



Rendering High Dynamic Range Images on the Web

By Peter Webb

Modern digital cameras may rival the human eye's perception of color and spatial resolution, but they cannot yet match the eye's ability to manage contrast. An image sensor or display medium's contrast ratio defines the distance between the darkest black and the lightest white that the device records or displays. The eye's contrast ratio of 1:100,000 is roughly 24 times greater than the 1:4096 of a typical digital camera. As a result, even in correctly exposed photographs, shadows are often too dark and highlights too bright, creating a noticeable loss of detail.

High dynamic range (HDR) digital imaging systems increase visual fidelity by integrating contrast ratio with observer-based models of color perception across multiple exposures of a single scene. They use mathematical transformations known as tone-mapping operators to display HDR images on low dynamic range (LDR) devices such as computer monitors.

In this article, we create HDR Explorer, a Web-based experimental test platform that can be used to quickly evaluate how different HDR algorithms render LDR images. We describe how to translate the mathematical expression of an algorithm into a MATLAB® program and then show how to turn that program into a Web application.

HDR Image Processing in MATLAB

The goal of HDR image processing is to use a series of photographs of a scene to produce an image that is as close as possible to what the human eye would see. To turn a photograph into an HDR image, we first extract the radiance information from the photographs and then tone-map it into a single low dynamic range image. Both the reconstruction and tone-mapping algorithms rely on matrix operations, making them easy to implement in MATLAB.

Reconstructing Radiance Information

The low dynamic range of a single digital photograph contains accurate radiance information only for correctly exposed pixels. Changing the exposure captures radiance

Products Used

- MATLAB®
- Image Processing Toolbox™
- MATLAB Builder™ JA

information for different sets of pixels. To accurately analyze a sequence of photographs to extract radiance information for the entire scene, we must take into account the way the camera sensor responds to different levels of light.

Digital cameras map the radiance of a scene into image pixel values via the *camera response function*, a nonlinear function that combines the radiance E falling on sensor location i and the exposure time Δt to produce color value C at pixel i .

$$C_i = f(E_i \Delta t)$$

In the approach developed by Debevec and Malik, the radiance-reconstruction process samples the exposure stack to determine the inverse of the camera response function.

Let g be the natural log of the invertible camera function, Z_{\min} and Z_{\max} the pixel value boundaries, N the number of samples, P the number of photographs, and w a ‘hat’ weighting function that encourages the smoothness of g . To find the range of g we minimize this objective function:

$$O = \sum_i^N \sum_j^P w(C_{ij}) [g(C_{ij}) - \ln E_i - \Delta t_j]^2 + \lambda \sum_{z=Z_{\min}+1}^{Z_{\max}-1} w(z) g''(z)$$

Once the matrices of coefficients have been initialized, two lines of MATLAB code solve this system of equations using Gaussian elimination:

```
x = A \ b;
g = x(1:256);
```

With g fully determined by a lookup table, we reconstruct the radiance map by combining the exposures, using a weighted average of the camera response function of the pixels in each exposure:

$$\ln E_i = \frac{\sum_{j=1}^P \sum_{i=1}^N w(C_{ij}) [g(C_{ij}) - \ln \Delta t_j]}{\sum_{j=1}^P w(C_{ij})}$$

Note that N is the total number of pixels in each image and not the number of sampled pixels.

The MATLAB code for the numerator in equation (3) is simply

```
for j = 1:P
    relRad = ...
    relRad + weightFcn(C(j)) .* ...
    (filmResponse(C(j)) - ...
    logExposure(j));
end
```

Here, a `for` loop replaces the outer summation operator. In MATLAB code, mathematical operators automatically iterate over their matrix inputs, making the inner summation operator implicit.

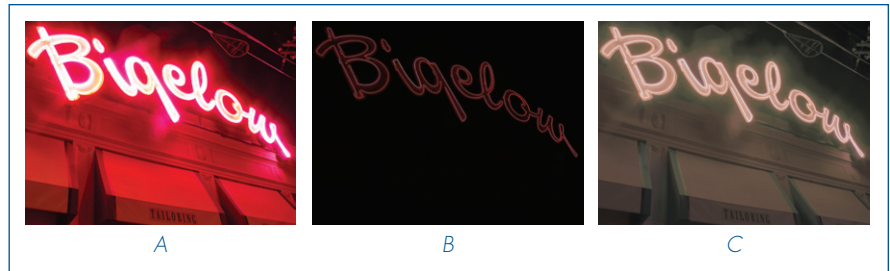


Figure 1. Single LDR exposure (A), linear scaling (B), and logarithmic compression (C).

Translating these mathematical equations into a working algorithm requires approximately 200 lines of MATLAB code.

MATLAB provides a large library of built-in functions to simplify the implementation. For example, loading a JPEG image requires only a single function call, as does extracting the exposure information from the image metadata.

Tone-Mapping Radiance Data

Tone mapping compresses the wide dynamic range of an HDR data set into the much narrower dynamic range of a display device. A good algorithm produces an image that is perceptually similar to the original scene. Because luminance largely governs our perception of contrast, most tone-mapping algorithms operate on the luminance channel of the HDR image. The simplest form of compression is linear scaling, which involves scaling the luminance values in the HDR data set to fit into the luminance range of the output device. However, linear scaling removes a significant amount of detail (Figure 1B).

Because the human eye’s response to light is logarithmic, a logarithmic transformation produces a more recognizable image. Taking the natural log of the color values before normalizing shifts many more of the HDR values into a range that is accessible by an LDR device (Figure 1C). This approach restores much of the detail

but desaturates the colors and reduces contrast. Many tone-mapping algorithms operate in the logarithmic domain, and each addresses the deficiencies of logarithmic compression differently.

Contrast-limited adaptive histogram equalization, introduced by Ward, helps maintain a perceptual equivalence between the HDR and LDR images. Recall that HDR data represents the amount of light in the scene rather than the perceived colors in an image. Ward’s tone-mapping algorithm takes the logarithm of this light, as our eyes would, increases the contrast of the image by histogram equalization to make full use of the available dynamic range, and restores the saturation and contrast lost by logarithmic luminance compression.

The algorithm performs five steps:

1. Compute the logarithm of the HDR image and normalize it to the interval [0 1], thereby compressing the dynamic range of HDR data to an interval that a LDR device can display.
2. Convert the normalized image to the CIE L*a*b* color space, in which luminance can be manipulated directly.
3. Apply contrast-limited adaptive histogram equalization to the luminance channel.
4. Adjust image contrast according to a user-specified parameter.
5. Increase or decrease color saturation as requested by the user.

```

% Compute normalized log of HDR data
hdr = lognormal(hdr);

% Convert from the sRGB color space to the CIE L*a*b* colorspace
Lab = sRGB2Lab(hdr);
Luminance = Lab(:,:,1);

% Increase contrast via histogram equalization
Luminance = adapthisteq(Luminance, 'NumTiles', numtiles);

% Adjust contrast (ignore shadows, clip highlights)
Luminance = imadjust(Luminance, LRemap, [0 1]);

% Restore color saturation by increasing a* and b* channels
Lab(:,:,2) = Lab(:,:,2) * saturation;
Lab(:,:,3) = Lab(:,:,3) * saturation;

```

Code sample 1. Implementing the tone-mapping algorithm.

Using MATLAB and Image Processing Toolbox™, we can implement these steps in just seven lines of code (Code sample 1).

Compare the logarithmic scaling of Figure 1C to the tone-mapped image in Figure 5. Clearly, tone mapping algorithms produce much more lifelike images. An examination of the histogram of each image reveals why. The histogram of the logarithmically scaled image is jagged (Figure 2),

indicating a loss of smooth tonal variation, and compressed along the luminance axis, which wastes available dynamic range.

The histogram of the tone-mapped image (Figure 3) differs from that of the logarithmically scaled image. Both peak in the same areas, but the LDR histogram stretches across the entire x -axis, while the HDR histogram leaves almost 10% of the x -axis unused.

Because the image contains a relatively high number of black pixels, the LDR histogram peaks in the low mid-tone range, but the smooth shape at either end of the range indicates that neither highlights nor shadows have been clipped. The smoothness of the histogram in Figure 3 shows that tone mapping avoided posterization (image blotching created by compression of the color space) and restored fine variations in

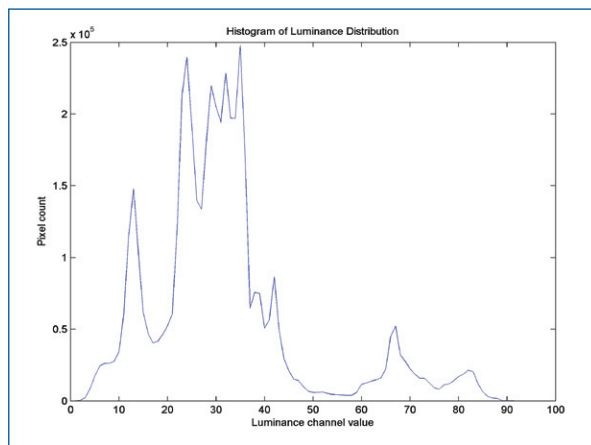


Figure 2. Luminance of log-compressed HDR data, taken from image C in Figure 1.

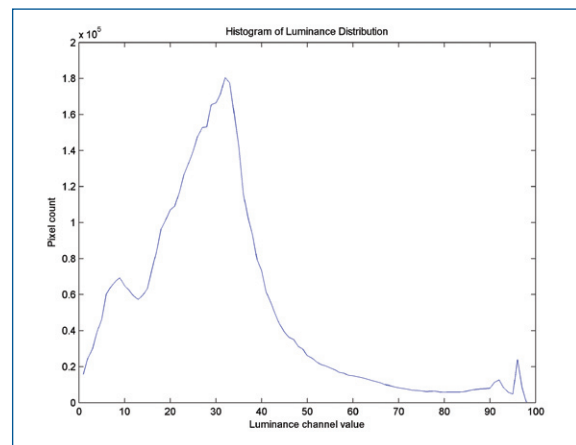


Figure 3. Histogram of tone-mapped image.

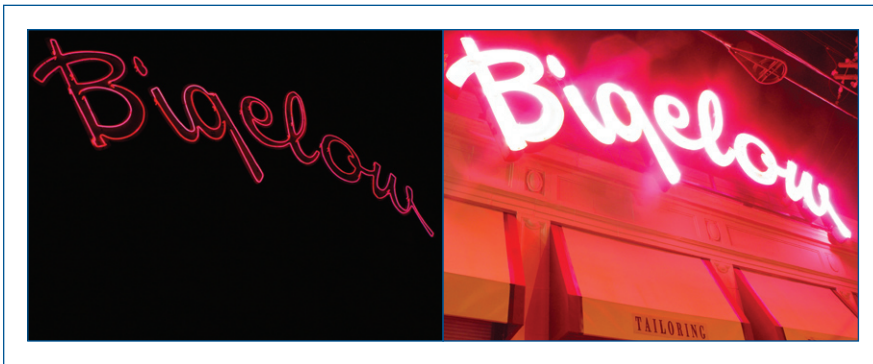


Figure 4. Minimum and maximum LDR exposure frames (from a stack of 10).

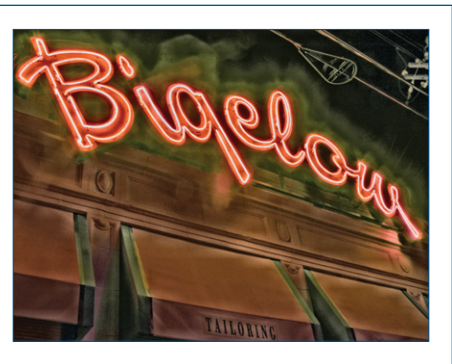


Figure 5. HDR data, tone-mapped to LDR image.

image tone, as is clearly seen by comparing Figure 1C with the image in Figure 5.

Figure 4 shows the boundary images from a stack of 10 LDR images of a high-contrast night-time scene. Combining all 10 images to form an HDR data set and then tone-mapping this data set for a computer monitor produced the image in Figure 5. The tone-mapped image has more detail than either boundary image, and more accurate color fidelity. Further processing could reduce the halo effect, which is caused partly by lens flare and partly by the edge-enhancing properties of histogram equalization.

Deploying HDR Explorer as a Web Application

A MATLAB based Web application requires a browser based GUI, a server-side interface to the MATLAB algorithm, and Java Script to integrate the two. The HDR Explorer application's browser-based GUI consists of a JavaServer Pages™ (JSP) form, the server-side interface of a Java servlet and a Java API generated by MATLAB Builder™ JA. The integration code is a manually written Java Script function. The application provides two functions: LDR image stack preview and HDR data generation and tone mapping.

Designing a Browser GUI

Our HDR Explorer application needs a way to specify input images, GUI controls to adjust transformation parameters, and a name for the tone-mapped LDR image. For simplicity, the application accepts only JPEG images and requires a separate directory for each LDR exposure stack. The GUI consists of several data input fields, two but-

tons, and an image display area (Figure 6). Once the user identifies a directory containing an LDR image stack, the HDR Explorer application displays that stack's median exposure image. The user can adjust the HDR creation and tone-mapping parameters and then create and display the tone mapped LDR image, which is automatically saved as a TIFF file.

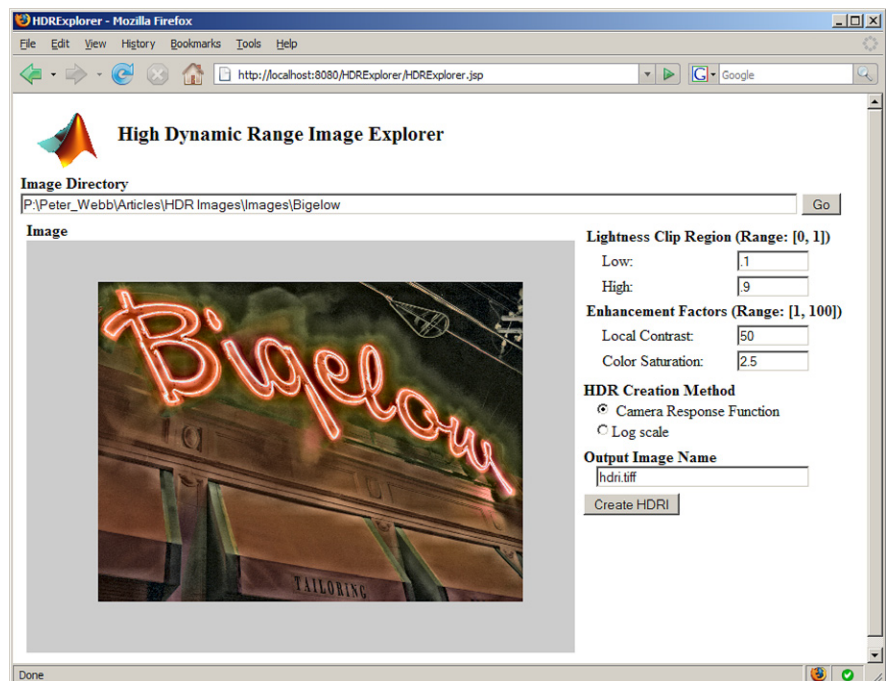


Figure 6. Web interface: a JSP form containing a MATLAB Web figure.

```

if (request.getParameter("SubmitHDR") != null)
{
    MWCellArray flags = new MWCellArray(2,1);
    flags.set(1,"Show");
    flags.set(2,"ReturnWebFigure");

    Object[] result =
        componentObject.jpg2hdr(
            1, // One output -- a structure array
            guiState.getDirectory(), guiState.getOutputName(),
            guiState.getCreateHDRGroup(), flags, 5, 250,
            guiState.getLowLight(), guiState.getHighLight(),
            guiState.getSaturation(), guiState.getContrast(),
            figureHandle);

    UpdateWebFigure(result, "webfig");
}

```

Code sample 2. Script to invoke `jpg2hdr`.

The GUI serves as a front end for experimenting with HDR data creation and tone-mapping algorithms. It is simple enough to be coded by hand, using raw HTML GUI components to create radio buttons or other selectors to choose between HDR image-creation algorithms and a POST method for submitting the JSP form.

Creating a Java API

We create the Java API automatically from the MATLAB HDR algorithm with MATLAB Builder JA. The following command creates a Java component that exports the HDR algorithm via two top-level wrapper functions, `previewImageStack` and `jpg2hdr`:

```

>> mcc -v -W ...
"java.(insert string here)" ...
jpg2hdr.m getWebFigure.m ...
previewImageStack.m

```

MATLAB Builder for JA does all the hard work: determining which functions to include in the component, generating functions that let you call the MATLAB HDR functions from Java, and packaging the

component into a JAR file. These generated functions also manage the redistributable MATLAB runtime and automatically convert data between native MATLAB and Java types. When installed on a server, they become accessible to JSP servlets.

Integrating a Browser GUI with a MATLAB Algorithm

The HDR Explorer GUI sends parameters to `previewImageStack` and `jpg2hdr` and hosts a MATLAB Builder JA Web figure capable of displaying the full range of MATLAB graphics. The JSP form communicates with the MATLAB runtime via the MATLAB Builder JA generated wrapper class `HDRExplorer` and a Java bean.

For example, after creating the wrapper class instance `componentObject` (of type `HDRExplorer`) in `jspInit`, the page embeds a scriptlet to invoke `jpg2hdr` when the user clicks the **Create HDRI** button, which is named `SubmitHDR` in the JSP form's HTML (Code sample 2).

Here, `guiState` is the Java bean used to pass data between client and server, and

the `MWCellArray` type is a Java proxy for the MATLAB cell-array data type. (Cell arrays are N -dimensional generalizations of name-value associative lists like the C++ standard template library type `map`). Since `jpg2hdr` modifies the image displayed in the Web figure, `UpdateWebFigure` notifies the client-side Web figure instance to fetch the serialized image data from the server.

A User-Friendly Platform

The matrix-based and highly mathematical nature of HDR image processing algorithms makes them easy to implement in MATLAB. Their mathematical formulas often map term-by-term into MATLAB expressions or statements. MATLAB Builder JA simplifies the process of turning a MATLAB program into a Web-based application by automatically generating a browser-callable Java API. With MATLAB Builder JA and your favorite HTML or servlet development tools, you have a programmer-friendly platform for delivering mathematically rich applications to any desktop on the Internet. ■

Acknowledgements

I wish to thank Jeff Mather, for introducing me to the techniques used to create and process HDR images and for his implementation of Ward's tone-mapping operator, and Bigelow Cleaners of Newton MA, for permission to use photographs of their sign.

For More Information

- Debevec, P. and Malik, J., "Recovering High Dynamic Range Radiance Maps from Photographs," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, 369-378, ACM SIGGRAPH, 1997
- *MATLAB Builder Java JA*
www.mathworks.com/hdri_java
- Reinhard, E., Ward, G., Pattanaik, S., and Debevec, P., *High Dynamic Range Imaging*, Morgan Kaufmann, Boston, 2006
- Ward, G., Rushmeier, H., and Piatko, C., "A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes," Lawrence Berkeley National Laboratories,
http://www.mathworks.com/hdri_paper

Resources

VISIT

www.mathworks.com

TECHNICAL SUPPORT

www.mathworks.com/support

ONLINE USER COMMUNITY

www.mathworks.com/matlabcentral

DEMOS

www.mathworks.com/demos

TRAINING SERVICES

www.mathworks.com/training

THIRD-PARTY PRODUCTS AND SERVICES

www.mathworks.com/connections

Worldwide CONTACTS

www.mathworks.com/contact

E-MAIL

info@mathworks.com

© 2008 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

91578v00 06/08