Project 3: Environment Light

B03902124 黃信元

I have implemented P. Debevec's paper: A Median Cut Algorithm for Light Probe Sampling and also their later modification: Variance Minimization Light Probe Sampling.

I. Light Probe Sampling Algorithms:

These algorithm takes the environment light's texture map as input, then they create **a-given-number = pow(2, a-given-depth)** of distant lights. This is done by dissecting the original texture map into several rectangular regions. Each subregion is then represented by a distant light, with direction at the energy centroid of the subregion and the emitting light as the sum of the emitting light in the subregion. This can be more precisely written in the following formula.

$$\theta_c = \frac{\sum_{i\in r} L_i\theta_i}{\sum_{i\in r} L_i} \quad \phi_c = \frac{\sum_{i\in r} L_i\phi_i}{\sum_{i\in r} L_i} \quad \vec{E}_c = \sum_{i\in r} \vec{E}_i$$

- **Median cut algorithm**: The original region (**width** x **height**) is cut in a DFS manner. For each subregion, if it's depth < the given depth then I would try to cut it in half. Cutting width (correspond to **phi**) or height (**theta**) depends on their length, we choose the longer one to cut it. But the length of the width is illy defined (due to the curvy sphere), so in my implementation I use the length of the width corresponding to the mean **theta**. Then I would dissect the subregion in half by minimising the difference of the energy of the resulting two regions. The energy of a base region, a pixel, is defined to be the weighted mean of it's emitting light's RGB value (in my implementation, I used the **y()** defined in **pbrt**). The true emitting light of a pixel has to be scaled properly from the input texture map's RGB, which is done by scaling **sin(theta) x pi x 2pi / number_of_pixels**. The energy of a region is then simply the sum over all pixels. I used sum area table to speed up the calculation of the energy of a region, via the following formula.

$$\sum_{i=a}^{b}\sum_{j=c}^{d} L_{ij} = \sum_{i=0}^{b}\sum_{j=0}^{d} L_{ij} - \sum_{i=0}^{b}\sum_{j=0}^{c-1} L_{ij} - \sum_{i=0}^{a-1}\sum_{j=0}^{d} L_{ij} + \sum_{i=0}^{a-1}\sum_{j=0}^{c-1} L_{ij}$$

- **Variance minimisation algorithm**: Similar to median cut algorithm, it also dissect the region in a DFS manner. But for every region, we cut the region such that the maximum of the variance of the 2 resulting regions is minimised. And in my implementation, I do not decide whether to cut width or height at the beginning, it is purely determined by their minimised max-variance. The variance is defined as follows. The equation is based on the above definition for **theta_c** and

$$\sum_{i\in r} L_i((\phi_i-\phi_c)^2 + (\theta_i-\theta_c)^2) = \sum_{i\in r} L_i\phi_i^2 - \frac{(\sum_{i\in r} L_i\phi_i)^2}{\sum_{i\in r} L_i} + \sum_{i\in r} L_i\theta_i^2 - \frac{(\sum_{i\in r} L_i\theta_i)^2}{\sum_{i\in r} L_i}$$

  phi_c. From the right hand side of the equation, we can use five sum area table to speed up the calculation of the variance. We first pre-calculate the following five expressions. Then we can calculate the variance of any rectangular region in O(1) time.

$$\sum_{i=0}^{a}\sum_{j=0}^{c} L_{ij} \quad \left| \quad \sum_{i=0}^{a}\sum_{j=0}^{c} L_{ij}\theta_{ij} \right. \quad \left| \quad \sum_{i=0}^{a}\sum_{j=0}^{c} L_{ij}\theta_{ij}^2 \right. \quad \left| \quad \sum_{i=0}^{a}\sum_{j=0}^{c} L_{ij}\phi_{ij} \right. \quad \left| \quad \sum_{i=0}^{a}\sum_{j=0}^{c} L_{ij}\phi_{ij}^2 \right.$$

II. My Implementation in PBRT Functions:

- **IsDeltaLight**: Since we are transforming the environment light into a finite set of distant lights, we simply return **true** for this function.

- **Power**, **Sample_L(scene)**, **Pdf**: I have traced the code and found that although these functions are pure virtual functions, they are actually not called in our cases. Thus I have created these functions with a single **while(1)** inside of them.

- **Le**: This is not a pure virtual function, so we don't really have to implement this. This function takes a ray as input. This ray does not hit any scene object and will propagates to infinity. **Le** outputs the background colour for such ray. If we indeed transform the environment light into distant lights, then **Le** should return RGB = (0, 0, 0). But this would be very unnatural, so I left the original implementation, which uses a **MIPMAP** to query the background colour of that ray in the original environment map.

- **Constructor**: Contains a lot of creation, including MIPMAP used in **Le**, sum area table used for dissecting the region and the creation of the finite set of distant light using the algorithms described in the previous page.

- **Sample_L(point)**: I use **LightSample**'s **uComponent** to uniformly sample one of the distant light. The return ray is then simply determined. And we return the emitting light of the sampled light (since it is a sum over all pixels, which is very large, so I would then divide it by an arbitrary number, 10000 in my implementation), with pdf set to **1 / (# of distant lights)**. For median cut algorithm, the energy for different distant light is roughly the same, thus uniform sampling is fine. But in the variance minimisation algorithm, the energy may vary greatly for different distant lights, using importance sampling should be better. Nevertheless I still implemented uniform sampling due to its simplicity. Note that in variance minimisation, some region may even have total energy equal to 0 due to the unequal partition of energy.

III. Performance Comparison

The operating system for experiments is Mac OS. The processor is 4 GHz Intel Core i7 with memory 16GB 1600MHz DDR3. Cores detected by pbrt is 8 cores. Real time (in second) is shown.

	Median Cut	Variance Cut	PBRT Default
envlight-4	2.894	2.781	3.599
envlight-16	3.139	3.229	5.562
envlight-64	4.829	5.017	14.078
envlight-256	11.404	12.745	48.267
envlight-new-4	4.018	3.829	5.225
envlight-new-16	4.667	4.468	7.809
envlight-new-64	6.582	6.823	17.976
envlight-new-256	14.981	16.285	58.338

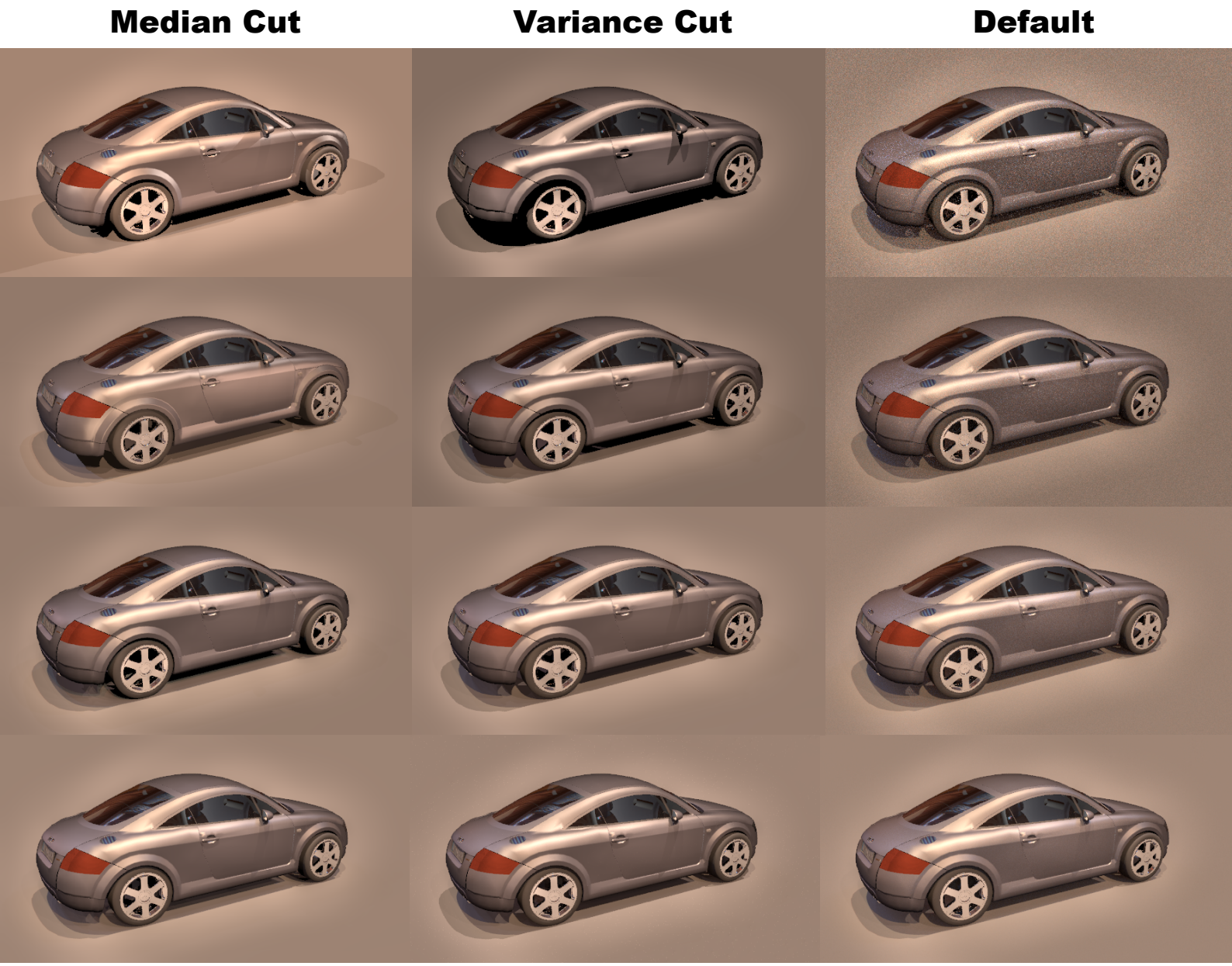
IV. My Results

The results shown here has been tone mapped using Photomatix in Photoshop with the default setting. The vertical order is the same as the table in the previous page. In general, variance cut is indeed better than median cut. This can be easily seen from envlight-256, where the reflection light on top of the car is obviously wrong for median cut, while variance cut is very close to the true answer. Nevertheless there are some weird behaviours that need clarifications.

1. When the distant light number is 4, there tend to have many black regions due to the lack of lights. The most noticeable one is envlight-4 using variance cut, which is almost completely dark. I think its simply because the 4 distant lights chosen by the variance cut algorithm are coincidentally all blocked in this scene. Since in envlight-new-4 using variance cut, we can still clearly see the car.

2. But why is the fuel filler cap, the wheels … etc., still visible in envlight-4 using variance cut? This is actually because of the **Le** function. After changing the environment light into a set of distant light, **Le** always has to return 0 (just as a distant light would do). But then the window and the filler cap would be completely dark for all the pictures, thus I use the default implementation which is actually slightly incompatible with our distant lights.

Median Cut **Variance Cut** **Default**


Median Cut
Variance Cut
Default