# Patch-Based Texture Synthesis on 3D Fluids

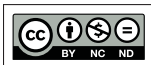
by

Jonathan GAGNON

MANUSCRIPT-BASED THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE IN PARTIAL FULFILLMENT FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
Ph. D.

MONTREAL, 5 AOUT 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC



**BOARD OF EXAMINERS**

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS:

Prof. Eric Paquette, Thesis Supervisor
Département de génie logiciel et des technologies de l'information, École de technologie supérieure

Prof. Catherine Laporte, President of the Board of Examiners
Département de génie électrique, École de technologie supérieure

Prof. Adrien Gruson, Member of the jury
Département de génie logiciel et des technologies de l'information, École de technologie supérieure

Prof. Jean-Michel Dischler, External Examiner
Laboratoire des sciences de l'ingénieur, de l'informatique et de l'imagerie, Université de Strasbourd

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON 18 JUILLET 2022

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

## ACKNOWLEDGEMENTS

This work was first funded by Mokko Studio and Digital District Canada, and then funded by FOLKS VFX. It was also funded by MITACS, Prompt, and ÉTS. We gratefully acknowledge the involvement of the former Mokko Studio and Digital District R&D team. We thank the employees and R&D team of FOLKS VFX for their feedback and involvement throughout the project. We would also like to thank SideFX for providing Houdini licenses for research. We thank Hippolyte Mounier, from Photosculpt, for sharing some of the texture exemplars used in this project.

**Synthèse de texture à l'aide de parcelles sur des fluides 3D**

Jonathan GAGNON

## RÉSUMÉ

Nous proposons trois approches différentes pour synthétiser à partir d'exemples de texture une texture sur la surface libre de fluide animée. Globalement, nos approches sont appliquées en post-traitement à une simulation de fluide. Nous avons utilisé des parcelles polygonales pour stocker les coordonnées *uv* de la texture. Nous advectons des marqueurs ou des parcelles polygonaux pour déplacer la texture avec l'écoulement du fluide. Les parcelles couvrent toute la surface du fluide à chaque image de l'animation. Nous utilisons un atlas de texture pour calculer la texture animée résultante.

Dans la première approche, les particules *marqueurs* sont dispersées sur la surface du fluide et utilisées pour suivre les déformations et les changements topologiques. Pour chaque image de l'animation, les marqueurs sont advectés et tournés de manière cohérente avec l'écoulement du fluide. Les polygones récepteurs sont identifiés sur la surface du fluide et sont utilisés pour transférer les coordonnées *uv*, tout en garantissant une quantité contrôlable de distorsion de texture. La densité des marqueurs est ajustée lors de la construction de l'atlas de texture utilisé pour le rendu. Les marqueurs qui restent inutilisés lors du remplissage de l'atlas sont supprimés, tandis que les texels de l'atlas sans tracker correspondant identifient les zones où de nouveaux marqueurs seront ajoutés. Associé à notre approche de superposition de parcelles, ce processus de création et de suppression de marqueurs réduit les artefacts de changement soudain.

Dans la deuxième approche, nous synthétisons des textures à base de parcelles temporellement cohérentes sur la surface libre des fluides à l'aide de parcelles déformables. Nous cherchons à maintenir une distribution de disque de Poisson pour la position des parcelles, et après l'advection, le critère du disque de Poisson détermine où ajouter de nouveaux parcelles et quels parcelles doivent être signalés pour suppression. La suppression tient compte du nombre local de parcelles : dans les régions contenant trop de parcelles, on accélère la suppression temporelle. Cela réduit le nombre de parcelles tout en respectant le critère du disque de Poisson. La réduction des zones avec trop de parcelles accélère le calcul et évite les artefacts de mélange de parcelles.

Dans la troisième approche, en utilisant des textures superposées combinées à des parcelles déformables, nous avons réussi à supprimer l'artefact de mélange et l'artefact rigide observés dans les méthodes précédentes. Nous restons fidèles à l'exemple de texture en supprimant les texels de parcelles déformés à l'aide d'un processus d'érosion de parcelle. L'érosion des parcelles est basée sur une carte des caractéristiques fournie avec l'exemple de texture en tant qu'entrées de notre approche. L'érosion favorise l'élimination des texels vers la frontière du parcelles ainsi que des texels correspondant à des régions plus déformées du parcelle. Là où les texels sont supprimés en laissant un espace sur la surface, nous ajoutons de nouveaux parcelles sous ceux existants. Le résultat est une texture animée suivant le champ de vitesse du fluide.

Nous montrons que nos approches fournissent de bons résultats pour de nombreux scénarios de simulation de fluides, et avec de nombreux exemples de texture.

**Mots clés:** synthèse de texture, simulation de fluide

**Patch-Based Texture Synthesis on 3D Fluids**

Jonathan GAGNON

## ABSTRACT

We propose three different approaches to synthesise a texture on an animated fluid free surface based on texture exemplars. Overall, our approaches are applied as a post-process to a fluid simulation. We used polygon patches to store texture *uv* coordinates. We advect trackers or polygon patches to move the texture along the fluid flow. The patches are covering the whole surface every frame of the animation. We use a texture atlas to compute the resulting animated texture.

In the first approach, 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.

In the second approach, we synthesize temporally coherent patch-based textures on the free surface of fluids using deformable patches. We seek to maintain a Poisson disk distribution of patches, and following advection, the Poisson disk criterion determines where to add new patches and which patches should be flagged for removal. The removal considers the local number of patches: in regions containing too many patches, we accelerate the temporal removal. This reduces the number of patches while still meeting the Poisson disk criterion. Reducing areas with too many patches speeds up the computation and avoids patch-blending artifacts.

In the third approach, using lapped textures combined with deformable patches, we successfully remove blending artifact and rigid artifact seen in previous methods. We remain faithful to the texture exemplar by removing distorted patch texels using a patch erosion process. The patch erosion is based on a feature map provided together with the exemplar as inputs to our approach. The erosion favors removing texels toward the boundary of the patch as well as texels corresponding to more distorted regions of the patch. Where texels are removed leaving a gap on the surface, we add new patches below existing ones. The result is an animated texture following the velocity field of the fluid. We show that our approaches provide good results for many fluid simulation scenarios, and with many texture exemplars.

**Keywords:** texture synthesis, fluid simulation

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

Page

## LIST OF ALGORITHMS

Page

# INTRODUCTION

This thesis is the fruit of a collaboration that started with the ÉTS and Mokko, a visual effect company. Later on this project, we included Carleton University and when Mokko closed, moved to Folks VFX with whom we finished this project.

In computer graphics, fluid simulation has been used to reproduce natural phenomena. For liquid fluids, several approaches have been developed, such as SPH (Smoothed-particle hydrodynamics) (Müller *et al.*, 2003) and FLIP (Fluid Implicit Particle) (Zhu & Bridson, 2005). We can also use SPH to simulate more complex phenomena, such as the transition from solid to liquid (Dagenais *et al.*, 2012), or the animation of snow imprints (Dagenais *et al.*, 2016). It can also be used in the medical field for blood simulation, as presented in the approaches of Dagenais *et al.* (2018b) and Dagenais *et al.* (2018a).

A common objective with fluid simulation is to have a highly detailed surface. It can be achieved by increasing the apparent resolution (Roy *et al.*, 2020, 2021). However, the most common way to add details on a surface in computer graphics is to use texture mapping.

This project started at Mokko where I was working as a research and development scientist. At that time, Mokko was working on the post-production of Riddick (2013). In that movie, there was a scene where the protagonist was fighting with an alien in a pond filled with liquid mud.. Our mandate was to simulate a muddy fluid with the 3D alien animation. For the muddy look, we wanted to use texture mapping on the fluid surface. This is where we found the limitation that was the origin of this thesis.

With the first analysis of the scene, we discovered that it was impossible to use standard texture mapping for that typical case, using the existing technologies. Indeed, the resulting fluid animation had a lot of splashes and thin films. The software we were using to simulate fluids did not handle texture mapping. Our tests on advecting *uv* coordinates and vertex colors on the

fluid surface were producing distorted textures. In the literature, the only approaches that were working in three dimensions at that time (Kwatra *et al.*, 2007; Bargteil *et al.*, 2006a; Narain *et al.*, 2007) were not giving the visual quality we were trying to reach.

**Problem statement**

In computer graphics, we use texture mapping to add details to a surface. Texture mapping is normally used on static or slightly deforming meshes. However, when the surface has substantial deformations and topological changes, texture mapping results in distorted images.

Using texture mapping to add surface details is used with the majority of assets in computer graphics. The complexity of the surfaces we use in the visual effect industry is constantly growing. To address existing and future complex texturing scenarios, we wanted to extend the knowledge on texture synthesis.

**Hypotheseses**

In the literature, there were approaches to synthesize a texture on a surface based on exemplars, but nothing providing highly detailed textures in 3D. Indeed, it is possible to use surface texture synthesis on a fluid surface using vertex color advection (Kwatra *et al.*, 2007; Bargteil *et al.*, 2006a; Narain *et al.*, 2007). However, the results are limited in term of quality of the details as these approaches all use a texture synthesis with optimization (Kwatra *et al.*, 2005) which fails to preserve the structure of the texture exemplar when it has a high-definition.

Based on that knowledge, we thought it would be interesting to investigate patch-based approaches to overcome the limitation of Kwatra *et al.* (2005) of low resolution texture exemplars. Therefore, our first hypothesis was that we could use lapped textures (Praun *et al.*, 2000) on a fluid simulation if we can advect the patches using the surface velocity field. The lapped texture hypothesis is explored in Chapter 2.

With patch-based texture synthesis, we cannot advect colors as in Kwatra *et al.* (2007). We need a way to have an efficient patch distribution over the fluid animation. The second hypothesis was that it would be possible to update the patch distribution adding new patches below existing ones where gaps were created by the fluid animation. The patch distribution hypothesis is also validated in Chapter 2.

Advecting rigid patches introduced a stiffness artifact. The third hypothesis was that we could use deformable patches to overcome stiffness in the resulting texture animation. This deformable patches hypothesis is validated in Chapter 3.

Advecting deformable patches introduced distortion, which we dealt with using temporal fading as in Yu *et al.* (2011). However, using fading to remove distorted patches introduced ghosting. The fourth and last hypothesis was that we could do a patch erosion to remove distorted patch to overcome ghosting. The patch erosion hypothesis is validated in Chapter 4.

Utility

Applying a texture on a complex surface is a recurrent problem. In the visual effect industry, having new approaches to texture fluids will allow the creation of new effects that were impossible before. For a visual effect company, having such approaches, would allow it to enhanced its service offer, and therefore making it more competitive.

**Section enumeration**

This thesis is organized as followed. First, we present the literature review as it pertains to the whole project. Then, we present the first paper that has been published in the Visual Computer: Dynamic Lapped Texture for Fluid Simulations Gagnon *et al.* (2016) in Chapter 2. In this chapter, we are adressing the limitation of popping artifact that can be seen in the appraoches

of Kwatra *et al.* (2007); Bargteil *et al.* (2006a); Narain *et al.* (2007) with the help of lapped textures advected on the free surface of the fluid.

Chapter 3, presents the second paper that has been published in the Computer Graphics Forum: Distribution Update of Deformable Patches for Texture Synthesis on the Free Surface of Fluids Gagnon *et al.* (2019). In this chapter, we address the limitation of rigid patches, from the previous chapter, with the help of deformable patches, extending the work of Yu *et al.* (2011).

This is followed by the third paper, in Chapter 4, that was also published in the Computer Graphics Forum: Patch Erosion for Deformable Lapped Textures on 3D Fluids Gagnon *et al.* (2021). In this chapter, to overcome the blending artifact, we explore the use of deformable patches and lapped textures, removing distored patches using erosion. Finally, the thesis ends with the discussion and a conclusion.

# CHAPTER 1

# LITERATURE REVIEW

In this chapter, we will review the existing approaches to fluid texture synthesis. First, we will describe texture synthesis methods in two dimensions that use *example-based texture synthesis* in Section 1.1. Then, we extend our review to two dynamic dimensional fluids in Section 1.2. To address texture synthesis in three dimensions, we are first taking a look at static *surface textures synthesis* in Section 1.3. Finally, three dimensional fluid texture synthesis is discussed in Section 1.4.

## 1.1 Example-Based Texture Synthesis

In computer graphics, example-based texture synthesis is used to create large textures from smaller texture exemplars. A texture exemplar is an image in two dimensions where each pixel is called a texel. There are two main approaches to synthesizing a texture using texture exemplars: using texels and using patches.

Texel-based methods synthesize one texel at a time (Efros & Leung, 1999; Wei & Levoy, 2000; Kwatra *et al.*, 2005). The first step of these methods is to initialize the output texture with random colors. Then, the second step is, for each texel of the output texture, to use a window of neighbor texels to identify the corresponding best match in the texture exemplar. A window of neighbor is a subset of texels in the vicinity of a texel inside both the texture exemplar and the texture to compute. The window can have a L-shape (Wei & Levoy, 2000) or a square shape (Efros & Leung, 1999; Kwatra, Essa, Bobick & Kwatra, 2005) and have different sizes, typically from 7x7 to 32x32. By definition, the best match is the texel in the texture exemplar where the window of neighbor texels has the smallest difference with the window of neighbor texels on the output texture. The bigger the window is, the more accurate the result will be. However, it does not always preserve the large scale pattern from the texture exemplar. To preserve the large scale patterns, a Gaussian pyramid can be used (Heeger & Bergen, 1995). A Gaussian pyramid is made of several resolution of a texture where the fine level is the full

resolution and the coarse level is the lowest resolution. The best match computation is done from the coarse level to the fine one. To have more accuracy, Kwatra *et al.* (2005) proposed to do multiple iterations. The main limitation of the texel-based methods is that we loose the overall structure of the texture exemplar with resolution greater than 128x128 pixels. This limitation can be overcome using patch-based methods.

Patch-based methods (Efros & Freeman, 2001; Kwatra *et al.*, 2003; Wu & Yu, 2004) copy several adjacent texels from the texture exemplar at once into the output texture. A patch is by definition a set of texels from the texture exemplar. Patch-based methods however don't search for best matches but copy a patch randomly on the output texture. The main limitation is the boundary of the patches which might be visible in the resulting texture. It can be overcome using minimum error boundary cut as in Efros & Freeman (2001) or graph cut as in Kwatra *et al.* (2003). Patch-based approaches have two main advantages. First, their simplicity. Indeed, patch-based approaches are simple to implement as they don't require multiple steps with Gaussian pyramids, best match searches and the use of different window sizes. Second, the possibility to use high-definition textures. For these reasons, the approaches presented in this thesis are using patch-bases texture synthesis.

## 1.2 Texturing 2D Fluids

The use of texture synthesis can be extended to the animated surface of a fluid. Texturing fluids is difficult, and consequently, many methods (Kwatra *et al.*, 2005; Jamriška *et al.*, 2015; Yu *et al.*, 2011) concentrate on the sub-problem of 2D fluids and flows. The texture optimization method (Kwatra *et al.*, 2005) synthesizes its results in image space, making it hard to extend to the surface of 3D fluids. While the method uses patches for the synthesis, the patches are at fixed positions, and cannot deform, which makes it harder to adapt to various types of flows. LazyFluids (Jamriška *et al.*, 2015) uses per-pixel best-match searches instead of patch-based searches. While this increases the precision, the patterns remain stiff as the search windows do not deform. Furthermore, it is also limited to synthesis in image space, and as such, is hard to extend to 3D fluids.

The patch-based method of Lagrangian texture advection (Yu *et al.*, 2011) uses deformable patches which are advected based on the flow field. A deformable patch is defined as a 2D polygonized mesh where each vertex has $UV$ coordinates pointing to the texture exemplar. To have well distributed patches over the fluid, a Poisson disk distribution is used. Patches are deformed by advecting their vertices. To avoid having too much distortion, a deformation value is computed at the patch level and when it reaches a user defined threshold it is flagged to be removed. The patches to remove are fading out on multiple subsequent frames to avoid a popping artifact. The advecttion and the removal of patches can generate holes in the patch distrubtion. Using the Poisson disk distribution, new patches can be added. Newly added patches are then fading in on multiple frames the same way removing patches are fading out. Finally, on each frame, for each texel color to compute, using the underlying patches, a blending function is used. This approach is the first one to validate the accuracy of the synthesis using the optical flow. Indeed, they compared their results with previous approaches and we can see that with their method the optical flow is closer to the velocity field. The main limitations of this approach is that it does not work with 3D fluids and the fading of distorted patches does not follow the velocity field accurately. Also, after a few seconds of simulation, it is possible to have a ghosting artifact where the features of the texture exemplar are all blended together, as illustrated in Figure 1.1. These limitations are addressed in this thesis. In Chapter 3 we extend the work of Yu *et al.* (2011) to 3D fluids and in Chapter 4 we address the issues of the fading and the ghosting artifacts.

### 1.3 Surface Texture Synthesis

The methods described in the previous sections 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 (Turk, 2001; Wei & Levoy, 2001). Vertex-based methods inherit the same limitations than the pixel-based texture synthesis: we can't use high-definition texture exemplars.

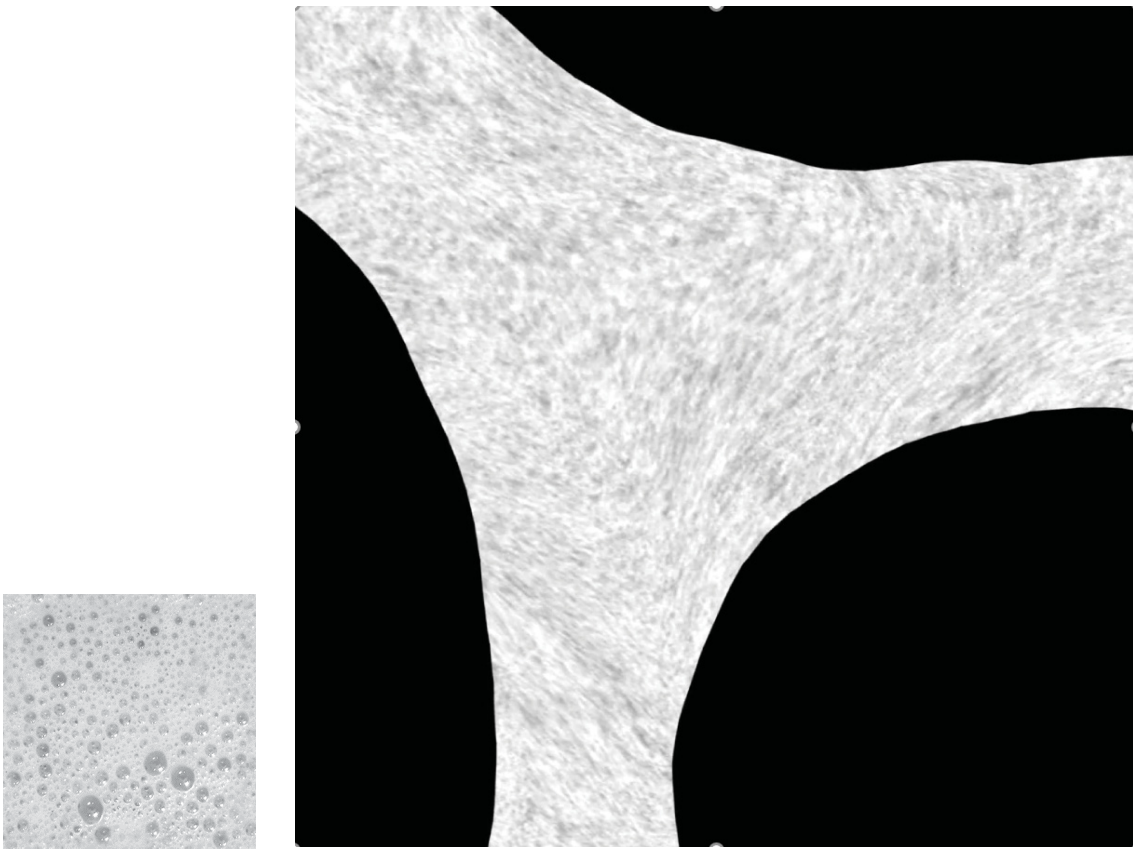



Figure 1.1 After a few seconds of simulation, using the approach of Yu *et al.* (2011), it is possible to see the blended features of the texture exemplar (see the accompanying video)

Patch-based methods in 3D use overlapping patches to synthesize a texture on a curved surface (Praun *et al.*, 2000; Soler *et al.*, 2002) in a similar fashion than in 2D with the extra step of defining patches of polygons on the 3D mesh. When a patch of polygons is selected, it needs to be flattened in order to define texture coordinates (called $UV$) in 2D. In order to avoid visible patch boundaries, lapped textures introduced by Praun *et al.* (2000) uses masks with a boundary that follows the features of the texture exemplar. These masks are a user input. When there is no relevant feature in the texture exemplar that the boundary could follow, an irregular boundary for the masks can be used. To overcome the uses of masks and avoid discontinues while adding a new patches, Soler *et al.* (2002) proposed an extra step of patch best match. These methods perform well with static surfaces, but are not designed for dynamic surfaces such as fluids. In Chapter 2 we use a similar mask approach on rigid patches, where in Chap-

ter 4 we use a dynamic mask based on the erosion of features from the boundary of the texture exemplar.

### 1.4 Texturing 3D Fluids

Few methods deal with dynamic surfaces like 3D fluids. The surface of a simulated fluid goes through deformations, distortions, and topological changes, making it difficult to texture map. Bargteil *et al.* (2006b) and Kwatra *et al.* (2007) extend the 2D texture optimization method (Kwatra *et al.*, 2005) to 3D fluids. They store the colors on the vertices of the fluid's free surface instead of in image space. The method of Bargteil *et al.* (2006b) can follow the optical flow, but it does not handle rotational flows: the features of the texture do not rotate. Kwatra *et al.* (2007) correct this problem by advecting orientations. However, as reported by Yu *et al.* (2009), the resulting optical flow of these methods (Bargteil, Sin, Michaels, Goktekin & O'Brien, 2006b; Kwatra *et al.*, 2007) does not show a very accurate match with the input flow. Moreover, these methods introduce popping artifacts. While the methods of Bargteil *et al.* (2006b) and Kwatra *et al.* (2007) can texture 3D fluids, it has the disadvantage of locking the texture resolution to the resolution of the simulation mesh; we cannot generate more or fewer individual colored points than there are vertices. Furthermore, by still relying on rigid patches for their best match search and synthesis, they need to re-sample the vertices to a square grid where they conduct the texture synthesis, and then need to re-sample in order to transfer the new colors back to the vertices. Each such resampling incurs some bluring of the features. Narain *et al.* (2007) and Bargteil *et al.* (2006c) essentially extend the work of Kwatra *et al.* (2007) to allow the use of a feature map, but their texture synthesis methods rely on the same basis as Kwatra *et al.* (2005), and consequently inherit the same drawbacks. For instance, as mentioned by Jamriška *et al.* (2015), the pixel-based synthesis is prone to suffer from a wash-out effect: after a few frames, the resulting texture converges to a blurred local minimum. The wash-out effect is illustrated in Figure 1.2 where the yellow part in the middle is extending over time, diverging from the structure of the texture exemplar. Event though the method of Jamriška *et al.* (2015) is free from the wash-out effect and improves the texture flow,

when dealing with a rotational flow, its resulting texture patterns have a tendency to locally retain their orientation, especially at the center. Moreover, Jamriška *et al.* (2015) are limited to 2D simulations. The main limitations of these approaches are the limited texture exemplar resolution, and the discrepancy between the input flow and the optical flow. Through this thesis, we address the issue of texture exemplar resolution by using a patch-based approach in Chapter 2. Then, to solve the issue of the input flow matching the optical flow, we use deformable patches in Chapter 3 and Chapter 4.

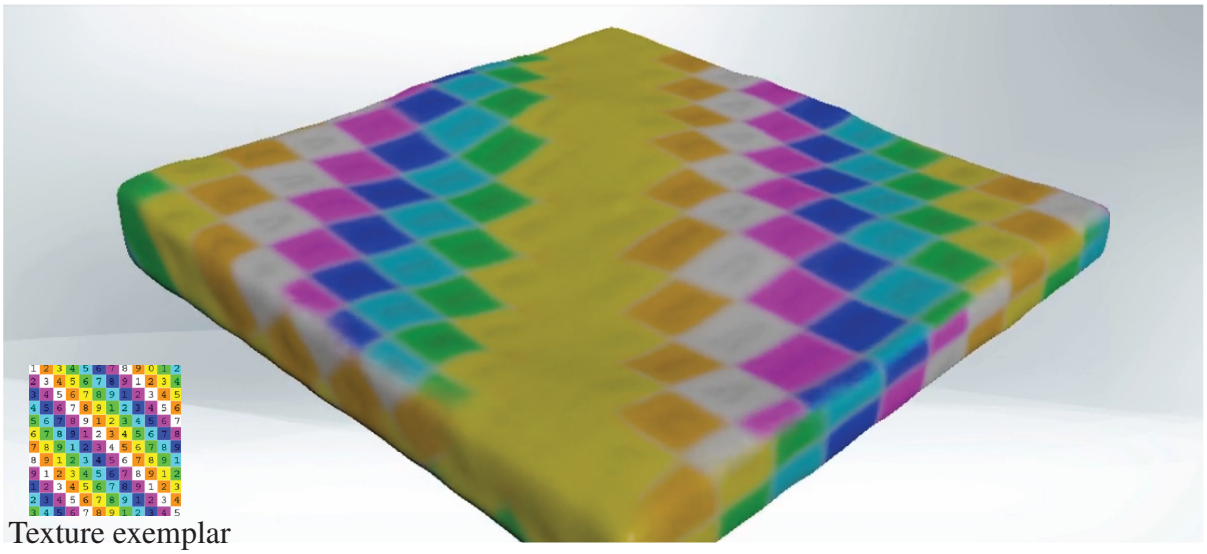



Figure 1.2 In this example of vertex-based texture synthesis using the approach of Kwatra *et al.* (2007), the wash-out artifact can be seen after 10 seconds of simulation

Recently, neural networks have been used for style transfer on 2D smoke simulation (Christen *et al.*, 2020) and later extended on volumetric fluid animation, changing also the shape of 3D fluid surfaces (Kim *et al.*, 2020). Where these methods focus on volumetric data, our approaches concentrate on surface texture synthesis.

With Gagnon *et al.* (2016), we propose a rigid patches advection approach on the free surface. This approach is described in Chapter 2. Our patch-based method does not use best-match searches, avoiding the problem of texture exemplar resolution. However, the resulting texture animation shows stiff texture patterns. To deal with the stiffness problem, with the approach

of Gagnon *et al.* (2019), presented in Chapter 3, we extend the deformable patches method of Yu *et al.* (2011) to 3D fluids. We also add a new sampling method to reduce the number of patches in order to reduce ghosting, but it was still an issue. Finally, to overcome ghosting, with the approach of Gagnon *et al.* (2021), we use the lapped texture method (Praun *et al.*, 2000) to texture fluids as in the approach of Gagnon *et al.* (2016). We also combine deformable patches texture synthesis (Gagnon *et al.*, 2019), replacing the patch fading by a feature-aware erosion method that removes distorted regions of patches. This latest approach is discussed in Chapter 4.

# CHAPTER 2

# DYNAMIC LAPPED TEXTURE FOR FLUID SIMULATIONS

Jonathan Gagnon [a], François Dagenais[b], Eric Paquette[b]

[a] Mokko Studio
[b] Département de génie mécanique, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

Published in "Visual Computer",
June 2016

## 2.1 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.

## 2.2 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 (Bargteil, Sin, Michaels, Goktekin & O'Brien, 2006b; Kwatra *et al.*, 2007; Mihalef, Metaxas & Sussman, 2007; Neyret, 2003; Yu, Neyret, Bruneton & Holzschuch, 2011) 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.

## 2.3 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 (Efros & Leung, 1999), using a window of neighbor texels to identify the best match from the texture exemplar. Even with improvements relying on Gaussian pyramids (Wei & Levoy, 2000) and optimizations (Kwatra *et al.*, 2005), they remains time-consuming, especially when considering larger input or output textures.

Instead of computing and copying colors on a per-texel basis, other methods (Efros & Freeman, 2001; Kwatra, Schödl, Essa, Turk & Bobick, 2003; Wu & Yu, 2004) 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 (Efros & Freeman, 2001; Kwatra *et al.*, 2003) or feature maps (Wu & Yu, 2004).

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 (Turk, 2001; Wei & Levoy, 2001). Patch-based methods use overlapping patches to synthesize a texture on a curved surface (Praun, Finkelstein & Hoppe, 2000; Soler, Cani & Angelidis, 2002). 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 (Bargteil, Sin, Michaels, Goktekin & O'Brien, 2006b; Kwatra *et al.*, 2007; Mihalef, Metaxas & Sussman, 2007; Narain *et al.*, 2007; Neyret, 2003). Surface tracking Bojsen-Hansen *et al.* (2012) 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 (Bargteil, Sin, Michaels, Goktekin & O'Brien, 2006b; Kwatra *et al.*, 2007). The method of Bargteil *et al.* (2006b) can follow the optical flow, but it does not handle rotational flows: the features of the texture do not rotate. Kwatra *et al.* (2007) correct this problem by advecting orientations. However, according to Yu *et al.* (2009), the resulting optical flow of these methods (Bargteil, Sin, Michaels, Goktekin & O'Brien, 2006b; Kwatra *et al.*, 2007) does not show a very accurate match with the input flow. Moreover, these methods (Bargteil, Sin, Michaels, Goktekin & O'Brien, 2006b; Kwatra *et al.*, 2007) introduce popping artifacts. Narain *et al.* (2007) use a parameter map, which allows for a better similarity with the optical flow, but still has popping artifacts. According to Jamriška *et al.* (2015), 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 Jamriška *et al.* (2015) 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.

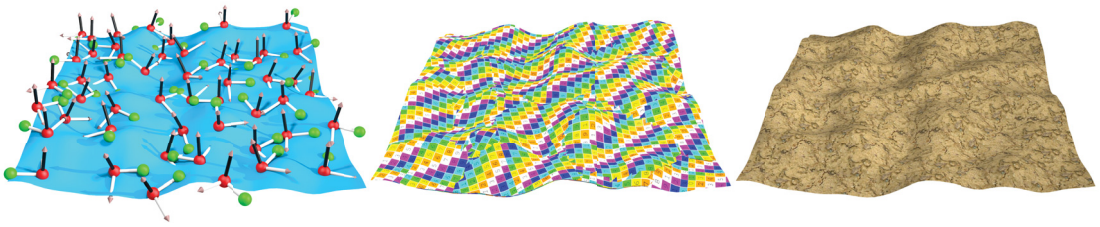
To improve the texture flow in accordance with the fluid flow, the patch-based approach of Yu *et al.* (2011) proposes to use deformable overlapping patches. This approach allows the use of large texture exemplars without increasing the computation time thanks to the patch-based strategy. It also corrects 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 (Yu *et al.*, 2011) or with point repulsion (Bargteil, Sin, Michaels, Goktekin & O'Brien, 2006b; Kwatra *et al.*, 2007) 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.

Vertex-based methods (Bargteil, Sin, Michaels, Goktekin & O'Brien, 2006b; Kwatra *et al.*, 2007) 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.* (2011). Although vertex-based methods can handle free surface meshes, the patch-based method of Yu *et al.* (2011) has been developed for 2D and $2\frac{1}{2}$D fluid simulations. Therefore, to our knowledge, there is no accurate tracking method for patch-based texture synthesis on free surfaces. Compared to previous methods, our approach introduces a new accurate 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.

## 2.4 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



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

Figure 2.1 Overview of our approach

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. 2.1. Trackers (particles) are first advected and their local frames updated according to the velocity field of the fluid and the tessellated surface (see Sec. 2.4.1). In the second phase, receiver polygons are identified around each tracker (see Sec. 2.4.2), and the last phase computes the texture atlas using an overlapping principle (see Sec. 2.4.3).

### 2.4.1 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.

#### 2.4.1.1 Advecting Trackers

Our tracker advection brings together advantages from the Poisson disk advection of Yu *et al.* (2011) and the orientation advection of Kwatra *et al.* (2007). We advect trackers in the same fashion as Yu *et al.* (2011). Similarly to Kwatra *et al.* (2007) 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 ($P_i''$).

#### 2.4.1.2 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. 2.2). 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

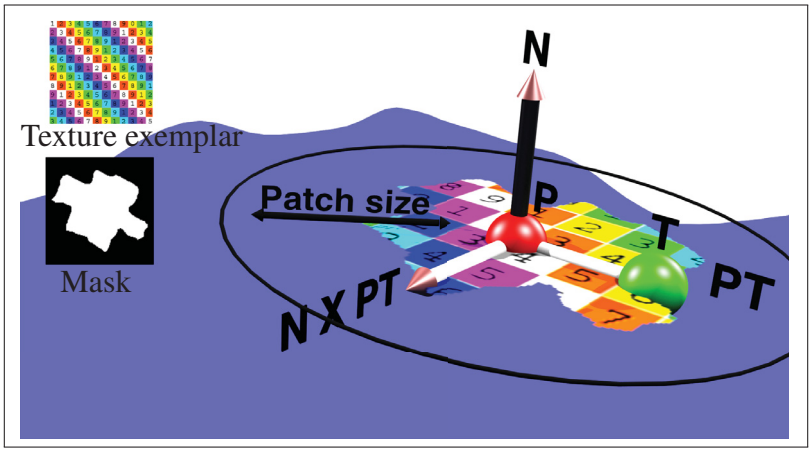



Figure 2.2 We track the flow of the fluid with the tracker (in red), the tangent tracker (in green), and the surface normal (in black)

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.

#### 2.4.1.3 Updating Tracker 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 (Bargteil, Sin, Michaels, Goktekin & O'Brien, 2006b; Kwatra *et al.*, 2007; Yu, Neyret, Bruneton & Holzschuch, 2011) 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. 2.4.3). 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.

### 2.4.2 Patch-to-Receiver-Polygon 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.

#### 2.4.2.1 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 chapter, we exclude polygons with an angle larger than $\phi_{\max} = 130^\circ$. 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.* (2000). As we rely on an orthogonal projection (described in Sec. 2.4.2.2), the distortion of the texture increases as the angle $\theta$ 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 $\psi_{\alpha=1} < \theta < \psi_{\alpha=0}$.

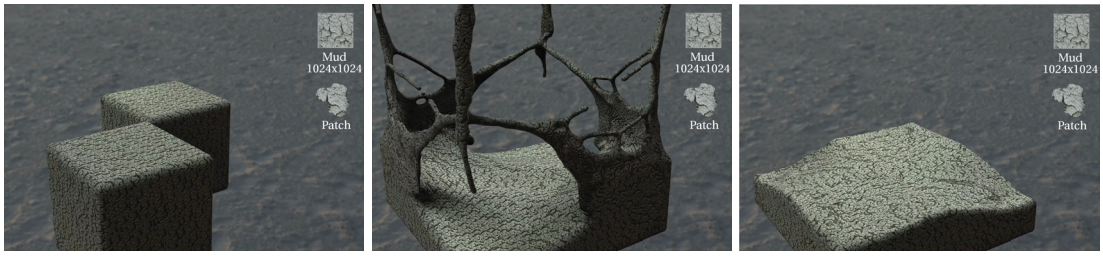
The angle thresholds should be set so that $\psi_{\alpha=1} < \psi_{\alpha=0} < \phi_{\max}$. These thresholds are controllable parameters which can be used to reduce distortion if required. The receiver polygon

identification and the alpha blending are not time-consuming, produce satisfying results (see Fig. 2.3), and are smooth over time as can be seen in the accompanying video.


Figure 2.3 Texturing a viscous fluid using overlapping patches. Visualization of the patches using a single random color per patch.

#### 2.4.2.2 Patch Property 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.




a) b) c)

Figure 2.4 Double dam break using a mud texture exemplar. There is only limited distortion and it is very difficult to distinguish the patches.

### 2.4.3 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 (Zwicker *et al.*, 2001) 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. 2.1). For the first step, several mesh unwrapping methods could be used. In our implementation, we rely on SideFX™ 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. 2.4.2.1. 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.

Algorithm 2.1 Per-frame atlas creation.

```
1  Unwrapped polygons = unwrap(fluid surface)
2  foreach Unwrapped polygon do
3      foreach Texel of Atlas in Unwrapped polygon do
4          TColor = null
5          foreach Patch in layered order do
6              uv_E =
7                AtlasToExemplarCoords(uv_A, Patch)
8              EColor = Exemplar(Patch, uv_E)
9              M_α = Mask(Patch, uv_E)
10             Falloff_α =
11               FalloffInterp(Unwrapped polygon)
12             TColor = (1 − Falloff_α) TColor+
                        Falloff_α M_α EColor
```

## 2.5 Results

To validate our approach, we tested a variety of fluid scenarios, as shown in Figs. 2.4-2.9, as well as in the accompanying video. Our approach works well with structural patterns. Figs. 2.4 and 2.5 show that the structure of two different texture exemplars is preserved. Compared to the method of Narain *et al.* (2007), 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.


Figure 2.5 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.

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 flow of the liquid, as shown in Fig. 2.6 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 textures such as mud, lava, or any type of fluid having small fragments of different types as illustrated in Figs. 2.7 and 2.8.

The lava example (Fig. 2.7) has been created using three different texture exemplars of melted rock. A standard FLIP simulation (Zhu & Bridson, 2005) 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

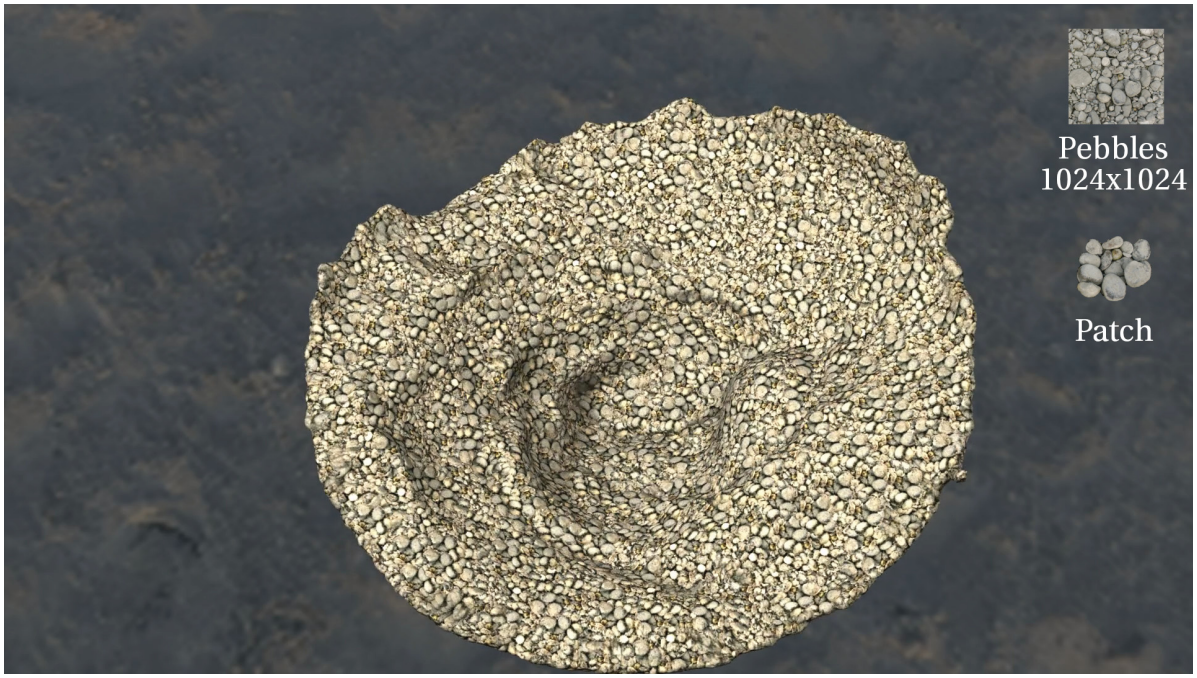



Figure 2.6 Rotational flow: there are only a few blurring artifacts and no popping artifacts in the animation

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.* (2007). A second strategy is to rely on a user-defined discrete probability distribution among 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.

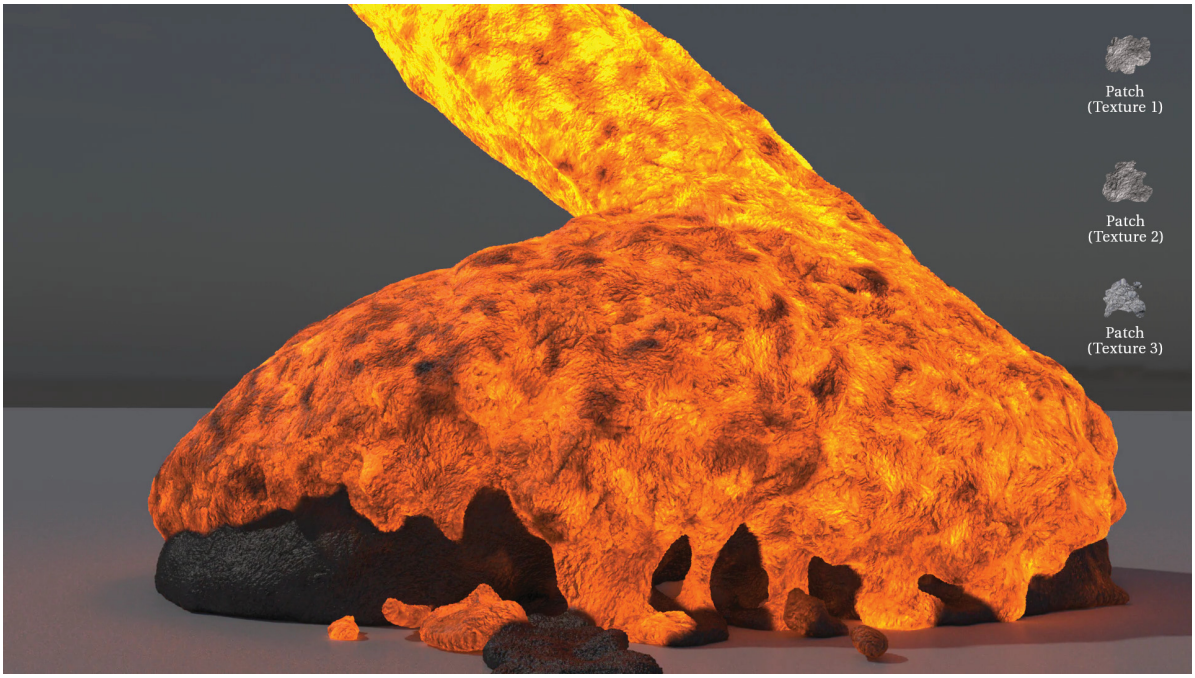



Figure 2.7 Texture exemplars combined with a color shader based on surface temperature

Figure 2.8 Multiple texture exemplars

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 2.4-2.9. 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 pacth 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 chapter are in high-definition ($1024 \times 1024$) and include a displacement map, except for the green exemplar ($64 \times 64$, 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 $3\text{k} \times 3\text{k}$ and $8\text{k} \times 8\text{k}$.

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 2.1 and 2.2. As can be seen in Fig. 2.10, 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.

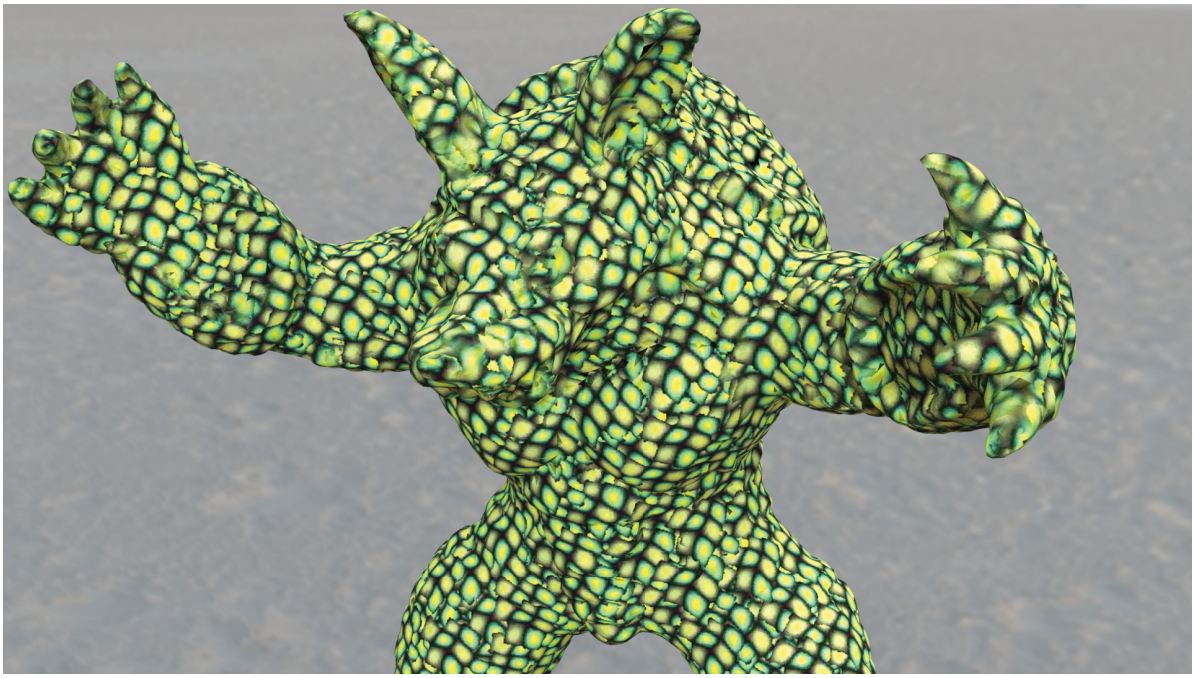


Figure 2.9 With this structured pattern, its high-contrasting colors, and the designed mask, it is easier to guess where the seams between neighbor patches are located

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. 2.4 takes an average of 75.5 seconds per frame. Rendering at a resolution of $1920 \times 1080$, with global illumination, displacement map, and a $5 \times 5$ supersampling, is around one minute per frame using SideFX Mantra$^{TM}$.

Table 2.1 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. 2.4) | 0.55 | 30 | 45 | 75.55 |
| Viscous fluid (Figs. 2.5 & 2.8) | 0.86 | 60 | 42 | 102.86 |
| Vortex (Fig. 2.6) | 2.2 | 50 | 75 | 127.5 |
| Lava (Fig. 2.7) | 1.25 | 175 | 180 | 356.25 |
| Armadillo (Fig. 2.9) | 0.5 | 15 | 70 | 85.5 |

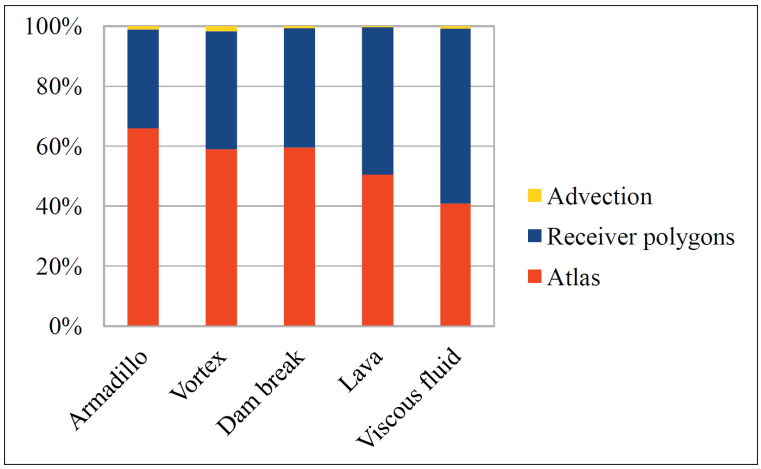



Figure 2.10 Relative computation times for the advection, identification of receiver polygons, and creation of the atlas

Table 2.2 Average number of polygons, number of trackers, and output resolution of the atlas in texels

| | Surface resolution | # of trackers | Output resolution |
|---|---|---|---|
| Dam break (Fig. 2.4) | 65k | 10k | 3.5k × 3.5k |
| Viscous fluid (Figs. 2.5 & 2.8) | 200k | 5.2k | 5k × 5k |
| Vortex (Fig. 2.6) | 175k | 11k | 6k × 6k |
| Lava (Fig. 2.7) | 260k | 5k | 8k × 8k |
| Armadillo (Fig. 2.9) | 80k | 3k | 6k × 6k |

## 2.6 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 of the mask influence the quality of the result. Fig. 2.9 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 (Praun *et al.*, 2000).

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, patch sizing is important: using smaller patches will fit the curved surface more accurately.

## 2.7 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 (Pérez *et al.*, 2003) 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.

# CHAPTER 3

# DISTRIBUTION UPDATE OF DEFORMABLE PATCHES FOR TEXTURE SYNTHESIS

Jonathan Gagnon [a], Julian Guzman [b], Valentin Vervondel [b], François Dagenais[b], David Mould[c], Eric Paquette[b]

[a] Folks VFX
[b] Département de Génie Mécanique, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[c] Caleron University

Published in "Computer Graphics Forum",
October 2019

## 3.1 Abstract

We propose an approach for temporally coherent patch-based texture synthesis on the free surface of fluids. Our approach is applied as a post-process, using the surface and velocity field from any fluid simulator. We apply the texture from an exemplar through multiple local mesh patches fitted to the surface and mapped to the exemplar. Our patches are constructed from the fluid free surface by taking a subsection of the free surface mesh. As such, they are initially very well adapted to the fluid's surface, and can later deform according to the free surface velocity field, allowing a greater ability to represent surface motion than rigid or 2D grid-based patches. From one frame to the next, the patch centers and surrounding patch vertices are advected according to the velocity field. We seek to maintain a Poisson disk distribution of patches, and following advection, the Poisson disk criterion determines where to add new patches and which patches should be flagged for removal. The removal considers the local number of patches: in regions containing too many patches, we accelerate the temporal removal. This reduces the number of patches while still meeting the Poisson disk criterion. Reducing areas with too many patches speeds up the computation and avoids patch-blending artifacts. The final step of our approach creates the overall texture in an atlas where each texel is computed from the patches using a contrast-preserving blending function. Our tests

show that the approach works well on free surfaces undergoing significant deformation and topological changes. Furthermore, we show that our approach provides good results for many fluid simulation scenarios, and with many texture exemplars. We also confirm that the optical flow from the resulting texture matches the fluid velocity field. Overall, our approach compares favorably against recent work in this area.

## 3.2 Introduction

Texture mapping is commonly used to add details to 3D surfaces. However, the texture may be distorted by surface curvature. Dynamic meshes introduce further complexity, and fluid animations are especially challenging, since the surface topology can change. Texturing a dynamic mesh is still a nontrivial problem with unanswered questions.

One category of methods applicable to fluids synthesizes the texture with a patch-based search (Kwatra *et al.*, 2007). This is very effective in maintaining the overall look of the texture exemplar. Nevertheless, it makes it much harder for the pattern to be deformed according to the flow of the fluid.

Another category of methods synthesizes the texture by advecting patches on the surface. This greatly improves the temporal coherence of the synthesized texture, which is why we decided to focus our attention on such methods. Some methods rely on "rigid" patches (Gagnon *et al.*, 2016) and have a tendency to show "blocky" artifacts, where the resulting animation feels too rigid since it does not completely respect the velocity field.

Other methods use 2D deformable grids (Yu *et al.*, 2011), and while this can be effective for deformations on mostly 2D fluids, it is not directly applicable to 3D fluids with splashes and topological changes. Small details such as splashes are also hard to handle, as it is difficult to have a good distribution of patches on elements with small surface areas. It is hard for the Poisson disk distribution techniques used by most patch-based methods to balance between the number of disks to assign to small regions and patch distortion concerns when wrapping the patches onto the fluid surface. Ultimately, concerns related to the deformation and distribution

of patches necessitate the use of small texture patches, which limits the range of applicability of earlier patch-based texture synthesis methods. Furthermore, patch-based methods face difficulties in handling converging fluid flows. While some methods provide reasonable results for 2D flows, this problem is exacerbated when considering 3D fluids, as they are prone to accumulating many patches in some areas depending on the flow and surface curvature. The source of the problem is that patches are faded at a constant rate, causing an accumulation of fading patches in regions where the fluid converges or 3D fluids merge together. Patch-based methods lead to artifacts when the density of patches is too high.

Methods relying on layered patches (Gagnon *et al.*, 2016) suffer from a degradation of the synthesized texture which is composed of many small inconsistent pieces of the exemplar. Other methods that rely on contrast-preserving blending (Yu *et al.*, 2011) are hindered by greatly reduced blending quality, as the contrast-preserving blending can lead to colors outside the $[0, 1]$ range, and the likelihood of this increases with the number of blended patches.

In contrast to previous patch-based methods, our patches are created directly on the surface of the fluid, by taking a subsection of the surface mesh. This ensures that the patches are not distorted at the time of creation; they can later deform according to the velocity field, providing greater ability to represent surface motion than rigid or 2D grid-based patches. We validated this advantage of our approach by comparing the texture's optical flow with the fluid's velocity field. To prevent patch accumulation during advection, we propose a temporal patch removal approach that considers the local number of patches: in regions containing too many patches, we accelerate the temporal removal. The opposite problem of patch accumulation consists of a lack of patches on small details, such as splashes. We improve the distribution of patches by refining the Poisson disk criterion, making it better adapted to the context of small details and patches that need to be wrapped to the fluid's surface. To this end, we use the normal of the surface and an ellipsoid form instead of a sampling sphere. The main contributions of the proposed texture synthesis method can be summarized as follows:

- Distortion reduction by creating patches from the surface;

- Dynamic fade-in and fade-out with adaptive speed related to density;
- Poisson disk criterion adapted to small features and splashes.

We obtain best results for high-frequency, stochastic, isotropic textures, as is characteristic of lapped texture synthesis. However, we do not make any assumptions with respect to the exemplar, and the results are reasonable for more structured textures. We tested our approach on many scenarios, and show that the resulting textures are temporally coherent and well adapted to the flow of the fluid.

### 3.3 Related Work

Two texture synthesis strategies are predominantly used for texturing fluids: pixel-based and patch-based (Barnes & Zhang, 2017; Wei, Lefebvre, Kwatra & Turk, 2009). Texturing fluids is difficult, and consequently, many methods (Kwatra, Essa, Bobick & Kwatra, 2005; Jamriška *et al.*, 2015; Yu, Neyret, Bruneton & Holzschuch, 2011) concentrate on the sub-problem of 2D fluids and flows. The texture optimization method (Kwatra *et al.*, 2005) synthesizes its results in image space, making it hard to extend to the surface of 3D fluids. While the method uses patches for the synthesis, the patches are at fixed positions, and cannot deform, which makes it harder to adapt to various types of flows. LazyFluids (Jamriška *et al.*, 2015) uses per-pixel best-match searches instead of patch-based searches. While this increases the precision, the patterns remain stiff as the search windows do not deform. Furthermore, it is also limited to synthesis in image space, and as such, is hard to extend to 3D fluids. The pioneering work of Stora *et al.* (1999) advected lava clinker with simple texture from noise functions. Lagrangian texture advection (Yu *et al.*, 2011) uses deformable patches which are advected based on the flow field. Even though this method works only in 2D, it investigated the problem of distortion of the texture pattern. Our approach extends the 2D deformable patches to deformable surface patches, allowing us to conduct texture synthesis on the free surface of arbitrary fluid simulations, and not only on 2D surfaces.

Few methods deal with 3D fluids. Kwatra *et al.* (2007) extend the 2D texture optimization method (Kwatra *et al.*, 2005) to 3D. They store the colors on the vertices of the fluid's free surface instead of in image space. While their method can texture 3D fluids, it has the disadvantage of locking the texture resolution to the resolution of the simulation mesh. Furthermore, by still relying on rigid patches for their best match search and synthesis, they need to resample the vertices to a square grid where they conduct the texture synthesis, and then need to resample in order to transfer the new colors back to the vertices. Narain *et al.* (2007) and Bargteil *et al.* (2006c) essentially extend the work of Kwatra *et al.* (2007) to allow the use of a feature map, but their texture synthesis methods rely on the same basis as Kwatra *et al.* (2005), and consequently inherit the same drawbacks. Similar to Lagrangian texture advection (Yu *et al.*, 2011), the dynamic lapped texture method (Gagnon *et al.*, 2016) uses patches on the surface of the liquid. This method works in 3D and provides reasonable results for scenarios including splashes. However, it has two main drawbacks. First, the patches do not deform with respect to the flow of the 3D liquid, and instead, only deform to conform to the surface of the fluid. Second, the method uses layered patches, as opposed to a contrast-preserving blending. The layering has the disadvantage of showing many small and inconsistent pieces of the exemplar in regions where many patches accumulate because of the temporal removal.

Our goal is to take advantage of the best strategies from patch-based texture synthesis methods. We move away from 2D synthesis methods, and while our approach can still handle 2D fluids, it is built for 3D fluids. We improve upon the deformable patches of Lagrangian texture advection (Yu *et al.*, 2011), while also using the 3D patch advection of dynamic lapped textures (Gagnon *et al.*, 2016). Given our new deformable patches, our texture deformation better follows the fluid flow than the non-deforming patch-based search methods (Bargteil, Sin, Michaels, Goktekin & O'Brien, 2006c; Kwatra *et al.*, 2007; Narain *et al.*, 2007).

## 3.4 Overview

In this section, we present an overview of the proposed approach. Key concepts are illustrated in Fig. 3.1.

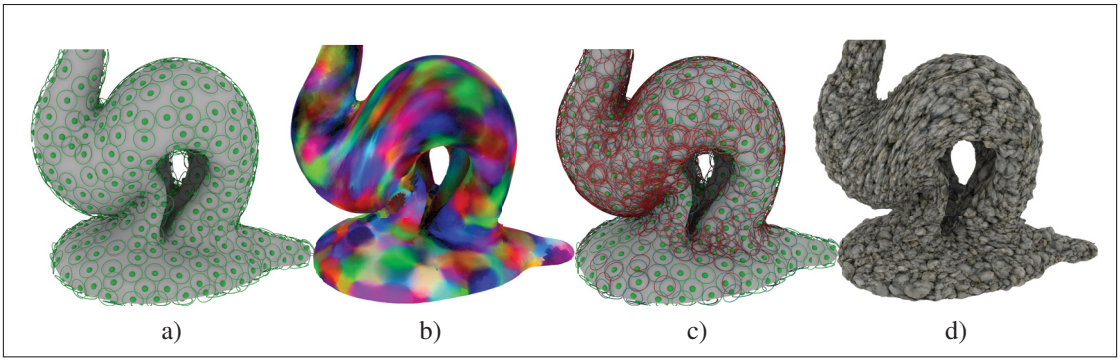



Figure 3.1 Key concepts of the proposed approach on a single frame. (a) Initial Poisson disk distribution, as if this frame would be the fist of the animation. (b) Visualization of deformable patches, after 123 frames of animation, where each patch is assigned a different color. (c) Patch distribution update after 123 frames of animation; green disks are kept, red disks will be removed, blue disks are added. (d) The final synthesized texture.

We start with an animated mesh, where each vertex has a velocity vector, usually arising from a fluid simulation. The goal is to use patch-based synthesis to synthesize a texture over the mesh. The first step of the approach is to sample the surface with a Poisson disk distribution on the first frame of the animation, as illustrated in Figure 3.1a. Next, we create a *deformable patch* which contains *uv* coordinates for each Poisson disk based on the mesh vertices surrounding it. For each patch and each patch vertex, we compute a contribution weight that will be used to compute the color of each texel.

In subsequent frames of the animation, we advect the Poisson disks and the vertices of the deformable patches according to the velocity field. To minimize distortion, we repair the overall texture by updating the patch representation: we remove patches with excessive distortion, eliminate patches in areas that are too crowded, and add new patches where there are gaps on the surface, as illustrated in Figure 3.1c. The new patch distribution is used to compute the texture for the current frame. Once the weights have been computed, we can finalize the texture synthesis by calculating, for each texel, a final color, which is a weighted combination of the contributing patches, as illustrated in Figure 3.1d. Patch removal and addition are not instantaneous. Rather, patches fade out (or in) over time at a rate determined by the local patch density. Details are provided in Sec. 3.7.3.

Our approach is inspired by the method of Yu *et al.* (2011), which we extend to work on free surfaces. The differences between our approach and that of Yu *et al.* (2011) are listed in Table 3.1.

Table 3.1 Comparison between our approach and that of Yu *et al.* (2011)

| **Yu *et al.* (2011)** | **Our approach** |
|---|---|
| Poisson Disk sampling in 2D | Poisson Disk sampling in 3D |
| 2D grid of vertices | 3D set of vertices |
| Orthogonal *uv* projection | *uv* flattening (Lévy *et al.*, 2002) |
| 2D advection | 3D advection on free surface |
| No topological changes | Handling topological changes |
| Linear patch fading | Dynamic patch fading |

A Poisson disk distribution is not designed to handle details smaller than the Poisson disk radius. Therefore, we propose a solution using the plane of the surface's normal, combined with an ellipsoid volume per patch as the Poisson disk criterion, which is discussed in Sec. 3.5. Creating deformable patches is tricky when we have a curved surface and complex mesh topology. The creation of deformable patches is discussed in Sec. 3.6. Deformable patches need to be updated over time in a consistent fashion in order to preserve their texture exemplar's features spatially and temporally. We fade in and out patches at a rate influenced by density of patches; details of the entire approach are discussed in Sec. 3.7.

## 3.5 Distribution on Curved Surfaces

This section describes our approach to surface sampling. We want to avoid two problematic conditions: areas of the surface with no patches, and areas on the surface with too many patches. We sample the surface with a Poisson disk distribution on the first frame. On subsequent frames, we compute a new Poisson disk distribution using the advected disks as a starting point. This allows us to check if it is possible to add new patches or if we need to delete some of them.

There are multiple Poisson disk sampling methods available; some of these work in 2D, and others in 3D. For our purpose, we need to sample curved surfaces, and we want to be able to distribute more samples in high curvature areas, such as splashes or thin threads. These concerns are important since we do not want our patches to deform too much when wrapping on the fluid surface, as this would cause unwanted texture stretch. As shown in Fig. 3.2a, when dealing with nearby surfaces, and only considering a 3D Poisson radius $r$, samples on one surface count toward the Poisson criterion of the other surface, leading to an undesired undersampling.

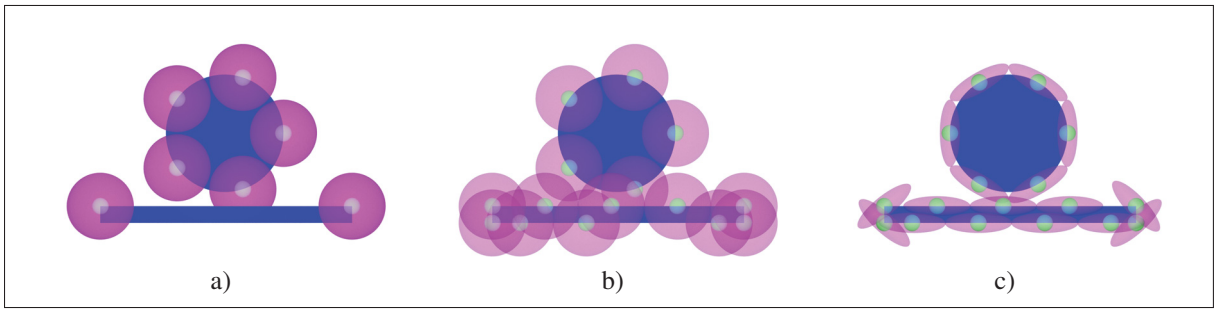



Figure 3.2 Example of underlying surfaces (in blue), with Poisson disks (in magenta) and their related Poisson points (in green): (a) Poisson disks with 3D radius $r$ criterion. (b) Poisson disks with distance and plane criteria. (c) Poisson disks with ellipsoid distance and plane criteria.

To fix this issue and provide an appropriate surface sampling, we propose using two criteria:

- no other Poisson disk should be inserted within the radius $r$ of another disk only if both disks are in the same plane;
- no other Poisson disk should be inserted inside an ellipsoid defined by the surface and disk.

The plane of a specific Poisson disk is computed from the normal $\vec{N}$ evaluated where the disk lies on the surface. When testing to add a new disk, we reject it only if it is in the same plane *and* if it is within the radius $r$. We thus test if the normal $\vec{N}_c$ at the candidate location is too

close to the normal $\vec{N}$ of the disk, determined by checking whether the value of the dot product is above a given threshold: $\vec{N_c} \cdot \vec{N} > pa$. For results shows in this chapter, we set $pa$ to 0.5.

These criteria are illustrated in Fig. 3.2. We can see in Fig. 3.2a that there are regions not covered by any sample when only the distance $r$ is used. In Fig. 3.2b, we add the orientation to the criterion, but some regions remain uncovered. Finally, Fig. 3.2c shows the result of using the ellipsoid in combination with the orientation, producing a better surface sampling.

We will refer to the ellipsoid criterion as the $k$-criterion. A candidate disk located at $\vec{p}$ is too close to another one if the following condition holds:

$$k(\vec{p}) = \frac{x^2}{r^2} + \frac{y^2}{r^2} + \frac{z^2}{cs^2} < 1, \tag{3.1}$$

where $cs$ is the ellipsoid thickness and $x$, $y$, and $z$ are the coordinates of the tested point in the tangent space of the Poisson disk, where $z$ points in the same direction as the normal $\vec{N}$. The geometry is illustrated in Fig. 3.3. This ellipsoid thickness should be smaller than the smallest detail of the fluid animation. The point $(x, y, z)$ is inside the ellipsoid when $k(\vec{p})$ is less than 1. With the $k$-criterion, we are able to fit thin surface details, as well as to address the undersampling problem shown in Fig. 3.4a; see Fig. 3.4b for an illustration of the result.

## 3.6 Deformable Patches

This section describes the proposed approach to handle surface deformation by using deformable patches. A deformable patch consists of the surface polygons surrounding the patch center; the polygon's vertices can then be advected independently while maintaining their texture coordinates, allowing the patch to deform.

### 3.6.1 Patch creation using surrounding polygons

A deformable patch is created by duplicating the polygons of the surface mesh surrounding the Poisson disk sample. The distance used to select polygons is slightly greater than the

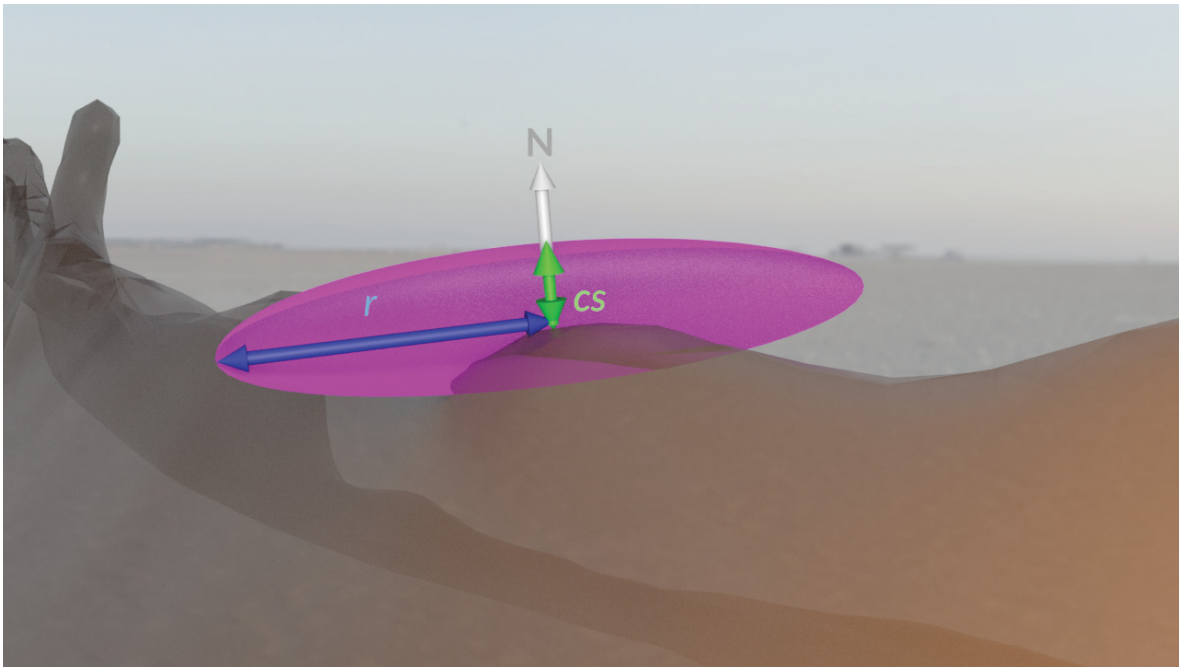



Figure 3.3 Our $k$-criterion corresponding to an ellipsoid (shown in magenta) having thickness $cs$ in the normal direction $\vec{N}$ and width $r$. Candidate samples within this ellipsoid are rejected.

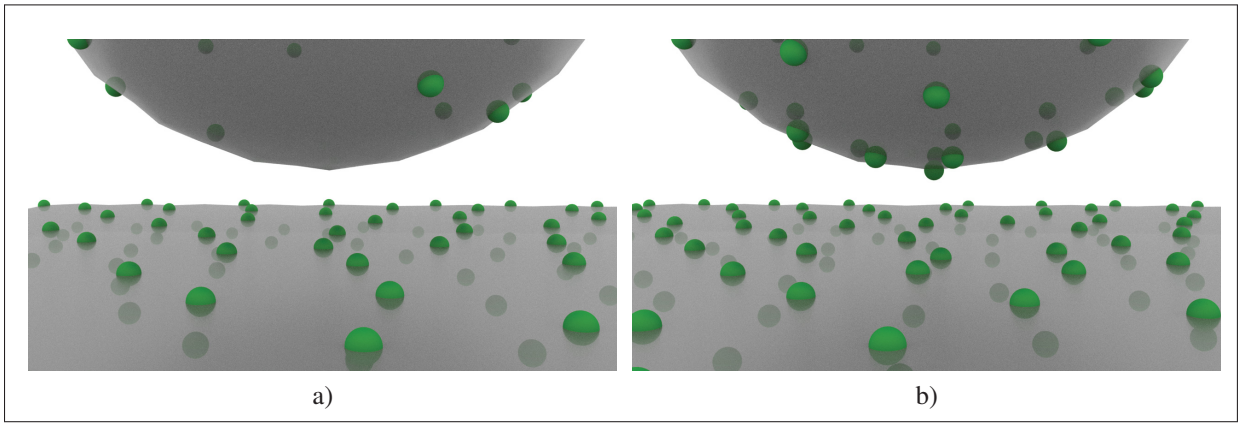



Figure 3.4 (a) Regular 3D Poisson disk distribution does not distribute enough disks in the region where the lower part of the sphere is close to the plane; (b) samples cover the lower part of the sphere when using our $k$-criterion.

ellipsoid used in equation 3.1 to ensure that there will be enough polygons on the border of the patch (Fig 3.5). We exclude polygons that are not in the same plane as the Poisson disk, using the dot product test described in Sec. 3.5. The resulting set of polygons can be flattened

to the *uv* coordinate space of the exemplar. To ensure minimal distortion of the *uv* coordinate assignment over the 3D patch, we use the parameterization given by Least Squares Conformal Maps (Lévy *et al.*, 2002).

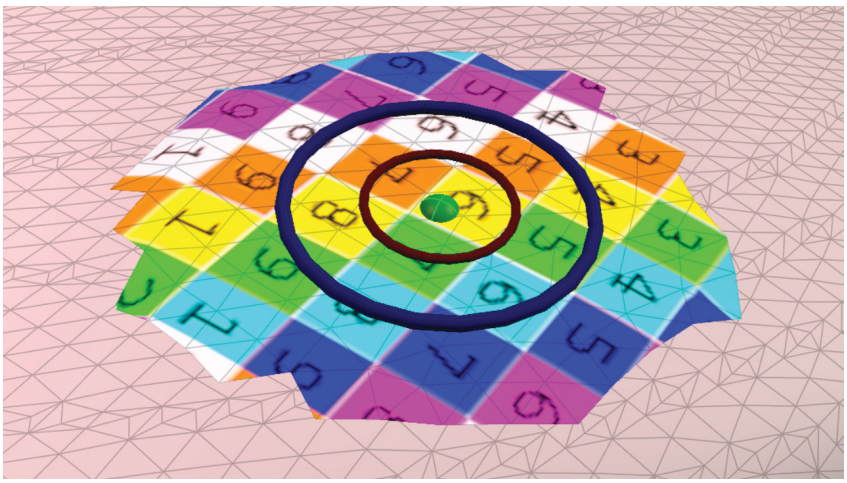


Figure 3.5 Representation of a deformable patch with the Poisson disk (outer ring), the selected polygons, and the *uv* coordinates. The kill zone (inner ring) is the threshold where we identify patches too close to each other.

### 3.6.2 Patch advection

We advect the vertices of each patch according to the velocity field on the surface. The examples presented in this chapter relied on an implicit surface reconstruction (with independent meshes at each frame), but our method would work as well with an explicit mesh (Dagenais *et al.*, 2017). Since the surface can undergo drastic and sudden topological changes, such as splits and merges, we must update vertices accordingly and remove any part of the patch that cannot follow the surface. Using an Euclidean distance threshold *mpt*, we remove from the patch all vertices that end up too far from the surface of the fluid. This removal allows for the introduction of new patches, leading to an overall patch distribution that is more faithful to the motion of the fluid's surface. Vertices below this threshold are projected onto the surface. The value of the threshold *mpt* is adjusted considering the scene scale and the fluid simulation. If

a patch ends up stretched between two disconnected components of the fluid, it is identified as distorted and removed, as explained in the next section.

### 3.6.3 Distortion estimation

Over the animation, every patch can deform, and the deformation can produce visible distortions in the resulting synthesis. Therefore, as in the work of Yu *et al.* (2011), we detect distortion with the distortion metric of Sorkine *et al.* (2002). Excessively distorted patches will be removed and replaced with undistorted ones; we consider a patch to be excessively distorted when the distortion value $\delta_{max}$ exceeds 3.

### 3.6.4 Poisson disk distribution update

With the creation and subsequent advection of deformable patches, we obtain deformed and displaced patches. We previously detected distorted patches, as described in Sec. 3.6.3. To maintain a uniform distribution of patches, we use the new patch positions to flag the patches to remove and insert new patches where there is a gap. In addition to distorted patches, we also want to remove patches that are too close to each other. These are the patches whose centers fall within the "kill distance" $(1-\alpha)d$ of another patch, as illustrated in Fig. 3.5. Where there is a gap on the surface, we insert a new Poisson disk and its corresponding patch.

The Poisson disk distribution is updated frame by frame, allowing us both to flag patches that are too close to each other, and to identify regions where new patches are needed. Finally, the distortion metric points out which patches need to be deleted because they are too distorted. The Poisson disks have four states:

- fading in
- active
- fading out (too distorted)

- fading out (too close)

The transition between states is illustrated in the accompanying video. Note that during the Poisson disk distribution update, we ignore the fading out patches.

## 3.7 Blending

The last step of the approach is to blend all patches together to synthesize the final texture atlas. This process is done on every frame of the fluid animation. First, for each vertex of each patch, a vertex weight is computed. Using this value, we then calculate the color of each texel using the blending function described in Sec. 3.7.4.

### 3.7.1 Patch update

After the advection, with the updated distribution, we can compute the quality of each vertex of a deformable patch at a time $t$. To do so, we compute a weight per vertex. A vertex weight $w_V(t)$ calculation was introduced by Yu *et al.* (2011):

$$w_V(t) = K_s(V)K_t(t). \tag{3.2}$$

The overall weight $w_V$ is the product of a spatial weight $K_s(V)$ and a temporal weight $K_t(t)$. The weight $w_V$ will be used in the blending computation to synthesize the texture.

### 3.7.2 Spatial component

The spatial component $K_s$ improves the continuity at the borders of the patches. The idea is to get the full color of the texture at the center of the patch, and to let this color contribution fall off towards the borders. To this end, we have $K_s$ equal to 1 at the center and 0 at the border. We use a linear blending, where $K_s$ varies according to the distance between the vertex $V$ and the patch center $\vec{p}$. To compute the 3D distance between $V$ and $\vec{p}$, we modify the $K_s$ computation given by Yu *et al.* (2011): instead of computing the distance between vertex $V$ and the Poisson

disk center $\vec{p}$ in three dimensional space, we compute the distance between vertex $V_{uv}$ and the disk center $p_{uv}$ in $uv$ space. Thus, when the patch is deformed, we always have the same distance in texture space, and the linear blending is consistent over time. Moreover, to give the user more control over the look of the texture synthesis, we also allow scaling $s$ of the $uv$ coordinates. The equation from Yu et al. is as follows:

$$K_s(V) = (1 - \frac{||V - p||}{d}) d_V Q_v. \tag{3.3}$$

We replace this with the following:

$$K_s(V) = (1 - \frac{||V_{uv} - p_{uv}||}{d_{uv}}) s d_V Q_v. \tag{3.4}$$

### 3.7.3 Density-based temporal component

We also modify the temporal component $K_t(t)$ from Yu *et al.* (2011), using a density-based fading. Fading out and fading in patches over a fixed number of frames works well when the velocity field does not change too rapidly. However, in fluid simulations, the velocity on the surface can change quickly, resulting in large amounts of patches that get stacked and have to be deleted. We propose to change the fading according to the density of patches in the patch's ellipsoid volume. Yu et al. fade over a fixed number of frames $\tau$:

$$K_t(t) = \begin{cases} \frac{t}{\tau} & if \quad t < \tau \\ 1 & if \quad \tau < t < t_k \\ 1 - \frac{t - t_k}{\tau} & if \quad t_k < t < t_k + \tau \end{cases} . \tag{3.5}$$

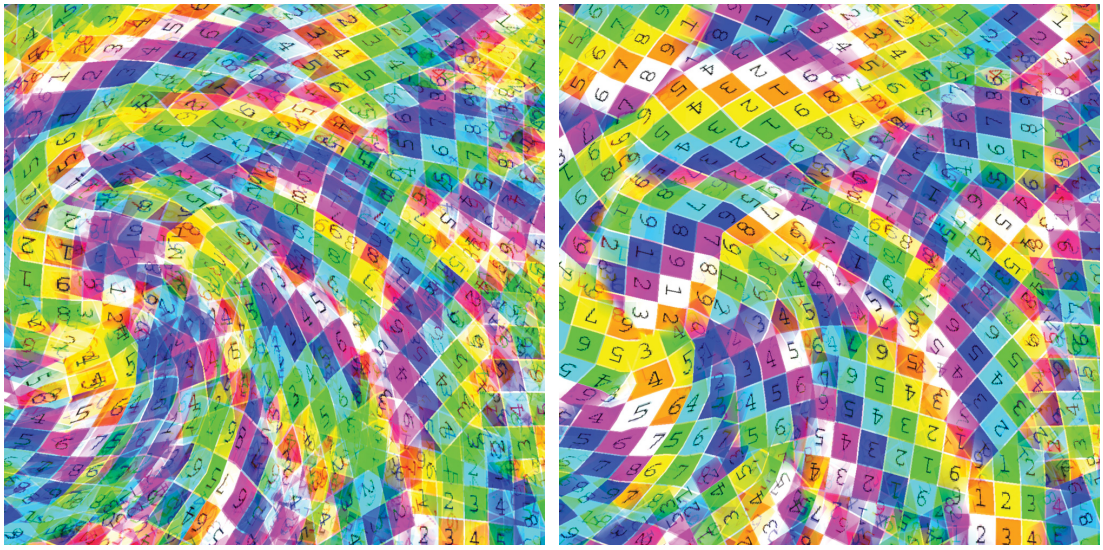
In contrast to the fixed linear evolution of $K_t(t)$ in the equation above, our fading value is adjusted dynamically based on the density:

$$K_t(t) = \begin{cases} \min\left(1, \frac{t}{\tau}(\rho+1)\right) & \text{if} \quad t < \tau \\ 1 & \text{if} \quad \tau < t < t_k \\ \max\left(0, 1 - \frac{t-t_k}{\tau}(\rho+1)\right) & \text{if} \quad t_k < t < t_k + \tau \end{cases}, \tag{3.6}$$

where $t_k$ is the time at which the patch is flagged to be removed, and $\rho$ is the density of the current patch's Poisson disk. When $K_t(t)$ reaches 1 (fading in) or 0 (fading out), we terminate the fading process, thus dynamically changing the number of frames over which we apply fading. The density $\rho$ is computed using the number of neighbors of the Poisson disk in the same plane and inside the radius $r$. We consider all patches when computing the density, irrespective of their current state. Patches will both fade in and fade out faster when there are more overlapping patches. Locations where we have large distortions trigger a lot of distorted patch removal, and the last line of Eq. 3.6 ensures their quick removal. However, removing many patches in turn triggers their replacement by many new patches. If patches here were to fade in slowly, there would be a good chance that they would be highly distorted by the time they have completely arrived, and thus they would need to be removed immediately. The first line of Eq. 3.6 ensures that patches fade in quickly, increasing the time during which the surface is covered by undistorted patches.

Under Equation 3.6, it is possible to encounter very large $\rho$, potentially producing a negative $K_t(t)$. When $K_t$ becomes negative for any patch, we remove the patch immediately.

In practice, the number of overlapping patches is related to the local velocity and deformation. Changing the rate of the fading at these locations looks visually plausible and involves fewer blending artifacts, as shown in Fig. 3.6. Indeed, we can see in this example that the numbers from the texture exemplar are easier to read because they involve less blending. In the end, we have fewer stacking artifacts, and the final texture can be computed more quickly as it involves blending fewer patches.



a) b)

Figure 3.6 (a) Linear fading vs. (b) density-based fading

### 3.7.4 Blending function

The vertex weight $w_V(t)$ (defined in Equation 3.2) is then used in the following blending function introduced by Yu *et al.* (2011) as $w_i(x)$:

$$R'(x) = \frac{\sum w_i(x)(R(u_i(x)) - \widehat{R})}{\sqrt{\sum w_i^2(x)}} + \widehat{R} \tag{3.7}$$

where $R'(x)$ is the pixel to compute, $\widehat{R}$ is the mean of the texture sample, and $u_i(x)$ is the texture mapping.

All the pixels are computed on a texture atlas. We use the approach of Lévy *et al.* (2002) to generate the texture mapping between the atlas and the fluid surface. The results of the blending are illustrated in Fig. 3.7.

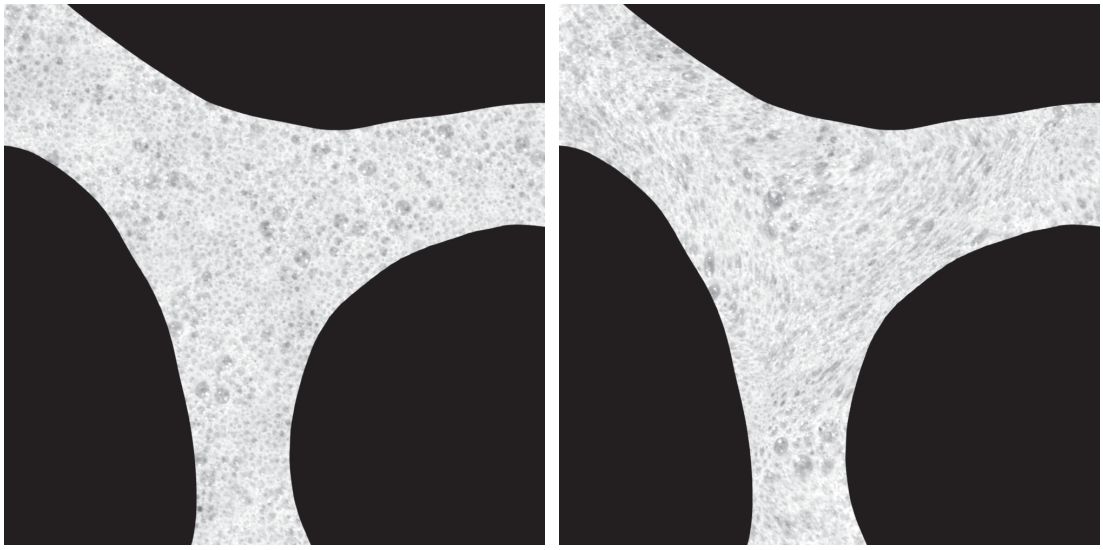
## 3.8 Results

We tested our approach using several fluid simulation scenarios similar to those from related work:

- 2D rotation flow, as in the paper of Yu *et al.* (2011) (Fig. 3.7)
- 2D with split and merge (Fig. 3.9)
- 2D flow with split, as in the paper of Yu *et al.* (2011) (Fig. 3.8)
- 3D viscous drop, as in the paper of Gagnon *et al.* (2016) (Fig. 3.12)
- 3D liquid dam break, as in the papers of Kwatra *et al.* (2007) and Gagnon *et al.* (2016) (Fig. 3.13)
- 3D lava drop, as in the paper of Gagnon *et al.* (2016) (Fig. 3.11c)

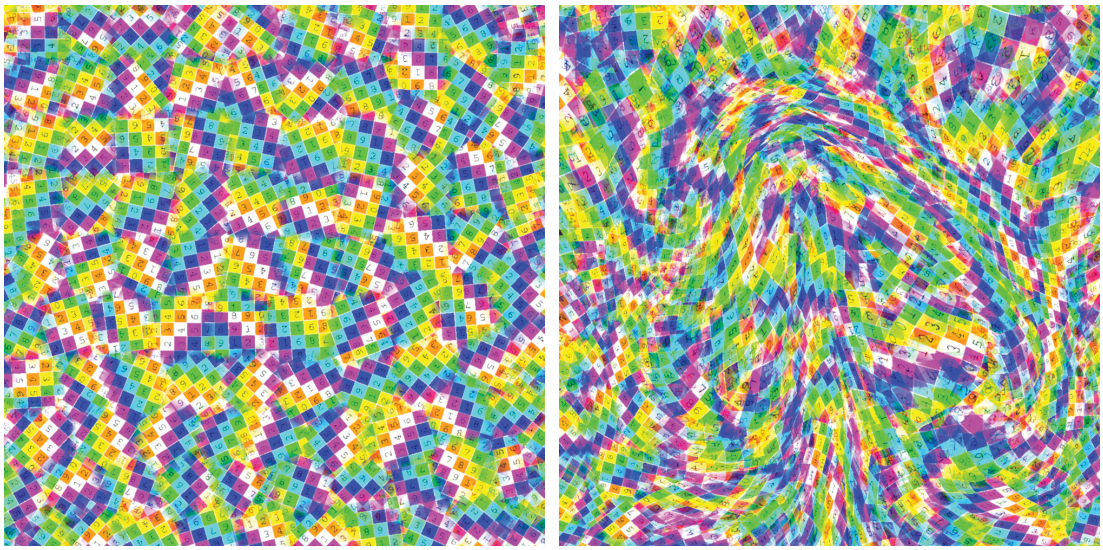
a) b)

Figure 3.7 2D rotational flow using a gravel texture exemplar, at frame 1 (a) and 240 (b)



a) b)

Figure 3.8 2D split using a bubble texture exemplar, at frame 1 (a) and 240 (b)



a) b)

Figure 3.9 2D fluid with split and merge, at frame 1 (a) and 140 (b)

Table 3.2 summarizes the statistics for the scenarios shown in the chapter and video.

We compared our results with the method of Kwatra *et al.* (2007). Their method works best with small exemplars (for example, resolutions $64 \times 64$ or $128 \times 128$). Using larger exemplars

Table 3.2 Statistics of our examples. "Nb poly" is the average number of polygons on the fluid's mesh, "Nb patch" is the average number of patches, and "Atlas res" is the resolution (in pixels) of the computed atlas. Computation times for the Poisson disk creation, patch creation, and atlas creation are in seconds. The patch advection is negligible, taking less than 0.6 seconds for all of our examples. The patch and atlas creation steps are computed in parallel (on 20 cores in our tests). Our computations were conducted on a 2.80 GHz Intel Xeon® E5-2680 CPU.

| | Number of Polygons | Number of Patch | Atlas Resolution | Poisson Disk | Patch | Atlas | Total |
|---|---|---|---|---|---|---|---|
| Dam Break | 7k | 6900 | $6k^2$ | 15 | 30 | 45 | 90 |
| Viscous Drop | 28k | 1740 | $6k^2$ | 10 | 4 | 101 | 115 |
| Split & Merge | 5k | 674 | $1k^2$ | 3 | 1.1 | 2 | 6.1 |
| Split Y | 6k | 584 | $1k^2$ | 0.5 | 1 | 1.3 | 2.8 |
| Lava | 51k | 5700 | $6k^2$ | 40 | 4 | 142 | 186 |

has a severe negative impact on computation times, and complicates the selection of the window size for the best match search, since the size of the window is related to the size of the patterns to replicate. Larger window sizes, required for larger exemplars, in turn also have a negative impact on computation times. For our method, either small or large exemplars can be used with negligible impact on computation time. As noted by Jamriška *et al.* (2015), the method of Kwatra *et al.* (2007) does not work for all texture exemplars, as it can lead to a wash-out effect, where the synthesized texture becomes too homogeneous and blurry. As can be seen in the video, the method of Kwatra et al. works better than ours when dealing with structured patterns. On the other hand, stochastic and isotropic textures are one of the causes of the wash-out observed in the synthesized results of Kwatra et al.'s method. In the accompanying video, we can see that our method preserves more of the exemplar colors and structure, as compared to the washed-out result from Kwatra et al.'s method.

We also compared our results with those of Gagnon *et al.* (2016) (Fig. 3.11). In their results, we can sometimes identify individual patches. With the large and rigid patches they use, we can observe blocky artifacts, and the optical flow is less respected. In comparison, with our approach, it is difficult to guess the boundaries of the patches, and very fine texture details

remain visible in our output. In addition, we maintain a correspondence between the texture movement and the free surface movement despite using large patches.

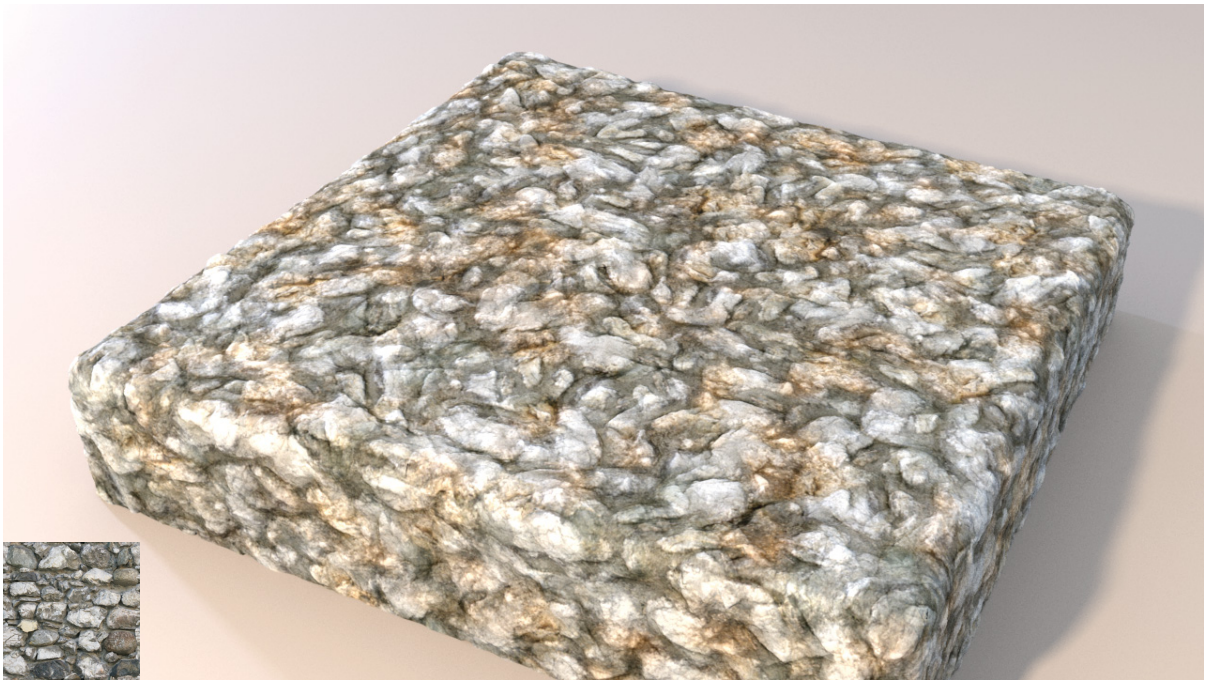


Figure 3.10 Dam break with stone texture exemplar (inset).

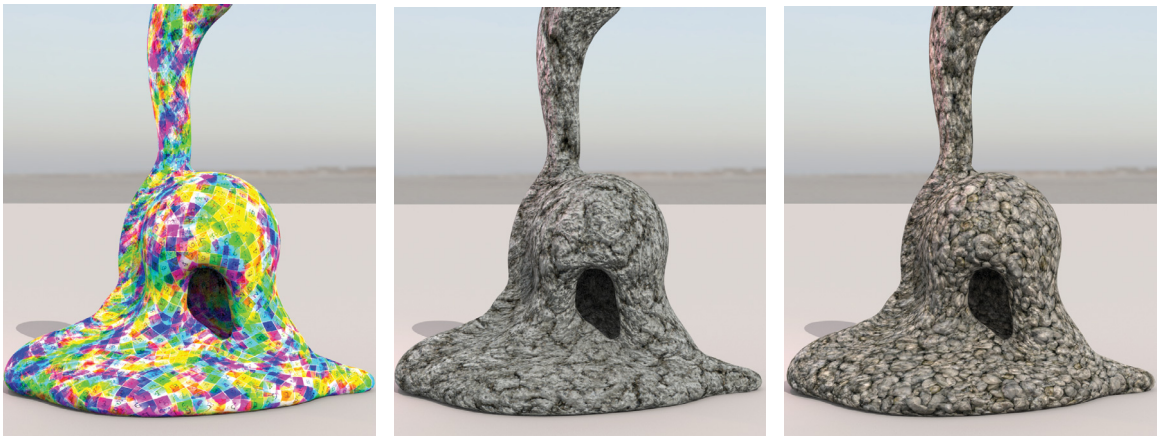
As can be seen in the figures, as well as the accompanying video, our approach works well on 2D examples (Fig. 3.7); with 3D viscous fluids (Figs. 3.11c and 3.12); and with 3D liquids (Fig.3.13). We can also synthesize different textures using the same deformable patch distribution, simply by changing the texture exemplar (Fig 3.12).

We should take particular note of Fig. 3.13, where the simulation has produced a complex mesh with a fine structure. The texture covers the mesh seamlessly and with good fidelity to the exemplar. In the accompanying video, it can be seen that our approach copes well with the entire animation despite the presence of multiple topological changes in the splashing liquid.

With the deformable patch representation, the texture conforms well to the movement of the fluid surface irrespective of any topological changes. We further demonstrate this in Fig. 3.14

a) b) c)

Figure 3.11 Lava simulation. Using texture exemplar (a), we compare between (b) Gagnon *et al.* (2016) and (c) our approach.



a) b) c)

Figure 3.12 Different textures using the same underlying patch distribution

and in the accompanying video, where we compare the fluid's velocity field (shown in blue) with the optical flow of the rendering (shown in green). The two fields are in close agreement, thus showing that the texture synthesis is able to match the liquid flow.

Thanks to the patch advection and Poisson distribution update, there are no regions with excessive numbers of patches or severe distortion. Moreover, with the density-based fading, we have fewer patches, and thus faster computation and fewer ghosting artifacts.

An added benefit of our approach is that we can apply other texturing techniques on fluids. For example, Fig. 3.1, 3.10, 3.11c, and 3.13 use displacement mapping, while Fig. 3.12 uses normal

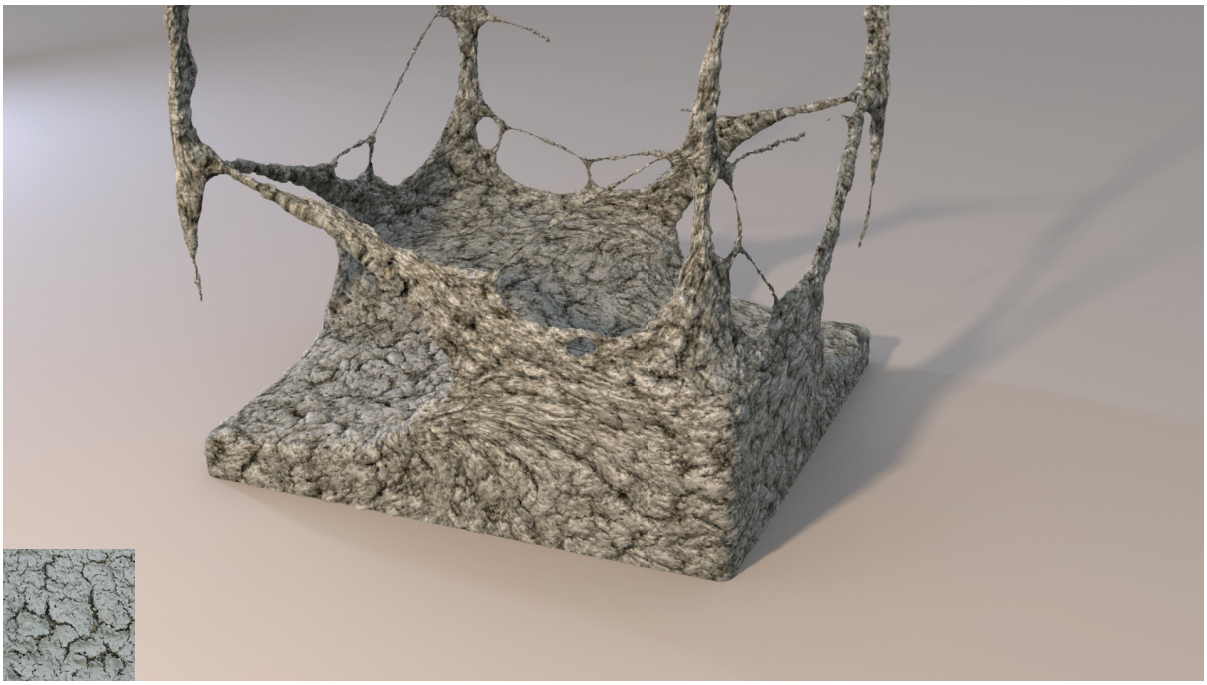


Figure 3.13 Dam break example with splash filaments

mapping. Corresponding scenarios found in the accompanying video also use displacement and normal mapping.

### 3.8.1 Limitations

One limitation of our approach appears in scenarios where the fluid stops or barely moves. Patches will continue to fade in and out for several frames, even if the velocity is very small or null. The fading sometimes leads to the impression that the fluid continues to move slightly. In the video of the rotational flow (Fig. 3.7), the slow movement makes it easier to note the fading in and out near the center.

Our approach also exhibits lengthy computation times; each frame can take a few minutes to compute, depending on the complexity of the fluid's animation. The advection and the patch creation times are similar to those of Gagnon *et al.* (2016). However, the texture synthesis time can be more significant. Indeed, since for each output texel we are using a blending of all overlapping patches, we need to compute the exact position of the texel on the 3D surface. With

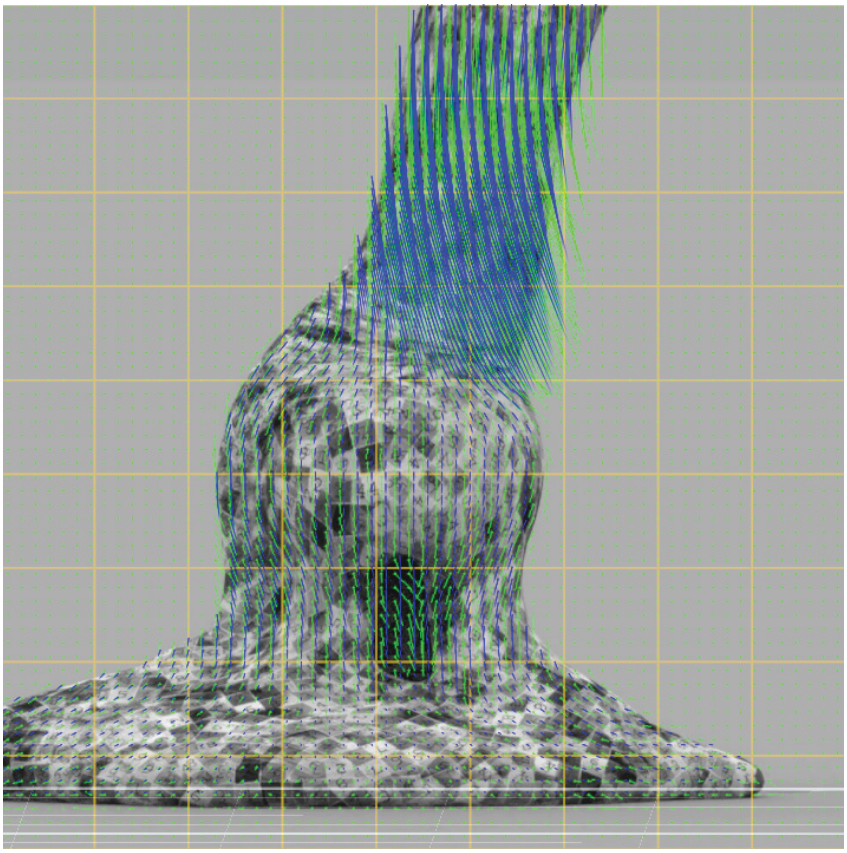


Figure 3.14 Test case demonstrating that the optical flow extracted from the texture advection matches the velocity field from the fluid simulation

this 3D texel position, we perform a projection through all intersecting patch triangles to get the *uv* coordinate that will provide the patch's texel. The large number of required projections acts as a bottleneck for the overall texture synthesis process.

Even with the density-based fading, some scenarios still produce visible ghosting effects. Just as with Yu et al.'s method (Yu *et al.*, 2011), this is inevitable because blending is used to combine the color of the patches overlapping the same region of the surface. The contrast-preserving blending (Eq. 3.7) reduces blurring at the expense of increased ghosting. Regular blending could be used when results with less ghosting are preferable. The ghosting is also worsened with structured textures, as illustrated in Fig. 3.15.

Figure 3.15 Ghosting artifacts can become visible, especially when using a structured texture

## 3.9 Conclusion

We have presented an approach for synthesizing texture on the free surface of animated fluids. The proposed approach creates textured patches with limited distortion, based on a Poisson disk distribution. We want a good distribution for the specific case of fluids with small details, such as splashes and thin films, and we need to account for patches that will be wrapped on the fluid's surface. The update of the distribution and of the patches fully supports topological changes. We presented a more elaborate temporal fading out of patches, which allows control over their accumulation; this is most noticeable in areas where the flow converges or where different regions of the surface come into contact. Finally, we showed that the surface texture follows the velocity field of the fluid simulation, demonstrating this by comparing the optical flow with the fluid velocities.

We believe this work can be extended to include a measure of local velocity in our adaptive fading, further improving the texture evolution. The blending function is another area of po-

tential development, further reducing ghosting beyond the improvements seen in this chapter. Another avenue to reduce the ghosting could involve an extra step using a best-match search as in the method of Kwatra *et al.* (2007), applied only in regions with excessive ghosting.

## 3.10 Acknowledgements

This work was funded by Digital District Canada, MITACS, Prompt, and ÉTS. We gratefully acknowledge the involvement of the former Digital District R&D team. We thank SideFX for providing Houdini licenses, and Hippolyte Mounier, from Photosculpt, for sharing some of the texture exemplars used in this project. The collaboration between ÉTS and Carleton University resulted from the 2017 Bellairs Multidisciplinary Computer Science Workshop held at the Bellairs Research Institute of McGill University.

## 3.11 Appendix: Input variables

Tables 3.3 and 3.4 describe the parameters of our approach.

Table 3.3 Input variables

| Variable | Description |
|---|---|
| $d$ | Poisson disk radius |
| $pa$ | Poisson disk plane angle |
| $pva$ | Patch vertex angle threshold |
| $\delta_{max}$ | Is the maximum deformation allowed, we use 3 |
| $s$ | Scaling of the $uv$ coordinates |
| $mpt$ | Maximum projection threshold to handle topological changes |
| $\tau$ | Fading speed |
| $cs$ | Ellipsoid thickness representing the smallest detail |

Table 3.4 Constants

| **Constant** | **Description** |
|---|---|
| $(1-\alpha)d$ | Kill distance where $\alpha$ is equal to 0.25 |
| $Q_v min$ | Quality vertex min is equal to 0.5 |

# CHAPTER 4

# PATCH EROSION FOR DEFORMABLE LAPPED TEXTURES ON 3D FLUIDS

Jonathan Gagnon [a], Julian Guzman [b], David Mould[c], Eric Paquette[b]

[a] Folks VFX
[b] Département de Génie Mécanique, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[c] Caleron University

Published in "Computer Graphics Forum",
June 2021

## 4.1 Abstract

We propose an approach to synthesise a texture on an animated fluid free surface using a distortion metric combined with a feature map. Our approach is applied as a post-process to a fluid simulation. We advect deformable patches to move the texture along the fluid flow. The patches are covering the whole surface every frame of the animation in an overlapping fashion. Using lapped textures combined with deformable patches, we successfully remove blending artifact and rigid artifact seen in previous methods. We remain faithful to the texture exemplar by removing distorted patch texels using a patch erosion process. The patch erosion is based on a feature map provided together with the exemplar as inputs to our approach. The erosion favors removing texels toward the boundary of the patch as well as texels corresponding to more distorted regions of the patch. Where texels are removed leaving a gap on the surface, we add new patches below existing ones. The result is an animated texture following the velocity field of the fluid. We compared our results with recent work and our results show that our approach removes ghosting and temporal fading artifacts.

## 4.2 Introduction

Texture mapping is the technique of choice to add details to surfaces. It is largely applied in settings where the surfaces are either rigid or deforming in an isometric fashion. However, fluid

animations pose difficult problems with challenges arising from severe deformations of the surface, even including changes in surface topology. Texture mapping under such circumstances can yield significant artifacts, and dealing with these artifacts is still a nontrivial problem with only a few solutions.

Recent fluid texturing methods use patch-based texture synthesis, where patches contain a set of pixels from a texture exemplar (Bargteil, Sin, Michaels, Goktekin & O'Brien, 2006b; Kwatra *et al.*, 2007; Narain *et al.*, 2007; Yu, Neyret, Bruneton & Holzschuch, 2011). Blending deformable patches may produce ghosting artifacts. Gagnon *et al.* (2016) used rigid opaque patches; their method significantly reduced ghosting, but the rigid patches were not able to follow the fluid flow closely. Gagnon *et al.* (2019) proposed deformable patches with alpha blending to combine overlapping patches, but this method suffered from ghosting. Moreover, the fading of distorted patches was not always consistent with the velocity field, especially in areas with little fluid motion. The introduction of new patches and the removal of distorted patches over multiple frames introduced temporal inconsistencies in their method. Our objective is to create a texture on a fluid animation with neither ghosting artifacts nor temporal inconsistencies when the fluid barely moves.

Our solution combines lapped textures and deformable patches. To remove distorted patches, instead of fading out the whole patch, we apply an erosion process to remove distorted patch texels using a feature map. The main contributions of the proposed texture synthesis approach can be summarized as follows:

- Adaptation of the lapped textures framework for deformable patches on fluids;
- Feature-aware patch erosion preserving consistency to the texture exemplar;
- Automatic creation of a multitude of lapped texture shapes based on a feature map;
- Patch distribution update based on texture erosion.

## 4.3 Related Work

Neural networks have been used for style transfer on 2D smoke simulation (Christen *et al.*, 2020) and later extended on volumetric fluid animation, changing also the shape of 3D fluid surfaces (Kim *et al.*, 2020). Where these methods focus on volumetric data, our approach concentrates on surface texture synthesis.

Kwatra *et al.* (2005) and Jamriška *et al.* (2015) proposed an animated 2D texture synthesis using a best-match search. However, the resulting animated texture looks stiff as the search windows do not deform. To avoid stiffness, it is possible to use textured patch advection, as proposed by Neyret (2003) and Yu *et al.* (2011); in these methods, patches can deform, resolving the stiffness concern. Overly distorted patches are replaced by undistorted ones. Alpha blending is used to phase out a distorted patch over time; unfortunately, the blending can create ghosting artifacts where structural patterns are blurred.

Only a few methods deal with 3D fluids. Bargteil *et al.* (2006b), Kwatra *et al.* (2007), and Narain *et al.* (2007) extended the work of Kwatra *et al.* (2005) for a best-match search texture synthesis on a 3D mesh. Unfortunately, they inherited the limitations of the 2D texture synthesis of Kwatra *et al.* (2005), with stiff-looking texture animations. Another limitation is inherent in best-match search methods: larger exemplars drastically increase computation times. Gagnon *et al.* (2016) proposed rigid patches advected on the free surface. Their patch-based method does not use best-match searches, avoiding the problem of texture exemplar resolution. However, the resulting texture animation shows stiff texture patterns. To deal with the stiffness problem, Gagnon *et al.* (2019) extended the deformable patches method of Yu *et al.* (2011) to 3D fluids. They also added a new sampling method to reduce the number of patches in order to reduce ghosting, but it is still an issue.

Our goal is to take advantage of deformable patches to avoid stiffness, and lapped textures to reduce blending. Lapped textures is a patch-based texture synthesis method (Praun *et al.*, 2000), successfully used to texture fluids by Gagnon *et al.* (2016). We also improve upon deformable patches texture synthesis (Yu, Neyret, Bruneton & Holzschuch, 2011; Gagnon *et al.*,

2019), replacing the patch fading by a feature-aware erosion method that removes distorted regions of patches.

## 4.4 Overview

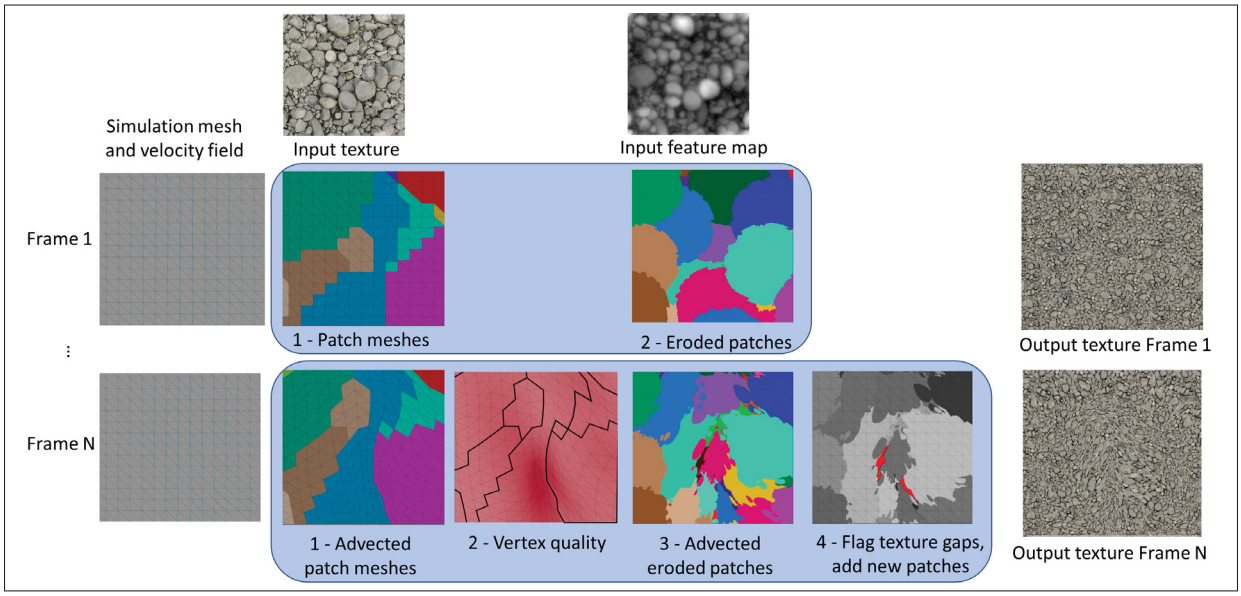



Figure 4.1 Overview of our approach

Our approach is designed to add texture details to the surface of a fluid. It is applied as a post-process and does not interfere with the fluid simulation. We take as input an animated mesh and a velocity field. The animated mesh of the fluid is typically a completely different mesh at every frame (e.g., surface reconstruction from an implicit function). The velocity field has no restrictions other than allowing sampling at arbitrary locations on the fluid surface.

The steps of the approach are illustrated in Fig. 4.1. Our approach uses patch-based synthesis, with patches being textured with a texture exemplar. Patches are represented as small polygon meshes distributed over the whole surface. We cover the fluid surface with overlapping textured patches, applying a Poisson disk distribution on the surface of the fluid at the first frame of the animation.

The polygonal patches are advected with the velocity field. Advection is done on a per-vertex basis, and consequently the patches deform over time, causing the texture to become distorted. We remove distorted textured regions of patches using a feature-aware erosion. Features are defined by a feature map provided as an input by the user. The erosion considers three criteria: removal of lower-importance features, removal of distorted regions of patches, and removal progressing from the outside toward the inside of the patches. At each frame, an output texture is synthesized over the entire input mesh with the layered patches.

The patch distribution is updated during the synthesis to ensure full coverage of the surface. Any time a surface texel has no covering patch texel, a new patch is added. Also, if a patch is not used in the texture synthesis, either because it is underneath other patches or it has no remaining texels, then it is removed. In the next sections, deformable patches are discussed (Sec. 4.5), followed by the texture synthesis using feature-aware erosion (Sec. 4.6).

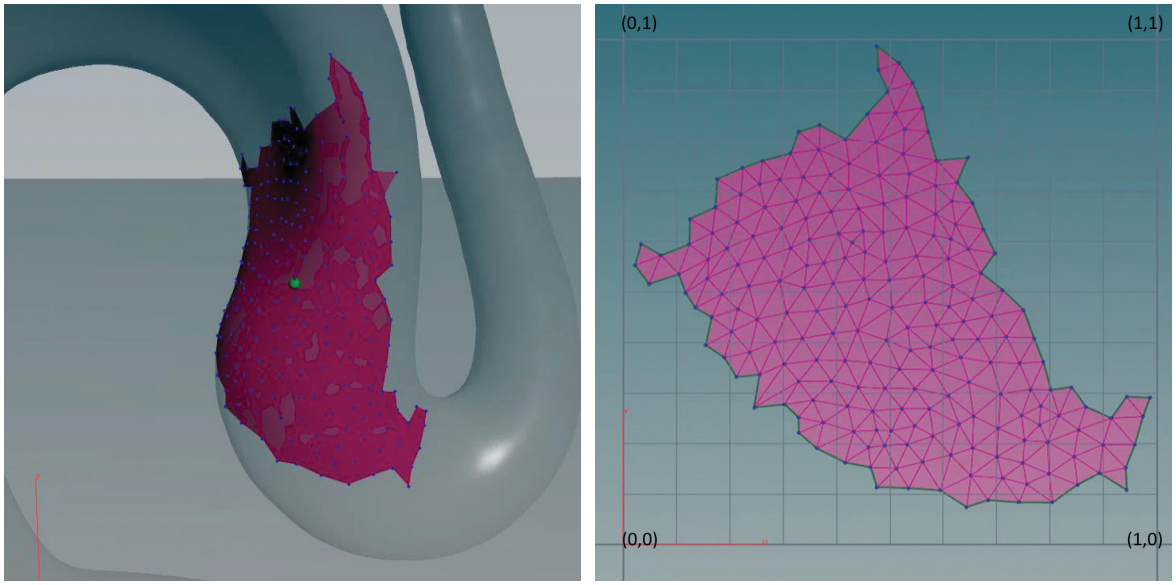
## 4.5 Deformable Patches

A patch is an open triangle mesh covering a local region from the surface of the fluid. Together, the patches cover the entire fluid surface. Patch vertices are advected by the fluid motion, causing initially well-shaped patches to deform over time; excessively deformed patches are progressively removed, using a process we detail in Sec. 4.6.2.

We initialize the set of patches at the first frame of the animation, placing patch centers on the fluid surface with the Poisson disk distribution of Gagnon *et al.* (2019). Each patch is then created by copying, from the mesh of the fluid, a subset of the polygons near the patch center, as illustrated in Fig. 4.2a. That is, the polygons for the patch perfectly match a portion of the fluid mesh at the moment when the patch is created. The subset of the fluid mesh is determined by proximity to the patch center $p$: we take polygons within Euclidean distance $r$, where $r$ is the user-specified patch radius. Of the nearby polygons, we exclude those that face away from the surface normal at $p$. With these criteria, each patch is guaranteed to contain at least one

triangle, though most patches are considerably larger. After creation, the mesh topology of each patch is independent of the mesh topology of the fluid simulation.

The polygonal patches are flattened with the parameterization of Lévy *et al.* (2002). Each flattened patch thus has associated *uv* coordinates, used to map the exemplar onto the patch. The patch is translated so that the position corresponding to $p$ is at the center of the *uv* coordinate space. The outcome is illustrated in Fig 4.2b.




a) b)

Figure 4.2 Patch creation using polygons from the mesh of the fluid (a) around position $p$ (highlighted in green) and the resulting *uv* coordinates (b) after flattening

Throughout the animation, we advect the individual vertices of the deformable patches according to the velocity of the fluid. To ensure that the patch matches the fluid surface, patch vertices are projected onto the fluid mesh, finding the closest location on the mesh to each vertex. After advection, vertices are sometimes too far from the surface, for example when there are topological changes due to splashes and droplets. Such vertices, and their related polygons, are removed from the patch. In this chapter, we removed vertices at a distance greater than $r/4$ from the surface.

Rejecting any portion of a patch – whether from polygons pointing away from the normal at $p$, erosion (Sec. 4.6.2), or removing vertices far from the surface – can produce a gap, where part of the fluid surface is not covered by a patch. Any gaps in coverage will be detected when filling our atlas (Sec. 4.6.1), and we add new patches to cover gaps. Newly added patches are full size, covering a larger region than just the detected hole. We thus avoid repeatedly introducing new patches in regions where a hole is growing.

## 4.6 Texture Synthesis

Our texture synthesis computation is inspired by the lapped textures method. For every frame of an animation, we compute an output texture atlas covering the whole surface of the fluid. This section describes how the texture is computed using the overlapping patches (Sec. 4.6.1) and how to remove distorted patch texels using feature-aware erosion (Sec. 4.6.2).

### 4.6.1 Lapped Textures

Inspired by the lapped textures method (Praun *et al.*, 2000), our approach relies on a set of overlapping patches covering the 3D mesh. While Praun *et al.* (2000) use a static and manually created patch mask, our mask is dynamic, automatically derived from a patch erosion process. As such, for each patch, the set of texels considered as *valid* (not eroded) changes over the animation.

An output texture atlas is computed with the method of Lévy *et al.* (2002). For each output texel $c$, we go through all patches that overlap $c$. These patches are sorted: the older patches are on top and the newest ones are below. Beginning with the top-most patch overlapping $c$, if the patch texel overlapping $c$ is flagged as eroded, the patch is rejected and we continue to the next patch below. When we find the first patch that is not rejected, we set the color of texel $c$ to the corresponding texel color from the patch.

Should all patches be rejected for output texel $c$, we add a new patch as described in Sec. 4.5. The center $p$ of the new patch on the fluid mesh is set to the location of the uncovered texel $c$. This patch distribution update is inspired by that of Gagnon *et al.* (2016).

In contrast to the transparent patch texels of Gagnon *et al.* (2016), our patch texels are either fully visible or rejected by the erosion process. This prevents any ghosting artifacts that might be caused by alpha blending.

### 4.6.2 Feature-Aware Patch Erosion

Repeated advection and projection of patch vertices can cause a patch to become distorted, and we remove regions (texels) of the patch that have undergone excessive distortion. To select the portions to remove, we combine three strategies: (1) removal that considers the features of the texture exemplar, (2) removal of regions close to distorted patch vertices, and (3) removal progressing from the outside toward the inside of the patches.

The features of the texture exemplar are defined by a feature map. The feature map could come from various sources such as a feature distance (Lefebvre & Hoppe, 2006), a height map, or it could be hand-created by an artist. For the results shown in this chapter, we relied on height maps. The feature map contains values in the range $[0,1]$, and the erosion will first favor regions with lowest values. An example of feature map values is illustrated in Fig. 4.3.

On each frame of the animation, patches are advected and thus deform. As the simulation proceeds, the initially well-shaped patches become distorted as vertices move in different directions and at different speed following the velocity field. Deformed patches distort the texture features of the exemplar. With the goal of identifying distorted patch regions that we will remove, we evaluate the quality $Q_T$ of patch triangles. We compute the triangle quality with the same distortion metric as in other papers (Sorkine, Cohen-Or, Goldenthal & Lischinski, 2002; Yu, Neyret, Bruneton & Holzschuch, 2011; Gagnon *et al.*, 2019):

$$Q_T = max((\delta_{max} - \delta_t)/(\delta_{max} - 1), 0), \tag{4.1}$$

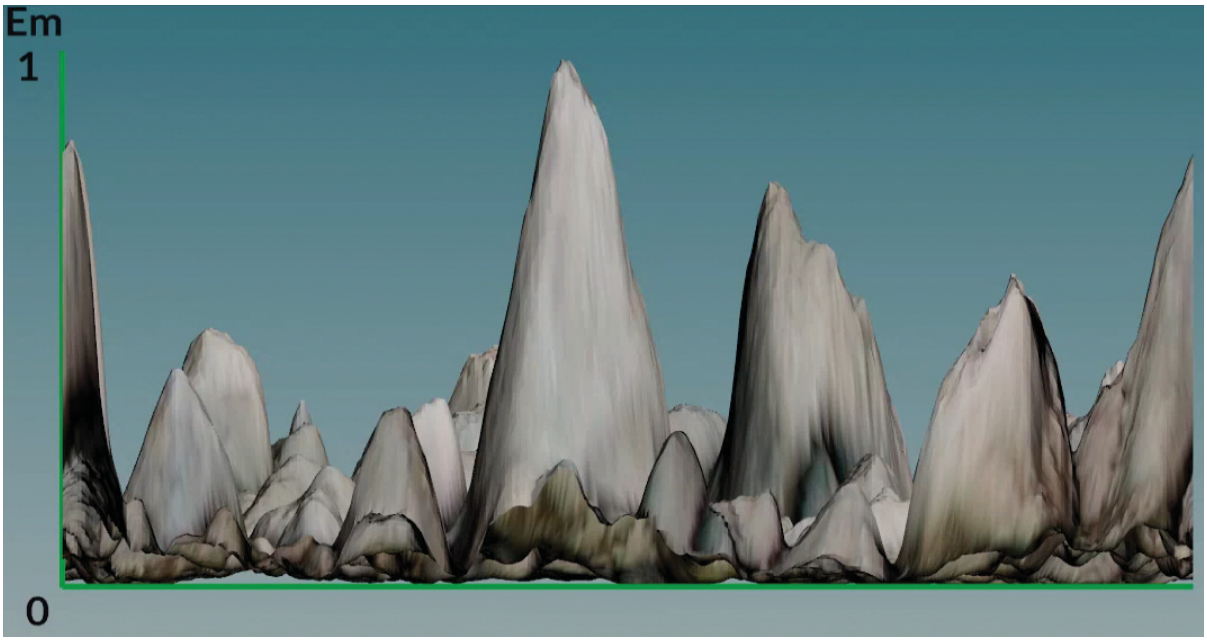



Figure 4.3 Side view of a feature map $F_m$

where $\delta_{max}$ is a constant defined by the user. We consider a triangle to be too distorted when its distortion $\delta_t$ exceeds a maximal distortion $\delta_{max} = 3$ . We then compute a per vertex quality $Q_V$ by averaging $Q_T$ from the neighboring triangles.

As we saw in Sec. 4.5, we might reject polygons within the radius $r$ and thus get an irregular patch shape as seen in Fig. 4.4. When such a patch is not distorted ($Q_V = 1$), we will not erode the patch and the boundary of the rendered patch will not be related to the features, but to the polygonal patch boundary instead. To avoid this, we supplement the vertex quality with a boundary value $B_V$ that drops to zero at the boundary edges of the mesh:

$$B_V = \begin{cases} 1, & V \text{ is an interior vertex of the patch} \\ 1, & V \text{ is within } r/4 \text{ from } p \\ 0, & V \text{ is a boundary vertex further than } r/4 \text{ from } p \end{cases} . \tag{4.2}$$

The second case is used to avoid eroding the center $p$ of the patch. Fig. 4.5 shows an example where, very close to $p$, polygons are rejected since they are pointing away from the normal

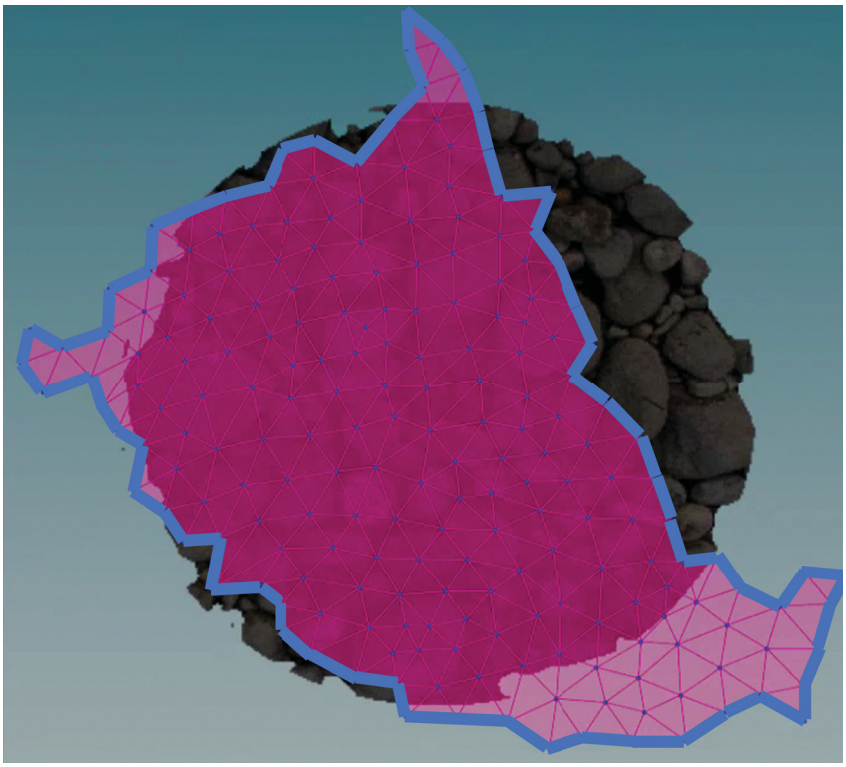
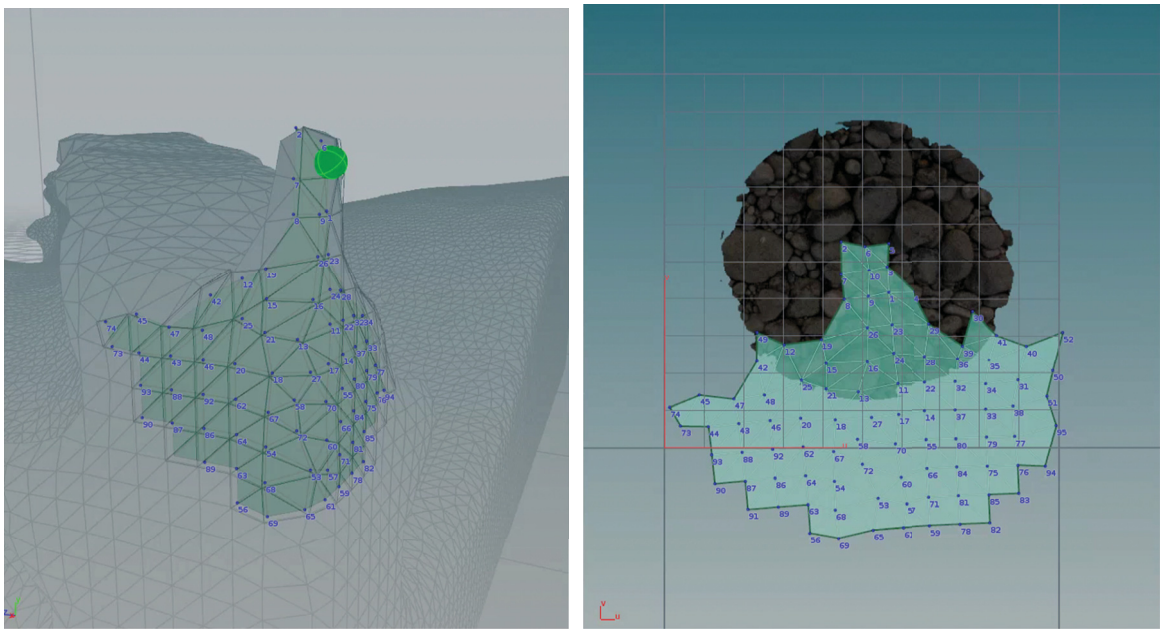


Figure 4.4 Example of a patch mesh in *uv* domain, where the boundary ($B_V = 0$) is shown as the blue line around the patch

vector at $p$. If we had $B_V = 0$ for one or more of the vertices of the triangle containing $p$, the interpolation of $B_V$ could lead to a very small value near $p$. In such a case, the same texel that triggered the patch creation could be eroded, leading to an infinite loop constantly failing to add a patch with a valid texel at $p$. We avoid this with the second case, enforcing $B_V = 1$ within a small distance from the patch center. Note that in all circumstances, the radius $r$ is always fixed and the same for all patches.

Similarly to Yu *et al.* (2011), we combine quality and boundary values in a vertex weight $w_V$:

$$w_V = Q_V B_V . \tag{4.3}$$



a) b)

Figure 4.5 Patch in 3D (a), where the corresponding patch center $p$ in the $uv$ domain (b) is very close to the patch boundary edges

For the vertex weight, Yu *et al.* (2011) and Gagnon *et al.* (2019) also include a temporal component in Eq. 4.3. Their temporal component $K_t(t)$ is a variable to fade out a distorted patch over a number of frames. Our approach does not use $K_t(t)$, which has the advantage of removing the issue related to fading patches over several frames even when the fluid does not move (see the accompanying video). The last piece of our erosion strategy favors erosion from the outside toward the inside of the patch. We define $D(u,v)$ based on the $uv$ coordinates: $D(u,v) = \max(0, 1 - ||(0.5,0.5) - (u,v)||/0.5)$. Thus $D(u,v)$ is 1 at the middle of the parameterization and decreases to zero toward the limits of the $uv$ space.

We combine our three erosion criteria into the following erosion selection:

if $\left( \frac{\sqrt{F_m(u,v)+D(u,v)}}{\max_{(s,t)}\left(\sqrt{F_m(s,t)+D(s,t)}\right)} - \varepsilon < 1 - w(u,v) \right)$

erode ,

else

keep

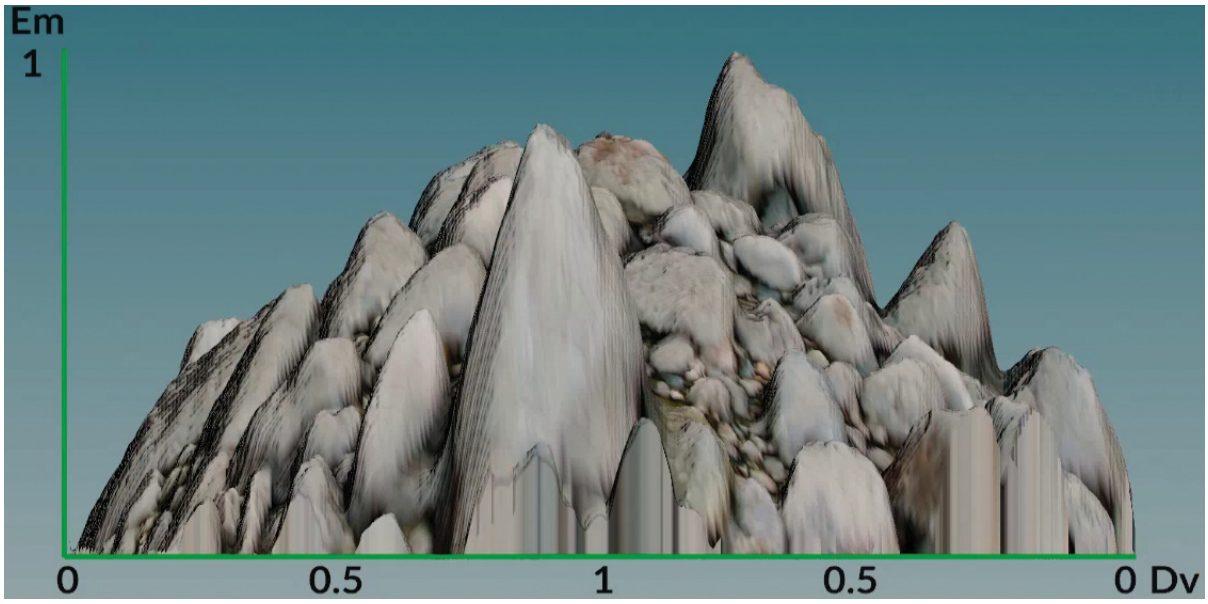



Figure 4.6 Normalized $\sqrt{F_m(u,v)+D(u,v)}$ which is compared to $1-w(u,v)$ in the erosion process

where $(u,v)$ is the texel position, $F_m(u,v)$ is the feature map value, and $w(u,v)$ is the texel weight interpolated from the per-vertex weights (Eq. 4.3) of the enclosing triangle. The square root over the feature map and the distance ensures that the area of the patch will be proportional to $1-w(u,v)$. By normalizing the square root by its maximal value over all texels, we keep the values of this term in the range of $[0,1]$, which is the same range as the weight. Fig. 4.6 shows the global shape of the normalized square-root term. The final element in our selection criterion is $\varepsilon \in ]0,1[$. By subtracting a small value $\varepsilon$ (we use $\varepsilon = 0.1$ for the results in this chapter), all patches are initially eroded. This ensures that the boundary of the rendered patch has the irregular shape necessary for the lapped texture technique to hide the patch seams.

In Fig 4.7 and the accompanying video, we show the erosion of a single patch. As the patch deforms, the erosion removes texels where the distortion is higher (on the right side for the patch seen in Fig 4.7d). A patch remains in use as long as at least one of its texels is used in the synthesized texture.

a) b) c) d) e)

Figure 4.7 A patch example using a pebble texture exemplar (a) with its feature map (b). We see the progression of the erosion when the patch has no distortion (c), when it is partially distorted (d), and when it is almost completely distorted (e).

## 4.7 Results

We have validated our approach using several fluid simulation scenarios similar on those from related work:

- 2D fluid with split and merge, as in the paper of Gagnon *et al.* (2019) (Fig. 4.8 and 4.14)
- 3D viscous drop, as in the papers of Gagnon *et al.* (2016) and Gagnon *et al.* (2019) (Fig. 4.9, 4.12 and 4.13)
- 3D liquid dam break, as in the papers of Gagnon *et al.* (2016) and Gagnon *et al.* (2019) (Fig. 4.10)
- 3D lava drop, as in the papers of Gagnon *et al.* (2016) and Gagnon *et al.* (2019) (Fig. 4.11)

As can be seen in the figures, as well as the accompanying video, our approach works well on 2D examples (Fig. 4.8), with 3D viscous fluids (Figs. 4.9), and with 3D fluids with no viscosity (Fig. 4.10). We also tested our approach with several texture types from structured to stochastic texture exemplars, as illustrated in Fig. 4.14. Table 4.1 reports statistics for some of our test cases. The patch and atlas creation steps are computed in parallel (on 12 cores in our tests). Atlas resolution was typically $6k^2$. Our tests were conducted on an Intel Xeon® Silver 4214 CPU. The majority of the time is taken by the patch update.

In the comparison with Gagnon *et al.* (2016) and Gagnon *et al.* (2019), we can see that our approach has no rigid artifact nor ghosting. The only visible artifact may be the visible patch boundaries when using large patches. We can also see that, when the texture exemplar has irregular patterns, such as pebbles or lava, our approach provides a significant visual improvement.

Figure 4.8 2D fluid with split and merge using texture exemplar and feature map from Fig. 4.7 and the resulting texture synthesis

Fig. 4.12 compares our results with the method of Gagnon *et al.* (2016). Their results show limited but noticeable distortion in curved regions. They use an orthogonal projection on each patch, for each frame of the animation, which can create more distortion than our flattening approach using Least Squares Conformal Maps (Lévy *et al.*, 2002). With their method, thanks to the use of rigid patches, on a still frame, we can see clearly every feature from the texture exemplar. With our approach we can notice more deformations. However, when looking at

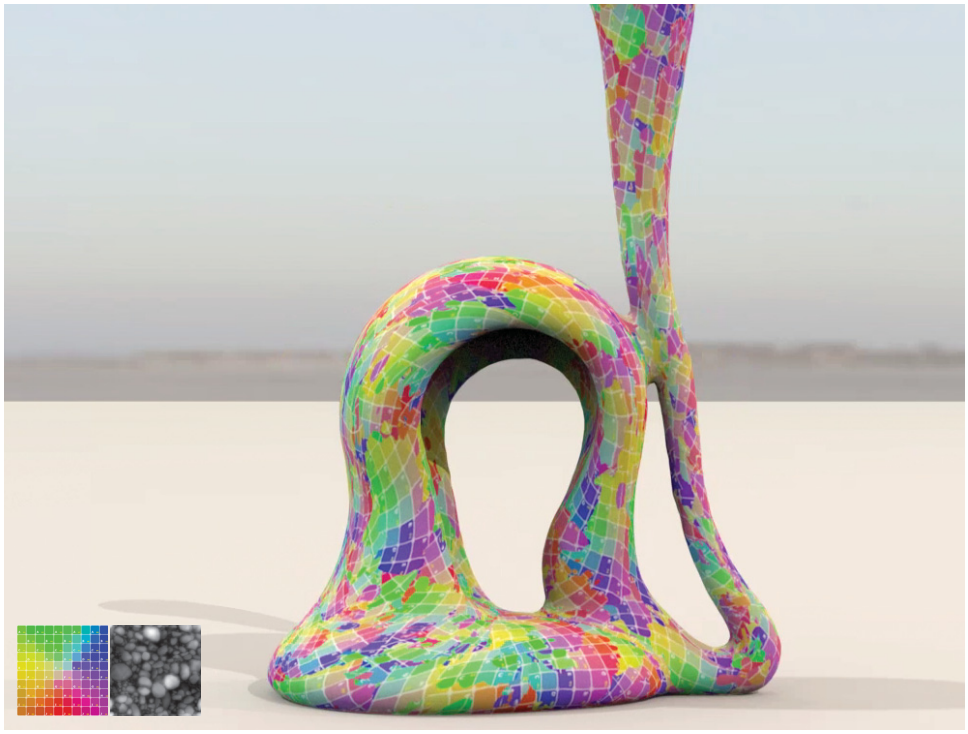


Figure 4.9 Viscous drop using colored squares as the texture exemplar. This example uses the same feature map as Fig. 4.7.

Table 4.1 Statistics of our examples. All times are in seconds per frame. Other steps in our process (included in the Total column), take very little time compared to the patch creation and update.

| | Number of Polygons | Number of Patch | Advection | Patch Creation | Patch Update | Total |
|---|---|---|---|---|---|---|
| Dam break | 85k | 4881 | 0.74 | 34.89 | 74.23 | 110.95 |
| Lava | 271k | 4664 | 2.07 | 96.69 | 197.39 | 96.15 |
| Viscous drop | 61k | 617 | 0.67 | 11.96 | 33.20 | 45.83 |

the animation of Gagnon *et al.* (2016) in the accompanying video, it is possible to identify individual patches: the rigid patches produce a visual artifact. Our approach is free of such artifacts thanks to the deformable patches.

Fig. 4.13 compares our results and those of Gagnon *et al.* (2019). Their results contain ghosting artifacts since they blend all overlapping patches to compute each texel. We are better able to preserve the exemplar structure; in the previous method, ghosting gives the impression of an

Figure 4.10 Dam break simulation using a pebbles texture exemplar

incorrect feature size. Moreover, as illustrated in the accompanying video, the fading of the patches can be visible when the fluid is slowing down, giving the impression that the update does not follow the velocity field. Conversely, our approach does not have any ghosting since only one patch texel is used for each output texel. Since we do not fade patches in or out using dynamic transparency, we also do not have visible fading that does not follow the velocity field. Our patch distribution update is also different as we do not have to delete patches too close to each other. Furthermore, we do not need to advect trackers with the patches as past methods did (Yu *et al.*, 2011; Gagnon *et al.*, 2019).

With the deformable lapped texture representation, the texture conforms well to the movement of the fluid surface. Fig. 4.15 and the accompanying video compare the fluid's velocity field

Figure 4.11 Result of our approach on a lava simulation. We synthesized both a color and a displacement texture.


Figure 4.12 Comparison between Gagnon *et al.* (2016) (left) and our approach (right)

(shown in blue) with the optical flow of the rendering (shown in green). The two fields are in close agreement, demonstrating that the texture synthesis is able to match the liquid flow.


Figure 4.13 Comparison between Gagnon *et al.* (2019) (left) and our approach (right)

## 4.8 Limitations

While our approach solves several problems (including ghosting, stiffness, and exemplar resolution), it has some limitations. Our approach relies on the exemplar texture having an accompanying feature map. When a height map is available, we can use that as a reasonable proxy. Arbitrary textures, however, may not have a feature map available, and hence the user must separately draw or compute one (e.g., using feature distance (Lefebvre & Hoppe, 2006)).

Regarding the mesh of the animated fluid, our approach is flexible and would work as well if the mesh is consistent or completely different from frame to frame. Nevertheless, our approach imposes some restrictions on the triangle mesh of the animated fluid, requiring reasonably homogeneous triangles. Flat regions of the fluid cannot be represented with larger triangles as this would be incompatible with the patch creation process. Moreover, elongated patch triangles will have a faster falloff in quality as they deform following the velocity field, and will unnecessarily accelerate the erosion. In our implementation, we used SideFX$^{TM}$ Houdini to create the mesh of the fluid, and we used the remesh operator to obtain triangles closer to the equilateral shape.

Another limitation is due to our texel removal strategy. When a patch is distorted and parts of it are being removed, it is sometimes possible to notice the moving boundary, especially when the patches are relatively big. The texel removal produces an illusion of motion as some portions of a patch are eliminated to reveal the patch beneath. This can be seen in the accompanying video.

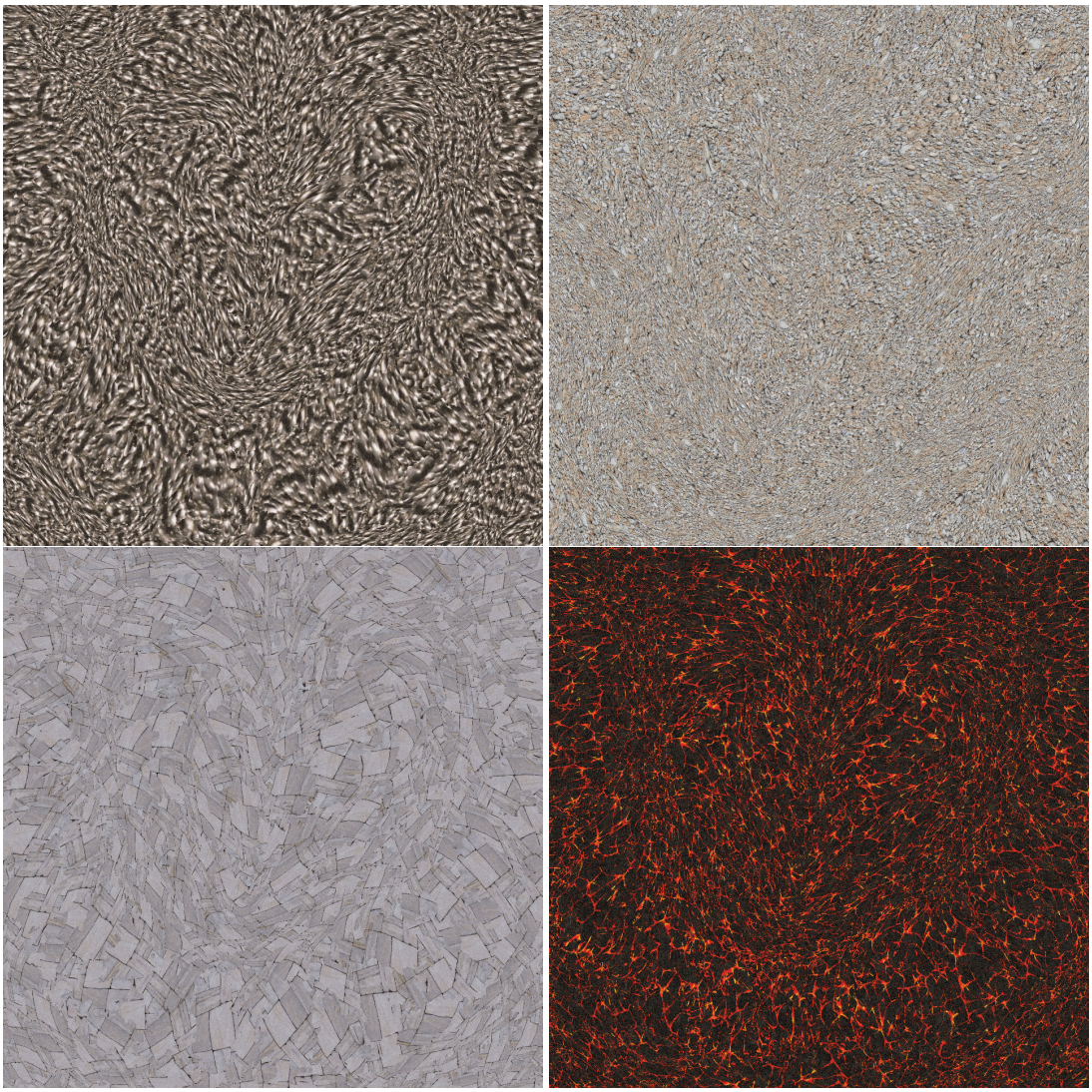


Figure 4.14 Split and merge test with various texture exemplars

Moreover, the erosion toward the boundary could result in a different look if the exemplar is significantly different at the center compared to the boundary. Using stationary exemplars, as we did in most of our examples, prevents this issue.

Another limitation comes from the patch update process. When we need to add a patch, it is added at the location of the first texel of the atlas that is not covered by any texel from the patches. However, this local strategy might be outperformed by a global strategy that takes

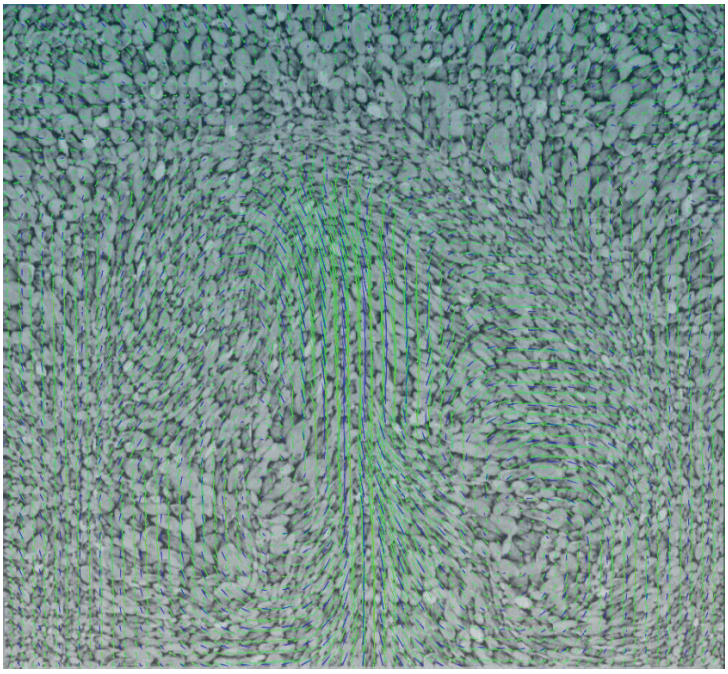


Figure 4.15 Test case demonstrating that the optical flow of the texture synthesis matches the fluid simulation's velocity field

into account all missing texels for the current frame. Future work could explore alternative distribution processes, but we have not noticed quality issues resulting from our current design.

## 4.9 Conclusion

In this chapter, we presented a new patch-based approach for synthesizing textures on fluid surfaces. Our approach extends lapped textures (Praun *et al.*, 2000) to fluids, building on past work by Gagnon *et al.* (2016). We use a feature-aware patch erosion approach in order to better preserve similarity to the exemplar. Using erosion rather than fading eliminates ghosting artifacts. Further, unlike methods depending on temporal fading, our texture stops changing when the underlying fluid stops moving. We showed numerous comparisons with previous work on fluid textures. We confirmed that our approach is coherent spatially and temporally by showing that the surface texture follows the velocity field of the fluid simulation, comparing the optical flow with the fluid velocities.

There are still some opportunities for future work. We would like to investigate more deeply the patch replacement strategy, ensuring that we get good surface coverage without excessive number of patches. Future work could explore automatic methods for computing a feature map from an exemplar. Finally, texturing surfaces with dynamic and non-homogeneous textures is an area of interest.

# CHAPTER 5

# DISCUSSION

In this chapter, we will discuss the results in perspective of the initial hypotheses. We will discuss the results and our contribution to the field of research.

Our first hypothesis was that we could use lapped textures (Praun *et al.*, 2000) on a fluid simulation if we can advect the patches using the surface velocity field. With the approach of Gagnon *et al.* (2016), described in Chapter 2, we were able to have patches that were moving according to the velocity field. The optical flow was close to the velocity field especially when the patches were small. Indeed, one limitation of advecting rigid patches is the bigger the patch, the more stiff the resulting animation looks. In term of contribution, it is the first approach that was able to advect patches on a free surface for a fluid simulation.

With patch-based texture synthesis, we cannot advect colors as in the method of Kwatra *et al.* (2007). We need a way to have an efficient patch distribution over the fluid animation. The second hypothesis was that it would be possible to update the patch distribution by adding new patches below existing ones where gaps were created by the advection. The result is an animation where, when the fluid is diverging, new textures appear in a natural fashion. This new patch update mechanism allows a patch animation avoiding popping artifact. This hypothesis is discussed in Chapter 2 and Chapter 4.

Advecting rigid patches introduced a stiffness artifact, especially with large patches. The third hypothesis was that we could use deformable patches to overcome stiffness in the resulting texture animation. This hypothesis has been experimented in the approach Distribution Update of Deformable Patches for Texture Synthesis on the Free Surface of Fluids (Gagnon *et al.*, 2019) and is discussed in Chapter 3. Thanks to the deformable patches, the optical flow was closer to the velocity field than the Dynamic Lapped Texture approach. Therefore, we were able to overcome the stiffness artifact. To ensure minimal distortion of the *uv* coordinate assignment over the 3D patch, we used the parameterization given by Least Squares Conformal

Maps (Lévy *et al.*, 2002), but our approach is versatile and could have used another method like Ghafourzadeh *et al.* (2020). It was the first approach where texturing a 3D fluid with deformable patches was possible.

Advecting deformable patches introduced distortion, which we dealt with using temporal fading as in the method of Yu *et al.* (2011). However, using fading to removed distorted patches introduced ghosting. The fourth hypothesis was that we could do a patch erosion to remove distorted patch to overcome ghosting. This last hypothesis has been experimented in the approach Patch Erosion for Deformable Lapped Textures on 3D Fluids (Gagnon *et al.*, 2021) discussed in Chapter 4. With the erosion mechanism, we were able to remove the ghosting artifact. The main contribution of the approach is the removal of deformed patches based on their distortion and texture features.

# CONCLUSION AND RECOMMENDATIONS

We have presented three approaches to texture the surface of fluids that handle any kind of topological changes including splashes. This thesis is the first to achieve patch advection on free surfaces.

In the first approach presented in Chapter 2, 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 first approach works well with rotational flows, thanks to the introduction of the tangent tracker. The 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.

The second approach, presented in Chapter 3, creates textured patches with limited distortion, based on a Poisson disk distribution. We want a good distribution for the specific case of fluids with small details, such as splashes and thin films, and we need to account for patches that will be wrapped on the fluid's surface. The update of the distribution and of the patches fully supports topological changes. We presented a more elaborate temporal fading out of patches, which allows control over their accumulation; this is most noticeable in areas where the flow converges or where different regions of the surface come into contact. Finally, we showed that the surface texture follows the velocity field of the fluid simulation, demonstrating this by comparing the optical flow with the fluid velocities.

The third approach, presented in Chapter 4, extends lapped textures (Praun *et al.*, 2000) to fluids, building on the work of Chapter 2. We use a feature-aware patch erosion approach in order to better preserve similarity to the exemplar. Using erosion rather than fading eliminates ghosting artifacts. Further, unlike methods depending on temporal fading, our texture stops changing when the underlying fluid stops moving. We showed numerous comparisons with

previous work on fluid textures. We confirmed that our approach is coherent spatially and temporally by showing that the surface texture follows the velocity field of the fluid simulation, comparing the optical flow with the fluid velocity.

Each of our approaches has artifacts. The first approach has rigid artifact that is especially visible when patches are large. Therefore, using texture exemplars with large patterns is not suited with this method. However, using small patches with high frequency texture patterns works well as there is less divergence between the optical flow and the velocity field.

If the objective of the synthesis on the fluid is to have a blend of texture exemplar, the second approach would be appropriate, regardless of the patch size. It will work with regular or stochastic textures and will follow the fluid simulation accurately thanks to the deformable patches.

For large structured patterns, the third approach is recommended. The erosion of distorted patches ensures that the patterns are better preserved and that the texture will deform according to the flow. However, using large patches can introduce a wave artifact.

Overall, all the artifacts of these approaches are less visible when we use small patches and stochastic texture exemplars. However, using small patches requires much more patches to cover the fluid surface and can result in a very slow computation.

There are still some opportunities for future work. We would like to investigate more deeply the patch replacement strategy, ensuring that we get good surface coverage without excessive number of patches. Future work could explore automatic methods for computing a feature map from an exemplar. The blending function is another area of potential development, further reducing ghosting beyond the improvements seen in Chapter 3. Another avenue to reduce the ghosting could involve an extra step using a best-match search as in the method of Kwatra *et al.*

(2007), applied only in regions with excessive ghosting. Also, texturing surfaces with dynamic and non-homogeneous textures is an area of interest.

The control that the user has can also be considered as a limitation. Indeed, apart from the texture exemplar and the size of the patch, there are no other ways to change the output of these approaches. Future work could explore user control on texturing fluid surfaces.

As mentioned in Chapter 1, recently, neural networks have been used for style transfer on 2D smoke simulation (Christen *et al.*, 2020) and later extended on volumetric fluid animation, changing also the shape of 3D fluid surfaces (Kim *et al.*, 2020). Where these methods focus on volumetric data, our approaches concentrate on surface texture synthesis. Neural texture synthesis is a subject that has not been explore in this thesis. If we exclude the training time of the model, we could synthesis a 2D texture with the method of Zhou *et al.* (2018) much faster than our approaches. Indeed, the training time is several hours, but computing a 600x400 texture takes 4 to 5 milliseconds. One of the main limitation of our approaches is the computation time. We believe it would be possible to use neural texture synthesis to overcome this limitation.

## BIBLIOGRAPHY


Bargteil, A. W., Goktekin, T. G., O'brien, J. F. & Strain, J. A. (2006a). A semi-Lagrangian contouring method for fluid simulation. *ACM Trans. Graph.*, 25, 19–38.

Bargteil, A. W., Sin, F., Michaels, J. E., Goktekin, T. G. & O'Brien, J. F. (2006b, Sept). A Texture Synthesis Method for Liquid Animations. *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

Bargteil, A. W., Sin, F., Michaels, J. E., Goktekin, T. G. & O'Brien, J. F. (2006c). A texture synthesis method for liquid animations. *ACM SIGGRAPH 2006 Sketches*, (SIGGRAPH '06).

Barnes, C. & Zhang, F.-L. (2017). A survey of the state-of-the-art in patch-based synthesis. *Computational Visual Media*, 3(1), 3–20.

Bojsen-Hansen, M., Li, H. & Wojtan, C. (2012). Tracking surfaces with evolving topology. *ACM Trans. Graph.*, 31(4).

Christen, F., Kim, B., Azevedo, V. C. & Solenthaler, B. (2020). Neural Smoke Stylization with Color Transfer. *41st Annual Conference of the European Association for Computer Graphics, Eurographics 2020 - Short Papers, Norrköping, Sweden, May 25-29, 2020 [online only]*, pp. 49–52.

Dagenais, F., Gagnon, J. & Paquette, E. (2017). Detail-Preserving Explicit Mesh Projection and Topology Matching for Particle-Based Fluids. *Computer Graphics Forum*, 36(8), 444-457.

Dagenais, F., Gagnon, J. & Paquette, E. (2012). A Prediction-Correction Approach for Stable SPH Fluid Simulation from Liquid to Rigid. *Proceedings of Computer Graphics International 2012*.

Dagenais, F., Gagnon, J. & Paquette, E. (2016). An Efficient Layered Simulation Workflow For Snow Imprints. *The Visual Computer*, 32, accepted for publication.

Dagenais, F., Guzmán, J., Vervondel, V., Hay, A., Delorme, S., Mould, D. & Paquette, E. (2018a). Extended virtual pipes for the stable and real-time simulation of small-scale shallow water. *Computers  Graphics*, 76, 84–95.

Dagenais, F., Guzmán, J., Vervondel, V., Hay, A., Delorme, S., Mould, D. & Paquette, E. (2018b). Real-Time Virtual Pipes Simulation and Modeling for Small-Scale Shallow Water. *Proceeginds of VRIPHYS 2018 Workshop on Virtual Reality Interaction and Physical Simulation*.

Efros, A. A. & Freeman, W. T. (2001). Image Quilting for Texture Synthesis and Transfer. *Proceedings of SIGGRAPH '00*, (Annual Conference Series), 341–346.

Efros, A. A. & Leung, T. K. (1999). Texture Synthesis by Non-Parametric Sampling. *Proceedings of the Intl. Conf. on Computer Vision - Volume 2*, (ICCV '99), 1033–1038.

Gagnon, J., Guzmán, J., Vervondel, V., Dagenais, F., Mould, D. & Paquette, E. (2019). Distribution Update of Deformable Patches for Texture Synthesis on the Free Surface of Fluids. *Computer Graphics Forum*, 38(7), 491–500.

Gagnon, J., Dagenais, F. & Paquette, E. (2016). Dynamic Lapped Texture for Fluid Simulations. *The visual computer*, 32(6-8), 901–909.

Gagnon, J., Guzmán, J. E., Mould, D. & Paquette, E. (2021). Patch Erosion for Deformable Lapped Textures on 3D Fluids. *Computer Graphics Forum*, 40(2), 367-374.

Ghafourzadeh, D., Ramachandran, S., Lasa, M. D., Popa, T. & Paquette, E. (2020). Local Editing of Cross-Surface Mappings with Iterative Least Squares Conformal Maps. *Proceedings of Graphics Interface 2020*, (GI 2020), 192 – 205. doi: 10.20380/GI2020.20.

Heeger, D. J. & Bergen, J. R. (1995). Pyramid-Based Texture Analysis/Synthesis. *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, (SIGGRAPH '95), 229–238. doi: 10.1145/218380.218446.

Jamriška, O., Fišer, J., Asente, P., Lu, J., Shechtman, E. & Sýkora, D. (2015). LazyFluids: Appearance Transfer for Fluid Animations. *ACM Trans. on Graph.*, 34(4).

Kim, B., Azevedo, V. C., Gross, M. & Solenthaler, B. (2020). Lagrangian Neural Style Transfer for Fluids. *ACM Trans. Graph.*, 39(4).

Kwatra, V., Schödl, A., Essa, I., Turk, G. & Bobick, A. (2003). Graphcut Textures: Image and Video Synthesis Using Graph Cuts. *ACM Trans. Graph.*, 22(3), 277–286.

Kwatra, V., Essa, I., Bobick, A. & Kwatra, N. (2005). Texture optimization for example-based synthesis. *ACM Trans. Graph.*, 24, 795–802.

Kwatra, V., Adalsteinsson, D., Kim, T., Kwatra, N., Carlson, M. & Lin, M. (2007). Texturing Fluids. *IEEE Trans. Visualization and Computer Graphics*, 13(5), 939-952.

Lefebvre, S. & Hoppe, H. (2006). Appearance-space Texture Synthesis. *ACM Trans. Graph.*, 25(3), 541–548.

Lévy, B., Petitjean, S., Ray, N. & Maillot, J. (2002). Least Squares Conformal Maps for Automatic Texture Atlas Generation. *IN SIGGRAPH 02 CONFERENCE PROCEEDINGS*, pp. 362–371.

Mihalef, V., Metaxas, D. & Sussman, M. (2007). Textured Liquids based on the Marker Level Set. *Computer Graphics Forum*, 26(3), 457–466.

Müller, M., Charypar, D. & Gross, M. (2003). Particle-based fluid simulation for interactive applications. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (SCA '03), 154–159.

Narain, R., Kwatra, V., Lee, H.-P., Kim, T., Carlson, M. & Lin, M. (2007). Feature-Guided Dynamic Texture Synthesis on Continuous Flows. *EGSR '07*.

Neyret, F. (2003). Advected textures. *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 147–153.

Pérez, P., Gangnet, M. & Blake, A. (2003). Poisson Image Editing. *ACM Trans. Graph.*, 22(3), 313–318.

Praun, E., Finkelstein, A. & Hoppe, H. (2000). Lapped textures. *Proceedings of SIGGRAPH '00*, (Annual Conference Series), 465–470.

Roy, B., Paquette, E. & Poulin, P. (2020). Particle Upsampling as a Flexible Post-Processing Approach to Increase Details in Animations of Splashing Liquids. *Computers Graphics*, 88, 57–69.

Roy, B., Poulin, P. & Paquette, E. (2021). Neural UpFlow: A Scene Flow Learning Approach to Increase the Apparent Resolution of Particle-Based Liquids. *CoRR*, abs/2106.05143. Consulted at https://arxiv.org/abs/2106.05143.

Soler, C., Cani, M.-P. & Angelidis, A. (2002). Hierarchical Pattern Mapping. *ACM Trans. Graph.*, 21(3), 673–680.

Sorkine, O., Cohen-Or, D., Goldenthal, R. & Lischinski, D. (2002). Bounded-distortion piecewise mesh parameterization. *Proceedings of the conference on Visualization '02*, (VIS '02), 355–362.

Stora, D., Agliati, P.-O., Cani, M.-P., Neyret, F. & Gascuel, J.-D. (1999). Animating Lava Flows. *Proceedings of the 1999 Conference on Graphics Interface '99*, pp. 203–210.

Turk, G. (2001). Texture synthesis on surfaces. *Proceedings of SIGGRAPH '01*, (Annual Conference Series), 347–354.

Wei, L.-Y. & Levoy, M. (2000). Fast Texture Synthesis Using Tree-structured Vector Quantization. *Proceedings of SIGGRAPH '00*, (Annual Conference Series), 479–488.

Wei, L.-Y. & Levoy, M. (2001). Texture Synthesis over Arbitrary Manifold Surfaces. *Proceedings of SIGGRAPH '01*, (Annual Conference Series), 355–360.

Wei, L.-Y., Lefebvre, S., Kwatra, V. & Turk, G. (2009). State of the Art in Example-based Texture Synthesis. *Eurographics 2009, State of the Art Report, EG-STAR*.

Wu, Q. & Yu, Y. (2004). Feature Matching and Deformation for Texture Synthesis. *ACM Trans. Graph.*, 23(3), 364–367.

Yu, Q., Neyret, F., Bruneton, E. & Holzschuch, N. (2009). Scalable real-time animation of rivers. *Computer Graphics Forum*, 28(2), 239–248.

Yu, Q., Neyret, F., Bruneton, E. & Holzschuch, N. (2011). Lagrangian Texture Advection: Preserving both Spectrum and Velocity Field. *IEEE Trans. Visualization and Computer Graphics*, 17(11), 1612 -1623.

Zhou, Y., Zhu, Z., Bai, X., Lischinski, D., Cohen-Or, D. & Huang, H. (2018). Non-Stationary Texture Synthesis by Adversarial Expansion.

Zhu, Y. & Bridson, R. (2005). Animating Sand As a Fluid. *ACM Trans. Graph.*, 24(3), 965–972.

Zwicker, M., Pfister, H., van Baar, J. & Gross, M. (2001). Surface Splatting. *Proceedings of SIGGRAPH 01*, (Annual Conference Series), 371–378.