Pacific Graphics 2017
J. Barbic and O. Sorkine-Hornung
(Guest Editors)

Volume 36 (2017), Number 7

Efficient Gradient-Domain Compositing Using an Approximate Curl-free Wavelet Projection

Xiaohua Ren[1] , Luan Lyu[1], Xiaowei He[2], Yanci Zhang[†3] and Enhua Wu[‡1,2,4]

[1]University of Macau, [2]State Key Lab. of CS, ISCAS & Univ. of CAS, [3] Sichuan University,
[4]Zhuhai-UM Science and Technology Research Institute

Figure 1: *Red rock: A* 19588×4457 *(83-megapixel) panorama (top row) from 9 photos produced by our curl-free wavelet projection in* $26.11s$ *on CPU and* $0.45s$ *on GPU. The bottom row is the extracted wavelet coefficients by our method. (Data is courtesy of Aseem Agarwala.)*

Abstract

Gradient-domain compositing has been widely used to create a seamless composite with gradient close to a composite gradient field generated from one or more registered images. The key to this problem is to solve a Poisson equation, whose unknown variables can reach the size of the composite if no region of interest is drawn explicitly, thus making both the time and memory cost expensive in processing multi-megapixel images. In this paper, we propose an approximate projection method based on biorthogonal Multiresolution Analyses (MRA) to solve the Poisson equation. Unlike previous Poisson equation solvers which try to converge to the accurate solution with iterative algorithms, we use biorthogonal compactly supported curl-free wavelets as the fundamental bases to approximately project the composite gradient field onto a curl-free vector space. Then, the composite can be efficiently recovered by applying a fast inverse wavelet transform. Considering an n-pixel composite, our method only requires $2n$ *of memory for all vector fields and is more efficient than state-of-the-art methods while achieving almost identical results. Specifically, experiments show that our method gains a 5x speedup over the streaming multigrid in certain cases.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms

1. Introduction

Gradient-domain compositing is one of the most important techniques in image processing, which has been widely used in applications such as seamless cloning [PGB03,JSTS06,FHL*09], seamless stitching [Aga07, YHLX13, LDLM15], gradient domain painting [MP08]. Its basic idea is to find a composite image whose gradients best match a composite gradient field extracted from two or more registered images. This matching process is actually equivalent to solving a Poisson equation. Although there exist a lot of algorithms that can be used to efficiently solve the Poisson equation, gradient-domain compositing for large images still remains a challenge. One reason is that direct solvers like Cholesky factorization and Gaussian elimination become impractical for large linear system of equations. Iterative solvers like conjugate gradients are applicable to large sparse systems, but would require many iterations to get the desired solution if no preconditioning technique, which is usually non-parallelizable, is used. Another reason is that efficient iterative solvers like traditional multgrid consume $8/3n$ memory for two dimensions. As the number of pixels increases, solving the linear system entirely in-core quickly becomes impossible.

According to Helmholtz-Hodge decomposition, any sufficiently smooth vector field can be decomposed into the sum of a curl-free

† yczhang@scu.edu.cn
‡ ehwu@umac.mo

vector field and a divergence-free vector field for a simply connected domain (like an image). Therefore, we can reformulate the Poisson equation as a projection problem which projects the composite gradient field onto a curl-free vector space. The key now becomes how can we find an appropriate curl-free basis for the curl-free space so that the projection is both lightweight and efficient. As we know, the most commonly used curl-free basis is the orthogonal basis based on cosine functions, which can lead to an accurate Poisson-equation solution. Unfortunately, cosine functions are global, and the complexity is $O(n\log n)$. Its total computation cost can be twice that of the modern iterative methods according to [McC08, BCCZ08].

Considering the Poisson-equation solution will be rounded off to integer numbers ranging from 0 to 255 in image applications, it is actually not necessary to get a solution with high accuracy. This motivates us to propose an approximate projection method to solve the Poisson equation. Inspired by works in fluid dynamics [Urb00, DP06], we use spline functions to form compactly supported biorthogonal curl-free bases for the projection. Then, we show that the projection can be highly parallelized on GPU with only a small extra memory cost by applying the lifting wavelet transform [SS96]. Compared to an exact projection based on orthogonal bases, our approximate projection based on biorthogonal curl-free bases is sufficient to get almost identical results for gradient-domain compositing with the following advantages

- Our approximate projection has an $O(n)$ time-complexity, which only consists of three fast wavelet transforms.
- By applying the lifting wavelet transform, our method only needs $2n$ of memory for an n-pixel composite, making it possible to process larger in-core images.
- Our method is highly parallelizable, which can be fully exploited on modern GPUs to achieve an order of magnitude speedup over the CPU implementation.

2. Related Work

The Poisson equation arises from many areas and a lot of practical algorithms are available to solve this problem. Direct methods such as Cholesky decomposition and Gaussian elimination are very accurate at solving small-scale problems. However, for large-scale problems, iterative methods are better choice, in items of computation time and memory storage (e.g., direct methods would need extra storage for the factored matrix). Simple iterative methods like Jacobi and Gauss-Seidel are efficient for each iteration, but they require many iterations to remove low-frequency errors. Jeschke et al. [JCW09] point out that the convergence rate of the Jacobi method can be accelerated by appropriately choosing the right stencil size. Alternatively, the number of iterations can be greatly reduced by applying a multigrid scheme [BHM00], which has an $O(n)$ time cost and $8/3n$ memory cost for 2D cases. The multigrid method has also been widely used in gradient-domain compositing, such as high dynamic range compression [FLW02] and real-time painting [MP08]. Unfortunately, traditional multigrid methods require multiple V-cycles, which are inefficient for solving large linear systems with out-of-core data. Kazhdan and Hoppe [KH08] address this problem by proposing a streaming multigrid solver, which needs just two sequential passes over out-of-core data. Their method is later extended to handle spherical images in [KH10] and to run on a distributed computer cluster with multiple nodes in [KSH10].

Besides the multigrid methods, conjugate gradient methods also converge much faster than the Jacobi or Gauss-Seidel methods, especially when a preconditioner is also used. Pérez et al. [PGB03] use the preconditioned conjugate gradient method for seamless cloning and Agarwala et al. [ADA*04] for gradient-domain compositing. Later, Agarwala [Aga07] solves a reduced linear system by exploiting the fact the difference between a simple color composite and its associated gradient-domain composite is largely smooth. Szeliski et al. [SUS11] propose a similar technique using low-dimensional B-splines to represent the offset field. Similarly, Farbman et al. [FHL*09] use mean value coordinates (MVC) defined over an adaptive triangulation of the cloned region to interpolate the offset field at the region boundary or the seams. To accelerate the convergence rate, Szeliski [Sze90] proposes to use hierarchical basis functions as the preconditioner, which is later improved in [Sze06]. Although these two methods are efficient at solving the Poisson equation, they typically require more memory usage as discussed in [Aga07]. Besides, parallelizing the preconditioning step is usually not an easy task (e.g., preconditioning via Gauss-Seidel steps requires advanced techniques).

Over the last decades, wavelets have been widely used in image processing, such as denoising, compression, fusion, etc. But there are very few works that apply wavelets to improve the efficiency of gradient-domain compositing. Burt and Adelson [BA83] first propose to use multiresolution B-splines to hide the seams at different scales. Urban [Urb00] constructs curl-free wavelets to supplement the divergence-free wavelets [LR92]. With both the curl-free and divergence-free wavelets, Deriaz and Perrier [DP06] propose a novel iterative algorithm to decompose a general vector field into the sum of a curl-free part and a divergence-free part by alternatively performing curl-free and divergence-free projections. The problem with their work is that it is only applicable to problems with periodic boundary conditions. Manson et al. [MPS08] propose a wavelet method to reconstruct the indicator function of a solid from an oriented point cloud. Farbmann et al. [FFL11] introduced a pyramidal convolution approach to solve linear translation-invariant problems with $O(n)$ time cost and $8/3n$ memory cost. Recently, Edge-avoiding wavelets are constructed using the lifting scheme in [Fat09] to avoid the difficulties in solving large and poorly-conditioned systems of equations. Later, this method is extended by applying the À-Trous wavelet transform [DSHL10, HDL11]. Compared to the lifting wavelet transform, the À-Trous wavelet transform has $s \cdot n$ extra memory cost with s being the number of scales to transform. Note that edge-voiding wavelets are designed to process scalar fields, we focus on handling vector fields.

3. Background

In this section, we first review the problem of gradient-domain compositing and its solution techniques. Then, we present a solver based on curl-free cosine functions. Finally, we briefly discuss its disadvantages, which motivate us to use more general curl-free bases to overcome them.

Notation. Scalars appear in lower case: x and vectors in bold lower case: $\mathbf{x}$. $\langle \mathbf{u}, \mathbf{v} \rangle$ and $\mathbf{uv}$ denote inner product and the component-wise multiplication of $\mathbf{u}$ and $\mathbf{v}$, respectively. A vector field whose curl is zero is called a curl-free field, which can be represented as the gradient of a scalar field p, i.e. $\mathbf{u}_c = \nabla p$. We denote $\mathrm{L}^2(\Omega)$ the vector space of all *scalar* functions $f(x) : \Omega \to \mathbb{R}$ of finite energy and $\mathbf{L}^2(\Omega) := \mathrm{L}^2 \times \mathrm{L}^2$ the vector space of all *vector* fields over Ω. Both spaces are equipped with the Euclidean norm. The curl-free space denoted by $\mathbf{H}^0_c(\Omega)$ is the subspace formed by all the curl-free fields in $\mathbf{L}^2(\Omega)$. $\mathbb{R}$, $\mathbb{N}$ and $\mathbb{Z}$ are used to denote the set of all real numbers, natural numbers and integers, respectively. A 2^j-scaled and k-shifted function of $f(x)$ is written as $f_{j,k} := f(2^j x - k)$ for $j,k \in \mathbb{Z}$.

3.1. Gradient-domain compositing

In gradient-domain compositing, a composite vector field $\mathbf{u} \in \mathbf{L}^2(\Omega)$ is generated by copying and blending the gradients of one or more registered images, which may not be a curl-free or conservative field. Our purpose is to find $\mathbf{u}_c \in \mathbf{H}^0_c(\Omega)$ that is closest to $\mathbf{u}$ by solving the minimization problem:

$$\min_{\mathbf{u}_c \in \mathbf{H}^0_c(\Omega)} \frac{1}{2} \int_\Omega \|\mathbf{u} - \mathbf{u}_c\|^2. \tag{1}$$

One solution of the problem is to solving a Poisson equation. Substituting $\mathbf{u}_c = \nabla p$ and using the Euler-Lagrange equation yield

$$\Delta p = \nabla \cdot \mathbf{u} \tag{2}$$

After discretizing the Poisson equation, direct or iterative solvers can be used to solve it.

Another solution is by projecting $\mathbf{u}$ onto $\mathbf{H}^0_c(\Omega)$, if orthogonal bases of $\mathbf{H}^0_c(\Omega)$ are given. As an example, we use the well-known curl-free cosine functions to illustrate the basic idea.

3.2. A solver using curl-free cosine functions

Curl-free cosine functions. It's known that the set of cosine functions $\{\varphi_{\mathbf{k}} = \cos(k_1 x)\cos(k_2 y)\}$ is an orthogonal basis of $\mathrm{L}^2(\Omega = [0,\pi]^2)$ with Neumann BCs, where $\mathbf{k} = [k_1, k_2]^{\mathrm{T}} \in \mathbb{N}^2$. Then, the curl-free cosine functions are defined as

$$\mathbf{\Phi}_{c,\mathbf{k}} = \nabla \varphi_{\mathbf{k}} = -\mathbf{k}\mathbf{\Phi}_{\mathbf{k}}, \tag{3}$$

where

$$\mathbf{\Phi}_{\mathbf{k}} = \begin{bmatrix} \sin(k_1 x)\cos(k_2 y) \\ \cos(k_1 x)\sin(k_2 y) \end{bmatrix}. \tag{4}$$

Finally, we have $\{\mathbf{\Phi}_{c,\mathbf{k}}\}$ and $\{\mathbf{\Phi}_{\mathbf{k}}\}$, which are orthogonal bases of $\mathbf{H}^0_c(\Omega)$ and $\mathbf{L}^2(\Omega)$, respectively.

Curl-free cosine projection. With these bases, we take three steps to obtain the closest $\mathbf{u}_c$ of $\mathbf{u}$ as well as the scalar field p. First, the coefficients $\tilde{\mathbf{u}} = [\tilde{u}_x, \tilde{u}_y]^{\mathrm{T}}$ of $\mathbf{u} = \sum_{\mathbf{k}} \tilde{\mathbf{u}}(\mathbf{k})\mathbf{\Phi}_{\mathbf{k}}$ are computed by applying the sine and cosine transforms of $\mathbf{u}$. Then, the coefficients $\tilde{u}_c$ of $\mathbf{u}_c = \sum_{\mathbf{k}} \tilde{u}_c(\mathbf{k})\mathbf{\Phi}_{c,\mathbf{k}}$ are computed by projecting $\tilde{\mathbf{u}}(\mathbf{k})$ onto $\mathbf{k}$:

$$\tilde{u}_c(\mathbf{k}) = -\frac{\langle \mathbf{k}, \tilde{\mathbf{u}}(\mathbf{k}) \rangle}{\|\mathbf{k}\|^2}. \tag{5}$$

Please refer to Appendix A for a derivation of the above equation. Finally, $\mathbf{u}_c$ and p are reconstructed by

$$p = \sum_{\mathbf{k}} \tilde{u}_c(\mathbf{k})\varphi_{\mathbf{k}}, \quad \mathbf{u}_c = \nabla p. \tag{6}$$

Discussion. Because curl-free cosine functions are orthogonal, the solution is exact. However, since sine/cosine functions are global, the cosine/sine transform is time-consuming, whose best time complexity is $O(n\log n)$ in case that the size of signal is power-of-two. In this paper, we explore to use *biorthogonal compactly supported* curl-free wavelets, whose transforms have $O(n)$ time complexity.

4. Our Solver using Curl-free Wavelets

Multiresolution analysis (MRA). A MRA of $\mathrm{L}^2(\mathbb{R})$ is a sequence of closed subspaces $\{\mathrm{V}_j = \mathrm{span}\{\varphi_{j,k}\}_{k\in\mathbb{Z}}\}_{j\in\mathbb{Z}}$ satisfying $\mathrm{V}_j \subset \mathrm{V}_{j+1}$ and some other properties defined in [Mal08], where $\varphi(x)$ is referred to as a *scaling function* of the MRA. Wavelet spaces $\mathrm{W}_j = \mathrm{span}\{\psi_{j,k}\}_{k\in\mathbb{Z}}$ are the complements such that $\mathrm{V}_{j+1} = \mathrm{V}_j \oplus \mathrm{W}_j$, where $\oplus$ indicates the direct sum. $\psi(x)$ is referred to as a *wavelet* of the MRA. Then we have the wavelet space decomposition $\mathrm{L}^2(\mathbb{R}) = \mathrm{V}_0 \oplus_{j\in\mathbb{N}} \mathrm{W}_j$. For simplicity we write $\{\varphi_{0,k}, \psi_{j,k}\}$ to denote the basis of $\mathrm{L}^2(\mathbb{R})$ generated by a MRA with associated scaling function and wavelet (φ, ψ).

By tensor-products of $\mathrm{L}^2(\mathbb{R}) = \mathrm{V}_0 \oplus_{j\in\mathbb{N}} \mathrm{W}_j$, the wavelet space decomposition of the 2D function space is $\mathrm{L}^2(\mathbb{R}^2) = \mathrm{V}_{\mathbf{0}} \oplus_{\mathbf{j_1}} \mathrm{W}_{\mathbf{j_1}} \oplus_{\mathbf{j_2}} \mathrm{W}_{\mathbf{j_2}} \oplus_{\mathbf{j_3}} \mathrm{W}_{\mathbf{j_3}}$, where $\mathbf{0} = (0,0), \mathbf{j}_1 = (j_1, 0), \mathbf{j}_2 = (0, j_2), \mathbf{j}_3 = (j_1, j_2)$ and $j_1, j_2 \in \mathbb{N}$. The base functions of $\mathrm{V}_{\mathbf{0}} = \mathrm{span}\{\varphi_{\mathbf{0},\mathbf{k}}\}$ and $\mathrm{W}_{\mathbf{j_e}} = \mathrm{span}\{\psi_{\mathbf{j}_e,\mathbf{k}}\}$ for $e \in \{1,2,3\}$ are defined by tensor-products of $\varphi_{j,k}, \psi_{j,k}$:

$$\begin{aligned} \varphi_{\mathbf{0},\mathbf{k}} &= \varphi_{0,k_1}\varphi_{0,k_2}, \psi_{\mathbf{j}_1,\mathbf{k}} = \psi_{j_1,k_1}\varphi_{0,k_2}, \\ \psi_{\mathbf{j}_2,\mathbf{k}} &= \varphi_{0,k_1}\psi_{j_2,k_2}, \psi_{\mathbf{j}_3,\mathbf{k}} = \psi_{j_1,k_1}\psi_{j_2,k_2}. \end{aligned} \tag{7}$$

Similarly, we write $\{\varphi_{\mathbf{0},\mathbf{k}}, \psi_{\mathbf{j}_e,\mathbf{k}}\}$ to denote the basis of $\mathrm{L}^2(\mathbb{R}^2)$ formed by above base functions.

Notice that curl-free cosine functions are actually the gradients of cosine functions. In one dimension, the derivative of $\{\cos(kx)\}$ is $\{-k\sin(kx)\}$ and either of them forms a basis of $\mathrm{L}^2([0,\pi])$ with corresponding BCs. So in order to construct more general curl-free wavelet bases, a crucial step is to find two bases linked by differentiation like cosine and sine functions. The existence of such pairs of bases is guaranteed by Lemarié-Rieusset's proposition in [LR92]. The following English version is borrowed from [DP09].

Proposition: Let $\{V_j^1\}$ be a MRA of $\mathrm{L}^2(\mathbb{R})$, with associated scaling function and wavelet (φ^1, ψ^1). Then, there exists a MRA $\{V_j^0\}$ of $\mathrm{L}^2(\mathbb{R})$, with associated scaling function and wavelet (φ^0, ψ^0), satisfying:

$$(\varphi^1)'(x) = \varphi^0(x) - \varphi^0(x-1), \tag{8a}$$

$$(\psi^1)'(x) = 4 \cdot \psi^0(x). \tag{8b}$$

Famous pairs of (φ^1, ψ^1) and (φ^0, ψ^0) satisfying Equations (8a, 8b) are B-splines of degree n and $n-1$, with corresponding wavelets of *vanishing moment* m and $m+1$. These scaling functions and

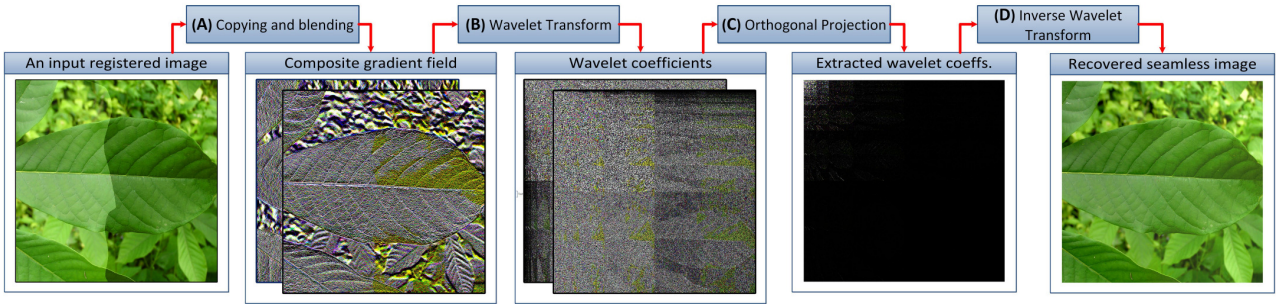

Figure 3: *An illustration of our curl-free wavelet projection method to find a seamless composition. From left to right: Given a registered image, (**A**) a composite gradient field is first generated by copying and blending the gradients of the image. (**B**) This field is then transformed into the wavelet domains, where wavelet coefficients are orthogonally projected onto the curl-free wavelets to extract the wavelet coeffs. of the seamless composition (**C**). Finally, a fast inverse wavelet transform of the extracted coeffs. is taken to recover the composition (**D**).*

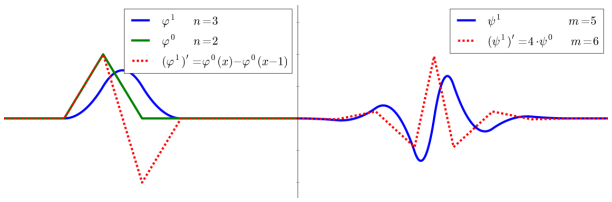

Figure 2: *An illustration of Lemarié-Rieusset's proposition. Left: The derivative (dashed red) of a B-spline of degree n (blue) can be expressed as the difference of two shifted B-splines of degree $n-1$ (green). Right: The derivative of the former's wavelet with m vanishing moments (blue) is equal to the latter's wavelet with $m+1$ vanishing moments (dashed red) up to a multiplier 4 (not scaled for display). As an example, we plot cdf3.5 and cdf2.6.*

wavelets denoted by cdf$(n+1).m$ were constructed by Cohen-Daubechies-Feauveau in [CDF92]. For example, cdf3.5 and cdf2.6 satisfy Equations (8a, 8b), whose scaling functions are 2nd-order (quadratic) and 1st-order (linear) B-splines, respectively, and corresponding wavelets have 5 and 6 vanishing moments, see Figure 2. In [KH08], the property of B-splines in Equation (8a) were used to derive the streaming multigrid to solve the Poisson equation (2), where 2nd-order B-splines were used. In contrast, we use the property of wavelets in Equation (8b) to derive a wavelet projection method without solving a linear system.

Curl-free wavelets. Let $\{\varphi^1_{0,k}, \psi^1_{j,k}\}$ and $\{\varphi^0_{0,k}, \psi^0_{j,k}\}$ be a pair of bases of $\mathrm{L}^2([0,1])$ with periodic BCs satisfying Lemarié-Rieusset's proposition. By Equation (7), we get a basis $\{\varphi^{\mathbf{1}}_{\mathbf{0},\mathbf{k}}, \psi^{\mathbf{1}}_{\mathbf{j}_e,\mathbf{k}}\}$ of $\mathrm{L}^2(\Omega=[0,1]^2)$ with periodic BCs, where $\mathbf{1}=(1,1)$. Taking the derivatives of the basis and using $(\varphi^1_{0,k})' = 0$, we get the curl-free wavelets

$$\begin{aligned} \Phi_{\mathrm{c},\mathbf{0},\mathbf{k}} &= \nabla\varphi^{\mathbf{1}}_{\mathbf{0},\mathbf{k}} = [0,0]^{\mathrm{T}}, \\ \Psi_{\mathrm{c},\mathbf{j_e},\mathbf{k}} &= \nabla\psi^{\mathbf{1}}_{\mathbf{j_e},\mathbf{k}} = 2^{\mathbf{j_e}}\Psi_{\mathbf{j_e},\mathbf{k}}, \end{aligned} \tag{9}$$

where $2^{\mathbf{j_1}} = [2^{j_1+2},0]^{\mathrm{T}}, 2^{\mathbf{j_2}} = [0,2^{j_2+2}]^{\mathrm{T}}, 2^{\mathbf{j_3}} = [2^{j_1+2},2^{j_2+2}]^{\mathrm{T}}$ and

$$\Psi_{\mathbf{j_1},\mathbf{k}} = \begin{bmatrix} \psi^0_{j_1,k_1}\varphi^1_{0,k_2} \\ 0 \end{bmatrix}, \Psi_{\mathbf{j_2},\mathbf{k}} = \begin{bmatrix} 0 \\ \varphi^1_{0,k_1}\psi^0_{j_2,k_2} \end{bmatrix}, \Psi_{\mathbf{j_3},\mathbf{k}} = \begin{bmatrix} \psi^0_{j_1,k_1}\psi^1_{j_2,k_2} \\ \psi^1_{j_1,k_1}\psi^0_{j_2,k_2} \end{bmatrix}. \tag{10}$$

Now we have obtained two wavelet bases $\{\Psi_{\mathrm{c},\mathbf{j_e},\mathbf{k}}\}$ and $\{\Psi_{\mathbf{j_e},\mathbf{k}}\}$ for $\mathbf{H}^0_{\mathrm{c}}(\Omega)$ and $\mathbf{L}^2(\Omega)$, respectively. Note that Equations (9, 10) are similar to Equations (3, 4). Figure 6 illustrates wavelet functions $\psi^{\mathbf{1}}_{\mathbf{j_3},\mathbf{k}}$ and $\Psi_{\mathbf{j_3},\mathbf{k}}$.

Curl-free wavelet projection $\mathbf{P}_{\mathrm{W}}$. We take the same three steps as the curl-free cosine projection to compute $\mathbf{u}_{\mathrm{c}}$ and p for $\mathbf{u} \in \mathbf{L}^2(\Omega)$. First, we compute the wavelet coefficients $\tilde{\mathbf{u}}$ of $\mathbf{u} = \sum_e \sum_{\mathbf{j}_e} \sum_{\mathbf{k}} \tilde{\mathbf{u}}(\mathbf{j}_e,\mathbf{k})\Psi_{\mathbf{j}_e,\mathbf{k}}$ by the wavelet transform. Second, we compute $\tilde{u}_{\mathrm{c}}$ by orthogonally projecting $\tilde{\mathbf{u}}(\mathbf{j}_e,\mathbf{k})$ onto $2^{\mathbf{j_e}}$:

$$\tilde{u}_{\mathrm{c}}(\mathbf{j}_e,\mathbf{k}) = \frac{\langle 2^{\mathbf{j_e}}, \tilde{\mathbf{u}}(\mathbf{j}_e,\mathbf{k})\rangle}{\|2^{\mathbf{j_e}}\|^2}. \tag{11}$$

Please refer to Appendix A for a derivation of the above equation. Finally, $\mathbf{u}_{\mathrm{c}}$ and p are reconstructed by

$$p = \sum_{k} \tilde{p}(\mathbf{0},\mathbf{k})\varphi^{\mathbf{1}}_{\mathbf{0},\mathbf{k}} + \sum_e \sum_{\mathbf{j}_e} \sum_{\mathbf{k}} \tilde{u}_{\mathrm{c}}(\mathbf{j}_e,\mathbf{k})\psi^{\mathbf{1}}_{\mathbf{j}_e,\mathbf{k}}, \quad \mathbf{u}_{\mathrm{c}} = \nabla p. \tag{12}$$

Note that $\tilde{p}(\mathbf{0},\mathbf{k}) = 0$ if Neumann/periodic BCs are imposed. We will discuss how to set this value in the next section. Figure 3 gives an intuitive description of our curl-free wavelet projection for gradient-domain compositing.

Biorthogonality. Unlike curl-free cosine functions, compactly supported curl-free wavelets are biorthogonal bases. We cannot get curl-free wavelets that are both compactly supported and orthogonal [LR92, DP09]. Although the loss of orthogonality can lead to a sacrifice of accuracy, the gain of the compact support property is $O(n)$ time to compute the wavelet transform. Considering the discretization error is $1/256 \approx 4\cdot 10^{-3}$ in 8-bit/channel image compositing, it's worth making such a tradeoff between accuracy and time. Our experiments show that the accuracy is sufficient for gradient-domain compositing.

Time and Memory Complexity. The first and third steps are the most time-consuming in $\mathbf{P}_{\mathrm{W}}$, which compute two forward wavelet transforms of $\mathbf{u} = [u_x,u_y]^{\mathrm{T}}$ and one inverse wavelet transform of $\tilde{u}_{\mathrm{c}}$ to recover p. The wavelet transform is actually the convolution of the input and the corresponding wavelet filters $\{h,g\}$. Because we use compactly supported wavelets, the size of filters is small. For example, cdf3.5 has 4 and 12 coefficients for h and g, respectively. So the time complexity of the transform is $O(n)$. Furthermore, Daubechies and Sweldens [DS98] show that any discrete wavelet

transform can be decomposed into a finite sequence of simple filtering steps, called lifting steps. Their method called the lifting wavelet transform not only reduces the computational complexity of the convolution-based wavelet transform by a factor of two, but also can be taken in place with constant memory.

Thanks to the lifting wavelet transform, only two pieces of memory (each with size n) are needed, which are first used to store each component of $\mathbf{u}$. After being transformed in place, one of them is reused to store extracted wavelet coefficients $\tilde{u}_c$, which is finally transformed in place to recover p. Thus, the memory cost is $2n$.

Parallelism. Since the size of wavelet filters is small and the computation of each wavelet coefficient is independent, the wavelet transform (also the lifting wavelet transform) is highly parallelizable [TSP*08, vdLJR11]. Experiments show that our GPU implementation can achieve $\sim 20x$ speedup.

Boundary conditions. In gradient-domain compositing, Neumann BCs are commonly used. But it's also reasonable to use periodic BCs and Lemarié-Rieusset's proposition is applicable to $L^2[0,1]$ with periodic BCs. In next section, we will introduce the implementation of our method and the support for Neumann BCs.

5. Implementation

The procedure of the curl-free wavelet projection is the same as that of the curl-free cosine projection. Algorithm 1 is an outline of our implementation. The key part of the algorithm is the *standard* two-dimensional wavelet transform [SDS95]. To obtain the standard wavelet transform of a two-dimensional signal, we first apply the one-dimensional wavelet transform to each row of the signal (row transform), and then to each column of the row-transformed signal (column transform). In the following, we briefly introduce the one-dimensional wavelet transform based on convolution, for the lifting wavelet transform we refer the reader to the excellent course [SS96].

One-dimensional wavelet transform. Let $(\tilde{\varphi},\tilde{\psi})$ be the dual scaling function and dual wavelet of (φ,ψ). They satisfy the following two-scale relations

$$\begin{aligned}\varphi_{j-1,k} &= \sum_m h(m,k)\varphi_{j,2k+m}, \psi_{j-1,k} = \sum_m g(m,k)\varphi_{j,2k+m},\\ \tilde{\varphi}_{j-1,k} &= \sum_m \tilde{h}(m,k)\tilde{\varphi}_{j,2k+m}, \tilde{\psi}_{j-1,k} = \sum_m \tilde{g}(m,k)\tilde{\varphi}_{j,2k+m},\end{aligned} \tag{13}$$

where $\{h,g\}$ and $\{\tilde{h},\tilde{g}\}$ are called synthesis and analysis filters, respectively. We will denote by $\mathbf{H} = [h(m,k)]$ the matrix whose (m,k) entry is $h(m,k)$. By using block-matrix notation, we define the synthesis matrix as $\mathbf{S} = \left[\mathbf{H}|\mathbf{G}\right]$ and the analysis matrix as $\mathbf{A} = \left[\tilde{\mathbf{H}}|\tilde{\mathbf{G}}\right]^{\mathrm{T}}$, where $\mathbf{H} = [h(m,k)], \mathbf{G} = [g(m,k)]$ and $\tilde{\mathbf{H}} = [\tilde{h}(m,k)], \tilde{\mathbf{G}} = [\tilde{g}(m,k)]$. Given a one-dimensional signal $\mathbf{f}_j = [f_j(k)]^{\mathrm{T}}$, its one-scale wavelet transform and inverse are given by

$$\tilde{\mathbf{f}}_j = \begin{bmatrix}\mathbf{f}_{j-1}\\ \mathbf{d}_{j-1}\end{bmatrix} = \mathbf{A}\mathbf{f}_j, \tag{14a}$$

$$\mathbf{f}_j = \mathbf{S}\tilde{\mathbf{f}}_j = \mathbf{S}\begin{bmatrix}\mathbf{f}_{j-1}\\ \mathbf{d}_{j-1}\end{bmatrix}, \tag{14b}$$

where $\mathbf{f}_{j-1}$ and $\mathbf{d}_{j-1}$ are called approximate and wavelet coefficients, respectively. The full-scale wavelet transform of $\mathbf{f}$ is obtained by recursively performing Equation (14a) from $\mathbf{f}_j$ to $\mathbf{f}_1$. The inverse transform is done by recursively performing Equation (14b) from $\tilde{\mathbf{f}}_1$ to $\tilde{\mathbf{f}}_j$.

Constructions of A and S. Here we only introduce the construction of the synthesis matrix $\mathbf{S}$. The construction of $\mathbf{A}$ is similar. $\mathbf{S}$ consists of two blocks: $\mathbf{H}$ and $\mathbf{G}$. The assemblies of $\mathbf{H}$ and $\mathbf{G}$ depend on the domain and its boundary conditions over which scaling functions and wavelets are constructed. For the domain $\mathbb{R}$, $\mathbf{H}$ has a simple structure: The columns $h(\cdot,k)$ of $\mathbf{H}$ are shifted versions of each other, as are the columns $g(\cdot,k)$ of $\mathbf{G}$. For the interval $\mathbb{I} = [0,1]$ with periodic BCs, the $h(\cdot,k)$ are circular-shifted versions of each other, as are $g(\cdot,k)$. For $\mathbb{I}$ with Neumann/Dirichlet BCs, we need to pay attention to the columns that intersect with the boundaries. We use reflection and skew-reflection methods to enforce Neumann and Dirichlet BCs, respectively. More specifically, assume that $h(\cdot,0)$ and $g(\cdot,0)$ intersect with the left boundary of $\mathbb{I}$, the new filters are obtained by $\bar{h}(m,0) = h(m,0) + h(-m,0)$ for Neumann BCs and $\bar{h}(m,0) = h(m,0) - h(-m,0)$ for Dirichlet BCs. The same process is also applied to $g(\cdot,0)$. In the additional files, we give the values of $\mathbf{S}$ and $\mathbf{A}$ of cdf3.5 with Neumann BCs and cdf2.6 with Dirichlet BCs for different scales.

Note on $\mathbf{P}_{\mathrm{W}}$. Here we give some details of the curl-free wavelet projection $\mathbf{P}_{\mathrm{W}}$ as listed in Algorithm 1. The input of the algorithm is a composite gradient field and the output is a composite. In Line 1, 2 and 4, WT2 (iWT2) represents the standard forward (resp. inverse) two-dimensional wavelet transform as described in [SDS95], in which the one-dimensional wavelet transform for each row and column transform has been introduced in the above paragraphs. We list the pseudocode of WT2 in Algorithm 3. The first parameter of WT2 (iWT2) is a two-dimensional signal and the other two are the wavelet types of the row and column transforms, respectively. In this paper, we use cdf3.5 and cdf2.6 as (φ^1,ψ^1) and (φ^0,ψ^0), respectively. Line 3 is the computation of curl-free wavelet coefficients $\tilde{u}_c$ as listed in Algorithm 2.

GPU Implementation. We implement both the convolution and lifting wavelet transforms on GPU with CUDA. We find that for small images (<0.5 megapixel), the former is 70% faster than the latter. But, for large images, the latter is $10 \sim 12\%$ faster than the former. The reason is that the size of the former's filters is larger than that of the latter's filters, which leads to more cache misses. On the other hand, the convolution transform needs one extra memory. We refer the reader to [TSP*08, vdLJR11] for more comprehensive comparisons of these two kinds of transforms on GPU.

Non-power-of-two images. For non-power-of-two images, we pad the input image of resolution $n = (n_x, n_y)$ using the boundary pixel values to a resolution $(2^{J_1}, 2^{J_2})$ with $J_1 = \lceil \log 2(n_x) \rceil$ and $J_2 = \lceil \log 2(n_y) \rceil$. Thus, the gradients of the padded pixels are zeros. In practice, we do not store them when preparing the composite gradient field $\mathbf{u}$. In the wavelet transform stage, we need additional memory to store wavelet coefficients of the padded parts of $\mathbf{u}$. However, since we use compactly supported wavelets, the size of the memory, which linearly depends on the size of filters $\{\tilde{h}_j, \tilde{g}_j\}$, is small.

Setting the image mean. For Neumann/periodic boundary conditions, the solution of the Poisson equation has an unconstrained

Algorithm 1 Curl-free wavelet projection: $\mathbf{P}_{\mathrm{W}}$

Input: $\mathbf{u} = [u_x, u_y]^{\mathrm{T}} \in \mathbf{L}^2([0,1]^2)$
Output: $p \in \mathrm{L}^2([0,1]^2)$
1: $\tilde{u}_x = \mathrm{WT2}(u_x, \mathrm{cdf2.6}, \mathrm{cdf3.5})$ ▷ *2D wavelet transform of* u_x
2: $\tilde{u}_y = \mathrm{WT2}(u_y, \mathrm{cdf3.5}, \mathrm{cdf2.6})$ ▷ *2D wavelet transform of* u_y
3: $\tilde{u}_{\mathrm{C}} = \mathbf{P}_{\mathrm{O}}(\tilde{\mathbf{u}})$ ▷ *Algorithm 2*
4: $p = \mathrm{iWT2}(\tilde{u}_{\mathrm{C}}, \mathrm{cdf3.5}, \mathrm{cdf3.5})$ ▷ *inverse 2D wavelet transform*

Algorithm 2 Orthogonal projection of $\tilde{\mathbf{u}}$ onto $2^{\mathbf{j}_e}$: $\mathbf{P}_{\mathrm{O}}$

Input: $\tilde{\mathbf{u}} = [\tilde{u}_x, \tilde{u}_y]^{\mathrm{T}}$ ▷ *wavelet coefficients of* $\mathbf{u}$
Input: $n = (n_x, n_y)$ ▷ *the size of* $\tilde{u}_c$
Output: $\tilde{u}_c$
1: $J = (\lceil \log 2(n_x) \rceil, \lceil \log 2(n_y) \rceil)$ ▷ $\lceil \cdot \rceil$ *is the ceil function.*
2: $\mathbb{J}_1 = \{0,1,...,J_1-1\}, \mathbb{J}_2 = \{0,1,...,J_2-1\}, \mathbb{O} = \{0\}$
3: $\mathbb{K}_{j_1} = \{0,1,...,2^{j_1}-1\}, \mathbb{K}_{j_2} = \{0,1,...,2^{j_2}-1\}$
▷ *In the following,* $\times$ *is the Cartesian product of two sets.*
4: $\mathcal{J}_1 = \mathbb{J}_1 \times \mathbb{O}, \mathcal{J}_2 = \mathbb{O} \times \mathbb{J}_2, \mathcal{J}_3 = \mathbb{J}_1 \times \mathbb{J}_2$
5: $\mathcal{K}_{\mathbf{j}_1} = \mathbb{K}_{j_1} \times \mathbb{O}, \mathcal{K}_{\mathbf{j}_2} = \mathbb{O} \times \mathbb{K}_{j_2}, \mathcal{K}_{\mathbf{j}_3} = \mathbb{K}_{j_1} \times \mathbb{K}_{j_2}$
6: **for** $e \in \{1,2,3\}$ **do**
7: **for** $\mathbf{j}_e \in \mathcal{J}_e$ **do**
8: $\tilde{u}_{\mathrm{C}}(\mathbf{j}_e, \mathbf{k}) = \frac{\langle 2^{\mathbf{j}_e}, \tilde{\mathbf{u}}(\mathbf{j}_e, \mathbf{k}) \rangle}{\|2^{\mathbf{j}_e}\|^2}$ for all $\mathbf{k} \in \mathcal{K}_{\mathbf{j}_e}$ ▷ *Equation (11)*
9: **end for**
10: **end for**

mean value, which leads to the coarsest coefficient $\tilde{p}(\mathbf{0}, \mathbf{k})$ in Equation (12) is zero. A common method is to explicitly add the average of the simply copy composition of registered images to the solution. Our approach is to set $\tilde{p}(\mathbf{0}, \mathbf{k}) = A \cdot (\sqrt{2})^{J_1+J_2}$ with A being the average. This is because the wavelet coefficients of a C-value constant image of size $(2^{J_1}, 2^{J_2})$ are all zeros except the coarsest coefficient whose value is $C \cdot (\sqrt{2})^{J_1+J_2}$.

6. Experimental Results

The method proposed in this paper has been implemented on a desktop PC with an Intel i7-3930K processor with 12G RAM and an Nvidia GTX 980 graphics card. All other algorithms used for comparison are also tested on the same machine.

Two different gradient fields are employed as the input for our method. Because we use second-order B-spline wavelets, the gradient is discretized on a staggered-grid like finite difference, see [KH08].

Copying Gradients. Similar to the approach of [KH08], we generate a composite gradient field by copying the gradients from the images and zeroing out the gradients across the seams. And then this field is employed as the input of Algorithm 1. One composite result generated by our method is shown in Figure 5, which demonstrates that our method works well for large exposure and hue variations in the registered photos. Another result is shown in Figure 1 to illustrate that our method works well for large images.

Mixing Gradients. The second type of input gradient field is generated by choosing the gradient values with largest absolute values among two or more images [PGB03]. The composition of this type field is more challenge than that of copying gradients since it may have complex structures. Two mixing gradient compositions shown in Figure 7 and Figure 8 demonstrate that our method works well. They are vivider than the simply-copy compositions.

Image name	RMS residual		RMS error		Max error	
	Ours	JM	Ours	SM	Ours	SM
Leaf	$6{\cdot}10^{-3}$	$6{\cdot}10^{-3}$	$2{\cdot}10^{-3}$	$2{\cdot}10^{-5}$	$2{\cdot}10^{-2}$	$4{\cdot}10^{-3}$
Flower	$4{\cdot}10^{-3}$	$4{\cdot}10^{-3}$	$3{\cdot}10^{-3}$	$2{\cdot}10^{-5}$	$2{\cdot}10^{-2}$	$6{\cdot}10^{-3}$
Leaf2	$8{\cdot}10^{-3}$	$8{\cdot}10^{-3}$	$3{\cdot}10^{-3}$	$1{\cdot}10^{-4}$	$3{\cdot}10^{-2}$	$4{\cdot}10^{-3}$
Paper flower	$5{\cdot}10^{-4}$	$3{\cdot}10^{-3}$	$1{\cdot}10^{-3}$	$3{\cdot}10^{-3}$	$1{\cdot}10^{-2}$	$3{\cdot}10^{-2}$
PNC Park	$8{\cdot}10^{-3}$	$7{\cdot}10^{-3}$	$7{\cdot}10^{-3}$	$3{\cdot}10^{-3}$	$7{\cdot}10^{-2}$	$1{\cdot}10^{-2}$
Edingburgh	$4{\cdot}10^{-3}$	NA	$2{\cdot}10^{-3}$	$5{\cdot}10^{-3}$	$4{\cdot}10^{-2}$	$2{\cdot}10^{-2}$
Redrock	$3{\cdot}10^{-3}$	NA	NA	NA	NA	NA

Table 2: *Error and residual statistics of our method. The last image is too large to practically obtain the ground truth by PCG.*

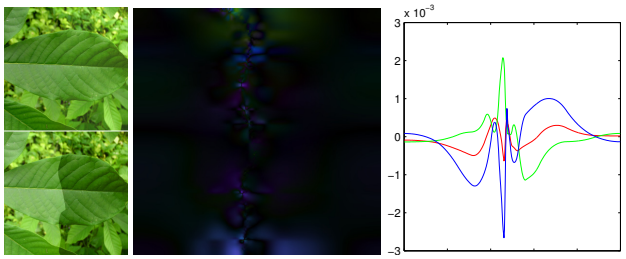

Figure 4: *Error distribution of the composition of a leaf. Middle: The* 60x*-magnified absolute errors of the generated seamless image (left top). Right: The middle row of the errors. The lines of the errors of RGB channels are marked using the corresponding color.*

Now we demonstrate our method can achieve almost identical results by numerical error analysis and visual comparisons.

Accuracy Analysis. A theoretical discussion on the reason why our method provides approximate solutions is given in Appendix A. Here we give a numerical accuracy analysis. We use two measures to numerically evaluate the accuracy: the relative residual $\|\nabla \cdot \mathbf{u} - \Delta p\| / \|\nabla \cdot \mathbf{u}\|$ of the Poisson equation and the root-mean-square (RMS) and maximal errors compared to ground truth solutions solved by PCG with a tolerance factor of 10^{-12} when the sizes of images are suitable.

We run a number of stitching examples. Our residuals and solution errors are listed in Table 2. As seen in the table, most of results can achieve a RMS error around 4×10^{-3} and maximal error around 10^{-2}. Note that these errors are low frequency and actually the difference of brightness. Figure 4 shows the error distribution of the leaf case. In the right part of the figure, the middle row of the errors is plotted, which illustrates that the errors are low-frequency. Figure 8 shows that our method can achieve visually identical results for mixing gradient compositions. We refer the reader to our additional materials for more visual comparisons of our results to the ground truth solutions.

Now we demonstrate the efficiency of our algorithm. Two other algorithms are employed as comparison basis. The first is the CPU-based streaming multigrid (SM) method [KH08] which uses

Image name	Size (MP)	Mem. (MB)			CPU-Ver. Time (s)			GPU-Ver. Time (ms)	
		Ours (Data + Solver)	SM (in)	SM (out)	Ours (I/O + Solver)	SM (in)	SM (out)	Ours	JM
Leaf	1.0	4 + 4 = 8	69	83	0.11 + 0.13 = 0.24	0.6	1.43	6	9
Flower	1.7	7 + 7 = 14	107	123	0.17 + 0.23 = 0.40	1.11	2.26	10	41
Leaf2	2.0	8 + 8 = 16	135	153	0.25 + 0.30 = 0.55	2.29	3.38	12	39
Paper flower	9.4	37 + 38 = 75	564	169	1.18 + 1.38 = 2.56	5.93	8.82	49	126
PNC Park	27.3	109 + 111 = 220	1623	130	3.75 + 3.97 = 7.72	18.37	23.68	142	189
Edingburgh	47.8	189 + 196 = 385	2853	216	7.26 + 7.44 = 14.7	32.56	33.06	254	NA
Redrock	83.3	333 + 337 = 670	5003	143	14.78 + 11.33 = 26.11	52.17	68.43	451	NA

Table 1: *A comparison of memory and run-time performance of our CPU version to the (in)-core and (out)-of-core streaming multigrid (SM) with one V-cycle, and our GPU version to MaCann and Pollard's multigrid (JM). The results of MaCann and Pollard's mutigrid of last two cases are not available since their method was implemented by OpenGL by which the maximal size of textures supported is* 8196×8196.

second-order B-splines for the Poisson equation and Gauss-Seidel as the smoother thus has high convergence rate, and the second method is the GPU-based multigrid (JM) proposed by Macann and Poallard [MP08], who use Jacobi as the smoother for its high parallelism. All performance data, including the memory cost and running time are listed in Table 1.

Comparison to SM. The SM method has in-core and out-of core versions. The speed of the in-core SM is better than that of the out-core SM, but at high memory cost. For a fair comparison, both SM and our method use 16-bit floats to store the composition gradient fields and run on a single CPU core. Our method has average **2.6x** and **4.1x** speedups over the in-core and out-of-core SM, respectively. As shown in the Table 1, our method has the fastest speed and lowest memory cost (including data term) for normal size (<20 megapixel) images. Our solver time is approximately the same as the I/O time. For very large images, the out-of-core SM has a better tradeoff between the time and memory cost. On the other hand, as shown in Table 2, SM has better precision than ours, thus is more suitable for applications with higher accuracy requirements.

In [KSH10], the SM is to run on a distributed computer cluster and parallelized using multiple threads within each node while preserving the same precision. The distributed SM can achieve linear speedup versus the number of nodes. As reported in their work, for example, 4.6x and 6.5x speedups are achieved on the Edingburgh and Redrock datasets, respectively, on a 4-node computer cluster with each node equipped an 8-core CPU. Currently, our method is parallelized using GPU and achieves $\sim 20x$ speedup on a modern GPU.

Comparison to JM. MaCann and Pollard's multigrid (JM) solver is a variant of standard GPU multigrid solvers such as [BFGS03] customized for gradient-domain painting thus suitable for gradient-domain compositing, which has no pre-smoothing and 2 post-smoothing steps per V-cycle. To make a fair comparison, we run their solver until convergence to a solution with a comparable residual like ours for each case, whose values we choose are listed in Table 2. Our method achieve average **2.5x** speedup over their method.

7. Conclusion and Future Work

We have introduced a fast approximate curl-free wavelet projection method for gradient-domain compositing that outperforms the state-of-the-art methods both on CPU and GPU. Experiments show that it is a competitive tool for gradient-domain compositing of 8-bit/channel images. However, our method is not applicable to gradient-domain applications with a high requirement of accuracy or with irregular boundaries, such as gradient-domain high dynamic range (HDR) compression [FLW02], which requires high precision solutions otherwise "halo" artifacts will appear, and image cloning [PGB03] which has arbitrary boundaries. Since our solver is an in-core solver, the memory becomes the bottleneck when processing gigapixel images. In this case, an out-of-core solver like the streaming multigrid or its distributed version is a good choice.

There are two directions to extend our method. The first one is

Figure 5: *Example result of copying gradient composition. Top: A 7963x3589 (27-megapixel) panorama from 7 photos, obtained by our method. Bottom: Close-ups, comparing our result to the simply-copy composition. (Data is courtesy of Michael Kazhdan.)*

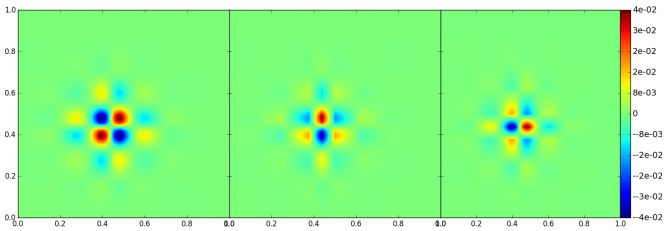

Figure 6: *An illustration of 2D wavelet functions. Left:* $\psi^{\mathbf{1}}_{\mathbf{j_3},\mathbf{k}}$*; Middle and Right: the x- and y-component of* $\mathbf{\Psi}_{\mathbf{j_3},\mathbf{k}}$.

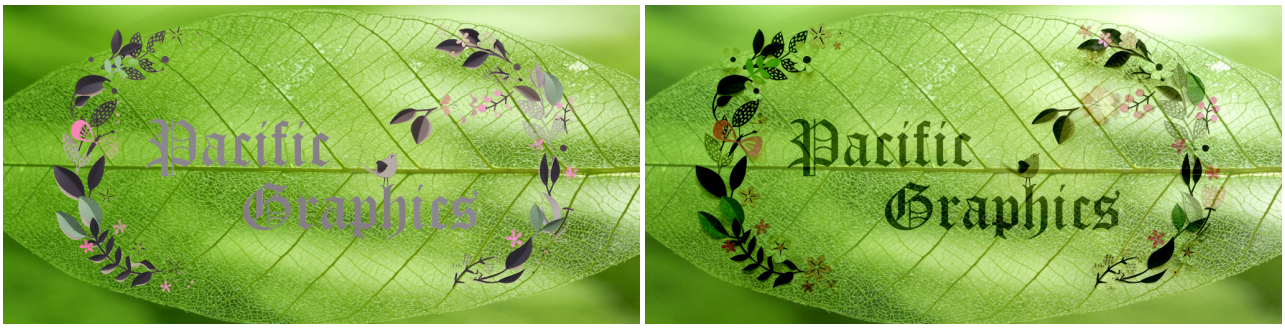

Figure 7: *A mixing gradient composition (2048x1024) of a "pacific graphics" image with a leaf solved by our method. Note that the mixing gradient composition (right) is vivider than the simply-copy composition (left).*

Figure 8: *A mixing gradient composition (1024x1024) of a paper-flower image (left top) with a leaf (left middle) solved by our method (middle column) and by PCG as the ground truth (right column). The simply-copy composition is shown in the bottom of the left column. Our solution is visually identical to the ground truth.*

to improve the accuracy and apply it to HDR compression. The other one is to extend our in-core version to out-of-core to support gigapixel image processing. Our method is efficient for out-of-core data since it needs only one projection and thus does not require multiple accesses over out-of-core data like iterative methods. However, one challenge is that an out-of-core transposition of the data would be required when performing the wavelet transform, which is expensive. We are also interested in applying our method to real-time gradient-domain painting.

Acknowledgements

The authors would like to thank the anonymous reviewers for their constructive comments and Wei Cao for preparing the images. The work is supported by the Macao Science and Technology Development Fund (FDCT: 068/2015/A2, 136/2014/A3), NSFC (61672502, 61632003, 6140051239, 61472261, 61502109), University of Macau Research Fund (MYRG2014-00139-FST), National High Technology Research and Development Program of China (2015AA016405), and Natural Science Foundation of Guangdong Province (2016A030310342).

References

[ADA*04] AGARWALA A., DONTCHEVA M., AGRAWALA M., DRUCKER S., COLBURN A., CURLESS B., SALESIN D., COHEN M.: Interactive digital photomontage. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 294–302. 2

[Aga07] AGARWALA A.: Efficient gradient-domain compositing using quadtrees. In *ACM SIGGRAPH 2007 Papers* (2007), SIGGRAPH '07. 1, 2

[BA83] BURT P. J., ADELSON E. H.: A multiresolution spline with application to image mosaics. *ACM Trans. Graph. 2*, 4 (Oct. 1983), 217–236. 2

[BCCZ08] BHAT P., CURLESS B., COHEN M., ZITNICK C. L.: Fourier analysis of the 2d screened poisson equation for gradient domain problems. In *Proceedings of the 10th European Conference on Computer Vision: Part II* (2008), ECCV '08, pp. 114–128. 2

[BFGS03] BOLZ J., FARMER I., GRINSPUN E., SCHRÖODER P.: Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. *ACM Trans. Graph. 22*, 3 (July 2003), 917–924. 7

[BHM00] BRIGGS W. L., HENSON V. E., MCCORMICK S. F.: *A Multigrid Tutorial: Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. 2

[CDF92] COHEN A., DAUBECHIES I., FEAUVEAU J.-C.: Biorthogonal bases of compactly supported wavelets. *Communications in Pure and Applied Mahtematics 45*, 5 (1992), 485–560. 4

[DP06] DERIAZ E., PERRIER V.: Divergence-free and curl-free wavelets in two dimensions and three dimensions:application to turbulent flows. *Journal of Turbulence 7*, 3 (Feb. 2006), 1–37. 2

[DP09] DERIAZ E., PERRIER V.: Orthogonal helmholtz decompostion in arbitrary dimension using divergence-free and curl-free wavelets. *Applied and Computational Harmonic Analysis 26*, 2 (Mar. 2009), 249–269. 3, 4

[DS98] DAUBECHIES I., SWELDENS W.: Factoring wavelet transforms into lifting steps. *Journal of Fourier Analysis and Applications 4*, 3 (1998), 247–269. 4

[DSHL10] DAMMERTZ H., SEWTZ D., HANIKA J., LENSCH H. P. A.: Edge-avoiding À-trous wavelet transform for fast global illumination filtering. In *Proceedings of the Conference on High Performance Graphics* (2010), HPG '10, pp. 67–75. 2

[Fat09] FATTAL R.: Edge-avoiding wavelets and their applications. In *ACM SIGGRAPH 2009 Papers* (2009), SIGGRAPH '09, pp. 22:1–22:10. 2

[FFL11] FARBMAN Z., FATTAL R., LISCHINSKI D.: Convolution pyramids. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 175:1–175:8. 2

[FHL*09] FARBMAN Z., HOFFER G., LIPMAN Y., COHEN-OR D., LISCHINSKI D.: Coordinates for instant image cloning. *ACM Trans. Graph. 28*, 3 (July 2009), 67:1–67:9. 1, 2

[FLW02] FATTAL R., LISCHINSKI D., WERMAN M.: Gradient domain high dynamic range compression. *ACM Trans. Graph. 21*, 3 (July 2002), 249–256. 2, 7

[HDL11] HANIKA J., DAMMERTZ H., LENSCH H.: Edge-optimized à-trous wavelets for local contrast enhancement with robust denoising. *Computer Graphics Forum 30*, 7 (2011). 2

[JCW09] JESCHKE S., CLINE D., WONKA P.: A gpu laplacian solver for diffusion curves and poisson image editing. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 116:1–116:8. 2

[JSTS06] JIA J., SUN J., TANG C.-K., SHUM H.-Y.: Drag-and-drop pasting. In *ACM Transactions on Graphics (TOG)* (2006), vol. 25, ACM, pp. 631–637. 1

[KH08] KAZHDAN M., HOPPE H.: Streaming multigrid for gradient-domain operations on large images. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 21:1–21:10. 2, 4, 6

[KH10] KAZHDAN M., HOPPE H.: Metric-aware processing of spherical imagery. *ACM Trans. Graph. 29*, 6 (Dec. 2010), 149:1–149:10. 2

[KSH10] KAZHDAN M., SURENDRAN D., HOPPE H.: Distributed gradient-domain processing of planar and spherical images. *ACM Trans. Graph. 29*, 2 (Apr. 2010), 14:1–14:11. 2, 7

[LDLM15] LU S.-P., DAUPHIN G., LAFRUIT G., MUNTEANU A.: Color retargeting: Interactive time-varying color image composition from time-lapse sequences. *Computational Visual Media 1*, 4 (Dec 2015), 321–330. 1

[LR92] LEMARIÉ-RIEUSSET P. G.: Analyses multi-résolutions non orthogonales, commutation entre projecteurs et dérivation et ondelettes vecteurs à divergence nulle. *Rev. Mat. Iberoamericana 8*, 2 (Mar. 1992), 221–237. 2, 3, 4

[Mal08] MALLAT S.: *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*, 3rd ed. Academic Press, 2008. 3

[McC08] MCCANN J.: Recalling the single-fft direct poisson solve. In *ACM SIGGRAPH 2008 Posters* (2008), SIGGRAPH '08, pp. 71:1–71:1. 2

[MP08] MCCANN J., POLLARD N. S.: Real-time gradient-domain painting. In *ACM SIGGRAPH 2008 Papers* (2008), SIGGRAPH '08, pp. 93:1–93:7. 1, 2, 7

[MPS08] MANSON J., PETROVA G., SCHAEFER S.: Streaming surface reconstruction using wavelets. In *Proceedings of the Symposium on Geometry Processing* (2008), SGP '08, pp. 1411–1420. 2

[PGB03] PÉREZ P., GANGNET M., BLAKE A.: Poisson image editing. *ACM Trans. Graph. 22*, 3 (July 2003), 313–318. 1, 2, 6, 7

[SDS95] STOLLNITZ E. J., DEROSE T. D., SALESIN D. H.: Wavelets for computer graphics: A primer, part 1. *IEEE Comput. Graph. Appl. 15*, 3 (May 1995), 76–84. 5

[SS96] SWELDENS W., SCHRÖDER P.: Building your own wavelets at home. In *Wavelets in Computer Graphics* (1996), ACM SIGGRAPH Course notes, pp. 15–87. 2, 5

[SUS11] SZELISKI R., UYTTENDAELE M., STEEDLY D.: Fast poisson blending using multi-splines. In *2011 IEEE International Conference on Computational Photography (ICCP)* (2011), pp. 1–8. 2

[Sze90] SZELISKI R.: Fast surface interpolation using hierarchical basis functions. *IEEE Trans. Pattern Anal. Mach. Intell. 12*, 6 (June 1990), 513–528. 2

[Sze06] SZELISKI R.: Locally adapted hierarchical basis preconditioning. *ACM Trans. Graph. 25*, 3 (July 2006), 1135–1143. 2

[TSP*08] TENLLADO C., SETOAIN J., PRIETO M., PIÑUEL L., TIRADO F.: Parallel implementation of the 2d discrete wavelet transform on graphics processing units: Filter bank versus lifting. *IEEE Trans. Parallel Distrib. Syst. 19*, 3 (Mar. 2008), 299–310. 5

[Urb00] URBAN K.: Wavelet bases in **H**(div) and **H**(curl). *Mathematics of Computation 70*, 234 (May 2000), 739–766. 2

[vdLJR11] VAN DER LAAN W. J., JALBA A. C., ROERDINK J. B. T. M.: Accelerating wavelet lifting on graphics hardware using cuda. *IEEE Transactions on Parallel and Distributed Systems 22*, 1 (2011), 132–146. 5

[YHLX13] YAN T., HUANG Z., LAU R. W., XU Y.: Seamless stitching of stereo images for generating infinite panoramas. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology* (2013), ACM, pp. 251–258. 1

Appendix A: Derivations of Equation (5) and Equation (11)

For a general derivation, let $\{\mathbf{B}_{\mathbf{c},\mathbf{i}}\}$ and $\{\mathbf{B}_\mathbf{i}\}$ denote two general bases of $\mathbf{H}^0_\mathbf{c}(\Omega)$ and $\mathbf{L}^2(\Omega)$, respectively. Equations (3, 4) or Equations (9, 10) are two types of such bases. A common important property of these two types of bases is that $\mathbf{B}_{\mathbf{c},\mathbf{i}} = \mathbf{w}\mathbf{B}_\mathbf{i}$ for each $\mathbf{i}$, see Equation (3) with $\mathbf{w} = -\mathbf{k}$ and Equation (9) with $\mathbf{w} = 2^{\mathbf{j}_e}$.

Our goal is to solve Problem (1). Substituting the linear representations $\mathbf{u} = \sum_\mathbf{i} \tilde{\mathbf{u}}(\mathbf{i})\mathbf{B}_\mathbf{i}$ and $\mathbf{u}_\mathbf{c} = \sum_\mathbf{i} \tilde{u}_\mathbf{c}(\mathbf{i})\mathbf{B}_{\mathbf{c},\mathbf{i}}$ into it, we get

$$\min_{\tilde{u}_\mathbf{c}} \frac{1}{2}\int_\Omega \|\sum_\mathbf{i}(\tilde{\mathbf{u}}(\mathbf{i})\mathbf{B}_\mathbf{k} - \tilde{u}_\mathbf{c}(\mathbf{i})\mathbf{B}_{\mathbf{c},\mathbf{i}})\|^2. \tag{15}$$

Instead of directly solving this global problem, we solve a series of sub-problems, i.e.

$$\min_{\tilde{u}_\mathbf{c}(\mathbf{i})} \frac{1}{2}\int_\Omega \|\tilde{\mathbf{u}}(\mathbf{i})\mathbf{B}_\mathbf{i} - \tilde{u}_\mathbf{c}(\mathbf{i})\mathbf{B}_{\mathbf{c},\mathbf{i}}\|^2, \tag{16}$$

for each $\mathbf{i}$. Substituting $\mathbf{B}_{\mathbf{c},\mathbf{i}} = \mathbf{w}\mathbf{B}_\mathbf{i}$ into it and taking the derivative w.r.t. $\tilde{u}_\mathbf{c}(\mathbf{i})$, the minimum is obtained at $\tilde{u}_\mathbf{c}(\mathbf{i}) = \langle \mathbf{w}, \tilde{\mathbf{u}}(\mathbf{i})\rangle / \langle \mathbf{w}, \mathbf{w}\rangle$. From a geometrical view, this is the orthogonal projection of $\tilde{\mathbf{u}}(\mathbf{i})$ onto $\mathbf{w}$. Then, we get Equation (5) when $\mathbf{w} = -\mathbf{k}$, and Equation (11) when $\mathbf{w} = 2^{\mathbf{j}_e}$.

Discussion. If $\{\mathbf{B}_\mathbf{i}\}$ is orthogonal, by Parseval's identity, it can be proved that the sub-problem is equal to the global one; otherwise an approximate solution is obtained. Thus Equation (5) is the exact solution since Equation (4) is orthogonal and Equation (11) is an approximate solution since compactly supported Equation (10) is not orthogonal. However, we note that Equation (11) is also exact if $\mathbf{u}$ is itself a curl-free vector field. That is, if $\mathbf{u}$ is the gradients of an unknown image, we can exactly reconstruct the image.

Appendix B: Pseudocode of the 1D and 2D wavelet transform

The pseudocode of the standard 2D WT is listed in Algorithm 3, which takes the 1D WT listed in Algorithm 4 for each row and column of input.

Algorithm 3 2D wavelet transform: WT2

Input: $S, n = (n_x, n_y)$ ▷ *a 2D signal, the size of S*
Input: t_x, t_y ▷ *the wavelet type of the row and column WT*
Output: $\tilde{S}$
1: $S(i,:) = \text{WT1}(S(i,:), t_x, \text{true})$ for $i = 1...n_y$ ▷ *row WT*
2: $\tilde{S}(:,i) = \text{WT1}(S(:,i), t_y, \text{true})$ for $i = 1...n_x$ ▷ *column WT*

Algorithm 4 1D wavelet transform: WT1

Input: $\mathbf{f}/\tilde{\mathbf{f}}, t$ ▷ *a 1D signal or wavelet coeffs., the wavelet type*
Input: $\mathbf{A}, \mathbf{S}$ ▷ *analysis & reconstruction matrix according to t*
Input: *isForward* ▷ *the forward/inverse WT*
Input: $J = \lceil \log 2(n) \rceil$ ▷ *n is the size of s*
Output: $\mathbf{f}/\tilde{\mathbf{f}}$
1: **if** *isFoward* **then**
2: $\tilde{\mathbf{f}}_j = \mathbf{A}\mathbf{f}_j$ for $j = J...1$ ▷ *forward WT, see Equation (14a)*
3: **else**
4: $\mathbf{f}_j = \mathbf{S}\tilde{\mathbf{f}}_j$ for $j = 1...J$ ▷ *inverse WT, see Equation (14b)*
5: **end if**