

Interactive Java Viewer for HDR-panoramas

H. Dersch

University of Applied Sciences Furtwangen, Germany

December 28, 2003

Abstract

A panorama viewer applet for interactive viewing of high-latitude panoramic images has been developed. The basic functionality of ordinary VR-viewers has been implemented together with manual and optional automatic brightness adjustment. Fast and accurate rendering is achieved by using lookup tables for linearization and dynamic range compression. A highly compressed image format suitable for internet applications is used, which typically requires 25% more space than JPEG-compressed low dynamic range images.

1 Introduction

The limited dynamic range of imaging and display devices is a standard problem in photography. Recording of high contrast scenes with standard analog or digital cameras is limited by two main factors. Sensors in digital cameras and films in analog devices have limited dynamic range, the maximum being approximately 10-12 f-stops for color negative film. Independently, dynamic range is limited by lens flare to typically 7-9 f-stops even in the presence of antireflection coatings. Image file formats are adjusted to these limitations and commonly provide 8 bits/color luminance range.

Particular difficult is the creation and display of panoramic views. Panoramas, especially ones with full spherical coverage ($360^\circ \times 180^\circ$), often include extremely different luminance levels depending on the viewing direction: In landscape scenes very high values up into the sky, almost always including views directly into the sun, whereas shadows may be many orders of magnitude less luminous.

For an exact registration of high contrast scenes we are forced to combine several images taken with varying exposures, which are then assembled to one high-dynamic range image (HDR-image) in a special file format. This method has been extensively studied and described in the literature [1], [2], [3], [4], [5], [6], [7], [8], [9], and automatic versions of panorama capturing devices are currently being offered commercially [10].

Display of HDR-images on limited range devices is accomplished by combining two basic ideas:

1. Compression of contrast by applying a nonlinear transformation to source luminance. Any amount of contrast may be compressed into the limited device space. However, images may appear dull and detail is lost.
2. Spatially varying brightness adjustment. Regions of the panoramic image are brightened or darkened depending on the scene content and average local luminance.

Viewers or tone mapping algorithms for static HDR-images have been proposed and developed in the past [11], [12], [13], [14], [15] and they implement one or both of the methods mentioned above with a multitude of variations. Recently, panoramic viewers for interactive viewing of HDR- panoramas have been suggested [16]. Brightness adjustment is achieved by mixing several in-memory rendered low-latitude versions of the image. These methods provide fast response on suitable graphics hardware.

Today many if not the majority of interactive panoramas are served and viewed over the internet using small footprint Java-based viewers which are unable to access

specific hardware or operating system features directly. This is not just a limitation of resources but usually desirable in order to reach a large audience. Also, slower and less powerful devices like PDAs may not provide these features even in the foreseeable future. A second problem is the high bandwidth required by current HDR-image formats, which makes serving over slow connections difficult. Both problems are addressed in this work.

2 Java Based Panoramic Viewer

The proposed viewer is based on recent work resulting in an interactive viewer capable of displaying low dynamic range panoramic images. This Java viewer (“PTViewer”) uses either equirectangular or cubic source panoramas similar to Quicktime [19] and VRML [20], and has been described elsewhere [21]. The user interactively selects viewing parameters (pan, tilt, zoom level), and the viewer renders the viewport content using well known projection transformations. The conventional program parts are shown in the upper part of figure 1. The lower part shows

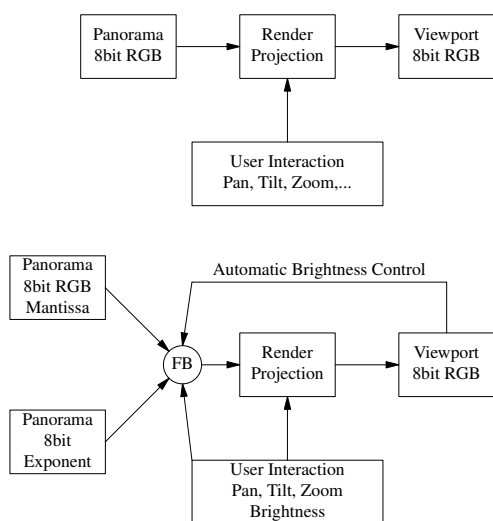


Figure 1: Schematic function of panoramic viewers using 8 bit source color data (upper part) and floating point color data (lower part). The module FP converts floating point to byte data and is described in the appendix.

Table 1: Speed of HDR-viewer as compared to standard viewer. Units are frames per second for a 420×300 pixel viewport.

Interpolator	Source Image type	Speed (fps)
Nearest Neighbor	8bit JPEG	18.9
Nearest Neighbor	HDR	15.6
Bilinear	8bit JPEG	14.7
Bilinear	HDR	9.9

the added functions of the viewer when displaying HDR-panoramas. Internally we use a memory representation of the HDR-image proposed by Ward [22] and used in the Radiance file format [23][24]. In this format each pixels data consist of four bytes, one corresponding to the binary exponent of the luminance of the brightest colour. Three other bytes code the mantissas of the luminance data for the three RGB-colors. The numeric values correspond to linear luminance which means that they scale linearly with physical light intensity. Physically correct brightness adjustment is achieved by scaling these linear data.

Handling the floating point data is done using several lookup tables as described in the appendix. One integer multiplication and shift operation is added for each color channel for each viewport pixel compared to the low dynamic range viewer. The speed penalty is demonstrated in the comparison of table 1 and is generally acceptable. The results in this table were obtained using nearest neighbor and bilinear interpolators. The program executed on a slightly outdated consumer PC-configuration (Win 98, AMD Duron 800MHz, Sun Java 1.4). No hardware graphics acceleration is used in either configuration, and considerable higher speeds are usually achieved with native viewers. The larger relative penalty for bilinear interpolation is due to the requirement of fetching and converting four floating point source data for each destination pixel as compared to just one in the other case.

2.1 Automatic Brightness Control

Several schemes for automatic adjustment of brightness and display of high dynamic range images have been proposed [14],[13], [12]. For the interactive viewer we se-

lected a method similar to the one used in [16]: The viewport center (i.e. center area with dimensions $(width/2) \times (height/2)$) is constantly monitored and the average viewport pixel luminance $s1$ corresponding to no brightness correction is calculated. This value is compared with a preset optimum value $p0$ and the brightness adjusted accordingly. Instead of a mass-spring dynamic used by Cohen et al. [16] a simpler autorange function has been implemented, allowing the user to select the value of $\alpha = (p1 - p0)/(s1 - p0)$. The feedback loop (see figure 1) then adjusts the viewport luminance to $p1$. $\alpha = 0$ corresponds to strict adjustment, which fixes the pixel average of the viewport center region to the predefined optimum value $p0$, whereas $\alpha = 1$ does not change luminance at all.

This action is demonstrated in figure 2.1 which exhibits two viewports generated by PTViewer using the source image of figure 3.

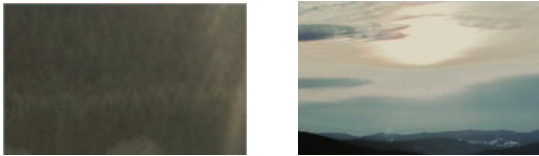


Figure 2: Two partial views generated by PTViewer using the second image of figure 3 in automatic brightness adjustment mode. Autorange is set to 0.3.

Both images are rendered using the autorange parameter α set to 0.3 and the optimum viewport luminance $p0$ set to 86. The first image is much darker, and its center averages to $s1 = 18$ without adjustment. If α were one, this image would be brightened to the optimum value 86. Due to the smaller value of α , it is brightened to just $p1 = 65$. This makes details visible while preserving the sensation of darkness. The same procedure is applied to the bright view into the sun in the second image of figure 2.1. Here the average without adjustment is $s1 = 378$, whereas the actual adjustment is targeted at $p1 = 173$. The speed comparison in table 1 has been obtained using $\alpha = 0.3$.

Finally, manual adjustment is provided by mapping keyboard keys to incrementing and decrementing brightness.

3 File format

A number of file formats has been proposed for storage of digitized high dynamic range images: The Radiance format by Ward [23] described in the last section, the logluv-format by Larson [25] implemented in the current TIFF-library [26], 16 bit variants of standard formats like TIFF [26] or PNG [27], and raw floating point formats which store the full precision of four byte floating point data for each pixel and color. These formats focus on high fidelity but require substantially more space than is usually tolerated for internet based applications. For this work we have used a simple extension of the standard JPEG format which implements the basic idea of the Radiance [23], [24] format. Basically, the RGB mantissas of the Radiance file are stored as standard JPEG image, while the exponents are attached as separate compressed data block.

The JPEG format in its most often used baseline variant [28] divides an image into subimages each 64 pixels large (8×8) and encodes each subimage separately. Breaking up the pixel data into mantissas and exponents interferes with the JPEG compression and leads to additional compression artefacts. This can be avoided by adjusting the mantissas in each 64 pixel subimage to use the same exponent (the largest in this block). That way the standard JPEG compression scheme becomes applicable to the mantissa data. Additionally, the storage requirement for the exponents decreases drastically by a factor of 64, which is further reduced by compressing the data. The disadvantage of this method is the limited dynamic range within each 8×8 pixel block, which is reduced to the usual 8 bit range. For photography based images this can almost always be tolerated, see the remarks about lensflare in the introduction. The overall dynamic range of this format covers the full range of 256 binary exponents, i.e. 256 f-stops or intensity ratios of 10^{77} .

An example of this technique is shown in figure 3. The first image is derived from the low dynamic range version, while the second image is adjusted for use by PTViewer. The compressed exponent attachment amounts to only 2336 bytes compared to 312 kBytes for the JPEG-compressed RGB-mantissas. Nevertheless, the fjpeg file is 66 kBytes larger than the low dynamic range JPEG image created using the same JPEG-quality setting. This is due to the higher contrast of the RGB-mantissas,

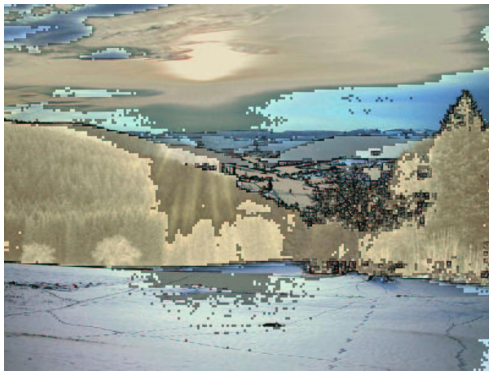


Figure 3: Winter scene photographed directly into the sun. The first image is a standard 8 bit file (JPEG) showing clipping in the sky and black featureless shadows. The second image features the mantissas of a floating point JPEG image. The source has been generated by combining 5 images with different exposure levels. Image size is 1600×1200 pixels. Each square dot in the right image corresponds to one 64 pixel subimage.

where each 64 pixel subimage has at least one pixel with one byte valued above 128.

The file specification for the floating point JPEG format (fjpg) are summarized as follows:

- Standard JPEG image representing the RGB-mantissas. Mantissas are adjusted for each 8×8 pixel subimage to use the same exponent.
- Followed by a block of byte sized binary exponents (ordered left to right, top to bottom) representing the

Table 2: File sizes of image shown in figure 3

Format	Filesize
JPEG	249 kBytes
fjpg	315 kBytes
Radiance	5.42 MBytes
8 bit TIFF	4.91 MBytes
Raw float	29.2 MBytes

largest Radiance exponent in each 64 pixel block. This attachment is additionally compressed by the lossless deflate-method (zlib-library [29]).

Typical file sizes for the image shown in figure 3 are displayed in table 2. The JPEG compression for both the JPEG and fjpg versions were carried out using the open-source JPEG-library[28] with quality set to 80%. TIFF data are packbits-compressed[26], and Raw float data are uncompressed 12 byte/pixel RGB floating point representations.

It should be added that the fjpg format is quite easily generated and decoded using standard programming techniques. The current Java API [30] implements methods for decoding both JPEG and zlib compressed data, and the Java-viewer of this work requires less than 30 lines of program code for this task. This is an important consideration since the applet code has to be transmitted together with the image data.

The Java viewer is demonstrated on this website[21]. A program for conversion from and to the fjpg file format has also been developed and is available from the same site.

4 Conclusion

An interactive Java viewer capable of displaying high dynamic range panoramas has been developed and its function has been demonstrated. The speed penalty for using extended range data is 20-35%. Using an extended JPEG-format the additional space requirement for transmitting the HDR-data becomes small and amounts to typically 25%. It is mainly due to the increased size of the JPEG-compressed RGB-mantissas. The size of the exponent data is negligible and below 1% of the total file size in

our examples. Finally, memory requirement is slightly increased due to the storage of the HDR-exponents. While the low range viewer stores 5 bytes per panorama pixel (3 color channels plus 1 alpha-mask plus 1 hotspot indicator), we now need 6 bytes.

5 Appendix: Brightness Adjustment

A lookup table Lgamma (datatype byte, size lmax) to generate viewport pixel data (0..255) from linear luminance data (0....lmax) is generated at program initialization time. This table is normalized so that linear luminance 1 corresponds to viewport luminance 1. lmax is the smallest linear luminance value leading to saturated viewport pixels (ViewportColor=255). The current implementation uses a gamma curve $ViewportColor = LinearColor^{1/\gamma}$ but any curve depending on the specific graphics hardware could be used. A second lookup table Lexp (datatype integer, size 256) to generate linear luminance data for each of the 256 possible exponents of the HDR data multiplied with the desired brightness adjustment factor a , i.e. $Lexp[n] = a \cdot 2^{n-128}$, is calculated prior to each viewport drawing request. Most of the numbers in this array are outside the integer range and are clamped to INTEGER_MAX or 0. Pixel data are then processed by multiplying Lexp with the HDR mantissa. The result is inserted in Lgamma to generate viewport pixel data. Example: Suppose the renderer determines pixel i of the HDR-image to be used for pixel k in the viewport. The calculations to determine viewport color become

```
int LinearColor = (HDR_mantissa[i]
    * Lexp[HDR_exponent[i]]) >> 8;
if(LinearColor >= lmax)
    Viewport[k] = 255;
else
    Viewport[k] = Lgamma[LinearColor];
```

References

- [1] R.Ginosar and Y.Y.Zeevi. Adaptive sensitivity/intelligent scan image processor. *SPIE Visual Communications and Image Processing*, 1001, November 1988.
- [2] P.J.Burt and R.J.Kolczynski. Enhanced image capture through fusion. In *Proc.of the ICCV*, pages 173–182, 1993.
- [3] B.C.Madden. *Extended Intensity Range Image*. Grasp Lab,UPenn, 1993.
- [4] S.Mann and R.Picard. Being undigital with digital cameras:extending dynamic range by combining differently exposed pictures. In *Proceedings of IS&T,46th annual conference*, pages 422–428, 1995.
- [5] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of SIGGRAPH 97*, pages 369–378, 1997.
- [6] T.Mitsunaga and S.K.Nayar. Radiometric self calibration. In *Proc.of CVPR*, volume 2, pages 374–380, June 1999.
- [7] M.A.Robertson, S.Borman, and R.L.Stevenson. Estimation theoretic approach to dynamic range improvement through multiple exposures. In *Proc.of ICIP*, pages 159–163, 1999.
- [8] Y.Y.Schechner and S.K.Nayar. Generalized mosaicing. In *Proc.of the ICCV*, volume I, pages 17–24, 2001.
- [9] Michael D. Grossberg and Shree K. Nayar. High dynamic range from multiple images: Which exposures to combine? In *Workshop on Color and Photometric Methods in Computer Vision*, 2003.
- [10] <http://www.spheron.com/products/spherocamh-dr/hdri.html>.
- [11] C. Schlick. High dynamic range pixels. *Graphics Gems*, IV:422–429, 1994.
- [12] K.S.N. Pattanai, J.E. Tumblin, H. Yee, and D.P. Greenberg. Time-dependent visual adaptation for realistic image display. In *SIGGRAPH*, pages 47–54, 2000.

- [13] A. Scheel, M. Stamminger, and H.P. Seidel. Tone reproduction for interactive walkthroughs. In *Eleventh Eurographics Workshop on Rendering*, 2000.
- [14] J. Tumblin, J.K. Hodgins, and B.K. Guenter. Two methods for display of high contrast images. *ACM Transactions on Graphics*, 18(1):56–94, January 1999.
- [15] Michael Ashikhmin. A tone mapping algorithm for high contrast images. In *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering*, pages 145–156, 2002.
- [16] Jonathan Cohen, Chris Tchou, Tim Hawkins, and Paul Debevec. Real-time high-dynamic range texture mapping. In *12th Annual Eurographics Workshop on Rendering*, pages 313–320, June 2001.
- [17] Erik Goetze. <http://radio.weblogs.com/0110138/stories/2003/02/06/hdrforpanoramas.html>.
- [18] Aldo Hoeben. <http://fieldofview.nl/rd-adr.php>.
- [19] <http://www.quicktime.com>.
- [20] <http://www.vrml.org>.
- [21] <http://www.fh-furtwangen.de/~dersch>.
- [22] Greg Ward. Real pixels. *Graphics Gems*, II:80–83, 1991.
- [23] Greg J. Ward. The radiance lighting simulation and rendering system. In *SIGGRAPH*, pages 459–472, July 1994.
- [24] <http://radsite.lbl.gov/radiance>.
- [25] G. W. Larson. Logluv encoding for full-gamut, high-dynamic range images. *Journal of Graphics Tools*, 3(1):15–31, 1998.
- [26] <http://www.libtiff.org>.
- [27] <http://www.png.org>.
- [28] <http://www.jpeg.org>.
- [29] <http://www.zlib.org>.
- [30] <http://www.javasoft.com>.