Coordinates for Instant Image Cloning

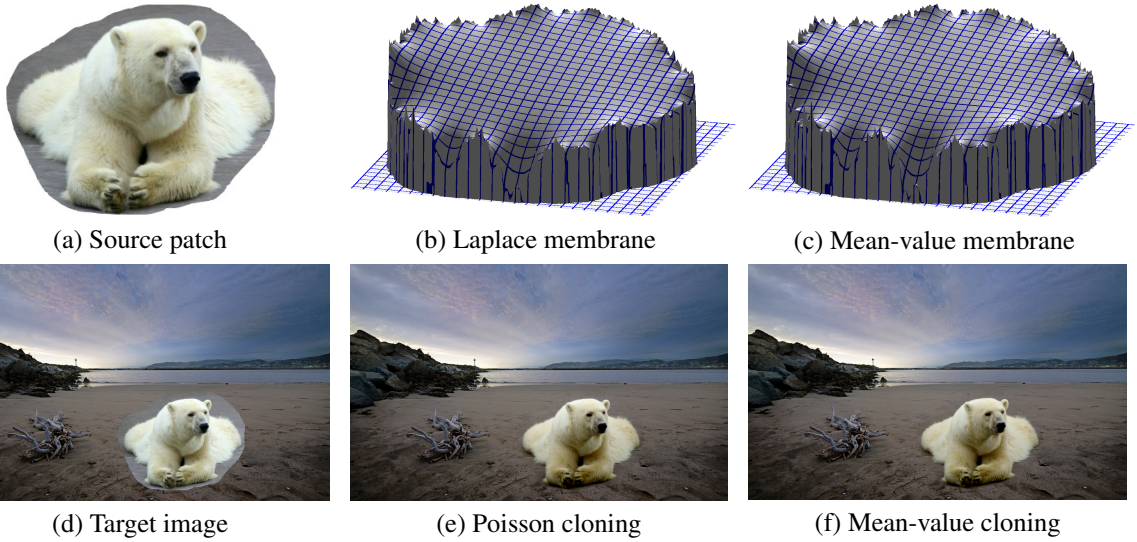
Zeev Farbman
Hebrew University

Gil Hoffer
Tel Aviv University

Yaron Lipman
Princeton University

Daniel Cohen-Or
Tel Aviv University

Dani Lischinski
Hebrew University



(a) Source patch (b) Laplace membrane (c) Mean-value membrane

(d) Target image (e) Poisson cloning (f) Mean-value cloning

Figure 1: *Poisson cloning smoothly interpolates the error along the boundary of the source and the target regions across the entire cloned region (the resulting membrane is shown in (b)), yielding a seamless composite (e). A qualitatively similar membrane (c) may be achieved via transfinite interpolation, without solving a linear system. (f) Seamless cloning obtained instantly using the mean-value interpolant.*

Abstract

Seamless cloning of a source image patch into a target image is an important and useful image editing operation, which has received considerable research attention in recent years. This operation is typically carried out by solving a Poisson equation with Dirichlet boundary conditions, which smoothly interpolates the discrepancies between the boundary of the source patch and the target across the entire cloned area. In this paper we introduce an alternative, *coordinate-based* approach, where rather than solving a large linear system to perform the aforementioned interpolation, the value of the interpolant at each interior pixel is given by a weighted combination of values along the boundary. More specifically, our approach is based on Mean-Value Coordinates (MVC). The use of coordinates is advantageous in terms of speed, ease of implementation, small memory footprint, and parallelizability, enabling real-time cloning of large regions, and interactive cloning of video streams. We demonstrate a number of applications and extensions of the coordinate-based framework.

Keywords: gradient domain, image editing, mean-value coordinates, Poisson equation, matting, seamless cloning, stitching

1 Introduction

A wide variety of image and video editing tasks may be effectively accomplished by *gradient domain* techniques, which operate directly on the gradient field of an image [Fattal et al. 2002; Pérez et al. 2003; Levin et al. 2004; Agarwala et al. 2004; McCann and Pollard 2008]. One of the most useful gradient domain tools is *Poisson cloning*: seamless insertion of a source image patch into a target image (Figure 1). This operation has attracted significant research attention in recent years [Pérez et al. 2003; Agarwala et al. 2004; Wang et al. 2004; Jia et al. 2006] and it is featured in professional image editing products [Georgiev 2004].

All gradient domain techniques eventually solve a large sparse linear system, the Poisson equation. This motivated a number of works proposing fast Poisson solvers for various scenarios [Szeliski 2006; Agarwala 2007; Kazhdan and Hoppe 2008] and for solving the Poisson equation on the GPU [Bolz et al. 2003; McCann and Pollard 2008].

In this paper, we introduce a new, *coordinate-based* approach that performs seamless cloning, as well as a number of other related operations in a direct manner, without ever having to form and solve systems of equations. Our approach is fast, straightforward to implement, and features a small memory footprint. The bulk of the computation may be performed completely in parallel, making it an ideal candidate for a GPU implementation.

When performing Poisson cloning, one typically solves the Poisson equation, where the gradients inside the cloned region come from the source patch, and the Dirichlet boundary conditions are prescribed by the target image. Perez *et al.* [2003] observed that solving this Poisson equation is equivalent to solving the Laplace equation with the Dirichlet boundary conditions set to the *difference along the boundary* between the source patch and the tar-

get image. In other words, Poisson cloning constructs a harmonic (or *membrane*) interpolant that smoothly spreads the discrepancies along the boundary to the entire cloned area. While the gradient field of this membrane has minimal L^2-norm, there is no evidence that this particular membrane is necessarily optimal from the perceptual standpoint. Thus, our key idea is to construct a different smooth interpolating membrane *directly*, i.e., without solving a linear system. While the membrane we construct is not identical to the harmonic one, our final results are nevertheless typically indistinguishable from Poisson cloning (Figure 1).

Specifically, our objective is to find a harmonic-like interpolant to some values along the boundary. Recent advances in the field of transfinite interpolation allow solving this problem using generalized barycentric coordinates. An important instance is Floater's Mean-Value Coordinates (MVC) [Floater 2003]. These coordinates were specifically designed for constructing smooth harmonic-like interpolants by mimicking the mean-value property of harmonic functions, and they are given by a simple closed-form formula. Thus, the resulting membrane may be evaluated *in parallel* for any point inside the region at a cost linear in the number of boundary vertices.

We further observe that due to the smoothness of the membrane away from the boundary, it is not necessary to evaluate it at each and every pixel inside the cloned area. Instead, it suffices to evaluate the membrane only at the vertices of an adaptive mesh, and obtain the values at the remaining pixels by linear interpolation. A similar optimization was recently utilized by Agarwala [2007] to solve large Poisson systems, such as those arising in gradient domain stitching, with a small memory footprint. Another important optimization that we introduce is adaptive hierarchical sampling of the boundary.

After presenting the use of mean-value coordinates for seamless image cloning in Section 3, describing an efficient implementation on the CPU and on the GPU, and comparing to existing approaches, we go on to present a number of applications and extensions of our approach (Section 5). Specifically, we discuss real-time interactive seamless video cloning, seamless stitching of large panoramas, removal of "smudging" artifacts that sometimes occur with seamless cloning, and MVC-based matte extraction.

In summary, our specific contributions are:

- A new, coordinate-based method for seamless cloning, which is easy to implement, features a small memory footprint, and is highly parallelizable.
- Real-time seamless cloning and healing of still images and video sequences on the CPU, as well as on the GPU.
- Extensions to related operations, such as seamless stitching and matting.

2 Background

Gradient domain methods

Psychologists have long discovered that the human visual system is much more sensitive to local contrasts than to absolute luminances or to slow changes in the luminance [Land and McCann 1971; Palmer 1999]. In particular, slow luminance changes, which are suppressed by the human visual system as part of lightness constancy, may be often superimposed over an image without a noticeable effect.

Gradient domain methods take advantage of the above properties, and modify images by manipulating their gradient field to perform a variety of tasks, ranging from shadow removal [Weiss 2001; Finlayson et al. 2002], to tone mapping [Fattal et al. 2002], seamless stitching [Levin et al. 2004; Agarwala et al. 2004], image cloning [Pérez et al. 2003; Georgiev 2004; Jia et al. 2006], seamless video editing [Wang et al. 2004], and, recently, gradient domain painting [McCann and Pollard 2008].

Reconstructing a new image from the modified gradient field typically requires solving the Poisson equation, which yields the image whose gradient field is closest (in the L^2-norm sense) to the modified one, subject to some boundary conditions. For example, in Poisson cloning [Pérez et al. 2003], the gradient field (sometimes referred to as the *guidance field*) inside the cloned region is taken from the source image, while the values of the target image along the boundary of the cloned region are used to define the Dirichlet boundary conditions for the equation.

Solving the Poisson equation for large images is a computational and memory intensive task. Agarwala [2007] observed that in the case of gradient domain stitching, one essentially solves for an offset function that is smooth away from the seams. This makes it possible to obtain an accurate solution by constructing a reduced linear system using an adaptive quadtree subdivision of the domain. This method has been shown to be significantly faster and more scalable than general Poisson solvers for stitching large images. We also take advantage of smoothness and use an adaptive mesh to speed up our computation and to make it scalable; however, in contrast to Agarwala we avoid solving a linear system altogether.

McCann and Pollard [McCann and Pollard 2008] describe a fast GPU implementation of a multi-grid Poisson solver, with which they achieve real-time interactive performance for gradient domain image editing operations, including seamless cloning. While their system outperforms previous methods, it does involve a substantial memory footprint, and the authors report that performance drops down once this footprint exceeds the available video memory.

Mean-Value Coordinates

Recently, there has been significant interest in using generalized barycentric coordinates for solving transfinite interpolation problems [Wachpress 1975; Floater 2003; Warren 1996]. In his seminal paper, Floater [2003] introduced the Mean-Value Coordinates (MVC) which are motivated by the Mean-Value Theorem for harmonic functions. These coordinates approximate a harmonic-like solution to the boundary interpolation problem. They are well-defined over the entire plane for arbitrary planar polygons without self-intersections, smooth (C^∞, except at the polygon vertices where they are C^0), and invariant under similarity transformations [Hormann and Floater 2006]. MVC coordinates have also been extended to 3D polyhedra and used for space deformation [Ju et al. 2005; Floater et al. 2005; Joshi et al. 2007]. In this work, we explore the novel use of MVC as a computationally attractive alternative for solving the Poisson equation in certain image editing tasks.

In the remainder of this section, we quickly define the 2D mean-value interpolant, and refer the reader to the references mentioned above for detailed derivations in 2D and in 3D. Consider a closed 2D polygonal boundary curve (with counter-clockwise ordering) $\partial P = (\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_m = \mathbf{p}_0)$, $\mathbf{p}_i \in \mathbb{R}^2$. The mean-value coordinates of a point $\mathbf{x} \in \mathbb{R}^2$ with respect to ∂P are given by

$$\lambda_i(\mathbf{x}) = \frac{w_i}{\sum_{j=0}^{m-1} w_j}, \quad i = 0, \ldots, m-1, \tag{1}$$

where

$$w_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|\mathbf{p}_i - \mathbf{x}\|}, \tag{2}$$

and α_i is the angle $\sphericalangle \mathbf{p}_i, \mathbf{x}, \mathbf{p}_{i+1}$ (see Figure 2). Once computed,

these coordinates may be used to smoothly interpolate any function f defined at the boundary vertices:

$$\tilde{f}(\mathbf{x}) = \sum_{i=0}^{m-1} \lambda_i(\mathbf{x}) f(\mathbf{p}_i). \quad (3)$$

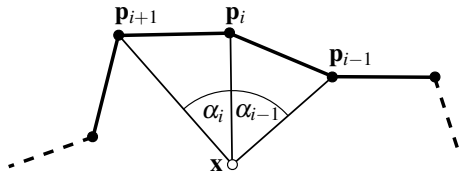


Figure 2: *Angle definitions for mean-value coordinates.*

3 Mean-Value Seamless Cloning

In this section we explain in detail how mean-value coordinates may be used to perform instant seamless image cloning.

Let $S \subset \mathbb{R}^2$ be the domain of the source image and $T \subset \mathbb{R}^2$ be the domain of the target image for cloning. Let us denote by $g : S \to \mathbb{R}$, $f^* : T \to \mathbb{R}$ the source and target image intensities over their respective domains. Let $P_s \subset S$ denote the source patch that we would like to clone seamlessly into $P_t \subset T$. We assume that these patches are isomorphic, and that their boundaries, ∂P_s and ∂P_t, are polygonal curves with the same number of vertices.

Poisson cloning computes a function $f : P_t \to \mathbb{R}$ by solving the Poisson equation:

$$\Delta f = \mathrm{div} \nabla g \quad \text{w/ Dirichlet boundary conditions } f\big|_{\partial P_t} = f^*. \quad (4)$$

In other words, Poisson cloning seeks a function f that agrees with the target image f^* on the boundary of the target region ∂P_t, whose gradient field is as close as possible to that of the source image g. Pérez *et al.* [2003] noted that solving the above Poisson equation is equivalent to solving the Laplace equation:

$$\Delta \tilde{f} = 0, \quad \text{w/ Dirichlet boundary conditions } \tilde{f}\big|_{\partial P_t} = f^* - g. \quad (5)$$

The final outcome of the cloning is then simply defined as

$$f = g + \tilde{f}. \quad (6)$$

This formulation reveals that Poisson cloning in fact constructs a smooth membrane (a harmonic function) $\tilde{f}$ that interpolates the difference $f^* - g$ between the target and source images on the boundary of P_t across the entire region.

As stated earlier, we propose to construct a similar smooth interpolating membrane $\tilde{f}$ in an entirely different manner, using mean-value interpolation, as described below The most obvious advantage of using the mean-value interpolant is that there exists a simple closed-form formula for constructing it, hence eliminating the need to solve a large linear system.

Consider a point $\mathbf{x} \in P_t$ with boundary $\partial P_t = (\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_m = \mathbf{p}_0)$. The mean-value interpolant obtaining the values $f^* - g$ at the boundary ∂P_t is given at point $\mathbf{x}$ by

$$r(\mathbf{x}) = \sum_{i=0}^{m-1} \lambda_i(\mathbf{x})(f^* - g)(\mathbf{p}_i), \quad (7)$$

where $\lambda_i(\mathbf{x})$, $i = 0, \ldots, m-1$ are the mean-value coordinates with respect to ∂P_t, as defined by equations (1–2). The result of mean-value cloning is then given, similarly to eq. (6), by

$$f = g + r. \quad (8)$$

Algorithm 1 MVC Seamless Cloning

1: {Preprocessing stage}
2: **for each** pixel $\mathbf{x} \in P_s$ **do**
3: {Compute the mean-value coordinates of $\mathbf{x}$ w.r.t. ∂P_s}
4: $\lambda_0(\mathbf{x}), \ldots, \lambda_{m-1}(\mathbf{x}) = MVC(x, y, \partial P_s)$
5: **end for**
6: **for each** new P_t **do**
7: {Compute the differences along the boundary}
8: **for each** vertex $\mathbf{p}_i$ of ∂P_t **do**
9: $diff_i = f^*(\mathbf{p_i}) - g(\mathbf{p}_i)$
10: **end for**
11: **for each** pixel $\mathbf{x} \in P_t$ **do**
12: {Evaluate the mean-value interpolant at $\mathbf{x}$}
13: $r(\mathbf{x}) = \sum_{i=0}^{m-1} \lambda_i(\mathbf{x}) \cdot diff_i$
14: $f(\mathbf{x}) = g(\mathbf{x}) + r(\mathbf{x})$
15: **end for**
16: **end for**

An unoptimized mean-value cloning procedure is given in pseudocode in Alg. 1. This routine precomputes the mean-value coordinates of each pixel inside the source patch P_s once the patch is selected and then repeatedly performs mean-value interpolation for each location P_t in the target image. It is easy to see that the number of operations is $O(nm)$, where n is the number of pixels in the cloned region, while m is the number of boundary pixels. Since the mean-value coordinates are precomputed and stored, the memory footprint is also $O(nm)$. To make MVC cloning fast and scalable, we introduce two optimizations, which are described below.

Adaptive mesh. The mean-value interpolant is very smooth away from the boundary of the cloned region. Thus, for all practical purposes, much of the computation in Alg. 1 may be avoided by constructing an adaptive triangular mesh over P_s. We use the CGAL [Cgal 2007] library to generate the adaptive mesh. An example is shown in Figure 3. Once the mesh is available, we only need to compute and store the mean-value coordinates (line 4 in Alg. 1) at each mesh vertex. Likewise, the evaluation of the interpolant (line 13 in Alg. 1) is also only performed at the mesh vertices, and the value at each pixel is obtained by linear interpolation of the three values at the vertices of the containing triangle. The number of these vertices is in practice roughly linear in the number of boundary pixels. This reduces the total complexity of computing the coordinates and of evaluating the interpolant to $O(m^2)$, enabling interactive performance when cloning regions of moderate size.

Hierarchical boundary sampling. A further significant speedup is achieved by hierarchically sampling the boundary, rather than using all of the boundary pixels. This idea is inspired by adaptive hierarchical approaches, such as fast particle simulation algorithms [Carrier et al. 1988] and hierarchical radiosity [Hanrahan et al. 1991]. Similarly to Coulomb potential fields and solid angles, the mean-value weight of each boundary vertex decays quickly with distance. Thus, an accurate approximation of the membrane may be achieved by sampling the boundary with density that is inversely proportional to the distance, as demonstrated in Figure 3. In practice, only a constant number of boundary vertices are used when computing the coordinates and the membrane at each mesh vertex, reducing the total cost of these operations to $O(m)$.

Specifically, we first construct a 1D hierarchy over the sequence of boundary pixels. Each coarser level in the hierarchy is obtained by dropping every other point in the previous (finer) level. Note that by this construction, if a vertex is present at some coarse level in the hierarchy, it is also present in all the finer levels. The process stops once the number of points in the coarsest level falls below a predefined constant (16 in our implementation).

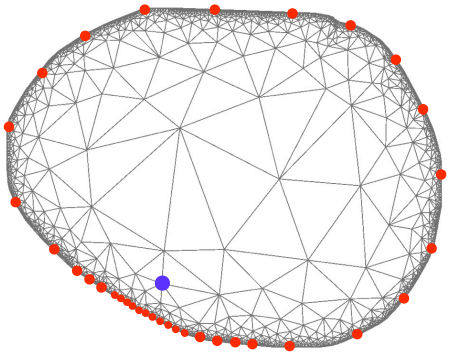


Figure 3: *An adaptive triangular mesh constructed over the region to be cloned. The red dots on the boundary show the positions of boundary vertices that were selected by adaptive hierarchical subsampling for the mesh vertex indicated in blue.*

Next, for each mesh vertex $\mathbf{x}$, we traverse to hierarchy from the top (coarse) level down. Let $\mathbf{p}_{i-s}^k, \mathbf{p}_i^k, \mathbf{p}_{i+s}^k$ be three consecutive vertices at the k-th level of the hierarchy, where s is the index step between successive vertices at that level. If each of the following three conditions hold:

$$\begin{aligned} \|\mathbf{x} - \mathbf{p}_i^k\| &> \varepsilon_{dist} \\ \sphericalangle(\mathbf{p}_{i-s}^k, \mathbf{x}, \mathbf{p}_i^k) &< \varepsilon_{ang} \\ \sphericalangle(\mathbf{p}_i^k, \mathbf{x}, \mathbf{p}_{i+s}^k) &< \varepsilon_{ang} \end{aligned}$$

then the mean-value weight (2) corresponding to $\mathbf{p}_i^k$ at $\mathbf{x}$ is sufficiently small and no further refinement of the boundary is necessary around $\mathbf{p}_i^k$. If this is not the case, denser sampling is required in order to provide a better approximation of the membrane. Therefore, we insert two additional points, and repeat the same test for each of the three vertices at the next (finer) level: $\mathbf{p}_{i-s/2}^{k+1}$, $\mathbf{p}_i^{k+1}$, and $\mathbf{p}_{i+s/2}^{k+1}$. In our current implementation we set the distance and angle thresholds ε_{dist} and ε_{ang} to:

$$\varepsilon_{dist} = \frac{\#\text{ boundary pixels}}{16 \cdot 2.5^k} \quad \text{and} \quad \varepsilon_{ang} = 0.75 \cdot 0.8^k,$$

where k is the current depth in the hierarchy ($k = 0$ at the coarsest level). While these expressions were found to provide a good trade-off between speed and visual quality in our experiments, they are not necessarily optimal, and could benefit from further tuning.

When given an error function $f^* - g$ on the boundary, care must be taken to avoid aliasing due to subsampling of the boundary. Thus, we progressively low-pass filter $f^* - g$ to obtain adequately band-limited values at each hierarchy level, before computing the interpolants at any of the mesh vertices.

With both of the above optimizations in place, the total cost of computing the MVC coordinates and of evaluating the membrane (lines 4 and 13 in Alg. 1) becomes roughly linear in the number of boundary pixels $O(m)$, which in practice grows as $O(\sqrt{n})$, where n is the number of cloned pixels. Of course, because we linearly interpolate the membrane values to all n pixels, the asymptotic behavior is still $O(n)$, similarly to Agarwala [2007].

4 Implementation and Performance

We have implemented MVC cloning both on the CPU and on the GPU. Both implementations target the interactive seamless cloning scenario, where the user first selects an image region to clone and then moves it across the target image, while the seamlessly cloned result is instantly generated and displayed at each target position.

CPU implementation. Once a selection has been made, a short pre-processing stage takes place, during which the adaptive mesh is created, and a vector of MVC coordinates is computed and stored with each mesh vertex. We also precompute, for each pixel in the selected region, the index of the mesh triangle containing this pixel and the three barycentric coordinates with respect to the containing triangle. As the region is moved to each new target location, we compute the error $f^* - g$ at each boundary point, evaluate the mean-value membrane $r(\mathbf{x})$ at each mesh vertex, linearly interpolate to each pixel, using the precomputed barycentric coordinates, and finally compute the sum $g + r$.

For a region with 133K interior pixels and 1,562 boundary pixels, the preprocessing stage takes 0.3 seconds (on a single core of an AMD Athlon 2.5GHz). The interactive cloning then proceeds at a rates exceeding 90 updates per second. More timings and statistics are given in Table 1. As expected, the number of mesh vertices grows linearly with the length of the boundary, but the number of boundary points sampled by each vertex remains roughly constant, thanks to the adaptive hierarchical subsampling scheme. As cloned regions become larger, the computation of barycentric coordinates eventually dominates preprocessing time, and the cloning time becomes dominated by the linear interpolation step. Thus, for large regions, performing MVC cloning is almost as cheap as performing a linear interpolation at each pixel. The memory footprint is modest, consisting mainly of storing the barycentric coordinates and a mesh triangle index for each pixel.

GPU implementation. MVC cloning is trivially parallelizable, since the membrane evaluation at each mesh vertex is performed completely independently of the other vertices. Our current GPU implementation also uses an adaptive mesh to approximate the membrane, and performs the hierarchical boundary subsampling. The adaptive mesh and the vector of MVC coordinates at each mesh vertex are precomputed on the host CPU as before, but it is no longer necessary to precompute and store the barycentric coordinates of each pixel, further reducing the memory footprint. At each frame, a simple vertex shader (30 lines of GLSL) evaluates the error membrane $r(\mathbf{x})$ at each mesh vertex, the rasterizing hardware linearly interpolates these values to each pixel, and a trivial fragment shader (6 lines of GLSL) computes the final value of $g + r$. This results in seamless cloning at roughly 134 frames per second on a mobile GPU (NVIDIA GeForce 9600M GT), when cloning a region with 133K interior pixels. The speed advantage of the GPU implementation over the CPU increases with the size of the cloned region (see Table 1).

Table 1: *Performance statistics for MVC cloning. Times exclude disk I/O and sending the images to the graphics subsystem. Cloning rate is the number of region updates per second.*

#cloned pixels	#bdry pixels	#mesh vertices	coords /vertex	prep. time(s)	cloning rate	
					CPU	GPU
51,820	1,113	2,063	38.63	0.15	199.0	163
133,408	1,562	2,963	44.21	0.30	92.1	134
465,134	2,683	5,323	45.50	0.63	22.6	82
1,076,572	4,145	8,241	44.59	1.16	9.7	44
4,248,461	8,133	16,369	57.71	3.63	2.7	26
12,328,289	14,005	28,240	58.68	8.99	0.94	–

Comparison with previous approaches

We are not aware of any existing system that is able to perform seamless cloning on the CPU at the rates reported above. Testimonials by other researchers [Pérez et al. 2003; McCann and Pollard

2008], as well as our own experiments, indicate that common Poisson solvers on the CPU are able to handle regions with 256^2 pixels at a rate of 3–5 solutions per second. Another possibility, which we have not seen mentioned in the literature, is to precompute a factorization of the Poisson equation matrix during the preprocessing stage, and then quickly compute the solution via back-substitution at each target location. In our experiments, for a region with 125K pixels, computing the back-substitution takes 0.3 seconds. Thus, all of the above are significantly slower than the rates we are able to achieve.

McCann and Pollard [2008] also demonstrate real-time seamless cloning as one of the features of their gradient-domain painting system, reporting rates of 20 multigrid V-cycles per second for a one megapixel image. A screen captured session with their system is included in the accompanying video. Note that while the seamless cloning indeed takes place in real time, there is a fair amount of noticeable flicker, as the cloned region is dragged about. The flicker may be attributed to two factors: (i) the Poisson equation is solved over the entire image (with Neumann boundary conditions), thus the position of the cloned patch has a global effect on the result; (ii) the result is updated after each V-cycle, which is not always sufficient to achieve complete visual convergence. In fairness, it should be noted that region cloning is but one feature among several supported by the gradient-domain painting system. It is reasonable to assume that a GPU-based multigrid solver would perform better and avoid flicker, if applied to the cloned region only (with Dirichlet boundary conditions). Still, solving the Poisson equation on the GPU is a much more involved task than MVC cloning, and has a significantly larger memory footprint.

5 Results and Extensions

MVC vs. Poisson. We have compared MVC cloning to Poisson cloning on a large number of examples, using a variety of images and differently shaped cloning regions. Our conclusion is that although the corresponding membranes are by no means identical, the outcome of the cloning is typically difficult to tell apart visually. Even when (subtle) differences are visible, it is usually difficult to prefer one outcome over another. The differences between the two kinds of membranes tend to be smaller for convex shapes, such as a rectangle or a disk, as demonstrated in Figure 1. Cloning more concave regions, such as the one shown in Figure 4, typically results in more significant differences between the membranes, but the results are difficult to tell apart. The differences between the membranes become most apparent for extremely concave regions. For example, consider the synthetic example shown in the top row of Figure 5. Here, the goal is to fill an omega-shaped hole. While the Laplace membrane succeeds in eliminating the hole with almost no visible trace, MVC interpolation is less successful. The reason is that the MVC membrane in each half of the shape is affected by values along the boundary of the opposite half, despite there being no lines of sight (inside the shape) between the two halves. However, in a more typical scenario with less extreme gradients and some texture, the results become comparable in quality, even though the same concave region is used (Figure 5, bottom row).

Instant seamless cloning. The gains we achieve in performance translate into a significantly different interactive experience for the user. To illustrate this, the accompanying video includes a real-time screen capture of seamless cloning with the *Patch tool* in Photoshop CS4. Note that while the user is dragging the patch around no seamless cloning takes place, and thus the user is unable to assess the result of the operation in real time. There is also a noticeable delay from the time the patch is dropped in its target position until the final result appears. In contrast, when cloning with our approach, the seamless cloning result is displayed instantly, greatly assisting the user in selecting a suitable target position.

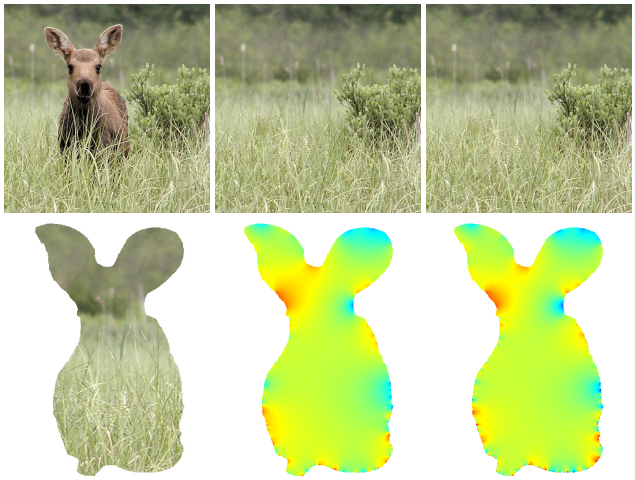


Figure 4: *Object removal with Poisson cloning (middle) and MVC cloning (right). Top left: original image; bottom left: source patch. The corresponding membranes are visualized using a colormap. Although the visualization reveals some numerical differences between the membranes (RMS difference of about 0.015), it is difficult to see the difference between the resulting images.*

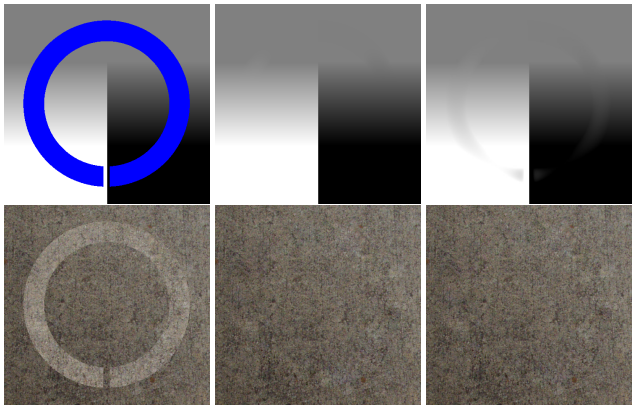


Figure 5: *Poisson vs. MVC over a highly concave region. Left: input image; Middle: Poisson cloning; Right: MVC cloning.*

Figure 6: *More seamless cloning results, obtained by rotating and scaling the source patch (left: original, right: after cloning).*

As was mentioned earlier, MVC are invariant under similarity transformations. Thus, during an interactive cloning session, the source region may be rotated and scaled without the need to repeat the preprocessing. Again, we found the ability to do this with instant feedback extremely helpful. Two results obtained with the use of such transformations are shown in Figure 6.

5.1 Mean-Value Video Cloning

Given the speed of MVC cloning, it is only natural to consider applying it to seamless cloning of video. Seamless video cloning has been attempted before by Wang *et al.* [2004], by forming and solving a 3D Poisson equation over the entire 3D space-time volume of the video. Since our goal is to clone interactively, while both the source clip and the target video are continuously playing, we opt instead for a frame-by-frame solution, with temporal smoothing between consecutive interpolating membranes to ensure temporal coherence.

In our current implementation, the shape of the source video patch and its position in the source video frames are kept fixed. We store with each mesh vertex (in addition to its MVC coordinates) a set of its membrane values in the last k frames. To form a membrane for the current frame we compute a weighted average of these values, with the weights of older frames decaying with time: $\Delta t^{-0.75}$, where Δt is the distance in frames between the current frame and the older one. However, for seamless results the membrane must respond quickly to changing discrepancies between the source and the target along the boundary. Thus, the weight of older membranes in the temporal averaging is further reduce at vertices near the boundary (by a factor of 2^{-d}, where d is the normalized distance of a vertex to the boundary).

The accompanying video demonstrates some results of interactive seamless video cloning (captured in real time). Snapshots from the interactive session are shown in Figure 7. Seamless cloning of video is a much more challenging task than cloning in still images: inserting and removing objects can be a time-consuming and frustrating task. Therefore, the kind of real-time feedback provided by our approach is instrumental to the user's ability to achieve satisfactory results.

5.2 MVC Stitching

Gradient-domain stitching and seamless cloning are closely related. For example, stitching may be done by setting up a guiding field that uses the gradients of the images being stitched away from the seams, and the average of the gradient at pixels along the seams [Agarwala 2007]. Solving the Poisson equation then yields a seamlessly stitched composite. This approach typically uses Neumann boundary conditions, which prescribe the value of the derivative normal to the boundary, and thus result in free-floating boundaries.

Since our MVC cloning machinery is based on interpolation of boundary values across the domain, it is suitable for Dirichlet boundary conditions, rather than Neumann boundary conditions. Nevertheless, it is quite easy to adapt our approach to perform stitching. Suppose that our goal is to stitch together two images A and B along a given seam. This may be accomplished by keeping one image, say B, fixed, and adding to A a smooth offset function, which interpolates the error between the two images along the seam and gradually goes to zero away from the seam. Specifically, we construct a polygonal boundary around image A consisting of the pixels on the seam between the images, as well as the "free" (unconstrained) corners of A. We set the offset values to the difference $B-A$ for each boundary vertex (pixel) along the seam, and to zero at the free corners of A. Note that the pixel values along the free edges of A are not constrained to any particular offset value, and the offsets along these edges vary smoothly between $B-A$ at the seam and zero at the free corners. This idea easily extends to any number of images, by computing a similar offset membrane for each image in the composite. Figure 8 shows an example result computed using the approach described above. This 7.5 Mpixel image took 3.7 seconds to stitch, which is slightly faster than the times reported by Agarwala [2007]. Additional experiments (up to 33 Mpixels) indicated that the cost of stitching with our approach grows linearly with the length of the seams, with stitching rates of above 1 Mpixel per second. The memory footprint is also linear in the length of the seams.

5.3 Selective Boundary Suppression

It is well known that Poisson cloning works best when the error along the boundary of the cloned region is nearly constant, or changes smoothly. When this is not the case, there is a visible "smudging" of the error from the boundary into the cloned area (Figure 9). Mixing source and target gradients [Pérez et al. 2003] and optimizing the boundary [Jia et al. 2006] offers a solution to this problem in some, but not all, cases. For example, in cases such as the one shown in Figure 9, no adjustment of the boundary is able to avoid the problem, since any boundary must cut across the trunk of the tree or the ground on which it stands in the source image. An alternative solution, is to construct a smooth membrane which does not attempt to interpolate large errors on the boundary. A similar workaround has been used in the context of gradient domain fusion [Agarwala et al. 2004].

Sections of the boundary where the error is too large to be interpolated may be detected automatically, or indicated by the user. In our current implementation, we let the user paint over a portion of the boundary that causes an undesirable artifact in the cloned result. This signals the cloning routine that the marked boundary vertices should not participate in the membrane evaluation. The mean-value weights λ_i corresponding to these vertices are then set to zero (and the remaining weights are, of course, re-normalized accordingly). Note that this does not involve recomputing the mesh, or the coordinates at each mesh vertex. Figure 9 and the accompanying video show an example of a result obtained in this manner, demonstrating that selective boundary suppression provides the user with more control over the result of the cloning operation and widens the range of scenarios where seamless cloning is possible.

Figure 9: *Selective boundary suppression. Left: source patch with boundary cutting across an object. Right: regular seamless cloning results in smudging (left tree), which is removed via selective boundary suppression (right tree, see also the video).*

Figure 7: *Seamless video cloning: snapshots from an interactive session. A bird is duplicated (left), another bird is removed (middle), a large rock to the left of the bear is removed (right).*

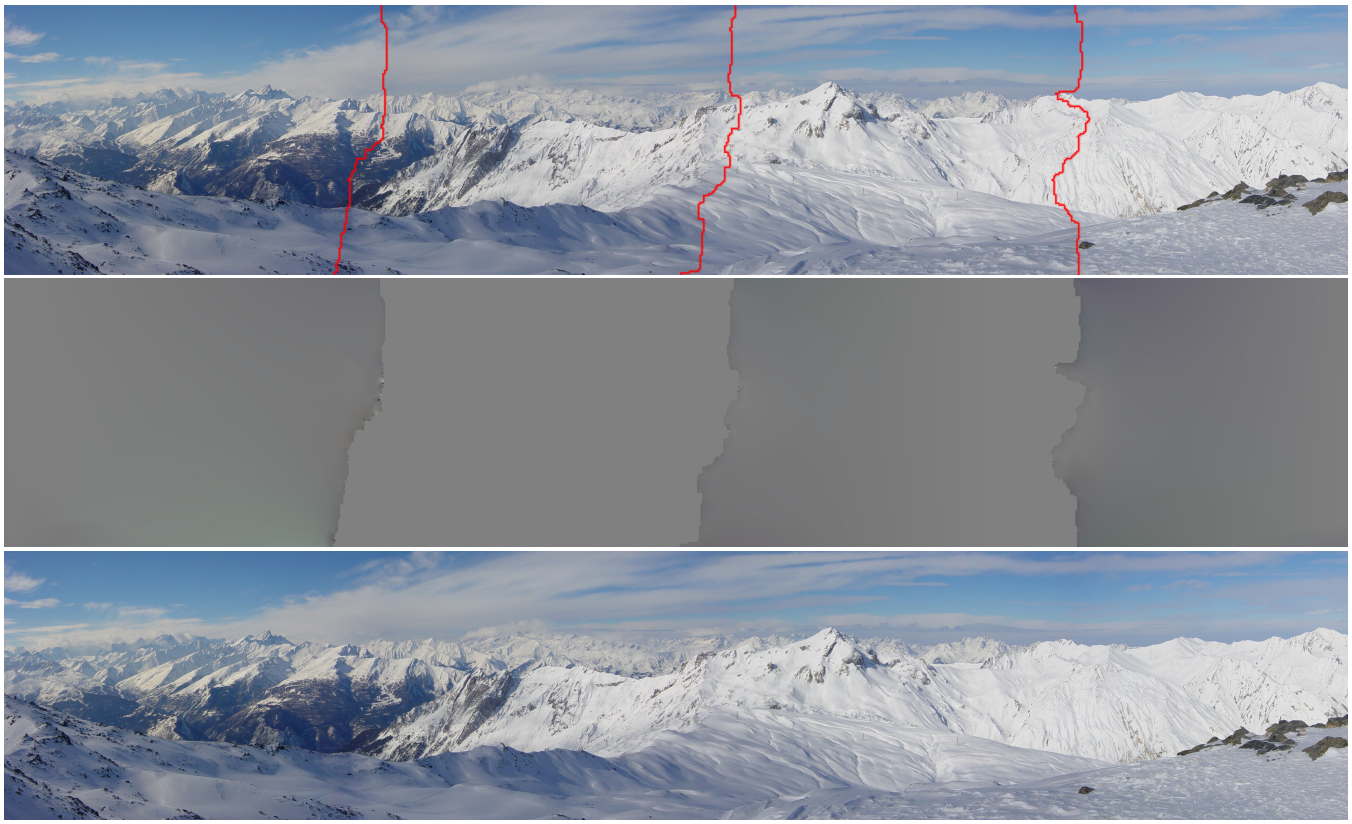


Figure 8: *MVC stitching. Top: composite with seams; Middle: MVC membrane; Bottom: seamlessly stitched panorama.*

5.4 MVC Matting

Poisson matting [Sun et al. 2004] is a gradient-domain technique for extracting the matte of a foreground object. Given an image I, the goal is to estimate the matte α and the foreground and background color functions F and B, such that:

$$I = \alpha F + (1-\alpha)B. \tag{9}$$

In order to accomplish this task, the user provides a *trimap*: a map classifying the image pixels into three disjoint regions: "definitely foreground" Ω_F, "definitely background" Ω_B, and the "unknown region" Ω between them, which contains the boundary of the foreground object.

Poisson matting relies on the assumption that in the unknown region, both the foreground color F and the background color B vary smoothly. Thus, the gradients in this region are assumed to be due to the matte α. More precisely, the matte gradient field over Ω is approximated as:

$$\nabla\alpha \approx \frac{1}{F-B}\nabla I \tag{10}$$

The matte is therefore estimated by solving the Poisson equation

$$\Delta\alpha = \operatorname{div}\frac{\nabla I}{F-B}, \quad \text{such that } \alpha = \begin{cases} 1 & \text{on } \partial\Omega_F \\ 0 & \text{on } \partial\Omega_B \end{cases} \tag{11}$$

In general, the above equation is not equivalent to a Laplace equation, because the vector field $\nabla I/(F-B)$ is not conservative. However, if F and B vary smoothly in the unknown region, as assumed by Poisson matting, we may approximate it by a conservative field:

$$\frac{\nabla I}{F-B} \approx \nabla\frac{I}{F-B}. \tag{12}$$

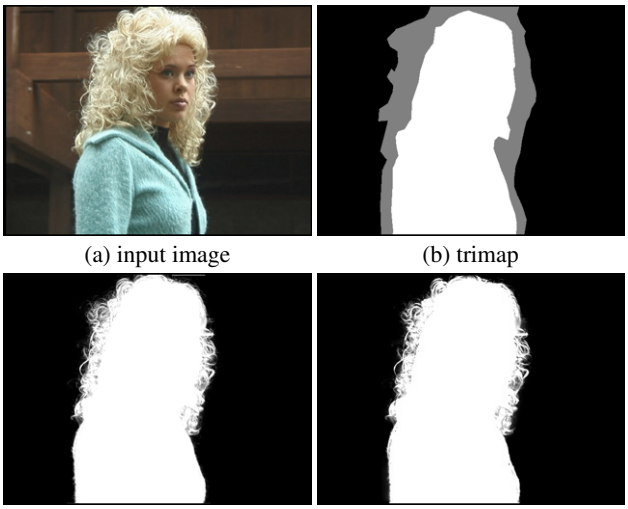
Thus, defining $g = I/(F-B)$, we obtain that solving the Poisson equation (11) is approximately equivalent to solving the Laplace equation:

$$\Delta\tilde{\alpha} = 0, \quad \text{such that } \tilde{\alpha} = \begin{cases} 1-g & \text{on } \partial\Omega_F \\ 0-g & \text{on } \partial\Omega_B, \end{cases} \tag{13}$$

and obtaining the alpha matte as: $\alpha = g + \tilde{\alpha}$. Exactly as before, it is possible to compute a similar membrane interpolant by using mean-value coordinates, instead of solving a linear system.

Specifically, given a trimap, we need to estimate $g = I/(F-B)$ on the boundaries of the unknown region Ω. We use mean-value interpolation to obtain these estimates. The colors B are smoothly interpolated from their known values along the boundary $\partial\Omega_B$, while F is smoothly extrapolated outward from their known values along $\partial\Omega_F$. Here, we take advantage of the fact that mean-value coordinates are well-defined and smooth over the entire plane (except on the boundary itself) [Hormann and Floater 2006].

Figure 10 shows an input image and a corresponding trimap, as well as the resulting mattes produced by Poisson matting and by our approach. It may be seen that although the mattes are not identical they are quite similar. It should be noted that Poisson matting is not the best matting method available today (see [Levin et al. 2008; Wang and Cohen 2007], where Poisson matting is compared with more state-of-the-art methods). However, when attempting to clone an object over a non-homogeneous target image, the kind of matte that we are able to obtain with our approach is often sufficient for a convincing composite. Figure 11c shows a case where seamless cloning fails to produce a satisfactory result. Compare this result with Figure 11d, where the transparency across the cloned region is modulated by the matte computed using our approach: the overall appearance of the eagle matches the target image, but the smudging of the surrounding background is avoided.



(a) input image (b) trimap

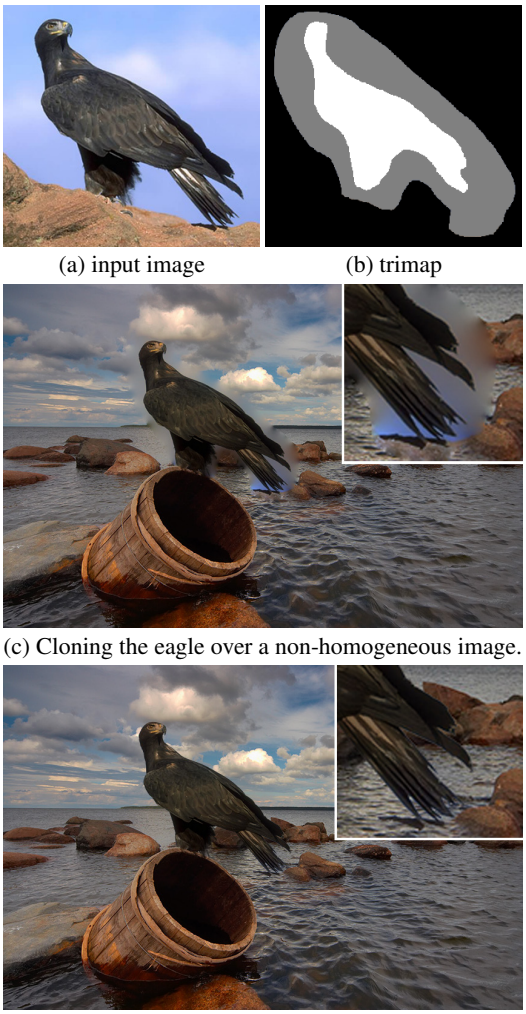
(c) Matte from [Sun et al. 2004] (d) MVC matte

Figure 10: *A comparison with Poisson matting.*

Interestingly, since we have an estimate of F and B at every point inside the unknown region, it is also possible to estimate α directly from these values:

$$\alpha = \frac{I-B}{F-B}. \tag{14}$$

The results are not identical, but comparable to those obtained as described earlier, so this observation merits further investigation.



(a) input image (b) trimap

(c) Cloning the eagle over a non-homogeneous image.

(d) Applying a matte to the cloned region.

Figure 11: *Matted cloning.*

6 Discussion and Conclusions

Using the general framework of mean-value coordinates, we have presented a new approach for seamless cloning of images and video, stitching, and matting. We have demonstrated a number of advantages that our approach offers over existing techniques.

Limitations. One limitation of our approach is that it is not applicable to every scenario where the Poisson equation might be used, as it relies on the ability to decompose the solution into a sum of a smooth interpolating membrane and a known function. Thus, we do not currently see a way of applying it to tasks such as gradient-domain HDR compression [Fattal et al. 2002], or Poisson cloning with mixed gradients [Pérez et al. 2003], where the resulting guiding field is not conservative.

Another limitation, already pointed out earlier, is that seamless cloning (be it MVC-based or Poisson-based) only works well when the texture in the surrounding target region is sufficiently similar to the texture near the boundaries of the source patch. This becomes particularly visible in some video cloning examples, where the textures should match both spatially and temporally for satisfactory results.

Future Work. Our current implementation of video cloning was meant as a proof of concept. We believe that a better, specialized user interface is needed in order to effectively work with seamless video cloning. The user should be able to adjust the shape of the source region for cloning, or the region where an object is to be removed, since having to use a fixed region throughout a video clip is quite limiting. It would also be interesting to investigate whether constructing a 3D interpolant (in the space-time volume of the video) offers any advantages over our current temporal smoothing scheme.

As pointed out in Section 1, a variety of generalized barycentric coordinates schemes have been proposed in recent years. In this paper we chose to focus on MVC, but it might be interesting to explore how some of these other schemes compare with MVC in the context of seamless cloning. For example, the higher order barycentric coordinates proposed by Langer and Seidel [2008] enable interpolation of both values and derivatives on the boundary.

Future work should also examine the possibility of using a coordinate-based approach to perform cloning of volumes, light-fields, and other high-dimensional data sets, as well as seek additional applications of this powerful framework.

Acknowledgments: This work was supported in part by grants from the Israel Ministry of Science, and from the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities. The authors would also like to thank the anonymous reviewers for their comments.

References

AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. *ACM Trans. Graph. 23*, 3, 294–302.

AGARWALA, A. 2007. Efficient gradient-domain compositing using quadtrees. *ACM Trans. Graph. 26*, 3, 94.

BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRÖDER, P. 2003. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Trans. Graph 22*, 3, 917–924.

CARRIER, J., GREENGARD, L., AND ROKHLIN, V. 1988. A fast adaptive multipole algorithm for particle simulations. *SIAM Journal on Scientific and Statistical Computing 9*, 669–686.

CGAL, 2007. Computational Geometry Algorithms Library. http://www.cgal.org.

FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. *ACM Trans. Graph. 21*, 3, 249–256.

FINLAYSON, G. D., HORDLEY, S. D., AND DREW, M. S. 2002. Removing shadows from images. In *Proc. ECCV*, Springer-Verlag, London, UK, vol. IV, 823–836.

FLOATER, M. S., KÓS, G., AND REIMERS, M. 2005. Mean value coordinates in 3d. *Comput. Aided Geom. Des. 22*, 7, 623–631.

FLOATER, M. S. 2003. Mean value coordinates. *Comput. Aided Geom. Des. 20*, 1, 19–27.

GEORGIEV, T. 2004. Photoshop healing brush: a tool for seamless cloning. *In Workshop on Applications of Computer Vission (ECCV 2004)*, 1–8.

HANRAHAN, P., SALZMAN, D., AND AUPPERLE, L. 1991. A rapid hierarchical radiosity algorithm. *Computer Graphics (SIGGRAPH '91 Proceedings) 25*, 4 (July), 197–206.

HORMANN, K., AND FLOATER, M. S. 2006. Mean value coordinates for arbitrary planar polygons. *ACM Transactions on Graphics 25, 4*, 1424–1441.

JIA, J., SUN, J., TANG, C.-K., AND SHUM, H.-Y. 2006. Drag-and-drop pasting. *ACM Trans. Graph. 25*, 3 (July), 631–637.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph. 26*, 3, 71.

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24*, 3, 561–566.

KAZHDAN, M. M., AND HOPPE, H. 2008. Streaming multigrid for gradient-domain operations on large images. *ACM Trans. Graph 27*, 3.

LAND, E. H., AND MCCANN, J. J. 1971. Lightness and Retinex Theory. *J. Opt. Soc. Amer. 61* (Jan.), 1–11.

LANGER, T., AND SEIDEL, H.-P. 2008. Higher order barycentric coordinates. *Computer Graphics Forum (Eurographics 2008) 27*, 2, 459–466.

LEVIN, A., ZOMET, A., PELEG, S., AND WEISS, Y. 2004. Seamless image stitching in the gradient domain. In *Proc. ECCV*, Springer-Verlag, vol. IV, 377–389.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2008. A closed-form solution to natural image matting. *IEEE Trans. Pattern Anal. Mach. Intell. 30*, 2, 228–242.

MCCANN, J., AND POLLARD, N. S. 2008. Real-time gradient-domain painting. *ACM Transactions on Graphics (SIGGRAPH 2008) 27*, 3 (Aug.).

PALMER, S. E. 1999. *Vision Science: Photons to Phenomenology*. The MIT Press, May.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Trans. Graph. 22*, 3, 313–318.

SUN, J., JIA, J., TANG, C.-K., AND SHUM, H.-Y. 2004. Poisson matting. *ACM Trans. Graph. 23*, 3, 315–321.

SZELISKI, R. 2006. Locally adapted hierarchical basis preconditioning. *ACM Trans. Graph 25*, 3, 1135–1143.

WACHPRESS, E. L. 1975. *A Rational Finite Element Basis*. Academic Press, New York.

WANG, J., AND COHEN, M. F. 2007. Optimized color sampling for robust matting. In *Proc. CVPR*, 1–8.

WANG, H., RASKAR, R., AND AHUJA, N. 2004. Seamless video editing. In *Proc. ICPR '04*, IEEE Computer Society, Washington, DC, USA, vol. 3, 858–861.

WARREN, J. 1996. Barycentric coordinates for convex polytopes. *Advances in Computational Mathematics 6*, 2, 97–108.

WEISS, Y. 2001. Deriving intrinsic images from image sequences. In *Proc. ICCV*, 68–75.