Image Recoloring Induced by Palette Color Associations

Gary R. Greenfield
Department of Mathematics & Computer Science
University of Richmond
Richmond, VA 23173, U.S.A.
ggreenfi@richmond.edu

Donald H. House
Visualization Laboratory
Texas A&M University
College Station, TX 77843, U.S.A.
house@viz.tamu.edu

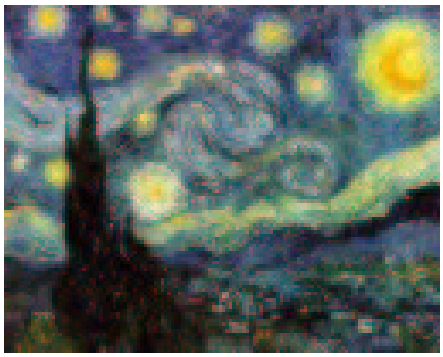
Abstract

In this paper we present a non-interactive method for recoloring a destination image according to the color scheme found in a source image. The approach is motivated by trying to invert the working process employed in oil painting, and results are demonstrated by application to several well-known oil paintings. The algorithm uses several color models, but leans most heavily on the $\ell\alpha\beta$ color space. We first color segment each image bottom-up by iteratively merging groups of pixels into connected regions of similar color. During color segmentation, a color "texture" tree is generated and associated to each region. Next, we construct classes of regions by compensating for color duplication and color similarity within the set of averaged color values obtained from regions. We extract a color palette for each image by choosing the colors of canonical region representatives from these classes. Once this palette is constructed for each image, any inverse map from the set of destination palette colors to the set of source palette colors induces a forward map from the classes of regions in the source image to sets of classes of regions in the destination image. For each source class in the range of the inverse map we transfer color from its canonical region representative to each of the associated destination regions. Color transfer occurs at the level of pixels, and uses the color texture trees associated to the regions. Our recoloring method attempts to maintain the destination image's original value structure. This is accomplished by transferring only the α and β channels from the source. To make our method computationally tractable, we work within an image pyramid, transferring color layer by layer.

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

1 Introduction

A painting's palette is responsible for how color is selected, contrasted, harmonized and blended. It exerts a powerful influence on how the viewer interprets a painting's imagery. Consider the two Impressionist paintings at the top of Figure 1. For "Starry Night", Van Gogh uses stark color contrast. The cool



Van Gogh, "Starry Night"

Cezanne, "Skulls"

Van Gogh to Cezanne

Figure 1: Two low resolution test images and an image recoloring. (Van Gogh and Cezanne from http://www.artchive.com)

saturated blues and greens of the sky and landscape are opposed to the warm saturated yellows and oranges of the moon and stars to create a visual intensity, as if the night sky were a battlefield. Compare this with Cezanne's "Skulls", where the painter smoothly blends warm reds and oranges of low saturation with just a few cooler brown elements to create a quiet glow, as if embers of life persist even though the imagery is that of death.

Now, imagine being able to somehow liquefy the color of a painting in such a way that it could be poured from one painting to another thereby transferring the chromatic "orchestration" i.e., the artistic intent of the color scheme, from one painting to the other. This would hold out the possibility that "Starry Night" could be transformed into a warm peaceful panorama, while a paradoxical energy could be imbued into the somber imagery of "Skulls". Our recent work has been inspired by the desire to be able to achieve just such an effect. A sample recoloring we made of "Skulls" from "Starry Night" is shown in Figure 1. Additional examples appear at the end of this paper.

Our goal in this paper is to investigate the problem of transferring the color scheme of a source image to a destination image and in the process learn more

about what it means to "liquefy the color of a painting" and to "transfer the artistic intent of a color scheme." The solution we describe is based on the notion that one can link an image's color organization to its underlying color palette and then, by making relevant associations between colors in the source palette and the destination palette, recolor the destination image while preserving the artistic intent of the color scheme used in the source image.

Image recoloring has potential practical applications. In movie making it could be useful for altering the mood of scenes, for rapidly prototyping set designs, or for assistance in matching CG and live-action plates. The idea could even have amusing (though possibly outrageous) commercial applications, allowing a consumer to custom order art prints whose color schemes harmonize with home or office décor. Conceivably, image recoloring could be incorporated into a photo-editing application to provide a digital effect for artistic purposes.

The idea of transferring the color content from one image to another is not new, but previous algorithms required manual selection of swatches to steer the algorithm. Moreover, images were carefully chosen so that their compositions had about the same color proportions whence, in the right color space, statistical methods could be used to effect the color transfer. In this paper we attempt to overcome such limitations by hierarchically analyzing images and computing palettes that can be used to forge color correspondences between images. Thus we fully automate the color transfer process and we consider the color transfer problem in its full generality.

In developing the conceptual framework for our color transfer method we have tried to take into consideration two ideas classically trained painters are taught: (1) Paintings should be organized by value — the pattern of lights and darks. (2) To develop a painting on canvas, first block out broad loosely defined regions of color, which determine the structure of the image, and then later refine these regions by creating image detail. These ideas furnish the two cornerstones for our design: (1) The value patterns of the destination image should survive color transfer from the source. (2) An image should be "deconstructed" using an image pyramid whose top layer or apex reveals only enough detail to determine a limited number of regions of color, and whose successively lower layers refine these regions, adding more and more image detail, until we reach the bottom or base layer which consists of the image at full resolution.

To recolor a destination image from a source image, we make "associations" between colors belonging to image palettes that are derived from the regions at the apexes of the image pyramids. These color associations are further refined by recursively descending layer by layer through the pyramid. Within each layer of the pyramid, color associations induce region pairings which, in turn, induce pixel pairings. Chromatic content is then transferred from source pixels to destination pixels. For the purposes of illustration, throughout the presentation we use the low resolution (maximum dimension of 128 pixels) "Starry Night" and "Skulls" images of Figure 1 as source and destination images.

2 Background

A color transfer method is by its very nature a non-photorealistic digital effect. In that sense it fits into the general framework of the field of Non-Photorealistic Rendering, particularly the subbranch of automatic painting systems [4]. The notion of color transfer *per se* is not widespread in the literature, although transferring the *texture* of one image to another has received considerable attention [6]. Reinhard et al [8] examine color transfer with the goal of color *correction* in mind. Their method is best suited for working with images (or portions of images) whose palettes are similar. Their results are quite dramatic and successful. A key contribution of their paper is the identification of the $\ell\alpha\beta$ color space of Ruderman et al. [9] as an ideal candidate for work in image recoloring. As a convenience, we give the details of the conversion scheme between the RGB system of computer graphics and the $\ell\alpha\beta$ system that they used (modulo a clamping step that we added) in Appendix A. Their recoloring method is statistical in nature: they modify destination pixels in such a way that upon transforming them to their color space they will have the same statistical *characteristics* as similarly transformed source pixels.

3 Color Spaces

In this paper, we work in four color spaces. The $\ell\alpha\beta$ color space has the property that for a range of color images of nature scenes the ℓ, α, and β axes are highly perceptually decorrelated. Further, the ℓ component of the space is a good measure of perceptual luminance while the α and β components measure chromatic content. Thus, $\ell\alpha\beta$ space can clearly differentiate between the notions of value and chroma. We rely on it to help organize the color content of an image and to help establish color associations between the palettes of two different images. We use RGB color space for three reasons: (1) it is needed for image input and output, (2) it does the best job of identifying extraneous colors, and (3) it determines the in-gamut color space. We use the familiar HSV color space to help identify color similarities within sets of colors from which palettes are extracted. Finally, one phase of out-of-gamut color correction is done with the help of the Y, or luminance, channel of YIQ color space.

4 Color Segmentation Preliminaries

Our primary objectives are to organize the color information within an image in such a way that it can be analyzed for artistic intent and in such a way that it can be quantitatively transferred from one image to another. These are mutually distinct and competing objectives. To discern artistic intent we need to make color comparisons on the basis of color metrics. To transfer color we need to capture the structure or "texture" of the color fields within an image. To handle both objectives simultaneously, we construct regions of pixels bottom-up using a region-merging algorithm that assigns binary trees

to regions. Colors obtained by averaging color over all the pixels within a region are used when we need to address the issue of artistic intent and binary trees are used when we need to address the issue of transferring color texture from one region to another. Bottom-up region merging is an example of an image *color segmentation* algorithm. Our two competing objectives are the reasons we do not use one of the standard color segmentation algorithms [2] [3]. Our region-merging color segmentation algorithm is in fact "reversible" and can control for the number of regions that are formed during segmentation. In this respect it resembles any one of a number of *synthesis* algorithms found in computer graphics e.g. wavelet based image representation, texture synthesis, image querying [7]. The algorithm's drawback, however, is that it is slow.

Our region-merging algorithm uses the pixel representation for a raster image shown in Figure 2 that was first suggested by Bieri and Kohler [1]. Geometrically, a *pixel* consists of a *vertex*, left and top *edges*, and a *face*. The vertex is determined by the pixel's row and column. Associated with each edge is a flag for determining whether or not the edge is currently serving as a boundary between two regions of pixels, and a time stamp for remembering which merge event caused the edge to become converted from a boundary edge to an interior edge. Each edge knows which pixel it belongs to, and associated with each face is the color information for the pixel as well as *mark* and *mask* flags needed during the execution of our recursive algorithms.

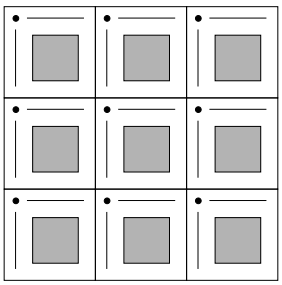


Figure 2: The geometry of pixels.

Geometrically, a *region* consists of a group of one or more pixels that are simply connected using 4-neighborhoods. As a data structure, a region is represented by a binary tree whose leaves are pixels. A region has attributes: a region identifier, area, average color, height, a flag to indicate whether or not the region is *active*, and a distinguished pixel. The distinguished pixel is used to help initialize region merging. Initially, each pixel is made into an active region which has unit area, is distinguished by the pixel's vertex, has height one, and whose *average* color is the pixel's color.

5 Color Segmentation Algorithm

Any two active adjacent regions — regions sharing a common boundary edge — are candidates to be merged to form a new active region. Any sequence of region merges will color segment an image (or a portion of an image) into a *forest* of binary trees. The left and right subtrees of a binary tree are the binary trees associated to the two child regions which were merged to form the parent region. Pixels know which active region they belong to.

During a scanline traversal, as each pixel is encountered and used to initialize a region, edges of the pixel that are boundaries with adjacent pixels are placed into a priority queue. A merge event occurs when a boundary edge is removed from the priority queue and passed to the `RegionMergeFromEdge` algorithm. This algorithm is responsible for the necessary bookkeeping to create a new active region from the two regions bounded by the edge. It invokes the recursive algorithm `UpdatePixel` to update the status of all the pixels and edges in the new active region. Figure 3 shows a pixel diagram for a merge of regions N_1 and N_2 that will be triggered by boundary edge e. It reveals that our algorithm must be able to take into account additional *shared* boundary edges such as e' and must be able to update the priorities of non-shared boundary edges such as f, g, and h thus bringing into play regions that are respectively interior, common, and exterior to the two regions being merged. The reason the algorithm uses merge-event time stamping is to help maintain edge integrity when it becomes necessary to remove more than one shared boundary edge during the course of a merge. This becomes especially critical when newly created regions will have resulting genus greater than zero.

RegionMergeFromEdge(e)
  $p_1 \longleftarrow$ PixelContainingEdge(e)
  $p_2 \longleftarrow$ PixelAdjacentToEdge(e)
  $N_1 \longleftarrow$ ActiveRegionContainingPixel(p_1)
  $N_2 \longleftarrow$ ActiveRegionContainingPixel(p_2)
  $t \longleftarrow$ NextTimeStamp()
  e.timestamp $\longleftarrow t$
  $N \longleftarrow$ NextRegion()
    N.color $\longleftarrow$ AreaWeightedAvgColor(N_1,N_2)
    N.area $\longleftarrow N_1$.area + N_2.area
    N.height $\longleftarrow 1 + \max(N_1$.height, N_2.height)
    N.distinguishedpixel $\longleftarrow p_1$
    N.left $\longleftarrow N_1$
    N.right $\longleftarrow N_2$
    N.timestamp $\longleftarrow t$
  UpdatePixel(p_1,N_1,N_2,N,t)
  UnmarkPixelsInRegion(N)
  N_1.active $\longleftarrow$ false
  N_2.active $\longleftarrow$ false
  N.active $\longleftarrow$ true

```
UpdatePixel (p,N1,N2,N,t)
    if (!p.marked ∧ (p ∈ N1 ∨ p ∈ N2))
        p.marked ⟵ true
        p.region ⟵ N
        for each edge e′ of p in the priority queue
            if (BoundaryEdge(e′,N1,N2))
                RemoveFromPriorityQueue(e′)
                e′.timestamp ⟵ t
            else
                UpdatePriority(e′)
        for each pixel p′ adjacent to p
            UpdatePixel(p′,N1,N2,N,t)
```

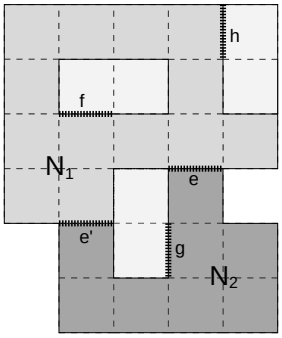


Figure 3: Merge of regions N_1 and N_2 triggered by boundary edge e.

6 Edge Merge Priority

As an aid in organizing paintings, a painter will often mentally de-focus or blur the subject matter of the image. We want to preserve the artist's broad color structure achieved using this technique. We do this by merging neighboring regions with the same perceived chromatic content. Merge priorities assigned to boundary edges are used to measure color differences between neighboring regions. These color differences are calculated on the basis of color component differences. Identical colors will have no differences in their color components, so edges bounding regions with identical color will have merge priority zero. Thus we use a *minheap* [10] for our priority queue of edges, whence the edge with the minimal merge priority will trigger the next region-merge event. The merge priority of an edge is defined to be the sum of the squares of the color component differences across the edge, computed in $\ell\alpha\beta$ space.

7 The Image Pyramid

We organize each digital image into a pyramid [5] by successively down sampling so that the *base*, at layer zero, has the image at full resolution, while the *apex* has the image at the lowest resolution we are willing to use for extracting image color content. Our down sampling method is nonstandard. From a block of four pixels we select the pixel which is closest to the average color of the block. This maintains color true to the original painting, preventing "muddy" colors that often result from averaging, and also helps to preserve the painterly style of many of our images. Figure 4 illustrates how we use the image pyramid. All the pixels in the apex form one "logical" active region. The lone active region of the apex is not obtained from our color segmentation algorithm, however, because we never transfer color at the apex layer, so we never need either its binary tree or its averaged $\ell\alpha\beta$ color. To descend through an image pyramid, we select an active region in the current layer (indicated by hatching in the figure), and we *mask* those pixels in the layer directly beneath (indicated by shading in the figure) which can be projected back up to pixels in the region selected. Now, turning our attention to the layer just masked, we color segment only the portion in that layer that has just been masked, and we repeat the process on down to the base. Pyramid descent is initialized by masking the entire apex layer. Since pyramid descent implements a divide-and-conquer strategy, it has the additional advantage of lowering the run-time complexity of our algorithm by reducing the size of our priority queues.

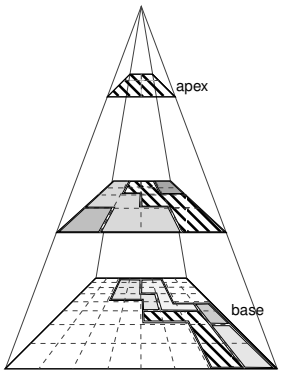


Figure 4: Synthesis within an image pyramid.

If we start in the layer beneath the apex of the image pyramid, and color segment until we have a reasonably small number of regions, then we can usually achieve good approximations to the color organization for the image. There are broad expansive areas of the image to work with which are broken up by areas containing highlights, shadows, and transitional colors. One problem that arises is that there can be too many small regions. Some of these small regions consist of single never-merged accent *pixels*, whose boundary edges are of such high merge priority that continued merging will preserve these "rogue" regions at

the expense of the structural integrity of the composition as a whole. The decision about how to handle rogue regions affects both the formation of image palettes and the actual color transfer that takes place at the level of pixels. To help prevent image corruption, we interrupt the region merging algorithm before the major structural elements are lost and work the rogue regions into the merged framework by *absorption* until the desired granularity is obtained. More precisely, once a merge-priority threshhold is reached, we override the algorithm for selecting edges that trigger merge events so that rather than selecting edges of minimum priority, it selects one of the edges associated to a region of minimal area i.e. the first pixel encountered during a scanline traversal that belongs to an active region of minimal area must have one of its *four* edges in the priority queue, and by examining those edges we can determine an edge to use that will cause this minimal area region to be merged with a region adjacent to the pixel that has *smallest* area. Figure 5 demonstrates how pixel p belonging to region N, which has area two, would cause region N to be merged with region X (but not X') using largest areas, or Y (but not Y') using smallest areas, and further shows how an edge that might be needed to trigger such a merge event can lie in an adjacent pixel.

Figure 6 offers evidence of the advantages to be gained from using absorption, but also points out the tradeoff we were faced with when confronting the decision about how to select companion regions to use for absorption. Starting with the image at the top left, the image at the top right shows, in blue, those pixels that were never merged when the number of regions was reduced from $32 \times 24 = 1068$ by successively removing edges of lowest merge priority until the merge-priority threshold was reached. Absorption was then used to further reduce the number of regions until no regions consisting of single pixels remained. Absorbing the smallest extant regions with their *largest* nearest neighbors gave the lower left image. Absorbing the smallest available regions with their *smallest* nearest neighbors gave the lower right image. Smallest nearest neighbors appear to give better image decompositions, while largest nearest neighbors appear to give better color distributions. Even though our goal is color transfer, we adopted the smallest-nearest-neighbor rule so that image recoloring would have fewer image artifacts. In other words, it turned out to be futile to make good color associations between palettes if the "features" the palette colors represented were garbled.

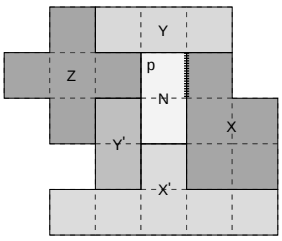


Figure 5: Absorption merging via a pixel.

In order for regions in layers that are lower in the pyramid to be able to reveal more detail when they are color segmented we *decrease* proportionally the threshhold we use to interrupt priority merging. Because absorption impacts color transfer at the pixel level, we do not absorb rogue regions in subsequent layers.

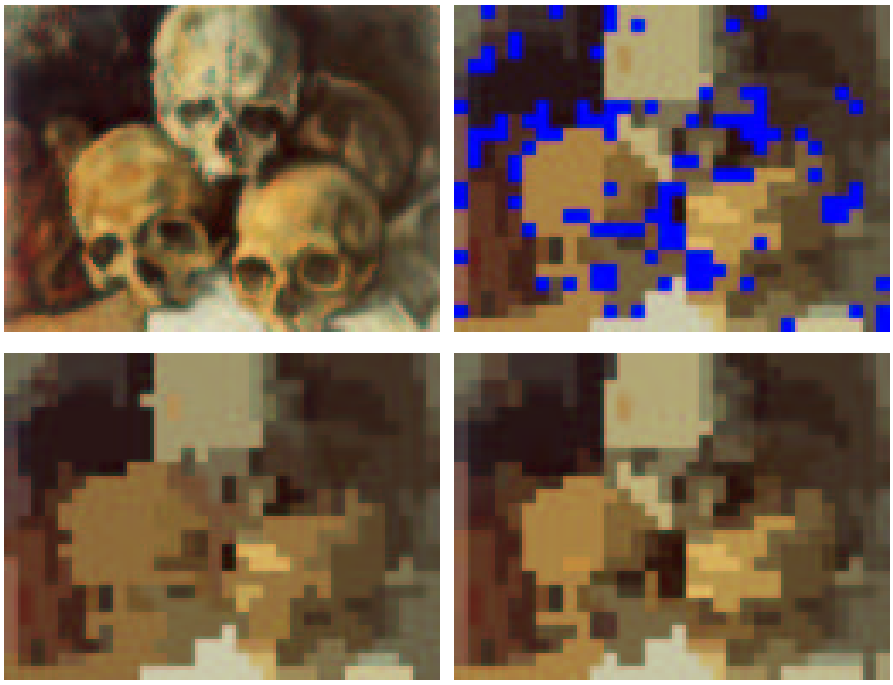


Figure 6: The effects of merging by absorption.

8 Palette Extraction

The set of average colors of active regions obtained from the color segmentation of an image, is the set of *segmented* colors. Because regions are 4-connected, it is to be expected that there will be a number of non-adjacent regions that will have perceptually indistinguishable colors. Our first step is to compensate for this color duplication within the set of segmented colors. We partition the set of segmented colors into subsets of colors and select one color representative from each subset. These representatives form the set of *identified* colors. To clarify this partitioning step, we consider what partitioning might look like for a hypothetical example. Figure 7 shows regions X, X', Y, Y', and Z obtained through region merging. The five colors associated to them are the segmented colors. Labeling indicates that the color for X has been identified with the color for X', the color for Y has been identified with the color for Y', and

that colors from regions X and Y have been selected as the nominees for the identified set of colors, i.e. the colors from regions X, Y, and Z are the canonical representatives. The color chosen as the canonical representative for a subset is the one that is associated to the region having the largest area. Working in RGB space, we define two colors to be perceptually identical provided their Euclidean distance does not exceed a specified threshold. We use RGB space for this purpose because $\ell\alpha\beta$ space is logarithmic and makes too fine a distinction between dark colors and too coarse a distinction between light colors. For our test images, following segmentation "Starry Night" has 159 colors and "Skulls has 119 colors. Following identification, these numbers are reduced to 22 and 13 respectively.

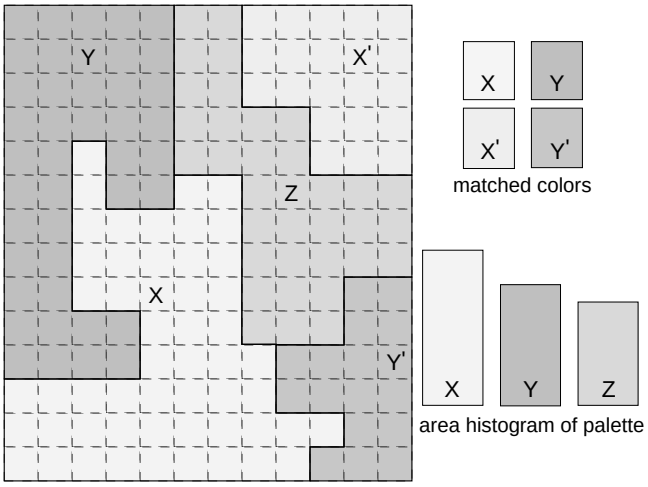


Figure 7: Identifying duplicate colors.

Since we are attempting to model the painter's palette, we want a bare minimum number of colors, thus we must compensate for the fact that within the set of identified colors there may be a number of shades of a given color. We need to partition the set of identified colors into subsets such that all colors within a subset are similar up to shading. Unfortunately, this is a very difficult task. No single color space accurately captures the notion of shades of a color. We again adopted a bottom-up approach. Sorting the identified colors by their saturation components in HSV space, we let each (unused) color serve as the representative for a cluster of colors that are similar with respect to shading as determined in some color space. Iterating this clustering algorithm first in HSV space, where similarity is defined using a "tapered wedge" neighborhood, then in $\ell\alpha\beta$ space, where similarity is defined using a "slab" neighborhood, allows us to "converge" to a set of color representatives for clusters/subsets of colors within the identified set of colors. Clustering in HSV space can be used to good advantage for lighter colors, while clustering in $\ell\alpha\beta$ can be used to good advantage for darker colors. The highest-saturated color representatives

responsible for the final cluster formation make up the color *palette* of the image we are working with. For our test images, shade clustering reduces the number of colors for "Starry Night" from 22 to 10 and the number of colors in "Skulls" from 13 to 8.

For most images it does not make sense to try and recolor true black or true white. True black and true white are mixing colors that the artist uses to make shades from palette colors. We do not want to consider true black and true white as belonging to our palette when we are trying to make decisions about color transfer. We will have more to say about this later, but for now we note that this has no effect on the "Skulls' palette, but does remove the darkest color from consideration in the "Starry Night" palette.

As lower layers in the pyramid are encountered, we continue to remove duplicates from sets of segmented colors, but because the "shade extraction" algorithm is used to lock-in the initial palette color associations, and the assumption is that we are trying to refine these "master" associations, we no longer extract shades. Instead, we *trim* the number of colors in the set of identified colors of the *source* image in an effort to prevent spurious colors from dominating color transfer from source to destination. Spurious colors often occur at feature boundaries. They also occasionally arise as an up sampling artifact. After each subsequent color segmenting of a portion of the source image, following the identification phase, we accumulate colors to use for the color transfer from that portion by sorting colors on the basis of the areas they are responsible for and accumulating colors until we account for 90% of the segmented area.

To continue with the theme of refinement, in the layer immediately below the layer that was segmented to extract the image's palette, whenever a projected region is to be segmented, it is broadened to include the projected regions from all the regions in the class of its parent. This is to ensure that all the shades from the source color and all the shades from the destination color get considered simultaneously when trying to make subsequent refinements of the initial source palette color to destination palette color pairing.

To summarize, an *image* palette color is a region color that is responsible for a set of shades, and a shade is a region color that is responsible for a subset of regions with perceptually indistinguishable colors. Thus image palette colors are responsible for partitioning the set of segmented colors into classes. If R is a region whose average color is a palette color, i.e. the canonical representative for a class, we denote its class of segmented colors by $[R]$. We denote the palette color vector by $[R]$ also.

Figure 8 shows the palettes and the segmented images colored with respect to the palette color of the class they belong to. These palettes were obtained by using a four layer pyramid with the apex having maximum pixel dimensions 16×16. Color segmenting to extract the palette was therefore done in the 32×32 pixel layer of the pyramid. Colors from the palette are sorted in descending order according to the total area they are responsible for. A histogram of these areas lies above each palette.

We are finally able to give (unoptimized) pseudocode for our color transfer algorithm. Details of several of the steps of the algorithm follow the pseudocode.

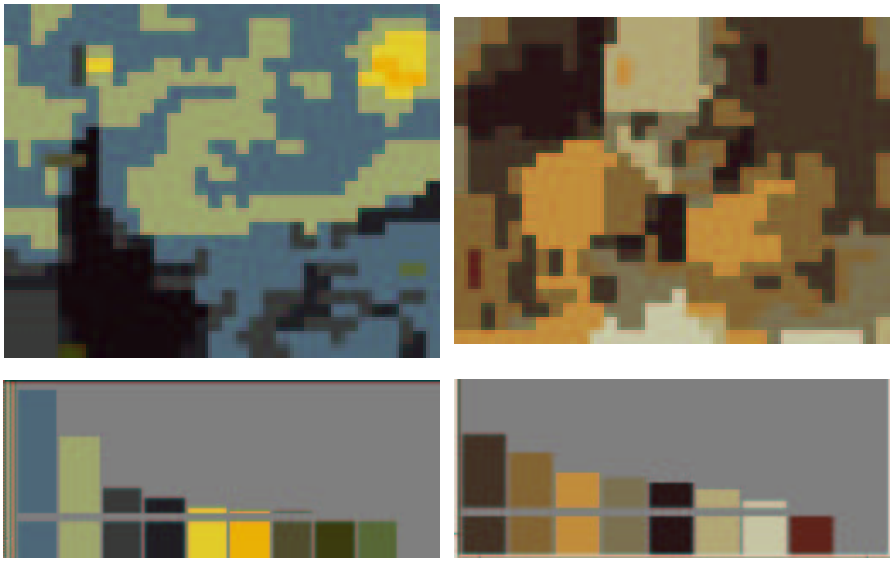


Figure 8: Palettes for the low resolution test images.

ColorTransfer(*srcfile*,*dstfile*)
    Pyramid Variables: *src*, *dst*
    src.Load(*srcfile*)
    dst.Load(*dstfile*)
    src.FormPyramidLayers()
    dst.FormPyramidLayers()
    src.MakeApexIntoOneRegion()
    dst.MakeApexIntoOneRegion()
    FormRegionPairs(*src*, *dst*, APEX)
    ColorTransferByDescent(*src*, *dst*, APEX−1)
    dst.OutputLayer(BASE)

ColorTransferByDescent(*src*, *dst*, i)
    dst.RefineProjectedBoundariesInLayer(i)
    for each region pair (S,D) of layer $i+1$
        src.MaskProjectedPixelsInLayer(i)
        src.SegmentMaskedPixelsInLayer(i),
                yielding regions $s_1, s_2, \ldots, s_u$
        src.ExtractPaletteInLayer(i)
        src.NormalizePaletteInLayer(i)
        dst.MaskProjectedPixelsInLayer(i)
        dst.SegmentMaskedPixelsInLayer(i),
                yielding regions $d_1, d_2, \ldots, d_v$
        dst.ExtractPaletteInLayer(i)

```
    dst.NormalizePaletteInLayer(i)
    FormRegionPairs(src,dst,i)
    if (i == BASE)
        for each region pair (s_j, d_k)
            ColorTransferByRegion(s_j, d_k)
    src.UnMaskPixelsInLayer(i)
    dst.UnMaskPixelsInLayer(i)
if (i > BASE)
    ColorTransferByDescent(src, dst, i − 1)
```

9 Normalized Palettes

Although making "intelligent" color associations between source palette colors and destination palette colors is not yet our primary focus, initial tests suggested that it was helpful to encourage the color responsible for the largest area in the source to be paired with the color responsible for representing the largest area from the destination. This observation forms the basis of a naive color transfer heuristic. The idea is to use this association to establish an anchor pairing and then make further pairings based on how the remaining colors deviate from their anchor colors. We begin by sorting the colors in the palette from largest to smallest using the area obtained by summing area over the subset of regions in the set of segmented colors that a palette color is responsible for. We then translate all of the colors in a palette so that the color in the palette with the largest summed area is the origin of the palette's color space. This serves to measure the remaining colors *relative* to the color with the largest summed area. Since the translation preserves relative values of ℓ, the value rankings of the colors do not change. In the spirit of Reinhardt et al [8], we turn to elementary statistics to try and establish a meaningful way to make comparisons between sets of relative colors. Within each palette, the mean and standard deviation of each component is calculated, and all components are replaced by their z-scores, i.e. the number of standard deviations from the mean. We then translate a second time so that the anchor color is again at the origin. (The subtle point is that the anchor color must be normalized so that "nearby" colors can be identified, and the second translation is needed so that inter-palette comparisons make sense.)

10 Palette Color Associations

Our algorithm for assigning source palette colors to destination palette colors forces the color representing the largest summed area of the source to be associated to the color representing the largest summed area of the destination. To *encourage* the color representing the second largest summed area of the source to be assigned to the color representing the second largest summed area of the

destination, we rotate each normalized palette around the ℓ-axis so that the β component of the normalized vector corresponding to each color representing the second largest summed area is zero. Palette color associations and their induced source to destination region pairings now arise as follows. Associate to each color $[D]$ in the destination palette the color $[S]$ in the source palette that is closest to it as measured by calculating Euclidean distance between normalized, rotated palette color vectors in $\ell\alpha\beta$ space. From the palette association $([S], [D])$, form source to destination region pairs (S, D') where D' ranges over all regions in the class $[D]$ i.e. over all regions in the segmented set of colors whose colors are "identical" to colors that are "similar" to the color of D. Figure 9 shows the color pairing this strategy produced for our test images following color segmentation of the 32×32 resolution layer. Source color is on the top and destination color is on the bottom. Since the palette color associations in Figure 9 respect the area sort that was imposed upon the destination palette, we are able to observe that some source colors (e.g. gray) are used only sparingly and some (e.g. gold) not at all.

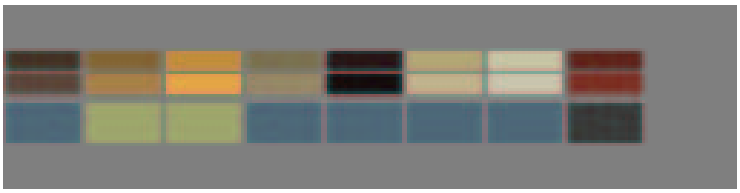


Figure 9: Palette color associations for the test images.

11 Color Transfer at the Pixel Level

The final step of our color transfer method requires us to use the binary trees from a paired source region and destination region to transfer color from source pixels to destination pixels. If the trees were isomorphic, then we could traverse them in parallel and achieve a one-to-one pixel matching. However, the two trees will almost certainly be topologically different. Real time visualizations of our region-merge algorithm convinced us that it was robust in the sense that merge sequences tended to yield good approximations to full trees. Therefore there was some hope that we could transfer color between paired regions by simultaneously traversing their binary trees in such a way as to approximate a parallel traverse between two identical full trees.

Our simultaneous traverse has the property that if it were presented with identical trees it would pair identical pixels. It is motivated by the idea that color transfer moves color from the source *area* to the destination *area*, and thus its key is the area attribute of the tree. Recall that the leaves of the trees are the pixels.

ColorTransferByRegion(s, d)

```
if (s.Leaf() ∧ d.Leaf())
    ColorTransferByPixel(s,d)
else if (s.Leaf())
    ColorTransferByRegion(s, d.left)
    ColorTransferByRegion(s, d.right)
else if (d.Leaf())
    ColorTransferByRegion(bestLeaf(s), d)
else
    RegionMatch(s.left, s.right, d.left, d.right)
```

If a source pixel is transferred to a destination region, then recursion distributes this pixel's color throughout the entire region, but if a source region is transferred to a destination pixel then we must choose which leaf pixel from the source to use. We tried using a random leaf, a leaf nearest the root, a leaf farthest from the root, and a leaf whose color was closest to the region's average color. The images shown here use the best approximation to the average color.

When neither the source region nor destination region is a leaf, we cannot discard destination subtrees or else their regions will not be colored! To make the best use of the source we must try to preserve as much of it as possible. The descent algorithm for region pairing that we settled upon is hierarchical — the order in which cases are considered matters.

```
RegionMatch(s_L, s_R, d_L, d_R)
    if (s_L.area ≤ d_L.area ∧ s_R.area ≤ d_R.area)
        CoplorTransferByRegion(s_L, d_L)
        ColorTransferByRegion(s_R, d_R)
    else if (s_L.area ≤ d_R.area ∧ s_R.area ≤ d_L.area)
        ColorTransferByRegion(s_L, d_R)
        ColorTransferByRegion(s_R, d_L)
    else if (s_L.area > d_L.area ∧ s_L.area > d_R.area)
        if (s_L.left.area > s_L.right.area)
            s' ⟵ s_L.left
        else
            s' ⟵ s_L.right
        ColorTransferByRegion(s', d_L)
        ColorTransferByRegion(s', d_R)
    else if (s_R.area > d_L.area ∧ s_R.area > d_R.area)
        if (s_R.left.area > s_R.right.area)
            s' ⟵ s_R.left
        else
            s' ⟵ s_R.right
        ColorTransferByRegion(s', d_L)
        ColorTransferByRegion(s', d_R)
    else if (s_L.area > s_R.area)
        ColorTransferByRegion(s_L, d_L)
        ColorTransferByRegion(s_L, d_R)
```

else
    ColorTransferByRegion(s_R, d_L)
    ColorTransferByRegion(s_R, d_R)

Since color is being transferred *from* the source, the algorithm begins by considering how to handle insufficient source material. Region merging does not orient subtrees — "left" and "right" have no meaning — so if there is a pairing which will fit the source subtrees inside the destination subtrees it uses that pairing. Next, if it can fit both destination subtrees inside the largest source subtree then it pairs both destination trees with the largest subtree of that largest subtree. This helps prevent pairing one large source subtree with *all* of the destination pixels thereby causing one pixel from that source subtree to be the only source pixel ever transferred, a situation which could give inferior results if a source region was involved in many pairings. Only as a last resort, when it is certain that the largest source subtree will have area smaller than the largest destination subtree does the algorithm pair both destination subtrees with the largest source subtree. The images in the top row of Figure 10 show the advantage we gain from subtree analysis. The image at the upper left shows the transfer of the average $\ell\alpha\beta$ color from the paired source *region*, while the image at the upper right shows the transfer of the $\ell\alpha\beta$ color from the paired source *pixel* for our test example. The pixel transfer approach can preserve more of the visually rich color texture of the source region.

12 Color Correction

Since we want to transfer chroma not value, we transfer only the α and β channels from source to destination. Figure 10 shows how important this is. Without value preservation (top row), the structure of the image is lost, whereas with preservation (bottom row) it is retained. However, two problems arise: (1) color corruption due to large discrepancies in the ℓ channels i.e. excessive lightening or darkening of a color and (2) out-of-gamut colors due to significant difference in hues. When the ℓ component of a color is low, its chroma should not matter. Since many digital images have high saturations in their dark colors, if we are not careful, transferring dark colors to destinations with moderately higher value components often yields unnaturally saturated colors. As a partial solution to this problem, when we transfer the α and β channels α_{src} and β_{src} from the source image to the channels α_{dst} and β_{dst} of the destination image, we desaturate them according to a factor based on the the value measures L_{src} and L_{dst} by setting

$$\begin{aligned} \alpha_{\text{dst}} &= \alpha_{\text{src}} F(L_{\text{src}}, \min(L_{\text{dst}}, 0.4)) \,, \\ \beta_{\text{dst}} &= \beta_{\text{src}} F(L_{\text{src}}, \min(L_{\text{dst}}, 0.4)) \,, \end{aligned}$$

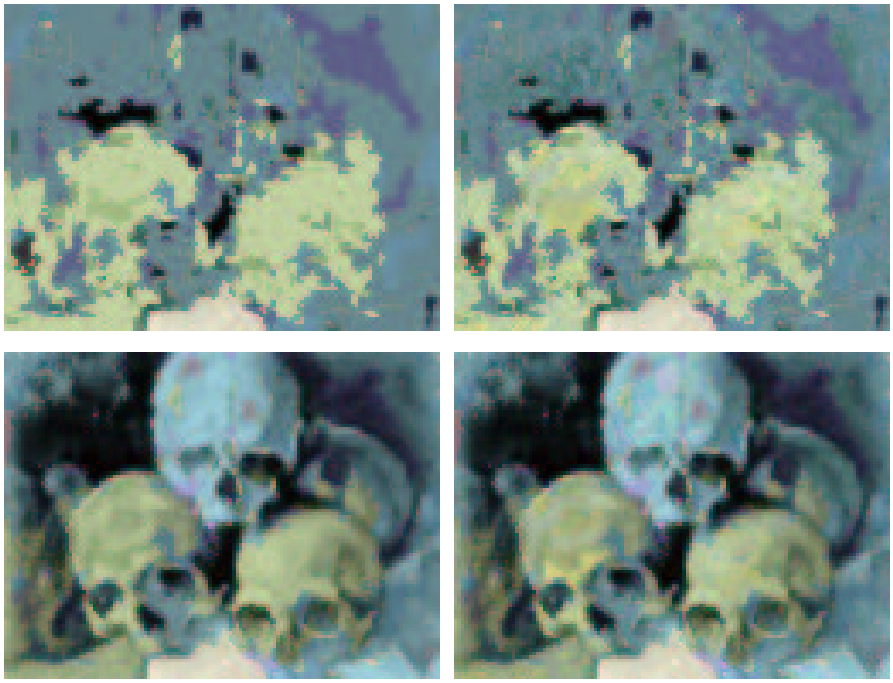


Figure 10: Comparison of color transfer by regions (left column) versus pixels (right column) and by $\ell\alpha\beta$ (top row) versus $\alpha\beta$ (bottom row).

using the saturation curve

$$F(t,T) = \begin{cases} 1 & \text{if } t \geq T\ , \\ -2(t/T)^3 + 3(t/T)^2 & \text{if } 0 \leq t < T\ . \end{cases}$$

Note that L_{src} and L_{dst} lie in the intermediate LMS space (see Appendix A).

Even though desaturation will not be invoked when highly saturated colors are transferred to destination colors of lower value, out-of-gamut colors may still result because the "strength" of the chroma being transferred may artificially inflate one or more of the RGB components when we transform from $\ell\alpha\beta$ space. Regardless of how out-of-gamut colors arise, after we transform all of the destination image pixels back to RGB space we must often make a global correction of the RGB image so that all colors are in gamut. This is accomplished by identifying the 96% trimmed maximum taken over the R, G, and B components of all pixels and then scaling the entire image by this value, if necessary. We found that scaling by the global maximum darkens the image too much, so the use of the 96% trimmed maximum is a compromise which seems to work well in practice. For those pixels whose maximum taken over the R, G, and B components lie above the 96% trimmed maximum that are out-of-gamut we do a local correction as follows: the maximal channel value is clamped and the values of the remaining channels are raised in an effort to try to preserve

the pixel's luminance. The luminance adjustment is performed holding the Y component (of YIQ space) constant. For the image recoloring found in Figure 1, the trimmed maximum color correction did not come into play and no pixels were out of gamut.

Desaturating colors as they are transferred from source pixel to destination pixel and then invoking global color corrections can produce images with excessive amounts of gray. This problem is most acute when (pure) whites are involved. Trying to add chroma to whites in $\ell\alpha\beta$ space causes colors to soar out of gamut in RGB space. Further complicating matters is the fact that when trying to transfer saturated colors to dark areas, global correction can create "hot spots" caused by clamping only a few pixels within a region. Issues such as these revealed to us how difficult the *general* problem of color transfer really is.

In making images for this paper we added one more feature to try and head off out-of-gamut problems. It was suggested to us during discussions of our preliminary results with an artist. When artists want to incorporate a disparate color into their palette at say a later stage of image composition, often they must re-adjust the value structure of the entire painting. Therefore prior to color transfer we conditioned our destination images by implementing an algorithm to modify ℓ values of the pixels in the destination image so that the resulting histogram of their ℓ values would match a similar histogram of ℓ values of the source image. Figure 9 shows the colors in the destination palette on top split in half. The top half is the color as it appears in the image, and the bottom half is the color as it appears after conditioning. This explains why the recoloring lightened the original image. The fact that a destination image might need conditioning in order to be "value compatible" with a source image helps explain why it is virtually impossible for certain recolorings to be done. For example, Figure 11 shows what happened when we tried to condition a Fragonard using a Kandinsky and vice versa. The Fragonard became useless after this step, and color transfer was halted. The Kandinsky held up somewhat better. Our recoloring of Kandinsky by Fragonard is discussed in the next section.

13 Lossyness and High Resolution Recolorings

Image recolorings based on *palette* color associations necessarily lead to 'lossy" recolorings because (1) we have no way to guarantee that all of the source palette colors will be transferred and (2) only one source region from each color pairing participates in the recoloring while all destination regions from each color pairing participate. As a test of our algorithm's integrity we tried recoloring an image with itself. Figure 12 shows such a recoloring of "Starry Night." The two images have only subtle differences. We turned off source palette "trimming" when making this image in order to gauge the degree of lossyness that results.

When producing high resolution image recolorings (maximum dimension of 512 pixels), to prevent "blockiness" from propagating during pyramid descent,

we turn off absorption after the initial layer is color segmented, and as each new layer within the destination pyramid is encountered, we examine its pixels one by one in order to resolve any questionable parent-region assignments for pixels that are children of *boundary* pixels in the parent layer. A high resolution recoloring for our test images is shown in Figure 13. A high resolution recoloring of Kandinsky by Fragonard is shown in Figure 14. Unfortunately, our naive transfer heuristic prevented any of the yellow in the Fragonard from being transferred to the Kandinsky.

14 Bi-directional Imaging Recoloring

While developing our approach, we leaned heavily on the "Starry Night" and "Skulls" images as test examples. The top row of Figure 15 shows test images by Franz Marc ("Yellow Cow") and Emil Nolde ("Mask Still Life III") that we used to evaluate the generality and the limitations of our approach. They were selected for their range of palette colors. The middle row of Figure 15 shows low resolution bi-directional recolorings and the bottom row shows the top-level palette color associations for each of these recolorings. The high resolution recolorings are given in Figure 17 and Figure 16. Our naive color matching heuristic has trouble using all the palette colors when making the initial palette color associations. Notice, however, that the recoloring of "Yellow Cow" does find a red from the "Masks" image at a lower layer of the source pyramid. Curiously enough, our effort to match second largest summed area colors of the palettes failed for both these recolorings. This is especially unfortunate in the "Masks" recoloring since the two yellow masks in the image draw color from a saturated, neutral sky color extracted from "Yellow Cow" and the saturation curve fades this color even further upon transfer. On balance, both of these recolorings are quite successful. In large measure, this is due to the fact that true black and and true white were excluded from their palettes. This causes a default color association of white with white and black with black. If these colors were not excluded, then palette normalization would force them to be associated to chromatic colors which would lead to global color correction that would make both recolorings quite gray. There is a cost. The color representative used for white in the "Skulls" image is not perceived as a true white. Subsequent refinement of the white regions in "Skulls" causes only one of the two white patches at the bottom of the image to be perceived as true white. Fig 1 shows the somewhat jarring effect that occurs when one of these patches is recolored and the other is not. Be that as it may, we wish to emphasize once more that our recolorings are all done by the same program, using identical parameter settings, and without any user intervention.

15 Conclusions

The goals of this paper were limited — we attempted to establish a framework to investigate the problem of transferring the chromatic content from one image to another in such a way that its artistic intent was preserved, and to consider some preliminary results. The images we presented were not meant to be our *coup-de-grace*, but simply demonstrations of the potential of this framework. It is our hope that they will provide a benchmark by which to measure the success or failure of future work. In that sense, we feel that we have succeeded in our original quest. We were able to make a number of compelling recolored images. We did not, however, have the opportunity to focus as much attention as we would have liked on the problem of deciding which colors within a palette are the most important ones to use for associations, or to investigate "rule sets" for making color associations between palettes.

Automated color transfer is a very challenging problem. One nuance is level of detail. We attempted to refine color associations while descending through an image pyramid. This caused new colors to come into play which were not easily reconciled in accordance with the top-level "master" color associations. Also, the divide and conquer strategy sometimes led to a communications breakdown about the master color associations. A second nuance is sensitivity to initial conditions. Because of down sampling artifacts, the number of layers in a pyramid could confound the computations that yielded average colors and region areas thereby altering the formation of palette colors — classes of regions are *not* equivalence classes because the color similarity relation is not transitive, therefore region class formation depends heavily on the color of the "seed" representative that is chosen — and palette color associations. A third nuance is the difficulty in *forcing* source color to flow into destination color without causing out-of-gamut difficulties. For us this meant that the consideration of sophisticated rule sets for classifying palettes, making source to destination palette color associations, or identifying the best shades (i.e. finding the best region representatives) was premature.

There are other less technical nuances to consider. For example, we were never wholly successful at making a bi-directional recoloring of "Starry Night" with "Skulls" primarily because the narrow range of earth tones in "Skulls" meant that nearly all the colors in its palette needed to be transferred before the "Starry Night" recoloring made sense. Given the chromatic range of "Starry Night" this is not an easy task. Not only will out-of-gamut problems arise but there are issues about the roles the colors white and black should play in the recoloring that, if not handled carefully, could lead to only partial recolorings. In general, we found that the more colors that *needed* to be transferred from the source to the destination, the more likely it was that errors in artistic intent would occur. Experience taught us that even minor chromatic inconsistencies in recolorings invoked strong negative reactions from viewers. This explains why we resorted to trimming source palettes and why we chose well-known paintings as test sets of images for recolorings.

References

[1] H. Bieri and A. Kohler. Computing the area, the circumference, and the genus of a binary digital image. *Graphics Gems II*, J. Arvo (editor), Academic Press, 107–111, 1991.

[2] Comaniciu, D., and P. Meer, Robust analysis of feature spaces: color image segmentation, *Proceedings of CVPR '97*, 1997, 750-755.

[3] Y. Deng, B. Manjunath, and H. Shin, Color image segmentation, *Proceedings of CVPR '99*, 1999.

[4] B. Gooch and A. Gooch. *Non-Photorealistic Rendering*. A K Peters, 2001.

[5] D. Heeger and J. Bergen. Pyramid-based texture analysis/synthesis. *ACM Computer Graphics (Proc. of SIGGRAPH '95)*, 229-238, 1995.

[6] A. Hertzmann, C. Jacobs, N. Oliver, B. Curless, and D. Salesin. Image analogies. *ACM Computer Graphics (Proc. of SIGGRAPH '01)*, 327–340. 2001.

[7] C. Jacobs, A. Finkelstein, and D. Salesin. Fast multiresolution image querying. *ACM Computer Graphics (Proc. of SIGGRAPH '95)*, 277–286, 1995.

[8] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley. Color transfer between images. *IEEE Computer Graphics and Applications*, **21**(5):34–41, 2001.

[9] D.L. Ruderman, T.W. Cronin, and C.C. Chiao. Statistics of cone responses to natural images: implications for visual coding. *J. Optical Soc. of America*, **15**(8):2036–2045, 2001.

[10] R. Sedgewick. *Algorithms, Second Edition*. Addison-Wesley, 1989.

Appendix A

We give the essentials of the transformation of Reinhard et al [8] to convert colors from RGB space to logarithmic $\ell\alpha\beta$ space. The only modification we make is to add a clamping operation when logarithms are taken. A color vector (R, G, B) is first transformed to (L, M, S) space

$$\begin{pmatrix} L \\ M \\ S \end{pmatrix} = \begin{pmatrix} 0.3811 & 0.5783 & 0.0402 \\ 0.1967 & 0.7244 & 0.0782 \\ 0.0241 & 0.1288 & 0.8444 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}.$$

Second, logarithms compensate for perceptual sensitivity

$$\begin{pmatrix} L' \\ M' \\ S' \end{pmatrix} = \begin{pmatrix} \max(-10.0, \log L) \\ \max(-10.0, \log M) \\ \max(-10.0, \log S) \end{pmatrix}.$$

Third, axes are decorrelated by rotation

$$\begin{pmatrix} \ell \\ \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \\ 1/\sqrt{6} & 1/\sqrt{6} & -2/\sqrt{6} \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 \end{pmatrix} \begin{pmatrix} L' \\ M' \\ S' \end{pmatrix}.$$

The process is easily inverted.

Fragonard Kandinsky

Figure 11: Top row: Low resolution test Fragonard and Kandinsky. Lower left: Fragonard value-conditioned from Kandinsky. Lower right: Kandinsky value-conditioned from Fragonard. (Fragonard and Kandinsky from http://www.artframed.com)

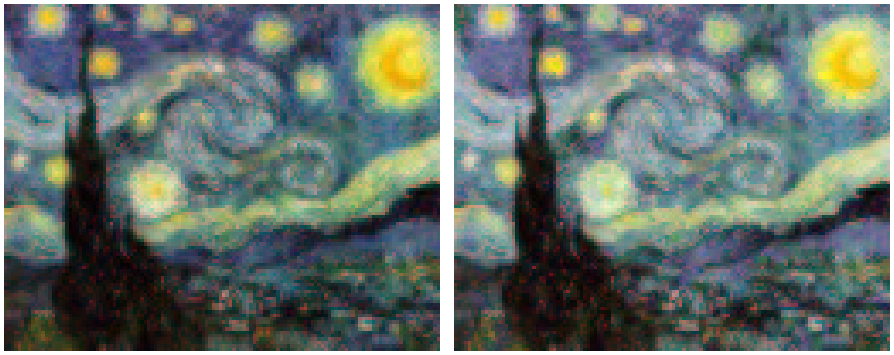


Figure 12: A recoloring of "Starry Night" with itself. The original is on the left, and the recoloring is on the right.

Figure 13: High resolution recoloring of "Skulls" by "Starry Night."

Figure 14: High resolution recoloring of Kandisky by Fragonard.

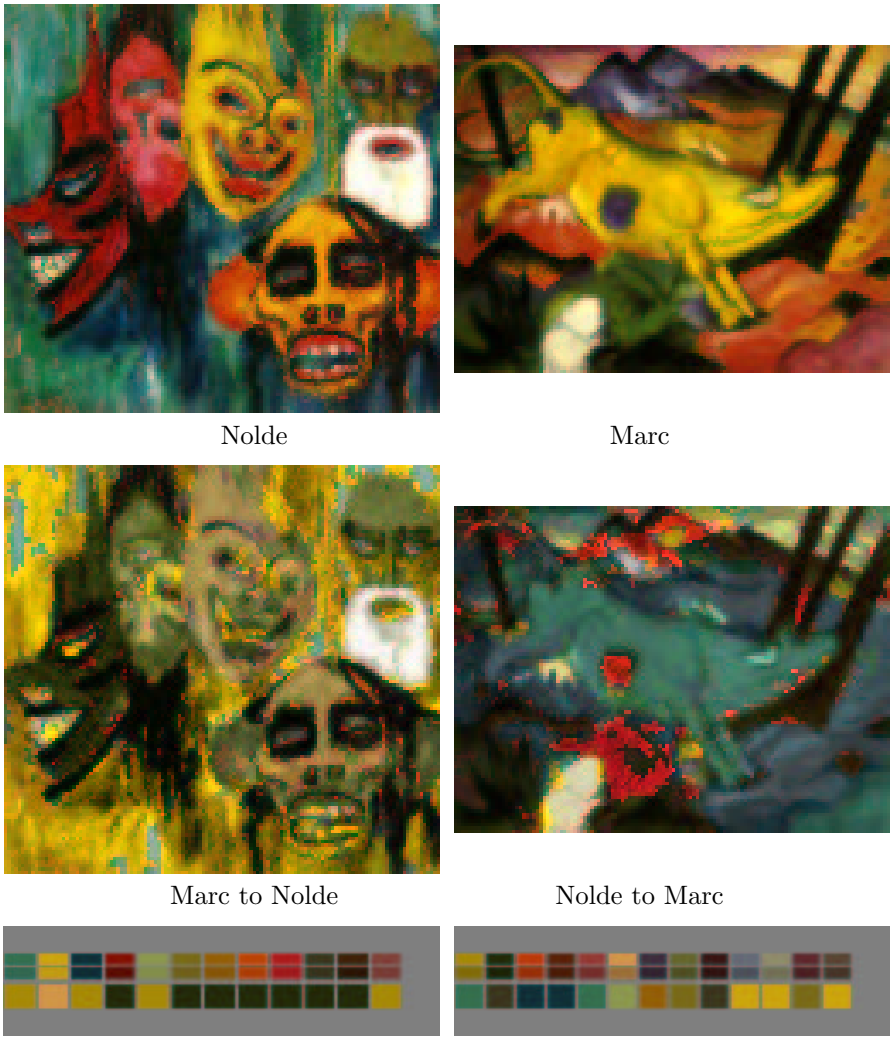


Figure 15: Top row: Low resolution Nolde and Marc images. Middle row: High resolution bi-directional recolorings. Bottom row: Palette color associations. (Nolde and Marc from http://www.artframed.com)

Figure 16: High resolution recoloring of Marc by Nolde.

Figure 17: High resolution recoloring of Nolde by Marc.