Scalable gradient-domain rendering

Vishesh Gupta

1. Introduction

There has been a considerate amount of research in trying to reduce the noise of a render in path tracing. Some methods look into providing a different integrating technique, while some are trying to use different path tracing strategies, some go as far as to reuse multiple path that have already been traced, there is also a method where we are taking the primal image and applying a reconstruction on it to create our final render. Since the color value at each pixel is a solution to a monte carlo estimator, there is bound to be a lot of noise in our renders.

Gradient Domain Rendering [LKL*13, KMA*15, MKA*15] is one such method that provide significant improvement on visual convergence by exploiting image-space smoothness and coherence in sensor path subspaces. That is it uses the gradient of an image as a intermediary to reconstruct the primal render into a final render, with its variance being the same as the gradient calculated.

We would like to introduce a new scalable method to generate our gradient image, which would require us to design a new shift mapping technique for gradient domain rendering. For the following project, we are trying to incorportate VPLs [HSA91], and stochastic light cuts [Yuk19] with reconnection shift mapping [BSH02] to help us get a better gradient with lower down time on the gradient path calculation.

We would begin our report by explaining different concepts used in our project one by one, building on top of each other to give us the basic knowledge needed to make what we want. starting with Light transport equation [Kaj86] describing the working of a light / camera path and how it interacts with the scene, then describing the data structure and use of VPLs [HSA91], followed by a small review of gradient domain rendering [HGP*19], and stochastic light cuts [Yuk19]. Finally we share our novel shift map and compare its results with different rendering algorithms.

2. Light Transport Equation

We need to formalize how rays going into the camera from the light source interacts with various types of objects to give us the color of the pixel. That's where the light transport equation comes [Kaj86] in, it tells us how to get the luminance of a ray that goes into the scene, while factoring in all possible parameters such as material properties, geometry, light emitted etc.

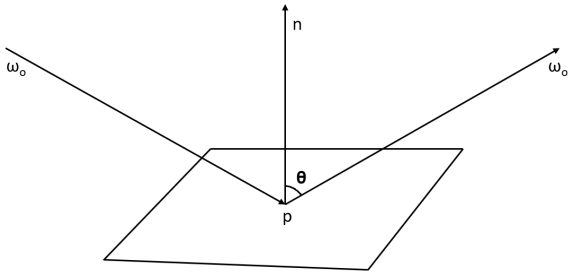

Figure 1: *Local interaction of the light on a surface with normal n*

$$L(p,\omega_o) = L_e(p,\omega_o) + \int_{S^2} f(p,\omega_o,\omega_i) L(t(p,\omega_i),-\omega_i) |\cos\theta_i| d\omega_i \quad (1)$$

Here, a ray coming from direction ω_o, hits a point p. L is the radiance that is going back to the ray from point p into the direction ω_o. L_e is the light emitted by the surface of point p, in the direction ω_o. Finally the integral is the summation of all radiance that is coming into point p, from all possible set of points in the scene. Letting ω_i be any direction from point p going into the scene, and $t(p,\omega_i)$ being the point on a surface that ray hits, if it propagates in that direction from that point. f is the material coefficient of the surface at which the ray hits it and leaves it, finally the cosine is the correction factor for light bouncing. In short the equation basically means that the light radiance for a ray going in direction ω_o, and hitting a point p, is given by the light emitted by the point, and the summation of all possible light from other surfaces that bounce off of point p, and go into the direction of our ray.

Due to the complexity and high dimensionality of the integral, it is impossible to solve this equation analytically. This is why we rely on Monte Carlo estimators to generate an es-

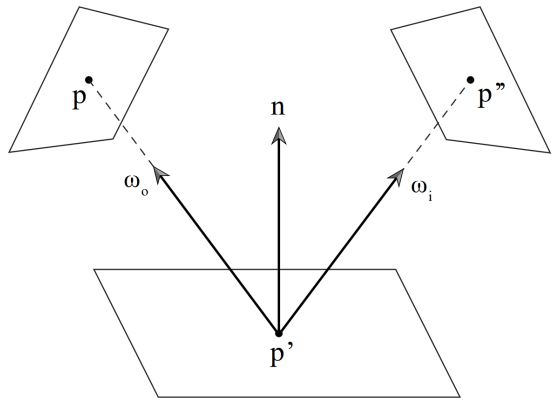

Figure 2: *interaction of a light path moving from point p towards point p" via point p' [PJH16]*

timate to the final value by taking multiple samples for each ray.

2.1. Monte Carlo

Taking the light transport equation without the self emitted illuminance, and we can solve the big integral by using Monte Carlo estimation, which gives us

$$\frac{1}{N}\sum_{k=1}^{N}\frac{f(p,\omega_o,\omega_k)L(t(p,\omega_k),-\omega_k)|\cos\theta_k|}{P(\omega_k)}$$

for point p, using N samples from probability distribution P over all possible points that are accessible from p.

2.2. Path Integral

In a lot of cases we would rather represent our LTE as a path integral instead of the normal integral [PJH16]. We will try to reform the equation to the surface form instead of angle form. We would shed some light on the working of the equation by making some behaviors of the function explicit in the integrand. We do this by rewriting LTE as an integral over area instead of an integral over directions on the sphere. First, we define exitant radiance from a point p' to a point p by $L(p' \rightarrow p) = L(p,\omega)$ as if p and p' are mutually visible $\omega = \widehat{p - p'}$ and the surface property function can be written as $f(p'' \rightarrow p' \rightarrow p) = f(p',\omega_o,\omega_i)$.

We also need to multiply by the Jacobian that relates solid angle to area in order to transform the LTE from an integral over direction to one over surface area, which is $|\cos\theta'|/r^2$, where θ' is the angle made by $-\omega_i$ and normal at p''. We will now combine this with the $\cos\theta$ term that was already there and create a new term

$$G(p \leftrightarrow p') = V(p \leftrightarrow p')\frac{|\cos\theta||\cos\theta'|}{\| p - p' \|^2}$$

where V is the visibility function, its either 1 or 0 depending if the points are mutually visible or not.

These lets us re-write LTE as below

$$L(p' \rightarrow p) = L_e(p' \rightarrow p) + \int_A f(p'' \rightarrow p' \rightarrow p)L(p'' \rightarrow p)G(p'' \leftrightarrow p')dA(p'') \quad (2)$$

Here, A is all possible surface area over all different points. This equation can be extended to volume by changing the domain and adapting the jacobian.

3. Virtual Points Lights

We can also break down the integration process into a two stage process. Here in the initial pass, we send out light paths from the light source into the scene and storing every all the interaction it makes with the scene. We call these interactions Virtual Point Lights (VPL) [HSA91] as they can act like individual point light sources. As for the second stage of the process we send paths from the camera into the scene and sample lights sources and VPLs to give us the final integral. This helps us get faster camera path propagation with lesser bounces while still achieving global illumination.

VPLs help us store a lightpath in the scene, which can be reused across pixels, making it really efficient in exchange for higher memory use. Later we will discuss ways in which we can improve light sampling of VPLs in our scene, converting its time complexity from linear time to sub linear.

As for VPLs, we produce them using light rays, and spawn / save VPLs at every diffuse bounce only, because storing. We only store our VPLs at diffused surface because for specular surfaces the re-connection probability of our shading point with that light path VPL will be zero.

4. Gradient Domain Rendering

The idea for gradient domain rendering comes from the fact that for natural scenes, we usually have a very sparse gradient. In gradient domain rendering, we estimate the primal image and calculate its gradient, often calculated at the same time, which we later use to reconstruct the image with the same variance of the gradient. But there is a catch, if we naively, calculate the gradient of the image by measuring the difference between two pixels color values, the gradient will have the same variance as the primal image hence no benefits can be reaped from this method.

To tackle this problem, we use the path correlation, between two pixels, to calculate a gradient with lower variance. We call this path correlation - shift mapping. A shift map is a transformation that deterministically transform a path from a given image-space location to another one. This is used to reduce the variance of the gradient of the image, which in turn makes the final image less noisy.

We design a shift map with the goal of reducing the difference between the radiance of primal path and shifted path,

which in turn results in lower variance of the gradient image. Lower variance gradient image will result on lower variance reconstructed image.

Following the derivation by [HGP*19] we let I_p represent the radiance of the light entering pixel p and I_q^p represent the shifted ray from pixel p to pixel q, where

$$I_p = \int_P h_p(x) f(x) dx \tag{3}$$

and

$$I_q^p = \int_P h_q(T_{pq}(x) f(T_{pq}(x)) | \frac{dT_{pq}(x)}{dx} | dx \tag{4}$$

where T_{pq} is a shift mapping function that transforms a path through pixel p (base path) into a path through pixel q (offset path). The Jacobian determinant $|\frac{dT_{pq}(x)}{dx}|$ accounts for the change of integration domain from p to q. As we estimate pixel q using paths from pixel p shifted exactly by one pixel, the reconstruction filter at pixel q can be defined as $h_q(T_{pq}(x)) = h_p(x)$.

As by design $T_{pq}(x)$ is supposed to be as close to x as possible, its difference represented by -

$$\Delta_{pq} = I_q^p - I_p = \int_P h_p(x)(f(T_{pq}(x)) | \frac{dT_{pq}(x)}{dx} | - f(x)) dx \tag{5}$$

Since the shift mapping function T_{pq} implies $T_{pq}(x) \approx x$, thus integrand in the equation becomes zero almost everywhere.

4.1. Shift Mapping

Now we go into a bit more into detail about shift mapping in this section. In simpler terms, shift mapping is a method to use an already existing path in our scene, and transform or "shift" a path that we want to create now, into a path that is similar to the path that is already there. We do this so that we can generate a set of paths with lower variance than two independent path generated normally, which is used here to generate a low variance gradient image. A few examples of existing shift mapping can be random sequence replay or path re-connection.

Random Sequence Replay: In this form of shift mapping, we store the sequence of random numbers used by the initial path that we generated, and we reuse the same sequence of random numbers to generate our shifted path [MKD*16]. By following this simple set of rules, we generate a path that is co-related with our initial path with a different image space location.

Path Reconnection: In this form of shift mapping, we store the points of possible reconnection in our initially generated path, and when generating our shifted path, we try to reconnect our shifted path to the starting path [BSH02]. This helps us reuse the data from our primal path, while also helping us co-relate the two paths.

5. Lightcuts

In most systems, rendering cost is increases linearly with the number of lights. The lightcuts framework [WFA*05] provides a quick way to approximate the illumination from a group of lights, and puts cheap and reasonably tight bounds on the maximum error in doing so. It also uses an adaptive method for partitioning the lights into groups by creating a light tree to control the tradeoff between the cost and error. All this leads to the rendering cost strongly scaling sublinearly with the number of point lights in the scene while maintaining perceptual fidelity.

Given a set of point light sources S, the radiance L caused by their direct illumination at the surface point x viewed from a direction ω is a product of each light's material, geometry, and intensity terms summed all over the light

$$L_S = \sum_{i \in S} M_i(x,\omega) G_i(x,\omega) V_i(x,\omega) I_i \tag{6}$$

Let us define a cluster, $C \subseteq S$, to be a set of point lights along with a representative light $j \in C$. The direct illumination from a cluster can be approximated by using the representative light's material, geometric, and visibility terms for all the lights to get:

$$L_C = \sum_{i \in C} M_i(x,\omega) G_i(x,\omega) V_i(x,\omega) I_i \tag{7}$$

$$\approx M_j(x,\omega) G_j(x,\omega) V_j(x,\omega) \sum_{i \in C} I_i \tag{8}$$

The amount of cluster error will depend on how similar the material, geometric, and visibility terms are across the cluster.

In lightcuts we first take the list of all the point lights, store them in a binary tree (construction discussed below) based on their spatial coordinates, and for each shading point we get a different tree which we then traverse to generate a cut. These cuts are basically a group of nodes in the light tree, with each node having a representative for all the lights under it. We keep the node if the error percentage of the representative light is under a certain mark otherwise, we repeat the process with its child nodes.

5.1. Tree

A light tree serves as the foundation of our approach. It's a binary tree where leaves represent individual lights and internal nodes represent clusters containing lights beneath them in the hierarchy. A cut through the tree is defined as a set of nodes where any path from the root to a leaf must pass through precisely one node in the chosen set. This implies that each cut translates to a valid partitioning of the lights into individual clusters.

The light tree facilitates the grouping of point lights into clusters. Ideally, we strive to optimize the quality of these

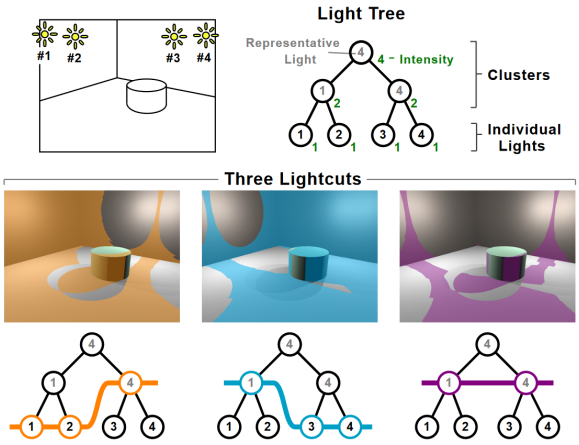

Figure 3: *A light tree and three example cuts. The tree is shown on the top with the representative lights and cluster intensities for each node. Leaves are individual lights while upper nodes are progressively larger light clusters. Each cut is a different partitioning of the lights into clusters. Above each cut, the regions where its error is small are highlighted [WFA*05].*

generated clusters, aiming to combine lights that exhibit the highest degree of similarity in their material properties, geometry, and visibility characteristics. To achieve this, we employ an approximation technique that groups lights based on their spatial proximity and similar orientation.

Construction of the light tree involves taking the set of virtual point lights (VPLs) and arranging them based on their spatial coordinates. For each pair of nodes starting from the beginning, a new parent node is created, linking the two children beneath it. This node creation process is iteratively repeated over the newly formed layers of nodes until only a single node remains at the top, representing the tree's root.

5.2. Choose a cut

Using a relative error criterion for cluster acceptability requires prior knowledge of the total radiance. To circumvent this dependency, we begin with a coarse cut, typically the root node of the light tree, and progressively refine it until our desired error threshold is met.

For each node in the current cut, we compute both its approximate lighting contribution and an upper bound on its potential error. Each refinement step focuses on the node with the largest error bound. If this bound exceeds a specified error ratio multiplied by the current total radiance estimate, we remove it from the cut. We then replace it with its two child nodes from the light tree, calculate their respective lighting contributions and error bounds, and update the total radiance estimate. If the resulting bound satisfies our error criterion, the cut is considered a lightcut, and the process concludes.

To improve efficiency, we impose a constraint: the representative light of a cluster must be the same as one of its children. This eliminates the need to compute the cluster estimate and visibility for one child, saving valuable processing time, especially when calculating visibility is computationally expensive. Additionally, we utilize a heap data structure to efficiently identify the cluster node with the highest error bound within the cut. Finally, individual lights (leaf nodes in the light tree) are considered perfectly accurate and assigned a zero error during the process.

```
cutList = {};
cutList.push_back(rootNode);
while(true) {
    if(cutList.size() > cutSize){
        break;
    }
    //Sort cutList based on
    //Error of each node
    if(cutList[0].error > errorBound){
        //Add child of the node
        continue;
    }
    else{
    break;
    }
}
```

6. Stochastic Lightcuts

Stochastic lightcuts [Yuk19] builds up on lightcuts as it that incorporates stochastic sampling into the illumination estimation framework of lightcuts. As a result, it remove the sampling correlation from the lighting estimation. It also uses a robust hierarchical sampling strategy to minimize the stochastic sampling noise.

This correlation arises because of the representative lights in lightcuts, ignoring the representative light in the tree can solve this issue. Instead, we randomly pick a light source within a given subtree. Light source in the subtree can be sampled using any importance sampling scheme. Thus, when we move deeper into the light tree, the light estimation (including shadow computation) for the parent node is not wasted, similar to the original lightcuts method. Three significant advantages come from this modification:

1. It completely removes the sampling correlation by substituting a random order for the predetermined arrangement of lights.
2. It allows using any type of light source.
3. Without risking the danger of excessive correlation artifacts, we can limit the number of lights assessed during lighting estimation.

6.1. Hierarchical Importance Sampling

As mentioned above we can use any Importance Sampling scheme to get consistent and converging unbiased estimates. However, we require our sampling probability to be proportional to the illumination contribution of each light source for faster convergence in lighting estimation. Similar phenomenon can be seen in multiple importance sampling if done incorrectly.

In Stochastic Lightcuts [Yuk19], instead of directly sampling a random light within a subtree, we employ a step-by-step traversal until reaching a leaf node. We start with the root node and randomly pick one of its child nodes at each step using importance sampling.

To estimate the lighting at a shading point x we represent the probabilities of selecting either one of the child nodes as p_1 and p_2. Following the error estimation mechanism of lightcuts, we assign importance weights w_1 and w_2 for picking the child nodes based on a metric that maximises the possible illumination coming from each node. The corresponding probabilities are defined as $p_1 = w_1/(w_1 + w_2)$ and $p_2 = w_2/(w_1 + w_2)$.

7. Results

For our experiment we chose 840,000 lightrays to create the VPLs in our scene with upto 5 bounces each, for stochastic lightcuts the cut size was chosen to be 10, and 64 samples per pixel for Gradient domain with stochastic lightcut algorithm (GradSLC). After GradSLC image was produce we would generate images for the same time, using Stochastic lightcut algorithm (SLC) and simple path tracing (SPT), to compare it with our algorithm.

It can be seen that in our SLC and GradSLC images, when there is a presence of glossy material there is a loss of energy caused by improper generation of VPLs and linking the rays to the VPLs. Also we can see fireflies in SLC images due to edges which make the reconnection value of the VPL really high. These fireflies result in fireflies in the gradient domain images, which inturn during reconstruction cause spots on the final GradSLC image as can be seen in the figures.

We can improve the image by also increasing the amount of initial lightrays for VPL generation but we are facing segmentation fault while creating the light tree for the following, which we think can be tackled by more memory efficient tree creation, and debugging.

As with increasing the cut size the image becomes a bit more clearer for much higher processing times.

We would also like to try out a different reconstruction algorithm to see if it helps us get a better result for GradSLC, and look more into the VPL connection and shift mapping to get a better result for our image.

References

[BSH02] BEKAERT P., SBERT M., HALTON J.: Accelerating path tracing by re-using paths. doi:10/ggdwkn. 1, 3

[HGP*19] HUA B.-S., GRUSON A., PETITJEAN V., ZWICKER M., NOWROUZEZAHRAI D., EISEMANN E., HACHISUKA T.: A survey on gradient-domain rendering. 455–472. doi:10/ggd8m5. 1, 3

[HSA91] HANRAHAN P., SALZMAN D., AUPPERLE L.: Instant radiosity. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques (SIGGRAPH)* (1991), pp. 27–36. 1, 2

[Kaj86] KAJIYA J. T.: The rendering equation. 143–150. doi:10/cvf53j. 1

[KMA*15] KETTUNEN M., MANZI M., AITTALA M., LEHTINEN J., DURAND F., ZWICKER M.: Gradient-domain path tracing. 123. doi:10/gfzrhn. 1

[LKL*13] LEHTINEN J., KARRAS T., LAINE S., AITTALA M., DURAND F., AILA T.: Gradient-domain metropolis light transport. *ACM Trans. Graph. 32*, 4 (2013). 1

[MKA*15] MANZI M., KETTUNEN M., AITTALA M., LEHTINEN J., DURAND F., ZWICKER M.: Gradient-domain bidirectional path tracing. doi:10/gf2hcw. 1

[MKD*16] MANZI M., KETTUNEN M., DURAND F., ZWICKER M., LEHTINEN J.: Temporal gradient-domain path tracing. doi:10/f9cpsw. 3

[PJH16] PHARR M., JAKOB W., HUMPHREYS G.: *Physically Based Rendering: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., 2016. 2

[WFA*05] WALTER B., FERNANDEZ S., ARBREE A., BALA K., DONIKIAN M., GREENBERG D. P.: Lightcuts: A scalable approach to illumination. 1098–1107. doi:10/dhp5d3. 3, 4

[Yuk19] YUKSEL C.: Stochastic lightcuts. Steinberger M., Foley T., (Eds.). doi:10/ggjcgw. 1, 4, 5

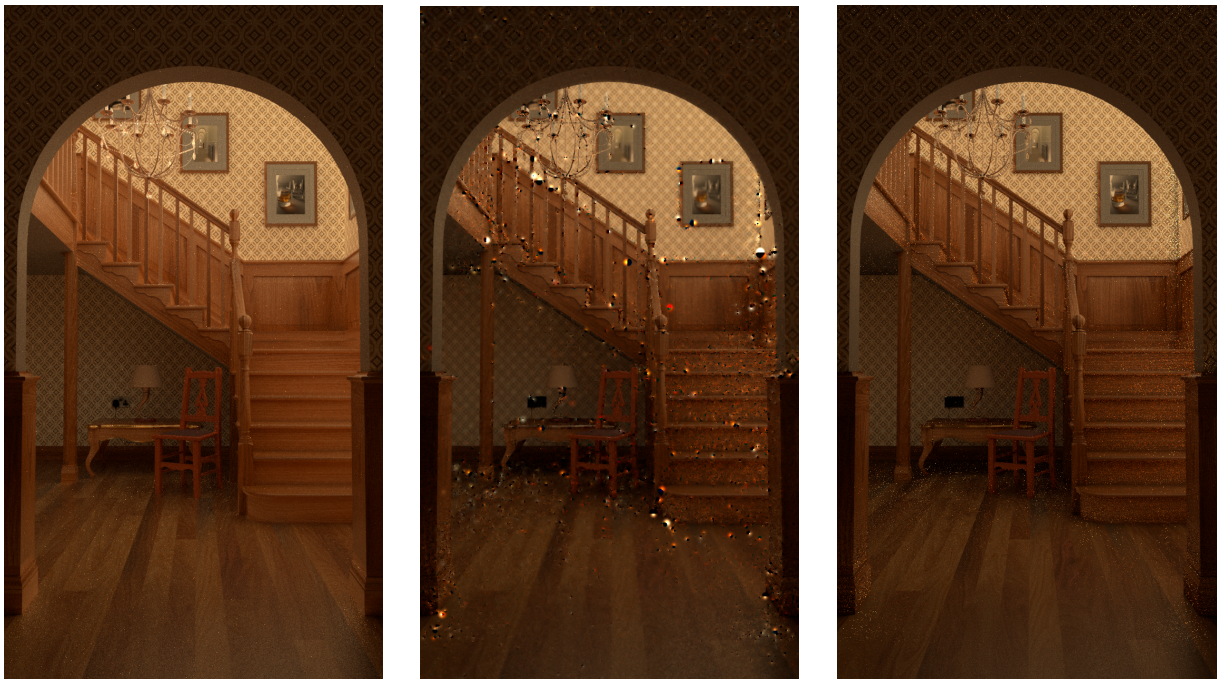

Figure 4: *Stair case scene a. Simple Path tracer only indirect lighting 320spp - 1000s, b. Gradient Domain with stochastic lightcuts only indirect lighting 64spp - 1001s, and c. Stochastic Lightcuts only indirect lighting 128spp, 1001s*

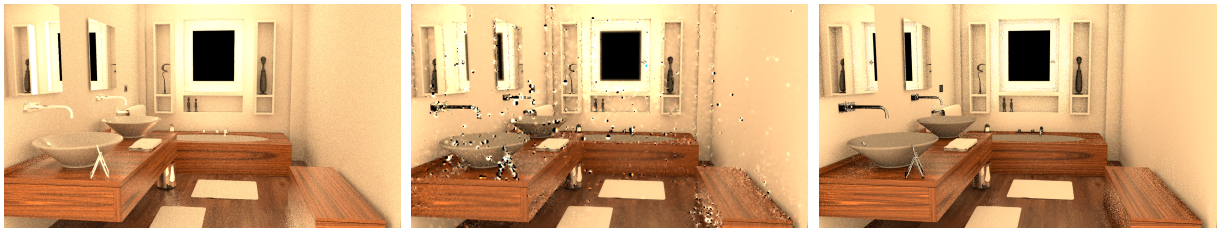

Figure 5: *Bathroom scene a. Simple Path tracer only indirect lighting 100spp - 680s, b. Gradient Domain with stochastic lightcuts only indirect lighting 64spp - 687s, and c. Stochastic Lightcuts only indirect lighting 128spp - 688s*

Figure 6: *House scene a. Simple Path tracer only indirect lighting 200spp - 280s, b. Gradient Domain with stochastic lightcuts only indirect lighting 64spp - 280s, and c. Stochastic Lightcuts only indirect lighting 150spp - 280s*

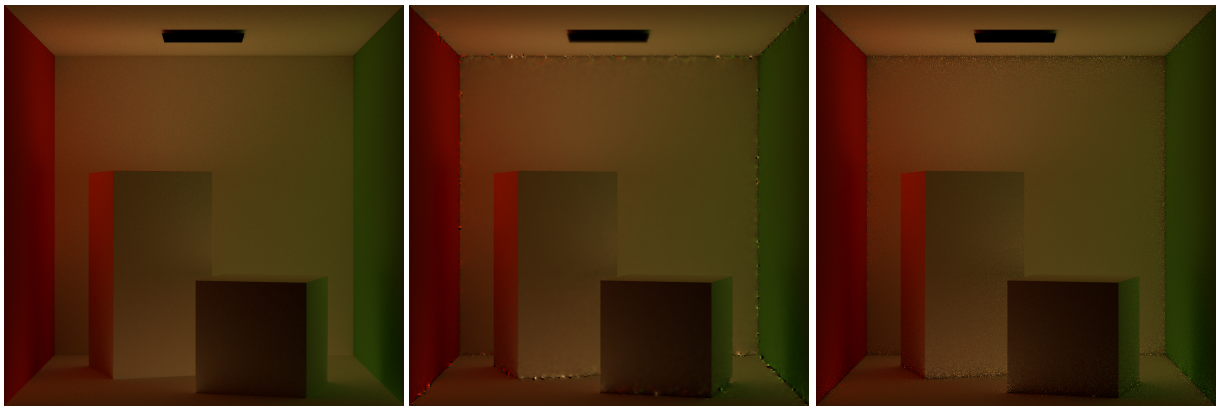

Figure 7: *Cornell Box scene a. Simple Path tracer only indirect lighting 300spp - 620s, b. Gradient Domain with stochastic lightcuts only indirect lighting 64spp - 620s, and c. Stochastic Lightcuts only indirect lighting 128spp - 620s*

Figure 8: *Lamp scene a. Simple Path tracer only indirect lighting 1650spp - 870s, b. Gradient Domain with stochastic lightcuts only indirect lighting 64spp - 865s, and c. Stochastic Lightcuts only indirect lighting 160spp - 870s*