第五章 分析與結論

根據前面第三章與第四章所提出的轉換模型（Transformation meta model），我們實做使用 Javascript 來控制的 SVG，使其能使用 SVG 來呈現 Fractal 的範例圖形與 L-system 的範例樹型。5.1 節中對於第三章與第四章的概念做整合；5.2 節中將依照實做的過程對 SVG 與 Fractal、SVG 與 L-system 的比較來作圖形呈現效能分析；5.3 節為結論；5.4 節為未來發展。

5.1 Fractal 與 L-system 範例整合

第三章與第四章中提出了轉換模型的概念與 subdivision method、segment recursive grow method 等方法。在這一節當中藉由表 5-1 對 Fractal、L-system 與 SVG 的特性作比較；也由表 5-2 將所有的範例與轉換方式、碎形維度做一個整理。

	Fractal	L-system	SVG
描述內容方式	以『線段』為主軸的 Fractal 可經由規則形成結果字串來描述；而其他的 Fractal 需要藉由文字說明來敘述。	可以用遞迴次數、起始符號及公理字串的參數來描述。	以座標點來定位各個圖形物件的大小、顏色或是位置等參數，可藉由遞迴的方式來描述 Factal 與 L-system。
圖形呈現方式	多半以圖檔來呈現	可藉由 L-system Generation Program 來呈現	以內含標籤的純文字檔利用瀏覽器及 SVG plug-in 來呈現
改變遞迴次數時的物件個數	若是以圖檔來呈現，不需要討論物件個數	改變遞迴次數只需改動一個參數值	物件個數成指數成長

表 5-1 Fractal、L-system 與 SVG 特性比較

範例名稱	轉換方式	碎形維度
Cantor Set	1D subdivision method	0.6309
Koch snowflake	2D line subdivision method	1.2618
Sierpinski gasket	2D area subdivision method	1.5850
2D Tree	2D segment Recursive grow method—Standard Type	unkown
2D Willow	2D segment Recursive grow method—Willow Type	unkown
3D Tree	3D segment Recursive grow method—Standard Type	unkown
3D Willow	3D segment Recursive grow method—Willow Type	unkown

表 5-2 Fractal、L-system 轉換方式與碎形維度

藉由表 5-1 將 Fractal 與 L-system 轉換成 SVG 作了整理，可以看出 Fractal 與 L-system 雖然在概念上大致相同，作法也很相像，不過 Fractal 是自我細分為更小的部分，L-system 是依照原有的骨架越長越大。比較完了 Fractal 與 L-system 的轉換之後，下一節將介紹使用 SVG 來繪製 Fractal 與 L-system 的圖形呈現效能分析。

5.2 圖形呈現效能分析

由於本論文中首先是把 Fractal 的樣本範例以 SVG 的方式呈現出來，接著將以 Fractal 為基礎的 L-system 樣本範例也以 SVG 來呈現。所以在分析的過程中也分為兩部分：SVG 與 Fractal 的分析、SVG 與 L-system 的分析。

5.2.1 SVG 與 Fractal 的呈現效能比較

在第三章中已經比較過 Fractal 與 SVG 的不同之處，因此在本節當中直接就實做出來的結果作一個統整的分析，如表 5-1。一般看到 Fractal 的教學都以圖片來呈現，因此我們就各種圖檔的類型與 SVG 做一個比較。

- Cantor set

	GIF 圖檔	JPEG 圖檔（壓縮比：90%）	PNG 圖檔	SVG	
				靜態	動態（含 script）
遞迴一次	755B	2.11KB	1.34KB	354B	2.27KB
遞迴兩次	783B	2.43KB	1.37KB	435B	2.27KB
遞迴六次	841B	2.54KB	1.41KB	5.54KB	2.27KB
可動性	有	無	無	無	有
互動性	無	無	無	無	有

（以上圖檔的 size：450X380 為本論文的樣本）

表 5-3 Cantor set 與 SVG 的檔案比較

- Koch Snowflake

	GIF 圖檔	JPEG 圖檔（壓縮比:90%）	PNG 圖檔	SVG	
				靜態	動態（含 script）
遞迴一次	1.66KB	4.76KB	2.95KB	506B	3.64KB
遞迴兩次	2.02KB	5.86KB	3.28KB	1.22KB	3.64KB
遞迴六次	4.56KB	10.8KB	8.58KB	253KB	3.64KB
可動性	有	無	無	無	有
互動性	無	無	無	無	有

（以上圖檔的 size：450X380 為本論文的樣本）

表 5-4 Koch Snowflake 與 SVG 的檔案比較

- Sierpinski gasket

	GIF 圖檔	JPEG 圖檔（壓縮比:90%）	PNG 圖檔	SVG	
				靜態	動態（含 script）
遞迴一次	1.62KB	4.63KB	2.97KB	351B	2.35KB
遞迴兩次	1.91KB	5.86KB	3.46KB	530B	2.35KB
遞迴六次	4.21KB	11.5KB	4.66KB	19.74KB	2.35KB
可動性	有	無	無	無	有
互動性	無	無	無	無	有

（以上圖檔的 size：450X380 為本論文的樣本）

表 5-5 Sierpinski gasket 與 SVG 的檔案比較

在第一章當中已經概略的介紹過了這幾種圖檔的不同之處。再由上面的三個表中的不同的遞迴次數比較之後，可以看出一些值得討論的地方：

1. 表格當中的『可動性』是對於能否動態的看出 Fractal 的變化而言；『互動性』方面是對於使用者能否操縱此 Fractal 的變化而言。在可動性來說，由於 GIF 可以有動畫效果，所以可以製作 Fractal 的動畫以供使用者瞭解，當然 SVG 在這一方面也可以達成。而互動性來說，GIF、JPG 與 PNG 都無法達到可以給使用者控制的效果，而 SVG 也可以達成。
2. 由於 GIF、JPG 以及 PNG 都是網路常用的圖檔格式，因此也都經過壓縮，所以當圖檔的內容作改變時，檔案大小也會有部分的改變。不過就圖檔格式來看都還是以 pixel 為主，所以對於同樣大小的圖（pixel 數目不變）， Fractal 樣本的變化對於檔案大小並不會有驚人的變化。
3. 根據第一章中對 GIF 特性的描述，由於 GIF 是採用色盤的壓縮方式，而本論文中的樣本是在平面上繪製單一顏色的線條或圖形，因此 Fractal 樣本中的壓縮效果相當好，若是圖形有複雜的顏色，檔案會變的相當大。
4. 靜態的 SVG 方面，由於 SVG 是 XML based 由標籤組合而成。所以若是將 SVG 的檔案當中的標籤數增加，檔案也將跟著變大。正如 Koch Snowflake 的部分，遞迴一次時有 3 個物件（3 條線），檔案只有 506B；遞迴兩次時有 12 個物件（12 條線），檔案增加到了 1.22K；若是遞迴六次之後，由於物件是成指數倍成長，總共有 3072 個物件，因此檔案增加到了 253K。
5. 對於 Koch Snowflake 與 Sierpinski gasket 來說，這兩個樣本都是屬於複雜的圖形，並且在 GIF、JPG 以及 PNG 的圖檔大小方面都差距不太大，但是在靜態的 SVG 檔方面就有很顯著的不同。其中最主要的原因是 Sierpinski gasket 雖然圖形複雜，但是其中的物件數比 Koch Snowflake 少很多，所以檔案也小很多。

根據以上的幾點的歸納，針對 Fractal 圖形來說，形成的物件個數多寡對於採用靜態的 SVG 檔案有很大的影響。因此對於靜態的 SVG 檔案來說，若是圖形可以分

割為數量不是太多的物件來繪出，絕對可以達到減少檔案傳送時間的目的。另外由於 SVG 的顏色只需給定著色的物件一個屬性即可，所以無論顏色多麼複雜都對檔案大小沒有影響。再則由於 SVG 是向量圖形，所以若是圖形很大，GIF、JPG 以及 PNG 的檔案大小勢必跟著增加，而 SVG 只要調整參數即可，不會影響檔案的大小。

接下來討論的是動態 SVG 的部分，前面我們說到靜態 SVG 檔案的大小是跟物件的多寡有關。然而針對 Fractal 圖形來說，Fractal 是有規則性的，在前面的第三章中可知，Fractal 的自我描述性都可以用程式中的迴圈來達成。由這一點看來，若是 Fractal 圖形遞迴了很多次之後，物件個數是成指數倍成長，所以靜態 SVG 檔案才會比較大。若是根據 Fractal 的特性，在圖檔中以迴圈來代替已經固定的物件標籤，改由在 client 端才藉由 DOM 的概念動態產生物件標籤，我們在傳遞檔案的時候便可以只要傳遞含有 script 的檔案即可，不論遞迴的次數多寡、圖形如何複雜，檔案的大小也必定也可以減少許多，這便是 SVG 這種由標籤組成的圖檔一個很大的優勢，就是可以採用與 HTML 相同的方式動態的產生標籤。

5.2.2 SVG 與 L-system 的呈現效能比較

在經由前面 SVG 與 Fractal 的比較之後，接著再對 SVG 與 L-system 做進一步的分析。我們知道 L-system 是有規則的，其中包含了敘述時需要的幾項參數，例如：遞迴次數、偏轉的角度、樹枝粗細以及替換的規則。當要改變 L-system 的遞迴次數時，只需改變 L-system 的一個參數值即可；而在 SVG 的部分，物件數目則會成指數倍成長。下面是 2Dtree 與 2Dwillow 兩個樣本的物件數目成長圖：

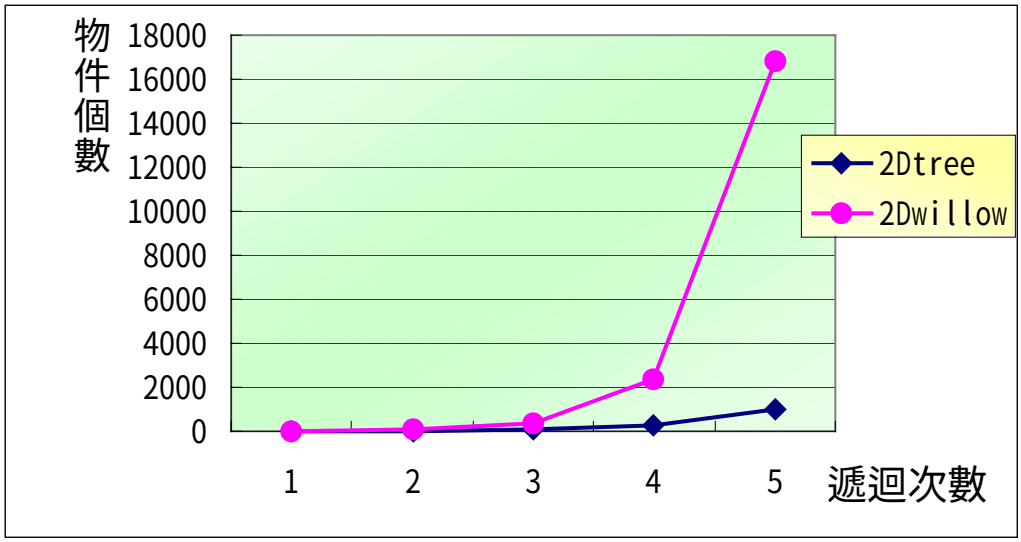

表 5-6 2Dtree 與 2Dwillow 遞迴一到五次的物件個數成長圖

由上圖可知，2D 的部分由於 L-system 只要改動一個參數值即可，而 2Dtree 的物件數目會呈指數倍成長（4、4^2、4^3…..）。同樣的，2Dwillow 的物件數目也會呈指數倍成長（7、7^2、7^3……），所以要是以靜態 SVG 來顯示，遞迴次數一高，檔案一定相當大，並且缺少了與使用者間的互動性。由這幾點看來，使用動態 SVG 來呈現是很好的選擇。

在 3D 的物件部分，由於現在的 3D 的技術多半都是結構龐大的環境，所以我們採用了 SVG 這個純文字的格式來呈現 3D 的物件更顯得輕便。與前面所敘述的道理相同，當使用者在傳送 3D SVG 的檔案時所需的時間是很少的，因為只需要傳送一些文字檔案，而呈現的部分都在 client 端才執行。

3D 的環境中，使用者除了想看到 3D 的物件之外，更需要能轉動來觀察各個角度的情形，所以並不適用於將 3D 的樹畫在 2D 的畫布上，因此我們將 3Dtree 建立在 3D 的 SVG 上面。同樣的，在這一方面也採用了 JavaScript 使使用者能旋轉並將物件停止在想觀察的角度。不過在 3D SVG 的的環境當中，物件數量有著很大的影響性，因為我們使物體旋轉的時候，必須依照物體旋轉的角度來動態的計算出每一點在這個時刻應該轉到了哪一個座標位置，接著定點、組合物件，所以在旋轉的過程當中，

必須一直動態的做坐標點的計算。而物件越多，要算的點也越多，速度就會比較慢一些。其中若是立體樹型只有骨幹的架構，3Dtree 的物件數目是成指數成長（6、6^2、6^3……），而 2Dwillow 的物件數目也呈指數成長（9、9^2、9^3……）。若是要更有立體敢，3Dtree 的每一根樹枝以六角柱來代替直線（六角柱是由 8 個面，也就是 8 個 polygon 的標籤組合而成），物件數目便成為（8*6、$8*6^2$、$8*6^3$……），3Dwillow 以六角柱來代替線狀的樹枝，物件數目便成為（8*9、$8*9^2$、$8*9^3$……），而 3Dtree 遞回一到五次的物件個數如下表 5-5 所示；3Dwillow 遞回一到五次的物件個數如下表 5-6 所示，物件數目的龐大由此可知。

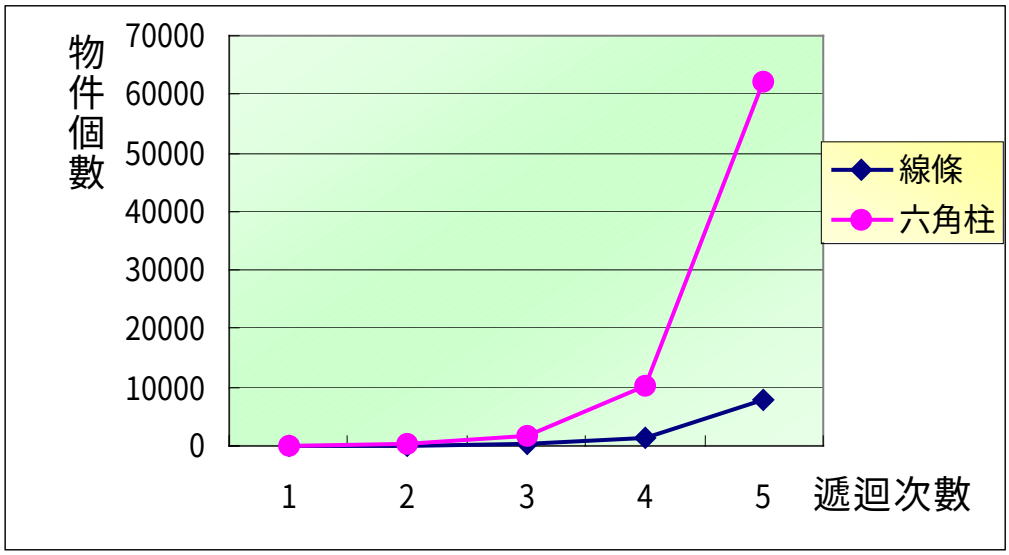

表 5-7 3Dtree 遞迴一到五次的物件個數

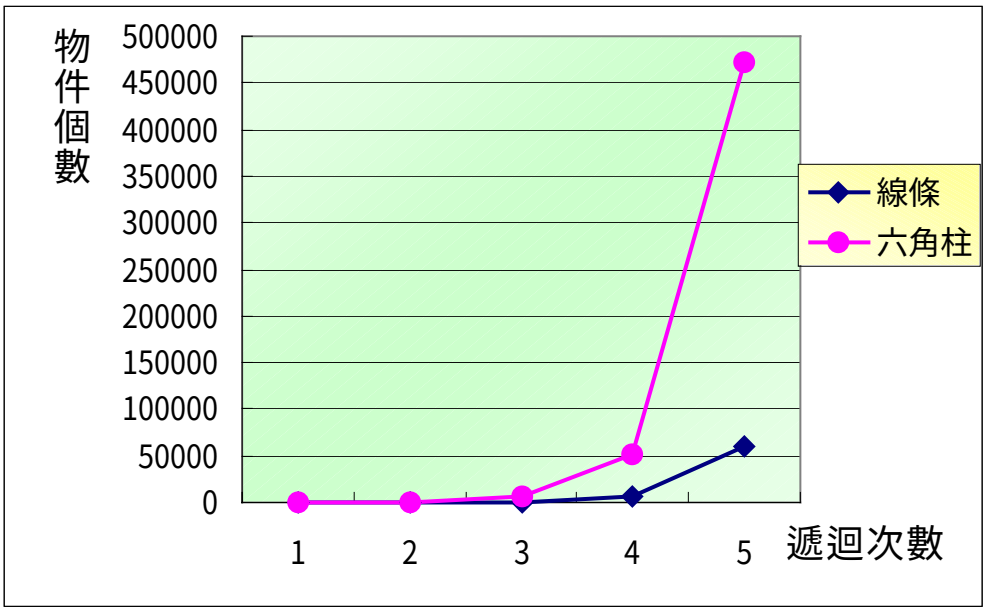

表 5-8 3Dwillow 遞迴一到五次的物件個數

換句話說，若是使用 L-system Generation Program，只要改動一個參數值便可以達到不同遞迴次數的效果，並且可以旋轉物體。不過前面也提過 L-system Generation Program 必需要使用者安裝才能使用，這是不方便的。在這裡 3D SVG 提供了使用者一個在網路上只要安裝 SVG plug-in 便可以觀看的 3D 環境，雖然他的效能與物件的數量與計算有關，但也已經方便了許多，成為一個很大的優勢。因此將 L-system 與 SVG 做結合，以 L-system 當作向量圖的格式，便可以傳遞少量的資料，再使用 Javascript 在 client 端做計算與繪圖，這樣便可以達到很好的效果。

5.3 結論

在本論文中提出了轉換模型將 Fractal 及 L-system 轉換成 SVG，也提出了 subdivision method 與 segment recursive grow method 兩大類的轉換法則。因為 Fractal 及 L-system 都是有規則的，藉由他們簡單的規則，可以繪製出複雜美觀的圖形。但是在一般的圖檔當中，複雜的圖形勢必佔了很大空間，而在網路的傳輸上也必定要花費比較多的時間。對於複雜的圖形來說，SVG 這個以標籤來繪製的語言佔了很大的優勢，因為 SVG 不需要將圖片中每一個 pixel 的資料都花好幾 byte 個別儲存起來，只要將圖形存成描述圖形的標籤，傳送時只需傳送純文字檔案。對於調整圖形的大小或是調整顏色等功能，只需改變標籤中的一些參數即可，圖形的檔案並不會變大。除此之外，有規則的 Fractal 與 L-system 將可以達到更好的效果。因為藉由其特性可以使用 Javascript 來控制、動態的產生 SVG 文件當中的標籤，也就是以 Fractal 與 L-system 的規則當作傳遞的向量圖格式，讓 Javascript 接收到參數值後直接在 client 端繪製圖形，這將可以節省很多在網路上傳遞圖形的時間，當然對於在網路上觀看 3D 的物件時有更顯著的效果。這也是只有 SVG 這個以標籤來繪製圖形的語言特有的優點與特色，是其他圖形格式辦不到的。

隨著 XML 的蓬勃發展，以 XML 為基礎發展而成的 SVG 也逐漸的受到大家的重視。雖然就普及度來說，SVG 尚未達到一般使用者都瞭解的程度，因此安裝 SVG plug-in 的比率上面還不算是太高。但是由於 SVG 的許多的優點，有很多的軟體廠商開始注意到 SVG，並且發展配合 SVG 的軟體或是能閱讀 SVG 的 viewer。所以我們相信，在不久之後的將來，瀏覽器一定會內含 SVG plug-in，這一天指日可待。

5.4 未來發展方向

本論文中提出了轉換模型的概念，並利用這個概念實作了 Fractal 與 L-system 的一些樣本範例，以達到藉由 Fractal 與 L-system 的規則在 client 端繪製複雜圖形的效果，也將 L-system 由 2D SVG 的環境推向了 3D SVG 的環境。但是由於 SVG 本身即為 2D 向量圖形語言，所以在 3D 的環境之下尚未十分成熟，在本論文所使用的 3D SVG 環境中，也還需要對於景深、視角的轉換……等等再做處理。另外在 L-system 的 3D 樹型方面，也可以再增加一些供使用者調整的參數，以及可以模擬更真實的樹枝生長情形（例如越到頂端樹枝越細…等等）使樹型能作更多的變化，達到更好的效果。

而在 3D SVG 的執行效率方面，如何能合併一些物件，使得在 Client 端觀看的速度能更快，也是很重要的議題。