**Rendering HW3 Report**

**Median Cut Algorithm**

Median cut algorithm is a simple algorithm which uses several point lights to simulate environment light. To achieve that, median cut algorithm turns a light probe image into several light sources, then samples lights from the point lights.

To create point lights, we have to subdivide the image into several regions of equal energy. First, we take whole image as one region and add it to region list. Second, we subdivide along the longest side of all regions in region list such that the two regions divided from original region have equal energy. Third, if the number of regions is less than the number we want, return to second step. Finally, put a point light at the centroid of each region in region list and set the point light spectrum to the sum of all pixels’ spectrum within the region.

**Implementation Details**

To implement the median cut algorithm, I copied infinite light implementation and modified three functions of it, constructor, Sample\_L, isDeltaLight.

In constructor, I subdivided regions and constructed point lights here. To subdivide a region into two sub-regions of equal energy, I had to get energy of each pixel first. I took Y(luminance) as energy instead of RGB value. To speed up the calculation of sum energy in region, I used sum area table. Each pixel in sum area table represents sum energy in the region of (0,0) to the pixel. We can construct the table in a dynamic programing fashion because the value of each pixel can be calculated by the equation( left pixel + upper pixel – left-upper pixel). When we subdivide regions, we can use binary search to speed up the dividing point. At the end, I normalized the spectrum by solid angle (2\*pi^2/(image\_width\*image\_length)) and sintheta when I set the spectrum to point lights.

In Sample\_L function, I uniformly chose a point light created in constructor because all lights have roughly equal energy, and set its pdf to 1/number\_of\_lights.

In isDeltaLight function, I made it always return true to avoid BRDF importance sampling.

**Performance Comparison**

Table 1

Grace\_latlong execution time(sec)

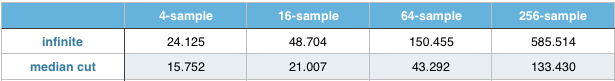
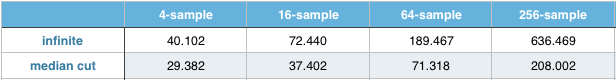


Table 2

Grace-new\_latlong execution time(sec)



From two table above, we can find that the median cut algorithm has better performance than infinite. When the sample rate goes higher, the difference in performance becomes larger. Although median cut algorithm needs more time to construct its light sources, it’s very fast to execute the sample\_L because it only needs to uniformly choose one light in light sources and avoid the importance sampling. Besides, it spend very few time to construct its light sources.

From the Result section below, we can find that the results rendered by infinite have many noise but the results rendered by median cut are noise-free. Besides, the shadow in results generated by median cut is unnatural. We can easily find that the car looks like being in an environment with many point lights from my results under 64-sample because the number of point lights is the same as the sample rate. To render a more natural picture, it’s better to make the number of point lights more than 64.

**Result**

Grace\_latlong(after tone mapping):

4-sample

**** ****

infinite median cut

16-sample

** **

infinite median cut

64-sample

** **

infinite median cut

256-sample

**** ****

infinite median cut

Grace\_new\_latlong:

4-sample

**** ****

infinite median cut

16-sample

** **

infinite median cut

64-sample

** **

infinite median cut

256-sample

**** ****

infinite median cut

**Environment**

Mac OS X EI Capitan 10.11.2, 2.7GHz Intel Core i5, 8G Ram, Intel Iris Graphics 6100, build pbrt project by Xcode