CGI2016 manuscript No.
(will be inserted by the editor)

Dynamic Lapped Texture for Fluid Simulations

Jonathan Gagnon · François Dagenais · Eric Paquette

Abstract We present a new approach for texturing fluids. Particle *trackers* are scattered on the surface of the fluid, and used to track deformations and topological changes. For every frame of the animation, the trackers are advected and rotated coherently with the flow of the fluid. Receiver polygons are identified on the fluid surface and are used to transfer *uv* coordinates, while ensuring a controllable amount of texture distortion. The density of the trackers is adjusted when constructing a texture atlas used for rendering. Trackers that remain unused when filling the atlas are safely removed, while texels of the atlas without any corresponding tracker identify areas where new trackers will be added. Together with our patch layering approach, this tracker creation and removal process reduces popping artifacts. Both the input (fluid surface mesh and velocity field) and the output (texture atlas) of our approach make it easy to integrate into a typical production pipeline. We tested our approach on several types of fluid simulation scenarios, including splashes, rotational flows, and viscous fluids. The resulting animations of textured fluids are free from temporal artifacts and popping, and show a limited amount of distortion, blurring, and discontinuity.

Keywords Fluid Animation · Texture Synthesis

J. Gagnon[1,2]
jonathangagnon@gmail.com

F. Dagenais[1]
francois.dagenais.2@ens.etsmtl.ca

E. Paquette[1]
eric.paquette@etsmtl.ca

[1]Multimedia Lab, École de technologie supérieure, Montreal, Canada
[2]Mokko Studio, Montreal, Canada

1 Introduction

Several types of liquids require texturing: mud, foam, and soiled water to name a few. The typical surface parametrization method using *uv* coordinates performs well when texturing rigid or partially deformable objects. Nevertheless, for liquids, the surface can undergo severe distortions that lead to disturbing results if relying on basic *uv* coordinates advection. Current methods for texturing fluids [1,6,9,11,20] rely on either sprite-based advection or texture synthesis. Methods from both of these classes have several limitations and result in spatial and temporal artifacts. When the fluid surface extends or shrinks, disturbing blurring or popping artifacts become visible as texture patterns appear or vanish. Moreover, the global structure of the pattern does not deform accurately when subjected to a rotational flow, which breaks the illusion of a continuous surface pattern. Finally, changes in topology and splashes often result in distracting flickering.

While sprite-based methods have their limitations, they fit well within a conventional visual effects pipeline and are much less prone to popping and jittering artifacts. Moreover, in a typical visual effects context, high-resolution textures will be used. Compared to texture synthesis, the computation times of sprite-based methods will increase at a lower rate as the resolution of the texture increases.

To overcome the limitations of sprite-based methods, we propose an approach extending the lapped texture method to fluids. Our approach tracks the surface changes and updates the texture using overlapping patches. Although the proposed approach shares some similarities with sprite-based methods, it brings forward innovative improvements. First of all, our method tracks the surface movement with both particle trackers

and local coordinate frames, which increases the precision and eases the texturing of rotational flows. We also present an entirely new approach to maintain an appropriate distribution over the surface, while preventing popping artifacts as well as allowing thin splashes and drops to be textured in a realistic manner. Finally, the proposed method fills a texture atlas with the texture exemplars, making it easy for the textured fluids to be integrated into a typical rendering pipeline. To summarize, the contributions of the proposed approach are:

- an atlas-based texture synthesis;
- a distortion control over the receiver polygons identification and atlas filling;
- an atlas coverage tracker distribution update;
- a tracker plus local frame advection.

Our expandable texture synthesis approach is easily amenable to texturing with several exemplars at the same time, and to exemplars evolving over time. Compared with previous methods, this new approach provides results with increased realism for texturing fluids with rotational flows, high-curvature areas, and splashes.

2 Related Work

Example-based texture synthesis is often used to create large textures from smaller texture exemplars. We look at two main approaches to synthesizing a texture: using texels and using patches. Texel-based methods synthesize one texel at a time [4], using a window of neighbor texels to identify the best match from the texture exemplar. Even with improvements relying on Gaussian pyramids [16] and optimizations [7], it remains time-consuming, especially when considering larger input or output textures.

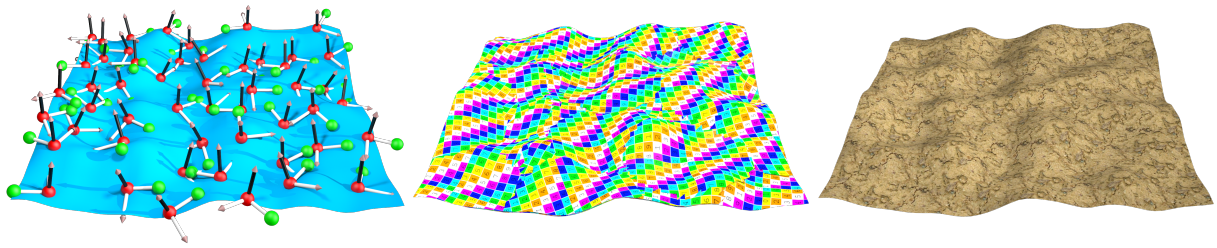
Instead of computing and copying colors on a per-texel basis, other methods [3,8,18] copy several adjacent texels at once. With this set of texels, the synthesis corresponds to juxtaposing patches. As patch-based methods do not optimize texel colors individually, they are often augmented with methods to improve the color transition between patches, such as minimum error boundary cut [3,8] or feature maps [18].

The methods discussed so far are designed to synthesize textures on a flat surface. When considering curved surfaces, there are two main ways to synthesize a texture: using vertices or using patches. Vertex-based methods synthesize a texture using vertex colors and vertex neighborhoods [15,17]. Patch-based methods use overlapping patches to synthesize a texture on a curved surface [13,14]. These methods perform well with static surfaces.

The surface of a simulated fluid goes through deformation, distortion, and topological changes, making it difficult to texture map. The advection of vertex colors was used to track these changes [1,6,9–11]. Surface tracking [2] can be used in a similar manner to determine the movement of the texture. For both advected vertices and surface tracking methods, a large amount of stretching is introduced, the global texture patterns quickly diverge, and these do not provide an adequate solution for texturing newly exposed areas. In a similar fashion as for texel-based methods, it is possible to correct the vertex colors by finding best match colors in the exemplar [1,6]. The method of Bargteil et al. [1] can follow the optical flow, but it does not handle rotational flows: the features of the texture do not rotate. Kwatra et al. [6] correct this problem by advecting orientations. However, according to Yu et al. [19], the resulting optical flow of these methods [1,6] does not show a very accurate match with the input flow. Moreover, these methods [1,6] introduce popping artifacts. Narain et al. [10] use a parameter map, which allows for a better similarity with the optical flow, but still has popping artifacts. According to Jamriska et al. [5], the pixel-based synthesis can also suffer from a wash-out effect: after a few frames, the resulting texture converges to blurred local minimum. While the method of Jamriska et al. [5] is free from the wash-out effect and improves the texture flow, when subject to a rotational flow the texture patterns have a tendency to locally retain their orientation, especially at the center. Furthermore, the method relies on 2D motion flows instead of 3D free surfaces. Finally, the texture exemplar used with these vertex-based methods are often quite small. Using high-definition exemplars with texel-based methods significantly increases computation times.

To improve the texture flow in accordance with the fluid flow, the patch-based approach of Yu et al. [20] proposes to use deformable overlapping patches. This approach allows the use of large texture exemplars without increasing the computation time thanks to the patch-based strategy. It also corrects distortion and popping artifacts, but suffers from the blurring.

While tracking deformations, fluid texturing methods need to deal with changes in topology, as well as shrinking and expanding surfaces. All of these changes affect the distribution of points tracked on the surface. Homogeneous scattering with Poisson disk distribution [20] or with point repulsion [1,6] can introduce a loss of details; on features such as splashes and thin threads, the target distribution can be achieved without generating a sufficient number of trackers to cover all sides of these thin features.



(a) Trackers and local frames (b) Receiver polygons (c) Atlas creation from layered receiver polygons

Fig. 1 Overview of our approach.

Vertex-based methods [1,6] track deformations using advected colors on the vertices of the fluid surface, while deformable grids are used for the patch-based method of Yu et al. [20]. Although vertex-based methods can handle free surface meshes, the patch-based method of Yu et al. [20] has been developed for 2D and $2\frac{1}{2}$D fluid simulations. Therefore, to our knowledge, there is no precise tracking method for patch-based texture synthesis on free surfaces. Compared to previous methods, our approach introduces a new precise patch tracking on free surfaces. It also proposes a coherent patch distribution update allowing topological changes. It performs well with complex fluid movements involving thin features, splashes, and rotational flows.

3 Dynamic Lapped Texture

To texture the surface of a fluid, the user provides an input texture exemplar and a mask, together with the per-frame tessellated surface and the velocity field supplied by any fluid simulator. Our approach uses these to output a texture atlas that can be processed through any typical rendering pipeline and tools. The approach consists of three main phases as outlined in Fig. 1. Trackers (particles) are first advected and their local frames updated according to the velocity field of the fluid and the tessellated surface (see Sec. 3.1). In the second phase, receiver polygons are identified around each tracker (see Sec. 3.2), and the last phase computes the texture atlas using an overlapping principle (see Sec. 3.3).

3.1 Tracker and Local Frames Advection

The first phase of our approach is the local frame advection process, which is split into three steps: advecting the trackers, updating the local frames, and updating the tracker distribution.

3.1.1 Advecting Trackers

Our tracker advection brings together advantages from the Poisson disk advection of Yu et al. [20] and the orientation advection of Kwatra et al. [6]. We advect trackers in the same fashion as Yu et al. [20]. Similarly to Kwatra et al. [6] who use orientation vectors on mesh vertices, we use an orientation per tracker. The input to the tracker advection is the tessellated surface of the fluid and the velocity field of the fluid simulation (either through particles or a grid). These properties are straightforward to extract from most fluid simulators. In our approach, each tracker particle P_i is on the surface of the fluid. For each frame, tracker P_i is advected to P_i' based on the fluid simulation. This process requires the velocity, which we store at the vertices of the surface to avoid transferring the whole velocity field. Errors tend to accumulate if the advection uses a low-order scheme. To maintain accurate results, especially for turbulent fluids, we use a fourth-order Runge-Kutta integration. After advection, each tracker P_i' is projected toward the closest location on the fluid surface (P_i'').

3.1.2 Updating Local Frames

For each tracker, we use a *tangent tracker* (T_i) to create a local coordinate frame defining the orientation of the patch (see Fig. 2). Unlike P_i'', tracker T_i' is not projected onto the surface. The local frame is updated using the advected trackers P_i'' and T_i'. The orthonormal basis is defined from the normal $\overrightarrow{n_i}$ of the tracker and the binormal $\overrightarrow{b_i} = \overrightarrow{n_i} \times P_i''T_i'$. The position of the tangent tracker is then adjusted to T_i'' such that $P_i''T_i''$ is perpendicular to $\overrightarrow{n_i}$ and $\overrightarrow{b_i}$. The local frame orientation is important as it allows a better similarity with the velocity field. As it can be seen in the accompanying video, if we remove the orientation, the result looks unrealistic, especially in rotational flows.

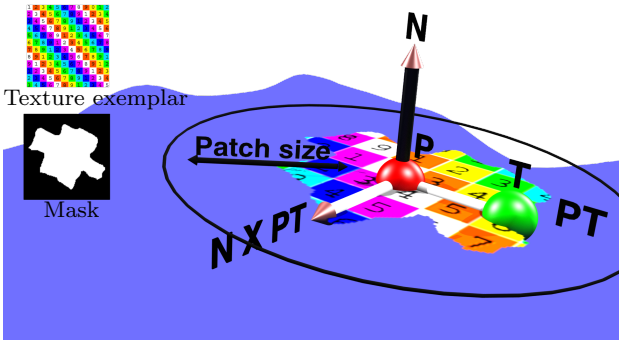


Fig. 2 We track the flow of the fluid with the tracker (in red), the tangent tracker (in green), and the surface normal (in black).

3.1.3 Updating Trackers Distribution

Initially, a Poisson disk sampling is done to cover the surface with a uniform distribution of trackers. The radius of the Poisson disk matches the patch size set by the user. Previous methods [1,6,20] use a density approach to determine if it is possible to add or remove Poisson disks to preserve a homogeneous distribution. Although this works well on flat and moderately curved surfaces, it could be difficult to achieve the target distribution while having enough trackers to cover thin threads or splashes.

Our solution ensures that the surface is completely covered by patches with the relation between the patches and the texture atlas used for rendering (see Sec. 3.3). When rasterizing the receiver polygons to the atlas, we identify uncovered areas of the surface by finding the texels that are mapped to a triangle, but that do not get colored. A random distribution of trackers is done on the uncovered areas of the surface. This process is repeated until every atlas texel mapped to a triangle is colored. The appearance of new patches happens only where newly uncovered areas are exposed. Furthermore, the new patches are layered below older ones through the use of a layering number. Thus, the new patches affect only the newly uncovered areas, leaving the already covered areas free of any popping artifacts.

The distribution update also involves finding the patches that are fully concealed. Each patch that remains unused when filling the atlas is not visible and thus removed. A sink is a particular case where the patch on top is likely to stop moving, but will never disappear because of our layering scheme. To circumvent this, we apply an additional test where we measure if velocity vectors at the four corners of the patch are oriented toward the center of the patch. Such patches are temporally deleted by blending their masks, at a rate proportional to the speed of the inward-pointing velocity vectors.

3.2 Patch-to-Receiver-Polygons Parametrization

The second phase identifies the polygons of the fluid surface that will receive the properties from each patch. This phase is performed in two steps. First, we identify the polygons based on the tracker position and the local frame orientation. In the second step, the uv coordinates and layering of the patches are transferred to the receiver polygons through an orthogonal projection. This process is repeated for each patch and results in a list of uv coordinates and layering IDs for each vertex of the fluid surface.

3.2.1 Receiver Polygons

The potential receiver polygons are those around tracker P_i within a distance equal to the patch size. From this set of polygons, we reject those not connected to the polygon closest to the tracker; this avoids selecting polygons that are nearby, but from a disjoint part of the liquid. Since flattening a closed or excessively curved surface introduces too much distortion, we exclude polygons pointing away from the local frame's normal with a threshold angle $\phi_{\max}$. For the examples presented in this paper, we exclude polygons with an angle larger than $\phi_{\max} = 130°$. This is a user-controllable parameter, enabling a compromise between the amount of distortion and the number of trackers required to cover the surface. As we reject more polygons, more trackers will be required to completely cover the surface.

When defining a parametrization between a curved and a flat surface, there are often remaining distortions. To address this issue, we use an alpha falloff inspired by the method of Praun et al. [13]. As we rely on an orthogonal projection (described in Sec. 3.2.2) , the distortion of the texture increases as the angle θ between the normal of the polygon vertices and the normal of the local frame increases. Thus, receiver polygon vertices with an angle $\theta < \psi_{\alpha=1}$ are set as opaque, whereas polygon vertices with an angle $\theta > \psi_{\alpha=0}$ are set as completely transparent. Linear interpolation of the alpha value is used when $\psi_{\alpha=1} < \theta < \psi_{\alpha=0}$.

The angle thresholds should be set so that $\psi_{\alpha=1} < \psi_{\alpha=0} < \phi_{\max}$. These thresholds are controllable parameters which can be used to reduce distortion if required. The receiver polygon identification and the alpha blending are not time-consuming, produce satisfying results (see Fig. 3), and are smooth over time as can be seen in the accompanying video.

Fig. 3 Texturing a viscous fluid using overlapping patches. Visualization of the patches using a single random color per patch.

3.2.2 Patch Properties Transfer

Given a specific patch on the surface of the fluid, we transfer its properties to the receiver polygons of the tessellated surface. Each vertex of the receiver polygons is orthogonally projected on the supporting plane of the patch, along the direction of the local frame's normal. The corresponding locations in the patch texture space are computed based on the patch size and the projection on the patch local frame. The *uv* coordinates from the local frame and the layering IDs of the patch are then added to the per-vertex list of properties. A texture ID could also be stored, should we want to use multiple texture exemplars. The properties transferred from the patch to the atlas is typically the color, but it could be any other texture properties, for example a normal map or a displacement map.

3.3 Dual-Purpose Texture Atlas

The last phase is the preparation of a texture atlas, a general representation enabling the use of any rendering tool. Surface splatting [22] could be used to render the patches instead of using our atlas construction. While splatting would require a dedicated shader, relying on the atlas has the advantage that it can be used with any standard renderer and shader, making it much more easy to integrate in the set of shading and lighting tools used in a visual effects studio. The atlas is created in two steps: (1) the surface of the fluid is unwrapped to the texture space of the atlas, and (2) the texels of the atlas are filled based on the patch *uv* and layering ID stored with the fluid surface (see Alg. 1). For the first step, several mesh unwrapping methods could be used. In our implementation, we rely on SideFX^{TM} Houdini's "UV Unwrapping" operator. In our experience, the unwrapping does not even have to be temporally coherent. Strong temporal inconsistencies in the unwrapping will still provide a temporally-coherent rendering of the patches.

After unwrapping, each polygon of the fluid surface has a corresponding location in the atlas texture space. In the second step, the layered texture patches and their masks are combined, yielding the atlas texel values. For each texel of the atlas, the corresponding polygon is fetched together with its list of layering IDs and texture exemplar *uv* coordinates. The layered patches are handled from bottom to top. The texture exemplar colors are accumulated, considering the mask color and the alpha falloff. The atlas rendering is repeated for every frame of the animation.

In highly curved regions, alpha falloff is used, as described in Sec. 3.2.1. In this case, if none of the patches affecting this texel are opaque, a patch is added below in the patch distribution update step. This process ensures the surface is fully covered with patches even if we use alpha blending.

```
Unwrapped polygons = unwrap(fluid surface)
foreach Unwrapped polygon do
    foreach Texel of Atlas in Unwrapped polygon do
        TColor = null
        foreach Patch in layered order do
            uv_E =
              AtlasToExemplarCoords(uv_A, Patch)
            EColor = Exemplar(Patch, uv_E)
            M_α = Mask(Patch, uv_E)
            Falloff_α =
              FalloffInterp(Unwrapped polygon)
            TColor = (1 − Falloff_α) TColor+
                     Falloff_α M_α EColor
```

Algorithm 1: Per-frame atlas creation.

4 Results

To validate our approach, we tested a variety of fluid scenarios, as shown in Figs. 4-9, as well as in the accompanying video. Our approach works well with structural patterns. Figs. 4 and 5 show that the structure of two different texture exemplars is preserved. Compared to the method of Narain et al. [10], we do not need a parameter map in order to avoid the washout effect of the convergence to a local minimum. Since it is a patch-based method, and because of our receiver polygon identification and alpha falloff, the proposed approach introduces a limited and controllable amount of distortion.

The texture follows the velocity field in a realistic way with the help of the trackers and tangent trackers. The translations and rotations of the patches follow the

(a) (b) (c)

Fig. 4 Double dam break using a mud texture exemplar. There is only limited distortion and it is very difficult to distinguish the patches.

Fig. 5 Viscous fluid animation. Even with the slow movement, and the significant amount of topology changes, the animation is temporally coherent and does not suffer from flickering.

Fig. 6 Rotational flow: there are only a few blurring artifacts and no popping artifacts in the animation.

flow of the liquid, as shown in Fig. 6 and the accompanying video.

Liquids often carry different materials visible at their surface. It is very easy to adapt our approach to support the simultaneous use of several texture exemplars. In fact, since each has an associated texture exemplar, we can have a different texture per tracker. This way, we can reproduce complex texture such as mud, lava, or any type of fluid having small fragments of different types as illustrated in Figs. 7 and 8.

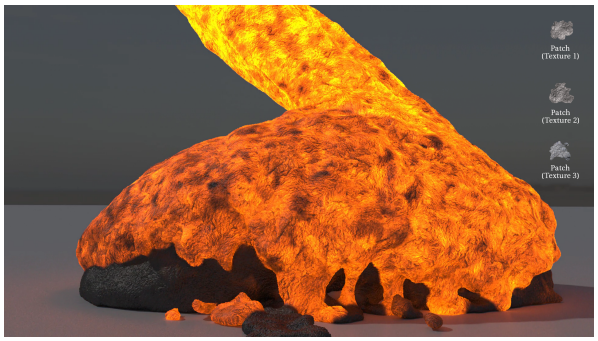


Fig. 7 Texture exemplars combined with a color shader based on surface temperature.

The lava example (Fig. 7) has been created using three different texture exemplars of melted rock. A standard FLIP simulation [21] was used to animate the lava. The fluid is initially set to a maximal temperature, and the temperature of each particle follows an exponential decay. Throughout the animation, the same exemplars are used, regardless of the lava temperature. We used a temperature interval from a minimal temperature at which the rock becomes solid to a maximal temperature when the rock is considered completely liquid. Then, to simulate solidifying rocks, the viscosity of the fluid is affected by the temperature: maximal temperature is fully liquid and minimal temperature is highly viscous. Finally, a shader modulates the color from the temperature by interpolating between black for the minimal temperature and bright yellow-orange for the maximal temperature.

When dealing with a set of multiple exemplars, the initialization and update of the tracker distribution need to take into account the selection of a specific exemplar for each new tracker. Our approach can support various exemplar selection strategies. First, the selection can be determined by regions where a specific exemplar will be selected, such as the parameter map of Narain et al. [10]. A second strategy is to rely on a user-defined discrete probability distribution among

Fig. 8 Multiple texture exemplars.

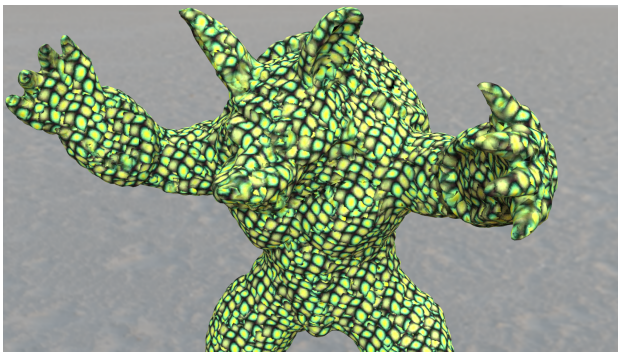


Fig. 9 With this structured pattern, its high-contrasting colors, and the designed mask, it is easier to guess where the seams between neighbor patches are located.

the set of exemplars. When a more uniform distribution is required, the selection could rely on the computation of a local histogram of the neighbor exemplar types. The exemplar for the new tracker is then selected to steer the local histogram toward the user-specified target distribution.

Our approach avoids blocky artifacts and prevents popping artifacts thanks to the layering strategy. As shown in the accompanying video, the patch distribution update is smooth. Our approach also limits blurring artifacts when updating the distribution, by only deleting concealed patches. Only two cases remain where our approach can introduce blurring artifacts: when deleting patches because of inward-pointing velocity vectors, and on receiver polygons of high curvature regions. The first case of deleting patches with inward velocity never happened for the animations related to Figs 4-9. We created a fake velocity field with velocities all pointing to a sink position. Even considering the fact that this scenario was explicitly set up to force alpha blended path deletions, only 1% of the patch deletions were done based on inward velocity vectors, introducing a limited amount of blurring artifacts. The second case where our approach can introduce blurring is when we use alpha blending based on the normal of the local frame and the normals for the receiver polygons' vertices. This affects 2% of the atlas texels on average for our examples. Overall, our animations have a very limited amount of blurring artifacts.

Since our approach is patch-based, the use of high-resolution exemplars has a negligible effect on computation time. All texture exemplars used in this paper are in high-definition (1024×1024) and include a displacement map, except for the green exemplar (64×64, no displacement map). There are between 60k and 200k polygons in the tessellated fluid surfaces of our animations. The number of trackers is between 5k and 20k, and the resolution of the atlas is between $3k \times 3k$ and $8k \times 8k$.

We used a 12-core Intel i7-3960X with 64 GB of RAM and a Quadro 4000 graphics card to run our texturing approach. All of our animations are 10 seconds long at 24 frames per second. Computation times and other statistics are illustrated in Tables 1 and 2. As can be seen in Fig. 10, the most time-consuming part is the atlas creation. This phase takes around 60% of the total time. For our examples, the whole texturing process takes between one and six minutes per frame.

Our current implementation takes advantage of parallel computation for the atlas creation, but uses a single core for the advection and the receiver polygons stages. It outputs frames every few seconds to a few minutes depending on the scene complexity. For example, the dam break of Fig. 4 takes an average of 75.5 seconds per frame. Rendering at a resolution of 1920×1080, with global illumination, displacement map, and a 5×5 supersampling, is around one minute per frame using SideFX Mantra^{TM}.

Table 1 Per frame computation times (in seconds) for the advection, identification of receiver polygons, and creation of the atlas.

	Advect	Receiver poly	Atlas	Total
Dam break (Fig. 4)	0.55	30	45	75.55
Viscous fluid (Figs. 5 & 8)	0.86	60	42	102.86
Vortex (Fig. 6)	2.2	50	75	127.5
Lava (Fig. 7)	1.25	175	180	356.25
Armadillo (Fig. 9)	0.5	15	70	85.5

5 Discussion

Even though our method outputs texture atlases roughly as fast as they can be rendered, the whole process can be time-consuming, and this could be a limitation. The input texture exemplar and manual design

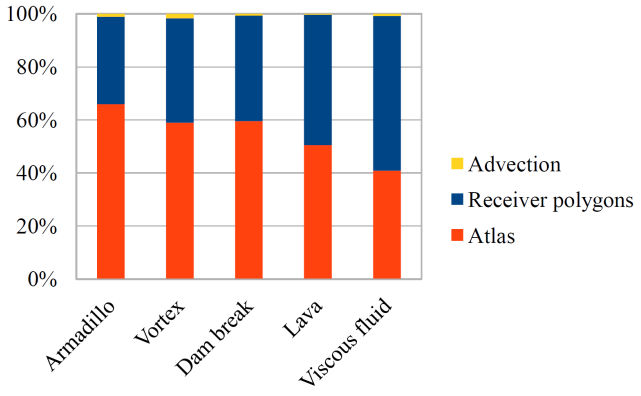


Fig. 10 Relative computation times for the advection, identification of receiver polygons, and creation of the atlas.

Table 2 Average number of polygons, number of trackers, and output resolution of the atlas in texels.

	Surface resolution	# of trackers	Output resolution
Dam break (Fig. 4)	65k	10k	3.5k × 3.5k
Viscous fluid (Figs. 5 & 8)	200k	5.2k	5k × 5k
Vortex (Fig. 6)	175k	11k	6k × 6k
Lava (Fig. 7)	260k	5k	8k × 8k
Armadillo (Fig. 9)	80k	3k	6k × 6k

of the mask influence the quality of the result. Fig. 9 shows an example with layered patches, where borders of the pattern introduce contrasting edges not found in the input. This is an inherent problem from the lapped texture method [13].

Furthermore, large patches will not follow the velocity field appropriately, and the texture will locally look rigid. In that case, it is possible to observe seams, especially with more regular texture exemplars. Moreover, large patches on high-curvature areas can introduce visible blurring artifacts and distortions caused by the orthogonal projection and the alpha falloff. Large patches on high-curvature areas can also introduce visible popping artifacts since there could be texels without any opaque color from the layered patches. This texel requires a new patch that could create a popping artifact. To avoid such a scenario, the patch size is important: using smaller patches will fit the curved surface more accurately.

6 Conclusion

We have presented an approach to texture the surface of fluids that handles any kind of topological changes including splashes. With the spatial and temporal coherences ensured by the surface trackers and local frames, we achieve patch advection with improved precision. Moreover, the update of the tracker distribution is done in the atlas process, thus ensuring that every polygon of the surface is fully covered by texels, including the polygons of splashes, and thin threads. Popping artifacts are avoided using the overlapping patches principle: new patches appear underneath existing ones in a natural way. Blurring artifacts are also avoided by deleting only patches that are concealed, once identified during the generation of the atlas. The approach works well with rotational flows, thanks to the introduction of the tangent tracker. Our texture projection is simple and efficient, and it allows for a controllable amount of distortion within the cross parametrization among the texture exemplar, receiver polygons, and texture atlas.

Deriving a method using Poisson blending [12] to hide seams in a temporally coherent manner is an interesting direction for future work. Our approach propagates patches in a spatially and temporally coherent manner throughout the animation. It would be interesting to propagate painting in the same manner by transferring surface or atlas texture edits to the patches representation. It would also be interesting to rethink the patch representation, replacing flat texture exemplars with volumetric textures. This should avoid distortions and provide a better pattern continuity, especially on high curvature areas.

7 Acknowledgements

This work was funded by Mokko Studio, NSERC, Prompt/CINQ, ÉTS, and FRQNT; We gratefully acknowledge the involvement of Danny Bergeron, president of Mokko Studio and the Mokko Studio R&D team. We thank SideFX for providing Houdini™ licences, and Hippolyte Mounier, from Photosculpt, for sharing some of the texture exemplars used in this project.

References

1. Bargteil, A.W., Sin, F., Michaels, J.E., Goktekin, T.G., O'Brien, J.F.: A texture synthesis method for liquid animations. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. ACM (2006)
2. Bojsen-Hansen, M., Li, H., Wojtan, C.: Tracking surfaces with evolving topology. ACM Trans. Graph. **31**(4), 53:1–53:10 (2012)
3. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: Proceedings of SIGGRAPH '00, Annual Conference Series, pp. 341–346. ACM (2001)
4. Efros, A.A., Leung, T.K.: Texture synthesis by non-parametric sampling. In: Proceedings of the Intl. Conf. on Computer Vision - Volume 2, ICCV '99, pp. 1033–1038. IEEE Computer Society (1999)

5. Jamriška, O., Fišer, J., Asente, P., Lu, J., Shechtman, E., Sýkora, D.: Lazyfluids: Appearance transfer for fluid animations. ACM Trans. on Graph. **34**(4) (2015)
6. Kwatra, V., Adalsteinsson, D., Kim, T., Kwatra, N., Carlson, M., Lin, M.: Texturing fluids. IEEE Trans. Visualization and Computer Graphics **13**(5), 939–952 (2007)
7. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. ACM Trans. Graph. **24**, 795–802 (2005)
8. Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: Image and video synthesis using graph cuts. ACM Trans. Graph. **22**(3), 277–286 (2003)
9. Mihalef, V., Metaxas, D., Sussman, M.: Textured liquids based on the marker level set. Computer Graphics Forum **26**(3), 457–466 (2007)
10. Narain, R., Kwatra, V., Lee, H.P., Kim, T., Carlson, M., Lin, M.: Feature-guided dynamic texture synthesis on continuous flows. In: EGSR '07 (2007)
11. Neyret, F.: Advected textures. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 147–153. Eurographics Association (2003)
12. Pérez, P., Gangnet, M., Blake, A.: Poisson image editing. ACM Trans. Graph. **22**(3), 313–318 (2003)
13. Praun, E., Finkelstein, A., Hoppe, H.: Lapped textures. In: Proceedings of SIGGRAPH '00, Annual Conference Series, pp. 465–470 (2000)
14. Soler, C., Cani, M.P., Angelidis, A.: Hierarchical pattern mapping. ACM Trans. Graph. **21**(3), 673–680 (2002)
15. Turk, G.: Texture synthesis on surfaces. In: Proceedings of SIGGRAPH '01, Annual Conference Series, pp. 347–354. ACM (2001)
16. Wei, L.Y., Levoy, M.: Fast texture synthesis using tree-structured vector quantization. In: Proceedings of SIGGRAPH '00, Annual Conference Series, pp. 479–488. ACM (2000)
17. Wei, L.Y., Levoy, M.: Texture synthesis over arbitrary manifold surfaces. In: Proceedings of SIGGRAPH '01, Annual Conference Series, pp. 355–360. ACM (2001)
18. Wu, Q., Yu, Y.: Feature matching and deformation for texture synthesis. ACM Trans. Graph. **23**(3), 364–367 (2004)
19. Yu, Q., Neyret, F., Bruneton, E., Holzschuch, N.: Scalable real-time animation of rivers. Computer Graphics Forum **28**(2), 239–248 (2009)
20. Yu, Q., Neyret, F., Bruneton, E., Holzschuch, N.: Lagrangian texture advection: Preserving both spectrum and velocity field. IEEE Trans. Visualization and Computer Graphics **17**(11), 1612 –1623 (2011)
21. Zhu, Y., Bridson, R.: Animating sand as a fluid. ACM Trans. Graph. **24**(3), 965–972 (2005)
22. Zwicker, M., Pfister, H., van Baar, J., Gross, M.: Surface splatting. In: Proceedings of SIGGRAPH 01, Annual Conference Series, pp. 371–378 (2001)