國立暨南國際大學資訊工程學系

碩士論文

L-Grammar 之文法產生器

L-Grammar Generator

指導教授：陳履恆 博士

研究生：黃俊育

中華民國九十四年七月十六日

致謝

我非常感謝我的指導教授陳履恆博士,感謝老師從我大學到研究所對我的教導與照顧,讓我在學業與待人處世都受益良多。縱使我研究所延畢了一年,老師也願意以包容的心不辭辛勞的指導我,甚至老師也願意犧牲自己的休息時間來督促我。

我也感謝我的家人,多虧有父母的彈藥補給,讓我能不需要煩惱三餐的專心做研究,雖然多留一年難免讓你們唸了一下,可是你們還是願意支持與體諒我,才能讓我順利完成學業。

感謝 bird 學長,當我程式方面有問題的時候,你總是能夠適時的幫我一把,感謝小黑不辭辛勞的幫我拉了許多葉子的 model,感謝 Taco、小包、fishman、蛋蛋、winner、阿富、沛宇、襄安、Jimmy、翁爺、蕭大俠、劉明進、獼猴、辣的、笑雲、叮噹、小馬達、家禾以及 m3lab 的全體成員,感謝這些夥伴們平日的幫助,當我心情煩悶的時候,你們的搞笑總是能帶給我精神,讓我從低潮中走出來,除了感謝之外,我也想說,在我研究所的旅程上有你們真好。

論文名稱：L-Grammar 之文法產生器
校院系：國立暨南國際大學資訊工程學系　　頁數：31
畢業時間：中華民國 94 年 7 月　　學位別：碩士
研究生：黃俊育　　指導教授：陳履恆 博士

論文摘要

在植物的 modeling 方面，有二個主要的考量：一、植物學上的正確性，二、電腦圖學上擬真效果。而在模擬植物的生長中，L-System 一直是個主流的手法，最主要的原因在於 L-System 可以把植物的生成特性，整合進 formal language 的 self-developing grammars 中，使得由 L-System 所產生的字串具有植物學上的正確性。可是 L-System 是藉著文法來模擬植物的生長，而目前文法這部分都要靠人工來撰寫，這樣不只費時，而且錯誤率高，對於無經驗的使用者而言，要學習如何撰寫這些文法更是難上加難。

因此，我們的研究目的是研發一些輔助的工具來產生文法，除了讓使用者能透過此系統方便的產生植物文法，也希望讓無經驗的使用者不需經過複雜的文法學習訓練，就能夠透過此系統產生出夠逼真的植物影像。

我們的方法分成四個部分，第一部分是從植物學上的生長敘述找出適當的關鍵字。第二部分是從植物影像來建立其三維骨架(3D skeleton)，我們的做法是先取得三張分別由彼此正交的正面、側面以及頂視圖三個方向的照片，在系統載入圖片之後，分別手動選取其相互對應的 stem，系統會從這些 stem 的資料產生該植物的三維骨架。第三部分則是從三維骨架計算得到我們所需的量化數據。最後則是將上述步驟得到的量化資訊轉換成初始狀態的 L-grammar，並且從初始狀態的文法整理出一些規則，進而產生出預測狀態的 L-grammar。

關鍵字：L-System，L-grammar，關鍵字，量化資訊

Title of Thesis：L-Grammar Generator

Name of Institute：National Chi Nan University, Dept. of CSIE Pages：31

Graduation Time：July 2005 Degree Conferred：Master

Student Name：Jiun-Yu Huang Advisor Name：Dr. Lieu-Hen Chen

Abstract

There are two main considerations in modeling of plants: one is to preserve the accuracy of botany, the other is to render the photorealistic images of computer graphics. L-System is the mainstream method of simulating the growth of plants. The reason is that the L-System is able to simulate the botanic properties of plants by integrating with the self-developing grammars of formal language. However, grammars are written manually, it may cause many errors and is very time consuming. It is very difficult for a user with no experience to learn how to writing L-grammar.

The research purpose of this paper is to design a CAD system to help users to produce grammars easily and conveniently. We also hope that inexperienced users could use our system to get photorealistic plant images without complicated training.

There are four steps in our method. First step is finding keywords from growth description of plants. Second step section is reconstructing 3D skeleton by selecting corresponding stems among photos. Third step section is calculating quantity information from 3D skeleton. The last step is transforming quantity information into L-grammar.

Keywords: L-System, L-grammar, keyword, quantification

目錄

圖片目錄

第一章 緒論

1.1 研究動機與目的

L-System 是藉著文法來模擬植物的生長，但是為了加強 L-System 的表現能力，會導致文法過度複雜化，複雜的文法使得可讀性降低，造成文法難以使用及學習。而目前文法都要靠人工來撰寫，並且要不斷地修正文法與參數才能產生逼真的植物影像，這樣不只費時，而且錯誤率高。對於無經驗的使用者而言，他們必須先了解 formal language 的相關知識，才有能力去撰寫文法，要使他們寫出令人滿意的文法更是難上加難。

在近年來有關 L-System 的論文中，我們看到許多有關改進 L-System 文法本身的研究成果，但是，對於 L-System 在產生文法的困難點上，始終沒有好的解答。因此，我們的研究目的是研發一些輔助的工具來產生文法，除了讓使用者能透過此系統方便的產生植物文法，也希望讓無經驗的使用者不需經過複雜的文法學習訓練，就能夠透過此系統產生出夠逼真的植物影像。

有鑒於網路上大量的植物資料是以文字敘述的方式存在著，這些文字敘述裡面包含了葉序、葉形、草本或灌木這一類的特徵關鍵字，而我們就是透過這些特徵來辨認植物的種類，所以我們認為找出生長敘述中特定的關鍵字將有助於我們建立 L-grammar。在撰寫文法時，需要大量的植物量化資料，例如葉子大小、莖的長短、粗細以及夾角等等的數據，但是從植物的生長描述當中，可以取得的量化資訊十分有限，所以我們的想法就是可以從植物的影像中找這一些量化資訊，來彌補文字敘述的不足。基於上述理由，我們的系統將以目標植物的文字敘述與植物影像當成輸入的資料來轉換成 L-grammar。

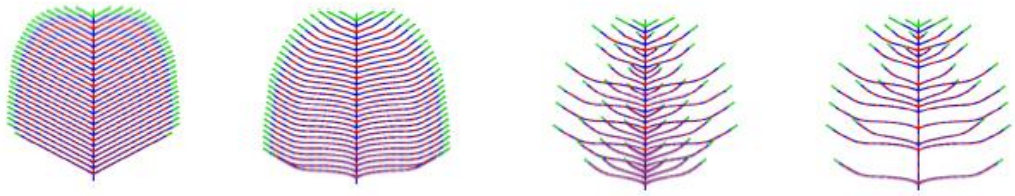
1.2 論文編排方式

本篇論文的第二章會探討相關研究，第三章介紹本研究的系統架構，第四章是研究成果與未來展望。第五章是參考文獻。

第二章 相關研究

因為 L-System 可以把植物的生成特性，整合進 formal language 的 self-developing grammars 中，使得由 L-System 所產生的字串具有植物學上的正確性，所以自從 L-System 被提出之後，就一直是植物 modeling 的主流手法。在 1995 年發表的 The Artificial Life of Plants[4]論文當中，作者提出的 parametric L-System 對於 L-System 來說是個重大的進展，parametric L-System 的手法是藉由在 L-System 的符號中加入許多數字屬性以達成過去做不到的控制，可是相對的卻也造成了文法複雜度的提升以及可讀性的下降。

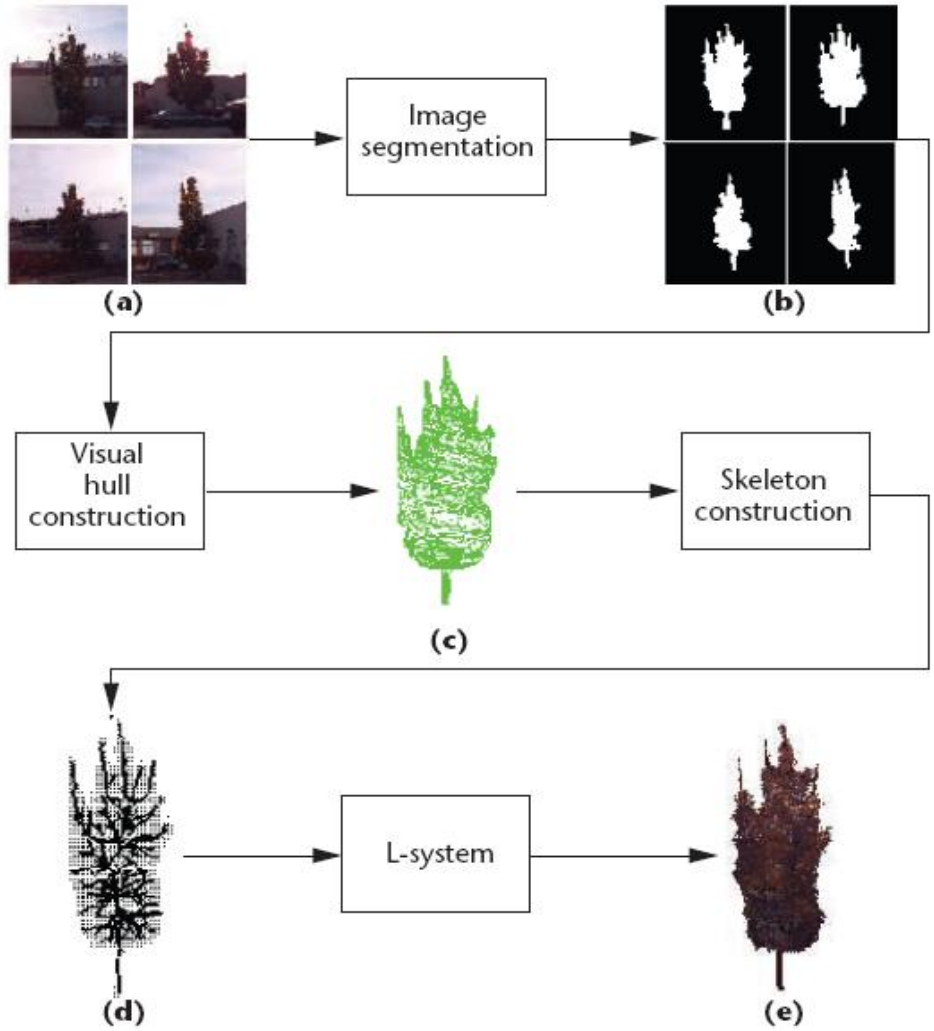
Callum Galbraith發表的Implicit Visualization and Inverse Modeling of Growing Trees [3]提出用hierarchical implicit modeling system的方法來處理生長點以及分枝造成的non-smooth features，透過global to local的inverse modeling方法來建立北美白楊樹(Populus deltoides)的model，並且產生出很好的成果影像，不過他們只針對北美白楊樹這種植物來建立model，而我們的目標則放在能對各種植物來建立model。



圖(1)：建立北美白揚木 model 的過程(圖片引用自 Implicit Visualization and Inverse Modeling of Growing Trees[3])

Ilya Shlyakhter發表的Reconstructing 3D tree models from instrumented photographs [5]提出從植物照片來重建植物model的研究，他們的方法是從植物影像的silhouette建立visual hull，接下來從visual hull建立skeleton並且在上面設定葉子的生長點，最後透

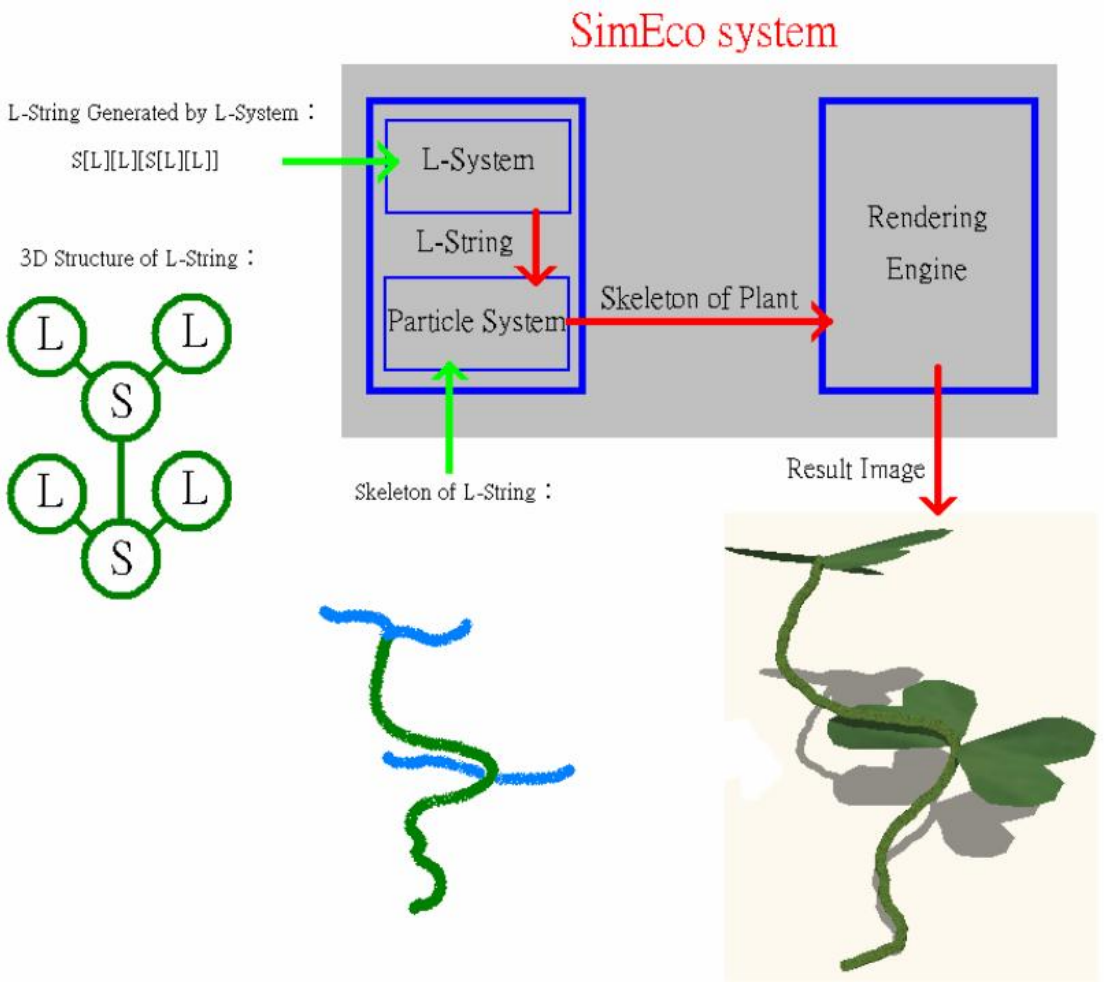
過L-System的方法在生長點長出葉子，跟本研究不同的地方在於他們只有在葉子的部分才有用到L-System，而我們則是著重於產生L-grammar，並透過L-System來模擬出整個植物。



圖(2)：Reconstructing 3D tree models from instrumented photographs 的系統流程圖(圖片引用自 Reconstructing 3D tree models from instrumented photographs[3])

An Adaptive L-System [8]裡面的 SimEco 是整合 particle system 與 parametric L-System 的一套系統，透過紀錄 particle 的運動軌跡來當成植物生長的 skeleton，並

且能夠把環境對植物的影響直接反映在 particle 的運動軌跡上，進而把環境的影響直接反應在植物的形變上，本研究建構在 SimEco 之上，因此本研究產生的文法都是以 SimEco 的語法為基準。



圖(3)：SimEco 系統概觀(圖片引用自 An Adaptive L-System [8])

第三章 系統架構

我們的系統主要分成四個部分：

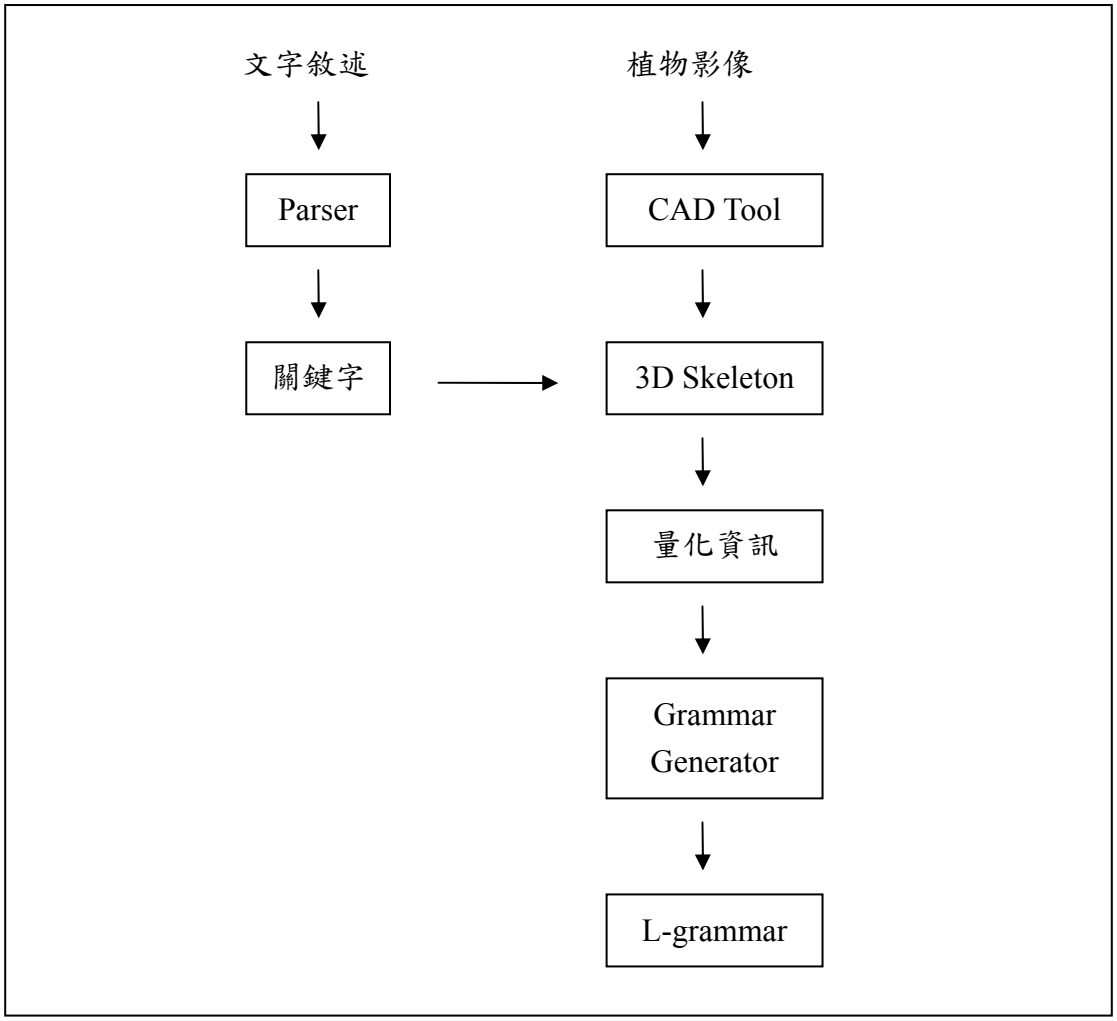
〈1〉 關鍵字擷取：從植物的文字敘述裡面取得我們所需的關鍵字。

〈2〉 建立植物三維骨架：從植物的影像透過使用者介面的互動來建立出該植物的 3D skeleton。

〈3〉 量化資訊擷取：從三維骨架計算出我們所需的量化資訊。

〈4〉 文法轉換：將上述步驟獲得的資訊整合並以 L-System 的文法格式輸出。

下圖(4)是我們的系統流程圖。



圖(4)：系統流程圖

3.1 關鍵字擷取

很多介紹植物的書籍或是網站，他們在介紹植物的時候，都會用一段文字敘述來描述該植物的特徵以及生長情況，而這些特徵就是我們在辨識植物上重要的依據，我們稱這些描述特徵以及生長情況的文字為“關鍵字”，見下圖(5)，這些關鍵字對我們建立 L-grammar 有很大的幫助，因此我們系統的第一步就是從這些文字敘述來取得我們所需要的關鍵字。

翅果鐵刀木

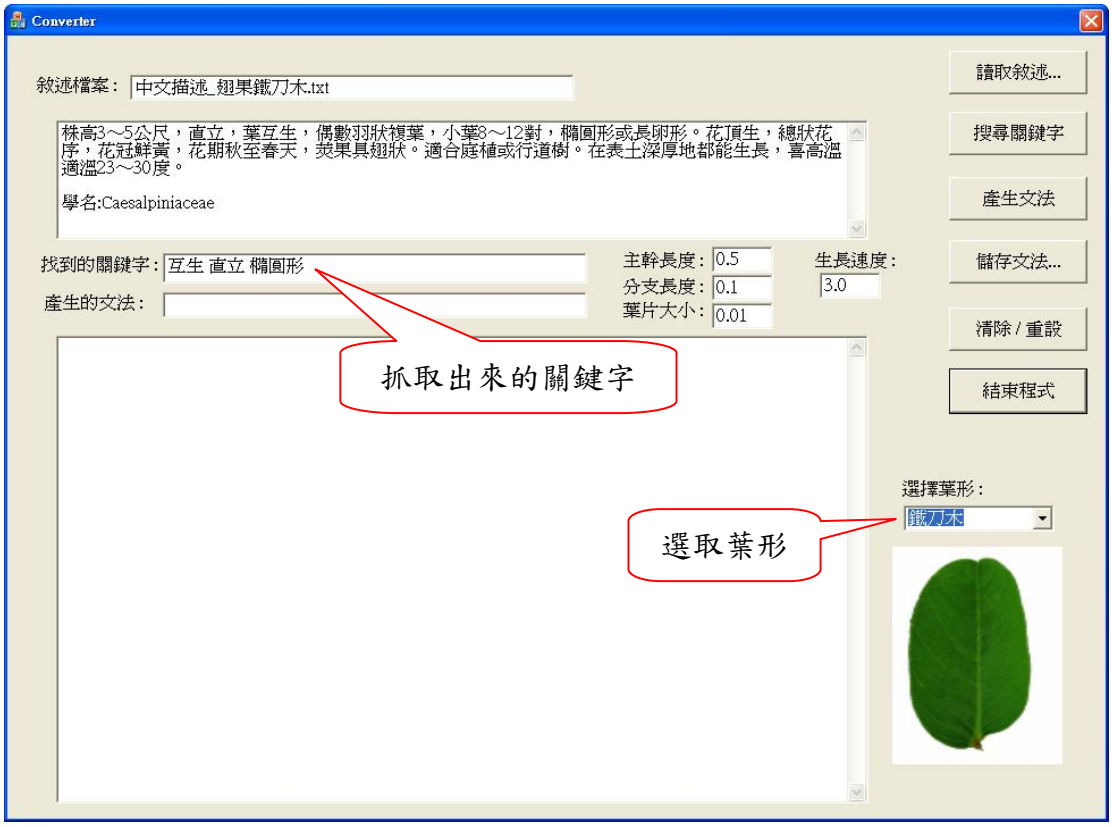
學名：Caesalpiniaceae

特徵：株高 3～5 公尺，直立，葉互生，偶數羽狀複葉，小葉 8～12 對，橢圓形或長卵形。花頂生，總狀花序，花冠鮮黃，花期秋至春天，莢果具翅狀。適合庭植或行道樹。在表土深厚地都能生長，喜高溫適溫 23～30 度。

關鍵字：直立、互生、橢圓形

圖(5)：翅果鐵刀木生長敘述與關鍵字

這部分我們的做法是事先定義好我們要處理的關鍵字，並將這些關鍵字整合成關鍵字表格，之後我們的系統在載入文字敘述之後，會執行字串分析比對的動作，把與關鍵字表格裡面相同的關鍵字找出來。接下來使用者必須選取葉形，系統會依照選取的葉形去找出相對應的葉子材質貼圖設定到系統裡面，這部分系統介面如圖(6)所示。



圖(6)：關鍵字擷取的系統介面

3.2 建立植物的三維骨架

在撰寫文法的時候，我們需要大量的植物量化數據，例如莖的長短、粗細、夾角、葉子大小以及植物高度等等諸如此類的數據，但是從植物的生長描述當中，可以得到的量化資訊十分有限，因此我們這個部分的主要目的是透過與使用者介面的互動，從植物影像中擷取出我們所需的量化資訊。這個部分在這裡又分成三個流程：

〈1〉 取得植物影像：取得符合條件的植物照片。

〈2〉 建構植物的 skeleton：透過使用者介面建立出植物的 skeleton。

〈3〉 設定葉序結構：設定部份參數來產生出目標植物的葉序結構。

接下來我們將分別詳述每個步驟的詳細做法。

3.2.1 取得植物影像

在 Ying Sun 發表的 Automated 3D reconstruction of tree-like structures from two orthogonal views [6]有提到如何從兩張正交的影像當中重建回樹狀物體的 3D model，因此我們必須取得植物的正視、右視與頂視三個方向的照片，而且三張照片的拍攝角度必須互相垂直。實際上只需要正視圖與右視圖兩張照片就能重建，頂視圖在我們的系統裡面當輔助之用，除非正視圖或右視圖有部分視點被遮蔽，這時候才需要頂視圖來當幫助使用者建立 skeleton，圖(7)是依照我們的條件所拍攝的植物影像。

(a) (b) (c)

圖(7)：鐵刀木的植物影像，(a)是正視圖，(b)是右視圖，(c)是頂視圖。

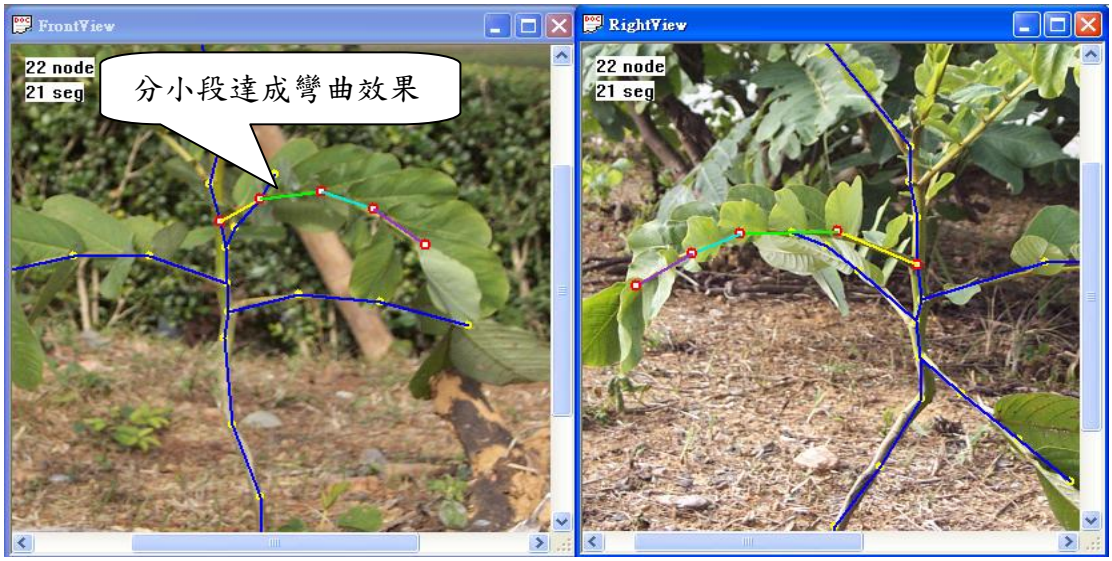
3.2.2 建立植物三維骨架

在 Automated 3D reconstruction of tree-like structures from two orthogonal views [6]也提到要先知道兩張影像中分枝(branch)的對應關係才能重建回唯一的 3D model，這個部分我們採取讓使用者手動選取植物影像中分枝的對應關係的方式。

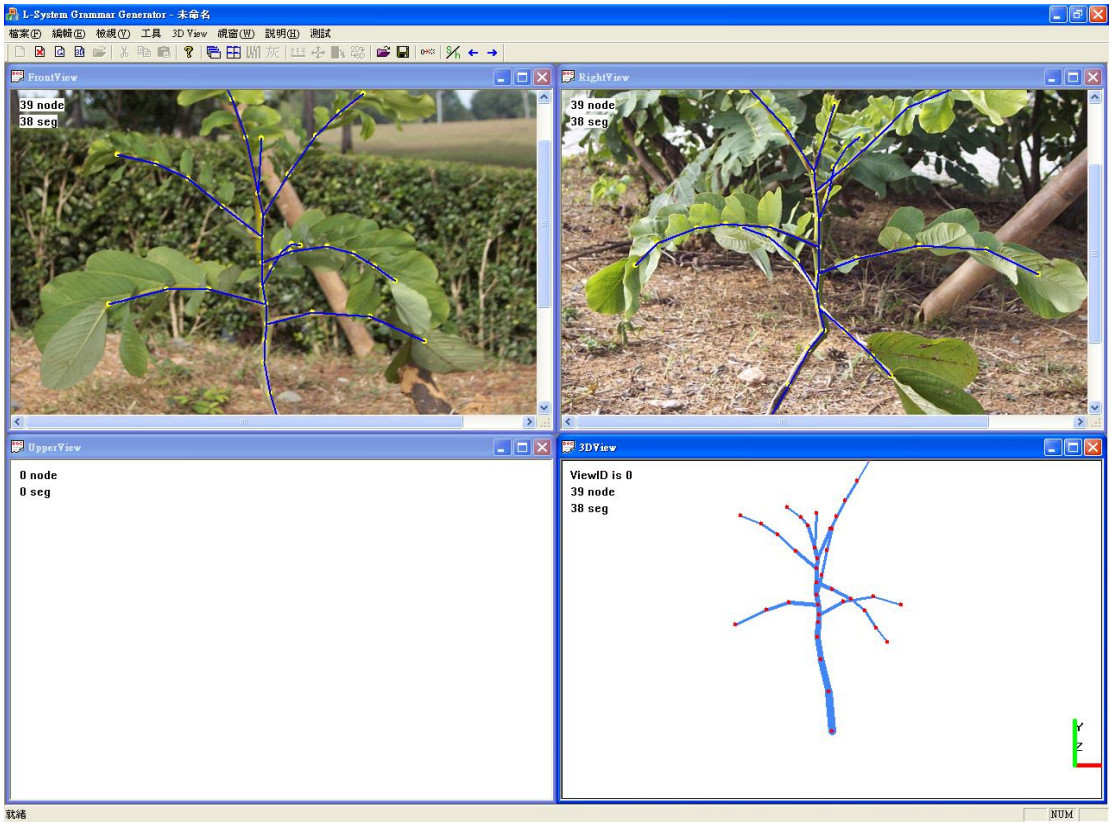
系統載入圖檔之後，使用者首先必須決定兩張圖片相對應的原點，也就是照片中植物的根部生長點，如圖(8)，之後選取的點資料在系統裡面會針對此原點作平移的動作，轉換成以根部生長點為原點的座標系統。接下來使用者必須以滑鼠點選的方式來選取圖片之間相對應的 stem，為了能夠達到植物彎曲的 stem 的效果，我們

採用分成小段的方式去模擬，如圖(9)，最後使用者必須手動設定 stem 的寬度。因為系統裡面會依照滑鼠選取的順序來當作對應的順序，所以使用者在選取對應 stem 的時候必須注意選取的順序是否對應正確，如圖(9)，另外還有一點限制是使用者選取出來的結構必須是樹狀結構，不能有循環結構出現，所以有經過人為外力影響的植物，例如觀賞用盆栽，是不在我們系統的處理對象之內。選取完成的結果如圖(10)所示。

圖(8)：將植物根部設定成原點



圖(9)：stem 選取說明圖。相同顏色表示相同順序。



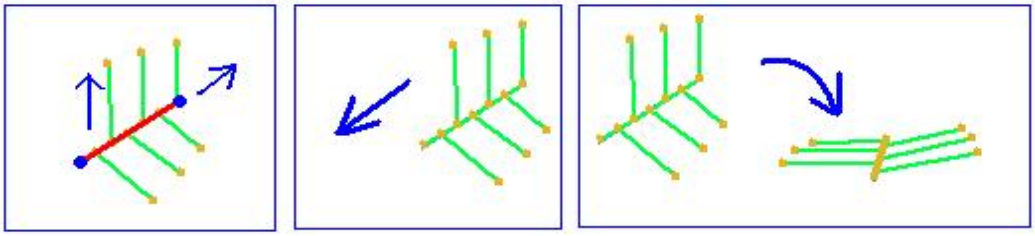
圖(10)：植物的三維骨架

3.2.3 設定葉序結構

到目前為止我們只得到植物莖幹與分枝的骨架結構，接下來我們必須在這個骨架上加上葉子，所以這個章節我們主要說明系統如何產生這些葉序結構。

我們的系統在這個部分採用關鍵字為主，參數設定為輔的方式來產生葉序結構。系統會取出 3.1 節得到的關鍵字當作主要的葉序，下一步系統會依照葉序的關鍵字來自動產生出該種葉序的基本 3D model，再配合上使用者給定的參數來對此基本葉序的 model 進行微調的動作，最後再附加到 skeleton 上。

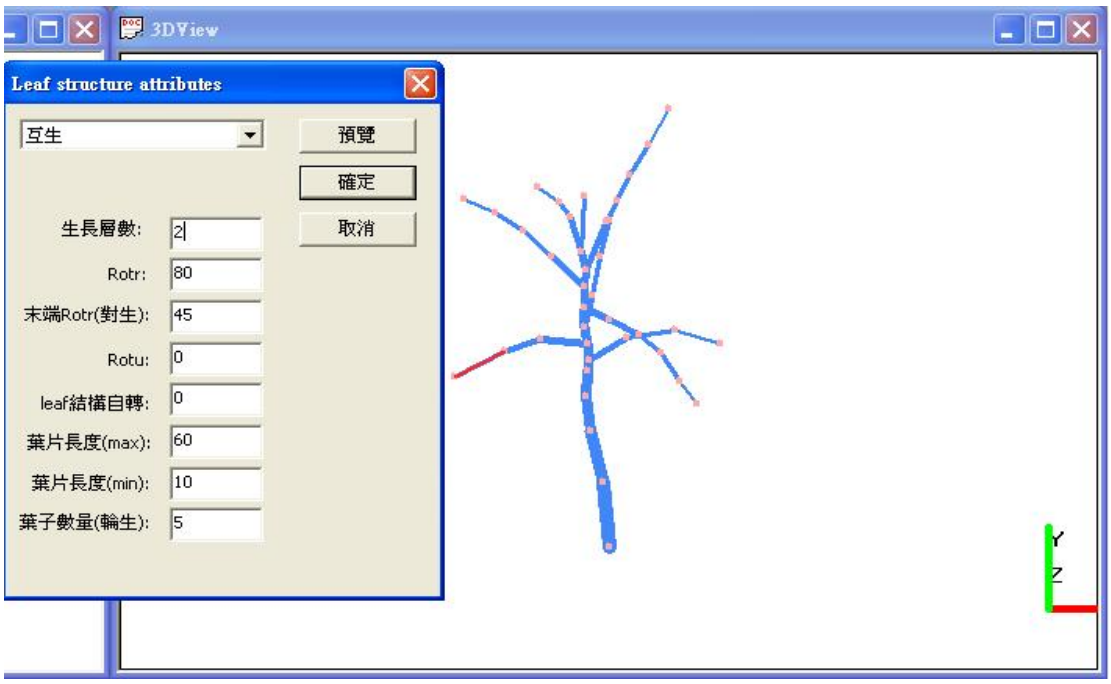
系統產生葉序結構的方法如下：首先系統依照關鍵字產生出基本的葉序結構，下一步依照參數來依序調整葉子的長度、密度、角度與擺向，如圖(11)。



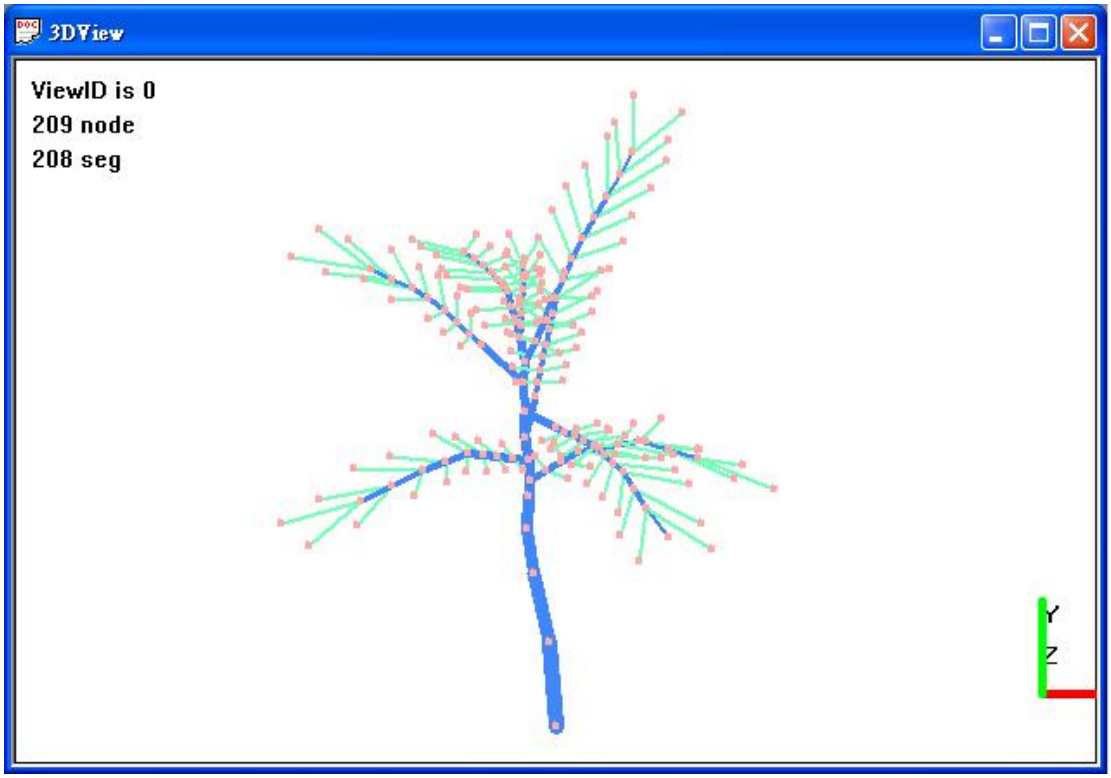
(a)　　(b)　　(c)

圖(11)：葉序產生說明，(a)調整葉子長度與密度，(b)調整角度，(c)調整擺向

操作流程如下：使用者首先將滑鼠移動到要設定長出葉序結構的 stem 上面，接下來點選右鍵呼叫出設定葉序的對話框，如圖(12)，對話框裡面的葉序會預設成之前得到的葉序關鍵字，接下來使用者必須設定生長層數、旋轉角度、葉片長度等等的參數，使用者可以用預覽功能來看目前參數產生出來的葉序結構是否合乎目標植物的葉序，一直到調整出滿意的結果再按下確定，此時系統才將葉序結構正式加入到 skeleton 裡面，設定完成的結構如圖(13)。



圖(12)：設定葉序屬性的對話框



圖(13)：葉序設定完成圖

3.3 量化資訊擷取

在取得植物帶有葉序結構的 skeleton 之後，接下來我們就能由這個 skeleton 來計算出我們所需的量化資訊，我們在這裡是以每一段 stem 當作基本單位來計算量化資訊，而每一段 stem 算出來的數據也都分別紀錄在各自的資料結構裡面。我們在計算量化資訊這個部分的流程如下：

〈1〉 首先我們的系統會先計算每段 stem 的長度。

〈2〉 第二步計算出每段 stem 與根部的層數。

〈3〉 第三步計算 stem 之間的 parent 與 child 的關係。

〈4〉 最後計算 stem 之間的夾角。

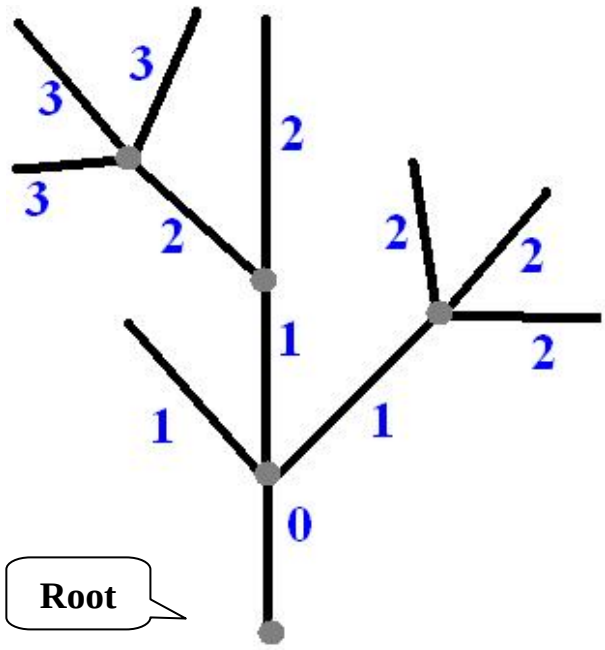
底下的章節將分別介紹每個步驟的詳細做法。

3.3.1 計算 stem 的長度

在求得 stem 的長度方面，我們只需要計算出 stem 兩端點間的距離，此距離就是該 stem 的長度。

3.3.2 計算 stem 的層數

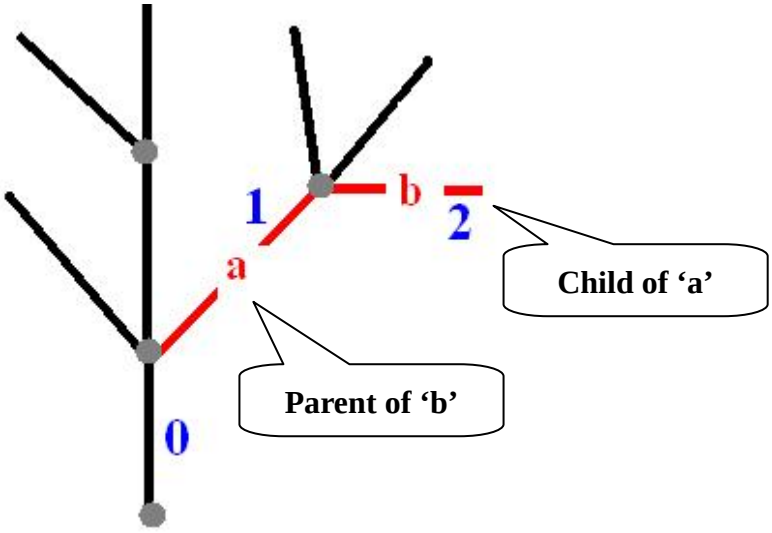
這裡說的層數指的是每段 stem 與根部間隔了幾段其他的 stem，如圖(14)所示。因為系統在下一個步驟計算 stem 之間的 parent 與 child 的關係需要用到 stem 的層數資訊，所以我們在這裡需要先算出每段 stem 的層數。我們的做法是先把每一段 stem 的距離都設定成一單位長度，然後用 Dijkstra's algorithm 來計算出根部到每一段 stem 的最短路徑，路徑長度就代表該 stem 的層數。



圖(14)：level of stem

3.3.3 stem 的 parent 與 child 關係

因為我們的系統接下來找角度與輸出文法的部分需要用參考到 stem 的 parent 與 child 關係，所以我們需要去尋找出每一段 stem 之間的 parent 與 child 的關係。這一部分我們的做法是先找出有相連接的 stem，接下來比較這些相連 stem 的 level，level 較大者表示比較遠離根部，所以是 child，反之 level 較小者則是 parent，如下圖(15)所示。



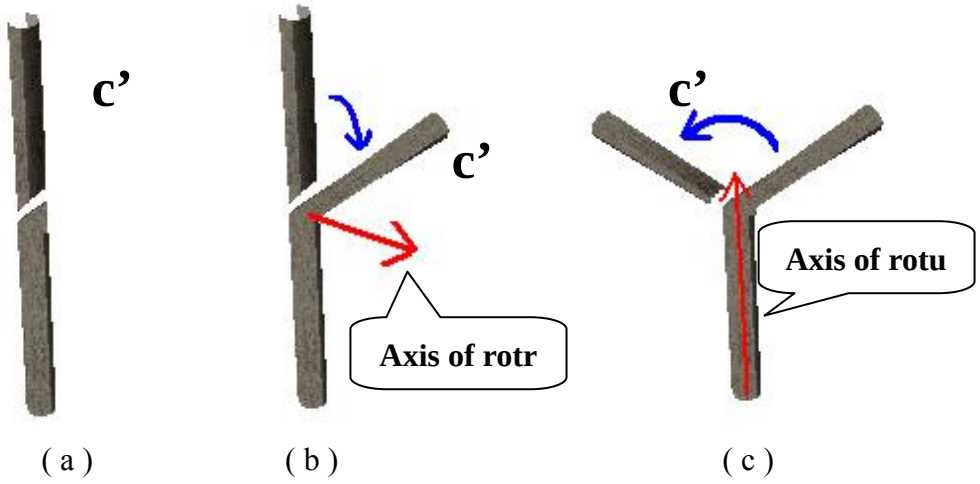
圖(15)：parent and child 關係說明圖

3.3.4 stem 的夾角

因為我們這裡的文法指的是跟本實驗室的 SimEco System 相容的文法，所以我們這部分表達角度的格式以及計算角度的規則都必須參照 SimEco 裡面的方式，才能保證我們計算出來的角度能夠符合 SimEco 的需求，因此接下來的部分我們會先介紹 SimEco 裡面處理角度的方式。

SimEco 是在產出 child stem 的時候處理角度的旋轉，所以我們要先說明一下 SimEco 產生 child stem 的方式，首先 parent stem 會先複製出一段自己的方向向量，

我們將它稱為 c’，如圖(16-a)，此時 c’會依照文法的旋轉角度先做垂直方向的旋轉，如圖(16-b)，此垂直方向的旋轉稱為 rotr，之後再做水平方向的旋轉，如圖(16-c)，此水平方向的旋轉稱為 rotu，c’經過垂直與水平方向旋轉完之後得到的方向向量，就是 child 長出去的起始方向，因此我們的系統必須分別找出垂直方向與水平方向這兩個角度，而且 SimEco 在垂直方向與水平方向的角度指的都是 child stem 相對於 parent stem 的相對夾角。

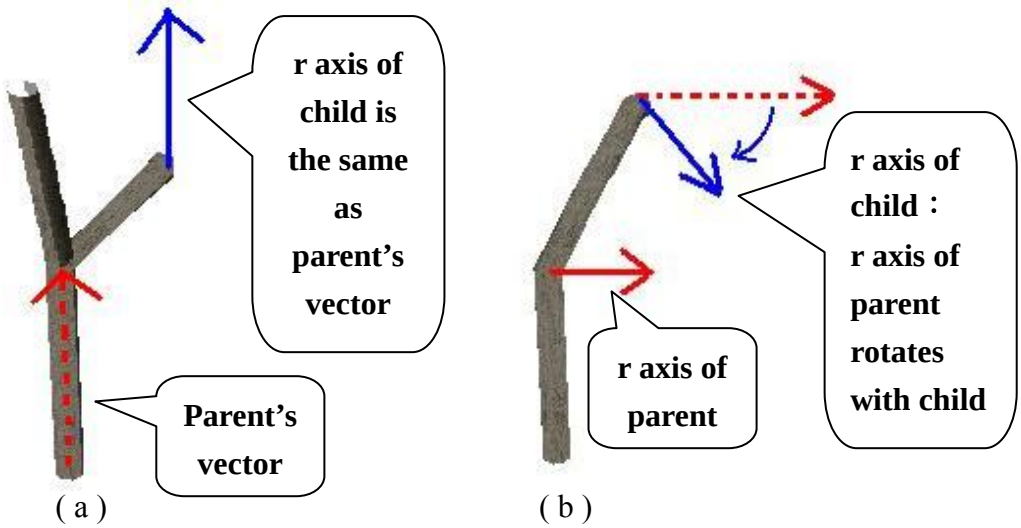


圖(16)：SimEco 產生 child 的方式，(a)複製，(b)做 rotr 旋轉，(c)做 rotu 旋轉。

上述 c’在做 rotr 方向旋轉的時候， c’是繞著一段叫 r 軸的參考軸旋轉，如圖(16-b)所示，c’在做 rotu 旋轉的時候，c’是以 parent 當軸心繞 parent 旋轉，如圖(16-c)所示。SimEco 產生 child 的整個流程為從圖(16-a)至圖(16-c)。

SimEco 在做 rotr 旋轉的時候是以 r 軸當旋轉軸，然而每一段 stem 的 r 軸在 SimEco 裡面並不是固定不變的，而是會依照 stem 在文法裡面是 branch 結構或是 main stem 結構來動態計算出該 stem 的 r 軸。在 SimEco 裡面第一根 stem 的 r 軸是固定的，接下來的 child stem 才會依照文法結構來動態計算出自己的 r 軸，如果 child stem 在文法裡面是 branch 結構的情況下，parent stem 的方向向量就是 child stem 的

r 軸，如圖(17-a)所示。如果 child stem 在文法裡面是 main stem 結構的情況下，child stem 的 r 軸就直接複製 parent 的 r 軸，並且依照 child stem 的 rotr 角度與 rotu 角度隨著 child stem 一起做旋轉的動作，旋轉後得到的 r 軸即是所求，如圖(17-b)所示。



圖(17)：r 軸計算方式，(a)child 是 branch 結構的情況，(b)child 是 main stem 結構的情況時

在了解 SimEco 處理角度的方式後，我們的系統同樣是依照此規則來計算角度，我們的系統計算角度的流程如下：

〈1〉 求得第一段 stem 的 r 軸。

〈2〉 將全部 stem 的 child 分類成 main stem 與 branch 的結構。

〈3〉 依照結構計算出全部 stem 的 r 軸。

〈4〉 依照 r 軸計算角度。

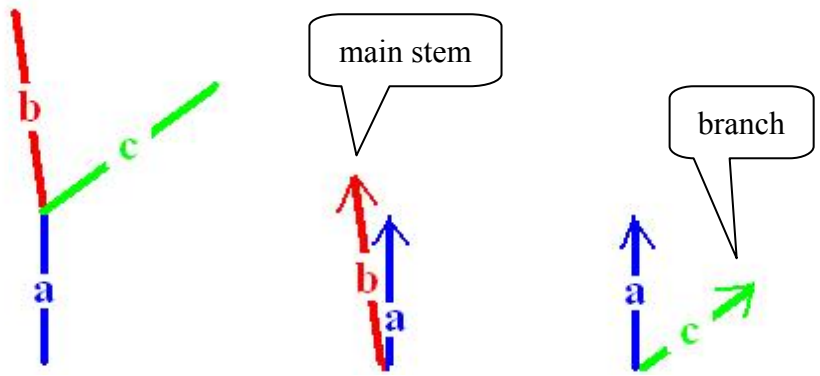
接下來我們將詳述每個流程的做法。

〈1〉相對於 SimEco 的第一段 stem 的 r 軸是固定的，我們的系統以第一段 stem 跟與其夾角最小的 child 做外積運算得到的向量來當第一段 stem 的 r 軸，我們稱第

一段 stem 為 P，稱與 S 夾角最小的 child stem 為 C，第一段 stem 的 r 軸計算方法如下式：

$$\text{第一段 stem 的 r 軸} = P \times C \qquad (1)$$

〈2〉有了第一跟 stem 的 r 軸之後，我們的系統同樣要依照 child stem 是否為 main stem 結構或是 branch 結構來套用應有的 r 軸計算方式來計算出 r 軸，所以我們這個步驟要先判斷 child stem 是否為 main stem 結構或是 branch 結構，我們的方法是依照 parent stem 跟 child stem 夾角的大小來判斷，首先把 parent stem 與 child stem 的向量計算出來，其中向量方向固定從比較靠近根部的點指向比較遠離根部的點，接下來就是計算 parent 向量與 child 向量的夾角，夾角最小者就表示該 child 是 main stem 結構，如圖(18-b)，其他夾角較大者全部歸類為 branch 結構，如圖(18-c)所示，也就是說我們的系統在這裡只允許有一段 main stem 結構，其他的都是 branch 結構。



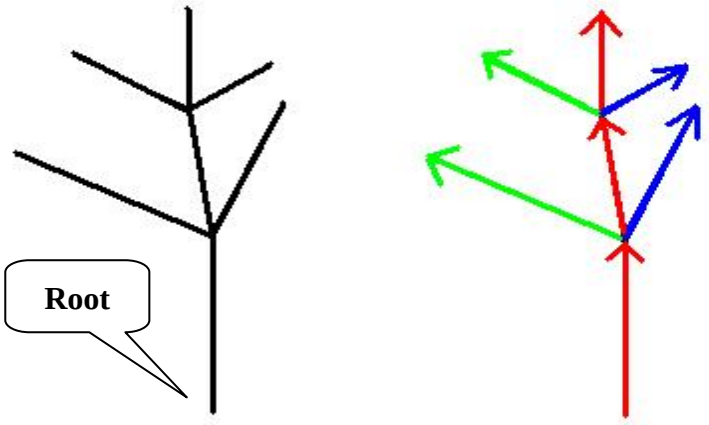
(a)　　(b)　　(c)

圖(18)：main stem 的找法，(a)原始結構，(b)夾角最小者為 main stem，(c)夾角較大者為 branch 結構

〈3〉Main stem 與 branch 結構分類完之後，這個步驟我們就依照 stem 的結構套用對應的 r 軸計算方式來計算出每一段 stem 的 r 軸。

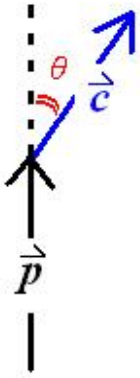
〈4〉有了每一段 stem 的 r 軸之後，我們就能夠開始計算角度，我們的系統在這裡計算出來的角度必須跟 SimEco 一樣是 child stem 相對於 parent stem 的相對夾角，因此我們這裡的做法是把每一段 stem 都跟自己的 parent stem 做角度計算，算出與 parent 在垂直方向與水平方向上的夾角。

我們計算角度的做法是先取得 parent stem 的向量 $\bar{p}$ 與 child stem 的向量 $\bar{c}$，其中向量的方向同樣是固定由靠近根部的點指向遠離根部的點，如圖(19-b)，向量 $\bar{p}$ 與向量 $\bar{c}$ 的夾角 θ 就是垂直方向上的角度，如圖(20)。



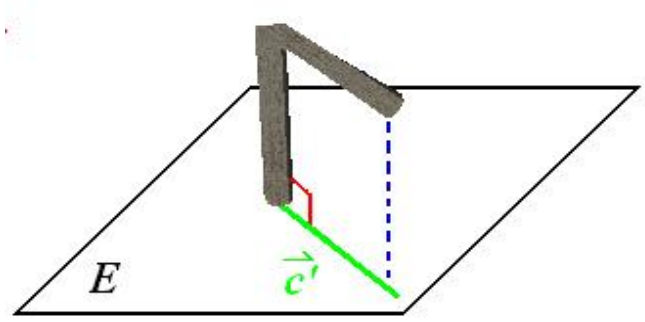
(a) (b)

圖(19)：向量方向說明圖，(a)原始結構，(b)標示出向量方向的結構

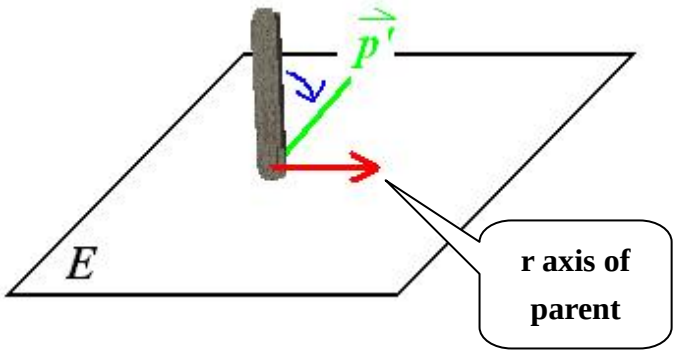


圖(20)：垂直方向角度計算方式

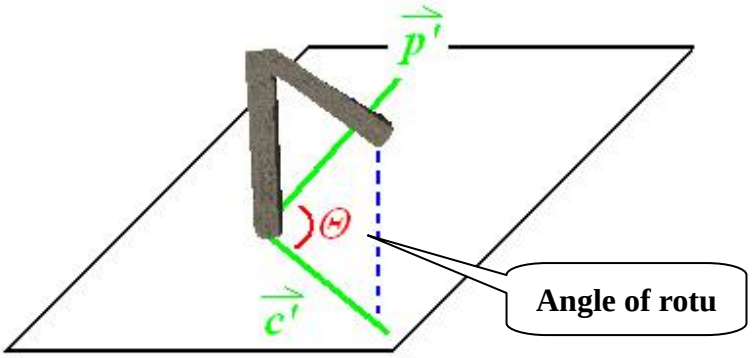
我們在計算水平方向角度的做法是先把 child stem 投影到通過 $\bar{p}$ 起點而且與 $\bar{p}$ 垂直的平面 E 上，如圖(21)，child stem 投影在平面 E 上得到投影向量 $\bar{c}$'。但是只有 $\bar{c}$'一個向量是無法計算角度的，因此我們要還要知道 child stem 在進行水平方向旋轉的起始位置，因為 SimEco 在處理旋轉的時候是先做垂直方向的旋轉，之後才進行水平方向的旋轉，所以 child stem 做完垂直方向旋轉之後的位置就是進行水平方向旋轉的起始位置，我們將 $\bar{p}$ 繞 parent stem 的 r 軸轉 90 度得到向量 $\bar{p}$'， $\bar{p}$'就是在 SimEco 裡面 child stem 在做水平方向旋轉之前的向量，也就是 child stem 在進行水平方向旋轉的起始位置，如圖(22)，接下來我們計算出 $\bar{c}$'與 $\bar{p}$'的角度，就是 child stem 在水平方向上的角度，如圖(23)所示。



圖(21)：child 投影到平面 E 上



圖(22)： $\bar{p}$ 轉 90 度得到 $\bar{p}$'



圖(23)：rotu 夾角求得方式

3.4 輸出文法

在求得我們所需的資訊之後，我們的系統接下來就是進行將這些資訊整合並轉換成文法的最後步驟，輸出文法這步驟我們的系統又分成兩個部分：

〈1〉 輸出初始狀態的文法。

〈2〉 加入預測的部分，得到預測狀態的文法。

我們的系統依照現有的資訊所輸出的文法稱為初始狀態的文法，初始狀態的文法經過 SimEco System 模擬出來的植物架構會跟我們系統建構出來的植物三維骨架非常相似，接下來我們的系統會針對此初始狀態文法做分析的動作，整理出結構類似的部分並加入到初始狀態的文法裡面，進而得到預測狀態的文法，預測狀態的文法則能夠進一步模擬預測該植物接下來可能成長的狀態，底下我們將分別說明這兩個部分。

3.4.1 初始狀態文法

我們先簡單的說明一下 SimEco 所定義的文法結構，整個文法巨觀的結構如下圖(24)：

```
KeywordEcosystem VersionNumber “SystemName” {
    Ecosystem Attributes
    Keyword Plant “PlantName” {
        Plant Attributes;
        OrganStage1 {
            ……
        }
        ……
        OrganStageN {
            ……
        }
        KeywordGrow {
            Axiom;
            Production Rules;
        }
    }
}
```

圖(24)：文法結構

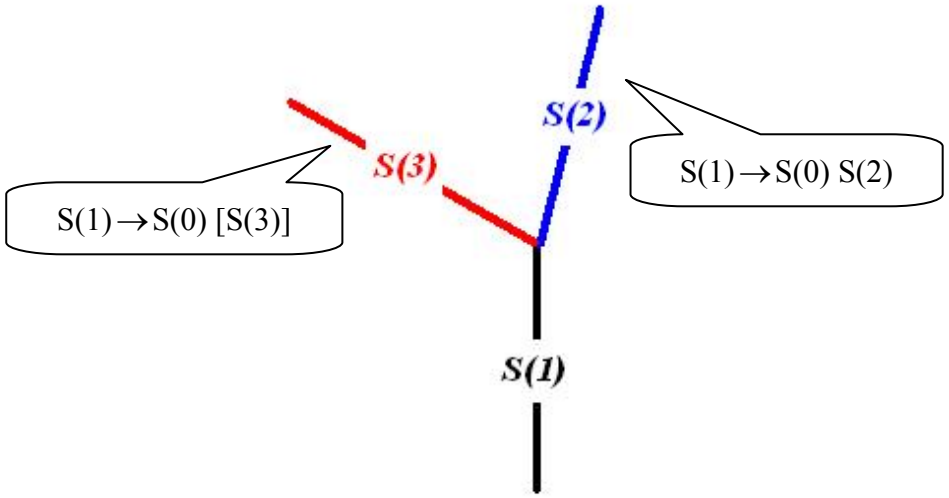
文法一開始是整個系統的屬性參數(Ecosystem attributes)，接下來是植物的屬性參數(Plant Attributes)，之後就是植物器官的特性與參數(OrganStage)，例如莖的長度，葉的大小都是在這部分設定的，最後是整個植物的生長規則區塊(Grow block)，這裡就是以文法的生長規則(Production Rules)來表現植物生長情況的部分，其中植物器官的參數以及生長規則就是我們系統主要的處理對象，有關文法結構更詳細的說明請參考 An Adaptive L-System [8]。我們的目的就是把抽取出來的量化資訊轉換成文法裡面 OrganStage 的參數，並且把整個植物的三維骨架以及葉序結構轉換成文法裡面的生長規則。

我們的系統首先會把抽取出來的量化資訊轉換成 OrganStage 的參數，在 3.3 節有提到每一段 stem 都有紀錄下各自的量化資訊，我們的做法就是把每一段 stem 當作一個 stage，接下來把量化資訊對應到適當的 stage 參數裡面，之後就是分別把這

些 stage 輸出成文法的格式。

在全部 stage 都輸出完畢之後，接下來就是處理生長規則的部分了，這個部分我們的做法是依照 parent 與 child 的關係來輸出生長規則，而且 main stem 結構與 branch 結構在文法裡面會有不同的格式，所以我們在輸出生長規則的時候必須考慮 child 是 main stem 或是 branch 結構。我們以圖(25)當例子來說明，S(1)是 parent，S(2)是 main stem 結構的 child，S(3)是 branch 結構的 child，單就 S(1)與 S(2)而言，S(2)是 main stem 結構，這時候輸出的生長規則是 S(1)→S(0) S(2) ，其中 S(0)是拿來當成生長點用的，接下來我們看 S(1)與 S(3)，S(3)是 branch 結構，這時候輸出的生長規則是 S(1)→S(0) [S(3)]，要注意的是 branch 結構必須用中括弧括起來。圖(25)整個結構所輸出的生長規則加上角度的參數大致上會像下式：

S(1) → S(0) [rotr(…) rotu(…) S(3)] rotr(…) rotu(…) S(2) ; (2)



圖(25)：文法輸出結構說明圖例

我們輸出生長規則的順序是從根部出發，之後依序將每一段有 child 的 stem 輸出成生長規則，直到每一段有 child 的 stem 皆輸出完成為止，此時得到的文法稱為

初始狀態的文法。因為我們這部分在輸出文法的時候是採用把每段 stem 逐一輸出成文生長規則，所以初始狀態的文法經由 SimEco system 模擬出來的植物結構會跟我們系統建構出來的植物三維骨架一模一樣。

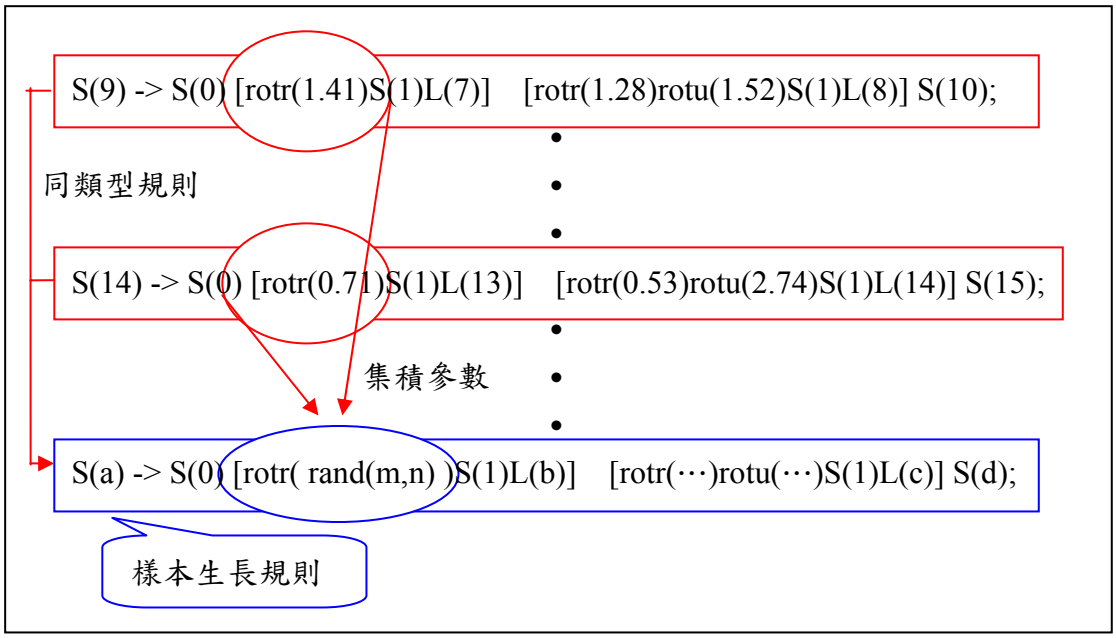
3.4.2 預測狀態的文法

除了產生出初始化狀態的文法之外，我們接下來的目標是從初始化狀態的文法去找出某些規則，讓我們的文法在經由 SimEco 生長至初始狀態之後，更能套用這些規則來進行更進一步的生長預測與模擬。我們的做法分成兩個部分：

〈1〉 建立預測生長規則的樣本。

〈2〉 將樣本加入疊代時間的控制與足標來達到自我展開的特性。

第一個步驟是先從初始狀態的文法裡面找出具有相同結構的生長規則，之後建立出此結構尚未包含數值資料的樣本，接下來我們集積這些同類型生長規則的參數，就可找出亂數的範圍，最後我們把亂數的數值資料與我們建立的樣本整合在一起，並且加入到文法裡面，如圖(26)所示。

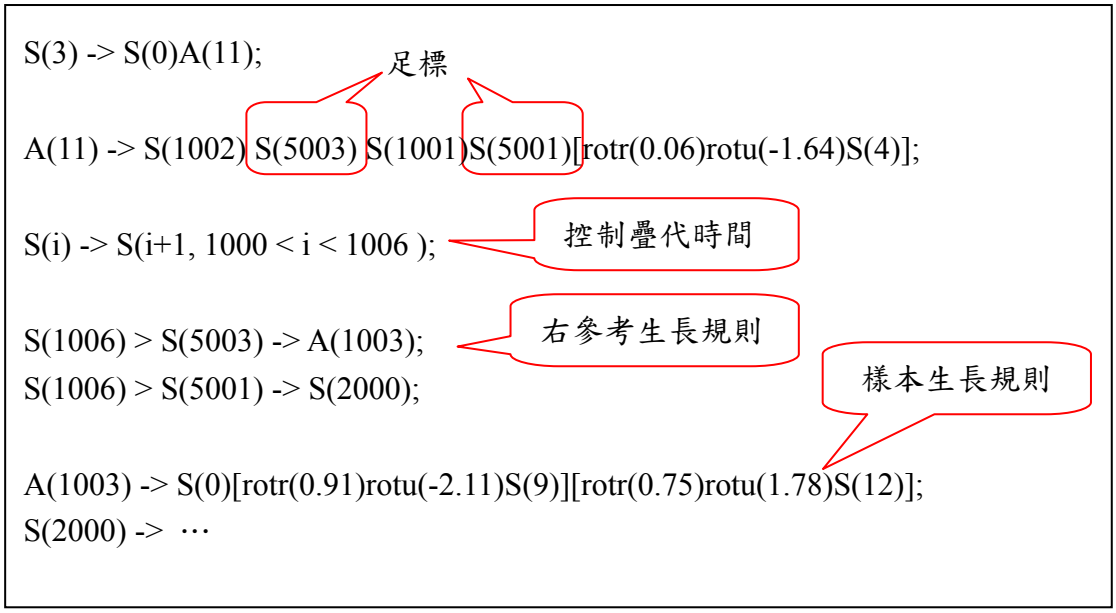


圖(26)：樣本生長規則的產生方式

第二個步驟是加入一道如下式(3)的生長規則：

S(i) -> S(i+1, start number < i < end number); (3)

我們稱這道生長規則為 step function，其中 i 只要介於起始數值與終止數值之間就會執行此規則，這道規則的功能是每經過一次的疊代 i 就會遞增 1，直到終止條件才結束。除了加入上式之外，我們還需要對樣本規則加入疊代時間的控制與足標，並且配合右參考的規則才能達成樣本規則自我展開的特性，下圖(27)是加入疊代時間控制與足標的例子。

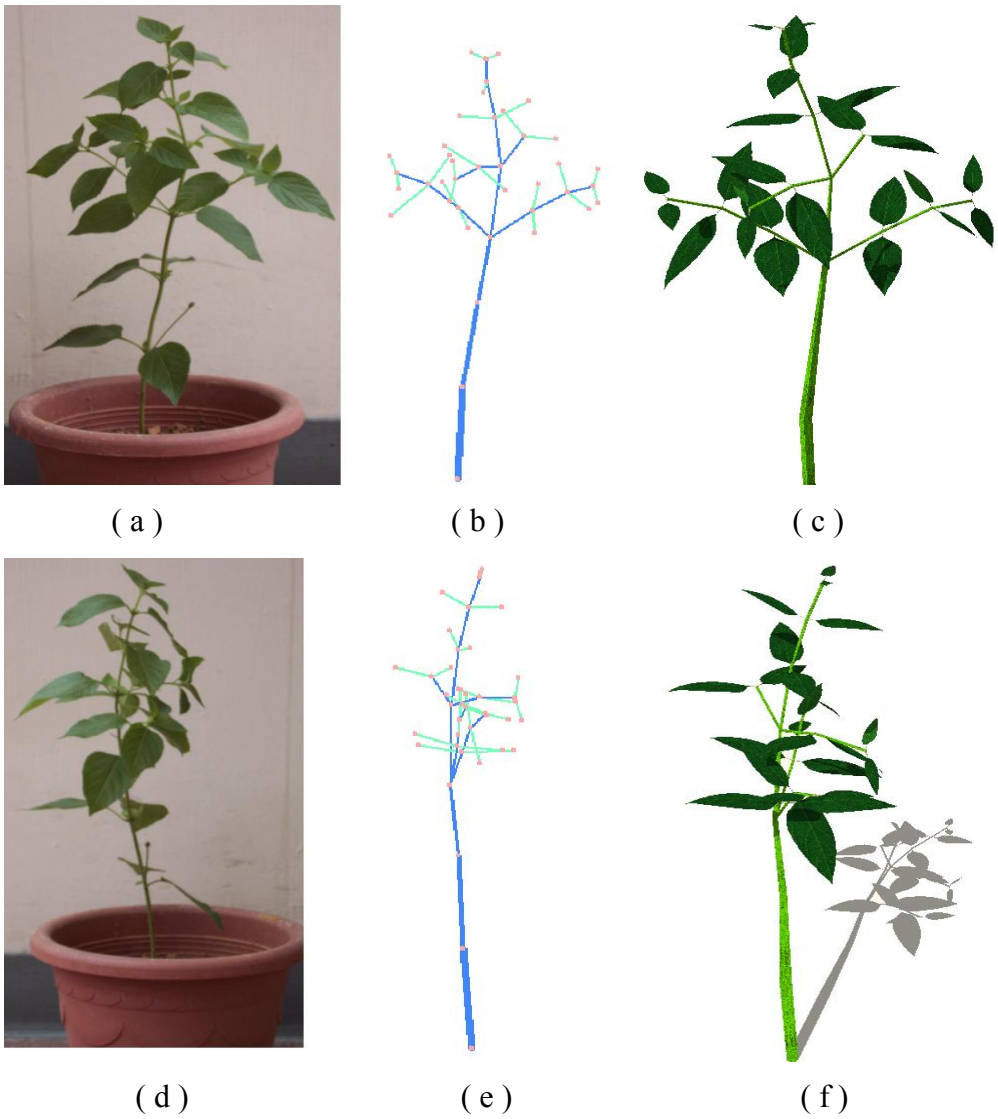


圖(27)：加入疊代時間與足標的例子

我們採用從植物三維骨架找出來的最大深度來當 step function 的終止數值，用意在於我們希望 SimEco 系統在初始狀態的文法全部處理生長完畢之後才開始處理預測的部分，另外從上圖(27)當中可以看出 S(1002)S(5003)經由 step function 疊代到 S(1006)S(5003)的時候，此時就會套用 S(1006) > S(5003) 這道右參考生長規則，進而疊代成 A(1003)，最後 A(1003)再疊代成樣本生長規則，由此可知我們透過足標與右參考生長規則成功的把樣本生長規則安插到文法當中。

第四章 研究成果與未來展望

使用者只需要了解3.1節的關鍵字擷取以及3.2節的建立植物三維骨架這兩部分的流程與操作方法，就能夠使用本系統來產生植物的 L-grammar，我們的系統在操作過程當中並不會需要由使用者直接去撰寫或修改文法，所以就算是沒有 formal language 的相關知識的使用者也是可以使用本系統來產生 L-grammar。以下是一些我們的研究成果：

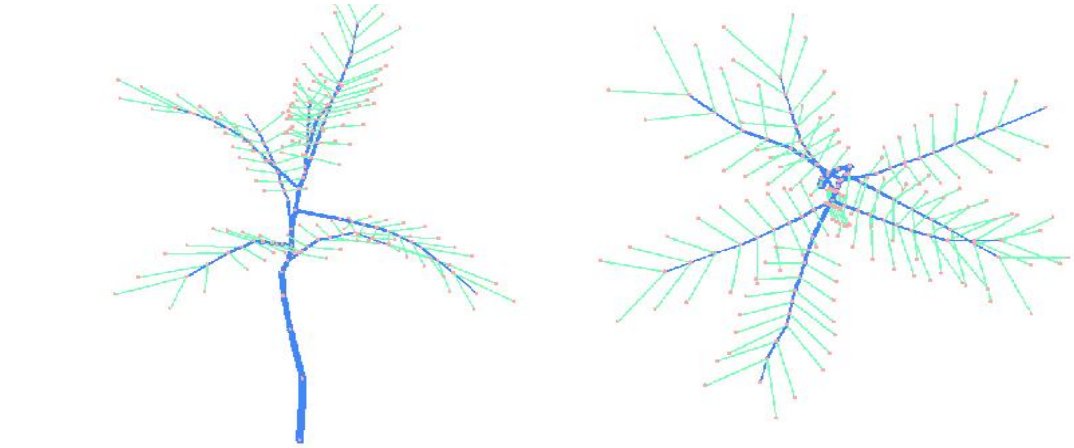


(a) (b) (c)

(d) (e) (f)

圖(28)：馬櫻丹成果圖。(a)、(d)馬櫻丹植物影像。(b)、(e) 是附有葉序結構的 skeleton。(c)、(f) 是系統產生出的 L-grammar 經由 SimEco 畫出來的

(a)　　(b)



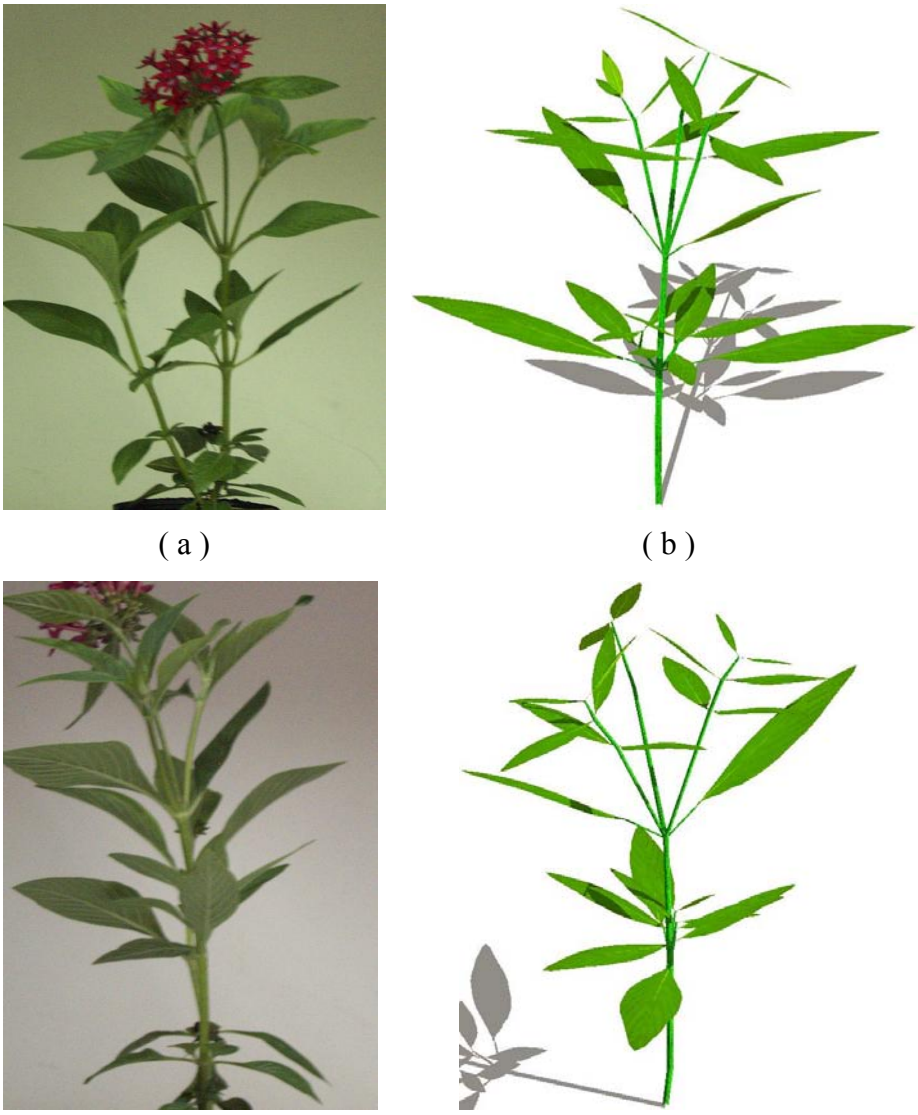
(c)　　(d)

(e)　　(f)

圖(29)：鐵刀木成果圖。(a)是翅果鐵刀木的正面影像，(b)是頂視影像。(c)、(d)是附有葉序結構的 skeleton。(e)、(f)是系統產生出的 L-grammar 經由 SimEco 畫出來的影像。

圖(30)：馬櫻丹初步預測狀態連續圖

圖(31)：翅果鐵刀木初步預測狀態連續圖



(a) (b)

(c) (d)

圖(32)：繁星花成果圖。(a)、(c)繁星花照片。(b)、(d)系統產生影像。

(a) (b) (c)

圖(33)：嫣紅蔓成果圖。(a)嫣紅蔓照片。(b)、(c)系統產生的影像。

(a)　　(b)　　(c)

圖(34)：玫瑰成果圖。(a)玫瑰照片。(b)、(c)系統產生的影像。

(a)　　(b)　　(c)

圖(35)：日日春成果圖。(a)日日春照片。(b)、(c)系統產生影像。

由上面的成果圖可以很明顯地看出來我們的研究有許多需要改進的地方，因為我們的系統是用線段來表植物的莖幹，而且輸出文法的時候是由 parent 與 child 的關係直接產生生長規則，因此造成畫出來的莖也是由一段一段的線段所構成，而且莖

與莖之間夾角過大的時候，就會造成明顯不自然的尖角，而且我們尚未處理莖的貼圖，導致全部的莖都是同一種顏色。

除了改善影像品質之外，量化資訊的正確性也很重要，因為在我們的系統中，樹葉以及莖幹寬度是用參數設定來產生的，並不是從植物影像中取得真實數據來產生，只要輸入的參數跟植物本身的數據有差距的時候，對於產生出來的影像就會有很大的影響，這部分我們預計透過影像處理的手法來取得正確的量化資訊。

目前我們預測生長規則樣本的數值是從每個相同結構的生長規則集積而來，但是這意味著我們把不同層結構的數據拿來當同一層的數據使用，導致數據集積出來的結果有問題，如何分層來集積生長規則樣本的數據也是我們未來要改進的地方。而且從多棵同種植物取得的量化資訊比只從單一棵植物所取得的更為客觀，未來我們也預計從多棵同種植物的影像來取得量化資。

文法結構也有許多需要加強的部分，我們希望藉由把生長規則結構化來產生出具有自我疊代性質的生長規則，並且希望能控制其最大疊代深度，進而提高文法的可讀性。目前預測狀態的文法只能進行一個階段的預測，未來我們要加強預測的廣度與深度。此外因為植物生長點的位置會因為種類的不同而改變，所以我們未來預計要參考植物學上的專業知識來得到正確的生長點位置，上述部分都是我們在文法結構上需要努力改進的目標。

L-grammar 複雜且難懂的語法結構是讓使用者望之卻步的主要原因之一，未來我們會朝著降低 L-System 的門檻為目標繼續邁進，希望讓系統能夠更自動化的產生文法之外，並且建立 L-grammar 的資料庫，讓文法的取得更加容易。

SimEco 與 L-grammar 研究成果的網頁：
http://staffweb.ncnu.edu.tw/lhchen/simeco/Lgrammar.htm

第五章 參考文獻

1. Alex Reche-Martinez, Ignacio Martin, George Drettakis, “Volumetric reconstruction and interactive rendering of trees from photographs”, ACM Transactions on Graphics, Volume 23 , Issue 3, 2004, Page(s): 720 – 727.

2. Alvy Ray Smith, “Plants, fractals, and formal languages”,ACM SIGGRAPH Computer Graphics , Proceedings of the 11th annual conference on Computer graphics and interactive techniques, 1984, Page(s): 1 - 10

3. Callum Galbraithy, Lars Mündermann, Brian Wyvill, “Implicit Visualization and Inverse Modeling of Growing Trees”, Eurogrphics 2004, Volume 23, Session 3

4. Przemyslaw Prusinkiewicz, Mark Hammel, Radomír Měch, “The Artificial Life of Plants”, Artificial life for graphics, animation, and virtual reality, volume 7 of SIGGRAPH ’95 Course Notes, pages 1-1–1-38. ACM, Press, 1995.

5. Shlyakhter, I.; Rozenoer, M.; Dorsey, J.; Teller, S.; “Reconstructing 3D tree models from instrumented photographs”, Computer Graphics and Applications, IEEE , Volume 21 , Issue 3 , May/Jun 2001 , Page(s):53 – 61.

6. Ying Sun; Bergerson, E.; “Automated 3D reconstruction of tree-like structures from two orthogonal views”, Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on , 11-14 April 1988, Page(s):1296 - 1299 vol.2

7. 張瑜宏, “A Growth Simulation System of Taiwanese Common Plants Based on Lindenmayer System”, Master's degree thesis, National Chi Nan University, Department of Computer Science and Information Engineering, 2003.

8. 石凌霖, “An Adaptive L-System”, Master's degree thesis, National Chi Nan University, Department of Computer Science and Information Engineering, 2004.

附錄 系統產生的文法實例

```
EcoSystem 1.0 "TestSys" {
    SoilPH = 7.000000;
    RainRate = 0.900000;
    DayLight = 12.000000;
    Plant "TestPlant" {
        Position = vector(0.000000, 0.000000, 0.000000);
        Direction = vector(0.000000, 1.000000, 0.000000);
        DeriveTime = 3.000000;
        Apex {}
        Stem {
            Stage 0 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.000100;
                GrowTime = 0.000100;
            }
            Stage 1 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.100000;
                GrowTime = 0.100000;
                BroadenRate = 0.500000;
            }
            Stage 2 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.193752;
                GrowTime = 1.000000;
                BroadenRate = 1.000000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
            Stage 3 {
                Model = "DefaultStem.i";
```

```
                TextureFront = "seafloor.bmp";
                GrowRate = 0.179444;
                GrowTime = 1.000000;
                BroadenRate = 0.600000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
            Stage 4 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.137899;
                GrowTime = 1.000000;
                BroadenRate = 0.400000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
            Stage 5 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.157328;
                GrowTime = 1.000000;
                BroadenRate = 0.200000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
            Stage 6 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.112925;
                GrowTime = 1.000000;
                BroadenRate = 0.200000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
```

```
        }
        Stage 7 {
            Model = "DefaultStem.i";
            TextureFront = "seafloor.bmp";
            GrowRate = 0.090863;
            GrowTime = 1.000000;
            BroadenRate = 0.200000;
            ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
            MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
        }
        Stage 8 {
            Model = "DefaultStem.i";
            TextureFront = "seafloor.bmp";
            GrowRate = 0.062129;
            GrowTime = 1.000000;
            BroadenRate = 0.200000;
            ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
            MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
        }
        Stage 9 {
            Model = "DefaultStem.i";
            TextureFront = "seafloor.bmp";
            GrowRate = 0.098874;
            GrowTime = 1.000000;
            BroadenRate = 0.200000;
            ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
            MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
        }
        Stage 10 {
            Model = "DefaultStem.i";
            TextureFront = "seafloor.bmp";
            GrowRate = 0.082898;
            GrowTime = 1.000000;
            BroadenRate = 0.200000;
```

```
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
            Stage 11 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.075631;
                GrowTime = 1.000000;
                BroadenRate = 0.200000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
            Stage 12 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.104403;
                GrowTime = 1.000000;
                BroadenRate = 0.200000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
            Stage 13 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.078256;
                GrowTime = 1.000000;
                BroadenRate = 0.200000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
            Stage 14 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
```

```
                GrowRate = 0.054074;
                GrowTime = 1.000000;
                BroadenRate = 0.200000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
            Stage 15 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.076420;
                GrowTime = 1.000000;
                BroadenRate = 0.200000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
            Stage 16 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.071021;
                GrowTime = 1.000000;
                BroadenRate = 0.200000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
            Stage 17 {
                Model = "DefaultStem.i";
                TextureFront = "seafloor.bmp";
                GrowRate = 0.079221;
                GrowTime = 1.000000;
                BroadenRate = 0.200000;
                ColorFront = RGBA(0.000000, 1.000000, 0.000000, 1.000000);
                MaterialFront = MTRL(0.500000, 1.000000, 0.000000, 0.100000,
0.100000, 0.100000);
            }
```

```
}
Leaf {
	Stage 1 {
		Model = "馬櫻丹.i";
		TextureFront = "馬櫻丹.bmp";
		GrowRate = 0.074000;
		GrowTime = 1.000000;
	}
	Stage 2 {
		Model = "馬櫻丹.i";
		TextureFront = "馬櫻丹.bmp";
		GrowRate = 0.070000;
		GrowTime = 1.000000;
	}
	Stage 3 {
		Model = "馬櫻丹.i";
		TextureFront = "馬櫻丹.bmp";
		GrowRate = 0.060000;
		GrowTime = 1.000000;
	}
	Stage 4 {
		Model = "馬櫻丹.i";
		TextureFront = "馬櫻丹.bmp";
		GrowRate = 0.058000;
		GrowTime = 1.000000;
	}
	Stage 5 {
		Model = "馬櫻丹.i";
		TextureFront = "馬櫻丹.bmp";
		GrowRate = 0.030000;
		GrowTime = 1.000000;
	}
	Stage 6 {
		Model = "馬櫻丹.i";
		TextureFront = "馬櫻丹.bmp";
		GrowRate = 0.030000;
		GrowTime = 1.000000;
```

```
}
Stage 7 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.098000;
	GrowTime = 1.000000;
}
Stage 8 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.090000;
	GrowTime = 1.000000;
}
Stage 9 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.072000;
	GrowTime = 1.000000;
}
Stage 10 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.094000;
	GrowTime = 1.000000;
}
Stage 11 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.046000;
	GrowTime = 1.000000;
}
Stage 12 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.048000;
	GrowTime = 1.000000;
}
```

```
Stage 13 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.086000;
	GrowTime = 1.000000;
}
Stage 14 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.084000;
	GrowTime = 1.000000;
}
Stage 15 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.058000;
	GrowTime = 1.000000;
}
Stage 16 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.064000;
	GrowTime = 1.000000;
}
Stage 17 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.046000;
	GrowTime = 1.000000;
}
Stage 18 {
	Model = "馬櫻丹.i";
	TextureFront = "馬櫻丹.bmp";
	GrowRate = 0.042000;
	GrowTime = 1.000000;
}
Stage 19 {
```

```
            Model = "馬櫻丹.i";
            TextureFront = "馬櫻丹.bmp";
            GrowRate = 0.072000;
            GrowTime = 1.000000;
        }
        Stage 20 {
            Model = "馬櫻丹.i";
            TextureFront = "馬櫻丹.bmp";
            GrowRate = 0.082000;
            GrowTime = 1.000000;
        }
        Stage 21 {
            Model = "馬櫻丹.i";
            TextureFront = "馬櫻丹.bmp";
            GrowRate = 0.086000;
            GrowTime = 1.000000;
        }
        Stage 22 {
            Model = "馬櫻丹.i";
            TextureFront = "馬櫻丹.bmp";
            GrowRate = 0.082000;
            GrowTime = 1.000000;
        }
        Stage 23 {
            Model = "馬櫻丹.i";
            TextureFront = "馬櫻丹.bmp";
            GrowRate = 0.046000;
            GrowTime = 1.000000;
        }
        Stage 24 {
            Model = "馬櫻丹.i";
            TextureFront = "馬櫻丹.bmp";
            GrowRate = 0.042000;
            GrowTime = 1.000000;
        }
    }
    Root {}
```

```
        Fruit {}
        Flower {}
        Grow {
            w = A(0);
            A(0) -> S(2) ;
            S(2) -> S(0) A(10) ;
            A(10) -> rotr(0.118453)rotu(0.000000)S(3) ;
            S(3) -> S(0) A(11) ;
            A(11) -> rotr(0.060320)rotu(-1.640751)S(4) ;
            S(4) -> S(0) [rotr(0.915944)rotu(-2.102210)S(9)]
[rotr(0.753473)rotu(1.781339)S(12)] rotr(0.143584)rotu(-2.926000)S(5) ;
            S(5) -> S(0) [rotr(0.782279)rotu(-2.720525)S(15)]
[rotr(1.327108)rotu(0.355939)S(16)] rotr(0.253139)rotu(0.829668)S(6) ;
            S(9) -> S(0) [rotr(1.437030)rotu(-2.948652)S(1)L(7)]
[rotr(1.412685)rotu(0.192941)S(1)L(8)] rotr(0.129264)rotu(-0.740093)S(10) ;
            S(12) -> S(0) [rotr(1.461646)rotu(-0.441349)S(1)L(13)]
[rotr(1.455861)rotu(2.700244)S(1)L(14)] rotr(0.151972)rotu(-2.562562)S(13) ;
            S(6) -> S(0) [rotr(1.304935)rotu(-3.040756)S(1)L(1)]
[rotr(1.243223)rotu(0.100836)S(1)L(2)] rotr(0.128525)rotu(-0.589314)S(7) ;
            S(10) -> S(0) [rotr(1.398658)rotu(1.861781)S(1)L(9)]
[rotr(1.438476)rotu(-0.979812)S(1)L(10)] rotr(0.296512)rotu(-1.367003)S(11) ;
            S(13) -> S(0) [rotr(1.278759)rotu(0.481072)S(1)L(15)]
[rotr(1.252177)rotu(-2.660521)S(1)L(16)] rotr(0.272038)rotu(1.113786)S(14) ;
            S(15) -> S(0) [rotr(1.254986)rotu(-3.115658)S(1)L(20)]
rotr(1.250299)rotu(0.025934)S(1)L(19) ;
            S(16) -> S(0) [rotr(1.386465)rotu(-2.566244)S(1)L(21)]
[rotr(1.368044)rotu(0.135349)S(1)L(22)] rotr(0.249618)rotu(-1.884596)S(17) ;
            S(7) -> S(0) [rotr(1.297974)rotu(-0.500724)S(1)L(3)]
[rotr(1.306198)rotu(2.640868)S(1)L(4)] rotr(0.165558)rotu(-1.632350)S(8) ;
            S(11) -> S(0) [rotr(1.446511)rotu(2.246520)S(1)L(12)]
rotr(1.421713)rotu(-0.895073)S(1)L(11) ;
            S(14) -> S(0) [rotr(1.363551)rotu(-2.864109)S(1)L(18)]
rotr(1.347926)rotu(0.277484)S(1)L(17) ;
            S(17) -> S(0) [rotr(1.301708)rotu(-3.008880)S(1)L(24)]
rotr(1.259434)rotu(0.132713)S(1)L(23) ;
            S(8) -> S(0) [rotr(1.085069)rotu(-0.439142)S(1)L(5)]
rotr(1.070187)rotu(2.702451)S(1)L(6) ;
```

```
            }
        }
    }
```