Real-Time High-Dynamic Range Texture Mapping

Jonathan Cohen, Chris Tchou, Tim Hawkins, and Paul Debevec
University of Southern California Institute for Creative Technologies
13274 Fiji Way, Marina del Rey, CA 90292
{jcohen,tchou,timh,debevec}@ict.usc.edu

Abstract. This paper presents a technique for representing and displaying high dynamic-range texture maps (HDRTMs) using current graphics hardware. Dynamic range in real-world environments often far exceeds the range representable in 8-bit per-channel texture maps. The increased realism afforded by a high-dynamic range representation provides improved fidelity and expressiveness for interactive visualization of image-based models. Our technique allows for real-time rendering of scenes with arbitrary dynamic range, limited only by available texture memory.

In our technique, high-dynamic range textures are decomposed into sets of 8-bit textures. These 8-bit textures are dynamically reassembled by the graphics hardware's programmable multitexturing system or using multipass techniques and framebuffer image processing. These operations allow the exposure level of the texture to be adjusted continuously and arbitrarily at the time of rendering, correctly accounting for the gamma curve and dynamic range restrictions of the display device. Further, for any given exposure only two 8-bit textures must be resident in texture memory simultaneously.

We present implementation details of this technique on various 3D graphics hardware architectures. We demonstrate several applications, including high-dynamic range panoramic viewing with simulated auto-exposure, real-time radiance environment mapping, and simulated Fresnel reflection.

1 Introduction

Real-world scenes usually exhibit a range of brightness levels greatly exceeding what can be accurately represented with 8-bit per-channel images. This is typical in outdoor scenes where some surfaces are in sunlight and others are in shadow, in environments with both interior and exterior areas, or in unevenly illuminated interior environments. Previous work has described techniques for acquiring high-dynamic range (HDR) images of real world scenes, and for representing such images efficiently. Other work has described techniques for visualizing high-dynamic range images by compressing the dynamic range into the 8 bits of dynamic range that most display systems will accept.

We propose a technique for visualizing HDR image-based scenes in graphics hardware without compressing the dynamic range. Borrowing from real imaging devices, we add an "exposure level" parameter to the virtual camera model. The exposure level can be dynamically adjusted to map a particular section of the full dynamic range of the scene to the displayable dynamic range of the output device. Our technique is similar in motivation to the dynamic visual adaptation work recently presented by Pattanaik *et al*. [22], but focuses on how such visual adaptation can be accomplished in real time using current graphics hardware. Central to our technique is the use of high-

dynamic range texture maps (HDRTMs) for real-time rendering of HDR image-based environments. Our implementations of HDRTMs use standard 8-bit per-channel texture memory to store 8-bit sections of arbitrary bit-depth textures.

In image-based rendering, texture maps often store the exitant radiance from a surface rather than albedo. Since light can be arbitrarily intense, standard clamped 8-bit textures have limited usefulness for image-based rendering. HDRTMs, on the other hand, allow us to store arbitrarily large radiance values in texture maps. During rendering, these radiance values are scaled according to the camera's exposure level and any light attenuation due to reflection or transmission, and then clamped to the valid range of the display device. Thus HDRTMs are an appropriate technique for visualizing global illumination solutions interactively and properly simulating bright lights reflected in dark surfaces.

Most devices display images with 8 bits per-channel, employing a nonlinear mapping between pixel values and brightness levels. This mapping is typically described by a *gamma curve*, in which the intensity i displayed on the monitor for a pixel with value p is computed as $i = (p/255)^{\gamma}$. In this work, we create texture maps that have a greater number of bits per pixel than the display device, allowing for increased dynamic range. For example, a texture map with 16 bits per pixel allows us to represent 256 times the displayable pixel value range, and with a $\gamma = 2.2$ mapping [1] the maximum representable intensity increases by $256^{2.2}$, or nearly a factor of 200,000. This dynamic range is enough to represent bright outdoor scenes and darker indoor regions in the same texture map, and is therefore adequate for most applications.

2 Related work

An early source of high-dynamic range (HDR) images in computer graphics were the renderings produced by radiosity and global illumination algorithms. As a particular example, Greg Ward's RADIANCE synthetic imaging system [32] outputs each of its renderings in Ward's "Real Pixels" [31] high-dynamic range Red-Green-Blue-Exponent format, representing HDR images in just 32 bits per pixel. Schlick [27] later presented a nonlinear mapping process to encode color HDR in 24 bits per pixel, and Ward has presented two other formats based on separating high-dynamic range luminance from perceptually represented chrominance as the LogLuv extension to the TIFF image format [15]. HDR images of real-world radiance may also be acquired using standard digital cameras as demonstrated in [4]. We use HDR images acquired with this technique in this paper.

Displaying HDR images on low-dynamic range (LDR) devices such as video monitors and has been studied in the context of *tone reproduction* [29]. [10] performed this process through quantization, [16] through histogram adjustment, and [30] through a form of anisotropic diffusion. [21] modeled the effect of visual adaptation in viewing high-dynamic range images, and later work [22] modeled global time-dependent effects. [30] discussed a foveal display program in which the exposure of a high-dynamic range image was adjusted according to user mouse input, but was implemented in software only. [26] presented a technique for performing Ward's tone reproduction algorithm interactively to visualize radiosity solutions.

Texture mapping has been used in scan line renderers, ray tracers, and hardware-accelerated computer graphics applications for over two decades. A survey of applications of texture mapping is presented by Haeberli and Segal in [11]. We take advantage

[1] 2.2 is a typical value for γ on PC monitors.

of the powerful new additions to the texturing pipeline that allow for programmable texture preprocessing such as NVidia's register combiner and texture shader architectures [19].

Of particular relevance to our work is the use of texture mapping for real-time image-based rendering [5, 7, 17, 24]. Most applications of these techniques have been limited to reproducing scenes with low-dynamic range, such as flatly illuminated room interiors or cloudy outdoor environments. This paper provides a way to extend these techniques to environments with dramatic lighting, such as both interior and exterior areas, by allowing high-dynamic range images to be used as texture maps.

[3] used high-dynamic range images of incident illumination to render synthetic objects into real-world scenes. This work used non-interactive global illumination rendering to perform the lighting calculations. Significant work has been done to approximate these full lighting calculations in real-time using graphics hardware. For example, [1] and [12] described how texture prefiltering and standard texture mapping could be used to interactively render convincing approximations to the illuminated objects in [3], while [13, 14, 23] use multi-pass rendering methods to simulate arbitrary surface reflectance properties. This paper presents a potential step towards extending these techniques by allowing high-dynamic texture maps to be rendered with hardware acceleration and used for hardware-based lighting calculations. An alternate implementation by Simon Green uses the new two-channel 16-bit HILO texture format available in the NVidia GeForce3 architecture [8] to perform similar calculations.

3 Representation and rendering of HDRTMs

3.1 General technique

Most graphics hardware allows only 8-bit per-channel color values to be used in texture maps.[2] However, many graphics cards allow texels from multiple textures to be combined during texture fetching or in the framebuffer to produce the final rendered color. In our technique, we use multiple textures to represent high-dynamic range textures. These textures are then combined in hardware to produce a correct 8-bit texture for any exposure setting.

Consider a 16-bit texture that stores values directly proportional to the exitant radiance from a surface. We simulate the appearance of the surface as seen by a linear-response camera with a given exposure value and white point by computing:

$$I(v) = clamp(ev).$$

Here e is a virtual exposure level that may take on any non-negative value, and the *clamp* function simply clamps the result to 8 bits, that is, to a pixel value of 255. To provide for hardware computation of $I(v)$, we can represent the 16-bit texture v as two 8-bit textures, storing the low bits into v_0 and the high bits into v_1. We then have

$$\begin{aligned} I(v) &= clamp(e(v_0 + 256v_1)) \\ &= clamp(ev_0 + 256ev_1) \\ &= clamp(clamp(ev_0) + clamp(256ev_1)) \end{aligned} \tag{1}$$

where the last equality follows from the observation that for $a, b \geq 0$, $clamp(a+b) = clamp(clamp(a) + clamp(b))$. This is demonstrated graphically in Figure 1.

[2] The newest GeForce3 cards from NVidia allow for a two-channel 16 bit per-channel format.

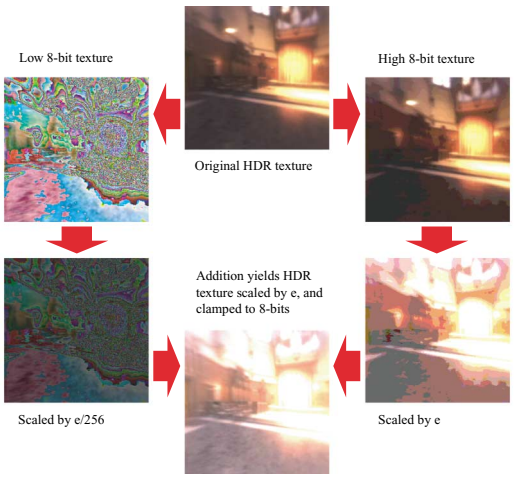

Fig. 1. Overview of HDRTM pipeline.

Many graphics systems have multitexturing that supports texture computations of the form of Equation 1, although all systems we have investigated have required some modifications to accommodate limitations such as restricted ranges for operands (e.g. multiplication by values greater than 1 is not supported).

It should be noted that when $e < 1/256$, any regions that were clamped (saturated) in the 16-bit texture will begin to dim incorrectly. Similarly, when $e > 1$, the low 8 bits of the texture will be amplified and the displayed texture will begin to appear quantized for lack of sufficient low-order bits. This means that restricting e to the range $[256^{-1}, 1]$ allows the 16-bit texture to be viewed without visible artifacts.

For textures with greater than 16 bits, we can modify Equation 1 by adding one term to the sum for each additional 8 bits. However, adding many textures together during rendering may impact performance. By precomputing and storing additional texture maps, we can compute Equation 1 by adding only two 8-bit textures together. We illustrate this with an example. Consider a 24-bit texture value v. From v we compute $x = v/256$, which is representable in 16 bits. We also compute $y = clamp_{(2^{16}-1)}(v)$ which clamps v to the 16-bit maximum. We split the 16-bit values x and y into 8-bit textures x_0, x_1 and y_0, y_1 as above. For a given value of e, we compute $I(v)$ as follows. If $e \in [256^{-1}, 1]$, $I(v) = clamp(clamp(ey_0) + clamp(256ey_1))$, as in Equation 1. If $e \in [256^{-2}, 256^{-1}]$, $I(v) = clamp(clamp(256ex_0) + clamp(256^2 ex_1))$. In general, the number of textures required to store a texture of bit depth $8n$ with $n \geq 2$ is $2n-2$. Note that for an $8n$-bit texture, the useful exposure range is $[256^{1-n}, 1]$.

3.2 Gamma-corrected textures

Most display devices apply a gamma curve to the output intensities as described in Section 1. To produce the correct display values, we must therefore gamma correct an HDRTM by exponentiating the radiance values by $1/\gamma$.

Let $v' = v^{1/\gamma}$ be a gamma-corrected texture value. To apply our technique to gamma-corrected values, we need only observe that gamma-correcting the result of scaling radiance value v by e is equivalent to scaling v' by $e^{1/\gamma}$. If we let $e' = e^{1/\gamma}$, this implies

that

$$I(v)^{1/\gamma} = clamp(v'e') = clamp(clamp(e'v'_0) + clamp(256e'v'_1). \quad (2)$$

Thus, provided the values in the HDRTM are gamma-corrected before they are converted to 8-bit textures, we can correctly render this HDRTM by gamma-correcting the exposure level as well. Note that this extends the effective exposure range of the texture map to $[1/256^{\gamma}, 1]$. The downside is that we can not correctly add two gamma-corrected HDRTMs using the standard addition operator, for example to render diffuse and specular passes as shown in Figure 3e.

3.3 Implementation on SGI

We can implement HDRTMs using the hardware-accelerated pixel transfer pipeline operations on SGI systems. Assuming gamma-corrected 16-bit texture values, we first render the high 8 bits of the texture into the framebuffer, v'_1. We turn off blending, set the pixel transfer function to scale all color channels by $256e'$, and call glCopyPixels to read the framebuffer back onto itself and apply the scale factor. Next, we turn on additive blending, set the current color to be (e',e',e'), set the texture environment mode to modulate the texture values by the current color, and render the least-significant portion of the texture into the framebuffer, v'_0. This directly computes the quantity as in Equation 2. On an Onyx3 Infinite Reality, this runs in full screen mode (1280 x 1024 resolution) at 30 Hertz.

3.4 Implementation on NVidia GeForce2

We have also implemented HDRTMs using the programmable texture combiner system in NVidia's GeForce2 and TNT architectures. This is inherently difficult because these architectures do not directly support multiplying textures by values greater than one. Texture combiners process texture values after they have been fetched from memory before they are written into the framebuffer. The GeForce2 contains two texture combiners, each of which can take 4 inputs (A,B,C,D) interpreted as 8-bit fixed point numbers between 0 and 1 and generate the output $A*B+C*D$. The inputs to the combiners can be either the value 0, the value 1, the current texel color, a constant color, an interpolated color, or the result of a previous texture combiner. Finally, the output of the combiner can be multiplied by 1, 2, or 4, with the result clamped back to the 8-bit range. The details of this operation may be found in [19, 25].

Within this architecture, we can compute $clamp(ex_0 + 16ex_1))$ for any value of e between 0 and 1 as follows:

Exposure Range	Combiner Name	A	B	C	D	Final Multiplier
$e \in [\frac{1}{2}, 1]$	TC0	x_1	1	x_1	1	x4
	TC1	TC0	e	x_0	$e/2$	x2
$e \in [\frac{1}{4}, \frac{1}{2}]$	TC0	x_1	1	x_1	1	x2
	TC1	TC0	$2e$	x_0	$e/2$	x2
$e \in [\frac{1}{8}, \frac{1}{4}]$	TC0	x_1	1	x_1	1	x1
	TC1	TC0	$4e$	x_0	$e/2$	x2
$e \in [0, \frac{1}{8}]$	TC0	x_1	1	0	0	x1
	TC1	TC0	$8e$	x_0	$e/2$	x2

We carefully set up the combiners so that if a value has been clamped at any point in the pipeline, future computations will never multiply the resulting number by less than $1/2$. This correctly ensures that the final output of TC1 after the multiplication by 2 will be 255 if any previous computations clamped.

If the texture combiners performed calculations in 8-bits of precision, the x2 multiplier at the end of TC1 would result in the low bit of the output always being zero. In the TNT2 and later architectures, however, all texture calculations are performed in 9 bits (although this is not guaranteed by the specifications), so we do not lose precision.

The ability to modulate a texture value by up to a factor of 16 implies that we can store 8-bit sections of an n-bit texture with each successive 8-bit chunk shifted 4 bits from the previous. Thus with two textures, we can store a 12-bit texture map. We may chain these 12-bit sections to achieve higher bit depth as in Section 3.1. Since we can only multiply by up to 16, however, this representation requires $4n-4$ textures to represent an $8n$-bit texture map. Again, only two textures are ever used simultaneously.

With this method, a 16-bit texture would usually require 4 textures; however, in this particular case it is actually possible to use a trick to reduce the number of textures required to only 3. We start with two 8-bit textures, x_0 and x_1, representing the least-significant and most significant 8 bits of a 16-bit texture as in Equation 1. We then precompute a third texture $z = clamp(16 \cdot clamp(x_1))$. Since we can scale a texture by up to 16, we can obtain $clamp(256x_1)$ using $clamp(16z)$. In the case that $e > 1/16$, we replace x_1 with z in the above texture combiner setup. If $e \leq 1/16$, we multiply e by 16, divide the scale factors in variables B and D in the TC1 stage by 16, and use x_1 as described in the table. This is the technique we used to create the images in this paper.

4 Applications

4.1 Visualizing HDR scenery

We have implemented a high-dynamic range panoramic image viewer using hardware-accelerated HDRTMs, similar to panoramic image viewers like Quicktime VR [2]. Our viewer is implemented with hardware-accelerated cube-based environment mapping, where we render a cube that surrounds the viewpoint with the panoramic texture appropriately mapped onto it. The user can change the viewpoint by rotating and zooming, as in Quicktime VR. Our viewer has an extra degree of freedom, however, since the user can additionally set the exposure level at which the panorama is rendered. The system is demonstrated in Figure 2.

The exposure can be set either by a slider or with an "auto-exposure" mode. In auto-exposure mode, the system samples pixels near the center of the visible portion of the panorama. The system then adjusts the exposure level to correctly expose the image for these pixel values, similar to [28, 22]. To keep the exposure from fluctuating in an unpleasant manner, we do not adjust exposure directly, but model it as a high friction spring system, where the rest state is set to the desired exposure. More sophisticated models of adaptation as in [28, 22] could also be investigated.

4.2 Lighting

We can also use HDRTMs to correctly model sharp specular reflections of bright environments from dark objects. We demonstrate this with high-dynamic range environment mapping of a teapot and simulating Fresnel reflection from the polished surface of a dark monolith.

Perfect specular reflection of convex objects can be simulated using graphics hardware by environment mapping techniques [9, 11] or by changing the camera's projection matrix according to the position of the reflector, as in [6] and [20]. While these techniques correctly calculate the position at which reflected objects appear on the sur-

face of the reflector, [3] notes that the intensity of the reflected light will be incorrect unless the reflected color is computed in full dynamic range before being clamped to the range of the display device.

For a point s on a perfectly specular surface, the reflected imaging function $I_s(r)$ calculates the value to render to the framebuffer given incident environment radiance r along the ray that is reflected towards the camera about s. Assume s modulates incident radiance by specular coefficient ϕ, where $\phi \in [0,1]$. To display the result of the reflection, we compute $I_s(v) = clamp(e\phi v)$. Thus we set our exposure level to $e \cdot \phi$ and render the value v that is stored in a HDRTM. To account for a gamma curve we must actually set the exposure level to be $(e\phi)^{1/\gamma}$ and use the texture value v'. An example using this technique for real-time environment mapping is shown for a dark polished teapot in Figure 3a; the specular reflection produces an image of the environment $1/8^{th}$ of the environment's brightness. Figure 3b shows the same result using just standard 8-bit texture maps incorrectly clamped at the exposure level seen in the background environment.

We can also simulate glossy materials with a two-pass technique shown in Figure 3e. First, we render the object with a pre-convolved diffuse environment map. Then we render the object with a darker specular environment map and add the results in the framebuffer. Note that we cannot correctly add gamma-corrected pixel values in the framebuffer, so the result is not physically correct, although it looks plausible. Figure 3f shows the correct result, obtained by compositing the two passes in software.

Most real-world polished surfaces will behave as near-perfect mirrors ($\phi = 1$) at glancing angles, while dropping to as low as $\phi = 0.05$ at angles normal to the surface (the so-called "Fresnel effect"). Because the reflectivity can change by a factor of 20, the environment map must have dynamic range of at least 20 times that of the display device. Figures 3c-d show the wide dynamic range seen by rotating about a polished black monolith in a bright environment. To model Fresnel effects on curved surfaces efficiently, we would need a way of continuously varying the exposure level at which the HDRTM is rendered over the surface. This may be possible using different hardware-based techniques on newer archnitectures such as the GeForce3 [8].

5 Conclusion and Future Work

We have demonstrated that high-dynamic range texture maps can be stored and rendered efficiently using current hardware texturing architectures by storing high and low bits in different texture maps and recombining them during rendering. While native support for high-dynamic range would be ideal, the relatively small memory and processor overhead in our implementations on the SGI Onyx3 and NVidia GeForce2 systems suggest that HDRTMs could be applied in real applications on current hardware.

While the applications to lighting and scene visualization demonstrated in this paper are useful, we believe that much more work could be done in this area. We would like to experiment with performing hardware lighting calculations in full dynamic range and using these results to illuminate textured surfaces, which may be possible with the latest geometry and lighting engine on NVidia's GeForce3 cards [18]. Other applications include high-dynamic range lightfields and image-based models, as well as interaction techniques that exploit the ability to adjust exposure level. The addition of direct support for arbitrary multiplication of texture values in the texture fetching stage, and the ability to correctly add gamma corrected values in the framebuffer and multitexturing units would enable more physically realistic real-time lighting simulation. We leave this as future work in the area of hardware design.

Acknowledgements

We thank Simon Green from NVidia for his helpful comments. This work was supported by funds from the University of Southern California and the United States Army but does not necessarily reflect any corresponding positions or policies and no official endorsement should be inferred.

References

1. CABRAL, B., OLANO, M., AND NEMEC, P. Reflection space image based rendering. *Proceedings of SIGGRAPH 99* (August 1999), 165–170.
2. CHEN, S. E. Quicktime VR - an image-based approach to virtual environment navigation. *Proceedings of SIGGRAPH 95* (August 1995), 29–38. ISBN 0-201-84776-0. Held in Los Angeles, California.
3. DEBEVEC, P. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH 98* (July 1998).
4. DEBEVEC, P. E., AND MALIK, J. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH 97* (August 1997), pp. 369–378.
5. DEBEVEC, P. E., YU, Y., AND BORSHUKOV, G. D. Efficient view-dependent image-based rendering with projective texture-mapping. In *9th Eurographics workshop on Rendering* (June 1998), pp. 105–116.
6. DIEFENBACH, P. *Pipeline Rendering: Interaction and Realism through Hardware-based Multi-Pass Rendering*. PhD thesis, University of Pennsylvania, 1996.
7. GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. The Lumigraph. In *SIGGRAPH 96* (1996), pp. 43–54.
8. GREEN, S. Personal communication, May 2001.
9. GREENE, N. Environment mapping and other application of world projections. *IEEE Computer Graphics and Applications 6*, 11 (November 1986), 21–29.
10. HAEBERLI, P., AND SEGAL, M. Quantization techniques for visualization of high dynamic range pictures. In *Fourth Eurographics Workshop on Rendering (Paris, France)* (June 1993), pp. 7–18.
11. HAEBERLI, P., AND SEGAL, M. Texture mapping as A fundamental drawing primitive. In *Fourth Eurographics Workshop on Rendering* (June 1993), M. F. Cohen, C. Puech, and F. Sillion, Eds., Eurographics, pp. 259–266.
12. HEIDRICH, W., AND SEIDEL, H.-P. Realistic, hardware-accelerated shading and lighting. *Proceedings of SIGGRAPH 99* (August 1999), 171–178. ISBN 0-20148-560-5. Held in Los Angeles, California.
13. KAUTZ, J., AND MCCOOL, M. D. Approximation of glossy reflection with prefiltered environment maps. *Graphics Interface* (2000), 119–126. ISBN 1-55860-632-7.
14. KAUTZ, J., AND MCCOOL, M. D. Interactive rendering with arbitrary BRDFs using separable approximations. *Eurographics Rendering Workshop 1999* (June 1999).
15. LARSON, G. W. Logluv encoding for full-gamut, high-dynamic range images. *Journal of Graphics Tools 3*, 1 (1998), 15–31. ISSN 1086-7651.
16. LARSON, G. W., RUSHMEIER, H., AND PIATKO, C. A visibility matching tone reproduction operator for high dynamic range scenes. *IEEE Transactions on Visualization and Computer Graphics 3*, 4 (October - December 1997), 291–306.
17. LEVOY, M., AND HANRAHAN, P. Light field rendering. In *SIGGRAPH 96* (1996), pp. 31–42.
18. LINDHOLM, E., KILGARD, M., AND MORETON, H. A user-programmable vertex engine. In *SIGGRAPH 2001* (2001).
19. NVIDIA CORPORATION. NVIDIA OpenGL extension specifications. Tech. rep., NVIDIA Corporation, 2001.
20. OFEK, E., AND RAPPOPORT, A. Interactive reflections on curved objects. *Proceedings of SIGGRAPH 98* (July 1998), 333–342. ISBN 0-89791-999-8. Held in Orlando, Florida.
21. PATTANAIK, S. N., FERWERDA, J. A., FAIRCHILD, M. D., AND GREENBERG, D. P. A multiscale model of adaptation and spatial vision for realistic image display. *Proceedings of SIGGRAPH 98* (July 1998), 287–298.
22. PATTANAIK, S. N., TUMBLIN, J. E., YEE, H., AND GREENBERG, D. P. Time-dependent visual adaptation for realistic image display. *Proceedings of SIGGRAPH 2000* (July 2000), 47–54. ISBN 1-58113-208-5.
23. PEERCY, M. S., OLANO, M., AIREY, J., AND UNGAR, P. J. Interactive multi-pass programmable shading. *Proceedings of SIGGRAPH 2000* (July 2000), 425–432. ISBN 1-58113-208-5.
24. PULLI, K., COHEN, M., DUCHAMP, T., HOPPE, H., SHAPIRO, L., AND STUETZLE, W. View-based rendering: Visualizing real objects from scanned range and color data. In *Eighth Eurographics Workshop on Rendering* (June 1997), pp. 23–34.
25. ROGERS, D. TNT 8-stage setup in Direct3D. Tech. rep., NVIDIA Corporation, 2001.
26. SCHEEL, A., STAMMINGER, M., AND SEIDEL, H.-P. Tone reproduction for interactive walkthroughs. In *Eleventh Eurographics Workshop on Rendering* (2000).
27. SCHLICK, C. High dynamic range pixels. *Graphics Gems IV* (1994), 422–429.
28. TUMBLIN, J., HODGINS, J. K., AND GUENTER, B. K. Two methods for display of high contrast images. *ACM Transactions on Graphics 18*, 1 (January 1999), 56–94. ISSN 0730-0301.
29. TUMBLIN, J., AND RUSHMEIER, H. E. Tone reproduction for realistic images. *IEEE Computer Graphics & Applications 13*, 6 (November 1993), 42–48.
30. TUMBLIN, J., AND TURK, G. Lcis: A boundary hierarchy for detail-preserving contrast reduction. *Proceedings of SIGGRAPH 99* (August 1999), 83–90. ISBN 0-20148-560-5. Held in Los Angeles, California.
31. WARD, G. Real pixels. *Graphics Gems II* (1991), 80–83.
32. WARD, G. J. The RADIANCE lighting simulation and rendering system. In *SIGGRAPH 94* (July 1994), pp. 459–472.

Fig. 2. Screen snapshots from our high-dynamic range panorama viewer. The viewer implements our HDRTM technique allowing these results to be rendered interactively at 50 Hertz at 640×480 pixels with 4x antialiasing. The exposure changes by a factor of 100 over these 5 images.

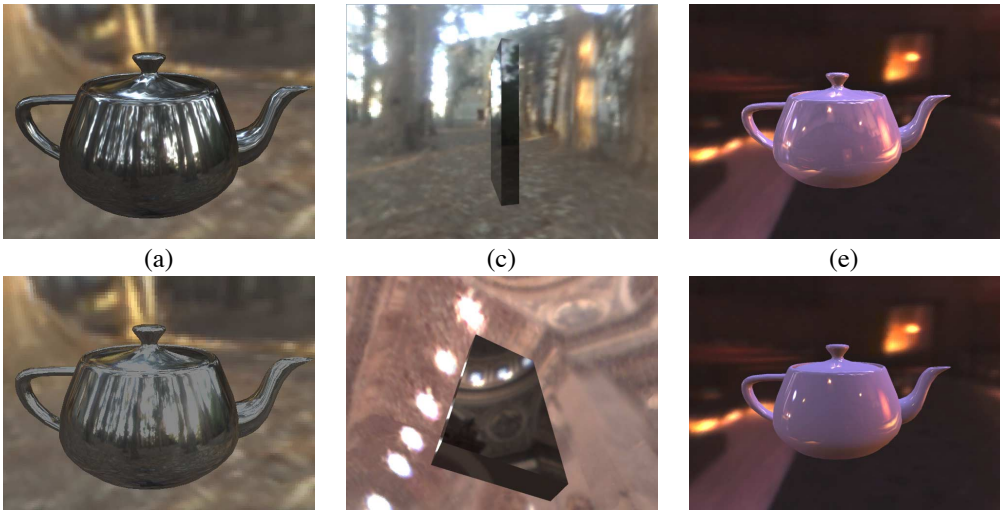

Fig. 3. **(a)** Environment mapping with a specular coefficient of $\phi = 0.125$ using HDRTMs. **(b)** Environment mapping with 8-bit per-channel texture maps. The attenuation due to reflection is not displayed correctly. **(c)** and **(d)** Simulated Fresnel reflection on a monolith in two environments. For the faces seen at glancing angles, the monolith is a near-perfect mirror. For the faces viewed more directly, the specular reflectance drops to just 5 percent, revealing the colors of the bright light sources. **(e)** A colored glossy material is simulated by adding a diffuse environment map to a specular environment map with $\phi = 0.08$. Because the textures have been gamma-corrected, the result of compositing in the framebuffer is incorrect. **(f)** The simulated correct result.