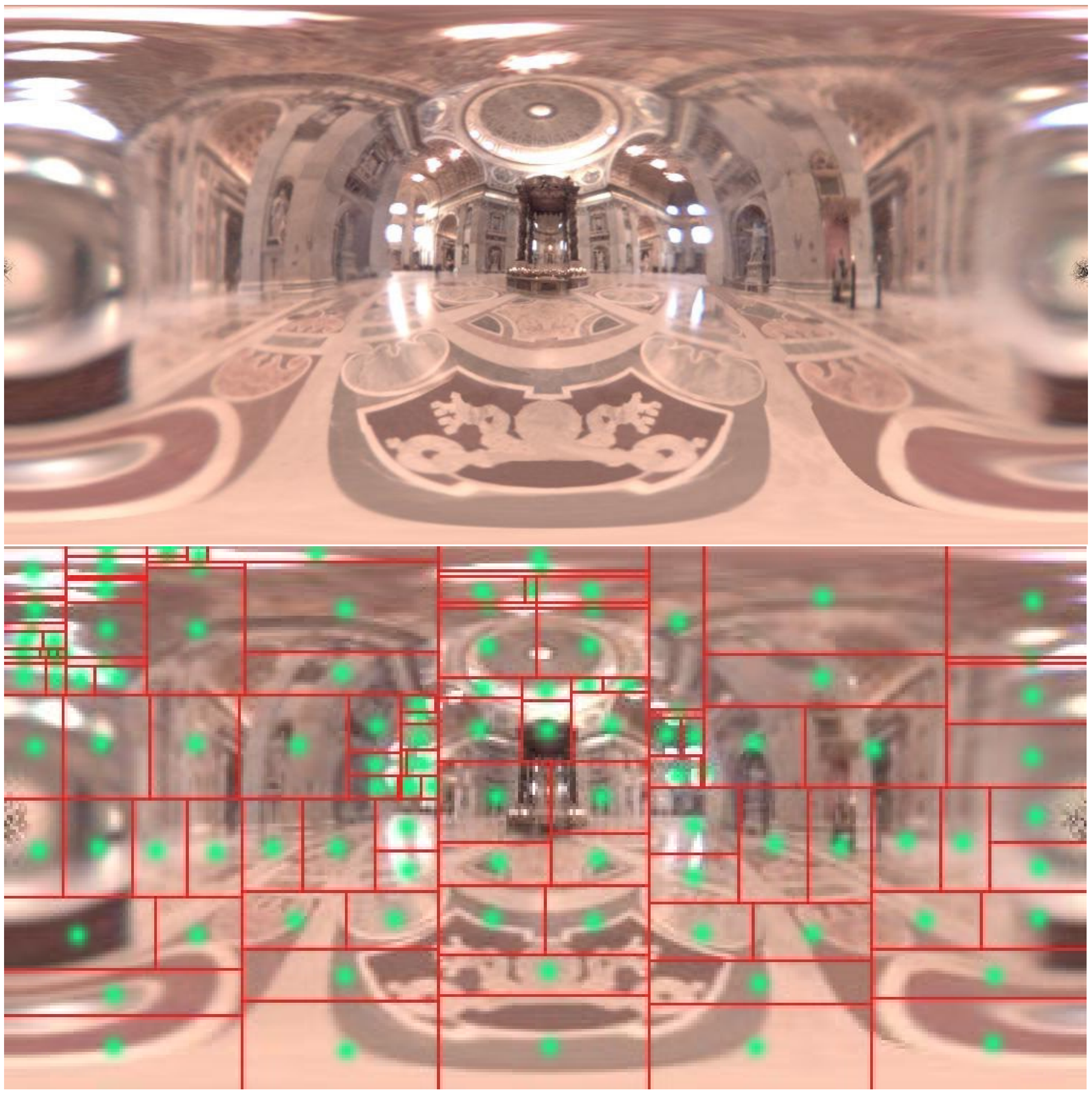
Median Cut Algorithm in Image Based Lighting

Olof Landahl
TNM078 Image Based Rendering
Linköpings Universitet, 2007



Abstract

The idea with this project was to implement the Median Cut Algorithm to convert an HDR light probe image to a set of light sources, as described in [1]. The basic idea is to divide an image (in latitude-longitude format) into regions with equal light energy and then represent each region with a light source.
This was supposed to be implemented in Matlab and result in a script file for 3D Studio Max where it can be used to create these light sources.
By combining virtual objects with the light sources and the scene environment, the goal was to get a realistic rendering of the virtual objects in the real scene.

Median Cut Algorithm

The Median Cut Algorithm has been applied in many areas before but in this implementation it's used in image based lighting.
By dividing a light probe into regions with equal light energy, conventional light sources for 3D rendering can be created, representing the light in scene where the light probe was captured. This can be very useful when adding virtual objects into the scene. This gives relatively short rendering times and a good visual result for 64 light sources or more.
Having a set of conventional light sources also makes modifying and adjusting the light in the scene very easy.

Starting with the whole image as a region, in every itreration each region is split into two new regions with equal light energy (or equal sum of pixel intensities). The split is done in the longest dimension of the region. The way of splitting makes the number of regions (light sources) increase exponentially with the number of iterations n:

$$\text{number of light sources} = 2^n$$

When the number of iterations is reached each region is converted into a light source with position, color and intensity.

Implementation

The algorithm was implemented in Matlab.
The input image has to be in HDR latitude-longitude format. I used an HDR light probe image from [2] (see figure 1). In figure 2 it's transformed into latitude-longitude format.

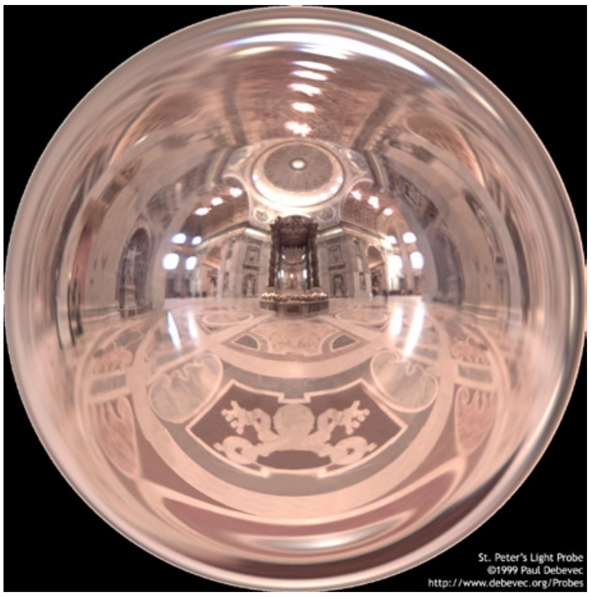


Figure 1. An image in angular map format taken from [2].

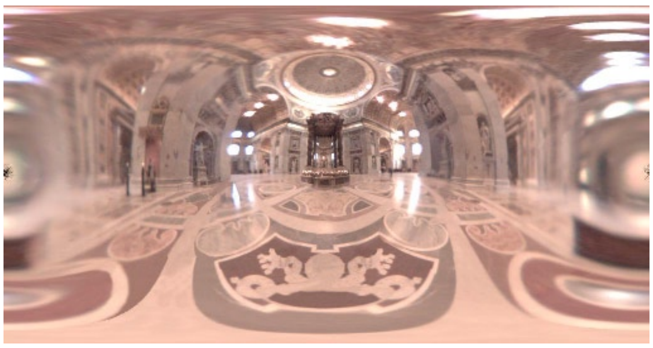


Figure 2. An image in latitude-longitude format, representing the light in a scene.

Converting a light probe angular map into latitude-longitude was done in HDR Shop [3].
Running the algorithm also needs the number of iterations and distance to the light sources to be set. Before starting the splitting, the image is converted into a monochrome representation to make the light energy comparison easier. Since the three color channels (R, G and B) don't represent the same amount of light, ITU-R Recommendation BT.709 [4] was used to represent the light intesity from the three channels in one single channel:

$$Y = 0.2125R + 0.7154G + 0.0721B$$

Also, the pixel values has to be scaled according to their angle of inclination, since the light in the latitude-longitude image is over-represented at the poles. The result is a monochrome, scaled image which is used when executing the recursive cutting function.

The cutting function calls itself in the end of each iteration until the user-set number of iterations has been reached and it works like this:

1. Find the longest dimension in the current region (scale region width according to the angle of inclination) and set that dimension to be split.

2. Move the "splitting line" along that dimension until the difference of the two resulting region sums is as small as possible. This means that the regions should have as equal amount of light as possible.

3. Specify the two new regions borders and split them by calling the cutting function again for each region (unless the number of iterations has been reached). To see how the splitting is done, take a look at figures 3-6.

4. When the number of iterations has been reached another function is called, a function that specifies the light source corresponding to that region.

Now the color image is used to set the light color by taking the sum of each color channel in that region and scale it according to the angle of inclination (this had only been done on the monochrome image before).
The color is normalised to fit the RGB color space by dividing each color channel by the largest color value. An intesity (multiplier) is also specified but I had problems finding the best way to make it general for any scene and make it look good in 3D Studio Max. Instead the multiplier can be specified and fiddled with in the program file to get a good result.
Since the light sources are directional lights the target is set to origo in the scene and raytrace shadows are used.
The light sources are positioned in the centre of each region (see figure 7).

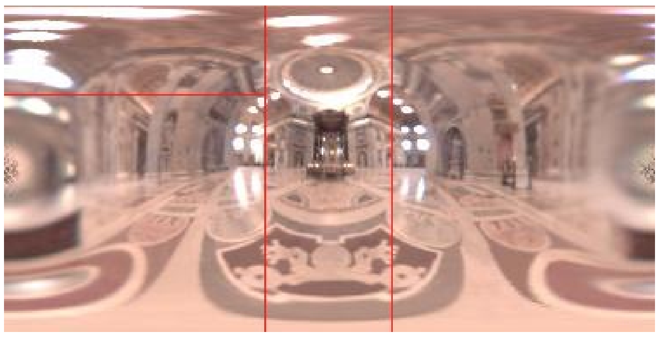


Figure 3. Light regions after 2 iterations.

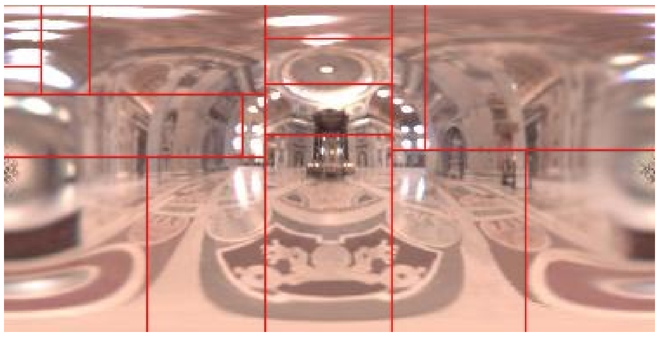


Figure 4. Light regions after 4 iterations.

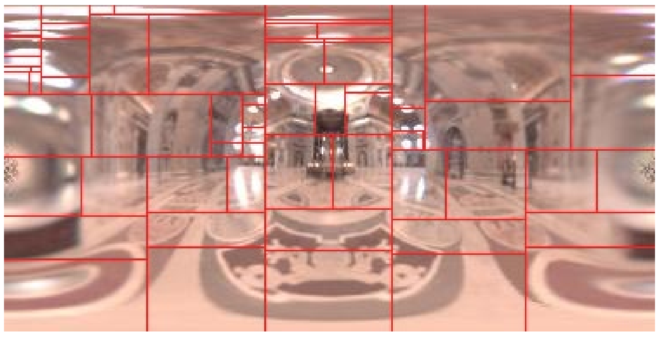


Figure 5. Light regions after 6 iterations.

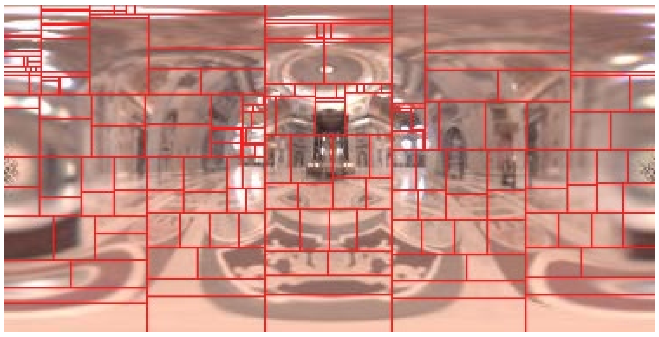


Figure 6. Light regions after 8 iterations

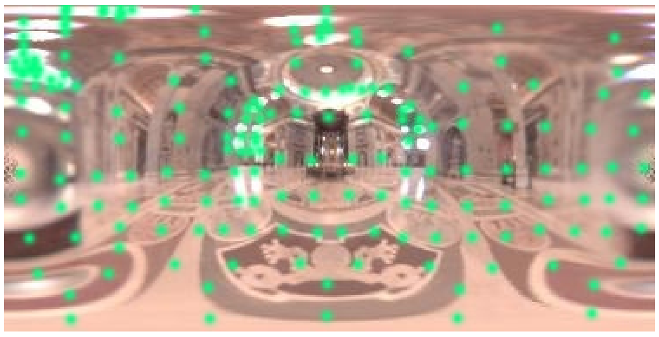


Figure 7. Light source positions after 8 iterations.

By converting the latitude-longitude (PI x 2PI) coordinates to polar coordinates (where the radius has been set by the user) a 3D cartesian coordinate position can be specified for the light source. The position and color information is stored in a MaxScript file.

To split a region, it can't be too small. It must be possible to split it into two new regions or the recursion stops and a light source is specified. This can result in the number of light sources to be less than 2^n.

Practical information (How to use the program)

Open program.m and specify the name of the light probe image file (hdrImage), the number of iterations (n), radius (r) and multiplier (m). The script file name can also be specified (fileName). Run the program to generate the script.
The resulting script file is implemented in 3D Studio Max by entering the MAXScript menu and click on Run Script. Specify the desired script file and the lights are imported into the scene (see figure 8). If an environment map is used, make sure it's mapped on the same distance as the lights are positioned (the specified radius).

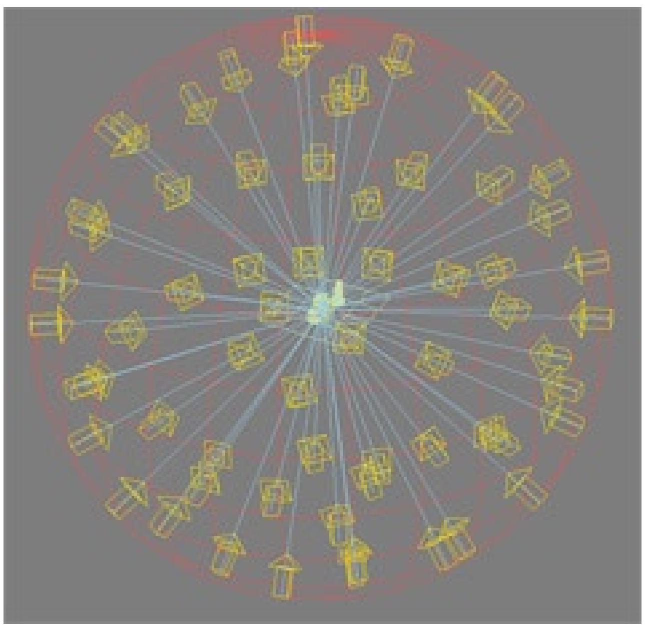


Figure 8. Light sources script imported in 3D Studio Max

Result

When the script file was imported into 3D Studio Max, I added an environment map and created some objects in the scene. The rendered image should look realistic with 64 or more light sources (6 iterations). But because of my problem with finding a good specififcation of the multiplier for each light source the result wasn't satisfying. The exact region sums can't be used since this will "overlight" the objects.
I had two ideas how to decide the correct value. The first one was to simply apply the multiplier values so that their sum was 1, ie every multiplier was set to 1/(2^n). The other idea was to multiply 1/(2^n) with the same value that the color was normalised with. Neither one of these made a good result. Instead I had to fiddle around with the multiplier to get a realistic rendered image.

The result (which can be seen in figure 9) was pretty good. The colors and shadows looks accurate and the light intensity was ok after trying some different multipliers for the light sources.

Although the result wasn't as good as I had hoped (mainly because of the problems with specifying multipliers) it's still acceptable.

References

[1] Paul Debevec, A Median Cut Algorithm for Light Probe Sampling, 2005

[2] Paul Debevec, Light Probe Image Gallery, http://www.debevec.org/Probes/

[3] HDR Shop, http://gl.ict.usc.edu/HDRShop/

[4] International Telecommunication Union-Radiocommunication, http://www.itu.int/ITU-R/

[5] E. Reinhard, G. Ward, S. Pattanaik & P. De bevec; High Dynamic Range Imaging, Morgan Kaufmann, 2006

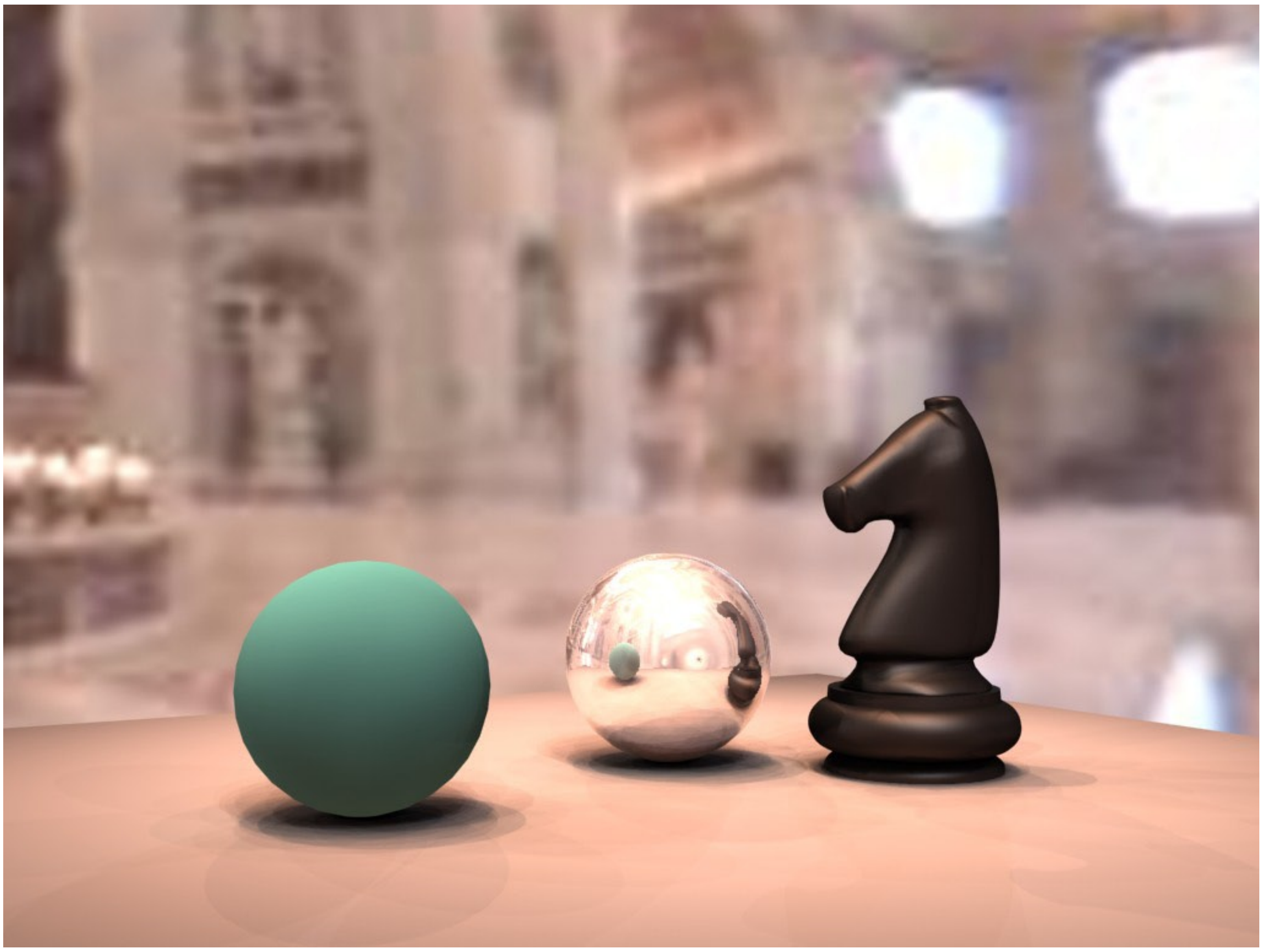


Figure 9. A rendered scene with 3D objects and light from the real scene.