3 and 4 using the finite difference form:

$$\nabla^2 U_{ij} = U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j}$$

where U is the current solution, and subscripts i and j refer to the grid's row and column indices. The standard solution for Poisson equations is a simple relaxation scheme that starts with an approximate solution and improves it through a number of Jacobi iterations, using the results of the previous iteration as input to the next. Earlier GPU-based fluid simulators have used the Jacobi iteration because of its simplicity and easy implementation.[6] But some more sophisticated methods—such as conjugate gradient and multigrid—converge faster.[7-10] We adopted a V-cycle multigrid algorithm in our current implementation.

Figure 1 shows an illustration of two levels in the multigrid algorithm. Our approach performs an iteration toward a solution at the finest level by starting with an initial guess and improving it through repeated application of the V-cycle algorithm. A multigrid V-cycle starts with a few presmoothing steps. After that, we calculate the residual value at each grid cell by applying the Laplacian operator to the current resolution and restrict the residual value to a coarser grid. We then perform a V-cycle on this restricted residual and so on recursively to some coarsest level. For simplicity we use the Jacobi iteration for the coarsest level solver. Finally, we interpolate back the approximate solution to a higher resolution grid and perform postsmoothing. Both pre- and postsmoothing steps use the Jacobi iteration as well. For our simulation, we only use two V-cycles, with two presmoothing steps, two postsmoothing steps, and ten iterations (default) on the coarsest grid. Since the major part of iterations occurs on the low-resolution grid, the multigrid algorithm demonstrates much higher performance. More details about the solver are available elsewhere.[7,9]

Multilayer density field

Until now, the simulation gives only the fluid velocity. A straightforward way to represent color paints in the flow field is to maintain an additional density field, which is only carried along by the fluid, but does not affect the flow itself. Other approaches use a scalar field as the density field to show a gray-scale fluid,[3,10] while another method

uses three scalar fields corresponding to the RGB components of color.[6] Because a texture has four channels, these three scalar fields can be layered into a single texture.

An accurate simulation to the advection of colors requires solving the advection diffusion equations taking into consideration color paint properties, such as the diffusion and dissipation of color paints in the fluid. In the case of marbling, the liquid viscosity is usually large. Therefore, the diffusion and dissipation rates of each color should be quite low. Therefore, currently we only consider the color advection, using the following equation:

$$\frac{\partial d}{\partial t} = -(\mathbf{u} \cdot \nabla) d$$

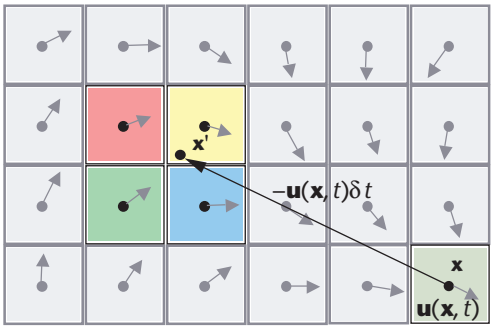
where d is the density field that represents color concentration carried by the fluid. We can solve this equation just like the advection step for velocity. To generate marbling designs suitable for seamless tiling, color advection should also satisfy the periodic condition.

As shown in Figure 2, to calculate the color for a point $\mathbf{x}$, we trace back from $\mathbf{x}$ to $\mathbf{x}'$ and perform bilinear interpolation of the four cells closest to $\mathbf{x}'$. If we use the most straightforward approach—that is, represent the color with three scalar fields corresponding to RGB components and interpolate each scalar field independently—the resulting color at $\mathbf{x}$ would be a mixture of the colors at the four cells. This, however, is physically incorrect, because different paint types usually don't mix together due to the effect of ox gall, which acts as color separator in marbling. To prevent the blending of different paints, we define a multilayer density field. A multilayer density field is composed of an arbitrary number of layers (virtually, 8 to 16 is enough), each corresponding to a special paint color the user applies. To use all four channels in the floating point units, we use one texture for four layers. We generate the final effect by composing the colors for the layers according to their densities. Another advantage of the multilayer method is that users can easily adjust the paint colors on each layer—they can even change the color schemes of a ready-made marbling texture.

Image sharpening

In real-world marbling, ox gall added to the paint forms a wall of fat around each color and prevents colors from blending with each other. This is an important property contributing to the sharp and vivid impression of marbling designs.[1] Although, as introduced previously, we succeeded in preventing the mixture of different colors by using a multilayer density field, the colors still diffuse into the background and can result in a texture without sharp boundaries among different colors. An accurate simulation to the ox gall effect involves modeling the surface tension and interaction among different materials. However, we have chosen to keep the sharpness of marbling textures through a simpler way—that is, shock filtering.

Shock filtering's underlying principle is based on diffusing energy among neighboring pixels.[11] A dilation or erosion process is applied depending on whether the pixel belongs to the influence zone of a maximum or a minimum. A sharp shock is created between two zones after several passes.[12] Compared to other image sharpening schemes—for example, high-pass filtering—the shock filter method is more appropriate for transforming smooth transitions resulting from texture interpolation, which is the main cause of blurriness in our algorithm.



2 Advection of colors. Colors at the four cells closest to $\mathbf{x}'$ are interpolated to get the new color of $\mathbf{x}$ at time $t + \delta t$.

We apply the shock filter on the composition of the multilayer density field with eight iterations. Notice that the result of image sharpening is only used for displaying, but not as the input for the simulation at the next time step, so that it doesn't affect the physical model's accuracy.

Interactive designing system

We have implemented an interactive marbling system as a Windows application using Visual Studio.NET and OpenGL. We implemented the fluid solver as a series of pixel shaders, using the Cg language coupled with an Nvidia GeForce FX 7800 GS GPU. Figure 3 (next page) shows a snapshot of our designing system.

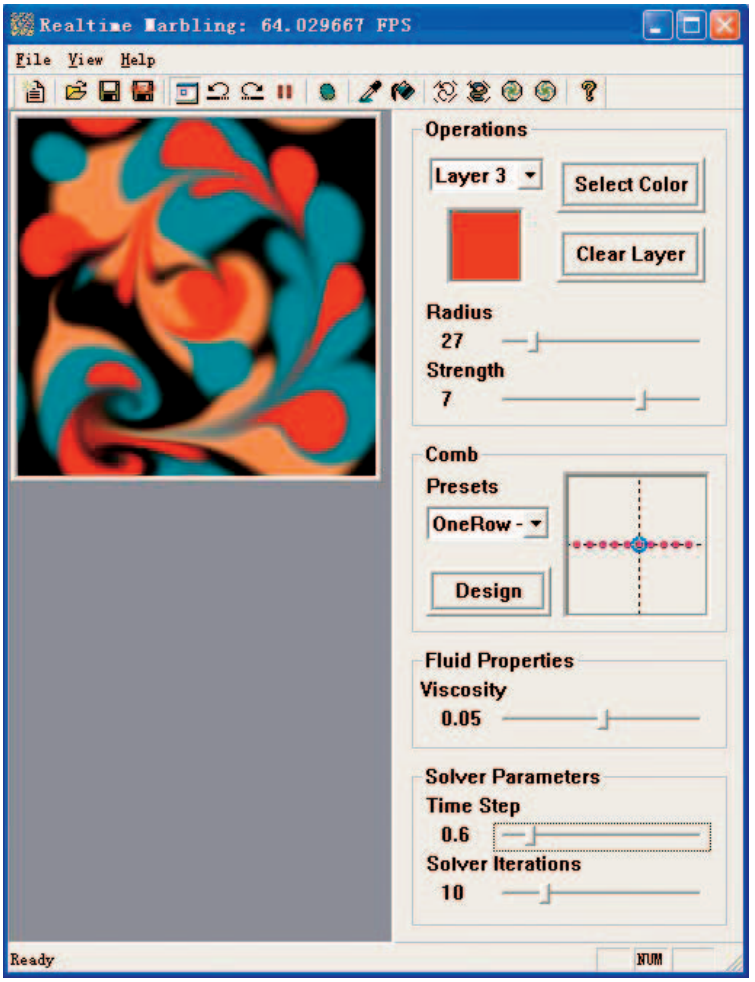
The control panel in Figure 3 contains all the parameters that users can specify. Besides the comb pattern, which we'll discuss later, users can also determine the paint color and drop size, patterning operation strength, liquid viscosity, and time step and coarsest-level iterations used in the fluid solver. Many earlier GPU implementations of fluid simulation did not perform diffusion ($\nu = 0$) on the velocity field, because visually the numerical diffusion is sufficient.[4,6,7] On the contrary, the marbling process requires high viscosity. In our application we use $\nu = 0.05$ as the default setting. We recommend a higher viscosity when the magnitude of patterning force is large.

Our system supports the following functions:

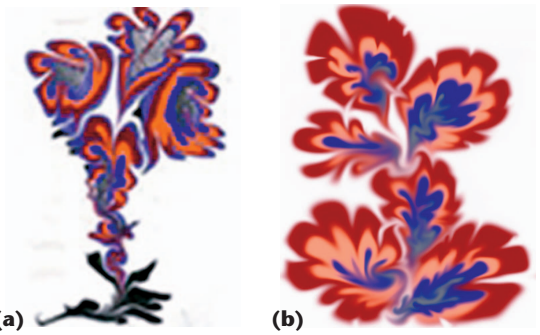
- color application,
- stick/comb patterning,
- swirl patterning,
- tool design,
- history function,
- color scheme, and
- image sharpening.

For the color application, users apply paint colors onto the liquid to create a basic design. Our system provides two color applicators: a pipette for dropping color spots, and a bucket for pouring a continuous stream of dye. Each color can also be cleared to empty according to its layer.

For the stick and comb patterning, users make patterns by dragging the mouse in the simulation window.



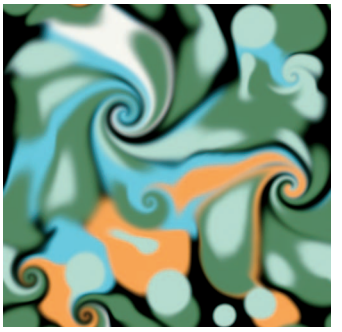
3 Screen capture of our real-time marbling system.



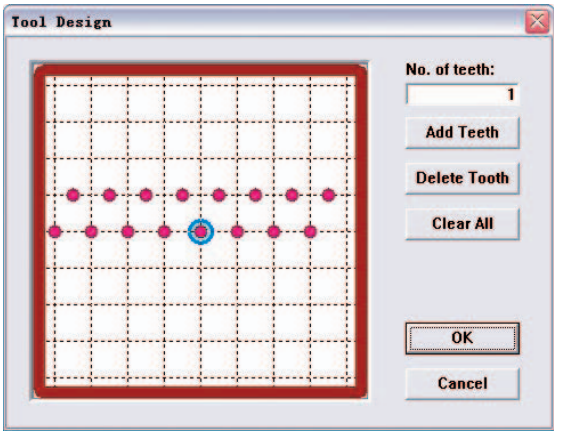
4 (a) A real marbling design. (b) The similar pattern generated with our interactive designing system. We ran the simulation with a domain size of 384 × 512 pixels.

With comb patterning, users can choose one of the predefined combs or design a new comb with the tool design function. The comb pattern in current use is displayed in the middle of the control panel. In Figure 4, we present a pattern created using only the stick patterning tool.

To make swirl patterns, users create clockwise or counterclockwise marbling patterns with a user-specified radius. Figure 5 shows a swirl pattern created with this function.



5 A green swirl pattern with a domain size of 512 × 512 pixels.



6 Control window for the tool design function. The red dots denote teeth on the comb; the blue one stands for the handle (the mouse position when holding the tool).

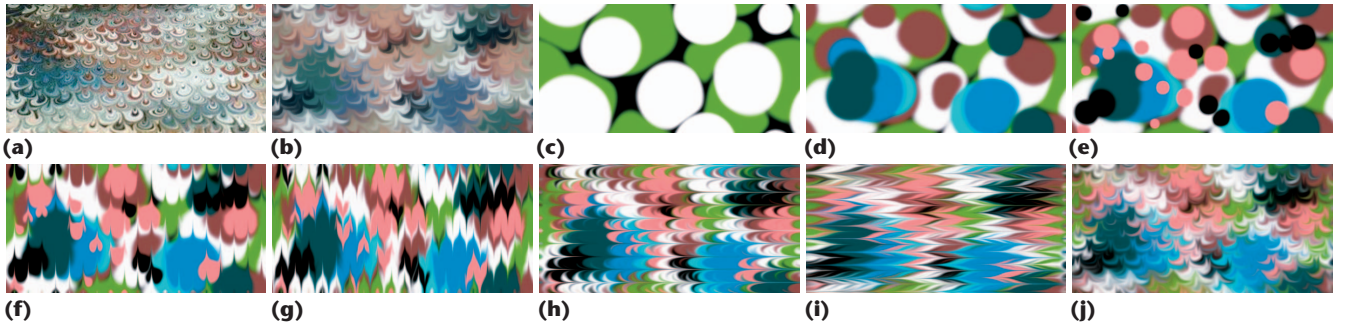
The tool design function cooperates with the comb patterning function to let users design patterning tools. The control window in Figure 6 shows a comb with staggered teeth, which is suitable for creating the classic bouquet patterns.

With the history function turned on, users can undo or redo their previous operations, and select a particular step to continue the designing process.

In addition, users can change the paint color scheme on each layer. With this function, it's convenient for users to alter the color scheme of a ready-made marbling texture.

In image sharpening mode, the marbling texture looks more vivid with an additional image sharpening step. Since this operation does not really affect the simulation, we recommend that users turn this mode off during the designing process to get a higher frame rate, and only turn it on to see the improved visual effect.

Here we demonstrate the process of generating the bouquet pattern, one of the most popular and famous marbling patterns. Figure 7a is a real-world example of the bouquet pattern that has been combed five times; Figure 7b shows a similar design created using our system. Once the basic design has been made (see Figures 7c through 7e), then we draw a comb in a straight line from bottom to top (see Figure 7f). The pattern generated by this action is called *nonpareil*, a commonly used intermediate step during the marbling process. We then draw back the same comb in the other direction, the teeth passing in between where they have passed before,



7 (a) A real-world bouquet pattern. (b) A design similar to that in (a) created with our system. (c – j) Images showing the steps in the bouquet pattern-generation process. We ran the simulation with a domain size of 512 × 256 pixels.

forming the gel git pattern (see Figure 7g). Repeating the last two steps in the orthogonal direction yields another gel git (see Figures 7h and 7i). We perform the final combing using a comb with two staggered rows of teeth, such as the one shown in Figure 6. A side-to-side motion while combing produces the curved shapes (see Figure 7j). We finally open the image sharpening mode and tune the color scheme. All the images shown in Figure 4b, Figure 5, and Figures 7b through 7j satisfy the periodic condition, and we can directly use them for seamless texture mapping—this is the case with all images created with our system.

In Figures 8 and 9 we demonstrate the potential applications of computer-generated marbling textures to 3D image synthesis. The top-left image in Figure 8 is a marbling texture that our system created. The top-right image in Figure 8 is a 2 × 2 tiling of the marbling texture, which we've applied to the Venus model using projection mapping. The seamless feature of our created marbling texture can be easily observed. In Figure 9, we apply our created texture to the design of a facial tissue box.

We have run a series of marbling simulations using our designing system, and compared the performance of our multigrid–multilayer solver to a standard Jacobi solver on the GPU (see Table 1). Both implementations are reasonably optimized, and comparable in visual effect. All tests used a 3-GHz Pentium 4 and an Nvidia GeForce FX 7800 GS graphics card.

Conclusions and future directions

Marbling is fun and easy at the beginner level. Unlike many artistic media, with decent instruction users can produce some fine papers on their first few tries. There is such an infinite variety of patterns and color combinations that it never gets boring. It's even easier for beginners to start with a virtual environment, because they can simply undo their misoperations, and discover the right way through trial and error.

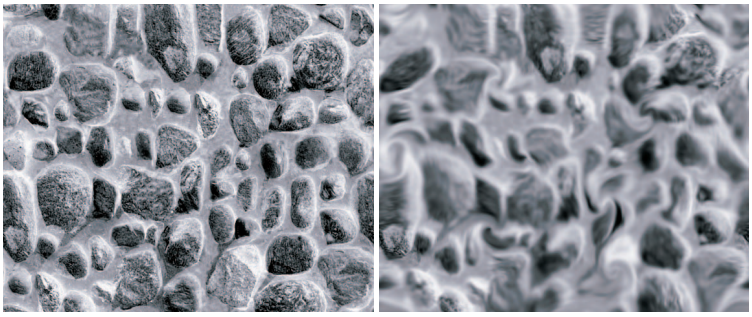
An extended application of our marbling system is texture synthesis. We might not start every designing process from scratch, but take a refined texture image as input. Then we can use the system to apply some marble-like effect onto it, without disturbing the seamless-tiling property of the original texture image. Figure 10 (next page) demonstrates an example of this. Currently we don't support the third step of the traditional marbling process in our implementation. Adding this function involves modeling the absorption and

8 Design application to 3D image synthesis.

9 Design application to a facial tissue box design.

Table 1. A standard Jacobi solver using 20 iterations per pass versus our multigrid–multilayer solver using the 10 coarsest-level iterations.

Domain Sizes (pixels)	Jacobi (frames per second)	Ours (frames per second)	
		No Image Sharpening	With Image Sharpening
256 × 256	37.6	64.3	37.6
512 × 512	12.2	24.4	15.0
1,024 × 1,024	3.1	6.3	4.2



(a) (b)

10 Application to texture synthesis: (a) the original texture and (b) the marbled one.

Web Extra

For a video of the virtual marbling system in use, go to http://opac.ieeecomputersociety.org/opac?year=2007&volume=27&issue=2&acronym=cga.

diffusion of colors on the paper or other materials, which is our next goal.

On the other hand, we have yet to perfect the model of the first two steps. For example, we still suffer from blurred color boundaries using shock filtering. In a newly published paper on digital marbling, the advection model is improved by the combination of a high-order spatial interpolation and a sharp interface method to match the highly detailed, sharp fluid interfaces in marbling patterns.[13] This suggests a potential way to overcome this limitation in our future work. Another future research direction is to further improve the system's speed. Although we have mapped the fluid solver to the GPU coupled with a multigrid Poisson solver, the performance is not yet satisfactory under high grid resolutions. We believe that performance can be boosted largely if we further exploit the parallelism provided by today's graphics hardware. ■

Acknowledgments

This project is supported by the National Natural Science Foundation of China (grant nos. 60340440422, 60573153, and 60533080) and the Natural Science Foundation of Zhejiang Province (grant no. R105431).

References

1. X. Mao, T. Suzuki, and A. Imamiya, "AtelierM: A Physically Based Interactive System for Creating Traditional Marbling Textures," *Proc. 1st Int'l Conf. Computer Graphics and Interactive Techniques in Australasia and South East Asia*, ACM Press, 2003, pp. 79-86.
2. J. Stam, "Stable Fluids," *Proc. Siggraph*, ACM Press, 1999, pp. 121-128.
3. J. Stam, "Real-Time Fluid Dynamics for Games," *Proc. Game Developer Conf.*, 2003, http://www.dgp.toronto.edu/people/stam/reality/Research/pub.html.
4. R. Fedkiw, J. Stam, and H.W. Jensen, "Visual Simulation of Smoke," *Proc. Siggraph*, ACM Press, 2001, pp. 15-22.
5. A.J. Chorin and J.E. Marsden, *A Mathematical Introduction to Fluid Mechanics*, 3rd ed., Springer, 1993.
6. M.J. Harris, "Fast Fluid Dynamics Simulation on the GPU," *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, Addison-Wesley, 2004, pp. 637-665.
7. J. Bolz et al., "Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid," *ACM Trans. Graphics*, vol. 22, no. 3, 2003, pp. 917-924.
8. C. Garcia et al., "Beowulf Performance in CFD Multigrid Applications," *Proc. 10th Euromicro Workshop Parallel, Distributed and Network-Based Processing*, IEEE CS Press, 2002, pp. 7-14.
9. N. Goodnight et al., "A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware," *Proc. Eurographics/Siggraph Workshop Graphics Hardware*, Eurographics Assoc., 2003, pp. 102-111.
10. J. Kruger and R. Westermann, "Linear Algebra Operators for GPU Implementation of Numerical Algorithms," *ACM Trans. Graphics*, vol. 22, no. 3, 2003, pp. 908-916.
11. F. Guichard and J.M. Morel, "A Note on Two Classical Shock Filters and Their Asymptotics," *Proc. 3rd Int'l Conf. Scale-Space and Morphology in Computer Vision*, Springer, 2001, pp. 75-84.
12. S. Osher and L.I. Rudin, "Feature-Oriented Image Enhancement Using Shock Filters," *SIAM J. Numerical Analysis*, vol. 27, no. 4, 1990, pp. 919-940.
13. 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.

Xiaogang Jin *is a professor at the State Key Laboratory of CAD & CG at Zhejiang University, People's Republic of China. His research interests include implicit surface computing, cloth animation, crowd and group animation, computer animation, and digital geometry processing. Jin has a BSc in computer science, and an MSc and PhD in applied mathematics, all from Zhejiang University. Contact him at jin@cad.zju.edu.cn.*

Shaochun Chen *is a software engineer at Ubisoft Shanghai. Her research interests include computer games, texture design, and computer animation. Chen has an MS in computer science from Zhejiang University. Contact her at chenshaochun@cad.zju.edu.cn.*

Xiaoyang Mao *is an associate professor at the University of Yamanashi in Japan. Her research interests include flow visualization, texture synthesis, nonphotorealistic rendering, and human–computer interactions. Mao has an MS and PhD in computer science from Tokyo University. Contact her at mao@yamanashi.ac.jp.*

Article submitted: 16 May 2006; revised: 17 Aug. 2006; accepted: 17 Aug. 2006.