

Color-Space CAD: Direct Gamut Editing in 3D

Neophytos Neophytou and Klaus Mueller ■ *Stony Brook University*

As digital photography grows at an increasing pace, so do the number of image- and color-manipulation tools and their capabilities. Currently, these tools are almost always confined to traditional 2D interfaces. Color, however, is a 3D entity, most naturally manipulated in a suitable 3D interface. Yet two hurdles have prevented these types of interfaces from becoming mainstream. First, there are concerns regarding

usability and the learning curve such a new paradigm would impose on the current user community. Second, pre-GPU hardware is incapable of facilitating an interactive, responsive, and compelling interface that would support these direct manipulations in 3D color space. We present an interface that addresses these two obstacles.

We extend the current set of image-processing tools by introducing a technique for transforming the range of colors in any region of an image using geometric operations in a perceptually uniform color space. The provided environment lets

users directly operate on an image region's color gamut using a 3D CAD-like interface. The color-visualization environment, along with real-time feedback on the effects of geometric manipulations on the image, let the artist better understand and explore the color-space relationships. To alleviate some of the 3D manipulations' complexity, we provide the complete editing environment used in CAD applications, which combines a set of simultaneous 2D orthographic projections and

a 3D perspective view. This provides additional contextual information and much better control to the artist.

Color-Space CAD has inherently different goals from other classes of colorization methods (see the "Related Work on Colorization" sidebar, page 90). Our primary goal is to generalize the traditional 1D and 2D color-manipulation mechanisms resident in many photo-processing software packages into 3D interaction techniques. We use linear mapping operations, which are similar to traditional methods, and are far less computationally expensive than the statistical or optimization methods of the more involved colorization methods mentioned previously. From this simplicity, we gain interactive processing speeds, which are a must-have in an artistic and creative setting where users tend to experiment extensively until they reach a satisfying result.

Direct manipulation in the color space

Figure 1 gives an overview of our color-space manipulation method. Initially, the user imports a color photograph into the host system (Adobe Photoshop, for example). The user selects the area to be edited and calls the Color-Space CAD plug-in to modify the selected region's color gamut. Alternatively, the user can import a gamut from other reference images to serve as the target gamut. The system then transforms the source colors of the selected image regions into the reference gamut's target colors using a 3D transformation in the color space. This transformation can take place in CIELAB or HSV color spaces, depending on the user's preference.

Using CIELAB

We chose the CIELAB color space (whose coordinates are L^* , a^* , b^*) as the most uniform and intuitive for our 3D spatial color visualization.

Color-Space CAD is an interactive image-processing framework that lets users manipulate colors directly in 3D perceptual color space. Unlike traditional 2D color-manipulation tools, which often require multiple iterations, Color-Space CAD allows direct 3D navigation of the solution space. The framework uses graphics hardware to accelerate the computation-intensive mapping operations.

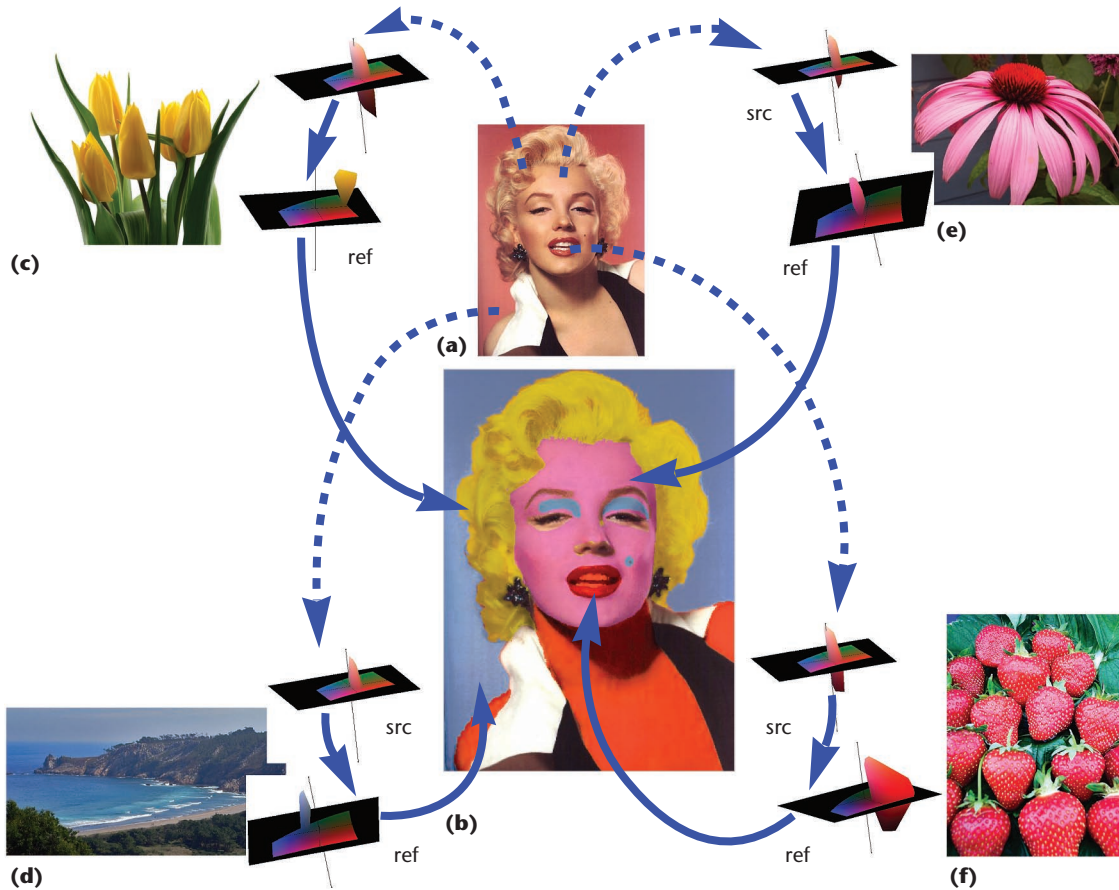


Figure 1. Creating a virtual silkscreen using Color-Space CAD. (The term “virtual silk-screen” refers to Andy Warhol’s famous prints using this technique.) The user imports (a) the original image into the system then creates (b) the target regions in the original image, using advanced selection tools. After importing the colors from the reference images (c–f), the system converts, for every selected region, the colors using a geometric transform from the source color objects (src) to the reference color objects (ref). (FreeDigitalPhotos.net provided the reference images.)

Psychophysical experiments have shown that even though CIELAB isn’t perfectly isotropic, the spatial relationships between colors are more natural to human observers. Thus, this color space is more satisfactory for image-understanding applications.¹

With perceptual uniformity as the main motivation, we defined all the interactive geometric operations within CIELAB to offer a better correspondence to Euclidian 3-space. Further, the manipulations on the color objects in this space are quite straightforward to the user. Any movement along the z-axis (mapped to L^*) affects the luminance (like using a television’s brightness control), and any movement along the x- (mapped to a^*) and y-axes (mapped to b^*) corresponds to changes in the image region’s chroma. Because the images are originally in RGB, a nonabsolute color space, we require a proper International Color Consortium (ICC) profile for the conversions to and from the CIELAB color space. In the absence of such a profile, we assume the parameters of a generic CRT monitor. Because the artist’s creative process

takes place using a specific monitor when loading the first image into Color-Space CAD, this monitor becomes the device profile attached to the resulting images and is used to standardize the image for subsequent edits, possibly by other users on different monitors.

Given these circumstances, we don’t claim that CIELAB provides better accuracy. We use this color space merely because it’s a convenient tool for geometric user interaction with 3D color objects.

Using HSV

Because the CIELAB color space provides a perceptually more uniform alternative, it represents a natural choice for performing 3D manipulation and transformations. However, the greater familiarity and comfort that most graphic artists and professionals have with HSV poses certain limitations to using CIELAB for production color-editing tasks.

We therefore extended our application to also use the HSV color space as a working environment for performing 3D color manipulation and

Related Work on Colorization

Work on the colorization of gray-level images^{1,2} and the recolorization of color images^{2,3} is slightly relevant to our work. Both of these approaches are semiautomatic offline processes that have enjoyed much recent interest. They use statistical analysis to impose one image's characteristics onto another. First, they convert the image into a perception-based color space ($\alpha\beta$) and then apply the color-correction process in a way that preserves the desired statistical characteristics along the three axes of the color space. Welsh and colleagues use swatches to control the gray-scale-to-color conversion.² This lets users better resolve ambiguities when the luminance statistics are similar in the source and reference images. Another approach lets users define new colors by scribbling a strip of uniform paint directly on the image to be processed.¹ An offline process then produces the colorized image based on these user parameters and hints (the scribbles).

Color science researchers have demonstrated color transfer using a direct representation of the image's color gamuts in ways quite different from ours. Here, the gamut-mapping algorithms deal with the problem of transferring an image across different media such as CRT monitors, ink-jet color printers, and color laser printers, with minimal loss of contrast and faithful color representation. Morovic and Luo provide an overview and evaluation

of such gamut-mapping functions, with some emphasis on contrast-preserving scaling functions.⁴ A related approach introduces the use of Alpha shapes—a generalization of convex hulls—to provide a geometric analytical description of the gamut's surface.⁵ This facilitates the comparison of gamuts and the computation of simple figure-of-merit quantities related to the device's quality (such as a gamut's volume). It also aids in out-of-gamut mappings using geometric techniques.

References

1. A. Levin, D. Lischinski, and Y. Weiss, "Colorization Using Optimization," *ACM Trans. Graphics (Proc. Siggraph)*, vol. 23, no. 3, 2004, pp. 689–694.
2. T. Welsh, M. Ashikhmin, and K. Mueller, "Transferring Color to Greyscale Images," *ACM Trans. Graphics (Proc. Siggraph)*, vol. 21, no. 3, 2002, pp. 277–280.
3. E. Reinhard et al., "Color Transfer between Images," *IEEE Computer Graphics and Applications*, vol. 21, no. 5, 2001, pp. 34–41.
4. J. Morovic and M.R. Luo, "The Fundamentals of Gamut Mapping: A Survey," *J. Imaging Science and Technology*, vol. 45, no. 3, 2001, pp. 283–290.
5. T. Cholewo and S. Love, "Gamut Boundary Determination Using Alpha-Shapes," *Proc. 7th Color Imaging Conf., Soc. for Imaging Science and Technology*, 1999, pp. 200–203.

transformation operations. For this, we modified the hue and saturation channels to form a 2D Cartesian plane. The assumption that hue varies along the circumference of a circle with a radius of 1.0, while saturation varies [0.0 . . . 1.0] from the circle's center to its perimeter, motivated this modification. Next, we convert these polar coordinates into Cartesian space to obtain the new HS_x and HS_y coordinates in $[-1 . . . 1]$, which express the color variations on the hue-saturation plane. This approach also resolves the problem of the hue component being modulus—that is, as hue ranges from 0 to 360 degrees, hue is red at both 0 and 360 degrees.

Color range definition

In our system, a *color range* is the gamut that includes the group of colors appearing in a selected image region. The defining colors are expressed as 3D points in the perceptual color space (CIELAB/HSV), and the color range is simply the convex hull enclosing these points. To find the color range of a particular region within the image, we convert the colors of all pixels within that region into points in the perceptual space, then create their enclosing 3D convex hull. At this point, the user can manipulate the 3D color objects in the provided CAD

environment, or define a completely new object by drawing the enclosed points and performing geometric operations on the resulting polyhedra in the perceptual color-space domain. The system visualizes the color-range objects in the GUI using GPU fragment programs that convert the associated CIELAB or HSV position into an RGB color for every drawn point in real time, even while the user manipulates the color objects.

In the CIELAB color space, the convex hull of an entire image region's colors includes not only all the colors that currently appear in that image region but also any possible combination of these colors. We can explain this through the color range's geometric interpretation. Every contributing color is actually a 3D point in the color space. A color mix is the interpolation of two points, which will always be somewhere between those points in 3D space. Hence, it will still fall within the convex hull already enclosing these two points. We derive an interesting observation from this property. If we create a digitized painting's color-range gamut, the most basic colors used will always reside on the gamut's edges, and, in all likelihood, these were the colors on the artist's palette. On the other hand, everything in the interior of this gamut is the result of mixing the outside colors of

the basic palette. Later, we describe an application that uses this property.

Color transformation

The transform operation converts all colors of a source color gamut so they'll fit as a group into the target (or reference) color gamut. Here, the reference gamut might be the result of geometric manipulations on the original gamut or an entirely different gamut imported from another image or artificially created using the 3D interactive environment.

The transformation process converts every color of the source image gamut into its corresponding color in the reference gamut by taking the resulting color's distance from the reference gamut's center to be proportional to the distance of its counterpart from the source gamut's center. Chang and colleagues also used this correspondence to transfer the mood among different images.² By allowing the transformation only between color gamuts of the same perceptual color category, they preserve the source and reference images' basic perceptual color categories.³ In contrast, our system allows drastic changes in the color category, in the spirit of artistic editing.

We define the correspondence between the source and reference gamut (where the gamut is a convex hull) using the distance vector of the source point to the source hull's center. The new point in the reference gamut will lie on the segment drawn from the reference hull's center along the same direction, at a relative distance from the center given as:

$$P_{\text{ref}} = \frac{|P_{\text{src}} - C_{\text{src}}|}{|E_{\text{src}} - C_{\text{src}}|} \cdot (E_{\text{ref}} - C_{\text{ref}}) + C_{\text{ref}} \quad (1)$$

Here, P_{ref} is the corresponding point in the reference hull, P_{src} is the source point, C_{src} is the source hull's center, C_{ref} is the reference hull's center, E_{src} is the intersection of a ray of direction $P_{\text{src}} - C_{\text{src}}$ with the source hull's shell, and E_{ref} is the intersection of a ray of the same direction with the reference hull. Figure 2 illustrates this transformation.

During the color transformation, the system first converts the source image's pixels from RGB to perceptual space. After computing the corresponding reference point according to this method, the system converts it back to RGB in the final image.

Gamut clipping

In some cases, the user's manipulations on a color gamut could transform parts of the resulting 3D shape to the outside of the display device's bound-

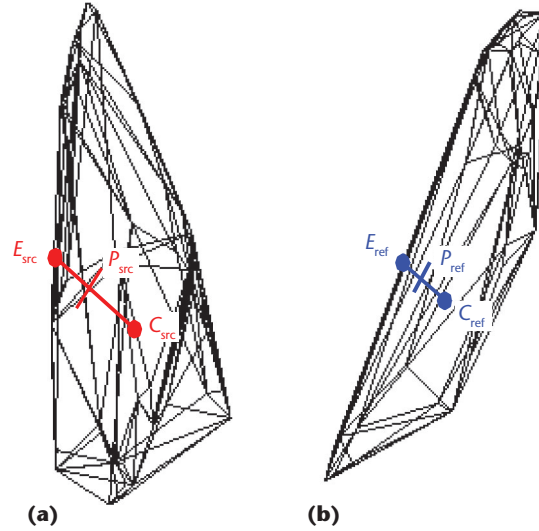


Figure 2. (a) Given the position of the color point P_{src} inside the source hull, the point E_{src} is the intersection of the vector $P_{\text{src}} - C_{\text{src}}$ with the hull. (b) Using the same direction vector, we can find E_{ref} . We can then position P_{ref} on $E_{\text{ref}} - C_{\text{ref}}$ at a relative distance from the reference hull's center.

aries. Left untreated, this situation would produce colors that are impossible to represent and visualize on the user's device. One way to deal with the problem is to use interactive collision-detection techniques to prohibit the user from moving or resizing color objects outside of this visible region. Our current application uses the much simpler approach of clamping the resulting colors to their closest RGB representation during visualization, while preserving the out-of-gamut representation for the internal computations.

Slight gamut clipping can also occur during the transformation process. This occurs when using the perceptual color space CIELAB, in which some device gamuts, including common CRT/LCD displays, are concave in certain regions. So, if any image colors lie on these concave boundaries, their convex hull will also contain some slightly out-of-gamut colors. This problem isn't acute in 8-bit-per-channel images; however, it's becoming a major concern with the continuous availability of much higher resolution raw images and Photoshop's support for 16-bit-per-channel images. This specific case isn't a problem for the HSV implementation, which is also a nonabsolute color space and in which all color points are fully enclosed within a predefined cylinder. We plan to extend our plug-in with a modified transformation process that uses freeform and mass-spring model-deformation techniques, which will better address this issue.

System implementation

To be practical for use in a creative trial-and-error process in which artists try many settings until they find one that works, such a system must be interactive. However, the methods we've described involve considerable computation to obtain the transformed color for each pixel. In our initial software implementation, most of the processing cost was consumed in finding the intersections between the

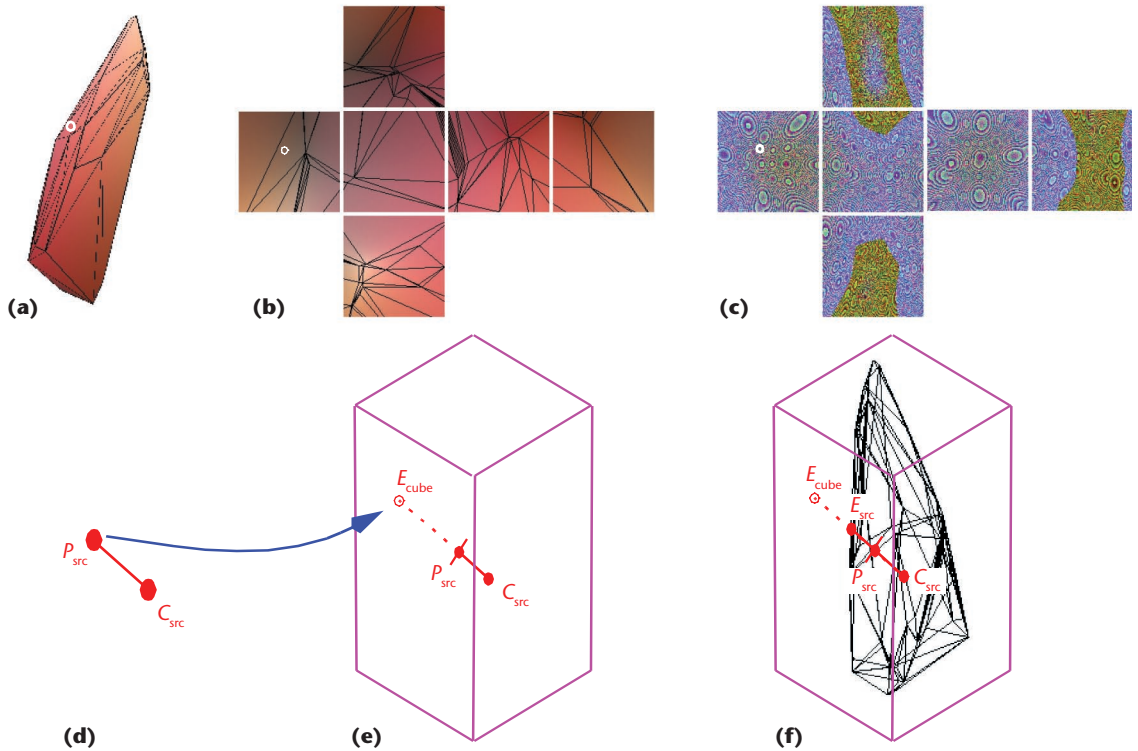


Figure 3. Using a cube map to speed the intersection lookups. We render (a) the color-range convex hull onto (b) a set of six panels, each representing a different viewing angle from inside a cube. The fragment program that we use for rasterization computes each point’s distance to the hull’s center and encodes it into the texture as a 32-bit float, resulting in (c) the final cube map. (d) During transformation, we compute P_{src} by converting the current pixel’s RGB color to an L^*a^*b point. (e) Using the direction of the vector $P_{src} - C_{src}$ we can look up texel E_{cube} from the cube map. (f) The lookup’s result will give the distance $E_{src} - C_{src}$, which we plug into Equation 1 and use to produce the transformed color.

ray and the convex hulls, E_{src} and E_{ref} , respectively. A GPU hardware-accelerated version of the system addresses this bottleneck operation by reducing it to a single texture lookup in a specially encoded cube map. This modification effectively reduces the response time to fractions of a second.

The intersection of an arbitrary ray with a convex hull (3D polyhedron) requires cycling through all the polyhedron’s facets, performing an intersection test for each. However, all the intersections required to calculate E_{src} involve rays that pass through the hull’s center and span to all possible directions. The same is true for E_{ref} in Equation 1. Cube maps encode the environment using six projections onto a surrounding cube’s sides.⁴ They’re easy to index and are currently implemented in most graphics hardware accelerators.

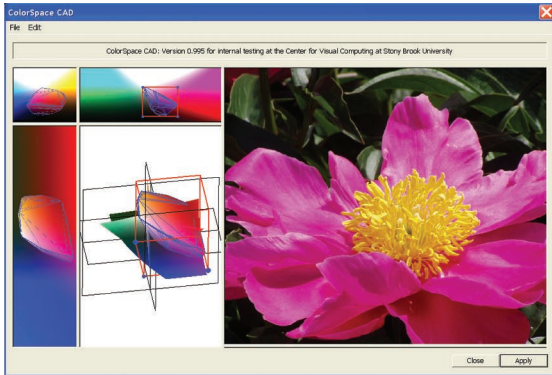
In our application, the surrounding scene is the convex polyhedron for the current color range. The information that must be stored in this structure is the distance from the object’s center to the intersecting polygon along every direction. Because this distance, $E_{src} - C_{src}$, doesn’t change throughout the color-range polyhedron’s lifetime, the sys-

tem can store it in the cube map data structure and use it per pixel. This operation’s lookup key is the direction of the $P_{src} - C_{src}$ vector, which the fragment shader computes for every pixel.

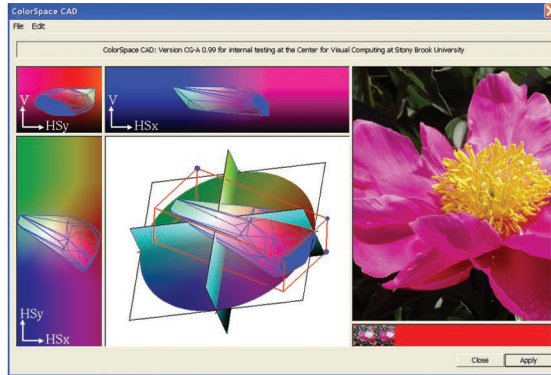
To create the cube map, we use a fragment program that only computes the distance of every point on the surface to the hull’s center. As we rasterize each triangle, we pass the coordinates of the triangle’s vertices as texture parameters. The hardware interpolates these parameters for every fragment rasterized. The fragment program subsequently uses them to compute the distance of every point of the triangle to the polyhedron’s center. To properly render each side of the cube map, we draw the structure from six different viewing angles. We encode the final result into 32-bit floats using the shaders’ pack and unpack utilities (this allows compatibility to previous-generation hardware, which doesn’t support rendering to float textures). Figure 3 illustrates this process.

User environment and features

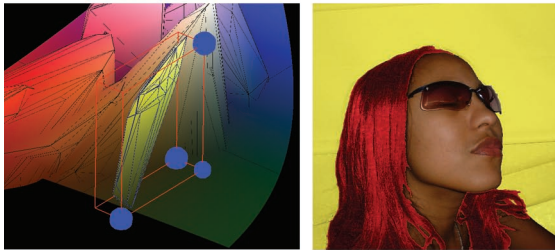
Our system lets users interactively manipulate (through an affine sculpting operation) the 3D



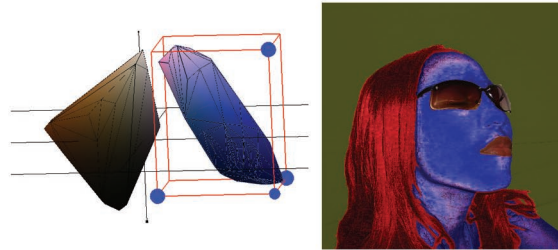
(a)



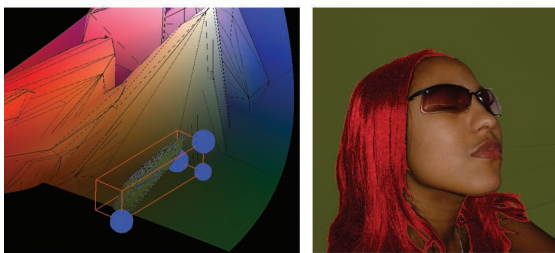
(b)



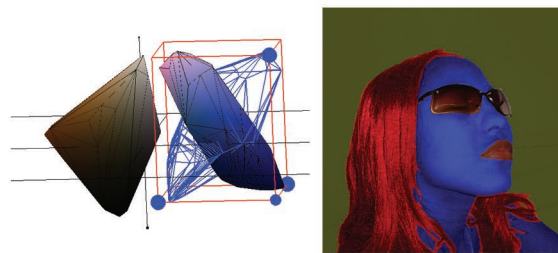
(c)



(e)



(d)



(f)

Figure 4. (a) The Color-Space CAD plug-in interface with a selection for editing in CIELAB mode. On the left is the color-space object in a combination of orthographic and a perspective editor views as in a general CAD system. The changes are immediately reflected in the preview image on the right of the window. The bottom right section browses the gamut library to activate gamuts from other images for a color transfer. (b) The plug-in interface when working in HSV mode. Editing actions in the color-space domain include: (c) initial color transfer for the background; (d) resizing (contracting) along the z-direction modifies the dynamic range, resulting in a flat color; (e) the source (tan) and reference (blue) hulls aren't aligned, resulting in unpleasant artifacts; and (f) an alternative approach rotates only the internal vectors.

color space. Our Color-Space CAD Photoshop plug-in interface (detailed in Figure 4) has two main components:

- the color-space windows, where the color ranges are visualized and manipulated as solid shapes in perceptual space through a combination of 2D orthogonal projections and a 3D perspective projection; and
- the image preview window, which provides immediate feedback regarding the user's actions' effect on the selected image region.

As the user interacts with the 3D color-space interface, the system applies the transformations

directly to the selected image region and renders them in real time in the image preview window.

Selection mask

Photoshop invokes the Color-Space CAD plug-in after the user selects a region using the advanced selection tools. These tools include color-range selection, magic-wand, lasso, and magnetic lasso tools, in combination with selection modifications such as antialiasing, feathering, and smoothing. These tools all result in an 8-bit selection mask, which our plug-in uses to adjust the selection boundaries. Photoshop uses the 8-bit range to mark every pixel with tags varying from not selected (mask = 0) to fully selected (mask = 255).

Color-space visualization

As Photoshop calls the plug-in, the plug-in creates a 3D solid representation of the selected region's color gamut. We represent the $L^*a^*b^*$ color space in 3D by assigning the L^* channel to the z axis and the a^* and b^* channels to the x and y axes, respectively. Fragment programs convert each pixel's coordinates back to the corresponding RGB colors, so that the objects visually carve out a 3D color volume. We represent the modified HSV color space (shown in Figure 4b) using the transformation discussed earlier, with the V (intensity value) component mapped to the z axis, and the HSx and HSy components mapped to the x and y axes. We divide the color-space visualization display into four resizable components, consisting of three orthogonal projections and a perspective view of the color object. The 2D and 3D views include a host of editing features so users can further fine-tune the color-transfer process, with support available at multiple levels.

The main display (perspective view) handles gamut clipping by discarding all nonvisible pixels when carving the axial planes, but clamps the colors toward their closest boundary in displaying the color gamut shapes. In contrast, the 2D orthogonal displays extend the out-of-gamut regions by drawing the nonvisible colors clamped to their closest in-gamut equivalents.

Alternative 2.5D visualization and editing

To make Color-Space CAD more accessible to users familiar with the existing 2D manipulation tools, we provide an alternative 2.5D depth-visualization capability. This additional display augments the existing three orthogonal 2D displays by adding a depth-perception layer. This display uses a checkerboard pattern in which the front- and back-most layers of the projected color gamut appear in alternating squares. A large color difference in the front and back layers (caused by a large depth difference) becomes immediately apparent in the checker display as high contrast, giving a better perception of the shape's actual depth. A more homogeneous look would suggest a thin object of much less depth.

Basic editing support lets users directly edit the selected region's 3D gamut object. The interface's most intuitive operation is translating the 3D color objects. Translating the color object along the z axis affects the target image region's luminance (CIELAB) or intensity value (HSV). Moving the object along the xy plane affects the target image region's chroma in both perceptual color spaces. The color of the objects in the 3D ma-

nipulation window gives additional hints regarding the final image's appearance, simultaneous to updating the resulting image. Resizing the objects intuitively affects the color region's dynamic range along the resize direction. For example, we can accentuate tone differences within the same color by enlarging the color object, or flatten the resulting region's color by contracting the color hull (Figure 4d). We perform these operations in the orthographic view using a 2D resizer box, or in the perspective view using a 3D cube widget. We initially adjust the cube's orientation to the minimum enclosing cube for the given object. We compute this cube by performing principal component analysis (PCA) on the color gamut. We can also rotate the resizer cube to allow resizing along arbitrary directions. Finally, the system also lets us rotate the gamut around its own center or around the color-space center. This operation makes more sense in HSV mode, where rotations of the object around the V axis result in consistent hue changes.

Import reference gamut

In addition to the free-hand approaches described so far, Color-Space CAD also supports example images, allowing the direct transfer of the color gamut from a separate reference image to the current user selection. Users can further modify the imported color gamut using the controls described previously. The plug-in interface supports this functionality by letting users select the desired reference gamut from a library of previously processed image regions with their color gamuts. Allowing color transfers between any arbitrary color gamuts can cause quantization artifacts when either the shape or the orientation of the source and reference gamuts differ significantly. That is, the problem arises when a short axis in the source gamut coincides in orientation with a longer axis in the reference gamut. In this case, a neighborhood of pixels spanning a small range of contrast in the original image must map onto a larger contrast range in the resulting image. The new contrast difference between neighboring pixels in the selected image region is now much steeper, resulting in the pronounced quantization artifacts illustrated in Figure 4e.

Gamut alignment dequantization

The system automatically aligns the selected reference gamut to the orientation of the image source gamut. It does this by performing PCA on both gamuts and rotating the major axis of the reference gamut to match the orientation of the source

gamut axes. This approach eliminates the quantization artifacts; however, the resulting rotation effectively changes the reference gamut's color statistics. Hence, the resulting color transfer doesn't completely represent a transfer to the selected reference image color region.

As an alternative to rotating the reference gamut and changing the resulting color mood, we use a different rotation in the transformation process, which we apply on the reference vectors ($P_{ref} - C_{ref}$) just before using Equation 1. This internal vector transformation avoids the quantization artifacts explained previously, without rotating or changing the reference gamut itself, and effectively preserves the intended mood of the target image region. Figure 4f illustrates the effect of vector-alignment dequantization. This method ensures that a short axis's orientation from the source hull won't coincide with the orientation of a long axis of the reference hull by applying an appropriate rotation to the reference hull's coordinate system. We compute the rotation using PCA similarly to its use in the gamut alignment (the wireframe hull in Figure 4d). We then rotate the vectors into the original reference hull, using the intended color gamut in the mapping. This rotation can result in mappings in which a source image area of varying hue changes to varying luminance in the result image.

This rotation approach isn't conceptually wrong; in fact, it's in keeping with the reference gamut statistics. It also complies with the approach's two main goals: to preserve the user's reference gamut, and to avoid mapping quantization artifacts. The tradeoff associated with our dequantization technique is similar to the tradeoff associated with replacing aliasing with blur in sampling tasks. Nevertheless, if the effects of the dequantization are unacceptable, yet users want to avoid the quantization without severe gamut changes, they can alternatively widen or shorten the reference gamut using the scaling tools.

An additional switch lets users convert the region to gray scale. This instructs the rendering fragment programs to use only the luminance channel for the final image. Users can then adjust the region's brightness and contrast by manipulating the color object's position and size (stretch) along the z-direction to enhance some features. However, to preserve subtle features that depend only on chromatic contrast, users must pursue a saliency-preserving approach.⁵

We can also save the system's entire state, including the color gamuts and their associated modifications, on disk for later retrieval using an extensible XML format. This lets Color-Space



Figure 5. The “virtual silkscreen” Marilyns. These images illustrate how we can use this method to produce extreme results for fun and as an artistic tool. These images were the original inspiration for the Photoshop tool.

CAD maintain its library of preprocessed gamuts for users, which can include gamuts shared over the Internet.

Artistic applications

We tested Color-Space CAD on a standard configuration comprising a Pentium 4 processor running at 2 GHz with 512 Mbytes RAM and an NVidia GeForceFX 6800 graphics board. The hardware-accelerated implementation's response time when operating on 3-megapixel images is less than 30 milliseconds, enabling real-time manipulations for even large images.

We tested the system in different image-processing application scenarios, mostly for artistic purposes. One example application is the virtual silkscreen, in which our artistic collaborator sought to create enhanced silkscreen-like images resembling the style of Andy Warhol's silkscreens. The images in Figure 5 illustrate our attempts to create a set reminiscent of Andy Warhol's 1962 Marilyn Monroe silkscreen series. He made the originals by projecting the positives of black and white photos onto a silkscreen. The original photograph was from Monroe's 1953 movie, *Niagara*. However, in contrast to Warhol's original technique, color-space sculpting capitalizes on the statistics of color images. To illustrate this capability, we used an alternative image of Monroe from the same movie in a similar setting and recreated an

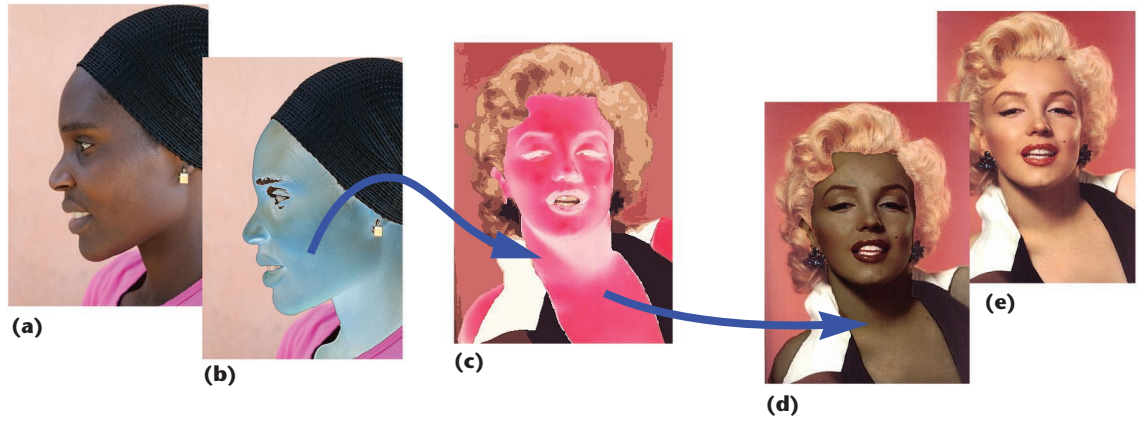


Figure 6. Using color-space sculpting to transfer skin color: (a) reference image (African woman), (b) selected region in reference image, (c) source image (Marilyn Monroe) with selected region, (d) Monroe after the conversion, and (e) original image for comparison.



(a)



(b)

Figure 7. Using color-space sculpting to enhance selected image features. We transformed selected features to Americana-inspired colors using Color-Space CAD manipulation and gamut imports from other images. (We used aqualegia flowers for the purple-bluish chair and mustard seeds for the grass.)

enhanced silkscreen-style effect. Transferring an entire color gamut onto a region results in full-range images, which also preserve some texture and lighting from the original image. These results can't be reproduced simply by changing some regions' hue element, because the entire transformation process also depends on the source's shape and reference color gamuts.

Another application of the color-range transfer process involves transferring the color gamut of a selected region from one image to the selected region in another image. So, if we're working with portrait or full-body images of individuals with different color skin complexions, we could transfer only the skin color from one picture to another. Figure 6 shows the results of transferring the skin complexion from a photograph of an African woman to a photograph of Monroe. For this transfer, we isolated a selection that includes the African woman's face as the reference color range, and Monroe's face and neck as the source color range. Both the source face and the reference selection include a multitude of colors that define the blonde Caucasian and the dark African complexions.

Similarly, we can achieve other difficult color transfers involving a range of colors to be converted, such as hair color. This ability could be useful in modeling and styling applications that must provide a convincing preview before a make-over takes place. The images in Figure 7 illustrate similar uses of Color-Space CAD.

Applications in visualization

As Bauer and colleagues demonstrated in their fundamental perception study,⁶ given a set of colors enclosed by their color-space convex hull, additional colors only pop out (become noticeable pre-attentively) if they fall outside this convex

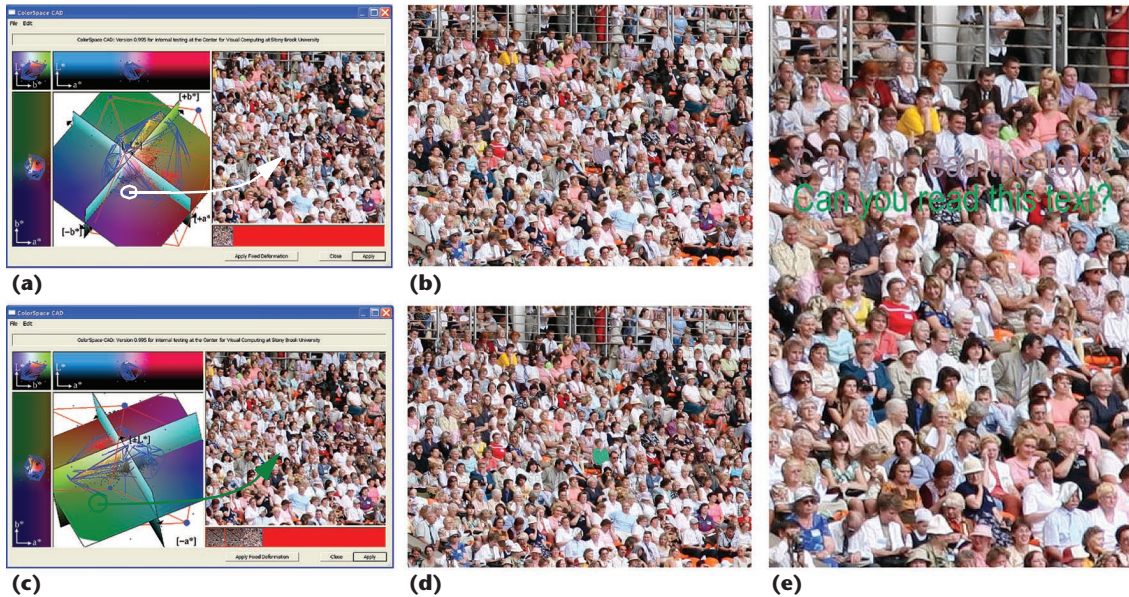
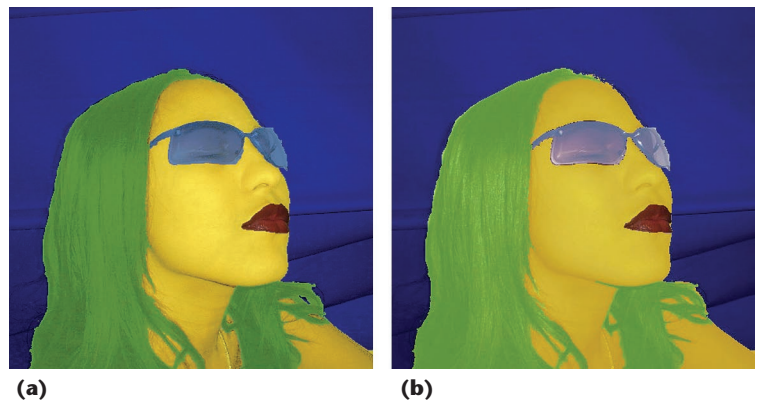


Figure 8. Photograph-highlighting and text-annotation example. (a) The user selects a purple color from inside the image’s color gamut to highlight a person’s t-shirt. (b) The plug-in screen shows the highlighted photo. (c) We choose a light green lying outside the gamut and (d) the color highlighted is immediately apparent. (e) We compare three colors for text annotation as sampled from the image color space. As we expect, line 1 has been sampled from inside the gamut and line 2 was sampled outside of the gamut.

hull. Our system provides a natural interface for using this fundamental finding. For example, a potentially useful application is the semiautomatic selection of colors for highlighting and annotating color photographs. Figure 8 demonstrates such an application. Here, the system first calculates the gamut convex hull of the photograph to be highlighted or annotated. Next, it lets the user choose from a selection of colors situated outside the gamut. If the image’s gamut completely fills the color space, leaving no apparent choices for colors outside the gamut, the user can resize the gamut using any of the gamut operations discussed to reduce its dynamic range in an arbitrary direction to make space for the color of the new feature to be introduced.

Comparison to existing packages

Color-Space CAD offers a versatile environment for color transformations by letting users manipulate 3D color shapes and practically sculpt an image’s color-space domain. This feature offers an advantage over similar features in the most advanced photography processing packages, such as Adobe Photoshop. To test our claim, we tried to replicate a virtual silkscreen (produced by Color-Space CAD) using Photoshop’s layer operations. We started with the original color photograph and, after segmenting the image, matched the desired colors for each layer. We defined the colors as flat color overlays over the image’s segmented areas. We assigned each



layer/overlay a mixing operation with the layer underneath it, which in our case was the original image. These predefined operations included darken, multiply, difference, and exclusion. For our example, we used Photoshop’s linear dodge operation for applying hair color and its hard light operation for applying face color. We used the color operation to assign the background and other regions. This operation keeps the original luminance channel and replaces the chrominance channels with the assigned colors. We selected these operations to emulate the results of color-space sculpting.

Figure 9 shows the two images side-by-side. Although we used most of Photoshop’s available options in our effort to achieve a comparable image, these global operations don’t offer enough control to fine-tune the results to the extent that color-space sculpting permits. As the example shows, the

Figure 9. Comparison with layer operations in Adobe Photoshop: (a) Color-Space CAD and (b) Adobe Photoshop.

image in Figure 9b matches the colors and some features of the image in Figure 9a, but it's still less refined. Our system's flexibility let the artist eliminate the unwanted hair highlights, while keeping the overall color assignment. The system offers this unexplored level of flexibility by allowing interactive and more direct manipulation and sculpting of the color space.

Of course, other packages could replicate all of Color-Space CAD's functionality through a long trial-and-error process, using the provided 1D layer operations. However, our system provides fast shortcuts to these functionalities by allowing the manipulation to take place in 3D, directly on the color object.

During recent interactions with various artists and color professionals in the context of our software demonstrations and an oral sketch presentation of an earlier prototype,⁷ we found that such an interface has long been awaited and promises to be useful. To provide better accessibility of our plug-in to the color artists and color professionals communities, we've extended our system to support the HSV color space in addition to CIELAB. Our current focus is on exploring further applications with these professionals and investigating additional functionalities specific to the perceptual color spaces we use. The current version of the plug-in is due for a public release in the summer of 2008, in the hope of wider usage and feedback. ■■

Acknowledgments

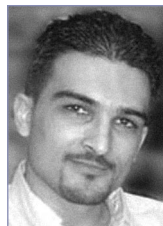
We thank our artistic collaborator, Odaly Cruz, for her feedback during the development and testing of this system, and for creating the "Virtual Silkscreen Marylins" and other examples in this article. We greatly appreciate the artistic feedback of Philip Sanders, associate professor at the College of New Jersey, and Aviv Yaron, 2D technical director at Cinesite Europe. Many thanks to Youngha Chang for our useful conversations about color transfers, and to Keiji Uchikawa for allowing us access to his perceptual color research data. Our gratitude also goes to Adobe Systems for providing complimentary access to the Photoshop software developer's kit. US NSF Career grant ACI-0093157 and US NIH grant 5R21EB004099-02 partially supported this work.

References

1. G. Wyszecki and W.S. Stiles, *Color Science: Concepts*

and Methods, Quantitative Data and Formulae, 2nd ed., Wiley-Interscience, 2000.

2. Y. Chang, S. Saito, and M. Nakajima, "A Framework for Transfer Colors based on the Basic Color Categories," *Proc. 2003 Computer Graphics Int'l (CGI 2003)*, IEEE CS Press, 2003, pp. 176-181.
3. K. Uchikawa, I. Kuriki, and H. Shinoda, "Categorical Color-Name Regions of a Color Space in Aperture and Surface Color Modes," *J. Light and Visual Environment*, vol. 10, no. 1, 1996, pp. 26-35.
4. R.A. Hall, "Hybrid Techniques for Rapid Image Synthesis," *Image Rendering Tricks, Course Notes 16 for Siggraph 1986*, T. Whitted and R. Cook, eds., ACM Press, 1986, p. 986.
5. A.A. Gooch et al., "Color2gray: Saliency-Preserving Color Removal," *ACM Trans. Graphics (Proc. Siggraph)*, vol. 24, no. 3, 2005, pp. 634-639.
6. B. Bauer, P. Jolicoeur, and W. Cown, "Distractor Heterogeneity Versus Linear Separability in Colour Visual Search," *Perception*, vol. 25, no. 11, 1996, pp. 1281-1293.
7. N. Neophytou and K. Mueller, "Color-Space CAD," *ACM Siggraph 2006 Sketches*, ACM Press, 2006, p. 122.



Neophytos Neophytou is a senior scientist at Intelipix LLC, where he leads research and development on GPU-accelerated geospatial intelligence and visualization. His research interests include graphics, scientific visualization, medical imaging, and geospatial visualization. Neophytou has a PhD in computer science from Stony Brook University. Contact him at nneophyt@cs.sunysb.edu.



Klaus Mueller is an associate professor in the Computer Science Department at Stony Brook University, where he also holds coappointments in the Biomedical Engineering and Radiology Departments. His research interests are computer and volume graphics, visualization, visual analytics, medical imaging, and computer vision. Mueller has a PhD in computer science from the Ohio State University. He is a senior member of the IEEE and the IEEE Computer Society. Contact him at mueller@cs.sunysb.edu.

For further information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.