國立暨南國際大學資訊工程學系

碩士論文

應用 WebGL 實作 Ray Casting 之研究

An Implementation of Ray Casting based on WebGL

指導教授： 陳履恆 博士

研究生： 徐國峰

中華民國 103 年 8 月

國立暨南國際大學碩士論文考試審定書

資訊工程 學系（研究所）

研究生 徐國峰 所提之論文

應用 WebGL 實作 Ray Casting 之研究
An Implementation of Ray Casting based on WebGL

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

委員兼召集人

委員

委員

中華民國 103 年 6 月 24 日

致謝

家人永遠是我最重要的後盾，雖然你們無法給我研究上實質的幫助，但你們卻是我心裡最強大的支柱。感謝我的父母讓我能無後顧之憂的完成學業，沒有你們無條件的背後支持，我的學生生涯不會如此順遂，也謝謝你們體諒聚少離多的相處陪伴，你們的恩情今世難報。也謝謝老妹的時相鼓勵與小妹妞妞的陪伴。

能完成研究取得碩士學位，最要感謝的是我的指導教授，陳履恆博士，謝謝老師給我極大的包容與空間讓我選擇自己的興趣來發揮，多了您的提點與鼓勵，使我更能堅持到最後，對於老師永遠有源源不斷的想法可以激發引領我們，這點令我感到相當的佩服。同時感謝口試委員吳曉光教授及蔡孟峰教授的寶貴指教與建議，你們讓我學習到如何有效展現與表達自己研究成果的方法。

感謝 CGDA LAB 的成員們，承蒙學長姊的照顧和同學們的伴隨，還有一群優秀學弟妹們的相扶持，我從這裡得到很多幫助，謝謝你們。唯深感遺憾的是因自己能力不足而沒能盡到帶領實驗室及學弟妹的責任。很高興能認識你們，最後一起走出暨大的校門，祝你們未來有很棒的前程跟發展。

話說在我還沒正式進入暨大前，就已經在認識的學長引薦下進入了暨大運動舞蹈校隊 XD，在校隊和社團裡，真的是有很多酸甜苦辣的點點滴滴，何其有幸的在此碰到諸位愛舞同好，一起練舞比國際賽，一起揮汗成長，一起遊玩同樂，我們一起度過很多美好的時光，謝謝你們。尤其是生日時，社團夥伴更是每每讓我感到十分的窩心與溫暖。而這段校隊經歷也讓我碰到一生的摯友兼舞伴，謝謝妳。期待未來一同攜手征戰，拚出一番成績，圓我們此生最美的夢。順帶一提，其實最早曾跟指導老師有討論過個人興趣搭配電腦圖學領域做一套舞蹈學習系統的計畫，但無奈執行難度過高，大幅降低了可行性。畢竟舞蹈不是固定僵化的技藝，它極富音樂性且具有豐富的肢體動作跟空間移動性，舞序的構思和創意也是靈活的，是一個既有標準規範而自由度卻

又相當高的雙人組競技運動。也因此若要建構一個輔助學習系統，那必是個挑戰性極高的跨領域任務，只可惜現階段我能力尚不足以做到。

在研究所期間，很幸運地在校園旁聽的課中認識一位前輩，Ubuntu 台灣社群負責人，BlueT，感謝你帶領我們進入 Ubuntu 的世界，更讓我下定決心在我研究所期間開始直接以 Ubuntu 做為日常生活中主要的桌面作業系統。

在這三年間碰到的所有師長、歷屆室友、實驗室同學、國標社大家族、一起習舞的外校夥伴、你們都是我的貴人。由衷的感謝一路走來關心我並支持我的人，有你們的相伴，使我的人生更加繽紛多彩，是你們豐富了我的生命，最後我終於走過這個階段了，我抱持著感恩的心，謝謝你們。

在暨大的這些日子裡，有很多特別的回憶，期間接觸了些許新事物，也體悟出一些新思維，我陸續實現了很多以前沒做過或未完成的事，甚至還因緣際會下參與拍攝商業廣告的微電影。由於暨大就座落在風景區環繞的地理位置上，這也讓我在研究之餘，走訪了許多名勝景點，姑且不論校外，暨大校園本身就是個因時變換的美麗桌布，一次偶然照下的相片就讓我無心插柳拿到科院盃攝影比賽的次獎，生命彷彿就是這樣一連串意外形塑的，自然也包括我到暨大念書的種種歷程。起初，在我服完兵役後隻身來到這個完全人生地不熟的埔里時，確實常感到有些孤獨無助，還好一路走來有很多朋友的相助與陪伴，讓我順利度過低潮難關。這趟研究之路雖然走的不算順，幸好在摸索許久後，尋得自己想做的方向與目標，研究工作終究隨之完成了。接下來更重要的是，我將離開校園展開新的旅程，邁向未知的挑戰。沒有最好，只有更好！這是我秉持的信念，期許自己面對未來的一切，要不斷的精進發展。

論文名稱：應用 WebGL 實作 Ray Casting 之研究

校院系：國立暨南國際大學科技學院資訊工程學系　　頁數：30

畢業時間：103 年 8 月　　學位別：碩士

研究生：徐國峰　　指導教授：陳履恆博士

中文摘要

現今的個人電腦設備雖然已經能執行很多複雜的運算程式，但在後 PC 時代，行動裝置的蓬勃發展下，電腦的影響力已經被爆發性成長的智慧型手機和平板電腦所蓋過，尤其體現在我們今日的日常生活中。顯然能在移動裝置上順利執行任務的重要性已經不言可喻了。

近年來，由於各家瀏覽器的激烈競爭，興起一連串的網路技術改革，也因此推動了新一代的網頁技術規範，於是催生出所謂 HTML5 相關的 Web 技術，隨之而來的發展，出現了 WebGL，從此突破以往必須額外加裝外掛程式來輔助瀏覽器呈現 3D 繪圖的限制。

光線追蹤的過程是一高計算成本用來產生高質量 3D 電腦圖學成像的方法。而電腦圖學的照明效果就是通過光線追蹤的演算法，利用模擬光跟物體之間的相互作用而使得照明效果更真實。本研究，我們利用 WebGL 實作此電腦圖學理論呈現在不同的平台上。

關鍵字：HTML5, WebGL, Ray Casting

Title of Thesis：An Implementation of Ray Casting based on WebGL

Name of Institute：Department of Computer Science and Information Engineering, College of Science and Technology, National Chi Nan University Pages：30

Graduation Time：08/2014 Degree Conferred：Master

Student Name：Kuo-Feng Hsu Advisor Name：Dr. Lieu-Hen Chen

Abstract

Though today's personal computer equipment has been able to carry out many complicated computation programs, but in the post-PC era, mobile devices began to flourish. After the explosive growth of mobile device, the influence of the PCs has become less than smartphones and tablets, especially on our daily life. Obviously, the importance of successful running programs on mobile device is needless to say.

In recent years, due to the fierce competition in the browser, the rise of a series of Internet technology reform, thus promotes the new generation of web technology specification, such as HTML5 and WebGL and so on. By the new technological breakthroughs, nowadays browser can render 3D graphics without installing additional plug-ins for assistance.

The ray tracing process is a computation expensive method used to produce high-quality 3D computer graphics image. The illuminating effects of computer graphics are more realistic by adopting ray tracing techniques, which simulates the interactions between lights and objects. In this paper, we use WebGL implement this computer graphics theory presented on different platforms.

Key words: HTML5, WebGL, Ray Casting

目次

圖次

第一章　緒論

1.1　研究背景與動機

過去我們只能單純地在特定的裝置平台（通常是安裝某種作業系統的個人電腦）上撰寫和執行程式，但問題是在某一作業系統上撰寫的原生程式，實在難以在多個平台流通並執行展示。特別是行動裝置蓬勃發展的今日，在行動裝置上也能順利執行任務，並且呈現出同樣程式的效果，這顯然是件有意義且重要的事。而傳統電腦圖學的實作，絕非只能侷限在某些裝置或平台上，應嘗試突破與更多媒介協作，以產生更多應用的可能性。

由於各個作業系統環境的程式仍無法輕易移植或跨平台執行，但現在則可透過 Web 介面來達成，因為新一代瀏覽器可以支援原生三維繪圖物件，故以瀏覽器當窗口來呈現跨平台的多媒體內容會是一個極佳的解決方案。我們更可選擇新一代 Web 技術（HTML5）來取代過去必須使用諸如 Flash 或 Java Applet 之類的技術所製作的網頁繪圖動畫，而舊有技術最大的缺點便是必須在瀏覽器額外安裝第三方外掛程式才能運作，如今這問題可藉此獲得相當良好的改善，免除使用者的麻煩手續，甚至可以因此提升系統安全性。

我們知道 HTML5 中的 <canvas> 標籤正是提供網頁瀏覽器一個絕佳的圖形容器，只要透過程式定義繪製方法後交給 canvas 這個元素，便可由網頁瀏覽器顯示出來。我們發現 WebGL 這個 JavaScript API，它允許開發者在瀏覽器中直接嵌入支援硬體加速的互動 3D 圖形，故藉由腳本語言可直接定義繪製管線來操作 GPU 做圖形的計算。此外 WebGL 做為 HTML5 中的 <canvas> 標籤裡的一個特殊的上下文（experimental-webgl），用以實作在瀏覽器中，因此 WebGL 也可以與所有 DOM 介面完全整合到一起。由於 WebGL API 是基於 OpenGL ES 2.0 的版本，換句話說就是

OpenGL ES 2.0 的子集，所以 WebGL 可以運行在許多不同的硬體設備之上，例如桌上型電腦、智慧型手機、平板電腦和智慧電視等。這就是我們決定採用 HTML5 這個新一代的 Web 技術整合電腦圖學領域知識來做應用。

1.2 研究目的

本研究主要目的是利用 HTML5 + WebGL 來建構一個純使用前端或所謂客戶端硬體資源（主要是指 GPU）的 3D 場景渲染，其中包含了電腦圖學中光線模型理論的應用，並配合現代的新型瀏覽器來呈現即時繪圖的動畫影像。此外我們也設計了可以監看畫面更新率（FPS）的方法來評估動畫執行效能。

第二章　相關研究

2.1　HTML5

HTML5 現正處於發展階段，目標是取代 1999 年所制定的 HTML 4.01 和 XHTML 1.0 標準。為了因應網際網路應用日益發展的需求，必須把某些過時的規格汰換並增補一些符合現代網路應用的元素。

一般廣義論及 HTML5 時，其實指的是包括 HTML、CSS 和 JavaScript 等前端技術在內的技術組合。它能夠滿足豐富性網路應用服務（plug-in-based rich internet application，RIA）的功能，並減少瀏覽器對於需要外掛程式，如 Adobe Flash、Microsoft Silverlight，及 Oracle JavaFX 的依賴，同時提供更多能有效增強網路應用的標準集。

2.2　WebGL

WebGL（Web Graphics Library）是一基於 OpenGL ES 2.0 規格的 JavaScript API，提供網頁瀏覽器渲染互動式 3D/2D 畫面的技術。它使用 HTML5 Canvas 元素作為網頁繪圖的圖形容器，並以文件物件模型（DOM, Document Object Model）作為存取介面。它允許開發者在瀏覽器中嵌入支援硬體加速的互動 3D 圖形，改善以往需要外掛程式輔助才能讓瀏覽器呈現 3D 畫面的技術。

由於 WebGL 是根據 OpenGL ES 2.0 之規格設計，而在 OpenGL ES 2.0 之後已經捨棄先前的 fixed function pipeline 架構，改採用 programmable pipeline，所以在實作 WebGL 繪圖程式時，必須使用 GLSL 著色器語言自行編寫 Vertex Shader 和 Fragment Shader。

WebGL 還在實驗階段，已有初步規格釋出，尚在發展當中，故程式語法仍可能

會變更，目前是由非營利組織 Khronos Group 所管理。雖然各家瀏覽器對 WebGL 的實作不一，然而目前看來，WebGL 已在多種最新的瀏覽器中被廣泛支援了，列舉如下：Firefox 4+, Google Chrome 9+, Opera 12+, Safari 5.1+ and Internet Explorer 11+.。

2.3 著色器與著色語言

著色器用途是指一組供計算機圖形資源在執行渲染任務時使用的指令碼，程式設計師會將著色器應用於 GPU 的可程式化的管線中，在 WebGL（OpenGL ES）裡，著色器主要分成兩者，一個 Vertex Shader，另一個是 Fragment Shader。著色器的重要特性是用來同時處理大量的數據，比如螢幕上的一整塊像素群，或者一個模型結構的所有頂點。而平行運算正適合這樣的情況，且當今的 GPU 也被設計用來提升高效率的平行運算。

GLSL ES（OpenGL Shading Language for Embedded Systems），簡言之就是用在著色器的 OpenGL 著色語言，是一個類 C 語言的高階著色語言。它提供開發者對繪圖管線更多的直接控制，而無需使用低階的 ARB 組合語言或硬體規格語言。

2.4 Ray Casting

光線追蹤是電腦圖學領域中最基本的成像理論，是一種來自幾何光學的通用技術，它透過追蹤與物體表面發生交互作用的光線，通過計算一束束光線的反射與折射路徑，來對 3D 模型及場景做最接近真實的著色，只要硬體性能夠好，計算光與物體相交測試的深度夠多，就可以模擬出相對擬真度高的畫面。 Ray Casting 即為光線追蹤的第一層相交測試步驟。

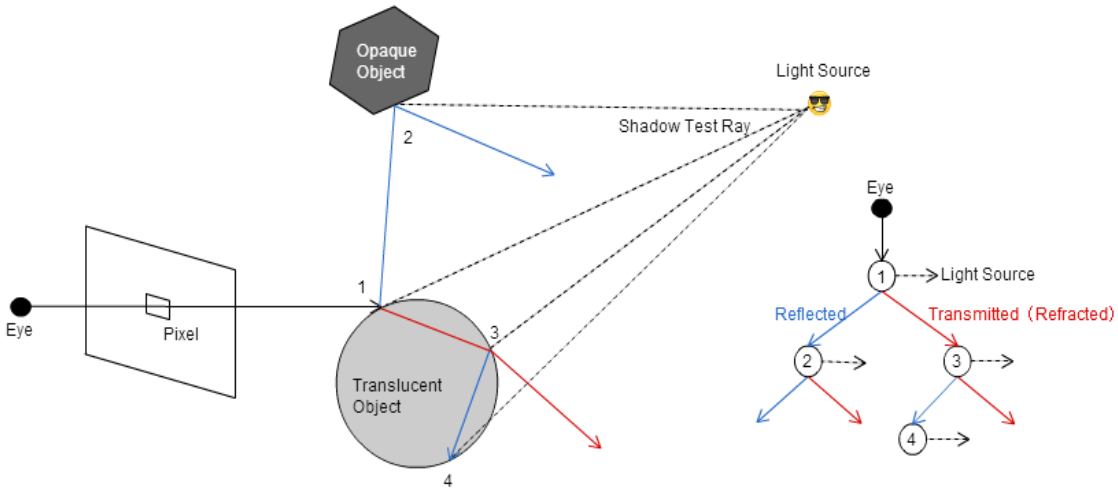

圖 1　光線追蹤總體示意圖

光線的反射向量推導：

$where\ \theta_i = \theta_{r,}\ I, R, N\ on\ the\ same\ plane, \|I\| = \|N\| = \|R\| = 1$

$\because R = (I_{\parallel}) + (-I_{\perp}) = (I - I_{\perp}) + (-I_{\perp}) = I - 2I_{\perp}$

$\because I_{\perp} = \|I\| \cos\theta_i \cdot (-N) = \|I\| \cdot \|-N\| \cos\theta_i \cdot (-N) = [I \cdot (-N)] \cdot (-N) = (I \cdot N) \cdot N$

$\therefore R = I - 2I_{\perp} = I - 2(I \cdot N) \cdot N$

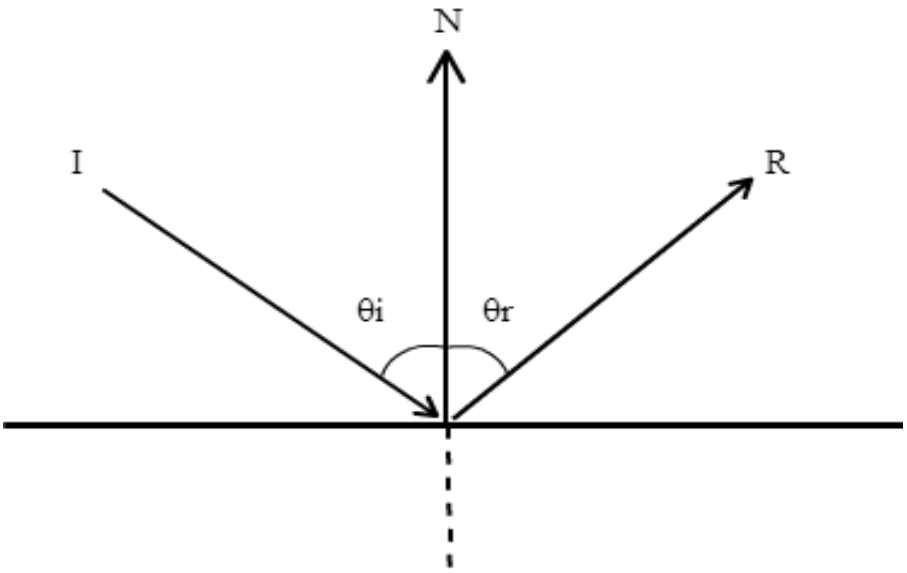

圖 2　反射向量示意圖

2.5 瀏覽器的排版引擎與 JavaScript 引擎

排版引擎又被稱作瀏覽器核心、頁面渲染引擎或樣版引擎，它負責取得網頁的內容，整理訊息，以及計算網頁的顯示方式，然後輸出至顯示器。而所有需要根據表示性的標記語言（Presentational markup）來顯示內容的應用程式都需要排版引擎。現時主要的四大排版引擎分別為：Gecko（用在 Mozilla Firefox）； WebKit（用在 Google Chrome 和 Apple Safari）；Presto（用在 Opera）；Trident（用在 Internet Explorer）。

JavaScript 引擎是一個獨立但內嵌在排版引擎裡面配合運作的程式，用來專門處理 JavaScript 的指令碼，可被嵌入於許多不同應用的環境，特別是網頁瀏覽器，幾乎被廣泛地納入使用。而每款引擎的特色與實作都不盡相同，最早是純粹用來為腳本語言解釋執行的直譯器，後來有些引擎開始採用即時編譯成位元碼的方式，又例如 V8 引擎甚至為了提升效能而將程式碼編譯成機器碼來執行。現時主要瀏覽器的 JavaScript 引擎分別為：Mozilla Firefox 的 Monkey 系列；Google Chrome 有著名的 V8；Opera 用的是 Carakan；Safari 採 Nitro ；IE 則是 Chakra。

在進行網頁程式開發時必須要特別留意不同瀏覽器之間的差異，畫面效果將因不同的排版引擎及 JavaScript 引擎而有所不同，必要時得針對個別差異進行調校。

第三章　系統架構與實作

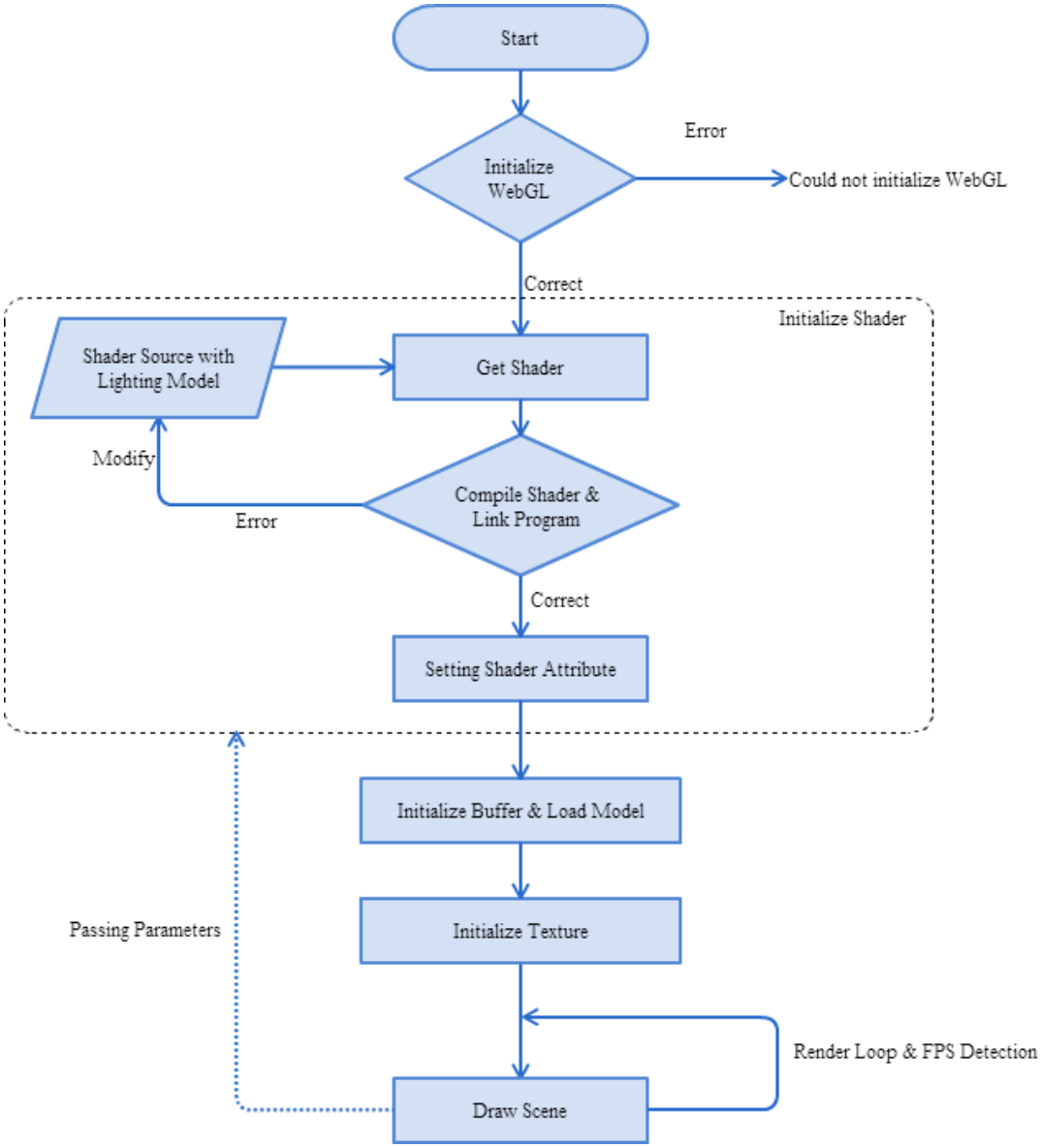

圖 3　系統架構流程圖

3.1 系統建置實作

3.1.1 網頁架構基礎佈建

首先我們需要先佈建好一個基本的網頁結構，完成 <head> 及 <body> 兩大部份的規劃。<head> 裡面要做最重要的設置，包含 meta data （內容形態、文字編碼、語言區域等的一些解釋文件的資料）、標題、層疊樣式表（CSS，Cascading Style Sheets）及最重要的腳本語言區塊（其中又包含外部程式的引入及本體程式區塊），然而本系統有三種 script type 分別是 text/javascript、x-shader/x-vertex 和 x-shader/x-fragment，後兩者為著色器。<body> 則是網頁版面的規畫，也是一個網頁畫面最直接的內容呈現，更是人機互動操作介面相關的設計所在點之一，首先我們選擇在頁面載入時設定一個程式進入點，然後做些該頁面相關的文字說明區塊，以及預埋幾個 區塊做為程式執行時動態顯示資料的區塊，另外還在頁面上做了使用者自訂參數的區塊，有表格輸入區塊、有下拉式選單以及多選核取方塊，用來讓程式做即時的效果轉換設置。本研究不著墨用來結構化網頁文檔及決定網頁內容外觀顯示效果的層疊樣式表。

3.1.2 主程式的建構

開始進入程式流程階段，一開始要從 <body> 取得 HTML5 畫布元素之識別碼（Canvas ID），之後將再藉由 Canvas ID，進一步獲取一個標準的上下文名稱（目前仍是用一個實驗階段的 WebGL 識別程式碼，未來將可能正式全面啟用一個特定的名稱）來建立 WebGL 渲染內容物件，如果沒有錯誤就可以開始使用 WebGL 物件來做接下來的一系列工作，例如設定出一個自定義的空白畫布，至此即完成了 WebGL 的初始化。

再來我們做一個瀏覽器相關資訊的偵測，之後會在頁面上顯示一個明細，例如開啟這個網頁程式是使用甚麼瀏覽器，其名稱版本訊息及瀏覽器所處平台為何（也就

是代表瀏覽器所處的作業系統環境，此功能的重要性在於可用來區別是哪個裝置啟動了這個程式系統，例如行動式裝置與個人電腦很輕易的就可以由此判斷)。

接著我們使用 getShader 函數來獲取兩個區塊的程式碼，分別是 Fragment Shader 和 Vertex Shader，透過 DOM 架構將預先撰寫的兩個著色器區塊程式碼讀出，然後把它們同時附加到一個叫做「program」的 WebGL 物件裡，用以創建著色器物件並編譯之，隨後把編譯好的著色器用來創造程式物件，然後設定著色器的程式物件的頂點屬性（包括位置與顏色的陣列位址)，再透過 WebGL 函數去啟用頂點的位置跟顏色屬性，在最後初始化著色器階段結束前再將著色器的程式物件設定位於著色器程式碼區塊的 Uniform 變數的位置資訊屬性。

接下來我們做初始化緩衝區的步驟，以下針對某個多邊形物體建立一個存放頂點陣列的緩衝區，並設定好該物件的各頂點座標位置和顏色資訊，然後將緩衝區繫結至著色器程式，此舉是告訴 WebGL 讓這個緩衝區成為使用中，再把 JavaScript 的浮點數陣列型別轉成可傳遞給 WebGL 處理的形式。

進入最終階段，繪製場景，此時將需要明白指定好欲繪製的畫布尺寸（即視口，viewport)，等同於一個三維空間的投影平面，我們建立一個用於觀測場景的透視圖。在預設情況下，WebGL 會把近處的物體和遠處的物體用同樣的尺寸繪製（這在電腦圖學中被稱作「正交投影」)。為了產生遠近感，遠處物體看起來比近處物體小，我們需要明確指定使用透視方法。對於這個場景，我們指派垂直視野為 45°、畫布的寬高比以及從我們的視點看到的最近距離是 0.1 個單位，最遠距離是 800 個單位。給定相關參數後便可建立投影矩陣，再來建造模型視圖矩陣來指定物體將要繪製何處，一樣要將現在欲處理的物體之頂點陣列緩衝區綁定為當前處理狀態，接著呼叫 WebGL 函數來接收該物體的相關參數，最後把模型視圖矩陣和投影矩陣操作推送到 GPU 來執行，同時使用 WebGL 繪製陣列方法把物體繪出於三維場景中。

3.1.3 3D 物件的運動

在 WebGL 中的 3D 場景裡移動物體的原理非常單純，就是重複的繪製物體，即在每個時刻繪製不同運動狀態下的物體。這也就代表著，我們一直用來繪製場景的函數（drawScene、animate、tick）就需要被重複呼叫，每個時間點每次繪製的內容都有些許差異。tick 函數用於更新場景的運動狀態（比如控制場中物體的各軸旋轉角度，依時間改變其變數值）、繪製場景以及調整好參數為下次呼叫做準備。另外我們得留心一個問題，即瀏覽器在需要重繪 WebGL 場景時呼叫時，比如說刷新顯示，這種每隔固定微秒就重新呼叫一次的工作，得由瀏覽器提供 API 支援，但各個瀏覽器都對需要循環繪製場景以實現動畫效果的任務提供了名稱皆不相同的函數，例如在 Firefox 中，這個函數的名字是 mozRequestAnimationFrame，但在 Google 瀏覽器和 Apple 瀏覽器中，它又是不一樣的名稱 webkitRequestAnimationFrame。或許在未來，這些瀏覽器廠商會在 API 函數名稱上達成一致共識。但現在解決之道便是自行撰寫兼容程式碼來解決，或者也可利用第三方函式庫如 Oak3D 提供的 okRequestAnimationFrame 函數來方便我們一次搞定所有瀏覽器。總而言之，就是每當瀏覽器需要繪製一幀圖時，就會呼叫一次 tick 函數，然後進行繪製，然後更新相關的參數來為下次呼叫做準備。

3.1.4 物件紋理

紋理貼圖的方法其實就如同設定 3D 物體中某個點的顏色一樣的做法，由於顏色是由 fragment shader 指定的，所以我們需要載入圖片然後將它傳送到 fragment shader。另外，fragment shader 也需要知道當前像素應當使用紋理的哪一個部分，所以我們也需要把紋理的使用位置訊息， 也就是紋理座標， 傳給 fragment shader。

我們使用 gl.createTexture 來建立了一個紋理物件，然後我們建立了一個 JavaScript Image Object，並把它放到我們給紋理物件添加的一個新屬性中（利用 JavaScript 語言優勢，可以給任何物件自由新增任何屬性的能力）。在圖片文檔載入到圖片物件前，

我們先放一個函數來處理當它在圖片被完全載入後，將隨即被呼叫。最後，我們設置好圖片的 src 屬性。完成後，圖片將被異步加載。

由於一般數學座標系與大部份計算機圖形系統之座標系不同，故所有被載入紋理的圖片都需要做一個垂直翻轉。然後下一步使用 texImage2D 函數的方法將圖片中的所有資訊傳到 GPU 的 Texture Space 中，函數的參數分別是，圖片類型、細節層次、通道的大小（也就是用於儲存 R、G、B 值的資料類型）、最後是圖片本身。再來則需指定紋理的特殊縮放參數。接著我們建立包含紋理座標的陣列物件，然後將它綁定合適的屬性，以便著色器可以呼叫它，如此一來，WebGL 就知道各個頂點該使用紋理的哪個部分了。然而 WebGL 可以呼叫 gl.drawElements 這種函數，處理最多共 32 個紋理對象（WebGL 的規格定義），從 TEXTURE0 到 TEXTURE31，我們將剛才載入的紋理指定為 0 號紋理，我們將 0 這個值推送到著色器的 uniform 變數中，以通知著色器我們要使用 0 號紋理。這和單純在 vertex shader 填充顏色的作法非常相似，我們所做的就是將紋理座標設置為頂點屬性，然後以 Varying 變數的形式直接從 vertex shader 中傳出。當每個頂點都設定完畢後，WebGL 會將頂點與頂點之間的像素進行線性內插。然後在 fragment shader 中獲取到了線性內插後的紋理座標，並且以 sampler2D 型態儲存在變數中，它在著色器中代表紋理。在 drawScene 函數中，我們的紋理與 gl.TEXTURE0 綁定在一起，而 uniform 變數 uSampler 的值是 0，所以這個 sampler2D 變數代表的正是我們的紋理。著色器所做的就是 呼叫 texture2D 函數，並根據座標從紋理中獲得相對應的顏色。最後，在 fragment shader 拿到顏色之後，我們就成功地在畫布上繪製出了一個帶有紋理貼圖的物體。

3.1.5 3D 物件中的光線

在這個系統中，我們的目的是在三維場景內模擬光源。這些光源並不用顯示出來，也就是無需是可見的（在模擬情況中，可見情形會影響我們觀察），但它們發出的光必須能夠打在整個三維空間理，並作用於 3D 物體的表面，讓物體面對光照的那一面

是光亮的，同時背向光源的部分將顯得陰暗。更深入的說，我們要做的事就是對 vertex shader 撰寫程式碼來處理光照。我們需要計算出光照對每個頂點及每個像素的影響，並調整物體的顏色。

在電腦圖學中我們將光的類型按照光與物體表面的作用進行區分：一種是從特定方向特定角度平行射入並只會照亮面對入射方向的物體，我們稱之為平行光。另一種光是來自四面八方所有方向並且會照亮所有物體，不管這些物體是否面相或背向光源，我們稱之為環境光。環境光在真實世界裡，其實就是光線照到其他物體，然後間接經由反射光照到的情況，中間的媒介可能是氣塵、牆壁、地板等等。在此，我們把它單獨作為一個光照模型列出來。當光照到物體表面，會發生兩種情況：一、漫反射（Diffuse）：無論光的入射角度如何，都會向所有方向發生反射。反射光的亮度只和光線的入射角度有關，與觀察角度無關。也就是光線愈平行於物體表面，則反射光愈弱，表面愈暗；光線愈垂直於表面，反射光愈強，表面愈亮。計算方法為：倒轉入射光向量與法線內積。二、鏡面反射（Specular）：鏡面反射通常會造成物體表面上的「閃爍」和「反光」現象。鏡射光將按照和入射角相同的角度反射出來。我們能否看到物體鏡射光，取決於眼睛和光反射的方向是否在同一直線上，也就是說，反射光的亮度不僅與光線的入射角有關，也跟視線和物體表面之間的觀察角度有關。鏡面反射的強度也與物體的材質有關，無光澤的木材很少會有鏡面反射發生，而高光澤的金屬則會有大量鏡面反射。計算方法為：$(R_m \cdot V)^{\alpha}$，R_m 是鏡射光方向的單位向量，V 是觀察方向（視線）的單位向量，α 是一個描述光澤度的常數，常數的值越大，光澤度越高。

我們實作主要用到馮氏反射模型，一個物體呈現完整的光反射，會包含以下三種光線：環境光 + 漫射光 + 鏡射光，除了採用基本的馮氏反射模型外，實作時還會考量材質屬性，例如反光度或說光澤度，它決定物體光反射的細節。對於場景中的每一點，它的顏色都是由照射光的顏色、材質本身的顏色和光照效果混合起來的。由於環境光本來就是自然的一部分，而不是特定的某束光線，但我們需要找到一種方法來產生整個場景中的環境光，為了計算簡便，我們使用最簡單且具有效益的方法，就是

為每個場景設定一個環境光等級。著色器需要做的就是分別計算出在環境光、漫射光和鏡射光下每個頂點的 RGB 值，然後加總起來，最後輸出結果。

我們系統場景中的光線設定都來自於同一個固定的方向，也就是平行光，而且這個方向對於場中物體每個頂點都不會改變。於是我們把它放到 uniform 變數中，然後提供給著色器來呼叫。由於頂點上的光照情況取決於光線的入射角，所以我們需要知道一個可以代表物體表面朝向的法向量。我們可以利用建模軟體所產生的三維幾何模型，知道指定頂點所在表面的法線向量。於是我們可以利用內積函數，利用反射光公式來完成反射光的計算，如第二章中的公式推導說明，本來是有三角函數的問題，但在向量的推演下，我們可以巧妙避開了處理角度問題，更減少了運算複雜度。

3.2 FPS 偵測演算法

FPS（Frame Per Second），中文稱「畫面更新率」或簡稱「影格率」，就是說每秒可以顯示幾張影像，這是我們可以用相對直覺客觀的角度來評估繪圖效能的方法，我們參考相關論述，得知目前只有火狐瀏覽器有特別設計一個用來提供繪圖渲染的計數器（window.mozPaintCount）。在此我們共設計實作了三種不同的 FPS 演算法。

3.2.1 FPS Algorithm 1

使用專用在 Mozilla Firefox 的繪圖渲染計數器方法，由於 window.mozPaintCount 的值是隨繪圖次數不斷累加的，它會忠實紀錄下每一次頁面重繪的次數，所以我們只要定時每一秒去抓取當下的繪圖次數再減去上一次的繪圖數據即可知影格率。這個方法極為方便有效，也是諸君公認最準確的作法，只是此方法僅能跑在火狐瀏覽器。而此法的 FPS 值會是整數的。程式執行數據穩定，不會有忽高忽低的現象。唯一要挑剔的就是需要額外呼叫一個 setInterval 函數來定時執行。

3.2.2 FPS Algorithm 2

關鍵是利用時間差來達成的，概念就是先從『繪製每個影格需要幾秒（SPF）』下手，再藉由將其倒數處理來達成我們的目的：『每秒可繪製幾個影格（FPS）』。而此法計算出的 FPS 值必然是浮點數。也需要額外呼叫一個 setInterval 函數來定時執行。每一秒抓出前一次完成繪圖的時間（單位：微秒），然後做出乍看下是相當即時的計算資訊，可是其實網頁程式在執行的過程裡，有一些因素會影響計算數據，所以程式執行數據相對不穩，有時數值跳動較大。

3.2.3 FPS Algorithm 3

方法是每當程式呼叫一次繪圖函數做繪圖時，便把影格計數器的值加 1，然後隨著程式不斷呼叫繪圖函數的同時，我們利用累積時間差的做法，直到滿足一秒鐘，才印出那一秒鐘所渲染的影格數。此方法接近第一種演算法，故此法的 FPS 值也是整數的。優點是，不再需要呼叫一個 setInterval 函數來計時執行。程式執行數據穩定。因此在非火狐瀏覽器，我們會採用這個演算法來實作。

第四章　實驗結果

4.1　實驗測試的相關資訊

- 硬體測試環境與所用瀏覽器
  - Mobile Phone – Samsung GT I9100 Galaxy SII
    - OS: Android 4.1.2
    - CPU: Dual-core 1.2 GHz Cortex-A9
    - GPU: Mali-400 MP
    - Browser: 主要為 Firefox
  - NB – GIGABYTE U2442F
    - OS: Windows 7
    - CPU: Intel i7 3537U
    - GPU: NVIDIA GeForce GT 650 M
    - Browser: 主要為 Firefox

本研究實驗所使用的 Utah Teapot 三維模型資訊：

Vertexs: 608

Faces: 1024

除了硬體顯卡要支援 WebGL 外，也要測試瀏覽器是否支援 WebGL，我們可以簡單藉由網路上一些現成的檢測頁面來得知，也可以查看關於 GPU 的相關資訊。圖 4 是 http://get.webgl.org/的檢測畫面。圖 5 是 http://webglreport.com/的檢測畫面。

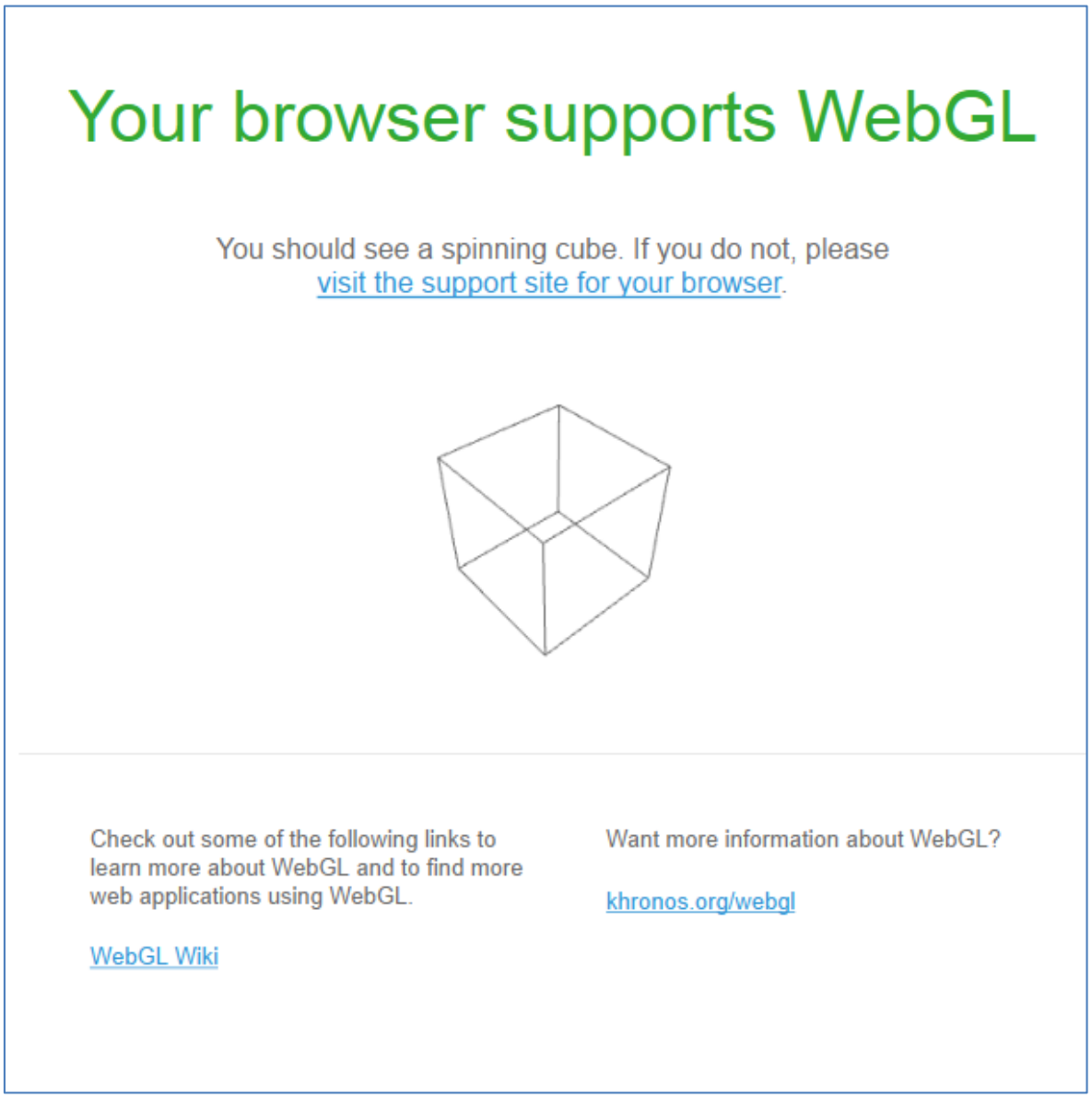

圖 4　測試瀏覽器是否支援 WebGL

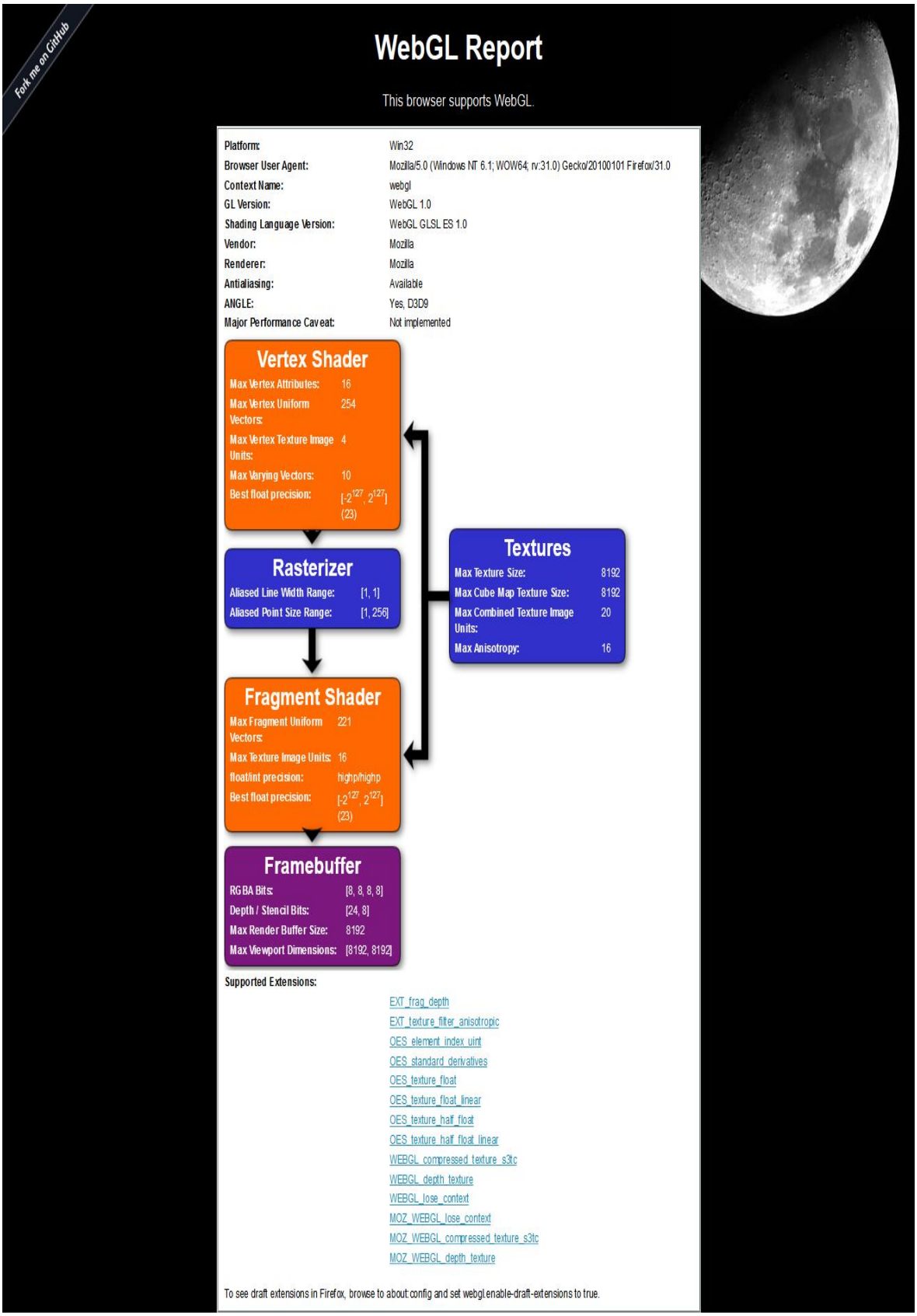

圖 5　WebGL Report - Chrome 資訊與繪圖顯示晶片相關資訊

4.2 實驗比較結果分析

我們知道行動裝置與個人電腦，他們的計算能力以及硬體規格是有差距的，如圖6，我們將分別列出手機及電腦的顯示卡或顯示繪圖晶片的詳細規格。

GPU	Cores	Clock	Fill Rate (Pixel)	Processing Power	Memory
Mali-400 MP	4	500 MHz	0.5 GP/s	5 GFLOPS	X
GeForce GT 650 M	384	835 MHz	13.4 GP/s	641.3 GFLOPS	1024 GB

圖 6　手機與電腦之 GPU 規格比較

圖 7 是電腦與手機在跑網頁程式時的效能情形，很明顯的當 3D 場景中的物體或者說多邊形的數量增加時，也就是場中的點跟面增多時，每秒的畫面更新率會降低，隨著場景中的複雜度增加，硬體資源會出現吃緊的狀況，尤其是 GPU 運算能力相對較差的行動裝置，相較於一般電腦會有很明顯的落差。

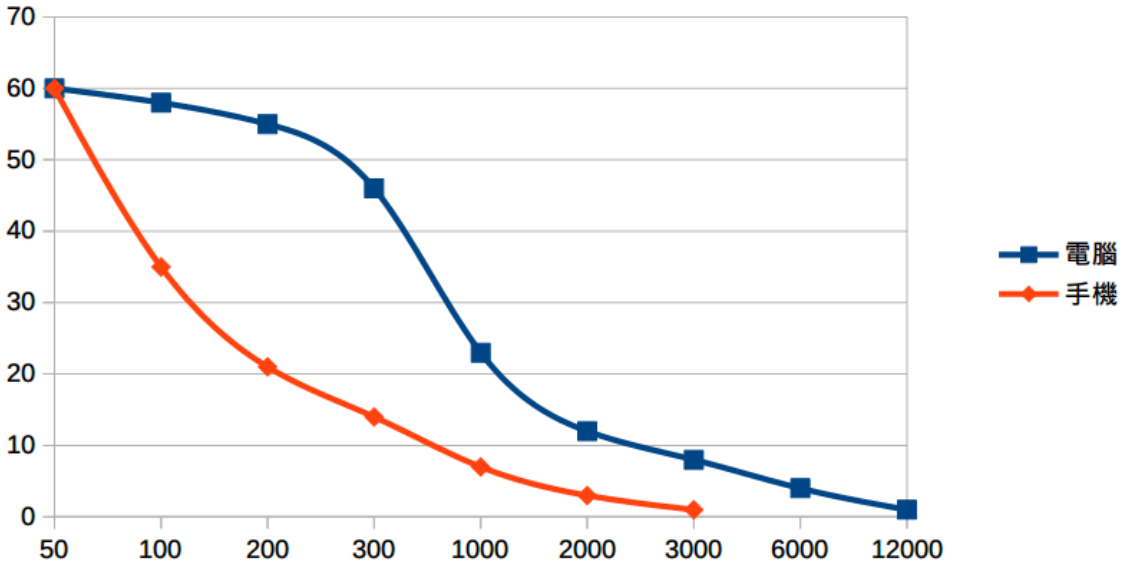

圖 7　手機與電腦之效能比較

註：

X 軸：Teapot 個數。

[個數, 三角形面數] = {50, 51200 => 100, 102400 => 200, 204800 => 300, 307200 => 1000, 1024000 => 2000, 2048000 => 3000, 3072000 => 6000, 6144000 => 12000, 12288000}

Y 軸： FPS 值

4.3 實驗成果圖

由於本研究是可跨平台顯示的，故本章所有展示之網頁 3D 動畫均可在各平台瀏覽。以下我們先展示手機的 demo 畫面。

首先是一個在空間中散布的茶壺，總共數量是 36 個。手機作業系統是 Android，所以程式會偵測到 Linux，然後瀏覽器非 Firefox，則 FPS Algorithm 1(for firefox only) 的畫面更新率演算法不會執行也不會顯示，以下手機分段截圖是展示電鍍材質的茶壺，其他設定參數如光澤度及各光線的顏色亮度，將如圖 8 所示。

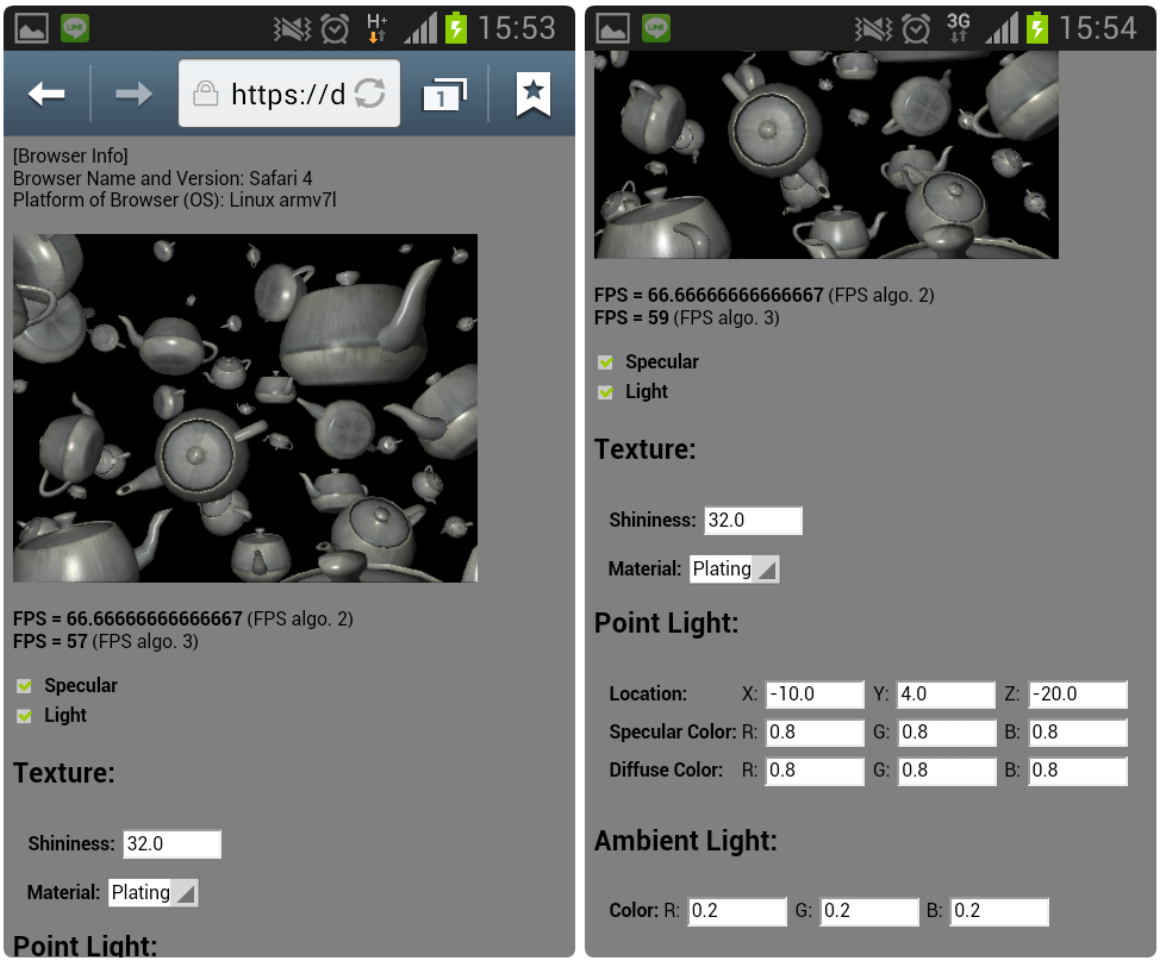

圖 8　Mobile 執行畫面一

其次是展示一個在空間中並列的茶壺，總共數量是 90 個。在此數量下，我們可以觀察到手機的畫面更新率明顯下降。手機作業系統是 Android，瀏覽器為 Firefox，FPS Algorithm 1（for firefox only）的畫面更新率演算法會執行並顯示，如圖 9。

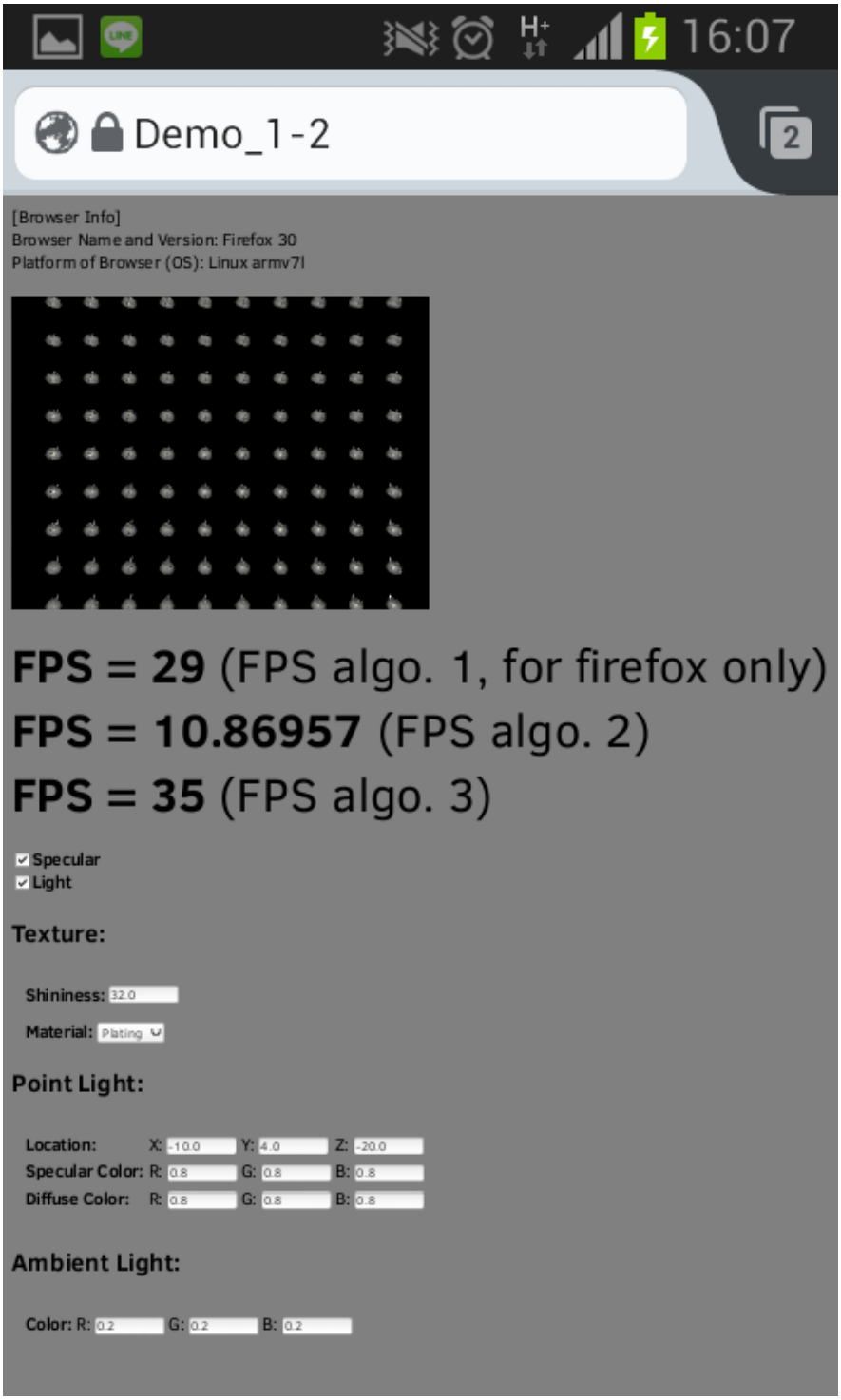

圖 9　Mobile 執行畫面二

再來展示一個在空間中，單一個茶壺。我們可以細看各項設定產生的紋理效果及光照變化。手機作業系統仍是 Android，瀏覽器為 Firefox，如圖 10。

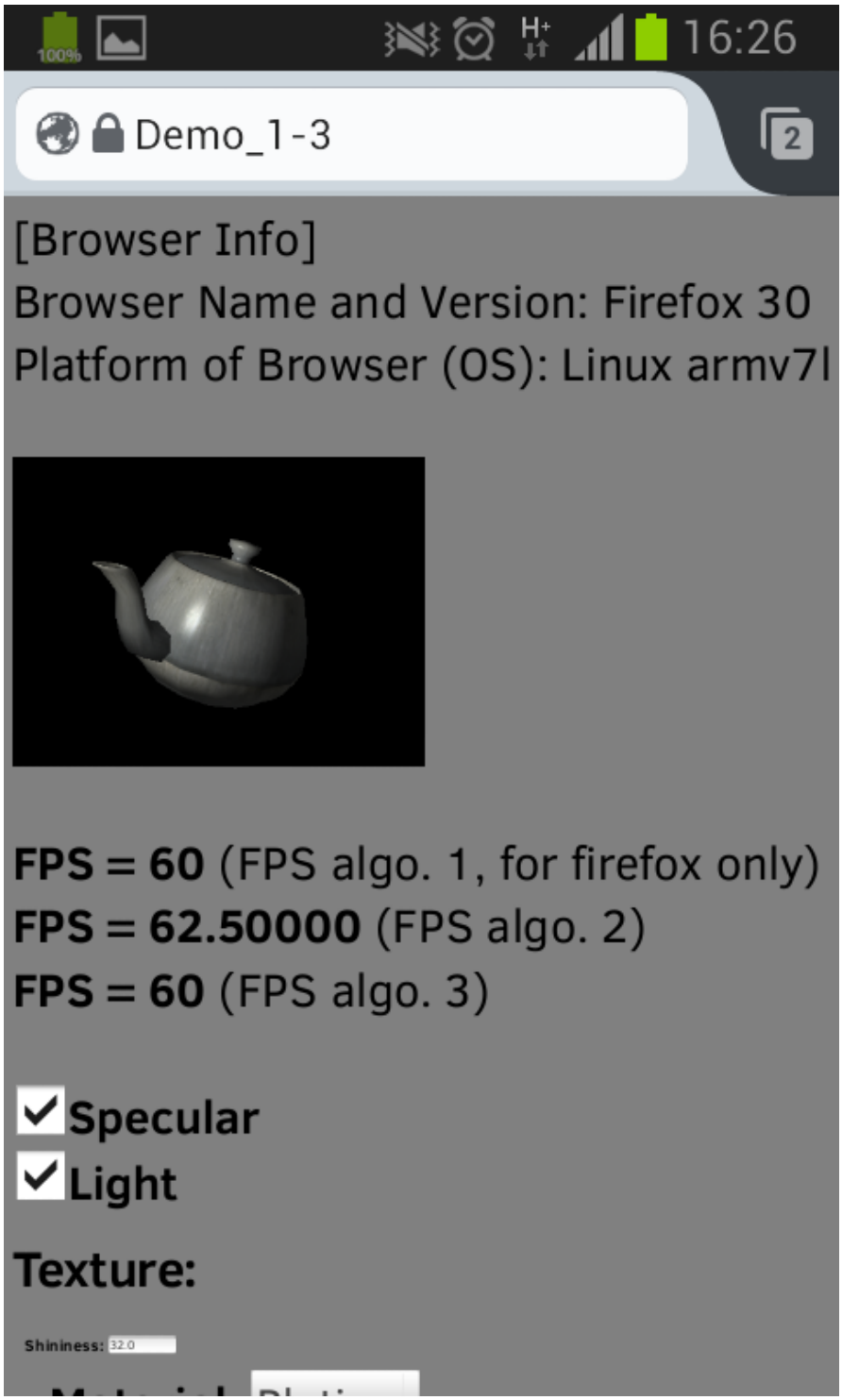

圖 10　Mobile 執行畫面三

下面一幅合併四張比較圖 demo 馮氏反射模型，圖中展示了一個可監看 FPS 並具有材質貼圖的光照模型圖，全都是在電腦平台上執行的，作業系統是 Windows，瀏覽器是 Firefox 的環境。以下是在開啟光照的場景中不開啟材質的狀況，分別針對四種情形做對照比較，依序從左上、右上到左下、右下，分別為：環境光、漫射光、鏡射光，以及前三者組合起來的效果圖。

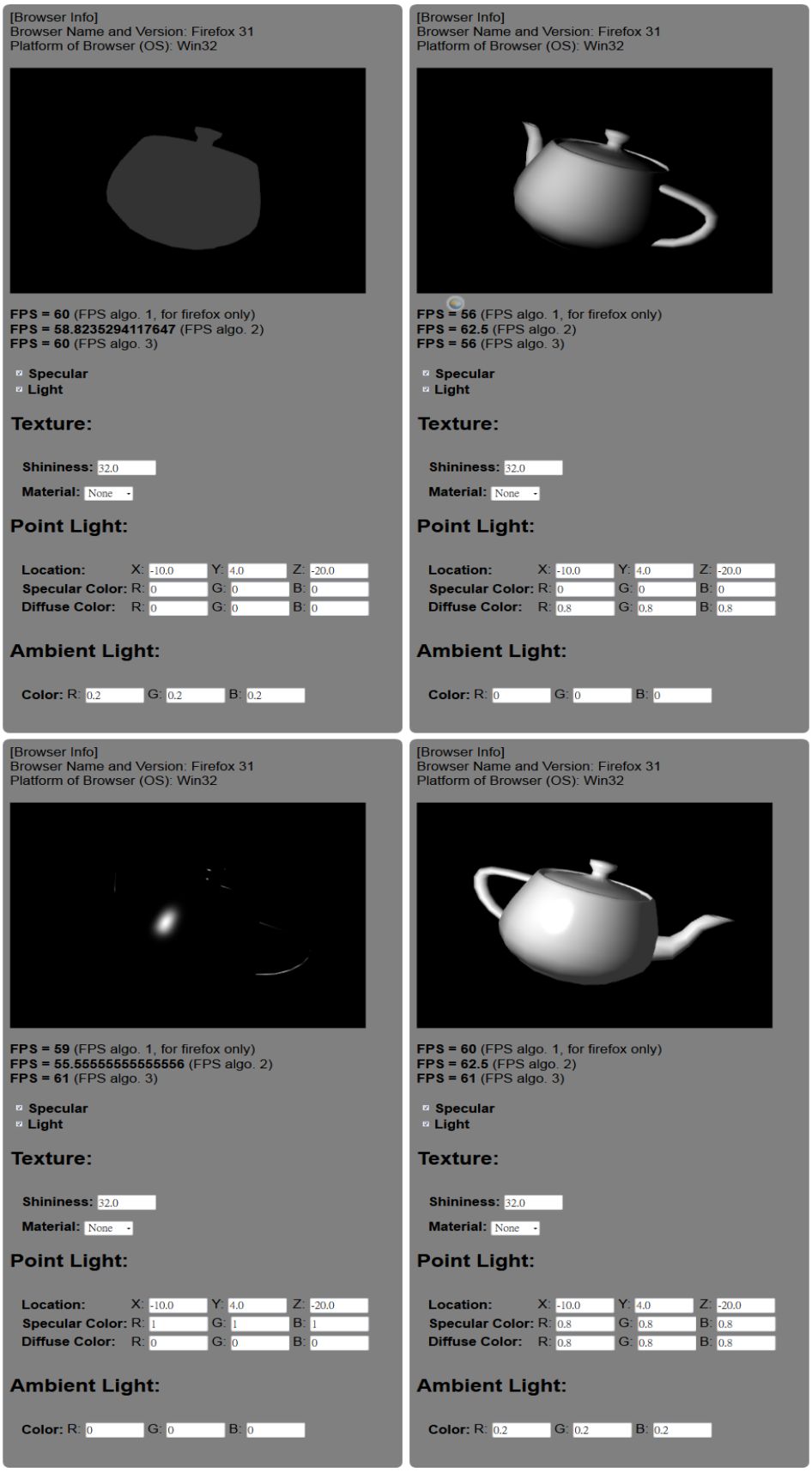

圖 11　PC 執行畫面 - Phong's Reflection Model

我們除了證明可跨平台跑程式跟顯示外，也證明了可以跨瀏覽器執行，以下兩張圖的個人電腦環境分別是 Ubuntu 作業系統和 Windows 作業系統。Ubuntu Linux 示範的瀏覽器是 Chromium 及 Firefox；Windows OS 示範的是 Chrome、Firefox、IE 及 Opera。

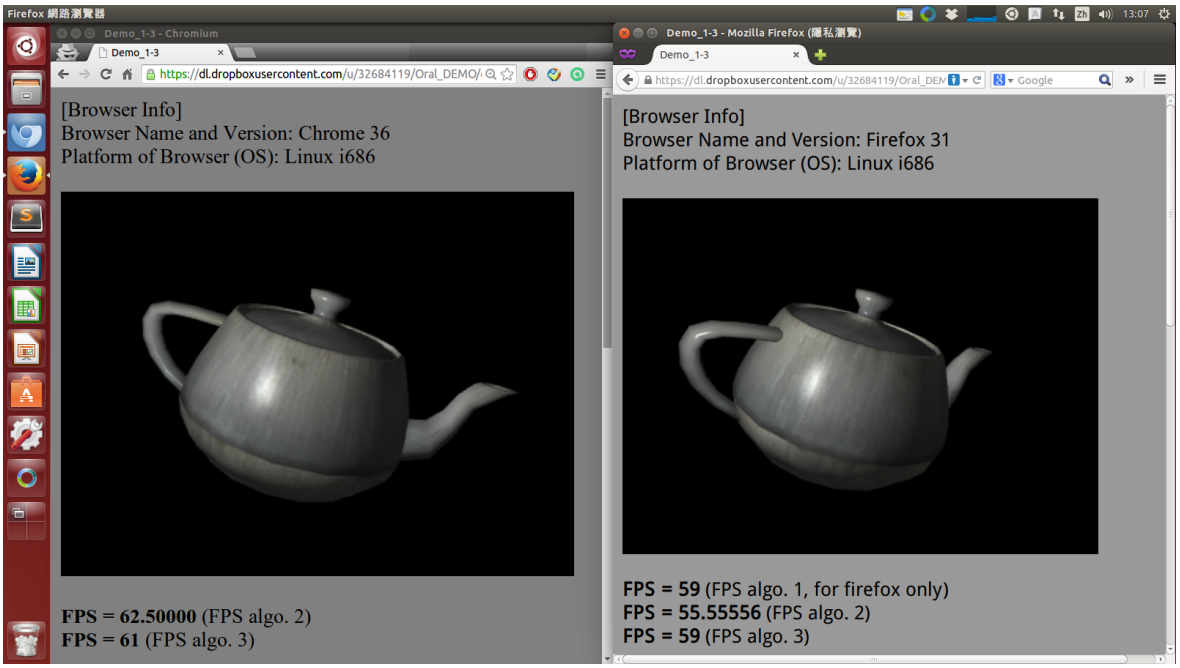

圖 12　Ubuntu 14.04 with Chromium and Firefox

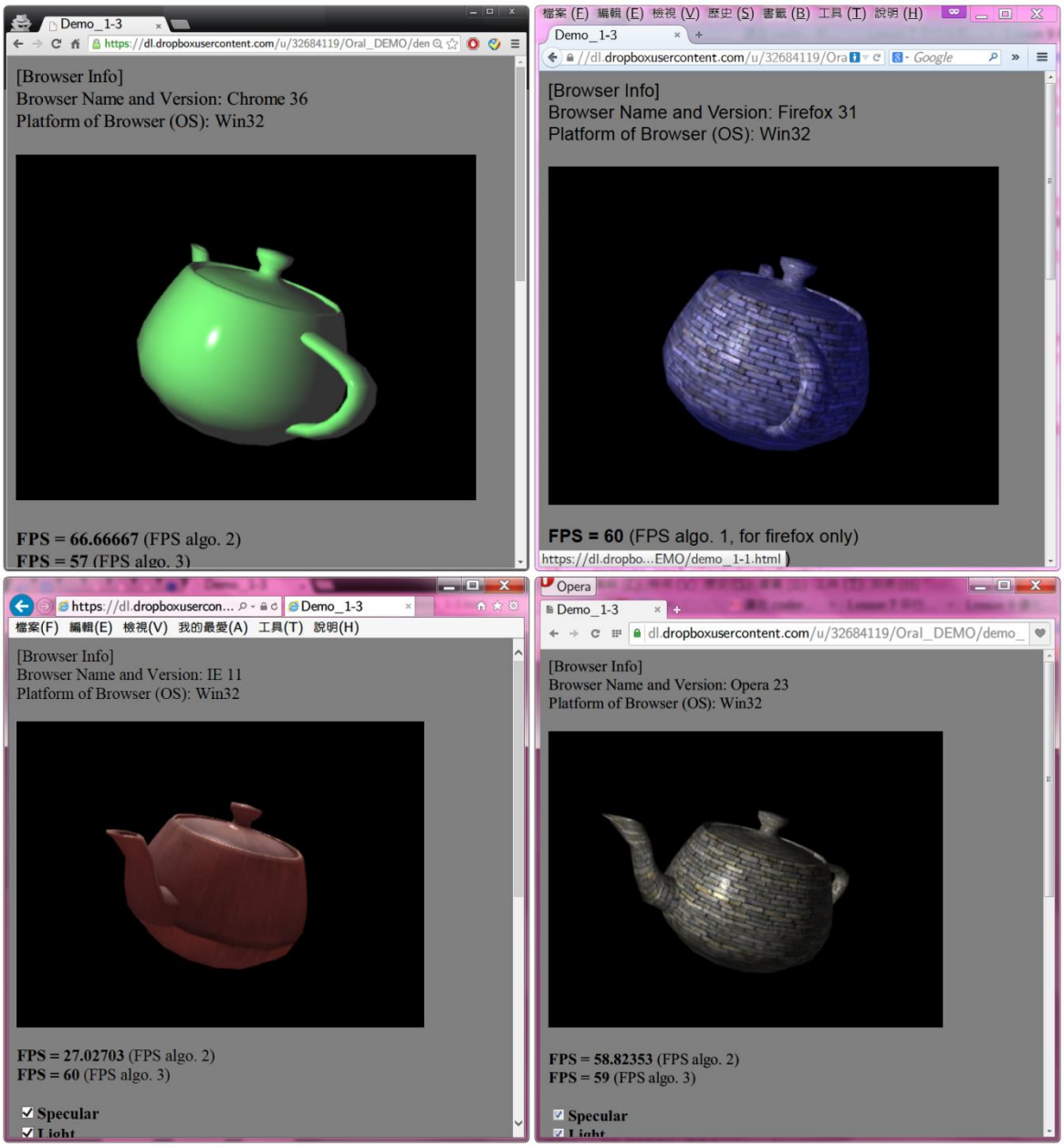

圖 13　Windows 7 with Chrome, Firefox, IE and Opera

極限壓力測試部分，當場景中的物體（此例即茶壺）數量多到一個程度時，畫面更新率將會降到每秒一幀左右的情況。圖 14 為 PC 的畫面（茶壺數量 12000, 也就是三角形數 12288000），圖 15 為手機的畫面（茶壺數量 3000, 也就是三角形數 3072000）。註：因為場景物件數量龐大，螢幕大小有限，無法盡覽，有很多在三維空間內的運算的茶壺無法被看到。

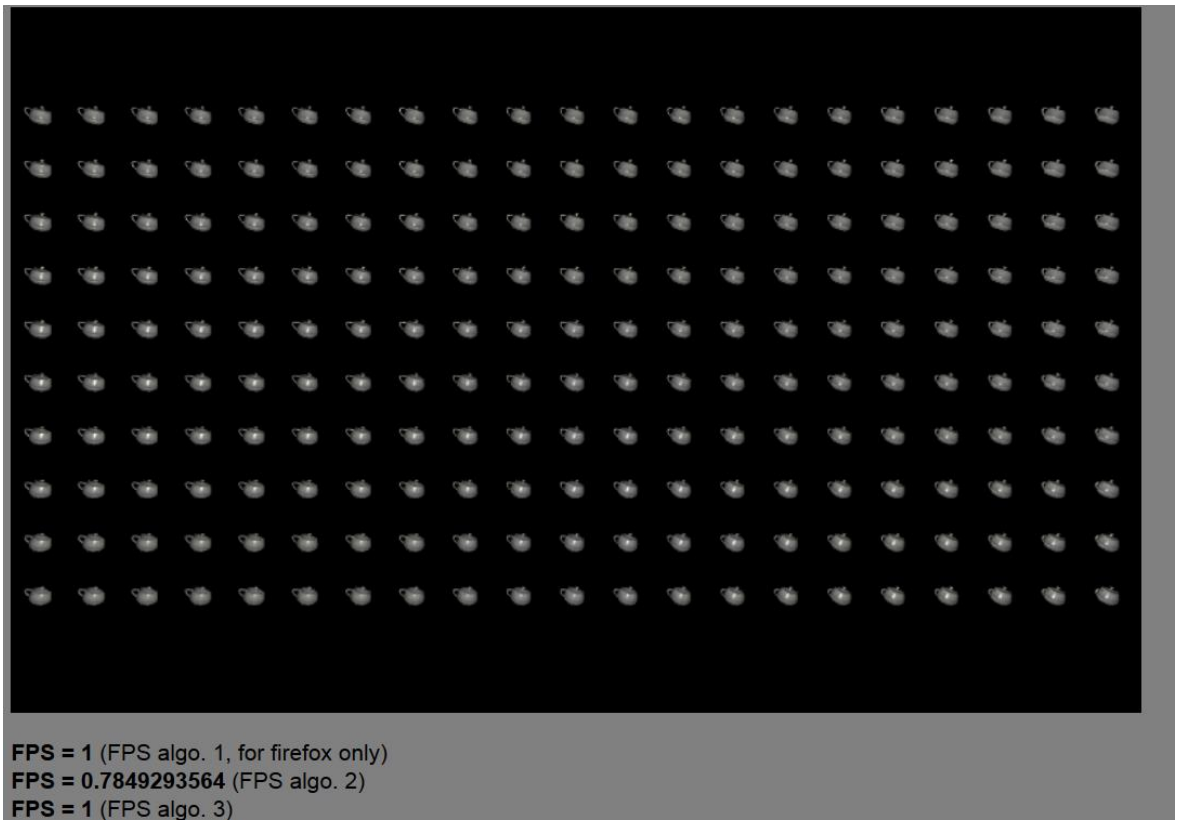
FPS = 1 (FPS algo. 1, for firefox only)
FPS = 0.7849293564 (FPS algo. 2)
FPS = 1 (FPS algo. 3)

圖 14　電腦壓力測試

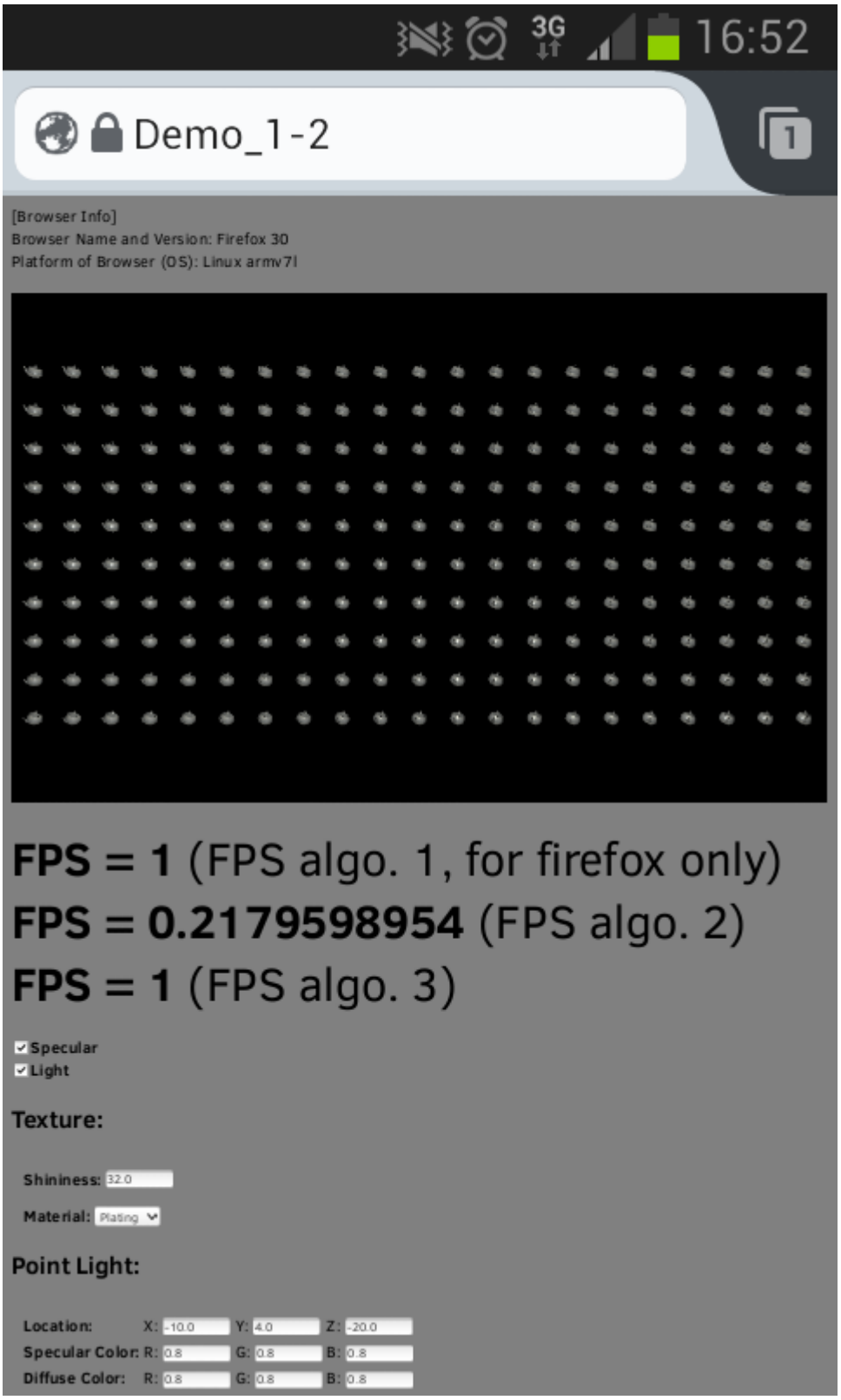
16:52
Demo_1-2
[Browser Info]
Browser Name and Version: Firefox 30
Platform of Browser (OS): Linux armv7l
FPS = 1 (FPS algo. 1, for firefox only)
FPS = 0.2179598954 (FPS algo. 2)
FPS = 1 (FPS algo. 3)
Specular
Light
Texture:
Shininess: 32.0
Material: Plastic
Point Light:
Location: X: -10.0 Y: 4.0 Z: -20.0
Specular Color: R: 0.8 G: 0.8 B: 0.8
Diffuse Color: R: 0.8 G: 0.8 B: 0.8

圖 15　手機壓力測試

第五章　結論與未來展望

有別於過去只能單純地在各家電腦平台上用原生的程式來執行電腦繪圖的呈現，現在則可以 Web 形式藉由瀏覽器利用 WebGL 來透過 GPU 去執行即時的圖形運算，達到跨平台的目的。

現代瀏覽器幾乎都支援 GPU 硬體加速（有些瀏覽器可能因為還在實驗階段或者可能導致瀏覽器不穩定以及基於資訊安全的理由，預設並未開放，需要手動去設定啟用），讓許多以 Canvas 為基礎的 Web App 線上應用程式運作更加流暢，甚至能比擬本機程式的速度，這將強化線上應用程式的實用性。

如今只要在硬體規格許可下（使用有支援 OpenGL 的顯示卡或繪圖晶片），並配合支援 WebGL 的網頁瀏覽器，再使用 JavaScript 來利用 WebGL 及 HTML5 Canvas 撰寫渲染器，便能在瀏覽器中實現瘋狂的動畫效果。本研究的結果也意味著未來將有更多的 3D 互動程式應用在 Web 上，對於行動式裝置開發有著重要的地位，對於跨平台問題的解決方案更是有著重大的意義。

由於電腦圖學領域的研究往往具有大量高度平行運算的特質，因此更適合運行在 GPU 上，故未來可再搭配 WebCL 做平行運算，只是目前因 WebCL 仍在草案階段，技術規格未定，且持續變動中，研究資料也相對較少，不過仍可預期這將是未來一個重點方向。

參考文獻

[1] HTML5, http://www.w3.org/TR/html5/

[2] WebGL, http://www.khronos.org/registry/webgl/specs/1.0/

[3] Shader, http://en.wikipedia.org/wiki/Shader

[4] The OpenGL® ES Shading Language, http://www.khronos.org/files/opengles_shading_language.pdf

[5] Ray Tracing, http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtrace0.htm

[6] Web browser engine, http://en.wikipedia.org/wiki/Web_browser_engine

[7] Javascript engine, http://en.wikipedia.org/wiki/JavaScript_engine

[8] WebGL | MDN, https://developer.mozilla.org/en-US/docs/Web/WebGL

[9] Frame rate, http://en.wikipedia.org/wiki/Frame_rate

[10] Browser rendering loop, http://fhtr.blogspot.tw/2011/04/browser-rendering-loop.html

[11] Window.mozPaintCount, https://developer.mozilla.org/en-US/docs/Web/API/Window.mozPaintCount

[12] Roth, Scott D., "Ray Casting for Modeling Solids", Computer Graphics and Image Processing pp. 109–144

[13] Golubovic D., Miljkovic G., Miucin S., Kaprocki Z., Velisavljev V., WebGL implementation in WebKit based web browser on Android platform, 2011

[14] G. Anthes, “HTML5 leads a Web revolution,” Communications of the ACM, 2012.

[15] Rama C. Hoetzlein, Graphics Performance in Rich Internet, 2012

[16] D. Geary, Core HTML5 Canvas: Graphics, Animation, and Game Development. Prentice Hall, 2012.