Available online at www.sciencedirect.com
ScienceDirect
J. Vis. Commun. Image R. 18 (2007) 387–396
JOURNAL OF VISUAL Communication & IMAGE Representation
www.elsevier.com/locate/jvci

Painting in High Dynamic Range

M. Colbert [a,*,1], E. Reinhard [b,a], C.E. Hughes [a,2]

[a] School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816, USA
[b] Department of Computer Science, University of Bristol, Merchant Venturers Building, Woodland Road, Bristol BS8 1UB, United Kingdom

Received 15 November 2006; accepted 14 March 2007
Available online 10 April 2007

Abstract

We present a novel approach that allows users to intuitively and interactively manipulate High Dynamic Range (HDR) images using commonly available Low Dynamic Range (LDR) displays. This solves the problem of how to draw with contrasts that are much larger than the monitor can display. Whereas commercial HDR-enabled drawing programs manipulate tone mapped representations of HDR images, we provide an intuitive brush interface that supports interaction with the unmapped HDR imagery. Our approach introduces two new brush constructs to a typical virtual painting interface, such as Adobe Photoshop. First, we present a brush that locally adjusts the display of the HDR image to a dynamic range specified within a real-time, interactive, local histogram of the region around the cursor. This affords precise, quantitative control of the HDR contrast values produced by the brush. Second, we demonstrate a brush that uses the perception of glare as the underlying basis for determining the contrasts painted onto the HDR image, giving artistic control over the HDR contrasts. By maintaining an HDR image, the result is available for further manipulation and processing by algorithms, such as those used in image-based rendering, for which an LDR representation is inadequate. Finally, we use the Graphics Processing Unit to provide real-time visual feedback for the effects of each image manipulation.

1. Introduction

Conceptually, High Dynamic Range (HDR) imaging can be viewed as using floating point numbers to represent radiance values, providing a wider gamut of color compared with the standard 8-bit, Low Dynamic Range (LDR), counterparts [21]. High dynamic range imaging affords fascinating new opportunities in imaging. For the first time, it is possible to capture, manipulate, and archive absolute radiance values, rather than quantized and clamped relative values. Imaging may therefore transform itself into a quantitative discipline, which has so far not been possible.

Although the conceptual difference between conventional imaging and high dynamic range imaging is straightforward, the entire imaging pipeline requires a redesign. Techniques have emerged to capture HDR images using multiple exposures [5,17,19] or with specially designed hardware [1]. File formats and encoding schemes have emerged to enable storage of HDR data [15,18], and a significant amount of research has gone into preparing HDR images for display on conventional display devices [5,20,21]. Applications that make direct use of HDR data are also emerging, including image-based lighting [4], and image-based material editing [13].

Although most image editing software can now read and write HDR images, drawing with unquantized radiance values presents a significant problem that has yet to be addressed. The reason that HDR images are difficult to manipulate stems from the fact that typical display devices cannot display the full range of values found in HDR images. This means that the radiance values that are to be drawn cannot be displayed directly, but have to be mapped to the display range first.

* Corresponding author. Fax: +1 407 823 5419.
E-mail address: colbert@cs.ucf.edu (M. Colbert).
[1] Partially supported by the UCF I^2 Lab Fellowship Program & the Media Convergence Lab.
[2] Partially supported by the National Science Foundation (SES0527675) and the Army Research Institute VIRTE Program.

1047-3203/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jvcir.2007.03.002

The implications are that picking a color cannot proceed in the normal way. Similarly, the visualization of images using tone reproduction operators leads to a cumbersome and ineffective way to draw HDR images. The latter means that the color selected for drawing is not necessarily the same as the color that is reproduced on the monitor, making drawing an unintuitive process.

In this paper, we present a solution to both problems (Fig. 1). Image regions around the cursor are visualized in an intuitive manner, and picking colors is split into a two stage process. First, the user selects hue and saturation values in the same way that all color pickers allow hue and saturation to be selected. The luminance value, however, is selected with the aid of a histogram which is computed over a region around the cursor. By clicking on a position within the histogram, the user is able to select luminance values that are related to the image in a quantitative and meaningful way.

As not all image editing applications are quantitative in nature, we also present an artistically motivated approach to drawing with high dynamic range imagery. This technique is aimed to be more intuitive for artists, while still overcoming the limitations of tone reproduction, which is a necessary step to display the results.

With these two very different approaches, one quantitative and one artistic, we are for the first time able to give users of drawing programs the ability to truly interact with high dynamic range images. Our technique uses the Graphics Processing Unit (GPU) to perform a painting operation at an average of 1300 Frames Per Second (FPS) on a 512×768 sized image rendered with an nVidia GeForce 7900.

The following sections survey related work in HDR image editing (Section 2) and explain the tone reproduction method we use for HDR image display (Section 3). We present our two approaches for solving the HDR editing problem (Section 4) as well as our implementation (Section 5). We conclude with a discussion of our algorithm's performance (Section 6) as well as possible applications for painting in high dynamic range (Section 7).

2. Background

To our knowledge, there does not exist previously published research on interfaces for HDR editing and manipulation. However, several commercial packages, such as Adobe Photoshop, HDR Shop, Idruna Photogenics, and Artizen HDR, provide tools for directly painting onto an HDR image.

Adobe Photoshop provides a limited ability to edit HDR images, allowing the user to apply a subset of the available filters in the software package. The interface also has a set of proprietary tone-mappers, which allow the user to map an HDR image to a 16-bit fixed precision image for brush-based manipulation.

HDR Shop is a tool that allows the user to manipulate an HDR image by specifying a linear compression for the image, exporting the compressed LDR image to an arbitrary LDR image editor, editing the image, and recombining the modified image into the original HDR image. However, the tool does not provide an integrated work flow or a real-time feedback mechanism for the painting operation.

Idruna Photogenics provides a technique to directly manipulate the HDR values with a brush as well as a means to choose an HDR color value. The interface linearly scales an HDR image to compress the luminance range for an LDR display. Independently, the user can choose a color for painting from an LDR color picker. However, if

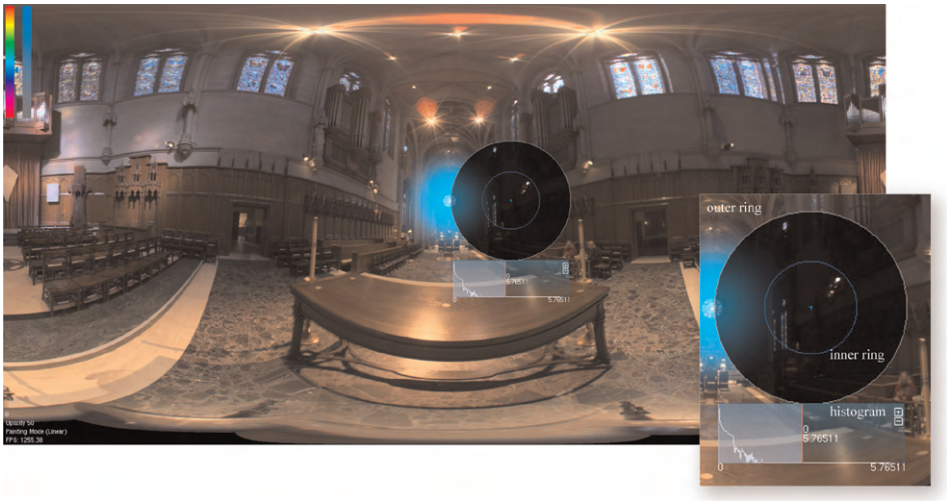

Fig. 1. Quantitative HDR painting interface. In the upper left, a hue and saturation selector is displayed and clearly visible when rendered in color; and in the center and inset, a brush interface is provided that moves with the cursor. In the histogram, the entire luminance domain is visualized and the user selects the luminance associated with the selected hue and saturation values. The highlighted area of the histogram represents the visualized luminance range in which the region within the outer ring is linearly compressed and displayed. The inner ring represents the region that will be painted by the brush. Background image courtesy of Paul Debevec.

users wish to operate outside the displayable color gamut, they must use a slider bar associated with each color channel to select an HDR color. The selected color appears either over-saturated or black, if it is out of the displayable range of the monitor, but will appear correct if applied to a proper linearly-scaled LDR mapping.

Artizen HDR solves the problem of manual linear scaling by providing a variety of automatic, non-linear tone mappers to view HDR images. When using locally adapting operators, such as photographic tone reproduction [20], the interface has a noticeable delay as it re-calculates the HDR values. Moreover, the interface does not provide a means to paint values outside of the displayable color gamut, as the color picker is limited to the low dynamic range of displayable colors.

Our approach provides a brush-based interface and the ability to paint and choose values outside of the displayable luminance range. We offer two solutions, one for quantitative and precise control and another for qualitative and visual-based manipulation. We provide the user with sufficient constructs to control the selection of HDR colors as well as interact with the interface at an average rate of 1300 FPS on a commodity graphics card (nVidia GeForce 7900).

3. Photographic tone mapper

In editing an HDR image, the area under the cursor can be linearly scaled to enable quantitative drawing operations, but the remainder of the image must be tone mapped for display on conventional display devices. While recent research has produced a large variety of tone reproduction operators, any of which can be used to prepare images for display [21], we have chosen the photographic tone reproduction operator [20]. This choice is based on the operator's strong overall performance, as shown in several validation studies [2,6,14,16] and computational image quality metrics [24], and the fact that it can be implemented with the aid of graphics hardware [9] to support real-time applications.

Reinhard et al.'s tone reproduction is inspired by photographic practices [20], and in particular Ansel Adams' zone system. Each pixel in the image is reduced in dynamic range using a measure of local contrast, similar to dodging and burning a film negative. The operator first globally scales the image based upon the log average of the luminance values, $\bar{L}_w$, and a user defined parameter, γ, which effectively sets the exposure of the HDR image:

$$L_m(x,y) = \frac{\gamma}{\bar{L}_w} L_w(x,y). \tag{1}$$

The image is then compressed in dynamic range, pixel by pixel, using a sigmoidal compression scheme:

$$L_d(x,y) = \frac{L_m(x,y)}{1 + L_m(x,y)}. \tag{2}$$

A refinement to this basic approach, is to make this operator locally adaptive. Here, the division is replaced by a value that depends on a spatial neighborhood around each pixel. A good estimator of local adaptation is afforded by the largest region around each pixel that does not overlap with sharp contrasts. To find this region, a scale-space approach is used. By computing the difference of Gaussians (DoG) at a given scale, we can detect if the neighborhood under the filter kernel has a sharp contrast or not. If this value is close to zero, then no sharp contrasts were found. If this is the case, then this procedure is repeated for a larger set of scales until the desired region is found.

The notation used for Gaussian convolution is:

$$L_s^{\text{blur}}(x,y) = L_m(x,y) \otimes R(x,y,\sigma_s). \tag{3}$$

Here, $R(x,y,\sigma_s)$ is a two-dimensional Gaussian defined with a standard deviation of σ_s for the current scale, s. The difference of Gaussians is then defined as the normalized difference between L_s^{blur} and L_{s+1}^{blur}:

$$V_s(x,y) = \frac{L_s^{\text{blur}} - L_{s+1}^{\text{blur}}}{2^{\phi}\gamma/s^2 + L_s^{\text{blur}}}. \tag{4}$$

The normalization factor, $2^{\phi}\gamma/s^2 + L_s^{\text{blur}}$, allows the function V_s to quantify relative differences, instead of absolute luminance differences. Therefore, we search for the maximum scale for which the relative difference of Gaussians is smaller than some ϵ value, which effectively finds the maximum area of similar luminance values for the given pixel:

$$\arg\max_{s(x,y)} |V_{s(x,y)}(x,y)| < \epsilon. \tag{5}$$

The Gaussian blurred pixel, which is computed in this manner, is then a measure of local adaptation. In photographic terms, this is equivalent to dodging and burning. We can use this value directly to steer the sigmoidal compression function:

$$L_d(x,y) = \frac{L_m(x,y)}{1 + L^{\text{blur}}_{s_{\max}(x,y)}(x,y)}. \tag{6}$$

The tone mapped luminance values compress the radiance values by scaling each color channel by the ratio between the displayable luminance, $L_d(x,y)$, and the original luminance, $L(x,y)$. Effectively, the procedure compresses the luminance values, via the sigmoid, while maintaining sharp contrast in highly compressed regions.

4. Algorithm

We approach the problem of choosing and painting with color exceeding the dynamic range of the display from two perspectives: quantitative and qualitative. Quantitatively, the user must be able to operate on an image by defining specific luminance ranges. Being able

to select and draw with absolute luminance values is crucial for drawing programs to become useful tools within a quantitative imaging pipeline. For instance, a lighting designer would need to be able to set the brush to a precise luminance value specified in cd/m^2, while at the same time being able to visualize, in real-time, the overall final result.

On the other hand, artists are likely to be less interested in drawing with absolute luminance values. Here, a good interface is one that the artist is able to intuitively overcome the differences between the image's luminance range and the limited display range. Qualitatively, the user must be able to design in HDR solely based upon visual aesthetics. Most artists are concerned with the final appearance of the image, and are unaware of the exact luminance measures inherent to the brush or the image. An appropriate, intuitive optical phenomenon such as glare may be used to visually model changes in contrasts.

In either case, we use the same underlying framework, whereby the tool allows the user to paint values, while the system transparently and in real-time updates the HDR image as well as re-compresses the image via the photographic tone mapper. The ability to continuously tone-map the image is significant to the user interface because it affords a reasonable impression of the pixel values, and how they interact. Note, however, that we choose a different visualization in the region immediately surrounding the cursor, allowing the appropriate selection of luminance values. Hence, the cursor and the pixels surrounding it become part of the interface for picking colors, which in our opinion is the key factor that allows users to directly interact with HDR data on LDR monitors.

Our work uses a variant of the technique by Goodnight et al. [9], which is a real-time GPU implementation of the photographic tone mapper [20]. The photographic tone reproduction operator has several parameters that may affect the appearance of the tone compressed image [20]. Therefore, we also provide the user with an interactive interface for updating the different parameters. Specifically, since the technique uses local tone reproduction, we provide the user with the ability to visualize different locally adapted scales (Fig. 2).

In the following sections, we will present the quantitative color selection, painting, and qualitative interaction techniques that we have found most useful for editing HDR images.

4.1. Quantitative color selection

Our quantitative interface provides a two-step approach for selecting HDR color values. First, the user chooses the hue H and saturation S of the color by using a conventional hue and saturation color picker, as seen in Figs. 1 and 2. We do not need any special selection mechanism for these color attributes, because hue and saturation in HDR images do not span a larger range than in LDR images.

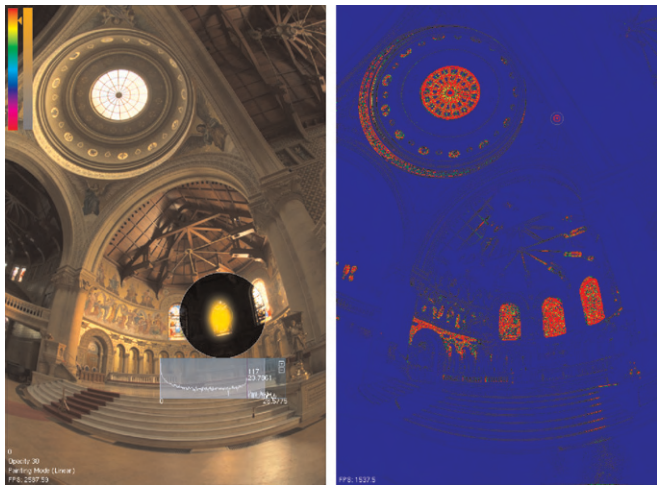

Fig. 2. On the left, the memorial church is shown, modified via the quantitative controls. The luminance value for the color is selected within the histogram and is denoted by the bar. The peak in the histogram around the bar results from the painted region being modified by the selected luminance value. The region around the cursor also linearly compresses the luminance values between 28.6 and 0 cd/m^2 to visualize the dynamic range selected in the highlighted region of the histogram. On the right is an example of the interactive scaling visualization within our interface, where the different shades represent different scales.

In conventional color pickers, the user is able to select either a value for the 'lightness', 'brightness', 'value' or 'intensity' axis. The resulting triplet, *HSL*, *HSB*, *HSV*, or *HSI*, can then be transformed into an RGB triplet for drawing using a conventional transform [7,8,11].

However, in our case, we wish to select a luminance value that is related to the image in some meaningful way. We have found that a useful visualization is the histogram, computed over a neighborhood of pixels around the cursor. Such a histogram shows the distribution of luminance values for a region of interest, and allows a quick assessment of the program material being visualized. By using the location on the horizontal axis of the histogram as a measure of luminance value, the desired luminance value L_{abs} can be selected without the need to resort to trial and error.

The triplet of values now at our disposal consists of hue, saturation, and luminance. The first two values are relative, with hue specified as an angle between 0° and 360°, and saturation defined as a fraction between 0 and 1. The luminance value is an absolute value, which requires a redesigned transform between HSL (for hue, saturation, and luminance) and the RGB color space that is used for drawing.

This transform may be accomplished by introducing a constant value for lightness, which we arbitrarily set to 0.5. We then convert the resulting HSL (hue, saturation, lightness) triplet to RGB, which now encodes relative values. Assuming that we know the primaries of the RGB color space of our choosing, we can com-

pute the relative luminance of this RGB triplet using a weighted average of RGB. Alternatively, if the primaries of this color space are not known, we can approximate the luminance of this triplet by making the assumption that the image is given in the sRGB color space [12]. The relative luminance L_{rel} is then computed with:

$$L_{\text{rel}} = 0.2126R + 0.7152G + 0.0722B \tag{7}$$

The absolute RGB values are then computed by scaling the relative RGB values using:

$$R' = R\frac{L_{\text{abs}}}{L_{\text{rel}}} \tag{8}$$

$$G' = G\frac{L_{\text{abs}}}{L_{\text{rel}}} \tag{9}$$

$$B' = B\frac{L_{\text{abs}}}{L_{\text{rel}}} \tag{10}$$

The histogram approach provides a powerful and intuitive tool for HDR image editing because it is not limited by the dynamic range of the display. The local luminance information provides a good measure for the different radiance values already present within the region, so the user can naturally choose luminance values with respect to other existing luminance values. Additionally, the linearly scaled display presents a visual guide for understanding the displayed luminance values.

Algorithm 1 Linear painting algorithm

```
1: for all p in pixels do
2:     α = f(b_pos, p_pos)          ▷ Compute the brush's opacity
3:     R_p = R' · α + R_p · (1 − α)  ▷ Blend the values
4:     G_p = G' · α + G_p · (1 − α)
5:     B_p = B' · α + B_p · (1 − α)
6: end for
```

4.2. Painting

For painting, we perform linear interpolation on all pixels underneath the brush between the chosen HDR color, i.e. the triplet (R', G', B'), and the source image using the opacity defined by the brush's falloff function. Here, a falloff function outputs a fraction between 0 and 1 representing opacity of the brush with respect to the brush's position, $\mathbf{b}_{\text{pos}}$, and the pixel's position, $\mathbf{p}_{\text{pos}}$. We detail this operation in Algorithm 1, where f is our brush's falloff function, and (R_p, G_p, B_p) is the pixel's color triplet. In our case, f is the Gaussian function, $\exp(-\|\mathbf{b}_{\text{pos}} - \mathbf{p}_{\text{pos}}\|^2/2r^2)$, where r is the radius of the brush. However, we leave f as a generic function, since any arbitrary falloff function will operate in a fashion similar to our Gaussian function.

4.3. Artistic interaction

For qualitative, artistic controls, we look at the visual cue of glare to define the mapping of an LDR color to an HDR color. Glare is the scattering of light in the cornea, lens, and vitreous humor that produces the "bloom" effect we see around relatively bright light sources [22,23]. The effect is commonly seen in nature as well as photography, and therefore affords an intuitive visual representation; albeit the user may not understand the underlying reasons for the phenomenon.

Our interface provides a similar two-step process as with the quantitative process. First, the user selects an LDR color value using a standard LDR color picker, such as an *HSL* color picker. Next, the user specifies the amount of glare via a double-ringed brush, where r_i represents the radius of the inner ring, or the region that will be painted, and r_o represents the radius of the outer ring, or the region that will be affected by glare. Since the artist can create qualitative versions of glare in LDR painting interfaces, commonly by performing some blurring operation on light color values, we assume the introduction of a double-ringed construct will not impede their creation process. The glare effect is only introducing a similar blur-like effect around the painted region, as seen in Fig. 3.

Vos [23] showed that appearance of glare can be modeled by the affine combination of three functions, whose domain is the viewing angle, θ, between the light source and the point on the glare effect. The first function is a Gaussian generating spikes approximately 2–4° from the light source. The result of the spikes is a halo effect surrounding the light source. The remaining two components represent an intensity falloff, with respect to the glare effect, modeled via a θ^{-2} falloff function and a θ^{-3} falloff function. However, the combination can be roughly approximated by the θ^{-2} function. In our case, we do not have the original geometry or viewing position associated with a picture, and thus we also approximate the viewing angle, θ, as the distance between pixels. Therefore, if the maximum area affected by a glare effect is known, which in our case is defined by the user

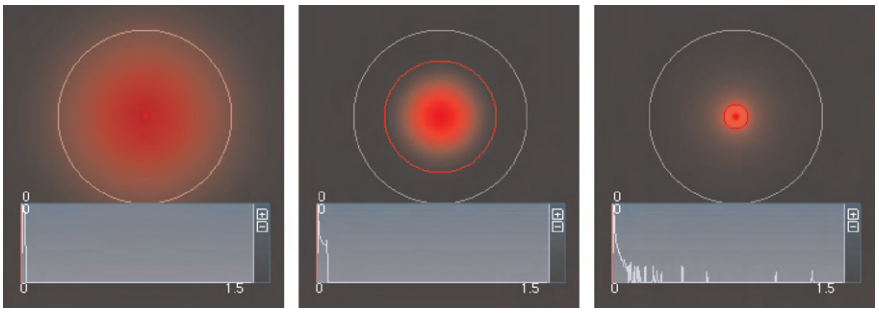

Fig. 3. Three different inner radii and their resulting glare effects. As the inner radius decreases and the glare effect increases, the dynamic range of the luminance expands as seen in the respective histograms.

parameter r_o, the luminance value necessary to generate the glare effect can be calculated.

We start by defining the scale as the squared ratio of the ring radii, r_o and r_i.

$$L_s = \frac{r_o^2}{r_i^2} \tag{11}$$

Conceptually, if the radius of the outer ring doubles, it would take approximately four times as much intensity of the painted region to propagate the glare effect out to the same distance, due to the inverse squared falloff. Therefore, the squared relationship effectively models the growth of the glare effect.

We also scale the HDR color by the ratio $\bar{L}_w/\gamma$ as a means to adjust for the exposure setting of the photographic tone mapper. For images with a very low log average luminance, i.e. when $\bar{L}_w$ is extremely small, we use a δ value as a minimum value for the scaling, allowing the brush to operate even if starting from a blank image. In this way, we produce the HDR color value from the LDR RGB triplet:

$$R' = L_s \cdot \max\left(\frac{\bar{L}_w}{\gamma}, \delta\right) R \tag{12}$$

$$G' = L_s \cdot \max\left(\frac{\bar{L}_w}{\gamma}, \delta\right) G \tag{13}$$

$$B' = L_s \cdot \max\left(\frac{\bar{L}_w}{\gamma}, \delta\right) B \tag{14}$$

When painting, we update all pixels underneath the brush via a linear interpolation between the brush falloff and glare effect functions with the original source image. Specifically, we choose the maximum of the two values, since the functions represent the opacity of the inserted color. However, since we are using visual cues, the linear scaling from Eq. (11) changes the appearance of the brush's falloff behavior. In other words, the scaling will cause the painted region to visually appear larger than the original, unscaled version of the same function. In our case, we use a Gaussian falloff function for the brush and we adjust the standard deviation variable, σ. We calculate the necessary σ parameter knowing that we want the scaled Gaussian function at point r_i to be equivalent to the unscaled Gaussian function with a standard deviation of r_i. We choose a standard deviation of r_i because a majority of the energy of the Gaussian function should be contained within the inner radius of the brush.

$$L_s e^{-\frac{r_i^2}{2\sigma^2}} = e^{-\frac{r_i^2}{2r_i^2}}$$

$$L_s e^{-\frac{r_i^2}{2\sigma^2}} = e^{-\frac{1}{2}}$$

$$\ln L_s - \frac{r_i^2}{2\sigma^2} = -\frac{1}{2}$$

$$\ln L_s + \frac{1}{2} = \frac{r_i^2}{2\sigma^2}$$

$$\sigma^2 = \frac{r_i^2}{2\ln L_s + 1}$$

In calculating the glare effect, we assume the center of the brush is a point light source and has an inverse squared falloff as the distance increases. Typically, when one adds a point light source, the emitted radiance is added to the existing radiance values, due to the additive nature of light. However, we are performing a painting operation where the light is replaced. Therefore, we use a linear interpolation between the source image and glare based upon the falloff, β. Additionally, we adjust the values by the inverse of the falloff opacity values produced by the brush falloff function, where the inverse for an opacity value is one minus the original value. We require the adjustment since we are using a point light source only as an approximation, and we must delineate between the region that is the painted light source and the region that is effected by glare. Here, we also normalize the falloff with respect to the luminance scale since β is a measure of opacity and must map between 0 and 1.

$$\beta = \left(1 - \exp\left(-\frac{\|\mathbf{b}_{\text{pos}} - \mathbf{p}_{\text{pos}}\|^2}{2\sigma^2}\right)\right) \max\left(\frac{L_s}{\|\mathbf{b}_{\text{pos}} - \mathbf{p}_{\text{pos}}\|^2} - \frac{1}{r_i^2}, 0\right) \Big/ L_s \tag{15}$$

The remainder of the details are presented in Algorithm 2.

Algorithm 2 Glare-based painting algorithm

$R' = L_s \cdot \max(\bar{L}_w/\gamma, \delta)\ R$ ▷ Scale from brush size and exposure
$G' = L_s \cdot \max(\bar{L}_w/\gamma, \delta)\ G$
$B' = L_s \cdot \max(\bar{L}_w/\gamma, \delta)\ B$
for all p in pixels **do**
  $d^2 = \|(\mathbf{b}_{\text{pos}} - \mathbf{p}_{\text{pos}})\|^2$
  $\rho = \exp(-d^2/2\sigma^2)$ ▷ Compute the brush falloff
  $\beta = (1-\rho)\max(L_s/d^2 - 1/r_i^2, 0)/L_s$ ▷ Compute the glare falloff
  $\alpha = \max(\beta, \rho)$
  $R_p = R' \cdot \alpha + R_p \cdot (1-\alpha)$ ▷ Blend the values
  $G_p = G' \cdot \alpha + G_p \cdot (1-\alpha)$
  $B_p = B' \cdot \alpha + B_p \cdot (1-\alpha)$
end for

5. Implementation

Our implementation of painting in HDR is a hybrid approach utilizing both the Graphics Processing Unit (GPU) and the Central Processing Unit (CPU), whereby painting, tone reproduction, and histogram creation operate on both processors. Specifically, the GPU is responsible for painting, tone mapping, and displaying both the image and image statistics, while the CPU updates the image's luminance log average, $\bar{L}_w$, as well as local histogram statistics. The following details the implementation of our real-time photographic tone mapper, our GPU-accelerated painting interface, and the corresponding performance of our procedures.

5.1. Real-time photographic tone mapping

Our real-time photographic tone mapper is an extension of the work presented by Goodnight et al. [9]. The procedure consists of a series of pixel shaders applied to a single quadrilateral, whereby the output of each pixel shader has a one-to-one correlation with a pixel in the original image.

First, we perform a pass on the GPU to scale the luminance values, as described in Eq. (1). Unlike the implementation of Goodnight et al., we perform the average luminance calculation on the CPU. In our case, this approach is efficient because we limit the scope of our application to painting operations. A painting operation only requires an update of $4r_e^2$ pixels on the CPU, where r_e is the effected radius of a painting operation and $4r_e^2$ is the corresponding area of a box underneath the region.

In Goodnight et al.'s implementation, the log average luminance, $\bar{L}_w$, value is calculated on the GPU, but to obtain $\bar{L}_w$ on the GPU requires a costly reduction technique. A GPU reduction is multi-pass approach, whereby each pass uses a pixel shader that sums its neighboring values and outputs the sum for the next pass. A subsequent pass, with the same pixel shader, will then read the neighboring values from the previous pass and output the sum for another pass. The process repeats until eventually propagating the final, summed value into one pixel position. This expensive approach is due to the hardware design of the GPU, whereby the processor cannot output global values as it does not have the necessary registers. Therefore, a sequence of concurrent local operations, such as summing the neighboring pixels, must occur to efficiently obtain global values. In the case of reduction, typically a neighborhood of 4 values is chosen, thereby reducing each image dimension by half. In total, this would require approximately $\log_2 n$ passes, where n is the maximum of the width or height of the image. This is more expensive than our approach, since we can perform significantly fewer computations on the CPU.

In each subsequent pass, the pixels whose normalized difference of Gaussians (DoG) is greater than the ϵ threshold, as defined in Eq. (5), are compressed using the sigmoid function, Eq. (6). The remaining pixels are tagged, so they can be compressed when the appropriate scale, $s_{\max}$, is found or the user-defined number of scales is reached.

For efficient calculation of the DoGs on the GPU, Eq. (5), we first must compute the convolution of a Gaussian with a luminance image, Eq. (3). We employ a 4-pass approach that uses the separability of a two-dimensional Gaussian convolution to accelerate the computation. In the first pass, we pack 4 neighboring luminance values along the x-axis of the image into a single RGBA value. This reduces the number of texture look ups by a factor of 4 during the next pass, where the pixels are convolved with the Gaussian kernel. Without using the packing step, kernel sizes greater than 30 become too expensive for the GPU and performance is no longer real-time. In the remaining two passes, we perform the packing and convolution along the y-axis using the output of the x-axis convolution. We then observe that Eq. (5) looks at the DoG with respect to only its scale and the next scale, i.e. L_s^{blur} and L_{s+1}^{blur}. Thus, we can always save the computed L_{s+1}^{blur} image for a subsequent pass. Hence, for an arbitrary scale, s, we have L_s^{blur} from a previous pass and only need to compute L_{s+1}^{blur}. In the special case of the first scale, L_0^{blur} is equivalent to the unfiltered luminance image.

Therefore, we need to perform only one Gaussian convolution for each scale used in the tone reproduction.

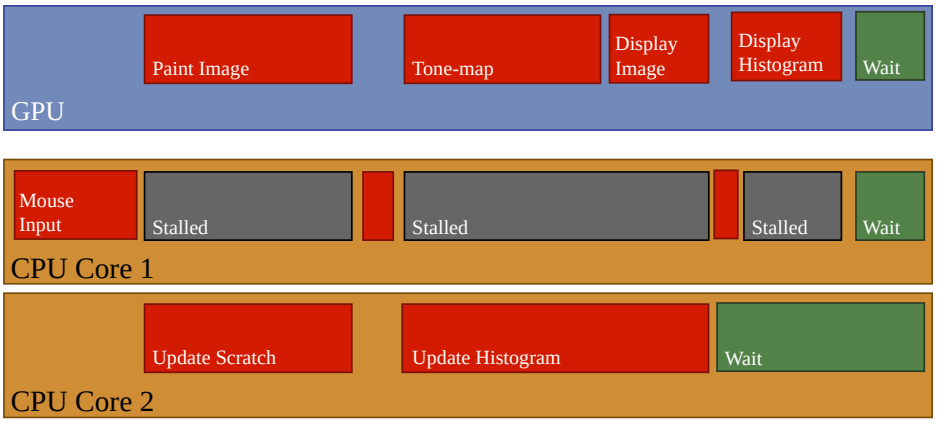

Fig. 4. Process flow of the painting interface. The flow starts from a mouse input on the main thread of the program, executing in the primary core. Once triggered, the image is painted on the GPU while the CPU updates the luminance values. Upon synchronization, the second thread reports the log average of the luminance values and sends the value to the GPU to perform the tone mapping operation. Concurrently, the second thread updates the local histogram information and reports the data to the main thread. Finally, the histogram information is displayed on the GPU and the program waits for the next input from the user.

The one exception is the last scale, since all remaining pixels will be compressed with the already calculated $L_{s_{\max}}^{\text{blur}}$ convolution no matter the result of the DoGs. Thus, no convolution operation is necessary in this pass.

5.2. GPU-accelerated painting

In either the qualitative or artistic approach, the evaluation of the painting operation is performed as described in Algorithms 1 and 2, whereby for all pixels, we perform linear interpolation between an HDR color value and the source image, using either the brush falloff function or the glare effect function. However, as mentioned in the previous section, we limit the affected pixels to the region beneath the brush, whose area is denoted as $4r_e^2$. In the quantitative approach, we define the affected radius, r_e, by setting the scaled brushes's falloff function to a small value, ϵ, and calculating the necessary distance for the function to reach ϵ, giving:

$$r_e = \sqrt{2r_i^2(\ln L_s - \ln \epsilon)}. \tag{16}$$

In the artistic approach, the affected region is defined by the outer ring, and therefore the outer radius r_o is used for the effected radius, r_e.

Additionally, the painting operation is performed on both the CPU and the GPU. On the CPU, we maintain a luminance image within main memory, and perform a painting operation simply by running either the linear or glare painting algorithm over the effected region. On the GPU, we maintain a frame buffer, or a GPU construct for reading and writing image data. We execute a pixel shader, defined by the inner loop of either algorithm, to update the effected pixels.

We perform the operation as such for two reasons: histogram creation and finding the luminance log average, $\bar{L}_w$, of the image. Our hybrid approach reduces the amount of bus traffic and decreases CPU/GPU stalls by better utilizing the multi-core processors available in commodity hardware. Maintaining the luminance information on the CPU also provides a fast way to gather the image and histogram statistics, since we can scan the region surrounding the brush and calculate the corresponding statistics. A pure GPU approach would required the histogram to be built via Occlusion Queries [10]. This approach requires a pass for each bucket in the histogram, thereby making high fidelity histograms operate at less than real-time performance.

Therefore, we use our multi-core processor approach, as depicted in Fig. 4. Upon receiving a mouse input, we instruct the GPU and another CPU core to execute the painting operation. We prevent stalls and allow the driver to utilize more CPU resources by leaving the primary core free with respect to the painting calculation. After completing the painting operation on both the GPU and the CPU, the statistical information computed on the CPU is sent to the GPU, so the GPU can perform the necessary tone mapping operation on the modified image. Concurrently, the CPU updates the histogram information around the brush. Finally, we overlay the tone mapped image with a visual representation of the histogram and wait for further user input.

6. Performance

Our interface performs at approximately 1300 FPS on a 512×768 image, with a brush radius of 15 pixels, running on an nVidia GeForce 7900 using a 2.8 GHz Intel Pentium D processor with 1.0 GB of RAM. We found similar performance with the Grace Cathedral image, where at 1536×768 pixels our algorithm performed at approximately 800 FPS. The speed of our implementation is limited by two factors: image size and brush size.

The image size changes the performance of the tone mapping and the painting operation. During tone repro-

duction, the GPU must perform $5(s_{\text{gmax}} - 1) + 2$ passes, where s_{gmax} is the user-defined maximum number of scales. In the first pass, the luminance must be scaled. The Gaussian convolution (4 passes) and the sigmoid compression and scale evaluation (1 pass) must be performed for each scale, up to the global maximum, $s_{\text{gmax}} - 1$. In the last pass, no convolution operation is necessary, thus only one pass is required for the compression evaluation. Overall, most passes invoke relatively simple pixel shaders and take little time to execute on newer graphics cards. However, there is a fill rate dependence, whereby the pixels are displayed as fast as the GPU can output them. Thus, for each pass on the GPU, all pixels must be redisplayed, making the operation dependent on image size. In the painting operation, the GPU does one additional pass over the image, as discussed in the previous section. However, this operation is trivial in comparison to the $5(s_{\text{gmax}} - 1) + 2$ passes for the tone reproduction.

Brush size is probably the more constrictive of the two performance bottlenecks. When the brush size increases, there does not exist an effect on the GPU, since it performs all operations over all pixels. However, the increased brush size directly affects the amount of time the CPU requires to update the luminance image in main memory, and subsequently the luminance log average, $\bar{L}_w$, as well as local histogram statistics. Typically, painting operations involving over 160,000 pixels cause reduced performance in our implementation.

7. Conclusions and future work

We present an approach for HDR interaction, manipulation, and editing that is novel from both a quantitative and artistic perspective. The use of a double-ringed brush in combination with real-time histogram feedback provides a sufficient basis for building similar tools within commercial painting packages. The importance of our work is that it provides the capability to incorporate image editing software into a fully quantitative imaging pipeline, which was not previously possible. The novelty of our work relates to the design of a user interface that in a meaningful and intuitive manner overcomes the limitations of conventional display devices in the context of image drawing applications.

While we establish our ideas within a painting interface, we believe that the same constructs can be applied to lighting or material design tools. Specifically, material design tools, such as BRDF-Shop [3], provide a painting interface for generating bidirectional reflectance distribution functions (BRDFs). The increased fidelity of HDR editing would endow the interface with more explicit control over the amount of reflectance of a painted highlight.

We demonstrate that our quantitative and artistic HDR editing techniques are capable of running at an average of 1300 FPS on commodity hardware. Even though we present our work in the image editing domain, since it is directly applicable to image-based lighting techniques and image-based material editing, our work lends itself to other applications, such as HDR lighting design and HDR material design, making painting in high dynamic range a powerful tool for future research.

References

[1] Manoj Aggarwal, Narendra Ahuja, Split aperture imaging for high dynamic range, International Journal of Computer Vision 58 (1) (2004) 7–17.
[2] Michael Ashikhmin, Jay Goral, A reality check for tone mapping operators, ACM Transactions on Applied Perception 4 (1) (2007).
[3] Mark Colbert, Sumanta Pattanaik, Jaroslav Křivánek, BRDF-Shop: Creating physically correct bidirectional reflectance distribution functions, IEEE Computer Graphics & Applications 26 (1) (2006) 30–36.
[4] Paul Debevec, Image-based lighting, IEEE Computer Graphics & Applications 22 (2) (2002) 26–34.
[5] Paul Debevec, Jitendra Malik, Recovering high dynamic range radiance maps from photographs, in: SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, 1997, ACM Press/Addison-Wesley Publishing Co., pp. 369–378.
[6] Frédéric Drago, William L. Martens, Karol Myszkowski, Hans-Peter Seidel, Perceptual evaluation of tone mapping operators with regard to similarity and preference. Technical Report MPI-I-2002-4-002, Max Plank Institut für Informatik, 2002.
[7] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, Computer graphics principles and practice, second ed., Addison-Wesley, 1990.
[8] Adrian Ford, Alan Roberts, Colour space conversions, 1998.
[9] Nolan Goodnight, Rui Wang, Cliff Woolley, Greg Humphreys, Interactive time-dependent tone mapping using programmable graphics hardware, in: EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering, Aire-la-Ville, Switzerland, Switzerland, 2003, Eurographics Association, pp. 26–37.
[10] Simon Green, Image processing tricks in OpenGL, Games Developer Conference (2005).
[11] Donald Hearn, M. Pauline Baker, Computer graphics with OpenGL, third ed., Prentice Hall, 2003.
[12] ITU, International Telecommunication Union ITU-R Recommendation BT.709, Basic Parameter Values for the HDTV Standard for the Studio and for International Programma Exchange, Geneva, 1990.
[13] Erum Arif Khan, Erik Reinhard, Roland W. Fleming, Heinrich H. Bolthoff, Image-based material editing, ACM Transactions on Graphics 25 (3) (2006) 654–663.
[14] Jiangtao Kuang, Hiroshi Yamaguchi, Garrett M. Johnson, Mark D. Fairchild, Testing HDR image rendering algorithms, in: Proceedings of IS& T/SID 12th Color Imaging Conference, Scottsdale, 2004, pp. 315–320.
[15] Greg Ward Larson, LogLuv encoding for full-gamut, high dynamic range images, Journal of Graphics Tools 3 (1) (1998) 15–31.
[16] Patrick Ledda, Alan Chalmers, Tom Troscianko, Helge Seetzen, Evaluation of tone mapping operators using a high dynamic range display, ACM Transactions on Graphics 24 (3) (2005) 640–648.
[17] S. Mann, R.W. Picard, Being undigital with digital cameras: extending dynamic range by combining differently exposed pictures, in: IS & T's 48th Annual Conference, Society for Imaging Science and Technology, Washington, DC, 1995, pp. 422–428.
[18] Rafal Mantiuk, Grzegorz Krawczyk, Karol Myszkowski, Hans-Peter Seidel, Perception-motivated high dynamic range video encoding, ACM Transactions on Graphics 23 (3) (2004) 733–741.

[19] Tomoo Mitsunaga, Shree K. Nayar, Radiometric self calibration, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Fort Collins, CO, June 1999, IEEE, pp. 374–380.
[20] Erik Reinhard, Michael Stark, Peter Shirley, James Ferwerda, Photographic tone reproduction for digital images, in: SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, New York, NY, USA, 2002, pp. 267–276.
[21] Erik Reinhard, Greg Ward, Sumanta Pattanaik, Paul Debevec, High dynamic range imaging: acquisition, display, and image-based lighting, Morgan Kaufmann, 2005.
[22] Greg Spencer, Peter Shirley, Kurt Zimmerman, Donald P. Greenberg, Physically-based glare effects for digital images, in: SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, New York, NY, USA, 1995, pp. 325–334.
[23] J. Vos, Disability glare—a state of the art report, C.I.E. Journal 3 (2) (1984) 39–53.
[24] Akiko Yoshida, Rafal Mantiuk, Karol Myszkowski, Hans-Peter Seidel, Analysis of reproducing real-world appearance on displays of varying dynamic range, Computer Graphics Forum 25 (3) (2006) 415–426.