Multiscale Image Based Flow Visualization

Alexandru Telea[a] and Robert Strzodka[b]

[a]Department of Mathematics and Computer Science, Eindhoven University of Technology, Netherlands

[b]Centre of Advanced European Studies and Research (caesar), Bonn, Germany

ABSTRACT

We present MIBFV, a method to produce real-time, *multiscale* animations of flow datasets. MIBFV extends the attractive features of the Image-Based Flow Visualization (IBFV) method, i.e. dense flow domain coverage with flow-aligned noise, real-time animation, implementation simplicity, and few (or no) user input requirements, to a multiscale dimension. We generate a multiscale of flow-aligned patterns using an algebraic multigrid method and use them to synthesize the noise textures required by IBFV. We demonstrate our approach with animations that combine multiple scale noise layers, in a global or level-of-detail manner.

Keywords: flow visualization, texture-based animations, multiscale decomposition, algebraic multigrid

1. INTRODUCTION

Texture-based methods for flow field visualization are an important class of tools for getting insight on complex datasets, such as produced by flow simulations, weather and environment prediction systems, earth sciences, and so on. Texture-based methods produce a dense pixel image that covers the whole flow domain with patterns which, by their orientation, color, and distribution, efficiently and effectively convey insight into the vector data. Texture-based methods have several attractive aspects. First, the pixel image produced, usually consisting of flow-aligned fine-scale noise patterns, is easy to comprehend even by non-specialists. Secondly, efficient methods exist that produce such images in real time, for complex flow datasets, by using graphics hardware. This allows producing real-time animations, which are probably the most effective way to visually convey the flow motion.[1] Finally, most texture-based flow methods require little or no user input, as compared to e.g. streamline based methods. This makes them ideal for quick exploration of new datasets. Image-based flow visualization (IBFV) is an excellent example of texture-based methods.[1] IBFV produces real-time dense animated displays of flow data by injecting, advecting, and blending a periodic fine-grained noise image. It uses a simple, but very efficient graphics hardware implementation which yields hundreds of frames per second on nowadays graphics cards.

Another important dimension of flow visualization methods is their ability to show the flow data in a *multiscale* fashion: Fine levels show local flow details, whereas coarse levels show more global, dominant flow behavior. Multiscale methods are attractive especially for complex datasets, as they simplify the user's task of interpreting the flow field by emphasizing salient features and/or providing on-demand local small-scale details.

However, up to now, there is only little work done to combine the attractive aspects of texture-based flow animations with those of multiscale methods. We propose here a framework that extends the basic idea of IBFV to a multiscale dimension. We combine all attractive points of IBFV (dense coverage, real-time flow data animation, implementation simplicity, automated visualization) with the main attractive aspect of multiscale methods (show fine details on finer scales and/or dominant details on coarse scales). For the multiscale flow decomposition, we use the method presented by Griebel *et al.*[2] based on the algebraic multigrid (AMG) technique. This method decomposes a flow field in a multiscale of flow-aligned, fine-to-coarse basis functions, or patterns. We use these patterns as the noise input for the IBFV method. This allows us to produce real-time flow animations in the spirit of IBFV, at any spatial scale. Next, we extend this idea by combining several layers of such texture-based flow animations to convey both fine-scale and coarse-scale information in the same image. Performing this multi-layer animation composition locally, leads to a local level-of-detail visualization. The multiscale flow representation produced by the AMG method lends itself perfectly to be combined with the IBFV animation approach.

[a]E-mail: alext@win.tue.nl; [b]E-mail: strzodka@caesar.de

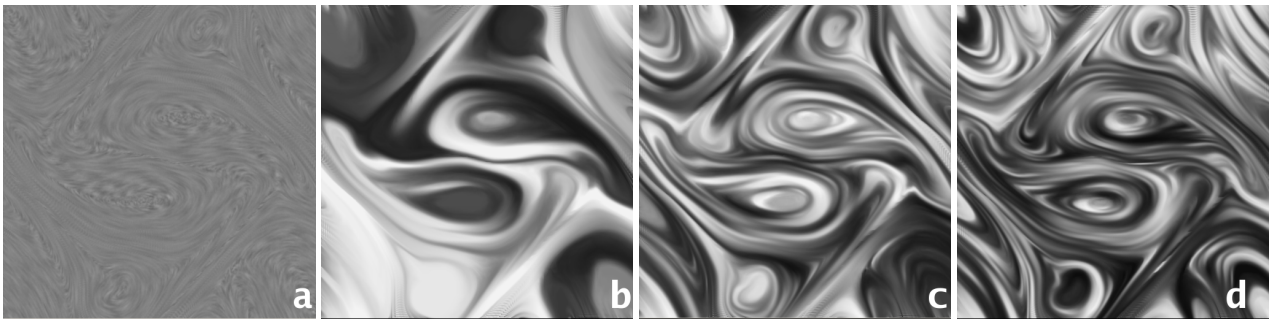

Figure 1. Fine-scale IBFV and first three coarse-scale MIBFV visualizations.

The structure of this paper is as follows. In Sect. 2 we review related work on texture-based and multiscale flow visualization. Section 3 explains how the multiscale flow-aligned noise produced by the AMG method is combined with the IBFV animation to yield the multiscale flow animation. Section 4 presents several enhancements of the basic idea: multi-layered flow animations and local level-of-detail animations. Section 5 explains the implementation details. Section 6 concludes the paper.

2. PREVIOUS WORK

Numerous texture-based flow visualization methods exist, starting with spot noise[3] and line integral convolution (LIC),[4] which visualize flow fields by dense images of fine-scale flow-aligned noise. Due to advancements in the performance of graphics processor units (GPUs), recent methods have focused more on animating such textures. A well-known method here is the Image-Based Flow Visualization (IBFV) method.[1] IBFV produces real-time flow animations by injecting (blending) and advecting a time-dependent periodical, fine-scale white noise texture. A main feature of IBFV is its simplicity: its basic OpenGL implementation has less than 100 C code lines. IBFV was also extended to produce flow animations on curved surfaces[5] and in 3D volumes.[6] A similar idea to the animation on curved surfaces[5] is proposed by the Image-Space Advection (ISA) method.[7] Whereas IBFV (and related) methods advect textures by warping polygon meshes, the Lagrangian-Eulerian Advection (LEA) method[8] and its followers[9, 10] do a per-pixel advection using programmable GPUs. Both LEA and IBFV are unified in the Unsteady Flow Advection Convolution (UFAC) framework[10] which also uses programmable GPUs. For more details on the techniques above, see the review of Laramee *et al*.[11]

However, none of the above texture-based flow animations can produce a *spatial multiscale* view of the flow. Essentially, the above methods follow the original LIC and spot noise techniques, i.e., display a flow-aligned fine-scale noise pattern of near pixel size. Often, though, one wants to produce a more simplified view of the flow, where only important, i.e. geometrically large, flow patterns show up. However, simply increasing the noise granularity size above the pixel size is not a solution. As explained by Van Wijk,[1] this leads to random high frequency visual patterns, since the shape of the noise 'dots' is not correlated (aligned) with the flow.

Other approaches exist to produce such multiscale flow-aligned dense patterns. Preußer and Rumpf use anisotropic, flow-aligned, diffusion of fine scale noise to produce static multiscale flow-aligned patterns.[12, 13] Adding transport produces flow animations.[14] Diffusion time plays the role of a scale parameter, as larger patterns appear after longer diffusion times. Both above methods use a contrast-enhancing term to sharpen the flow patterns. Still, diffusion yields visible blurring, especially on larger scales. Another multiscale dense pattern generation method simulates the physical two-phase Cahn-Hilliard clustering model.[15] The two-phase model nicely overcomes the blurring problem. However, this method has not been used to produce flow animations. Moreover, both this and the diffusion-based methods[12, 14] are quite complex to implement and much slower than the real-time IBFV animation.

Finally, we mention the flow field clustering method using the algebraic multigrid (AMG).[2] This method produces a multiscale of flow-aligned basis functions on the flow domain. In the following, we shall use these basis functions to create a flow-aligned noise signal which is fed to the IBFV animation.

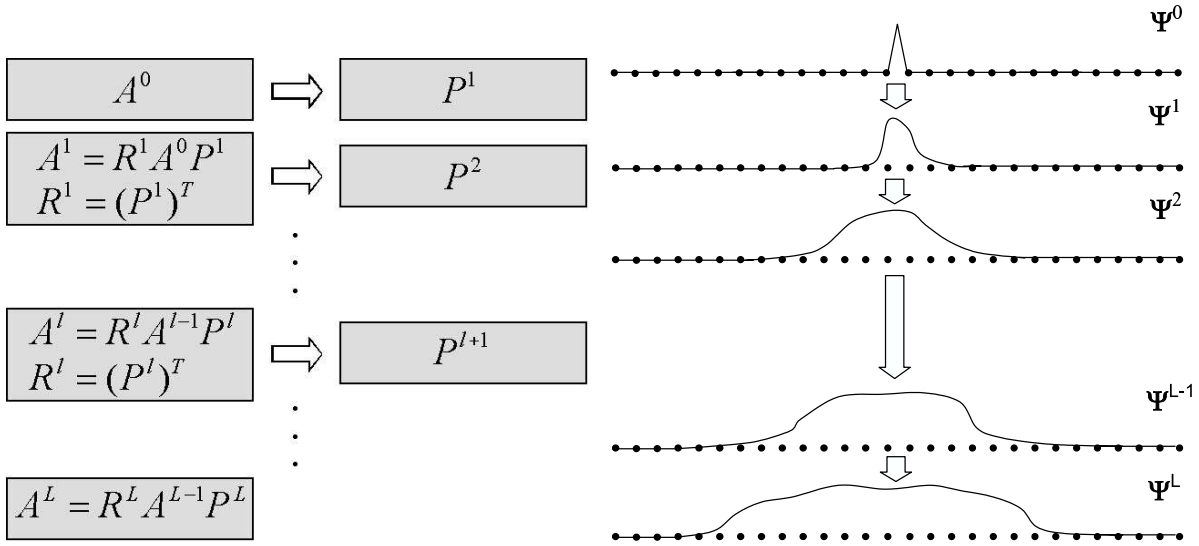

Figure 2. Sketch of the AMG algorithm (left). Fine to coarse hierarchy of basis functions (right).

3. MULTISCALE IBFV

Let us consider the IBFV process. A vector field $\mathbf{v}(\mathbf{x}, t)$ is visualized by an animated dense texture

$$F(\mathbf{x}, t) = (1 - \alpha)F(\mathbf{x}_p, t_p) + \alpha G(\mathbf{x}, t) \tag{1}$$

obtained by blending a noise signal $G(\mathbf{x}, t)$ together with an advected version $F(\mathbf{x}_p, t_p)$ of the texture itself. The advection $\mathbf{x} = \mathbf{x}_p + \mathbf{v}(\mathbf{x}_p, t_p)\Delta t$ is done by warping a polygon mesh that discretizes the flow domain. Time $t = t_p + \Delta t$ is uniformly discretized. This produces fine-scale animations as shown in Fig. 1 a. The vector field, originating from a magnetohydrodynamic simulation, is visualized with similar methods in Ref.[2] and Ref.[13] The noise $G(\mathbf{x}, t)$ is a periodic function with per-pixel random phase $n(\mathbf{x}) \in [0, 1]$, e.g. $G(\mathbf{x}, t) = w(t + n(\mathbf{x}))$, where $w(t)$ is a periodic function with values in $[0, 1]$. As explained in Sect. 2, we cannot simply spatially up-scale the noise $n(\mathbf{x})$, i.e. use $n'(\mathbf{x}) = n(K\mathbf{x})$ where K is some scaling factor, to produce spatially coarse, yet flow-aligned patterns. We must somehow correlate $n(\mathbf{x})$ with the flow. We do this using the AMG-based flow decomposition,[2] described next. Our final multiscale results are depicted in Fig. 1 b,c,d.

3.1. AMG field decomposition

In a nutshell, the AMG-based technique for flow clustering presented by Griebel *et al.*[2] works as follows. Given a two-dimensional field $\mathbf{v}$, we first define a field-aligned coupling tensor $a(\mathbf{v})$

$$a(\mathbf{v}) = B(\mathbf{v})^T \begin{pmatrix} K(\|\mathbf{v}\|) & 0 \\ 0 & \epsilon \end{pmatrix} B(\mathbf{v})$$

where $B(\mathbf{v})$ is a rotation from the canonical basis (e_0, e_1) in $\mathbb{R}^2$ to $(\mathbf{v}, \mathbf{v}^\perp)$. K is a constant a few orders of magnitude larger than ϵ, which ensures much stronger coupling along, rather than across, $\mathbf{v}$. Next, we consider a finite element discretization of the flow domain Ω, i.e., a meshing of Ω with nodes $(X_i)_{i=1,\cdots,n}$ and a set of piecewise linear basis functions $\Phi^i(X_j) = \delta_{ij}$, where δ_{ij} is the Kronecker symbol. Basically, these are tent linear functions defined on the mesh triangles. Having this basis, we encode the tensor $a(\mathbf{v})$ in a finite element matrix $A_{ij} := \int_\Omega a(\mathbf{v})\ \nabla\Phi^i \cdot \nabla\Phi^j\, dx$. That is, the entry A_{ij} is high (K) if vertices i and j are neighbors and the edge (i, j) is aligned with the flow $\mathbf{v}$, low (ϵ) if the edge is perpendicular on $\mathbf{v}$, and zero if i,j are not neighbors.

Now the AMG technique comes into play. Algebraic multigrid techniques were introduced in the 1980's to accelerate solving linear systems $AU = b$. Below we give only a very simple overview of AMG. For more practical as well as

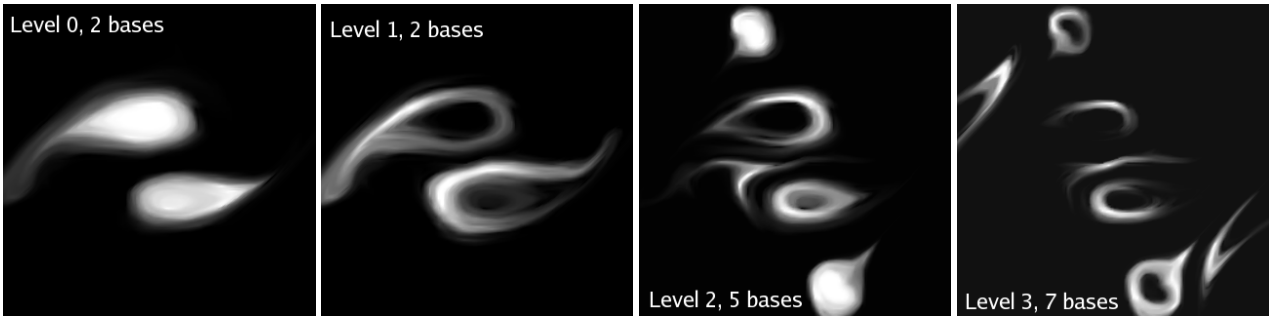

Figure 3. Selected basis functions on different AMG decomposition levels.

theoretical details, we refer to Ref.[2] Given some matrix A, AMG produces a sequence of progressively coarsened (simplified) matrices A^l, such that each A^l approximates the operator encoded by A. For every level l, AMG computes A^l using only the previous level $l-1$ in the so-called Galerkin projection $A^l := R^l A^{l-1} P^l$, where $R^l := (P^l)^T$ is the restriction and P^l the prolongation from level $l-1$ to l. Figure 2 left illustrates the AMG process. How are the prolongations P^l constructed? To each level l, a corresponding basis $\Psi^l := \{\Psi^{l,i}\}$ exists. AMG starts with the finest level $l=0$ on which we have the piecewise linear finite element basis functions mentioned above in the construction of $A^0 := (A_{ij})_{ij}$. Such a basis function is sketched in 1D at the top in Fig. 2 right, the dots in the figure represent the mesh points. Next, AMG tries to merge fine level basis functions from Ψ^{l-1} to construct a coarse basis Ψ^l, using the notion of *algebraic smoothness*. Common to nowadays algebraic smoothness criteria is the general observation that a simple relaxation scheme – most often Gauss–Seidel smoothing – efficiently damps components in the direction of matrix eigenvectors associated with large eigenvalues. Consequently, the coarse basis functions are chosen such that they deal with the remaining components of an eigenvector decomposition. Given that our matrix A encodes an anisotropic diffusion tensor $a(\mathbf{v})$, AMG will basically join flow-aligned fine-scale basis functions to construct flow-aligned, coarse-scale basis functions. Figure 2 right illustrates this bottom-up basis construction for the 1D case. In other words, each matrix A^l delivered by AMG, starting with the initial finest-scale A^0 up to the coarsest A^L, where L is the number of decomposition levels, approximates the anisotropic diffusion operator using the matrix-dependent, flow-aligned basis $\{\Psi^{l,i}\}$, where $A^l_{ij} = A\overline{\Psi^{l,i}} \cdot \overline{\Psi^{l,j}} = \int_\Omega a(\mathbf{v}) \nabla\Psi^{l,i} \cdot \nabla\Psi^{l,j}$. Here, $\overline{\Psi^{l,i}}$ is the nodal vector corresponding to the basis function $\Psi^{l,i}$, i. e. $\Psi^{l,i} = \sum_{j=1,\cdots,n} (\overline{\Psi^{l,i}})_j \Phi^j$ where Φ^j are the finest-scale basis functions, $\Phi^j = \Psi^{0,j}$. The whole process is sketched in Fig. 2.

Although the AMG technique and its implementation are rather involved, we just use it as a black box: We take a field $\mathbf{v}$, encode the matrix A using $\epsilon = 10^{-3}$, $K=1$, feed it to the AMG, and obtain a sequence of bases $\Psi^l = \{\Psi^{l,i}\}$, $l = 0..L$. Figure 3.1 shows, for the flow field in Fig. 1, several basis functions picked on a few decomposition levels, with a black (0) to white (1) colormap. The bases are aligned with the flow on all levels. Finer level bases have smaller supports. Finest-level ones cover exactly one mesh triangle. Coarser bases have increasingly large, yet flow aligned, supports. Basis functions on the same level may overlap, but still have localized (compact) support. Supports on the same level l are of about the same size. The average support size is reduced by a factor roughly equal to 2 from l to $l+1$, a property inherent to the AMG coarsening scheme. Given these properties, we use the basis functions to design our multiscale noise signal, as described next.

3.2. Multiscale, field-aligned noise

The basic noise model used by the IBFV is $G(\mathbf{x}, t) = w(t + n(\mathbf{x}))$ (Sect. 3). We can express G also as:

$$G(\mathbf{x}, t) = \sum_i \delta_i(\mathbf{x}) w(t + n_i) \tag{2}$$

where the domain of G is discretized as a set of pixels i, δ_i is a unit (discrete Dirac) pulse function centered at pixel i, $w(t) = w(t+\tau)$ is a function of period τ, and n_i is a random phase value in $[0, 1]$. Essentially, the above expresses the noise G using δ_i, a set of constant basis functions, centered on the pixel grid. Now we can generalize the noise G, and thus the IBFV method, to a multiscale dimension. For this, we simply replace the above basis δ_i with any of the bases Ψ^l

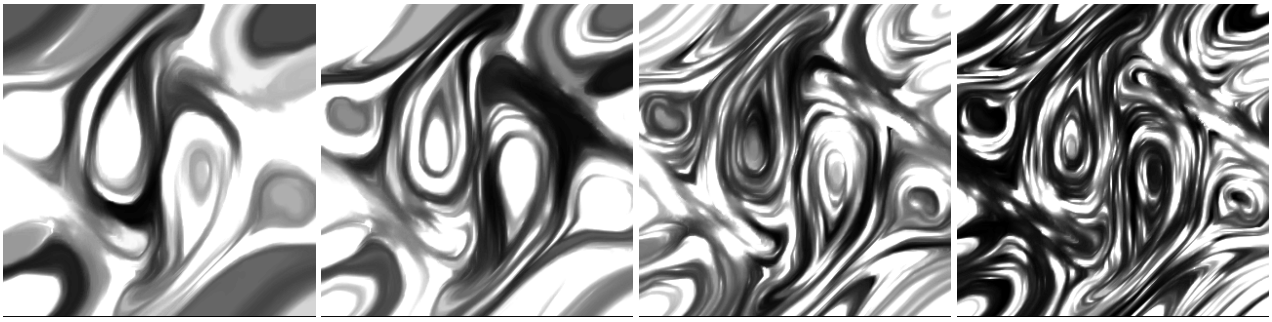

Figure 4. Multiscale noise textures, four coarsest levels.

delivered by the AMG decomposition. Figure 4 shows four multiscale noise images G for the first four coarsest scales of the field in Fig. 1.

There are two main reasons why the AMG basis functions are good for building multiscale noise. First, the cross-overlap between different basis functions at the *same* time moment, i.e. between $\Psi^{l,i}$ for the same l and different i, is rather small, as mentioned in Sect. 3. This is a by-product of the AMG coarsening scheme. Given this, we can sum up basis functions modulated with random noise (Eqn. 2) without obtaining high-frequency artifacts. A small amount of overlap is good, as this leads to a bit of smoothing between the differently weighted basis functions, thus a nice continuous-looking image. Too sharp borders between these basis functions would be visually disturbing in the animation, especially on coarse levels. No 'overshooting' effects appear: All terms in Eqn. 2 sum up to 1, given the partition of unity property of the basis functions, i.e. $\sum_i \Psi^{l,i} = 1$ for any l. If desired, the amount of cross-overlap can be decreased by e.g. raising the basis functions delivered by the AMG to a power of 2 or 3. Since AMG basis functions smoothly decrease from 1 somewhere in their support center to zero at the borders, this narrows their support and concentrates their energy towards the support center.

Secondly, our basis functions have elongated, flow-aligned supports (Sect. 3). Hence, when advection is done with a small time step Δt by the IBFV (Eqn. 1), the overlap between the original and advected supports of the same basis function $\Psi^{l,i}$ is quite large. Thus, the self-overlap (of a basis function with its advected version) is maximized, whereas the cross-overlap (of an advected basis function and other basis functions) is minimal. The advected basis function gets mostly blended with its own previous version. Since Δt is small, the difference $w(t + n_i + \Delta t) - w(t + n_i)$ between the colors, or luminances, of the basis and its advected self, is also quite small. Hence, the blending (weighted summation) of the two does not lead to high-frequency visual artifacts. If however, we used some basis functions $\Psi^{l,i}$ whose supports were *not* flow-aligned, then the advection would cause large cross-overlaps. Over these overlaps, different colors (or luminances) get summed up, since different basis functions have random phases n_i, hence different colors. This shows up as high-frequency visual noise that destroys the flow animation effect, as shown in Ref.[1]

4. ENHANCEMENTS

So far, we have visualized the flow with a single AMG basis $\Psi^l = \{\Psi^{l,i}\}$. However, this yields the same level-of-detail over the whole flow domain. Combining several AMG bases on different levels l in the visualization yields more insight into the flow data. We present two such techniques: Blending independent MIBFV animations running on different levels (Sect. 4.1) and modulation of noise from different levels into a single MIBFV animation (Sect. 4.3). We can also apply these techniques to user specified areas of the flow domain, rather than its entirety, to achieve a local level-of-detail effect (Sect. 4.2). Finally, we may use color to emphasize either the global structure of the flow or the dynamics therein (Sect. 4.4).

4.1. Bi-Level Visualization

To produce a bi-level visualization, we simply run two MIBFV animations using bases from two different AMG levels l_1 and l_2 in parallel, and then additively blend the results, yielding the result in Fig. 6 left. This type of visualization combines both coarse flow structures (such as Fig. 1 b,c,d) with fine-level structures (such as Fig. 1 a) in the same animation. Figure 4.1 shows four coarsest levels of a bi-level visualization of a flow field from a Benard convection simulation. These

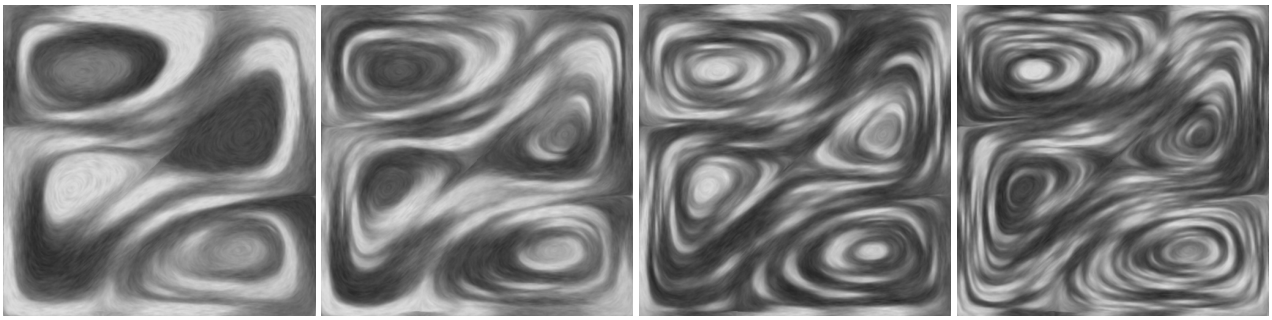

Figure 5. Bi-level MIBFV visualization, four coarsest levels.

images should be compared with the same field visualized by the Cahn-Hilliard clustering method[15] and anisotropic noise diffusion method.[13]

The bi-level animation is most useful when the difference between l_1 and l_2 is quite large, otherwise the result is similar to a one-level animation. A good choice is to set l_2 to the finest level. As explained in Sect. 3.2, the finest-level MIBFV is identical to the original IBFV. Thus we may use IBFV directly, as originally described in Ref.,[1] to implement the finest-level animation. Given the high number of l_2-level basis functions, saved by the AMG as textures (Sect. 5), this saves several seconds of preprocessing. For an even clearer visual separation of the two animation levels, one can also use a different period of the noise signals w on the two levels: A quickly oscillating fine noise easily stands out against a slowly changing coarse background.

4.2. Level-of-Detail Lens

The bi-level animation shows both overview and detail over the whole flow domain. However, sometimes we want the detail displayed only in a limited, e.g. user-chosen focus, area. This is the classical *context and focus* approach on data visualization. For this, we use a coarse level for the context animation and a fine level for the focus animation (Fig. 6 right). We limit the focus to circular areas around some points of interest, interactively chosen by the user, e.g. using the mouse. Such points are indicated by black dots in Fig. 6 right. We achieve a smooth transition from focus to context by blending the fine level animation image (focus) with the coarse level animation image (context) using a smooth radial decaying function, e.g. a radial cosine or exponential. This blending only effects the display. The fine and coarse MIBFV animations must be computed in parallel, independently of each other. Indeed, if we were to inject fine noise at the focus point, and advect it together with the coarse-noise background in a single animation, the fine noise would 'bleed' when the user interactively changes the focus point, yielding undesired artifacts.

4.3. Noise Modulation

A different strategy combining multiple spatial levels is to run a single MIBFV animation but to compose the noise from different levels. Suppose we pick two levels and their fine and coarse noise signals G_f and G_c. Figure 7 illustrates these in the one-dimensional case. We can now use a noise $G = \beta G_f + (1 - \beta)G_c$, i.e. compose the fine and coarse noises by a weighted averaging. The resulting signal (Fig. 7 middle) resembles both the coarse and the fine structure. Indeed, averaging keeps both low and high frequencies. Accordingly, the advected results still show both fine and coarse structures in the animation (Fig. 8 left). However, since averaging reduces the signal contrast even further than done by frame blending, coarser structures are more visible in the animation. Still, an advantage of this technique over the basic coarse-level visualization (Fig. 1 b,c,d) is the reduction of the artificial effect of pop-in and pop-out of large basis functions in the animation (see accompanying videos and demo). The effect is lessened because now the coarse basis functions do not appear all white or all black, allowing to convey the impression of motion also within the basis functions' supports. By adjusting the blending factor β between the fine and coarse noises G_f, G_c, we also obtain a smooth transition from one level to the other, e.g. we can run the IBFV with only a slight overtone of the coarse structure.

A second possibility is to multiply the two noise signals G_c, G_f (Fig. 7 right). Again, both fine and coarse structures are preserved, but this time contrast stays high. The coarse noise G_c serves as an envelope for the fine noise G_f, yielding a

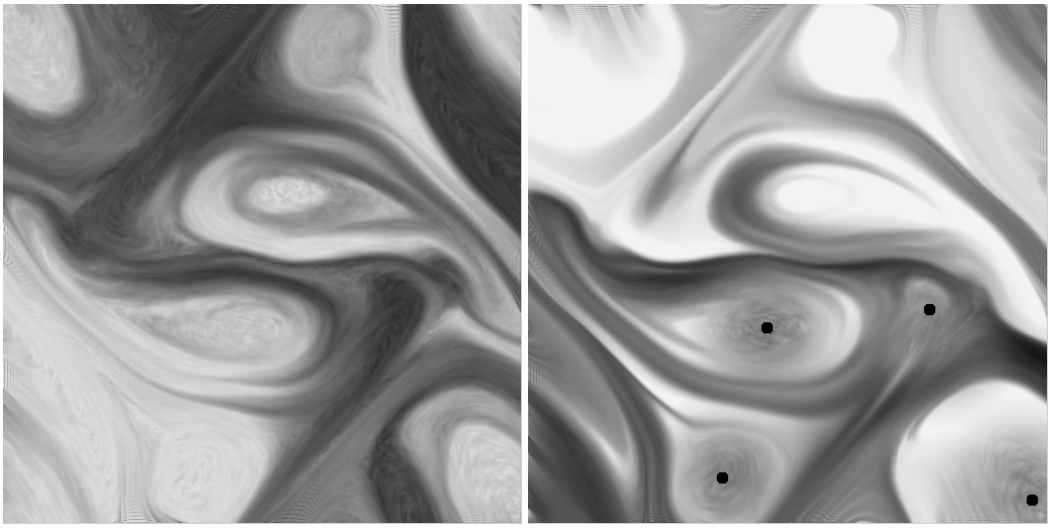

Figure 6. Bi-level visualization (left). Local level-of-detail (right).

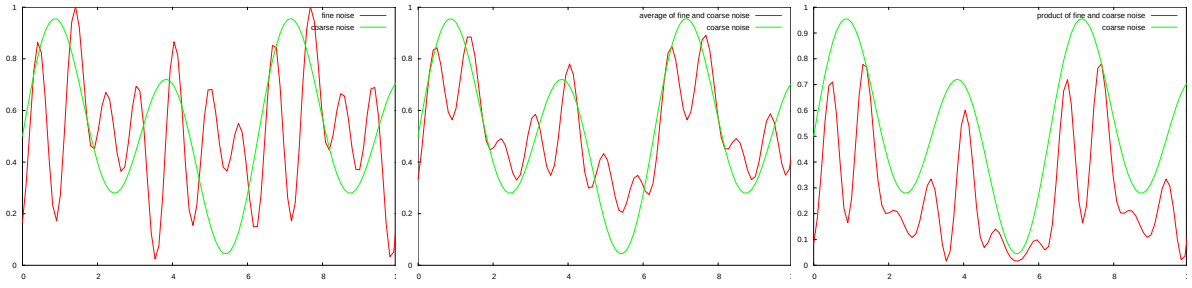

Figure 7. Fine and coarse noise signals (left). Weighted average (middle) and product (right).

fine noise modulation onto the coarse noise. Now coarse basis functions carrying a fine pattern are advected in the MIBFV. This generates a highly correlated coarse structure as well as a fine scale motion within the basis functions. Moreover, the high contrast is still present in the animation (Fig. 8 middle).

Finally, we can combine the two noises G_f and G_c as before (addition or multiplication), but use different frequencies for the two. Usually we propose a higher frequency for G_f than for G_c (see Eqn. 2). While a coarse basis function is faded in, the pattern modulated on it changes (Fig. 8 middle). By swapping the frequencies, i.e. using a higher frequency for the coarse noise G_c than for the fine noise G_f, structures perpendicular to the flow are emphasized. The animation then reveals a very high dynamics in comparison to the other combination modes (Fig. 8 right).

4.4. Colored Noise

So far, we used just the luminance channel to construct our MIBFV animation. We see several possibilities to enhance the visualization with color while the rest of the MIBFV process remains the same:

(a) Each basis function is assigned a constant color.

(b) The phase function maps to a color, instead of luminance, map.

(c) Application-specific information, e.g. pressure, is shown via a colormap.

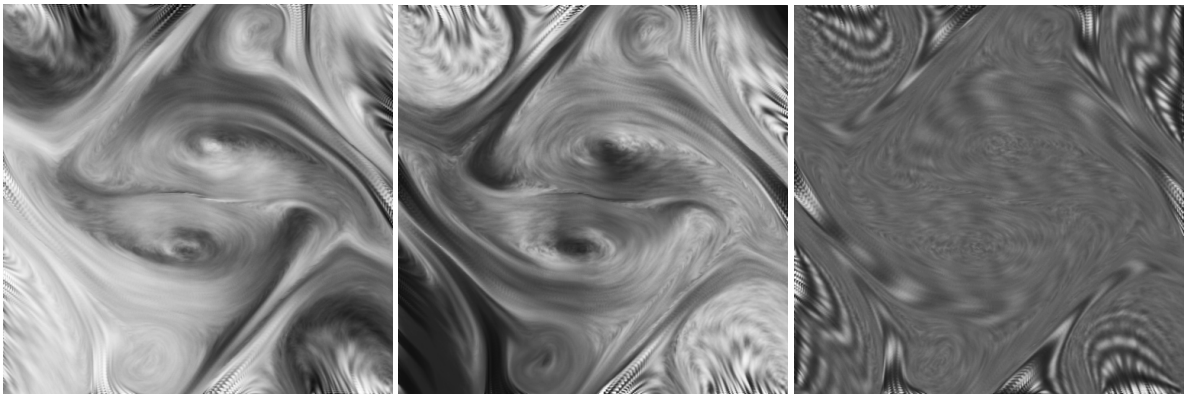

Figure 8. Noise modulation by summation (left), multiplication (middle), and using different frequencies (right).

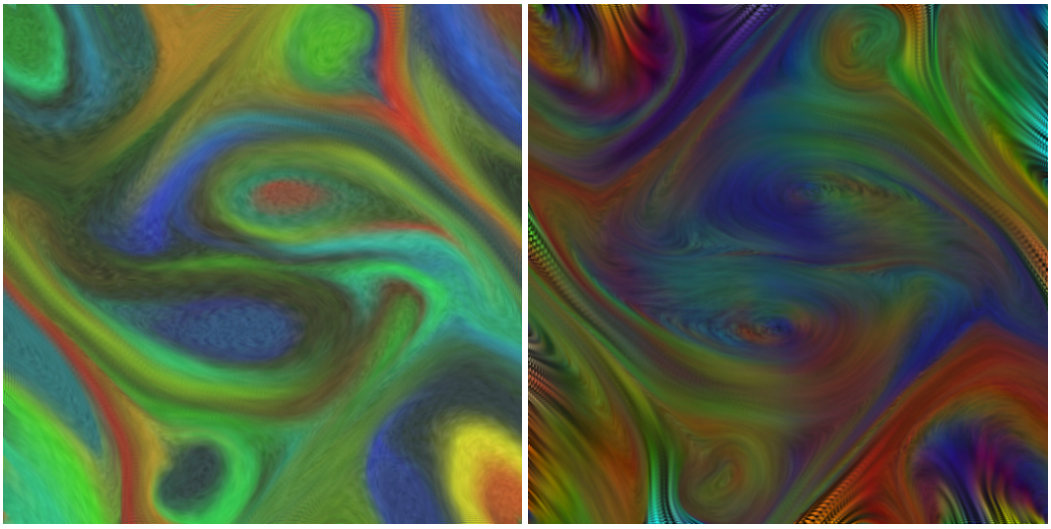

Figure 9. Color use in MIBFV: Constant-hue (left) and time-periodic hue (right) basis functions.

While (c) only requires to blend in a color signal at the very end, (a) and (b) require RGB, instead of luminance, noise textures, which triples the memory demand. However, as graphics cards are optimized for the processing of RGB textures, the overall performance penalty is very low.

In case (a), each basis function is assigned a color depending on some enumeration of the functions within the AMG level. This color is constant over time but its intensity is still modulated by the periodic function $w(t)$ (Eqn. 2). We see now coarse-level advection, but with a color emphasis of the different basis functions (Fig. 9 left).

A second technique maps the value range $[0, 1]$ of the periodic function $w(t)$ to a rainbow blue-to-red colormap. The basis functions encoded in the noise frames do not change luminance now, but hue (Fig. 9 right). This can be used to emphasize the flow dynamics. However, it should be applied with moderation, to avoid producing too dynamic, colorfully blinking, animations. If we compare the color technique (Fig. 9 right) with the monochrome one (Fig. 8 middle), the first one shows a higher dynamic range, but the monochrome one is actually easier to interpret. Overall, we believe color (hue) can best convey application-specific flow information, such as vorticity, pressure or temperature, leaving the flow itself to be encoded in the luminance channel variation.

5. IMPLEMENTATION

A main concern for us was to make the MIBFV as efficient and simple to implement as the original IBFV. The implementation consists of 3 parts:

(a) AMG field decomposition (Sect. 3.1): basis functions on chosen scales are saved as images.

(b) Noise synthesis (Sect. 3.2): noise frames are created by adding basis functions biased by random phases.

(c) MIBFV animation itself: the current image is advected and blended with the current noise frame.

Step (a) is computed in software with the same AMG preprocessing as for the visualization presented in Ref.[2] This is not a interactive procedure and is performed once on a vector field, prior to visualization. However, this step can be (and is actually) seen as a black box: one can use any tool that delivers a set of luminance images encoding flow-aligned patterns, with an approximate partition of unity property. Indeed, even though our AMG tool computes the basis functions in double floating point precision, we pass these to MIBFV via 8-bit luminance textures. This is done to reduce the memory bandwidth requirement in the later processing. There are no visible artifacts which can be attributed to this quantization, leading us to conclude that even an approximate partition of unity is fine for our purposes. This allows using a variety of multiscale pattern generation tools, possibly simpler than the AMG, together with our MIBFV.

Step (b) computes a series of noise textures, i.e. regularly samples one period of the function $G(\mathbf{x}, t)$ (Sect. 3.2). For the periodic function $w(t)$ (Eqn. 2), we use a cosine, which we sample with 128 or 256 samples per period $\tau = 2\pi$. This yields a corresponding number of noise frames, encoded as textures. For each noise frame, the addition of basis functions modulated by periodic random-phase functions (Eqn. 2), is done in graphics hardware, by a few image multiplications and additions. This is over an order of magnitude faster than in software, but the generation still requires around 10 seconds for 256 frames and average AMG levels (see Fig. 10). The two bottlenecks are the file access, for reading the basis images from the AMG tool, and the bad computational intensity of the operations. However, once the noise frames are generated and passed to the GPU as textures, switching between the scales in the MIBFV animation happens instantly.

Step (c) uses the standard polygon mesh warping and frame-buffer blending implementation introduced by the IBFV.[1] The procedure is a little bit more complicated when two levels are combined for the bi-level visualization (Sect. 4.1) and the level-of-detail lens (Sect. 4.2). Here, we perform two independent animations on a fine and a coarse level in parallel and blend their results for every frame. This costs two mesh warps instead of one but, as explained before, this is necessary to prevent the fine level-of-detail being advected by the flow. On a GeForceFX 5950 GPU, this drops the performance by about 20%, whereas on a NVIDIA Quadro FX Go card we could not measure any difference.

Overall, implementing the MIBFV is just slightly more complex than the IBFV. MIBFV still requires only the minimal OpenGL 1.1 fixed pipeline graphics hardware. The code for each of the two different bi-level animations is now roughly 150 lines of C code, compared to about 100 lines of code of the original IBFV. The noise texture synthesis takes another 150 lines of code.

Benard field

level	basis functions	synthesis(seconds)	memory
L	36	4	3.3 MiB
$L-1$	92	7	8.1 MiB
$L-2$	205	14	19.3 MiB
$L-3$	449	21	47.3 MiB

Magnetohydrodynamic field

level	basis functions	synthesis(seconds)	memory
L	34	3.5	3.5 MiB
$L-1$	83	7	9.0 MiB
$L-2$	198	11.5	20.0 MiB
$L-3$	447	20.5	43.8 MiB

Figure 10. Noise texture synthesis timings.

Basis functions are manipulated as 8-bit luminance textures. Noise frames are either also 8-bit luminance or RGB textures for the color applications. Arithmetic operations are performed with the texture environment modes and the frame-buffer blending modes, e.g. the current noise frame is injected by inserting the injection coefficient alpha (encoded as a vertex color) with the GL_MODULATE texture environment into the alpha channel, and using it in a convex combination frame-buffer blending mode, which combines the advected frame and the noise frame. The MIBFV uses a lower alpha value (0.05) as compared to the original IBFV (0.2 and higher) for injection of noise. The reason for this is that the pattern size in MIBFV is much larger than in IBFV. Thus, an advected pattern overlaps with itself much more, and for a much longer time. Consequently, it must be blended in less aggressively, i.e. with a lower alpha. However, the advection speed, measured as number of pixels a pattern is advected per animation time step, is the same for both methods, since we want to give the same visual speed impression.

Essentially, MIBFV is as fast as IBFV during the animation, i.e., we get 500 frames per second or more for graphics cards such as NVIDIA's Quadro FX Go, FX 5950, or ATI's Radeon 9800 Pro. The main price to pay for MIBFV is its pre-processing phase, i.e. the AMG decomposition and noise frame construction. The AMG decomposition takes between 1 and 3 seconds for 2D flow fields of up to 256^2 grid cells on a Pentium 4 at 1.8 GHz with 1MiB cache.[2] Figure 10 shows timings for the noise frame synthesis for the four coarsest levels $L, ..., L-3$ of the two vector fields illustrated previously, both sampled on a 256^2 grid. These timings could be significantly lowered if we passed basis functions from the AMG to MIBFV via memory, not files. The table shows also the total memory taken by the basis functions transferred from the AMG to MIBFV via image files.

IBFV can work with small size (32^2 or 64^2) noise textures which are tiled on the flow domain. For 128 time samples of the noise function (Eqn. 2) and 32^2 noise textures, this requires 128 KiB of texture memory. As MIBFV's noise textures must capture large flow aligned patterns (Fig. 4), they must be of the flow domain size. For 128 time samples and 512^2 8-bit luminance textures, this requires 32 MiB of texture memory. This is clearly higher than for the basic IBFV, but easily fits most nowadays cards with 64 MiB or more. Often, even very high memory requirements can be handled, e.g. 256 time samples of 512^2 RGB textures require 192 MiB, but the Radeon 9800 Pro card with only 128 MiB can animate this in real-time by an efficient swapping of textures to and from the main memory.

An interactive demo of the MIBFV as well as videos illustrating the method are publicly available from:
`www.win.tue.nl/~alext/MIBFV`

6. CONCLUSIONS

We have presented MIBFV, a method that extends the Image-Based Flow Visualization (IBFV) technique to a multiscale dimension. MIBFV generalizes IBFV's per-pixel, fine-scale noise structure to a spatial multiscale of flow-aligned noise patterns. We create these patterns by using an existing flow decomposition technique based on the algebraic multigrid (AMG). This produces flow-aligned basis functions with supports on different spatial scales. Mathematically, IBFV is nothing else than our MIBFV applied to the finest multiscale level, on the pixel grid. Indeed, AMG's finest-level piecewise linear basis functions (Sect. 3.1) correspond to an IBFV method using linear noise texture interpolation. IBFV's nearest neighbor texture interpolation corresponds to finest-level constant basis functions in MIBFV. From a model point of view, MIBFV is intimately related to the anisotropic diffusion flow visualization (ADFV) of Preußer *et al.*[12] Indeed, as outlined in Sect. 3.1 and detailed in Ref.,[2] AMG's decomposition of our flow matrix corresponds to a coarsening of an anisotropic diffusion operator. The main modeling difference between MIBFV and ADFV is that MIBFV first constructs the flow-aligned multiscale bases and then modulates them with noise, whereas ADFV directly applies anisotropic diffusion on the noise itself, making time play the scale parameter role. Since the AMG already offers this coarse-to-fine hierarchy

to MIBFV, we use time to run an animation of the advected noise. MIBFV has the advantage that it allows visualizing separate coarse and fine patterns in the same display. The price for this freedom is the AMG pre-processing This is roughly similar to ADFV, which must restart the diffusion after passing its parameter changes to its diffusion tensor.

Both IBFV and ADFV and can handle time dependent vector fields. For MIBFV, this is also possible, if all data are available in advance rather than being streamed from some external source. Although the preprocessing time for the noise frames is high, these can be stored, or streamed, on the graphics card, just as the original time-dependent vector fields are stored or streamed by the IBFV or ADFV.

If desired, MIBFV can combine more than two animation layers. In practice, the resulting number of parameter combinations is rather confusing. Moreover, we believe that the bi-level animation, which combines one coarse with one fine level, achieves the best visual results.

Extending MIBFV to handle curved surfaces would be trivial. The noise textures must be synthesized exactly as explained here, as the AMG decomposition[2] works also for such curved surfaces. Next, these textures can be simply fed in the IBFV variant for curved surfaces described in Ref.[5]

Finally, MIBFV can be extended to synthesize noise dynamically. Rather than keeping pre-generated noise frames as textures, one could store the basis functions in textures and generate the noise frames on-the-fly during the animation. This dynamic noise synthesis would have the great advantage of allowing instant changes to the noise parameters, e.g. time frequency, biasing basis functions in a level-of-detail manner, and so on.

REFERENCES

1. J. J. van Wijk, "Image based flow visualization," *Computer Graphics (Proc. SIGGRAPH '01), ACM Press, pp. 263-279* , 2001.
2. M. Griebel, M. Rumpf, T. Preusser, M. A. Schweitzer, and A. Telea, "Flow field clustering via algebraic multigrid," *Proc. Visualization'04* , pp. 35–42, 2004.
3. J. J. van Wijk, "Spot noise texture synthesis for data visualization," in *Computer Graphics (SIGGRAPH '91 Proceedings)*, **25**, pp. 309–318, 1991.
4. B. Cabral and L. C. Leedom, "Imaging vector fields using line integral convolution," *Computer Graphics (Proc. SIGGRAPH '93), pp. 263-279* , 1993.
5. J. J. van Wijk, "Image based flow visualization for curved surfaces," *Proc. Visualization'03* , pp. 123–130, 2003.
6. A. Telea and J. J. van Wijk, "3d ibfv: Hardware-accelerated 3d flow visualization," *Proc. Visualization'03* , pp. 233–240, 2003.
7. R. Laramee, B. Jobard, and H. Hauser, "Image space based visualization of unsteady flow on surfaces," *Proc. Visualization'03* , pp. 131–138, 2003.
8. B. Jobard, G. Erlebacher, and Y. Hussaini, "Hardware-accelerated texture advection," *Proc. Visualization'00* , pp. 155–162, 2000.
9. B. Jobard, G. Erlebacher, and Y. Hussaini, "Lagrangian-eulerian advection for unsteady flow visualization," *Proc. Visualization'01* , pp. pp. 439–446, 2001.
10. B. Jobard, G. Erlebacher, and Y. Hussaini, "Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization," *IEEE TVCG* **8**(3), pp. 211–222, 2002.
11. R. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. Post, and D. Weiskopf, "The state of the art in flow visualization: Dense and texture-based techniques," *Computer Graphics Forum* **23**(2), pp. 775–792, 2004.
12. T. Preußer and M. Rumpf, "Anisotropic nonlinear diffusion in flow visualization," *Proc. Visualization'99* , pp. 325–332, 1999.
13. U. Diewald, T. Preußer, and M. Rumpf, "Anisotropic diffusion in vector field visualization on euclidean domains and surfaces," *IEEE TVCG* **6**(2), pp. 139–149, 2000.
14. D. Bürkle, T. Preußer, and M. Rumpf, "Transport and anisotropic diffusion in time-dependent flow visualization," in *Proc. Visualization'01*, pp. 61–67, IEEE CS Press, 2001.
15. H. Garcke, T. Preusser, M. Rumpf, A. Telea, U. Weikard, and J. J. van Wijk, "A phase field model for continuous clustering on vector fields," *IEEE TVCG* **7**(3), pp. 230–241, 2000.