國立臺灣師範大學

資訊工程研究所碩士論文

指導教授：　張鈞法　博士

虛擬實境中基於物理之擬真材質的繪製

Physically Based Material Renderer for Virtual Reality

研究生：陳佑欣　撰

中華民國　一〇六 年 七 月

摘要

基於物理繪圖(Physically Based Rendering)是近年來十分熱門繪圖概念，有別於傳統的上色模型，從物質本身和周遭環境以真實物理的方式去計算光線，如此不但能夠畫出更真實的影像，也方便美術人員可以使用真實物理材質的參數來創作物件。由於參數的算法都是從真實物理的角度出發，擺設場景的時候不太需要去微調數據，只管放置物件，得到的影像也會十分真實。

在即時繪圖領域，需要使用許多的技巧才能簡化這些計算過程，如何去簡化繪圖方程式(Render Equation)才可以快速又不失真，有許多問題需要考量。於此我們將呈現一種即時繪圖的方式，使用和遊戲引擎相同的貼圖和環境光源來繪製場景，並且接入虛擬實境裝置中，實現即時而又逼真的場景繪製。

關鍵字：基於物理繪圖、即時繪圖、虛擬實境

Abstract

Physically Based Rendering became the most popular rendering techniques recently. Comparing with the traditional shading theory, it is more or less based on the same underlying theory which more closely matches to the physical world. Not only render the more photo-realistic image, but also help the artist design based on material's physical attributes. Because the method is based on physics, it's not necessary to tweak some specific parameter for building a scene, we can still get the correct results.

There are lots of hacking methods for real-time rendering. The key point is how to simplify the render equation and still get the acceptable lighting and shading. In this thesis, we introduce an method that using the same type of texture as game engine, we also implement using the environment texture as image-based lighting. And then render the results in virtual reality device.

Keywords: physically based rendering, real-time rendering, virtual reality.

誌謝

念研究所的這兩年是我這一輩子最快樂的念書時光，可以使用自己已經學會的知識去探索自己有興趣的領域，同時感受到這些知識都是自己專業的基石，一點一滴累積起來的充實感真棒。最最感謝指導教授張鈞法老師，其實我一路學圖學都沒甚麼開竅，就是憑藉一股我很喜歡，從大三黏到現在，中間還一度想說大學畢業就直接出去工作好了，是一段有點擺爛的日子。繼續唸書的契機可能世俗到我自己不好意思在這裡說，不過碩一開始重新再跟著張老師學習卻點燃了心中的小宇宙，怎麼能不被精美的動畫影片或是遊戲畫面所吸引呢？兩年過去了，可能可能我現在算是有點東西了，感謝張老師一路引導。

回想一路走來漫長的求學過程，要感謝的人實在太多了，不如就列出來吧！

感謝蔣宗哲老師，總是關心我們的學習狀況，以及不時開導做研究的心態。

感謝實驗室的各位學長、同學、學弟們，威豪、浩庭、柏元、毓緯、昌宇、祐瑄、俞慶、新予、廷賜，圖學的路上，有你們相伴，真好。

特別感謝昌宇、柏元、祐瑄，碩二努力的路上，最常聊的還是你們。

感謝廷翰，的咖啡還有羊羹，以及一些便當。

感謝俊豪，雖然我都叫你連哥，是交大的圖學好夥伴，也是沙盒好朋友。

感謝師大的學長姐、同學、學弟妹，陪我聊天，分享學習經驗。

最後，感謝我的家人和我的朋友們，你們的支持是我最大的動力。

陳佑欣　中華民國壹佰零陸年柒月

目錄

目錄 ..... **V**

附圖目錄 ..... **VI**

附表目錄 ..... **VII**

圖片引用來源 ..... **VIII**

模型引用來源 ..... **VIII**

授權引用聲明 ..... **VIII**

第一章 簡介 **1**

第一節 研究背景 1

第二節 研究目的 2

第三節 論文架構 2

第二章 文獻探討 **3**

第一節 OpenGL 傳統光柵化繪圖 Rasterization 3

第二節 測量光的傳輸 Measurement of Light Transport 4

第三節 基於物理繪圖 Physically-Based Rendering 6

第四節 基於影像光源 Image-Based Lighting 13

第五節 OpenVR API 17

第三章 實作基於物理材質 **19**

第一節 五種貼圖 5 Types of textures 19

第二節 能量守恆 Energy Conservation 21

第三節 基於物理雙向反射分佈函數 Physically Based BRDF 21

第四章 實驗結果 **26**

第一節 實驗環境 26

第二節 測試場景 26

第三節 比較 30

第五章 結論與未來展望 **32**

參考文獻 **33**

相關研究論文 33

附圖目錄

圖 1：GLSL PIPELINE 的各階段流程 3
圖 2：繪圖方程式示意圖 5
圖 3：漫射光(DIFFUSE)與反射光(SPECULAR) 6
圖 4：金屬物質與非金屬物質，金屬物質會吸收所有的漫射光 6
圖 5：距離與次表面散射 7
圖 6：半路向量 H 8
圖 7：左邊和中間這些因為不平滑造成的內部陰影會被右邊的內部漫射給解決 8
圖 8：正確的光能量和粗糙度關係 8
圖 9：菲涅耳效應示意圖，越遠入射角越大，光容易反射；越近入射角越小，光容易透射 9
圖 10：一些物質因為菲涅耳效應產生的反射光比例 10
圖 11：一些物質於垂直表面觀察時的反射光比例及顏色 10
圖 12：環境貼圖經過轉換後變成輻射照度貼圖(IRRADIANCE MAP) 14
圖 13：五種粗糙程度的環境紋理貼圖 15
圖 14：BRDF 的二維查表貼圖(2D LOOK UP TABLE)，橫軸是觀察者視線的餘弦值，縱軸是粗糙度 16
圖 15：HTC 的虛擬實境裝置 VIVE 17
圖 16：漫射光貼圖、反照率貼圖、法向量貼圖 19
圖 17：金屬度貼圖、粗糙度貼圖、環境光遮蔽貼圖 21
圖 18：由左往右，二維的貼圖轉成環境貼圖 22
圖 19：越高緯度的地區，離散積分的面積越小 23
圖 20：反射光路徑和不同粗糙程度表面比較 24
圖 21：一般的隨機取樣序列和低差異序列 25
圖 22：繪製球的模型 27
圖 23：繪製茶壺模型 27
圖 24：繪製頭像模型 28
圖 25：繪製茶房場景，由於無法達成物件全域影響，所以桌面其實是直接反應環境貼圖 28
圖 26：閹割版的 SPONZA 模型，使用的貼圖為最高解析度的貼圖 29
圖 27：高解析度貼圖會嚴重拖累效能 29
圖 28：基於物理繪圖、PHONG SHADING 30
圖 29：基於物理繪圖、CYCLES 繪圖引擎 31

附表目錄

表 1：漢默斯里低差異序列 24
表 2：個人電腦設備 26
表 3：實驗室電腦設備 26
表 4：測試場景相關效能數據 26
表 5：基於物理繪圖和 PHONG SHADING 效能比較 30

圖片引用來源

圖 1：**Lighthouse3d** - Tutorials » GLSL 1.2 Tutorial » Pipeline Overview

http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/pipeline-overview/

圖 2：**Wikipedia** - Render Equation

https://en.wikipedia.org/wiki/Rendering_equation

圖 3、4、5、6、7、10、11：Hoffman, Naty, “Background: Physics and Math of Shading”, part of “Physically Based Shading in Theory and Practice”, SIGGRAPH 2013 Course Notes.

圖 8：**Learn OpenGL** - PBR » Theory

https://learnopengl.com/#!PBR/Theory

圖 9：**Blender Guru** - tutorial » Making Realistic PBR Materials - Part 1

https://www.blenderguru.com/tutorials/pbr-shader-tutorial-pt1

圖 12、19：**Learn OpenGL** - PBR » IBL » Diffuse irradiance

https://learnopengl.com/#!PBR/IBL/Diffuse-irradiance

圖 13、14、20、21：**Learn OpenGL** - PBR » IBL » Specular IBL

https://learnopengl.com/#!PBR/IBL/Specular-IBL

圖 15：**HTC VIVE**

https://www.vive.com/tw/

圖 16、17：**Poliigon -** 材質編號：Tiles13、MetalOxidizedGoldPlating001、BricksOld02

https://www.poliigon.com

模型引用來源

本篇所使用四種模型 Teapot, Head, Crytek Sponza, Dabrovic Sponza 下載來自

Morgan McGuire's Computer Graphics Archive https://casual-effects.com/data

授權引用聲明

第一章 簡介

第一節 研究背景

電腦圖學研究發展之今，大概可以分成兩個大方向，其一是使用極為接近真實物理的方式模擬光線穿梭於物質之中，將蒐集到的資訊繪製成照片，進而將數張照片結合而成為影片。這種繪圖方式我們稱為光線追蹤，繪製過程往往因為光線的特性而相當費時，當加上物體材質本身的特性之後，又會使得計算量劇增，是十分消耗繪製時間的方式。然而儘管費時，此法通常都可以做到媲美照片等級的質感，因此，除了藝術創作，展示設計產品、電影製作也會使用此法，適合不與使用者互動產生回饋的媒體。

另一種繪圖方式，為傳統光柵化繪圖，通常是圖學入門會使用的繪製方式，因為 Khronos Group 提供的 OpenGL 可編成應用程式介面，使得在學習初期就可以快速繪製出不錯的圖片，也很容易做出能夠和使用者互動的三維影像。關鍵在於，光柵化是一種非常簡化的光線計算繪製方式，藉由大量減少甚至是忽略光線打到物體後產生的折射與反射來加快繪製的過程。而另一個被犧牲掉的，是物體的全域性，光柵化所繪製出的物體互相無法影響，就算旁邊有容易反射光線的材質，也不會呈現出來，因而造成失真。

雖然說傳統光柵化繪圖有許多限制，我們還是可以透過許多方式來前處理這些問題，將預先算好的數據存在記憶體中或是貼圖中，在繪製的時候直接去取數

據來使用，節省計算的時間。現今的許多遊戲引擎，就是使用這種技巧，一個材質使用數種貼圖去運算，來盡可能達到逼真，又能兼顧即時繪製畫面。

第二節 研究目的

基於物理材質是近幾年非常熱門的材質計算方式，透過把計算資訊儲存在帶有物理資訊的貼圖中，達成讓藝術創作者容易製造而繪圖引擎容易使用這些資訊的目的，這使得在即時運算繪圖領域也能夠畫出不輸給光線追蹤的材質真實質感。而虛擬實境是近兩年內推出最熱門的新的畫面觀賞模式，能夠有身歷其境的感覺，隨著硬體越來越進步，相信在未來會在設計、遊戲、展示領域有更佳的應用。

本研究將結合基於物理材質於虛擬實境中展示，藉由著色器語言結合新版 OpenGL 程式介面，嘗試實現基於物理材質的即時繪製。最後再比較即時繪製與繪圖引擎畫出來的差異，進而理解即時繪製的極限。

第三節 論文架構

本篇論文總共分成六個章節：第一章為簡介，講述近幾年內即時繪圖的主流，以及虛擬實境的崛起；第二章為文獻探討，講述基於物理材質目前世界研究的成果；第三章為實作基於物理材質所需要理解的數學物理公式以及它們帶來的效果；第四章為實驗的結果與分析，比較基於物理材質與其他繪圖方式的差別；第五章為論文總結；第六章為相關參考資料。

第二章 文獻探討

第一節 OpenGL 傳統光柵化繪圖 Rasterization

光柵化繪圖是電腦圖學中最基本的繪圖方式，在讀入模型檔案之後，會把模型在空間中的座標分布使用數個矩陣轉換成二維平面上輸出成像。而 OpenGL 提供的可編程式流程管線可以讓使用者更簡易地透過程式繪製場景，這套系統主要包含兩大要素，OpenGL 主體程式和 GLSL 著色器語言。

OpenGL 在 3.3 版本以後多是透過設定頂點緩衝物件(Vertex Buffer Object, VBO)來傳送需要被計算的資料以及撰寫著色器語言來完成繪圖，其中常見的有頂點著色器(Vertex Shader)與像素著色器(Fragment Shader)，可以分別對不同的資料作上色處理，達到使用者自訂的效果。頂點著色器會透過矩陣運算出頂點的位置、顏色、貼圖座標以及法向量，並且在之後的階段，傳遞給像素著色器內插出中間的資訊，並且使用這些資訊結合觀察者位置以及光源的位置，去完成更細部的著色。

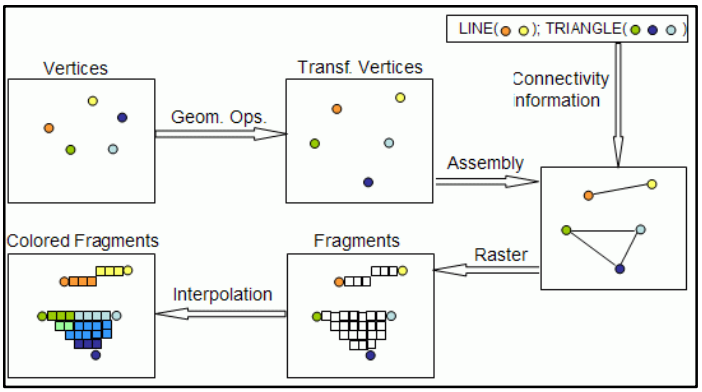

圖 1：GLSL pipeline 的各階段流程

第二節 測量光的傳輸 Measurement of Light Transport

2-2-1 輻射強度 radiant intensity

輻射通量(Radiant Flux)，符號表示 **Φ**，單位為瓦特。

輻照度(Irradiance)，符號表示 **E**，單位入射表面的輻射通量。

$$\mathrm{E(x)} = \frac{d\Phi}{dA}$$

輻射強度(Radiant Intensity)，符號表示 **I**，單位立體角的輻射通量。

$$\mathrm{I(x)} = \frac{d\Phi}{d\omega}$$

輻射率(Radiance)，符號表示 **L**，單位立體角與單位投射表面的輻射通量。

$$\mathrm{L(x,\omega)} = \frac{d^2\Phi}{d\omega dA^{\perp}}$$

其中，輻射率當入射角大的時候較弱，而與平面垂直時較強，故可以改寫成：

$$\mathrm{L(x,\omega)} = \frac{d^2\Phi}{d\omega dA cos\theta}$$

2-2-2 雙向反射分布函數 BRDF

由 B. Duvenhage 於 2013 年整理[1]，當光線從ω_i立體角射入至一平面，其反射比例可以由此函數算出，每種材質都有屬於自己的 BRDF：

$$f_r(x,\omega_i,\omega_e) = \frac{dL_o(x,\omega_e)}{dE(x,\omega_i)} = \frac{dL_o(x,\omega_e)}{dL_i(x,\omega_i)cos\theta_i d\omega_i}$$

此外，基於物理的 BRDF 會具備以下三項特質：

1. 恆正：$f_r(x, \omega_i, \omega_e) \geq 0$

2. 對稱性（亥姆霍茲互換性）：$f_r(x, \omega_i, \omega_e) = f_r(x, \omega_e, \omega_i)$

3. 能量守恆：$\forall \omega_i, \int_\Omega f_r(x, \omega_i, \omega_e) cos\theta_i d\omega_i \leq 1$

2-2-3 繪圖方程式 rendering equation

由 Kajiya 於 1986 發表的繪圖方程式[7]，描述光能在場景中傳遞的強度，遵守能量守恆定律，也符合物理學原理，如今大部分的繪製技術，都是設法去逼近這個結果。其公式如下：

$$L_0(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o) L_i(x, \omega_i)(\omega_i \cdot \vec{n}) d\omega_i$$

最終獲得的射出光強度，為物體本身發光加上半球積分內的反射比例、入射光、入射角度衰減有關。

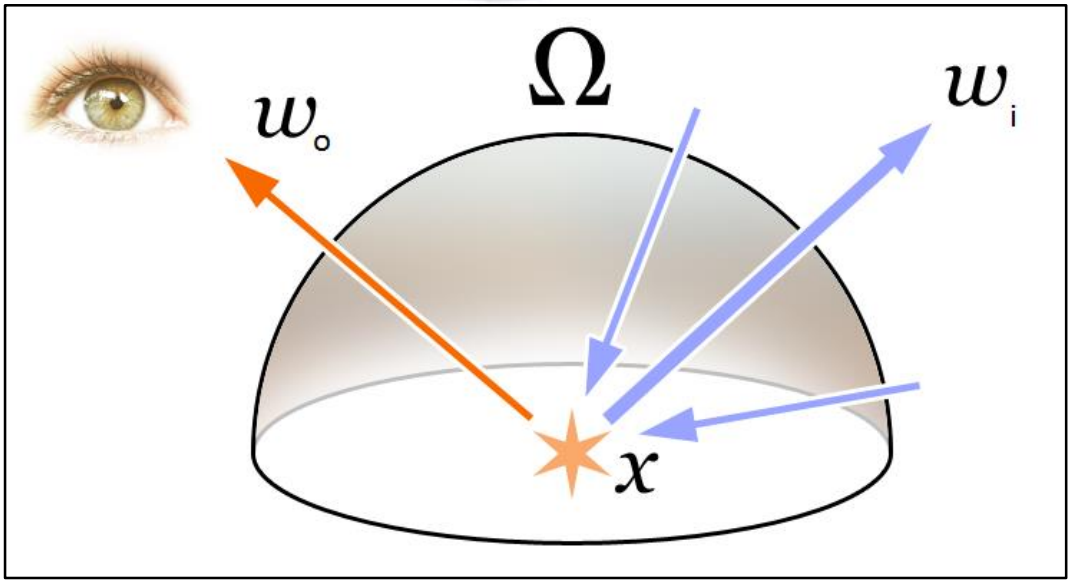

圖 2：繪圖方程式示意圖

第三節 基於物理繪圖 Physically-Based Rendering

有別於傳統貼圖的色彩、法向量、高度圖，即時的基於物理繪圖多了許多必須從物理角度下去考量的要點，Naty Hoffman 於 2013 年 SIGGRAPH 研討會課程中，整理了許多計算細節。[6]

2-3-1 漫反射 diffuse & specular light

入射光線照射到物體後會分成兩個部分，一部分被物體表面反射成為反射光(specular)，另一部分射入物體表面，在表面粒子內經過無數的碰撞，被吸收部分波長的光後無規則地彈射出來，成為漫射光(diffuse)。

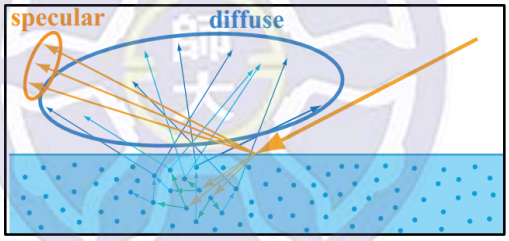

圖 3：漫射光(diffuse)與反射光(specular)

此現象發生於大部分非導體物質。而金屬材質光線也會進入表層內部，只是金屬中的自由電子會吸收所有的漫射光線，只保留反射光線。

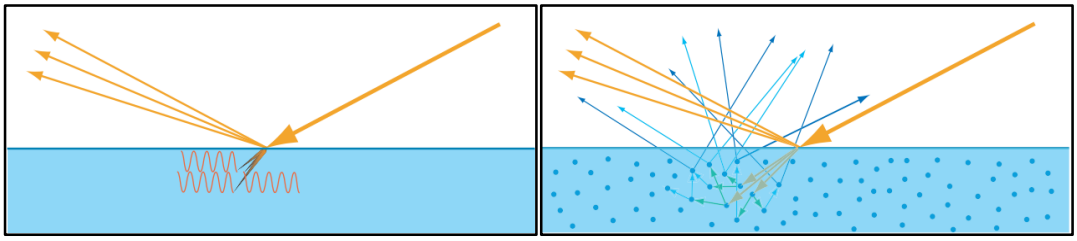

圖 4：金屬物質與非金屬物質，金屬物質會吸收所有的漫射光

2-3-2 次表面散射 subsurface scattering

一些更複雜的物質，像是蠟、皮膚，會使得射入的光線在內部彈射而微微透出物質內部的顏色，然而這種現象其實是和觀察者的距離有關。當觀察距離很遠的時候，次表面散射的光線路徑其實很像是一般的漫射路徑，而距離很近的時候，就算是一般的漫射材質也會有次表面散射的特性。如下圖所示，綠色的圈圈為一個像素的範圍，上面兩張圖可以視為很近距離看物體，就算是發生漫射光也像是次表面散射；下面的圖為很遠距離看物體，光線路線很像是漫射光。

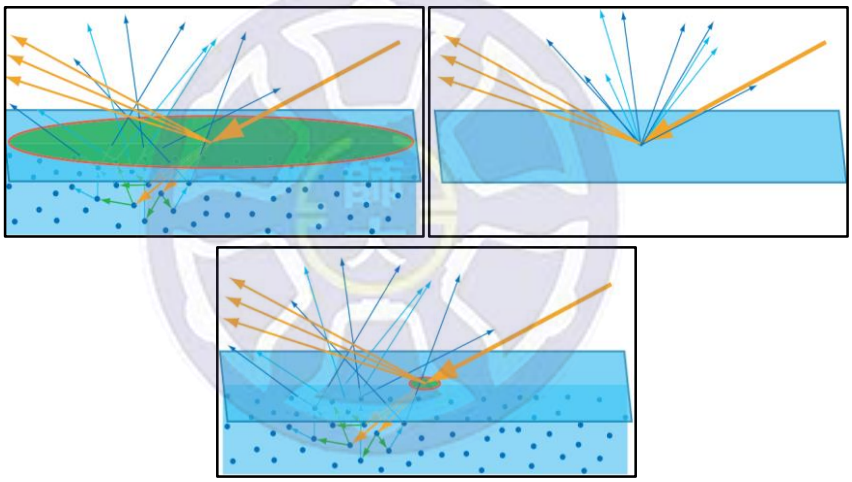

圖 5：距離與次表面散射

2-3-3 微表面 microfacet

真實物體的表面在微觀時絕對不是平滑的，然而實際請美術創作者去刻劃這些細節又太過於困難，因此可以使用一個粗糙度數值(roughness)或是一張粗糙度貼圖表示物體的粗糙程度，當粗糙程度越高時，反射的光線越模糊。

除了粗糙度，還會創造一個半路向量 h (halfway vector)來表示光線 l 和視線 v 的作用程度，沿著此向量可以決定觀察到的微表面粗糙程度。

$$h = \frac{l + v}{|l + v|}$$

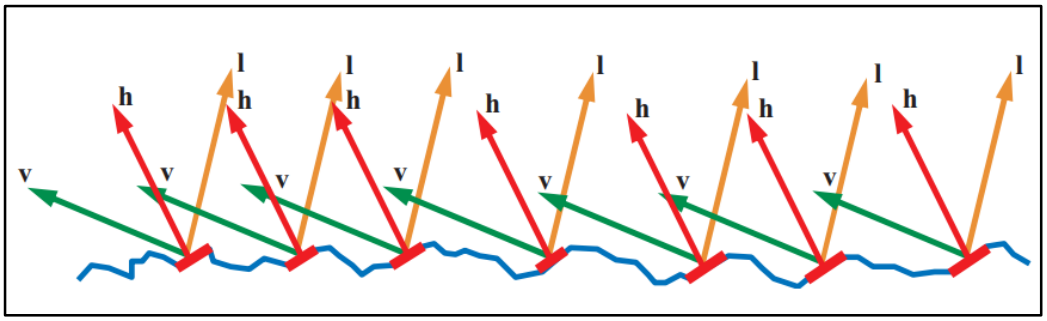

圖 6：半路向量 h

微表面理論無法解釋因為表面粗糙阻擋光線而造成的陰影，但是這些陰影會被其他表面散射的光線給照亮，因此不會造成問題。

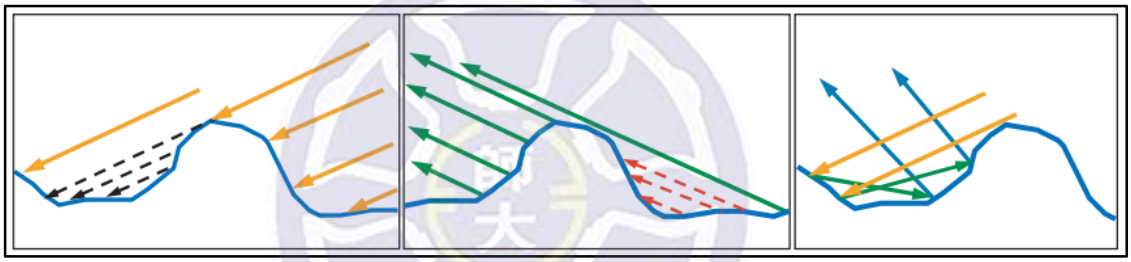

圖 7：左邊和中間這些因為不平滑造成的內部陰影會被右邊的內部漫射給解決

2-3-4 能量守恆 conservation of energy

由於是基於物理性質，反射光能量加上透射光能量絕對不可能超過入射光能量，會有部分的能量轉換成熱能被吸收或是產生次表面散射。當表面的粗糙度上升時，反射光的亮度會下降，其餘整體亮度提升，才不會使得整體能量變多。

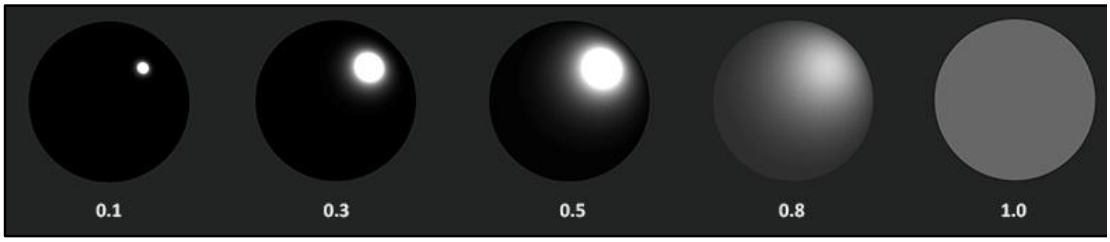

圖 8：正確的光能量和粗糙度關係

2-3-5 菲涅耳效應 Fresnel effect

菲涅耳效應指得是觀察者實際看到的反射與透射比例，會因為視線的入射角大小而有所區別，當入射角趨近於垂直角的時候，物體表面趨近於全反射，而入射角趨近於零度的時候(與物體表面垂直)，物體有最小的反射比例。

圖 9：菲涅耳效應示意圖，越遠入射角越大，光容易反射；越近入射角越小，光容易透射

每個物體會有不同程度的菲涅耳效應，存在著實際的物理數據。對於絕緣體物質，我們可以使用 Schlick 的公式[9]由物體的垂直表面反射率F_0估算出反射比例：

$$F_{Schlick}(F_0, n, v) = F_0 + (1 - F_0)(1 - (n \cdot v))^5$$

$F_{Schlick}$會隨著入射角增加而漸漸從F_0增加至趨近於 1。至於金屬物質，可以拿金屬本身的F_0沿著絕緣體的$F_{Schlick}$曲線逼近出可以接受的結果。大部分的絕緣體物質F_0都介於 2%~5%之間，鑽石擁有絕緣體中最高 17%，半導體和一些非常稀有的物質會擁有 20%~40%的F_0，金屬則都是 50%以上。此外，當物體表面的粗糙程度

增加，菲涅耳效應會急遽地變得不明顯。

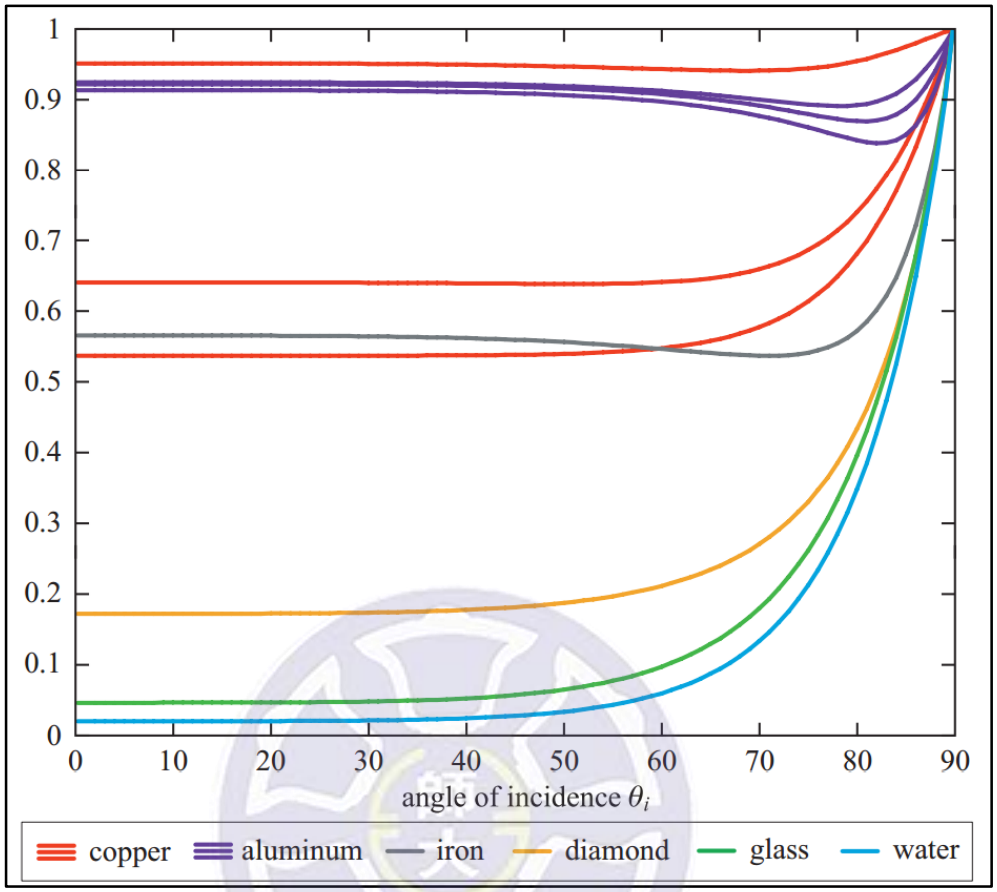

圖 10：一些物質因為菲涅耳效應產生的反射光比例

Material	F_0 **(Linear)**	F_0 **(sRGB)**	**Color**
Water	0.02,0.02,0.02	0.15,0.15,0.15	
Plastic / Glass (Low)	0.03,0.03,0.03	0.21,0.21,0.21	
Plastic High	0.05,0.05,0.05	0.24,0.24,0.24	
Glass (High) / Ruby	0.08,0.08,0.08	0.31,0.31,0.31	
Diamond	0.17,0.17,0.17	0.45,0.45,0.45	
Iron	0.56,0.57,0.58	0.77,0.78,0.78	
Copper	0.95,0.64,0.54	0.98,0.82,0.76	
Gold	1.00,0.71,0.29	1.00,0.86,0.57	
Aluminum	0.91,0.92,0.92	0.96,0.96,0.97	
Silver	0.95,0.93,0.88	0.98,0.97,0.95	

圖 11：一些物質於垂直表面觀察時的反射光比例及顏色

2-3-6 微表面 BRDF microfacet BRDF

由 Brian Karis 代表 Epic Game 於 2013 發表的 Unreal Engine 4 的技術細節[2]，使用了 Lambertian 的漫射光模型[8]結合 Cook-Torrance 的 BRDF 模型[4, 5]，公式如下：

1. $f_r = k_d f_{Lambert} + k_s f_{Cook-Torrance}$
2. $f_{Lambert}(l, v) = \frac{c_{diffuse}}{\pi}$
3. $f_{Cook-Torrance} = \frac{D(h)F(l,h)G(l,v,h)}{4(n\cdot l)(n\cdot v)}$

而 Cook-Torrance 的 BRDF 模型又包含以下三項：

1. 常態分佈式(Normal distribution function)：決定觀察到的表面粗糙程度。
2. 幾何公式(Geometry function)：決定粗糙表面造成的細小陰影。
3. 菲涅耳公式(Fresnel equation)：決定觀察者看到的反射光線比例。

這三個算式都有許多學者研究出非常多種算法，有些接近真實物理表現，有些則加強效能，Epic Game 挑選了以下幾種方式來計算。

常態分佈式(Specular NDF)，反射光 D 項目：

使用同 Disney 的作法[3]，Trowbridge-Reitz GGX 的方式計算。

1. $\alpha = Roughness^2$
2. $D(h) = \frac{\alpha^2}{\pi((n\cdot h)^2(\alpha^2-1)+1)^2}$

幾何公式(Specular G)，反射光 G 項目：

使用 Smith's Schlick GGX 和 Walter 的方式計算[9, 10]。

1. $k_{direct} = \frac{(Roughness+1)^2}{8}, k_{IBL} = \frac{Roughness^2}{2}$
2. $G_1(v) = \frac{n \cdot v}{(n \cdot v)(1-k)+k}$
3. $G(l, v, h) = G_1(l) \times G_1(v)$

菲涅耳公式(Specular F)，反射光 F 項目：

使用 Fresnel-Schlick approximation 的方式計算[9]。

1. F_0為視線垂直表面時，物體的反射率
2. $F(F_0, n, v) = F_0 + (1 - F_0)(1 - n \cdot v)^5$
3. $F(F_0, n, v) = F_0 + max((1 - roughness), F_0) \times (1 - (n \cdot v))^5$

總和上述公式，繪圖方程式可以改寫成如下(去除物體本身發亮材質)。

$$L_0(x, \omega_o) = \int_\Omega (k_d \frac{c}{\pi} + k_s \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}) L_i(x, \omega_i)(\omega_i \cdot \vec{n}) d\omega_i$$

第四節 基於影像光源 Image-Based Lighting

一般繪製時使用的點光源已經可以使用上一節的公式算出，然而要更進一步繪製有真實感的圖像，最常使用的是環境光源，又稱為基於影像光源。此方法整體來說，是放一張高解析度的環境貼圖當作背景，然後讓物件可以使用這張貼圖上面的每一的像素當作光源資訊。

觀察繪圖方程式會發現可以將其拆開成兩個分開的項目，一邊是漫射光，另一邊是反射光，公式如下：

$$L_0(x,\omega_o) = \int_\Omega (k_d \frac{c}{\pi}) L_i(x,\omega_i)(\omega_i \cdot \vec{n}) d\omega_i + \int_\Omega (k_s \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}) L_i(x,\omega_i)(\omega_i \cdot \vec{n}) d\omega_i$$

2-4-1 漫射光 diffuse term

$$L_{Diffuse}(x,\omega_o) = k_d \frac{c}{\pi} \int_\Omega L_i(x,\omega_i)(\omega_i \cdot \vec{n}) d\omega_i$$

公式經過拆分後，常數項目可以提出來，經過觀察後發現漫射光和觀察者的視線立體角ω_o沒有關係，因此可以預先算好漫射光的結果，透過卷積(convolution)的方式，將輻射照度(irradiance map)暫存於一張環境貼圖(environment map or cubemap)當中。值得注意的是，由於 OpenGL 無法像是其他遊戲引擎使用額外的方式(reflection probes)去處理多張環境貼圖，這裡假設點 x 是在場景的正中心位置，因此越接近中心的物體，效果越好。為了要算出卷積，可以把上述公式作以

下修改，改成極座標的積分：

$$L_{Diffuse}(x,\phi_o,\theta_o) = k_d \frac{c}{\pi} \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\frac{\pi}{2}} L_i\,(x,\phi_i,\theta_i)\,cos(\theta_i)\,sin(\theta_i)\,d\theta_i d\phi$$

然後把積分改成離散的方式，即可算出輻射照度的環境貼圖。

$$L_{Diffuse}(x,\phi_o,\theta_o) = k_d \frac{c}{\pi} \times \frac{2\pi}{n} \times \frac{\pi}{2m} \sum_{\phi=0}^{n} \sum_{\theta=0}^{m} L_i\,(x,\phi_i,\theta_i)\,cos(\theta_i)$$

圖 12：環境貼圖經過轉換後變成輻射照度貼圖(irradiance map)

2-4-2 反射光 specular term

$$L_{Specular}(x,\omega_o) = \int_{\Omega} (k_s \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)})\,L_i(x,\omega_i)(\omega_i \cdot \vec{n}) d\omega_i$$

反射光處理起來會比較複雜，原因是 k_s 的比例會受到視線 v 和光線 l 的影響，如果要全部去計算所有情況將會花費太多的資源。Epic Game 提出了簡化方式，使用分割總和(split-sum approximation)的方式可以概略地逼近原公式：

$$L_{Specular}(x,\omega_o) = \int_{\Omega} L_i(x,\omega_i)\,d\omega_i \int_{\Omega} (k_s \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)})\,(\omega_i \cdot \vec{n}) d\omega_i$$

此時觀察前者，會發現和漫射光類似的情況(沒有ω_o)，但是由於是反射光，

會因為材質粗糙程度而改變反射光的強度，因此還是必須使用上述的 NDF 公式來算出粗糙程度影響的比例。在此 Epic Game 又提出另外一項省略的方式，直接假設ω_o垂直於平面，這樣簡化的反射光其實不太容易察覺，而我們可以不用考慮到觀察者的角度，僅需要考慮粗糙程度去預處理反射光的環境貼圖，之後再將環境貼圖存成紋理貼圖(mipmap)。

圖 13：五種粗糙程度的環境紋理貼圖

接下來就只剩下 BRDF 項目的積分而已：

$$\int_{\Omega}(k_s\frac{DFG}{4(\omega_o\cdot n)(\omega_i\cdot n)})(\omega_i\cdot\vec{n})d\omega_i$$

$$=\int_{\Omega}f_r(x,\omega_i,\omega_o)\frac{F(\omega_o,h)}{F(\omega_o,h)}(\vec{n}\cdot\omega_i)d\omega_i$$

$$=\int_{\Omega}\frac{f_r(x,\omega_i,\omega_o)}{F(\omega_o,h)}F(\omega_o,h)\,(\vec{n}\cdot\omega_i)d\omega_i$$

$$=\int_{\Omega}\frac{f_r(x,\omega_i,\omega_o)}{F(\omega_o,h)}[F_0+(1-F_0)(1-l\cdot n)^5](\vec{n}\cdot\omega_i)d\omega_i$$

$$=\int_{\Omega}\frac{f_r(x,\omega_i,\omega_o)}{F(\omega_o,h)}[F_0+(1-F_0)(1-\omega_o\cdot h)^5](\vec{n}\cdot\omega_i)d\omega_i$$

令 $\alpha = (1 - \omega_o \cdot h)^5$

$$F_0 + (1 - F_0)(1 - \omega_o \cdot h)^5 = F_0 + (1 - F_0)\alpha = F_0(1 - \alpha) + \alpha$$

原式可以拆分解成：

$$F_0 \int_\Omega \frac{f_r(x, \omega_i, \omega_o)}{F(\omega_o, h)}[1 - \alpha](\vec{n} \cdot \omega_i)d\omega_i + \int_\Omega \frac{f_r(x, \omega_i, \omega_o)}{F(\omega_o, h)}\alpha(\vec{n} \cdot \omega_i)d\omega_i$$

此公式中，前者可以表示成F_0的比例，後者可視為F_0的偏差量。藉由$n \cdot \omega_o$和 roughness 可以透過此公式，產生一張查表貼圖(Look Up Table)。此表格貼圖中，橫軸代表$n \cdot \omega_o$或是$cos\theta_o$，縱軸代表 roughness，紅色表示F_0的比例，綠色表示F_0的偏差量。接下來，結合上述所有公式即可算出基於圖片擬真的光照效果。

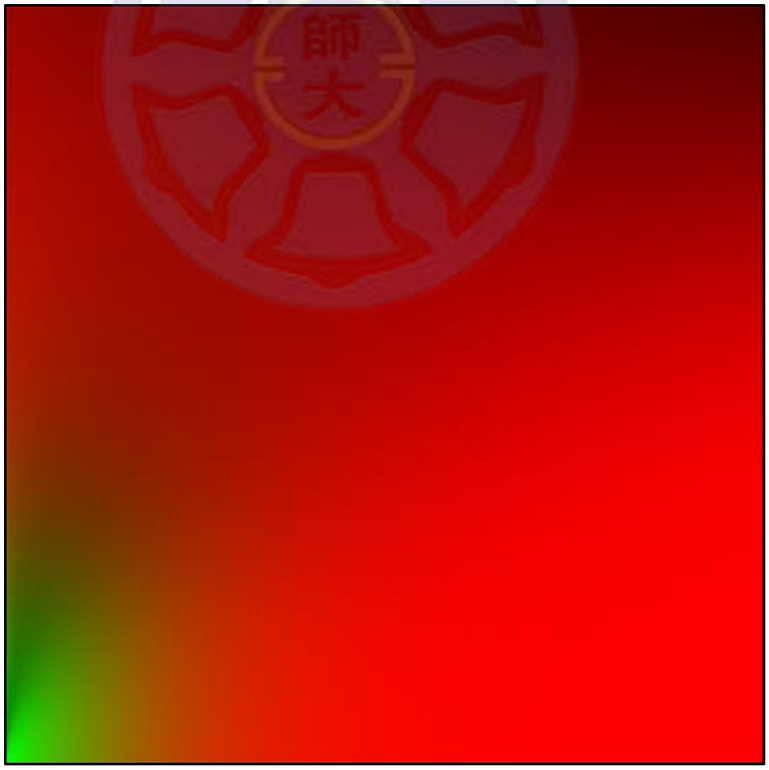

圖 14：BRDF 的二維查表貼圖(2D Look up table)，橫軸是觀察者視線的餘弦值，縱軸是粗糙度

第五節 OpenVR API

OpenVR API 提供了方法可以直接和虛擬實境裝置互動，而不用依賴供應商提供的特殊硬體規格標準程式庫。可以獨立支援軟硬體的更新而不影響遊戲或是軟體本身。

API 是由 C++物件導向概念中的虛擬函式(pure virtual function)實作而成。當有遊戲或是軟體初始化系統後，它會回傳符合該應用程式標準程式庫中標頭檔的介面。

每當有新的版本發布，它同時也會支援過去的版本，因此應用程式端不必強迫更新標準程式庫，簡化程式流程如下頁。

圖 15：HTC 的虛擬實境裝置 VIVE

初始化 Initialization (pseudo code)

[a] 初始化裝置

vr::IVRSystem hmd = vr::VR_Init();*

vr::IVRCompositor compositor = vr::VRCompositor();*

[b] 開 2 個 framebuffers, 2 張 color textures, 2 張 depth textures 給之後雙眼使用

[c] 初始化頭戴裝置變數

vr::TrackedDevicePose_t trackedDevicePose[vr::k_unMaxTrackedDeviceCount];

OpenGL 繪圖迴圈 Main Loop (pseudo code)

[a] 由 *GetProjectionMatrix(vr::Eye_XXX, near_Plane, far_Plane);* 取得 projection matrix

[b] 由三組陣列相乘的逆矩陣算出 view matrix，其中：

body to world 為身體偏移量，推測為校正使用(手動設定)

head to body 由 *trackedDevicePose* 取得

eye to head 由 *GetEyeToHeadTransform(vr::Eye_XXX);* 取得

[c] 至於 model matrix 依然是改變物件位置，和傳統使用方式相同

[d] 傳給裝置顯示

vr::VRCompositor()->Submit(vr::EVREye(eye), &tex);

[e] 使裝置立刻動作

vr::VRCompositor()->PostPresentHandoff();

終止 Terminate (pseudo code)

[a] 關閉裝置

vr::VR_Shutdown();

第三章 實作基於物理材質

第一節 五種貼圖 5 Types of textures

僅只針對絕緣體與金屬物質，使用五種貼圖提供資訊，除了反照率貼圖提供色彩以外，其餘都是提供數據供運算使用。

3-1-1 反照率貼圖 Albedo texture

很像是漫射光貼圖(diffuse texture)，但是稍微亮一點點，因為漫射光貼圖可能會包含有陰影資訊，而計算色彩的時候，我們不需要這些陰影。

3-1-2 法向量貼圖 Normal texture

決定表面凹凸的貼圖，會影響每個像素的法向量資訊。貼圖的 RGB 三色分別表示法向量於貼圖座標 U 方向、V 方向、法向量本身的偏移量，必須透過 TBN 矩陣將 Normal texture 提供的偏移從 tangent space 轉回 world space。

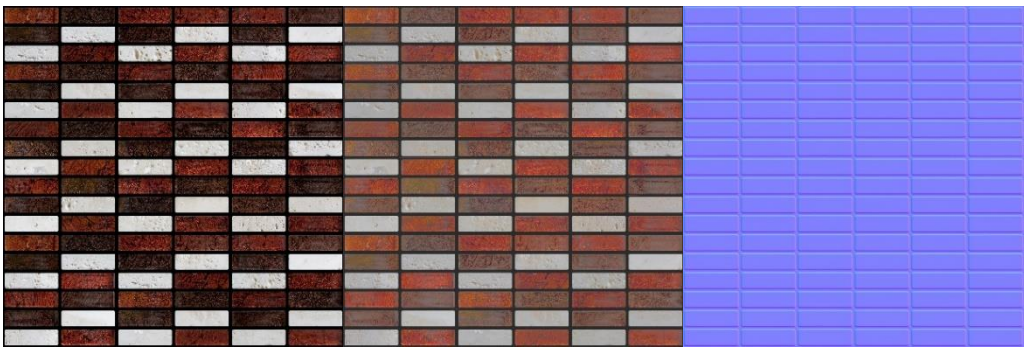

圖 16：漫射光貼圖、反照率貼圖、法向量貼圖

3-1-3 金屬度貼圖 Matellic texture

為了要讓著色器(shader)精簡化，會多加入一張金屬度貼圖，直接指出一塊區域的金屬程度或是反光程度。一般的絕緣體假設有 4%的垂直反射比例 F_0：

$$F_0 = (0.04, 0.04, 0.04)$$

加入金屬度貼圖取得較為正確的F_0，此行會依據金屬程度的數值取F_0(絕緣體)或是本身的顏色(金屬)，如果是絕緣體，F_0會十分接近黑色，如果是金屬，則會顯示本來的顏色

$$F_0 = mix(F_0, albedo, matellic)$$

同樣的方式處理k_d，由於金屬材質會全部吸收漫射光，可以利用來去除k_d值：

$$k_d = k_d \times (1 - matellic)$$

3-1-4 粗糙度貼圖 Roughness texture

表示材質表面的粗糙程度，會直接影響到 BRDF 的計算。

3-1-5 環境光遮蔽貼圖 AO texture

有些材質內部會有許多細小的縫隙，像是磚塊之間或是石牆，這些細小的陰影可以使用 AO texture 來直接繪製陰影效果。

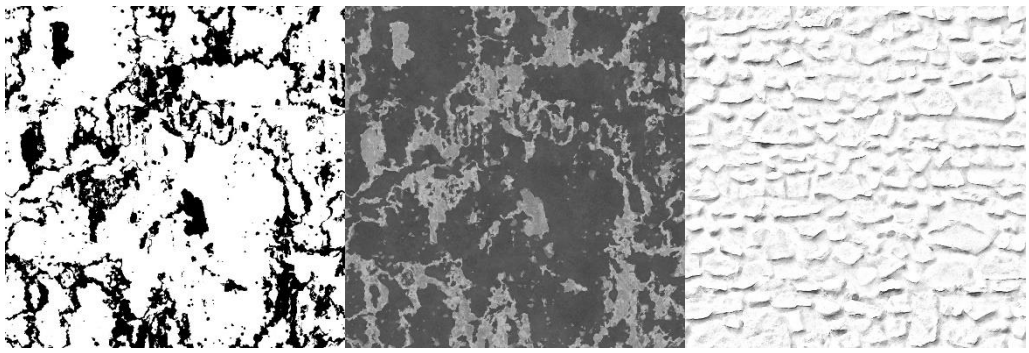

圖 17：金屬度貼圖、粗糙度貼圖、環境光遮蔽貼圖

第二節 能量守恆 Energy Conservation

能量守恆實作十分容易，只要在計算注意 k_d 和 k_s 的處理方式即可。

$$k_s = F(F_0, l, n)$$

$$k_d = 1 - k_s$$

要注意的是，由於金屬材質會吸收所有的漫射光，金屬度貼圖的計算必須添加在能量守恆之後。

第三節 基於物理雙向反射分佈函數 Physically Based BRDF

3-3-1 點光源 point light

$$L_0(x, \omega_o) = \int_\Omega (k_d \frac{c}{\pi} + k_s \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}) L_i(x, \omega_i)(\omega_i \cdot \vec{n}) d\omega_i$$

點光源的做法直接將數據帶入公式即可，會遇到困難的是 $L_i(x, \omega_i)$，由於程式只能使用離散的方式來累加，我們無法得知每一次被切割加入的 $L_i(x, \omega_i)$ 是多少，只知道光線的強弱和距離衰減有某種程度的關係，這裡採用 Inverse-square

Law 計算距離衰減：

$$attenuation = \frac{1.0}{distance^2}$$

然後手動調整點光源的顏色強度，因為這裡的單位不明確，其餘的計算都是比例上的多寡，所以必須手動調整到適當的顏色強度。

3-3-2 環境漫射光 diffuse irradiance

$$L_{Diffuse}(x, \omega_o) = k_d \frac{c}{\pi} \int_{\Omega} L_i(x, \omega_i)(\omega_i \cdot \vec{n}) d\omega_i$$

3-3-2-1 從 360 環景圖轉換成環境貼圖

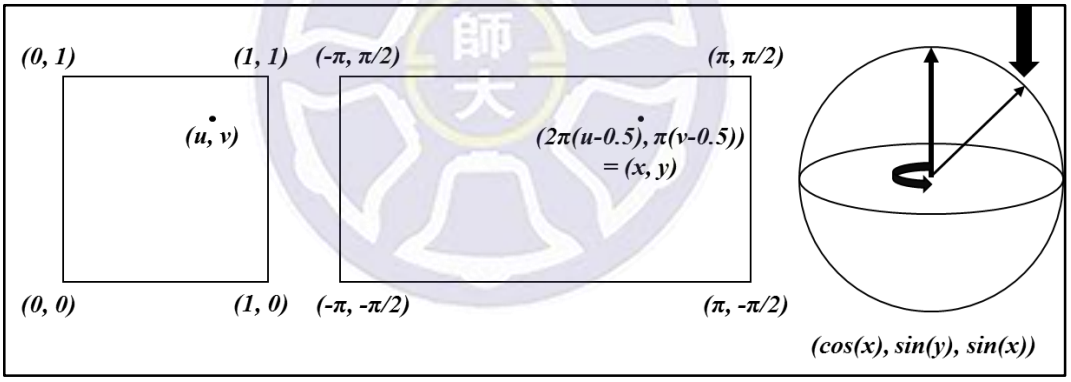

圖 18：由左往右，二維的貼圖轉成環境貼圖

我們有 360 環景圖(Equirectangular Panorama)，想要貼成環境貼圖(Cubemap)，必須知道要去貼圖上取哪個點(u, v)，於此可以使用逆推導的方式思考轉換流程。處理貼圖座標一般會使用 U 和 V 兩軸來表示，而不管多大張的圖片都會使用比例的形式壓成 0~1 之間的貼圖座標。將這張貼圖 U 方向拉長 2π 倍、V 方向拉長 π 倍後(變成原始的比例放大 π 倍)，剛好可以貼在一個單位球體上。原本的點(u, v)在經過轉換後也變成：

$$P:\ (cos(2\pi(u-0.5)), sin(\pi(v-0.5)), sin(2\pi(u-0.5)))$$

有了這個公式後，只要逆推回去即可以解出(u, v)：

$$u = \left[tan^{-1}\frac{P_z}{P_x}\right] \times \frac{1}{2\pi} + 0.5$$

$$v = \left[sin^{-1}P_y\right] \times \frac{1}{\pi} + 0.5$$

3-3-2-2 卷積環境貼圖 convolution cubemap

$$L_{Diffuse}(x, \phi_o, \theta_o) = k_d \frac{c}{\pi} \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\frac{\pi}{2}} L_i(x, \phi_i, \theta_i)\, cos(\theta_i)\, d\theta_i d\phi$$

由於程式使用離散的算法角度間隔是固定的，因此在高緯度的半球體會發生區塊沒有變小的狀況，此時加入一項$sin(\theta_i)$可以彌補這個問題，公式如下：

$$L_{Diffuse}(x, \phi_o, \theta_o) = k_d \frac{c}{\pi} \times \frac{2\pi}{n} \times \frac{\pi}{2m} \sum_{\phi=0}^{n} \sum_{\theta=0}^{m} L_i(x, \phi_i, \theta_i)\, cos(\theta_i)\, sin(\theta_i)$$

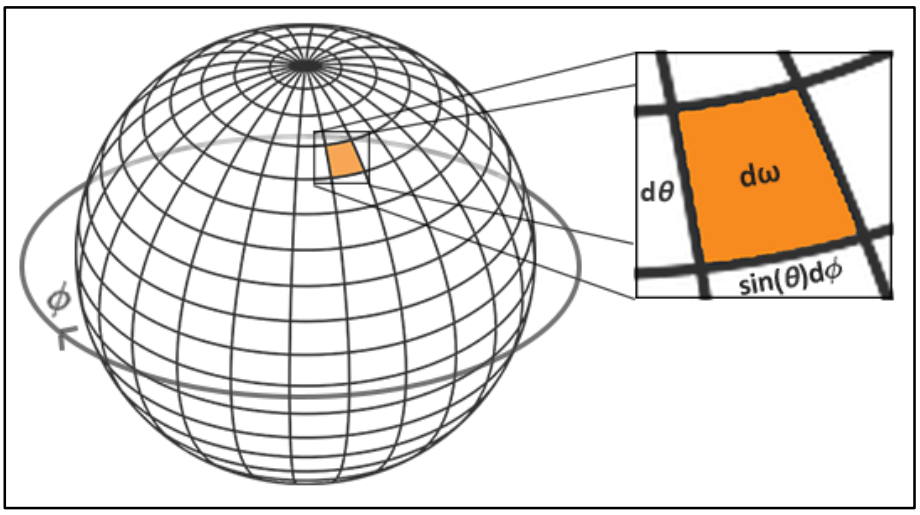

圖 19：越高緯度的地區，離散積分的面積越小

3-3-3 環境反射光 specular IBL

$$L_{Specular}(x, \omega_o) = \int_{\Omega} L_i(x, \omega_i)\, d\omega_i$$

3-3-3-1 多種粗糙程度的反射光環境貼圖

有別於上面的漫射光輻射照度貼圖對著整個半球取樣，這裡只需要對著反射角周圍取樣即可，取樣時必須考慮到整體粗糙程度(第幾層 mipmap)，越光滑越集中，越粗糙越發散，之後將產生的紋理貼圖存起來。

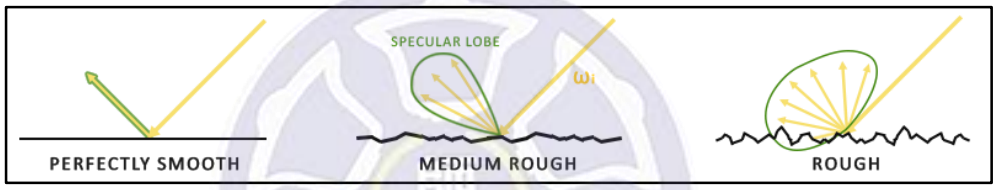

圖 20：反射光路徑和不同粗糙程度表面比較

3-3-3-2 擬蒙地卡羅方法：使用低差異序列

一般會使用蒙地卡羅積分產生隨機序列來取樣，我們知道越接近反射角會有越多反射光，但是要等亂數取樣後取得正確的機率分布實在太花費時間，即時繪圖必須捨棄隨機取樣，而採用有偏差的擬蒙地卡羅方法，運用低差異序列(low-discrepancy sequence)來取樣。我們使用漢默斯里序列，H.Dammertz 仔細描述了此種序列生成方式[11]，它是一種反向的二進位數列可用來快速產生低差異序列。

序列數	0	1	2	3	4	5	6	7	8
二進位	0.1	0.01	0.11	0.001	0.101	0.011	0.111	0.0001	0.1001
十進位	0.5	0.25	0.75	0.125	0.625	0.375	0.875	0.0625	0.5625

表 1：漢默斯里低差異序列

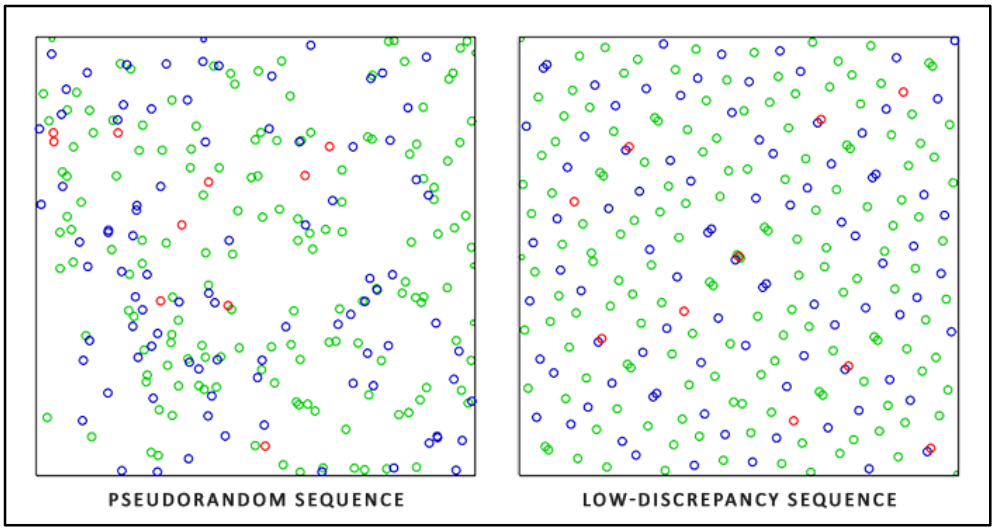

圖 21：一般的隨機取樣序列和低差異序列

3-3-3-3 GGX 重要取樣 GGX importance sampling

有了取樣序列，還必須要結合粗糙程度才能得知取樣向量，Epic Game 提供了一種結合 NDF-GGX 的取樣產生方式，將紋理貼圖(mipmap)的粗糙度和低差異序列算出取樣向量和 h 向量，如此一來就可以直接從環境貼圖算出反射光的強度。

3-3-3-4 預處理 BRDF pre-computing the BRDF

$$\alpha = (1 - \omega_o \cdot h)^5$$

$$L_{Specular}(x, \omega_o) = F_0 \int_\Omega \frac{f_r(x,\omega_i,\omega_o)}{F(\omega_o,h)}[1-\alpha](\vec{n}\cdot\omega_i)d\omega_i$$

$$+ \int_\Omega \frac{f_r(x,\omega_i,\omega_o)}{F(\omega_o,h)}\alpha(\vec{n}\cdot\omega_i)d\omega_i$$

將數字套入公式即可，記得 G 函式要改變 k 值，因為是取環境光。

$$k_{IBL} = \frac{Roughness^2}{2}$$

第四章 實驗結果

第一節 實驗環境

本章節所有圖片都是使用我個人電腦繪製，可用作即時繪圖運算能力的參考，虛擬實境環境測試是使用實驗室的電腦和 HTC VIVE 裝置。

作業系統	Windows 10 Education
處理器	Intel® Core™ i5-4460 CPU @ 3.40GHz
記憶體	16GB DDR3 1333MHz RAM
顯示卡	NVIDIA GeForce GTX 760

表 2：個人電腦設備

作業系統	Windows 10 Education
處理器	Intel® Core™ i7-6700 CPU @ 4.00GHz
記憶體	16GB DDR4 2133MHz RAM
顯示卡	NVIDIA GeForce GTX 1070

表 3：實驗室電腦設備

第二節 測試場景

4-2-1 效能數據

Model	Num. Vertices	Num. Faces	Performance	Performance (VR)
sphere	10241	20479	1000 fps	100 fps
teapot	8478	15704	1000 fps	100 fps
head	17684	9923	1000 fps	100 fps
tearoom	2802	5397	1000 fps	100 fps
Dabrovic Sponza	59810	66450	~333 fps	100 fps
Sponza	184330	262267	~40 fps (Hi-Res. tex.)	~15 fps
Sponza	184330	262267	~90 fps (default tex.)	~35 fps

表 4：測試場景相關效能數據

4-2-2 繪製結果

4-2-2-1 球 sphere

頂點數：10241 // 三角片數：20479 // 效能：1000 fps // 虛擬實境效能：100 fps

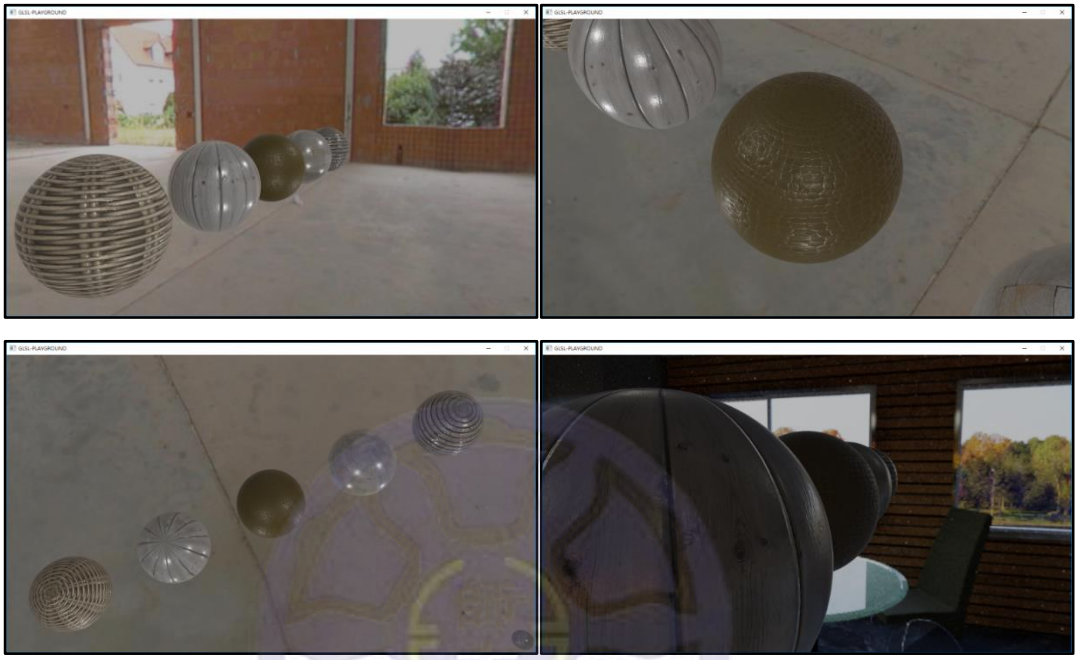

圖 22：繪製球的模型

4-2-2-2 茶壺 teapot

頂點數：8478 // 三角片數：15704 // 效能：1000 fps // 虛擬實境效能：100 fps

圖 23：繪製茶壺模型

4-2-2-3 頭像 head

頂點數：17684 // 三角片數：9923 // 效能：1000 fps // 虛擬實境效能：100 fps

圖 24：繪製頭像模型

4-2-2-4 茶房 tearoom

頂點數：2802 // 三角片數：5397 // 效能：1000 fps // 虛擬實境效能：100 fps

圖 25：繪製茶房場景，由於無法達成物件全域影響，所以桌面其實是直接反應環境貼圖

4-2-2-5 Dabrovic Sponza

頂點數：59810 // 三角片數：66450 // 效能：~333 fps // 虛擬實境效能：100 fps

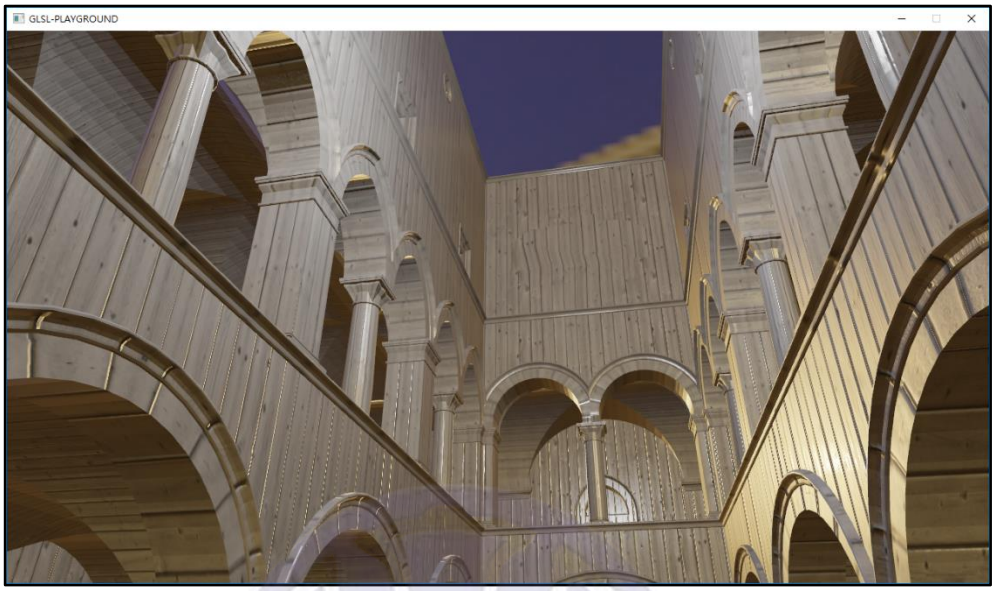

圖 26：閣割版的 sponza 模型，使用的貼圖為 6k 高解析度的貼圖

4-2-2-6 Sponza

頂點數：184330 // 三角片數：262267 // 效能：~40 fps // 虛擬實境效能：~15 fps

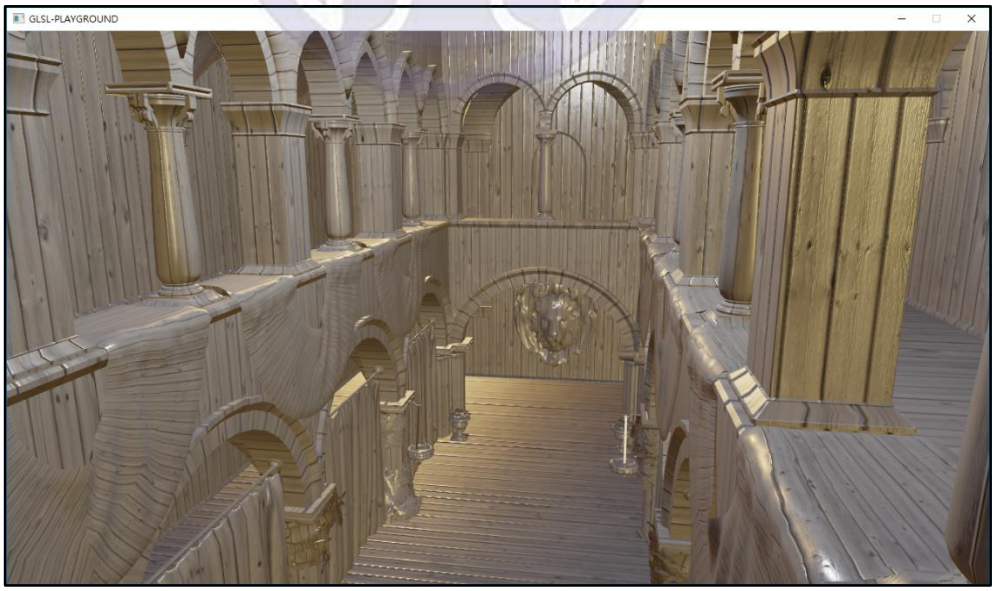

圖 27：高解析度貼圖會嚴重拖累效能

第三節 比較

4-3-1 基於物理繪圖和 Phong shading

圖 28：基於物理繪圖、Phong shading

這裡將基於物理繪圖和傳統的 Phong shading 上色方式做比較，會發現基於物理繪圖的方式能夠反應環境光源。法向量貼圖可以模擬材質的凹凸，看起來表面不會是平的；環境光遮蔽貼圖則能夠讓溝紋變得更深，呈現出立體感；粗糙度貼圖和預處理的漫反射環境貼圖把環境光源模糊化而不會像是鏡面。

4-3-2 基於物理繪圖和 Phong shading 效能比較

	Model	Num. Vertices	Num. Faces	Performance
PBR	Sponza	184330	262267	~40 fps
Phong shading	Sponza	184330	262267	~180 fps

表 5：基於物理繪圖和 Phong shading 效能比較

4-3-3 基於物理繪圖和 Cycles 繪圖引擎(光線追蹤)

圖 29：基於物理繪圖、Cycles 繪圖引擎

此處比較出即時演算和光線追蹤的差異，前者只能模擬出類似的效果，而後者卻是計算光線實際穿梭於場景中，正確的陰影、正確的光照、正確的遮蔽效果都大幅超越即時運算，然而這張圖片花了大約 40 分鐘繪製，而即時繪圖卻能一秒運算超過 1000 張影像(以此模型為例)。

第五章 結論與未來展望

此實驗展示了即時繪圖也能做出接近真實的材質效果，預處理環境光雖然需要大量的數學來幫助簡化，但是得到的效果相當不錯，如果就單一凸面物件放在場景的正中間，就是此方法最完美的展示。

不過實際創作一個場景絕對還有更多的因素要考量，受限於編成介面，無法實際製作一個房間製造出室內的感覺，如果要取得更正確的光照資訊，只能先繪製好室內的全景圖當作環境貼圖來使用，此外，模型的精細度與貼圖座標也是個大問題，使用傳統圖學的模型會受限於無法自訂貼圖座標而沒有太多的選擇。

未來希望有機會能夠實作物件的全域影響，並且加入陰影和攝影機移動，如此一來就更接近虛擬實境該有的體驗了。

參考文獻

相關研究論文

[1] B. Duvenhage, K. Bouatouch, and D. G. Kourie, "Numerical verification of bidirectional reflectance distribution functions for physical plausibility," presented at the Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference, East London, South Africa, 2013.

[2] Brian, Karis, “Real Shading in Unreal Engine 4”, part of “Physically Based Shading in Theory and Practice”, SIGGRAPH 2013 Course Notes.

[3] Burley, Brent, “Physically-Based Shading at Disney”, part of “Practical Physically Based Shading in Film and Game Production”, SIGGRAPH 2012 Course Notes.

[4] Cook, Robert L., and Kenneth E. Torrance, “A Reflectance Model for Computer Graphics”, Computer Graphics (SIGGRAPH ’81 Proceedings), pp. 307–316, July 1981.

[5] Cook, Robert L., and Kenneth E. Torrance, “A Reflectance Model for Computer Graphics”, ACM Transactions on Graphics, vol. 1, no. 1, pp. 7–24, January 1982.

[6] Hoffman, Naty, “Background: Physics and Math of Shading”, part of “Physically Based Shading in Theory and Practice”, SIGGRAPH 2013 Course Notes.

[7] J. T. Kajiya, "The rendering equation," SIGGRAPH Comput. Graph., vol. 20, pp. 143-150, 1986.

[8] Lambert, Johann H., “Photometria Sive de Mensure de Gratibus Luminis, Colorum Umbrae,” Eberhard Klett, 1760.

[9] Schlick, Christophe, “An Inexpensive BRDF Model for Physically-based Rendering”, Computer Graphics Forum, vol. 13, no. 3, Sept. 1994, pp. 149–162.

[10] Trowbridge, T. S., and K. P. Reitz, “Average Irregularity Representation of a Roughened Surface for Ray Reflection,” Journal of the Optical Society of America, vol. 65, no. 5, pp. 531–536, May 1975.

[11] Walter, Bruce, Stephen R. Marschner, Hongsong Li, Kenneth E. Torrance, “Microfacet Models for Refraction through Rough Surfaces,” Eurographics Symposium on Rendering (2007), 195–206, June 2007.

[11] Holger Dammertz - Hammersley Points on the Hemisphere
http://holger.dammertz.org/stuff/notes_HammersleyOnHemisphere.html

[12] Joey de Vries - LearningOpenGL
https://learnopengl.com/#!PBR/Theory

[13] Marco Alamia - Coding Labs
http://www.codinglabs.net/article_physically_based_rendering.aspx