以可縮放式向量圖形語言呈現碎形圖形

Fractal Rendering Using SVG

葉耀明
國立台灣師範大學資訊教育所
ymyeh@ice.ntnu.edu.tw

林芃君
國立台灣師範大學資訊教育所
lesly@ice.ntnu.edu.tw

摘要

現今網際網路發展相當迅速，如何能傳送大量的資訊而速度也維持一定的標準是很重要的課題。而以 XML 為概念發展出來的 SVG（Scalable Vector Graphics）可縮放式向量圖形也漸漸的受到重視。SVG 相較於 Bitmap 有許多的優點，對於存取一般的圖形能降低儲存空間，並且可以節省在網際網路中傳輸的時間，有效的對網際網路的發展做出貢獻。

本論文中提出了將 Fractal 與 L-system 轉換為 SVG 來呈現的轉換模型(Transformation meta model)以及細分法則（subdivision method)、線段遞迴生長法則（segment recursive grow method)，再透過演算法用 SVG 平台來實作 Fractal 與 L-system 圖形並探討其特性。再則，SVG 本身為 2D 的向量圖形語言，但是其優點非常顯著，因此本論文也將 SVG 由 2D 推向 3D 的階段，繪出 3D 的 L-system 樹型範例以增加樹型的真實呈現。

關鍵詞：SVG、Fractal、L-system、可縮放式向量圖形語言、碎形

一、 簡介

對於一個結構複雜的圖形來說，用向量圖形來表示比用點陣圖形來表示具有更多的優勢，例如點陣圖放大的呈現方式是將 pixel 放大、粒子變粗，並不能看到更細微的變化，所以使用向量圖形來呈現碎形（Fractal）圖形並且配合 script 語法來提供使用者動態調整參數是一項很好的選擇。在網路圖形的發展中，SVG 逐漸成為 Web 上的主要向量圖形格式，故在本論文中將 Language based 的 Fractal 與 L-system 以 SVG 平台來呈現，並且討論其特性。

本研究分別對於 Fractal 與 L-system 提出了將其轉換為 SVG 的 Transformation meta model，並且提出以下的轉換方法：1D subdivision method、2D line subdivision method、2D area subdivision method、2D segment Recursive grow method 與 3D segment Recursive grow method，也對每一種方法實作了樣本的轉換，樣本的功能為提供使用者在 client 端調整 Fractal 的參數來改變圖形。在 3D 的部分，使用了中研院翁維瓏先生的 3D SVG 點與線的環境為基礎，修改並增加繪製 3D L-system Tree 樣本的功能，使得 L-system Tree 在不需要安裝除了 SVG 之外的 Plug-in 或是應用程式便可以在 client 端展示。在 2D 及 3D 的部分同樣不需要透過 client 端與 server 端的圖形傳送，因此不會佔用頻寬或因為連線的速度而影響觀看複雜的 L-system Tree。

本文共分為五節。第二節為圖形物件表示法之概念；第三節為碎形呈現演算法；第四節為 L-system 呈現演算法；第五節為圖形呈現效能分析與結論。

二、圖形物件表示法之概念

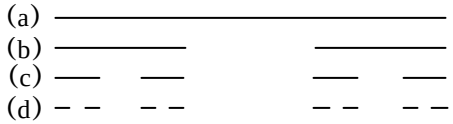
2.1 圖形物件表示法與模型

模型（model）是相對於真實世界的一個高階的概念。舉例來說，在物理學當中，我們常使用數學公式將真實世界中的情形抽象化。然而在計算機圖學的領域中有五種主要對於模型的研究：階層模型（hierarchical models）、語言模型（language models）、物理模型（physical models）、碎形模型（fractal models）以及資料整合模型（data set models）[1]。而在何種情況下使用哪一種模型，則是依照該模型的特色來決定，在接下來的內容當中我們將介紹碎形模型、以語言模型為基礎的林登梅爾系統與 SVG 圖形表示法。

2.2 碎形表示法

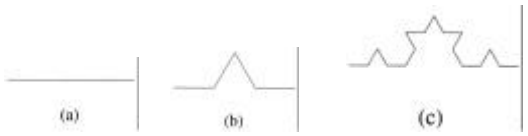
所謂碎形是指一個幾何物件在越來越細微的尺度上，不斷重複自我模仿，以固定的方式重複製造細節。其維度的計算如下：

若一直線長度為 1，將其平均分成 N 等分，每段長則為原邊長的 $r=\frac{1}{N}$ 倍。相對於一個正方形來說，把一正方形分成 N 等分，每段長則為原邊長的 $r=\frac{1}{N^{1/2}}$ 倍。再相對於一個正立方體來說，把一正方體分成 N 等分，每段長則為原邊長的 $r=\frac{1}{N^{1/3}}$ 倍。由此可知，若物件的維度為 D，則 $r=\frac{1}{N^{1/D}}$。故可以得到：$D=\frac{\log(N)}{\log(1/r)}$，其中 D 便是碎形的維度[2]。舉例來說，圖 1 的 Cantor set、圖 2 的 Koch Snowflake 及圖 3 的 Sierpinski gasket 也可依照此規則算出碎形維度。



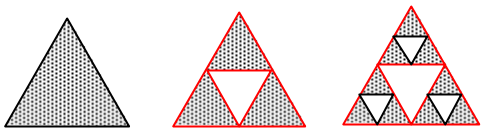
D=log(2)/log(3)　0.6309

圖 1 Cantor set



D=log(4)/log(3)　1.2618

圖 2 Koch Snowflake



D=log(3)/log(2)　1.5850

圖 3 Sierpinski gasket

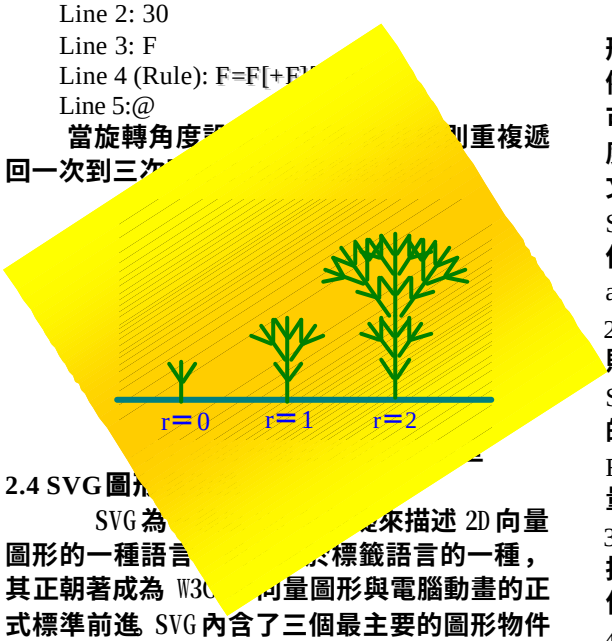
2.3 林登梅爾系統表示法

林登梅爾系統（L-system）是最典型的「語言模型」，其中心概念在於重新寫入字串。L-system 運用了重複自我模仿的特性，以一個公理字串及一到數個規則字串來描述一個樹型物件，公理字串可藉由規則字串以遞迴式來取代以取得結果字串，接著再以該結果字串利用海龜式繪圖法繪製出圖形[3]。

當要在畫面上顯示 L-system 的樹型時，可以利用海龜式繪圖法繪製出圖形。假設有一個 L-system 系統 G＝<V,ω,P>，而字母 V 代表{F,f,＋,－,[,]}，起始符號ω為 F，P 代表了規則 F[+F][-F]F。以下為此樹型結構在林登梅爾系統中的範例：

Line 1: 1
Line 2: 30
Line 3: F
Line 4 (Rule): F=F[+F
Line 5:@

當旋轉角度 則重複遞回一次到三次



2.4 SVG 圖形

SVG 為 來描述 2D 向量圖形的一種語言 標籤語言的一種，其正朝著成為 W3C 向量圖形與電腦動畫的正式標準前進。SVG 內含了三個最主要的圖形物件部分：向量圖形（例如：圓形、長方形或是直線等）、影像與文字；而這些圖形物件可以被群組化、合併及轉換座標。SVG 的圖形也可以達到人機互動（Interactive）和動態（Dynamic）的效果，並且藉由嵌入 SVG 的動畫標籤或是 scripting 的語法來達到動畫（Animation）顯示[4]。

除了 SVG 本身的優點之外，相對於一些圖形產生軟體，其中有一些部分是設計者無法知道與控制的，繪製 SVG 的 Designers 有完全的控制權力。並且 SVG 的編寫環境相當單純、容易，可用 Notepad、SimpleText 等純文字環境，所以可以在任何機器上編寫，如 286 PC、Mac Plus 甚至是 Palm Pilot。

三、碎形呈現演算法

3.1 轉換模型（Transformation meta model）

使用 SVG 來呈現碎形圖形需要一個轉換的方法與過程，本論文稱其為「轉換模型」。轉換模型有圖 5 的四個步驟：

Fractal 轉換模型的步驟

1. **觀察圖形**：Fractal 的基本概念是運用重複自我描述的特性，所以 Fractal 是由一個起始圖形開始，再將自己分為幾個小圖形；而這些小圖形在下一次的遞迴當中，每一個又將依照同樣的規則再分裂為更小的部分，一直持續的做下去，所以在本論文中稱其為細分法則（subdivision method）。在 0　1 維的 Fractal 圖形當中，本論文稱其為 1D subdivision method，例如 Cantor Set。在 1　2 維的 Fractal 圖形中，可以分為兩部分：一種是圖形的座標有兩個維度，如（x,y）。但圖形的是由線段來構成，本論文稱其為 2D line subdivision method，例如 Koch Snowflake ；另為一種座標同樣也有兩個維度，但圖形是由區域來構成的，本論文稱其為 2D area subdivision method，例如 Sierpinski gasket。
2. **幾何圖形分析**：在觀察出重複自我描述的規則之後，接著要由 SVG 的觀點來探討。已知 SVG 的圖形是由絕對或相對座標點來決定圖形的位置，所以在這一階段當中必須分析出這個 Fractal 要用 SVG 來表示時需要知道的座標參數量和物件類型。
3. **數學函式推導**：在分析完了幾何圖形之後，接著必須討論這些座標點之間有何關連性與如何利用數學函式求出這些座標點。
4. **細分法則演算法**：經過精密的分析之後，可以將導出來的三角函數與遞迴概念套入演算法當中，實際由程式去執行，就可以用 SVG 來呈現 Fractal 圖形。

3.2 一維細分法則（1D subdivision method）

依照上一節的轉換模型步驟，本節將探討屬於一維細分法則的例子：Cantor Set，並探討將其轉換為 SVG 的方法。

1. **觀察圖形：**

Cantor Set 在拓樸學中是一個非常有名的例子，其繪製的方式如圖 3-2 所示。首先給定[0,1] 閉區間，挖掉其三分之一等份的中間段開區間$(\frac{1}{3},\frac{2}{3})$，再將現有的兩個閉區間$[0,\frac{1}{3}]$、$[\frac{2}{3},1]$，各挖掉其三分之一等份的中間段開區間$(\frac{1}{9},\frac{2}{9})$、$(\frac{7}{9},\frac{8}{9})$，無窮的依此步驟將現有的每一線段的三分之一等份的中間段開區間挖掉，最後所成的集合即為 Cantor Set。

2. **幾何圖形分析：**

藉由觀察圖形的結果可知，每遞迴一次需要知道的座標點為前一次的 2 倍（因為將每條線段切割成 3 部分，取左邊及右邊的部分），而我們所要知道的線段長度也為上一次遞迴時的$\frac{1}{3}$。

3. **數學函式推導：**

同樣的，在這個步驟當中必須分析出如何找到需要的座標點的方法。假設第一條線的起始點座標為 a，線段長度為 L，可以得到遞迴一到四次的結果，如圖 6。

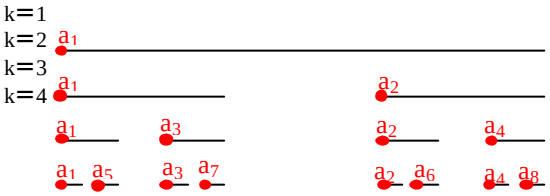


圖 6 遞迴一到四次的 Cantor Set（含座標點）

而此圖中的始點個數與座標、線段長度的分析如表 1。

遞回次數	始點個數	始點座標	線段長度
k=1	2^0 (a_1)	$a_1=a$	$L_1=L$
k=2	2^1 (a_1、a_2)	a_1同上 $a_2=a_1+2L_2$	$L_2=\frac{1}{3}L_1$
k=3	2^2 (a_1、a_2、a_3、a_4)	a_1、a_2同上 $a_3=a_1+2L_3$ $a_4=a_2+2L_3$	$L_3=\frac{1}{3}L_2$
k=4	2^3 (a_1、a_2、a_3、a_4、a_5、a_6、a_7、a_8)	a_1、a_2、a_3、a_4同上 $a_5=a_1+2L_4$ $a_6=a_2+2L_4$ $a_7=a_3+2L_4$ $a_8=a_4+2L_4$	$L_4=\frac{1}{3}L_3$

表 1 遞迴一到四次的概念

4. **細分法則演算法：**

經過了上面三個階段，接著把這些分析與架構套入演算法當中。

1. 宣告二維陣列存放始點及終點的 x 軸坐標
2. 當 k=1 時，以始點的 x 軸座標 a、終點的 x 座標 a+L 繪出第一條線
3. 當 k＝2 到 n 時，

將線段長設為原長的三分之一（$L=\frac{L}{3}$），並且把上次遞迴的每一始點依下列方式處理：

a. 以原來這一點為起點，起點＋L 為終點畫出左邊線段。

b. 以原來此點＋2L 為起點，起點＋L 為終點畫出右邊線段。

3.3 二維線段細分法則(2D line subdivision method)

本論文以寇茲雪花圖（Koch Snowflake）作為二維線段細分法則的代表，以下是根據轉換模型將 Koch Snowflake 轉換為 SVG 的方法。

1. **觀察圖形：**

寇茲雪花圖（Koch Snowflake）是應用 Fractal 的自我描述性概念所形成的。首先給定一正三角形，切掉每邊的三等份的中間段，並以此段為底邊向外做一正三角形，此為遞迴一次的製作步驟。對現有的圖形的每一邊做一次製作步驟的處理，且無窮的重複下去，最後得到的圖形即為有名的寇茲雪花圖。圖 7 為遞回一次到四次的圖形：

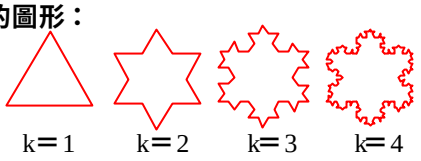


圖 7 Koch Snowflake 遞回一到四次的情形

2. **幾何圖形分析：**

藉由觀察圖形可以知道， Koch Snowflake 的原始圖形有三個頂點，並且必須把三個頂點連起來。遞迴第二次時每邊會變成需要四個頂點，所以整個圖形需要十二個頂點，線段長度為原來的$\frac{1}{3}$，遞迴 n 次的結果以此類推。所以每遞迴一次的頂點數都是上一次的 4 倍，而所需要的線段長度也為上一次遞迴時的$\frac{1}{3}$。

3. **數學函式推導：**

根據上面的分析，將遞迴一到二次的頂點存放方式以圖 8 表示。由於連線方式是以頂點的編號依序連結，所以必須設最後一點座標與原點相同，才能順利的連成圖形。

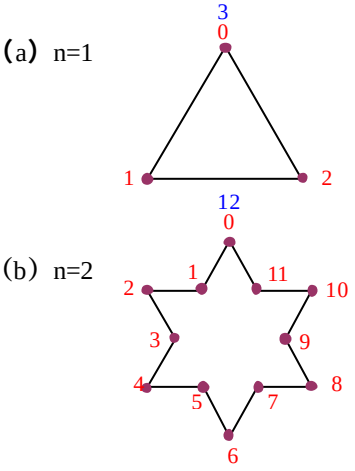


圖 8 Koch Snowflake 頂點存放方式

而在每次遞迴的過程當中，要從某點找尋到相關的四個頂點，可以按照以下的方法來求得。首先將頂點分為三組：

a.由已知端點加上原線段長的$\frac{1}{3}$向量即可求出的點，如圖 9 中的編號 1.5.9。

b.由已知端點加上原線段長的$\frac{2}{3}$向量即可求出的點，如圖 9 中的編號 3.7.11。

c.由已知端點兩兩求出中點，再加上中間向量才能求出的點，如圖 9 中的編號 2.6.10。

方法如下： 以點 2 為例，由點 1 與點 3 可求出中間向量 a，a 與 b 垂直可求出向量 b，由點 1 與點 3 也可求出中點 P，而 P+b 可得到點 2 之座標，同理可求出 6.10 的座標。若遞回更多次座標點求法相同，最後依照編號的順序用線段連起來。

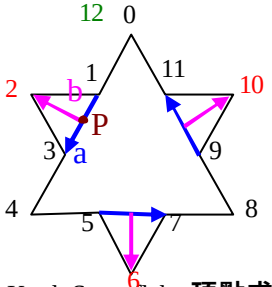


圖 9 Koch Snowflake 頂點求法

4. **細分法則演算法：**

根據上面的分析與架構所延伸出的演算法如下:

1. 宣告三個陣列 B 存放頂點坐標 （x, y)、X 暫存中點與 Y 暫存中間向量

2. k=1 時， 定位上頂點（a, b)、左頂點（$a-\frac{L}{2}$，$b+\frac{\sqrt{3}L}{2}$)、右頂點（$a+\frac{L}{2}$，$b+\frac{\sqrt{3}L}{2}$）

並且將最後一點（第四點）設為與第一點座標相同，連成第一個三角形。

3. k=2 到 n 時，

將最後一點（$B_{3*4^{k-1}+1}$）設為與第一點（B_0）座標相同，線段長設為原線段的三分之一（$L=\frac{L}{3}$），並將點分為三組，遞迴求出：

a. 由已知端點加上原線段長的$\frac{1}{3}$向量即可求出的點，如圖 3-5 中的編號 1.5.9。

$BX_{4*i+1}=(BX_{4*i}*2)/3+BX_{4*(i+1)}/3;$
$BY_{4*i+1}=(BY_{4*i}*2)/3+BY_{4*(i+1)}/3;$

b. 由已知端點加上原線段長的$\frac{2}{3}$向量即可求出的點，如圖 3-5 中的編號 3.7.11。

$BX_{4*i+3}=BX_{4*i}/3+(BX_{4*(i+1)}*2)/3;$
$BY_{4}*_{i+3}=BY_{4*i}/3+(BY_{4*(i+1)}*2)/3;$

c. 由已知端點兩兩求出中點，再加上中間向量才能求出的點，如圖 3-5 中 2.6.10。

$XX_{4*i+2}=(BX_{4*i+1}+BX_{4*i+3})/2;$
$XY_{4*i+2}=(BY_{4*i+1}+BY_{4*i+3})/2;$
$YX_{4*i+2}=BX_{4*i+3}-BX_{4*i}+1;$
$YY_{4*i+2}=BY_{4*i+3}-BY_{4*i}+1;$
$BX_{4*i+2}=XX_{4*i+2}-(YY_{4*i+2}*0.866);$
$BY_{4*i+2}=XY_{4*i+2}+(YX_{4*i+2}*0.866);$

最後依照編號的順序用線段連起來。

3.4 二維區域細分法則（2D area subdivision method)

Sierpinski gasket 為二維區域細分法則的代表，以下是根據轉換模型將 Sierpinski gasket 轉換為 SVG 的方法。

1. **觀察圖形：**

Sierpinski gasket 這個有名的 Fractal 圖形製作的方式如圖 10。首先，給定一正三角形，取各邊中點，挖掉中間那塊正三角形(其邊界需留下)，剩下三個相同的正三角形，接下來對剩下的每個三角形做同樣的步驟，挖掉其各邊中點連線所成正三角形，重複此步驟無窮次，即形成 Sierpinski Triangle。

2. **幾何圖形分析：**

藉由觀察圖形的結果，Sierpinski gasket 的起始圖形為一正三角形，所以若以三角形為一個物件單位，則需要知道的頂點只有一個。遞迴一次時，取各邊中點，挖掉中間那塊正三角形，剩下三個相同的正三角形。所以需要的頂點個數為三個，並畫出三個小三角形，邊長為原來的一半，若遞迴 n 次則以此類推。其中每遞迴一次的頂點數都是上一次的三倍，而所需要知道的線段長度也為上一次遞迴時的一半。

3.**數學函式推導：**

根據上述討論，假設初始的正三角形頂點座標為（a, b)，而三角形物件的左頂點以及右頂點可由上面三項使用者輸入的變數得出：左頂點座標為（a-$\frac{1}{2}$L, b+$\frac{\sqrt{3}}{2}$L)、右頂點座標為（a+$\frac{1}{2}$L, b+$\frac{\sqrt{3}}{2}$L)。因為在 Sierpinski gasket 中所繪製的三角形為正三角形，所以只要知道三角形的上頂點，便可以計算出左頂點及右頂點，所以在演算法的基本概念中，只需找出三角形的上頂點。表 2 為 Sierpinski gasket 遞迴一到四次的基本概念，圖 10 Sierpinski gasket 演算法頂點存放方式，其餘遞迴次數以此類推：

遞回次數	始點個數	始點座標	線段長度
n=1	3^0 (a_1, b_1)	$a_1=a$、$b_1=b$	$L_1=L$
n=2	3^1 (a_1, b_1)，(a_2, b_2)，(a_3, b_3)	a_1同上 $a_2=a_1-L_2/2$，$b_2=b_1+\sqrt{3}L_2/2$ $a_3=a_1+L_2/2$，$b_3=b_1+\sqrt{3}L_2/2$	$L_2=\frac{1}{2}L_1$
n=3	3^2 (a_1, b_1)，(a_2, b_2)，(a_3, b_3)，(a_4, b_4)，(a_5, b_5)，(a_6, b_6)，(a_7, b_7)，(a_8, b_8)，(a_9, b_9)	a_1、a_2、a_3同上 $a_4=a_1-L_3/2$，$b_4=b_1+\sqrt{3}L_3/2$ $a_7=a_1+L_3/2$，$b_7=b_1+\sqrt{3}L_3/2$ $a_5=a_2-L_3/2$，$b_5=b_2+\sqrt{3}L_3/2$ $a_8=a_2+L_3/2$，$b_8=b_2+\sqrt{3}L_3/2$ $a_6=a_3-L_3/2$，$b_5=b_3+\sqrt{3}L_3/2$ $a_9=a_3+L_3/2$，$b_9=b_3+\sqrt{3}L_3/2$	$L_3=\frac{1}{2}L_2$

表 2 Sierpinski gasket 遞迴一到四次的基本概念

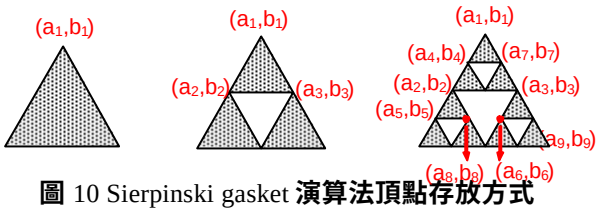


圖 10 Sierpinski gasket 演算法頂點存放方式

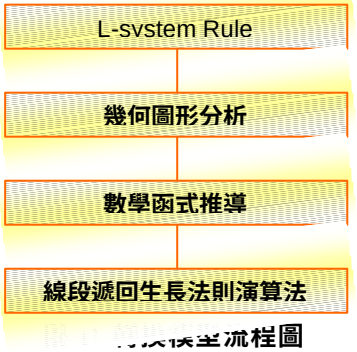
4. **細分法則演算法**：

1. 宣告二維陣列 B 存放上頂點的 x座標及 y 座標
2. K=1 時，由原始的正三角形上頂點畫出第一個三角形
3. 當 K=2 到 n 時，每次邊長減半（$L=\frac{L}{2}$），並且藉由上一次遞回的每一個頂點及現在邊長依照下列方式找出左頂點及右頂點。最後可由左頂點：（上頂點－L/2，上頂點＋$\sqrt{3}$ L/2）繪出左三角形；右頂點：（上頂點＋L/2，上頂點＋$\sqrt{3}$ L/2）繪出右三角形。

四、林登梅爾系統呈現演算法

4.1 轉換模型（Transformation meta model）

林登梅爾系統是由典型的語言模型而來的，而 SVG 是一種使用標籤來繪製圖形的標籤語言。因此要使用 SVG 來呈現林登梅爾系統的圖形需要一個轉換的方法與過程，本論文稱其為「轉換模型」。轉換模型有圖 11 的四個步驟：



轉換模型流程圖

1. L-system Rule： L-system Rule 包含了幾項最重要的資訊：遞迴次數、偏轉角度、起始符號以及生長的規則。其生長方式是藉由生長規則得到一個起始樹型之後，遞迴第二次時以起始樹型放大後當作下一次遞迴的骨架，而放大後的每一根樹枝構造與起始樹型相同，在本論文中，稱其為線段遞迴生長法則（segment Recursive grow method）。在 2 維空間中的 L-system 樹型，我們稱其法則為 2D segment Recursive grow method；在 3 維空間中的 L-system 樹型，稱其法則為 3D segment Recursive grow method。
2. **幾何圖形分析**：在得到 L-system Rule 之後便要由 SVG 的觀點來探討。已知 SVG 的圖形是由絕對或相對座標點來決定圖形的位置，所以在這一階段當中，必須分析出這個 L-system 樹型要用 SVG 來表示時需要知道的坐標點數量及樹枝長度。
3. **數學函式推導**：在分析完了幾何圖形之後，接著必須討論這些座標點之間有何關連性及如何用數學函式來求出這些座標點。
4. **線段遞回法則演算法**：經過分析與推導之後，可以將導出來的三角函數與遞迴概念套入演算法當中，實際由程式去執行，即可將 L-system 以 SVG 的方式來呈現。

4.2 柳樹樹型的二維線段遞回生長法則

（2D segment Recursive grow method— Willow Type）

1. L-system Rule：

本論文設計了一個樹型規則，這個樹型的規則如表 3，遞迴一到三次的圖形如圖 12，我們稱其為柳樹樹型（willow type）：

說明	L-system 語法
遞迴次數	1
旋轉角度	15
樹枝寬度	2
起始符號	F
規則	F= FF[---F+F][-F-F-F]
結束符號	@

表 3 L-system 的柳樹樹型語法

圖 12 柳樹樹型遞迴一到三次

2. **幾何圖形分析**：首先畫出這個 L-system 柳樹樹型的起始圖形。分析過後得知，每遞迴一次所需要知道的座標點數目為上一次遞迴的七倍。如圖 13(a)

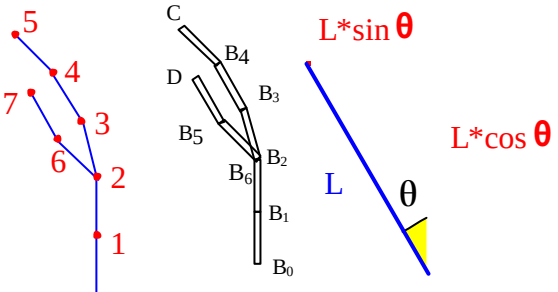


圖 13 柳樹樹型的數學理論圖解

3. **數學函式推導**：

由於柳樹樹型有幾項可控制的變數，例如：遞回的次數、起點座標、樹枝的長度以及樹枝張開的角度；因此我們也提供了這四樣的變數可供調整：遞回的次數（n）、起點座標（a,b）、樹枝的長度（L）以及樹枝張開的角度（q）。

由圖 13（c）可知，若樹枝往左邊偏轉 q 角時，偏轉後長度為 L 的樹枝之座標頂點（x,y）可以由下列公式求出：x=x－ L*sin q ；y=y－L*cos q 。

而圖 13（b）為此柳樹樹型形成所需的座標點，敘述如下：

B_0 為始點，所以座標即為（a,b）

$B_1 = B_0$ －（0,L）……（公式 4.2.1）

$B_2 = B_1$ －（0,L）……（公式 4.2.2）

$B_3 = B_2$ －（L*sin θ , L*cos θ）……（公式 4.2.3）

$B_4 = B_3$ －（L*sin 2* θ , L*cos 2* θ）（公式 4.2.4）

C = B_4 －（L*sin 3* θ , L*cos 3* θ）（公式 4.2.5）

$B_6 = B_2$ ……（公式 4.2.6）

$B_5 = B_2$ －（L*sin 3* θ , L*cos 3* θ）（公式 4.2.7）

D = B_6 －（L*sin 2* θ , L*cos 2* θ）（公式 4.2.8）

接著進行遞回的動作，每一枝樹枝長度都將為原來的兩倍。但由於 C、D 兩點都不用再長出樹枝，所以存在另外的陣列中。接著只看編號為 B 的點：當遞回第一次時總共有 7 個頂點，遞回第二次時總共有 7^2 個頂點，遞回第三次時總共有 7^3 個頂點。

4. **線段遞迴生長法則演算法：**

1. 首先宣告 B、C、D 三個主要陣列：B 陣列存放數學基礎中與 B 有關的點、C 陣列存放數學基礎中與 C 有關的點、D 陣列存放數學基礎中與 D 有關的點。

2. 當遞回次數為 1 時，則依據上面的概念求出 C、D 及 B_0 到 B_6 的座標，再將 B_0 到 B_6 依順序連起來，再連 B_4 到 C 及 B_5 到 D，即連成第一株柳樹。

3. 當遞迴次數為 2　n 時，

將所有已知與 B 有關的點座標都存到陣列中 7 倍的位置，例如：當遞迴次數為 2 時，$B_0 \rightarrow B_0$；$B_1 \rightarrow B_7$；$B_2 \rightarrow B_{14}$；$B_3 \rightarrow B_{21}$……；當遞迴第三次時，再把第二次的每一點移到 7 倍的位置…..以此類推。

最後根據相同的方式把每一枝樹枝的其他頂點補齊，再將與 B 有關的點依序連結，最後 C_i 與 B_{4+7i} 連結，D_i 與 B_{5+7i} 連結。

4.3 基本樹型的三維線段遞回生長法則

（3D segment Recursive grow method—Standard Type）

1. L-system Rule：

根據 4.2 的樹型概念，本論文將 2D 的基本樹型推到 3D 的環境當中，稱其為 3D segment Recursive grow method—Standard Type。而在 3D 基本樹型的結構中，我們讓主要的樹枝生長的方向除了能讓其向右及向左偏轉一個角度（θ）外，也讓樹枝可以向前及向後偏轉另一個角度（φ），如圖 14 所示。

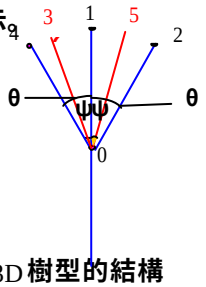


圖 14 3D 樹型的結構

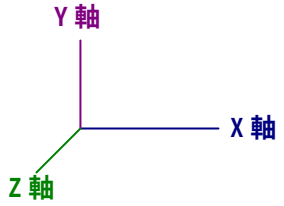
2. **幾何圖形分析：**

首先畫出這個 3D L-system 基本樹型的起始圖形。分析過後得知，每遞迴一次所需要知道的座標點數目為上一次遞迴的 6 倍。

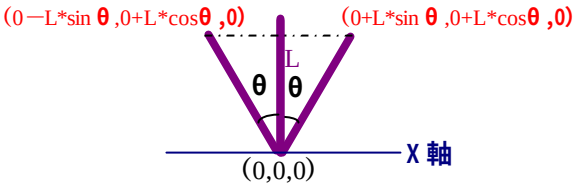
3. **數學函式推導：**

而在 3D 樹型當中，最主要提供給使用者改變的參數就是遞迴的次數（n）與改變的角度 θ 與 φ。

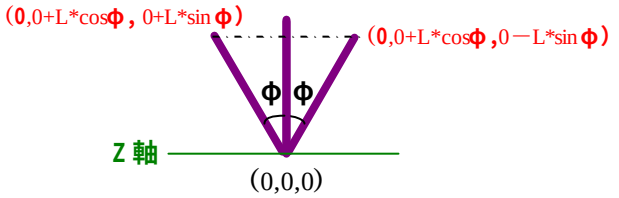
a. 環境中的座標軸如下：



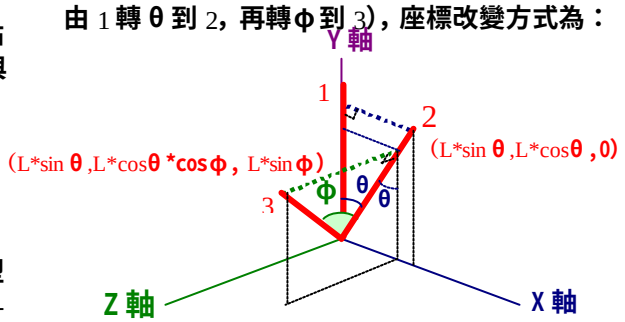
假設『右方』為『X 軸的正向』時，樹枝向左右兩方偏轉 θ 時，座標改變的方式為：



b. 由 a.可知，若樹枝向前後兩方偏轉 φ 時，座標改變的方式為：



c. 若是樹枝向 θ 及 φ 都有偏轉的方向時（樹枝由 1 轉 θ 到 2，再轉 φ 到 3），座標改變方式為：



不論是 θ 和 φ 旋轉的順序如何，道理都是一樣的。

4. **線段遞迴生長法則演算法：**

1. 先宣告一個二維陣列來存放頂點的位置，陣列的大小為 $6^n \times 5$（5 欄分別存放點的 X、Y、Z 座標以及旋轉的 θ 角和 φ 角。
2. 設定好起點與主幹。
3. 建立迴圈遞迴 1 到 n 次，
   每次樹枝長度減半。
   再建立一個子迴圈，遞迴次數為 1 到上一次做完的總頂點數：
   依照上面的分析將頂點本身定位後，再依下列公式找出折半點。

$X_{mk+5*i} = X_i + L \sin(A_i \theta)$；
$Y_{mk+5*i} = Y_i - L \cos(A_i \theta) \cos(B_i \phi)$；
$Z_{mk+5*i} = Z_i + L \sin(B_i \phi)$；
$A_{mk+5*i} = A_i$；
$B_{mk+5*i} = B_i$；

再建立一個子迴圈，遞迴次數為 1 4（找出折半點向右、後、左、前再長出的分枝的頂點），用到的方法如下：

遞迴次數	計算方式	結果
1	$\sin(1*\frac{p}{2})=1$；$\cos(1*\frac{p}{2})=0$	畫出右枝，將θ方向角度加一倍
2	$\sin(2*\frac{p}{2})=0$；$\cos(2*\frac{p}{2})=-1$	畫出後枝，將φ方向角度減一倍
3	$\sin(3*\frac{p}{2})=-1$；$\cos(3*\frac{p}{2})=0$	畫出左枝，將θ方向角度減一倍
4	$\sin(4*\frac{p}{2})=0$；$\cos(4*\frac{p}{2})=1$	畫出前枝，將φ方向角度加一倍

$X_{mk+5*i+j} = X_i + L\sin((A_i + \sin(j*p))\theta)$;
$Y_{mk+5*i+j} = Y_i - L\cos((A_i + \sin(j*p))\theta)\cos((B_i + \cos(j*p))\phi)$;
$Z_{mk+5*i+j} = Z_i + L\sin((B_i + \cos(j*p))\phi)$;
$A_{mk+5*i+j} = A_i + \sin(j*p)$;
$B_{mk+5*i+j} = B_i + \cos(j*p)$;

再將 i 點與 mk+5*i+j 點連起來。

4.3.1 柱狀樹枝

由於前面的 2D tree 呈現出來的是平面的長方形樹枝，而目前在 3D 的環境當中，我們的樹枝都是顯示骨架的線條，若要是樹型在 3D 環境中更加的真實，我們藉由下面的方法將樹枝改以六角柱代替。

1. **先由底面六角形的中間點找出六角形的六個點。**
2. **當樹枝為直立的時候，經由平移 y 軸（即向上平移）找到六角柱的上平面之六角形的六個座標點。畫出六角柱的上平面、底面以及側面的六個長方形，以形成六角柱。**
3. **當六角柱生成分枝時，使用旋轉變換公式：**
   $x = x' \cos\theta - y' \sin\theta$
   $y = x' \sin\theta - y' \cos\theta$

五、圖形呈現效能與分析

5.1 Fractal 與 L-system 範例整合

第三節與第四節中提出了轉換模型的概念與 subdivision method、segment recursive grow method 等方法。在這一節中藉由表 4 對 Fractal、L-system 與 SVG 的特性作比較。

藉由表 4 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 是依照原有個骨架越長越大。

	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 4 Fractal、L-system 與 SVG 特性比較

5.2. SVG 與 Fractal 的圖形呈現效能比較

在本節當中直接就實做出來的結果作一個統整的分析，一般看到 Fractal 的教學都以圖片來呈現，因此就各種圖檔的類型與 SVG 做一個比較。

1. **『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. **由於網路常用的圖檔格式都還是以 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. **GIF 是採用色盤的壓縮方式，而本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. **SVG 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；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，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，但是物件數量比 Koch Snowflake 少很多，所以檔案也小很多。**

針對 Fractal 圖形來說，Fractal 是有規則性的。由前面可知，Fractal 的自我描述性都可以用程式中的迴圈來達成。若根據 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。這也可以克服 SVG 由於圖形內含有的標籤越多，檔案便越大的缺點。

5.3 SVG 與 L-system 的圖形呈現效能比較

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。在 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。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……）。若是要更有立體感，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……）。 3Dtree 遞回一到五次的物件個數如下表 5 所示，物件數目的龐大由此可知。

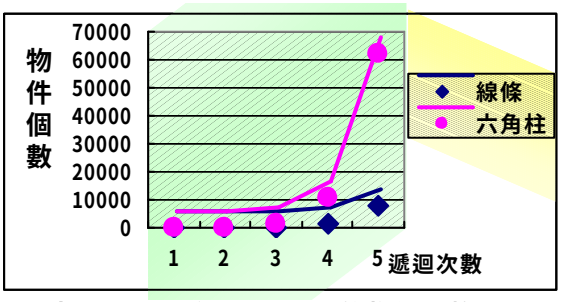


表 5 3Dtree 遞回一到五次的物件個數

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，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，雖然效能與物件的數量與計算有關，但也已經方便許多，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。

5.4 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，其中轉換模型與法則的概念是依照 Fractal 與 L-system 的特色所歸納出來藉以提高傳輸的效率，而本文依照各個不同類型的 Fractal 與 L-system 也實做了各項的範例，若是欲實做其他的函數可依照轉換模型與法則修改演算法。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。這只有 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。

5.5 未來發展

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。對於 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。

參考文獻

[1] E. Angel. Working with Models. Interactive computer graphics, a top-down approach with OpenGL, Pages 303-340, Addison Wesley, 1997.

[2] F.S Hill, JR. “Approaches to Infinity. Computer Graphics Using OpenGL”, Second Edition, Pages 477-478, Prentice-Hall, 2001.

[3] Yao-Ming Yeh, Kuan-Sheng Lee. A XML-based Plant Modeling Language. Cvgip2001.

[4] W3C Candidate Recommendation, " Scalable Vector Graphics (SVG) 1.0 Specification" HTTP DOC, November 2000. <<http://www.w3.org/TR/2000/CR-SVG-20001102/index.html>>