

Enhancing Realism and Visual Coherence in Fluid Simulation for Computer Graphics

by

Julián Edgardo GUZMÁN CORTÉS

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

MONTREAL, JUNE 26TH 2026

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC



Julián Edgardo Guzmán Cortés, 2026



This Creative Commons license allows readers to download this work and share it with others as long as the author is credited. The content of this work cannot be modified in any way or used commercially.

BOARD OF EXAMINERS

**THIS THESIS HAS BEEN EVALUATED
BY THE FOLLOWING BOARD OF EXAMINERS**

Prof. Eric Paquette, Thesis supervisor
Department of Software and IT Engineering, École de technologie supérieure

Prof. Mohamed Cheriet, Chair, Board of Examiners
Department of Systems Engineering, École de technologie supérieure

Prof. David Labbé, Member of the Jury
Department of Software and IT Engineering, École de technologie supérieure

Prof. Guillaume Gilet, External Independent Examiner
Faculty of Science, Department of Computer Science, Université de Sherbrooke

**THIS THESIS WAS PRESENTED AND DEFENDED
IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC
ON MAY 6TH 2026
AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE**

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to all the people and organizations who supported me throughout my doctoral studies.

First and foremost, I would like to thank my advisor, Professor Eric Paquette, for his constant guidance and support throughout this journey. His mentorship was fundamental to the completion of this work. I am also sincerely grateful to Professor David Mould from Carleton University in Ottawa for his involvement in the projects, as well as for his valuable advice, support, and expertise.

I would like to thank my colleagues and friends at the ÉTS multimedia lab, Jonathan Gagnon, François Dagenais, Valentin Vervondel, and Bruno Roy, for their continuous support and collaboration throughout the development of these projects. I am also grateful to my friends Pilar Motta, Dr. Srinivasan Ramachandran “Ashok” and Rodrigo Campos for their encouragement during difficult times. I would especially like to thank my parents and family for their unconditional love and support during this time.

I am grateful for the financial support provided by Mokko Studio and MITACS during the early years of my doctoral studies. I also thank OSSim Tech and NSERC for funding the real-time fluid simulation project, and I further acknowledge NSERC for supporting both texture synthesis projects. I am grateful to Professor Oliver Van Kaick and my colleague Christopher Palazzolo from Carleton University for their involvement in the texture synthesis projects. I also thank Folks VFX, which allowed me to work and continue my research during the challenges of the COVID-19 pandemic. Finally, I would like to thank SideFX for providing the Houdini licenses used throughout all the projects and during my doctoral research.

Augmentation du réalisme et de la cohérence visuelle pour la simulation de fluide en informatique graphique

Julián Edgardo GUZMÁN CORTÉS

RÉSUMÉ

Les simulations de fluides sont souvent utilisées dans un large éventail d'applications en informatique graphique, notamment dans les effets visuels, les jeux vidéo, la visualisation scientifique et la simulation médicale en temps réel. Malgré les avancées considérables réalisées dans les modèles physiques existants, obtenir des simulations réalistes combinant à la fois réalisme physique et haute qualité visuelle demeure un problème difficile. Dans cette thèse, nous apportons des contributions selon deux axes complémentaires de la simulation de fluides : l'amélioration du réalisme physique et l'amélioration de la qualité visuelle des simulations animées. Afin d'améliorer le réalisme physique, nous proposons une approche qui renforce le réalisme des simulations de liquides peu profonds en temps réel. Cette approche s'appuie sur le modèle des "tuyaux virtuels" (*virtual pipes*) et l'étend au moyen de cartes d'élévation multicouches. Elle introduit le concept de "tuyaux étendus" (*extended pipe*), capables de résoudre les écoulements à travers des passages entièrement inondés, ce qui n'est pas possible avec les techniques multicouches existantes. Deux variantes sont proposées : les tuyaux étendus alignés sur les axes, qui gèrent les passages alignés avec la grille, et les tuyaux étendus unifiés, qui traitent les passages à branches multiples et permettent l'écoulement du liquide à travers des passages d'orientation et de forme arbitraires, y compris autour des coins. En permettant un écoulement à travers des passages entièrement inondés, des cavités et des structures à branches multiples, cette approche améliore significativement le réalisme physique de la simulation. Concernant l'amélioration de la qualité visuelle, cette thèse se concentre sur l'enrichissement visuel des simulations de fluides en proposant des contributions relatives à la synthèse de textures à la surface de liquides animés. Nos données d'entrée comprennent une texture d'exemple, une séquence de maillages 3D (correspondant au liquide animé) avec une vitesse définie par sommet, ainsi qu'un champ d'orientation 3D initialisé à la première trame. Nous proposons une approche de synthèse adaptative multi-résolution qui équilibre la fidélité à la texture d'exemple et la cohérence avec le mouvement du fluide. Notre méthode simule le déplacement de la texture ainsi que du champ d'orientation au fil des trames, produisant une séquence cohérente de textures conformes à la géométrie de chaque trame. L'adaptativité est guidée par des mesures locales de distorsion en 2D et en 3D, qui déterminent quand il est nécessaire de resynthétiser ou de préserver le contenu résultant de la simulation du déplacement. De plus, un mécanisme de prévention du scintillement est introduit afin d'éviter des changements brusques et marqués de couleur, en imposant des transitions progressives dans le temps. Nous introduisons également une seconde stratégie de synthèse de texture : une méthode hybride pour les surfaces de liquides animés combinant une synthèse basée sur des pièces de texture avec notre synthèse adaptative multi-résolution fondée sur la texture d'exemple. Les principales pièces de texture sont extraites de la texture d'exemple d'entrée, tandis que les zones de fond restantes servent à créer une texture d'exemple de fond. Les pièces de textures sont déplacées par simulation selon le champ d'orientation, conjointement à un processus visant à maintenir leur distribution. En parallèle, la

texture d'exemple de fond est utilisée pour synthétiser les détails d'arrière-plan à l'aide de la synthèse adaptative multi-résolution. Les deux textures sont ensuite fusionnées afin de produire une texture animée finale qui préserve les caractéristiques distinctives de la texture d'exemple tout au long de l'animation. Ces approches de synthèse de textures donnent de bons résultats aussi bien pour des liquides à écoulement lent que pour des écoulements turbulents, et pour une variété de textures d'exemple, améliorant significativement la qualité visuelle des liquides animés.

Mots-clés: : Simulation de fluides, Simulation en temps réel, Tuyaux virtuels, Synthèse de textures, Informatique graphique

Enhancing Realism and Visual Coherence in Fluid Simulation for Computer Graphics

Julián Edgardo GUZMÁN CORTÉS

ABSTRACT

Fluid simulations are widely used in a broad range of applications within computer graphics, including the entertainment industry, visual effects, interactive games, scientific visualization, and real-time medical simulation. Despite considerable advancements in existing physical models, achieving realistic simulations that combine physical realism with high-quality visual appearance remains a challenging problem. In this thesis, we make contributions toward two aspects of fluid simulation: improving the physical realism and improving the visual quality of animated fluid simulations. Toward improving the physical realism, we propose an approach that enhances the physical realism of real-time shallow water simulations. The approach builds upon the virtual pipes model and extends it using multi-layered heightmaps. It introduces the extended pipes that can resolve flows through fully flooded passages, which is not possible with existing multi-layered techniques. Two variants are proposed: axis-aligned extended pipes, which handle passages aligned with the grid, and unified extended pipes, which handle multi-branch passages and allow liquid flow through passages of arbitrary orientation and shape, including around corners. By enabling reliable flow through fully flooded passages, cavities, and multi-branch structures, this approach significantly improves the physical realism of the simulation. Regarding the improvement of the visual quality, this thesis focuses on enhancing the visual richness of fluid simulations by making contributions related to texture synthesis on the animated free surfaces of fluids. Our inputs consist of a 2D exemplar, a sequence of 3D meshes (corresponding to the animated liquid) with per-vertex velocity, and a 3D orientation field on the first frame. We propose an adaptive multiresolution synthesis approach that balances fidelity to the exemplar and consistency with the fluid motion. Our approach advects the texture and the orientation field across frames, yielding a coherent sequence of textures conforming to the per-frame geometry. Adaptivity is guided by local 2D and 3D distortion measures, which determine when to resynthesize or preserve advected content. Additionally, a popping prevention mechanism is introduced to prevent sudden, drastic color changes by enforcing gradual color transitions over time. We also introduce another texture synthesis strategy: a hybrid texture synthesis method for animated fluid surfaces that combines texton-based texture synthesis with our adaptive multiresolution exemplar-based synthesis. The main textons from the input exemplar are extracted, while the remaining background areas are used to create a background exemplar. The textons are advected given the orientation field together with a process to maintain their distribution. In parallel, the background exemplar is used to synthesize the background details with adaptive multiresolution synthesis. The two textures are then merged to produce a final animated texture that preserves both the features of the exemplar and background details throughout the animation. These texture synthesis approaches perform well on both slow-moving and turbulent liquids, and across a variety of texture exemplars, significantly enhancing the visual quality of animated fluids.

Keywords: Fluid simulation, Real-time simulation, Extended virtual pipes, Texture synthesis, Advection, Computer Graphics

TABLE OF CONTENTS

	Page
INTRODUCTION	1
CHAPTER 1 LITERATURE REVIEW	5
1.1 Fluid Simulation	5
1.1.1 Real-Time Fluid Simulation	6
1.1.2 Virtual Pipes	7
1.2 Texture Synthesis	7
1.2.1 Vector Texture Synthesis	9
1.2.2 Texture Synthesis with Deep Generative Models	10
1.2.3 Solid Texture Synthesis	11
1.2.4 Texture Synthesis on Animated Mesh	12
CHAPTER 2 REAL TIME FLUID SIMULATION WITH EXTENDED VIRTUAL PIPES	13
2.1 Multi-Layered Virtual Pipes Simulation	15
2.2 Extended Pipes	17
2.2.1 Axis-aligned Extended Pipes	19
2.2.2 Unified Extended Pipe	19
2.3 Results	23
2.3.1 Axis-Aligned Extended Pipe Results	24
2.3.2 Unified Extended Pipes Results	27
2.4 Discussion	28
CHAPTER 3 TEXTURE SYNTHESIS ON FLUIDS	31
3.1 Appearance-space Texture Synthesis	33
3.2 Texture Synthesis on Fluids	36
3.2.1 First Frame Synthesis	38
3.2.2 Surface Parameterization	38
3.2.3 Advection	39
3.2.3.1 Texture Coordinates Advection	40
3.2.3.2 Orientation Field Advection	41
3.2.4 Stretching Detection	42
3.2.4.1 3D stretching	43
3.2.4.2 UV stretching	44
3.2.5 Synthesis Pyramid	46
3.2.6 Popping Reduction	47
3.2.7 Implementation Considerations	48
3.3 Results	51
3.3.1 Comparison	55
3.3.2 Ablation study	58

3.4	Discussion	59
CHAPTER 4 HYBRID EXEMPLAR-BASED TEXTON TEXTURE SYNTHESIS WITH MULTIREOLUTION BACKGROUND SYNTHESIS FOR ANIMATED FLUIDS		
4.1	Textons and Background Texture	63
4.1.1	Texton Extraction	65
4.1.2	Background Exemplar Generation	66
4.2	Hybrid Texture Synthesis on Animated Fluids	69
4.2.1	Texton Synthesis	70
4.2.2	Background Texture Synthesis	74
4.3	Results	79
4.3.1	Texton Extraction and Background Generation Results	80
4.3.2	Texton and Background Synthesis Results	81
4.3.3	Comparison	82
4.3.4	Limitations	83
4.4	Discussion	85
CONCLUSION AND RECOMMENDATIONS		87
LIST OF REFERENCES		91

LIST OF TABLES

	Page
Table 2.1	Statistics for all the real-time fluid simulation examples 23
Table 3.1	Average per-frame computation times (in seconds) for various stages of our approach 54
Table 4.1	Average per-frame computation times (in seconds) for various stages of our hybrid approach 84

LIST OF FIGURES

	Page
Figure 2.1	Flow blockage in passages with the classical VP method 15
Figure 2.2	Grid representation of the VP method 16
Figure 2.3	The VP method prevents flow between b_{in} and b_{out} (it is blocked at p_{in}) 18
Figure 2.4	Our approach permits flow between b_{in} and b_{out} using an extended pipe 20
Figure 2.5	Complex passages not handled by axis-aligned extended pipes 21
Figure 2.6	Unified extended pipes with many-to-many connections 22
Figure 2.7	The average simulation time per frame (excluding rendering) 24
Figure 2.8	Simulation of multi-layer VP scenario 25
Figure 2.9	Simulation of a tunnel with obstacles using axis-aligned extended pipes 25
Figure 2.10	Simulation of a container with passages using axis-aligned extended pipes 26
Figure 2.11	Simulation of a circular container with passages using axis-aligned extended pipes 26
Figure 2.12	Surgical simulation from a vertebra using extended pipes 27
Figure 2.13	S-Shape passage simulation using unified extended pipes 28
Figure 2.14	Cross-Shape passage simulation using unified extended pipes 28
Figure 2.15	X-Shape passage simulation using unified extended pipes 29
Figure 3.1	Synthesis process 32
Figure 3.2	Left: Input exemplar. Right: appearance-space exemplar 34
Figure 3.3	Appearance-space texture synthesis on a 3D mesh 35
Figure 3.4	Best-match search using a hollow window of texels (L-shaped corners) . 36
Figure 3.5	Pipeline of our approach 37

Figure 3.6	Advection of texture coordinates	40
Figure 3.7	Advection of the orientation field	42
Figure 3.8	Stretching detection: comparison without synthesis and with our approach	43
Figure 3.9	3D stretching detection process	44
Figure 3.10	UV stretching detection process	45
Figure 3.11	Typical atlas with UV coordinates	49
Figure 3.12	Results of our approach in the sphere scenario	52
Figure 3.13	Results of our approach in the double dam break scenario	52
Figure 3.14	Results of our approach in the viscous drop scenario	53
Figure 3.15	Results of our approach in the viscous dam break scenario	53
Figure 3.16	Comparison of synthesis results for various threshold values for 3D stretching, UV stretching, and popping detection	55
Figure 3.17	Synthesis results using atlases at different resolutions	56
Figure 3.18	Comparison of Kwatra <i>et al.</i> (2007) and Gagnon <i>et al.</i> (2019) vs. our approach	57
Figure 3.19	Comparison of Gagnon, Dagenais & Paquette (2016) vs. our approach ..	57
Figure 3.20	Comparison of Gagnon, Guzmán, Mould & Paquette (2021) vs. our approach	58
Figure 3.21	Ablation study results showing texture synthesis on fluid simulation over time	59
Figure 3.22	Random-walk best match loses features from the input exemplar. Selected frames from an animation	60
Figure 4.1	Segmentation of the input exemplar into textons and background areas ..	66
Figure 4.2	Constrained texture synthesis method that performs synthesis in a scan-line order using an L-shaped window neighborhood	67
Figure 4.3	Background texture synthesis through iterative inpainting	68

Figure 4.4	Window configurations with different shapes	68
Figure 4.5	Texton advection	71
Figure 4.6	Background exemplar and its appearance-space representation	74
Figure 4.7	Synthesis without texel usage verification	76
Figure 4.8	Synthesis without and with texel usage verification	79
Figure 4.9	Texton extraction and background texture synthesis results	80
Figure 4.10	Results showing only the synthesized background texture	81
Figure 4.11	Results of our hybrid synthesis approach	82
Figure 4.12	Comparison of Guzmán, Mould & Paquette (2026) (showing stretching) and our approach (no stretching)	83
Figure 4.13	Comparison of Guzmán <i>et al.</i> (2026) (lost features) and our approach (uniform textons distribution)	84
Figure 4.14	Texton extraction and background texture synthesis results	85

LIST OF ALGORITHMS

	Page
Algorithm 2.1	Simulation loop 17
Algorithm 3.1	Overview of our approach 39
Algorithm 4.1	Background Synthesis. Colored lines highlight the modifications introduced with respect to the approach presented in chapter 3 78

LIST OF ABBREVIATIONS

ANN	Approximate Nearest Neighbor
CNN	Convolutional Neural Networks
EP	Extended Pipes
GAN	Generative Adversarial Networks
PCA	Principal Component Analysis
SVD	Singular Value Decomposition
SWE	Shallow Water Equations
VFX	Visual effects
VP	Virtual Pipes
VR	Virtual Reality

INTRODUCTION

The simulation of physical phenomena in computer graphics has become an essential mechanism for both scientific exploration and realistic visual experiences. Visualizing complex phenomena through computer graphics enhances our understanding of natural behaviors and also enables immersive experiences for entertainment and for scientific and professional training. Among these phenomena, fluids are particularly challenging to reproduce accurately in computer graphics. From the entertainment industry, VFX and interactive games to real-time medical simulations, the capability to simulate and render fluids with both physical realism and visual richness remains a challenge. Achieving such realism requires methods with stable and efficient numerical simulation, and visually coherent representation of surface details such as textures. Despite current research, important limitations remain, especially in the computational efficiency, physical realism, and quality of the visual outcomes.

Simulating fluids in computer graphics involves capturing highly dynamic behavior, interactions with complex surfaces, and free-surface phenomena such as splashes. Fluid simulations can be performed offline or in real time, and each of these simulations has distinct constraints and objectives. For example, in the field of visual effects and feature animations, simulations are typically computed offline. These simulations can reproduce accurate results with high levels of detail and visual fidelity, but they are often computationally expensive. In contrast, for interactive applications such as video games or interactive medical simulations, achieving visually convincing fluid behavior requires real-time methods that are both stable and computationally efficient. In these simulations, each frame must be computed within a few milliseconds, as computational resources must be shared with other tasks, including user input, game logic, rendering, and other system processes.

Models, such as heightmap-based or shallow water approaches (Chentanez & Müller, 2010; Mei, Decaudin & Hu, 2007), enable real-time simulations of liquids while capturing essential

features like thin layers and interactions with obstacles. These methods effectively reproduce shallow water phenomena like waves, ripples, and surface interactions while ensuring interactive frame rates. Although heightmap-based approaches are efficient for real-time simulations, they fail to represent liquid behavior over complex terrains, that contains overhangs or cavities. To address such cases, Kellomäki (2014) introduced multilayer heightmaps with virtual pipes, which establish interconnections between layers, allowing liquid to flow across them. Such configurations are particularly important for scenarios such as medical simulations, where fluids, such as blood, flow below organs. Nevertheless, current multilayer approaches still fail to handle flow through fully flooded passages beneath obstacles. Once such passages become completely filled, the flow toward these regions stops, preventing further transfer. These shortcomings significantly reduce the fidelity of real-time simulations, particularly when modeling realistic liquid transport in environments with complex geometry. In chapter 2, we describe an approach for real-time shallow water simulation, building upon the virtual pipes model with multi-layered heightmaps. In this approach, we introduce the use of extended pipes, which resolve flow through fully flooded passages, a feature not possible with current multi-layered techniques.

In fluid simulation, achieving realistic rendering requires not only the use of physical models that provide accurate motion but also the incorporation of a convincing visual appearance. The film and game industries often need textures for visual effects, and they require methods that can synthesize large textures from a small exemplar. At the same time, simulated fluids are often used for visual effects, and while texturing the surfaces of these fluids is of great interest, doing so is very challenging. Texture synthesis on the surface of fluids requires special attention to ensure that the patterns do not exhibit distortions or discontinuities, while being temporally coherent. Furthermore, splashes result in a surface that splits and merges, amplifying concerns related to distortions and discontinuities of the texture.

Texture synthesis from example aims to generate an output texture that resembles an input exemplar without being an exact duplicate. Most texture synthesis methods generate 2D textures, which are difficult to map onto arbitrary surfaces without introducing distortions or discontinuities. Some methods allow the creation of textures for static 3D surfaces (Wei & Levoy, 2001; Lefebvre & Hoppe, 2006). However, such strategies are not directly suited to dynamic and constantly changing fluid surfaces. Only a few methods allow the synthesis of textures on the animated free surface of a fluid. Some of these methods suffer from a wash-out effect caused by their texture optimization (Kwatra *et al.*, 2007; Narain *et al.*, 2007), while others suffer from a variety of problems such as exhibiting stiff texture patterns, resulting in a texture exhibiting some ghosting of the patterns from the exemplar, and showing moving discontinuities at the boundary between patches through the animation (Gagnon *et al.*, 2016, 2019, 2021). In chapter 3, we describe a new approach for texturing fluids that preserves fidelity to a texture exemplar, maintains temporal coherence, reduces distortions, and keeps the continuity of the patterns, even during complex deformations and topological changes. Furthermore, we introduce an adaptive multiresolution synthesis approach that balances fidelity to the exemplar with consistency to the fluid motion. Given a 2D exemplar texture, an orientation field from the first frame, an animated velocity field, and polygonal meshes corresponding to the animated liquid, our method advects both the texture and the orientation field across frames. The resulting animated liquids exhibit textures that remain faithful to the exemplar while being temporally consistent, even in the presence of severe surface disruptions such as splashes.

While the approach in chapter 3 achieves good results, it is less effective for non-stationary textures, particularly those with distinctive features, such as rocks or bubbles. To address these limitations, chapter 4 introduces a hybrid texture synthesis approach for animated fluid surfaces that extends and integrates the texel-based method from chapter 3 with a texton-based method. The approach separates the most prominent structures of the texture (called textons) from the background features and synthesizes them independently, preserving the distinctive

texture details that are often lost or distorted in standard texel-based methods. The textures are transferred to the next frame with the current frame velocity, while the background texture synthesis populates the next frame background texture by pulling information from the current frame using the next frame velocity. An intermediate velocity is used to ensure consistency between the two processes, and the final textures are merged using alpha blending to produce smooth, temporally coherent results.

Together, the contributions presented in this thesis address both the physical and visual challenges of fluids in computer graphics. On the one hand, we enhance real-time shallow water simulations by extending virtual pipes and multilayer heightmaps, allowing the flow through fully flooded passages. On the other hand, we propose two novel approaches for texture synthesis on animated fluid surfaces, producing temporally coherent, distortion-free textures that faithfully preserve the appearance of a given exemplar. With these contributions, this work aims to provide more realistic and visually compelling representations of fluids for interactive and visual effects applications.

CHAPTER 1

LITERATURE REVIEW

This chapter reviews previous work in the main research areas addressed in this thesis. The first section examines existing approaches to fluid simulation, with particular emphasis on real-time methods suitable for small-scale scenarios, such as blood flow in VR surgical training environments. Small-scale scenarios refer to simulations where the volume of liquid is limited, whereas large-scale scenarios involve a larger volume of fluid (oceans, rivers, etc.). The second section reviews the state of the art in exemplar-based texture synthesis, focusing especially on techniques designed for animated surfaces, such as fluids.

1.1 Fluid Simulation

Animating natural phenomena, such as fluids, using physics-based simulation methods is a challenging task in computer graphics due to their complexity and computational cost. There are two main approaches in fluid simulation: mesh-free methods (Lagrangian approach) and mesh-based methods (Eulerian approach). In mesh-free methods, the fluid is treated like a particle system where each particle represents a tiny amount of the fluid with its own characteristics (such as position, velocity, density, etc.). On the other hand, in mesh-based methods, fixed points are defined in the space to measure the variation of fluid in time (such as velocity, density, temperature, etc.).

In the computer graphics field, considerable research on animating liquids has focused on offline simulations using either Eulerian (Enright, Marschner & Fedkiw, 2002), particle-based (Ihmsen, Orthmann, Solenthaler, Kolb & Teschner, 2014; Dagenais, Gagnon & Paquette, 2012), or hybrid (Jiang, Schroeder, Selle, Teran & Stomakhin, 2015) methods. Some work focuses computations on areas with more details using adaptive grid structures (Aanjaneya, Gao, Liu, Batty & Sifakis, 2017), narrow band surfaces (Ferstl, Ando, Wojtan, Westermann & Thuerrey, 2016), or adaptive particle radii (Ihmsen *et al.*, 2014). Recent work has explored target-driven liquid simulation techniques that guide simulations toward predefined fluid shapes while

maintaining physically plausible motion (Sivakumaran, Paquette & Mould, 2022). These methods are capable of producing good visual results, but they are generally too slow for real-time applications. Macklin & Müller (2013) were able to achieve impressive visual results by simulating and rendering more than 100,000 particles in real time using their position-based dynamics framework. While this method is real-time, its ability to reproduce smooth thin layers of liquid is limited.

1.1.1 Real-Time Fluid Simulation

To reduce computational costs, it is more efficient to use a heightmap to represent the fluid and perform a 2D simulation to update the liquid's height. While methods using heightmaps as such cannot exhibit some more complex behaviors such as splashes and wave crests, they are adequate for a broad range of scenarios. For example, intricate wave patterns in shallow water can be encoded as height displacements by particles (Yuksel, House & Keyser, 2007) or packets of similar wavelengths (Jeschke & Wojtan, 2017). Furthermore, heightmap methods can simulate arbitrarily thin films of liquids with no impact on the simulation resolution. Lee & O'Sullivan (2007) allowed some compressibility in their 2D simulation based on the Navier-Stokes equations and adjusted the liquid's height based on its density. While simple and efficient, this technique does not account for the underlying terrain elevation. By assuming a vertical anisotropy of the liquid's velocity, the Navier-Stokes equations can be simplified, resulting in the shallow water equations, which can be further simplified to the shallow wave equations (Kass & Miller, 1990). Several papers focus on implicitly solving these equations on a regular grid (Kass & Miller, 1990; Layton & van de Panne, 2002) or on triangular mesh surfaces (Wang, Miller & Turk, 2007; Angst, Thuerey, Botsch & Gross, 2008). Methods with an implicit integration maintain stability at larger timesteps, but are prone to a lot of diffusion as well as volume gain when using a large timestep. Furthermore, faster-moving boundaries require a smaller timestep, which can considerably increase the computation time. On the other hand, Chentanez & Müller (2010) showed that their explicit integration of the shallow water equations is able to simulate large-scale scenarios in real time. For their part, our experiments show that the explicit integration requires

a considerably smaller timestep for small-scale scenarios because of the larger ratio between the liquid’s velocity and the simulation cell size, which limits its use for real-time applications in that context.

1.1.2 Virtual Pipes

Another model for simulating shallow water, the virtual pipes (VP) method, was introduced by O’Brien & Hodgins (1995). It is based on the hydrostatic pressure difference between neighbor cells of a uniform grid. Liquid is transferred between them through virtual pipes connected at their bottom, and the simulation uses an explicit integration. This method has been extended to support multi-layered heightmaps in order to allow simulations above partially submerged floating obstacles (Kellomäki, 2014) and on more complex terrains with overhangs (Borgeat, Massicotte, Poirier & Godin, 2011). However these multi-layered methods completely block the liquid’s flow under obstacles (e.g., overhangs), which results in an unrealistic behavior. We thus expand the VP method and propose an extended pipe model that handles the flow underneath obstacles (see chapter 2). Furthermore, our experiments show that the VP method allows a considerably larger timestep than with the shallow water method of Chentanez and Müller Chentanez & Müller (2010) for small-scale scenarios.

1.2 Texture Synthesis

Exemplar-based texture synthesis methods take as input an exemplar texture and synthesize a new output texture that is visually the same as the exemplar. Broader information about the texture synthesis area can be found in the survey by Wei, Lefebvre, Kwatra & Turk (2009).

Pixel-based methods synthesize the output texture on a pixel-by-pixel basis. Efros & Leung (1999) pioneered this method, using a best-match search to find a neighborhood in the exemplar similar to the neighborhood in the output. Wei & Levoy (2000) accelerated the search with a specialized data structure and later used the same technique to synthesize colors on 3D meshes (Wei & Levoy, 2001). While this is closer to what we want to achieve, they synthesize

the colors on the vertices of the mesh which links the resolutions of the mesh and the texture. This limits flexibility and makes it more complicated to generate high-resolution textures for lower-resolution meshes. Later, Lefebvre, Hornus & Neyret (2005) proposed a method for texturing 3D geometries at high resolution using texture sprites. Texture elements are splatted onto the surface, enabling efficient depiction of high-frequency detail. The method operates directly on the surface geometry without requiring a global surface parameterization. Lefebvre & Hoppe (2005) synthesize textures by working on the UV coordinates of the exemplar instead of RGB colors; this work was later extended (Lefebvre & Hoppe, 2006) to 3D objects by parameterizing the surface to a 2D atlas where the synthesis is conducted. Appearance-space texture synthesis (Lefebvre & Hoppe, 2006) introduced compressed search windows and drastically reduced synthesis time. It also supports 2D advection of the texture through a surface velocity field. While this is going in the direction of our work, it keeps the same atlas throughout the whole animation and as such does not support surfaces that evolve through time, like liquids. Another drawback of the work of Lefebvre & Hoppe (2006) is its long precomputation times; subsequent methods (Barnes, Shechtman, Finkelstein & Goldman, 2009; Busto, Eisenacher, Lefebvre, Stamminger *et al.*, 2010) replace the offline precomputation step with an online random walk search that is equally effective, greatly reducing computation. The approach presented in chapter 3 uses the appearance-space compression and atlas of Lefebvre & Hoppe (2006) together with the random walk search of Busto *et al.* (2010). We provide a significant extension to the method of Lefebvre & Hoppe (2006) which is limited to static meshes.

Texture optimization methods (Kwatra, Essa, Bobick & Kwatra, 2005; Darabi, Shechtman, Barnes, Goldman & Sen, 2012; Kaspar, Neubert, Lischinski, Pauly & Kopf, 2015) optimize to find a set of overlapping windows of texels from the exemplar. Optimization was also used for synthesizing texture based on a 2D orientation field (Liu, Jiang, Wong & Fu, 2012). On structured exemplars, such methods perform remarkably well. However, as noted by Jamriška *et al.* (2015), applying them to more stochastic textures tends to produce *wash-out*, where detail is lost in regions of the output texture. Jamriška *et al.* (2015) reduce wash-out by preventing excessive reuse of the same window of texels from the exemplar. Despite these improvements,

most optimization methods (Kwatra *et al.*, 2005; Darabi *et al.*, 2012; Kaspar *et al.*, 2015; Liu *et al.*, 2012) are limited to synthesizing static 2D textures. While the method of Jamriška *et al.* (2015) can synthesize animated textures, the textures remain 2D and cannot be applied to dynamic surfaces.

Some texture synthesis methods involve representing or decomposing texture exemplars into multiple components before synthesis. For instance, Lockerman *et al.* (2016) introduced a multi-scale label-map representation that captures texture structures at different scales and is used to guide texture synthesis, particularly for non-stationary textures. Guingo, Sauvage, Dischler & Cani (2017) propose a bi-layer texture model that decomposes a texture into a structural layer capturing large-scale patterns and a noise layer capturing fine-scale Gaussian patterns. This model combines the two layers, enabling structural deformations to create variations without introducing repetitive artifacts, while preserving fine-scale consistency. These methods are primarily designed for 2D texture synthesis. In chapter 4, we present an approach that decomposes the exemplar into components and extends this concept to animated meshes.

1.2.1 Vector Texture Synthesis

In contrast to traditional raster-based approaches, vector texture synthesis works with structured geometric primitives instead of pixel grids. Rather than synthesizing color values on a regular grid, these methods process vector elements such as parametric shapes, enabling resolution-independent and structurally coherent pattern generation. Prior work has explored the synthesis of textures with repetitive patterns from input exemplars using vector representations. Hurtut *et al.* (2009) synthesizes stroke-based vector textures by capturing the appearance and spatial relationships of elements in the exemplar. Moreover, Tu, Wei, Yatani, Igarashi & Zwicker (2020) generate textures of continuous curves by considering the geometry and topology of the exemplar elements. Similarly, Tu, Wei & Zwicker (2022) produce textures by sampling groups of vector shapes in clusters, producing structured and patterned results. More recently, Palazzolo, van Kaick & Mould (2025) proposed a method for synthesizing vector textures from a single raster exemplar. Their approach segments the exemplar into three components: primary textons,

secondary elements, and background regions. The primary textons are converted into vector primitives and clustered based on their visual similarity, using the median color of texton pixels. For each texton, a descriptor is defined to capture the local distribution of neighboring textons in the exemplar, which helps guide the synthesis process. The synthesis is performed by placing the primary and secondary elements according to these descriptors, while a gradient field is constructed to reproduce the background appearance. Finally, the three components are merged into a single vector image. Despite some qualities, such as resolution independence, current vector texture synthesis approaches remain limited to 2D textures and have not been applied to animated 3D fluid surfaces. We found the approach of Palazzolo *et al.* (2025) to be of particular interest due to its decomposition of a texture exemplar into several components, in particular the primary textons. In chapter 4, we make significant changes to their approach (Palazzolo *et al.*, 2025), changing its texton extraction strategy, redefining the texton representation to be faithful to the full range of colors found in the exemplar, adding texton advection through a velocity field, transforming its texton rendering technique so that it works on 2D texture atlases, and adapting it to the context of synthesis on the 3D surfaces of liquids.

1.2.2 Texture Synthesis with Deep Generative Models

Texture synthesis with Convolutional Neural Networks (CNN) (Sendik & Cohen-Or, 2017) and Generative Adversarial Networks (GANs) (Zhou *et al.*, 2018) can generate a great variety of textures through the generative capabilities of deep neural networks. Relying extensively on image convolutions, these synthesis methods work very well to output rectangular 2D images. In the case of synthesis on 3D surfaces, the continuity, regularity, and grid-type layout between adjacent texels, required by the convolution kernels, is not present. The work of Kovács, Hermosilla & Raidou (2024) partly solves the problem by defining CNN operations on meshes, then minimizing the distances of VGG-19 features measured over the mesh and the exemplar. The process is slow (around 50 minutes for a 1024×1024 output) and produces blurry patch seams related to overlapping CNN footprints. Instead of using CNNs, the method of Hu, Desai, Abu Alhaija, Kim & Shugrina (2024) uses diffusion-based inpainting. It is successful at allowing

users to paint a texture on a 3D mesh. One drawback of the methods of Hu *et al.* (2024) and Kovács *et al.* (2024) is that they rely on resampling (from the CNN footprint or camera-space 2D diffusion) onto a 2D atlas which introduces a certain level of blurring. Similarly, while methods employing latent diffusion models for texture synthesis look promising (Yu *et al.*, 2024; Perla, Wang, Mahdavi-Amiri & Zhang, 2024), such methods generate 2D images and are remapped onto the mesh; many images must be generated from different views so as to fully cover the mesh. An alternative proposed by Mitchel, Esteves & Makadia (2024) synthesizes texture directly on a mesh; synthesized texture quality is good, although expensive per-texture training is a concern. None of these methods have so far been applied to animated meshes, and temporal coherence and fidelity to the surface motion will pose formidable obstacles to adopting methods based on deep neural networks for texturing fluids.

1.2.3 Solid Texture Synthesis

Solid texture synthesis generates textures over entire volumes, avoiding reliance on surface parameterizations and UV mapping. Such methods are expensive to execute and solid textures are costly to store, as the texel count increases as the cube of the spatial resolution. Kopf *et al.* (2007) introduced a method for generating 3D solid textures from 2D images. It combines non-parametric texture optimization with histogram matching. To improve synthesis speed, Dong, Lefebvre, Tong & Drettakis (2008) proposed a method that takes advantage of the GPU. Chen, Johan & Mueller-Wittig (2013) proposed a method based on tiling, deformation, and resampling techniques, capable of working in real-time. Takayama, Okabe, Ijiri & Igarashi (2008) synthesized local volumetric patches with partial overlap, producing visually coherent solid textures with sharp features and smooth color variations. While solid texture methods can synthesize volumetric textures and bypass the need for surface parameterization, challenges remain for dynamic or animated surfaces. In order to obtain plausible animated textures, the solid texture contents would need to be advected by the surface motion, a task not considered by the preceding methods. Advecting full 3D data would be computationally expensive compared

to advecting a 2D atlas, and furthermore, no practical solution currently exists for temporal coherence in this setting.

1.2.4 Texture Synthesis on Animated Mesh

Currently, few methods can address texture synthesis over the evolving surface of simulated liquids. The method of Neyret (2003) advects a texture, but is limited to 2D advection. In a similar way, the methods of Lefebvre & Hoppe (2006), Yu, Neyret, Bruneton & Holzschuch (2011), and Jamriška *et al.* (2015) support 2D advection, but they are not directly applicable to the problem of texturing 3D fluids. Methods like those of Bargteil, Sin, Michaels, Goktekin & O’Brien (2006), Kwatra *et al.* (2007), and Narain *et al.* (2007) are able to texture the free surface of fluids. However, they store colors per-vertex directly on the mesh, interlinking the mesh and texture resolutions, which is far from convenient. Since these three methods are extensions of the texture optimization method of Kwatra *et al.* (2005), they also suffer from wash-out with some textures. Guingo *et al.* (2020) propose a texture deformation method for deformable surfaces. The method performs texture warping at texel resolution in a parametric domain, preserving texture features during deformation. However, it assumes a fixed topology, making it unsuitable for evolving fluid surfaces. Yet other methods (Gagnon *et al.*, 2016, 2019, 2021) texture liquids by performing a patch-based texture synthesis. These patch-based methods present issues like stiff texture patterns (Gagnon *et al.*, 2016), visible discontinuities in color between patches (Gagnon *et al.*, 2021), and ghosting (Gagnon *et al.*, 2019). While these methods (Gagnon *et al.*, 2016, 2019, 2021) rely on a texture atlas like ours, rendering to the atlas from the patches is a time-consuming process.

CHAPTER 2

REAL TIME FLUID SIMULATION WITH EXTENDED VIRTUAL PIPES

This chapter focuses on real-time shallow water simulation at small scales using extended virtual pipes, which enhance the fidelity of simulations in environments with complex terrains. The work was initially introduced in the paper *Real-Time Virtual Pipes Simulation and Modeling for Small-Scale Shallow Water* by François Dagenais, **Julián Guzmán**, Valentin Vervondel, Alexander Hay, Sébastien Delorme, David Mould, and Eric Paquette, presented at the VRIPHYS 2018 Workshop on Virtual Reality Interaction and Physical Simulation (Dagenais *et al.*, 2018a). This work was awarded the Best Paper Prize and subsequently led to an invitation to extend the research for publication in the *Computers & Graphics* journal. The resulting paper is titled *Extended Virtual Pipes for the Stable and Real-Time Simulation of Small-Scale Shallow Water* by François Dagenais, Valentin Vervondel, **Julián Guzmán**, Alexander Hay, Sébastien Delorme, David Mould, and Eric Paquette (Dagenais *et al.*, 2018b).

The research presented in these publications makes five main contributions:

1. Physically-based stable viscosity model
2. **Extended pipes interconnection framework allowing flow through passages**
3. Independent multi-layered surface reconstruction
4. Surface optimization to prevent unwanted interpenetrations between the heightmap and an arbitrary mesh
5. Distance-based correction of the normals to enhance the specular shading of the meniscus

The main contribution to this thesis, outlined in bold above, is the Extended Pipes interconnection framework, which allows flow through passages. In the first publication (Dagenais *et al.*, 2018a), the framework was developed to handle flow through fully flooded passages beneath obstacles, limited to axis-aligned configurations. In the second publication (Dagenais *et al.*, 2018b), the framework was further extended to support flow through passages of arbitrary orientation and shape, including configurations around corners. The other contributions that were mainly carried

out by the co-authors are not detailed in this thesis; for more information, readers are encouraged to refer to the original publications.

Real-time shallow water simulation has applications in interactive systems, such as video games and medical simulators, in scenarios like spilled coffee, bleeding during surgery, or flowing paint. These scenarios involve thin layers of liquid moving across surfaces, which may eventually fill up small cavities or flow through narrow passages. In real-time applications, various systems such as haptic feedback, collision detection, and additional physics modules run concurrently on the same hardware as the fluid simulator. Therefore, the fluid simulator must be computationally efficient, enabling it to share resources while producing visually convincing fluid behavior.

Full 3D models can generate simulations with a high level of detail, but they are computationally expensive. In real-time contexts, these models can only operate at very coarse resolutions, which fail to accurately capture thin films of liquids. To address this, models such as those proposed by Chentanez & Müller (2010) and Mei *et al.* (2007) perform 2D simulations on a heightmap. These methods are capable of running in real-time, capturing thin liquid layers and reproducing small-scale shallow-water phenomena, including surface interactions.

Our approach is based on the virtual pipes (VP) method (O'Brien & Hodgins, 1995; Št'ava, Beneš, Brisbin & Křivánek, 2008). To support more complex terrain geometries that include overhangs and cavities, the virtual pipe methods have been extended to utilize multi-layered heightmaps (Kellomäki, 2014). These heightmaps incorporate interconnections that allow the liquid to flow among layers. Such configurations are important for different scenarios, such as when blood should flow below organs in a surgery simulation. Nevertheless, current multi-layered methods do not handle flow through fully flooded passages below obstacles. When such passages become completely filled, the flow to these cells stops, preventing further transfer of liquid while the passages remain filled, resulting in unrealistic behavior (Fig. 2.1).

The work presented in this chapter addresses the issue of liquid flow through fully flooded passages. We extend the multi-layered VP method by introducing extended virtual pipes, which are capable of handling flow under obstacles. We propose two approaches: the first is limited to

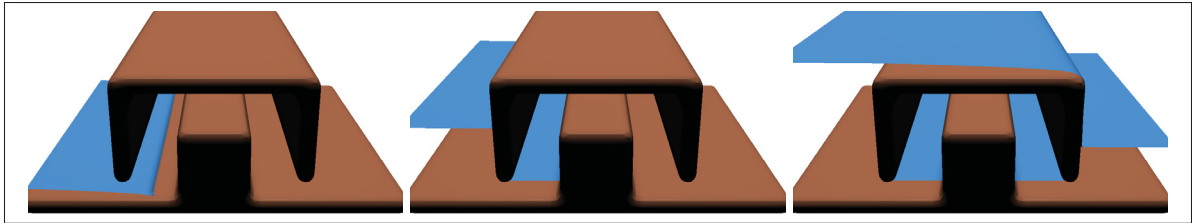


Figure 2.1 The fluid stops flowing within the passage when it reaches the maximum limit of the obstacle. Simulation parameters: resolution 100×100 cells, 2 layers, timestep 3 ms, cell size 0.0005 m

axis-aligned passages, while the second allows liquid flow of any orientation and shape including around corners. These enhancements improve the realism and visual fidelity of shallow water simulations under challenging conditions. In the following sections, we provide a detailed description of the extended virtual pipes framework. We begin with the implementation of our VP method (Sec. 2.1), followed by the presentation of the extended pipe framework (Sec. 2.2.1). Next, we introduce the extension of the extended pipe framework to support flow in passages of any orientation and shape (Sec. 2.2.2). Finally, we present the results (Sec. 2.3) and discuss the performance and visual fidelity of the proposed method (Sec. 2.4).

2.1 Multi-Layered Virtual Pipes Simulation

In this section, we describe our multi-layered VP model. Our base liquid simulation follows the VP method formulation introduced by O’Brien & Hodgins (1995), with the multi-layer structure of Kellomäki (2014). The VP method represents the fluid domain using a 2D simulation grid of cells, where each cell may contain one or more vertical columns in the multi-layered model. Each column corresponds to a range $[\min_i, \max_i]$ along the vertical axis, with base height b_i , liquid height h_i , and maximum range $\max_i$ (Fig. 2.2). Our terrain data comes from a 3D signed distance field representing solid objects. Throughout this chapter, we use the term *terrain* to refer to any obstacle (soft or rigid) lying below the columns of liquid. We define Δx as the size of a simulation grid cell, and we use the same value for the terrain cell size so that we capture all of the details from the terrain. At initialization and as needed when the terrain changes, the

column properties concerning the terrain ($\min_i$, b_i , and $\max_i$) are set by ray casting the terrain distance field upward through the center of a cell.

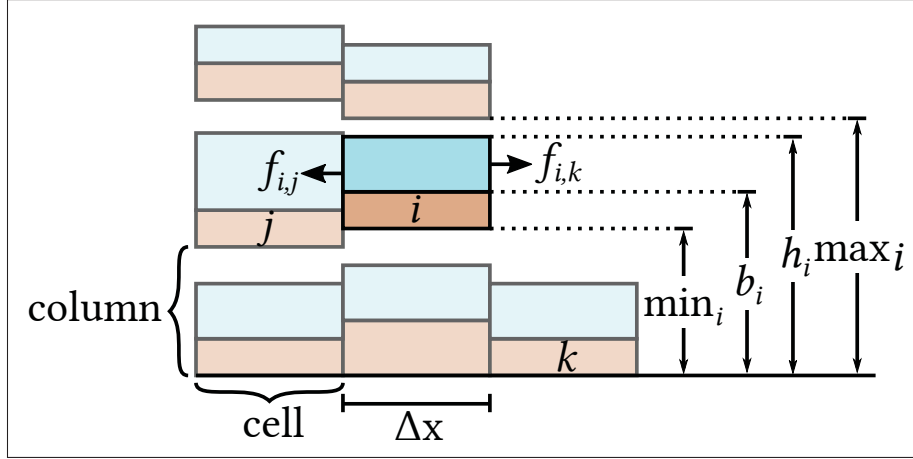


Figure 2.2 The simulation grid is divided into cells, which in turn contain one or more stacked columns. Here we see three linked adjacent columns i , j , and k , and column i 's properties: its range $[\min_i, \max_i]$ along the vertical axis, its base height b_i , and its liquid height h_i

Adjacent columns are connected using virtual pipes through which they exchange liquid. In the multi-layer case, a pipe connects two adjacent columns i and j when their liquid ranges $[b_i, \max_i]$ and $[b_j, \max_j]$ overlap. Note that one column can be connected to multiple columns from the same adjacent cell. As in the work of Mei *et al.* (2007), pipe connections are created only for the 4-neighborhood. These connections are created at the beginning of the simulation and adjusted when there is a change in the geometry of the obstacles. At each simulation step, the flux $f_{i,j}$ between a column i and its neighbor column j is updated using their difference in hydrostatic pressure and an explicit Euler integration:

$$f_{i,j}^{t+\Delta t} = \zeta f_{i,j}^t + \Delta t A \frac{g(h_i^t - h_j^t)}{l}, \quad (2.1)$$

where ζ is the friction parameter introduced by Mould & Yang (1997), Δt is the simulation timestep, g is the gravitational acceleration, A is the cross-section area of the virtual pipe, and l is the length of the virtual pipe. As in the work of Št'ava *et al.* (2008), we set l to be equal to the grid cell size Δx , and $A = \Delta x^2$. Note that damping is added in the simulation by scaling

the amount of flux from the previous timestep that is preserved using the friction parameter ζ , instead of adding a friction force in the simulation. This approach has the advantage of being stable for $\zeta \in [0, 1]$. We want ζ to have the same effect irrespective of the time step. As such, we propose to set $\zeta = \omega^{\Delta t}$, where $\omega = [0, 1]$ is the fraction of flux retained per unit time. In our examples, we set $\omega = 0.5$. Note that $f_{i,j} = -f_{j,i}$; thus we compute the flux once per pipe. In order to prevent the liquid height from dropping below the base height, the outflow fluxes are scaled as described by Mei *et al.* (2007). Similarly, to prevent the liquid height from going above the maximum height, we scale the inflow fluxes as explained by Kellomäki (2014). The scaled fluxes are then used to compute the new liquid heights, again using an explicit Euler integration:

$$h_i^{t+\Delta t} = h_i^t + \frac{\Delta t}{\Delta x^2} \sum_{j \in \text{neighbor}(i)} f_{j,i}^{t+\Delta t}. \quad (2.2)$$

The main steps of our simulation loop are shown in Alg. 2.1. Lines in bold are the steps that we added to the VP method to handle the flow under fully-flooded passages.

Algorithm 2.1 Simulation loop

- 1 **Update extended pipes connections (Sec. 2.2)**
- 2 **Update liquid fluxes, including those between extended pipes (Sec. 2.2)**
- 3 Apply viscosity
- 4 Scale liquid fluxes
- 5 Update liquid heights
- 6 Source liquid in the simulation

2.2 Extended Pipes

The multi-layer VP method can only exchange liquid between neighboring columns through height differences of their free surfaces. If there is a passage through or below obstacles, then as soon as the level of one column of the passage (Fig. 2.3, p_{in}) reaches its maximum, the VP method will block the flow of liquid for that column. This behavior occurs because when a column reaches the obstacle ceiling, no further increase in height is possible, and no additional fluid can be transferred to it. As a result, the VP model predicts zero flux between these

columns. However, in reality, liquid inside a fully flooded passage can continue flowing even when no height difference between adjacent columns is present. Since the VP method only allows exchange between neighboring columns, it cannot propagate flow from non-adjacent columns. Extending the flow therefore requires introducing additional connections beyond adjacent column interactions.

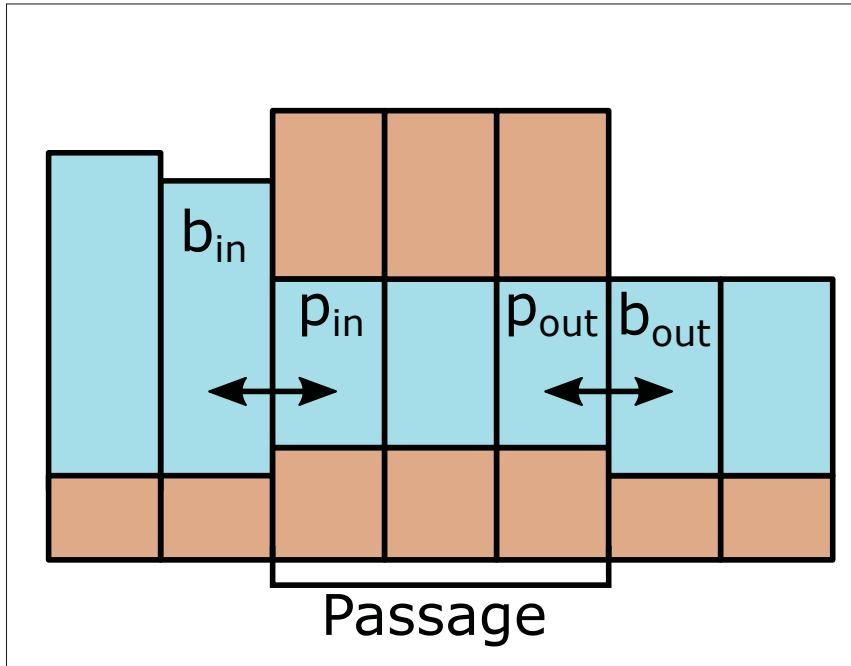


Figure 2.3 The VP method prevents flow between b_{in} and b_{out} (it is blocked at p_{in})

To address this limitation, we introduce extended pipes, which allow liquid to flow through fully flooded passages by creating direct connections between their endpoints. To allow liquid flow inside a passage, we identify fully-flooded passages and change the connections using extended pipes to link both ends of the passage (Fig. 2.4, Fig. 2.6). Thus, instead of only having connections between neighboring columns, we will also connect the ends of fully-flooded passages. Using extended pipes improves the simulation's ability to represent fluid flow beneath obstacles, enhancing visual realism without compromising real-time performance. In the following subsections, we present two variants of extended pipes: one designed for axis-aligned passages (Sec. 2.2.1), and a more general formulation that supports passages of arbitrary orientation and shape (Sec. 2.2.2).

2.2.1 Axis-aligned Extended Pipes

This section describes the first variant of our approach: *axis-aligned extended pipes*, as proposed in Dagenais *et al.* (2018a), which handle completely flooded passages aligned with the grid axes. A fully-flooded passage is identified by searching a group of consecutive connected columns having a level of liquid equal to their maximum limit. In this approach, we restrict the search for consecutive cells along two orientations, namely, alignment with the local x - and y -axes of the grid. By restricting the search for passages to the principal grid axes, the method ensures computational efficiency. The process of identifying fully-flooded passages, creating extended pipes, and updating fluxes can be performed in parallel on the GPU, allowing real-time simulation even in complex terrain.

Once a fully-flooded passage is identified, we use an extended pipe to make a connection between the columns at the boundary (b_{in} and b_{out} , Fig. 2.4). The flux between these boundary columns is initialized to zero to prevent the introduction of numerical imbalances. Our approach is compatible with VP method, allowing extended pipes to be incorporated without modifying its core operation. After adjusting the pipe connections of the columns on both sides of the passage, the standard VP method is used with the new connections. Extended pipes are dynamically connected and disconnected at each timestep based on the identified fully-flooded passages and the current fluid heights of the columns.

Extended pipes provide a solution by creating direct connections across fully flooded passages, allowing the fluid to flow smoothly from one end to the other and thereby enhancing the realism and visual fidelity of the simulation. However, because these connections are created and removed dynamically, the method may occasionally introduce small ripples when extended pipes are activated or deactivated, even when flux scaling is applied.

2.2.2 Unified Extended Pipe

The axis-aligned extended pipes discussed in Sec. 2.2.1 block the flow of liquid for each column that does not have an axis-aligned line-of-sight to another side of the passage. Furthermore,

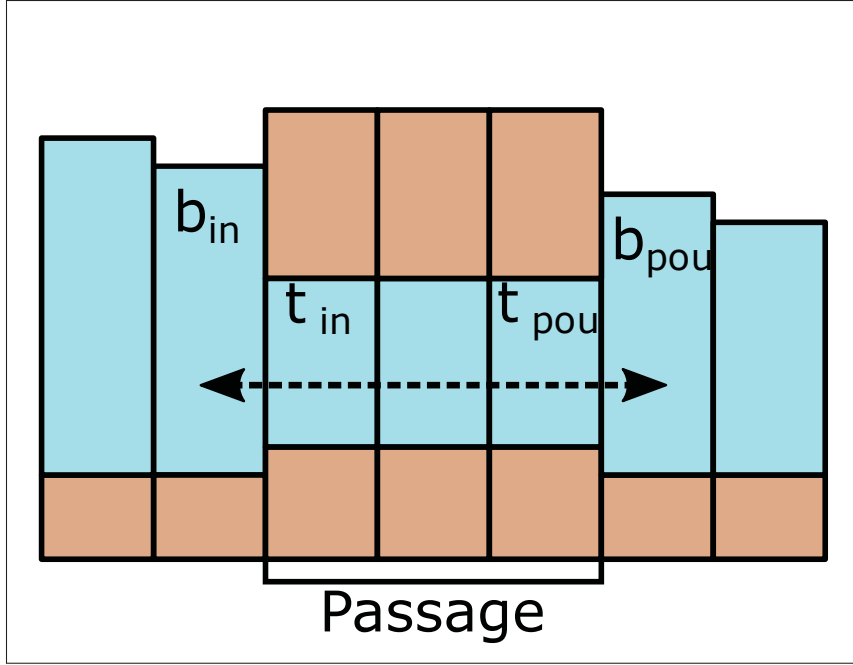


Figure 2.4 Our approach permits flow between b_{in} and b_{out} using an extended pipe

when the fluid turns inside the passage, liquid flow is drastically slowed because the columns at the beginning of the tunnel, from which the fluid originates, are no longer directly connected to columns along the new direction. Consequently, fluid transfer is limited to neighboring columns only, preventing a fast and continuous flow through curved or branching passages. As such, the axis-aligned extended pipes behave very poorly in scenarios exhibiting more complex passages, such as multiple branches or curves, where certain columns cannot establish a direct axis-aligned connection to the opposite boundary (Fig. 2.5).

To address this limitation, we introduce an alternative extended pipe approach, the *unified extended pipes* (Dagenais *et al.*, 2018b). Unlike the axis-aligned extended pipes variant, which establishes multiple independent one-to-one connections between columns facing each other on both sides of the passage, the *unified extended pipes* establishes a many-to-many connection for each passage. This unified connection ensures that liquid can flow through every part of the passage, independent of its geometric complexity, thereby improving the realism and stability of the simulation.

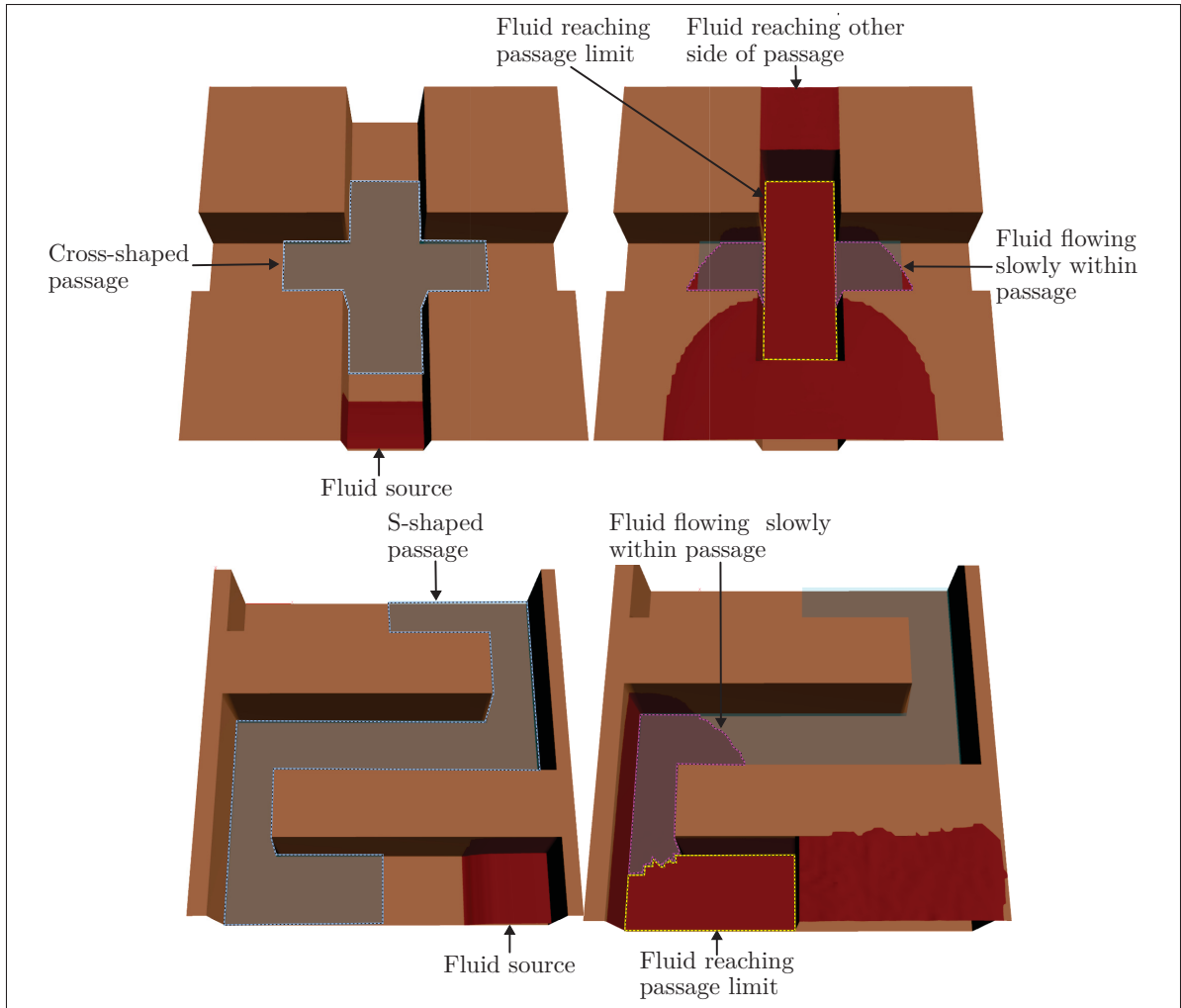


Figure 2.5 Examples of more complex passages that the axis-aligned extended pipes cannot handle (top: cross-shaped passage, bottom: S-shaped passage). The liquid flow is drastically slowed down when it should turn from the axis-aligned direction (pink). The simulated fluid is shown in red, and its apparent “climbing” along the walls is due to the grid mesh used to represent the fluid surface. The shaded area indicates the passage (blue), and the areas where the fluid reaches the passage limit are shown in yellow. See the video of the paper (Dagenais *et al.*, 2018b) for a comparison between the axis-aligned extended pipes and the unified pipe for the cross-shaped (left) scenario

For each passage P , we identify the set b_P of boundary columns connected to the passage (Fig. 2.6). These columns represent all valid entry and exit points of the passage and are used to establish the unified pipe connections. The selection of b_P ensures that liquid from every source column can reach all accessible destinations within the passage.

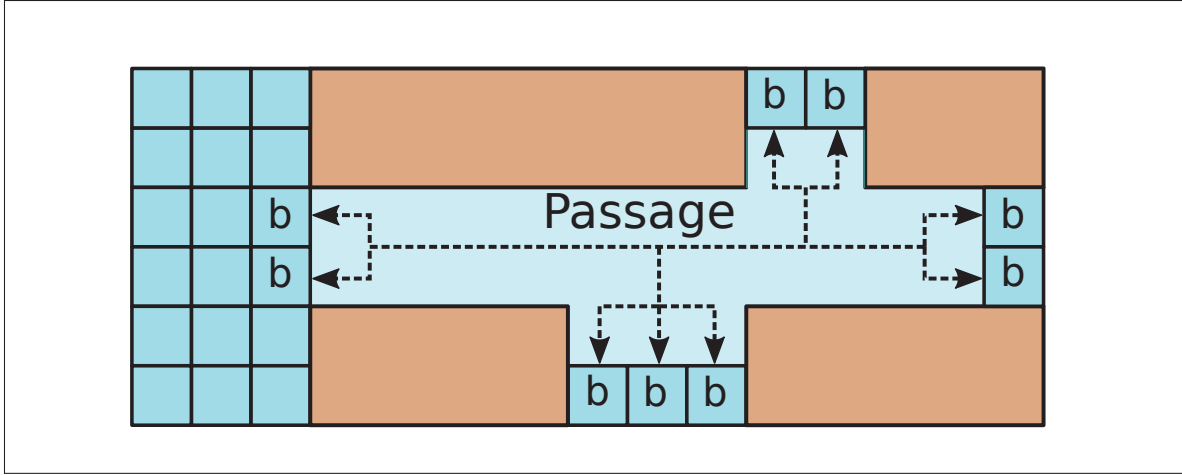


Figure 2.6 This figure presents the top view of a complex passage with several branches, some of different size. Each gateway to the passage cannot establish axis-aligned connections to each of the other gateways; multiple configurations of source of liquid will result in many gateways not getting their share of liquid. We thus use a many-to-many connection to ensure that all of the boundary columns at each gateway can exchange liquid

Once we have identified the boundary columns, we compute the liquid height average of these boundary columns $\overline{h_p^t}$ and we adapt Eq. 2.1 to compute the flux update between each boundary column b and the passage P :

$$f_{b,P}^{t+\Delta t} = \zeta f_{b,P}^t + \Delta t A \frac{g \left(h_b^t - \overline{h_p^t} \right)}{l} \quad (2.3)$$

Using the unified extended pipe, we need an extra step to ensure that the fluxes all around the passage sum up to zero. If the fluxes do not sum up to zero, which happens in most cases, we want to adjust them so that their sum is zero. This is equivalent to subtracting the average flux to each boundary flux around the passage:

$$f_{b,P}^{*t+\Delta t} = f_{b,P}^{t+\Delta t} - \overline{f_P}, \quad (2.4)$$

where $\overline{f_P} = \frac{1}{|b_P|} \sum_{i \in b_P} f_{i,P}$ is the average flux. These new fluxes are used in the same way with Eq. 2.2 to compute the new liquid heights.

Unified extended pipes, like axis-aligned extended pipes, are dynamically connected and disconnected at each timestep based on the identified fully flooded passages and current fluid heights of the columns. Despite the increased connectivity, the method remains highly efficient due to parallel GPU implementation. The results of this approach are presented and discussed in Sec. 2.3.

2.3 Results

In this section, we present the results of our tests evaluating the performance of the Multi-Layered Virtual Pipe model integrated with the extended pipes approach. Our method was tested across a wide range of scenarios to analyze its behavior and assess the impact of different simulation configurations, such as passage geometries and grid resolutions.

The multilayer Virtual Pipe model with extended pipes was implemented in C++, with a Qt-based graphical interface for scene configuration and visualization. The simulation core was implemented using CUDA for GPU parallel execution, significantly improving computational efficiency. Additionally, all tests were conducted on a machine equipped with an Intel Core i5-6600 CPU at 3.30 GHz, 16 GB of RAM, and a GeForce GTX 970 GPU.

Table 2.1 shows the measured times for the simulations, including the integration of the extended pipes approach. It also includes the computation times for other contributions from the original work by the co-authors, such as viscosity, surface tension, and rendering.

Table 2.1 Statistics for all the real-time fluid simulation examples: resolution ($N \times N \times Layers$), timestep Δt , cell size Δx , and kinematic viscosity ν as well as maximum/average timings (in ms) per frame for the simulation, surface generation, and rendering

Example	Resolution	Δt (ms)	Δx (m)	ν (m ² /s)	<u>Simulation</u>		<u>Surface</u>		<u>Rendering</u>		<u>Total</u>	
					Max	Avg	Max	Avg	Max	Avg	Max	Avg
Three layers (Fig. 2.8)	$100 \times 100 \times 3$	9.00	0.001	$4.0E-6$	0.68	0.32	0.85	0.37	0.89	0.17	1.53	0.86
Passages (Fig. 2.10)	$100 \times 100 \times 3$	9.00	0.001	$4.0E-6$	0.50	0.26	0.48	0.34	1.10	0.15	2.36	0.78
Surgery (Fig. 2.12)	$200 \times 200 \times 5$	3.00	0.0005	$4.0E-6$	5.43	4.86	2.11	1.10	0.66	0.34	7.04	6.30

As shown in Table 2.1 and Fig. 2.7, the simulation time grows approximately quadratically with the number of cells. Nevertheless, all of our examples run in real time, requiring less than 16.6ms per frame, leaving time for other computations such as soft body simulation, haptic feedback, rendering, and collision detection.

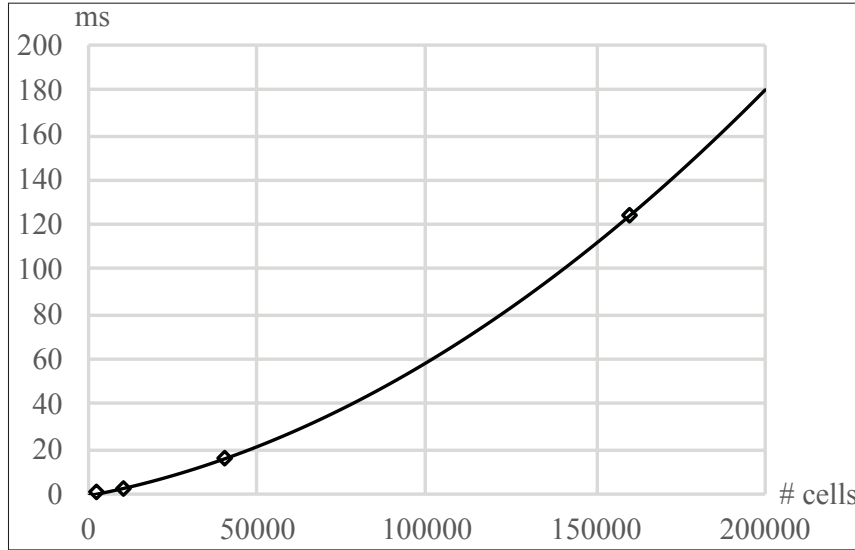


Figure 2.7 The average simulation time per frame (excluding rendering) is quadratic with respect to the number of cells

2.3.1 Axis-Aligned Extended Pipe Results

We first evaluate the performance and behavior of the axis-aligned extended pipe approach described in Sec. 2.2.1. This variant addresses passages aligned with the principal grid axes and establishes direct one-to-one connections between columns at opposite ends of fully flooded passages. We tested our approach with a wide range of scenarios, including straight passages of different lengths, widths, and orientations aligned with the grid axes. These tests allow us to assess the ability of axis-aligned extended pipes to enable liquid flow through passages while maintaining stability and real-time performance. The results of these tests are also available in the accompanying video of the original paper (Dagenais *et al.*, 2018a).

We evaluate the behavior of the multi-layered Virtual Pipe model using a three-layer scenario that includes cavities, a hole, and overhangs, as shown in Fig. 2.8. This scenario demonstrates

how the multilayer model handles vertical fluid distribution and interaction with the terrain while maintaining continuous flow.

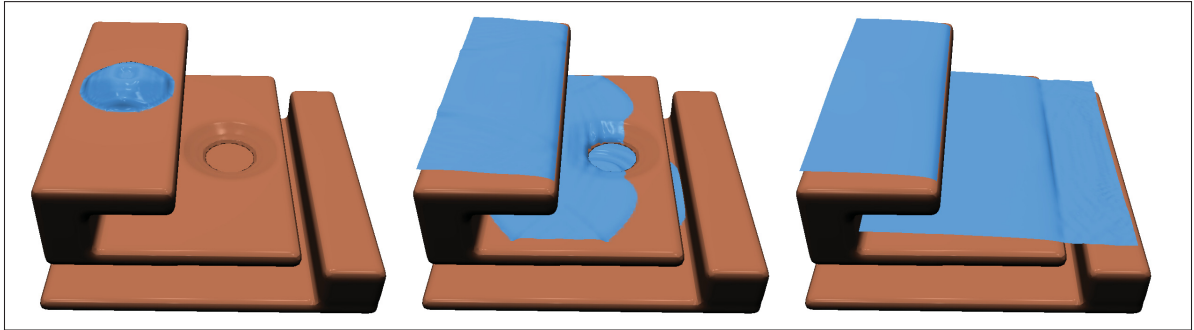


Figure 2.8 This scenario shows how the multi-layered Virtual Pipe model correctly handles complex geometries featuring cavities and overhangs. Simulation parameters: resolution 100×100 cells, 3 layers, timestep 9 ms, cell size 0.001 m

Next, we consider a tunnel scenario (Fig. 2.9), that contains several obstacles aligned with the grid axes. The goal of this test is to evaluate the ability of the extended pipes to maintain continuous liquid flow through the tunnel despite the presence of obstacles that would normally block the standard Virtual Pipe method. Axis-aligned extended pipes enable liquid to flow continuously through the passage, bypassing the obstacles, while the simulation remains stable and visually accurate.

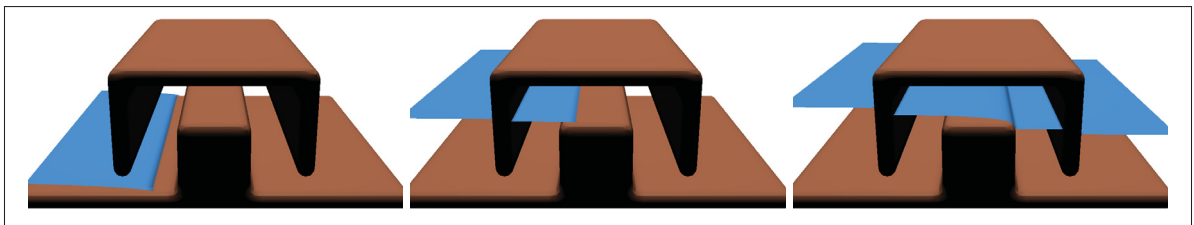


Figure 2.9 Simulation of a tunnel with obstacles using axis-aligned extended pipes. The fluid flows continuously through the passage despite obstacles. Simulation parameters: resolution 100×100 cells, 2 layers, timestep 3 ms, cell size 0.0005 m

In a more complex configuration, the four passages scenario (Fig. 2.10) shows how, after the passages are flooded, the liquid transfer does not stop, and is able to fill the other parts of the container. This ensures that the fluid propagates correctly even in more complex configurations.

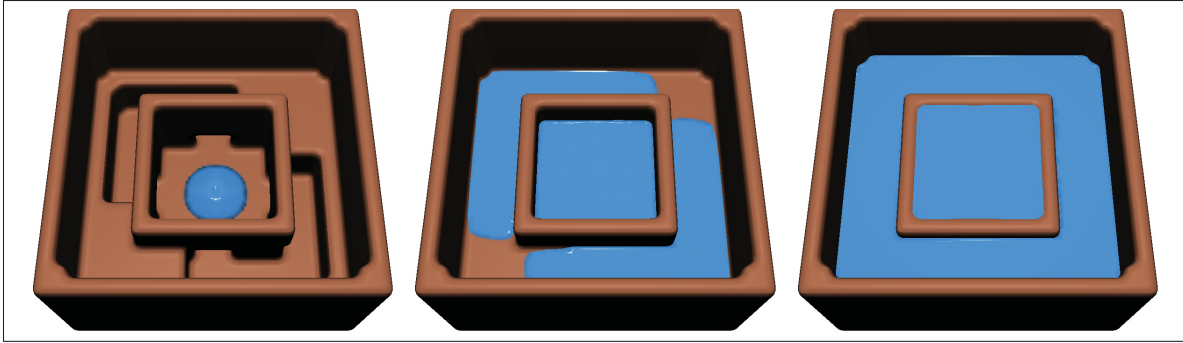


Figure 2.10 The liquid flows from the center and through four passages. When the passages are fully flooded, the extended pipes continue the transfer of liquid. Simulation parameters: resolution 100×100 cells, 2 layers, timestep 9 ms, cell size 0.001 m

The circular pool scenario with four tunnels (Fig. 2.11) illustrates how the fluid exits the container through the tunnels. As the liquid flows out, the level outside the pool rises until it matches the level in the central region, where the fluid source is located. This shows that the extended pipes effectively distribute the fluid across the connected passages.

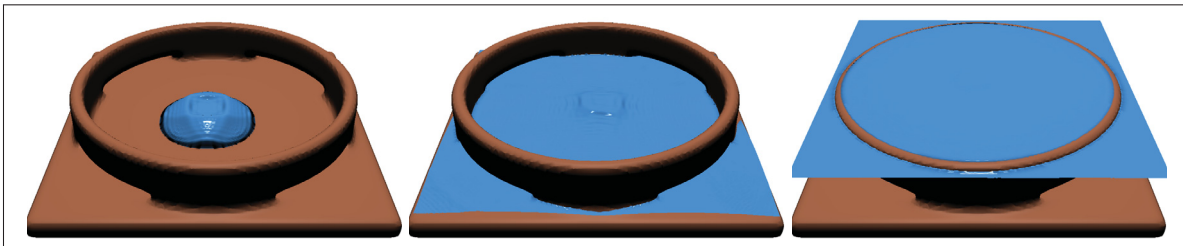


Figure 2.11 The fluid level outside the pool rises until it matches the level in the central region. Simulation parameters: resolution 200×200 cells, 2 layers, timestep 3 ms, cell size 0.0005 m

Finally, we validate our approach with a surgery simulation scenario, including multiple overhangs and holes. In Fig. 2.12, blood originates on top of a vertebra during surgery. As the simulation domain is filled, our surface correctly handles the evolving topology of the liquid surface, and no discontinuities are visible.

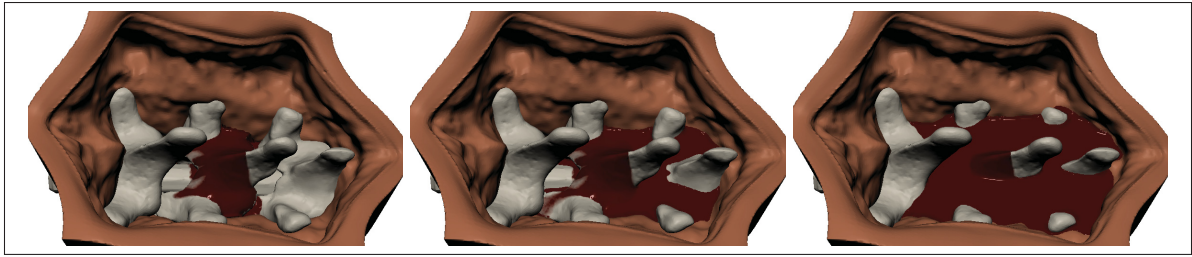


Figure 2.12 Frames from an animation where blood flows from a vertebra. Simulation parameters: resolution 200×200 cells, 5 layers, timestep 3 ms, cell size 0.0005 m

2.3.2 Unified Extended Pipes Results

We evaluate the performance and behavior of our Unified Extended Pipes approach, which enhances the axis-aligned variant to handle passages of arbitrary orientation and complex geometry. This new approach creates many-to-many connections across all boundary columns of a passage, ensuring that liquid can flow through every part of the passage, regardless of curves, branches, or intersecting tunnels, while also maintaining stability and visual accuracy.

We tested the unified extended pipes method on three challenging scenarios. In the first scenario, we test our approach in a curved S-shaped tunnel (Fig. 2.13). In this test, the unified extended pipes maintain continuous fluid flow along the curved path, ensuring that the liquid reaches the far end of the passage despite its curvature. This demonstrates the method's capability to manage changes in direction within a single passage while preserving stable flow.

To evaluate the ability of our approach to handle multiple branches within a single passage, we consider a cross-shape scenario (Fig. 2.14). In this scenario, the unified extended pipes maintain continuous fluid flow throughout the entire passage network, ensuring that the liquid flows through all branches without interruption.

Finally, we evaluate a more complex scenario consisting of diagonal tunnels forming an X-shape (Fig. 2.15). In this scenario, the unified extended pipes effectively manage intersecting passages, ensuring that the fluid is properly distributed across all directions, including diagonal paths.

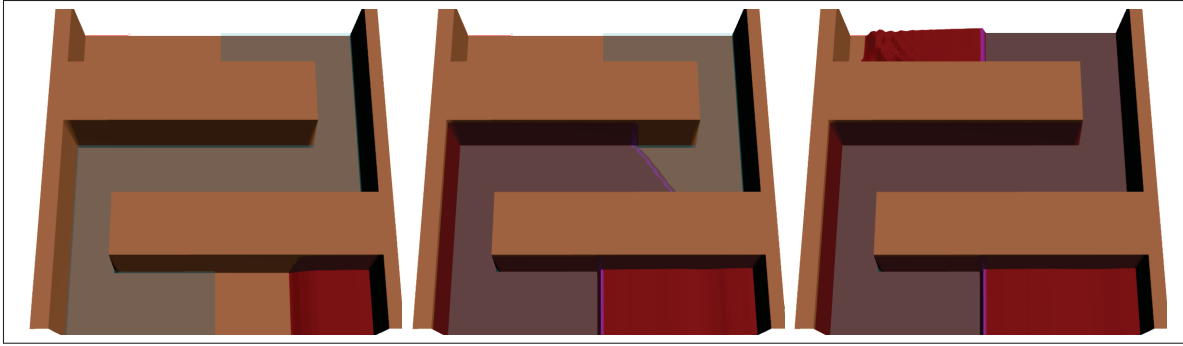


Figure 2.13 S-Shape passage. The fluid flows along the curved path, ensuring that the liquid reaches the far end of the passage despite the curvature. The shaded area indicates the passage. The red area corresponds to the simulated fluid, and its apparent “climbing” along the walls is due to the grid mesh used to represent the fluid surface. Simulation parameters: resolution 100×100 cells, 2 layers, timestep 3 ms, cell size 0.0005 m

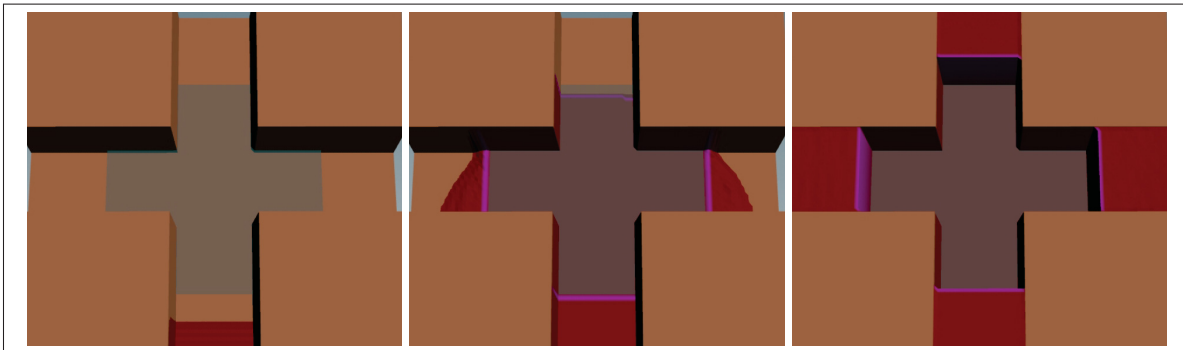


Figure 2.14 Cross-Shape passage. The fluid flows throughout all branches of the passage network, demonstrating the method’s capability to handle multi-branch configurations. The shaded area indicates the passage. The red area corresponds to the simulated fluid, and its apparent “climbing” along the walls is due to the grid mesh used to represent the fluid surface. Simulation parameters: resolution 100×100 cells, 2 layers, timestep 3 ms, cell size 0.0005 m

2.4 Discussion

In this chapter, we introduce the extended pipes approach to enhance the multi-layered virtual pipes method for real-time shallow-water simulation. The extended pipes significantly improve the visual realism of the simulation by enabling reliable flow through fully flooded passages, through cavities, and across multi-branch passages.

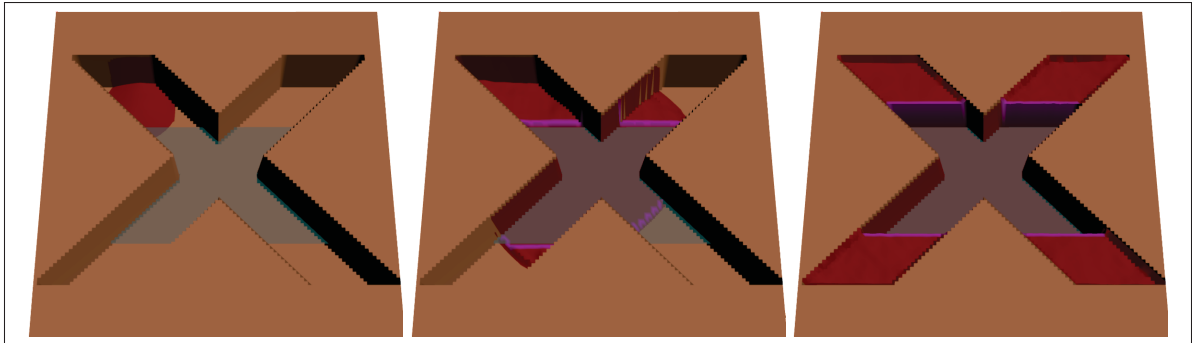


Figure 2.15 X-Shape passage. Fluid flows across all directions, including diagonal paths. The shaded area indicates the passage. The red area corresponds to the simulated fluid, and its apparent “climbing” along the walls is due to the grid mesh used to represent the fluid surface. Simulation parameters: resolution 100×100 cells, 2 layers, timestep 3 ms, cell size 0.0005 m

We introduce two extended pipes approaches: axis-aligned extended pipes, which can handle completely flooded passages aligned with the grid across multiple layers, and unified extended pipes, which can handle passages of arbitrary orientation and shape. The detection of passages and the connections of extended pipes are performed on a per-frame basis. Nevertheless, the approach remains computationally inexpensive, making it suitable for integration with other real-time systems commonly required in practical applications, such as surgical training and games. We have demonstrated that the approach operates in real time across various scenarios, including blood simulation for virtual surgery. Our experiments show that flow between the borders of passages does not interfere with other pipe connections, and that fluid transfer remains accurate without any loss due to the dynamic connection of the extended pipes.

Despite these improvements, some limitations remain. The extended pipes approach may introduce ripples when these pipes get connected and disconnected, even when using the scaling method of Mei *et al.* (2007). Furthermore, while axis-aligned extended pipes can work in scenarios with multiple layers, unified extended pipes are currently restricted to a single layer due to the computational cost of detecting passages and connecting boundary cells. Extending this mechanism to support multiple layers remains an open direction for future work. Additionally,

future work could extend our multi-layered virtual pipes model to handle dynamic scenarios in which the terrain changes at every frame or to support interactions with external objects.

CHAPTER 3

TEXTURE SYNTHESIS ON FLUIDS

The work described in this chapter presents an approach for synthesizing textures on the animated free surfaces of fluids. Our method introduces an adaptive multiresolution synthesis framework that balances fidelity to the exemplar with consistency to the underlying fluid motion. This work has been published in the article *Adaptive Multiresolution Exemplar-Based Texture Synthesis on Animated Fluids* by **Julián Guzmán**, David Mould, and Eric Paquette, in the journal *Computers & Graphics* (Guzmán *et al.*, 2026). In this chapter, we provide a detailed explanation of the approach and its underlying concepts.

Exemplar-based texture synthesis is a longstanding research topic in computer graphics. The goal is to synthesize an output texture that resembles an input texture exemplar without being an exact duplicate. The film and game industries often need textures for visual effects, and they require methods that can synthesize large textures from a small exemplar. At the same time, simulated fluids are often used for visual effects, and while texturing the surfaces of these fluids is of great interest, doing so is challenging. Texture synthesis on the surface of fluids requires special attention to ensure that the patterns do not exhibit too much distortion or discontinuity and to reduce temporally incoherent texture motion. Furthermore, splashes result in topological changes to the surface, amplifying concerns related to distortions and discontinuities of the texture.

Most texture synthesis methods generate 2D planar textures, which cannot be applied to surfaces without introducing discontinuities or distortions. Some texture synthesis methods allow the creation of a texture on a static 3D surface. Many such methods synthesize the texture colors on the vertices of the mesh, which improperly couples the geometric and texture resolutions. Other methods flatten the 3D surface into a texture atlas before conducting the texture synthesis, allowing independent mesh and texture resolutions. Only a few methods allow the synthesis of textures on the animated free surface of a fluid. These methods suffer from a variety of problems

such as exhibiting stiff texture patterns, producing ghosting of the patterns from the exemplar, and displaying objectionable discontinuities throughout the animation.

In this work, we present a new approach (Fig 3.1) for texturing fluids that aims to preserve fidelity to a texture exemplar, maintain temporal coherence, reduce distortions, and favor the continuity of the patterns, even during complex deformations and topological changes. Our

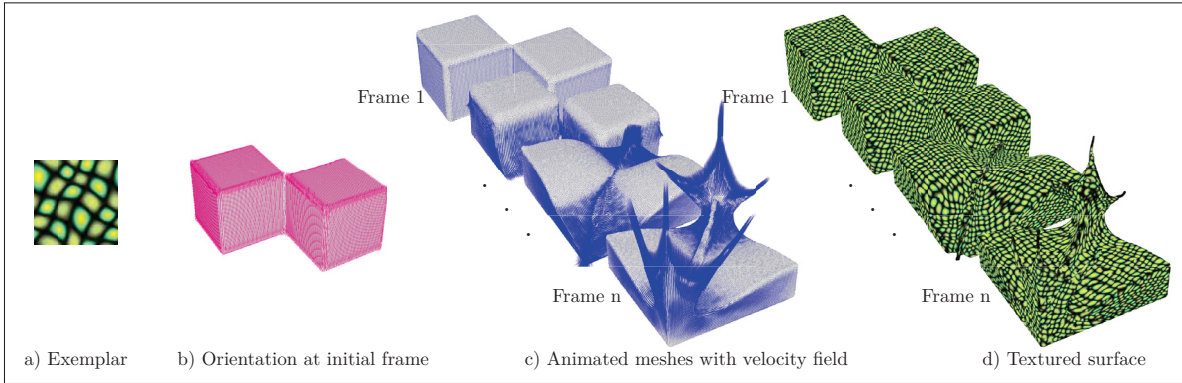


Figure 3.1 Our synthesis process. Inputs: a) texture exemplar, b) 3D mesh together with an orientation field, c) animated frames with velocity field. d) We synthesize an animated texture which can be used in off-the-shelf 3D software

approach requires as input a texture exemplar (we expect a stationary texture), an animation (velocity field together with sequence of meshes, obtained from a typical off-the-shelf simulator), and an orientation field on the first frame. An existing appearance-space method synthesizes the texture for the first frame. For subsequent frames, our approach uses the velocity field to advect the texture from frame to frame in 3D and in per-frame 2D atlas parameterizations; these are used to condition the synthesis to favor temporally consistent features.

Strict advection of the texture quickly results in heavily distorted features which diverge from the look of the exemplar, even if appearance-space texture corrections are applied to the texture. To address this, from the advected texture, we create a multiresolution *advection pyramid*, used together with a multiresolution *synthesis pyramid* which will contain the newly synthesized texture. One might imagine initializing the coarsest level of the synthesis pyramid from the coarsest level of the advection pyramid, and then creating higher-resolution levels by only

upsampling and conducting appearance-space texture correction. While this strategy produces good-quality textures that are faithful to the exemplar, the temporal continuity is very poor.

We thus introduce multiresolution per-texel decision criteria to set each synthesized texel either from the same-level advection pyramid (more faithful to the advected texture) or from upsampling the coarser synthesis pyramid level (more faithful to the texture exemplar). Our criteria to decide between the advection and synthesis pyramid are based on detecting stretching both in 2D and in 3D. Our approach also prevents frame-to-frame sudden and drastic changes in color (popping artifacts) by replacing them with gradual changes in color over a few frames.

Our final animated liquids show that our approach is successful in promoting textures faithful to the exemplar while encouraging temporal consistency, even in the presence of severe disruptions to the surface such as splashes. Our approach results include numerous examples of animated fluids with various textures, including both arbitrary textures and plausible textures such as foam. Our main contributions can be summarized as follows:

- Identification of locally distorted texels based on 2D and 3D features;
- Adaptive multiresolution texture synthesis approach with advection and synthesis pyramids;
- Advection of the texture and orientation field adapted to the appearance-space synthesis and texture atlases;
- Interpolation, extrapolation, and downsampling of the UV coordinates used in the appearance-space synthesis;
- Mechanism for gradual change in color to prevent popping artifacts.

3.1 Appearance-space Texture Synthesis

Since our approach extends the appearance-space texture synthesis method (Lefebvre & Hoppe, 2006), we will describe that method in more detail. To improve the synthesis quality and speed, their method transforms the texture exemplar into a compressed version. All of the 5×5 windows from the RGB exemplar form a set of vectors of $5 \times 5 \times 3$ dimensions onto which a Principal Component Analysis (PCA) decomposition is applied. The decomposition yields

the eigenvectors onto which all of the 5×5 windows of the exemplar are projected, yielding the PCA space coordinates. The compression takes place by discarding all but the first eight coordinates. This compressed representation is referred to as *appearance-space*. Figure 3.2 shows an exemplar and its corresponding appearance-space compressed version, obtained by mapping the first three coordinates in PCA space to the RGB channels for visualization purposes. Best-match search with the compressed exemplar compares only 8 floating point values instead

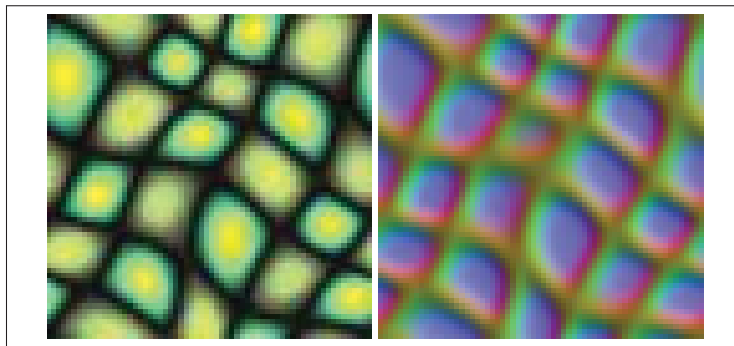


Figure 3.2 Left: Input exemplar. Right: appearance-space exemplar

of the equivalent $5 \times 5 \times 3$. Just as our approach does, the method of Lefebvre & Hoppe (2006) parametrizes 3D meshes to a 2D atlas where the output texture texels will be computed (Fig. 3.3, right). Instead of working with texture colors, the method uses the appearance-space 8D vectors. Furthermore, instead of storing such 8D vectors in the output texture directly, it stores UV coordinates (Fig. 3.3, center) linking back to the compressed exemplar where the 8D vectors are stored.

Lefebvre and Hoppe use texture pyramids for both the exemplar and the output texture to apply a hierarchical synthesis. After initializing the coordinates at the coarsest level of the output pyramid, the method starts at the next finer level of the output pyramid by upsampling from the coarser level. The upsampling step generates the intermediate UV coordinates necessary to cover the same UV range of the exemplar as the coarser texel did (and also accounting for an orientation field through its Jacobian, see original paper Lefebvre & Hoppe (2006) for further details). The correction step uses a best-match search with a hollow window around the output texel (Figure 3.4). For each output texel, the exemplar UV coordinates of the hollow window

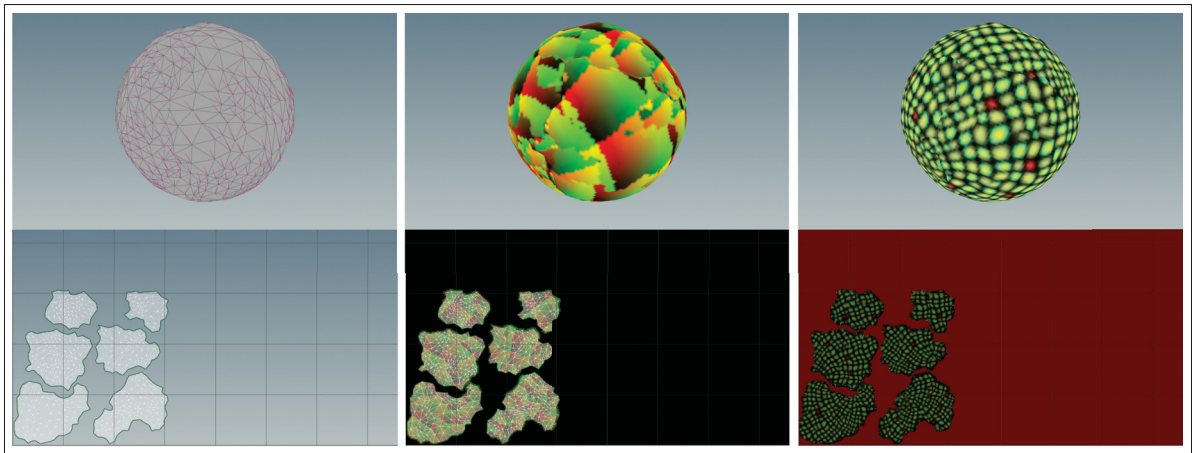


Figure 3.3 Left: Input 3D mesh with orientation field and 2D atlas. Center: Exemplar UV coordinates shown on the surface as well as in the atlas. Right: Exemplar UV coordinates converted to RGB colors

texels are temporarily converted to their equivalent 8D appearance-space vectors by fetching the corresponding 8D vectors from the compressed exemplar. The 8D values of each corner (L-shaped group of 3 texels) are averaged, leading to 4×8 values to be used for the comparison with the exemplar. The best match is the one with least L_2 distance on the 8D appearance-space vectors. A best-match search is then computed over hollow windows of the exemplar texels by comparing the 4×8 values from the output texel to those of the exemplar. The UV coordinates of the best-match from the exemplar are then written to the output texel. After computing all best-match searches of all levels, the output is filled with exemplar UV coordinates (Fig. 3.3, center) corresponding to the final texture. The last step is to perform a lookup process to get the colors corresponding to the UV coordinates (Fig. 3.3, right).

To avoid the precomputation required by the k-coherence best-match search used in Lefebvre & Hoppe (2006), the approximate best-match method of Busto *et al.* (2010) is employed. The search method of Busto *et al.* (2010) improves upon the k-coherence search of Tong *et al.* (2002) through a random walk best-match search. The approach searches at random locations in the exemplar around the previous best match. Each search is done within a decreasing search radius (decreased by half), similar to the PatchMatch method of Barnes *et al.*

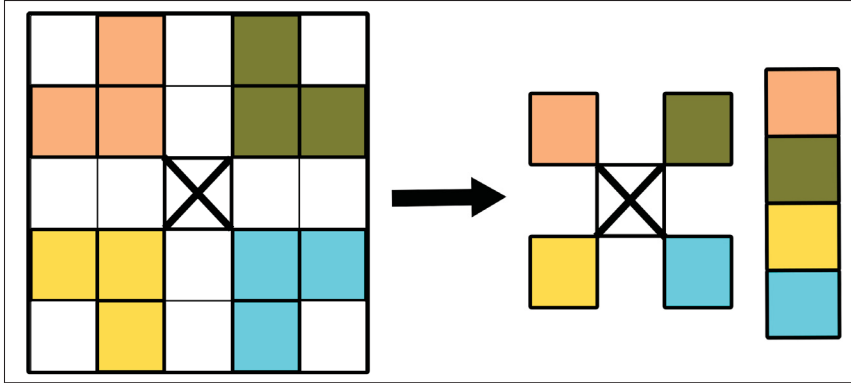


Figure 3.4 Left: Hollow window of texels (L-shaped corners) used to conduct the best match for the center texel. Right: Color values of the L-shaped corners are averaged and stacked together for the best-match

(2009), but by updating the center of the radius search at each iteration. The initial center of the search is set to the best candidate among the eight neighbor texels.

For the first frame of our animation, we apply the texture synthesis methods described in this section. For the subsequent frames, we designed a different approach so that we can deal with the animated meshes of the fluid, the advection, and maintaining a texture that is both faithful to the exemplar as well as temporally coherent.

3.2 Texture Synthesis on Fluids

We aim to texture the deforming surface of a 3D fluid animation, based on a 2D texture exemplar. The novel texture should be faithful to the exemplar, despite synthesizing it on a curved 3D surface; this implies reducing spatial discontinuities as well as spatial stretching and compression of the patterns. Furthermore, the texture evolves temporally which often produces popping. To address these challenges, we introduce a new adaptive, multiresolution texture synthesis approach.

The main inputs to our approach are the texture exemplar, the animated fluid provided in the form of per-frame meshes and velocity field (which are easy to obtain from typical fluid simulation software), and a 3D orientation field for the first frame (used to orient the texture features). The

output of our approach consists of a set of per-frame 2D textures, each parameterized to the corresponding per-frame mesh.

On the first frame, we synthesize a texture over the mesh with the method of Lefebvre & Hoppe (2006) which synthesizes UV coordinates that index back to the exemplar. This initial texture closely resembles the exemplar; no complications due to fluid motion have yet influenced the texture appearance.

At each subsequent frame, we execute multiple steps to synthesize that frame’s 2D texture (Fig. 3.5). First, we advect both the texture and the orientation field from the previous frame using the provided velocity field. A multiresolution *advection pyramid* is populated by downsampling the advected texture from fine to coarse (Sec. 3.2.3) up to the number of levels in the exemplar, $\log_2(w)$ where w is the exemplar width in texels. As we typically synthesize at a resolution higher than the exemplar, the advection pyramid is truncated at $\log_2(w)$ levels. The advection pyramid contains texture that can be used at the current frame, but may contain deficiencies such as stretching, which the remainder of our synthesis process strives to address.

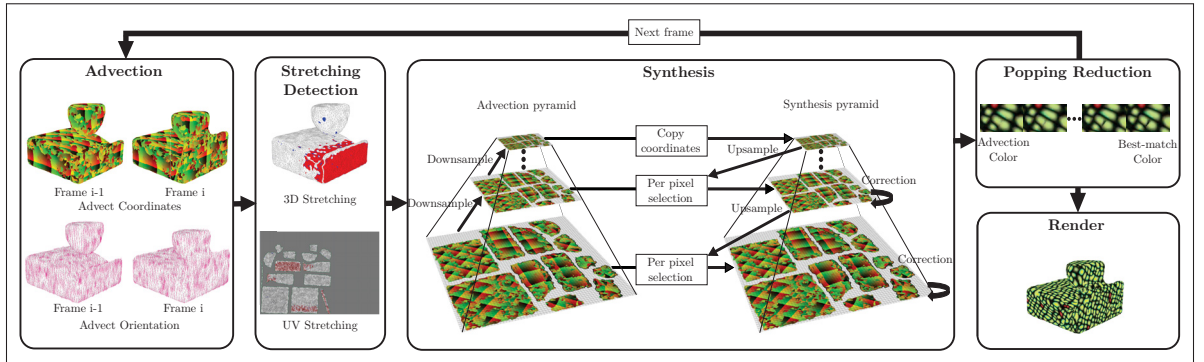


Figure 3.5 Pipeline of our approach. The synthesized UV coordinates are mapped to the red and green channels for depiction (boxes corresponding to advection and synthesis)

We copy the coarsest level of the advection pyramid to the coarsest level of a multiresolution *synthesis pyramid*. The synthesis pyramid, is the same size and shape as the advection pyramid; once complete, it contains the output texture in its finest-scale level. Following initialization, we iterate over the synthesis pyramid, level by level, from coarse to fine (Sec. 3.2.5). We detect

stretched areas (see Sec. 3.2.4) and wherever there is stretching, we use freshly synthesized data from the coarser synthesis pyramid level (faithful to the exemplar, but with weaker temporal coherence); in areas that are not stretched, we use data from the same level of the advection pyramid (good temporal consistency and good fidelity to the exemplar because stretching was absent). Before iterating to the next level, we run a window-based best-match method with the exemplar to improve texture quality.

After the synthesis pyramid is fully populated, we conduct an additional step to prevent popping artifacts (Sec. 3.2.6), as follows. We detect when a synthesized texel has a significantly different color from the previous frame, and eliminate the potential popping by gradually interpolating between the previous color and the synthesized color. Figure 3.5 and algorithm 3.1 summarize the steps of our approach.

3.2.1 First Frame Synthesis

At the first frame, we synthesize the texture using an existing appearance-space texture synthesis method (Lefebvre & Hoppe, 2006). This multiresolution method synthesizes UV coordinates that index into the exemplar. We later convert these coordinates to colors to get the final texture, and the final animations will be rendered with typical off-the-shelf 3D software.

3.2.2 Surface Parameterization

The objective of the surface parameterization is to obtain a 2D space in which to conduct our appearance-space texture synthesis. For every frame, we parameterize the 3D surface to a 2D atlas using the method of Sheffer, Lévy, Mogilnitsky & Bogomyakov (2005). The atlas is created independently at each frame, i.e., there is no relationship between atlases at different frames. While independent atlas creation simplifies atlas creation and imposes no constraints on the simulation or meshes, it complicates tracking texels between frames. We track texel positions by transforming 2D atlas positions to 3D spatial positions; this is further discussed in the next section.

Algorithm 3.1 Overview of our approach

```

1 Texture synthesis (Lefebvre & Hoppe, 2006) for  $frame_1$  in  $synthesis\_pyramid(level_m)$ ;
  // Pyramid levels:  $m$  is finest,  $0$  is coarsest
2 foreach  $frame_i, i \in 2, \dots, n$  do
  // advection pyramid
3   Advect  $synthesis\_pyramid(level_m, frame_{i-1})$  texels to  $advection\_pyramid(level_m, frame_i)$ 
4   foreach  $level_j, j \in m, \dots, 1$  do // fine to coarse
5     | Downsample  $level_j$  to  $level_{j-1}$ ;
6   end foreach
  // synthesis pyramid
7   Copy  $advection\_pyramid(level_0)$  to  $synthesis\_pyramid(level_0)$ ;
8   foreach  $level_j, j \in 1, \dots, m$  do // coarse to fine
9     | foreach  $element \in level_j$  do
10      | if there is stretching at element then
11      |   Set  $synthesis\_pyramid(level_j, element)$  by upsampling from
12      |    $synthesis\_pyramid(level_{j-1}, element)$ ;
13      | else
14      |   Copy  $advection\_pyramid(level_j, element)$  to  $synthesis\_pyramid(level_j, element)$ ;
15      | end if
16    | end foreach
17    Apply best-match search correction on  $synthesis\_pyramid(level_j)$ ;
18  end foreach
19  Convert UV coordinates from  $synthesis\_pyramid(level_m)$  to colors;
20 end foreach

```

3.2.3 Advection

Once the first frame of the simulation is synthesized using the method of Lefebvre & Hoppe (2006), our approach advects both the texture and the orientation field across frames, yielding a coherent sequence of textures conforming to the per-frame geometry. Advection ensures that texture features follow the fluid motion throughout the animation. In the following sections, we describe the two advection processes in detail: texture advection (Sec. 3.2.3.1) and orientation advection (Sec. 3.2.3.2).

3.2.3.1 Texture Coordinates Advection

For a given frame, we advect texture from the previous frame, ultimately producing an animated texture following the fluid motion. However, although the textures are stored in 2D atlases, the fluid velocity field is given in 3D (at vertex positions, in our implementation). We therefore need a process to transfer the 2D texture coordinate to 3D and back. The process is described in this section and illustrated in Figure 3.6.

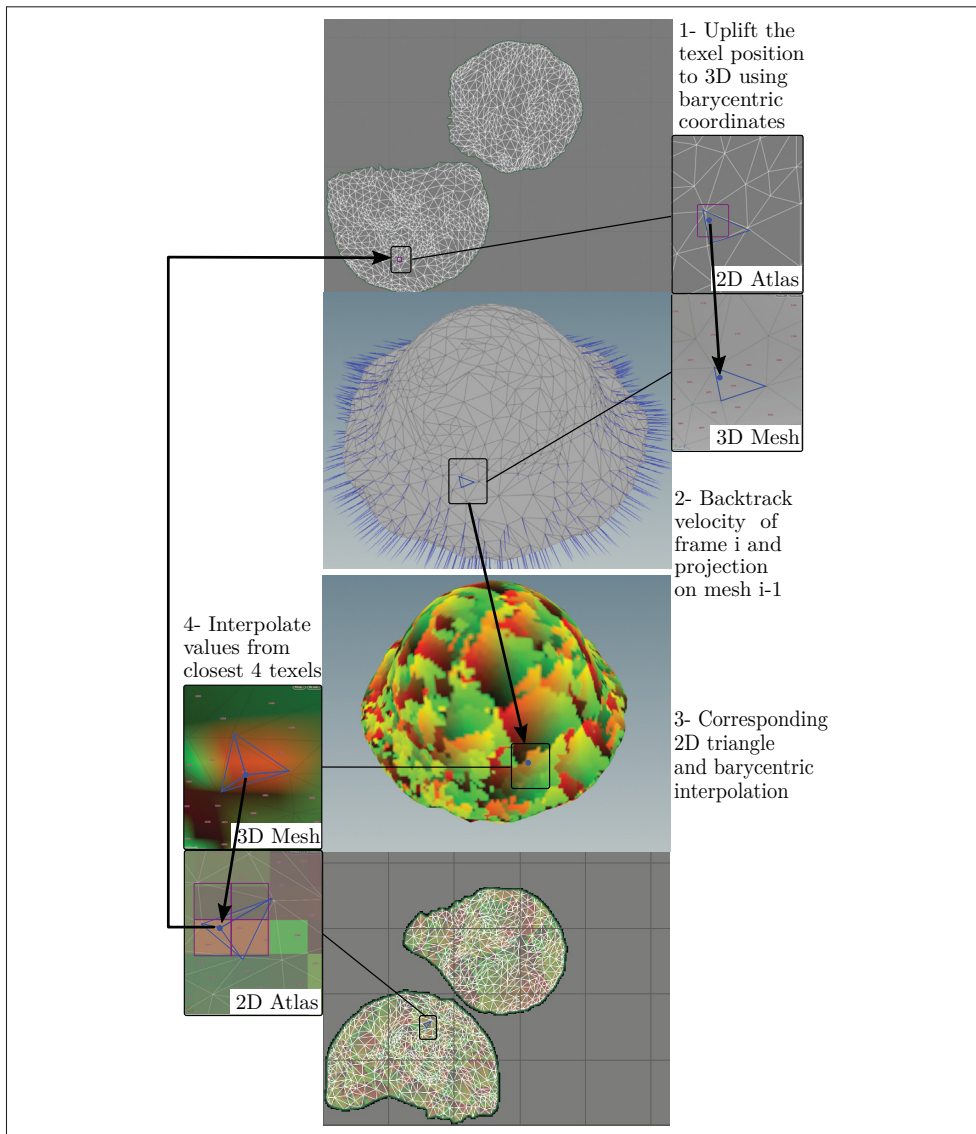


Figure 3.6 UV coordinates of a current frame i are computed by backtracking to the previous frame based on the velocity field

For each texel, we first convert its 2D position to a 3D point $\mathbf{p}$ using barycentric coordinates (w_1, w_2, w_3) within the 2D triangle containing the texel. We interpolate the velocity from the 3D triangle’s vertex velocities $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ using barycentric interpolation:

$$\mathbf{v} = w_1 \mathbf{v}_1 + w_2 \mathbf{v}_2 + w_3 \mathbf{v}_3 . \quad (3.1)$$

Next, we advect the 3D point $\mathbf{p}$ backwards with the time interval Δt between two frames:

$$\mathbf{p}' = \mathbf{p} - \Delta t \cdot \mathbf{v} . \quad (3.2)$$

The backtracked point $\mathbf{p}'$ is projected onto the closest triangle of the previous 3D mesh. We then compute the barycentric coordinates of this projected 3D point and apply them to the corresponding 2D triangle to retrieve the texel position in the previous frame’s atlas.

The obtained position will usually not align exactly with the center of a texel in the atlas. The advected texture value is therefore determined by interpolating from the texels around the position. After we are finished populating the finest level of the pyramid through advection, we downsample to coarser levels.

3.2.3.2 Orientation Field Advection

Akin to the method of Lefebvre & Hoppe (2006), our approach uses an orientation field to control the direction of the texture features. The user provides a 3D orientation field over the surface at the first frame, with an orientation vector at each mesh vertex. Unlike Lefebvre & Hoppe (2006), we need to deal with the advection of that orientation field throughout the animation. We use the method of Kwatra *et al.* (2007) to advect the orientation field from the past frame to the current frame. Doing so means backtracking from each 3D vertex in the current frame to a 3D position on the 3D mesh of the past frame, using the inverse velocity of the current frame. From that position, we obtain the orientation of the closest vertex.

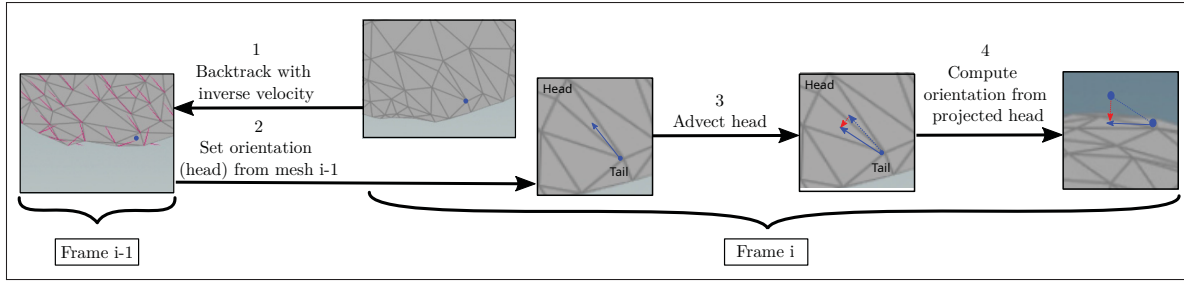


Figure 3.7 Advection of the orientation field by backtracking to the position on the previous frame

Having obtained the past frame’s orientation, we are not finished: the previous orientation also needs to be rotated by the velocity field. We follow the strategy of Kwatra *et al.* (2007), anchoring the orientation vector to the current frame vertex, and separately advecting the head of the orientation vector. The head is set at a distance d from the current frame vertex in the direction of the past frame orientation, where d is half the average length of the edges incident on the current frame vertex. We use the closest position to the head on the current-frame mesh to determine what velocity to apply to the head.

We do a barycentric interpolation of the velocity, advect the head using that interpolated velocity, and project it to the closest position on the current frame mesh. From the projected advected head, we compute the new orientation vector, which is the displacement from the current frame vertex to the newly computed head. The full advection process is illustrated in Figure 3.7.

As we aim to have an orientation field with as few discontinuities as possible, we complete the orientation field update with a smoothing step (Kwatra *et al.*, 2007). Smoothing is particularly important when there are topological changes in the surface (e.g., splashes).

3.2.4 Stretching Detection

When surfaces deform and experience significant topological changes, advection can generate textures with visible artifacts. Large texture stretching, as depicted in Figure 3.8 left, is a prominent and disturbing visual artifact produced by advection. Unlike other minor defects,

stretching cannot directly be repaired by best-match search correction; we therefore propose a specific approach to addressing it. We detect local stretching using two complementary criteria: the 3D displacement relative to neighboring texels in the previous frame, and the UV coordinate distance to neighbor in the current frame.

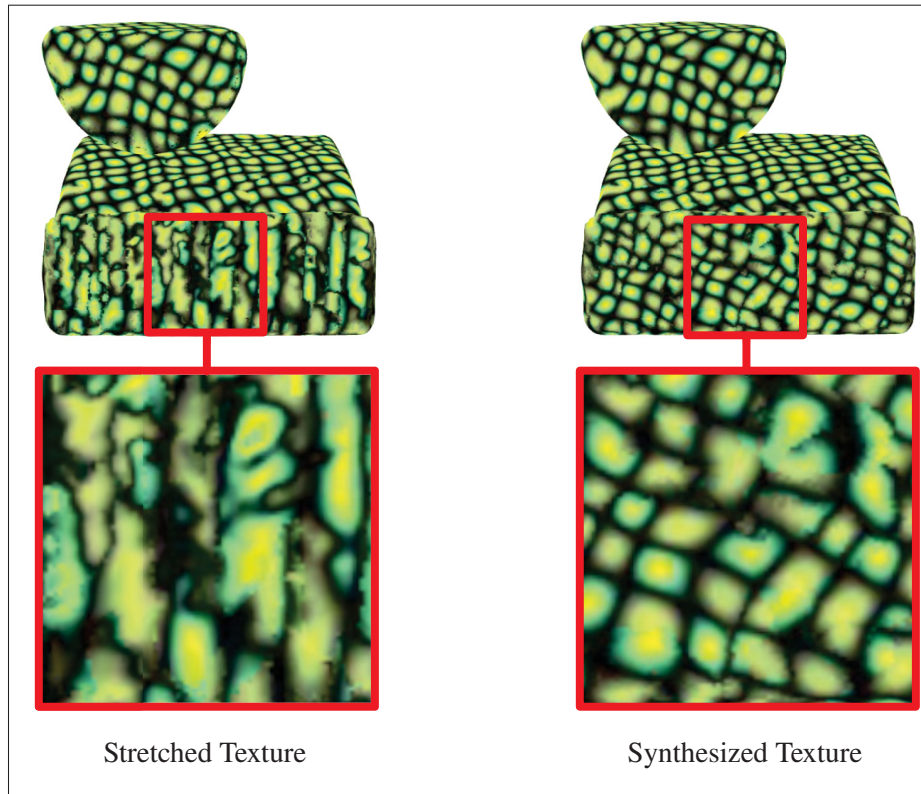


Figure 3.8 Advected texture after a single frame of animation compared to the result after our synthesis approach. Left: Without synthesis, the texture becomes severely stretched, even after a single frame. Right: Our approach detects stretched regions and synthesizes new texture there

3.2.4.1 3D stretching

The 3D stretching criterion is calculated for each texel at the finest resolution (Fig. 3.9). We first find the position of each texel in 3D space by computing its barycentric coordinates within the corresponding triangle of the surface mesh. We then backtrack this position to the previous frame using the velocity field. For each texel, we evaluate the Euclidean distances to its neighboring

texels in 3D space at their previous positions. The minimum of these distances is compared against a threshold τ_{3D} . If this distance is below the threshold, the texel is marked as stretching. A small distance in the previous frame indicates positions that were close together in the previous frame which have become far apart in the current frame, introducing stretching. We use the minimum distance as stretching is often anisotropic.

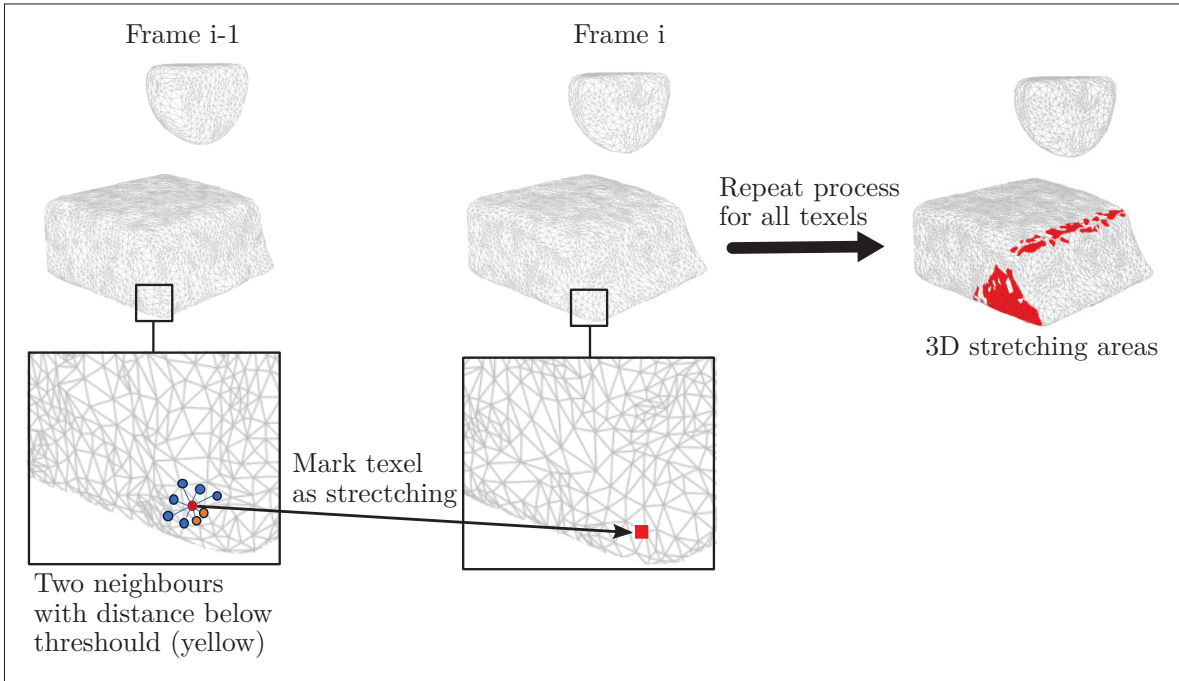


Figure 3.9 3D stretching detection process

3.2.4.2 UV stretching

The UV coordinate distance criterion is also computed per texel at the finest resolution (Fig. 3.10). Considering the UV coordinates stored in the advection pyramid, we determine the distances between the stored UV coordinates of a texel and the stored UV coordinates in its immediate neighbors. Then, we select the smallest UV distance and compare it against a threshold τ_{UV} . Small UV distances mean that the details will appear stretched, and here too we take the minimum as the stretching can be anisotropic. Texels with distances below the threshold are marked as stretching. Texels considered as either 3D or UV stretching are considered as stretching during the synthesis step.

The synthesized texture in Figure 3.8 right shows the result of the synthesis based on the stretching greatly improves the results. Despite that, Figure 3.8 right may seem of lower quality compared to the exemplar. One must keep in mind that here we deliberately selected a frame with large stretching. In this frame, large portions of new surface were created over a single frame, requiring the synthesis of a lot of new content, while trying to keep some consistency with the previous frame. As soon as the liquid movement introduces slightly less stretching, the frame-to-frame synthesis quickly improves the quality to the level of the exemplar.

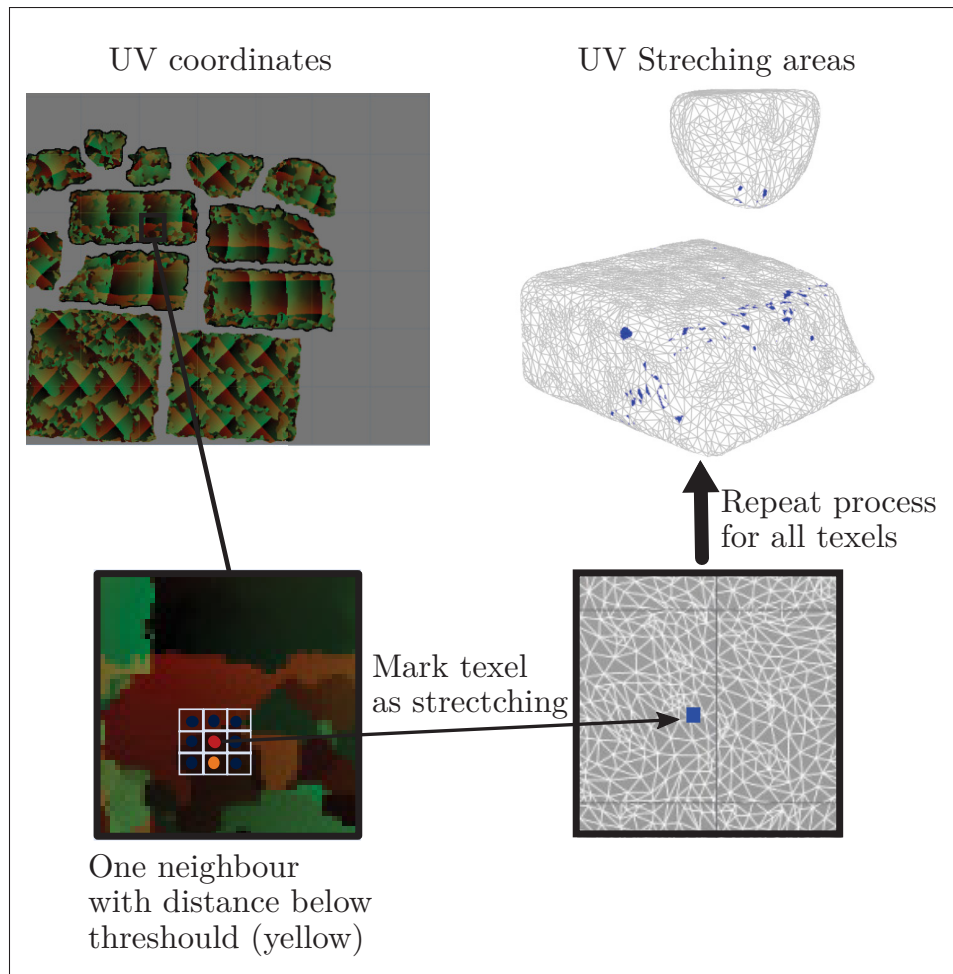


Figure 3.10 UV stretching detection process

3.2.5 Synthesis Pyramid

The advection pyramid contains the advected texture at multiple levels of resolution. We will now populate the synthesis pyramid from coarse to fine, leading to the synthesized texture coordinates at its finest level. Our goal is to get a good compromise between the advected texture data from the advection pyramid (to promote temporal coherence) and synthesizing coarse to fine without considering the advected texture data (for better fidelity to the exemplar).

With an animated liquid, the variable surface area can cause the texture features to become stretched and compressed. Compressed features require no special treatment: a best-match search method will automatically repair compressed areas, even working at a single level of resolution, since the support of the search window is larger than the size of the shrunken details. However, stretched features still pose a problem. When the surface stretches, the support of the search window is small compared to the texture feature size, and the best-match process at the finest level cannot recover the missing larger-scale features.

Conversely, we could conduct a synthesis solely from the coarsest level of the advection pyramid. This would have a severe downside: temporal continuity would be lost (see accompanying video of the paper) since the coarsest level of a downsampled advected texture does not retain enough of the advected features. This is nearly equivalent to resynthesizing the texture independently at every frame.

We thus designed an adaptive multiresolution approach that decides, for each texel of each level of the synthesis pyramid, whether the texel should get its value from upsampling the partially synthesized coarser level synthesis pyramid or from the corresponding texel in the advection pyramid. We initialize the coarsest level of the synthesis pyramid with the coarsest level of the advection pyramid, and conduct two rounds of coherence best-match search. For the next levels from coarse to fine, we decide, on a per-texel basis, if the synthesis gets its value by upsampling from the coarser synthesis pyramid level or from the same level advection pyramid. For each synthesis texel, if all texels in the corresponding window at the finest resolution level are marked as stretching, the texel coordinates are obtained through upsampling from the coarser level of the

synthesis pyramid. Otherwise, the advection pyramid at the same level provides the coordinates, favoring temporal continuity. After populating all texel values at a given level, we conduct two rounds of texture correction with best-match search through a coherence random walk best-match search (Busto *et al.*, 2010) based on PCA compressed exemplar features (Lefebvre & Hoppe, 2006).

3.2.6 Popping Reduction

Our synthesis process so far is effective, but when the liquid is slowly deforming, occasional local popping artifacts can occur, where there is a sudden and quite noticeable local change in color from one frame to the next. Popping prevention is a balance between remaining faithful to the exemplar and maintaining temporal continuity. Since the surface deforms, it is inevitable that we will need to introduce new patterns and it is not always possible to introduce them slowly while remaining faithful to the exemplar.

In our approach, we compute the 3D velocity of each texel and compare it to a user-specified velocity threshold (adjusted for a given simulation scenario). Only texels with a velocity slower than the threshold will be considered for our popping reduction. At faster velocity, many changes typically need to happen, and they are hardly noticeable. Furthermore, the popping reduction is restricted to the texels where there is UV stretching. The rationale behind this is that the UV stretching is often linked to modifications in the texture to introduce new patterns which will reduce the stretching. For exemplars with large color differences, this can lead to the sudden color changes that we want to avoid.

We detect abrupt popping by measuring the distance between the RGB color resulting from advection and the RGB color obtained from the best match. If the distance exceeds a threshold τ_{pop} , we want to transition between the advected color and the best-match color over a number of frames (5 in our examples). For such texels, we store advected and best-match colors. The detection could occur on an already transitioning texels, in which case the stored colors are reset, as well as the transition counter. Note that popping detection is computed only at the finest

resolution of the synthesis pyramid. For texels where a color transition is active, we replace the best match coordinates of the first round of correction with those of a *transition color*, computed by linearly interpolating between the stored advection and the best-match colors.

For a typical scenario, the size of the connected components of neighboring texels undergoing the popping reduction transition is small. For the more than 37k components detected on a frame-by-frame basis over the 240 frames of a representative animation, only 30 were of size 100 texels or more. This means that the regions are small and infrequent, making our approach immune to the ghosting artifacts found in other methods. Furthermore, the identification of popping occurs on a per-texel basis. This is another strength of the approach: the affected regions change over time (e.g., can grow or shrink by a few texels on a frame-by-frame basis), making them hard to identify and follow and hence unlikely to be perceived as spurious structures.

Because we store UV coordinates in the synthesis pyramid, we need to identify UV coordinates of an exemplar color close to the transition color. We search among exemplar texels until we find UV coordinates close to the transition color. In our implementation, for efficiency considerations, we search until we find a color with a distance to the transition color that is below a threshold τ_{interp} . Given this transition color, we gradually bring the synthesized color closer to the stored best-match color as we progress from frame to frame. When moving to the next frame, the stored advection and best-match colors are advected in a similar fashion to the UV advection (Sec. 3.2.3), but snapping to the closest texel in the previous frame, to avoid having the transition region grow every frame.

3.2.7 Implementation Considerations

Surface parameterization. The area of the free surface of animated fluids often significantly expands and contracts as waves and splashes form and then reintegrate into the bulk of the liquid. Because we conduct the synthesis in atlas space (Sec. 3.2.2), we need to account for distortions in the fluid surface. We aim to ensure that the ratio of surface area remains constant between 3D

and 2D atlas space. Frame to frame, we scale the 2D atlas surface to maintain the same 3D/2D ratio throughout the animation.

Interpolating UV coordinates. Advection requires us to fetch the UV coordinates from the past frame (Sec. 3.2.3.1). Simply snapping from the position toward the closest atlas texel introduces severe aliasing, potentially manifesting as a discordant vibrating appearance when the texture is animated. To minimize aliasing, we conduct interpolation, but care must be taken since we are interpolating UV coordinates instead of colors. Figure 3.11 shows the variation of UV coordinates in a typical atlas; U and V are respectively mapped to the red and green channels. Note the patches with smooth variation of coordinates, and also discontinuities at the boundaries of these patches.

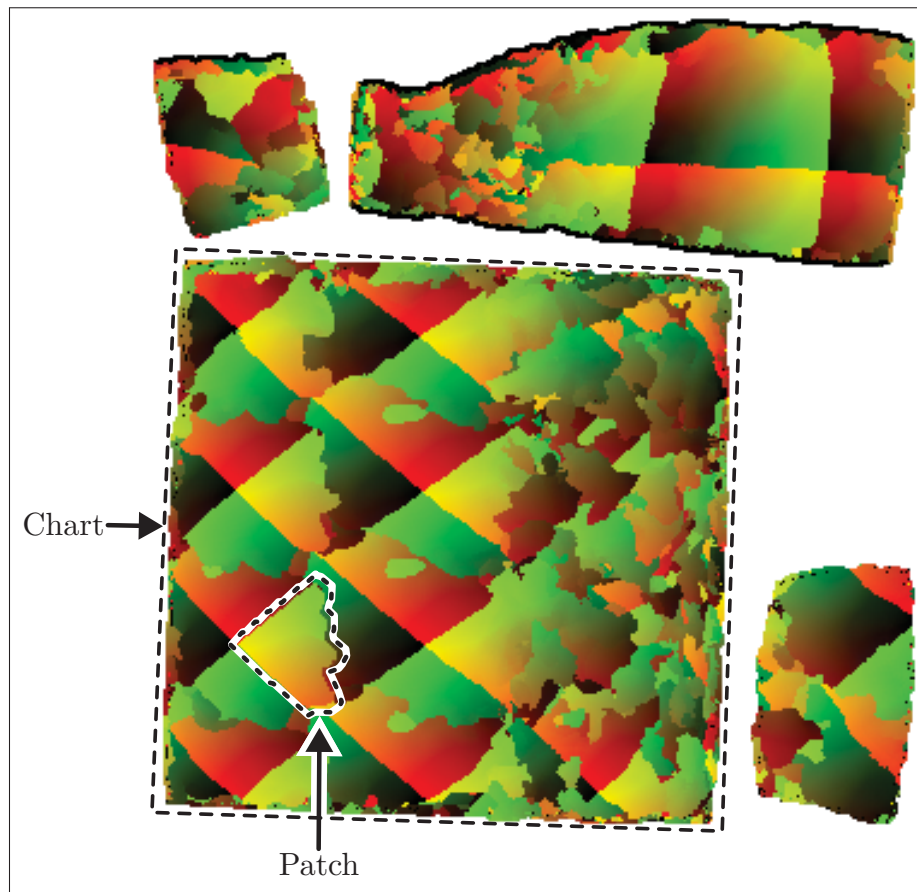


Figure 3.11 A typical atlas with what we refer to as *charts* (separate parts of the mesh flattened to 2D) and *patches* (contiguous group of texels with smoothly varying UV coordinates)

Interpolating UV coordinates across patch discontinuities would lead to interpolated coordinates spatially far away from each other in the exemplar, and consequently unrelated. Accordingly, the straightforward bilinear interpolation only makes sense in regions of smooth variation of UV coordinates. At patch boundaries, a different strategy is required. Given the four texels surrounding the position in the atlas, we select the closest one and retain only those of the remaining three whose stored UV distance to it is less than 5% of the UV extent. We are left with one, two, three, or four texels, and depending on the number, we choose a different interpolation strategy. When four texels remain, bilinear interpolation is used; when three texels remain, barycentric interpolation is applied. If only one or two texels remain, we extrapolate the UV coordinates of the closest texel based on the difference between the position in the atlas and the center of the texel. This difference leads to a vector which is then interpreted relative to the Jacobian of the orientation field. We add the UV coordinate stored in the closest texel to that vector to get the UV coordinate to copy to the current frame texel.

Downsampling UV coordinates. A problem similar to UV interpolation is that of downsampling the stored UV coordinates in the advection pyramid. Simply averaging the four UV values only works if all texels are in the same patch. Instead of averaging, we pick the top-left texel and add a UV vector corresponding to moving from the top-left texel center to the central position of the four texels accounting for the Jacobian of the orientation field. For patches that properly account for the orientation field, this is equivalent to computing the average of the UV values. Moreover, this strategy is robust to cases where the four texels are not in the same patch.

Thresholds for stretching detection. The thresholds in Section 3.2.4 are computed from user-specified ratios. Threshold τ_{3D} is defined from a user-specified ratio (varying within $[0.8, 0.95]$ in our experiments) multiplied by the mean 3D distance to neighboring texels in the synthesized first frame. Threshold τ_{UV} is defined from a user-specified UV stretching ratio (varying within $[0.3, 0.5]$ in our experiments) multiplied by the mean exemplar UV coordinate distance between each synthesized texel and its neighbors in the first frame.

Orientation in 2D. When doing the coherence random walk best-match search (Busto *et al.*, 2010) in Section 3.2.5, we need to convert the 3D surface orientation into per texel 2D orientation, done as follows. For each texel, we get its corresponding position in 3D, interpolate the orientation from the vertices of the corresponding triangle (by barycentric interpolation) and convert the 3D orientation to 2D through the Jacobian matrix corresponding to the surface parameterization.

Best match in atlas space. Some care must be taken when doing the best-match search for a texel at the boundary of a chart (see Fig. 3.11) in the atlas. As the neighbor texels are in a different chart, it is necessary to figure out where in the other chart to fetch the proper UV coordinate. Knowing that an edge at the boundary of the 2D charts corresponds to a 3D edge shared by two triangles, we identify the neighbor 3D and 2D triangle and the related neighbor chart in the atlas. We index on the other side of the 2D edge (in the other chart) at the corresponding location.

Popping prevention. The threshold τ_{pop} to detect abrupt changes in color (Sec. 3.2.6) is defined through a user-defined ratio (varying within $[0.2, 0.5]$) converted to an RGB distance threshold by multiplying it by the maximum distance between any RGB color in the exemplar. The threshold τ_{interp} , used to determine whether we found an exemplar color close enough to the interpolation color, is derived from a user-defined ratio (varying within $[0.02, 0.1]$) which is also converted to an RGB distance by multiplying it by the maximum distance between any two RGB colors in the exemplar.

3.3 Results

We tested our texture synthesis approach with different fluid simulation scenarios and multiple texture exemplars. The test cases encompass a range of behaviors, including both viscous and inviscid flows, with and without splashes. In the first scenario (Fig. 3.12), a spherical volume of fluid falls into a non-viscous liquid, creating significant splashes and rapid surface motion. This scenario tests our approach’s ability to handle sudden topological changes and large deformations.

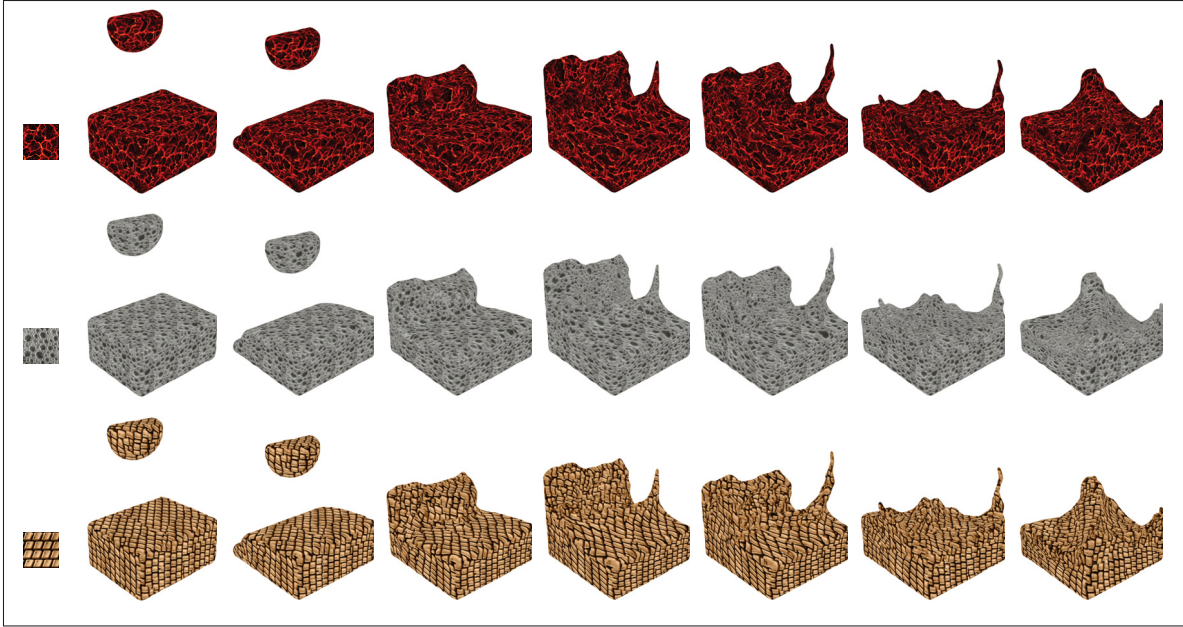


Figure 3.12 Results of our approach in the sphere scenario. The input exemplars are shown in the leftmost column, followed by frames (left to right) extracted from the animation

In the second scenario (Fig. 3.13), we simulate a double dam break in which two blocks of fluid collapse under gravity, producing complex surface flows and numerous splashes. This test evaluates the temporal coherence and stability of our approach under complex motion.

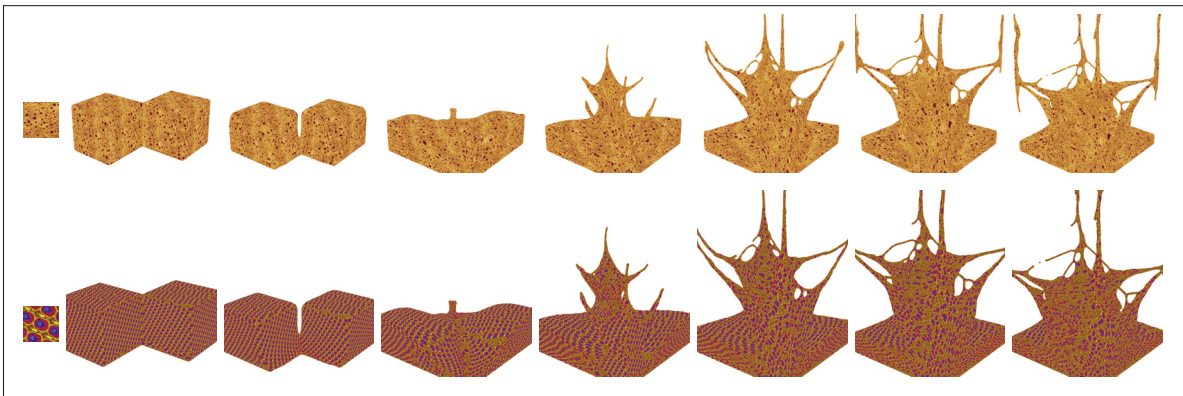


Figure 3.13 Results of our approach in the double dam break scenario. The input exemplars are shown in the leftmost column, followed by frames (left to right) extracted from the animation

In the third scenario (Fig. 3.14), a viscous fluid falls and deforms slowly on an object and the ground, with no splashes. This test demonstrates the ability of our approach to work under smooth and gradual deformation.

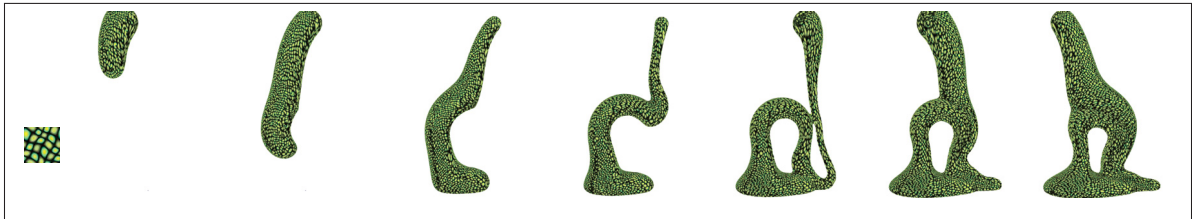


Figure 3.14 Results of our approach in the viscous drop scenario. The input exemplar is shown in the leftmost column, followed by frames (left to right) extracted from the animation

Finally, for the fourth scenario (Fig. 3.15), the flow evolves more slowly, without abrupt changes or fragmentation, emphasizing temporal consistency in more stable simulations.

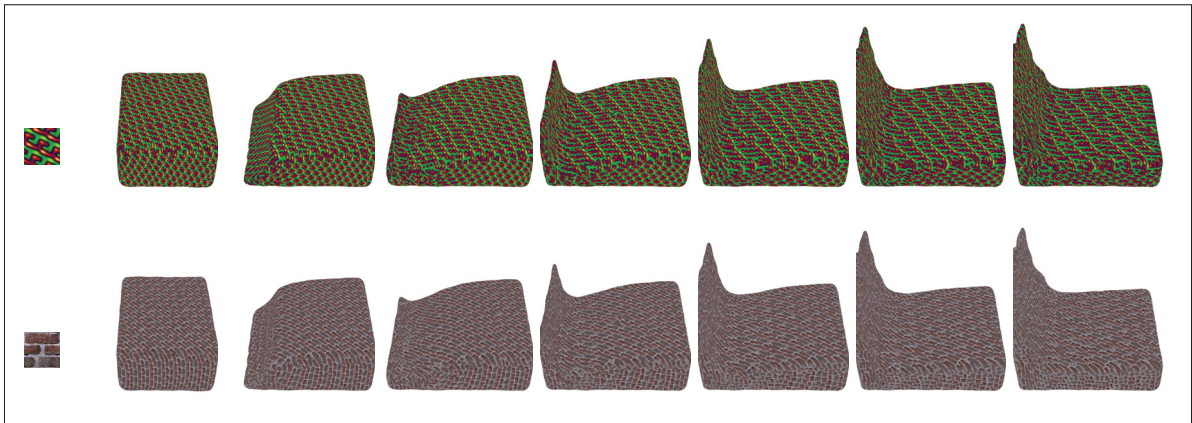


Figure 3.15 Results of our approach in the viscous dam break scenario. The input exemplars are shown in the leftmost column, followed by frames (left to right) extracted from the animation

Our selection of exemplars includes textures corresponding to real-world scenarios (lava, foam), textures commonly used in texture synthesis (purple cells, green cells, hooks), and structured textures (keyboard, bricks). In all cases, we can see that our approach encourages results with texture patterns faithful to the exemplar and promotes temporal coherence throughout the

animation. Full animations of these results can be seen in the accompanying video of the original paper (Guzmán *et al.*, 2026) and demonstrate good temporal continuity of the texture patterns.

All experiments were conducted on an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz (6 cores) with 32 GB of RAM and an NVIDIA RTX A2000 GPU. Table 3.1 shows performance statistics for several test cases. The reported times correspond to the average per-frame duration of each stage of our approach.

Table 3.1 Average per-frame computation times (in seconds) for various stages of our approach

Scenario	Pyramid Levels	Output Resolution	Atlas Texels	Advection	Stretch Detection	Synthesis	Popping Reduction	Total
Sphere	6	1024 ²	185k	0.34	1.43	25.27	0.07	26.77
Double dam break	5	1024 ²	115k	0.22	1.13	13.94	0.02	15.31
Viscous drop	5	2048 ²	621k	0.96	4.83	63.26	2.02	71.07
Viscous dam break	5	1024 ²	205k	0.34	1.45	27.23	0.51	29.19

In addition to the performance evaluation, we compare smaller and larger values of thresholds τ_{3D} , τ_{UV} , and τ_{pop} , modifying one threshold at a time to demonstrate their respective influence on the texture synthesis (Fig. 3.16). With smaller values of τ_{3D} and τ_{UV} , our approach more often uses the advection texture information; consequently, the temporal continuity is good, but larger stretched patterns appear and the overall texture is less faithful to the exemplar. Conversely, larger values of τ_{3D} and τ_{UV} result in stronger reliance on the freshly generated values from the synthesis pyramid. In this scenario, the temporal continuity is low, but the texture on static frames is more faithful to the exemplar. The value of τ_{3D} is the most sensitive; slight increases can significantly decrease the temporal continuity. When considering τ_{pop} , lower values result in good temporal continuity, but the multiple transitions make it harder to preserve fidelity to the exemplar. With larger values of τ_{pop} , the transition is rarely used, resulting in increased popping artifacts. In conclusion, the degrees of freedom provided by τ_{3D} , τ_{UV} , and τ_{pop} help the user to tune the results as wanted and the effects of each threshold are predictable, making them easy to adjust.

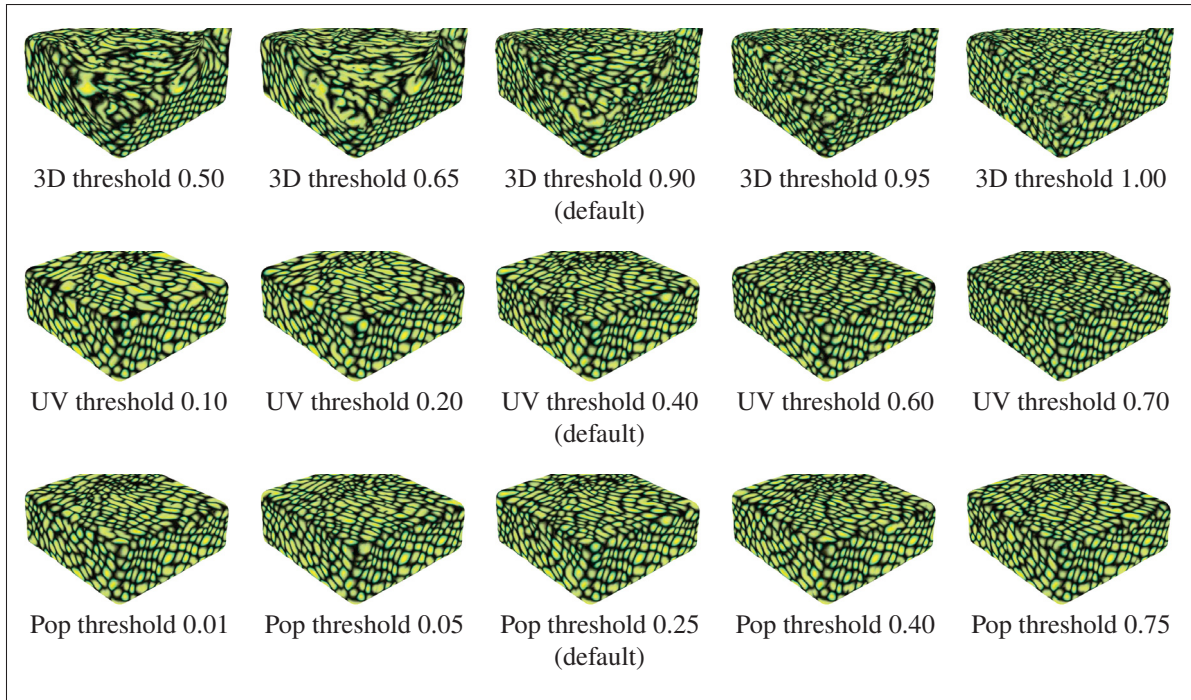


Figure 3.16 Comparison of texture synthesis results using different threshold values for 3D stretching, UV stretching, and popping detection. The middle column (default) shows the value we selected as best for this simulation and texture

Finally, Figure 3.17 illustrates the influence of the atlas resolution (512^2 , 1024^2 , and 2048^2) on the synthesized results. One should keep in mind that the Jacobian scale needs to be adjusted to keep the same feature size in the rendered images. Results tend to be better when the scale of the Jacobian is one; lower-resolution atlases tend to be blurrier, while higher-resolution ones sometimes result in slight amounts of higher-frequency patterns. Overall, our approach produces similar results across all resolutions, with synthesized patterns remaining visually consistent.

3.3.1 Comparison

We compared our approach against relevant previous methods: those which generate textures on the free surface of dynamic fluids. Figure 3.18 shows the comparison between our approach and the methods of Kwatra *et al.* (2007) and Gagnon *et al.* (2019). We can see that our texture colors are more faithful to the exemplar than those produced by the method of Kwatra *et al.*

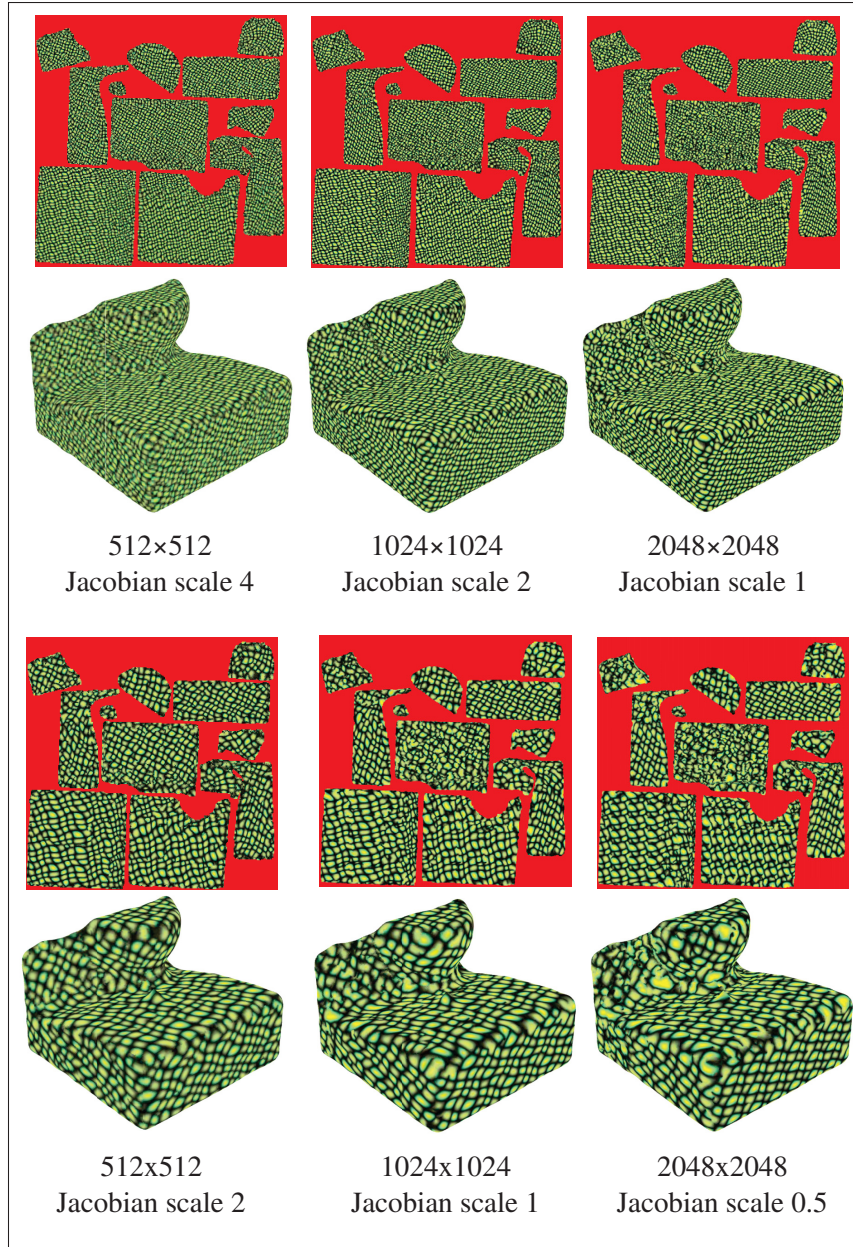


Figure 3.17 Synthesis results using atlases at different resolutions. The first and third rows show the texture atlases, while the second and fourth rows show the rendered fluid surfaces

(2007). The method of Gagnon *et al.* (2019) produces artifacts such as ghosting and double contours, which are not present in our results. Overall, our approach is free from these issues and generates more coherent texture animations.

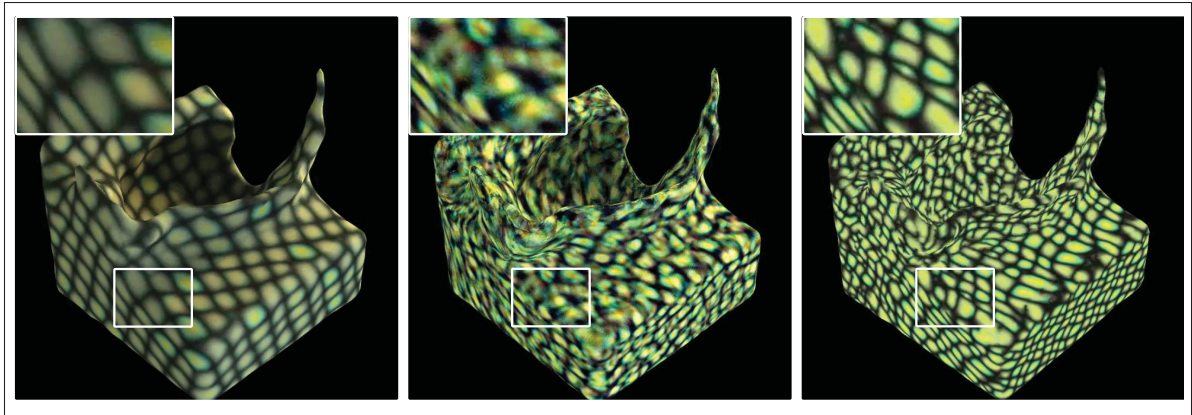


Figure 3.18 Comparison of Kwatra *et al.* (2007) and Gagnon *et al.* (2019) vs. our approach

In addition, we evaluated our approach against the method of Gagnon *et al.* (2016), which synthesizes textures using patches. As shown in Figure 3.19, even when small patches are used, the contour of the patches is quite perceptible and introduces sharp edges not found in the exemplar. This issue becomes more pronounced when larger patches are used, resulting in even more noticeable discontinuities. In contrast, our approach performs per-voxel synthesis and avoids such artifacts, producing smoother and more coherent results.

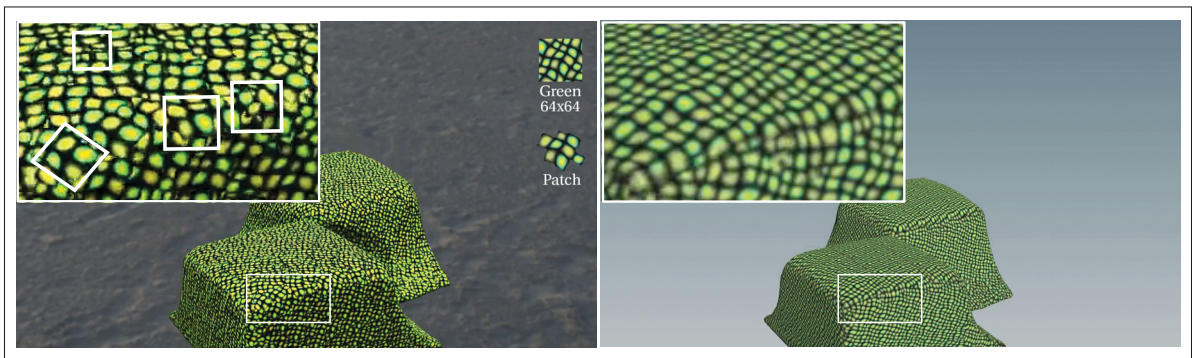


Figure 3.19 Comparison of Gagnon *et al.* (2016) vs. our approach

We also compared our approach to the method of Gagnon *et al.* (2021), which uses deformable patches to synthesize textures on fluids. Figure 3.20 shows that our result is more faithful to its respective exemplar compared to the result of Gagnon *et al.* 2021. Moreover, their method often produces motion artifacts near patch boundaries due to its erosion mechanism. As the

patches shrink, parts of them are removed, revealing portions of underlying patches and creating an illusion of motion. In contrast, our approach performs synthesis per texel instead of relying on patches. This avoids boundary artifacts and undesirable motion artifacts. As shown in the accompanying video, the textures generated by our approach preserve fine texture features and have good coherence across the simulation frames.

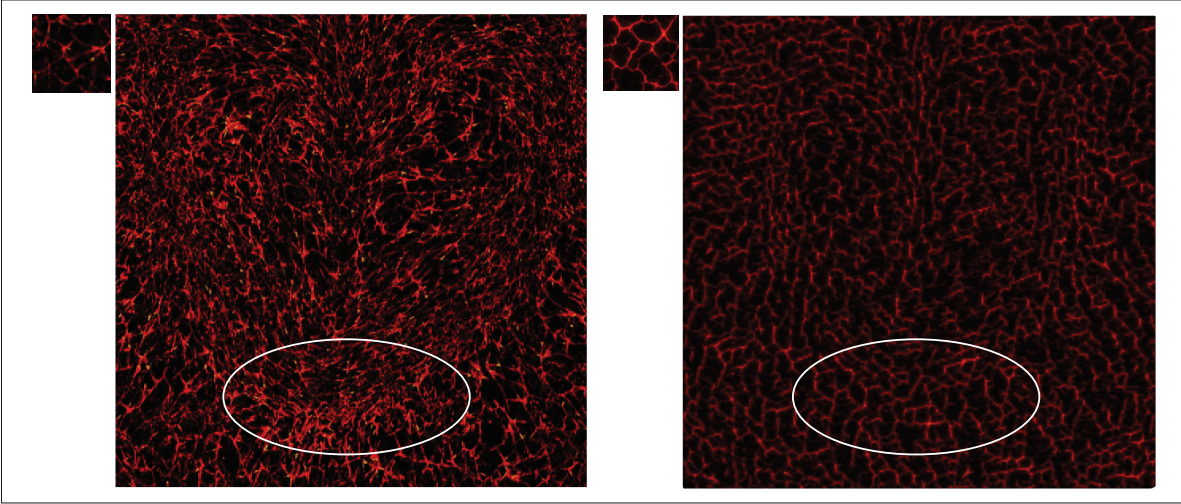


Figure 3.20 Comparison of Gagnon *et al.* (2021) vs. our approach

3.3.2 Ablation study

To better understand the contribution of the synthesis and advection pyramids in our approach, we performed an ablation study comparing three configurations. In the first, we used only the advection, propagating values from the previous frame only at the finest level and without performing any synthesis. In the second, we advect and downsample to get the advection pyramid, and then we perform synthesis for all texels and at all levels in each frame, only considering the advected data when copying the coarsest resolution of the advection pyramid to the synthesis pyramid. As shown in Figure 3.21, the advection-only variant preserves temporal continuity but quickly loses texture detail and fidelity, whereas the synthesis-only variant preserves texture features well but lacks temporal coherence. Our full approach combines the strengths of both, promoting results which are faithful to the exemplar while encouraging temporal stability throughout the animation.

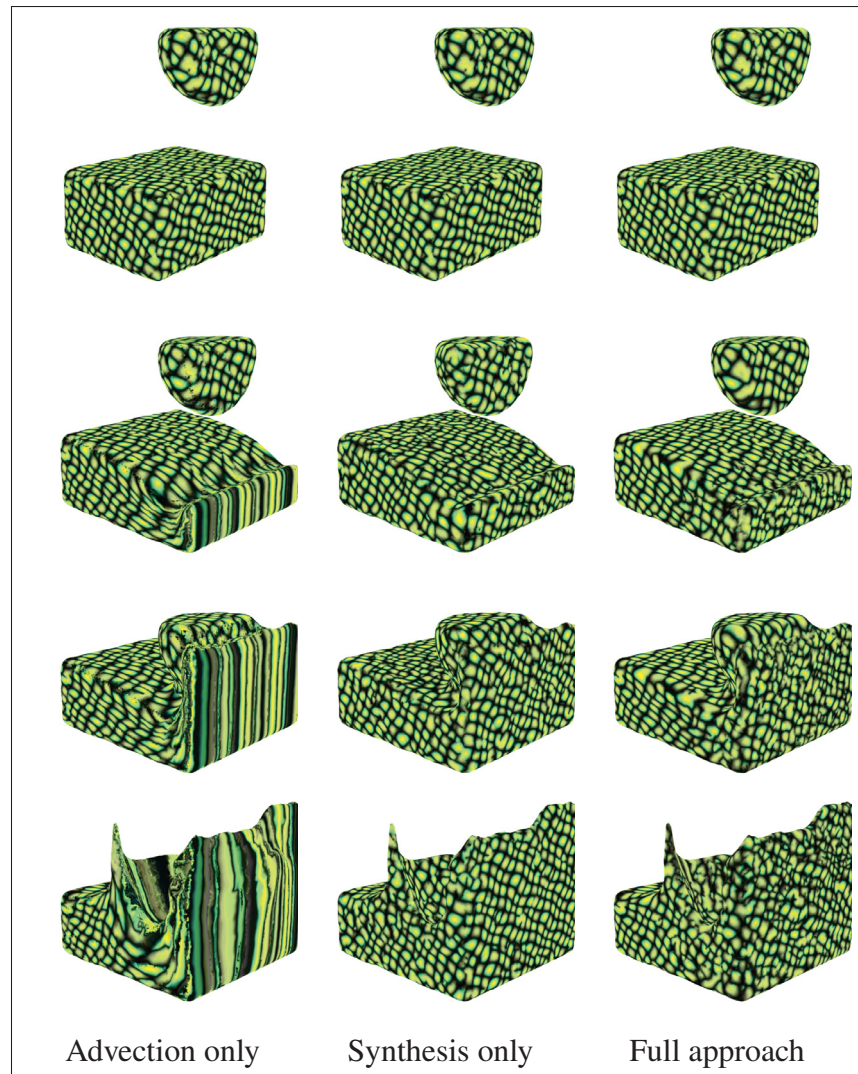


Figure 3.21 Ablation study results showing texture synthesis on fluid simulation over time. Left: When only relying on advection we see significant stretching. Middle: Only synthesis variant, showing good texture details but suffering from temporal incoherence (see accompanying video of the paper). Right: Our adaptive approach, which combines advection and synthesis to produce good fidelity and temporal continuity. The figure shows selected frames from an animation, in temporal order from top to bottom. Note that all variants begin from the same first frame

3.4 Discussion

In this chapter we presented an approach to conduct exemplar-based texture synthesis on animated meshes from fluid simulations. An advantage of our approach is that it is agnostic to the

specific fluid simulation method, as long as we can get animated 3D meshes and velocity fields. After a full multiresolution synthesis at the first frame, our approach advects the texture and orientation information from frame to frame. A multiresolution texture pyramid is built by downsampling the advected finest-resolution texture. We compute local stretch in 3D by advecting a neighborhood of texels to the previous frame and in 2D by comparing the stored UV coordinates within a neighborhood of texels. Where we identified stretching, we upsample the coarser levels of the synthesis pyramid to improve coherence to the exemplar. Conversely, where no stretching is detected, we use the advected data to favor temporal coherence. This process is done in a multiresolution fashion and together with best-match correction at every level of the synthesis pyramid. We showed that our approach works on different fluid simulation scenarios and texture exemplars.

Nevertheless, our approach suffers from some limitations: lower-fidelity reproduction on some exemplars, lower-quality synthesis on non-stationary textures, and reduced synthesis quality in later frames compared to the first frame. Figure 3.22 shows one exemplar for which the random-walk of Busto *et al.* (2010) does not recover compatible good matches and introduces areas corresponding to a local minimum in the best-match search. We thus lose the larger-scale features of that exemplar. Using a strategy similar to that of Jamriška *et al.* (2015), limiting repeated selection of the same texel, would likely mitigate this issue.

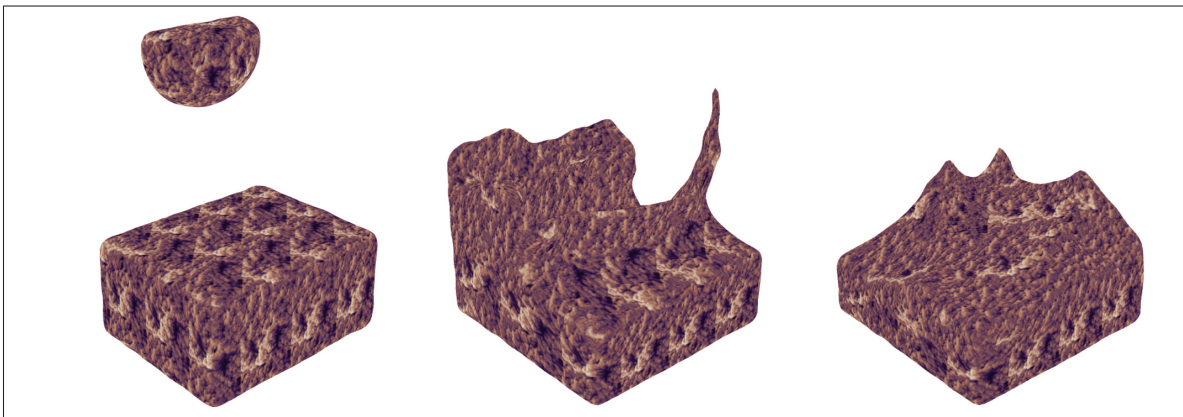


Figure 3.22 Random-walk best match loses features from the input exemplar. Selected frames from an animation

Additionally, while our approach produces good results across a range of textures, it performs best with stationary exemplars. For instance, results on the keyboard and brick textures are acceptable, but inferior to results for stationary textures.

At the first frame, the synthesis is computed without any temporal continuity constraints and therefore the quality is slightly better than at frames later in the animation. Doing a more thorough synthesis at later frames is an avenue for potential improvement.

Finally, our approach opens avenues for further research. An advantage of our approach is that it is agnostic to the simulation approach, making it amenable to methods which increase the resolution from coarser simulations (Roy, Poulin & Paquette, 2021; Roy, Paquette & Poulin, 2020). Moreover, instead of using separate texture atlases at every frame, it could be interesting to investigate the tracking of an explicit mesh throughout the animation (Dagenais, Gagnon & Paquette, 2017). Furthermore, a mesh refinement method could be explored to provide a more detailed surface representation where needed while reducing unnecessary geometric complexity (Li, Wang & Paquette, 2024). In addition, the 2D parameterization to an atlas and the frame-to-frame advection could be used in conjunction with other synthesis models such as diffusion models. The advection and multiresolution pyramid, together with the local stretching detection, could be used to control the diffusion process. Our popping prevention mechanism could as well be used should the diffusion model introduce abrupt changes. The problem of texture synthesis over fluid surfaces remains a challenging one and we anticipate further advances in the coming years.

CHAPTER 4

HYBRID EXEMPLAR-BASED TEXTON TEXTURE SYNTHESIS WITH MULTIRESOLUTION BACKGROUND SYNTHESIS FOR ANIMATED FLUIDS

In this chapter, we present another method for synthesizing textures on animated fluids. This approach will be presented in the paper *Joint Texton and Background Texture Synthesis for Animated Liquids* by **Julián Guzmán**, Christopher Palazzolo, Oliver van Kaick, David Mould, and Eric Paquette to be submitted soon. The method builds upon two techniques: By-Example Synthesis of Vector Textures (Palazzolo *et al.*, 2025) and the technique of chapter 3 Adaptive Multiresolution Exemplar-Based Texture Synthesis on Animated Fluids (Guzmán *et al.*, 2026). As in chapter 3, we address exemplar-based texture synthesis with the goal of having an output texture that resembles the exemplar without being an exact duplicate. As well, the approach is designed for texturing the surfaces of fluids. As opposed to chapter 3, we want our approach to work for non-stationary textures with identifiable features, such as rocks and bubbles.

The synthesis of vector textures method (Palazzolo *et al.*, 2025) synthesizes vector graphics rather than raster graphics, offering advantages such as editability and resolution independence. However that method is limited to 2D textures and has not been applied to animated 3D fluid surfaces. While the adaptive multiresolution texture synthesis method (Guzmán *et al.*, 2026) described in chapter 3 works well for generating temporally coherent textures on animated fluids, it has some limitations. The method has difficulty preserving large texture features, as they often deform into unfaithful shapes during advection and synthesis. Additionally, the method can suffer from a gradual reduction in synthesis quality across frames, especially for non-stationary textures or areas where texture features show little variation. This often shows up as washout effects, where texture details gradually fade, resulting in lower visual fidelity compared to the first frame.

In this work, we present a novel approach for synthesizing textures on animated fluid surfaces that transforms the vector texture synthesis method so that it generates raster-graphics textons distributed and advected on the 3D surface. We also propose an approach that extracts textons from the texture exemplar, and creates a new *background exemplar* exempt of the extracted

textons. As in chapter 3, the proposed approach requires as input a texture exemplar, the sequence of meshes corresponding to the animated free surface of the fluid with per-vertex velocities, and an initial orientation field defined on the first frame of the mesh sequence.

Motivated by the work of Julesz & Bergen (1983), which describes textures in terms of prominent local features, our approach begins by extracting the most significant structures from the input texture, referred to as textons. We then extend the texton texture synthesis method to be used on animated 3D fluid surfaces. The textons are synthesized frame by frame by advecting their positions over time in an atlas domain and then mapping them onto the 3D surfaces. This strategy preserves the integrity of the main texture features, reducing distortion during animation. In addition, the texton synthesis provides control over the number of synthesized textons, and when they appear or disappear during the animation.

To complement the synthesized textons, we create a background exemplar that contains only the less prominent background details of the original input exemplar, explicitly excluding the textons addressed by the texton texture synthesis. Using this background exemplar, we apply a texel-based adaptive multiresolution method to synthesize the background texture for each frame of the animation. Having removed the prominent textons, the remaining background texture is more uniform. As such, the synthesis process is more prone to issues such as washout, reduced fidelity, or lower visual quality. To address these problems, we enhance the synthesis process with a strategy that limits repeated selection of texels, ensuring a more uniform usage of all texels from the background exemplar. This strategy reduces stretching and washout effects, improving the overall quality and coherence of the generated background texture.

After completing both synthesis processes, the synthesized textons and background texture are merged in each frame to produce the final texture sequence. The background synthesis serves as the base onto which the textons are overlaid in the corresponding atlases. Alpha blending is applied at the edges of the textons to smoothly integrate them with the background. The resulting animated texture is consistent with the exemplar and is temporally coherent.

The main contributions of this work can be summarized as follows:

- **Hybrid texture synthesis method for fluid surfaces that integrates and extends texton-based and adaptive multiresolution methods.**
- **Advection of texton and texel positions on a 2D animated mesh using 3D velocity field.**
- Texton texture synthesis for 3D animated fluids.
- **Creation of a background exemplar by removing textons from the original exemplar.**
- **A texel usage mechanism enforcing a more uniform contribution of exemplar texels to the background synthesis.**

This work was conducted in collaboration with Carleton University. Our main contribution, in bold above, focused on developing the advection method for texton texture synthesis, creating the background exemplar, and implementing a strategy to limit repeated texel selection during background synthesis. A collaborating student at Carleton University developed the texton texture synthesis method and the final merging of both synthesis approaches.

4.1 Textons and Background Texture

Our approach starts by decomposing the exemplar into complementary components: the most prominent and structured elements and less structured background details. The extraction of the principal texture structures (Sec. 4.1.1) results in a collection of individual *textons*, while the background details serve to synthesize a *background texture* (Sec. 4.1.2).

4.1.1 Texton Extraction

To identify the principal texture structures, we follow the texton extraction method proposed by Palazzolo *et al.* (2025). In this method, the input exemplar is segmented using the Segment Anything model (Kirillov *et al.*, 2023), identifying regions corresponding to the most prominent and structured elements of the texture (textons, Figure 4.1). Unlike the original method by Palazzolo *et al.* (2025), which further decomposes the exemplar into three layers, our approach retains only the primary textons. A collaborating student carried out the implementation of

this texton extraction process; for further details, we refer the reader to the original method (Palazzolo *et al.*, 2025).

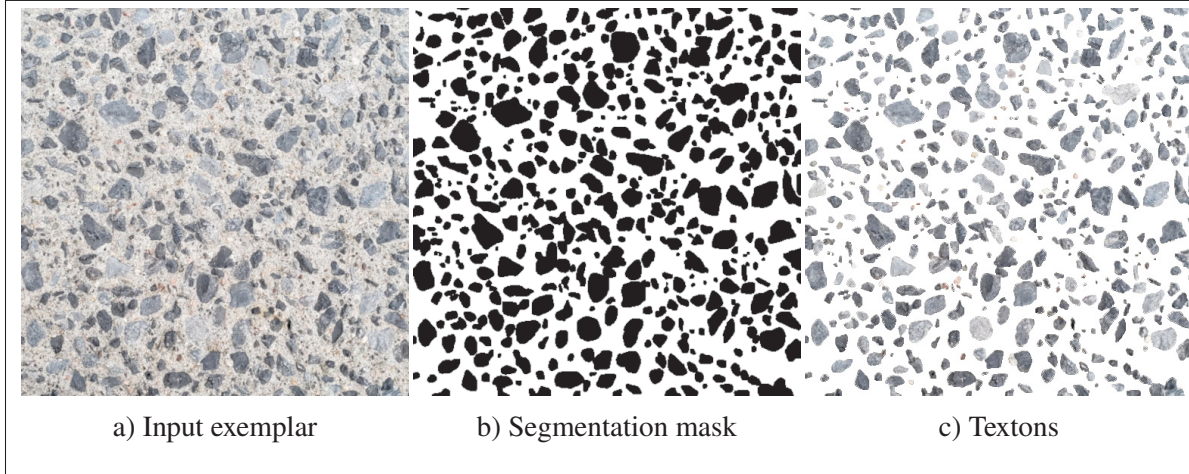


Figure 4.1 Segmentation of the input exemplar into textons and background areas

4.1.2 Background Exemplar Generation

To take advantage of the approach of chapter 3 for texture synthesis, we require a square texture that looks like the remaining background area and exempt of the holes left by the texton removal (Figure 4.1). The difficulty here is that, to create that background texture, all we have is the input exemplar which contains valid background texels inter-weaved with a multitude of texels that have the texton colors and patterns that are not part of the background.

One solution to fill these missing regions is to adapt a constrained texture synthesis method so that the search windows are constrained to the locations where each texel of the search window overlaps with a valid background texel. We initially explored the technique proposed by Wei & Levoy (2000). In this method, the regions to be filled are defined using a mask, and texels are synthesized based on neighborhood matching. However, such approaches that perform synthesis in a scan-line order using an L-shaped neighborhood often introduce visible artifacts and discontinuities, especially along the right and bottom boundaries of the filled regions (Fig. 4.2).

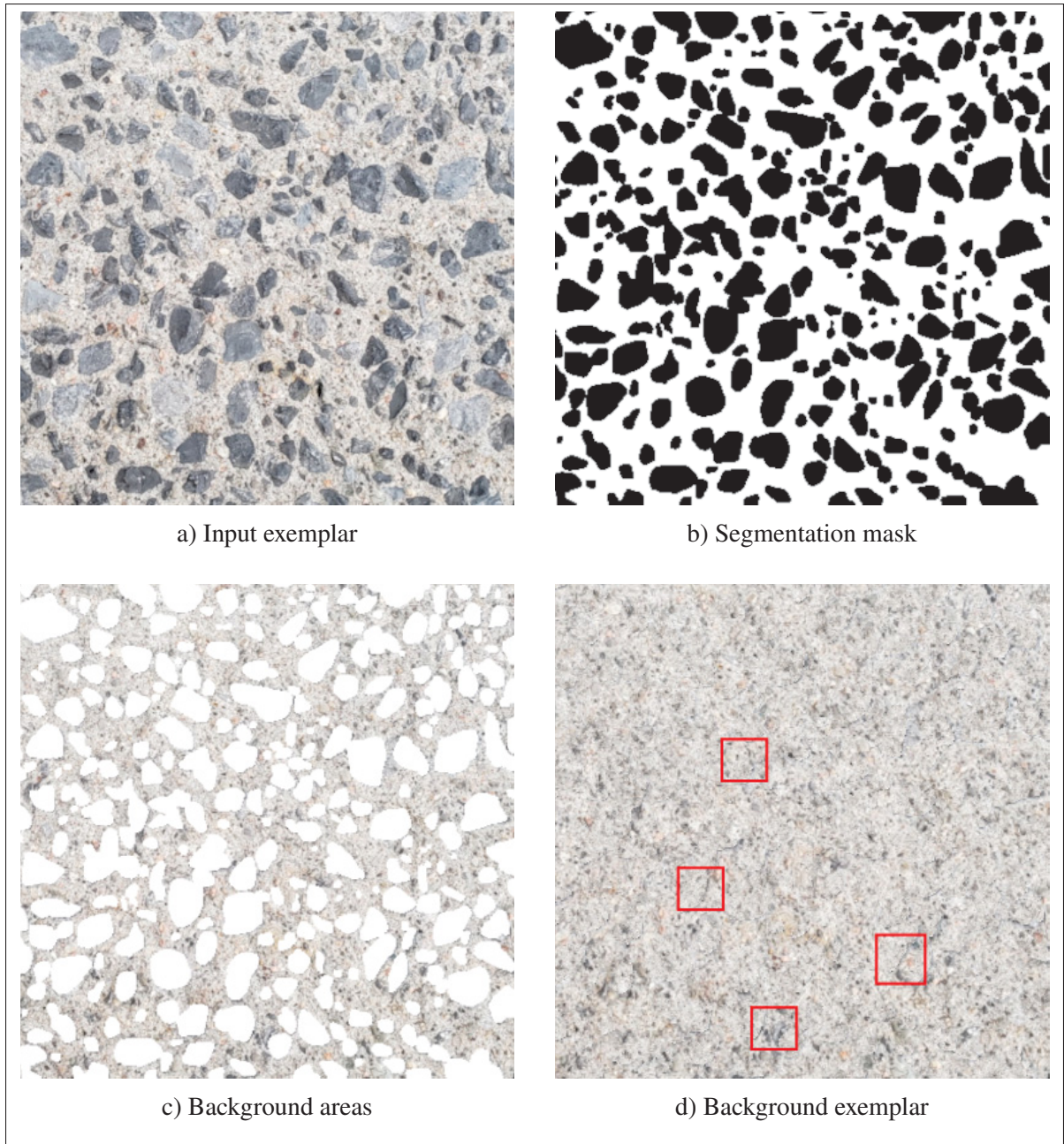


Figure 4.2 Constrained texture synthesis method that performs synthesis in a scan-line order using an L-shaped window neighborhood. Highlighted regions depict typical discontinuities caused by processes based on scan-line and L-shaped processes

Existing work in image and video inpainting has investigated methods for filling missing regions by restricting the search space and enforcing spatial consistency (Clément, Benoit & Paquette, 2007; Benoit & Paquette, 2015). We adopt the image inpainting approach proposed by Clément

et al. (2007) to fill missing regions iteratively, starting from the boundary between valid background pixels and the empty areas left by the removed textons. At each iteration, only pixels on the current boundary are synthesized, progressively filling the empty regions from the exterior toward their interior (Fig. 4.3).

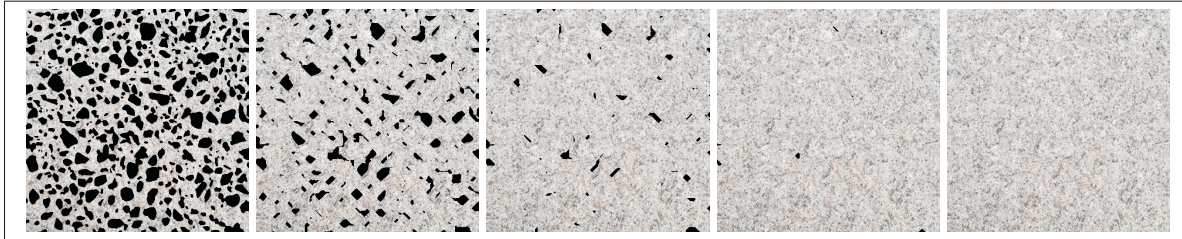


Figure 4.3 Background texture synthesis through iterative inpainting is illustrated. The synthesis results at iterations 1, 10, 20, 30, and 40 show the gradual filling of empty regions left by the removed textons

The approach uses four neighborhood window configurations with different shapes, instead of relying on a single fixed window (Fig. 4.4). Each configuration considers a subset of texels corresponding to a specific corner of the window (top-left, top-right, bottom-left, and bottom-right). For each boundary texel, an appropriate window is selected dynamically based on the availability of valid neighboring texels. If no suitable window is available for a given texel, its synthesis is deferred until neighboring texels are filled in subsequent iterations. Additionally, like in the original method, approximate nearest neighbor (ANN) search structures are precomputed for each window configuration to ensure computational efficiency.

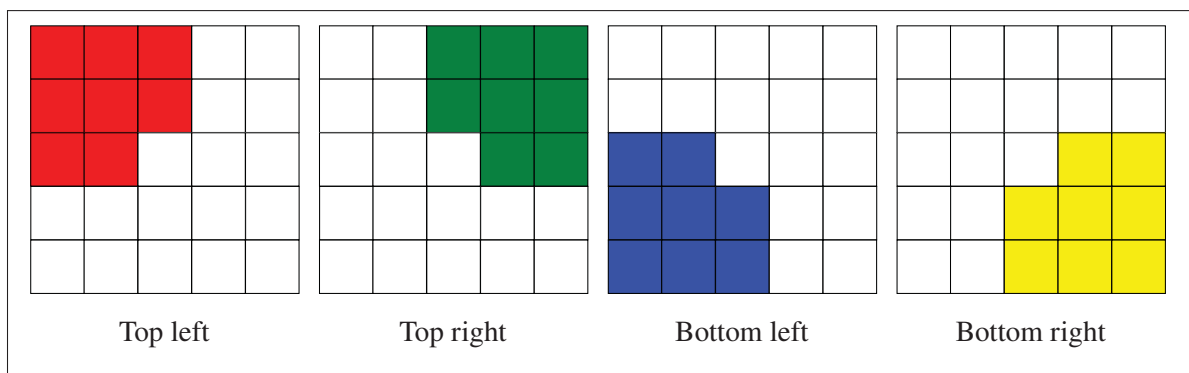


Figure 4.4 Window configurations with different shapes

This approach produces visually coherent results when filling empty areas in an image. Therefore, we use it to generate a background exemplar by filling the regions left empty after removing textons. This process preserves the overall appearance and background characteristics of the original texture.

4.2 Hybrid Texture Synthesis on Animated Fluids

After extracting the textons and preparing the background texture exemplar, we pursue with the animation part of the process. The texton-based method is responsible for synthesizing and advecting the main texture features, preserving their integrity and avoiding distortion during the animation. Meanwhile, the adaptive multiresolution method synthesizes the background texture. The outputs from both methods will subsequently be merged to produce the final textured fluid surface.

Although the fluid surface is represented as a sequence of 3D meshes, the texture synthesis process is performed in a 2D domain. As in chapter 3, the mesh sequence is mapped to a 2D atlas space using a surface parameterization. This parameterization is shared by both synthesis processes, ensuring consistency between the texton-based texture synthesis of textons and the adaptive multiresolution synthesis of the background.

The texton-based synthesis (Palazzolo *et al.*, 2025) needed to be significantly changed to account for texton orientation, advection, distribution maintenance, and the atlas composed of disjoint charts. Starting from an initial distribution defined on the first frame, the positions of textons are advected across successive frames using the 3D advection projected to 2D. This texton-based texture synthesis strategy helps prevent common artifacts such as texture stretching, ghosting, and deformation of prominent texture features, ensuring robust preservation of the main texture structures.

The second synthesis approach focuses on generating the background details of the texture using an adaptive multiresolution exemplar-based method. As explained in chapter 3, this strategy performs a per-textel synthesis process by advecting texel coordinates backward across frames

and using both a multiresolution advection pyramid and a multiresolution synthesis pyramid. For each texel, the synthesis process is applied at multiple resolution levels, after which a per-texel decision criterion determines whether to preserve the advected content or to resynthesize it from the exemplar. This approach maintains visual fidelity and temporal coherence throughout the animation. However, the method may lose some features of the exemplar and experience reduced synthesis quality in later frames. To address this, a new strategy is introduced here to limit repeated texel selection during the synthesis process, ensuring a more uniform usage of all texels from the exemplar.

Finally, the synthesized textons and background texture are merged to produce the final textured fluid surface for each frame. This hybrid approach reduces common artifacts such as texture stretching, ghosting, and loss of exemplar features. The resulting texture sequence maintains both strong temporal coherence and high visual fidelity, even in the presence of complex fluid motion. The following subsections describe each component of the hybrid synthesis process in detail.

4.2.1 Texton Synthesis

Our texton synthesis approach aims to synthesize the textons over the animated fluid surface. The synthesis is performed in a 2D atlas domain, where textons are placed, and their positions are represented using continuous UV coordinates. We define the position of a texton as the center of its axis-aligned bounding box in the UV exemplar space. As in the original method of Palazzolo *et al.* (2025), a descriptor is constructed for each texton extracted from the input exemplar. Each descriptor consists of a grid-based representation of the local neighborhood of the texton. The cells in the grid encode either the presence of a neighboring texton or empty space when no texton is present. These descriptors provide a compact representation of the local texton distribution in the exemplar and serve as the basis for guiding the synthesis process.

Synthesis begins on the first frame by generating an initial distribution of textons. The first texton is placed randomly in the atlas domain, and the following textons are gradually positioned

using their descriptors to ensure a coherent spatial distribution. For further details, we refer the reader to the original method (Palazzolo *et al.*, 2025).

To propagate the textons across the animation, their positions are advected according to the motion of the fluid surface. Texel advection follows a strategy akin to that proposed by Guzmán *et al.* (2026), in which each UV coordinate is mapped to a 3D point on the fluid surface. Each point is advected to a new 3D position and subsequently projected back to 2D. However, instead of performing backward advection as in Guzmán *et al.* (2026), we advect the texel positions forward across frames (Fig. 4.5). To map each texel 2D position to a 3D point $\mathbf{p}$ on the fluid surface, we use barycentric coordinates (w_1, w_2, w_3) within the corresponding triangle of the surface parameterization. The velocity $\mathbf{v}$ at point $\mathbf{p}$ is computed by interpolating the per-vertex velocities $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ of the associated 3D triangle using barycentric interpolation. The 3D point is then advected to the next frame mesh using this interpolated velocity over the time interval Δt :

$$\mathbf{p}' = \mathbf{p} + \Delta t \cdot \mathbf{v}. \quad (4.1)$$

The resulting advected point $\mathbf{p}'$ may not lie exactly on the mesh surface, so it is projected onto the closest triangle of the 3D fluid surface. The barycentric coordinates of this projected point are then computed and applied to the corresponding triangle in the atlas parameterization, yielding the texel position in the atlas of the next frame.

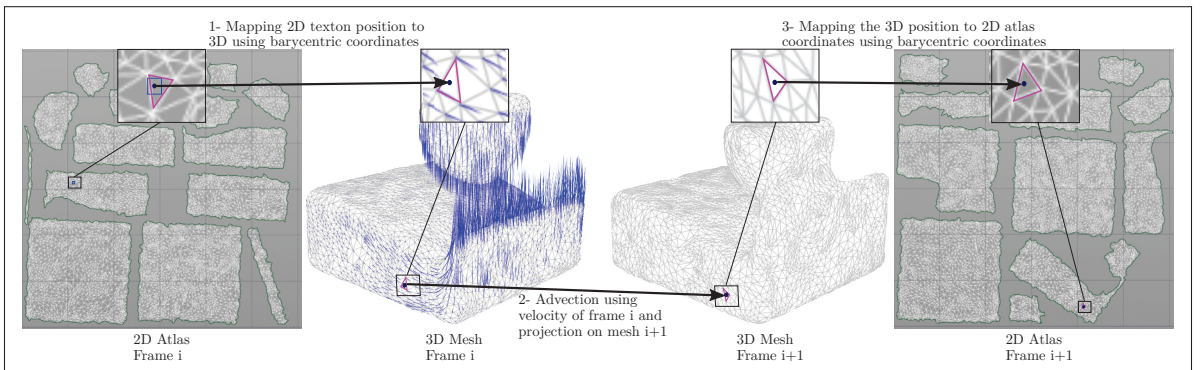


Figure 4.5 Texton advection

In the method of chapter 3 (Guzmán *et al.*, 2026), we used backward advection to make sure that we have a continuous assignment of UV coordinates for every texel of the current frame. As the texton advection is *forward* instead of backward, we sometimes got conflicting velocity vectors which resulted in the textons looking as if they were “sliding” on top of the background texture. To correct that, we “average” the 3D velocities from two consecutive frames in a fashion similar to Heun’s method (improved Euler’s method). Given a 3D position p_i at time i that we want to advect to time $i + 1$, we compute an initial position p_{i+1} by regular integration $p_i + \Delta t v_i(p_i)$ and projection on the mesh $p_{i+1} = \Pi_{\text{mesh}_{i+1}}(p_i + \Delta t v_i(p_i))$. From there, we compute an average of the velocities at the two frames to do the integration:

$$p_{i+1} = \Pi_{\text{mesh}_{i+1}} \left(p_i + \Delta t \left(\frac{v_i(p_i)}{2} + \frac{v_{i+1}(p_{i+1})}{2} \right) \right). \quad (4.2)$$

As we now get a new p_{i+1} which will in turn lead to a new velocity $v_{i+1}(p_{i+1})$, we iterate this process 10 times (it quickly leads to a good average velocity). We always use this process to compute the velocity, whether for forward or backward advection, and it leads to smooth and consistent results.

The texton synthesis uses an orientation field to control the direction of the textons. Akin to the method of Guzmán *et al.* (2026), a 3D orientation is defined over the surface in the first frame, advected across frames, and then converted into a per-texel 2D orientation in the atlas. The textons are synthesized using the orientation corresponding to their 2D position. This orientation field ensures that the textons maintain a coherent alignment throughout the simulation.

Since the atlas is composed of multiple disjoint charts, textons may cross chart boundaries during synthesis and advection. In such cases, a texton may partially belong to two or more charts, requiring special handling to ensure consistent positioning and orientation. To address this, we follow the strategy proposed by Lefebvre & Hoppe (2006) and construct an indirection map along chart boundaries. This indirection map stores, for each boundary texel, the corresponding location in the neighboring chart. When a texton is located near a chart boundary, it is positioned in its corresponding chart using the texton center. If the texton crosses the boundary, the portion

extending outside the chart is remapped to the neighboring chart using the indirection map, through a rigid transformation. The transformation is computed from two sets of points. The first set corresponds to the positions of the texels in the indirection map ring outside the chart which are inside the textons. We get the second set by transferring each position from the first set through the indirection map. We compute the 2D rigid transformation between these two sets of points through a Singular Value Decomposition (SVD). This transformation is then applied to the texton position and orientation vector. We then paste the related part of the texton to that new chart. This process works well even in cases where a texton is split across multiple charts.

Textons may appear or disappear as the fluid surface evolves. The texton distribution is dynamically regulated at each frame based on their local density, which is evaluated using the descriptors. If the density of textons is too high, some of them are tagged to be removed to avoid overcrowding. Conversely, if the density is too low or new surface areas appear, additional textons are added following the process controlled by the descriptors. We want the textons to appear and disappear gradually. For each texel inside a texton, we compute its distance to the texton boundary. When fading out a texel, at every frame we will decrease the distance by how many texels we want to erode from the texton boundary. That value can be less than a full texel as well (we use 0.75, determined empirically, for the results shown in this chapter). After decreasing the distance value, we completely discard any texel with a negative distance. To smooth the transition between the colors of the textons and those of the background texture, we apply a feathering over a few texels at the boundary of the texton. We integrate the two processes, erosion and feathering, by setting the alpha blending value to the distance divided by the number of texels over which we want to apply the feathering (three for the examples shown in this chapter) and clamped to the range $[0, 1]$. The process is reversed when fading in a texton.

The implementation of this texton synthesis method was carried out in collaboration with a student at Carleton University.

4.2.2 Background Texture Synthesis

The background texture synthesis stage focuses on synthesizing the less prominent details of the input exemplar that are not processed by the texton synthesis. For completeness, this section includes both the main aspects shared with the approach of chapter 3 and the novel aspects designed in this chapter. The method uses the background exemplar generated in Section 4.1.2, which is transformed into a compressed appearance-space representation using Principal Component Analysis (PCA) (Fig. 4.6).

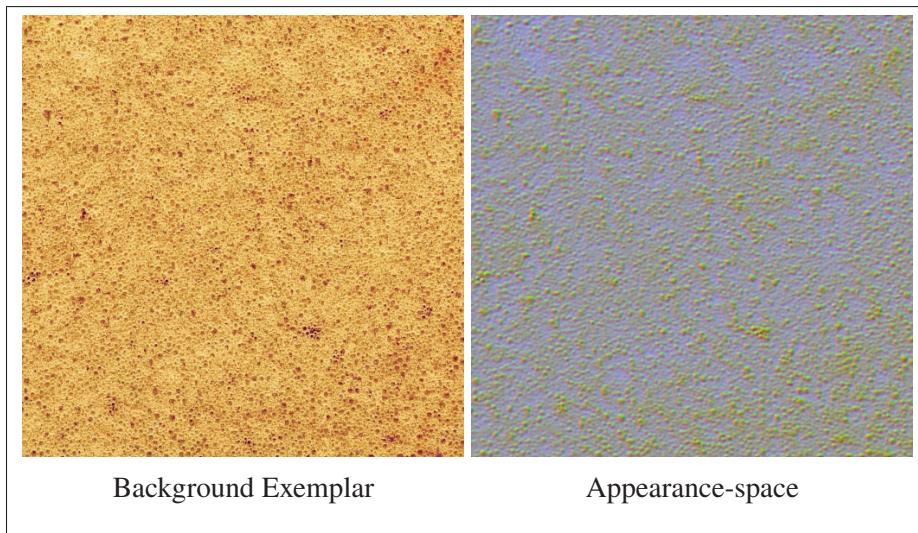


Figure 4.6 Background exemplar and its appearance-space representation

The process begins by synthesizing the texture for the first animation frame using the method of Lefebvre & Hoppe (2006). This texel-based method synthesizes UV coordinates from the exemplar in a 2D atlas domain (see Sec. 3.1). The synthesis of the first frame establishes a high-quality reference texture that serves as the foundation for all subsequent frames.

At subsequent frames, for each texel, the texture coordinates are advected from the previous frame according to the 3D velocity field. The synthesis process relies on two multiresolution structures: the advection pyramid and the synthesis pyramid. To determine which pyramid to use at each texel, local distortion measures are evaluated in both the 2D atlas domain and the 3D surface domain. These measures detect stretching and guide per-texel decisions, ensuring that

advected content is preserved where appropriate or replaced by resynthesized content from the exemplar as needed.

Following the original method of chapter 3 (Guzmán *et al.*, 2026), advection of UV coordinates is performed by obtaining the position of each texel in the atlas from the previous frame. As opposed to how we backtracked to the previous frame in chapter 3, here we use the averaged velocity explained in section 4.2.1. From the four texels surrounding that position, we select the closest texel and any additional texels whose UV distance to the closest texel is below 5% of the UV extent (same threshold as in the original method). Depending on the number of selected texels, an appropriate interpolation (bilinear or barycentric) or extrapolation is applied to obtain the final UV coordinate. Compared to the context of chapter 3, background exemplars contain fewer distinctive features, making the background synthesis process more susceptible to washout and stretching artifacts. To address this issue, we adjusted the stretch-detection thresholds, making the detection more sensitive than in chapter 3. In our experiments, the 2D stretching threshold τ_{UV} varies in the range $[0.45, 0.6]$, compared to the $[0.3, 0.5]$ range used in chapter 3. The 3D stretching threshold τ_{3D} varies in the range $[0.9, 0.95]$, rather than the $[0.3, 0.5]$ range used in chapter 3. These new threshold values allow stretch artifacts to be identified with higher sensitivity. The threshold τ_{pop} used for popping prevention uses the same values as in chapter 3, with values in the range $[0.2, 0.5]$. Similarly, the interpolation threshold τ_{interp} , which determines whether an exemplar color is sufficiently close to the interpolated color, is also maintained in the range $[0.02, 0.1]$. All other implementation considerations described in Section 3.2.7 are applied as in the original method.

Despite the adjustments to the stretch-detection thresholds, background synthesis may still be prone to undesired stretching and washout effects (Fig. 4.7). To mitigate these issues, we enhance the method of chapter 3 by limiting the repeated selection of texels during the best-match selection step, ensuring a more uniform use of all texels from the background exemplar. Our strategy is inspired by the approach of Jamriška *et al.* (2015), which promotes a more uniform usage of exemplar content. In their approach, synthesis is performed using patches extracted from the exemplar, and each patch is associated with a usage counter that limits how often it

can be assigned to the synthesis target, ensuring that all patches contribute more evenly to the synthesized result. As background synthesis is performed by texel rather than using patches, we derive a new approach by applying the principle directly to individual texels. Each exemplar texel is therefore monitored during synthesis to prevent excessive reuse, encouraging a more balanced distribution of texel assignments across the synthesized background. To determine an appropriate limit on how often exemplar texels can be reused, our method relies on the synthesis of the first animation frame, which provides a high-quality reference texture. At this initial frame, we record how many times each exemplar texel is assigned to the synthesized texture. From this data, we identify the maximum reuse count, corresponding to the most frequently selected texel. This maximum count then serves as a reference to constrain texel reuse in all subsequent frames.

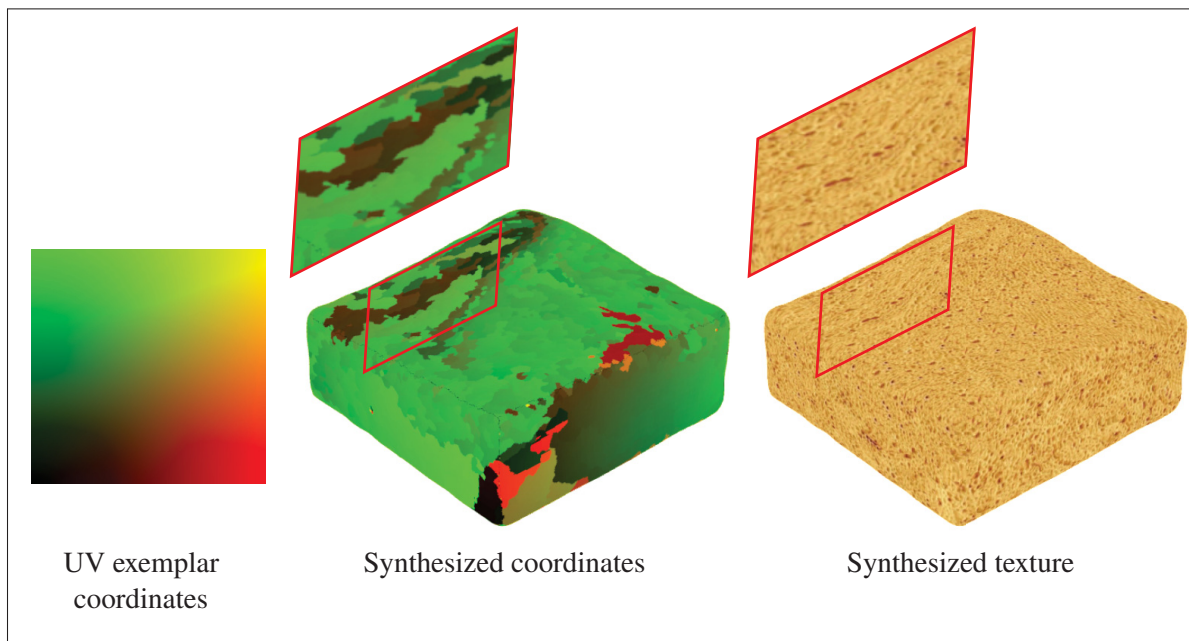


Figure 4.7 Synthesis without texel usage verification. Center: many surface regions reuse exemplar coordinates from the same region instead of more uniformly covering the whole UV space (left). Right: resulting synthesized texture, showing stretching in some areas

Beginning at the second frame, the best-match search correction on the synthesis pyramid is conducted in raster order as opposed to the interleaved order used in chapter 3 and inherited from the synthesis method of Lefebvre & Hoppe (2005). In addition, a texel usage verification

is performed at each level of the synthesis pyramid during the second round of best-match correction. For each synthesized texel, we check whether the texel obtained from the coherent random-walk best-match search has exceeded the maximum reuse count defined in the first frame. If the texel has reached this limit, a random best-match search is then performed to find a new texel. A texel candidate is selected if its L_2 window distance is smaller than that obtained by the coherent best-match and its usage does not exceed the maximum limit. If no such candidate is found after a predefined number of iterations (in our experiments, corresponding to a quarter of the number of exemplar texels), the texel with the smallest L_2 distance and that did not reach the usage limit, is selected. After completing the verification and best-match computations at all levels, the output texture is generated by converting the UV coordinates to RGB colors, as in the original method.

Texel usage verification produces a more uniform distribution of exemplar texels during background synthesis, significantly reducing stretching and washout artifacts. Figure 4.8 illustrates the improved texel assignment in the output texture, showing that regions of the background exemplar contribute more evenly to the synthesized result compared with synthesis performed without this texel verification. Algorithm 4.1 illustrates the steps of the approach and highlights the differences compared to the approach of chapter 3.

The last step is to combine the textons and background texture. This is done frame by frame to produce the final textured fluid surface. The background texture serves as the base onto which the textons are overlaid in the corresponding atlas. To smoothly integrate the textons with the background and avoid hard edges, alpha blending is applied along a narrow band around each texton as explained in Sec. 4.2.1. Within this band, the final color of each texel is computed as a weighted combination of the texton and background colors. This approach preserves both the structure of the textons and the visual continuity of the background, resulting in a temporally coherent and visually consistent animated texture.

Algorithm 4.1 Background Synthesis. Colored lines highlight the modifications introduced with respect to the approach presented in chapter 3

```

1 Texture synthesis (Lefebvre & Hoppe, 2006) for  $frame_1$  in  $synthesis\_pyramid(level_m)$ ;
  // Pyramid levels:  $m$  is finest,  $0$  is coarsest
2 Compute the usage of the most frequently used texel in the first synthesized frame;
3 foreach  $frame_i, i \in 2, \dots, n$  do
4   Advect  $synthesis\_pyramid(level_m, frame_{i-1})$  texels to  $advection\_pyramid(level_m,$ 
      $frame_i)$ 
5   foreach  $level_j, j \in m, \dots, 1$  do // fine to coarse
6     | Downsample  $level_j$  to  $level_{j-1}$ ;
7   end foreach
8   Copy  $advection\_pyramid(level_0)$  to  $synthesis\_pyramid(level_0)$ ;
9   foreach  $level_j, j \in 1, \dots, m$  do // coarse to fine
10    foreach  $element \in level_j$  do
11      | if there is stretching at element then
12        |   Set  $synthesis\_pyramid(level_j, element)$  by upsampling from
13        |    $synthesis\_pyramid(level_{j-1}, element)$ ;
14      | else
15        |   Copy  $advection\_pyramid(level_j, element)$  to  $synthesis\_pyramid(level_j,$ 
16        |    $element)$ ;
17      | end if
18    end foreach
19    foreach  $element \in level_j$  do
20      | Apply coherence best-match search correction on  $synthesis\_pyramid(level_j,$ 
21      |  $element)$  in raster order;
22      | if the best-match coordinate usage exceeds limit then
23        |   repeat
24        |     | Compute random best-match search on  $synthesis\_pyramid(level_j,$ 
25        |     |  $element)$ ;
26        |     | if (random best-match usage is under usage limit) and (random
27        |     | best-match has an  $L_2$  distance lower than the coherence best-match
28        |     | search) then
29        |     |   Apply random best-match to element;
30        |     |   break;
31        |     | end if
32        |     | until maximun number of iterations;
33        |     | end if
34      | end foreach
35    end foreach
36  end foreach
37  Convert UV coordinates from  $synthesis\_pyramid(level_m)$  to colors;
38 end foreach

```

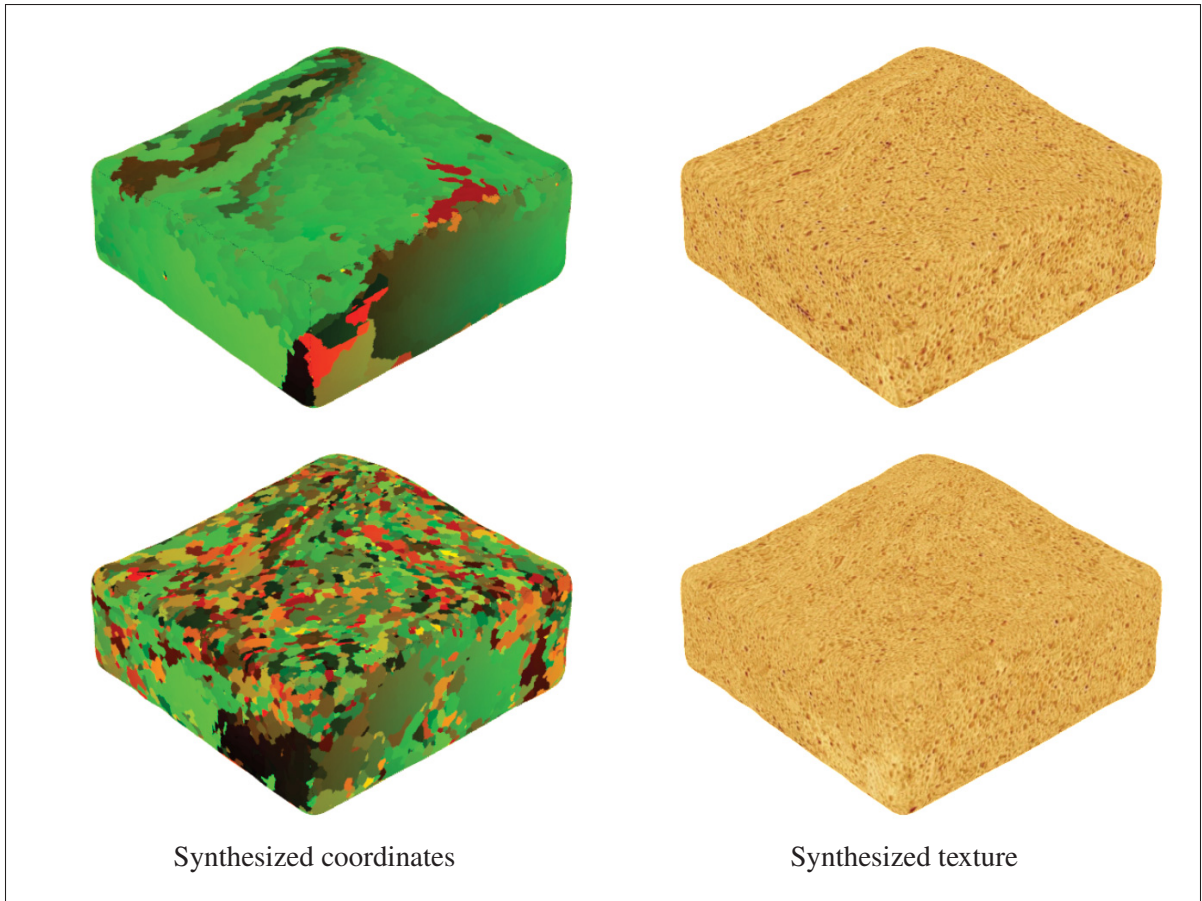



Figure 4.8 Synthesis without (top) and with (bottom) texel usage verification. Bottom-left: more uniform distribution of exemplar coordinates. Bottom-right: synthesized texture showing reduced stretching compared to the unverified case (top-right)

4.3 Results

This section presents the results of our proposed hybrid approach. We first show the results of texton extraction from the input exemplar and the generation of the background texture. We then present the results of texton and background synthesis over animated fluid surfaces, followed by the final merged textured sequence.

4.3.1 Texton Extraction and Background Generation Results

We have tested our approach on a variety of input examples containing different texton shapes and distributions, seen in Figure 4.9, a). The results show that our approach is able to successfully extract the textons, capturing the most representative patterns of each texture, as seen in Figure 4.9, c). For each tested exemplar Figure 4.9, e) shows the background texture generated from the remaining areas after removing the textons. This background texture preserves the global appearance of the exemplar while avoiding visible repetition artifacts.

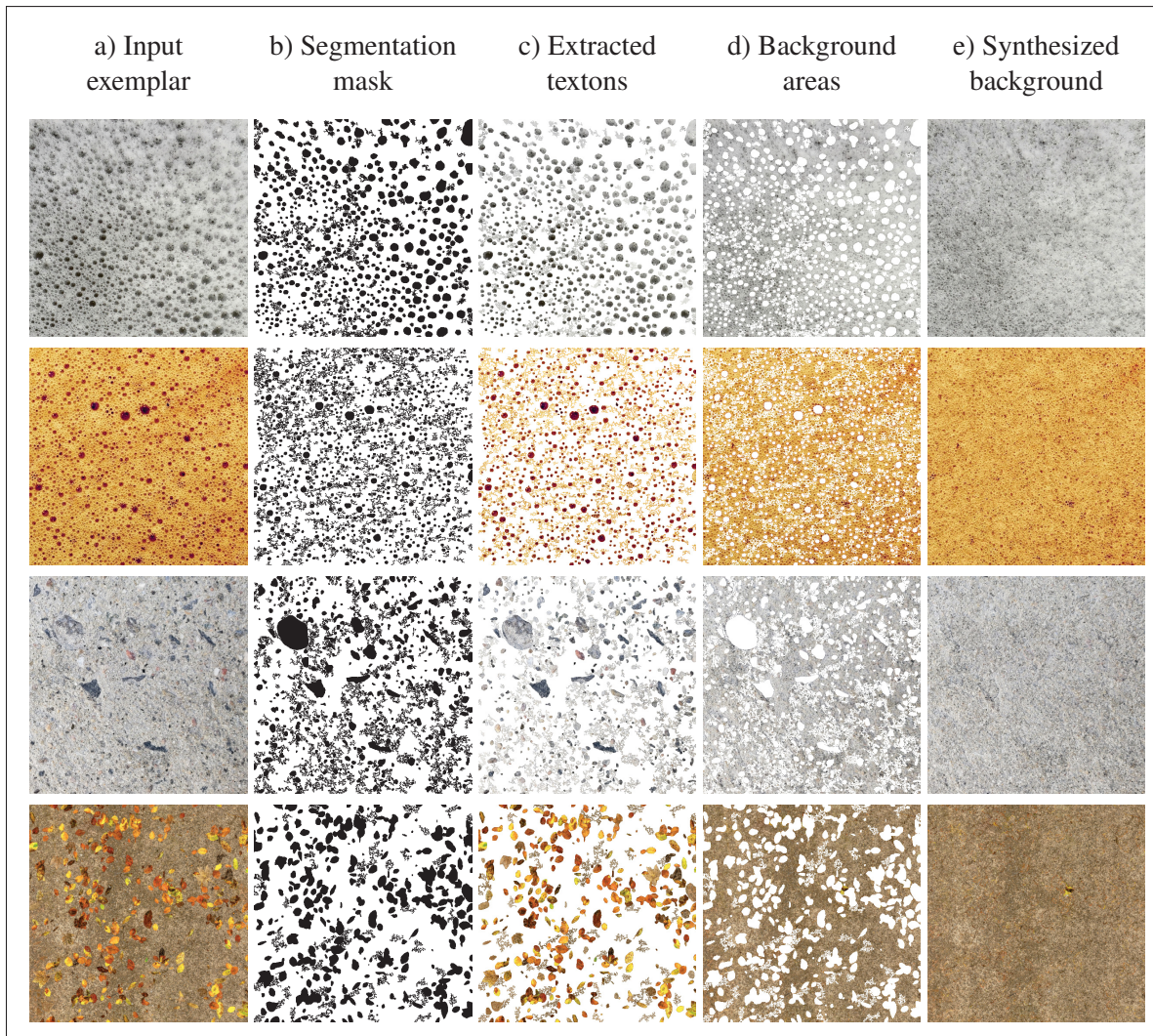


Figure 4.9 Texton extraction and background texture synthesis results

4.3.2 Texton and Background Synthesis Results

We evaluated our hybrid approach across a variety of fluid animation scenarios, including cases with both slow and fast surface motion, as well as scenarios with and without splashes. Figure 4.10 presents the results of the background synthesis simulations.

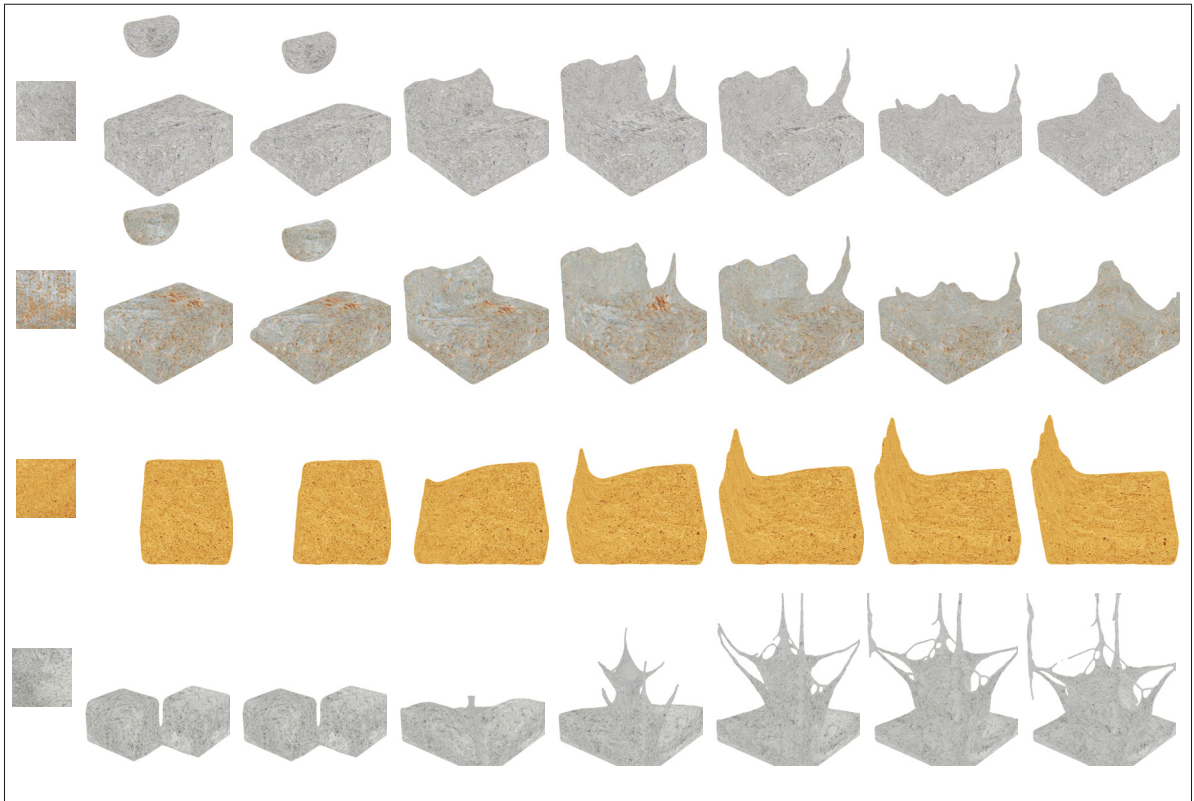


Figure 4.10 Results showing just the synthesized background texture. The input background exemplars are shown in the leftmost column, followed by frames (left to right) extracted from the animation

Figure 4.11 illustrates the final merged results, where the synthesized textons are combined with the background texture. Overall, the results show that our approach produces textures that preserve the main features of the exemplar without distortion, while maintaining fidelity to the input exemplar and ensuring temporal coherence throughout the animation.

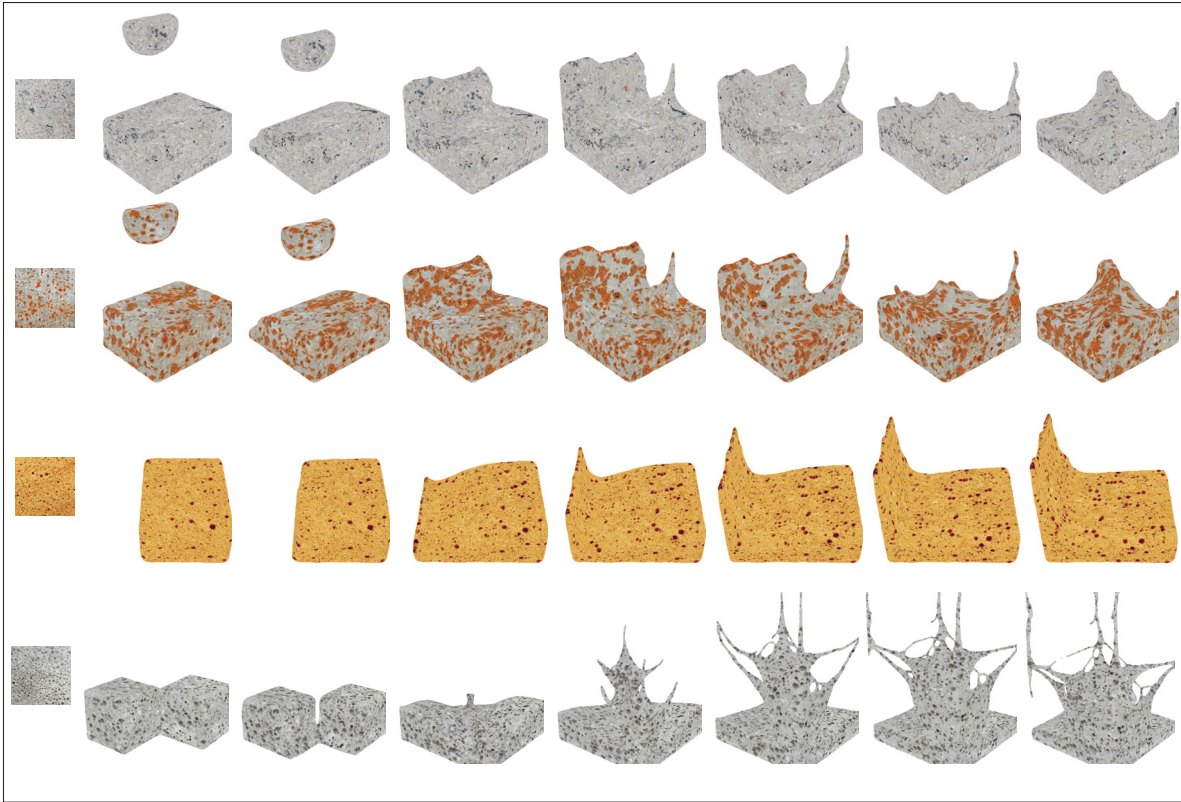


Figure 4.11 Results of our hybrid synthesis approach. The input exemplars are shown in the leftmost column, followed by frames (left to right) extracted from the animation

4.3.3 Comparison

We evaluate our hybrid approach against the method described in chapter 3 (Guzmán *et al.*, 2026). As shown in Figure 4.12, the method of Guzmán *et al.* (2026) suffers from stretching in some areas, leading to distortion of the main features of the exemplar. In addition, some regions exhibit washout effects. In contrast, our approach preserves the shape of the textons without distortion or stretching throughout the animation.

Figure 4.13 shows that, for non-stationary textures, the method of Guzmán *et al.* (2026) gradually loses important features of the input exemplar over time. This results in reduced synthesis quality in later frames compared to the first frame. In contrast, our approach maintains a uniform distribution of textons across the surface, preserving the main features and overall texture fidelity throughout the animation.

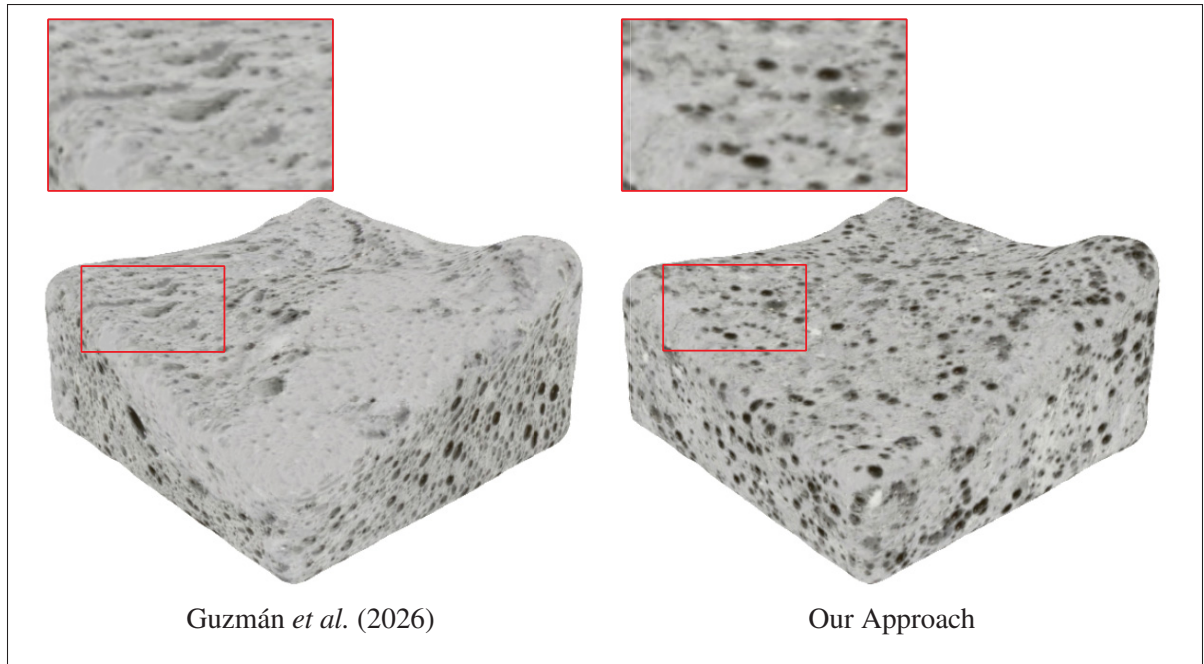


Figure 4.12 Guzmán *et al.* (2026) suffers from stretching (left), while our approach better preserves the main texture features (right)

4.3.4 Limitations

Our approach has some limitations. The quality of the results depends on the characteristics of the input textures. Textures with distinct and identifiable textons that have a reasonably irregular or stochastic distribution are handled well. However, textures with ambiguous or overlapping textons may yield less distinct or incorrect texton representations, which can affect the visual fidelity of the synthesized texture. Figure 4.14 shows some examples of textures in which our approach fails to perform correctly.

Additionally, the synthesis process is more computationally expensive than the traditional texel-based or patch-based methods, since the final results depend on two synthesis methods, despite the fact that they can be computed in parallel. Nonetheless, the approach is designed for offline rendering scenarios, where visual quality and temporal coherence are prioritized over real-time performance.

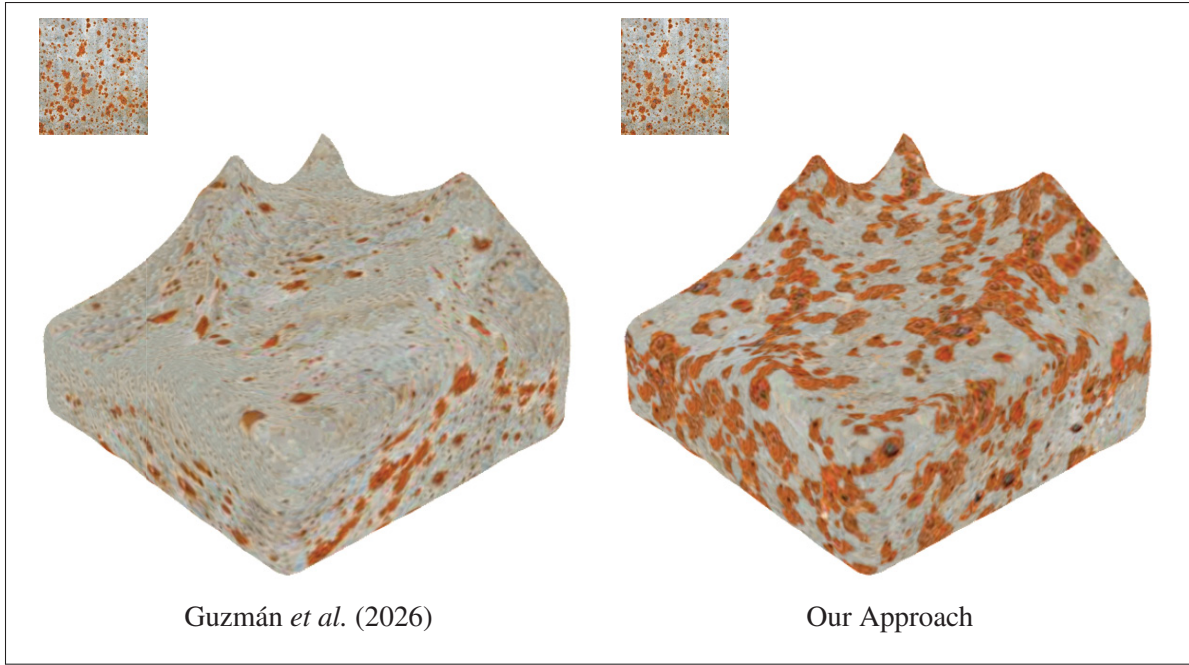


Figure 4.13 Guzmán *et al.* (2026) loses features of the texture (left), while our approach maintains a uniform distribution of the textons (right)

Table 4.1 shows performance statistics for our test cases. The reported times correspond to the average per-frame duration of each stage of our hybrid approach. The background synthesis experiments were performed on an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz (6 cores), with 32 GB of RAM and an NVIDIA RTX A2000 GPU. The texton synthesis experiments were performed on a 13th Gen Intel(R) Core(TM) i9-13900HX (2.20 GHz), with 32 GB of RAM and an NVIDIA GeForce RTX 4070 Laptop GPU.

Table 4.1 Average per-frame computation times (in seconds) for various stages of our hybrid approach

Scenario	Output Resolution	Atlas Texels	Background Advection	Background Synthesis	Texton Synthesis	Total
Sphere dam break	1024 ²	385k	0.567	80.424	52.941	133.932
Double dam break	2048 ²	460k	0.730	103.046	294.957	398.733
Dam break	1024 ²	427k	0.648	90.196	28.365	119.209

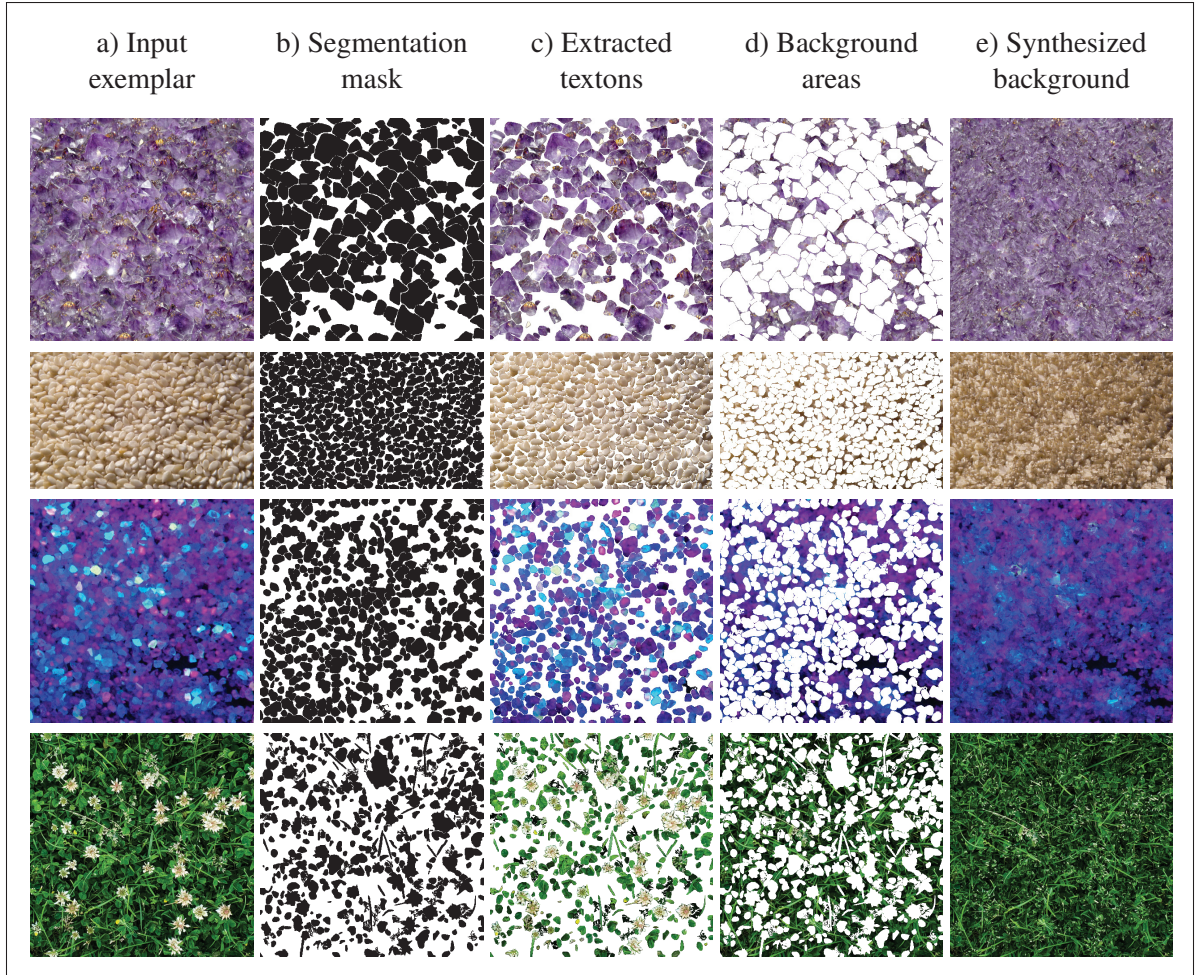


Figure 4.14 Texton extraction and background texture synthesis results

4.4 Discussion

In this chapter, we introduced a hybrid texture synthesis approach for animated fluid surfaces that integrates and extends both texton-based synthesis and adaptive multiresolution texture synthesis. Our approach separates textons from background features in the input exemplar and synthesizes them independently. This enables the preservation of distinctive structures of the input exemplar by explicitly synthesizing textons, which are often lost or distorted in texel-based synthesis methods. At the same time, the synthesized background texture provides a coherent base layer that supports the seamless integration of textons. The approach is effective for exemplars with

clear, identifiable textons and stochastic distributions. The resulting textures maintain visual fidelity to the exemplar and temporal coherence, even under complex fluid motion.

Future work could explore alternative inpainting methods based on diffusion models for generating the background exemplar. Another promising direction is to take the concept of separating textons from background and apply it to generative models, such as diffusion models, which could further improve the quality of the results, simplify the synthesis process, and potentially reduce computational cost.

CONCLUSION AND RECOMMENDATIONS

This thesis introduced a set of methods aimed at enhancing the realism and visual coherence of animated fluids. The main goal was to develop techniques capable of handling complex fluid simulations that exhibit highly dynamic behavior and interactions with complex surfaces. To improve realism in real-time fluid simulations, an enhanced multilayer VP approach was proposed, enabling fluid flow through passages while maintaining real-time performance. Additionally, two texture synthesis frameworks were introduced to enhance the visual appearance and surface textures of animated fluids. Together, these contributions represent a significant advance in the state of the art in real-time fluid simulation and appearance synthesis.

In chapter 2, we presented an enhanced multilayer Virtual Pipes (VP) approach augmented with extended pipes to improve the realism of real-time shallow-water simulations. Our method enables fluid to flow through fully flooded passages and complex configurations that include overhangs and cavities, which was not possible using multi-layered techniques. To achieve this, we first introduced axis-aligned extended pipes, which efficiently manage fully flooded passages aligned with the simulation grid across multiple layers. This approach detects flooded passages each frame and creates an extended pipe that connects the boundary columns of the passage, ensuring continuous and physically coherent flow from one end to the other. To address more complex passages that feature multiple branches or curved paths, we introduced a second approach called unified extended pipes. This approach establishes a unified many-to-many connection among all boundary columns of each passage, allowing the fluid to flow throughout every part of the passage regardless of its shape or orientation. The two approaches were evaluated across a variety of scenarios, demonstrating robust handling of passages with different shapes and configurations. Our experiments showed that the extended-pipe formulations significantly improve the capabilities of the multi-layered VP model, enhancing the visual realism of real-time shallow-water simulations in complex environments. Nevertheless, some limitations remain: the approach may introduce small ripples when extended pipes are connected or disconnected.

Moreover, extending the second approach, unified extended pipes, to support multiple layers remains an open direction for future work. Finally, testing our approach in dynamic scenarios where the terrain changes at every frame and in situations involving interactions with external objects is another avenue for further extension.

In chapter 3, we introduced an approach to synthesize textures for the animated free surfaces of fluids. Because fluids deform and undergo topological changes, it is challenging to maintain fidelity to a reference texture exemplar while avoiding visual artifacts such as distortion and discontinuities. Our method introduces an adaptive multiresolution synthesis framework that balances fidelity to the exemplar with consistency in fluid motion. The process begins with a full multiresolution synthesis on the initial frame and subsequently advects the texture and orientation field across frames, yielding a coherent sequence of textures conforming to the per-frame geometry. Our technique utilizes both a multiresolution advection pyramid and a multiresolution synthesis pyramid. Adaptiveness is achieved using local 2D and 3D distortion measures, which guide per-texel decisions. Each texel is either resynthesized from the synthesis pyramid, to better match the exemplar texture, or preserved from the advection pyramid, to maintain temporal coherence. Additionally, our approach reduces popping artifacts by enforcing gradual color changes over time. The method was evaluated across a variety of texture exemplars and fluid simulations involving complex deformations and topological changes. The results show that the adaptive multiresolution strategy performs well, preserving the fidelity to the exemplars while maintaining temporal coherence, particularly with stationary textures. Despite these good results, there are potential directions for future work. Performing a thorough synthesis, like the one at the first frame, could “reset” the quality of the results. Future work could investigate how to conduct this thorough synthesis on a regular basis and how to seamlessly blend it with the results from the regular frame-to-frame synthesis. Additionally, integrating the method with neural texture synthesis techniques could broaden its applicability to a wider range of visual

effects for fluids. Finally, we identified that enhancing the synthesis quality for non-stationary textures was an important future work and pursued it in chapter 4.

In Chapter 4, we presented a hybrid approach for texture synthesis on animated fluid surfaces that integrates texton-based and texel-based background synthesis methods. This work was developed in collaboration with Carleton University. Our approach synthesizes textons and background independently. To achieve this, the process begins by extracting the textons from the input exemplar and generating a new background exemplar with only the less prominent features. Texton-based synthesis preserves the main texture features by synthesizing the textons and advecting them from one frame to the next according to the fluid motion. Background synthesis also relies on the fluid motion, but this time to fill all texels of the current frame based on the texture from the previous frame. This is done by advecting current frame positions backward to the previous frame. As we want the regular advection of the textons to be consistent with the backward advection used in the background texture synthesis, we compute an average velocity between consecutive frames, similar to Heun’s method. To improve the background synthesis results, we introduced a mechanism that ensures more uniform usage of all texels from the exemplar. Finally, the synthesized textons and background texture are merged to produce the final textured surface, applying alpha blending around the textons to ensure smooth transitions. The results show that our hybrid approach effectively reduces common artifacts of traditional pixel-based or patch-based methods, such as texture stretching, ghosting, and wash-out. Furthermore, it preserves the main structures of the input exemplar and maintains high visual fidelity and temporal coherence, even under complex fluid motion. However, textures without distinct, identifiable textons do not perform well with our approach. Additionally, the method is computationally expensive, since it involves two synthesis processes. Future work could explore alternative inpainting methods for generating the background texture, and integration with generative models to further improve quality and reduce computational cost.

The methods presented in this thesis significantly improve both the physical realism and visual fidelity of fluid simulations. The approach proposed in chapter 2 improves the realism of real-time water simulations, while the approaches introduced in chapter 3 and chapter 4 enhance the visual quality of fluid simulations. These techniques can be applied in many areas, from the entertainment industry, VFX, and interactive games to real-time medical simulations. Together, these contributions provide a foundation for continued progress in realistic fluid animation and appearance synthesis.

LIST OF REFERENCES

- Aanjaneya, M., Gao, M., Liu, H., Batty, C. & Sifakis, E. (2017). Power Diagrams and Sparse Paged Grids for High Resolution Adaptive Liquids. *ACM Trans. Graph.*, 36(4), 140:1–140:12.
- Angst, R., Thuerey, N., Botsch, M. & Gross, M. (2008). Robust and Efficient Wave Simulations on Deforming Meshes. *Computer Graphics Forum*, 27(7), 1895–1900.
- Bargteil, A. W., Sin, F., Michaels, J. E., Goktekin, T. G. & O'Brien, J. F. (2006). A Texture Synthesis Method for Liquid Animations. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (SCA '06), 345–351.
- Barnes, C., Shechtman, E., Finkelstein, A. & Goldman, D. B. (2009). PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28(3), 24.
- Benoit, J. & Paquette, E. (2015). Localized Search for High Definition Video Completion. *J. WSCG*, 23(1), 45–54.
- Borgeat, L., Massicotte, P., Poirier, G. & Godin, G. (2011). Layered Surface Fluid Simulation for Surgical Training. *Proc. of Medical Image Computing and Computer-Assisted Intervention – MICCAI 2011 Conference*, pp. 323–330.
- Busto, P. P., Eisenacher, C., Lefebvre, S., Stamminger, M. et al. (2010). Instant Texture Synthesis by Numbers. *VMV*, pp. 81–85.
- Chen, K., Johan, H. & Mueller-Wittig, W. (2013). Simple and efficient example-based texture synthesis using tiling and deformation. *Proceedings of the ACM SIGGRAPH symposium on interactive 3D graphics and games*, pp. 145–152.
- Chentanez, N. & Müller, M. (2010). Real-time Simulation of Large Bodies of Water with Small Scale Details. *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, (SCA '10), 197–206.
- Clément, O., Benoit, J. & Paquette, E. (2007). Efficient editing of aged object textures. *Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pp. 151–158.
- Dagenais, F., Gagnon, J. & Paquette, E. (2012). A prediction-correction approach for stable sph fluid simulation from liquid to rigid. *Proc. of Comp. Graph. Intl*, 60, 65.
- Dagenais, F., Gagnon, J. & Paquette, E. (2017). Detail-Preserving Explicit Mesh Projection and Topology Matching for Particle-Based Fluids. *Computer Graphics Forum*, 36, 444–457.

- Dagenais, F., Guzmán, J., Vervondel, V., Hay, A., Delorme, S., Mould, D. & Paquette, E. (2018a). Real-Time Virtual Pipes Simulation and Modeling for Small-Scale Shallow Water. *Proceedings of VRIPHYS 2018 Workshop on Virtual Reality Interaction and Physical Simulation*.
- Dagenais, F., Vervondel, V., Guzman, J. E., Hay, A., Delorme, S., Mould, D. & Paquette, E. (2018b). Extended virtual pipes for the stable and real-time simulation of small-scale shallow water. *Computers & Graphics*, 76, 84–95.
- Darabi, S., Shechtman, E., Barnes, C., Goldman, D. B. & Sen, P. (2012). Image Melding: Combining Inconsistent Images Using Patch-Based Synthesis. *ACM Trans. Graph.*, 31(4).
- Dong, Y., Lefebvre, S., Tong, X. & Drettakis, G. (2008). Lazy solid texture synthesis. *Computer Graphics Forum*, 27, 1165–1174.
- Efros, A. A. & Leung, T. K. (1999, September). Texture Synthesis by Non-parametric Sampling. *IEEE International Conference on Computer Vision*, pp. 1033-1038.
- Enright, D., Marschner, S. & Fedkiw, R. (2002). Animation and Rendering of Complex Water Surfaces. *ACM Trans. Graph.*, 21(3), 736–744.
- Ferstl, F., Ando, R., Wojtan, C., Westermann, R. & Thuerey, N. (2016). Narrow Band FLIP for Liquid Simulations. *Computer Graphics Forum*, 35(2), 225–232.
- Gagnon, J., Dagenais, F. & Paquette, E. (2016). Dynamic Lapped Texture for Fluid Simulations. *Vis. Comput.*, 32(6-8), 901–909.
- Gagnon, J., Guzmán, J. E., 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., Guzmán, J. E., Mould, D. & Paquette, E. (2021). Patch erosion for deformable lapped textures on 3D fluids. *Computer Graphics Forum*, 40(2), 367–374.
- Guingo, G., Sauvage, B., Dischler, J.-M. & Cani, M.-P. (2017). Bi-Layer Textures: a Model for Synthesis and Deformation of Composite Textures. *Computer Graphics Forum*. doi: 10.1111/cgf.13229.
- Guingo, G., Larue, F., Sauvage, B., Lutz, N., Dischler, J.-M. & Cani, M.-P. (2020). Content-aware texture deformation with dynamic control. *Computers & Graphics*, 91, 95-107. doi: <https://doi.org/10.1016/j.cag.2020.07.006>.

- Guzmán, J. E., Mould, D. & Paquette, E. (2026). Adaptive multiresolution exemplar-based texture synthesis on animated fluids. *Computers & Graphics*, 134, 104490. doi: <https://doi.org/10.1016/j.cag.2025.104490>.
- Hu, A., Desai, N., Abu Alhaija, H., Kim, S. W. & Shugrina, M. (2024). Diffusion texture painting. *ACM SIGGRAPH 2024 Conference Papers*, pp. 1–12.
- Hurtut, T., Landes, P.-E., Thollot, J., Gousseau, Y., Drouillhet, R. & Coeurjolly, J.-F. (2009). Appearance-guided synthesis of element arrangements by example. *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*, (NPAR '09), 51–60. doi: 10.1145/1572614.1572623.
- Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A. & Teschner, M. (2014). SPH Fluids in Computer Graphics. *Eurographics 2014 - State of the Art Reports*.
- 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).
- Jeschke, S. & Wojtan, C. (2017). Water Wave Packets. *ACM Trans. Graph.*, 36(4), 103:1–103:12.
- Jiang, C., Schroeder, C., Selle, A., Teran, J. & Stomakhin, A. (2015). The Affine Particle-in-cell Method. *ACM Trans. Graph.*, 34(4), 51:1–51:10.
- Julesz, B. & Bergen, J. R. (1983). Human factors and behavioral science: Textons, the fundamental elements in preattentive vision and perception of textures. *The Bell System Technical Journal*, 62(6), 1619-1645. doi: 10.1002/j.1538-7305.1983.tb03502.x.
- Kaspar, A., Neubert, B., Lischinski, D., Pauly, M. & Kopf, J. (2015). Self Tuning Texture Optimization. *Computer Graphics Forum*, 34(2), 349-359.
- Kass, M. & Miller, G. (1990). Rapid, Stable Fluid Dynamics for Computer Graphics. *Comput. Graph.*, 24(4), 49–57.
- Kellomäki, T. (2014). Rigid Body Interaction for Large-scale Real-time Water Simulation. *Int. J. Comput. Games Technol.*
- Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y. et al. (2023). Segment anything. *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4015–4026.
- Kopf, J., Fu, C.-W., Cohen-Or, D., Deussen, O., Lischinski, D. & Wong, T.-T. (2007). Solid texture synthesis from 2D exemplars. *ACM Trans. Graph.*, 26(3).

- Kovács, Á. S., Hermosilla, P. & Raidou, R. G. (2024). Surface-aware Mesh Texture Synthesis with Pre-trained 2D CNNs. *Computer Graphics Forum*, 43(2), e15016.
- 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.
- Layton, A. T. & van de Panne, M. (2002). A Numerically Efficient and Stable Algorithm for Animating Water Waves. *The Visual Computer*, 18(1), 41–53.
- Lee, R. & O’Sullivan, C. (2007). A Fast and Compact Solver for the Shallow Water Equations. *Workshop in Virtual Reality Interactions and Physical Simulation (VRIPHYS)*, pp. 51–57.
- Lefebvre, S. & Hoppe, H. (2005). Parallel Controllable Texture Synthesis. *ACM Trans. Graph.*, 24(3), 777–786.
- Lefebvre, S. & Hoppe, H. (2006). Appearance-space Texture Synthesis. *ACM Trans. Graph.*, 25(3), 541–548.
- Lefebvre, S., Hornus, S. & Neyret, F. (2005). Texture sprites: texture elements splatted on surfaces. *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, (I3D ’05), 163–170. doi: 10.1145/1053427.1053454.
- Li, J., Wang, S. & Paquette, E. (2024). Texture-driven adaptive mesh refinement with application to 3D relief. *Computer-Aided Design*, 167, 103640.
- Liu, X., Jiang, L., Wong, T.-T. & Fu, C.-W. (2012). Statistical Invariance for Texture Synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 18(11), 1836–1848.
- Lockerman, Y. D., Sauvage, B., Allègre, R., Dischler, J.-M., Dorsey, J. & Rushmeier, H. (2016). Multi-scale label-map extraction for texture synthesis. *ACM Trans. Graph.*, 35(4). doi: 10.1145/2897824.2925964.
- Macklin, M. & Müller, M. (2013). Position Based Fluids. *ACM Trans. Graph.*, 32(4), 104:1–104:12.
- Mei, X., Decaudin, P. & Hu, B.-G. (2007, Oct). Fast Hydraulic Erosion Simulation and Visualization on GPU. *Pacific Conference on Computer Graphics and Applications*, PG ’07, pp. 47-56.

- Mitchel, T. W., Esteves, C. & Makadia, A. (2024, June). Single Mesh Diffusion Models with Field Latents for Texture Generation. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7953-7963.
- Mould, D. & Yang, Y.-H. (1997). Modeling Water for Computer Graphics. *Computers & Graphics*, 21(6), 801–814.
- Narain, R., Kwatra, V., Lee, H.-P., Kim, T., Carlson, M. & Lin, M. C. (2007). Feature-Guided Dynamic Texture Synthesis on Continuous Flows. *Rendering Techniques*, pp. 361–370.
- Neyret, F. (2003). Advected textures. *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 147–153.
- O’Brien, J. F. & Hodgins, J. K. (1995, Apr). Dynamic Simulation of Splashing Fluids. *Proc. of Computer Animation ’95*, pp. 198-205, 220.
- Palazzolo, C., van Kaick, O. & Mould, D. (2025). By-Example Synthesis of Vector Textures. *Proc. Pacific Graphics*.
- Perla, S. R. K., Wang, Y., Mahdavi-Amiri, A. & Zhang, H. (2024). EASI-TeX: Edge-Aware Mesh Texturing from Single Image. *ACM Trans. Graph.*, 43(4). doi: 10.1145/3658222.
- 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. *Proc. ACM Comput. Graph. Interact. Tech.*, 4, 40:1–40:26.
- Sendik, O. & Cohen-Or, D. (2017). Deep Correlations for Texture Synthesis. *ACM Trans. Graph.*, 36(5).
- Sheffer, A., Lévy, B., Mogilnitsky, M. & Bogomyakov, A. (2005). ABF++: Fast and Robust Angle Based Flattening. *ACM Trans. Graph.*, 24(2), 311–330.
- Sivakumaran, G., Paquette, E. & Mould, D. (2022). Time Reversal and Simulation Merging for Target-Driven Fluid Animation. *Proceedings of the 15th ACM SIGGRAPH Conference on Motion, Interaction and Games*, (MIG ’22). doi: 10.1145/3561975.3562952.
- Takayama, K., Okabe, M., Ijiri, T. & Igarashi, T. (2008). Lapped solid textures: filling a model with anisotropic textures. *ACM Trans. Graph.*, 27(3), 1–9.

- Tong, X., Zhang, J., Liu, L., Wang, X., Guo, B. & Shum, H.-Y. (2002). Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics (ToG)*, 21(3), 665–672.
- Tu, P., Wei, L.-Y., Yatani, K., Igarashi, T. & Zwicker, M. (2020). Continuous curve textures. *ACM Trans. Graph.*, 39(6). doi: 10.1145/3414685.3417780.
- Tu, P., Wei, L.-Y. & Zwicker, M. (2022). Clustered vector textures. *ACM Trans. Graph.*, 41(4). doi: 10.1145/3528223.3530062.
- Št'ava, O., Beneš, B., Brisbin, M. & Křivánek, J. (2008). Interactive Terrain Modeling Using Hydraulic Erosion. *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.*, (SCA '08), 201–210.
- Wang, H., Miller, G. & Turk, G. (2007). Solving General Shallow Wave Equations on Surfaces. *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.*, (SCA '07), 229–238.
- Wei, L.-Y. & Levoy, M. (2000). Fast Texture Synthesis Using Tree-Structured Vector Quantization. *Proc. of SIGGRAPH 00*, (Annual Conference Series), 479–488.
- Wei, L.-Y. & Levoy, M. (2001). Texture Synthesis over Arbitrary Manifold Surfaces. *Proc. 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 State of the Art Report*, pp. 93–117.
- 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.
- Yu, X., Yuan, Z., Guo, Y.-C., Liu, Y.-T., Liu, J., Li, Y., Cao, Y.-P., Liang, D. & Qi, X. (2024). TEXGen: a Generative Diffusion Model for Mesh Textures. *ACM Trans. Graph.*, 43(6). doi: 10.1145/3687909.
- Yuksel, C., House, D. H. & Keyser, J. (2007). Wave Particles. *ACM Trans. Graph.*, 26(3).
- Zhou, Y., Zhu, Z., Bai, X., Lischinski, D., Cohen-Or, D. & Huang, H. (2018). Non-Stationary Texture Synthesis by Adversarial Expansion. *ACM Trans. Graph.*, 37(4).