Computers & Graphics 134 (2026) 104490

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Adaptive multiresolution exemplar-based texture synthesis on animated fluids☆

Julián E. Guzmán [a],*, David Mould [b], Eric Paquette [a]

[a] École de technologie supérieure, 1100 Notre-Dame Street West, Montréal, H3C 1K3, Québec, Canada
[b] Carleton University, 1125 Colonel By Drive, Ottawa, K1S 5B6, Ontario, Canada

ARTICLE INFO

Keywords:
Texture synthesis
Animation
Fluid simulation

ABSTRACT

We propose an approach to synthesize textures for the animated free surfaces of fluids. Because fluids deform and experience topological changes, it is challenging to maintain fidelity to a reference texture exemplar while avoiding visual artifacts such as distortion and discontinuities. We introduce an adaptive multiresolution synthesis approach that balances fidelity to the exemplar and consistency with 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 approach advects the texture and the orientation field across frames, yielding a coherent sequence of textures conforming to the per-frame geometry. Our adaptiveness relies on local 2D and 3D distortion measures, which guide multiresolution decisions to resynthesize or preserve the advected content. We prevent popping artifacts by enforcing gradual changes in color over time. Our approach works well both on slow-moving liquids and on turbulent ones with splashes. In addition, we demonstrate good performance on a variety of stationary texture exemplars.

1. Introduction

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 resolution. 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. 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 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, the velocity field advects 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

☆ This article was recommended for publication by Peng-Shuai Wang.

* Corresponding author.
E-mail addresses: julian-edgardo.guzman-cortes.1@ens.etsmtl.ca (J.E. Guzmán), mould@scs.carleton.ca (D. Mould), eric.paquette@etsmtl.ca (E. Paquette).

https://doi.org/10.1016/j.cag.2025.104490
Received 11 August 2025; Received in revised form 12 November 2025; Accepted 13 November 2025
Available online 3 December 2025

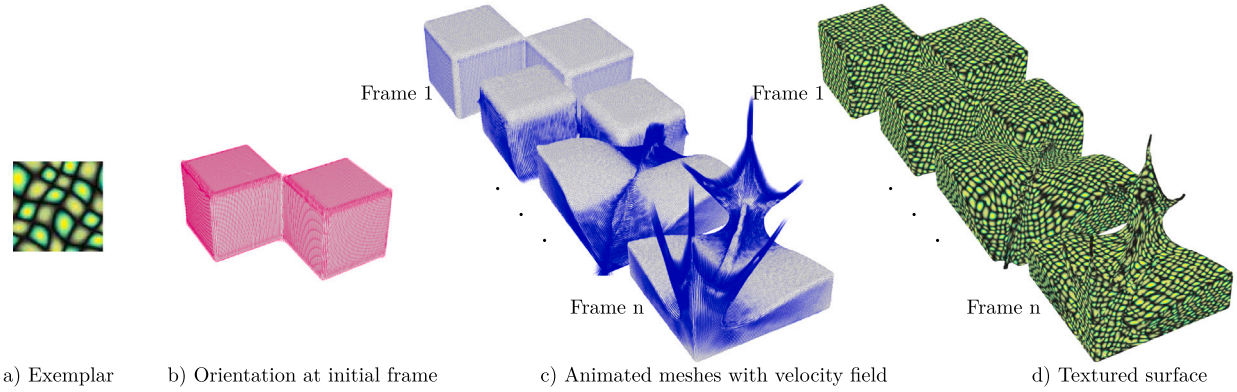

Fig. 1. Our synthesis process. Inputs: (a) texture exemplar, (b) 3D mesh together with an orientation field, (c) animated frames with velocity field. (d) We synthesize an animated texture which can be used in off-the-shelf 3D software.

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. The paper contains 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.

2. Related work

Our review concentrates on exemplar-based texture synthesis methods. Such methods take as input an exemplar texture and synthesize a new output texture that is visually the same as the exemplar. Additional information about the texture synthesis area can be found in the survey by Wei et al. [1].

Pixel-based methods synthesize the output texture on a pixel-by-pixel basis. Efros and Leung [2] pioneered this method, using a best-match search to find a neighborhood in the exemplar similar to the neighborhood in the output. Wei and Levoy [3] accelerated the search with a specialized data structure and later used the same technique to synthesize colors on 3D meshes [4]. 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. Lefebvre and Hoppe [5] synthesize textures by working on the UV coordinates of the exemplar instead of RGB colors; this work was later extended [6] to 3D objects by parameterizing the surface to a 2D atlas where the synthesis is conducted. Appearance-space texture synthesis [6] introduced compressed search windows and drastically reduced synthesis time. It also support 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 and Hoppe [6] is its long precomputation times; subsequent methods [7,8] replace the offline precomputation step with an online random walk search that is equally effective, greatly reducing precomputation. Our approach uses the appearance-space compression and atlas of Lefebvre and Hoppe [6] together with the random walk search of Busto et al. [8]. We provide a significant extension to the method of Lefebvre and Hoppe [6] which is limited to static meshes.

Texture optimization methods [9–11] optimize to find a set of overlapping windows of texels from the exemplar. Optimization was also used to synthesize texture based on a 2D orientation field [12]. On structured exemplars, such methods perform remarkably well. However, as noted by Jamriška et al. [13], applying them to more stochastic textures tends to produce *wash-out*, where detail is lost in regions of the output texture. Jamriška et al. [13] reduce wash-out by preventing excessive reuse of the same window of texels from the exemplar. Despite these improvements, most optimization methods [9–12] are limited to synthesizing static 2D textures. While the method of Jamriška et al. [13] can synthesize animated textures, the textures remain 2D and cannot be applied to dynamic surfaces.

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. [14] 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 et al. [15] proposed a method that takes advantage of the GPU. Chen et al. [16] proposed a method based on tiling, deformation, and resampling techniques, capable of working in real-time. Takayama et al. [17] 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.

Texture synthesis with Convolutional Neural Networks (CNN) [18] and Generative Adversarial Networks (GANs) [19] 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 et al. [20] partly solves the problem by defining CNN operations on meshes, then minimizing the distances of VGG-19 features measured over the mesh and the examplar. The process is slow (around 50 min for a 1024×1024 output) and produces blurry patch seams related to overlapping CNN footprints. Instead of using CNNs, the method of Hu et al. [21] 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. [21] and Kovács et al. [20] 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 [22,23], 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 et al. [24] 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.

Texturing Liquids. Currently, few methods can address texture synthesis over the evolving surface of simulated liquids. The method of Neyret [25] advects a texture, but is limited to 2D advection. In a similar way, the methods of Lefebvre and Hoppe [6], Yu et al. [26], and Jamriška et al. [13] support 2D advection, but they are not directly applicable to the problem of texturing 3D fluids. Methods like those of Bargteil et al. [27], Kwatra et al. [28], and Narain et al. [29] 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. [9], they also suffer from wash-out with some textures. Yet other methods [30–32] texture liquids by performing a patch-based texture synthesis. These patch-based methods present issues like stiff texture patterns [30], visible color discontinuities between patches [32], and ghosting [31]. While these methods [30–32] rely on a texture atlas like ours, rendering to the atlas from the patches is a time-consuming process.

Appearance-space texture synthesis. Since our approach extends the method of Lefebvre and Hoppe [6], we will describe that method in more detail. Their method parameterizes 3D meshes to 2D atlases where the output texture texels will be computed. Instead of working with texture colors, the method uses 8D appearance-space vectors; instead of storing such 8D vectors in the output texture directly, it stores UV coordinates linking back to the compressed exemplar.

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 as the coarser texel did (and also accounting for an orientation field through its Jacobian; see the paper [6] for additional details). After the upsampling step, two successive passes of best-match search are performed to refine the UV coordinates at the current resolution level. For each output texel, the exemplar UV coordinates are used to retrieve their corresponding 8D appearance-space vectors. The best match is the one with least L_2 distance on the 8D appearance-space vectors. The UV coordinates of the best match from the exemplar are then written to the output texel. The method processes the other levels in the same fashion, proceeding coarse to fine. After computing all best-match searches of all levels, the output texture is populated by converting the UV coordinates to RGB colors.

3. 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 and Hoppe [6] 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. 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 (Section 3.3) 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.

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 (Section 3.5). We detect stretched areas (see Section 3.4) 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 (Section 3.6), 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. Fig. 2 and algorithm 1 summarize the steps of our approach.

3.1. First frame synthesis

At the first frame, we synthesize the texture using an existing appearance-space texture synthesis method [6]. 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.

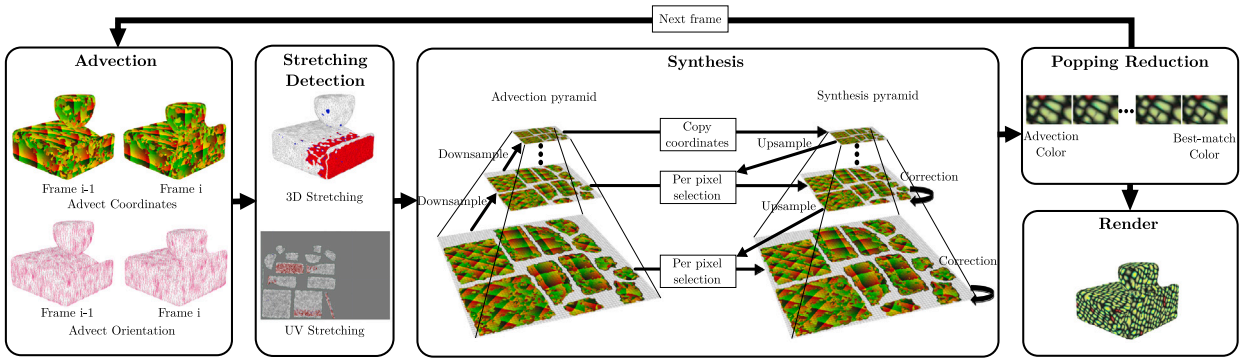

Fig. 2. Pipeline of our approach. The synthesized UV coordinates are mapped to the red and green channels for depiction (boxes corresponding to advection and synthesis).

Texture synthesis [6] for $frame_1$ in synthesis_pyramid($level_m$);
// Pyramid levels: m is finest, 0 is coarsest
foreach $frame_i, i \in 2, \ldots, n$ **do**
  // advection pyramid
  Advect synthesis_pyramid($level_m$, $frame_{i-1}$) texels to advection_pyramid($level_m$, $frame_i$) **foreach** $level_j, j \in m, \ldots, 1$ **do** // fine to coarse
    Downsample $level_j$ to $level_{j-1}$;
  // synthesis pyramid
  Copy advection_pyramid($level_0$) to synthesis_pyramid($level_0$) ;
  foreach $level_j, j \in 1, \ldots, m$ **do** // coarse to fine
    foreach *element* $\in level_j$ **do**
      if *there is stretching at element* **then**
        Set synthesis_pyramid($level_j$, element) by upsampling from synthesis_pyramid($level_{j-1}$, element) ;
      else
        Copy advection_pyramid($level_j$, element) to synthesis_pyramid($level_j$, element);
    Apply best-match search correction on synthesis_pyramid($level_j$) ;
  Convert UV coordinates from synthesis_pyramid($level_j$) to colors ;

Algorithm 1: Overview of our approach.

3.2. 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 et al. [33]. 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 subsection.

3.3. 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 Fig. 3.

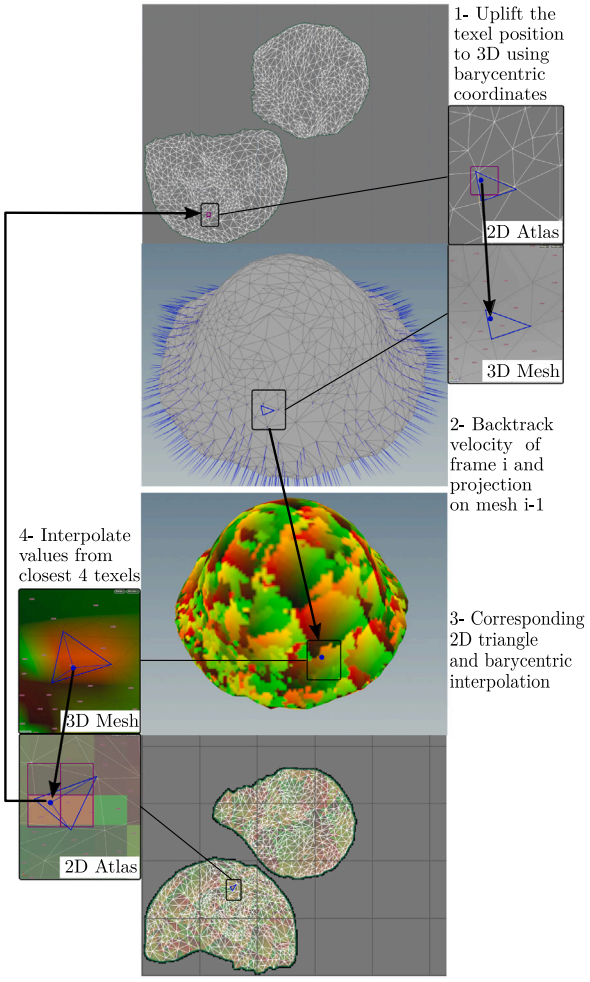

Fig. 3. UV coordinates of a current frame i are computed by backtracking to the previous frame based on the velocity field.

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

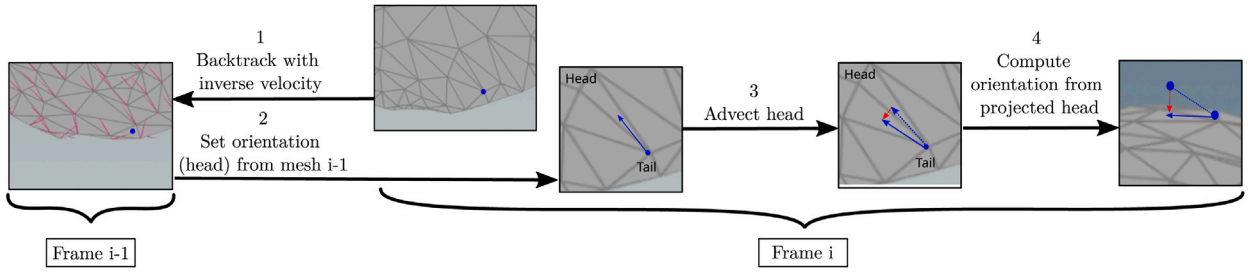

Fig. 4. Advection of the orientation field by backtracking to the position on the previous frame.

velocities $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ using barycentric interpolation:

$$\mathbf{v} = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + w_3\mathbf{v}_3.$$

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

$$\mathbf{p}' = \mathbf{p} - \Delta t \cdot \mathbf{v}.$$

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; details are found in Section 3.7. After we are finished populating the finest level of the pyramid through advection, we downsample to coarser levels.

Akin to the method of Lefebvre and Hoppe [6], 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 and Hoppe [6], we need to deal with the advection of that orientation field throughout the animation. We use the method of Kwatra et al. [28] 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.

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. [28], 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 Fig. 4.

As we aim to have an orientation field with as few discontinuities as possible, we complete the orientation field update with a smoothing step [28]. Smoothing is particularly important when there are topological changes in the surface (e.g., splashes).

3.4. Stretching detection

When surfaces deform and experience significant topological changes, advection can generate textures with visible artifacts. Texture stretching, as depicted in Fig. 5, 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 neighbors in the current frame.

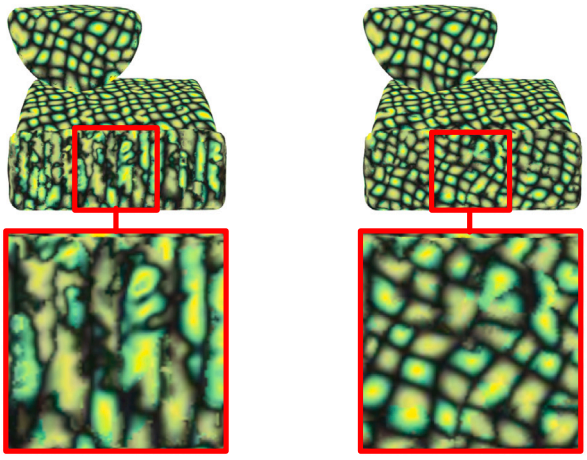

Fig. 5. Advected texture after a single frame of animation compared to the result after our synthesis approach. Left: Without synthesis, the texture becomes severely stretched, even after a single frame. Right: Our approach detects stretched regions and synthesizes new texture there.

The 3D stretching criterion is calculated for each texel at the finest resolution. 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.

The UV coordinate distance criterion is also computed per texel at the finest resolution. 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.

3.5. 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) 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 [8] based on PCA compressed exemplar features [6].

3.6. 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 (Section 3.3), but snapping to the closest texel in the previous frame, to avoid having the transition region grow every frame.

3.7. 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 (Section 3.2), 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 (Section 3.3). 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. Fig. 6 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. 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

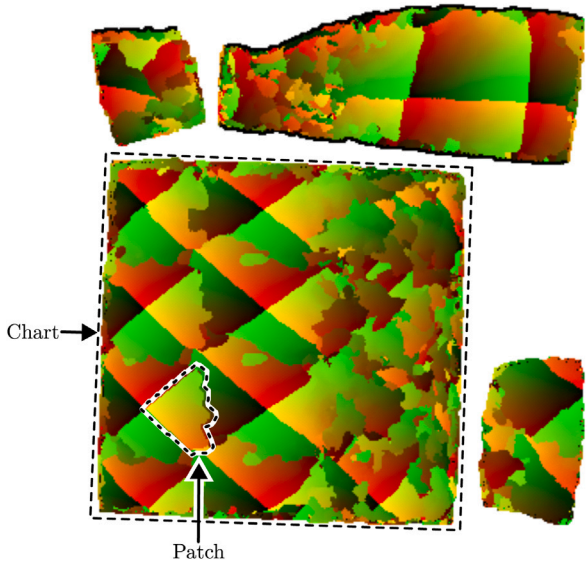

Fig. 6. A typical atlas with what we refer to as *charts* (separate parts of the mesh flattened to 2D) and *patches* (contiguous group of texels with smoothly varying UV coordinates).

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 3.4 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 [8] in Section 3.5, 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. 6) 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 (Section 3.6) 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.

4. 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. Fig. 7 shows representative frames. In the first scenario (sphere), 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. In the second scenario (double dam break), 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. In the third scenario (viscous drop), 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. Finally, for the fourth scenario (viscous dam break), the flow evolves more slowly, without abrupt changes or fragmentation, emphasizing temporal consistency in more stable simulations. 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 and demonstrate good temporal continuity of the texture patterns.

4.1. Comparison with existing methods

We compared our approach against relevant previous methods: those which generate textures on the free surface of dynamic fluids. Fig. 8 and the accompanying video show the comparison between our approach and the methods of Kwatra et al. [28] and Gagnon et al. [31]. We can see that our texture colors are more faithful to the exemplar than those produced by the method of Kwatra et al. [28]. The method of Gagnon et al. [31] 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.

In addition, we evaluated our approach against the method of Gagnon et al. [30], which synthesizes textures using patches. As shown in Fig. 9, 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-texel synthesis and avoids such artifacts, producing smoother and more coherent results.

We also compared our approach to the method of Gagnon et al. [32], which uses deformable patches to synthesize textures on fluids. Fig. 10 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

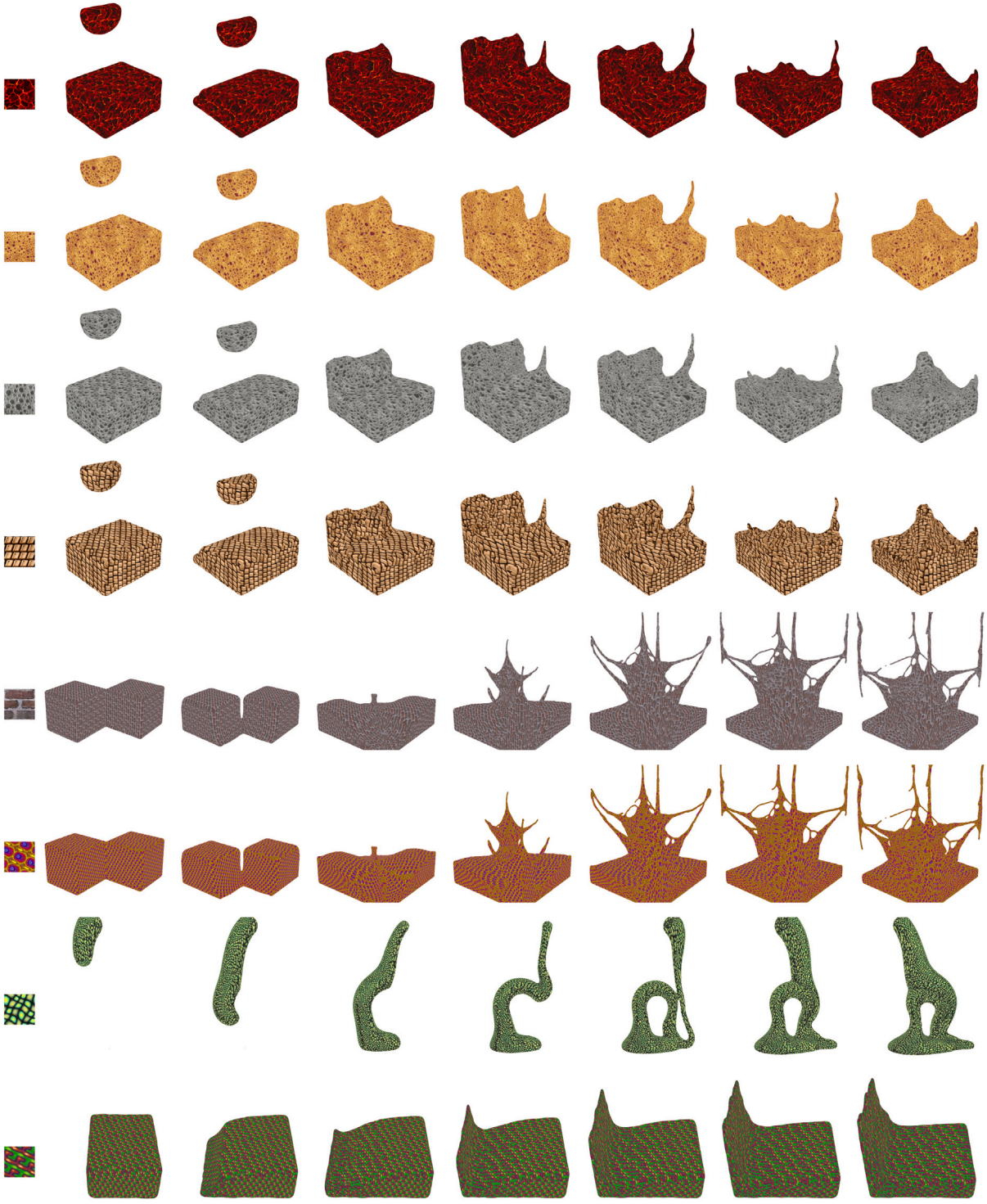

Fig. 7. Results from our approach. Scenario names, top to bottom: sphere, double dam break, viscous drop, viscous dam break. The exemplars are shown in the leftmost column. Left to right: frames extracted from the animation.

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.

Table 1 shows performance statistics for several test cases. The reported times correspond to the average per-frame duration of each stage of our approach. All experiments were conducted on an Intel(R) Core(TM) i7-8700 CPU @ 3.20 GHz (6 cores) with 32 GB of RAM and an NVIDIA RTX A2000 GPU.

4.2. Discussion

In Fig. 11 and the accompanying video, 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. With smaller values of τ_{3D} and τ_{UV}, our approach more often

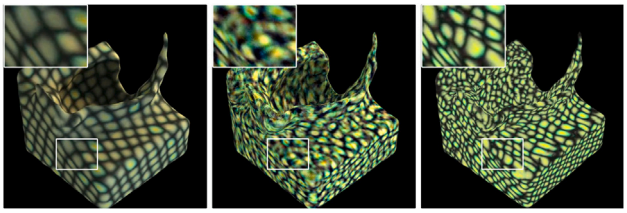

Fig. 8. Comparison of Kwatra et al. [28] and Gagnon et al. [31] vs. our approach.

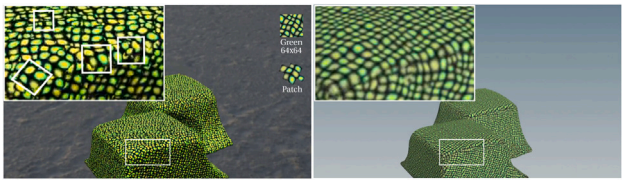

Fig. 9. Comparison of Gagnon et al. [30] vs. our approach.

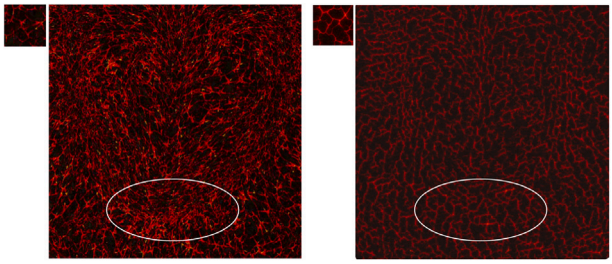

Fig. 10. Comparison of Gagnon et al. [32] vs. our approach.

Table 1
Average per-frame computation times (in seconds) for various stages of our approach.

Scenario	Pyr. Levels	Output Res.	Atlas Texels	Adv.	Stretch Det.	Synth	Pop. Red.	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

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.

Fig. 12 and the accompanying video illustrate the influence of the atlas resolution (512^2, 1024^2, and 2048^2) on the synthesized results. One needs to 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.

4.3. 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 Fig. 13 and the accompanying video, 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.

4.4. Limitations

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. The video and Fig. 14 show one exemplar for which the random-walk of Busto et al. [8] 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. [13], limiting repeated selection of the same texel, would likely mitigate this issue.

Our approach produces produces good results for a variety of textures, but is most effective for stationary textures. For instance, results on the keyboard and brick textures are acceptable (see accompanying video), 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.

5. Conclusion

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.

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 [34,35]. Moreover, instead of using separate texture atlases at every frame, it could be interesting to investigate the tracking of

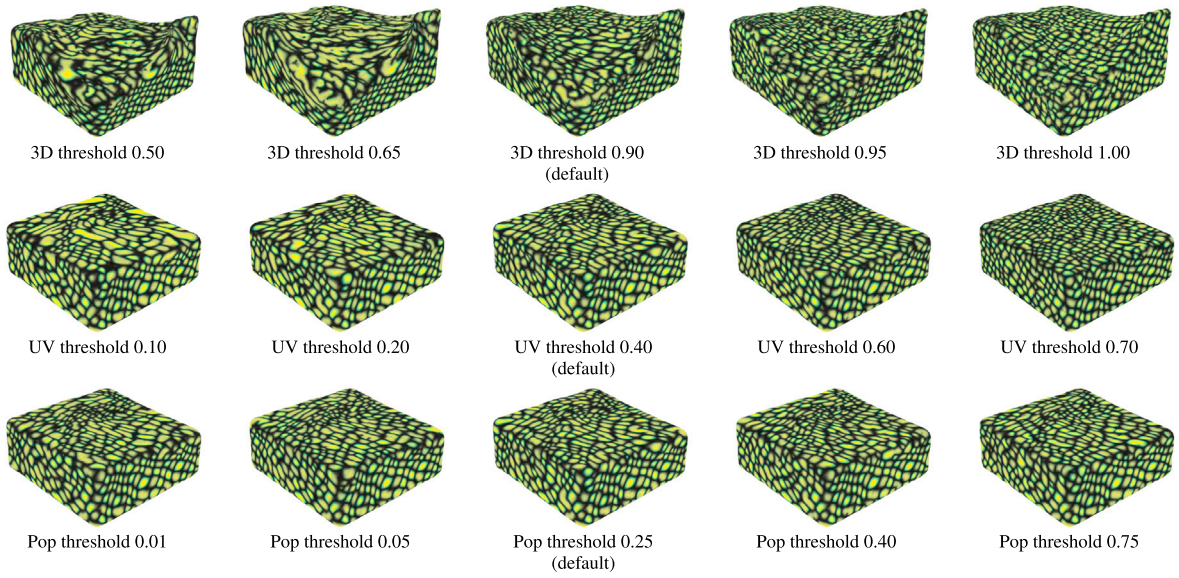

Fig. 11. Comparison of texture synthesis results using different threshold values for 3D stretching, UV stretching, and popping detection. The middle column (default) shows the value we selected as best for this simulation and texture.

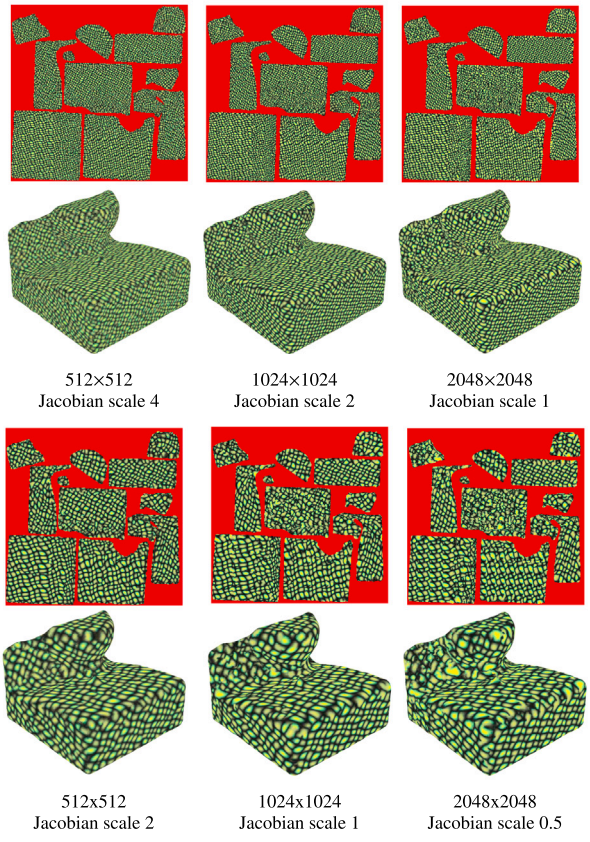

Fig. 12. Synthesis results using atlases at different resolutions. The first and third rows show the texture atlases, while the second and fourth rows show the rendered fluid surfaces.

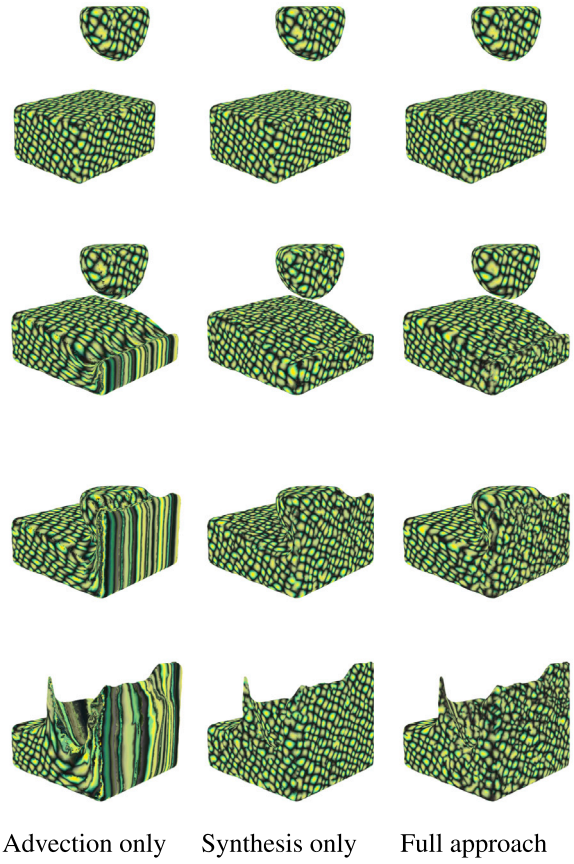

Fig. 13. Ablation study results showing texture synthesis on fluid simulation over time. Left: When only relying on advection we see significant stretching. Middle: Only synthesis variant, showing good texture details but suffering from temporal incoherence (see accompanying video). Right: Our adaptive 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.

an explicit mesh throughout the animation [36]. Also, 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

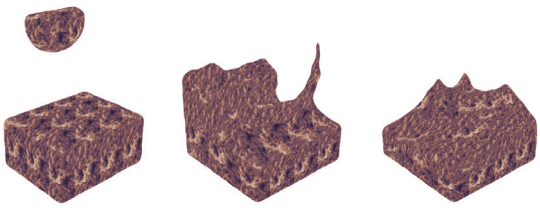

Fig. 14. Random-walk best match loses features from the input exemplar. Selected images from an animation.

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.

CRediT authorship contribution statement

Julián E. Guzmán: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **David Mould:** Writing – review & editing, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. **Eric Paquette:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Eric Paquette and Julián E. Guzmán report financial support was provided by NSERC CREATE VISION 584762-2024. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was funded by the NSERC CREATE VISION 584762-2024 program. The authors also acknowledge the use of Houdini software, courtesy of SideFX, which was used in the creation and rendering of the animations and simulations presented in this work. Lastly, we thank the anonymous reviewers for their thoughtful comments that helped improve the paper.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cag.2025.104490.

Data availability

Data will be made available on request.

References

[1] Wei L-Y, Lefebvre S, Kwatra V, Turk G. State of the art in example-based texture synthesis. In: Eurographics state of the art report. Eurographics Association; 2009, p. 93–117.
[2] Efros AA, Leung TK. Texture synthesis by non-parametric sampling. In: IEEE international conference on computer vision. Corfu, Greece; 1999, p. 1033–8.
[3] Wei L-Y, Levoy M. Fast texture synthesis using tree-structured vector quantization. In: Proc. of SIGGRAPH 00. Annual conference series, USA: ACM; 2000, p. 479–88.
[4] Wei L-Y, Levoy M. Texture synthesis over arbitrary manifold surfaces. In: Proc. of SIGGRAPH 01. Annual conference series, New York, NY, USA: ACM; 2001, p. 355–60.
[5] Lefebvre S, Hoppe H. Parallel controllable texture synthesis. ACM Trans Graph 2005;24(3):777–86.
[6] Lefebvre S, Hoppe H. Appearance-space texture synthesis. ACM Trans Graph 2006;25(3):541–8.
[7] Barnes C, Shechtman E, Finkelstein A, Goldman DB. PatchMatch: A randomized correspondence algorithm for structural image editing. ACM Trans Graph 2009;28(3):24.
[8] Busto PP, Eisenacher C, Lefebvre S, Stamminger M, et al. Instant texture synthesis by numbers. In: VMV. 2010, p. 81–5.
[9] Kwatra V, Essa I, Bobick A, Kwatra N. Texture optimization for example-based synthesis. ACM Trans Graph 2005;24:795–802.
[10] Darabi S, Shechtman E, Barnes C, Goldman DB, Sen P. Image melding: Combining inconsistent images using patch-based synthesis. ACM Trans Graph 2012;31(4).
[11] Kaspar A, Neubert B, Lischinski D, Pauly M, Kopf J. Self tuning texture optimization. Comput Graph Forum 2015;34(2):349–59.
[12] Liu X, Jiang L, Wong T-T, Fu C-W. Statistical invariance for texture synthesis. IEEE Trans Vis Comput Graphics 2012;18(11):1836–48.
[13] Jamriška O, Fišer J, Asente P, Lu J, Shechtman E, Sýkora D. LazyFluids: Appearance transfer for fluid animations. ACM Trans Graph 2015;34(4).
[14] Kopf J, Fu C-W, Cohen-Or D, Deussen O, Lischinski D, Wong T-T. Solid texture synthesis from 2D exemplars. ACM Trans Graph 2007;26(3).
[15] Dong Y, Lefebvre S, Tong X, Drettakis G. Lazy solid texture synthesis. In: Computer graphics forum, vol. 27, Wiley Online Library; 2008, p. 1165–74.
[16] Chen K, Johan H, Mueller-Wittig W. Simple and efficient example-based texture synthesis using tiling and deformation. In: Proceedings of the ACM SIGGRAPH symposium on interactive 3D graphics and games. 2013, p. 145–52.
[17] Takayama K, Okabe M, Ijiri T, Igarashi T. Lapped solid textures: filling a model with anisotropic textures. ACM Trans Graph 2008;27(3):1–9.
[18] Sendik O, Cohen-Or D. Deep correlations for texture synthesis. ACM Trans Graph 2017;36(5).
[19] Zhou Y, Zhu Z, Bai X, Lischinski D, Cohen-Or D, Huang H. Non-stationary texture synthesis by adversarial expansion. ACM Trans Graph 2018;37(4).
[20] Kovács ÁS, Hermosilla P, Raidou RG. Surface-aware mesh texture synthesis with pre-trained 2D CNNs. Comput Graph Forum 2024. http://dx.doi.org/10.1111/cgf.15016.
[21] Hu A, Desai N, Abu Alhaija H, Kim SW, Shugrina M. Diffusion texture painting. In: ACM SIGGRAPH 2024 conference papers. 2024, p. 1–12.
[22] Yu X, Yuan Z, Guo Y-C, Liu Y-T, Liu J, Li Y, Cao Y-P, Liang D, Qi X. TEXGen: a generative diffusion model for mesh textures. ACM Trans Graph 2024;43(6). http://dx.doi.org/10.1145/3687909.
[23] Perla SRK, Wang Y, Mahdavi-Amiri A, Zhang H. EASI-Tex: Edge-aware mesh texturing from single image. ACM Trans Graph 2024;43(4). http://dx.doi.org/10.1145/3658222.
[24] Mitchel TW, Esteves C, Makadia A. Single mesh diffusion models with field latents for texture generation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2024, p. 7953–63.
[25] Neyret F. Advected textures. In: Proc. of the ACM SIGGRAPH/Eurographics symposium on computer animation. Eurographics Association; 2003, p. 147–53.
[26] Yu Q, Neyret F, Bruneton E, Holzschuch N. Lagrangian texture advection: Preserving both spectrum and velocity field. IEEE Trans Vis Comput Graphics 2011;17(11):1612–23.
[27] Bargteil AW, Sin F, Michaels JE, Goktekin TG, O'Brien JF. A texture synthesis method for liquid animations. In: Proc. of ACM SIGGRAPH/Eurographics symposium on computer animation. Goslar, DEU: Eurographics Association; 2006, p. 345–51.
[28] Kwatra V, Adalsteinsson D, Kim T, Kwatra N, Carlson M, Lin M. Texturing fluids. IEEE Trans Vis Comput Graphics 2007;13(5):939–52.
[29] Narain R, Kwatra V, Lee H-P, Kim T, Carlson M, Lin MC. Feature-guided dynamic texture synthesis on continuous flows. In: Rendering techniques. Goslar, DEU: Eurographics; 2007, p. 361–70.
[30] Gagnon J, Dagenais F, Paquette E. Dynamic lapped texture for fluid simulations. Vis Comput 2016;32(6–8):901–9.
[31] Gagnon J, Guzmán JE, Vervondel V, Dagenais F, Mould D, Paquette E. Distribution update of deformable patches for texture synthesis on the free surface of fluids. Comput Graph Forum 2019;38(7):491–500.
[32] Gagnon J, Guzmán JE, Mould D, Paquette E. Patch erosion for deformable lapped textures on 3D fluids. Comput Graph Forum 2021;40(2):367–74.

[33] Sheffer A, Lévy B, Mogilnitsky M, Bogomyakov A. ABF++: Fast and robust angle based flattening. ACM Trans Graph 2005;24(2):311–30.
[34] Roy B, Poulin P, Paquette E. Neural upflow: A scene flow learning approach to increase the apparent resolution of particle-based liquids. In: Proc. ACM comput. graph. interact. tech., vol. 4, 2021, p. 40:1–26.
[35] Roy B, Paquette E, Poulin P. Particle upsampling as a flexible post-processing approach to increase details in animations of splashing liquids. Comput Graph 2020.
[36] Dagenais F, Gagnon J, Paquette E. Detail-preserving explicit mesh projection and topology matching for particle-based fluids. Comput Graph Forum 2017;36:444–57.