即時光線追蹤法的實作與效能分析 - 以 UE4 和 UE5 實作為例

108703017 資科三 邱彥翔

指導教授：紀明德

(一) 摘要

Unreal Engine 是 Epic Games 公司發布的遊戲引擎，其特點主要以彩現(Render)效果著名，是全球知名的遊戲引擎之一。光線追蹤(光追)技術是計算機圖學中的特殊彩現演算法，該演算法本質為沿著光線路徑去追蹤光源，進而解決光柵化難以解決之全局光照問題，使呈現之畫面逼近真實世界的光照效果。

Unreal Engine 5 (UE5)的關鍵技術：Lumen，一個動態全局光照和反射系統，是本研究主要探尋的方向。本研究主要分為兩部分：第一部分會先使用 UE5 的 Lumen 演示多個材質與多組場景，對照組使用 Unreal Engine 4 (UE4)實作，分析兩者場景之彩現效果與執行效能，系統性的探尋 Lumen 技術優勢及不足之處；第二部分將對於 Lumen 半透明體全局光照彩現的低品質開發一套解決方案以優化成像效果。

(二) 研究動機與研究問題

因為個人電腦的普及，遊戲逐漸進入人們視野，同時隨著電腦軟硬體配置與科研技術的新發展，遊戲產業亦逐漸多元化，畫面從黑白到彩色、視角從 2D 到 3D，遊戲複雜化的同時，玩家對於畫面質感的追求亦趨擴大，而光追技術也逐漸進入人們視野。

光源照射至物體表面稱為直接光照，直接光照經多次反射、折射照亮其它區域是為間接光照，而全局光照則是直接光照與所有間接光照的總合。過去遊戲的 3D 物件使用光柵化彩現，環境光的間接光照用同一理論往各方向模擬，產生的陰影稱為 AO(環境光遮蔽)，其可以提升畫面立體感，是一種全局光照的極粗略模擬。隨著光柵化技術逐漸進步，AO 變成 SSAO(螢幕空間環境光遮蔽)，是一種即時且可近似 AO 的技術，遊戲中靜態物體可預先烘焙光照貼圖以實現 SSAO，可達到極好的陰影效果，動態物體則無法預先操作，通常使用一些固定後期貼圖達成品質較低的 SSAO，雖然光柵化也有一些手法可以實現高品質的全局光照，但其不保證能達到良好的效果，可能產生諸多問題需要解決[11]。

幾何光學中忽略光的波動性，將光簡化成直線，此一假設也可套用於計算機圖學，利用此特點進而簡化著色過程，而光追就基於以上物理特性而衍生，把相機與物體連線並追蹤其與光照間的關係，進而加以著色彩現。動畫與電影等領域已長期使用光追技術彩現畫面，使其畫面達到逼真，然而這種方法為人垢病之處就是需要強大算力及耗時，所以一直難以用在即時遊戲上(30 FPS 可稱為即時)。

近年來，因為技術與算法的革新，即時光追從理想變為可能，原先的光柵化也加入了光追，變為直接光照採用光柵化，間接光照使用光追的 Hybrid Renderer，主流遊戲及多款應用程式開始支援即時光追，並且建立了多個跨

平台 API 標準以利廣泛使用。Epic 公司也將 UE4 加入實時光追功能(圖 2)，對比原先光柵化彩現的方式(圖 1)，其中幾處間接光照的呈現，如紅圈標示透明燈罩及箭頭指向之鏡子反射，在光追的幫助下，都能被極佳的即時彩現，然而 UE4 的即時光追極度著重硬體配置，高階 GPU 又逢挖礦風潮，價格水漲船高，導致不少人望而卻步。2020 年 UE5 搶先體驗版釋出，提出新的動態全局光照解法– Lumen，為一種基於 SDF(Signed Distance Field)實現的軟體即時光追，於示意圖中(圖 4)與 UE4 的光追(圖 3)相比更加明亮，搭配 Nanite[17](另一項 UE5 關鍵技術，非本研究重點)在相同硬體下能有更優於 UE4 的效能表現。

圖 1 : UE4 Raster 彩現[12]

圖 2 : UE4 Ray Traced 彩現[12]

圖 3 : UE4 Ray Traced 彩現[20]

圖 4 : UE5 Lumen 彩現[20]

雖然 UE5 的效能優於 UE4，但其彩現表現仍有待評估與測試，本研究第一部分將使用 UE5 的 Lumen 技術，測試 Lumen 的成效表現，對照 UE4 的即時光追 Hybrid Renderer，使用多個材質、多組場景同時測試其效能與彩現效果，藉此知悉 Lumen 技術不足待改進之處，以利日後使用該技術開發遊戲與動畫時能清楚了解何種材質應該避免或如何避免及其於遊戲中之實用程度；第二部分將使用 UE5 嘗試開發一個 Plugin，以此改善 Lumen 對於半透明物體全局光照的低品質彩現[6]。

(三) 文獻回顧與探討

光線追蹤

光線追蹤基於光路可逆性，從實際的光線終點處(眼睛/相機)發射光線，這種方法很好的解決了光柵化的問題，也不再需要多重運算複雜的 Z-buffer，而只需與光線射出至最近的物體表面求交並運算。

光追可以追溯到 1968 年，由 Arthur Apple 提出 Ray Casting [1]，是為了能擁有逼真的真實幻覺，解決可見性及深度問題，並能體現陰影與紋理的一種方法，其作法是從相機的每個像素開始投射光線，僅考慮光線投射一次，在可視範圍內尋找最接近物體的像素，將其與光源做連線(Shadow Ray)，判定是否為光源可見，並以此判斷產生陰影與否，之後返回相機各像素內進行著色，是使用計算機運用光追概念生成陰影圖片的第一人。

Apple 的方法只考慮光線彈射一次，Turner Whitted 提出 Recursive Ray Tracing[2]，用遞迴的光追方式呈現了光線多次複雜的反射及折射，一樣從相機出發，將每個碰到物體反射與折射的點逐一與光源連線，計算是否於陰影內，最終加總其對最終成像的貢獻再投射於相機的像素內著色，將 Apple 的方式進行了更進一步的改良，為光追技術吹響啟蒙的號角。

James Kajiya 提出 The Rendering Equation(1) [3]，在當時算是一個跨時代的公式，描述了光線的傳播方式，其中包括了考慮了物體自發光，在半球內所有的入射光大小、入射角與 Bidirectional Reflectance Distribution Function (BRDF 是基於輻射度量學的概念，衡量介質從入射方向收集之能量並反射至另一方向的能量大小，定義了各種材質於不同方向的反射能量大小)。

$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_{\Omega^+} L_i(p,\omega_i)\, f_r(p,\omega_i,\omega_o)\,(n\cdot\omega_i)d\omega_i \qquad (1)$$

單一像素反射 = 自發光 + 半球內積分[入射光 * BRDF(材質與方向) * 入射角餘弦]

實際的光線追蹤碰到物體可能反射出多條光線，如此反覆呈指數生長，造成了運算上的耗時；路徑追蹤則是光線追蹤的一種形式，使所有光線每次反彈僅產生一條光線，而產生之光線又將重覆進行反彈直至終止條件，並且隨機採樣形成最終之圖像，但若每個像素都只有一條光線，則會生成 noisy(噪聲)相當嚴重的圖像。

The Rendering Equation 的問題是不定積分內部過於複雜，難以計算，而 James Kajiya 也提出多種解決方案，其中 Monte Carlo Path Tracing(蒙特卡羅路徑追蹤)的解法將不定積分轉成機率形式的定積分，以合乎物理邏輯的方式，巧妙的解決了問題[3]，並得以彩現出接近真實的圖片，近代的光追技術都奠基於此算法。其做法是針對每個像素在積分域中進行多次的隨機採樣，最後加總求平均，並加入俄羅斯輪盤的概念，既可以為遞迴式設置終止

條件，亦可以維持期望值。該算法好處是利用機率統計理論的概念，只需要知道採樣區域之 Probability Density Function ((2) 之 $pdf(w_i)$)，即可求得反射近似。

$$L_o(p,\omega_o) \approx \frac{1}{N}\sum_{i=1}^{N}\frac{L_i\,(p,w_i)\,f_r\,(p,w_i,w_o)\,(n\cdot w_i)}{pdf(w_i)} \tag{2}$$

自 Rendering Equation 提出的 20 年來，學界一直在努力研究快速蒙特卡羅路徑追蹤的方法[7][8][9]，因為在隨機採樣下每像素需採樣數百至數千次才能使圖形收斂，這無疑是項龐大耗時的工程。2018 年，DirectX12 及 Vulkan 宣布支持光追，NVIDIA 亦針對光線追蹤推出 RTX 系列圖形處理器，使得每種光追演算法皆比上代的 GTX 顯示卡快上不少[10]，並且 RTX 20 系列顯卡使用了 DLSS(Depth Learning Super Sample)，以人工智慧與深度學習的方式進行超採樣，大幅提高幀率與畫質，使整體性能提升，這些技術都將硬體即時光追推向風口浪尖。

Unreal Engine 4

UE4 的即時光追實現大略分為三部分[4][5]

1. 將光線追蹤的實現方式集成至基於光柵化的即時彩現引擎。
   i. 建構光線追蹤的幾何體，以利更改時可構建或更新加速結構。
   ii. 建立 Hit Shader(圖 5)，並使其與 Unreal Engine 的 Existing Vertex Shader 和 Pixel Shader 相容，使用光柵化的方式，讓光線命中物體時計算材質參數。
   iii. 建立 Ray Generation Shader，跟蹤光線並產生陰影或反射。
2. 將程式重構，把原先的 Immediate Mode Style 改成 Retained Mode Style，讓光線追蹤僅在特定幀更動時進行更新，可大幅提升效率。
3. Partitioned Light Path Filtering，低採樣並為每個不同類型所生成之光線，包含反射、漫射等客製化過濾器(去噪方法)，同時搭配局部性質來提高降噪的彩現品質，如此就能以少量光線即時呈現接近離線彩現時耗時的圖像。

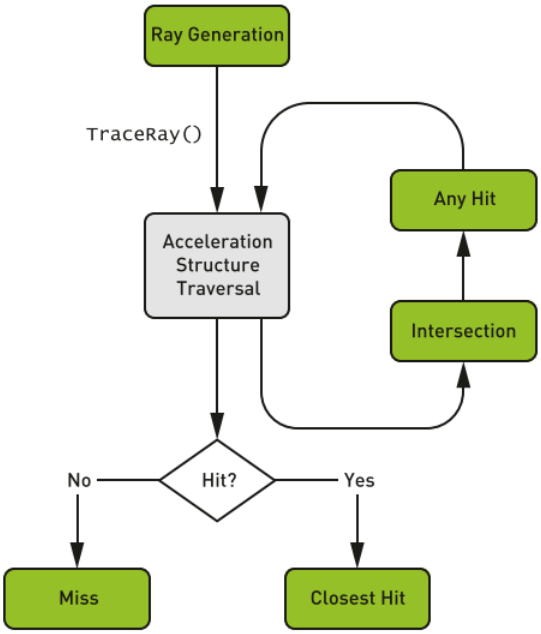

圖 5 : Hit Shader 流程示意圖[4]

Unreal Engine 5

在使用 UE4 的即時光追技術設計場景時，需先為靜態場景烘焙光照貼圖，使反射、陰影等效果更加柔和(Soft Shadow)，接著利用光柵化生成動態直接光照，並使用 SSGI(Screen Space Global Illumination，屏幕空間全局光照)添加動態間接光照[13]，雖可即時達成動態光照，但為龐大的靜態場景預烘焙時間往往需花費數小時甚至數天才能看到場景呈現效果，造成開發者的時間浪費，而 UE5 的 Lumen 不再需要預先烘焙貼圖，將靜動態場景的光照皆推向即時化，使開發效率大幅提升(Lumen 不支援光照貼圖，開啟 Lumen 時所有靜態光照會被禁用，光照貼圖也會被隱藏[14])。

Lumen 的實現大略分成兩部分(圖 6)[6][15][16]：

1. 離線生成(靜態模型匯入或修改後自動非同步生成，無需 Build 光線)
   i. 為靜態模型生成 Meshcard

      在每個 Meshcard 上獲取位置並取得各 Mesh 的材質，依材質生成 Surface Cache(僅覆蓋相機方圓 200 公尺內)並尋找靜動態光照，因此單一 Mesh 的材質不可過於複雜，否則將導致其失準。

   ii. 靜態模型生成 MDF(Mesh Distance Field)

      若 Meshcard 沒有精準的取得 Distance Field(可於場景中用 MDF Visualization 模式檢查)，可能導致漏光，此時可用 MDF 改善。

   iii. 靜態模型生成 GDF(Global Distance Field)

      雖然 MDF 呈現的畫面較好，但基於效能考量，離 Mesh 較遠處光線使用 GDF，較近處使用 MDF(預設 2 公尺內)。

2. 即時生成

i. LumenScene 更新

分為 FScene 的靜態模型(Lumen 離線生成的數據)及 Atlas Textures(Lumen 離線生成數據的 Cache)，這些數據與 Cache 在 Multiple Pipeline 中利用不同的的 LumenScene 更新程式可同步執行。

ii. 計算 Lumen 光照並生成 Irradiance Cache

使用三種資料結構並依精準度大小儲存光照 Cache，Meshcard 生成之 AtlasTextures (高精準度)；Voxel 3D Clipmap(中精準度)；GI Volume(低精準度，通常用於 Volume Rendering)，利用這三種類型的資料可覆蓋場景中所有的靜動態物體光照資訊。

iii. 使用 Cache 資訊計算最終的間接光照

對於最終的間接光照，依據物體與相機的距離區分為 Detail MeshCard Trace(~ 40 公尺)；VoxelLighting Trace(~ 200 公尺)；Distant MeshCard Trace(~ 1000 公尺)；SSGI(全範圍)。這些間接光照的計算並不嚴格拆分，對於細節較多處，Lumen 會採用較佳的呈現方式，以此平衡效能及彩現的效果。

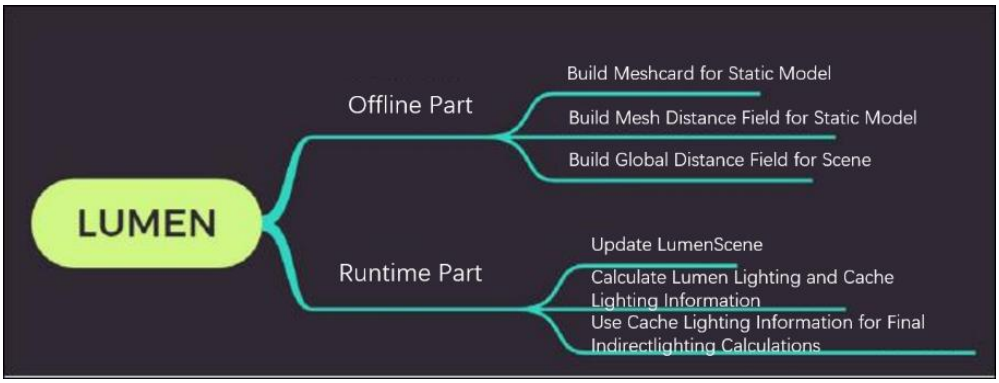

圖 6 : Lumen 主要流程與架構[15]

Lumen 底層的本質可以總結為軟體光線追蹤的加速器，其首先與 Z-buffer 建立 Screen Tracing(螢幕追蹤)；若未找到光線接觸點或光線跑出螢幕外則計算 Shader 和 SDF；對於接觸點使用 Surface Cache 進行處理。

(四) 研究方法與步驟

第一部分

希望能體現 UE5 裡 Lumen 的優勢及不足之處，對比 UE4 之彩現效果，並對各種材質與場景的畫面及效能做出對照分析，比較軟體光追與硬體光追在實作面上呈現之效果與效能差異。

1. **查閱 Unreal Engine 開發資源**：藉由官方文檔或網路資源熟悉 UE4 與 UE5 對於建立場景的框架與介面操作。
2. **蒐集各種材質素材**：利用可取得之材質素材進行單一或多個點光源及面光源的光照對照分析，其中包括但不限於植被、天空、雲霧、半透明、透明、金屬或其它單一物件；分析包括物件靜動態與碰撞時的高光、陰影、邊緣鋸齒、動態模糊等，觀察 UE4 及 UE5 在處理光照時的效果呈現。
3. **建立場景測試基準**：建立多套模擬場景於 UE4 及 UE5 中，此處測試不同於步驟二的單一物件，而是融合多組物件的場景，試圖模擬遊戲或現實之場景，並測試效能。最終使用 UE5 建立一套系統，該系統能讓使用者自由進入任一場景查看 Lumen 的光照效果，且能切換不同視角觀察，並同步顯示 UE5 於各場景中的 FPS 表現。
4. **整理與比較**：在 UE4 與 UE5 中統整各材質效果差異與各場景的平均 FPS 效能，並列出 Lumen 呈現效果與效能待改進之處。

第二部分

現今 Lumen 的半透明體全局光照尚有不足之處，將嘗試開發一套基於 UE5 的 Plugin 改善其效果。

1. **研究即時半透明體光照方法**：包括研究近期學界論文及書籍[18][19]、原始碼(在 UE4 硬體光追及 UE5 的 Lumen 中所實現方法)及業界常用手法等等。
2. **研究 Unreal Engine Plugin**：熟悉 UE5 的 Plugin 框架及場景套用之方法。
3. **設計 Plugin 並應用於場景**：最終結果希望呈現一個簡單易懂的介面，幫助開發者在 Lumen 下使用半透明材質能有更好的光照表現。
4. **UE5 的 Marketplace Plugins**：研究成果將嘗試發表至 UE5 中的 Plugin 商店。

(五) 預期結果

本研究預期之結果分為以下二部分

第一部分

比較各基準場景與材質在 UE4 及 UE5 下使用硬體光線追蹤與使用 Lumen 技術之優劣；建立一個以 UE5 為主的系統，該系統支援使用者得以自由切換場景與視角，並能在各場景中移動，便於查看物體使用 Lumen 所產生的光照效果。

第二部分

嘗試開發一套建立於 UE5，可套用於場景中的 Plugin，為半透明體呈現較佳的全局光照效果，並與 Lumen 比較半透明體成像細節與效果之差異。

(六) 參考文獻

[1] A. Appel. “Some techniques for shading machine renderings of solids”. In: *spring joint computer conf.* 1968, pp.37-45. DOI: 10.1145/1468075.1468082. URL: https://dl.acm.org/doi/10.1145/1468075.1468082

[2] T. Whitted. “An improved illumination model for shaded display”. In: *Communications of the ACM*, 23(6). 1980, pp.343-349. DOI: 10.1145/358876.358882. URL: https://dl.acm.org/doi/10.1145/358876.358882.

[3] J. Kajiya. "The rendering equation", In: *ACM SIGGRAPH Computer Graphics*, vol. 20, no. 4. 1986, pp.143-150. DOI: 10.1145/15886.15902. URL: https://dl.acm.org/doi/10.1145/15886.15902.

[4] E. Liu, et al. “Cinematic Rendering in UE4 with Real-Time Ray Tracing and Denoising”. In: *Ray Tracing Gems*. Publisher: Apress, Berkeley, CA. 2019, ch19, pp.289-379.

[5] Github, EpicGames | Unreal Engine, https://github.com/EpicGames/UnrealEngine

[6] Lumen Technical Details | Unreal Engine 5 Documentation, https://docs.unrealengine.com/5.0/en-US/RenderingFeatures/Lumen/TechOverview/

[7] Q. Fang. "Mesh-based Monte Carlo method using fast ray-tracing in Plücker coordinates", In: *Biomedical Optics Express*, vol. 1, no. 1. 2010, p. 165. DOI: 10.1364/boe.1.000165.

[8] A. Gruzdev, et al. "Practical approach to the fast Monte-Carlo ray-tracing", In: *Programming and Computer Software*, vol. 41, no. 5. 2015, pp.253-257. DOI: 10.1134/s0361768815050035. URL: https://dl.acm.org/doi/10.1134/s0361768815050035.

[9] L. Yan. “Physically-based Modeling and Rendering of Complex Visual Appearance”, In: *UC Berkeley Electronic Theses and Dissertations*. 2018, pp.131-150.

[10] 吳紹瑋，《使用光線追蹤在 OptiX 框架下之蒙地卡羅路徑追蹤演算法分析比較》，碩士論文，國立臺灣師範大學，2020。

[11] M. Pharr. “High-Quality Global Illumination Rendering Using Rasterization”. In: *GPU Gems 2*. Publisher: Addison-Wesley, 2006, ch38, pp.616-632.

[12] Real-Time Ray Tracing | Unreal Engine 4.27, https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/RayTracing/

[13] Global Illumination | Unreal Engine 4.27, https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/GlobalIllumination/

[14] Lumen - Global Illumination and Reflections | Unreal Engine 5, https://docs.unrealengine.com/5.0/en-US/RenderingFeatures/Lumen/

[15] UE5 Lumen Implementation Analysis, https://blog.en.uwa4d.com/2022/01/25/ue5-lumen-implementation-analysis/

[16] Github | Unreal Engine 5 Lumen, https://github.com/EpicGames/UnrealEngine/tree/ue5-early-access/Engine/Shaders/Private/Lumen

[17] Nanite Virtualized Geometry | Unreal Engine 5, https://docs.unrealengine.com/5.0/en-US/RenderingFeatures/Nanite/

[18] C. Chang, et al. “Real-Time Translucent Rendering Using GPU-based Texture Space Importance Sampling”. In: *EUROGRAPHICS*, vol. 27, no. 2. 2008, pp.517-526. DOI: 10.1111/j.1467-8659.2008.01149.x. URL: https://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2008.01149.x

[19] T. Zhang. “HANDLING TRANSLUCENCY WITH REAL-TIME RAY TRACING”, In: *Ray Tracing Gems II*. Publisher: Apress, Berkeley, CA. 2021, ch11, pp.127-138.

[20] Lumen in UE5: Let there be light! | Unreal Engine, https://www.youtube.com/watch?v=Dc1PPYl2uxA

(七) 需要指導教授指導內容

協助及指導內容包括軟體：Unreal Engine 界面操作、Plugin 操作及應用、偵測場景半透明體方法、Lumen 半透明體原始碼實現手法等；知識：業界對於即時光追、半透明彩現的技術新知及手法等；硬體：可能會用到圖學實驗室的電腦設備，及其它諸多實作上的問題尚需倚靠教授指導與循循善誘，方能使研究順利完成。