Multiresolution Painting and Compositing

Deborah F. Berman *Jason T. Bartell* *David H. Salesin*

Department of Computer Science and Engineering
University of Washington
Seattle, Washington 98195

Abstract

We describe a representation for *multiresolution images*—images that have different resolutions in different places—and methods for creating such images using painting and compositing operations. These methods are very easy to implement, and they are efficient in both memory and speed. Only the detail present at a particular resolution is stored, and the most common painting operations, "over" and "erase," require time proportional only to the number of pixels displayed. We also show how *fractional-level zooming* can be implemented in order to allow a user to display and edit portions of a multiresolution image at any arbitrary size.

CR Categories and Subject Descriptors: I.3.2 [Computer Graphics]: Picture/Image Generation — Display Algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques — Interaction Techniques.

Additional Key Words: compositing, infinite-resolution, multiresolution images, painting, wavelets, zooming.

1 Introduction

When editing images, it is important to be able to make sweeping changes at a coarse resolution, as well as to do fine detail work at high resolution. Ideally, the storage cost of the resulting image should be proportional only to the amount of detail present at each resolution; furthermore, the time complexity of the editing operations should be proportional only to the resolution at which the operation is performed. In addition, the user should be able to zoom in to the image to an arbitrary resolution, and to work at any convenient scale.

In this paper, we describe a very simple image painting and compositing system that meets these goals in large part. The system makes use of a Haar wavelet decomposition of the image, which is stored in a sparse quadtree structure. This wavelet representation has many advantages. First, the wavelet representation itself is concise in that it contains the same number of wavelet coefficients as there are pixels in the original image. Second, this representation supports compositing more efficiently than image pyramids. Finally, wavelets can also be used to achieve high compression rates on images [2]. By making use of a wavelet representation on-line, the editing system can be used to operate on compressed images directly, without first having to uncompress and then recompress afterward, making it much more practical for handling large images than a pyramid-based scheme.

The *multiresolution images* produced by our system can be thought of as having different resolutions in different places. There are many applications of these multiresolution images, including:

- Interactive paint systems, allowing an artist to work on a single image at various resolutions.
- Texture mapping, allowing portions of a texture that will be seen up close to be defined in more detail.
- Satellite and other image databases, allowing overlapping images created at different resolutions to be coalesced into a single multiresolution image.
- Storing and viewing the results of "importance-driven" physical simulations [9], which may be computed to different resolutions in different places.
- Virtual reality, hypermedia, and games, allowing for image detail to be explored interactively, using essentially unlimited degrees of panning and zooming.
- Supporting the "infinite desktop" user-interface metaphor [5], in which a single virtual "desktop" with infinite resolution is presented to the user.

The idea of using wavelets for multiresolution painting has also been explored simultaneously but independently by Perlin and Velho [6]. The system we describe differs from theirs in many respects, the most significant being the use of a Haar wavelet basis, support for fractional-level zooming and editing and a variety of compositing operations, and a different use of lazy evaluation in the algorithms employed.

In Section 2, we describe our multiresolution painting and compositing algorithm in detail. The algorithm is very simple, although its derivation requires some fairly sophisticated mathematics, which is deferred to Appendix A. In Section 3, we give examples of how the system can be used. Finally, in Section 4, we suggest directions for future work.

2 Algorithm

Here, we briefly describe a set of data structures and algorithms to support multiresolution painting and compositing.

2.1 Definitions and data structures

Let $\mathcal{I}$ be a *multiresolution image*—that is, an image with different resolutions in different places. One could think of $\mathcal{I}$ as an image whose resolution varies adaptively according to need.

More formally, we will define $\mathcal{I}$ as a sum of piecewise-constant functions $\mathcal{I}^j$ at different resolutions $2^j \times 2^j$. In this sense, $\mathcal{I}$ can be thought of as having "infinite" resolution everywhere: a user zooming into $\mathcal{I}$ would see more detail as long as higher-resolution detail is present; once this resolution is exceeded, the pixels of the finest-resolution image would appear to grow as larger and larger constant-colored squares.

We store the multiresolution image $\mathcal{I}$ in a sparse quadtree structure Q. The nodes of Q have the usual correspondence with portions of the image: the root of Q, at level 0, corresponds to the entire image; the root's four children, at level 1, correspond to the image's four quadrants; and so on, down the tree. Thus, each level j of quadtree Q corresponds to a scaled version of multiresolution image $\mathcal{I}$ at resolution $2^j \times 2^j$. Note that, by the usual convention, "higher" levels of the quadtree correspond to lower-resolution versions of the image, and vice versa.

Each node of the quadtree contains the following information:

```
**type** QuadTreeNode = **record**
    $d_i$: **array** $i \in [1,3]$ **of** RGBA
    $\tau$: real
    $child[i]$: **array** $i \in [1,4]$ **of pointer to** QuadTreeNode
**end record**
```

The d_i values in the *QuadTreeNode* structure describe how the colors of the children deviate from the color of the parent node. We will call these d_i values the *detail coefficients*. These coefficients allow us to compute the RGBA colors of the four children, given the color of the parent, as described in Section 2.2.1. We will refer to the "alpha" component of a color c or detail coefficient d_i as $c.\alpha$ or $d_i.\alpha$. The τ value represents the *transparency* of the node, initialized to 1. The τ values are used to optimize the painting and compositing algorithm, as explained later. The $child[i]$ fields are pointers to the four children nodes. Some of these may be null. To optimize storage, the $child[i]$ fields can alternatively be represented by a single pointer to an array of four children.

Note that each node N of the tree corresponds to a particular region of the display. We will denote this region by $A(N)$. The value $A(N)$ is determined implicitly by the structure of the quadtree and the particular view, and does not need to be represented explicitly in N. Except when displaying at fractional levels (Section 2.4), there is a one-to-one correspondence between pixels on the display and nodes at some level j in the quadtree.

The quadtree itself is given by:

```
**type** QuadTree = **record**
    $c$: RGBA
    $root$: **pointer to** QuadTreeNode
**end record**
```

The c value in the quadtree structure supplies the color of the root node; it corresponds to an average of all the colors in the image $\mathcal{I}$.

The quadtree is sparse in that it contains no leaves with detail coefficients that are all 0. Thus, the constant portions of the image at any particular resolution are represented implicitly. This convention allows us to support infinite resolutions in a finite structure. It also allows us to explicitly represent high-resolution details only where they actually appear in the image.

2.2 The algorithm

Multiresolution painting is easy to implement. The main loop involves three steps: *Display*, *Painting*, and *Update*.

2.2.1 Display

An image at resolution $2^j \times 2^j$ is displayed by calling the following recursive *Display* routine once, passing it the root and color of the overall quadtree:

```
**procedure** Display($N$ : QuadTreeNode; $c$ : RGBA):
    $c_1 \leftarrow c + d_1 + d_2 + d_3$
    $c_2 \leftarrow c - d_1 + d_2 - d_3$
    $c_3 \leftarrow c + d_1 - d_2 - d_3$
    $c_4 \leftarrow c - d_1 - d_2 + d_3$
    **for** $i \leftarrow 1$ to 4 **do**
        **if** $N$ is a leaf **or** $N$ is at level $j-1$ **then**
            Draw $c_i$ over the region $A(child[i])$
        **else**
            Display($child[i]$, $c_i$)
        **end if**
    **end for**
**end procedure**
```

For clarity, the pseudocode above recurses to level $j-1$ for the entire image; in reality, it should only recurse within the bounds of the portion of the image that fits in the display window. Note that if m pixels are displayed, the entire display operation takes just $O(m)$ time. (More precisely, the operation requires $O(m+j)$ time; however, since $j \ll m$ in almost any practical situation, we will ignore this dependency on j in the analyses that follow.)

2.2.2 Painting

Painting is implemented by compositing the newly-painted foreground buffer $\mathcal{F}$ with the background buffer $\mathcal{B}$ produced by *Display*, to create a new result image $\mathcal{R}$. We support several binary compositing operations: "over," which places new paint wherever it is applied; "under," which places paint only where the background is transparent; and "in," which places paint only where the background is already painted. We also support a unary "erase" operation, which removes paint from the background. The compositing algebra was originally described by Porter and Duff [7], and first described in the context of digital painting by Salesin and Barzel [8].

No special routines are required to implement painting itself. The only difference with ordinary painting is that in addition to the composited result $\mathcal{R}$, we must keep a separate copy of the foreground buffer $\mathcal{F}$, which contains all of the newly applied paint. This foreground buffer is necessary for updating the quadtree, as described in the next section. Ordinary painting proceeds until the user either changes the painting operation (for example, from "over" to "under"), or changes the view by panning or zooming. Either of these operations triggers an "update."

2.2.3 Update

The "update" operation is used to propagate the results of the painting operation to the rest of the multiresolution image, as represented by the quadtree. The update involves two steps: *decomposition*, in which the changes are propagated to all higher levels of the quadtree; and *extrapolation*, in which the changes are propagated to all the lower levels. We will consider each of these in turn.

Let j be the level at which the user has been painting, and let $c_r(x,y)$ be the color of each modified pixel in the result image $\mathcal{R}$. A decomposition of the entire image is performed by calling the following *Decompose* function once, passing the root of the quadtree $Q.root$ as an argument, and storing the result in $Q.c$:

```
function $Decompose(N : QuadTreeNode)$:
    if $N$ is at level $j$ then
        return $c_r(x, y)$
    end if
    for $i \leftarrow 1$ to $4$ do
        $c_i \leftarrow Decompose(child[i])$
    end for
    $d_1 \leftarrow (c_1 - c_2 + c_3 - c_4)/4$
    $d_2 \leftarrow (c_1 + c_2 - c_3 - c_4)/4$
    $d_3 \leftarrow (c_1 - c_2 - c_3 + c_4)/4$
    return $(c_1 + c_2 + c_3 + c_4)/4$
end function
```

For clarity, the pseudocode above assumes that the sparse quadtree Q already contains all of the nodes corresponding to the pixels in the result image $\mathcal{R}$; however, if $\mathcal{R}$ has been painted at a higher resolution than the existing image, then new nodes may have to be allocated and added to Q as part of the traversal. Furthermore, for efficiency, the *Decompose* function should be modified to only recurse in regions of the multiresolution image where changes have actually been made. Note that if the portion of the image being edited has m pixels, then the entire decomposition operation takes $O(m)$ time.

Extrapolation is a bit more complicated, and depends on the particular compositing operation used. For binary painting operations, let $c_f(x, y)$ be the color of the foreground image $\mathcal{F}$ at each pixel (x, y), and let $c_f.\alpha(x, y)$ be the pixel's alpha (opacity) value. For the "erase" operation, let $\delta(x, y)$ be the opacity of the eraser at each pixel. Extrapolation can then be performed by calling the following routine for the node N corresponding to each modified pixel (x, y) of the edited image:

```
procedure $Extrapolate(N : QuadTreeNode)$:
    for $i \leftarrow 1$ to $3$ do
        switch on the compositing operation
            case "over":
                $d_i \leftarrow d_i * (1 - c_f.\alpha(x, y))$
            case "under":
                $d_i \leftarrow d_i - d_i.\alpha * c_f(x, y)$
            case "in":
                $d_i \leftarrow d_i * (1 - c_f.\alpha(x, y)) + d_i.\alpha * c_f(x, y)$
            case "erase":
                $d_i \leftarrow d_i * (1 - \delta(x, y))$
        end switch
    end for
    if $N$ is not a leaf then
        for $i \leftarrow 1$ to $4$ do
            $Extrapolate(child[i])$
        end for
    end if
end procedure
```

Note that the extrapolation procedure takes time proportional to the amount of detail that appears "below" the modified parts of the image. In order to optimize this operation, at least for the most common cases of painting "over" and erasing, we can use a form of lazy evaluation. First, observe that the two formulas for "over" and "erase" in the pseudocode above merely multiply the existing detail coefficients by some constant, which we will call $\tau(x, y)$. (For painting "over," $\tau(x, y) = 1 - c_f.\alpha(x, y)$; for erasing, $\tau(x, y) = 1 - \delta(x, y)$.) Thus, for these two operations, rather than calling the *Extrapolate* procedure for each node N, we can instead just multiply the value $N.\tau$ stored at the node by $\tau(x, y)$. Later, if and when the d_i values for a node N are actually required, they can be lazily updated by multiplying each $N.d_i$ with the τ values of all of the node's ancestors. This product is easily performed as part of the recursive evaluation. This very simple form of lazy evaluation is a by-product of the underlying wavelet representation for the image, since the detail coefficients at higher resolutions depend only on the product of the opacities of all the paint applied and removed at lower resolutions. Any sort of lazy evaluation method would be much more complicated with image pyramids, since the high-resolution colors have a much more complex dependence on the colors of the paint applied and removed at lower resolutions. Note also that color correction, an important image operation, is a special case of compositing "over," and so can be performed on an arbitrarily high-resolution image in time proportional only to the resolution actually displayed.

2.3 Boundary conditions

Treating boundary conditions correctly introduces a slight complication to the update and display algorithms described in the sections above. The difficulty is that the *Decompose* function needs to have available to it the colors of the children of every node N that it traverses. However, some of these child nodes correspond to regions that are outside the boundary of the window in which the user has just painted, and therefore are not directly available to the routine. The obvious solution is to store color information in addition to the detail coefficients at every node of the quadtree; however, this approach would more than double the storage requirements of the quadtree, as well as introduce the extra overhead of maintaining redundant representations. Instead, we keep a temporary auxiliary quadtree structure of just the colors necessary for the decomposition; this structure can be filled in during the *Display* operation at little extra cost. The size of this auxiliary structure is just $O(m)$.

2.4 Display and editing at fractional resolutions

So far, we have assumed a one-to-one correspondence between the nodes of the quadtree at level j and the pixels of the image at resolution $2^j \times 2^j$. Since the levels of the quadtree are discrete, this definition only provides for discrete levels of zooming in which the resolution doubles at each level.

From a user-interface point of view, it would be better to be able to zoom in continuously on the multiresolution image being edited. A kind of *fractional-level* zooming can be defined by considering how the square region $A(N)$ corresponding to a given node N at level j in the quadtree would grow as a user zoomed in continuously from level j to $j+1$ to $j+2$. The size of $A(N)$ would increase exponentially from width 1 to 2 to 4 on the display. Thus, when displaying at a fractional level $j + t$, for some t between 0 and 1, we would like $A(N)$ to have size $2^t \times 2^t$.

On workstations that provide antialiased polygon drawing, this fractional zooming is implemented quite simply by drawing each node N as a single square of the proper fractional size. On less expensive workstations that support only integer-sized polygons efficiently, a slightly less pleasing but still adequate display can be achieved by rounding each rendered square to the nearest pixel. In either case, the only change to the *Display* routine is to bottom out the recursion whenever N is at level $\lceil j + t - 1 \rceil$ instead of at level $j - 1$, and to let the region $A(child[i])$ correspond to the appropriate fractional size.

Of course, from a user's standpoint, if it is possible to display an image at any level $j + t$, then it should also be possible to edit it at that level. This fractional-level editing is also easy to support. To update the quadtree representation, we simply rescale the buffer of newly painted changes $\mathcal{F}$ to the size of the next higher integer level, as if the user had painted the changes at level $j + 1$; the scaling factor required is 2^{1-t}. We can then perform the same update as before, starting from level $j + 1$.

3 Results

Figure 1 demonstrates our system with three examples.

In the first example (a)–(d), the user zooms into an image of Mona Lisa (a) and paints on some eye shadow and lipstick at higher resolution (b). To add a glint in the eye, the user zooms in slightly closer (c). Note that any (continuous) level of zooming is supported, so the user need not know anything about the underlying representation, which is actually discrete. The retouched image is then displayed at the original resolution (d).

In the second example (e)–(h), the user paints a tree at multiple resolutions, using different kinds of compositing operations. Most of the tree was painted at a coarse resolution. In the first frame (e), the user zooms way into the upper left corner of the tree and paints some leaves. In the second frame (f), the user zooms out to a coarser scale and changes the color of the leaves, using an "in" brush that only paints where paint has previously been applied. In the third frame (g), the user zooms out to a very coarse resolution and quickly roughs in the sky and grass, using an "under" brush that only deposits color where no paint already appears. Note that even though the sky color is applied coarsely, the new paint respects all of the high-resolution detail originally present in the image (h).

In the third example (i)–(l), we have created a single multiresolution image out of six successive images from the book, *Powers of Ten* [4], by compositing the images together at different scales. In the book, each of the images is a $10\times$ higher-resolution version of the central portion of its predecessor. In our multiresolution system, these six images become a single image with a 10^5 range of scale. (Note that representing power-of-10 images in our power-of-2 quadtree requires the fractional-level editing capability.) The first frame (i) shows a close-up of the innermost detail. The second frame (j) shows the image after zooming out by a factor of 100,000. In the third frame (k), the user retouches the low-resolution image using an "over" brush to give the impression of smog. This smog affects all of the closer views without eliminating any of the detail present, as demonstrated in the final frame (l).

4 Future work

There are many directions for future research, including:

Compression. Wavelet image transforms are most commonly used for image compression [2]. Our system already performs a simple kind of lossless compression by pruning any branches of the quadtree whose detail coefficients are all 0. We would also like to incorporate lossy image compression as part of our system. As a further extension, this compression could be applied interactively, with the user selecting increased compression ratios in the less vital parts of the image.

Progressive refinement. Another advantage of the wavelet representation is that it provides a natural ordering of the detail coefficients with respect to either L^2 (least squares) or L^∞ (max error) metrics. For example, the best L^2 approximation to an image using m coefficients is given by the largest m detail coefficients, assuming proper normalization of the basis functions. These largest coefficients, drawn as flat-shaded rectangles using polygon rendering and accumulation hardware, could be used to provide a fast indication of the image during interactive panning and zooming. The image could then be updated progressively from its most important to least important details.

User-interface paradigms. Multiresolution images can encode a great deal of complexity. New user-interface paradigms may therefore be required for navigating them. One useful tool would indicate the amount of detail present at different places of the image. Another would provide some measure of context when zoomed far into an image. For moving around, we would like to experiment with a movement akin to "flying," in which the user navigates through a large multiresolution image by smoothly zooming out, panning across, and zooming back in.

Automatic synthesis of detail. A fairly straightforward modification to our system would allow it to generate more detail procedurally whenever the user zoomed into an image, allowing for images with essentially infinite detail, such as fractals.

3-D and video. We would like to extend this work to three dimensions, allowing direct volumetric painting at arbitrary scales. We would also like to investigate the possibilities of multiresolution video, in which the temporal resolution of an animation might be varied to provide detailed slow-motion sequences, or to provide a low-bandwidth preview mechanism.

Antialiasing and higher-order wavelets. One drawback of the simple Haar-basis painting system described in this paper is that when the user zooms into an area where there is no further detail, the pixels of the lower-resolution image are displayed as large constant-colored squares. A number of possibilities exist for alleviating this problem. One approach would be to perform some kind of filtering on the displayed image so as to hide the pixel-replication artifacts; however, the inconsistency between the internal and external representations of the image that such an approach entails will likely be problematic. A more intriguing alternative is to extend the painting and compositing operations to higher-order wavelets, which might be used to achieve higher-order continuity across the image under any level of zooming. However, higher-order wavelets have a number of drawbacks as well. The supports of such wavelets are necessarily overlapping and larger than those of Haar wavelets, leading to a considerably more complex implementation, which is also likely to run at least an order of magnitude slower. More importantly, defining an accurate and basis-independent compositing operation appears to require that the wavelet basis be closed under products, which is not true of any higher-order wavelet basis of which we are aware. This requirement is discussed in more detail in Appendix A.

Acknowledgements

We would like to thank Tony DeRose for helpful discussions during the development of these ideas. This work was supported by an NSF National Young Investigator award (CCR-9357790), by the University of Washington Graduate Research and Royalty Research Funds (75-1721 and 65-9731), by an Air Force Laboratory Graduate Fellowship, and by industrial gifts from Adobe, Aldus, and Xerox.

References

[1] Charles K. Chui. *Wavelet Analysis and its Applications*, Volumes 1 and 2. Academic Press, Inc., San Diego, Califorinia, 1992.

[2] Ronald A. DeVore, Björn Jawerth, and Bradley J. Lucier. Image compression through wavelet transform coding. *IEEE Transactions on Information Theory*, 38(2):719–746, March 1992.

[3] Stephane Mallat and Sifen Zhong. Wavelet transform maxima and multiscale edges. In Ruskai et al., editor, *Wavelets and Their Applications*, pages 67–104. Jones and Bartlett Publishers, Inc., Boston, 1992.

[4] Philip Morrison, Phylis Morrison, and The Office of Charles and Ray Eames. *Powers of Ten*. Scientific American Library, New York, 1982.

[5] Ken Perlin and David Fox. Pad: An alternative approach to the user interface. Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993). In *Computer Graphics*, Annual Conference Series, 1993, pages 57–64.

[6] Ken Perlin and Luiz Velho. A wavelet representation for unbounded resolution painting. Technical report, New York University, November 1992.

[7] Thomas Porter and Tom Duff. Compositing digital images. Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July 23–27, 1984). In *Computer Graphics* 18, 3 (July 1984), pages 253–259.

[8] David Salesin and Ronen Barzel. Two-bit graphics. *IEEE Computer Graphics and Applications*, 6:36–42, 1986.

[9] Brian E. Smits, James R. Arvo, and David H. Salesin. An importance-driven radiosity algorithm. Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26–31, 1992). In *Computer Graphics* 26, 2 (July 1992), pages 273–282.

A Deriving the equations

The multiresolution paint algorithm we have described is an application of *wavelets*, a mathematical tool that has found a wide variety of applications in recent years, including image processing and compression [1, 2, 3]. In this appendix, we briefly describe how our algorithm fits into the larger context of wavelets, and we show how the formulas of Section 2 can be derived.

Let C^n be a matrix of size $2^n \times 2^n$ representing the pixel values of an image. We can associate with C^n a function $\mathcal{I}^n(x,y)$ given by

$$\mathcal{I}^n(x,y) \quad = \quad \Phi^n(y)\, C^n\, \Phi^n(x)^{\mathrm{T}}$$

where $\Phi^n(x)$ is a row matrix of basis functions $[\phi_1^n(x), \ldots, \phi_{2^n}^n(x)]$, called *scaling functions*. In our application, we use the *Haar basis*, in which each scaling function $\phi_i^n(x)$ is given by

$$\phi_i^n(x) \quad = \quad \begin{cases} 1 & \text{for } 0 \le 2^n x - i < 1 \\ 0 & \text{otherwise} \end{cases}$$

The wavelet transform allows us to decompose C^n into a lower-resolution version C^{n-1} and detail parts D_1^{n-1}, D_2^{n-1}, and D_3^{n-1}, using matrix multiplication as follows:

$$\begin{aligned} C^{n-1} &= A^n\, C^n\, (A^n)^{\mathrm{T}} && (1) \\ D_1^{n-1} &= A^n\, C^n\, (B^n)^{\mathrm{T}} && (2) \\ D_2^{n-1} &= B^n\, C^n\, (A^n)^{\mathrm{T}} && (3) \\ D_3^{n-1} &= B^n\, C^n\, (B^n)^{\mathrm{T}} && (4) \end{aligned}$$

In the Haar basis, the matrices A^n and B^n are given by:

$$A^n = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 & \cdots & & 0 \\ 0 & 0 & 1/2 & 1/2 & & & \\ \vdots & \vdots & & & \ddots & & \\ 0 & 0 & & & \cdots & 1/2 & 1/2 \end{bmatrix}$$

$$B^n = \begin{bmatrix} 1/2 & -1/2 & 0 & 0 & \cdots & & 0 \\ 0 & 0 & 1/2 & -1/2 & & & \\ \vdots & \vdots & & & \ddots & & \\ 0 & 0 & & & \cdots & 1/2 & -1/2 \end{bmatrix}$$

The detail coefficients d_i at level j in our algorithm are the entries of the D_i^j matrix. Thus, equations (1)–(4) provide the expressions used in the *Decompose* routine.

The four decomposed pieces can also be put back together again, using two new matrices P^n and Q^n:

$$\begin{aligned} C^n \quad = \quad & P^n\, C^{n-1}\, (P^n)^{\mathrm{T}} \; + \; P^n\, D_1^{n-1}\, (Q^n)^{\mathrm{T}} \\ & + Q^n\, D_2^{n-1}\, (P^n)^{\mathrm{T}} \; + \; Q^n\, D_3^{n-1}\, (Q^n)^{\mathrm{T}} \end{aligned}$$

This equation provides the expressions used in the *Display* routine. In the Haar basis, these matrices are given by $P^n = 2(A^n)^{\mathrm{T}}$ and $Q^n = 2(B^n)^{\mathrm{T}}$.

The original function $\mathcal{I}^n(x,y)$ can be expressed in terms of the lower-resolution pixel values C^{n-1} and detail coefficients D_i^{n-1} using a new set of basis functions $\Psi^j = [\psi_1^j(x), \ldots, \psi_m^j(x)]$, called *wavelets*, as follows:

$$\begin{aligned} \mathcal{I}^n(x,y) \quad = \quad & \Phi^{n-1}(y)\, C^{n-1}\, \Phi^{n-1}(x)^{\mathrm{T}} && (5) \\ & + \Phi^{n-1}(y)\, D_1^{n-1}\, \Psi^{n-1}(x)^{\mathrm{T}} \\ & + \Psi^{n-1}(y)\, D_2^{n-1}\, \Phi^{n-1}(x)^{\mathrm{T}} \\ & + \Psi^{n-1}(y)\, D_3^{n-1}\, \Psi^{n-1}(x)^{\mathrm{T}} \end{aligned}$$

In the Haar basis, there are $m = 2^j$ wavelets in Ψ^j, and each $\psi_i^j(x)$ is given by:

$$\psi_i^j(x) \quad = \quad \begin{cases} 1 & \text{for } 0 \le 2^j x - i < 1/2 \\ -1 & \text{for } 1/2 \le 2^j x - i < 1 \\ 0 & \text{otherwise} \end{cases}$$

Decomposing the first term $\Phi^{n-1}(y) C^{n-1} \Phi^{n-1}(x)^{\mathrm{T}}$ of equation (5) recursively allows us to represent a function $\mathcal{I}^n(x,y)$ in its *wavelet basis*, given by the row matrix

$$\left[\ \Phi^0 \ \middle|\ \Psi^0 \ \middle|\ \cdots \ \middle|\ \Psi^{n-1}\ \right].$$

In order to derive the expressions used for compositing detail coefficients in the *Extrapolate* routine, we must begin by defining compositing operations on functions $\mathcal{F}$, $\mathcal{B}$, and $\mathcal{R}$, built from the pixel values C_f, C_b, and C_r of the foreground, background, and result images:

$$\begin{aligned} \mathcal{F}^j(x,y) &= \Phi^j(y)\, C_f^j\, \Phi^j(x)^{\mathrm{T}} \\ \mathcal{B}^n(x,y) &= \Phi^n(y)\, C_b^n\, \Phi^n(x)^{\mathrm{T}} \\ \mathcal{R}^n(x,y) &= \Phi^n(y)\, C_r^n\, \Phi^n(x)^{\mathrm{T}} \end{aligned}$$

Note that the foreground image has its highest-resolution components in level j, the level at which the user is painting, whereas the background and resulting images have components in a potentially higher-resolution level n.

For example, the "over" operation can be defined on functions $\mathcal{F}$, $\mathcal{B}$, and $\mathcal{R}$ as follows:

$$\mathcal{R}^n(x,y) \quad = \quad \mathcal{F}^j(x,y) \; + \; (1 - \mathcal{F}^j.\alpha(x,y)) * \mathcal{B}^n(x,y)$$

The expressions for compositing detail coefficients can be derived by writing each function in its wavelet basis, multiplying out, and regrouping terms. The derivation is tedious, but the final expressions are quite simple, as the pseudocode for the *Extrapolate* routine attests.

Note that compositing multiresolution images, as defined here at least, requires taking products of basis functions. While the Haar basis is closed under products, we know of no other finite wavelet basis that has this property. Proving or disproving the existence of non-trivial finite wavelet bases that are closed under products is an interesting (and, as far as we know, open) theoretical question, which this research in compositing multiresolution images suggests.