DTOADQ

October 3, 2017

1 Introduction

2 SDFs

$$\begin{aligned} S(P,R) &= P - R^2 && \text{SDF of a sphere Radius} \\ S(P,N) &= P \cdot N && \text{SDF of a plane at Normal} \\ S(P,H,R) &= <|P_{x,z}| - R,\ P_y> -H && \text{SDF of a torus with Radius and Hole} \end{aligned}$$

3 OpenCL

4 Light Transport Introduction

5 Bidirectional Path Tracing

All the materials in here, about path tracing, have come from Veach's PhD thesis. So, in a way, DTOADQ, at this stage, is just an implementation of BDPT as described in veach's thesis to the GPU, along with pulling from other resources such as Physically Based Render [PBR] 3rd ed. In order to understand the mathematics for BDPT, it must be worked up starting from light transport. So this section is about transforming the algorithms and mathematical models into something easily and efficiently implementable for the GPU, even if some of it is a rewrite of the mathematical model so that a programmer could understand it. In order to compute a normal of an SDF, you must perform a gradient approximation: the central distance onf the SDF at P. For example, the difference quotient:

$$\frac{(f(x+h) - f(x))}{h}$$

Calculating probability distribution functions is a very vital component in Monte Carlo Sampling. The PDF states the probability of a random variable to take the form of a given input. Its usefulness comes when contributions from many Monte Carlo samples have a different weight. That is, one sample may have a higher probability distribution, and thus a higher contribution to the path. That's the strategy behind Monte Carlo;

$$\int f(x) \approx \frac{1}{N} \Sigma_0^N \frac{f(x)}{p(x)} \tag{1}$$

The PDF applies to everything from path contributions, solid angles of a BRDF, uniformly sampling a sphere, etc. I had a difficult time understanding these, and more importantly, how to calculate them for a given formula. The biggest conceptual barrier is that, for example, given a solid angle in a hemisphere

Ω, the probability for a random variable X to equal ω is 0. But this isn't what's happening here, this is all about the probability that X could potentially equal ω when compared to other values in Ω. It's all about comparing the probabilities of two potential ωs with each other.

I still have a hard time conceptualizing these things in my head, but the good thing is, is that calculating these are relatively easy. There's a lot that could be talked about, but the important thing is that I've found a reliable three-step process. The first is to find the cumulative distribution function $CDF = Pr(X \leq x)$. Where Pr is the probability, X is a uniformly random variable, and x is a known constant. In other words, this gives the probability that, for a formula P, where $X \in \Omega$, and Ω is the set of all possible values for CDF, that $X \leq x$. The second step is to take the derivative of this, so for PDF p

$$p(x) = \frac{dCDF(x)}{dx} \tag{2}$$

And the final step is to verify that the PDF is correct by integrating it over Ω, this final value should equal 1. This makes sense intuitively – it is a probability after all, so the sum of the probabilities of every possible input must be 1. For an example, how I calculated the PDF of an area light source $L_e^0(v) = L_o(P, \omega_o)$ is quite simple. Firstly,the area light is defined as a sphere, and thus our set of all possible values is correlated to surface area: $\Omega = [0,\ 2.0f * \tau * R^2]$. Thus for a given constant x, and a uniformly random X,

$$CDF(x) = Pr(X \leq x) = \frac{x}{2.0f * \tau * R^2} \tag{3}$$

$$p(x) = \frac{dCDF(x)}{dx} = \frac{1.0f}{2.0f * \tau * R^2} \tag{4}$$

$$\int_0^\Omega p(x)dx = 1.0f \tag{5}$$

In code

```
float Light_PDF ( float3 P, Emitter* m ) {
  float rad_sqr = m->radius*m->radius;
  return 1.0f/(2.0f*TAU*rad_sqr);
}
```

The more interesting aspect, in this example, is actually generating the uniformly distributed samples of a sphere. The naive method gives incorrect distributions.TODO page 774

The most obvious way to calculate this is the six-point gradient; where for each axis you compute $f(x+h) - f(x-h)$, and normalize the result. This returns a vector pointing in the direction where the map's SDF changes the most: the normal. A four point gradient is possible; as long as each each point of the gradient is multiplied by ϵ before normalizing. The speedup from six to four is worthwhile, as computing each gradient is expensive as it involves a sdf map call. There are more utilities with gradients not yet currently provided in DTOADQ but are worth mentioning, a few are: an additional normal call using an alternative (higher-quality) SDF map can be used to compute a 'normal' bumpmap, and analyzing the gradient by hand allows a cheap normal computation of primitives such as a spheres.

Rendering equation with solid angle:

$$L_o(P, \omega_o) = L_o(P, \omega_o) + \int_\Omega f_s(P, \omega_i, \omega_o) L_i(P, \omega_i) cos\Theta_i d\omega_i \tag{6}$$

To get from solid angle to path, consider:

$$f_s(v \to \dot{v} \to \ddot{v}) = f_s(P, \omega_i, \omega_o) \tag{7}$$

$$L_e(v \to \dot{v}) = L_e(P, \omega_o) \tag{8}$$

$$L_e(v \to \dot{v}) = L_e^0(v) L_e^1(v \to \dot{v}) \tag{9}$$

That is, instead of looking at a point P with an incoming and outgoing angle (ω_i, ω_o), look at the path from P_{-1} (v) to P_{+1} $(\ddot{v})$. More accurately, it's the transport between three different surfaces of the scene. In terms of SDFs, the surface is every point on an SDF map where $|map| \leq \epsilon$. There's no longer any integration over a hemisphere with solid angle ω_i, it's now over every single surface in the scene. This will be explained in detail later.
In the case of L_e, DTOADQ [at least, for now], only has support for area-light. Thus consider:

$$L_e(P, \omega_o) \equiv L_e(P) \qquad \text{[for area lights]} \tag{10}$$

$$L_e(v \to \dot{v}) = L_e^0(v) \tag{11}$$

The cos term, used for the projected solid angle in irradiance, no longer applies when dealing with area. This and $d\omega_i$ now become a generalized geometry term, G, that converts a PDF with respect to solid area using Jacobian mapping, which involves the inverse squared distance, and the cosine angle between the geometric normals at v and $\dot{v}$:

$$cos\theta_n = v \cdot N(v) \tag{12}$$

$$G(v \leftrightarrow \dot{v}) = V(v \to \dot{v}) \frac{(cos\theta_n cos\dot{\theta}_n)}{||v - \dot{v}||^2} \tag{13}$$

Where V is the visibility test. Thus the rendering equation with respects to area is:

$$L(v \to \dot{v}) = L_e(v \to \dot{v}) \int_M L(v \to \dot{v}) G(v \leftrightarrow \dot{v}) f_s(v \to \dot{v} \to \ddot{v}) d_A \tag{14}$$

where A is the area on M, and M is the union of all scene surfaces. This is known as the three-point form or the light transport equation. By recursively expanding[1], so that the three point form is integrated over every possible set of possible paths, this can be rewritten as the more usable format:

$$I_j = \int_\Omega f_j(\bar{v}) d\mu(\bar{v}) \tag{15}$$

Where $\bar{v}$ is a path $z_0 \to z_s \to y_t \to y_0 (eye \to light)$, Ω is the combination of all possible paths of any length, and μ is the area-product measure [the product of all the expanded d_A]. Special care has to be taken with the definition of path itself, as in the source code, there never exists a path $\bar{v}$, only eye-path $\bar{z}$ and light-path . Right now, this is analytically unsolvable. To compute this, we need to apply monte carlo sampling and limit the maximum path length:

$$I_j \approx \frac{1.0f}{N} \Sigma_0^N F \tag{16}$$

$$F = \Sigma_s \Sigma_t \frac{f_j(\bar{v}_{s,t})}{p(\bar{v}_{s,t})} \tag{17}$$

where s is the light-path length, and t is the eye-path length. In DTOADQ, they are limited to $s + t \leq 8$[TODO]. Finally, to transform $\bar{v}$ into two seperate paths, $\frac{f_j(\bar{v}_{s,t})}{p(\bar{v}_{s,t})}$ is split into two with a connection edge c

$$\frac{f_j(\bar{v}_{s,t})}{p(\bar{v}_{s,t})} \equiv \frac{f^L(\bar{y})}{p(\bar{y})} c(\bar{y}_s \leftrightarrow \bar{z}_t) \frac{f^E(\bar{z})}{p(\bar{z})} \tag{18}$$

And now

$$F = \Sigma_s \Sigma_t \alpha_s^L c_{s,t} \alpha_t^E \tag{19}$$

$$\alpha_i^{L|E} = \begin{cases} \frac{L_e^0(y_0 \to y_1)}{p_A(y_1)} | \frac{W_e^0(z_0)}{p_A(z_0)}, & i = 1, \\ (\frac{f_s(y_{i+1} \to y_i \to y_{i-1})}{p_\sigma(y_i \to y_{i-1})} | \frac{f_s(z_{i-1} \to z_{i-2} \to z_{i-3}}{p_\sigma(z_{i-2} \to z_{i-1})})\alpha_{i-1}^{L|E}, & i > 1 \end{cases} \tag{20}$$

There is no case for $i = 0$ as either, the path doesn't exist, or if one does exist, y_0 and z_0 do not contribute to the image. Veach (and many other sources), leave this in, but it's unnecessarily confusing. A special case needs to be handled, unfortunately, for α_0^E as it is being generated on the fly while $\bar{z}$ is being constructed. The two options is to unroll the first iteration of the construction loop, or just allow the special case to exist. The latter is the better choice, as all kernels will enter and exit the special case at the same time, and the GPU might just unroll the loop anyways. There may be cases where $s = 0$ ($t = 0$ is not possible without a physical camera lens), which is equivalent to just forward path tracing. There is also to consider, what happens on $s = 1$ and $t = 1$, Well, in either case where both paths exist or only the camera path exists, the connection strategy below describes how to connect the edges of the two paths

$$c_{s,t} = \begin{cases} L_e(z_{-2} \to z_{-1}), & s = 0, t > 0, \\ f_s(y_{-2} \to y_{-1} \to z_{-1})G(y_{-1} \leftrightarrow z_{-1})f_s(y_{-1} \to z_{-1} \to z_{-2}), & s > 0, t > 0 \end{cases} \tag{21}$$

One question to be made is if the strategy $s = 0, t > 0$ is worthwhile to compute; the most noticeable effect are for caustics, light sources in direct view of an emitter should be emitted from animations in this current state of DTOADQ. Another idea to consider is that under real physics circumstances, the photons are still allowed to bounce after hitting a source of light, but the contribution from these cases are negligible, and so most likely not worthwhile to implement. Anyways, the $s = 0$ strategy is a worthwhile calculation under these circumstances

One immediately obvious optimization that could be made with this model is in regards to the visibility check in the geometric term, if you were to expand I_j, and take the V term from outside the G function $[G(v \to \dot{v})V(v \to \dot{v})]$, it would be made obvious that all the visibility checks can be cancelled out for the entire equation, except for the connection term c. The expansion is not shown as it is very lengthy, and unnecessary. Paths $\bar{y}_0 \ldots \bar{y}_s$ and $\bar{z}_0 \ldots \bar{z}_t$ being visible is intuitively obvious as the paths have been generated using the same technique used to check for visibility; raymarching. Specifically, the definition of V is

$$V(v \to \dot{v}) = ||v - \dot{v}|| \leq m(v, \overrightarrow{v\,\dot{v}}) \tag{22}$$

where m is a march through the SDF map. A visibility check is only necessary for the connection edge

$$||v - \dot{v}|| = m(v, \overrightarrow{v, \dot{v}})$$
$$\text{for all paths but } y_{-1} \to z_{-1}$$

Moving on, for $\alpha_i^{L|E}$, these are equivalent for both L and E, the problem is that the $\bar{y}$ path is generated in perspective of the light source, so we have to call the BSDF in perspective of the camera; the path is flipped. Handling $\bar{y}$ is significantly easier than $\bar{z}$, as $\bar{y}$ is precomputed. For $\bar{z}$, the next vertex z_{+1} (ω_o in the perspective of SA), is unknown. For example α_1^L in OpenCL looks like:

```
Vertex* V0 = light_path.vertices - 1, * V1 = light_path.vertices;
float sigma_pdf = Light_PDF(V1.origin, &light);
float g = Geometric_Term(V0.origin, V1.origin, V0.normal, V1.normal);
float light_pdf = sigma_pdf * g;
float Le = light.colour;
light_contrib[1] = Le / light_pdf;
```

For $c_{s,t}$, there is no cases for $s = s, t = 0$ as the lens does not have a physical existence in the SDF map. p_A is the PDF w.r.t. area, for the light path; we only need to take the PDF of the solid angle of a sphere

$$p_A^L(y_0 \to y_1) = p_\sigma^L(y_0 \to y_1) G(v \leftrightarrow \dot{v}) \tag{23}$$

$$p_\sigma^L(y_0 \to y_1) = \frac{1.0f}{\tau * (1.0f - \sqrt{\frac{R^2}{|y_0 - y_1|^2}})} \tag{24}$$

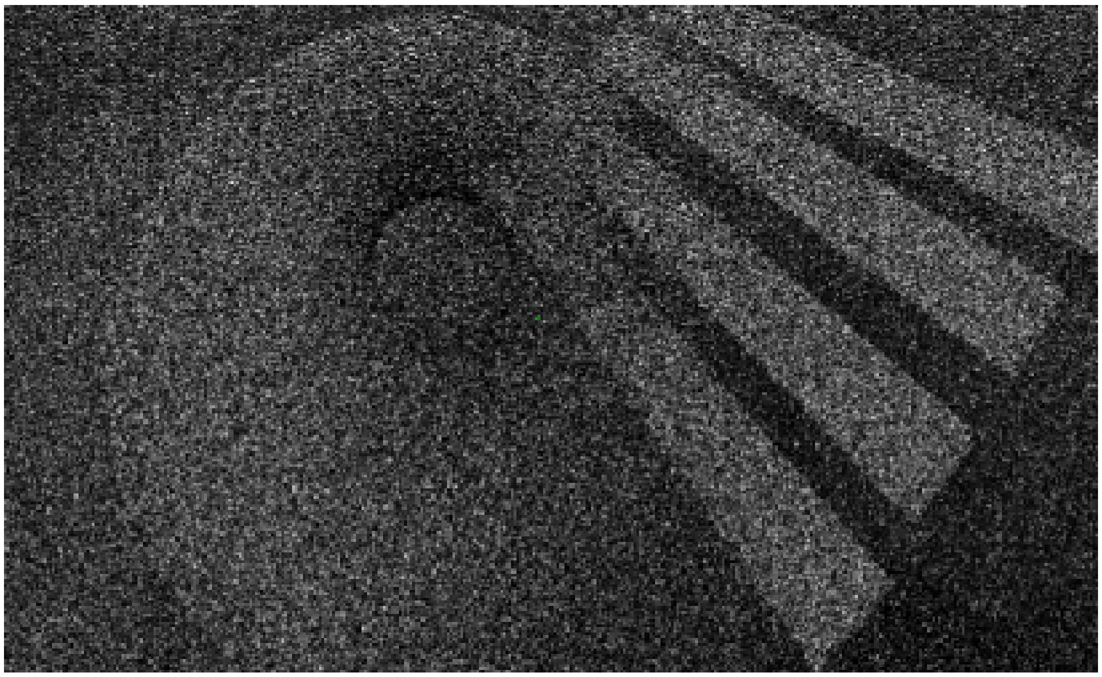
where R is the radius of the sphere, note y_0 is a point on the surface of the sphere, and not its origin. $p_A^E(x_0 \to x_1)$ doesn't exist, as there is no physical lens on the scene. [TODO - this might cahnge later]. One of the benefits of this, is that there are no special cases for pathes where $t = 0$, which is impossible to handle on the GPU as the path could contribute to any arbitrary pixel [directly hitting the lens].

$$P_A^E = 1.0f \tag{25}$$

The L_e^0 is contribution from the light source. As of right now, DTOADQ only supports area lights, so L_e^0 is set a constant for the emitter's spectrum value. The W_e^0 is contribution from the eye source, that is, the lens. Since the lens does not exist in the scene, as of now [TODO - this might change later], there is no W_e^0. While this should be implemented in the future, a hidden benefit of the current model is that there are no special cases for paths where $t = 0$, which is impossible to handle on the GPU as any such path could contribute to any arbitrary pixel [the light path is, after all, directly hitting the lens]. So this has now become:

$$\alpha_i^E = \Sigma_{i \leq 0} \tag{26}$$

The amount that each path contributes to F differ greatly, so naively weighing each contribution uniformly by something like 1.0f, or even, by inverse path length, will producy very noisy images, and would not be worth the additional computation time that BDPT requires.


(27)

In order to get good samples from the contributions, they must be combined in an optimal manner using Multiple Importance Sampling: each contribution is weighted by the PDF of the entire path itself:

$$F \equiv \Sigma_s \Sigma_t W_{s,t} C_{s,t}$$
$$C_{s,t} = \alpha_s^L c_{s,t} \alpha_t^E$$

where $W_{s,t}$ is the MIS weight of path $\bar{y} \leftarrow \bar{z}$, and $C_{s,t}$ is the unweighted contribution of the same path. For clarity, $C^U_{s,t}$ as described below is the unweighted contribution of the algorithm:

$$C^u_{s,t} = (\alpha^L_s c_{s,t} \alpha^E_t) \equiv \frac{f_j(\bar{v})}{p(\bar{v})} \tag{28}$$

thus

$$F = \Sigma_s \Sigma_t W_{s,t} C^u_{s,t} \tag{29}$$

The formula Veach gives for Multiple Importance Sampling on BDPT is as so:

$$MIS_{st}(\bar{x}) = \frac{P_s(\bar{x})}{\Sigma_i P_i(\bar{x})} \tag{30}$$

Conceptually, the numerator $P_s(\bar{x})$ is the path density that was generated, while the denominator is the path density of other connection strategies that could could have, in theory,[1] created the path. The straight forward implementation of this has a lot of problems gone unresolved in Veach's thesis; PBR 3rd Ed, chapter 16 pg 1014-1016 addresses and describe these problems. Primarily, $P_i(\bar{x})$ will overflow CLFloat (due to the distance in the Geometric term), and the straight forward implementation is very long, hard to debug, and has a poor time complexity. But the important part is the end result of their solution:

$$r_i(\bar{x}) = \begin{cases} 1.0f, & i = s, \\ \frac{\overleftarrow{P}(\bar{x}_i)}{\overrightarrow{P}(\bar{x}_i)} * r_{i+1}(\bar{x}), & i < s, \\ \frac{\overrightarrow{P}(\bar{x}_{i-1})}{\overleftarrow{P}(\bar{x}_{i-1})} * r_{i-1}(\bar{x}), & i > s \end{cases} \tag{31}$$

However, DTOADQ can simplify a bit further; unlike above, there are two paths for which to evaluate. Specifically, in the above term, for where $i < s$, we have $\frac{\overleftarrow{p}(\bar{x}_i)}{\overrightarrow{p}(\bar{x}_i)}$, which is the light path (i is iterating towards light-length s). The eye contribution is inversed from this, (also iterating towards s, hence why $\bar{x}_{i-1}$ is used). Another way of saying it, is $\bar{x} = \bar{y}||\bar{z}_{rev}$. In DTOADQ's case, the weights are calculated in the same direction relative to its originator; in this case the perspective of the eye path.

$$\mathcal{W}_i(\bar{x}) \equiv r_i(\bar{x}) \tag{32}$$

$$\mathcal{W}_i(\bar{x}) = \begin{cases} 1.0f, & i = 0, \\ \frac{\overleftarrow{p_i}(\bar{x}_i)}{\overrightarrow{p_i}(\bar{x}_i)} * \mathcal{W}_{i-1}(\bar{x}), & i > 0 \end{cases} \tag{33}$$

And then the expanded form of $\mathcal{W}_i(\bar{x})$ looks like

$$\mathcal{W}_i(\bar{x}) = \Pi^i_{n=2}\left(\frac{\overleftarrow{p_\sigma}(x_n)\overleftarrow{G}(x_n)}{\overrightarrow{p_\sigma}(x_n)\overrightarrow{G}(x_n)}\right) * \frac{\overleftarrow{p_A}(x_1)}{\overrightarrow{p_A}(x_1)} \tag{34}$$

And now the MIS looks like:

$$MIS_{s,t}(\bar{y}, \bar{z}) = \frac{1.0f}{\mathcal{W}_s(\bar{y}) + \mathcal{W}_t(\bar{z}) + 1.0f} \tag{35}$$

Thus, the final rendering equation for DTOADQ looks like such:

$$P = \frac{1.0f}{N} \Sigma^\infty_0 \Sigma_{s \le 0} \Sigma_{t \le 0} \alpha^L_s c_{s,t} \alpha^E_t MIS_{s,t}(\bar{y}, \bar{z}) \tag{36}$$

$$P_i(\bar{x}_{s,t}) = P_s^L p_t^E \tag{37}$$

$$p_i^{L|E}(\bar{x}) = \begin{cases} 1.0f, & i = 0, \\ P_A(x_i), & i = 1, \\ P_\sigma(x_{i-1} \to x_i) G(x_{i-1} \leftrightarrow x_i) P_{i-1}^{L|E}, & i > 1 \end{cases} \tag{38}$$

6 Video/Image Emitter

7 Future Improvements

Every emitter is chosen uniformly, however a better system would weight the probability of each emitter by

$$p_l = \frac{\phi_l r_l}{\Sigma_i \phi_i r_l}$$

Thus emitters that have a higher contribution to the scene, that is, a larger emittance and radius, will have a higher probability to be selected.

8 Bibliography

[1]

$$\Sigma_{k=1}^{\infty} \int_{M^{k+1}} L_e(v_0 \to v_1) G(v_0 \leftrightarrow v_1) \Pi_{i=1}^{k-1} F_s(v_{i-1} \to v_i \to v_{i+1}) G(v_i \leftrightarrow v_{i+1}) \\ W_e^j(v_{k-1} \to v_k) dA(v_0) \dots dA(v_k)$$