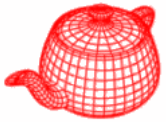
Homework #3
Environment Lights

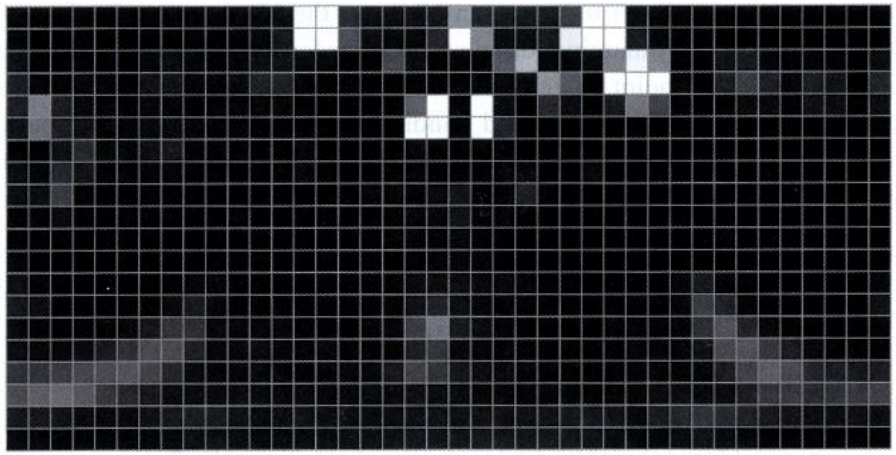
Digital Image Synthesis

Yu-Ting Wu

Project description



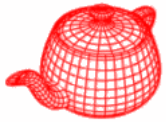
- Pbrt uses run-time importance sampling to render scenes with environment lights
  - The details will be given when talking about Monte Carlo integration



The brighter pixels (lights) have higher probability to be sampled

- In this homework, you are asked to provide an alternative approach

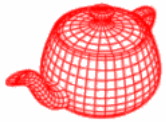
Project description



- Represent the environment map with a set of point (directional) lights

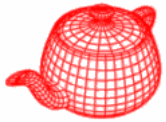
A light probe image is subdivided into 64 equal-energy regions. A point light is created for each region at its centroid.

Project description



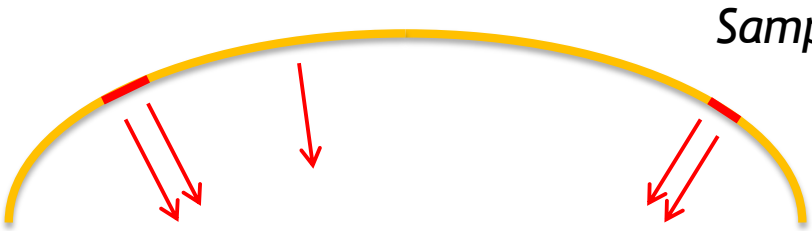
- Implement the “MedianCut” algorithm in pbrt
- There are three functions you should modify:
  - Constructor
    - Do MedianCut algorithm to generate a set of lights with roughly equal energy
  - Sample_L
    - When pbrt asks a sample from environment light, uniformly select one from all lights and return its direction, intensity, and PDF
  - IsDeltaLight
    - Tell pbrt whether this light can be sampled or not?
- For any other member functions, you can leave them alone at this time

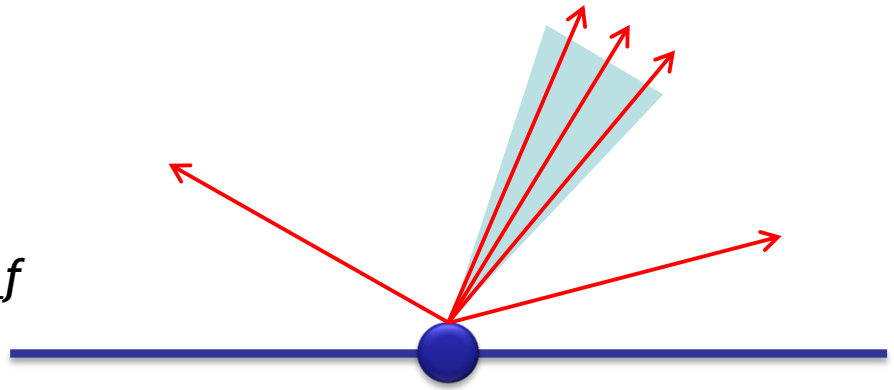
Importance sampling



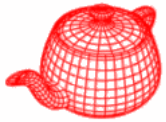
- In rendering, one important question is how to determine the sample distributions
- Importance sampling:
  - If the shape of sampling distribution is close to the shape of integrand, variance is reduced
  - In our case

$$f \propto L \cdot f_r$$

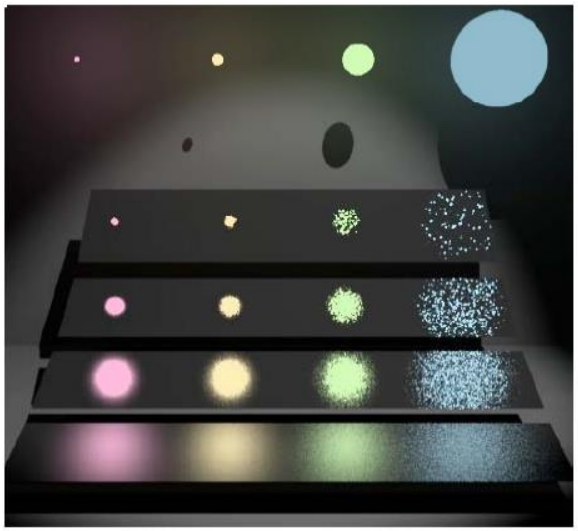




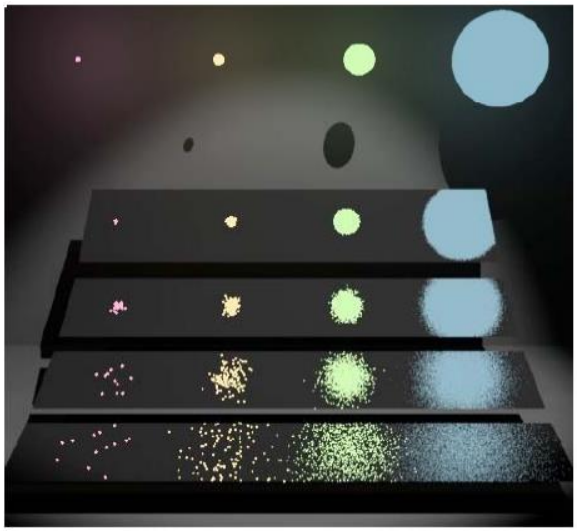
Multiple importance sampling



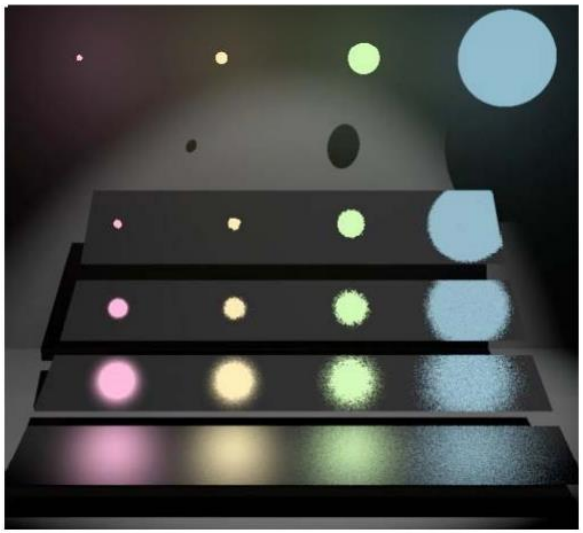
- Sample light or sample BRDF only cannot adapt to all configurations
- Multiple importance sampling:
  - Generate one sample using *Sample_L* and another using *Sample_f*, then combine the sample results
  - Better than either of the two strategies alone



Sample Light

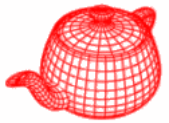


Sample BRDF



Multiple Importance Sampling

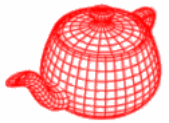
Multiple importance sampling



- Codes for estimating direct lighting in pbrt:

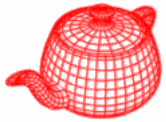
```
00109 Spectrum EstimateDirect(const Scene *scene, const Renderer *renderer,
00110         MemoryArena &arena, const Light *light, const Point &p,
00111         const Normal &n, const Vector &wo, float rayEpsilon, float time,
00112         const BSDF *bsdf, RNG &rng, const LightSample &lightSample,
00113         const BSDFSample &bsdfSample, BxDFType flags) {
00114     Spectrum Ld(0.);
00115     // Sample light source with multiple importance sampling
00116     Vector wi;
00117     float lightPdf, bsdfPdf;
00118     VisibilityTester visibility;
00119     Spectrum Li = light->Sample_L(p, rayEpsilon, lightSample, time,
00120                                   &wi, &lightPdf, &visibility);
00121     if (lightPdf > 0. && !Li.IsBlack()) {
00122         Spectrum f = bsdf->f(wo, wi, flags);
00123         if (!f.IsBlack() && visibility.Unoccluded(scene)) {
00124             // Add light's contribution to reflected radiance
00125             Li *= visibility.Transmittance(scene, renderer, NULL, rng, arena);
00126             if (light->IsDeltaLight())
00127                 Ld += f * Li * (AbsDot(wi, n) / lightPdf);
00128             else {
00129                 bsdfPdf = bsdf->Pdf(wo, wi, flags);
00130                 float weight = PowerHeuristic(1, lightPdf, 1, bsdfPdf);
00131                 Ld += f * Li * (AbsDot(wi, n) * weight / lightPdf);
00132             }
00133         }
00134     }
00135
```

Multiple importance sampling



```
00135
00136     // Sample BSDF with multiple importance sampling
00137     if (!light->IsDeltaLight()) {
00138         BxDFType sampledType;
00139         Spectrum f = bsdf->Sample_f(wo, &wi, bsdfSample, &bsdfPdf, flags,
00140                                     &sampledType);
00141         if (!f.IsBlack() && bsdfPdf > 0.) {
00142             float weight = 1.f;
00143             if (!(sampledType & BSDF_SPECULAR)) {
00144                 lightPdf = light->Pdf(p, wi);
00145                 if (lightPdf == 0.)
00146                     return Ld;
00147                 weight = PowerHeuristic(1, bsdfPdf, 1, lightPdf);
00148             }
00149             // Add light contribution from BSDF sampling
00150             Intersection lightIsect;
00151             Spectrum Li(0.f);
00152             RayDifferential ray(p, wi, rayEpsilon, INFINITY, time);
00153             if (scene->Intersect(ray, &lightIsect)) {
00154                 if (lightIsect.primitive->GetAreaLight() == light)
00155                     Li = lightIsect.Le(-wi);
00156             }
00157             else
00158                 Li = light->Le(ray);
00159             if (!Li.IsBlack()) {
00160                 Li *= renderer->Transmittance(scene, ray, NULL, rng, arena);
00161                 Ld += f * Li * AbsDot(wi, n) * weight / bsdfPdf;
00162             }
00163         }
00164     }
00165     return Ld;
00166 }
```

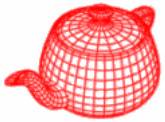
Multiple importance sampling



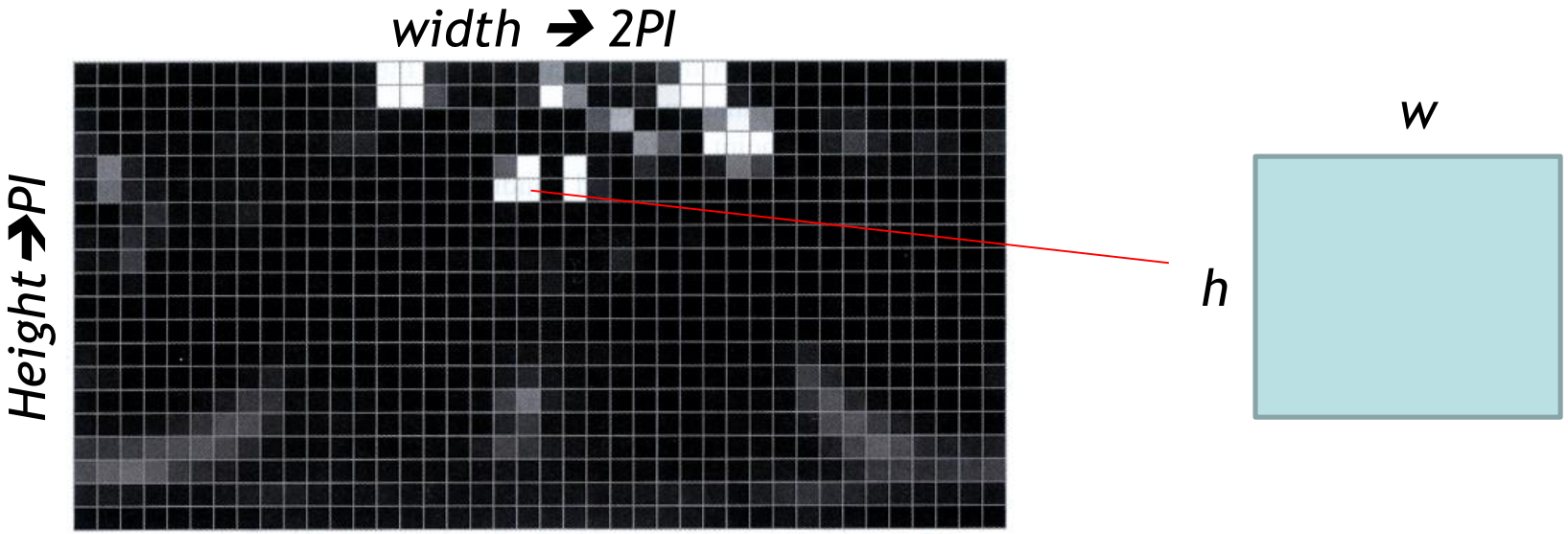
- IsDeltaLight
  - In homework #3, since we transform the environment to a set of directional lights which cannot be sampled, we should return true to avoid “BRDF importance sampling”

- Sample_L
  - Pbrt only asks one sample at a time.
  - In homework #3, all lights have roughly equal energy.
  - We can simply random choose one from *n* lights and return PDF with *1/n*

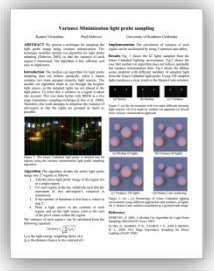
FAQ



- The rendered images have different radiance scale to the reference ones
  - Each pixel in the environment map represents an area on unit sphere
  - Remember to scale the light intensity with the areas (solid angles)

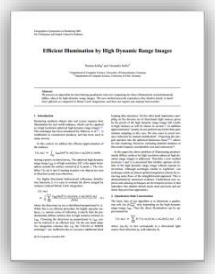


Related papers



Variance Minimization Light Probe Sampling
K. Viriyothai and P. Debevec
SIGGRAPH 2009 Poster

Structured Importance Sampling of Environment Maps
S. Agarwal, R. Ramamoorthi, S. Belongie, and H. W. Jensen
SIGGRAPH 2002



Efficient Illumination by High Dynamic Range Images
T. Kolling and A. Keller
EGSR 2003

Fast Hierarchical Importance Sampling with Blue Noise Properties
V. Ostromoukhov, C. Donohue, and P. –M. Hodoin
SIGGRAPH 2004