

# Design Memorandum

---

Date: Monday, April 21, 2008

Total Pages: 4

Subject: Key Abstractions in the Correction Operation for Hoppe & Lefebvre Texture Synthesis

---

**Refinement Level:** a zero-based index measuring the relative coarseness or fineness of synthesis images. The work image, the coordinate image, and the source image are all taken to exist not as flat, two-dimensional pixel matrices, but as multiscale, three-dimensional pyramids of 2D pixel matrices. A pixel in such a multiscale image  $I$  must thus be identified not by a simple ordered double  $(x, y)$  describing its 2D position in the image matrix, but by an ordered triple  $(l, x, y)$ , naming the pixel in matrix position  $(x, y)$  at refinement level  $l$ . Given a flat, square, two-dimensional input image  $I_{\text{pre}}$  of dimension  $m \times m$  pixels, we construct  $I$  by progressively downsampling  $I_{\text{pre}}$  by two and smoothing away the highest-frequency information. The per-level subimages of  $I$ ,  $I(l)$  for  $l = 0 \dots L$  are each flat, square, two-dimensional images. The most refined level subimage  $I(L)$  in  $I$ , is equivalent to  $I_{\text{pre}}$  and thus has dimension  $m \times m$  pixels. For any level  $l = 0 \dots (L - 1)$ ,  $I(l)$  has dimension  $\dim(I(l)) = \dim(I(l + 1)) / 2$  pixels. Since level dimensions are restricted to whole numbers, given an image  $I_{\text{pre}}$  where  $\dim(I_{\text{pre}}) = m$ , the corresponding multi-scale representation of can have at most  $\log_2 m$  refinement levels.

**Level Subimage:** let  $I$  be an image encoded as a three-dimensional multiscale representation over several refinement levels, as described in “Refinement Level.” The  $l$ -th level subimage,  $I(l)$  of  $I$  is a flat, square, two-dimensional pixel matrix of dimension  $2^l$  pixels, for  $l = 0 \dots L$ .

**Source Image:** denoted *srcimg*, the source image is a three-dimensional, multiscale representation of the artistic brush sample provided by the user. The  $l$ -th level subimage of the source image is denoted *srcimg*( $l$ ), and the pixel at location  $(x, y)$  in the  $l$ -th level subimage of the source image is *srcimg*( $l, x, y$ ).

**Work Image:** denoted *working*, the work image is a three-dimensional multiscale representation of the intermediate synthesis results image. The  $l$ -th level subimage of the work image is denoted

working( $l$ ), and the pixel at location  $(x, y)$  in the  $l$ -th level subimage of the source image is working( $l, x, y$ ). The algorithm synthesizes the work image from lower to higher levels, from coarser to finer detail. The level subimage for level  $l = L$  constitutes the overall output of the algorithm. Unlike the source image and the coordinate image, the work image exists as an explicit matrix of pixel values only on final output. During intermediate synthesis phases, the values of work image pixels exist only implicitly, as indirections into the source image on coordinate image pixel values. The value of the work image pixel at location  $(x, y)$  on level  $l$  is obtained by first referencing  $(l, x, y)$  up in the coordinate image, then using the value of this lookup as a pointer (or index) into the source image. Thus working( $l, x, y$ ) = srcimg(coording( $x, y, l$ ).red, coording( $x, y, l$ ).green,  $l$ ).

**Coordinate Image:** denoted coording, the coordinate image is a three-dimensional, multiscale image whose pixel values are taken not as colors, but as source image coordinates. As with all pixels, a coordinate image pixel  $p$  at location  $(x, y)$  on level  $l$  has as its value a color-component quadruple, coording( $x, y, l$ ) = ( $p$ .red,  $p$ .green,  $p$ .blue,  $p$ .alpha). By packing the red and green channels of  $p$  with the  $(x, y)$ -coordinates of some source image pixel  $q$ , we enable indirection into the source image, so that the coordinate image becomes an array of pointers into the source image. By performing two lookups in succession, first into the coordinate image and then into the source image, we derive the work image, as described above.

**Pixel Neighborhood:** a square pixel matrix of odd dimension, with a distinguished origin pixel at its center. Given a pixel  $p$ , we denote the pixel neighborhood of dimension  $d$  with  $p$  at its origin as  $N(p, d)$ .

**Perceptual Distance:** let  $n_1$  and  $n_2$  be two pixel neighborhoods of identical dimension  $d$  centered about pixels  $p_1$  and  $p_2$  respectively, such that  $n_1 = N(p_1, d)$  and  $n_2 = N(p_2, d)$ , we can calculate the perceptual distance  $\Delta$  between them as  $\Delta = \text{pdist}(n_1, n_2)$ . Metric  $\Delta$  provides a rough measure of how visually similar  $n_1$  and  $n_2$  appear to a human observer. We compute  $\Delta$  as the  $\ell^2$  norm of the vector of color component differences between corresponding pixels of  $n_1$  and  $n_2$ . Note that the terms in the norm's sum of squares may be weighted differently relative to each other, depending on the entries in the neighborhood mask.

**Neighborhood Mask:** a  $d \times d$  square matrix of real numbers, whose entries all fall within the closed interval  $[0, 1]$  that is used to partially correct the smoothing of differences introduced by employing the  $\ell^2$  norm as a perceptual distance metric. When computing the perceptual distance between two pixel neighborhoods, we want differences between the neighborhoods' center pixels to affect the computed distance relatively more than differences between their edge pixels. The neighborhood mask furnishes weights that determine how much each such pixel difference affects overall perceptual distance. For example, when computing the perceptual difference between two pixel

neighborhoods, the difference between the two origin pixels might be weighted 1.0, whereas the difference between the pixels at (1, 0) might only be weighted 0.85, and the difference between the two pixels at (2, 2) might be weighted a mere 0.52. These weights are applied to the terms of the sum of squares in the norm calculation before the square root is taken.

**Candidate Set:** the correction operation works by replacing each pixel in the work image independently with a *substitution pixel* sampled from the source image. For each work image pixel, we consider several distinct source image pixels before choosing one to serve as the substitution pixel. The set of pixels considered as substitutes for a work image pixel  $p$  is called the *candidate set* of  $p$ , and is given by  $C(p) = \{s_1, s_2, \dots, s_n\}$ , where each  $s_i$  is a pixel in the source image. This candidate set is partitioned into the proximity set  $C_P(p)$  and jump set  $C_J(p)$ , such that  $C_P(p) \cup C_J(p) = C(p)$ .

**Proximity Set:** given a work image pixel  $p$  under correction, we build a pixel neighborhood  $n$  of dimension  $d$  about  $p$ , such that  $n = N(p, d)$ . Now, consider the  $d^2$  pixels  $p_i$  for  $i = 0 \dots (d^2 - 1)$  in  $n$ . Each  $p_i$  has some position in  $n$  relative to  $p$ , and conversely  $p$  has some position shift relative to each  $p_i$ . For example, the pixel  $p_k$ , located one pixel position above and one pixel position to the right of  $p$  in  $n$  is shifted  $(-1, -1)$  relative to  $p_k$ . Given  $p$ , we form the proximity set of  $p$ ,  $C_P(p)$ , as follows. For each  $p_i$  in the neighborhood about  $p$ , determine its corresponding source image pixel  $s_i$ . For each such  $s_i$ , accumulate into  $C_P(p)$  the source image pixel  $s$  shifted relative to  $s_i$  in the same way that  $p$  is shifted relative to  $p_i$ .

**Jump Set:** denoted  $C_J(p)$ , the jump set of a work image pixel  $p$  is built by accumulating pixels whose neighborhoods are perceptually similar to those about the pixels of  $C_P(p)$ .  $C_J(p)$  is built from  $C_P(p)$  as follows. For each source image pixel  $s_i$  in  $C_P(p)$ , build a neighborhood  $n_i$  of dimension  $d$  about  $s_i$ . Then, for each of these  $n_i$ , enumerate the  $k$  source image neighborhoods  $n_{i,j}$ , for  $j = 0 \dots (k - 1)$  most perceptually similar to  $n_i$ . Then, accumulate the origin pixels of such each  $n_{i,j}$  into  $C_J(p)$ . Counting any duplicates,  $|C_J(p)| = k|C_P(p)|$ .

**Jump Penalization Parameter:** denoted  $\mathcal{K}$  (“script k”) the jump penalization parameter ensures that members of the proximity set are favored over members of the jump set in substitution pixel selection. During correction, each work image pixel  $p$  is replaced by its substitution pixel  $s_p$ , where  $s_p$  is chosen over all the elements of  $C(p)$  as the pixel that minimizes the perceptual distance between its neighborhood and the neighborhood about  $p$ , such that  $s_p = \min_{c \in C(p)} [\text{pdist}(N(c, d), N(p, d))]$ . However, the  $\ell^2$  norm is an imperfect measure of perceptual distance, and pixels chosen from the proximity set often yield better results than pixels chosen from the jump set, even if the jump set pixel’s neighborhood is closer in  $\ell^2$  distance. To correct for this, we multiply  $\text{pdist}(c, d)$  by  $\mathcal{K}$  when  $c \in C_J$ . The value of  $\mathcal{K}$  is a real number in the interval  $[1, \infty]$ . When

$\lambda = \infty$ , jump set pixels are wholly excluded from substitution pixel consideration; when  $\lambda = 1$ , jump set pixels aren't penalized at all relative to proximity set pixels.