什麼是光線追蹤 (Ray Tracing)？它如何實現即時 3D 圖形？

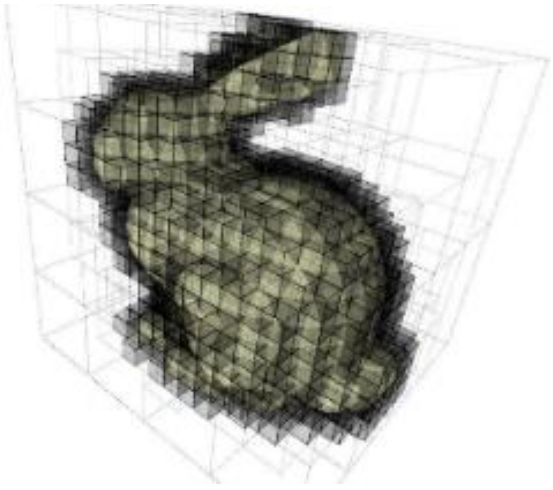
作者：Kristof Beets，Imagination Technologies PowerVR 產品管理資深總監

我們所處的世界充滿著由太陽或其他人造光源發射出的數以億計的光線。當光照射到物體表面時，光線會以各種方式反彈、分散和反射，直至最終到達我們的眼睛。正是這種複雜的交互作用創造了我們的現實「視界」。光線追蹤是一種用於 3D 圖形的照明技術，它可以模擬真實世界中的光線照射方式。雖然它能產生最逼真的效果，但是從傳統上看，其過程對於電腦而言，還是過於復雜，以致無法即時生成 3D 圖形。

如今，它被廣泛用於為廣告和電影，創造出超逼真的渲染效果，但是在這些應用中，即使利用當今功能非常強大的計算服務器，生成每一幀數據仍需花費數小時。光線追蹤是個時髦詞，作為即時繪圖技術的發展方向，它令人們感到非常興奮（也有人將其視為炒作，這取決於你的看法）。在本文中，我們將認識光線追蹤，並了解實現它的方法。

簡化問題

在 3D 遊戲中，場景由各種物體組成，當這些物體組合在一起時，就會形成數百萬個三角形。光線追蹤最基本的功能就是發射出一條光線，然後沿著它在 3D 場景中的路徑來定位它所到達的第一個物體，進而確定應該如何對該物體進行照明。然而，用場景中的每個物體去測試一條光線，以確定它們是否相交，這樣的做法效率太低、運算成本太高，完全無法即時進行。因此，為了使用光線追蹤技術，我們需要解決這一問題。



使用場景層次結構將兔子劃分到多個小方框中

這可以透過建立光線追蹤加速結構來實現，為了做到這一點，我們可以圍繞整個遊戲場景繪製一個方框，然後將其劃分為多個較小的方框，再將這些小方框細分為更小的方框，我們按這種方式不斷將方框進行細分，直到小方框中的三角形數量達到便於管理的程度為止。我們

將其稱為場景層次結構，它幫助我們將問題簡化到一定程度，使現有的圖形處理器可以有效地進行處理。

這種方法之所以有用，是因為當我們向遊戲場景中發射光線時，可以根據場景層次結構逐層進行檢查。首先，我們要檢查光線到底有沒有射入最大的方框（即我們的場景）。如果有，我們將繼續檢查下一層的小方框。在這個階段，我們會發現光線射入了一些方框，但沒有射入另一些方框。接下來，我們可以不斷將光線未射入的方框排除在外，只重點關注光線射入的那些方框，直至我們找到光線與三角形相交的地方。至此，我們終於找到了自己的目標。

這種層次結構使我們能夠找到光線和三角形最近的交點，而不必測試場景中的每個三角形。這極大地簡化了問題，因此可以更快地完成處理過程。

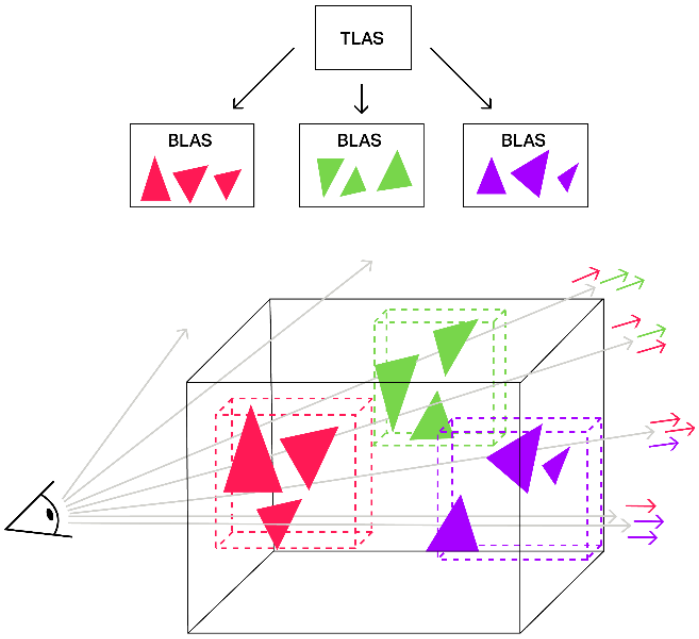
在幾何處理階段（該階段硬體會完成物體的動畫處理工作）之後，我們會將那些三角形置入一個稱為場景層次生成器的專用硬體中，該生成器可以生成上面所述的加速結構。我們還添加了一些專門的光線/方框/三角形測試器，它們是專用的固定功能硬體，透過加速結構進行光線追踪，並確定光線與三角形的交點。與使用軟體可編程管線相比，在專用硬體中完成所有這些操作要快得多，而且更節省面積和功耗。

那麼，當硬體確定光線照射到一個三角形之後，下一步該做什麼？接下來我們會觸發一個片段著色器，這是一個小程式，可以確定該三角形特定位置的顏色，這一步和傳統的渲染方式基本相似。然後，通過該片段著色器程序，我們將更多的光線發射到 3D 場景中，隨著此過程的不斷重複，就可以建立起我們的光線追蹤場景。

一致性問題

但是，現在我們又有一個新問題，我們向場景中發射了大量光線，那麼該如何高效地進行所有處理工作呢？我們需要從記憶體內的加速結構中獲取方框和三角形，並且當每條光線每次照射到一個物體上時，都會觸發一次片段程序。

不幸的是，光線是不穩定的，它們不一定會沿著同一方向傳播。在專業術語中，我們將此描述為不一致性—這會帶來問題。不一致的資料存取對現代的圖形處理器（GPU）來說是不利的。這有點像在按字母順序排列的名片簿中尋找資訊，但給我們的名字，卻是完全隨機排列的—代表我們必須不停地來回翻找，會佔用寶貴的時間和精力。

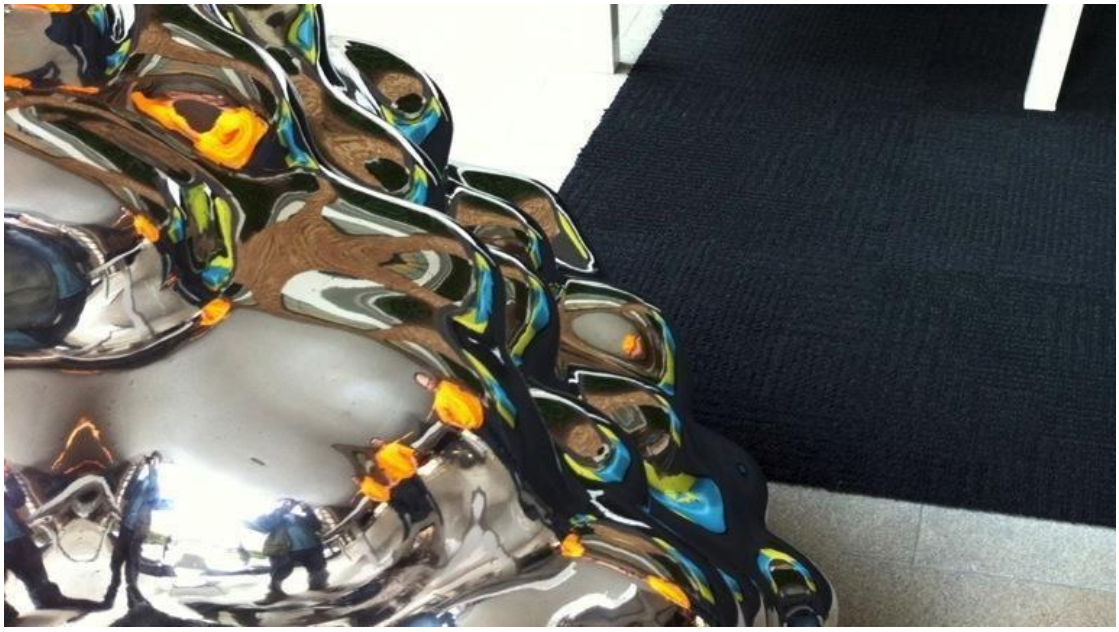


光線追蹤加速結構

更糟糕的是，當光線隨機地向四處反彈時，它們還會照射到不同的物體和三角形上，這些物體和三角形需要分別著色和添加陰影，這將觸發不同的著色程序。然而，GPU 喜歡以並行方式處理著色器。這正是 GPU 的強大之處：以大規模並行方式處理數據的能力使其比其他處理器（如 CPU）更具優勢。這是因為 GPU 的算術邏輯單元（ALU）本質上採用了單指令多線程（SIMT）方式。但是，如果每條光線會觸發一個不同的著色器，那麼將無法在 GPU 上運行，因為這需要多指令多線程（MIMT）架構，該架構在晶片面積和功耗方面的效率都很低。

針對此問題的一種解決方案是採用 Imagination Technologies 開發的一致性引擎，該引擎可以追蹤光線，並且在場景中所有混亂的光線之間找到秩序。

如果你看下面的圖片，一開始可能會覺得光線是隨機的。但是，如果你更仔細地觀察，會發現實際上是存在一致性的。



為了更清楚地對此進行解釋，請注意圖中物體的某些部分是如何反射相同黃色對象的。儘管看起來很混亂，但還是會發現有一些光線是沿著同一方向傳播的，並照射到了相似的對像上。我們的一致性引擎會對此進行查尋，並將這些光線分組，進而使它們更易於被 GPU 處理。這就是「魔法」，我們重新實現了高效的資料存取和執行，從而降低了處理的功耗以及對頻寬的需求。

混合渲染的好處

太好了，我們現在可以高效率地進行光線追蹤了。然而，正如我們前面所說的，現實世界中會有數以億計的光線向四面八方反射，從而形成我們眼睛看到的圖像。因此，即使考慮到我們實現的所有效率提升，使用光線追蹤來建立整個場景仍然是有問題的，那麼解決方案是什麼？混合渲染！

雖然傳統的柵格化渲染是一種很好的方法，但它卻受限於空間交互問題，例如燈光/陰影、反射和折射—而這些複雜的事情正是光線追蹤所擅長的。通過混合渲染方法，我們可以同時利用兩者的優勢，對簡單的物體使用柵格化渲染，然後從著色器發射一些光線，並有選擇性地創造數量有限的空間光線追蹤，進而生成超逼真的陰影、照明效果和精確的反射。通過使用這種混合渲染方法，我們極大地減少了所需追蹤的光線數量，這最終使我們實現了即時性能。

手機上的光線追蹤：真的會實現嗎？

答案很簡單：是的，會實現。如今智慧手機中的 GPU 相比其首次推出時，已經有巨大的進步，這不僅體現在功能方面，在實際性能方面亦是如此。事實上，高階智慧手機已經突破了 1 TFLOPS（每秒萬億次浮點運算）的計算壁壘，而這曾是專業遊戲機的專屬能力。這其中的核心問題是效率，智慧手機依賴於電池續航時間，而光線追蹤相比傳統的渲染方法更高效，因此它很有可能會很快地被添加到行動裝置體驗中。

利用上述創新，Imagination 可以實現高效的光線追蹤。在智慧手機中，使用傳統的柵格化方法在遊戲中「偽造」陰影和反射的成本非常高，在 Unity 或 Unreal 等現代遊戲引擎中，反射是使用 Cascaded Shadow Maps(CSM)生成的，這需要多次渲染屏幕中的幾何體，並將陰影貼圖查找表寫入記憶體中，所有這些操作都會消耗週期和頻寬，並佔用大量的 GPU 和系統功耗。

通過使用光線追蹤，我們可以向光源發射一條光線，如果該光線碰到了光以外的任何東西，我們就知道該片段處於陰影中。所以，使用我們簡化且高度優化的光線追蹤解決方案會簡單得多，相比 Cascaded Shadow Maps(CSM)所需的預先處理，它是一種功耗更低的解決方案。

在分析我們自 2016 年以來的原型光線追蹤硬體時，我們發現相比陰影、反射和其他技術，光線追蹤的功耗通常不到一半，但獲得的品質卻高得多。這裡需要意識到的是，一種複雜但「虛假」的技術，比簡單的光線追蹤技術功耗更高，而光線追蹤技術實現的效果卻逼真得多，這使得它不僅適合現代的高階智慧手機，而且是非常理想的選擇。

人工智慧和超解析度

雖然在智能手機中應用光線追蹤是一種選擇，但我們同樣對雲端遊戲的日益普及感到興奮，這得益於 5G 網路和邊緣計算的發展。在雲端遊戲中，我們的光線追蹤架構所實現的頻寬和功效很可能也是至關重要的。

我們需要不斷地進行創新，才能以更少的成本實現更多的成果，因此，我們對人工智慧（AI）處理的飛速發展感到非常興奮。再加上神經網路，它們可以與光線追蹤結合使用，進而提供更高的效率。例如，當我們為了提高效率僅追蹤相關光線時，可能會得到含有噪聲的結果。神經網路在降噪方面有很好的前景，可以利用學到的「智慧」來填補缺失的細節。這和現實中的工作方式是一致的，因為我們的大腦也會填補有限的人類視覺系統留下的許多空白。

神經網路可以用來提高圖片品質，同時無需使用更高的解析度

另一個極具潛力的概念是超解析度。它同樣是利用神經網路的能力，智慧地學習如何填補缺失的細節，以支持 GPU 以較低的分辨率進行渲染，從而提高性能並降低功耗，同時仍然保持視覺品質。

對於未來

毫無疑問，實時光線追蹤擁有光明的前景，這對於任何對 3D 圖形感興趣的人來說無疑是令人興奮的。由於光線追蹤基於真實世界的物理原理，因此可以提供最高水平的真實感，同時相比我們迄今一直在使用的技巧和近似方法，它還可以提供很高的效率。低功耗的柵格化圖形處理、開創性的光線追蹤操作，再加上人工智慧和神經網路的持續創新，將所有這些結合在一起，有助於將圖形處理提升到一個新的高度。

--

不要錯過 Imagination 的消息，您可以在 Twitter，Facebook 和 LinkedIn 上關注我們。

點擊訪問我們的網站去了解更多有關技術：https://www.imaginationtech.com/ray-tracing/