

Implementing Energy Redistribution Path Tracing

Christopher Batty

Department of Computer Science
The University of British Columbia

Abstract

Energy redistribution path tracing is a recent approach to performing global illumination that combines basic Monte Carlo path tracing with elements of Metropolis Light Transport in order to achieve the advantages of each. The result is an algorithm with reduced variance, better stratification, and the ability to handle direct lighting, indirect lighting, and caustics with a single method. It is also conceptually simpler than Metropolis, and eliminates several of the disadvantages of traditional MLT, such as start-up bias and poor convergence for direct illumination. This project describes the details and results of an implementation of ERPT within the framework provided by the freely available, educational renderer, PBRT.

1 Introduction

The problem of global illumination is a fundamental topic in computer graphics, and has received a great deal of attention over the past two decades. It focuses on the rendering of photorealistic images by computing an approximation of all the light paths in a scene. This is a difficult integration problem that is typically solved using numerical integration methods, such as Monte Carlo and Metropolis.

When Monte Carlo techniques are applied to this problem the result is the Monte Carlo path tracing algorithm, which generates random light paths, evaluates the radiance along each path, and weights them by the probability of each path being generated [Kajiya 1986]. This is the most common and also the most flexible global illumination algorithm. However, there are common cases where path tracing is less than ideal.

One difficult case is indirect illumination, which accounts for light that has hit a surface and scattered one or more times, producing effects such as colour bleeding. Because paths are created starting from the eye, it may be hard to generate paths that terminate at a light source, such as if the camera is in one room, and a light source is placed behind a partially closed door in an adjoining room. A second issue is the effect of caustics, generated by specular surfaces focusing light, and creating interesting patterns. This is also challenging for the forward path tracer, since it requires a path to be generated that hits a diffuse surface, and then is scattered in just the right direction for it to hit a caustic sur-

face and reflect towards a light source. The path tracer has no way of knowing in advance that this path is potentially important.

One major flaw in path tracing is that it fails to take advantage of correlation between paths. That is, for any two nearby points being rendered, they are likely to share a great deal of “information” in common. As a result, many unimportant paths are generated even though their contribution to the image should be minimal. This means that there is high variance in the samples taken by the path tracer, and this variance manifests as unsightly noise in the final render. The energy redistribution path tracing algorithm [Cline et al. 2005] seeks to exploit correlation between paths to reduce variance and allow better handling of the types of complex lighting situations described above.

2 Previous Work

Several possibilities have been explored in attempting to improve on basic path tracing. The most intuitively clear is bidirectional path tracing [Lafortune and Willems 1993; Veach and Guibas 1994], which generates sub-paths starting from both the eye and the light, and connects the two in the middle. In cases where a random path is unlikely to reach the light source, this can be helpful in sampling useful paths.

Another approach to reducing variance is known as importance sampling [Veach and Guibas 1995; Lawrence et al. 2004], and takes advantage of the flexibility provided by Monte Carlo sampling in choosing the probability distribution function of samples being generated. The idea is to generate samples using a distribution that attempts to approximate the actual function as closely as possible. This focuses sampling where the function is expected to be large, thus reducing variance.

Several “biased” algorithms use either caching or interpolation to estimate lighting for nearby pixels, which reduces variance and accelerates convergence. However, the unfortunate side effect is that they do not necessarily converge to the ideal, correct solution. Examples include irradiance caching [Ward et al. 1988], photon mapping [Jensen 1996], and density estimation [Shirley et al. 1995].

When Metropolis sampling approaches are applied to the rendering equation, the result is the Metropolis Light Transport algorithm [Veach and Guibas 1997]. It generates a single path, and then mutates it repeatedly to explore all of path

space. Mutations are accepted or rejected based on a rule that enforces “detailed balance,” ie. the concept that flow from one path sample to another should be equal to the reverse, ensuring that the stationary solution will eventually be reached. As mutations occur, a histogram records where the path mutates to, and the end result is proportional to the integral required by the rendering equation. Energy redistribution builds directly on these ideas.

Though not directly relevant to this project, there has been much additional work on MLT since its introduction. This includes adding support for participating media, [Pauly et al. 2000], an analysis of its statistical properties, [Ashikhmin et al. 2001], simplifications and speedups, [Kelemen et al. 2002], and an analysis of the start-up bias issue [Szirmay-Kalos et al. 1999].

3 Concepts

3.1 Energy Flow

Monte Carlo path tracing generates samples to compute an approximation of the integral at a given point in an image. If $f(x)$ is the value of a path x , and $p(x)$ is the probability of that path being generated for a given probability distribution, a sample has the value

$$X_f(x) = \frac{f(x)}{p(x)}$$

For each image pixel numerous paths are generated and evaluated in this way to produce an unbiased estimate of the integrals forming a complete image.

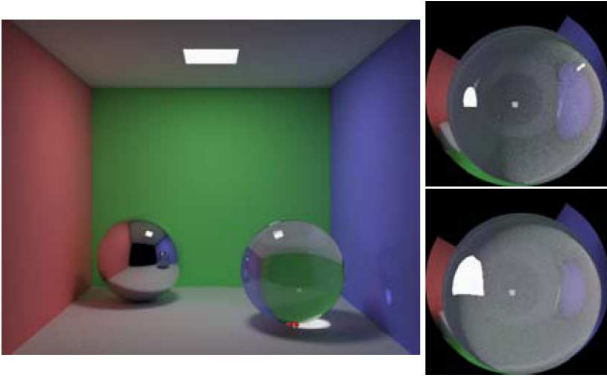


Figure 1: An example of correlation in paths. The above scene is rendered from the points of view of the two red dots in the image. The renders on the right illustrate that these points share a great deal of information, ie. they are highly correlated.

As described above, path tracing discards a great deal of information by generating an entirely new path on each iteration. If it could be modified to exploit the correlation between nearby paths, variance could be substantially reduced.

This is what energy redistribution does, by taking each path and mutating it to spread its sample energy around to nearby, closely related paths. At each mutated path in a long chain, it deposits some fraction of the original sample energy. To understand why this works, we must consider the details of energy flow. Energy flow is the idea of allowing energy to be transferred directly between related samples. The creation of a connection between two samples is the role performed by a mutation in the ERPT algorithm. The expected flow from one sample to another is given by:

$$E[\phi(x \rightarrow y)] = E[X_f(x)p(x)T(x \rightarrow y)q(x \rightarrow y)]$$

where x and y are two correlated samples, $\phi(x \rightarrow y)$ indicates flow from x to y , T is the transition function which gives the probability of a mutation from x to y occurring, and q dictates the fraction of the energy at x flowing to y , if such a mutation occurs.

The key to ensuring that such a strategy remains unbiased is satisfying a property known as “general balance”. This is the property of a mutation strategy that ensures it will converge to the stationary distribution, by requiring that total flow into a given point equals the total flow out of that point. If we consider an energy distribution that is already at the stationary solution, it is clear that general balance will ensure that it remains there. Similarly, enforcing general balance causes a non-stationary distribution to converge to the correct solution. A more restrictive requirement that inherently satisfies general balance is “detailed balance” which says that the flow from a point x to another point y must equal the flow from y to x . By enforcing detailed balance, ERPT can be made to be unbiased.

The mutation acceptance criterion used by MLT (which ERPT also adopts) that enforces detailed balance is:

$$a(x \rightarrow y) = \min \left(1, \frac{f(y)T(y \rightarrow x)}{f(x)T(x \rightarrow y)} \right)$$

where f is the radiance value of a path and T is again the transition probability. Taking the minimum of the two terms causes all mutations in one direction to be accepted, while only some in the opposing direction are accepted. This is done to increase the number of accepted mutations and speed convergence.

3.2 Mutations and Path Density

There are two types of mutations used by the algorithm. Caustic mutations perturb the outgoing direction from a light source or second diffuse vertex in a path. The modified path is then followed back through specular bounces until the diffuse vertex preceding the eye is reached. This point is then projected onto the image plane to get the new sample location. This mutation allows for exploration of paths that typically give large contributions to the interesting caustics generated by specular surfaces.

The second type is referred to as a lens perturbation in [Cline et al. 2005], but is actually a combination of lens and multi-chain perturbation. The image plane coordinates of an existing path are perturbed slightly, and a new ray is shot from that point. This path is then followed through the same types of bounces as before, perturbing the bounce direction at each diffuse vertex that is followed by a specular surface. When a diffuse vertex followed by a diffuse vertex is reached, the mutated path is connected directly up to the old path. Lens, multi-chain and caustic mutations are described in further detail in [Cline and Egbert 2005; Veach and Guibas 1997; Veach 1997].

In order to apply these mutations, we must compute the change between the probabilities of generating the existing and mutated paths with our path tracer. A small set of five rules determines how these “path densities” are computed change for each component of a mutation. For example, for perturbing the pixel coordinates of a sample, as done in a lens mutation, there is no change in density. Specific, easily implementable formulas are given in [Cline et al. 2005] for the remaining four cases, including connecting a path to the eye, and perturbing directions of light bounces. The only difficulty is knowing exactly when and how to apply these rules, which will be discussed in section 4.2.

4 Implementation

4.1 Integration with PBRT

I decided to add ERPT to the PBRT renderer [Pharr and Humphreys 2004], so that I would have time to focus on the details of the ERPT algorithm implementation, rather than the low-level minutiae inherent in creating a renderer from scratch. PBRT provides mechanisms for ray-shooting and intersection detection, sampling and filtering, shape representations, image output, several integration techniques, and a variety of other features necessary for successful rendering.

My initial hope was that ERPT could be implemented as a plugin, so that it might be distributed independently and used in the same fashion as the other integrators already provided. However, PBRT’s core rendering loop assumes that the integrator creates a single path and returns a single sample value to be added at that pixel. In contrast, ERPT generates a sample, and then deposits fractions of it at numerous points in a mutation chain. As a result, I was forced to modify the files in the core of PBRT. The main code for the implementation is in *scene.cpp*, which contains the main rendering loop, and *erpt.cpp*, which provides the path creation, mutation and evaluation routines.

For the path generation and evaluation component, I began with the standard PBRT path tracing implementation, *path.cpp*. This code performs tracing and evaluation in a single pass, and it sums the energy of all sub-paths of the computed path as well, improving convergence at the expense of biasing the result. I removed this latter feature,

and separated creation and evaluation into distinct functions operating on an array of vertices containing the path data (vertex location, incoming/outgoing direction vectors, specular/diffuse flags, etc). Next I created a *mutatePath* function that detects the path type and performs a mutation accordingly. Note that caustic mutations are possible only for paths of the form $L(S|D)*SDE$. I choose randomly between the two mutation types whenever both are applicable, and in other cases simply apply the appropriate. Once I thoroughly understood PBRT’s path tracing, writing code to perform these mutations was relatively straightforward, albeit slightly time-consuming and error-prone.

4.2 Issues and Clarifications

Although it isn’t terribly complicated, in the interests of clarity I present the exact algorithm I used for estimating the deposition energy, in algorithm 1.

Algorithm 1 Deposition Energy Estimation

```

function COMPUTED
  energy = (0, 0, 0)
  count = 0
  for each pixel in image do
    create a path x in the current pixel.
     $X_f(x) = f(x)/p(x)$ 
    energy = energy +  $X_f(x)$ 
    count ++
  end for
   $e_d = \text{luminance}(\text{energy}) / (\text{count} * \text{mutations})$ 
end function

```

One element of the original paper that is somewhat unclear is how to compute the value of the mutation acceptance function q in practice. Due to the organization of the original paper, it is implied that we must compute all the terms in:

$$q(x \rightarrow y) = \min \left(1, \frac{X_f(y)p(y)T(y \rightarrow x)}{X_f(x)p(x)T(x \rightarrow y)} \right)$$

However, since $X_f(x)p(x) = f(x)$, we just need to compute:

$$q(x \rightarrow y) = \min \left(1, \frac{f(y)T(y \rightarrow x)}{f(x)T(x \rightarrow y)} \right)$$

Furthermore, it is never explicitly stated exactly how the path density change from x to y , here denoted $\Delta \text{density}(x \rightarrow y)$, is used in computing the acceptance. Careful reading of [Cline and Egbert 2005] indicates that the following is the correct interpretation. Path density change computed using the supplied rules is equivalent to:

$$\Delta \text{density}(x \rightarrow y) = \frac{T(y \rightarrow x)}{T(x \rightarrow y)}$$

and recalling that we are actually dealing with luminance of the evaluated paths, we have:

$$q(x \rightarrow y) = \min \left(1, \frac{\text{lum}(f(y))}{\text{lum}(f(x))} \Delta \text{dens}(x \rightarrow y) \right)$$

Thus to find q we just need to compute the luminance of the current and mutated paths, and the path density change between the two.

A second issue I had trouble with was understanding how to apply the equal deposition flow rule, in terms of what exactly was deposited at each point. In practice it is very simple. If $e(r, g, b)$ is the rgb triple from the original sample location, and e_d is the scalar deposition energy computed as in 1, we deposit $depositionValue(r, g, b) = e/lum(e) * e_d$. The first way to look at this is that we are normalizing the colour given by the initial sample, and scaling by e_d to ensure we deposit the correct total energy. The other way to look at it is that the ratio of $lum(e) : e_d$ should equal the ratio of $e : depositionValue(r, g, b)$. This is valid because luminance is a linear function of the individual components of the spectrum.

Both of these details are spelled out explicitly in the modified version of the equal deposition flow algorithm, given in Algorithm 2.

Algorithm 2 Equal Deposition Flow

```

function EQUALDEPOSITIONFLOW( $x, e(r, g, b), m, e_d$ )
   $numChains = \lfloor random(0, 1) + lum(e)/(m * e_d) \rfloor$ 
   $depVal(r, g, b) = e/lum(e) * e_d/samplesPerPixel$ 
  for  $i = 1$  to  $numChains$  do
     $y = x$ 
    for  $j = 1$  to  $m$  do
       $(z, \Delta density(x \rightarrow y)) = mutate(y)$ 
       $lfz = lum(f(z))$ 
       $lfy = lum(f(y))$ 
       $q = lfz/lfy * \Delta density(x \rightarrow y)$ 
      if  $q >= random(0, 1)$  then
         $y = z$ 
      end if
      deposit  $depVal$  at  $y$ 
    end for
  end for
end function

```

I also implemented the second of the two filters described in the paper. ‘‘Consecutive sample’’ filtering reduces bright spots that occur as a result of a mutation chain getting ‘‘stuck’’ on the same path for a very large number of iterations. It does this by counting the number of iterations where a mutation fails, and if it exceeds a threshold of 10 or 20, it ceases to deposit energy at that sample until it mutates away from that point.

Another issue that the paper doesn’t go into is how to ensure that the energy deposition works correctly in the presence of an image filter being used for depositing samples. If the samples deposited at a point are averaged (as they normally would be) rather than summed (as required by ERPT) all the pixels in the image end up having a luminance the same as the deposition energy, which naturally gives a very poor result. The correct behaviour is for samples being de-

posited to simply be accumulated and not averaged, so that pixels that receive more energy depositions correctly become brighter. I also found it necessary to divide the deposition energy by the number of samples per pixel in the image, so the final sum works out to be the same as the desired average energy. (ie. if we don’t perform this division the image simply gets brighter and brighter as more samples per pixel are deposited.)

Although the original paper provides an appendix outlining how rules are used in two example paths, their caustic mutation fails to include any specular surfaces, despite the fact that they themselves do not apply caustic mutations to this type of path. Therefore I provide a full description of how each rule is applied in the renderer.

For a lens perturbation, when the pixel coordinates are adjusted on the image plane, we apply rule 1, which states that the path density does not change. When directions are perturbed outgoing from a diffuse surface, we apply rule 2, multiplying path density by the the ratio of two cosine values. Finally when we hit two consecutive diffuse vertices, we reconnect with the old path, and apply the formula from rule 3. If the path continues all the way to a light source, we apply rule 4, for connecting to lights. For an implicit path this implies again applying rule 3, and for explicit paths, we do nothing. We did not mutate paths of the form LE.

For a caustic perturbation, we apply rule 5 for connecting the final point back to the eye. This was the only rule that appeared to be necessary. The only other rule that might be applied is rule 4, as in the example in the appendix, but in our case these paths were implicit implying application of rule 3. However, rule 3 applies only to connecting diffuse vertices, and here we are connecting a specular vertex to the light (or diffuse vertex), so it doesn’t appear to apply. The results indicate that this is the correct choice.

5 Results

As demonstrated by the following collection of Cornell box images, the algorithm appears to work correctly, although it is still substantially slower than the render times quoted in [Cline et al. 2005]. This is at least partly because they performed renders on a 3.4GHz P4 Desktop, while all renders in this project were performed on a 2.4GHz Compaq laptop with 512MB of RAM.

The two images of mostly indirect lighting in figures 2 and 3 were rendered in approximately 2 hours each. The basic MC path tracing image used 1400 samples per pixel, while the ERPT image used 100 mutations per chain, with 24 samples per pixel. It is clear that the path tracing image produces much worse speckled noise, and ERPT produces much smoother images. However, ERPT has some problems with unevenness in the results, which would likely require a larger number of mutations to resolve. When the number of paths is relatively low, ERPT images possess a large degree of blotchiness and bright spots. Even for the multi-hour

Figure 2: MC path tracing with 1400 samples per pixel.

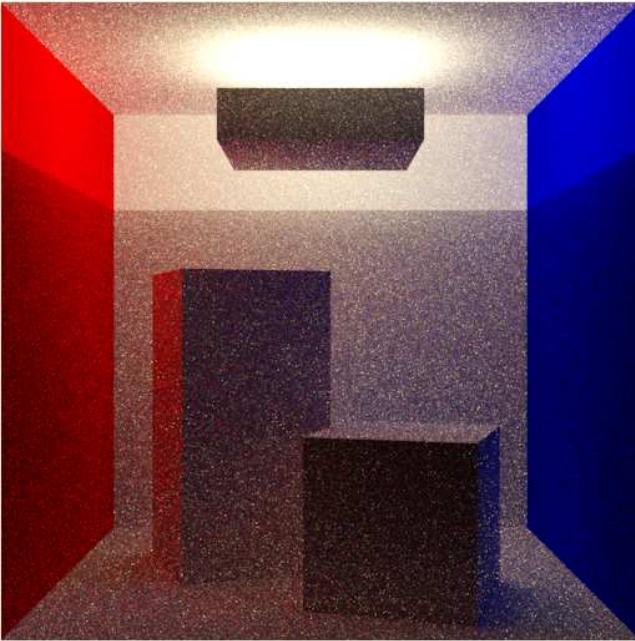
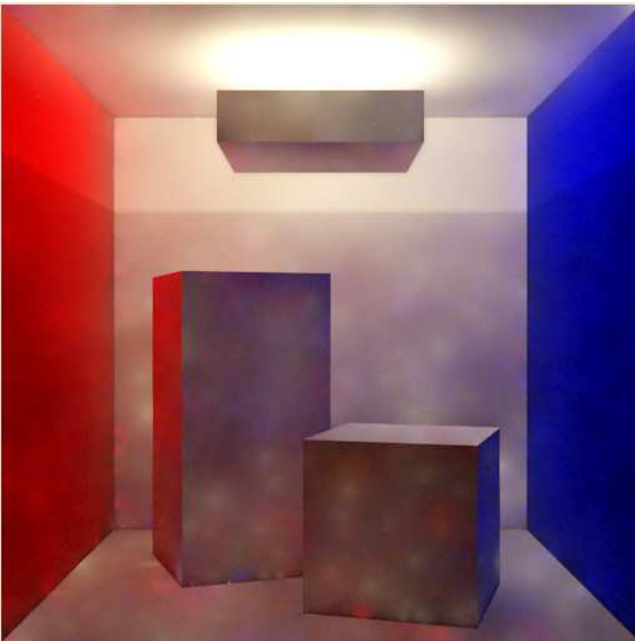


Figure 3: ERPT with 100 mutations and 24 samples per pixel.



renders shown above, the result in dark regions is very uneven. This issue was noted by Cline et al, and they suggest chains of 100-1000 are required. I used chains of 100 simply due to time constraints, and presumably longer chains would further smooth out the unevenness.

For caustic lighting, I set up a scene with a mirror ring against the back wall of a Cornell box, and a glass sphere closer to the viewer on the right. These images required

roughly 7.5 hours to render, although for the ERPT case the number of mutations and samples was identical to the preceding test. This is because in the previous example, with purely diffuse surfaces, there is no opportunity for multi-chain or specular mutations to occur, so all the mutations are very short, and quick to apply.

Considering the results in figures 4 and 5, MC path tracing successfully generates a nice cardioid caustic inside the ring, as one might expect. However the details of this caustic are blurry and indistinct, and there is a great deal of noise inside the ring. In contrast ERPT generates some very attractive cardioid caustics, with more detail and less noise. In front of the sphere and to the left, there is an additional caustic that is barely hinted at in the path tracing version. However, it is unclear whether the additional coloured caustic effects near the sphere and on the left and right walls are correct, or evidence of an error in the implementation.

There are also random bright spots that look like caustics, but are in fact artifacts of the algorithm. The original paper demonstrates these same types of artifacts in their “torus-in-a-glass-cube” renders. The explanation for this is that a low probability path with very high energy content was generated and deposited. Increased mutations and samples reduces these problems, although presumably a very large number is required to eliminate them entirely. In defense of ERPT, it is also worth noting that in renders taking just a few minutes to complete, ERPT already begins to flesh out the caustics, while a path tracing render shows nothing but bright speckled noise filling the ring.

Another interesting feature is the curved black square reflected in the sphere in the Monte Carlo images. This is a result of paths bouncing back out of the scene into the void beyond. Notice that this feature is entirely lacking in the ERPT images. The reason for this is that sample paths of this type have zero energy, and hence energy cannot be redistributed. One possible solution might be to split up the integral, and blend the ERPT solution with a Monte Carlo solution for just the zero energy paths. This would be similar in nature to how Metropolis typically uses MC path tracing to compute direct lighting, and MLT for the remaining paths.

The consecutive sample filter implementation worked exactly as claimed in the paper, despite the fact that the authors provided no examples demonstrating it. Figure 6 shows a render with and without this filter (using a consecutive sample cutoff of 10), clearly demonstrating that it has the desired effect.

6 Conclusion

Although ERPT is indeed easier to understand and apply than the original MLT formulation, some details of the algorithm were not crystal clear. This report has highlighted some of the difficulties encountered during implementation, and attempted to clarify details where possible. I have also successfully reproduced the qualitative behaviour of the re-

Figure 4: MC path tracing with 5625 samples per pixel.

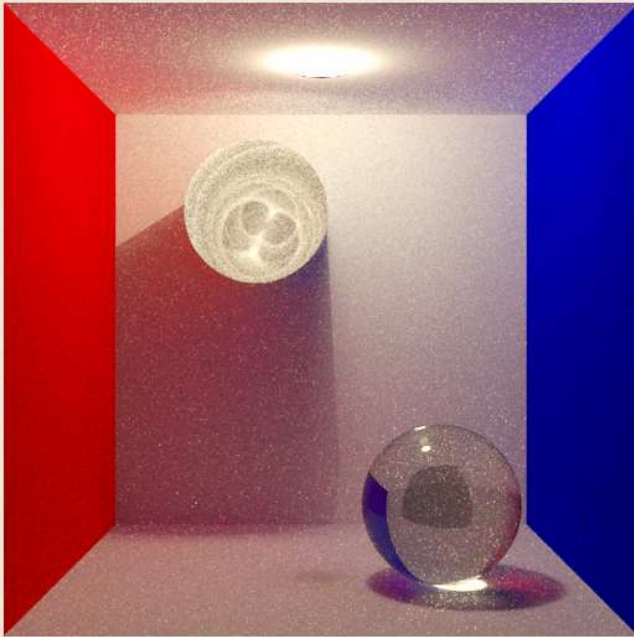
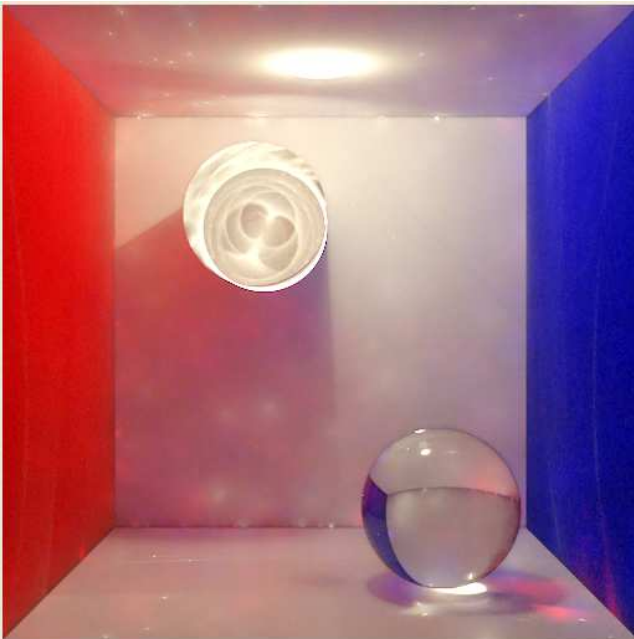


Figure 5: ERPT with 100 mutations and 24 samples per pixel.



sults shown in [Cline et al. 2005], as well as the problems with the algorithm, such as bright spots and blotchiness due to insufficient sampling or mutations.

There are several areas on which future work could potentially focus. In terms of the current implementation, the “proposed mutations” noise filter has not yet been implemented. In the original paper it is shown to improve noise in smooth areas while successfully retaining sharp edges. Fur-

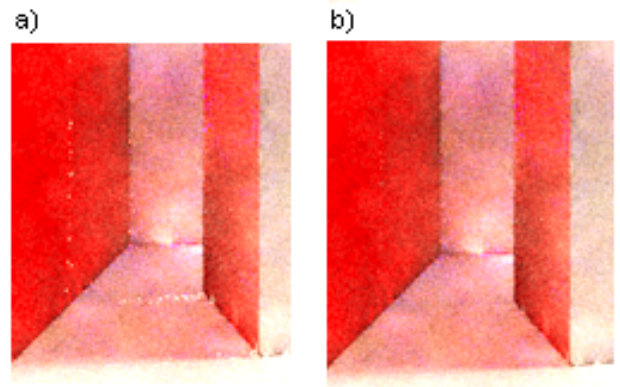


Figure 6: A comparison between (a) unfiltered and (b) filtered renders of a portion of a Cornell box. Note that the incorrectly bright pixels (due to mutation chains getting “stuck”) are eliminated in the filtered version.

ther code optimizations would also be helpful to accelerate the mutation and evaluation of paths, and thus produce faster renders.

As for extensions to the algorithm, deriving filters that would not bias the final result would be beneficial, as Cline et al suggest. Combining ERPT with existing variance reduction approaches such as importance sampling would be another useful avenue to explore. Lastly, looking for techniques to address the blotchiness of short mutation chains and the occurrence of bright spots in the presence of specular surfaces would also be helpful in making ERPT a more effective rendering method.

References

- ASHIKHMIN, M., PREMOŽE, S., SHIRLEY, P., AND SMITS, B. 2001. A variance analysis of the Metropolis Light Transport algorithm. *Computers and Graphics* 25, 2, 287–294.
- CLINE, D., AND EGBERT, P. 2005. A practical introduction to metropolis light transport. Tech. rep., Brigham Young University.
- CLINE, D., TALBOT, J., AND EGBERT, P. 2005. Energy redistribution path tracing. *ACM Trans. Graph.* 24, 3, 1186–1195.
- JENSEN, H. W. 1996. global illumination using photon mapping. In *Rendering Techniques*, 21–30.
- KAJIYA, J. T. 1986. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 143–150.
- KELEMEN, C., SZIRMAY-KALOS, L., ANTAL, G., AND CSONKA, F. 2002. A simple and robust mutation strategy for the metropolis light transport algorithm. *Computer Graphics Forum* 21, 3, 1–10.
- LAFORTUNE, E. P., AND WILLEMS, Y. D. 1993. Bi-directional Path Tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, H. P. Santo, Ed., 145–153.

- LAWRENCE, J., RUSINKIEWICZ, S., AND RAMAMOORTHY, R. 2004. Efficient brdf importance sampling using a factored representation. *ACM Trans. Graph.* 23, 3, 496–505.
- PAULY, M., KOLLIG, T., AND KELLER, A. 2000. Metropolis light transport for participating media. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, Springer-Verlag, London, UK, 11–22.
- PHARR, M., AND HUMPHREYS, G. 2004. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufman Publishers, San Francisco, CA.
- SHIRLEY, P., WADE, B., HUBBARD, P., ZARESKI, D., WALTER, B., AND GREENBERG, D. 1995. Global illumination via density estimation. In *Rendering Techniques*, 219–230.
- SZIRMAY-KALOS, L., DORNBACK, P., AND PURGATHOFER, W. 1999. On the start-up bias problem of metropolis sampling. Tech. rep., Technical University of Budapest.
- VEACH, E., AND GUIBAS, L. J. 1994. Bidirectional estimators for light transport. In *Rendering Techniques*, 147–162.
- VEACH, E., AND GUIBAS, L. J. 1995. Optimally combining sampling techniques for monte carlo rendering. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 419–428.
- VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 65–76.
- VEACH, E. 1997. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University.
- WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. 1988. A ray tracing solution for diffuse interreflection. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 85–92.

