Motion Field Texture Synthesis

Chongyang Ma[*‡] Li-Yi Wei[†] Baining Guo[‡*] Kun Zhou[§]

[‡]Microsoft Research Asia [†]Microsoft Research [*]Tsinghua University [§]Zhejiang University

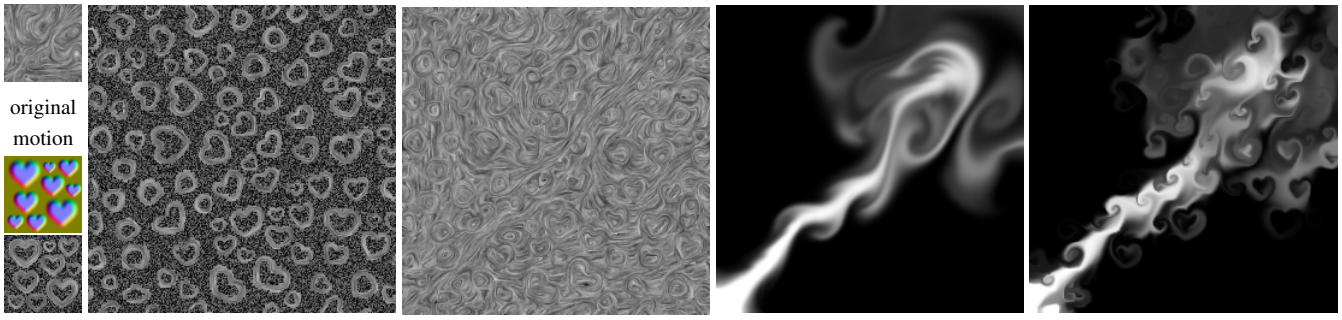


Figure 1: Motion field texture synthesis. Given a low-resolution motion field and an input exemplar, we synthesize a high-resolution detailed motion field that resembles the exemplar while follows the local orientation of the low-resolution field. This synthesized detail motion field is then combined with the original low-res motion field to produce the final motion. Our method can produce non-physics-based artistic effects such as fluids with heart-shaped motions.

Abstract

A variety of animation effects such as herds and fluids contain detailed motion fields characterized by repetitive structures. Such detailed motion fields are often visually important, but tedious to specify manually or expensive to simulate computationally. Due to the repetitive nature, some of these motion fields (e.g. turbulence in fluids) could be synthesized by procedural texturing, but procedural texturing is known for its limited generality.

We apply example-based texture synthesis for motion fields. Our technique is general and can take on a variety of user inputs, including captured data, manual art, and physical/procedural simulation. This data-driven approach enables artistic effects that are difficult to achieve via previous methods, such as heart shaped swirls in fluid animation. Due to the use of texture synthesis, our method is able to populate a large output field from a small input exemplar, imposing minimum user workload. Our algorithm also allows the synthesis of output motion fields not only with the same dimension as the input (e.g. 2D to 2D) but also of higher dimension, such as 3D volumetric outputs from 2D planar inputs. This cross-dimension capability supports a convenient usage scenario, i.e. the user could simply supply 2D images and our method produces a 3D motion field with similar characteristics. The motion fields produced by our method are generic, and could be combined with a variety of large-scale low-resolution motions that are easy to specify either manually or computationally but lack the repetitive structures to be characterized as textures. We apply our technique to a variety of animation phenomena, including smoke, liquid, and group motion.

Keywords: motion field, texture synthesis, fluids, group motion

1 Introduction

A variety of animation effects are characterized by a distinctive large scale motion plus repetitive small scale details, such as fluids with turbulence or herds with individual behaviors. Usually, a user would prefer to direct only the large scale motion, while avoiding tedious manual work by leaving the small scale details to be either roughly sketched or automatically generated.

One possible method to generate such small scale motions is physics simulation. This has the advantage of reality-based effects, but could be expensive to compute and difficult to control. Another possibility is procedural texturing (e.g. noise [Kim et al. 2008; Narain et al. 2008]). This is usually more computationally efficient than full fledged physics simulation, but procedural texturing is known for its limited generality, e.g. only applicable to specific phenomenon such as turbulence in fluids. Furthermore, both physical and procedural simulation might not offer the kind of artistic control that a user desires.

Our goal is to provide a method to generate detailed motion fields that is general, controllable, and easy to use. We attempt to achieve this by applying example-based texture synthesis to detailed motion fields. This method is general and allows us to take on a variety of user inputs as *texture exemplars*, include captured motion data, manual doodling, or physically/procedurally simulated animations. Due to the nature of texture synthesis, our method is able to populate a large output field from a small input exemplar, thus imposing minimum user workload. Our method also supports artistic effects that are difficult to achieve by physical/procedural simulation, such as heart-shaped swirls in fluids that could be achieved simply by providing a heart-shaped input exemplar to our algorithm.

Algorithm-wise, even though it does not require too much a leap of faith to apply texture synthesis to motion vector fields, we have to deal with the fact that existing texture synthesis algorithms are designed mainly for colors, which have potentially different perceptual implications from motion vectors. In addition, since we would like to provide a friendly 2D interface for specifying 3D motions, we need to figure out a way to synthesize 3D motion fields from 2D exemplars. Note that even though this has been achieved for color textures [Kopf et al. 2007; Dong et al. 2008], these meth-

ods are not directly applicable to our scenario as colors are invariant with respect to different 2D views whereas motion vectors are subject to projection (where the 2D views could only capture the projected vector components; see Figure 3) and coordinate transformation (where the output vector field has to be properly oriented not only during, but also after, synthesis; see Figure 2). One of our core technical contributions is to incorporate these projection and coordinate-transformation effects into the synthesis process for motion fields.

The texture motion fields produced by our technique are generic, and could be combined with a variety of large scale motions obtained from other sources, such as fluid simulation, group movements, or manual drawings. Such large scale motions are often distinctive and lack repetitive texture structures so that texture synthesis is not applicable, but on the other hand they are usually coarse enough to be easily specified either manually or computationally. We demonstrate the applicability of our method to a variety of phenomenon, such as smoke, liquid, and group motions.

2 Related Work

Animation A variety of animation effects contain a distinctive large scale motion plus a detailed motion field characterized by stochastic and/or repetitive patterns; some examples include individual behaviors in a large crowd/herd [Reynolds 1987; Treuille et al. 2006; van den Berg et al. 2008] and vortices [Fedkiw et al. 2001], bubbles [Hong et al. 2008], or turbulences [Kim et al. 2008; Narain et al. 2008] in fluids. It is usually desirable to intentionally direct only the large scale motion while leaving the detailed behaviors to certain automatic computation. Even though the individual phenomenon can be simulated with high fidelity by various methods as mentioned above, it might be desirable to have a more general method that is applicable to a wider variety of effects. Furthermore, the simulation based approaches could be difficult to control, and offers limited artistic choices. Our method aims to augment these prior methods via example-based texture synthesis. In particular, we aim at automatic synthesis of motion details, and can be considered as complementary to prior techniques for controlling large scale fluid motions such as [Fattal and Lischinski 2004; McNamara et al. 2004].

Texture Synthesis Texture synthesis is originated in the study of color images but it has also been applied to other data categories as well, such as geometry, videos, and motions; here, we focus on the application of texture synthesis to motions and refer the readers to [Wei et al. 2009] for a more comprehensive survey. For articulated motion signals containing repetitive patterns such as walking or running, the signals could be treated as one dimensional textures [Pullen and Bregler 2002; Li et al. 2002]. Another possibility is to treat the aggregate crowd positions as a texture [Kyriakou and Chrysanthou 2008]. Videos containing motion textures such as waterfalls could also be re-edited based on a target flow [Bhat et al. 2004]. Texture synthesis has also been applied to animate static images [Chuang et al. 2005; Okabe et al. 2009]. Our method is also related to static vector field generation [Liu et al. 2004; Palacios and Zhang 2007; Fisher et al. 2007; Wang et al. 2009], but we need to deal with motions. Beyond motions and videos, it is also possible to apply color textures over dynamic fluid surfaces to either enhance rendering or further simulate small scale effects such as bubbles [Bargteil et al. 2006; Kwatra et al. 2007; Narain et al. 2007]. A common trait among these techniques is that texture synthesis is often computationally cheaper than simulation, and offers more flexible control via the choice of input exemplars. However, to our knowledge, example-based texture synthesis has yet to be applied to general motion fields; even though there exists procedural texturing approaches to augment fluid details [Kim et al. 2008; Narain et al. 2008; Schechter and Bridson 2008], it is a well known fact that procedural texturing is less general than example-based synthesis. We draw inspirations from these methods, but take a step further by applying texture exemplars for motion fields.

3 Our Method

Here, we first describe our basic methodology, followed by details and variations. Our method consists of the following main steps :

1. Use some prior methods (e.g. manual, procedural, or physics based) to produce a large-scale, low-resolution motion vector field for the specific application (e.g. fluids or groups).

2. The user specifies texture exemplars to indicate the desired detail motion. The exemplars could come from a variety of sources, including captured data, manual art, or simulation.

3. Our system deploys texture synthesis to automatically propagate the characteristics of the input exemplars to the entire output domain. The input and output could be of the same dimensionality (e.g. 2D to 2D) or of different dimensionality (e.g. 2D input to 3D output).

4. We combine the synthesized high resolution details with the given low resolution motion field.

5. We optionally perform application specific processes such as incompressibility for fluids or boundary conditions. The former could be done as a post-process after Step 4 while the latter via constrained synthesis during Step 3.

Below we describe algorithm details. For easy reference, we summarize our algorithm in Pseudocode 1.

3.1 Motion detail synthesis - same dimension

When the input exemplar and the output motion field are of the same dimension, synthesizing motion vectors is analogous to synthesizing multi-channel texture colors. For this, our method basically follows [Kwatra et al. 2005], which performs texture synthesis by minimizing the following energy function

$$E_t(\mathbf{x}; \{\mathbf{z}_p\}) \quad = \quad \sum_{p \in X^\dagger} |\mathbf{x}_p - \mathbf{z}_p|^2 + O(\mathbf{x}) \qquad (1)$$

where E_t measures local neighborhood similarity across a subset X of the output $\mathbf{x}$, and $\mathbf{z}_p$ indicates the most similar input neighborhood to each output neighborhood $\mathbf{x}_p$. [Kwatra et al. 2005] solved this energy function via an iterative process alternating between neighborhood search and pixel assignment steps. The method could also be extended for other applications by adding extra energy terms $O(\mathbf{x})$, such as frame coherence for animation. As denoted in Pseudocode 1, our method follows a similar iterative search/assignment process. For same-dimension synthesis, the main differences are: (1) the need for coordinate transformation and (2) the perceptual differences between colors and motions.

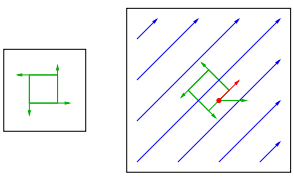


Figure 2: Coordinate transformation. The exemplar (left) is a swirl pattern. The output (right) is conditioned by a large scale diagonal pattern. The correct value at the red point should be the red arrow instead of the green one.

function $\mathbf{F} \leftarrow$ MotionFieldTextureSynthesis($\{\mathbf{I}_i\}_{i=1:m}$, $\mathbf{L}$, ω)
  // $\mathbf{F}$: *final output motion field*
  // $\{\mathbf{I}_i\}_{i=1:m}$: *exemplars across* m *views*
  // $\mathbf{L}$: *input large-scale coarse motion field*
  // $\mathbf{H}$: *synthesized detail motion field*
  $\mathbf{H} \leftarrow$ DetailFieldSynthesis($\{\mathbf{I}_i\}_{i=1:m}$, $\mathbf{L}$) // *Section 3.1 & 3.2*
  $\mathbf{H}' \leftarrow$ CoordinateTransform($\mathbf{H}$, $\mathbf{L}$) // *Section 3.1*
  // ω: *relative weighting for* $\mathbf{L}$ *and* $\mathbf{H}$
  $\mathbf{F} \leftarrow$ Combine($\mathbf{L}$, $\mathbf{H}'$, ω) // *Section 3.5*
  return $\mathbf{F}$

function $\mathbf{H} \leftarrow$ DetailFieldSynthesis($\{\mathbf{I}_i\}_{i=1:m}$, $\mathbf{L}$)
  $\mathbf{H} \leftarrow$ Initialize($\{\mathbf{I}_i\}_{i=1:m}$, $\mathbf{L}$) // *Section 3.4*
  iterate until convergence or enough # of iterations
    // *search phase, i.e. the "M-step" in [Kwatra et al. 2005]*
    foreach sample $s \in \mathbf{H}$
      // $\mu(s,i)$: *best match neighborhood for* s *with input* i
      $\{\mu(s,i)\}_{i=1:m} \leftarrow$ Search(s, $\{\mathbf{I}_i\}$, $\mathbf{L}$, $\mathbf{H}$)
    end
    // *assignment phase, i.e. the "E-step" in [Kwatra et al. 2005]*
    foreach sample $s \in \mathbf{H}$
      $s \leftarrow$ Assign($\{\mu(s,i)\}_{i=1:m}$)
    end
  end
  return $\mathbf{H}$

function $\mathbf{H} \leftarrow$ Initialize($\{\mathbf{I}_i\}_{i=1:m}$, $\mathbf{L}$)
  if first frame of animation
    $\mathbf{H} \leftarrow$ random pixels from $\{\mathbf{I}_i\}$
  else
    $\mathbf{H} \leftarrow$ advection from last frame [Kwatra et al. 2005]
  end
  return $\mathbf{H}$

function $\{\mu(s,i)\}_{i=1:m} \leftarrow$ Search(s, $\{\mathbf{I}_i\}_{i=1:m}$, $\mathbf{L}$, $\mathbf{H}$)
  foreach input i
    $\mathrm{N}(s,i) \leftarrow$ neighborhood sampled around s considering both orientation of $\mathbf{I}_i$ and local frame defined by $\mathbf{L}(s)$
    $\mathrm{N}(s,i) \leftarrow$ Project($\mathrm{N}(s,i)$, i) // *Section 3.2*
    $\mu(s,i) \leftarrow$ BestMatch($\mathbf{I}_i$, $\mathrm{N}(s,i)$)
  end
  return $\{\mu(s,i)\}$

function $\mu(s,i) \leftarrow$ BestMatch($\mathbf{I}_i$, $\mathrm{N}(s,i)$)
  find most similar neighborhood in $\mathbf{I}_i$ with $\mathrm{N}(s,i)$
  via tree search [Kwatra et al. 2005; Kopf et al. 2007]

function $s \leftarrow$ Assign($\{\mu(s,i)\}_{i=1:m}$)
  $s \leftarrow$ average the centers of $\{\mu(s,i)\}$

function $\mathbf{F} \leftarrow$ Combine($\mathbf{L}$, $\mathbf{H}$, ω)
  $\mathbf{F} \leftarrow$ Upsample($\mathbf{L}$) $+ \mathbf{H} \times \omega$ // *upsample* $\mathbf{L}$ *to have same size as* $\mathbf{H}$
  return $\mathbf{F}$

Pseudocode 1: Pseudo-code for our algorithm.

Given an input exemplar and a large-scale coarse-resolution vector field (Figure 2), our goal is to synthesize a detailed output motion field so that the local patterns (1) properly orient with respect to the large scale field and (2) resemble the input exemplar in the texture synthesis sense. If the input exemplar is a color texture instead of a vector field, these goals can be achieved by properly orienting the output texture neighborhoods during the search process, as described in prior algorithms such as [Turk 2001; Wei and Levoy 2001]. However, for vector fields, simply orienting the output neighborhoods is not enough, as the synthesized vector field will still retain the value from the original input coordinate frame, as indicated by the green arrow at the red point in Figure 2. The correct value, as indicated by the red arrow, is obtained by transforming the directly-synthesized green arrow by the local coordinate system (the blue vector field). Note that this coordinate transformation of the synthesized value should be performed as a post process, as we need to use the un-transformed values during texture synthesis to match the (un-transformed) values in the input exemplar.

Aside from this algorithm-wise coordinate transformation issue, motion vectors also differ from colors in that they have different perceptual implications. In particular, it has been found for colors, local coherence is very important and occasional high frequency discontinuities are more tolerable than a texture that is entirely continuous but subject to blur or noise; see discussions in coherence synthesis [Ashikhmin 2001; Tong et al. 2002; Han et al. 2006] and patch-based synthesis [Liang et al. 2001; Efros and Freeman 2001]. However, for motion vectors, we have found the opposite holds true: low frequency noise or blur is usually better than high frequency discontinuity. Thus, in our current implementation, we prefer the original least-squares based texture optimization algorithm [Kwatra et al. 2005] rather than the k-coherence enhanced version [Han et al. 2006] for quality reasons. However, for speed reasons, we have also developed a version of our algorithm based on k-coherence. This will be discussed in Section 3.7.

3.2 Motion detail synthesis - cross dimension

In Section 3.1 we have described how to perform same-dimension synthesis. Here we extend the idea further for cross-dimension synthesis [Kopf et al. 2007; Dong et al. 2008]. This could be useful for a variety of scenarios, such as synthesizing 3D output motions from 2D inputs. Just like same-dimension synthesis, cross-dimension synthesis for motion vectors is very similar to colors. The main difference, in addition to those mentioned for same-dimension synthesis, is the need to handle projection. Details are as follows.

The basic idea behind previous 2D-to-3D synthesis algorithms [Kopf et al. 2007; Dong et al. 2008] is to match up neighborhoods centered around each 3D output voxel with neighborhoods from several 2D inputs with different orientations. Specifically, the value of each output voxel s is determined by a two phase process as follows. In the *search* phase, m neighborhoods $\{\mathrm{N}(s,i)\}_{i=1:m}$ centered at s are built with orientations matching each one of the m input exemplars $\{\mathbf{I}_i\}_{i=1:m}$. (A classical scenario is when three input views are perpendicular to the three coordinate axes of the output volume.) A best match neighborhood $\mu(s,i)$ for $\mathrm{N}(s,i)$ is then found from $\mathbf{I}_i$ for each input i. In the *assignment* phase, the centers of the matches $\{\mu(s,i)\}_{i=1:m}$ are then combined together to yield the final value for the output voxel s. (This combination could be achieved either by weighted least squares [Kopf et al. 2007] or k-coherence [Dong et al. 2008].) These two phases are iterated enough times until convergence or a maximum number of iterations is reached.

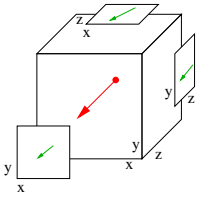


Figure 3: Vector projection. Each one of the three 2D input views specifies only the corresponding projected components of an output 3D field. In this example, the three views are aligned with the three coordinate axes, and thus the projection reduces to dropping one of the 3D vector components.

We follow a similar two phase process for motion vector synthesis, but unlike colors which remain invariant with respect to different views, motion vectors are subject to projections. For example, as illustrated in Figure 3, each one of the 2D views specifies only the corresponding projected components of the 3D motion field. This projection issue affects both the search and assignment phases of

our algorithm as follows. In the search phase, after building the output neighborhoods $\{\mathrm{N}(s,i)\}$, we project the vector components with respect to each one of the input views i before conducing the best match. For example, if $\mathrm{N}(s,1)$ corresponds to the view perpendicular to the x-axis in Figure 3, we then simply zero out the x components in $\mathrm{N}(s,1)$ and use only the y/z components for BestMatch(). In the assignment phase, each input match $\mu(s,i)$ contributes only to its vector components to the output voxel s. For example, if a direct average is to be performed from the three orthogonal views in Figure 3, then we have

$$\begin{aligned} s_x &= \frac{\mu(s,2)_x + \mu(s,3)_x}{2} \\ s_y &= \frac{\mu(s,3)_y + \mu(s,1)_y}{2} \\ s_z &= \frac{\mu(s,1)_z + \mu(s,2)_z}{2} \end{aligned} \tag{2}$$

In our implementation we perform a simple averaging for the assignment phase instead of the fancier weighted average + histogram match in [Kopf et al. 2007], as we have found the latter unnecessary. This could be another example of perceptual difference between colors and motion vectors.

3.3 Neighborhood orientation

As described in Section 3.1, to produce natural looking results the synthesized motion field $\mathbf{H}$ should be properly oriented with respect to the given coarse motion field $\mathbf{L}$. This can be achieved by collecting output neighborhoods with respect to properly defined local frames during texture synthesis. In 2D, the local frame is completely defined by $\mathbf{L}$ as we could have the x-axis follows $\mathbf{L}$ and y-axis rotated 90 degrees from x-axis. However, in 3D, $\mathbf{L}$ only specifies one of the three coordinate axes and the orientation of the other two are under-constrained. In general, we have found that the exact specifications of the additional constraint not very crucial, as long as the resulting local frames are coherent both spatially and temporally. In our experiments, we either define the additional constraint through specific application properties (e.g. gradient from the density for smoke rendering) or specify it manually at sparse locations followed by interpolation (similar to the specification of surface vector fields in [Turk 2001]).

3.4 Initialization

For the first frame of the detailed motion field $\mathbf{H}$ in the entire animation sequence, we initialize it by randomly copying pixels from the input exemplars $\{\mathbf{I}_i\}$. If the output $\mathbf{H}$ is of the same dimension as the exemplars (e.g. 2D to 2D), then there could be only one exemplar and the random copy could be done directly. If $\mathbf{H}$ is in 3D, we randomly take neighborhoods from the different input views $\{\mathbf{I}_i\}$ and blend them together via the *assign* phase algorithm as described in Section 3.2 to take into account projection.

For subsequent frames of the animation, we initialize the detailed motion field $\mathbf{H}$ by advecting $\mathbf{H}$ from the previous frame via the low-resolution motion field $\mathbf{L}$ as described in [Kwatra et al. 2005].

3.5 Final combination

After the detailed motion field $\mathbf{H}$ is synthesized, we could combine it with the given low resolution motion $\mathbf{L}$ to produce the final result. As shown in Pseudocode 1, we perform this combination via a user specifiable weighting ω, which could be either a global constant (essentially allowing the user to tune the amount of detail motion) or a spatially varying function with the same dimension as $\mathbf{H}$, such as setting ω proportional to the kinetic energy [Kim et al. 2008] or vorticity [Fedkiw et al. 2001] of $\mathbf{L}$.

3.6 Application specific process

Since the detailed motion field $\mathbf{H}$ is produced by texture synthesis instead of physics simulation, the final combined motion $\mathbf{F}$ might not satisfy the relevant physics properties of the target application scenario, such as incompressibility and boundary conditions for fluid motions. To handle these situations, we perform an optional application-specific process.

Incompressibility When the input exemplars $\{\mathbf{I}_i\}$ and coarse motion field $\mathbf{L}$ are both incompressible, we have found that the synthesized detail motion field $\mathbf{H}$ *visually* incompressible, even though it may not be really so. (Intuitively, texture synthesis, by matching spatial neighborhoods, does not tend to alter the divergence of the motion fields too much.) If strict incompressibility is desired, one could perform Helmholtz-Hodge decomposition [Tong et al. 2003] on the final combined $\mathbf{F}$ before rendering. We have not done so in our current implementation as strict incompressibility is not our main goal.

Boundary condition To observe the boundary condition, we need to ensure that the relevant components of $\mathbf{F}$ match that of the shared boundaries [Bridson and Müller-Fischer 2007]. We achieve this by performing constrained texture synthesis during *DetailFieldSynthesis* in Pseudocode 1. Constrained texture synthesis has been applied to color images for hole filling and object replacement [Fidaner 2008]. The basic idea is to keep the *constrained* portions of the image fixed (e.g. boundary of a hole) while trying to synthesize textures over the target region (e.g. a hole or an object to be replaced) so that the newly synthesized portion not only resembles the input exemplar but also remains consistent with the constrained regions.

We can apply a similar idea here for motion fields, with the main difference that unlike colors where the entire pixel values serve as constraints, boundary conditions for motion fields often only involve a specific vector component (e.g. normal to the boundary). This is related to the issue of projection in Section 3.2, and could be easily addressed by matching only the proper sub vector components of the boundaries during the synthesis process. Specifically, we add the following constraint energy term to the basic texture similarity term E_t in Equation 1:

$$E_n(\mathbf{x}) = \sum_{p \in X} \lambda_p |\mathbf{x}_p^n - \mathbf{b}_p|^2 \tag{3}$$

where $\mathbf{x}$ indicates the output motion field, $\mathbf{x}_p^n$ the sub-vector component at sample p corresponding to the boundary direction (e.g. normal to a wall), $\mathbf{b}$ the specified boundary condition (e.g. 0 velocity normal to a wall), and λ a Gaussian weighting function that peaks on and attenuates away from boundaries. This equation essentially enforces boundary conditions via a soft constraint as an additional energy term over Equation 1, and we opt this over a hard constraint for better synthesis quality and the need to deal with multi-resolution synthesis (where hard boundary constraints could not really apply at lower resolutions). In addition, since E_n formulates a quadratic energy term, the combined energy function $E_n + E_t$ can be solved via the search/assignment iterative process as depicted in Pseudocode 1. In our current implementation, we let λ be wider at lower resolutions for better quality, but shaper at higher resolutions for better boundary condition enforcement.

Discussion To synthesize motion fields that are known to be incompressible (e.g. fluids), one might conjecture that it would be better to convert the input exemplars $\{\mathbf{I}_i\}$ into potential fields $\{\Psi_i\}$, perform texture synthesis on $\{\Psi_i\}$ to produce an output Ψ_h, and derive the synthesized motion field as $\mathbf{H} = \nabla\times\Psi_h$. Doing so could have several benefits compared to synthesizing motion fields directly. Since Ψ is a scalar-valued function, it would be cheaper to synthesize than vector-valued motion fields. It is also easier to enforce both incompressibility and boundary conditions. However, even though this is a possibility for 2D-to-2D synthesis, we have encountered some difficulties for 2D-to-3D synthesis. Specifically, for 2D-to-2D synthesis, both the input Ψ_i and Ψ_h are scalar-valued 2D functions, and thus Ψ_h could be directly texture-synthesized from Ψ_i. However, for 2D-to-3D synthesis, the situation is more complex. First of all, even though $\{\Psi_i\}$ are all scalar-valued functions for 2D inputs, the output Ψ_h is a 3-vector. Thus, somehow we have to find a way to expand the number of channels from 1 on the input side to 3 on the output side. Second, even though one might come up with such a magic expansion by taking advantage the multiple input views $\{\mathbf{I}_i\}$, this is not guaranteed as the number of input exemplars might not necessarily be 3. Even if so, we have this cross-coupling issue of the curl operator where $\nabla\times\Psi = (\frac{\partial\Psi_3}{\partial y} - \frac{\partial\Psi_2}{\partial z}, \frac{\partial\Psi_1}{\partial z} - \frac{\partial\Psi_3}{\partial x}, \frac{\partial\Psi_2}{\partial x} - \frac{\partial\Psi_1}{\partial y})$ and thus each channel of $\nabla\times\Psi$ is coupled to two of the inputs, making the synthesis of Ψ_h from $\{\Psi_i\}$ a quite complex issue. All these are further compounded by the projection issue we discussed in Section 3.2. Due to the complexity for performing 2D-to-3D synthesis via potential fields, we opt for the direct synthesis of motion fields.

3.7 Acceleration

The main performance bottleneck of our algorithm described so far lies in the neighborhood search part, as indicated by BestMatch() in Pseudocode 1. This search time is proportional to the input exemplars size. Even though tree search is doable for small exemplars, the computation could become prohibitive for large ones. A classical solution to this problem is via k-coherence [Tong et al. 2002] which provides constant search time per output sample. The k-coherence idea has been employed in the 2D acceleration of [Kwatra et al. 2005] by [Han et al. 2006] as well as the 3D acceleration of [Kopf et al. 2007] by [Dong et al. 2008].

In principle, we could accelerate our algorithm by using [Han et al. 2006] for 2D synthesis and [Dong et al. 2008] for 3D synthesis (see Pseudocode 2), after taking care of the issues of coordinate transformation (for both 2D and 3D) and projection (for 3D) as mentioned earlier. Specifically, our accelerated algorithm in 3D runs just like [Dong et al. 2008] and stores for each output voxel the indices of the best matches from the input exemplars. During the *search* phase, we use k-coherence instead of tree search to find the best match neighborhoods. During the *assignment* phase, we take the candidate that is closest to the average of the centers of the best matches (essentially minimizing the neighborhood difference energy function [Han et al. 2006]), while still keeping the list of indices of the best matches from the inputs. These differences from our basic algorithm are summarized in Pseudocode 2.

```
function μ(s,i) ← BestMatch(I_i, N(s,i))
    find most similar neighborhood in I_i with N(s,i)
    via k-coherence search
    // [Han et al. 2006] for 2D or [Dong et al. 2008] for 3D

function s ← Assign({μ(s,i)}_{i=1:m})
    s ← k-coherence candidate most similar to
          the average of the centers of {μ(s,i)}
          // [Han et al. 2006] for 2D or [Dong et al. 2008] for 3D
```

Pseudocode 2: Pseudo-code for our accelerated algorithm. Here we show only the differences from our unaccelerated algorithm in Pseudocode 1.

4 Results

Our approach can be flexibly applied to a variety of combinations, including different input exemplars, different dimensions, different low resolution fields, as well as different physical phenomenon.

Different exemplars Similar to prior example-based texture synthesis techniques, our method could be applied to input exemplars with different texture patterns and formats. We show some results in Figure 1 and Figure 4 using smoke as the rendering medium. Since our technique is entirely data driven, we are able to produce results that are difficult to achieve via prior procedural or simulation-based techniques, such as smoke animations with square or lightning shaped motions. In fact, even the circular swirls might be achievable via physics simulation, our technique is still better at enforcing the circular shapes and thus better at producing a cartoonish effect.

In Figure 1 and Figure 4, we use primarily input exemplars that are generated by simple procedures (e.g. circles or squares) or bitmap images (from which we take curl on the gray scale magnitude to obtain the motion field), but other formats are certainly possible such as manual drawings or simulation results. In general, our method could handle input formats that could be represented or converted as vector fields.

One major difference between synthesizing color and motion textures we have observed is that to obtain visually salient effects, it is usually preferable to use simple inputs. This is kind of contradictory to color texture synthesis as the visual richness often relies on sufficiently complex inputs. However, for motions, we have found that an overly complex input might produce less intuitive results. This is especially true if the synthesized detail motion $\mathbf{H}$ is combined with a complex low resolution field $\mathbf{L}$. Among our examples shown in Figure 1 and 4, even though hearts, circles, and squares might be too simple to produce interesting color synthesis results, we have found them produce no less convincing results than more complex inputs (e.g. the 161 and 295 textures shown in Figure 4).

Different dimensions So far, we have shown results primarily in 2D because it is easier to visualize the effects. Figure 5 and 6 demonstrate the cross dimension capability of our algorithm. There, the user provides 2D inputs and our system automatically produces 3D outputs. This is a pretty handy application scenario as it could be difficult for the user to acquire or specify 3D inputs. For less structured textures, we have found it possible to produce reasonable results from multiple views. However, for more structured textures, it could be very difficult or downright impossible to produce similarly structured 3D outputs [Kopf et al. 2007; Dong et al. 2008]. For this latter case, we have found it beneficial to weigh the input views differently during synthesis so that one of them has a predominating effect to allow the formation of better output. The choice of the favored view would be application dependent, e.g. the one facing the viewer for static/pre-scripted view points or the one facing outward of a density field via the method we described in paragraph **Neighborhood Orientation** of Section 3.2. We use the latter method to produce the smoke results in Figure 5. Our method also allows the specification of different images for different views, such as swirls with square cross sections shown in Figure 6. We achieve this by providing square-shaped swirls for the top view and stripe patterns for the side views.

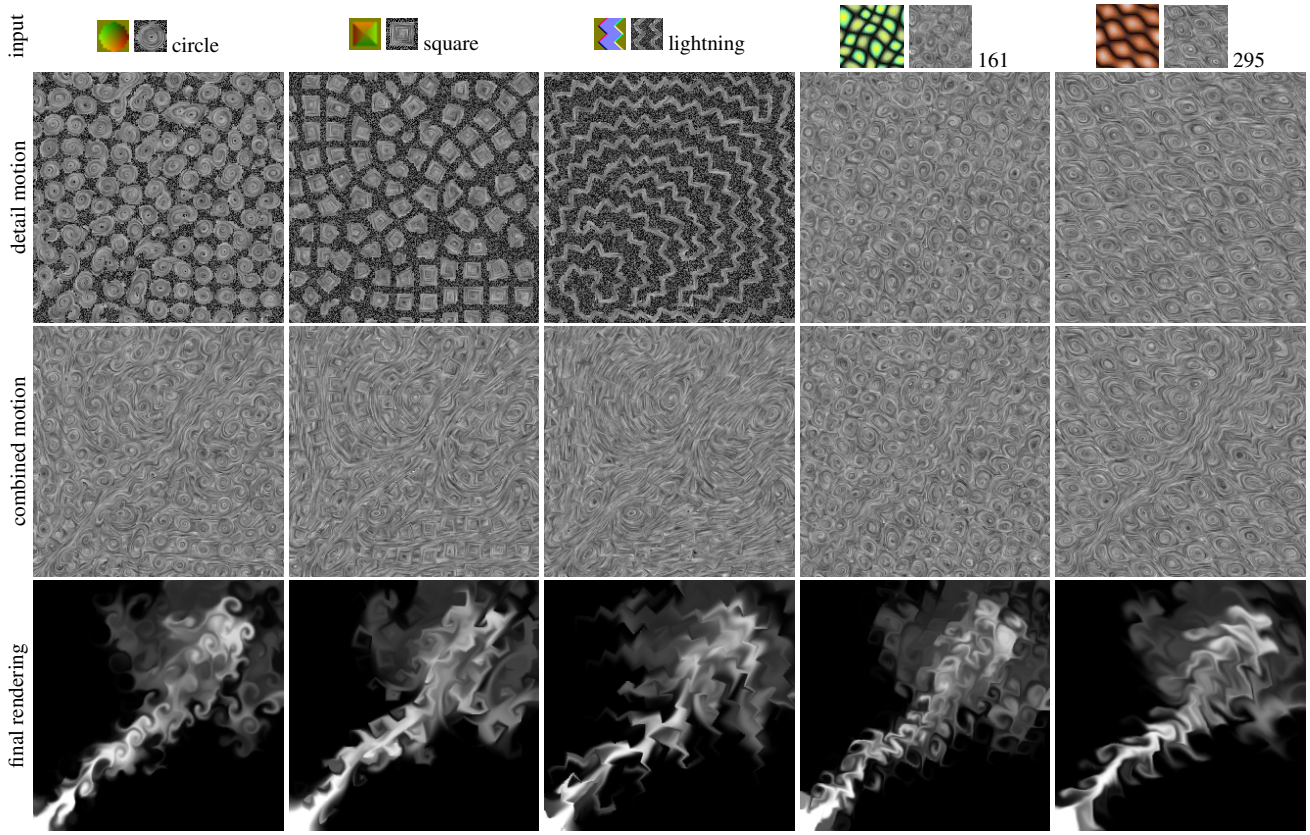


Figure 4: 2D synthesis results. Here, we use the original low resolution motion as in Figure 1, but show different inputs and the corresponding results. The low resolution motion field is produced via simulation [Fedkiw et al. 2001]. We visualize the motion fields via LIC [Cabral and Leedom 1993], and leave zero vectors as non-integrated white noises (with reduced amplitude). The circle and square inputs are generated by simple procedures, the lightning a simple bitmap, and the 161 and 295 more complex ones. For the input exemplars, in addition to the motion field visualized by LIC, we also show the color visualization for procedurals (circle and square) or the original bitmap image (lightning, 161, and 295) for clarity.

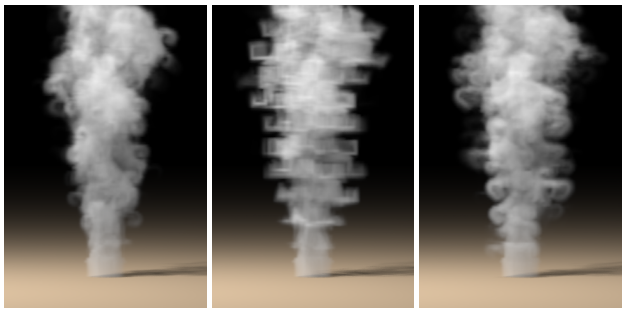


Figure 5: 3D synthesis results. The original low-res motion is shown on the left. The input textures are shown in Figure 1 & 4. The data sizes are $24^2 \times 32$ for the original and $96^2 \times 128$ for our results.

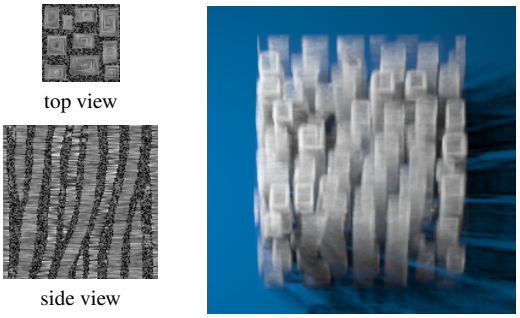


Figure 6: 3D synthesis result with different 2D views. To achieve this square-shaped swirls effect, we need different input exemplars for different views. The data sizes are: 64^2 for top view, 128^2 for side view, and 128^3 for the output motion.

Application specific process Our method also works for a variety of large-scale coarse-resolution vector fields. In Figure 4 and 5 we use physics simulation, and in Figure 7 we use the low resolution motion provided in the sample code from [Bridson et al. 2007]. We also utilize the incorporated scene (of river passing through a round stone) to enforce boundary conditions for our synthesized motion. Our approach is also applicable to more complex boundary conditions as shown in Figure 8.

Different phenomenon In addition to fluids, our method can also be applied to other phenomenon, such as group motions. In Figure 9, we demonstrate an example of under-water seaweed motions. There, we specify a 2D sinusoidal motion for both the top and side views, synthesize a 3D motion field, and use that to direct the movements of points on the seaweed polygonal models. As shown, even with such simple input exemplars, our method is able to produce natural-looking output motions. Note that even though this motion could also be achieved by pure procedural simulation, the user would have to inject a certain amount of randomness to avoid

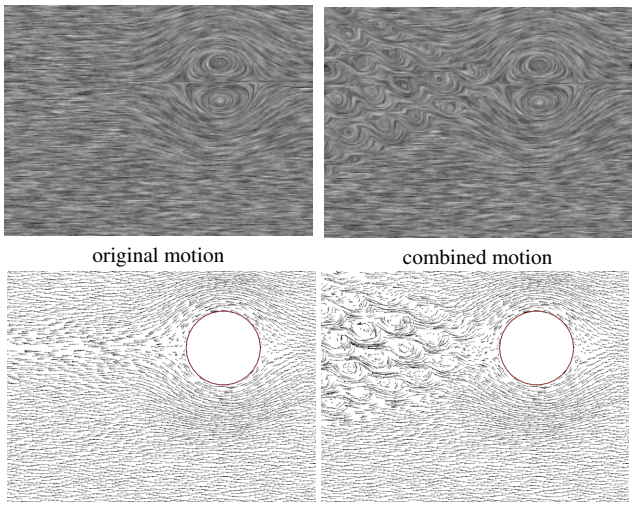


Figure 7: Boundary condition. We use the example scene from [Bridson et al. 2007] to enforce boundary condition of our result. The image sizes are: 64^2 for the exemplar (295 in Figure 4), 200×150 for the motion fields, and 640×480 for the renderings.

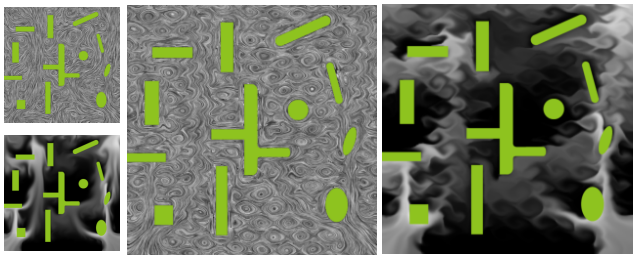
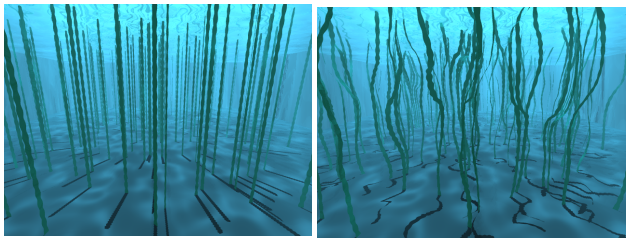


Figure 8: More complex boundary condition. Image sizes are 64^2 for the original and 256^2 for our result. The input texture is shown in Figure 7.

the motion being monotonic. Our technique, in contrast, does not require so; even if the input exemplar is purely regular, the nature of texture synthesis would induce a certain amount of randomness into the motions.

Parameters For the results shown in the paper, we use multi-resolution texture synthesis with 3 pyramid levels, except when the input size $\leq 16^2$ for which we use only 2 levels. Within each pyramid level, we perform 3 to 5 iterations with neighborhood size 17^2 followed by another 3 to 5 iterations with neighborhood size 9^2 for 2D synthesis, and fix the neighborhood size to 9^2 for 3D synthesis. The user is also free to choose the texture scale (e.g. the relative size of individual hearts on the output in Figure 1) and blending weight ω depending on the particular inputs and desired effects. In our experience, picking the texture scale is usually very intuitive, but ω might require some trials and errors. But tuning ω is usually quite easily as it does not involve any re-synthesis.

Performance Using a single thread running on a CPU, our current implementation takes about one minute to produce a 256^2 2D result and about one hour to produce a $256^2 \times 64$ 3D result. Our GPU implementation improves the corresponding timing to 3-to-5 seconds for 2D and 4-to-8 minutes for 3D, as measured on an NVIDIA GeForce 8800 GTX GPU. Our focus on this paper is not about performance, so we have not performed much speed optimization. We leave this as a potential future work.



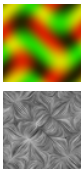


Figure 9: Group motion. We animate a group of seaweeds via our technique. For clarity, we visualize the exemplar by both LIC and RG color image encoding the xy motion components. Notice the lively motion produced by our method, even though the exemplars are produced by simple procedures. The input exemplar has size 64^2 and is applied for all three orthogonal views. The output field has size $256^2 \times 64$ and is advected by a constant horizontal flow.

5 Limitations and Future Work

There are several **limitations** of our approach:

- Based on texture synthesis, our technique is applicable only to detailed texture-like motions, not global coarse scale motion, which should be produced by prior methods, e.g. simulation, procedurals, or manual drawings. Our technique should be deployed only as a complement to these prior techniques for adding texture details.
- Similar to prior example-based texture synthesis algorithms [Wei et al. 2009] and especially 3D synthesis from 2D views [Kopf et al. 2007; Dong et al. 2008], our method will not work well for all input exemplars.
- Even for inputs that can be well generated by texture synthesis, the result might still not look good if the application needs to enforce conditions that are incompatible with the input. For example, if the input motion is very compressible (e.g. contains sources/sinks) whereas the application requires an incompressible output, our final result might lose resemblance to the input.
- In our current implementation, the synthesized detail motion is simply combined with the low resolution motion without considering any physical feedback or cross-coupling, even though we believe it is doable [Narain et al. 2008].

We envision several possible directions for **future work**:

- Given the recent advances in fast fluid simulation and rendering [Yuksel et al. 2007; Long and Reinhard 2009; van der Laan et al. 2009], it would be useful to add texture details via our technique at real-time. We plan to investigate further acceleration and parallelization, e.g. lazy evaluation [Dong et al. 2008] for certain phenomenon such as liquids where the most interesting effects happen near a narrow interface.
- Even though we have shown only texturing results for Eulerian fluid systems, our method is equally applicable to Lagrangian particles. This is analogous to texture synthesis over regular pixel grids versus irregular mesh vertices [Turk 2001; Wei and Levoy 2001] or surface samples [Bargteil et al. 2006; Kwatra et al. 2007].
- Extend our basic algorithm to other quantities that are also subject to projection or coordinate transformation, such as agent positions [Kyriakou and Chrysanthou 2008].

Acknowledgements Xin Tong, Xin Sun, Zhong Ren and Kangkang Yin suggested on improving results quality. Qiming Hou and Minmin Gong answered questions regarding GPU programming. Su Wang made the seaweed model. Kai Xu sketched the object contours in Figure 8. Matt Scott helped with video dubbing. Bridson et al. [2007] and Kim et al. [2008] provided source code online. Reviewers provided invaluable feedback. Kun Zhou is partially supported by the NSF of China (No. 60825201), the 973 Program of China (No. 2009CB320801) and NVIDIA.

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *SI3D '01*, 217–226.

BARGTEIL, A. W., SIN, F., MICHAELS, J. E., GOKTEKIN, T. G., AND O'BRIEN, J. F. 2006. A texture synthesis method for liquid animations. In *SCA '06*, 345–351.

BHAT, K. S., SEITZ, S. M., HODGINS, J. K., AND KHOSLA, P. K. 2004. Flow-based video synthesis and editing. In *SIGGRAPH '04*, 360–363.

BRIDSON, R., AND MÜLLER-FISCHER, M. 2007. Fluid simulation: Siggraph 2007 course notes. 1–81.

BRIDSON, R., HOURIHAM, J., AND NORDENSTAM, M. 2007. Curl-noise for procedural fluid flow. In *SIGGRAPH '07*, 46:1–3.

CABRAL, B., AND LEEDOM, L. C. 1993. Imaging vector fields using line integral convolution. In *SIGGRAPH '93*, 263–270.

CHUANG, Y.-Y., GOLDMAN, D. B., ZHENG, K. C., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2005. Animating pictures with stochastic motion textures. In *SIGGRAPH '05*, 853–860.

DONG, Y., LEFEBVRE, S., TONG, X., AND DRETTAKIS, G. 2008. Lazy solid texture synthesis. In *EGSR '08*, 1165–1174.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *SIGGRAPH '01*, 341–346.

FATTAL, R., AND LISCHINSKI, D. 2004. Target-driven smoke animation. In *SIGGRAPH '04*, 441–448.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *SIGGRAPH '01*, 15–22.

FIDANER, I. B., 2008. A survey on variational image inpainting, texture synthesis and image completion. `http://www.scribd.com/doc/3012627/`.

FISHER, M., SCHRÖDER, P., DESBRUN, M., AND HOPPE, H. 2007. Design of tangent vector fields. In *SIGGRAPH '07*, 56:1–9.

HAN, J., ZHOU, K., WEI, L.-Y., GONG, M., BAO, H., ZHANG, X., AND GUO, B. 2006. Fast example-based surface texture synthesis via discrete optimization. *Vis. Comput. 22*, 9, 918–925.

HONG, J.-M., LEE, H.-Y., YOON, J.-C., AND KIM, C.-H. 2008. Bubbles alive. In *SIGGRAPH '08*, 48:1–4.

KIM, T., THÜREY, N., JAMES, D., AND GROSS, M. 2008. Wavelet turbulence for fluid simulation. In *SIGGRAPH '08*, 50:1–6.

KOPF, J., FU, C.-W., COHEN-OR, D., DEUSSEN, O., LISCHINSKI, D., AND WONG, T.-T. 2007. Solid texture synthesis from 2d exemplars. In *SIGGRAPH '07*, 2:1–10.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. In *SIGGRAPH '05*, 795–802.

KWATRA, V., ADALSTEINSSON, D., KIM, T., KWATRA, N., CARLSON, M., AND LIN, M. 2007. Texturing fluids. *IEEE TVCG 13*, 5, 939–952.

KYRIAKOU, M., AND CHRYSANTHOU, Y. 2008. Texture synthesis based simulation of secondary agents. In *Motion in Games*, 1–10.

LI, Y., WANG, T., AND SHUM, H.-Y. 2002. Motion texture: a two-level statistical model for character motion synthesis. In *SIGGRAPH '02*, 465–472.

LIANG, L., LIU, C., XU, Y.-Q., GUO, B., AND SHUM, H.-Y. 2001. Real-time texture synthesis by patch-based sampling. *ACM TOG 20*, 3, 127–150.

LIU, Y., LIN, W.-C., AND HAYS, J. 2004. Near-regular texture analysis and manipulation. In *SIGGRAPH '04*, 368–376.

LONG, B., AND REINHARD, E. 2009. Real-time fluid simulation using discrete sine/cosine transforms. In *SI3D '09*, 99–106.

MCNAMARA, A., TREUILLE, A., POPOVIĆ, Z., AND STAM, J. 2004. Fluid control using the adjoint method. In *SIGGRAPH '04*, 449–456.

NARAIN, R., KWATRA, V., LEE, H., KIM, T., CARLSON, M., AND LIN, M. 2007. Feature-guided dynamic texture synthesis on continuous flows. In *EGSR '07*, 361–370.

NARAIN, R., SEWALL, J., CARLSON, M., AND LIN, M. C. 2008. Fast animation of turbulence using energy transport and procedural synthesis. In *SIGGRAPH Asia '08*, 166:1–8.

OKABE, M., ANJYO, K., IGARASHI, T., AND SEIDEL, H.-P. 2009. Animating pictures of fluid using video examples. In *Eurographics '09*, 677–686.

PALACIOS, J., AND ZHANG, E. 2007. Rotational symmetry field design on surfaces. In *SIGGRAPH '07*, 55:1–10.

PULLEN, K., AND BREGLER, C. 2002. Motion capture assisted animation: texturing and synthesis. In *SIGGRAPH '02*, 501–508.

REYNOLDS, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87*, 25–34.

SCHECHTER, H., AND BRIDSON, R. 2008. Evolving sub-grid turbulence for smoke animation. In *SCA '08*, 1–7.

STAM, J. 1999. Stable fluids. In *SIGGRAPH '99*, 121–128.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH '02*, 665–672.

TONG, Y., LOMBEYDA, S., HIRANI, A. N., AND DESBRUN, M. 2003. Discrete multiscale vector field decomposition. In *SIGGRAPH '03*, 445–452.

TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. 2006. Continuum crowds. In *SIGGRAPH '06*, 1160–1168.

TURK, G. 2001. Texture synthesis on surfaces. In *SIGGRAPH '01*, 347–354.

VAN DEN BERG, J., PATIL, S., SEWALL, J., MANOCHA, D., AND LIN, M. 2008. Interactive navigation of multiple agents in crowded environments. In *SI3D '08*, 139–147.

VAN DER LAAN, W. J., GREEN, S., AND SAINZ, M. 2009. Screen space fluid rendering with curvature flow. In *SI3D '09*, 91–98.

WANG, L., YU, Y., ZHOU, K., AND GUO, B. 2009. Example-based hair geometry synthesis. In *SIGGRAPH '09*, 56:1–9.

WEI, L.-Y., AND LEVOY, M. 2001. Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH '01*, 355–360.

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the art in example-based texture synthesis. In *EG STAR*, 93–117.

YUKSEL, C., HOUSE, D. H., AND KEYSER, J. 2007. Wave particles. In *SIGGRAPH '07*, 99:1–8.

Supplementary Materials

A Temporal integration scheme for smoke density rendering

In the main part of the paper we have described our algorithms for generating motions. Here, we discuss some related issues for rendering. Even though rendering is neither the main purpose nor the contribution of this paper, we have to rely on that to show our motions. For most of the rendering methods such as particles (Figure 7) or polygonal models (Figure 9), we could generate our results by direct rendering from the final motion fields $\mathbf{F}$. However, for smoke rendering, we have to take care of the temporal integration for the density field for best quality. Details are as follows.

For smoke rendering, a naive temporal integration scheme is to advect the density fields sequentially according to our generated motion fields. Given a sequence of our combined motion fields $\mathbf{F}_{i=1:n}$ where n is the number of frames, the output sequence of density fields $\mathbf{D}_{i=1:n}$ can be generated according to Pseudocode 3.

function $\mathbf{D}_{i=0:n} \leftarrow$ AdvectDensitySequenceNaive($\mathbf{F}_{i=1:n}$)
  // $\mathbf{F}_{i=1:n}$*: final combined motion fields*
  $\mathbf{D}_0 \leftarrow$ empty density field
  for $i = 1$ **to** n
    $\mathbf{D}_i \leftarrow \mathbf{D}_{i-1}$
    $\mathbf{D}_i \leftarrow$ inject density sources
    $\mathbf{D}_i \leftarrow$ advect $\mathbf{D}_i$ according to $\mathbf{F}_i$
        via the semi-Lagrangian scheme [Stam 1999]
  end

Pseudocode 3: Pseudo-code for a naive temporal integration scheme.

However, this naive method does not tend to produce compelling rendering for a very simple reason. Unlike particles (Figure 7) or polygonal models (Figure 9) whose motions could be clearly traced visually by the distinctive geometric elements, the smoke density could become messed up after sufficient accumulation, particularly at later frames. This could produce the perception that the output motion is stochastic, even though in reality it is not. (This problem is particularly pronounced in 3D when we essentially composite multiple layers of smoke for rendering.) Intuitive, this problem could be solved by letting the old density values gradually die out and disappear, but this is not very easy to implement in an Eulerian method for storing density values.

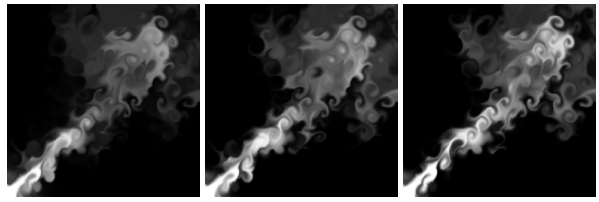
To address this issue, we have found a simple heuristic that works well. Instead of advecting smoke density according to all previous frames as in Pseudocode 3, we perform the advection across only a limited number frames so that only sufficiently recent density injections are integrated. Our proposed temporal integration scheme is shown in Pseudocode 4. We have applied our temporal integration scheme for all our smoke rendering results in the paper images and video demo. There, the user simply chooses a parameter m specifying the number of frames to look back for integration.

Figure 10 compares our results with the naive and our improved temporal integration scheme. As can be seen, our method produces better results. When m is too large, our scheme will have similar problems to the naive method as old density values could mess up with the rendering quality. However, when m is too small, the result might also not look good as there is not enough number of frames for integration to clearly depict the motion trajectory. (This aspect of our method is analogous to motion blur for the purpose of emphasizing motion trajectories, but instead of integrating scene radiance values as in traditional motion blur, we operate on the density values.) In our experience setting m in the range of $10 \sim 20$

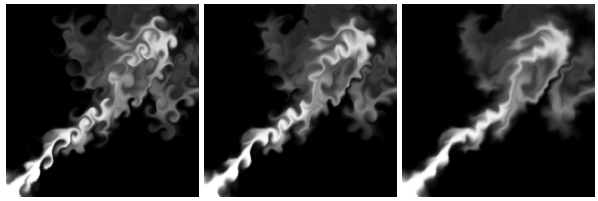
function $\mathbf{D}_{i=m:n} \leftarrow$ AdvectDensitySequenceOurs($\mathbf{L}_{i=1:n}$, $\mathbf{F}_{i=1:n}$, m)
  // $\mathbf{L}_{i=1:n}$*: coarse motion fields*
  // $\mathbf{F}_{i=1:n}$*: final combined motion fields*
  // m*: number of frames for integration*
  $\mathbf{O}_{i=0:n} \leftarrow$ AdvectDensitySequenceNaive($\mathbf{L}_{i=1:n}$)
  for $i = m$ **to** n
    $\mathbf{D}_i \leftarrow \mathbf{O}_{i-m}$
    for $j = 1$ **to** m
      $\mathbf{D}_i \leftarrow$ inject density sources
      $\mathbf{D}_i \leftarrow$ advect $\mathbf{D}_i$ according to $\mathbf{F}_{i-m+j}$
          via the semi-Lagrangian scheme [Stam 1999]
    end
  end

Pseudocode 4: Pseudo-code for our temporal integration scheme.

usually produces good results. We wish to emphasize that the proposed temporal integration scheme is for rendering only and affects only the density field, not our output motion. Thus, the rendering and the associated parameters can be tuned outside our main algorithm and do not require any re-synthesis of motions.



naive $m = 40$ $m = 20$

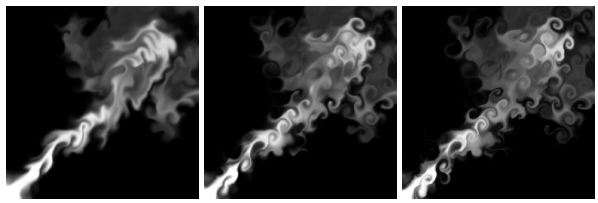


$m = 10$ $m = 5$ $m = 2$

Figure 10: Temporal integration scheme. Here, we compare the naive temporal integration scheme with our method under different m values.

B The ω parameters

In our implementation, we treat the weighting ω as a global constant, which can be tuned interactively after synthesis process. In Figure 4, ω is set to be 15, 50, 40, 30 and 30 respectively, while 30, 40 and 120 for the results in Figure 5. Figure 11 shows final rendering results under different values for ω while keeping all the other parameters fixed.



$\omega = 5$ $\omega = 15$ $\omega = 30$

Figure 11: The ω parameter for final combination. Here, we show results under different constant values for ω.