Assignment 6: High Dynamic Range Imaging

CS 148 Autumn 2012-2013

Due Date: Wednesday, November 7th, 2012 by 7pm

(a) Non-HDR Image (b) HDR Image

Figure 1: Comparison between Non HDR and HDR images

Introduction

In this assignment, you'll explore high dynamic range photography, generate a high dynamic range image, and explore a couple of ways of displaying HDR images on low dynamic range displays. You will also create an HDR light probe for use in a later assignment. There are many steps to this assignment, but they are essentially independent. We will list the steps in logical order, but you may want to do the simpler steps first to help your understanding. Don't feel overwhelmed by the number of steps, as each step is relatively short.

Steps

Read through all of the steps below to make sure you fully understand them. At the end of this document, a summary will list what you need to do more concisely.

Download and Compile the Starter Code

1. Download the starter code.
2. Build `libST`. This should require the same steps on your development system as it did in previous assignments.
3. Build the starter code. The subdirectory `/hdr` contains the starting code for your project. It should contain the C++ source file `main.cpp`. The directory also includes `response.h`, `response.cpp`, `STHDRImage.h`, `STHDRImage.cpp`, and some support code in the directory `/tnt`. You should understand the interface provided in `response.h` since you will use it in your code. `STHDRImage` is similar to `STImage` except that it stores its pixels as `STColor3fs` to make it easier to perform calculations on HDR images. You are not responsible for any of the `/tnt` code.
4. You can download the sample photos from the course website. You can also take your own photos, in which case you will also have to take pictures of the chrome ball in your scene.

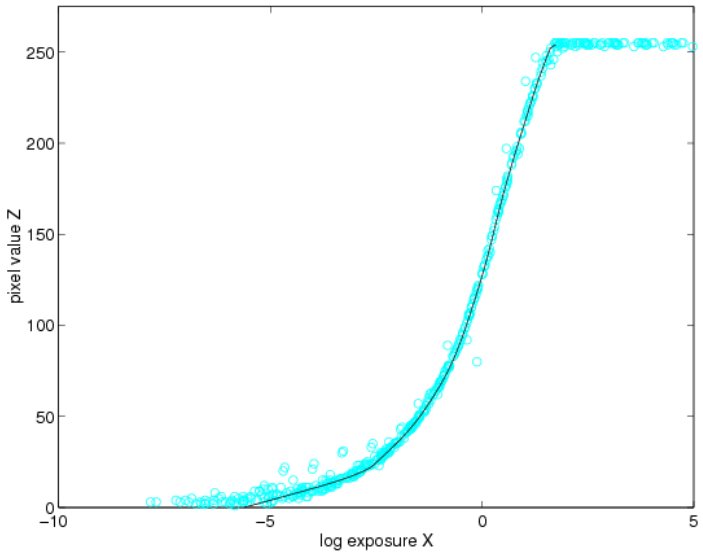

Figure 2: Sample Response Curve

Review of Camera Imaging and HDR

The input to a camera is the radiance of the scene. Light enters the camera, passing through the lens system and aperture and producing some irradiance $\mathbf{E}_i$ at each pixel i. The camera can control how much energy actually hits the sensors by increasing or decreasing the shutter speed. We can integrate the irradiance $\mathbf{E}_i$ over the shutter time **dt** to get the sensor exposure $\mathbf{X}_i$. The many steps from this point to producing an actual image (film exposure, development, digital conversion, remapping) act as a non-linear function which maps these input exposures in the range $[0, \infty]$, to image values $\mathbf{Z}_i$ in the range $[0, 255]$. This function $f(X_i) = Z_i$ is called the **camera response curve**. Our goal is to use some output images to get the input X_i values (and the E_i values). Therefore, we actually want to consider the inverse function $f^{-1}(Z_i) = X_i$. By substitution, we get $f^{-1}(Z_i) = E_i * dt$. Taking the natural logarithm we have $\ln(f^{-1}(Z_i)) = \ln E_i + \ln dt$. We'll rename this function $g(Z_i) = \ln E_i + \ln dt$. This is the representation we'll use for the response curve (given a pixel value, what is $\ln($`exposure`$)$). Note that this represents exactly the same information; we've just adjusted the representation a bit. Using the natural logarithm is convenient for our purposes because different shutter speeds are equivalent to translations of this function. We show a sample response curve in Fig. 2

Answer These Questions

Answer the following questions before doing the assignment:

1. What response do we get for exposure values beyond the ends of the response curve? What problem does this cause if we want to use the inverse?
2. Why do we have to tone map HDR images before displaying them? What would they look like if we didn't?

Answer the following questions after doing the assignment:

1. How well does the tone mapping work? Can you see everything you think you should be able to? Do you notice anything this tone mapping algorithm doesn't handle well? Does tone mapping an HDR image offer much improvement over simply taking a single photograph with an exposure time in the middle of the range?

Running the Assignment

This program uses command line arguments. If you're not sure how to specify command line arguments in visual studio or X code, look at the bottom of this page for screenshots.

```
hdr -response photos.list response_out.cr
hdr -create photos.list response.cr photo_out.pfm
hdr -view photo.pfm;
hdr -vp photo.pfm response.cr
hdr -tonemap photo.pfm
```

Options:

- **response** - Calculate the camera response file.
- **create** - Create an HDR image in PFM format.
- **view** - View the HDR image with simple linear mapping.
- **vp** - Take a virtual photograph.
- **tonemap** - View the HDR image with non-linear tone mapping.

File Formats

You should only need to edit the photo stack list files, but here's the information about all the formats used:

- Photo Stack Lists are stored in `.list` files. Each line of this file is of the form `<numerator> <denominator> <filename>`. The first two parameters give the shutter time in seconds. The last parameter is the name of the image file. Anything STImage supports will work here; presumably you'll be using JPEGs.
- Camera response curves are stored in `.cr` files. The file format is just a text dump of the 256 curve values, one for each color. See the CameraResponse class for details.
- HDR Images use the **Portable Float Map (PFM)** file format. This is an extremely simple format and doesn't support compression, so resulting files can be very large.

Taking Your Own Photographs

When taking photos using your own camera, you will need to recover the response curve for your camera. This can be done with the same set of images you take to create the HDR image, just follow the instructions listed below. Remember that your scene must be still while taking the pictures. Some small movements (trees swaying a bit) can be tolerated and generally shouldn't cause a problem. But trying to take pictures in the quad during midday is going to be difficult. To keep your camera steady for all the shots, you should either use a tripod or set your camera on a fixed object. You can also use a program like Photoshop to align your images and crop them to remove camera shake (although this won't remove camera shake blur for the long exposure shots).

Remember that the entire point of HDR images is that the scene you're photographing has high dynamic range, so make sure you have both relatively dark and bright regions. Pictures of sunset work very well, especially with some clouds. Be creative, just be sure your scene actually requires multiple exposures to recover the HDR image. Search online for some HDR images if you need inspiration.

Once you've found your scene, take some sample photos with various aperture and shutter speed settings. **Once you settle on an aperture and focus, do not change them while taking your sequence of photos.** The number of photographs you need and the shutter speeds to use will vary depending on the dynamic range of your scene. A good way to ensure good results is to make sure all pixels have at least one photo in which its response is in the range [64,192]. This means you want at least one with shutter speed fast enough that the scene is almost invisible - only the brightest objects are at half the brightest response. And you want one with shutter speed long enough that the image looks almost entirely washed out - only the

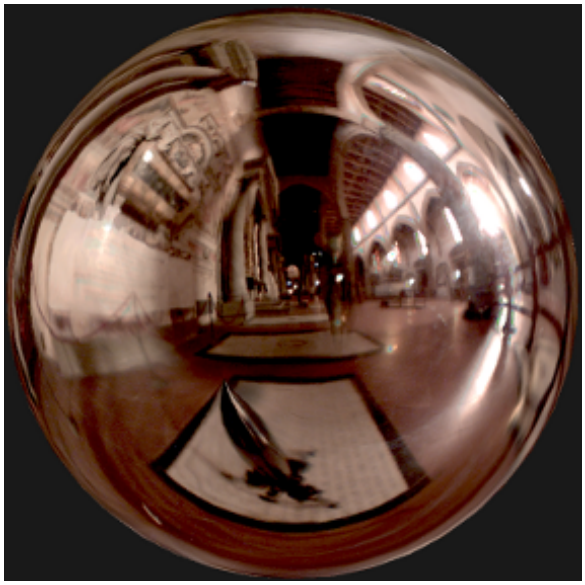

Figure 3: Sample light probe image

darkest areas aren't white but are not black either. The number of photos in between is harder to determine, but increasing by 1 stop for each photo seems to work well.

You should keep track of the shutter speeds while taking the photos, but you can also retrieve the information from the JPEG's EXIF data. Once you have all your photos you can construct a photo stack list file using the format described below.

Creating a Light Probe of Your Scene

Note that we are asking you to create 2 HDR images.

- One image will be of a scene of your choosing.
- The other image will be a light probe of the same scene.

A light probe is a hemispherical panoramic high-dynamic range image that captures all the lighting information of one half of a scene. There are many ways to construct a light probe image, but for this assignment, we will create a light probe image by taking pictures of a highly-reflective chromium ball. Fig. 3 is an example of a light probe image that was constructed using this technique.

As you can see from the above image, the spherical shape of the ball reflects the light rays heading towards the ball from virtually all directions. Information about the lighting behind the ball will be somewhat distorted around the edges of the ball, but is still present.

In order to construct an HDR light probe, you will need to place the ball into the scene of your HDR image, then take a series of pictures of the ball at different exposures in order to create an HDR image of the ball itself. Before you use the assignment program to merge the LDR pictures of the chromium ball, **make sure to crop the final image to the edges of the ball**, as this is required in order to receive credit. See the image above for a visual explanation of the crop specifications. Please come to a CA's office hours to check out a chrome ball.

Recovering the Camera Response Curve

We've provided the code for this step. Once you've compiled the skeleton code and have your photo stack list finding your camera response curve can be done using the following command

```
hdr -response photos.list cameraresponse.cr
```

which takes the list of photos in `photos.list` and generates the `cameraresponse.cr` file. This is accomplished by solving a linear least squares problem. Its a good idea to open and plot the response curve in Excel or another program to make sure your curve looks right and is smooth enough.

Creating an HDR Image

An HDR image is just the irradiance values $\mathbf{E}_i$ at each pixel due to the scene. In other words, an HDR image represents the actual power incident on each sensor, before any of the other effects of the camera after the lens and aperture, have taken place. Assume we have the $\mathbf{g(Z}_i\mathbf{)}$ representation of the response curve and just a single input image with known shutter speed $\mathbf{dt}$. Then we could simply rearrange the response curve formula to find $\mathbf{E}_i$:

$$\begin{aligned} g(Z_i) &= \ln E_i + \ln dt \\ \ln E_i &= g(Z_i) - \ln dt \\ E_i &= \exp(g(Z_i) - \ln dt) \end{aligned}$$

Why can't we just use a single photograph? The response curve can't give pixel values less than 0 or greater than 255, so all exposures above a certain level will result in a response of 255 and similarly for exposures below a certain level. This means that multiple exposure values map to the same response pixel value, so the function cannot actually be inverted. Therefore the above equations cannot determine the actual exposure value if the input pixel value is too small or too large.

So we need enough photographs such that every pixel is within a usable range in at least one photo (best if its, e.g., in [64,192]). Using this method, j photos will result in j exposure estimates for each pixel (that is, we have $E_{ij} = \exp(g(Z_{ij}) - \ln dt_j)$ for each pixel i in each image j). Then we need to combine them somehow. To do this we'll weight them so that E_{ij} will have high weight if pixel Z_{ij} has a high confidence value, i.e. those in the middle of the response curve. We've already created a weight function for you (`CameraResponse::Weight()`) which accomplishes this. Then we just calculate a weighted average (which we do on the ln values, not the actual values):

$$\ln E_i = \frac{\sum_j \texttt{weight}(Z_{ij}) * (g(Z_{ij}) - \ln dt_j)}{\sum_j \texttt{weight}(Z_{ij})}$$

and the E_i can be easily found.

You should implement this function in `recover_hdr()`. Only the arguments to the function are necessary to accomplish this. Carefully read the comments for the functions of the `CameraResponse` class to make sure you're getting the values you expect (i.e. whether you're getting the value itself or ln(`value`)).

You can perform this process by running

```
hdr -create photos.list response.cr photo_out.pfm
```

which will take the file `photos.list` and the response curve `response.cr` and create the HDR image and save it to `photo_out.pfm`. Note that pfm files are uncompressed, so for high resolution images such as those from cameras, the file will be quite large (close to 100 megabytes).

Taking a Virtual Photograph

Given an HDR image and a camera response we can actually find the image that would be generated by that camera. This is a straightforward use of the response curve. For some shutter speed $\mathbf{dt}$, we find the irradiance for each pixel and look up the response. We've given you a function in `CameraResponse` to lookup the response for some irradiance, so this should be simple to implement. The camera response curve we use

doesn't necessarily need to be the one the original photographs were taken with. You can actually find out what the same image, taken with different imaging systems, would look like! Implement this process in the function `virtual_photo()`. When you run the program with

```
hdr -vp photo.pfm response.cr
```

it will display calculate and display virtual photographs. You can increase/decrease the shutter speed using +/−. It increments by 1/3 stop. Hitting 's' will save the current virtual photograph to `vp.jpg`.

Displaying an HDR Image: Linear Mapping

In order to display an HDR image we need to map it to a low dynamic range display. Linear mapping is the simplest approach and simply scales and clamps the HDR values so a certain range is displayed and any values outside that range are clamped to the maximum LDR value. Implement this approach in `scale_hdr` so that `[0,max_val]` in the HDR image maps to [0,255] in the LDR image. When you run the program with

```
hdr -view photo.pfm
```

it will use this display scheme. Use +/− to increase/decrease the range of values that are scaled to the range [0,255]. Hitting 's' will save the current image to `view.jpg`.

Displaying an HDR Image: Tone Mapping

After implementing and experimenting with the linear mapping you'll realize that this is not a very effective way of displaying HDR images. A non-linear mapping from the HDR values to the LDR values can show detail at more levels. Here we'll describe a very simple tone mapping algorithm which is pretty effective. Your job is to implement it. This algorithm is based on the key of a scene. The key of a scene indicates whether it is subjectively light, normal, or dark. High-key scenes are light (think of a white room snow), normal-key scenes are average, and low-key scenes are dark (think of night shots). This algorithm uses the log-average luminance as an approximation to the key of the scene. The log average luminance is defined as

$$Lw_{\texttt{avg}} = \exp\left(\frac{1}{N}\sum_i \log\left(\delta + Lw_i\right)\right)$$

where i indexes pixels, Lw_i is the luminance of the pixel (look in the lecture notes for how to calculate luminance), and N is the number of pixels. δ is a small value which ensures we don't ever take the log of 0. We then scale all pixels E_i in the HDR image using this value and a user defined key setting, `a`:

$$E_{i\,\texttt{(scaled)}} = \left(\frac{a}{Lw_{\texttt{avg}}}\right) * E_i$$

So far all we've done is scale the values. We now need to compress all possible values into the range [0,1]. To do this we do the following

$$C_i = \frac{E_{i\,\texttt{(scaled)}}}{1 + E_{i\,\texttt{(scaled)}}}$$

When E_i is 0, C_i is also 0. As E_i gets larger, C_i gets larger as well, but its upper limit is 1. Finally we can scale this to the range for `STPixels` and store it in the output image. Code this algorithm in the `tonemap()` function. When you run the program with

```
hdr -tonemap photo.pfm
```

it will use this display scheme. Use +/- to increase/decrease the key value. Hitting '`s`' will save the current tonemapped image to `tonemapped.jpg`.

Summary

Here's a quick summary of everything you need to do to receive full credit on the assignment. Refer to this if you get confused by what we are asking for throughout the document.

1. Implement functionality for creating an HDR image from a series of LDR images in your project.
2. Implement functionality for viewing an HDR image using a linear mapping algorithm.
3. Implement functionality for creating virtual photographs.
4. Implement tone mapping on the HDR image.

Hints for Getting Started

- Note that this program only works with command line arguments. You must specify the function you want to do and its parameters. If you're not sure how to specify command line arguments in Visual Studio or Xcode, refer to Fig. 4, 5, and 6 respectively at the end of the document.
- The files you are dealing with are relatively large. If you use enough shots they can't all fit in memory at once. Make sure you write your code such that only 1 image from disk is in memory at a time.
- Running with Debugging can be painfully slow for large images such as those you'll get from the camera. We're providing some `.pfm` files for testing the second part of the assignment but when you're confident of your implementation of a function you'll want to run it in Release mode when possible.
- You'll probably want to scale down your images for testing. Doing so will make your debug cycle go much faster.

Grading

- (1 points) Questions
- (2 points) Questions + generated HDR Images (scene, light probe)
- (3 points) Questions + generated HDR Images (scene, light probe) + virtual photographs + linear mapping view
- (4 points) Questions + generated HDR Images (scene, light probe) + virtual photographs + linear mapping view + tone mapping

We will be looking for especially creative HDR imagery (taking your own photos and tonemapping them) for this week's project showcase.

Specifying Command Line Arguments in an IDE

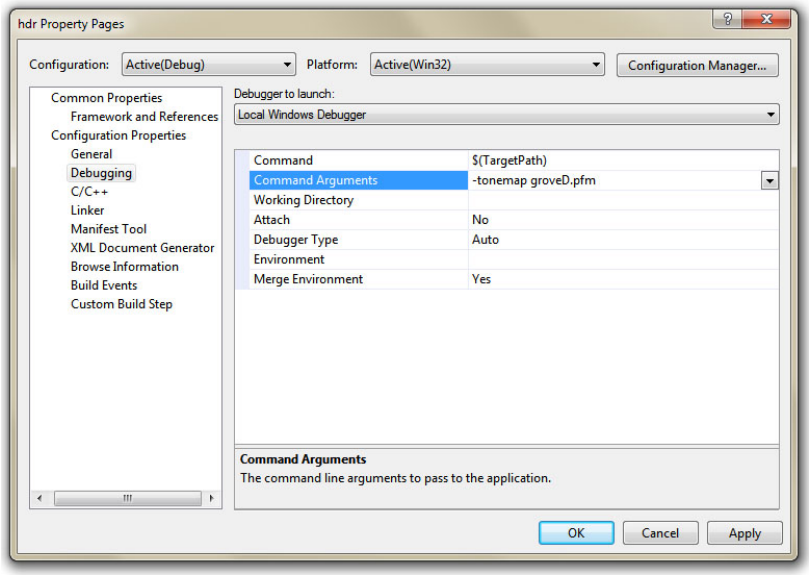

Figure 4: Specifying Command Line Arguments in Visual Studio

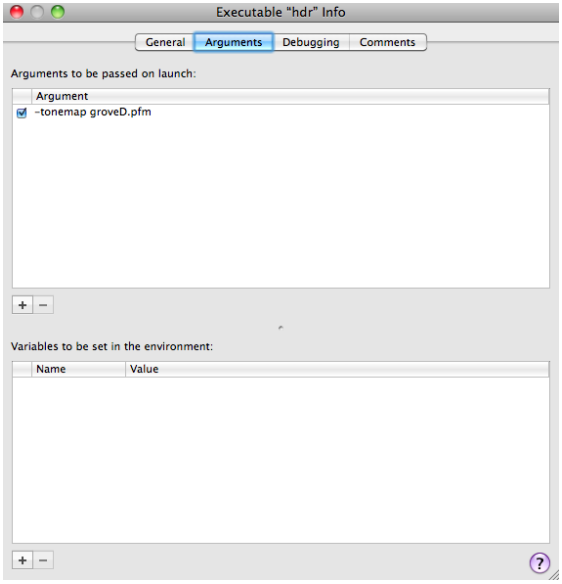

Figure 5: Specifying Command Line Arguments in Xcode 3

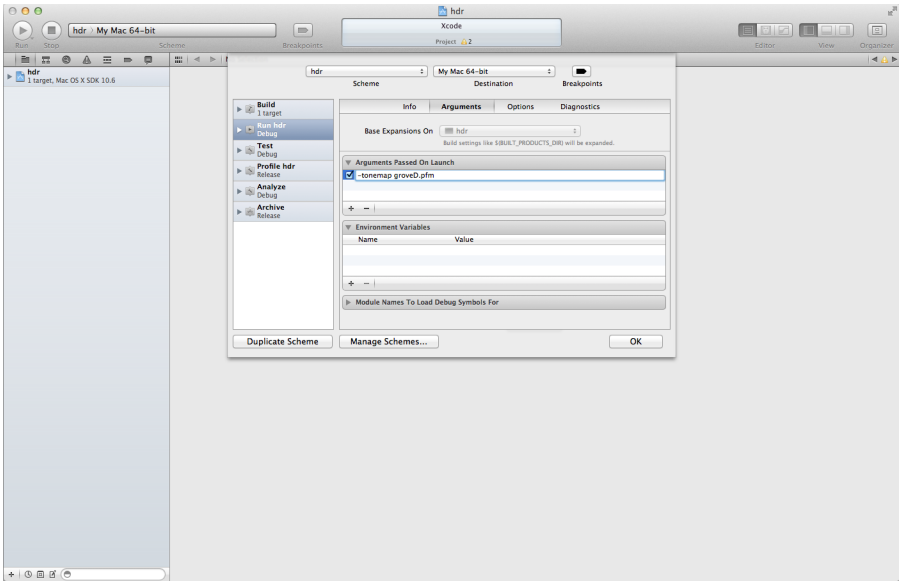

Figure 6: Specifying Command Line Arguments in Xcode 4