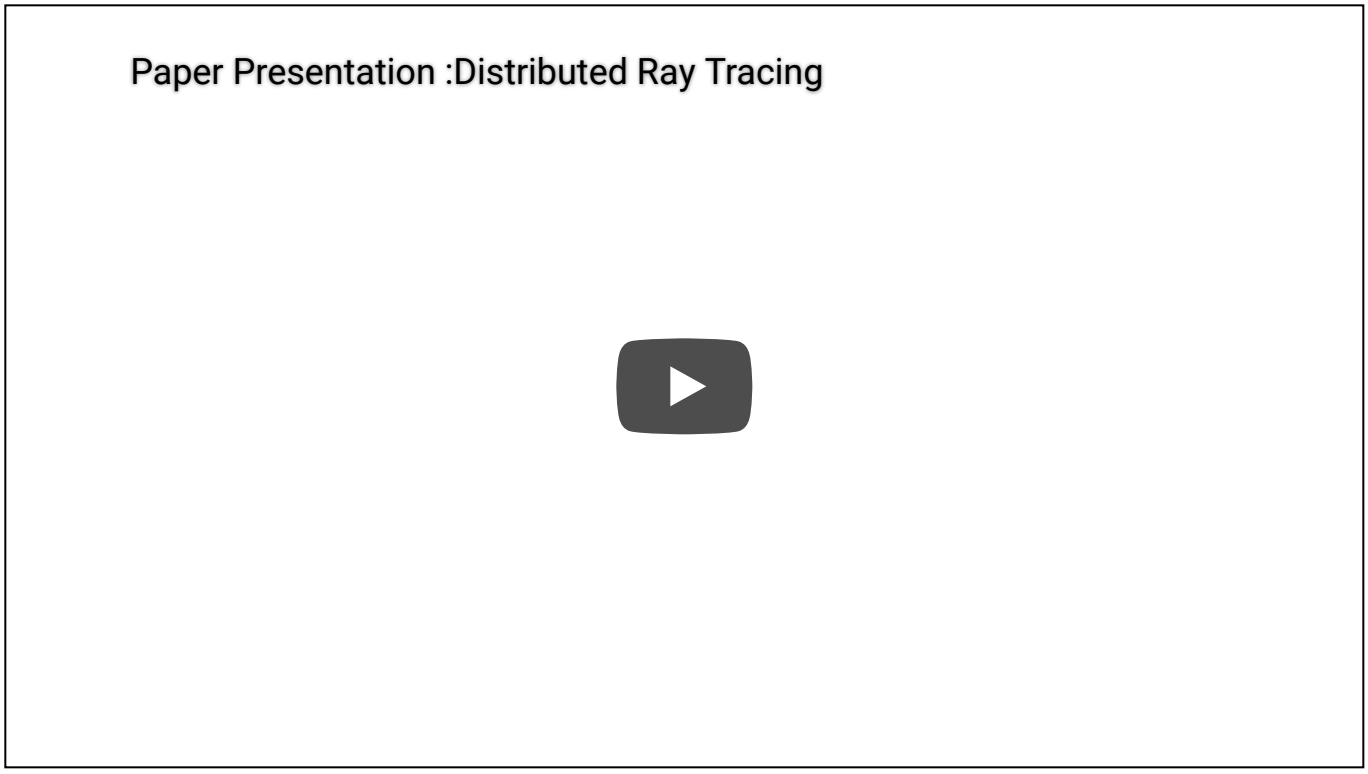
Paper: Distributed Ray Tracing

帽捲 Follow
Apr 14 · 20 min read

分布式光線追蹤，或者我們說隨機分布式光線追蹤(Stocastically Distribute Ray Tracing)，他是在Ray Casting、Ray Tracing之後提出的改良。

首先，我們可以先看看傳統Ray tracing有甚麼問題，傳統上每個pixel就用一條ray去遞迴找值，那結果其實會產生很明顯的鋸齒狀(aliasing)，所以第一個問題就是反鋸齒(Anti-aliasing)。

Presentation Video



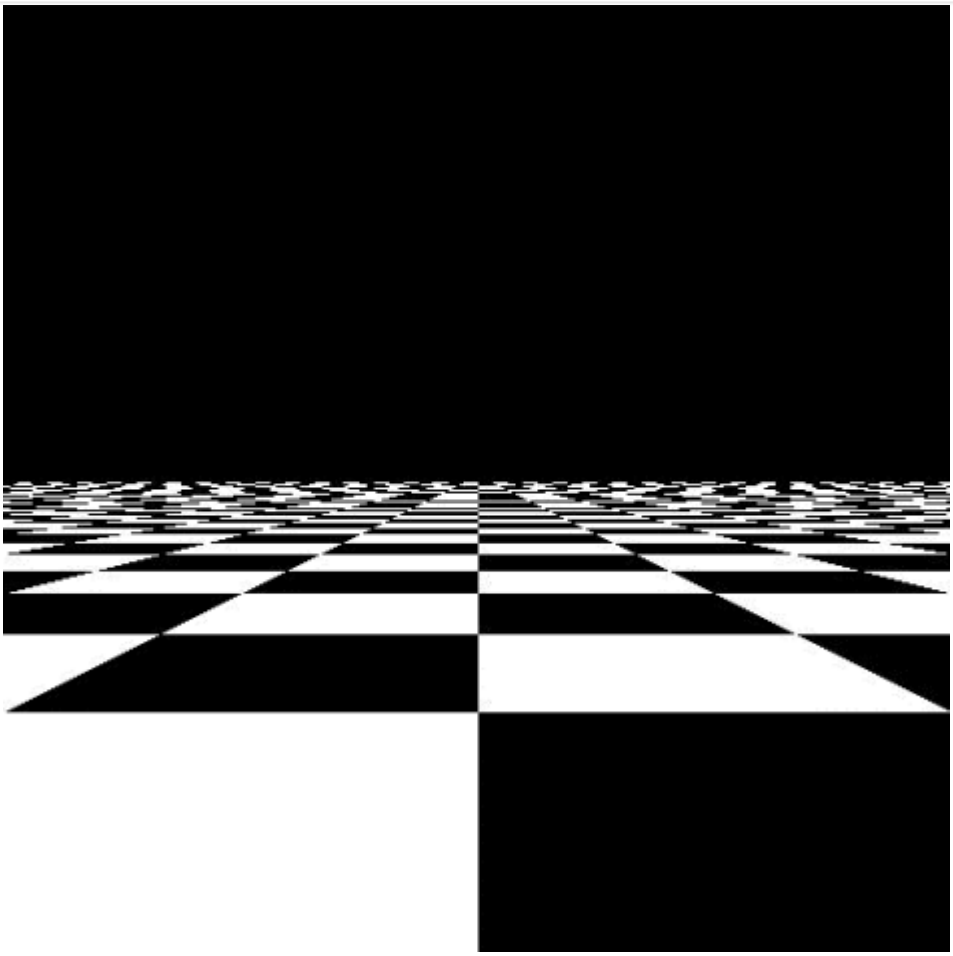


Figure 1(a) OpenGL rendering

可以發現後面的texture都亂掉了，那解法通常就是mipmap，也就等同於是在不同距離的texture我們用了不同的filter，也就是不同的採樣(sample)方式，由此來看，sample是相當重要的。

那回到傳統的Ray Tracing

其實也有類似的問題，那最明顯的問題就是鋸齒狀，那為甚麼會產生鋸齒呢?因為我們每個pixel只使用一條ray，那實際上我們從pixel打出去的並不是光線，而是有體積的光束，也就是說，我們每個pixel所表示的應該是一個面積裡的顏色，但我們只用一條ray的話，我們只能武斷的決定他是0或是1。

那最顯而易見的做法就是多打幾條:

Figure 1(c) 256 sample per pixel

非常明顯的，鋸齒沒有這麼明顯，因為每個pixel有了0~255階的範圍，所以在邊界的地方有了相對平滑的明度變化，因此原則上，只要我們能夠在一個pixel內打出無限條ray，那就可以無限精準。但這個方法也有個顯而易見的缺點，就是太沒有效率，原因是因為我們採用regular的方式打出多條ray，這個就是產生鋸齒的原因，只要是regular就會產生鋸齒，舉個例子來說，就是pixel，由於每張圖都是由regular排列的二維陣列，所以原則上只要是pixel-based的東西就會產生鋸齒，反過來說，鋸齒的原因就是regular的排列。

於是，就有了後面的做法，讓我們增加一些隨機性:

Figure 1(d) One jittering sample per pixel

原本最naive的做法是在每個pixel的中心(或是某個固定位置)打出ray，由於是固定(regular)位置，所以會產生鋸齒。現在我們則是在一個pixel的範圍內，隨機找一個點打出ray，也就是這邊稱為jittering(抖動)，從上圖可以看到鋸齒的痕跡變得不是這麼明顯，儘管每個pixel只有打一條ray，但相對來說，也產生了很多的noise。

那如果我們多打幾條呢?

Figure 1(e) Four jittering sample per pixel

我們僅僅使用4條ray，就可以做出不錯的結果，雖然相對於regular的打出256條是較差的，但是一定程度上，這是權衡之下的結果，也就是較有效率的方式。

那麼，為甚麼會這樣呢，我們可以用一些影像處理的例子來說明:

Figure 2 Two binary image, Regular pattern(a), Jittered pattern(b)

上圖都是binary的圖片，也就是每個pixel只有黑與白，那麼為甚麼我們感受得出明暗變化也就是灰階呢?因為我們的眼睛自帶有filter，也就是會把一個區域的東西整合在一起顯示，所以我們可以利用黑白點的不同密度來展現明暗。很明顯的，左圖有非常不自然的鋸齒，而右圖沒有，這就是因為左圖使用了regular的pattern來給不同明暗的區域密度，而右圖則是使用隨機抖動的方式產生。

回到我們的Distributed Ray Tracing，我們可以做個小結，我們利用加入隨機性的抖動來更有效的sample(Jittered sampling)，他雖然不精準會產生noise，但是原則上是相對有效率的做法。另外，這種作法其實說到底就是刻意增加noise，從訊號處理的角度來看，增加noise就是降低高頻訊號，而鋸齒則剛好也是高頻訊號，所以從這個角度也可以說明為甚麼可以去除鋸齒。

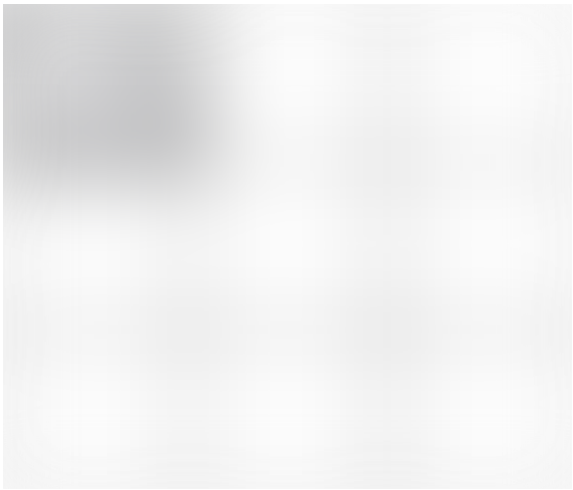


Figure 3

另外，如果從蒙地卡羅法來思考，這樣子的方式就是non-regular Stratified sampling，也就是所謂的分層式抽樣，值得一提的是，如果使用分層式抽樣，那麼這樣子的抽樣會與維度相關，而會一定程度上失去蒙地卡羅的優點，也就是與維度無關，幸好的事，原則上我們最多就在二維上分層，所以可以當作常數看待。

Shading model

那在我們討論shading的各種效果以前，我們先來了解一下paper所提到的shading model。

The intensity I of the reflected light at a point on a surface is an integral over the hemisphere above the surface. L is illumination function, R is reflection function

其實這邊所提出的model也只是概念上的model，這邊就簡單講解一下，我們ray tracing的目標就是我們從某個角度打到某個區域，並取出那個區域的顏色，而這邊的顏色就是Intensity，也就是I函數，而這個I函數則depend on光進入的反射角，也就是Intensity的數值只跟光的反射角有關。

我們來看看裡面的積分式，L表示illumination function，也就是來自某個方向的入射光的函數，所以她指相關於入射角，R則是reflect function，就是反射光的函數，整個積分式是對於所有入射光來積分。這邊舉個例子來說明，假設是一個鏡面的材質，且只有一個光束，那此時L則會變成delta function，也就是只有那條光的方向有值，其餘方向都會是0，由於是鏡面，所以也只有在對應的反射角會有光，所以R也會是delta function，那整個積分式就會退化成只有一項有值，也就是入射光對應的反射光的角度Intensity才會有值。

進一步來說，R都是使用者依據自己對於不同材質的理解去定義的。

Gloss

一個典型的例子就是鏡子或者是金屬，也就是非常光滑的表面，會有很明顯的入射角反射角的關係，另一方面也可以想像成Phong Shading的specular，也就是說是view-dependent的。

下面就借用[1]的例子來介紹。

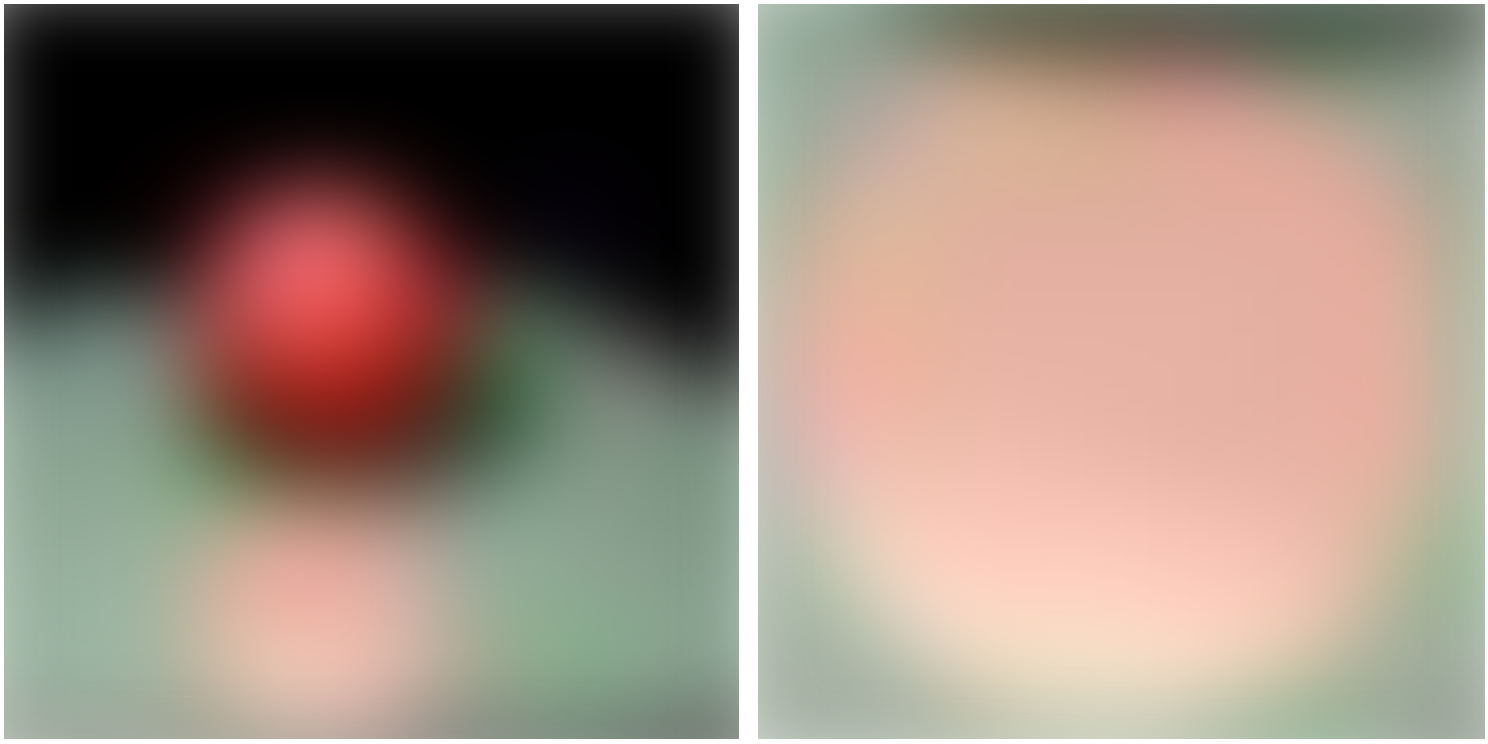


Figure 3(a) This example shows a perfect reflection produced by a traditional (non-distributed) ray tracer. Notice that the edges of the reflection are very sharp.

Figure 3(b) This example shows a glossy reflection generated by distributed ray tracing. The reflection was generated by distributing 10 rays.

可以很明顯的看到，使用傳統方法可以產生鏡面反射的效果，那相對來說，鋸齒的狀況又再度發生了，也就是我們對於鏡面反射的sample效率不好，另一方面，我們也知道大多數的材質並不會如此光滑，所以為了改善這個情況，我們使用相同的策略，利用隨機分布式的ray來sampling。

Figure 4 Gloss can be calculated by distributing these secondary rays about the mirror direction

我們在鏡面反射角的一定範圍內對反射角度做隨機的抖動，這樣就可以一定程度上消除明顯的鋸齒現象，因為我們更加有效率的對環境sampling。進一步來說，這個其實就是我們自己定義的R function，也就是我們依據對於材質的理解去調整R function在那些角度的值是多少。如果是完美鏡射的話，那這個bounding就是只有一個角度，也就是R function就變為delta function。

Translucency

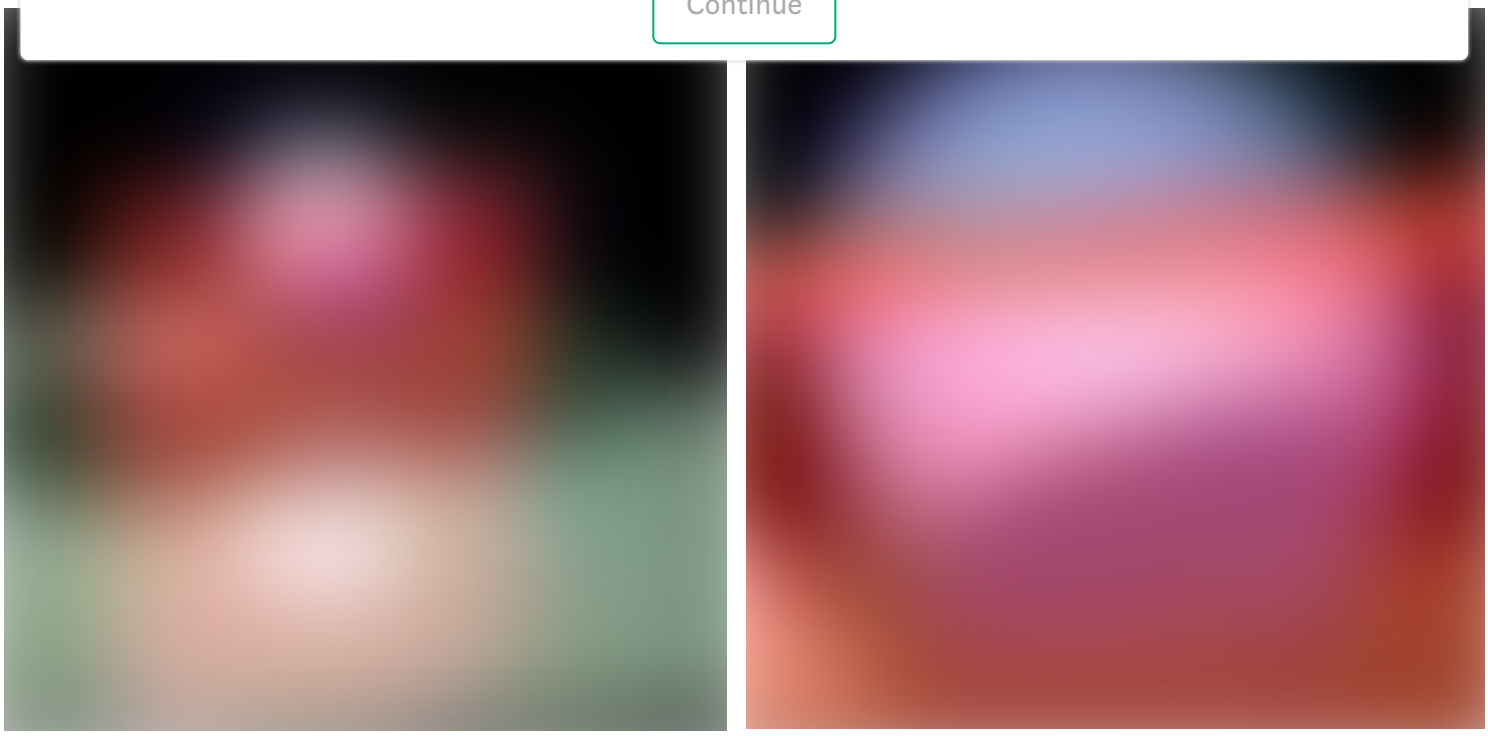


Figure 5(a) This example shows a perfectly translucent surface produced by a traditional (non-distributed) ray tracer. Notice that the edges of the sphere visible through the translucent pane are very sharp.

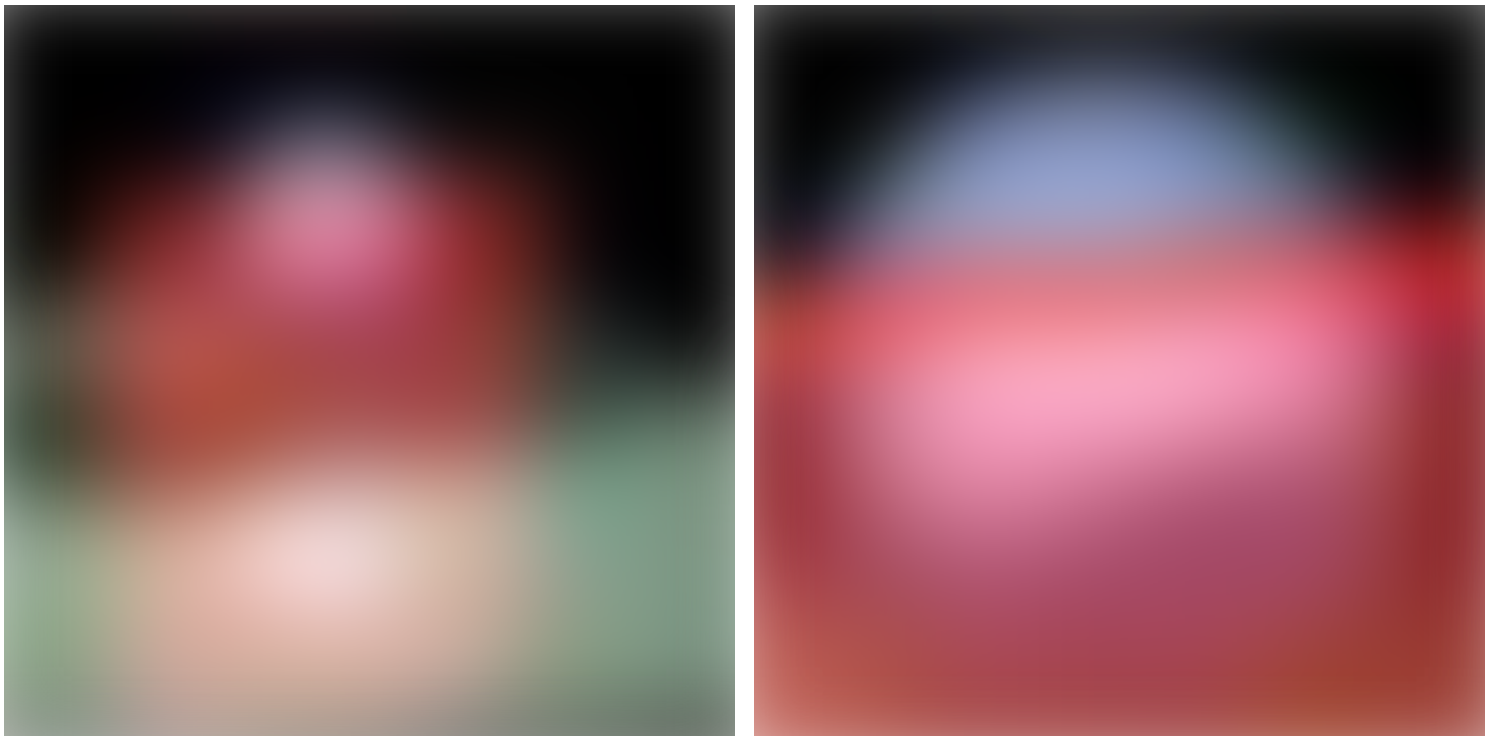


Figure 5(b) This example shows a fuzzy translucent surface generated by distributed ray tracing. The surface was generated by distributing 10 rays.

Figure 6 Translucency is calculated by distributing the secondary rays about the main direction of the transmitted light

Penumbras(Soft shadow)

陰影我們可以想像就是光沒照到的部分，那依據經驗，我們也知道陰影不會都是銳利的，更多的情況都是邊緣是帶有漸層的。

在傳統OpenGL的渲染中如果要做出shadow的話，最常見的做法就是shadow map，而在ray tracing中如果要形成影子，常見的方法就是shadow ray，簡單來說，就是檢查這個被sample到的點有沒有被光源照到，也就是從這個sample點朝著所有光源發射ray，如果有被其他東西擋住，那就算shadow，反之，如果沒有被東西擋住，就會把光的能量加上去，這就是shadow ray。

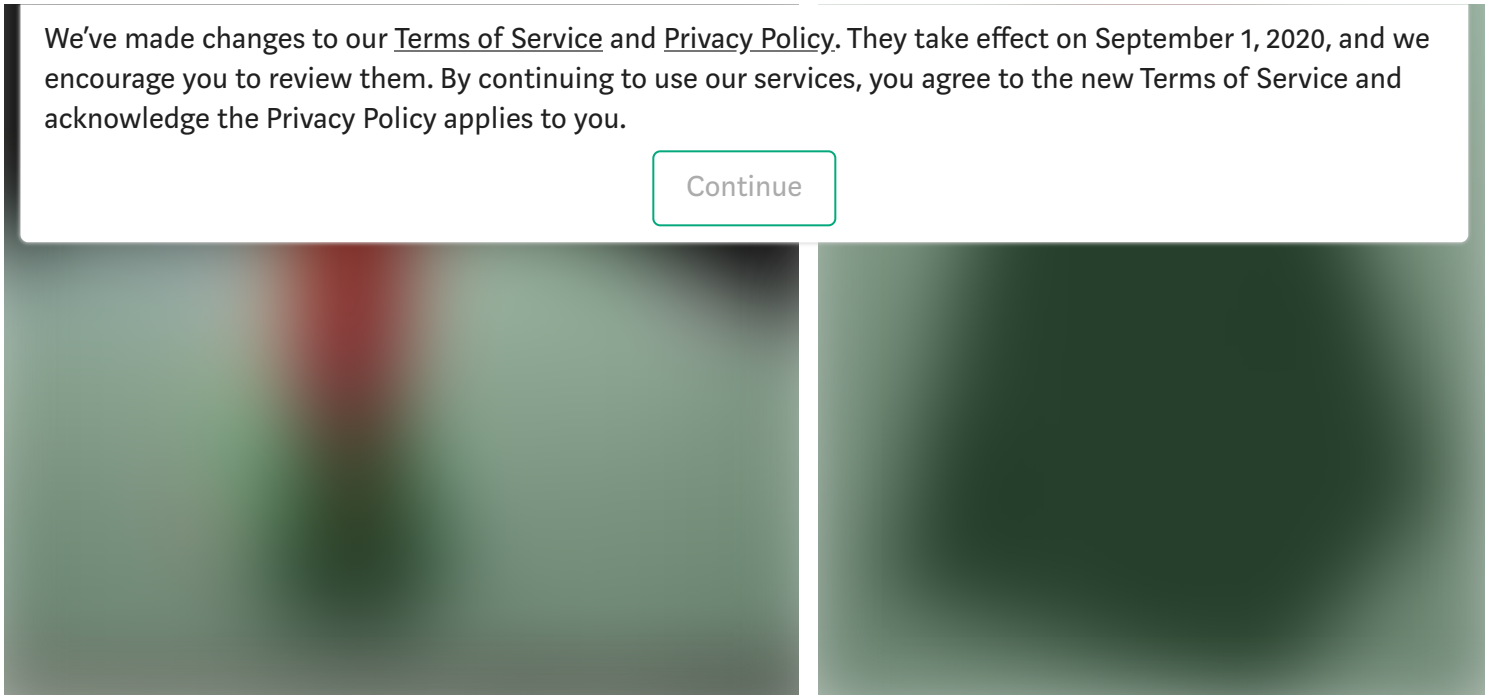


Figure 7(a) This example shows a hard shadow produced by a traditional (non-distributed) ray tracer. Notice the discrete boundary between points that are in shadow and points that are not.

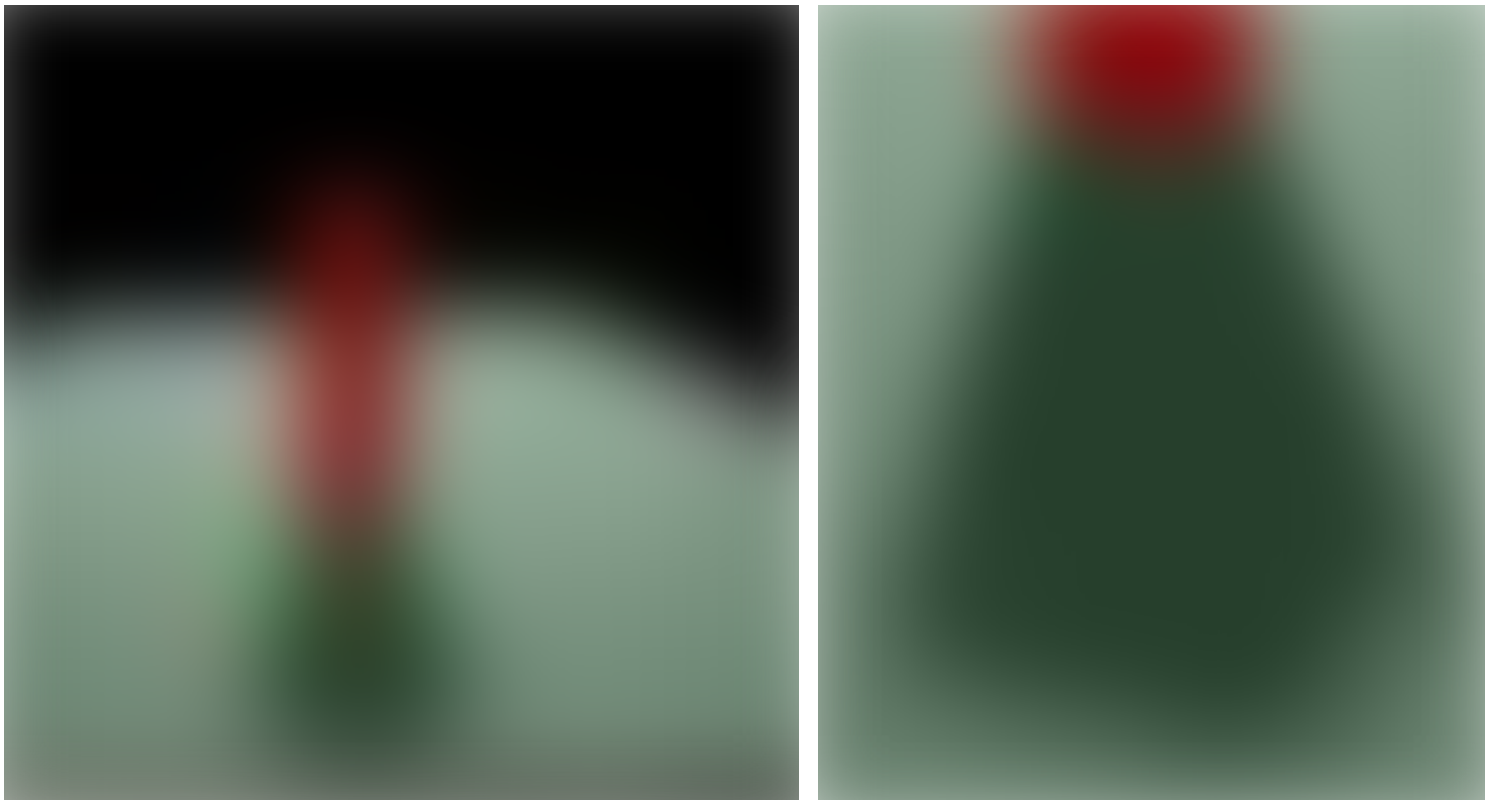
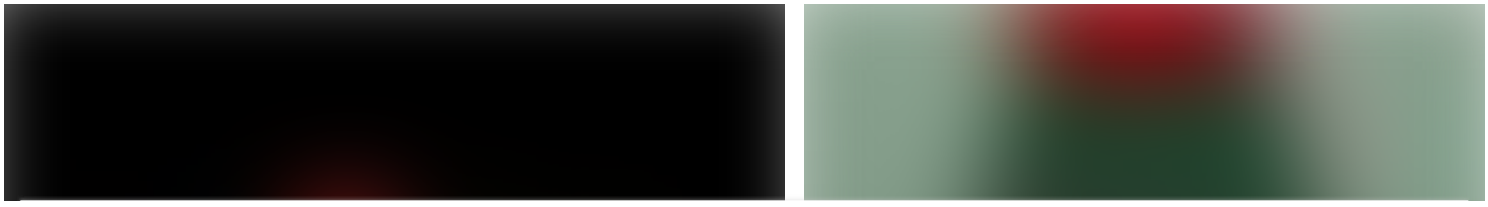


Figure 7(b) This example shows a technique for generating soft shadows that is not based on distributed ray tracing. This technique is based on uniformly sampling an area light source. Although the penumbra and umbra of the shadow are clearly visible, the variation between them is rigid.



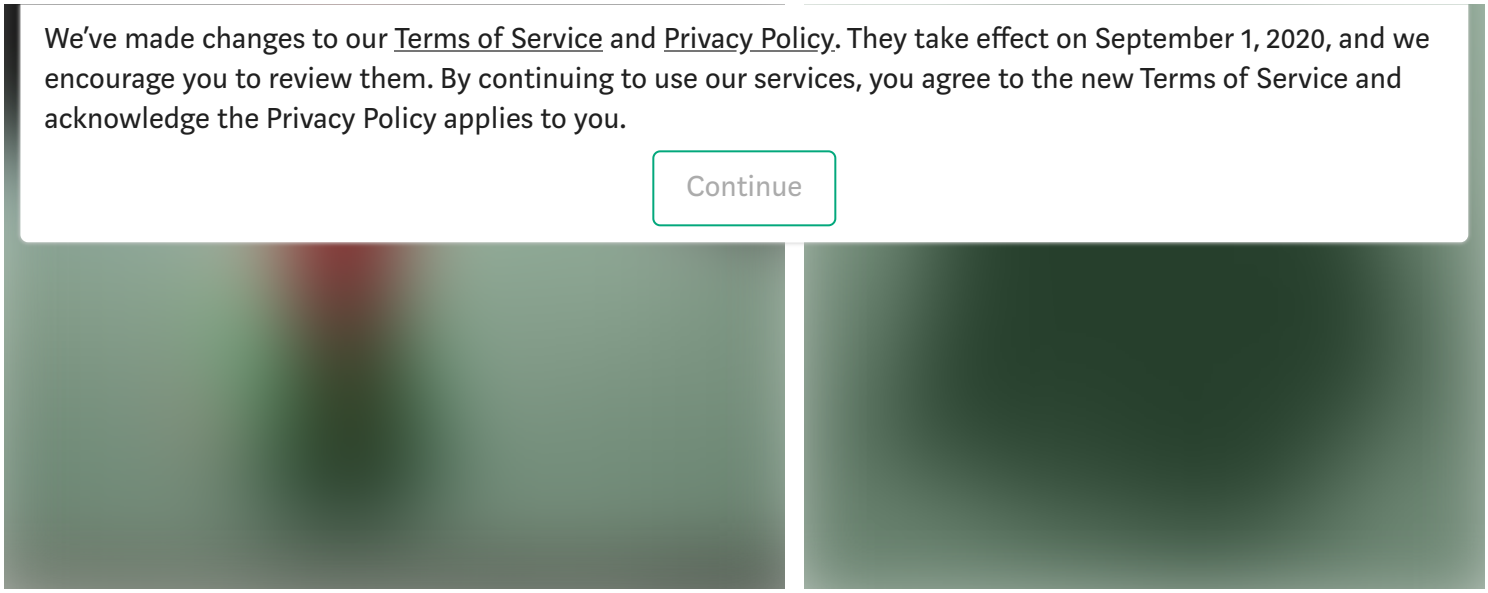


Figure 7(c) This example shows a soft shadow generated by distributed ray tracing. The light source was sampled using 10 rays.

Fig 7(a)是使用點光源的情況下，因為光源只是在3D空間是一個點，所以有無照到光就會變成0和1而已，但在一般情況下的光源都是所謂的area light，也就是光源不只是一個點，是一個面積甚至是體積，反過來說，我們之所以可以使用point light是因為我們假射光源離物體很遠，所以這樣子的簡化是合理的。

所以如果想要做出軟陰影的效果，我們必須把光源變成area light，Fig 7(b)則是每個sample點對area light的四個角發射shadow ray，可以發現更接近現實，但是這就是低效率的sample，Fig 7(c)則是使用了隨機分布式的shadow ray。

Figure 8(a)(b) The distribution of the shadow rays must be weighted according the projected area and brightness of different parts of the light source

shadow效果的一種hack。但有一說一，如果使用shadow ray，就無法產生Caustics效果，因為你因為hack，所以直接將後續的tracing中止掉。

. . .

這邊做個小結，對於各種材質我們都可以使用隨機分部式來達成各種效果，反過來說，我們是因為對於某個材質的理解，才設計不同的R function來試圖做出某種效果，所以這並不意味著一定需要設計一個大的bounding來做所有的材質，但是核心的概念上，我們會需要用　　的概念，因為這可以有效率的sample。

舉例來說，如果我們要產生沒這麼光滑的平面不一定需要隨機，我們可以只是定義一個bounding，然後regular或是固定pattern的方式做反射，這麼做仍然可以產生所謂Gloss的效果，只是還是會有鋸齒的產生，所以這篇論文的核心是在於

. . .

Depth of Field

除了上述那些屬於shading範圍的效果，還有一種效果叫做景深(Depth of Field)，這個效果是來自於相機的鏡頭產生的，更準確來說，是因為凸透鏡造成的。我們一樣可以回想OpenGL的render model，大多數的camera model就是來自於針孔相機。

Figure 9 pinhole camera model

無論是在什麼位置差一個平面他們的內容都與成像平面相同，只是大小、正倒立有差別而已，所以我們用的camera model就直接這樣子來簡化了。

Figure 10 OpenGL camera model

但在現實生活中，針孔相機不像電腦中，只要sample到就可以不損耗的把能量打在pixel，針孔相機的曝光往往不足，也就是非常暗，所以後來就使用凸透鏡來取代針孔，讓成像平面上的點更亮，但這相對遇到了另個問題，circle of confusion。

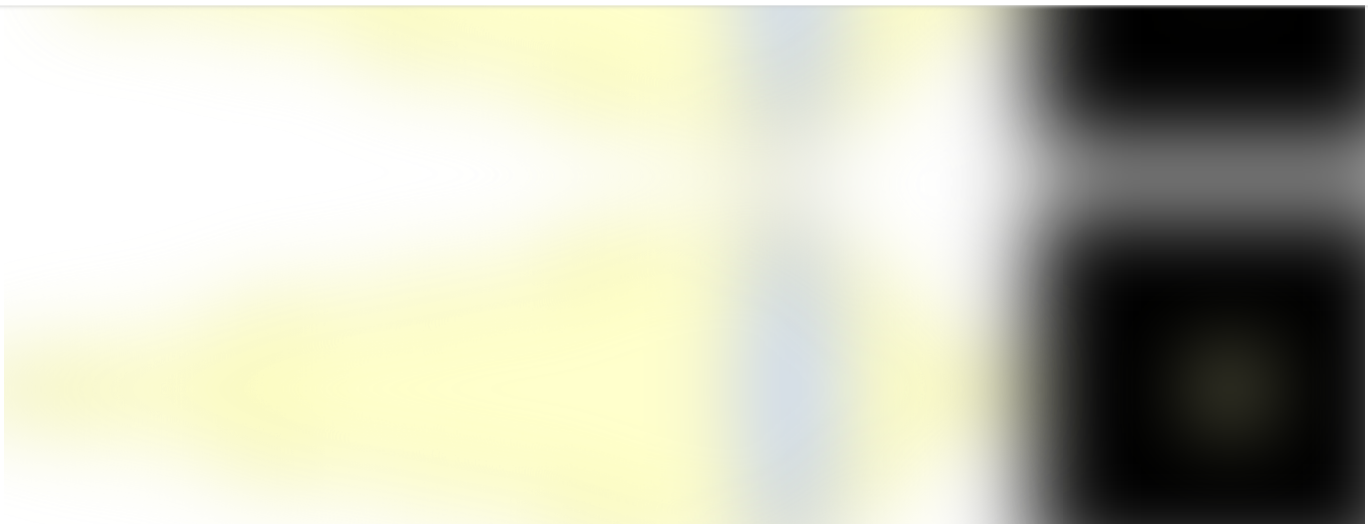


Figure 11 Circle of Confusion

因為凸透鏡說到底就是讓多個點都打在一個點，也就是多對1，那此時偌多的那邊如果資訊不同，那被對應到的這個1就會是各種混合，也就是會模糊，所以這就會有所謂的focal point的問題，這邊注意，focal point跟focus point是不同東西。也就是說如果東西不在focal point上，那個東西在成像平面上就會有模糊，幸運的是，focal point的距離是固定的，也就是說在focal point都會在一個平面上，也就是focal plane，那麼在這個focal plane上與成像平面就又會是1對1的，也就是在focal plane上的東西都能夠清晰成像。

Figure 12

那為了產生這樣子的效果，我們就必須把camera model成透鏡相機，而這在實作中，我們仍然可以用類似OpenGL的做法，也就是我們把透鏡位置當作相機位置，但此時我們不能隨便用near clip plane，因為這個plane必須要跟成像平面1對1，那麼此時我們就拿focal plane來當作虛擬成像平面。但因為我們不能在透鏡上做無限的sample，所以我們為了有效率的sample，我們仍採用相同的策略，也就是隨機分布式的ray。

Figure 13 Depth of field occurs because the lens is a finite size. Each point on the lens "looks" at the same point on the focal plane.

其實，針孔相機可以視為透鏡相機的特例，也就是只使用透鏡中心位置的透鏡相機，另外，透鏡相機除了可以產生景深這種效果，其實還可以讓sample到更多物體，因為他的ray可能可以剛好繞過擋在前面的物體，但如果是針孔相機，那就只能sample到最近的那個點了。

. . .

Motion Blur

這個效果也是來自於現實相機，原因就是曝光時間過長，所以成像平面上的一個點被多個點所照射，也就是多對1，這就會造成模糊。

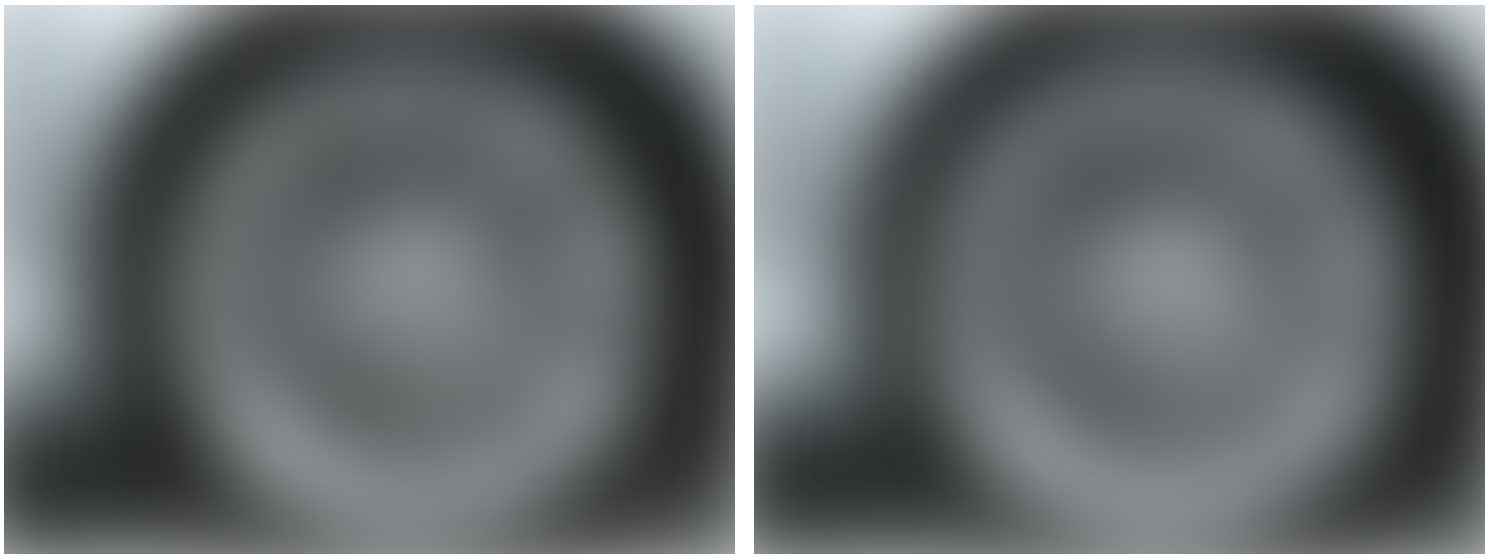


Figure 14 Motion blur

要做出motion blur最直觀的做法就是做一些blur就好，也就是某些區塊做不同mask的filter就好，但是困難點就是你要怎麼選取區塊，由於深度也不是簡單的幾個位數，所以要區分出所謂前後景分別做blur是相對複雜的，但利用這篇paper的架構就是相對簡單的，就是在時間上作隨機分布式ray，最後再把結果blur起來，因為理想上是我們在不同時間點上render在平均，但是這樣成本太高，所以我們用有效的sample來取代，也就是隨機分布式。

. . .

最後把所有sample個別平均起來就好了。

很明顯的雖然說是有效率的sample，但是究竟還是oversampling，尤其是在每個階段都要射出多個ray，這樣子的ray會是指數成長的，也就是只要遞迴的深度一多，整個算法就會很慢。所以才會有後面paper的改良，但大方向都是相同的，就是要更有效率的sample。

. . .

可以額外提一下的事，雖然論文原文並沒有提到蒙地卡羅法(Monte Carlo method)，但是後人都會為他附會上他是蒙地卡羅法，而就概念上，我們在shading model上有展示，任意點的顏色或者說是Intensity都是一個積分式，但對於這樣一個針對所有角度的積分式，並且場景複雜的情況下，這樣子的積分可以想像成無限複雜的多項式，而對

但蒙地卡羅法就在此時登場，他的error只與採樣數量有關，而與維度無關，而蒙地卡羅法的核心就是在一個機率分布下進行採樣，而這個機率分部則會是一個給定的隨機，所以回到整個論文，我們可以發現做的事情就是我們給定一個隨機的分布，並在這個分布之中進行採樣，而這麼做就跟蒙地卡羅法是一樣的。

. . .

Reference

[1]Distributed Ray Tracing by Allen Martin

[2]Distributed Ray Tracing

[3]Stochastic Sampling in Computer Graphics, ROBERT L. COOK

[4]CS319 Advanced Topics in Computer Graphics, John C. Hart

[5]CS655

[6]Modeling Motion Blur in Computer - Generated Image, Dmitriy Bespalov

[7]Dither wiki

[8]PBR Book 3rd chapter 7.3

[9]你沒見過的景深影響因素光學原理圖解

[10]3分鐘搞懂光圈景深原理

Computer Graphics Ray Tracing