Vol. 13, No. 4: 35–51

GPU Color Constancy

Marc Ebner
Eberhard Karls University Tübingen

Abstract. A sensor located inside a digital camera is only able to measure the light that is reflected by an object. The reflected light varies with the spectral power distribution of the illuminant. Hence, images taken with a digital camera may show a strong color cast if an incorrect white balance setting has been chosen. Such a color cast may also be due to an automatic white balance not working correctly. In contrast, colors perceived by a human observer appear to be approximately constant. Algorithms for automatic white balance try to mimic this ability and compute a color-corrected image that appears to have been taken under an illuminant with a uniform power distribution. I show how color-constancy algorithms can be implemented very efficiently on modern graphics processing units.

1. Introduction

A sensor inside a digital camera measures the light that is reflected from the objects of the scene. Some of the light is absorbed; the remainder is reflected and is able to enter the lens of the camera where it is measured. The reflected light varies with the type of illuminant used. Some light sources emit more light toward the red and green part of the spectrum and therefore appear to be very yellowish. If such an illuminant is used, then the scene will come out very yellowish in a photograph. The color temperature of daylight also varies during the day. Digital cameras can use post-processing to remove such a color cast. They try to compute an image that appears to have been taken under an illuminant with a uniform power spectrum. In digital photography, this

1086-7651/08 $0.50 per page

is termed automatic white balance. Many digital cameras also allow the user to select a particular color temperature. Some have pre-settings for sunlight, cloudy sky, neon light, or illumination by a light bulb.

In contrast to a machine sensor, the colors perceived by a human observer appear to be remarkably constant. This ability is called color constancy [Zeki 99, Ebner 07]. For machine-vision applications, it is very important to mimic this ability. For instance, color-based object recognition becomes very difficult if objects appear to change their color based on the type of illuminant used. Therefore, it makes sense to first compute a color-constant descriptor [Geusebroek et al. 01].

A number of algorithms have been developed to address the problem of color constancy. The problem can only be solved if some assumptions are made. I present a color-constancy method based on local space average color, which is easily implemented on a graphics processing unit (GPU). Algorithms based on local space average color have been shown to work very effectively in object-recognition tasks [Ebner 08]. The advantages of this method are: (1) it also works in the presence of multiple spatially varying illuminants, and (2) it is readily implemented with three or four lines of code. It can be used to (1) remove a color cast from textures provided that the assumptions made by the algorithm are fulfilled and (2) provide color constancy for GPU-accelerated computer vision, robotics, animation, or interactive gaming applications. I also show how intrinsic images, which only depend on the reflectance component, are efficiently computed through a pixel shader. The algorithm assumes that the illuminant can be approximated to have the spectral power distribution of a black-body radiator. Such intrinsic images are free from shadows, which makes them very useful for image segmentation.

2. Color Image Formation and the Gray World Assumption

A standard model of color image formation is given as follows (see Horn [Horn 86] for an introduction to radiometry). Suppose that a light source with radiance $L(\lambda)$ at wavelength λ is illuminating a scene. Let $R(x, y, \lambda)$ be the reflectance of an object patch that is depicted at image position (x, y). For a diffusely reflecting surface patch, i.e., a Lambertian reflector, the irradiance $E(x, y, \lambda)$ falling onto the object patch is given as $E(x, y, \lambda) = L(\lambda) \cos \alpha$ where $\cos \alpha = \mathbf{N}_S \mathbf{N}_L$ is the angle between the normal vector $\mathbf{N}_S$ of the surface patch and the unit vector $\mathbf{N}_L$ pointing from the surface patch into the direction of the light source. Usually three types of sensors are used that measure the light in the red, green, and blue parts of the spectrum. Let $S_i(\lambda)$ with $i \in \{r, g, b\}$ be the spectral sensitivity of the sensor i. Then the energy measured by the sensor i is given as

$$I_i(x, y) = \mathbf{N}_S(x, y)\mathbf{N}_L(x, y) \int S_i(\lambda) R(x, y, \lambda) L(\lambda) d\lambda.$$

Let us now assume that the sensitivity of the sensor is very narrow band, i.e., responds only to a single wavelength λ_i. We then have $S_i(\lambda) = \delta(\lambda_i - \lambda)$, where $\delta(\lambda)$ is Dirac's delta function. This gives us

$$I_i(x,y) = G(x,y)\int \delta(\lambda_i - \lambda)R(x,y,\lambda)L(\lambda)d\lambda = G(x,y)R_i(x,y)L_i.$$

Note that instead of $R(x,y,\lambda_i)$ we write $R_i(x,y)$, instead of $L(\lambda_i)$ we write L_i, and we replace the scalar product between the two vectors $\mathbf{N}_S(x,y)$ and $\mathbf{N}_L(x,y)$ with a geometry factor $G(x,y)$. We now see that the color of the illuminant L_i scales the energy $I_i(x,y)$ measured by the sensor i at position (x,y). If we allow for multiple spatially varying illuminants $L_i(x,y)$, then we have

$$I_i(x,y) = G(x,y)R_i(x,y)L_i(x,y).$$

Let $c_i(x,y)$ be the color information stored inside a texture at position (x,y). Then we have $c_i(x,y) = I_i(x,y)$ if we are working with texture data inside a linear color space. We will refer to the index i from now on as color channel i. We have $c_i(x,y) = I_i(x,y)^\gamma$ for some gamma factor γ if a gamma correction has been applied. Let us first assume that we work with a linear color space, i.e., $c_i(x,y) = I_i(x,y)$. Assuming that we had an estimate of the color of the illuminant $L_i(x,y)$ for every image pixel (x,y), we could compute a color-corrected output image by dividing each pixel $c_i(x,y)$ by this estimate of the illuminant:

$$\frac{c_i(x,y)}{L_i(x,y)} \approx \frac{G(x,y)R_i(x,y)L_i(x,y)}{L_i(x,y)} = G(x,y)R_i(x,y). \tag{1}$$

We obtain the product between the geometry factor $G(x,y)$ and the reflectance $R_i(x,y)$, which is independent of the color of the illuminant.

But how can we obtain an estimate for the color of the illuminant given a single input image? Clearly, some assumptions have to be made to solve the problem, as we have seven unknowns (the geometry factor, three reflectance components, and three illuminant components) but only three known values (the measured energy components) per texel. A simplifying assumption that is frequently made is to assume that the illuminant is constant across the entire image, i.e., we have $L_i(x,y) = L_i$. An additional assumption, due to Buchsbaum [Buchsbaum 80], is to say that on average, the world is gray. This assumption holds, provided that the viewed scene is sufficiently diverse and contains many differently colored objects. Let n be the number of texels in an image or texture. Then the global average a_i for color channel i of all texels is given by

$$\begin{aligned} a_i &= \frac{1}{n}\sum_{x,y} c_i(x,y) = \frac{1}{n}\sum_{x,y} G(x,y)R_i(x,y)L_i \\ &= L_i \frac{1}{n}\sum_{x,y} G(x,y)R_i(x,y) = L_i E[G]E[R_i], \end{aligned}$$

where $E[G]$ denotes the expected value of the geometry factor and $E[R]$ denotes the expected value of the reflectance values. The last equality holds because it can be assumed that geometry and reflectance are two independent properties.

If we now assume that a large number of differently colored objects are contained in the texture, then we may view the reflectances distributed over the texture as a random variable that is uniformly distributed over the range $[0,1]$. In other words, we are saying that all colors are equally likely. We can then compute the expected value of the reflectances. We have

$$E[R_i] = \frac{1}{n}\sum_{x,y} R_i(x,y) = \frac{1}{2}.$$

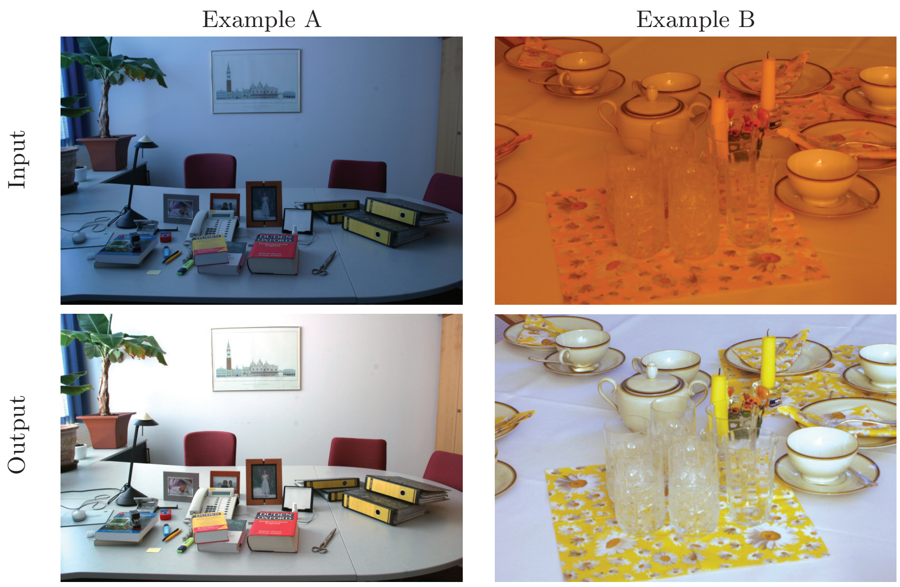

Figure 1. Output produced by the gray world assumption for two sample images.

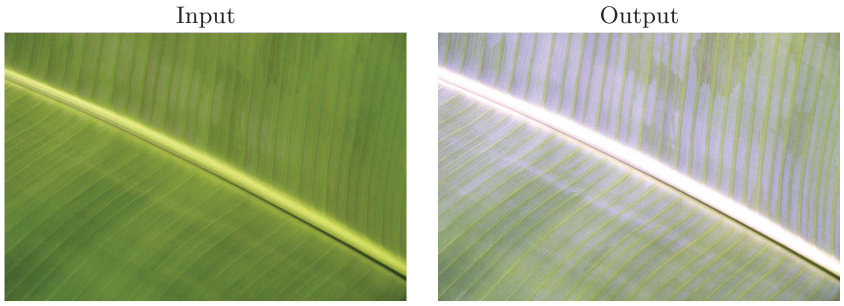

Figure 2. There has to be a sufficiently large number of different colors in the image for the gray world assumption to hold. The sample image on the left is a close-up of a leaf from a banana plant. In this case, the greenishness of the image is due to the leaf of the banana plant.

We now see that the color of the illuminant L_i can be estimated by computing the average color of all image pixels:

$$L_i = f a_i,$$

with $f = \frac{2}{E[G]}$. If our scene is composed of Lambertian reflectors that are illuminated at a right angle, then we have $f = 2$. We can now perform a color correction using Equation (1). Figure 1 shows how the gray world assumption works for two sample images. It removes the color cast quite nicely. Note that for the gray world assumption to hold, there has to be a sufficiently large number of different colors in the image. What happens if this assumption is not fulfilled is illustrated in Figure 2 where a close-up of a leaf from a banana plant is processed using the gray world assumption. These negative effects can be reduced if a segmentation of the image is performed [Gershon et al. 87] or if a histogram is computed.

3. A Color-Constant Pixel Shader Based on the Gray World Assumption

Ebner has shown that local space average color may be used to estimate the illuminant locally for each image pixel [Ebner 04b]. We estimate the color of the illuminant $L_i(x, y)$ as

$$L_i(x, y) = f a_i(x, y).$$

Using $f = 2$ works well in practice. Brightness is increased for dark areas and reduced in very bright areas. Algorithms based on local space average color

Pixel Shader 1 (GLSL code)

```
c=texture2D(textureColorSampler,gl_TexCoord[0].st,0.0);
a=texture2D(textureColorSampler,gl_TexCoord[0].st,level);
c/=2*a;
c=pow(c,0.4545); // gamma correction
// insert shading code here
gl_FragColor=c;
```

Figure 3. Pixel Shader 1. Each pixel is divided by twice the local space average color. This shader assumes that the image data is stored in a linear color space. Hence, a gamma adjustment is applied before output.

have been shown to work very well on histogram-based object recognition tasks [Ebner 08].

If a texture is applied to a rectangle, one can use mip mapping to compute local space average color. When mip mapping is used, a scale space of the texture is automatically constructed. It contains the original full-size image at the lowest level and a hierarchy of images where each image of the next higher level is constructed by averaging texels at the current level. The highest level contains a single texel with the average color of all of the texels. Local space average color may be obtained from intermediate hierarchies of the mip map.

The mip map level can be computed as follows. In order for the gray assumption to hold, the mip map level has to be sufficiently large. It should extend over at least 30% of the image. Let l be the mip map level and $s = \max\{\text{width}, \text{height}\}$, where width is the width of the image and height is the height of the image in pixels, then it should hold that

$$2^l = 0.3s.$$

In other words, we can compute the mip map level l as

$$l = \log_2(0.3s).$$

Assuming that the texture data is stored in a linear color space, then a color-corrected output can be computed using the pixel shader code as shown in Figure 3, where it is assumed that `level` is equal to the level l.

Usually the texture data is stored in a nonlinear color space such as the sRGB color space. In this case, a gamma correction has been applied to the RGB values. The following transfer function is used by the sRGB standard [Poynton 03]:

$$\text{gamma}_{\text{sRGB}}(x) = \begin{cases} 12.92x & \text{if } x \leq 0.0031308, \\ 1.055x^{\frac{1}{2.4}} - 0.055 & \text{if } x > 0.0031308. \end{cases}$$

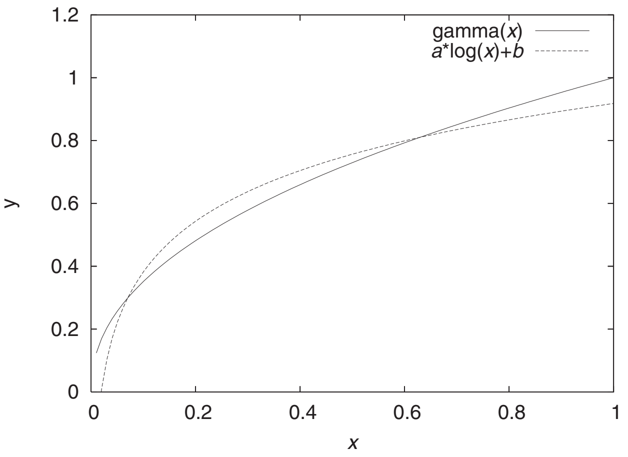

Figure 4. Fit of the function $a \log x + b$ with $a = 0.233359$ and $b = 0.918031$ to the function $x^{\frac{1}{2.2}}$. The general behavior of the two functions is similar for the range $[0, 1]$.

The overall transform is best described by

$$\text{gamma}(x) = x^{\gamma},$$

with $\gamma = 1/2.2$, i.e., we have $c_i = I_i^{\gamma}$. This gamma function can be approximated by the function $\text{gamma}(x) \approx a \log x + b$ with $a = 0.233359$ and $b = 0.918031$. Figure 4 shows these two functions. The behavior of the two functions is similar. This similarity between the logarithmic function and a power law was also noted by Wyszecki and Stiles [Wyszecki and Stiles 00].

If the logarithmic function is applied to the RGB values, then the mip mapping essentially computes the the geometric average of the measured data values I_i:

$$\begin{aligned} a_i &= \frac{1}{n} \sum_{x,y} c_i(x,y) = \frac{1}{n} \sum_{x,y} a \log I_i(x,y) + b \\ &= a \log \left(\prod_{x,y} I_i(x,y) \right)^{\frac{1}{n}} + b. \end{aligned}$$

We now have to subtract this average from the pixel data c_i to obtain a color constant descriptor o_i. Assuming $G(x,y) = 1$ and $L(x,y) = L_i$, we obtain

$$\begin{aligned} o_i(x,y) &= c_i(x,y) - a_i \\ &= a \log I_i(x,y) + b - a \log \left(\prod_{x,y} I_i(x,y) \right)^{\frac{1}{n}} - b \end{aligned}$$

$$
\begin{aligned}
&= a \log R_i(x,y) L_i - a \log \left(\prod_{x,y} R_i(x,y) L_i \right)^{\frac{1}{n}} \\
&= a \log R_i(x,y) + a \log L_i - a \log L_i - a \log \left(\prod_{x,y} R_i(x,y) \right)^{\frac{1}{n}} \\
&= a \log R_i(x,y) - a \log \left(\prod_{x,y} R_i(x,y) \right)^{\frac{1}{n}}.
\end{aligned}
$$

Using the gray world assumption again, i.e., we assume that the reflectances R_i are uniformly distributed over the range $[0,1]$, and using Stirling's approximation ($\log n! \approx n \log n - n$) for large n, we see that

$$\log \left(\prod_{x,y} R_i(x,y) \right)^{\frac{1}{n}} \approx \log \left(\prod_{x,y} \frac{i}{n} \right)^{\frac{1}{n}} = \log \left(\frac{n!^{\frac{1}{n}}}{n} \right) \approx \log \frac{(n^n e^{-n})^{\frac{1}{n}}}{n} = -1,$$

and we obtain

$$o_i(x,y) = a \log R_i(x,y) + a.$$

In order to obtain the reflectances, we need to apply the exponential function. We need to compute $o_i'(x,y) = \exp(o_i(x,y)/a)$. This would give us $o_i'(x,y) = c' R_i(x,y)$, which is linear in the reflectances R_i with $c' = e^{\frac{1}{a}}$. However, since we now have linear reflectances, we need to apply a gamma correction $\text{gamma}(x) = x^{\frac{1}{\gamma}}$ in order to display these linear values. The inverse of this gamma correction, $\text{gamma}(x) = x^{\gamma}$, can be approximated by $e^{\frac{x-b}{a}}$. Hence, we just need to render

$$o_i(x,y) = c_i(x,y) - a_i - a + b,$$

Pixel Shader 2 (GLSL code)

```
c=texture2D(textureColorSampler,gl_TexCoord[0].st,0.0);
a=texture2D(textureColorSampler,gl_TexCoord[0].st,level);
c-=a+0.244459-0.918031;
// insert shading code here
gl_FragColor=c;
```

Figure 5. Pixel Shader 2. Local space average color is subtracted from the color of each pixel, and a normalization that is based on the gray world assumption is applied.

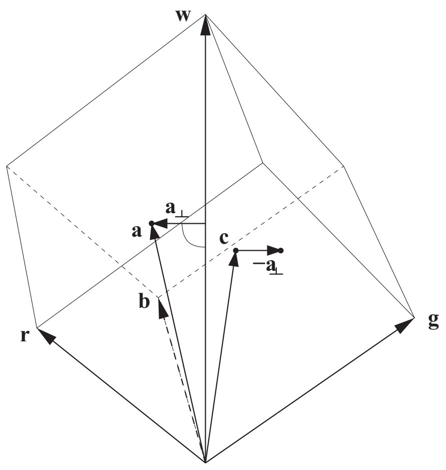

Figure 6. RGB color cube spanned by the three vectors **r**, **g**, **b**. The gray vector **w** runs from black to white. A color correction may be performed by shifting local space average color **a** onto the gray vector **w**. In this case, the perpendicular component $\mathbf{a}_\perp$ of **a** is subtracted from the color of the current pixel **c**.

with $a = 0.233359$ and $b = 0.918031$. Thus, the code performing this operation inside a pixel shader is given as shown in Figure 5.

An alternative method is to adjust only the color of the image and leave the brightness of the image intact. In this case, we use a shift that runs perpendicular to the gray vector as described by Ebner [Ebner 04a]. According to the gray world hypothesis, on average the world should be gray, i.e., local space average color should be positioned on the gray vector that runs through the RGB cube from black to white. If local space average color is offset from the gray vector, we can perform a color shift for each pixel that runs perpendicular to the gray vector, effectively moving local space average color onto the gray vector. This is illustrated in Figure 6. Such a color shift is also used by Ebner et al. [Ebner et al. 07] in integrating color constancy into the JPEG2000 framework. Let $\mathbf{a} = [a_r, a_g, a_b]^T$ be local space average color, then we first need to compute the component $\mathbf{a}_\perp$ that is perpendicular to the gray vector $\mathbf{w} = \frac{1}{\sqrt{3}}[1,1,1]^T$. The perpendicular component is given by

$$\mathbf{a}_\perp = \mathbf{a} - (\mathbf{a}^T\mathbf{w})\mathbf{w}.$$

We then subtract this component from the color of the image pixel $\mathbf{c} = [c_r, c_g, c_b]^T$ to obtain a color-corrected output $\mathbf{o} = [o_r, o_g, o_b]^T$:

$$\mathbf{o} = \mathbf{c} - \mathbf{a}_\perp = \mathbf{c} - \mathbf{a} + \frac{1}{3}(a_r + a_g + a_b)[1,1,1]^T.$$

Using pixel shader code, this operation is performed as shown in Figure 7.

Pixel Shader 3(GLSL code)

```
c=texture2D(textureColorSampler,gl_TexCoord[0].st,0.0);
a=texture2D(textureColorSampler,gl_TexCoord[0].st,level);
c-=a-(a[0]+a[1]+a[2])/3.0;
// insert shading code here
gl_FragColor=c;
```

Figure 7. Pixel Shader 3. Local space average color is computed, and only the component that is perpendicular to the gray vector **w** is subtracted from each pixel.

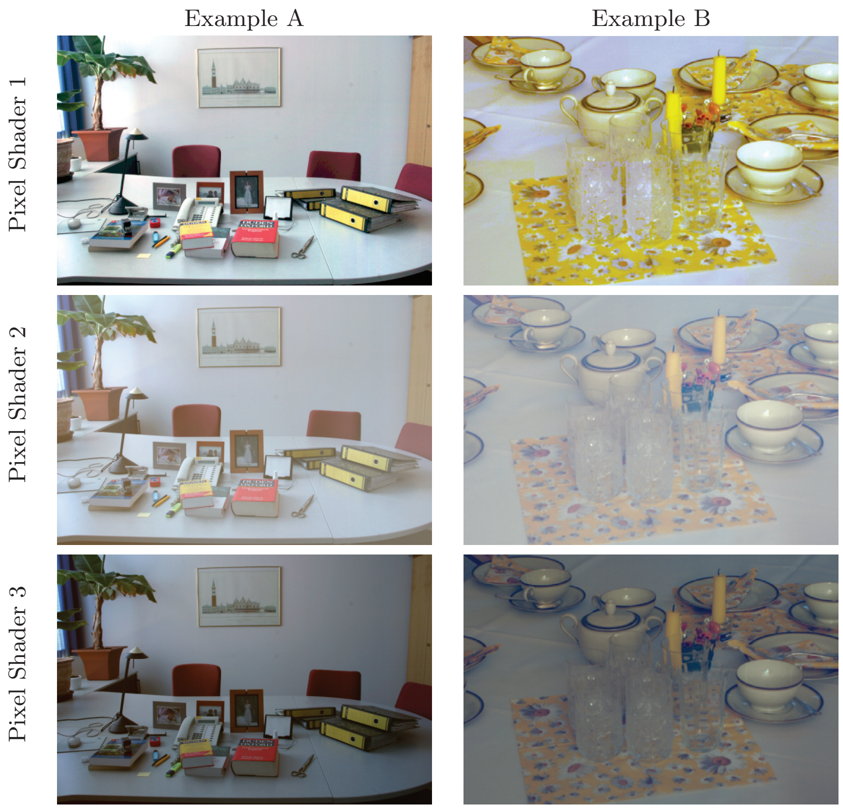

Figure 8. Output produced by the three pixel shaders for two sample images.

Figure 8 shows the output obtained for all three pixel shaders on the sample images from Figure 1. Pixel Shader 1 assumes a linear color space and works much like the gray world assumption except that it is based on local space average color. It is therefore able to handle nonuniform illuminants. Pixel Shader 2 assumes that colors are stored using the sRGB standard. Normalization is performed using the gray world assumption. Pixel Shader 3 works similarly except that the brightness of the input image is retained. In all cases, we see that the color cast is nicely removed. Pixel Shader 1 should be chosen if the image data is stored using a linear color space. Pixel Shaders 2 and 3 can be chosen if the image data is stored using the sRGB color space, i.e., a color space with a gamma correction already applied. The computed colors will differ between Pixel Shader 1 and Pixel Shaders 2 and 3 because the mip map approach computes the average color of image pixels, whereas for Pixel Shaders 2 and 3 the geometric average is computed.

In the following two sections, we show how intrinsic images can be computed by an appropriate pixel shader.

4. Computation of Intrinsic Images

Intrinsic images contain only one characteristic of the scene being viewed [Tappen et al. 02], e.g., a value that only depends on the reflectance of the object. Finlayson and Hordley [Finlayson and Hordley 01] have developed a method to compute intrinsic images for a calibrated camera. It is assumed that the camera's sensors are sufficiently narrow-band and that the illuminant can be approximated by a black-body radiator. Many natural light sources such as the flame from a candle or sunlight can be approximated by a black-body radiator. The radiance $L(\lambda, T)$ given off by a black-body radiator at a temperature T, measured in Kelvin, at wavelength λ is given by [Jähne 02]

$$L(\lambda, T) = \frac{2hc^2}{\lambda^5} \frac{1}{(e^{\frac{hc}{k_B T \lambda}} - 1)},$$

where $h = 6.626068 \cdot 10^{-34}$ Js is Planck's constant, $k_B = 1.3806 \cdot 10^{-23} \frac{\mathrm{J}}{\mathrm{K}}$ is Boltzmann's constant, and $c = 2.9979 \cdot 10^8 \frac{\mathrm{m}}{\mathrm{s}}$ is the speed of light. The temperature T, which is used to approximate many light sources, is usually no larger than 10,000 K. Thus, the equation of the black-body radiator can be simplified. Considering that the visible spectrum ranges from 400 nm to 700 nm, we have

$$e^{\frac{hc}{k_B T \lambda}} \gg 1$$

for $\lambda < 700$ nm and $T1$ 10,000 K. The result is a simple equation for the radiance $L(\lambda, T)$:

$$L(\lambda, T) = kc_1 \lambda^{-5} e^{-\frac{c_2}{T\lambda}},$$

with constants $c_1 = 2hc^2$, $c_2 = \frac{hc}{k_B}$, and k. The constant k can be used to model different intensities.

Given the above derivation for the color of the image pixels $\mathbf{c}$, we obtain

$$c_i = G \cdot R_i \cdot kc_1 \lambda_i^{-5} e^{-\frac{c_2}{T\lambda_i}}.$$

The coordinate index (x, y) has been omitted here. Note that the color of the illuminant only depends on the temperature of the black-body radiator. The product on the righthand side can be split into a sum by applying the logarithm to both sides:

$$\log(c_i) = \log(kG) + \log(c_1 \lambda_i^{-5} R_i) - \frac{c_2}{T\lambda_i}.$$

Only the last term depends on the temperature T. Finlayson and Hordley [Finlayson and Hordley 01] suggested to compute the difference ρ between two different color channels. This removes the first term. One obtains

$$\begin{aligned}
\rho_{rg} &= \log(c_r) - \log(c_g) = \log(\lambda_r^{-5} R_r) - \frac{c_2}{T\lambda_r} - \log(\lambda_g^{-5} R_g) + \frac{c_2}{T\lambda_g}, \\
\rho_{bg} &= \log(c_b) - \log(c_g) = \log(\lambda_b^{-5} R_b) - \frac{c_2}{T\lambda_b} - \log(\lambda_g^{-5} R_g) + \frac{c_2}{T\lambda_g},
\end{aligned}$$

for the differences between the red and green and the blue and green channels, respectively. Writing $R'_i = \lambda_i^{-5} R_i$ and letting $E_i = -\frac{c_2}{\lambda_i}$, one obtains

$$\begin{aligned}
\rho_{rg} &= \log\left(\frac{R'_r}{R'_g}\right) + \frac{1}{T}(E_r - E_g), \\
\rho_{bg} &= \log\left(\frac{R'_b}{R'_g}\right) + \frac{1}{T}(E_b - E_g).
\end{aligned}$$

The two equations define a line in (ρ_{rg}, ρ_{bg})-color space. The line is parameterized by the temperature T. The constants E_i only depend on the wavelength λ_i. For any given reflectance, one obtains a line with the same orientation because $[E_r - E_g, E_b - E_g]$ is independent of reflectance. Let $\mathbf{e}$ be this vector.

The dependence on the temperature of the black-body radiator can be removed by projecting the data points in a direction orthogonal to the line. The vector $\mathbf{e}_\perp$, which is orthogonal to the vector $\mathbf{e}$, is given by $[E_b - E_g, -(E_r - E_g)]^T$. When the coordinates (ρ_{rg}, ρ_{bg}) are projected onto this line, they become independent of the illuminant:

$$\begin{bmatrix} E_b - E_g \\ -(E_r - E_g) \end{bmatrix} \cdot \begin{bmatrix} \rho_{rg} \\ \rho_{bg} \end{bmatrix} = (E_b - E_g) \log\left(\frac{R'_r}{R'_g}\right) - (E_r - E_g) \log\left(\frac{R'_b}{R'_g}\right).$$

The terms $(E_b - E_g)$ and $(E_r - E_g)$ are constant for a given sensor. The derived reflectance term R'_i is only a function of the reflectance R_i and the wavelength to which the sensor responds.

Instead of choosing one of the sensors and then computing differences, one can also divide each channel by the geometric mean of the three channels and then take the logarithm [Finlayson and Drew 01]. The result is a three-dimensional color space ρ_i:

$$\rho_i = \log(c_i) - \log(c_M),$$

with $c_M = \sqrt[3]{c_r c_g c_b}$. If one carries out the same computations as described above and plots the colors of a single surface in this color space for different black-body illuminants, one again obtains a line parameterized by the temperature T.

The vector $\rho = [\rho_r, \rho_g, \rho_b]$ is orthogonal to the vector $\mathbf{u} = \frac{1}{\sqrt{3}}[1,1,1]^T$. Thus, all points ρ are located on a two-dimensional plane defined by $\mathbf{u}$. It is therefore sufficient to specify any point inside this color space by just two numbers. Finlayson et al. [Finlayson et al. 04] define a standardized coordinate system for the geometric mean chromaticity space using $\{\hat{\chi}_1$ and $\hat{\chi}_2\}$ as the two basis vectors. The two vectors are given by

$$\mathbf{U} = [\hat{\chi}_1, \hat{\chi}_2] = \begin{pmatrix} \sqrt{\frac{2}{3}} & 0 \\ -\sqrt{\frac{1}{6}} & -\sqrt{\frac{1}{2}} \\ -\sqrt{\frac{1}{6}} & \sqrt{\frac{1}{2}} \end{pmatrix}.$$

The χ chromaticity space is therefore defined as $[\chi_1, \chi_2] = \mathbf{U}^T\rho$. The matrix $\mathbf{U}$ simply rotates vectors inside the plane defined by $\mathbf{u}$ to a standard coordinate system.

The invariant direction $\mathbf{e}$ can be obtained by taking a sequence of images of a calibration target, such as a Macbeth color checker, at different times during the day. The colors of a single patch on the color checker will approximately line up along the invariant direction. We can do a covariance analysis on the data and find the direction of the largest spread. This is the invariant direction $\mathbf{e}$. It is also possible to compute an intrinsic image given a single image from an uncalibrated camera [Finlayson et al. 04]. This can be done by choosing the projective direction among all possible directions in which the entropy of the projected points is minimal.

Invariant data points g are obtained by projecting the original data points χ onto the vector $\mathbf{e}_\perp$, which is perpendicular to the vector $\mathbf{e}$:

$$g = \chi \cdot \mathbf{e}_\perp = \chi_1 \cos\theta + \chi_2 \sin\theta,$$

where θ denotes the direction of the vector $\mathbf{e}_\perp$. Now each color is described by a one-dimensional scalar. In order to display the data, it can be transformed to the range $[0,1]$. Figure 9(b)–(d) shows intrinsic images computed for a sample image with different orientations of the vector $\mathbf{e}_\perp$.

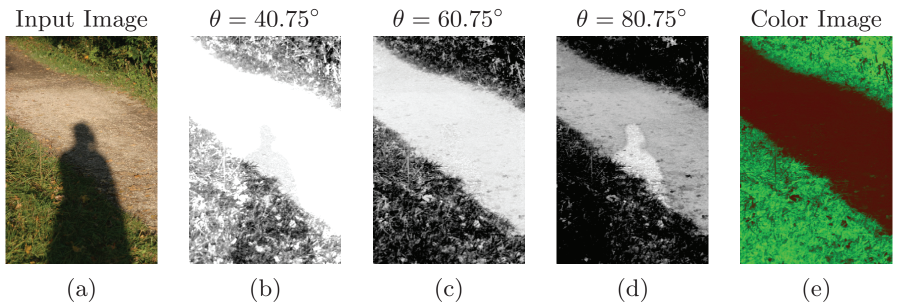

Figure 9. (a) Input image. (b) Intrinsic image that was computed by projecting the data onto the vector $\mathbf{e}_\perp$ with $\theta = 40.75°$. (c) Intrinsic image with $\theta = 60.75$. Notice how the shadow (which is caused by indirect illumination inside the shadow) disappears in the intrinsic image. (d) Intrinsic image with $\theta = 80.75$. (e) Color image computed from the intrinsic image.

5. A Pixel Shader for the Computation of Intrinsic Images and Shadow Removal

Intrinsic images that only depend on the reflectance of an object point are automatically free from shadows. This can clearly be seen in Figure 9(c), where the computed intrinsic image is free from shadows. Such an image can be used for object segmentation if one wants to extract an object but not the corresponding shadow.

It may be possible to obtain a full-color image in some cases from the intrinsic image. The ability to obtain a correct color image from the intrinsic image depends on the image content. Note that once we project the data points onto $\mathbf{e}_\perp$, reflectances that differ by a multiplier $[e^{\frac{s}{\lambda_r}}, e^{\frac{s}{\lambda_g}}, e^{\frac{s}{\lambda_b}}]^T$ with $s \in \mathbb{R}$ can no longer be distinguished. All colors that are located in the direction $\mathbf{e}$ cannot be distinguished. Some information is invariably lost.

Provided that the original image contains only a specific subset of all colors, a full-color image can still be obtained. One can either restore the reds, greens, browns, and yellows, or one can restore the blues, cyans, and magentas [Ebner 07]. This can be done by shifting the colors in χ chromaticity space along the direction $\mathbf{e}$ by a small amount as suggested by Drew et al. [Drew et al. 03]. The shift can be performed in the direction of either the negative or positive half-space created by the vector $\mathbf{e}_\perp$.

Figure 10 shows a pixel shader that computes the intrinsic images shown in Figure 9. The exact value that has to be used for the parameter `theta` depends on the camera model used. For the examples shown here, a Canon 10D was used. The optimal value for the parameter `theta` was determined to be 60.75° using a color Mondrian. Figure 9(e) show the results obtained when

Pixel Shader 4 for Intrinsic Images (GLSL code)

```
float offset=0.5;           // color offset
float theta=radians(60.75); // camera dependent angle
vec2 ePerp=vec2(cos(angle),sin(angle));
vec3 color=texture2D(textureColorSampler,
                     gl_TexCoord[0].st,0.0).rgb;
color=pow(color,2.2);
color+=vec3(0.001,0.001,0.001);
float gm=pow(color[0]*color[1]*color[2],1.0/3.0);
color=log(color/gm);
vec2 chi=vec2(dot(color,vec3(0.81650,-0.40825,-0.40825)),
              dot(color,vec3(0.0,-0.70711,0.70711)));
float s=dot(chi,ePerp); // projection
chi=offset*vec2(sin(angle),-cos(angle))+s*ePerp;
color=vec3(0.8165*chi[0],dot(chi,vec2(-0.40825,-0.70711)),
                         dot(chi,vec2(-0.40825,+0.70711)));
// insert shading code here
gl_FragColor=vec4(s+1);       // Figure 9(c)
gl_FragColor=vec4(color,1.0); // Figure 9(e)
```

Figure 10. Pixel shader 4 (intrinsic images). The color is first transformed to χ chromaticity space. All data points are projected onto the vector ePerp= $\mathbf{e}_\perp$.

a color image is computed by applying a shift of $\frac{1}{2}\mathbf{e}$ and then transforming the data back to the RGB color space. The resulting image is free from shadows.

References

[Buchsbaum 80] G. Buchsbaum. “A Spatial Processor Model for Object Color Perception.” *Journal of the Franklin Institute* 310:1 (1980), 337–350.

[Drew et al. 03] Mark S. Drew, Graham D. Finlayson, and Steven D. Hordley. “Recovery of Chromaticity Image Free from Shadows via Illumination Invariance.” In *ICCV’03 Workshop on Color and Photometric Methods in Computer Vision, Nice, France*, pp. 32–39. Los Alamitos, CA: IEEE Press, 2003.

[Ebner et al. 07] Marc Ebner, German Tischler, and Jürgen Albert. “Integrating Color Constancy into JPEG2000.” *IEEE Transactions on Image Processing* 16:11 (2007), 2697–2706.

[Ebner 04a] Marc Ebner. “Color Constancy Using Local Color Shifts.” In *Proceedings of the 8th European Conference on Computer Vision, Part III,Prague,* *Czech Republic, May, 2004*, edited by Tomáš Pajdla and Jiří Matas, pp. 276–287. Berlin: Springer-Verlag, 2004.

[Ebner 04b] Marc Ebner. "A Parallel Algorithm for Color Constancy." *Journal of Parallel and Distributed Computing* 64:1 (2004), 79–88.

[Ebner 07] Marc Ebner. *Color Constancy.* Chichester, UK: John Wiley & Sons, 2007.

[Ebner 08] Marc Ebner. "Color Constancy Based on Local Space Average Color." *Machine Vision and Applications Journal*, accepted, 2008.

[Finlayson and Drew 01] Graham D. Finlayson and Mark S. Drew. "4-Sensor Camera Calibration for Image Representation Invariant to Shading, Shadows, Lighting, and Specularities." In *Proceedings of the 8th IEEE International Conference on Computer Vision, Volume 2, Vancouver, Canada, July 9-12, 2001*, pp. 473–480. Los Alamitos, CA: IEEE Press, 2001.

[Finlayson and Hordley 01] Graham D. Finlayson and Steven D. Hordley. "Color Constancy at a Pixel." *Journal of the Optical Society of America A* 18:2 (2001), 253–264.

[Finlayson et al. 04] Graham D. Finlayson, Mark S. Drew, and Cheng Lu. "Intrinsic Images by Entropy Minimization." In *Proceedings of the 8th European Conference on Computer Vision, Part III,Prague, Czech Republic, May, 2004*, edited by Tomáš Pajdla and Jiří Matas, pp. 582–595. Berlin: Springer-Verlag, 2004.

[Gershon et al. 87] Ron Gershon, Allan D. Jepson, and John K. Tsotsos. "From [R,G,B] to Surface Reflectance: Computing Color Constant Descriptors in Images." In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Milan, Italy*, 2, edited by John P. McDermott, 2, pp. 755–758. San Francisco, CA: Morgan Kaufmann Publishers, 1987.

[Geusebroek et al. 01] Jan-Mark Geusebroek, Rein van den Boomgaard, Arnold W. M. Smeulders, and Hugo Geerts. "Color Invariance." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23:12 (2001), 1338–1350.

[Horn 86] Berthold Klaus Paul Horn. *Robot Vision.* Cambridge, MA: The MIT Press, 1986.

[Jähne 02] Bernd Jähne. *Digitale Bildverarbeitung*, Fifth edition. Berlin: Springer-Verlag, 2002.

[Poynton 03] Charles Poynton. *Digital Video and HDTV. Algorithms and Interfaces.* San Francisco, CA: Morgan Kaufmann Publishers, 2003.

[Tappen et al. 02] Marshall F. Tappen, William T. Freeman, and Edward H. Adelson. "Recovering Intrinsic Images from a Single Image." Technical Report AI Memo 2002-015, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 2002.

[Wyszecki and Stiles 00] Günther Wyszecki and W. S. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae*, Second edition. New York: John Wiley & Sons, Inc., 2000.

[Zeki 99] Semir Zeki. *Inner Vision: An Exploration of Art and the Brain*. Oxford, UK: Oxford University Press, 1999.

Web Information:

http://jgt.akpeters.com/papers/Ebner08/

Marc Ebner, Eberhard Karls Universität Tübingen, Wilhelm-Schickard-Institut für Informatik, Abt. Rechnerarchitektur, Sand 1, 72076 Tübingen, Germany
(marc.ebner@wsii.uni-tuebingen.de)

Received September 27, 2007; accepted in revised form January 15, 2009.