Vector Fluid: A Vector Graphics Depiction of Surface Flow

Ryoichi Ando*
Graduate School of Design
Kyushu University

Reiji Tsuruno†
Faculty of Design
Kyushu University.

Figure 1: ***Vector graphics depicted fluid:*** *These figures were generated interactively using our prototype and they can be described in terms of spline curves. Zoom in with your PDF viewer to see how our vector fluid preserves very high levels of detail.*

Abstract

We present a simple technique for creating fluid silhouettes described with vector graphics, which we call "Vector Fluid." In our system, a solid region in the fluid is represented as a closed contour and advected by fluid flow to form a curly and clear shape similar to marbling or sumi-nagashi (See Figure 1). The fundamental principle behind our method is that contours of solid regions should not collide. This means that if the initial shape of the region is a concave polygon, that shape should maintain its topology so that it can be rendered as a regular concave polygon, no matter how irregularly the contour is distorted by advection. In contrast to other techniques, our approach explicitly neglects topology changes to track surfaces in a trade off of computational cost and complexity. We also introduce an adaptive contour sampling technique to reduce this extra cost. We explore specific examples in 2D for art oriented usage and show applications and robustness of our method to exhibit organic fluid components. We also demonstrate how to port our entire algorithm onto a GPU to boost interactive performance for complex scenes.

CR Categories: I.3.5 [Computational Geometry and Object Modeling]: Curve, surface, solid, and object representations— [I.3.3]: Picture/Image Generation—Line and curve generation

Keywords: Non-photorealistic rendering,vector graphics,surface flow,marbling

*
†

1 Introduction

For more than a decade in the graphics community, the depiction of fluids has been mainly focused on capturing realistic behavior. Thanks to the considerable contribution of seminal studies, we now have extensive knowledge about how to achieve the photorealistic rendering of fluids. On the other hand, although some approaches to the artistic rendering of fluid have been proposed, it seems that *Non-Photorealistic Rendering* (NPR) of fluids has not been researched intensively and techniques for rendering fluids using vector graphics still remain untouched. In this paper, we specifically explore art oriented depictions of 2D surface flow similar to marbling or sumi-nagashi, using vector graphics. In order to achieve vector graphics rendering, we have to track solid regions in the fluid. However, traditional methods which address complex topological changes are often too dissipative, blobby or complex to capture clear and detailed silhouettes.

Instead, we adopt the principle that no parts of contours should collide by advection so that the topology of a region should not change, because the streaklines of fluid do not collide. Such solid regions may merge or split when they get very close, which does involve topological changes; despite exhausive efforts, such topological changes are still daunting to track precisely. In the context of surface detail preservation, a state-of-the-art topological changes algorithm also acts as a filter, which smoothes out surface detail. In our approach, we explicitly neglect topological changes to simplify the problem. We consider this to be a computational trade off. Ignoring topological events enables us to obtain detailed features and simplifies the algorithm, at the expense of considerable extra computational cost. Fortunately, we found that the cost in two dimensions was not so extreme as to monopolize the CPU. In fact, every figure shown in this paper was created interactively. In our model, a solid region is represented with explicit closed form contours so that rendering or export of the region into vector graphical form is done simply by plotting sequential contour vertices as a concave polygon.

The main drawback of our approach is that the computational cost rises quickly as the shape becomes complex due to the sharp propagation of vertices to track. Additionally, discretized contour representation causes collisions around adjacent contours. In order

to deal with these disadvantages, we introduce a simple technique to change the discretization space of contour vertices adaptively, which we call "*Adaptive Refinement*." We further introduce a method to implement the entire algorithm on a GPU to achieve sufficient performance to make real-time interaction with the simulation feasible.

The key components of our algorithm are listed as follows: (a) Ignorance of topological changes; since streaklines of fluid never collide, contours of the fluid flow never collide either. This leads to the idea that the topology of the solid region should not change no matter how irregularly shapes are distorted by advection. (b) Adaptive Refinement: if we let vertices propagate without bounds, the computational cost rises sharply as complexity increases. We suppress this propagation by controlling sampling spaces in the local environment. (c) Shapes are depicted with vector graphics; the closed form contours of solid regions are treated as regular concave polygons. By just plotting a sequence of vertices, the whole shape can be directly translated into vector graphics format.

2 Previous work

Our work is related to studies in two categories: artistic expression of fluids and surface tracking methods. The most relevant work is that of [Acar and Boulanger 2006] which was intended to reproduce the visual effects of marbling flow using a physically derived flow model. They observed surface flow based on mesoscale dynamics and produced effects of fluctuations at different scales. In order to advect clear silhouettes under Eulerian grids, they employed b-spline interpolation and extended the range of concentration temporarily in the semi-Lagrangian advection phase. Eulerian grid approximation is limited in terms of the degree of resolution, if we wish to obtain a reasonable simulation. If we simulate with high resolution, it requires a great deal of per-pixel computation and memory.

Eden [Eden et al. 2007] proposed a method for rendering liquids in a cartoon-style manner. Exploiting a physically created fluid surface, they rendered it by emphasizing properties of the liquid's shape and motion, inspired by the abstraction and simplification of cartoon animations. This method resembles our own in that both have clear silhouettes and very few colors. However, they used an implicit contouring method as an underlying liquid animation; hence thin line detail is inherently smoothed out before it is stylized.

Selle et al. [Selle et al. 2004] introduced a technique for generating cartoon-style animations of smoke. Based on a physically-based simulated output, they traced marker particles and rendered them using depth buffer differences to generate clear smoke animations. McGuire et al. [McGuire and Fein 2006] extended this technique and developed a system for rendering real-time animations of smoke, in addition to introducing a novel self-shadowing algorithm. However, these methods cannot track precise thin lines with the level of detail seen in marbling-like fluid flow since each particle (primitive) is visibly large.

Witting [Witting 1999] presented a system that uses computational fluid dynamics to produce two-dimensional animated films. They employed a compressive fluid dynamics and restricted it to two dimensions to develop user controllable fluid experiences. The work presented in this paper shares his motivation in that both offer artists fluid environments for design purposes; we focus particularly on a vector graphics format.

Our work is also comparable with surface tracking techniques used in physically derived liquid animations. This topic has a long historical background; here, we briefly review those works since they mainly focus on a well-designed topological changes algorithm, which is orthogonal to our approach. Roughly, a free surface is tracked using three approaches: Eulerian grids, Lagrangian particles and explicit surfaces.

Among Eulerian approaches, Hirt [Hirt and Nichols 1981] proposed a *volume-of-fluid* method that constructs an approximation of the interface from cells which contain portions of the interface. Osher [Osher and Sethian 1988] proposed a level set method which has become dominant in industrial applications. In the level set method, a signed distance function ϕ from the interface is advected and the surface is implicitly located where $\phi = 0$. The advantage of these two Eulerian based approaches is that they do not require a special post-process to handle topological changes. However, they do suffer from numerical diffusion or loss of mass conservation [Enright et al. 2005].

Among Lagrangian methods, Enright [Enright et al. 2002] extended the level set method with Lagrangian particles and increased accuracy. Harlow [Harlow and Welch 1965] advected particles in *marker-and-cell* grids to identify deforming surfaces. Müller, Harada [Müller et al. 2003; Harada et al. 2007] incorporated *Smoothed Particle Hydrodynamics* (SPH) and applied it to visual simulations. These methods introduced blobby or splitting artifacts for thin details.

Meanwhile, a number of explicit front tracking methods have been proposed, both for computational dynamics [Glimm et al. 1998; Brochu and Bridson 2009; Wojtan et al. 2009; Müller 2009; Tryggvason et al. 2001] and for image processing [Kass et al. 1988; McInerney and Terzopoulos 2000]. In front tracking methods, a surface interface is constructed with explicit surface elements and advected by the underlying motion. The front tracking method offers a precise representation of the interface free from grid resolution or numerical diffusion; however, it suffers from self-intersections or complexity. They detected topological changes by uniform grid traps or by searching close edges. It turns out to be challenging to do this without losing surface detail. Except for the fact that they tackle complex topological changes, their method is comparable to ours. Since we eliminate topological changes, we instead suffer from a huge computational cost.

3 Method

Before we discuss the concrete algorithm, let us first briefly outline the overall workflow in Figure 2. To depict vector graphics, we first place closed contours in a fluid flow and then advect them along the fluid flow. Rendering of the solid region is performed in just the same way as rendering a regular concave polygon. Since particle-based advection preserves accurate mass conservation, [Enright et al. 2002; Puckett et al. 1997] we paint regions with plain colors.

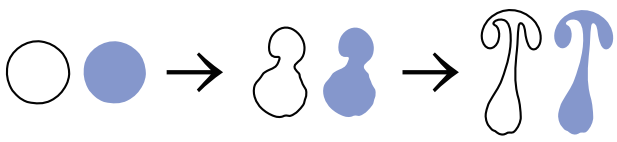

Figure 2: *Workflow of our method: We first place closed contours of a painted region in the fluid field, then advect or stretch them along the fluid flow. The rendering of the region is done just as a concave polygon is rendered.*

3.1 Contour Advection

In order to advect contours, we must generate the underlying fluid motion. For simplicity, we used the finite differential grid and semi-Lagrangian advection method to generate the fluid velocity field. Once we have fluid flow, we can place closed contours on the surface of water and advect them. In our system, we represent contours as a sequence of discrete points connected to each other and advect them in the Lagrangian manner. For a numerical method, it is inevitable that advection causes collisions; to prevent significant collisions, we employed a fourth order accuracy Runge-Kutta scheme. We denote this scheme as

$$\boldsymbol{v}^{t+\Delta t} = \phi(\Delta t, \boldsymbol{v}^t, \boldsymbol{u}^t). \tag{1}$$

where $\boldsymbol{v}^t$, $\boldsymbol{u}^t$ and ϕ denote the position of the contour vertex, the velocity of the fluid field and the Runge-Kutta scheme at time t, respectively. After all vertices have been advected ($t \leftarrow t + \Delta t$), we measure the distances between connected vertices and if this distance exceeds a threshold d, we insert a new vertex $\boldsymbol{v}_{\text{new}}$ midway between the relevant pair of vertices at the previous time step and advect it such that

$$\boldsymbol{v}_{\text{new}} = \phi\Big(\Delta t, \frac{1}{2}(\boldsymbol{v}_0^{t-\Delta t} + \boldsymbol{v}_1^{t-\Delta t}), \boldsymbol{u}^{t-\Delta t}\Big) \tag{2}$$

where $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$ denote the positions of each vertex in the pair. This time rewinding method helps position subdivided vertices more accurately than would linear subdivision with rapid advection. The effect may be slight; however, it also helps to reduce faceted edges or collisions, which allows us take larger time steps. On the contrary, if the distance is less than $d/2$ we collapse the vertex. This process is repeated until all connected vertices lie between d and $d/2$ apart.

3.2 Rendering and Export

To export the region as a vector graphic, we write the shape as a regular concave polygon. Starting from an arbitrary vertex, we move to the next connected vertex and write its position in sequence. Rendering of the concave polygon is performed efficiently by the stencil method [Woo et al. 1997]; for every pair of connected contour points $(\boldsymbol{p}, \boldsymbol{q})$, we draw a triangle polygon $(\boldsymbol{0}, \boldsymbol{p}, \boldsymbol{q})$ onto a framebuffer while inverting the existing values between 0 and 1. Finally, the solid region is filled with value 1.

3.3 Adaptive Refinement

With the aforementioned steps we may be able to create a vector fluid silhouette, however, implementation of this method can easily result in excessive computation due to rapidly increasing propagation of vertices over time. In this section, we introduce an "*Adaptive Refinement*" method to suppress this propagation.

From our observations, we found that we could omit vertices where the contour is not curved strongly, whereas where the contour is tightly curved or the fluid is eddying we needed to insert more points. Hence, we tuned the distance threshold d; we controlled it by taking its product with the local curvature $c_i^{\text{curvature}}$ and the turbulence $c_i^{\text{turbulence}}$, thus:

$$d_i = d_{\max} c_i^{\text{curvature}} c_i^{\text{turbulence}} + \varepsilon \tag{3}$$

where $d_{\max}$ denotes the maximum space between vertices. The value ε is a limit constant to avoid 0. To determine the local curvature, we employed a normalized second derivative:

$$c_i^{\text{curvature}} = \exp(-|\kappa|), \tag{4}$$

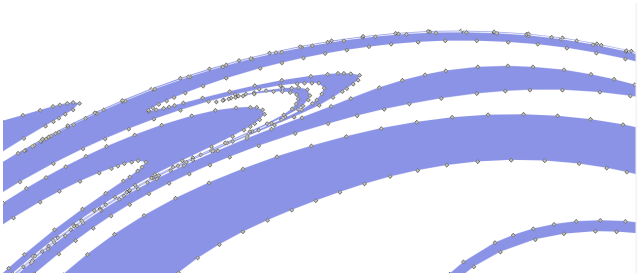

Figure 3: *Adaptive Refinement: Small diamond marks represent vertices to track. Notice that the space of vertices is sensitive to the curvature. In order to avoid collisions, more vertices are inserted where contours are highly curved.*

where κ denotes the finite differential curvature value. For local turbulence, we considered local vorticity:

$$c_i^{\text{turbulence}} = \exp(-|\nabla \times \boldsymbol{u}(\boldsymbol{v}_i)|), \tag{5}$$

where $\boldsymbol{v}_i$ is the position of i th vertex on a contour, $\boldsymbol{u}(\boldsymbol{v}_i)$ denotes the velocity of the fluid at position $\boldsymbol{v}_i$. Using such an adaptive d, we can unnoticeably remove a large fraction of the vertices while increasing the quality where contours have high curvature or turbulence. Notice that both $c_i^{\text{curvature}}$ and $c_i^{\text{turbulence}}$ range between 0 and 1. This strategy works well for most cases, although when d is sufficiently large it often fails to capture small perturbations of the fluid flow. To compensate for this drawback, we additionally diffused d_i along the contour. After we calculated each d_i value at $\boldsymbol{v}_i$ we iteratively assigned to d_i a value as follows:

$$d_i \leftarrow \sum_{n=-w}^{w} G(\alpha, n) d_{i+n}, \tag{6}$$

where w and $G(\alpha, x)$ denote the window radius and a gaussian function, respectively. Typically, $w = kd_{\max}^{-1}$. By diffusing d_i, rapid agitation around vertices triggers its neighbors to have a small d_i value; therefore, such rapid motion is well captured by the neighboring vertices. We illustrate the effect of Adaptive Refinement in Figure 3.

The Adaptive Refinement technique prevents large numbers of unnecessary vertices from being inserted; however it still produces intersections around thin or adjacent regions. Fortunately, we found that slight collisions of contours do not contribute to significant visual artifacts, however, for the purpose of generating high DPI images such as big posters, one may want to remove collisions as much as possible. In such cases, we add a proximity term $c_i^{\text{proximity}}$ in equation (3) as

$$c_i^{\text{proximity}} = 1 - \exp(-d_i^{\text{proximity}}), \tag{7}$$

where $d_i^{\text{proximity}}$ denotes the distance function from the nearest opposite contour. This magnifies contour subdivision; instead, generated collisions are minimized. For such a thin line, one may just split it during the simulation, however, thin lines are important visual keys for detailed silhouettes so we let them grow as it proceeds.

3.4 GPU Acceleration

Although our algorithm runs at an interactive rate on a CPU, it is limited to only around 40,000 vertices (10FPS). In this section we introduce a GPU accelerated algorithm to boost realtime performance. Similarly to particle markers, contours can be tracked

with a collection of particles except that in our model the number of particles varies and explicit connection information is dynamically reconstructed. The straightforward approach may be to store each initial vertex into individual kernels and watch them propagate, although computational cost varies among kernels since contour growth is uneven, which results in a slowdown. To disperse the computation evenly we introduce a method for task diffusion.

Our GPU acceleration strategy consists of the following three steps: (i) contour advection (ii) contour subdivision (iii) task diffusion. In our system, each vertex has information about its position and a reference to the next connected vertex. We refer to this kind of vertex as a "task." On a GPU, a contour is decomposed into a collection of tasks, each one referring to the next task; tasks are then stored in separate kernels. Device memory storage is pre-allocated and mapped onto kernels.

In the advection phase, positions of tasks stored in each kernel are advected as in section 3.1 in parallel. In the contour subdivision phase, each kernel probes every stored task for its distance from the referenced task, and then subdivides or collapses it if necessary, as also described in section 3.1,3.3. Inserted vertices are stored into the kernel of the originating task. In this phase, numbers of tasks stored in kernels become uneven. In the task diffusion phase, we choose random pairs of kernels and compare their numbers of stored tasks. In the case that the opposite kernel holds a smaller number of tasks, we move tasks into the opposite kernel. For consistency, we choose a pair $(\mathrm{kernel}_i, \mathrm{kernel}_{(i+r) \bmod n})$, where kernel_i denotes the i th kernel, r is a shared random integer and n is the number of kernels. We avoid conflict by letting i be even and r be odd. This task diffusion phase averages out the numbers of tasks among kernels, which disperses advection and subdivision cost evenly.

To evaluate equation (7) vertex indices are sorted into uniform grids as described in [Grand 2007] and we constrain searches within neighboring grids. Solid regions are rendered using the stencil method by randomly writing task polygons $(\mathbf{0}, \boldsymbol{p}, \boldsymbol{p}_{\mathrm{reference}})$ into a vertex array where $\boldsymbol{p}$ denotes the position of a task and $\boldsymbol{p}_{\mathrm{reference}}$ the position of the referenced task. Empty slots are filled with null polygons. In our case the whole procedure including a description of the underlying fluid was ported using CUDA.

4 Results

Comparison with competing methods is shown in Figure 4. The particle level-set result was generated with an existing library [Mokberi and Faloutsos]. Eulerian advection was generated with a semi-Lagrangian advection scheme. Clear edges were enforced using the concentration transformation function introduced by Acar [Acar and Boulanger 2006]. Front tracking methods were omitted since they behave similarly to ours if topological change is skipped. All of those results were computed for shared fluid motion. Our computation took 7 seconds while others took more than ten minutes for a 1600x1600 grid. The thin line property was well maintained with our method whereas the particle level-set method broke thin lines and Eulerian advection filtered them out even at high resolution. The performance with a single threaded CPU and GPU is illustrated in Figure 5(a). Our implementation on a GPU runs several times faster than that on a CPU for a complex scene. Note that the performance of the GPU depends on the size of the pre-allocated GPU memory because the entire memory space is sent into the rendering pipeline. The maximum size of this example was around 100,000 vertices.

On the other hand, Figure 5(a,b) also reveals two critical limitations of our method. Firstly, a tractable number of time steps is limited to around a few thousand (but this number depends heavily on the environment), which can be easily reached. Secondly, contours are interactively tracked up to around 100,000 vertices. Despite these limitations, we found that our method is quite practical and effective for creating organic fluid components.

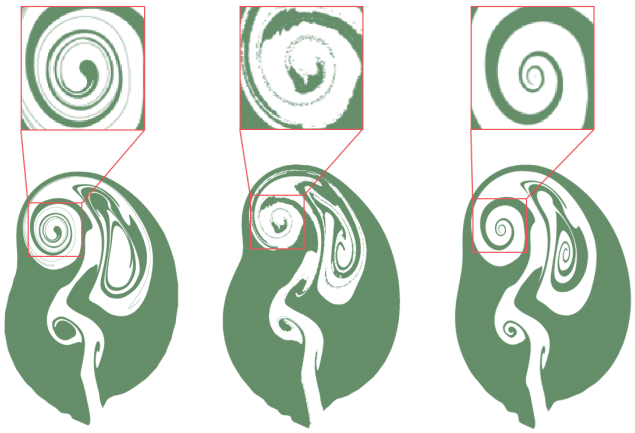

Figure 4: *Competing methods for a shared fluid field; the particle level-set method breaks thin lines and Eulerian advection filters out details whereas our method retains detailed features.*

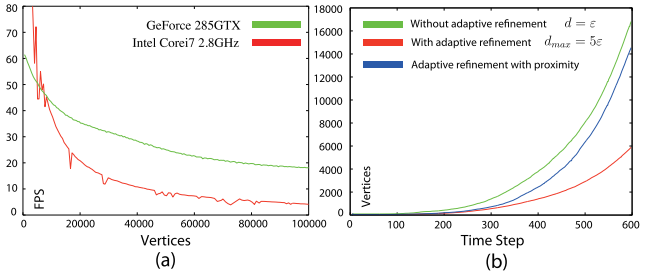

Figure 5: *Performance of our method with Adaptive Refinement (a): At 10,0000 vertices the CPU recorded 4 FPS while the GPU recorded 17 FPS. (b) Vertex propagation in time; there is a limit on the number of time steps that can be interactively simulated due to the sharp increase in the number of vertices.*

5 Applications

We built an interactive prototype both on an Intel Core i7 2.8GHz and a GeForce 285GTX running Linux. The underlying fluid was computed with a 64×64 grid resolution. With our prototype, users are allowed to drop pigments on the surface of water and disturb it simply by mouse dragging. In this section we introduce some interesting applications and explore the potential of our method.

5.1 Marbling and Sumi-nagashi

Figure 6 shows an example of a complex vector fluid generated by our method. When creating this silhouette, artists experienced responsive interactions just as in real marbling. As can be seen from the figure, our method is powerful enough to design stylish curly shapes similar to marbling or sumi-nagashi. The particular characteristic of our vector fluid method is that such closed contours are directly translatable into vector based programs. Figure 7 shows an example of a shape imported to Adobe illustrator and Adobe

FLASH. Since our vector fluid is wholly described in vector graphics, artists are allowed to print it with limitless high resolution.

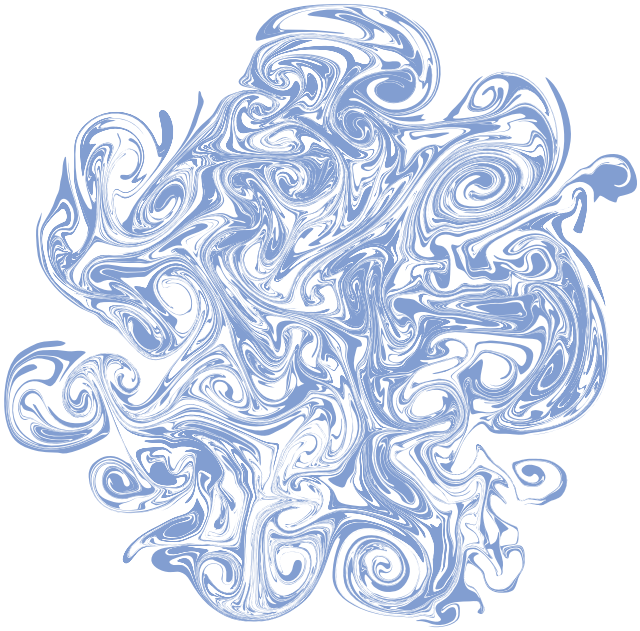

Figure 6: *Complex marbling-like silhouette: This vector fluid was created with our GPU prototype and during the simulation, rendering and interaction were responsive.*

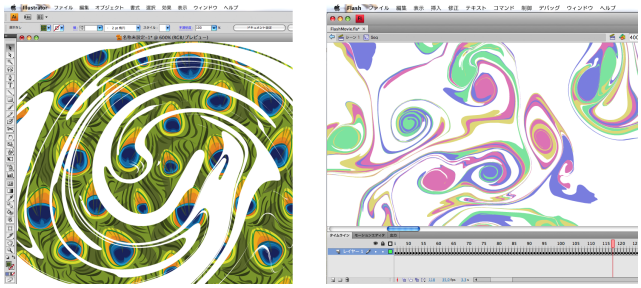

Figure 7: *Shape imported into Adobe illustrator and Adobe FLASH: Our vector fluid can be directly exported into a vector based program without losing detail.*

5.2 Shape Designing Tool

With a highly viscous fluid, the velocity diffuses through the entire space rapidly so that the effect of advection is only noticeable just after the user agitates the fluid. This allows artists a long time for each interaction and offers undo&redo functionality. One may exploit this effect for instant stylish curly designing. For example, Figure 8 was created by an artist within 10 seconds.

5.3 Target Driven Design

The underlying fluid can be controlled by external forces. For example we can combine with a Target-Driven fluid field [Fattal and Lischinski 2004] to guide regions into a specified target shape. In our implementation a solid region was rasterized and blurred at each time step. We skipped the smoke gathering procedure, since our vector field was not dissipative. Figure 9 shows an example of target-driven smoke animation effects. To generate

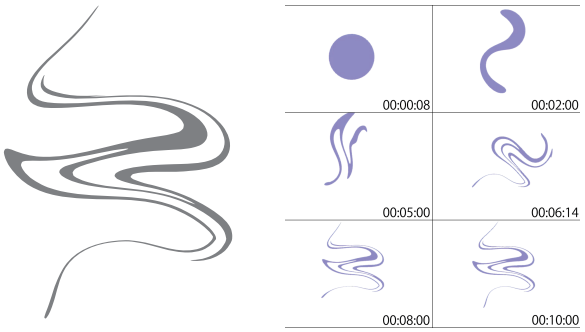

Figure 8: *Stylish logo created with viscous fluid; this figure was interactively created by an artist in only 10 seconds. The right side table shows snapshots of the design process.*

this figure, we first placed small ovals randomly on the canvas and gathered them according to the guiding flow.

Figure 9: *Effects of Target-Driven Smoke Animation: When combined, our vector fluid can be used to design user specified stencil art with curly fluid textures.*

5.4 FLASH Animation

Our vector fluid can be efficiently animated on a web browser using Adobe FLASH (but not interactively). To export it into a FLASH movie, each frame was computed and exported as a vector graphic. Finally, those sequential frames were imported into Adobe FLASH. The animation is rendered with the built-in FLASH engine so that it can be rotated or zoomed dynamically. As can be seen from Figure 10 our vector fluid offers a new medium for web page design.

6 Limitation

Our method does not handle changes in topology; therefore our method cannot be used for liquid animations. Despite our "*Adaptive Refinement*" technique and GPU acceleration, our method has inherent limitations with respect to the number of vertices and time steps to be interactively simulated as seen in Figure 5. The underlying fluid flow should be somewhat viscous to keep the contours tractable since small vortices increase subdivision quickly. Each time step should also be small otherwise advection produces many collisions. The dreadful drawback of excessive computational cost mainly arises from extremely thin lines of the solid region. For such

lines we are planning to employ an alternative simplified approximation, such as explicit lines with width.

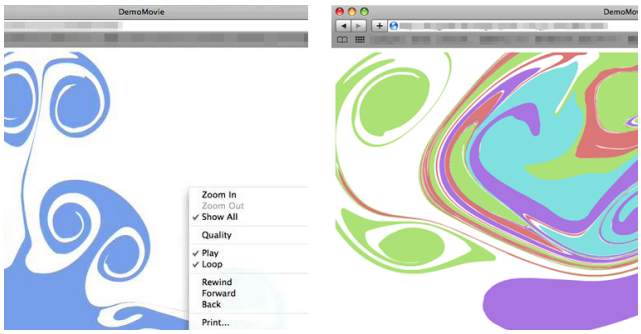

Figure 10: *Vector fluid animated in a web browser using the built-in FLASH renderer. To animate in FLASH, a sequence of vector graphics frames was precomputed and stitched.*

7 Discussion

We believe that the realtime interaction and the aesthetics of rendered silhouettes are the most impressive part of our method. Traditional approaches to fluid dynamics in computer graphics based on Eulerian grids or Lagrangian particles suffer from numerical diffusion or blobby artifacts when they are applied to generate clear silhouettes of surface flow. It may be possible to achieve our goal by employing state-of-the-art front tracking methods, but our method is greatly simplified and specifically tuned to produce reasonable results. The GPU acceleration and the fast rendering technique presented in this paper are only feasible within two dimensions. The overall idea may be conceptually extendable to three dimensions; however, it is not practically feasible.

8 Conclusion

In this paper, we have presented the simple idea of vector fluid and given a detailed explanation of the proposed method. The underlying principle of our method is that the topology of any concave polygon should not change by advection because streaklines of fluid field do not collide. Based on this consideration, we developed a simple front-tracking algorithm and introduced an adaptive refinement method to reduce subdivision cost. We further ported the entire algorithm onto a GPU and succeeded in achieving realtime performance. We also showed that our method can be used for marbling design, shape design, flash animation and target-driven design. We believe that our method opens up a new opportunity for vector artists. However, although our prototype is interactive, the running time of our method increases sharply as the contour stretches. In future work, we would like to modify our method to increase its ability to deal with more complex scenes.

References

ACAR, R., AND BOULANGER, P. 2006. Digital marbling: A multiscale fluid model. *IEEE Transactions on Visualization and Computer Graphics 12*, 4, 600–614.

BROCHU, T., AND BRIDSON, R. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing 31*, 4, 2472–2493.

EDEN, A. M., BARGTEIL, A. W., GOKTEKIN, T. G., EISINGER, S. B., AND O'BRIEN, J. F. 2007. A method for cartoon-style rendering of liquid animations. In *GI '07: Proceedings of Graphics Interface 2007*, ACM, New York, NY, USA, 51–55.

ENRIGHT, D., FEDKIW, R., FERZIGER, J., AND MITCHELL, I. 2002. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys 183*, 83–116.

ENRIGHT, D., LOSASSO, F., AND FEDKIW, R. 2005. A fast and accurate semi-lagrangian particle level set method. *Comput. Struct. 83*, 6-7, 479–490.

FATTAL, R., AND LISCHINSKI, D. 2004. Target-driven smoke animation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA, 441–448.

GLIMM, J., GROVE, J. W., LI, X. L., SHYUE, K.-M., ZENG, Y., AND ZHANG, Q. 1998. Three-dimensional front tracking. *SIAM J. Sci. Comput. 19*, 3, 703–727.

GRAND, S. L. 2007. *GPU Gems 3, Broad-Phase Collision Detection with CUDA*. Addison Wesley.

HARADA, T., KOSHIZUKA, S., AND KAWAGUCHI, Y. 2007. Smoothed particle hydrodynamics on gpus. In *Proc. of Computer Graphics International*, 63–70.

HARLOW, F. H., AND WELCH, E. J. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids 8*, 12, 2182–2189.

HIRT, C. W., AND NICHOLS, B. D. 1981. Volume of fluid /vof/ method for the dynamics of free boundaries. *Journal of Computational Physics 39* (January), 201–225.

KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1988. Snakes: Active contour models. *International journal of computer vision 1*, 4, 321–331.

MCGUIRE, M., AND FEIN, A. 2006. Real-time rendering of cartoon smoke and clouds. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, 21–26.

MCINERNEY, T., AND TERZOPOULOS, D. 2000. T-snakes: Topology adaptive snakes. *Medical Image Analysis 4*, 2, 73–91.

MOKBERI, E., AND FALOUTSOS, P. A particle level set library.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 154–159.

MÜLLER, M. 2009. Fast and robust tracking of fluid surfaces. In *SCA '09: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, 237–245.

OSHER, S., AND SETHIAN, J. A. 1988. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *J. Comput. Phys. 79*, 1, 12–49.

PUCKETT, E. G., ALMGREN, A. S., BELL, J. B., MARCUS, D. L., AND RIDER, W. J. 1997. A high-order projection method for tracking fluid interfaces in variable density incompressible flows. *J. Comput. Phys. 130*, 2, 269–282.

SELLE, A., MOHR, A., AND CHENNEY, S. 2004. Cartoon rendering of smoke animations. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, 57–60.

TRYGGVASON, G., BUNNER, B., ESMAEELI, A., JURIC, D., AL-RAWAHI, N., TAUBER, W., HAN, J., NAS, S., AND JAN, Y. 2001. A front-tracking method for the computations of multiphase flow. *Journal of Computational Physics 169*, 2, 708–759.

WITTING, P. 1999. Computational fluid dynamics in a traditional animation environment. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 129–136.

WOJTAN, C., THÜREY, N., GROSS, M., AND TURK, G. 2009. Deforming meshes that split and merge. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, ACM, New York, NY, USA, 1–10.

WOO, M., NEIDER, J., AND DAVIS, T. 1997. *OpenGL programming guide (2nd ed.): the official guide to learning OpenGL version 1.1*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.