國立政治大學資訊科學研究所

Department of Computer Science

National Chengchi University

碩士論文

Master's Thesis

以影像空間加速機制之多層折射繪圖

Image Space Real-time Multi-refraction Technology

研 究 生：王士豪

指導教授：紀明德

中華民國一百零三年四月

April 2014

以影像空間加速機制之多層折射繪圖

Image Space Real-time Multi-refraction Technology

研究生：王士豪　　Student：Shi-Hao Wang

指導教授：紀明德　　Advisor：Ming-Te Chi

國立政治大學

資訊科學系

碩士論文

A Thesis

Submitted to Department of Computer Science

National Chengchi University

in partial fulfillment of the Requirements

for the degree of

Master

in

Computer Science

中華民國一百零三年四月

April 2014

致謝

進入政大將近三年的時間，日子轉眼即逝，在這期間我非常由衷的感謝我的指導老師紀明德老師，從碩一剛入學時對於許多課業與研究方面皆遇到許多困難，而老師一直很耐心地指引我方向，隨著時間慢慢地過去，也逐漸進入狀況。這期間我非常謝謝我的家人，給予我精神上的支持，謝謝你們的關心與體諒。研究生的生活平常雖然乏味，日子卻相當充實，有實驗室的學長、學弟妹以及夥伴們的陪伴之下，生活當中也有許多不同的一面，謝謝漢光與承峰學長，有你們在實驗室才顯得有趣、謝謝凱彬與我度過神奇的三年以及對我的關心與照顧、謝謝維晉在程式上幫我許多忙、謝謝三個學弟以及汪雨在我碩三無聊的日子陪我打打電動玩玩桌游、謝謝阿樹成為博士生，讓我們實驗室的成果能夠傳承、謝謝雅欣的啟發我想成為遊戲設計師的夢想，謝謝琇媛陪我度過我的碩士生活，有妳就不會有無聊，謝謝我大學社團的朋友 Terry、風車、阿澤，在這個人生地不熟的台北，你們是我練舞的好夥伴與解悶的對象，要謝謝的人太多，總之謝謝所有幫助我的人。

以影像空間加速機制之多層折射繪圖

摘要

光線的折射和全內反射是透明物體中重要的光學現象，可用於玻璃藝術。本論文利用 depth-peeling 的技術將一個三維幾何網格物體拆解成四層深度與法向量貼圖，提出改良的 relief mapping 使用的光線相交演算法解決層與層之間的斷層問題，在影像空間中計算至多四次折射和全內反射，最後對計算出的折射向量配合環境貼圖取得對應的折射效果。以上的單一物體四次折射架構，可延伸至多物體的場景，進行多物體四次折射計算。本論文改良傳統 image-space 兩次折射的方法，可以處理更複雜物體與場景，並嘗試與 GPU 加速的光跡追蹤軟體(OptiX)與不同場景進行比較，驗證其演算法效率與品質。

Image Space Real-time Multi-refraction Technology

Abstract

Light refraction simulation is an important optical phenomenon for the realism of computer synthesized images. This thesis proposed a image-space for real-time multi-refraction. First, we apply the depth-peeling technique to disassemble a 3D polygonal object into four layers of normal texture and depth texture. Then a modified ray-height-field intersection algorithm is proposed to solve the fault zone between the layers and to compute the intersection, refraction, and the total internal reflection on image space. We can generate (at most) four pass refraction. At the last step, we use the environment map with the refraction vector to get the final color. The proposed algorithm can easily extend to multi-object refraction. Experiment results on various scenes demonstrate the feasibility and quality of the proposed multi-refraction method. A comparison to the GPU-based ray tracing (OptiX) is shown to support the efficient of our method.

目錄

圖目錄

第一章

緒論

1.1 研究動機與目的

在 3D 圖學中，光線的折射和全內反射，可以增加模擬物體的真實感。Ray-tracing 可以達到透明物體的效果，但是由於計算量非常龐大，要能夠即時顯示有一定的困難度，所以我們不使用這種方法，使用的是 Image-space 的方法，可以充分利用 GPU 的平行運算的特性，以達到即時顯示的效果，這種效果在遊戲中很重要，可以讓透明物體的光影表現在遊戲世界中更接近現實感。

這裡以 3D 模型作為輸入，來產生透明的 3D 物體，如此可以利用到現存豐富多樣的模型。將 3D 場景以環境貼圖的方式表現，由折射後的向量來繪製出物體的像素,最簡單的方式就是 Wyman 提出來的一系列方法[Wyman 2005, 2007]，但當 3D 模型太複雜時，造成多次折射的現象因為只有兩層表面而估計錯誤，所以我們以 Depth peeling 作法去改進。

全內反射（又稱全反射）是一種光學現象。當光線經過兩個不同折射率的介質時，部份的光線會於介質的界面被折射，其餘的則被反射。但是，當入射角比臨界角大時（光線遠離法線），光線會停止進入另一介面，反之會全部向內面反射，在 3D 圖學中，我們實作物體內的反射，以光線的角度計算出產生全反射的

位置，以達到與 Ray tracing 所繪製的 3D 物體相近的效果。

1.2 問題描述

因為效率上的考量，光跡追蹤法在即時的實現由於遞迴的計算量過於龐大而不容易實現，因此本篇論文參考了許多運用 Image-space 的折射效果，發現 3D 模型越複雜與 Ray tracing 所 render 出來的結果就越不相似，主要原因在於前面的方法只考慮了兩層的深度，也因此無法將整個物體形狀紀錄下來，而沒紀錄到的部分往往就無法正確的計算折射向量，此外 3D 物體的深度通常不是連續的，斷層經常出現在深度貼圖中，因此有些折射向量在計算時會被這些斷層誤導導致沒有辦法確實的計算，因此我們實作出四層深度的方法，在不失去大部分物體折射向量的情況下 render 出近似 ray tracing 的 3D 透明物體。此方法可分成四部分，1. depth peeling 的分層 2. 深度圖斷層的處理 3.層與層之間的折射 4. 全內反射的處理。

有了一個 3D 物體的折射，多個 3D 物體在之前的論文中並沒有被提及，為了更接近真實世界，我們發現可以利用多個物體分層並且合併來計算出折射貼圖，因此本論文提出以 render to texture 存取單個物體折射向量的方法，對排序過的物體依序處理折射的方法，並且利用貼圖傳送物體的資訊給下一個物題，可以得到多個折射物體的場景。

1.3 論文貢獻

在本研究論文當中，我們實作了多層折射的 3D 物體，其所產生的透明物體近似 ray tracing 的 render 結果並且即時的運行。本研究的方向主要可以分為以下兩大部分，各部分的內容敘述如下：

- 在影像空間中計算至多四次折射和全內反射，最後對計算出的折射向量配合環境貼圖取得對應的折射效果。

- 以單一物體四次折射架構，延伸至多物體的場景，進行多物體四次折射計算。

1.4 論文章節架構

在第二章中，我們將會介紹與本研究相關的研究背景知識，重真實世界光學的物理現象到光跡追蹤的演算法，以及對光學的分析，此外也介紹我們主要參考的技術:depth peeling、texture mapping、relief mapping 算法。在第三章當中則是主要的研究方法與步驟，針對單一物體分層後的結果，並考慮深度斷層的結果，此外我們也提出多個物體的折射算法。在第四章則是實驗結果的呈現與分析比較討論。第五章是結論與未來研究。

第二章

相關研究

在模擬折射之前，我們必須分析與觀察真實世界中的折射現象，因此我們在 2.1 節整理了許多的物理定律，並運用到相關的實作技術，我們的方法以光跡追蹤法的結果做為對照，因此也在此節說明光跡追蹤法的演算法。2.2 節針對折射物體的分析在許多論文中被提出，我們也以此為依據論述我們方法的折射層數與次數。由於我們實做折射的方法有別於光跡追蹤的方法而使用 image space 的技術，在 2.3 節討論 image space 的相關研究。另外我們的方法運用了 depth peeling 技術來切割我們的模型，在 2.4 節我們討論 depth peeling 技術以利於我們計算折射向量。最後 2.5 節討論折射實做的相關論文。

2.1 光學:折射與全反射

要模擬折射物體前，我們需要先了解真實世界中折射的原理，並以這些原理為基礎建構的數學模型。

折射為一種常見的物理現象，指的是當物體或波動由一種介質斜射入另一種介質中因為速度不同，因此引起角度上的改變。在這裡我們所說的折射特別是指光的折射，指的是光從一種介質進入另一種具有不同折射率之介質所發生的情形，或者在同一種介質中折射率不同的情況下(例如水溫差異導致折射率不同)，造成

光的波速變化，使光的移動方向改變。生活中常見的例子如，在河邊看的到魚卻抓不到他，或者吸管插入水杯中造成吸管折斷的情況。

光在發生折射時入射角與折射角符合司乃耳定律(Snell’ s law)。此定律指出光從真空進入某種介質發生折射時，入射角i的正弦跟折射角 r 的正弦之比數，等於這種介質的折射率n。這項定律以如下公式顯示：

$$\frac{\sin i}{\sin r} = n$$

或光從介質 1 進入另一不同折射率之介質 2 時，其入射角的正弦（θ）與其折射率（n）之乘積會相等：

$$n_1 \sin\theta_1 = n_2 \sin\theta_2$$

這二條公式被稱為司乃耳定律。

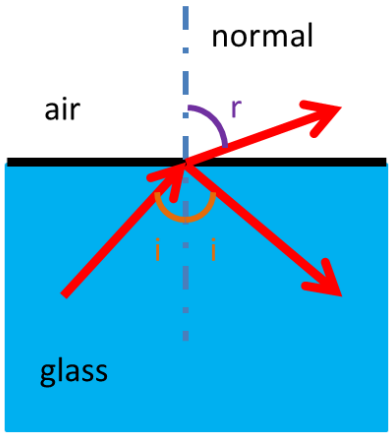


圖 2.1: 折射與反射

全內反射（Total internal reflection）是一種光學現象。當光線經過兩個不同折射率的介質時，部份的光線會於介質的界面被折射，其餘的則被反射。但是，當入射角比臨界角大時，光線會停止進入另一介面，並且全部向內面反射。

這只會發生在當光線從光密介質（較高折射率的介質）進入到光疏介質（較低折射率的介質）並且入射角大於臨界角時。因為沒有折射（折射光線消失）而都是反射，故稱之為全內反射。例如當光線從玻璃進入空氣時會發生，但當光線

從空氣進入玻璃則不會。在生活中出現的例子如沸騰水中的氣泡看起來十分明亮，就是因爲發生了全內反射的關係，另外鑽石會閃閃發光，也是因為多次全內反射的結果。

臨界角（Critical angle）即是發生全內反射時最小的入射角。入射角是從折射界面的法線量度計算的。臨界角（θ_c）可從從以下公式得到：

$$\theta_c = \arcsin\frac{n_2}{n_1}$$

其中n_2是較低密度介質的折射率，而n_1是較高密度介質的折射率。公式是以司乃耳定律所推導出來的結果。 當入射光線是準確的等於臨界角，折射光線會循折射界面的切線進行。以玻璃為例，臨界角約為 41.5°。

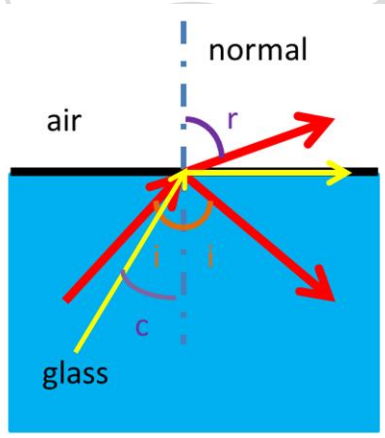


圖 2.2: 全內反射:c 為臨界角，i 大於 c 因此造成全內反射

Ray tracing(光線追蹤)，是幾何光學中一項基本的技術，透過跟蹤與光學表面發生互動作用的光線從而得到光線經過的路徑。也常用於生活中，如照相機，望遠鏡，顯微鏡等等的光學儀器，而在電腦圖學中則是特殊的繪圖演算法，最早是 1968 年 Arthur Appel 所提出的 [Arthur, 1968]，主要的概念是跟蹤從眼睛發出的光線而不是光源發出的光線，在用來繪製場景和模型。這種方法能得到較好的光學效果，例如對於反射與折射有更準確的模擬效果，所以本論文以此方法作為結果的對照組，圖 2.1 為簡單的示意圖。

Ray tracing 演算法:

對於每一個 pixel 都產生一條 view ray

如果與最相近的物體相交則設置此物體為最近物體並記錄最近距離。

對每個 view ray 射出一條 shadow ray 來檢測是否處在陰影中。

如果表面是反射面，生成反射光；對此反射光繼續遞迴。

如果表面透明，生成折射光；對此折射光繼續遞迴。

使用最近物體和最近距離來計算指像素的色彩。

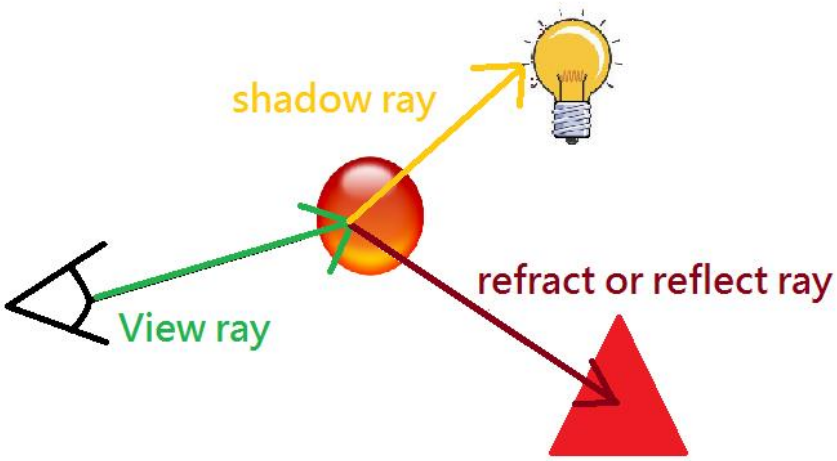


圖 2.3: ray tracing

而在電腦圖學中，運用 ray tracing 所繪製的 3D 物體如圖 3.3a:單純只有折射的 render，黑色的部分是因為全反射的產生而造成無法顯示折射，b 則為加入全反射後的 render，c 為場景設計。

(a) (b) (c)

圖 2.4: ray tracing 折射與全內反射比較(a)只有折射的結果(b)加入全反射後的結果 (c)場景設計。

而近年 NVIDIA 發表了“互動光線追踪引擎”OptiX (OptiX Application Acceleration Engine)。OptiX 是以 CUDA(Compute Unified Device Architecture)的架構，可以使用 C 語言編寫程式，繪製出以 GPU 運算的光線追踪的場景。OptiX 的應用既相當廣泛，包括照片式成像、車輛設計等圖形領域，還有諸如光學和聲學設計、輻射研究、容積計算、碰撞分析等等，由於 CUDA 以平行架構處理光線追踪，因此有別於傳統的光線追踪，可以較快速的生成場景，因此本論文的結果與 OptiX 所產生的折射場景作比較，我們發現我們的方法較為快速，並且接近其結果。

2.2 折射的分析

在了解物理的現象之後，考慮模型的複雜度與折射的次數，折射越多次之後的結果相對越不明顯，Yasuhiro 在 2010 年 CVPR 發表 [Yasuhiro, 2010]，其中分析光的強度如圖 2.6，我們可以發現，光線經過四次的作用之後其強度已經不明顯，其原因在於光線經過物體的折射反射與散射之後其能量會漸漸被物體給吸收，此外 Aner 在 2008 年 ACM TOG 發表[Aner, 2008]中也提到四次光學作用是他們認為最佳的光學效果。

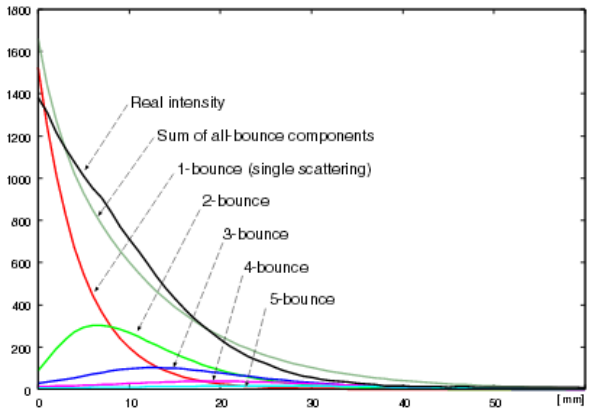


圖 2.5: Light Path(線條)，縱軸(光強度)，橫軸(光焦距)，摘自 [Yasuhiro, 2010]

Manmohan 在 2011 年 IEEE TPAMI 發表的[Manmohan, 2011]更精準的分析當前光學的運算方法:"逆向光線傳輸"，其結果如圖 2.6，顯示了 light pass 的次數越大結果越黑。因此在我們的方法中，我們也以分為四層的物體，最多產生四次折射為基礎。

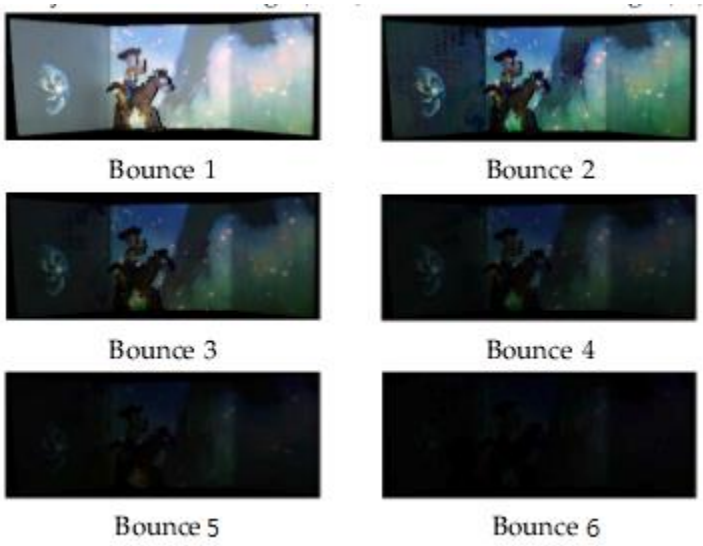


圖 2.6: Light Path 次數與顯示結果，摘自[Manmohan, 2011]

2.3 Texture mapping 技術與 Shader

Fabio 在 2005 年 I3D 發表[Fabio, 2005]，他的方法需要模型的高度場與法向量，以"光線高度場相交"演算法有效率的產生任意模型上的立體表面，由於[Manuel, 2007]的方法在計算折射交點時便是以使方法為基礎，因此我們在此仔細說明此演算法。

圖 2.7: relief mapping，摘自[Fabio, 2005]

圖 2.8 點 A 深度值為 0 ，點 B 關聯的深度值為 1.0 。在每一步裡面，都要計算出當前區域段的中點，然後把平均深度值和平均貼圖座標賦值給它。這個例子裡面，圈 1 表示了第一個中點。貼圖坐標的平均值用來訪問深度圖。如果儲存在深度圖中的深度值小於計算出來的深度值，表示這個點在高度域內（比實際高度低）。

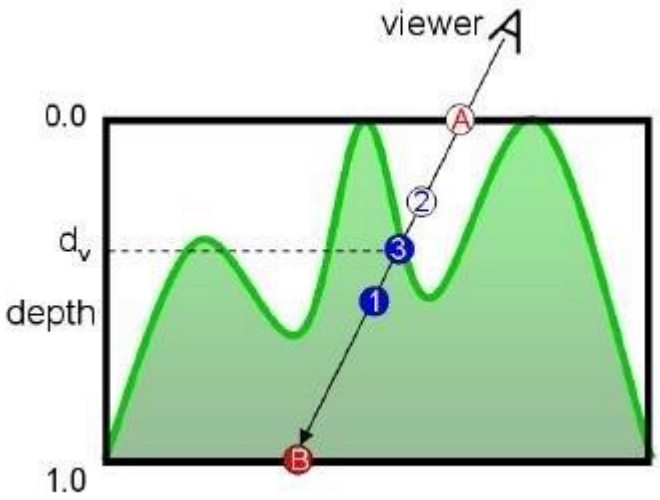


圖 2.8: 使用 A 和 B 之間的二分搜尋來計算 VD' 和高度域表面的相交點，數字表示計算中點的順序。

光線高度場相交演算法:

1. 計算從視點到多邊形表面上的 3D 位置點的方向(VD)，用向量來表示這個方向；
2. 把 VD 轉換到切線空間(tangent space)（定義為切線、法向量和次法向量組成的向量）；
3. 從 VD’（轉換過的 VD）和 A、貼圖座標(s,t) ，直到計算到 B，貼圖座標(u,v)，這個位置已經到了深度值是 1.0 的地方(圖 2.6)；
4. 使用 A 和 B 之間的二分搜尋來計算 VD’ 和高度域表面的相交點；
5. 使用計算出來的相交點的貼圖座標來繪製每一個 frangement 的屬性（比如：法向量、深度、顏色，等等）

然而這個演算法也有問題的存在，如果視線與高度域表面相交不止一個點的時候，二分搜尋的方法就可能會帶來不正確的結果，如下圖。在這個例子裡，第一次找到的中點對應的深度值要比深度圖中實際的深度值小。這個時候用二分搜尋會繼續找到圈 3 這個相交點，這顯然是錯誤的。為了避免錯過第一次相交，我們使用線形搜尋的方式來處理。從點 A 開始，我們沿著線段 AB 每次增加 d 的距離，直到找到在表面內的第一個點（圖 2.9 左）。

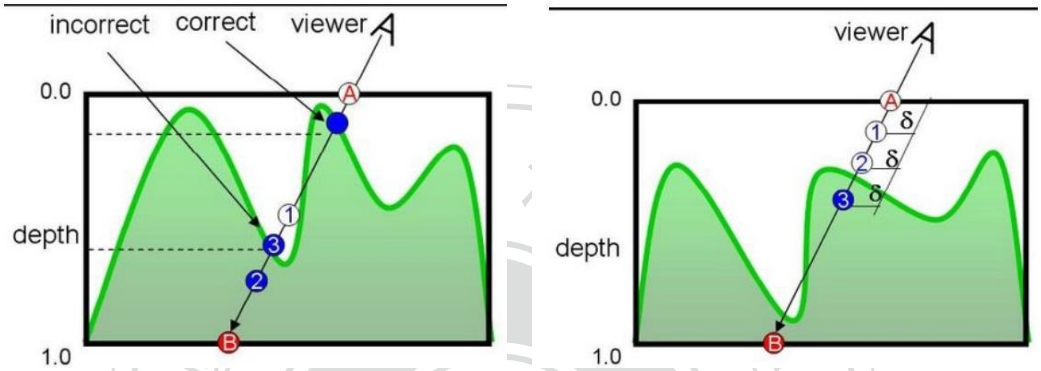


圖 2.9: 左:二分搜尋的錯誤，右:線性搜尋

至於背景我們採用[Fernando, 2003]的方法建構 environment mapping，以下為步驟:1、根據視線方向和法向量計算反射向量；2、使用反射向量或折射向量檢索環境貼圖得到貼圖；3、將貼圖融合到當前像素中。

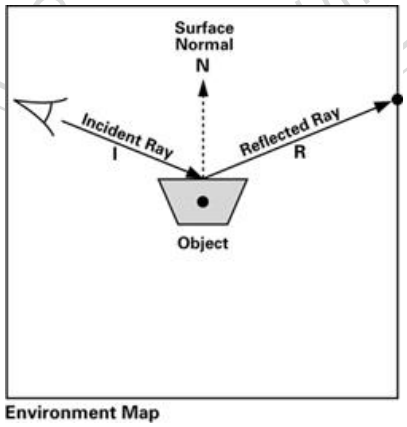


圖 2.10: environment mapping 示意圖，摘自[Fernando, 2003]

2.4 Depth peeling

Depth peeling(深度剝離)是為了處理透明物體的繪製而發明的技術，原始的版本可以在 NVIDIA Developer 網站上找到。depth peeling 是基於 z-buffer 的多層深度繪製，每一層的深度是基於上一層繪製的深度值基礎上進行的。它是由 Mamman 1989 年中提出[Mamman, 1989]。 Everitt 在 2001 年 [Everitt, 2001]給出該方法在 GPU 上的實作。在這之後，該方法得到大量的應用。Baoquan Liu 和 Liyi Wei 在 2006 年提出[Baoquan, 2006]，使用了 GPU 的 MRT(Multi render target)能力加速生成深度剝離圖層，比原始方法提高了一倍左右的效率。而在 2008 年 Nvidia SDK 給出了 dual depth peeling 的實現[Bavoil, 2008]，也就是前後深度 peeling 的技術進而提升效率。

圖 2.11: dual depth peeling，圖中顏色使觀察者明顯看出前後層，摘自[Louis, 2008]

2.5 折射

Wyman 在 2005 提出”An Approximate Image-Space Approach for Interactive Refraction”主要是以 image space 方法產生折射的效果，實作在 GPU 上。此篇的方法比起一般光跡追蹤的方法運算較容易且適合在 GPU 上實作，但是也有缺點: 當第三次折射發生時，就會產生問題，因為法向量與高度場無法正確的顯示類似茶壺這種拓樸結構，因此對於 3 次以上的折射會產生無法精確展示模型的問題。

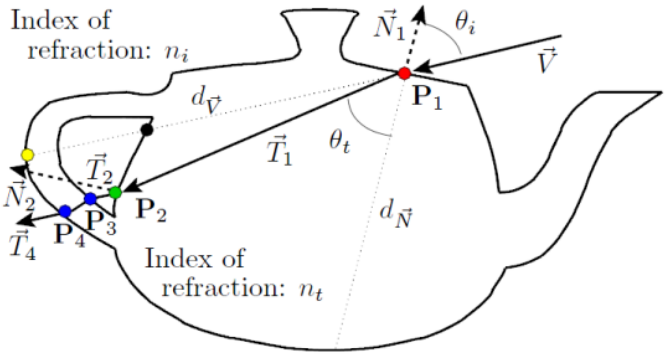


圖 2.12: 茶壺折射模型，摘自[Wyman, 2005]

以下為[Wyman 2005]主要的公式算法

由 P1 點以估計的方法估算到 p2 點，主要公式為:

$$\tilde{p}_2 = p_1 + \tilde{d}\vec{T}_1 \approx p_1 + d\vec{T}_1$$

其中 d 的算法則是以 dn 與 dv 的內插估計:

$$\tilde{d} = \frac{\theta_t}{\theta_i} d_{\vec{v}} + (1 - \frac{\theta_t}{\theta_i}) d_{\vec{N}}$$

得到 p2 點之後再取得 p2 點的視點入射向量、法向量，算出其射出方向得到估計的折射路徑，產生出折射的現象。

圖 2.13: 折射技術的比較 a.摘自[Wyman, 2005] b. 光跡追蹤法

在[Wyman,2005]之後，Wyman 緊接著發表了[Wyman, 2005]，也就是透明物體後的幾何模型如何顯示，因為原本的方法只能讀取到環境貼圖的顏色，並無法知曉後面的模型，因此 Wyman 將模型以貼圖的方式記錄其顏色並以物體折射後的向量計算與貼圖的距離來得到模型的顏色。

圖 2.14: a. 一層折射與後面的模型 b. 二層折射與後面的模型，摘自[Wyman, 2005]c. 光跡追蹤法

相較於[Wyman, 2005]，Manuel 在 2007 提出[Manuel, 2007]是以 image space 所做出的折射效果，最大的貢獻是可變物體的折射，主要的技術是運用了 relief mapping 的搜尋方式去尋找折射的交點，因此他修正[Wyman, 2005]估計上可能產生的錯誤。

圖 2.15: a. 光跡追蹤法 b. 摘自[Manuel, 2007] c. deformable objects

而 Charles 在 2011 提出[Charles, 2011]則是更改傳統的光跡追蹤方法，運用 BTDF 與 cone tracing 製造出即時的折射效果，另外表面也可以依照使用者輸入

的參數調整粗糙程度，變換不同的折射效果。

圖 2.16: 不同粗糙表面的折射，右邊為表面越粗糙，摘自[Charles, 2011]

第三章

研究方法與步驟

折射的計算取決於視點向量、法向量與折射率，因此我們首先需要運用 depth peeling 分層得到計算折射所需要的法向量與深度影像，並用環境貼圖建構 sky box，再實作兩層深度影像間的折射運算，並且以此方法為基礎，實作出多層深度影像間的折射效果。並處理全內反射的效果。

我們先在 3.1 節分析模型與分層，對折射模型有一定的理解後，我們以 3.2 節的程式架構實作，3.3 節說明 GPU 加速與光線相交的算法，3.4 節則將模型拆解四層深度貼圖以利於正確的多次折射，而拆解後遇到的斷層問題在 3.5 節提出了方法解決，單一模型之後，3.6 節則為多個模型的運算。

3.1折射模型的分析

在折射計算之前，我們必須先提出兩個問題，1. 折射次數與模型的關聯? 2. 多少次的折射會近似真實的效果?這兩個問題分別在 3.11 與 3.12 說明。

3.1.1 模型的分析

在實作之前，我們觀察發現我們可以將場景分成多個或單一模型，而單一模型之下又可細分為只需要兩層折射即可完成運算的簡單模型和至少需要四層以上的複雜模型，我們發現簡單模型都為凸多面體，而複雜模型必為凹多面體，以

下證明凸多面體只會產生二次折射:

凸多面體不會產生內角大於 180 度的形狀,如圖 3.1 為凹多面體的某一條 eye ray 的平行切面，因此我們利用反證法證明凹多面體可能會產生多次折射，得證凸多面體必定只會產生二次折射。

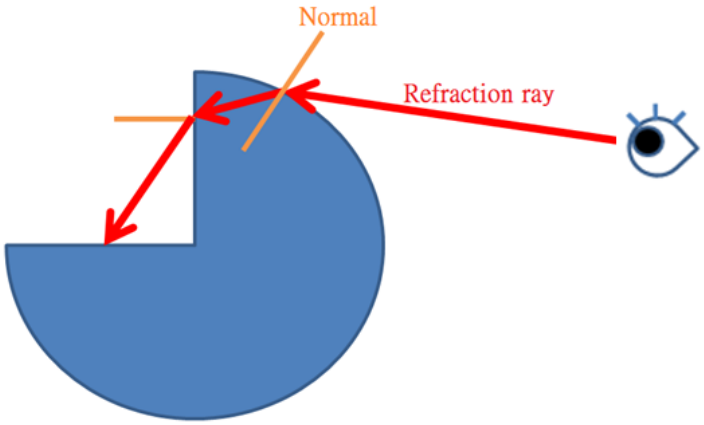


圖 3.1: 任意凹多面體的某一切面

圖 3.2 為詳細的折射角度示意圖，K、L 為邊的延伸向量 N 為第二次折射向量交點的法向量，角度 A 與 B 分別為第二次折射向量的內角與 K、L 的外角，當 B 角度大於 A 角度時,基於角度的定理 R 與 L 只會相交一點且 R 與 K 必定相交，因此第二次折射向量必定會與 K 向量的平面相交而再次進入模型中。

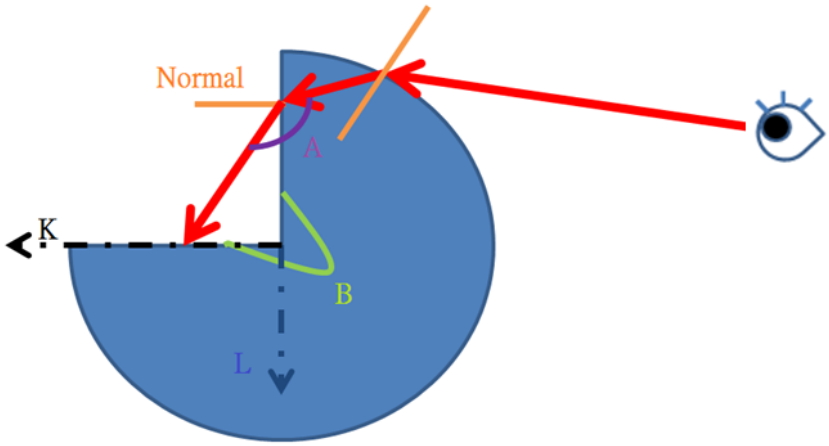


圖 3.2: 任意凹多邊形的折射

經由以上推論之後，凸多面體的模型均可以運用[Manuel, 2007]的方法來得到正確的折射向量，然而凹多面體卻沒有辦法，而我們觀察發現凹多面體內角大

於 180 度的地方往往也是 Depth pelling 所分割 layer 之處，因此我們採用了 Depth pelling 的技術來解決複雜模型的折射。

3.1.2 層數的分析

接著所產生的問題是:要切幾層才能表現出結果?，切太少層會失去許多的細節，切太多層又會增加不必要的計算，在[Manmohan, 2011]、[Yasuhiro, 2010]、[Aner ,2008]三篇論文中都對光線的行進路徑進行了分析，我們發現經過四次 pass 之後的光線強度大都不太明顯，在本論文中我們特別針對光的折射路徑進行分析，發現一般模型會產生兩次以上折射多半如甜甜圈模型的拓撲結構，又或者如動物耳朵的形狀容易在側面的角度產生四次的折射，以玻璃(折射率=1.45)的甜甜圈模型為例如圖 3.3，可以很明顯看出後面的柱狀體，然而也有如中國龍模型這種模型需要到六層或八層的折射但也因為太過複雜，隨這折射次數的增加結果也漸漸不明顯，因此如圖 3.4 所示在 OptiX render 的結果四層折射與六層折射結果非常接近，也因此本論文最後採用四層折射的結果可以顯示大致上相似，卻也不至於損失太多的細節。

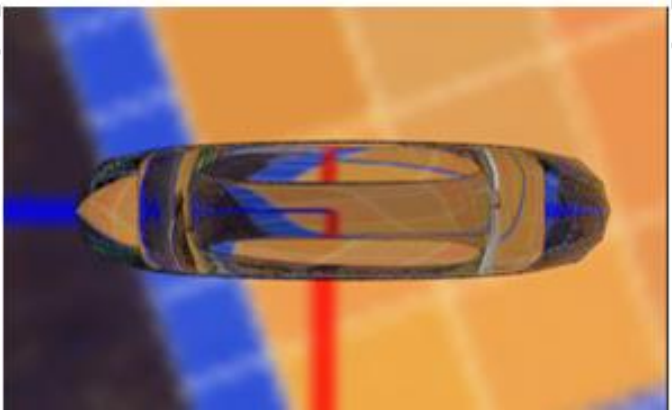


圖 3.3: 甜甜圈模型 Blender render 結果

圖 3.4: OptiX 中國龍模型四層(左)與六層(右)結果比較，圓圈為差異處

3.2 程式架構

本節為我們方法的程式架構，首先對每一個輸入的模型進行 Depth peeling 的分層與合併，得到四層的法向量與深度，以此資料進行每層進行折射交點的計算，計算之後的結果已貼圖的方式傳給下一個模型進行多個模型的折射。

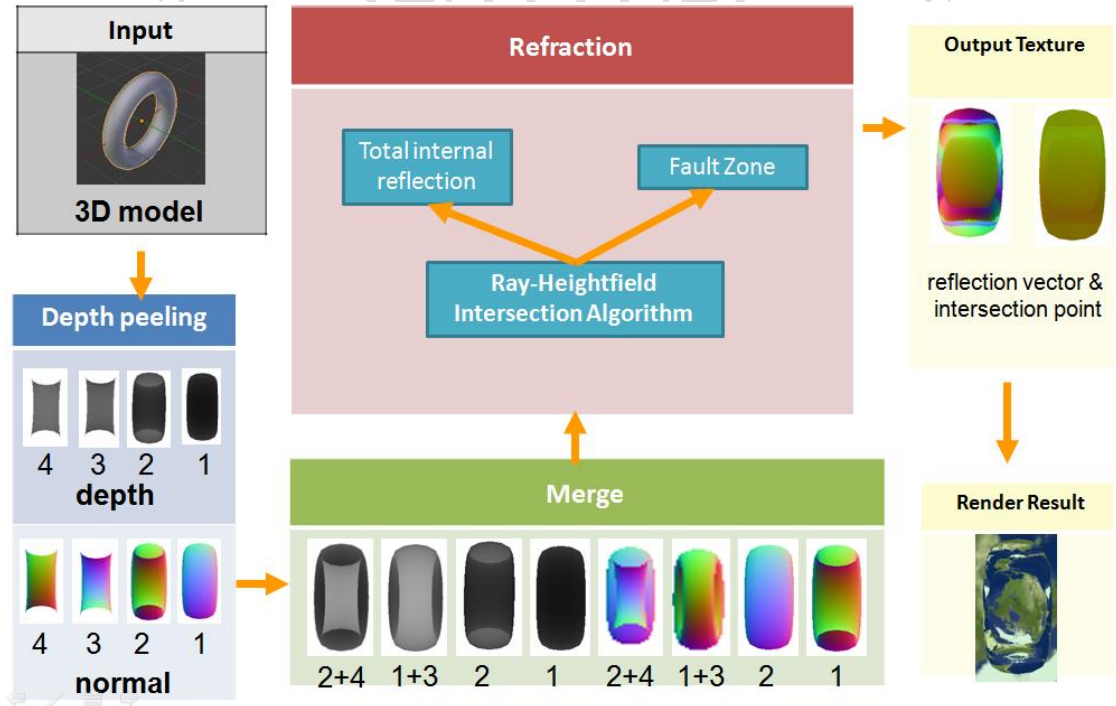


圖 3.5: 單一物體折射程式架構

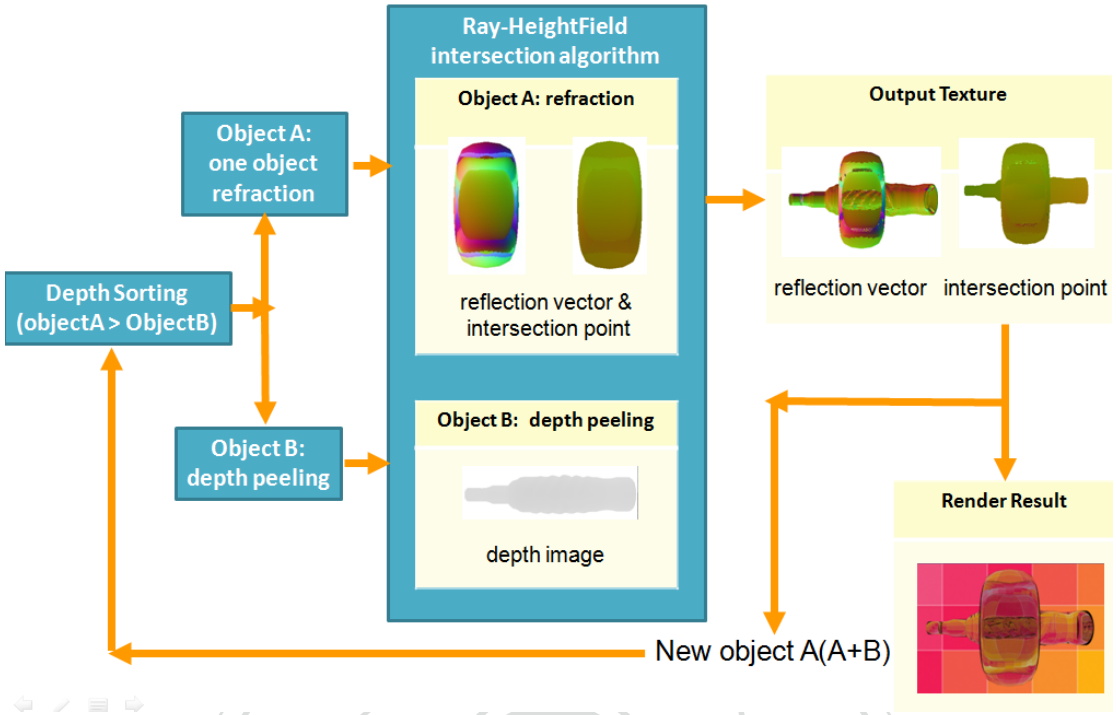


圖 3.6: 多物體折射程式架構

3.3 Render to texture 與光線相交演算法

Render to texture 是常用的一項基本的繪圖技術，顧名思義就是把 render 目標從 frame buffer 變成一個貼圖。這樣就可以把一個場景 render 後存成貼圖方便讀取，做出各種特效。而本論文在利用 GPU 做計算的時候也通過 render to texture 和 GPU 交換數據。

其技術關鍵在於 OpenGL 3.0 之後的版本，提供了一種預設不常顯示的 frame buffer 對象的界面。為了和原本 OpenGL 預設的 frame buffer 產生區別，這種 frame buffer 以物件的方式來表示。通過使用 frame buffer object（FBO），OpenGL 可以將 render 的結果輸出到 FBO，而不是傳統的 OpenGL 預設的 frame buffer。

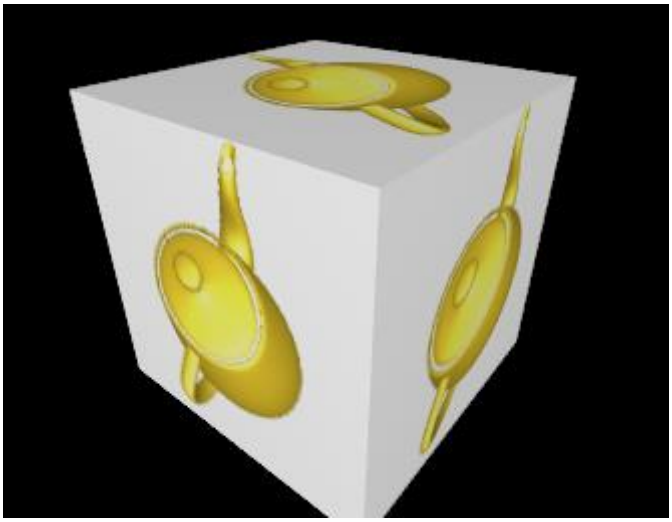


圖 3.7: 將茶壺以 render to texture，並貼圖到立方體上

在 render to texture 的技術之下，我們可以使 relief mapping 所使用的光線相交演算法得以提升效率，首先說明 Relief mapping 運用在折射中的方法:

在[Fabio 2005]所提到的光線高度場相交演算法是 relief mapping 的核心演算法。而在[Manuel 2007]中提出了運用光線高度場相交演算法來搜尋兩次折射向量與物體的的交點。

首先，以視點位置得出 frustum，以此 frustum 得出 Zmin，再以深度資訊得到 Zmax，作為之後查詢的深度 range。

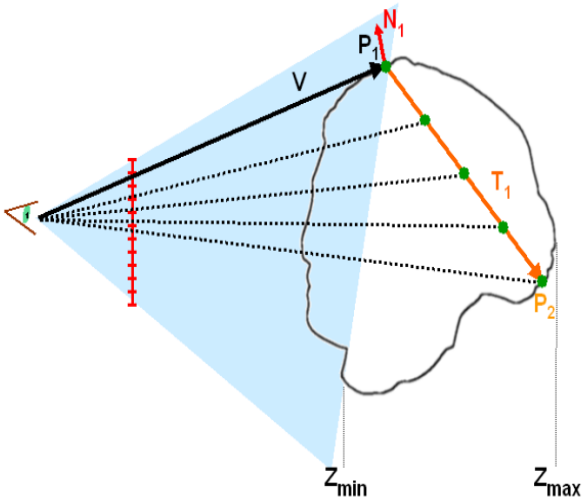


圖 3.8: Zmin 與 Zmax 求法[Manuel, 2007]

有了查詢的 range 之後，以 T1(第一次折射後之向量)正規化之向量為基底求得與 Zmin 和 Zmax 相交的 S, E 位置:

$S = P1 + (Z_{min} - Z_{p1})(\widehat{T1}/\widehat{T1}.z)$,

$E = P1 + (Z_{max} - Z_{p1})(\widehat{T1}/\widehat{T1}.z)$

再用插值公式，找出$\overrightarrow{SE}$中的點，及運用 relief mapping 的搜尋方式更精確地找到 P2(如下圖)

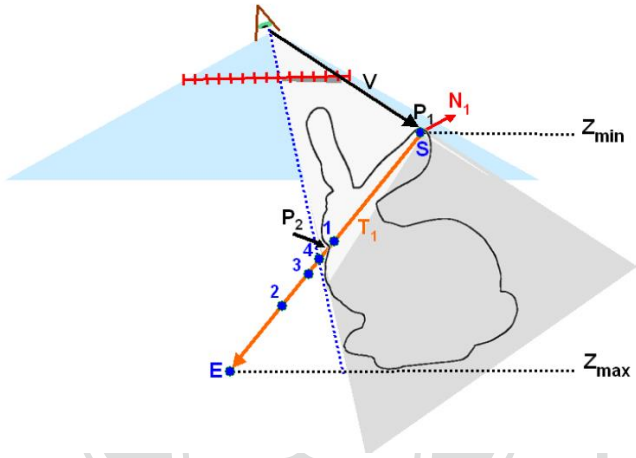


圖 3.9: 搜尋 P2 的方法[Manuel, 2007]

我們使用 GLSL 實作[Manuel 2007]，運用高度場與法向量資訊，求得正確的折射向量，並將所得的折射向量帶入環境貼圖，得到正確的折射影像。

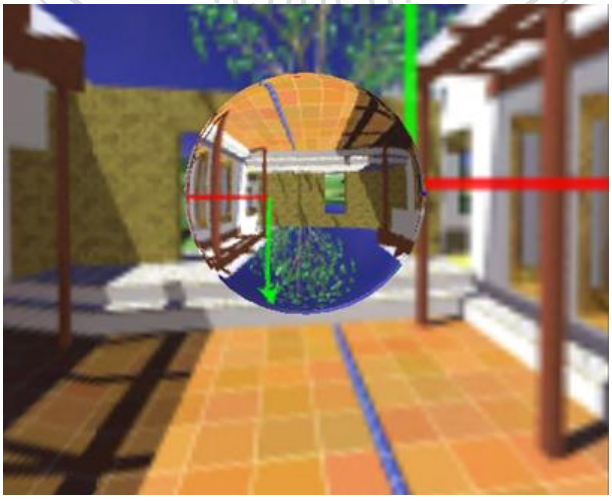


圖 3.10: 以 glsl 實作兩層折射

3.4 四層深度拆解

本論文採用了 2008 年 NVIDIA depth peeling 實作中由近到遠的剝離方法，主要概念就是對於每一個 fragment 比較剝離後的深度貼圖的深度值與當前 fragment 的深度，當 fragment 深度>=深度圖深度時即剝離，如圖 3.10(左)，而比深度圖深度小的則保留做為下一層的剝離深度圖如圖 3.10(右)。實作部份，我們在得到深度值得同時也記錄法向量的值，結果如圖 3.10。

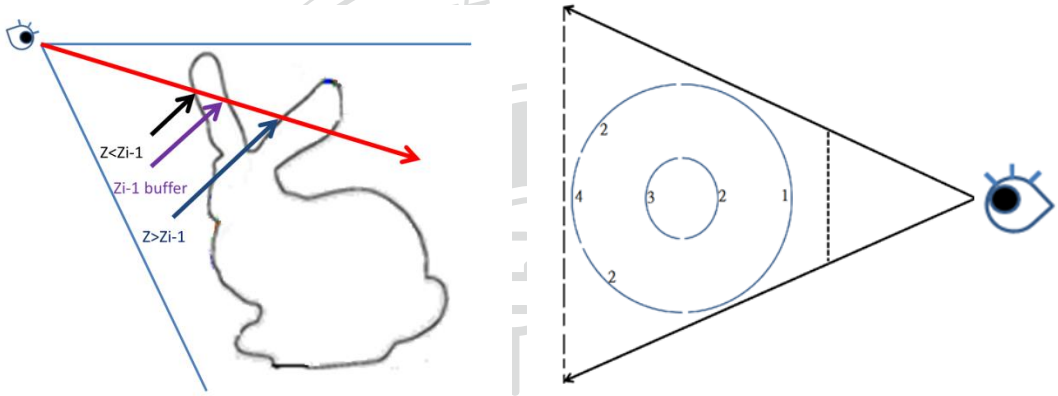


圖 3.11: depth peeling 示意圖(左) 甜甜圈模型為例的分層(右)

演算法如下:

```
//初始化第一層深度剝離的深度圖
For(深度剝離的層數)
    //比較當前 fragment 與深度圖的深度
    Fragment 深度>=深度圖 ->剝離
    fragment 深度<深度圖 ->保留並記錄深度成為下一層的深度圖
```

glsl 的 fragment shader:

```
uniform samplerRECT DepthTex;
void main(void)
{
    float frontDepth = textureRect(DepthTex, gl_FragCoord.xy).r;
    if (gl_FragCoord.z <= frontDepth) {
        discard;
    }
    gl_FragColor = vec4(gl_FragCoord.z, gl_FragCoord.z, gl_FragCoord.z,
1.0);
}
```

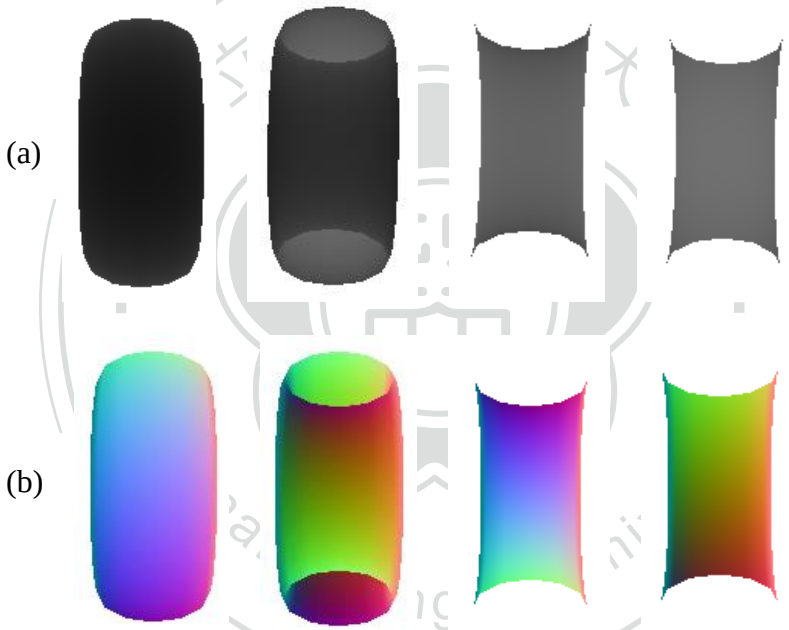


圖 3.12: 環形模型 depth peeling 後的四層深度(a)與法向量(b)，由左到右為 1~4 層

3.5 斷層處理

由圖 3.11 我們可以看到分層之後每層皆有跟其他層有重疊的地方，因而當折射光線靠近計算重疊處會產生判斷上的錯誤，因此必需進行斷層的處理。

3.5.1 斷層處理

擁有了四層的深度之後我們可以一、二層深度做光線相交算法再把結果與第三層做相交算法，最後再做第四層，然而由於內部深度會因為 3D 物體的形狀而有斷層的產生，如圖 3.13 紅色的部分即為甜甜圈模型在第二層深度產生斷層的地方。由於光線相交的演算法是以深度的大小來判斷其交點，因此斷層的出現，會造成判斷上的錯誤。

(a) (b)

圖 3.13: (a) 環形模型不產生斷層的角度 (b) 環形模型第二層深度產生斷層處

因此我們提出對於斷層深度處理的光線相交演算法，首先我們觀察發現，折射向量進入 3D model 後會被斷層所影響的情形有三種:

定義: 折射向量 ***T***，光線相交算法線性搜尋 step 的長度為 SIZE，搜尋到的深度比深度貼圖深度還小者差值為＋，反之為－。

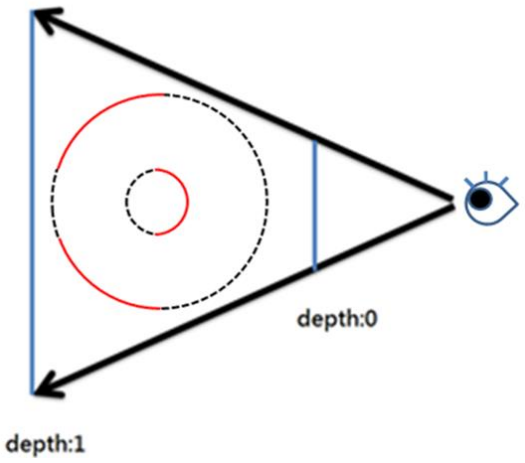
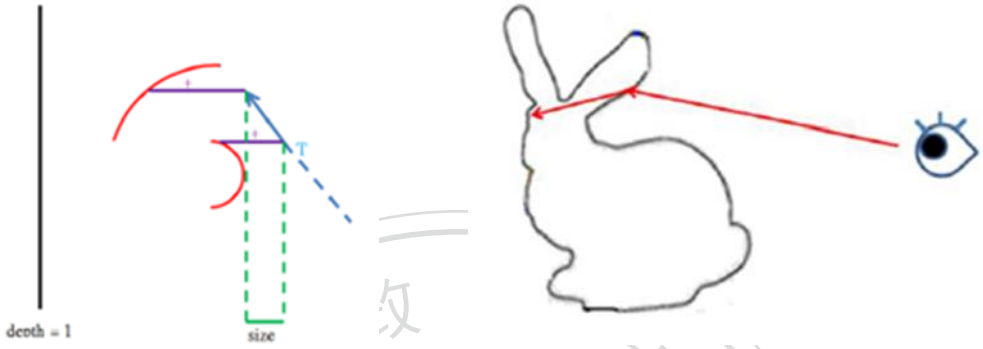


圖 3.14: 物體紅色實線為產生斷層的深度，黑色虛線為物體的輪廓

Case1:折射向量經過斷層的上方且不與之相交

此種情況下線性搜尋的前一個點與後一個點的深度雖然因為斷層而造成深度與深度貼圖的差值有劇烈變化，但因為差值還是為+。因此繼續線性搜尋，如圖 3.14。

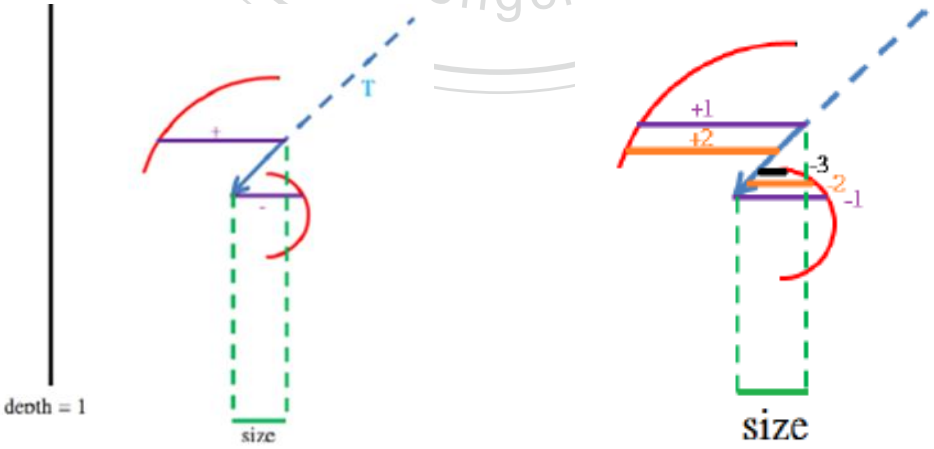


(a) (b)

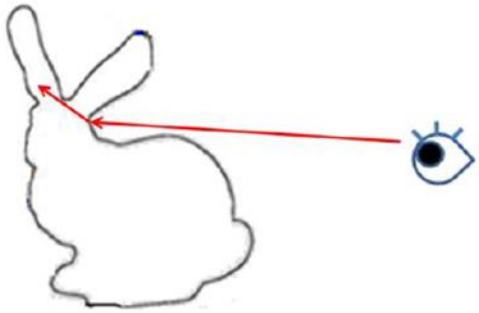
圖 3.15: (a) 3D model 產生的斷層 case1，T 為折射向量,(b) 以 bunny 為例

Case2:折射向量經過斷層中間且不與之相交

此種情況下線性搜尋的前一個點的差值為+，後一個點的差值為-，因此線性搜尋判斷停止，如圖 3.15(左)，接著進行二元搜尋，在搜尋時分別記錄最後一次搜尋到差值為+與-的深度，當二元搜尋結束時比較最後一次搜尋到+與-的深度貼圖深度，兩者差值必定會大於 size 的值，如圖 3.15(右)。



(a) (b)

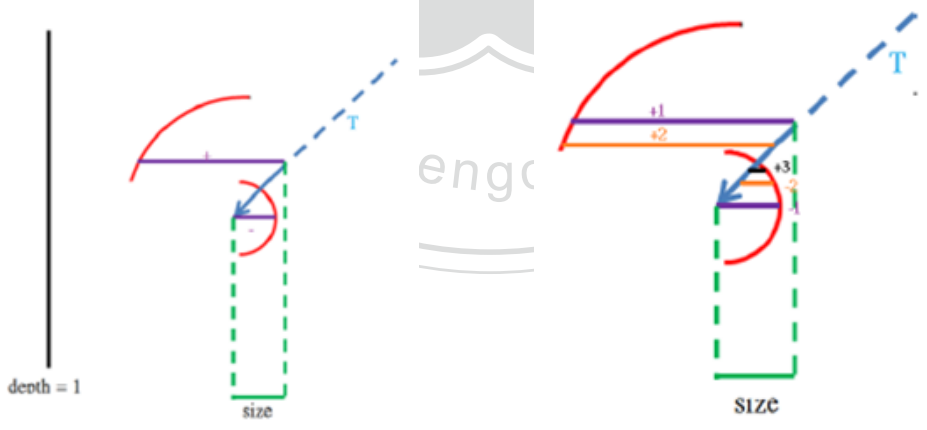


(c)

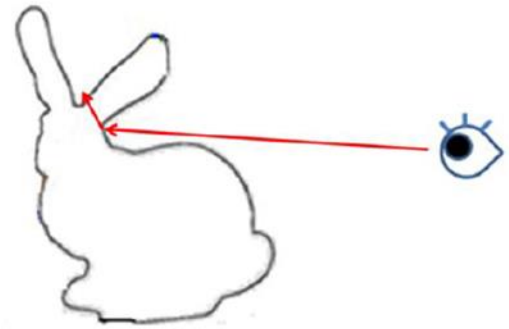
圖 3.16: (a) 3D model 產生的斷層 case2，(b) T 為折射向量 (右)二元搜尋(c) 以 bunny 為例

Case3 折射向量經過斷層且與之相交

此種情況下線性搜尋的前一個點的差值為+後一個點的差值為-，因此線性搜尋判斷停止，如圖 3.16(左)，接著進行二元搜尋，在搜尋時分別記錄最後一次搜尋到差值為+與-的深度，當二元搜尋結束時比較最後一次搜尋到+與-的深度貼圖深度，兩者差值在大部分的情況下小於 size 的值，如圖 3.16(右)。



(a) (b)



(c)

圖 3.17: (a) 3D model 產生的斷層，case3 (b) 二元搜尋，(c) 以 bunny 為例

分析 case2 與 case3 之差別處在於二元搜尋時 case2 會逼近斷層處，而 case3 則是交點處，因此可以產生區別。

Case3 二元搜尋後兩者差值在大部分的情況下會大於 SIZE 的值，也就是說，會有特例的情況，如圖 3.18，最後的深度圖差值>size，解決的方法就只有增加二元搜尋的次數或者將 size 的值調整變小。

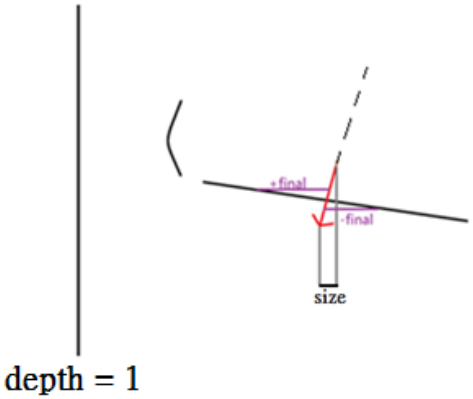


圖 3.18: case3 的特例

Case2 由於判斷線性搜尋停止但實際上並未停止，因此我們需要額外的 flag 來幫助我們判斷折射向量在斷層下的深度判斷，如圖 3.19。

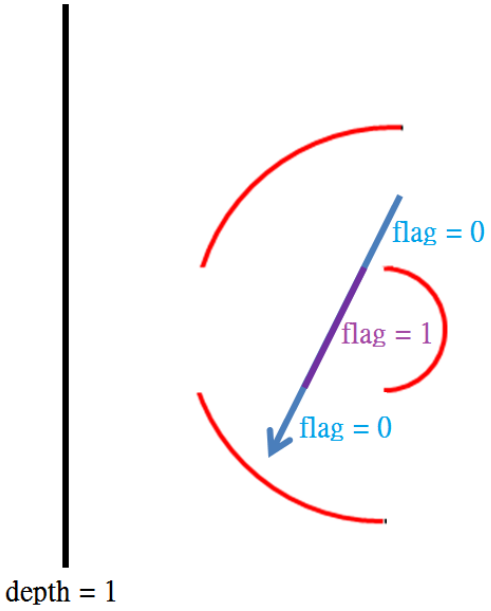


圖 3.19: 若 case2 不與斷層相交則繼續線性搜尋

以上敘述的斷層是 model 與 model 之間所產生的，另外也有模型與背景產生的斷層，而這種斷層在前兩層的計算並不會產生問題，因為前兩層的深度貼圖對於每一個 fragment 都可以找到對應的值，但到了第三層，由於模型的某些部份只有到兩層的深度，而造成第三層的計算會產生如圖 3.20(左)之錯誤。以甜甜圈第四層為例子產生錯誤位置如圖 3.20(右)所示，紫色圓圈圈起來的部分就有可能因為斷層太短而造成判斷錯誤的情況。

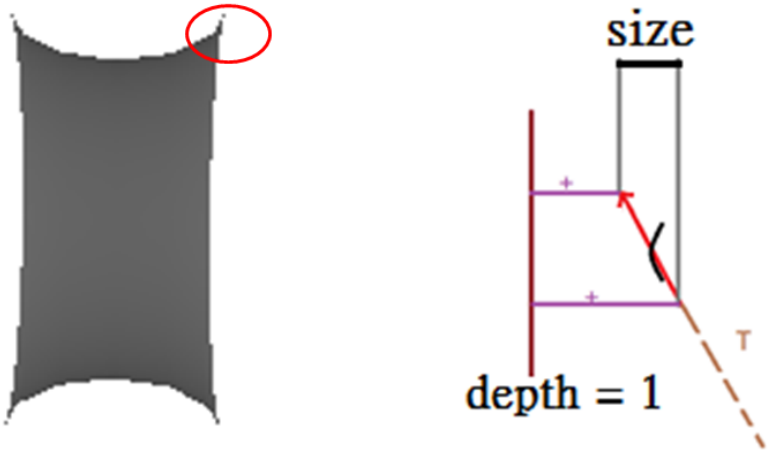


圖 3.20: (左)甜甜圈的第四層的錯誤 (右)二層深度後的錯誤

3.5.2 深度合併

因為圖 3.20 的錯誤，我們提出多層深度合併的方法來解決此問題，我們觀察到四層深度的情況下第二層與第四層深度往往會有相連的地方，而第一層與第三層也一樣，如圖 3.21 所示，黑色數字表示層數，藍色向量為某一折射向量，我們發現若能把第四層深度以第二層合併並保留第四層深度便可以解決圖 3.20 因為深度貼圖並未對應到交點的問題。

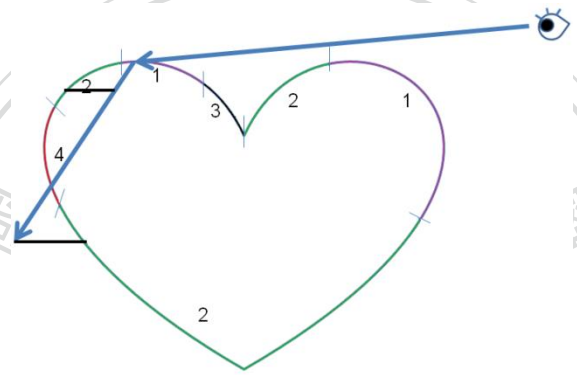


圖 3.21:可以合併第一層與第三層&第二層與第四層

合併方法以甜甜圈第四層為例，首先第四層深度減掉第二層深度得到圖 3.22 的橘色線(差值)，接著以第二層深度加上之前所產生的差值，if(產生的深度差值為 1)則為第二層深度 else 第四層深度，最後結果在取代原本的第四層深度。

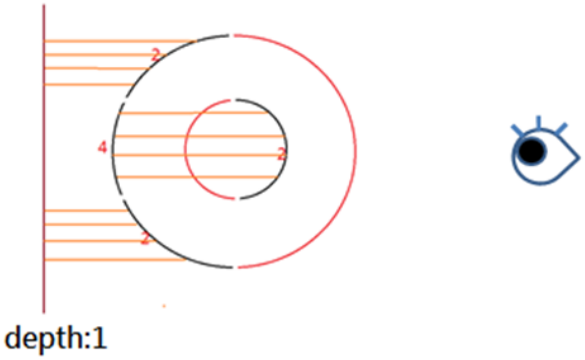


圖 3.22:合併第二層與第四層

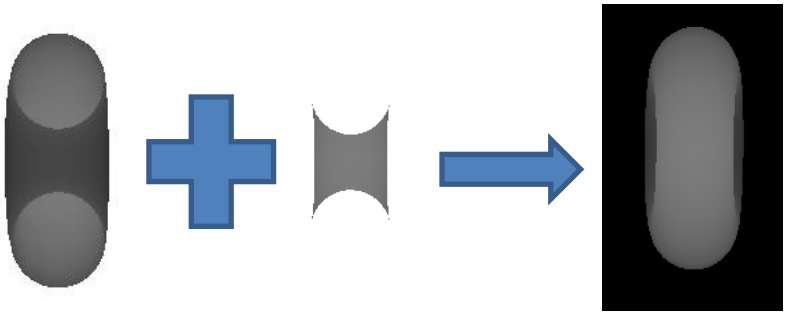


圖 3.23: 取代的第四層深度

3.5.3 Size 大小的影響

由於光線相交演算法的精準度取決於 size 的大小，如果太大或太小都會產生錯誤的結果，當 size 太大的時候容易產生如圖 3.24 的右邊:由於牛的模型中腳的部分比較纖細。我們設定 frustum 深度由 0 到 1，當 size 設定為 1/5(線性搜尋 5 次)時會產生找不到交點的情況而產生錯誤的結果，至於當 size 太小，如 1/500(線性搜尋 500 次)的時候又因為斷層的判斷方式導致判斷錯誤的情況如圖 3.25，因此在時間與品質的考慮下，1/50 最為適合(線性搜尋 50 次)。

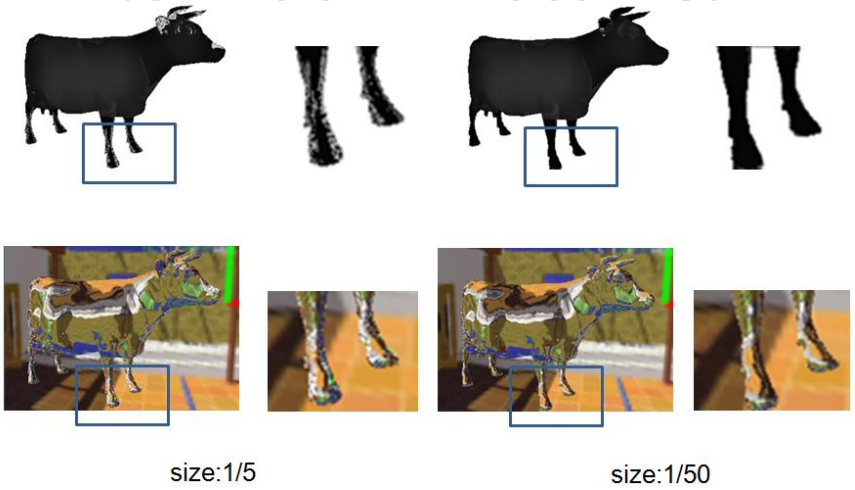


圖 3.24: 左: size=1/5 的深度圖於折射結果，右: size=1/50 的深度圖於折射結果

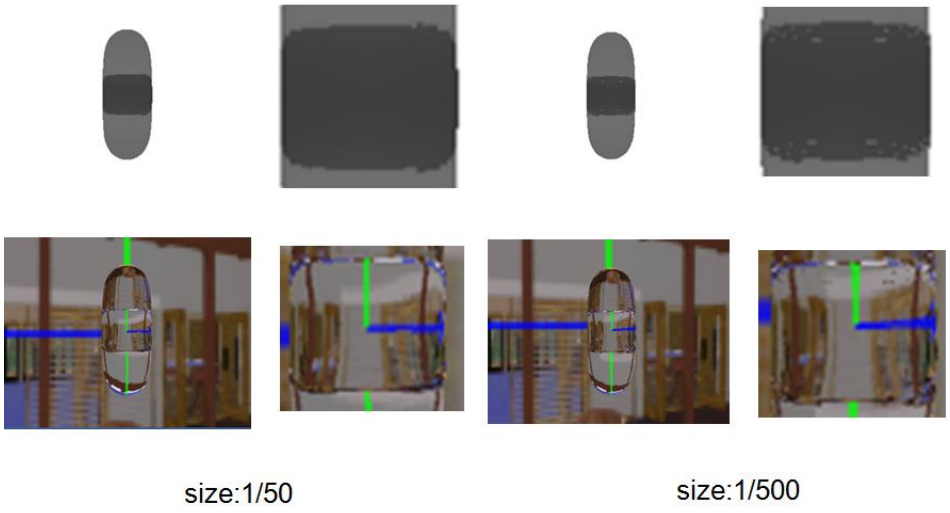


圖 3.25: 左: size=1/50 的深度圖與折射結果，右: size=1/500 的深度圖與折射結果

3.6 全反射的計算

全反射是折射必須要進行判斷的現象，由於全反射的產生只會在光線由折射物質內要穿出折射物質外時產生，因此我們只需要對第二層與的四層折射進行全反射的判斷，依照 GLSL 的折射函數，當全反射發生的時候會回傳-1 的值，因此我們在做完一次光線的搜尋之後判斷是否為全反射，若否則繼續下一層的搜尋，反之進行一次反射計算之後，以反射向量進行新的光線搜尋。

然而以原本的光線搜尋演算法去計算的時候往往出現錯誤，如圖 3.26 所產生的錯誤，主要的原因在於經過上一層的光相交演算法之後，所尋到的交點並無落在交點的平面上而是近似的結果，並且會超出一點點的值，如圖 3.27 藍色箭頭所示，因此在做反射的向量搜尋時，會以末端做為起點，然而搜尋時的 size 大小就會因為超出的起點而造成兩種錯誤的狀況，當 size 太小，小到小於起點到起點對應的貼圖深度的差值時，在經過原本產生全反射面時會判斷為找到新的交點(如圖 3.28(左)紫色箭頭所標示)，因此產生錯誤，而當 size 太大時容易突出纖細的模型因此在二元搜尋下找錯交點，為了解決此問題，我們採用了小的 size 先找尋第一個交點之後再繼續搜尋到第二個交點，如圖 3.29 所示，改善結果如

圖 3.30

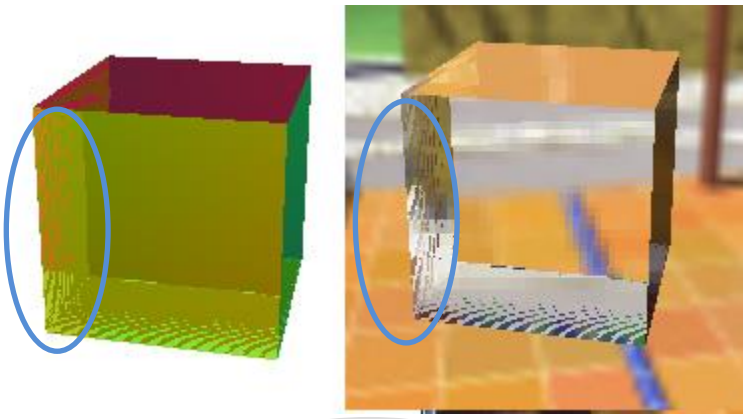


圖 3.26: 計算全反射之後產生的問題，左:全反射後的法向量 右: render 的結果

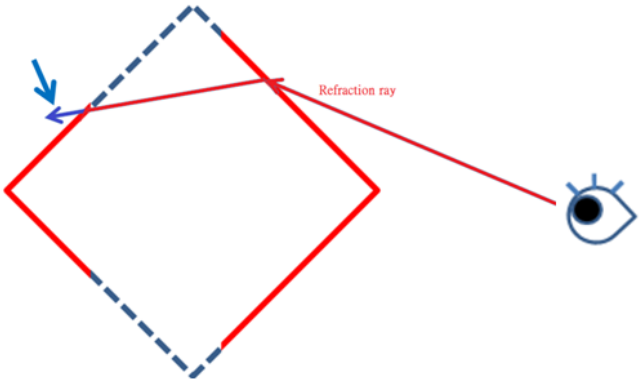


圖 3.27: 經過光線搜尋之後的結果，藍色箭頭為反射的起點 ，由於第一次搜尋的結果會逼近交點但是其深度會大於交點，因而產生全反射第一次搜尋時的錯誤

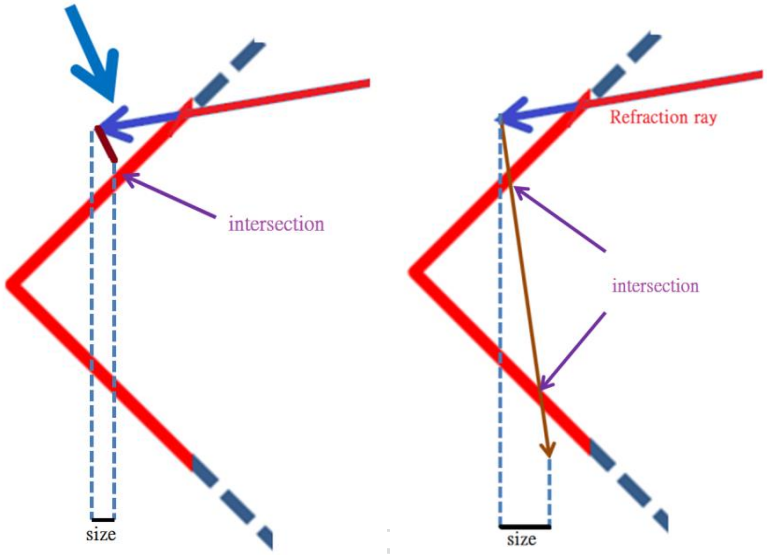


圖 3.28: (左)全反射 Size 太小，導致再以反射向量搜尋交點的時候受到第一次折射向量搜尋的誤差所影響，找到了與第一次搜尋交點相同平面的焦點，(右)全反射 Size 過大而物體太過纖細的情況下，反射向量在進行一次線性搜尋之後就進入二元搜尋，但因為第一次的交點涵蓋在這次線性搜尋中，因此有可能在二元搜尋的結果找到第一次搜尋的交點。

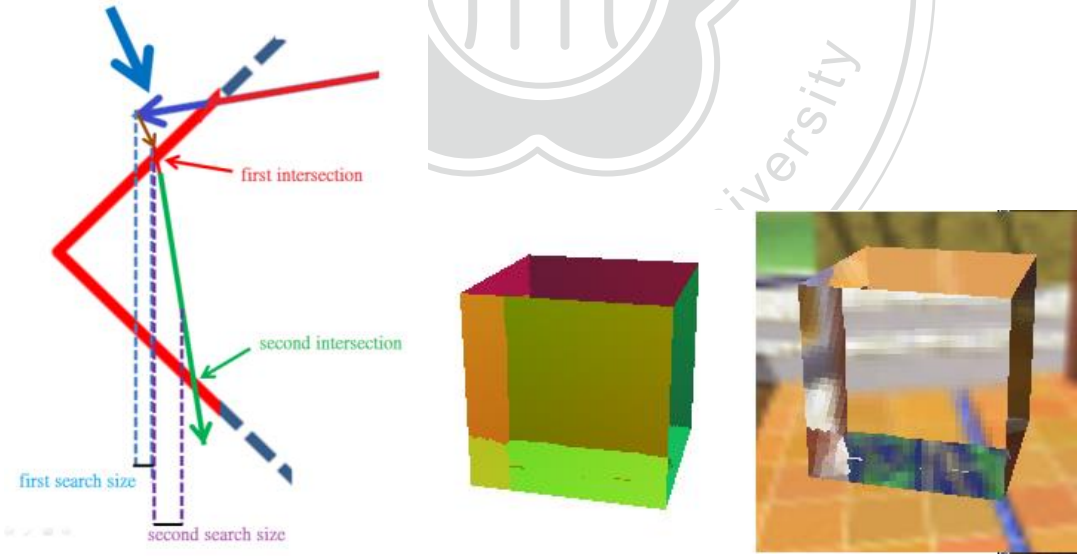


圖 3.29: (左)我們先以一個小的 size 搜尋第一個交點以確保在往後的搜尋不會找到，再進行正常的搜尋找到第二個交點(右)全反射修正結果，左:法向量，右 render 結果

3.7 單個物體與多個物體的折射

在已知單個物體的 image space 折射方法之後，接著由兩個物體的折射延伸到多個物體，我們先對所以物體與眼睛的深度做排序，排序之後我們以離眼睛最近的物體開始做單一物體的折射運算，如圖為兩個物體，我們先對物體 A 做折射運算並輸出折射向量與折射點，物體 B 以 depth peeling 分層，如圖 3.31，

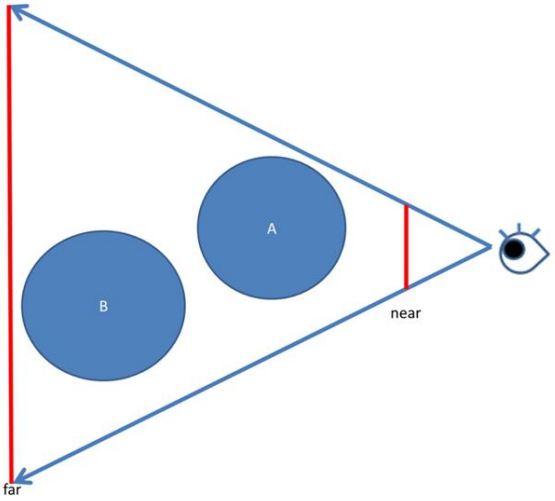


圖 3.30: 兩物體間的折射情況。物體 A 跟 B

圖 3.31: (左)物體 A 的 T2 與 P2 ；(右)物體 B 的深度

接著我們對兩物體之間的關係作分析，我們可以依照視點經過物體分成三種 case: 1. 視點向量並無經過物體 A 並直接與未被 A 遮蔽的 B 表面相交；2.視點向量與 A 表面相交且在經過物體 A 的折射向量不會與 B 表面相交；3.視點向量與 A 表面相交且在經過物體 A 的折射向量與 B 表面相交，如圖 3.32 所示

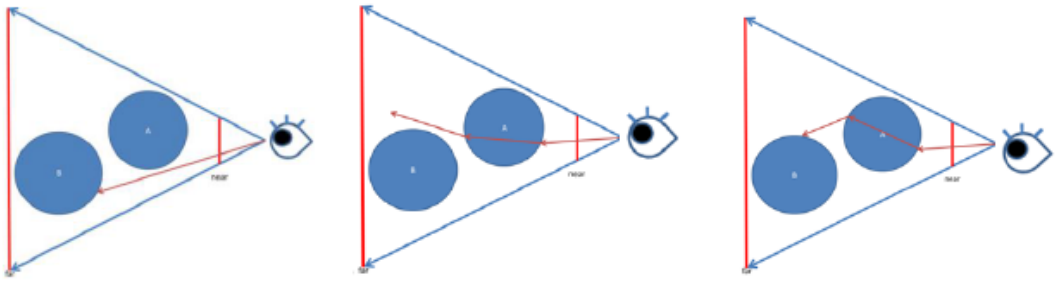


圖 3.31: 物體 A 跟 B 與視點的關係

在此分類之下，分別對每個 case 做計算:

case1:使用單個物體的 image space 折射即可。

case2 與 case3: 視點向量以物體 A 貼圖輸入的折射向量 T2 與折射向量的出發點 P2 作為判斷，接著以 T2 與物體 B 的深度進行光線高度場相交演算法，若有交點則為 case3 若無則為 case2，若是 case3，以搜尋到的交點與折射向量繼續進行物體 B 單一物體的折射運算，若是 case2 則只需繼續記錄原本的折射向量即可。演算法如下:

```
If (T2texture(eye ray.xy==1) //以視點讀取貼圖座標深度 等於 1 表示(case1)
    Final vector = T2&P2
else
    D = ray-height-field intersection algorithm(T2, P2,objectB d);
//以 T2,P2 尋找 objectB 的交點
If (D==1)//沒有與 objectB 相交(case2)
    Final vector = T2&P2
    Else (case3)
        ObjB_P1 =P2+D*T2;//計算物體 B 與折射向量的交點
        ObjB_T2&P2 = one obeect refraction;
    Final vector = ObjB_T2&P2;
```

有了兩個物體的折射算法，第三個物體的算法即是把前面兩個物體最後輸出的 T2 與 P2 當成一個有八層折射物體的輸出，然後與第三個物體作兩個物體的折射計算(圖 3.33)。結果如圖 3.34 所示。

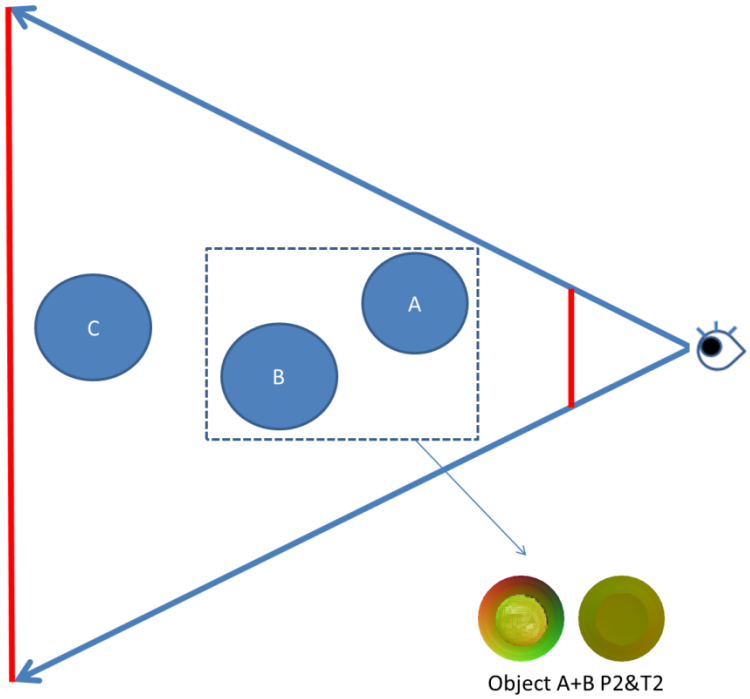


圖 3.32: 加入物體 C 後，物體 A 跟 B 視為同一個物體，輸入 Object A+B 的 P2&T2 計算兩個物體 Object A+B 與 Object C 的兩物體的折射運算

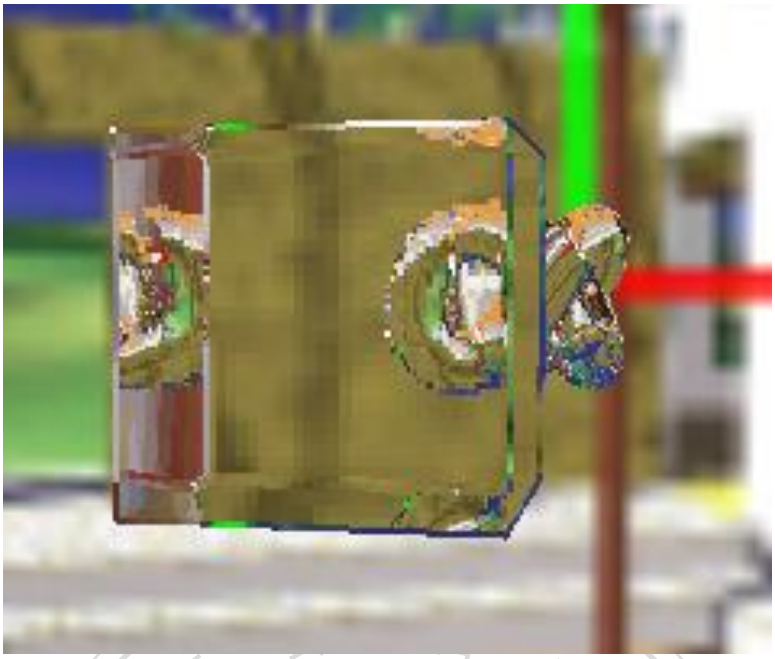


圖 3.33: 正方體 model 後面有 knot model 的折射結果

第四章

實驗結果與討論

我們在 windos 7 作業系統下開發以 C++語言編寫，以 OpenGL 為繪圖函式庫， 使用 opengl shading language 為著色語言，以真實世界的場景，光線追蹤的結果與一層折射、二層折射、四層折射比較在不同模型的折射情況，此外我們以 OptiX 作為效率的對照組，並且分析我們方法的時間複雜度與光線追蹤法差異。

4.1 真實世界的透明物體與光線追蹤法

我們可以從真實世界的透明圓球去觀察折射的現象，圖 4.1(左)為德國攝影師 Markus Reugels，利用攝影機拍出了水滴利用折射原理的藝術作品。他利用折射原理，將水滴與背景圖片完美的結合在一起。另一張圖為美國攝影師 Grayce Pedulla-Dillon，利用裝滿水玻璃杯，配合色彩強烈對比的背景，巧妙了運用折射效果，以強烈的色彩勾勒出了玻璃杯的曲線。

圖 4.1:: 現實世界的透明物體折射(左) Markus Reugels 攝(右) Pedulla-Dillon 攝

我們利用 blender 內建的光線追蹤法的結果可以明顯看出圓球折射結果如同現實世界，上下顛倒左右相反。

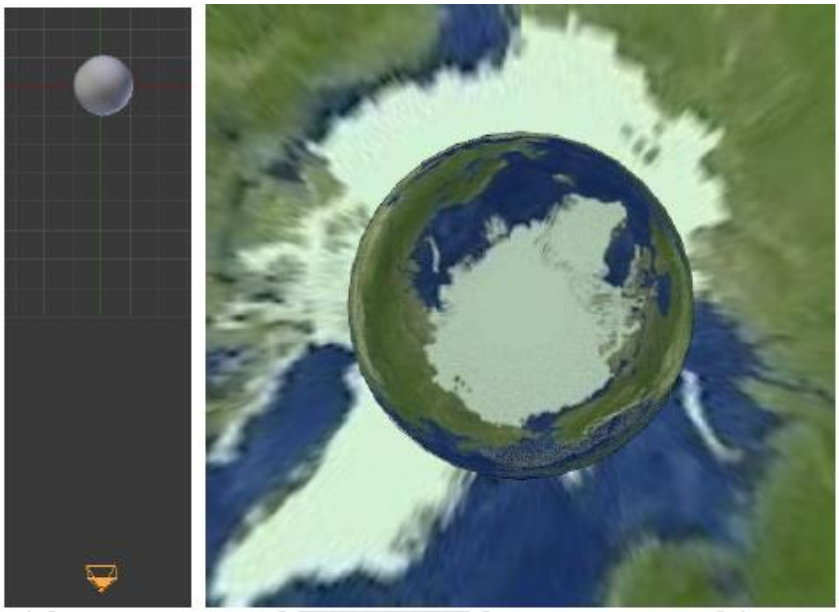


圖 4.2: (左) 場景設計 (右) ray tracing 透明球模型

以 glsl 本身所提供的折射函數所計算的一層結果如圖 4.3。可以明顯看出放大比例不真實。圖 4.4 是二層折射的結果，由於球的折射只需要二層因此可以看出與光線追蹤的結果極其相似。

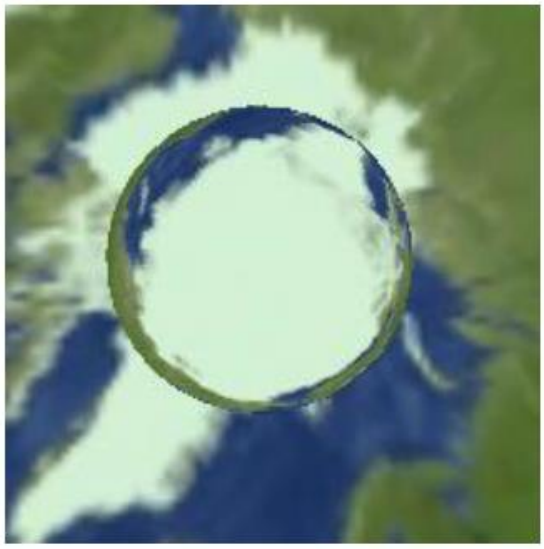


圖 4.3: 一層折射

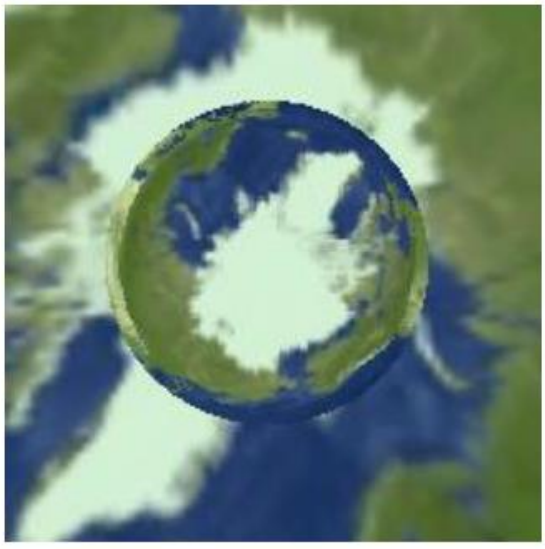
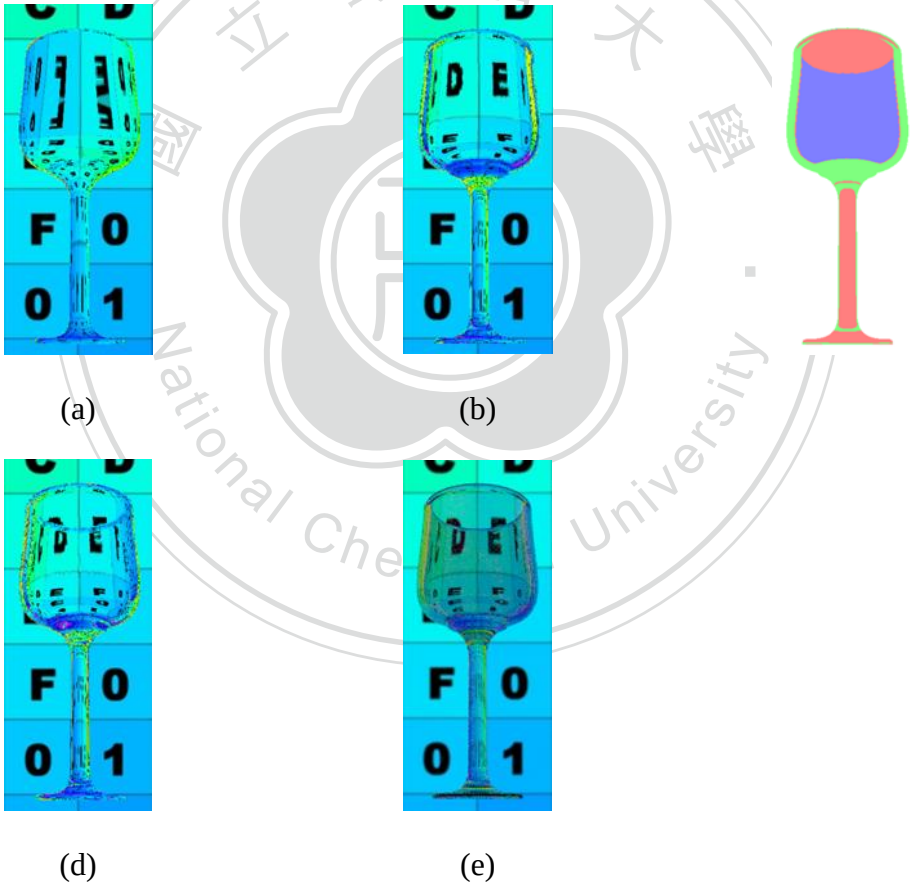


圖 4.4: 二層折射

4.2 四層深度的透明物體

圖 4.5 為高腳杯模型折射的結果，我們依照一層折射、兩層折射、四層折射與 ray tracing 做為比較，我們可以發現一層折射並無法清楚的知道物體的形狀，只能大略的知道表面的凹凸起伏，我們實做出來的兩層折射[Manuel 2007]雖然比起一層折射的方法更為接近光線追蹤法的結果，卻也無法正確得知彎曲的折射樣貌(例如杯緣部分的折射形狀)，我們方法所提出來的四層折射最為接近 ray tracing 的結果，我們可以立刻觀察到物體後面的正確的折射，並且折射之後的結果也接近對照組的結果。

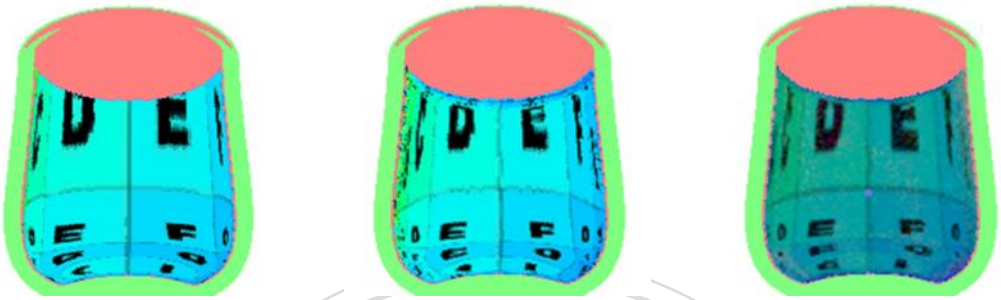


(a) (b)

(d) (e)

圖 4.5: 高腳杯模型(a) 一層折射結果，(b) 兩層折射結果，(c) 四層折射分析，紅色:兩層，藍色:四層，綠色:全反射，(d) 四層折射結果，(e) ray tracing 的結果

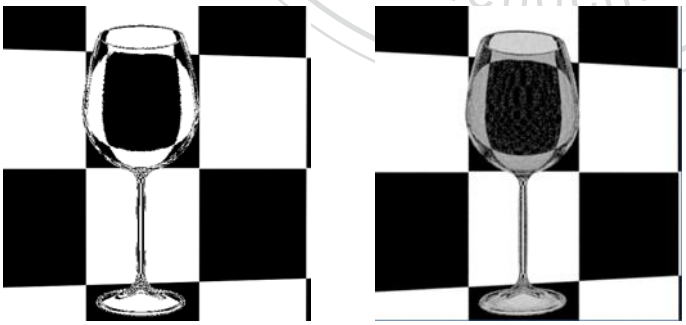
我們將高腳杯模型四層折射與兩層折射的解果拆開來並放大比較，如圖 4.6 所示，可以發現兩層折射的結果形狀與光線追蹤法的結果明顯的不同，圖 4.7 為不同貼圖的折射結果。



(a) (b) (c)

圖 4.6: 四次折射處比較(a) [Manuel, 2007]二層折射結果，(b) ray tracing 折射結果，(c) 我們的方法兩次折射結果

(a) (b) (c)



(d) (e)

圖 4.7: 高腳杯模型(a) 一層折射結果，(b) 兩層折射結果，(c) 四層折射分析，紅色:兩層，藍色:四層，綠色:全反射，(d) 四層折射結果，(e) ray tracing 的結果

圖 4.8 為環形模型折射的結果，我們依照一層折射、兩層折射、四層折射與 ray tracing 做為比較，我們可以發現內圈的部分在二層折射時所產生的結果並不特別明顯，在我們的方法中可以明顯看到後半部的環形形狀，圖 4.9 為不同貼圖。

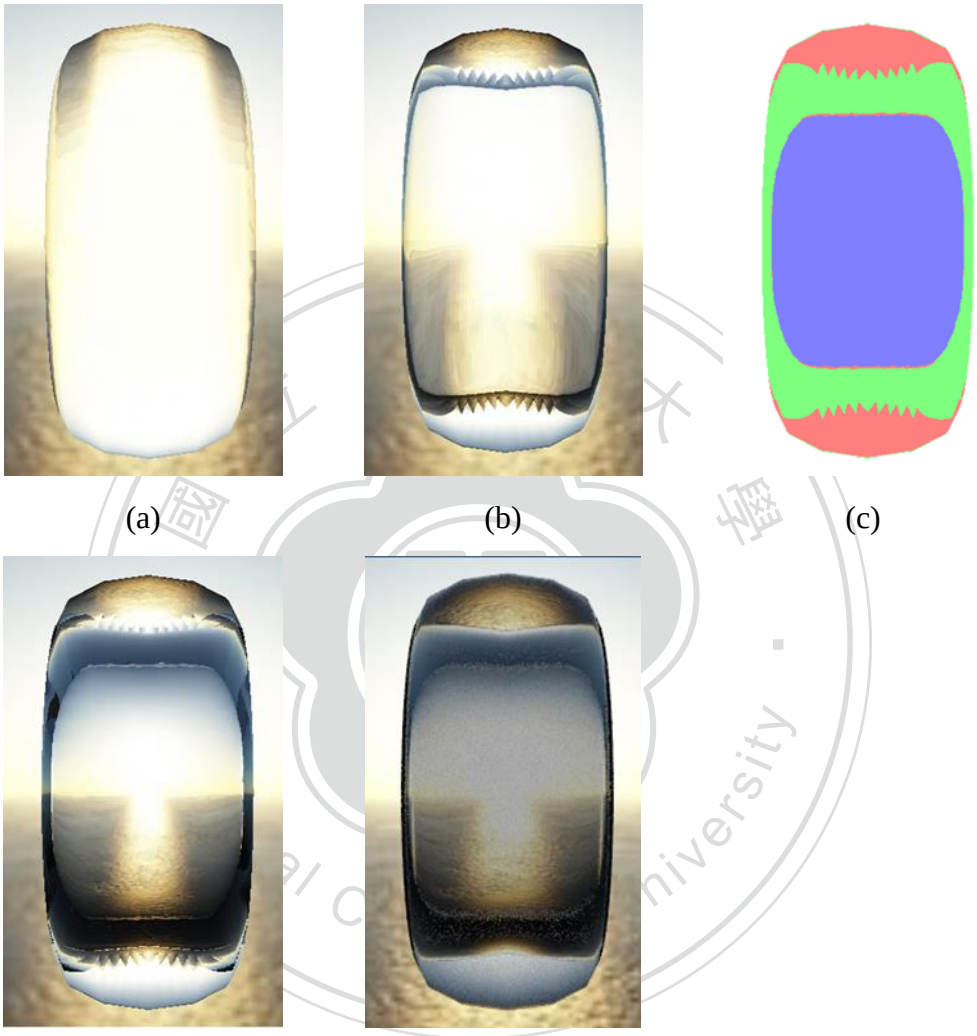


圖 4.8: 環形模型(a) 一層折射結果，(b) 兩層折射結果，(c) 四層折射分析，紅色:兩層，藍色:四層，綠色:全反射，(d) 四層折射結果，(e) ray tracing 的結果

我們以不同貼圖再看一次甜甜圈的模型(圖 4.9)，我們可以發現紅色方框為四次折射產生的地方，綠色方框為全反射產生的地方，兩處的結果接近光線追蹤法的結果。

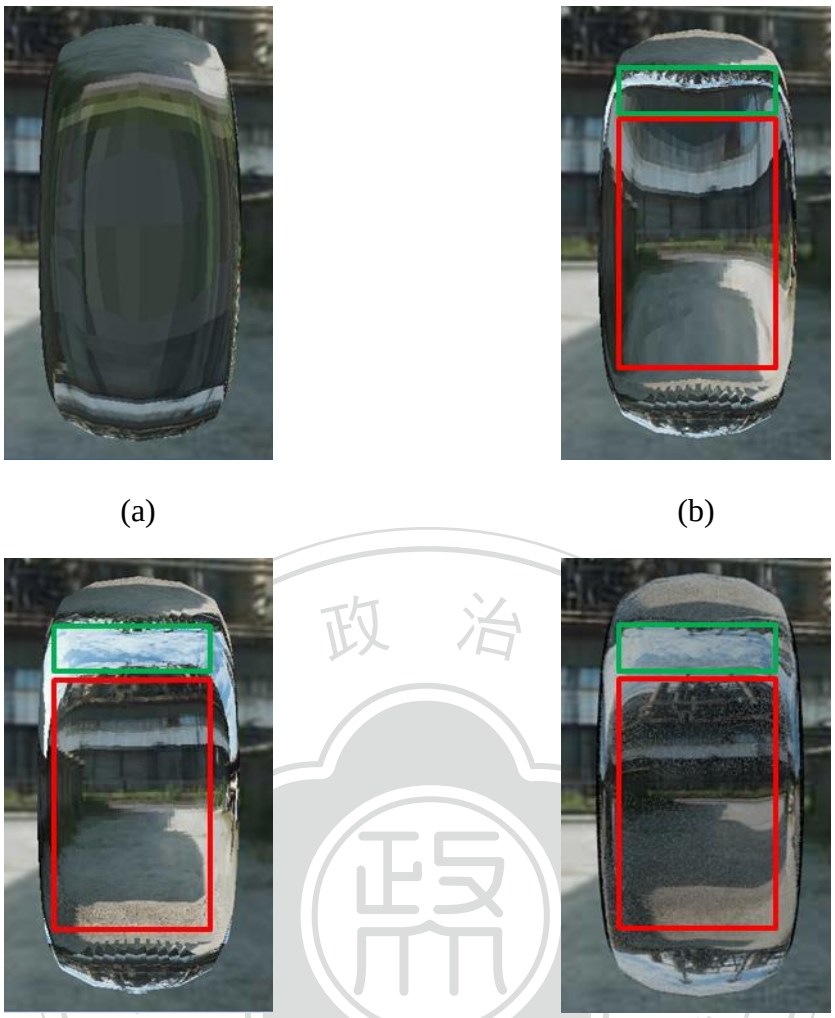
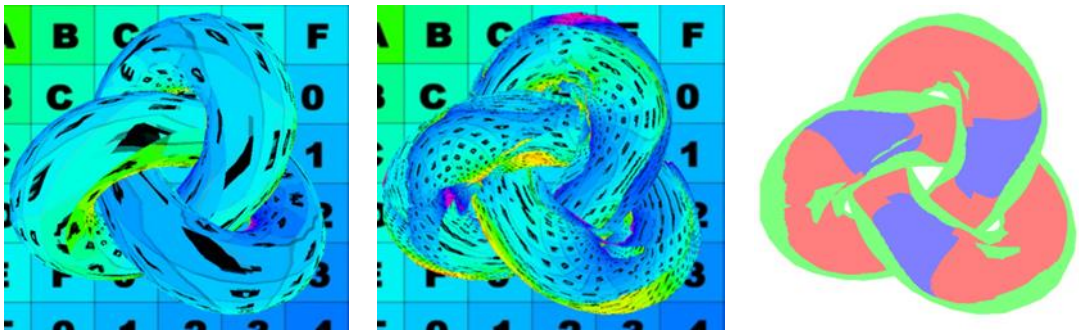
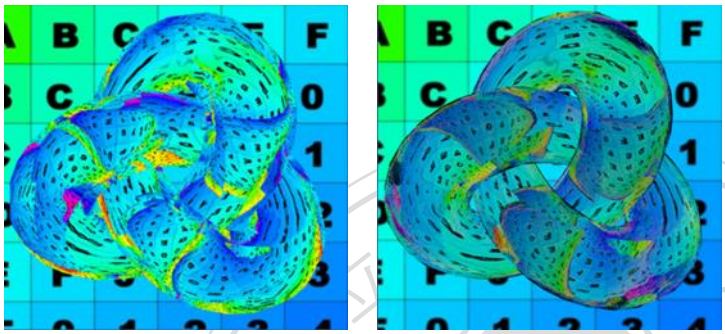


圖 4.9: 環形模型(a) 一層折射結果，(b) 兩層折射結果，(c) 四層折射結果，(d) ray tracing 的結果

knot 模型的特色在於有許多環形所組成，也因此四次折射較容易產生(圖 4.10~4.12)，我們可以發現一個場景中有三處產生四次折射，因此我們可以很清楚的觀察到四層折射處物體背後的環狀，且與光線追蹤法相似。



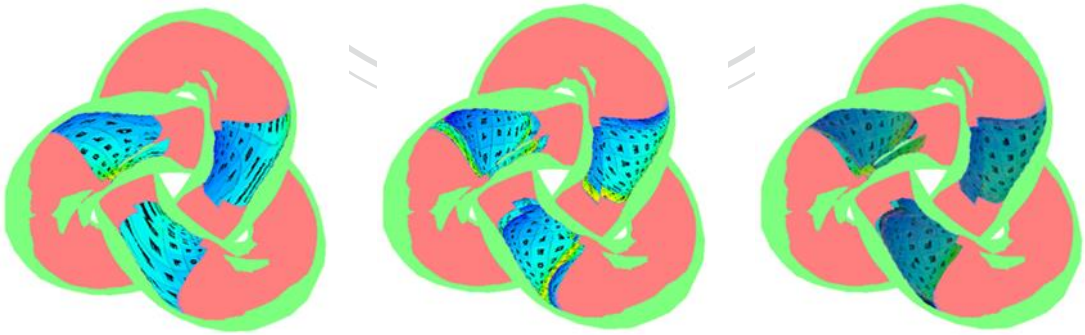
(a) (b) (c)



(d) (e)

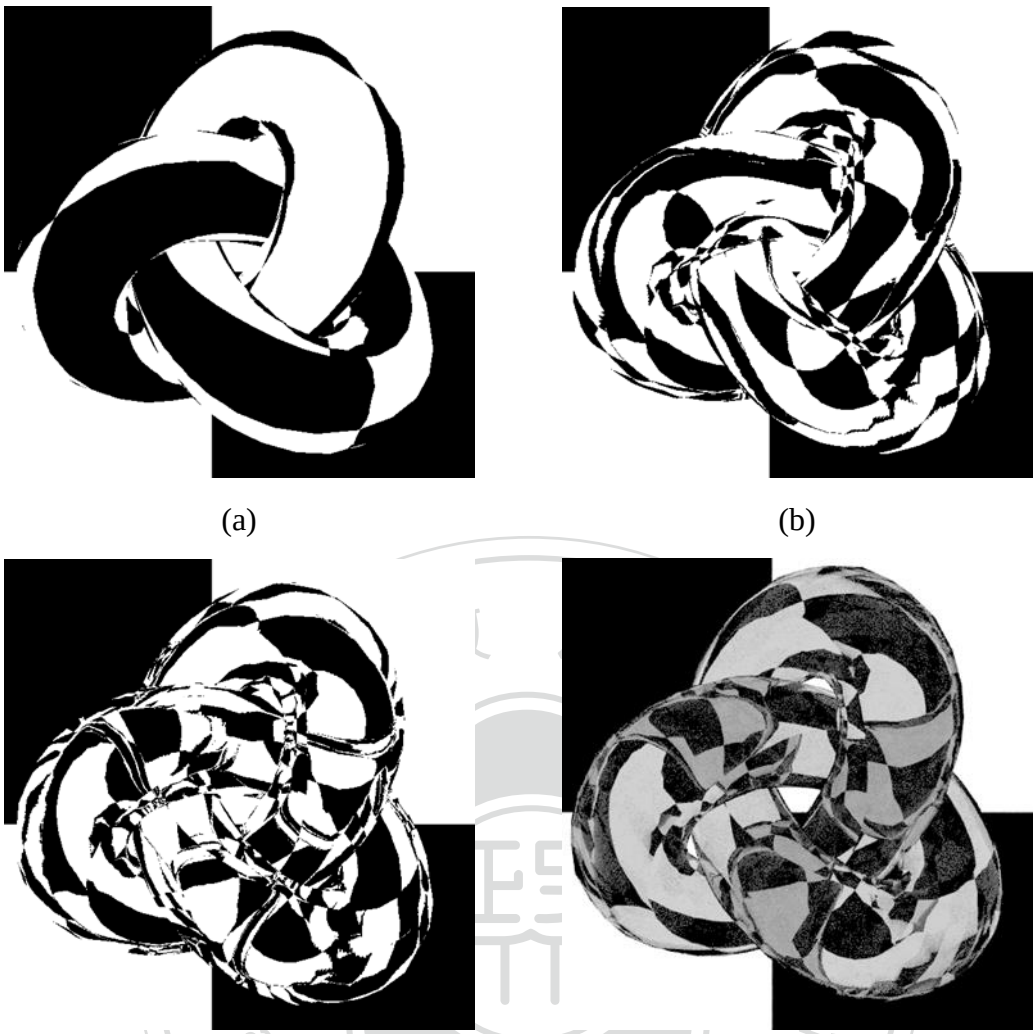
圖 4.10: knote 模型(a) 一層折射結果，(b) 兩層折射結果，(c) 四層折射分析，紅色:兩層，藍色:四層，綠色:全反射，(d) 四層折射結果，(e) ray tracing 的結果

我們將四次折射處放大來看，更清楚地發現[Manuel, 2007]的方法與我們的差異。



(a) (b) (c)

圖 4.11: knote 模型(a) [Manuel 2007]二層折射在四次折射處的結果，(b) 我們的方法四次折射結果，(c) ray tracing 折射結果

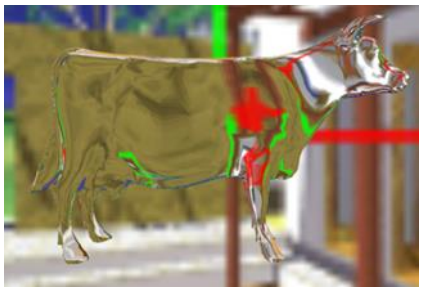


(a) (b)

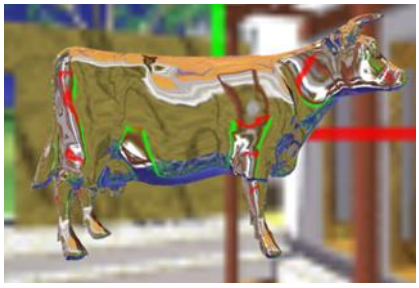
(c) (d)

圖 4.12: knote 模型(a) 一層折射結果，(b) 兩層折射結果，(c) 四層折射結果，(d) ray tracing 的結果

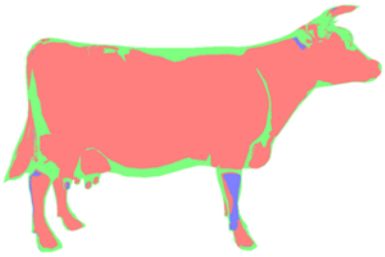
我們以更多較為複雜的模型所做出來的結果，牛的模型在腳與耳朵部分較容易出現四次折射(圖 4.13)，模型身體部分表面較為光滑，可以明顯表現折射的形狀，圖 4.14 我們以較為簡單的環境貼圖實驗，我們可以發現紅色方框為四次折射產生的地方，其結果接近光線追蹤法的結果。



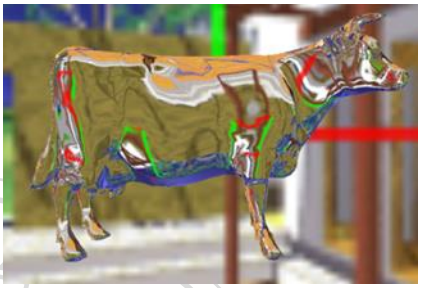
(a)



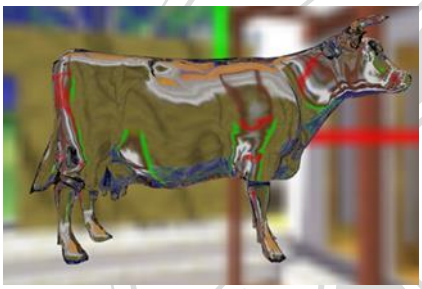
(b)



(c)

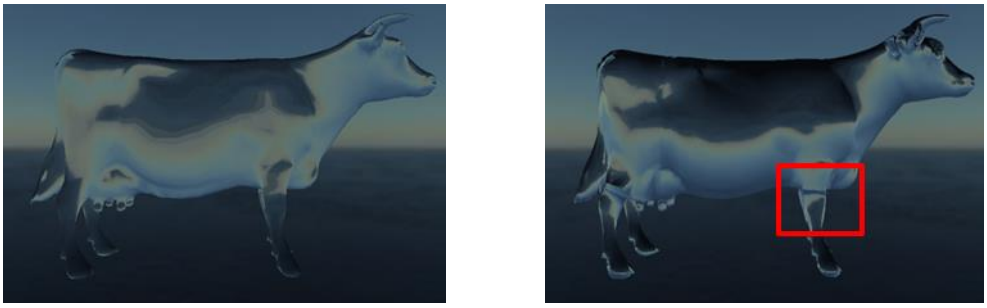


(d)



(e)

圖 4.13: 牛模型(a) 一層折射結果，(b) 兩層折射結果，(c) 四層折射分析，紅色:兩層，藍色:四層，綠色:全反射，(d) 四層折射結果，(e) ray tracing 的結果



(a) (b)

(c) (d)

圖 4.14: 牛模型(a) 一層折射結果，(b) 兩層折射結果，(c) 四層折射結果，(d) ray tracing 的結果

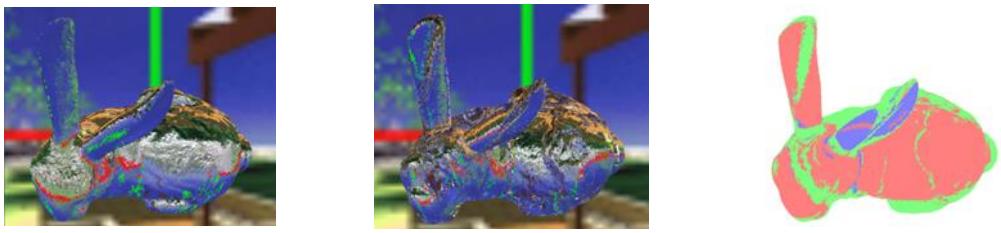
寶特瓶模型特色為表面細節變化明顯(圖 4.15)，經過折射後與光線追蹤法比對，能看出表面的細節與折射變化，並且與其相似。

(a) (b) (c)

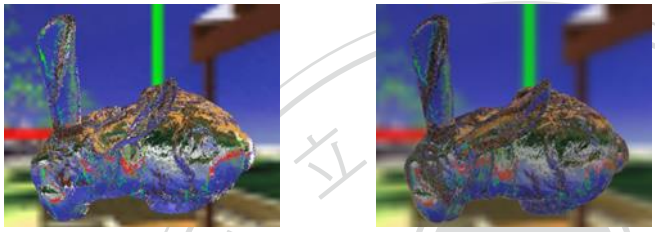
(d) (e)

圖 4.15: bottle 模型 (a) 一層折射結果，(b) 兩層折射結果，(c) 四層折射分析，紅色:兩層，藍色:四層，綠色:全反射，(d) 四層折射結果，(e) ray tracing 的結果

模型更為複雜的兔子(圖 4.16)，則容易在長耳朵的地方產生四次折射，此外由於模型精細度高，表面的凹凸起伏也較為細緻，也因此折射過後的顏色也較為混亂，女王頭模型也是如此(圖 4.17)。

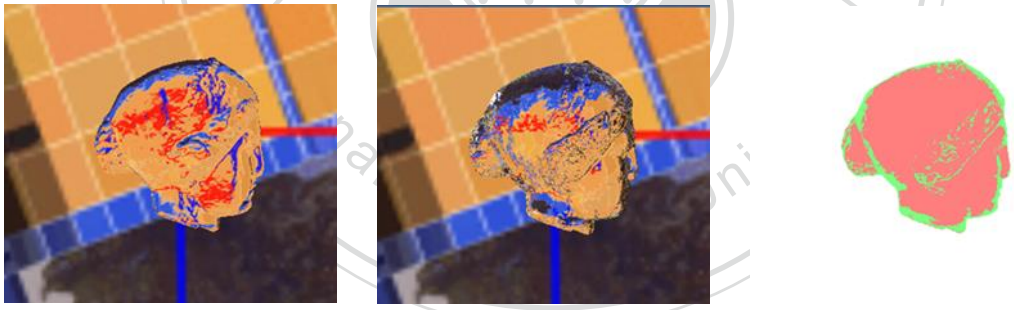


(a) (b) (c)

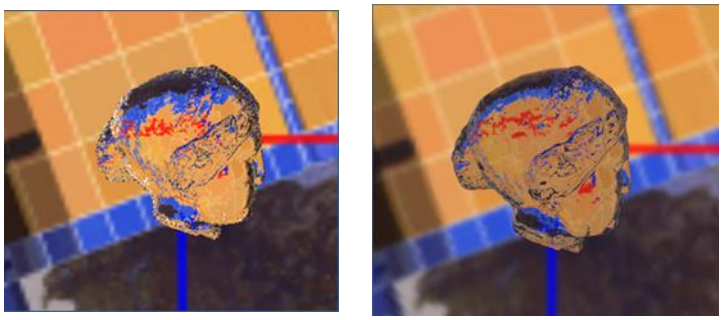


(d) (e)

圖 4.16: 兔子模型 (a) 一層折射結果，(b) 兩層折射結果，(c) 四層折射分析，紅色:兩層，藍色:四層，綠色:全反射，(d) 四層折射結果，(e) ray tracing 的結果



(a) (b) (c)



(d) (e)

圖 4.17: 女王頭模型(a) 一層折射結果，(b) 兩層折射結果，(c) 四層折射分析，紅色:兩層，藍色:四層，綠色:全反射，(d) 四層折射結果，(e) ray tracing 的結果

4.3 多個透明物體

多個透明物體的結果如圖 4.18 所示，我們的由簡單的場景設計開始，發現其折射後的結果近似，更加複雜的場景如圖 4.19,4.20,4.21 所示，觀察結果發現結果雖然並無百分之一百的相同，但其大致上的方向是正確的。相比於[Wyman 2005]的不透明幾何模型，我們更進一步得到多個物體折射的效果。

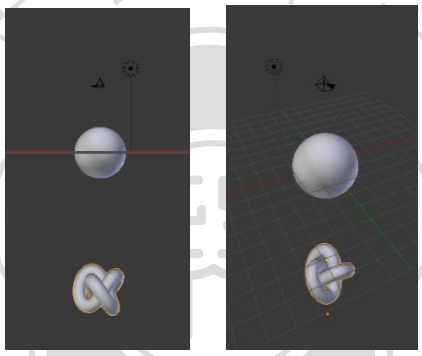
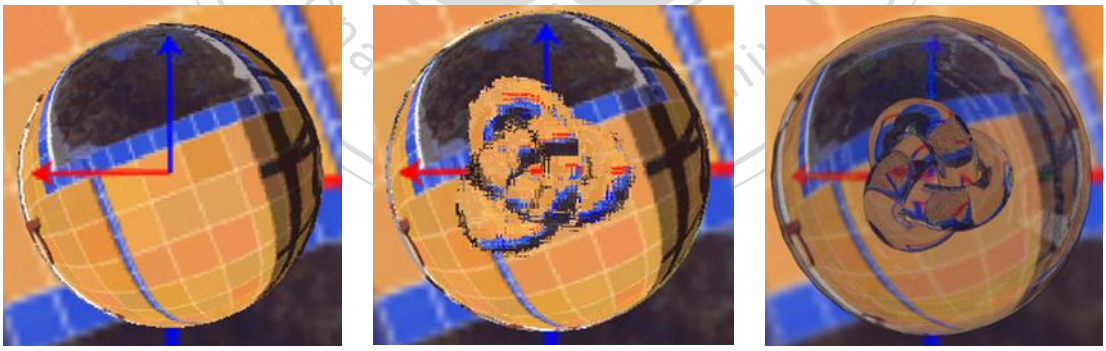
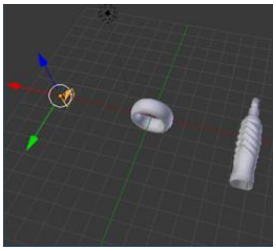


圖 4.18:blender 場景設計 render 結果為圖 4.18 的 a



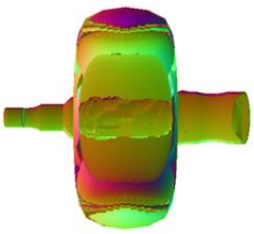
(a) (b) (c)

圖 4.19: 兩個物體折射(a) 圖 4.18 場景的 render (b) 我們方法的 render (c) 只有球 model 的情況



(a)

(b)



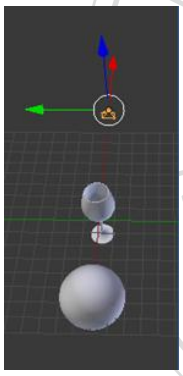
(c)

(d)

(e)

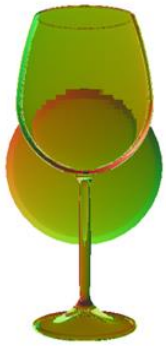
(f)

圖 4.20: 兩個物體折射(a) 場景設計 (b) 單一物體折射向量 (c) 兩個物體折射向量(d) 只由一個物體的結果 (e) 兩個物體的結果(f) ray tracing 的結果



(a)

(b)



(c)

(d)

(e)

(f)

圖 4.21: 兩個物體折射(a) 場景設計 (b) 單一物體折射向量 (c) 兩個物體折射向量(d) 只由一個物體的結果 (e) 兩個物體的結果(f) ray tracing 的結果

4.4 效能比較與複雜度分析

我們以 OptiX 來作為我們的對照(表 4.21)，我們可以發現我們的方法在簡單的模型(如球，立方體)上明顯優於 OptiX 的效能，複雜的模型(如兔子，維納斯頭像)在較遠的視點位置之下我們方法與 OptiX 的效能是差不多的，然而 OptiX 折射模型的畫面比例越多時(視點離物體越接近)fps 也越低，我們的方法則沒有此問題，如表 4.21，我們分別以不同視點 1.與物體距離 20 單位(物體佔畫面比例小), 2. 與物體距離 10 單位(物體佔畫面比例大)，與我們方法比較。

	球	牛	甜甜圈	無限環形	兔子
faces	512	5804	576	2400	69664
我們的方法	153 fps	126 fps	142 fps	133 fps	15 fps
OptiX 視點位置距離 20	112 fps	44 fps	92 fps	79 fps	12 fps
OptiX 視點位置距離 10	56 fps	23 fps	49 fps	34 fps	9 fps

表 4.1: 解析度 512*512

我們分析時間複雜度，假設我們每次搜尋交點做 n 次線性搜尋(線性搜尋每一步的深度為 1/n)與 m 次二元搜尋，因此我們每次搜尋交點的花費為 m+n，最佳的情況為兩次搜尋便得到最後的折射向量，表示我們只需要花一次 m+n 的搜尋，最差的情況即為四層搜尋加上一次全反射的計算，要花(3+1)(m+n)的時間複雜度。

4.5 限制

我們以 image space 的方式實作了多層的折射，雖然效能上優於光線追蹤法，然而當折射的層數提升，因為資料是以貼圖的方式傳遞，而受限於貼圖的解析度，我們方法的解析度也因而下降，也容易在斷層與全反射處產生錯誤的結果，如圖 4.22。

雖然一個場景中並不會有太多的折射模型，但隨著越多折射模型的或者太過複雜的模型加入都會造成效率的降低，此外，我們目前因效能的考慮而對每一個模型都只做四層的折射，因此會有無法正確顯示的部分。

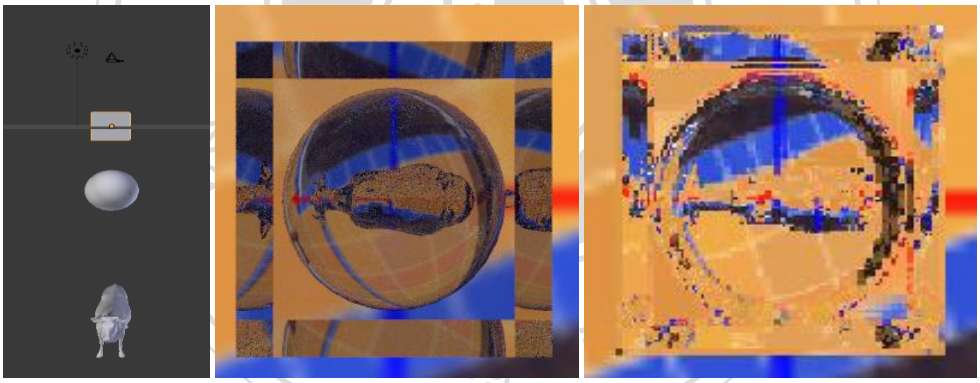
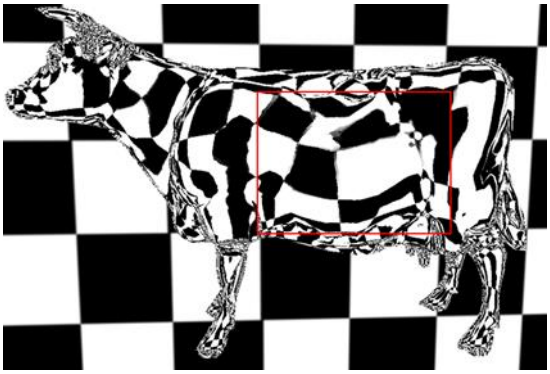


圖 4.22: (a) 場景設計 (b) blender render (c) 我們的方法

為了提升解析度的問題，我們以 Procedural texture 的技術繪製環境貼圖，我們發現隨著環境貼圖的解析度提升，其結果中許多的雜訊會消失，換言之越接近 Procedural texture 的結果，如圖 4.23 中紅色圈起的部分。



(a)

(a)

(b)

(c)

(d)

圖 4.23: (a) 環境貼圖解析度 256*256 (b) 環境貼圖解析度 512*512 (c) 環境貼圖解析度 1024*1024 (d) 環境貼圖以 Procedural texture 繪製

第五章

結論與未來發展

5.1 結論

本論文提出了一個用以實現即時多層折射，比起兩層折射，我們的方法更加接近光線追蹤法的結果，尤其在於可以在透明模型的表面較清楚的看見光學折射下物體的形狀，這是兩次折射的結果針對需要四次以上折射模型所無法做到的。

在整個研究中，我們先分析了折射的性質與模型的特性，發現模型的折射次數可以運用 depth peeling 分解的深度來運算，因此我們採用了 Depth peeling 的方法將模型分成四層，使得有別於光線追蹤的方法，在計算折射模型時不需在光線與物體表面不斷的遞迴，因而相較於光線追蹤的方法來的有效率，此外影像空間的貼圖處理也有助於我們紀錄於傳送向量與位置的資訊，使得折射計算更於有效率，本論文也提出了斷層搜尋的方法使得我們更加正確的找到模型內的交點，也因此我們解決了兩次折射無法正確的顯示整個物體內的形狀。

我們也提出了多個物體的折射方法，依造視點與物體的遠近，我們依序進行四層的折射，並且分析物體之間折射的可能性，得到正確的折射向量。

整體而言，我們在效率相較於[Manuel 2007]沒有減少太多的前提下增加了兩次折射的方法的準確度，使得我們的模型在透明的狀態下可以完整地獲知整個物體的資訊，此外我們也提供了多個透明物體間的折射方法。

5.2 未來發展

目前我們的方法尚在 image space 中和環境貼圖還有解析度的問題，在外來工作中，我們首先希望能提高我們方法的解析度，或許可以運用 mip-map 技術來提高貼圖解析度的不足，然而我們需要以折射深度來決定貼圖的解析度(mip-map 的層級)，這個問題的困難點在於貼圖的分層無法完全對應的模型每個點的折射深度，我們或許可以考慮在折射後再進行貼圖的計算。

考慮對四層以後的折射進行，目前我們只實做了四層的深度折射，然而並不是所有的物體都只會運行四次的折射計算，如前面提到了龍的模型，就可能需要六層以上的折射計算，但是由於光的直線性，越前面的層次需要考慮越多後面的層次，舉例來說，在計算第二層的折射，因為第四層的折射某些部分是與第二層重疊的，因此我們必須在第二層折射計算加入第四層的深度，一但層次到四層以上，光是分層時的判斷就會是一向繁複並且難以理解的工作，也因此每多加一層就會增加計算的時間和複雜的層度會是最大的困難點，怎麼樣保持時間複雜度與品質的平衡是這個問題的核心。

效率上我們可以考慮除了現有的搜尋方法之外，是否還有更佳的搜尋方法，畢竟越多的層次或者越多的物體，需要搜尋的次數就會相對提升，我們也可以考慮樣許多步驟一起執行，例如折射的計算有無可能與 depth peeling 同步處理，又或者多個物體的計算，以多層的場景來處理。

未來可以再往多層全反射(如閃耀的鑽石)方向研究，鑽石為何會有如此耀眼的光芒?原因就在於:較高的折射率與特殊加工後的表面，高折射率容易產生全反射，在特殊切割的表面折射，使得光線經過多次折射之後集中在某些點，因而鑽石閃閃發亮，運用在我們的方法上，我們可以多次的全反射，然而起反射向量射向 Depth peeling 切割後的哪一層是需要判定的，因此未來我們可以著手於此，設計出閃亮的鑽石或者折射的藝術作品。

參考文獻

[1] Fabio Policarpo, Manuel M. Oliveira, and João L. D. Comba L. D. Comba. 2005. real-time relief mapping on arbitrary polygonal surfaces. In Proceedings of the 2005 symposium on Interactive 3D graphics and games (I3D '05). ACM, New York, NY, USA, 155-162.

[2] Chris Wyman. 2005. An approximate image-space approach for interactive refraction. ACM Trans. Graph. 24, 3 (July 2005), 1050-1053.

[3] Chris Wyman. 2005. Interactive image-space refraction of nearby geometry. In Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia (GRAPHITE '05). ACM, New York, NY, USA, 205-211.

[4] Manuel M. Oliveira and Maicon Brauwers. 2007. real-time refraction through deformable objects. In Proceedings of the 2007 symposium on Interactive 3D graphics and games (I3D '07). ACM, New York, NY, USA, 89-96.

[5] Fernando, R. & Kilgard M. J. (2003). The CG Tutorial: The Definitive Guide to Programmable real-time Graphics. (1st ed.). Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA. Chapter 7: Environment Mapping Techniques

[6] Stephane Guy and Cyril Soler. 2004. Graphics gems revisited: fast and physically-based rendering of gemstones. In ACM SIGGRAPH 2004 Papers (SIGGRAPH '04), Joe Marks (Ed.). ACM, New York, NY, USA

[7] Arvo, J. (1986). "Backward Ray Tracing." Developments in Ray Tracing. ACM Siggraph Course Notes 12, pp. 259–263, 1986.

[8] Génevaux, O., F. Larue, et al. (2006). Interactive refraction on complex static geometry using spherical harmonics. Proceedings of the 2006 symposium on Interactive 3D graphics and games. Redwood City, California, ACM: 145-152.

[9] Günther, J., I. Wald, et al. (2004). Realtime caustics using distributed photon mapping. Proceedings of the Fifteenth Eurographics conference on Rendering Techniques. Norrköping, Sweden, Eurographics Association: 111-121.

[10] Guy, S. and C. Soler (2004). "Graphics gems revisited: fast and physically-based rendering of gemstones." ACM Trans. Graph. 23(3): 231-238.

[11] Hakura, Z. S. and J. M. Snyder (2001). Realistic Reflections and Refractions on Graphics Hardware with Hybrid Rendering and Layered Environment Maps. Proceedings of the 12th Eurographics Workshop on Rendering Techniques, Springer-Verlag: 289-300.

[12] Charles de Rousiers, Adrien Bousseau, Kartic Subr, Nicolas Holzschuch, and Ravi Ramamoorthi. 2011. real-time rough refraction. In *Symposium on Interactive 3D Graphics and Games* (I3D '11). ACM, New York, NY, USA.

[13] Arthur Appel. 1968. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30--May 2, 1968, spring joint computer conference* (AFIPS '68 (Spring)). ACM, New York, NY, USA, 37-45.

[14] Abraham Mammen. Transparency and antialiasing algorithms Implemented with the virtual pixel maps technique. IEEE Computer Graphics and Applications, 9(4): 43-55, July 1989

[15] Everitt C.: Interactive order-independent transparency. Tech. rep., NVIDIA Corporation, 2001.

[16] Liu B.-Q., Wei L.-Y., Xu Y.-Q.: Multi-layer depth peeling via fragment sort. Tech. rep., Microsoft Research Asia, 2006. 2.

[17] Bavoil L., Myers K.: Order independent transparency with dual depth peeling. NVIDIA OpenGL SDK, 2008.

[18] Bavoil L., Myers K.: Order independent transparency with dual depth peeling. NVIDIA OpenGL SDK, 2008.

[19] Y. Mukaigawa, Y. Yagi, and R. Raskar. Analysis of light transport in scattering media. In Proc. CVPR, pages 153-160, 2010

[20] Steven M. Seitz , Yasuyuki Matsushita , Kiriakos N. Kutulakos, A Theory of Inverse Light Transport, Proceedings of the Tenth IEEE International Conference on Computer Vision, p.1440-1447, October 17-20, 2005

[21] Aner Ben-Artzi , Kevin Egan , Frédo Durand , Ravi Ramamoorthi, A precomputed polynomial representation for interactive BRDF editing with global illumination, ACM Transactions on Graphics (TOG), v.27 n.2, p.1-13, April 2008