國立臺灣師範大學理學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Science

National Taiwan Normal University

Master's Thesis

基於多視角相片產生之 3D 點雲樹木重建

Tree Reconstruction from 3D Point Cloud based on Multi-view Photos

陳建壹

CHEN, Chien-Yi

指導教授: 張鈞法 博士

Advisor: CHANG, Chun-Fa Ph.D.

中華民國 114 年 7 月

July 2025

摘要

樹木是自然環境中不可或缺的元素，不僅是環境重建與山林模擬的重要目標，也廣泛應用於遊戲、動畫與電影場景中，能有效營造氛圍並提升沉浸感。然而樹木具有複雜的分枝結構與茂密的樹冠，傳統人工建模方式不僅耗時費力，也難以應對大量模型的產出需求。因此發展一種兼具效率與準確性的建模方法，有助於建立具備實用性與可重複利用價值的樹木模型資產，滿足大規模生成與多元應用的需求。

本研究提出一套結合運動回復結構技術（Structure-from-Motion, SFM）與 L-system 的程序化建模方法。首先透過多視角影像重建點雲，擷取樹木區域並濾除無效點，接著應用 AdTree 演算法提取骨架結構，建立初步樹狀模型。考量 AdTree 對雜訊與葉片分布的敏感性，可能造成末端分枝過密與不自然，本研究進一步透過規則修剪異常結構，並引入 L-system 根據樹種特性生成自然的細枝與葉片分布，以提升模型的結構合理性與視覺自然感。此外針對因視角不足造成的點雲缺失，亦可將現有分支轉換為可重用的 L-system 子樹，由使用者選擇節點進行接合補齊。最終將完整的樹木結構輸出為可編輯的三維網格模型（如.obj），可以應用於各類虛擬場景，展現本方法於樹木建模上的靈活性與實用性。

關鍵字：點雲、L-system、三維重建、樹木重建、運動回復結構、電腦圖學

Abstract

Trees are essential elements in natural environments, playing crucial roles not only in environmental reconstruction and forest simulation but also widely in games, animations, and film scenes, effectively creating atmosphere and enhancing immersion. However, trees have intricate branching structures and dense canopies, making traditional manual modeling methods both labor-intensive and inefficient for producing a large volume of models. Consequently, developing a modeling approach that balances efficiency with accuracy would significantly benefit the creation of practical and reusable tree model assets, fulfilling the requirements of large-scale generation and diverse applications.

This study proposes a procedural tree modeling method combining Structure-from-Motion (SFM) technology and L-systems. First, point clouds are reconstructed from multi-view images, and irrelevant points are filtered out to isolate tree regions. Then, the AdTree algorithm is applied to extract initial skeletal structures and construct preliminary tree models. Considering the sensitivity of AdTree to noise and uneven leaf distribution—

which can result in excessively dense or unnatural end branches—this research further employs rule-based pruning to eliminate anomalous structures. Additionally, the L-system is introduced to generate natural fine branches and leaf distributions based on specific tree characteristics, improving structural rationality and visual realism. Moreover, to address missing point cloud data due to insufficient viewpoints, existing branches can be converted into reusable L-system subtrees, allowing users to select nodes for seamless integration and completion. Finally, the complete tree structure is exported as editable three-dimensional mesh models (e.g., .obj), suitable for various virtual scenarios, demonstrating the flexibility and practicality of this modeling method.

Keywords: Point Cloud, L-system, 3D Reconstruction, Tree- Modeling, Structure-from-Motion, Computer Graphics

目錄

表目錄

圖目錄

第一章　緒論

1.1　研究動機

隨著計算機視覺與三維重建技術的快速發展，結構重建（Structure-from-Motion, SFM）已成為一種高效且廣泛應用的技術，能夠從多視角影像中生成高精度的三維點雲模型。然而，SFM 所產生的點雲資料在處理包含複雜幾何與豐富細節的自然場景時，經常面臨結構缺失與重建誤差，特別是樹葉、樹枝等細部構造。

樹木作為自然環境的重要組成部分，其結構複雜且變化多樣，傳統的建模方法難以在效率與完整性之間取得平衡。因此，如何利用現有技術對樹木進行快速建模與補全，成為了一項具有挑戰性的研究課題。利用手機或相機等設備進行拍攝，相對容易且成本低廉，能夠方便地取得多視角影像資料。基於這類影像進行樹木建模的方法在成本上更具彈性，特別適合針對特定樹種進行建模的需求。因此，若能研究出一種快速重建、品質良好且風格貼近目標樹木的建模方法，將具有大量生產可用資源的應用潛力。

1.2 研究目的

本研究旨在結合 SFM 技術生成的三維點雲數據與 L-system（Lindenmayer 系統）進行樹木模型的生成與補全。目標如下：

1. 利用 SFM 技術生成樹木的三維點雲數據，並對點雲進行降噪與預處理。
2. 探討現有基於點雲的樹木重建演算法（如 AdTree 與 SimpleTree），提取主要樹幹骨架，生成初步網格模型。
3. 結合 L-system 對樹木結構進行建模，修正 SFM 生成點雲中的細小分枝與樹葉分布，並補全修正區域，提升模型的結構完整性。
4. 將樹木模型轉化可編輯的網格（mesh），成為可以應用於多媒體領域的三維模型資產。

透過結合上述方法，本研究希望在樹木建模中面臨的分支結構誤判與不完整性問題，提供一種快速且靈活的解決方案。

第二章　文獻探討與相關工作

2.1 Structure From Motion

Structure From Motion（SFM）[13] 是一種自動重建三維結構的技術，核心原理是從多張不同角度拍攝的影像中提取特徵點，透過特徵匹配與相機姿態估計，進而重建場景的三維幾何結構 [15]。由於 SFM 無需事先提供場景的幾何資訊，且具備操作簡便、成本低廉的特性，因此廣泛應用於文化資產保存、地形建模與建築掃描等領域。

隨著電腦視覺與計算能力的進步，已發展出許多穩定而成熟的 SFM 實作，如 VisualSfM[16]、OpenSFM[11]、Colmap[14] 等，可提供完整的影像對齊、稀疏點雲重建與密集重建流程。這些工具大多內建關鍵點偵測（如 SIFT[10]）、匹配過濾（如 RANSAC[7]）、與增量式或全域式重建流程，能夠有效處理中小型場景的三維重建任務。

然而將 SFM[13] 應用於含有樹木之自然場景時，會面臨以下挑戰，例如：(1) 密集的枝葉容易造成遮蔽與遮擋，使部分結構無法被完整觀測；(2) 枝葉晃動導致相鄰影像間出現特徵點匹配錯誤。為提升此類場景的重建品質，需輔以其他策略與處理方法。

2.2 基於點雲的樹木重建方法

2.2.1 Simple Tree

SimpleTree [8] 是一種開源工具，可以從地面雷射掃描（TLS）點雲數據中建模出高精度的圓柱體樹木模型。此演算法主要步驟為先導入透過地面雷射掃描獲得的樹木點雲資料，接著 SimpleTree 會檢測樹幹點，利用主成分分析（PCA）在體素網格過濾的輸入雲的每個輸入點鄰域上執行。演算法會評估特徵值，若滿足使用者隱藏的閾值，則該點會被接受為初步檢測到的樹幹點。在這些點上執行聚類分析，輸出會被標記為進一步檢測到的樹幹點，並以此建立樹幹模型。

SimpleTree 利用球體表面切割點雲的方法，遞迴地搜尋和檢測樹木的分支結構。以樹木骨架上的點為球心，以大於底層分支段的半徑的球體，來提取球體表面上的子點雲，如果球體不在分支的尖端，則子點將表示底層分支段的橫截面積，橫截面積是圓形結構。透過圓擬合，確定分支的圓柱體參數(大小、方向)，並使用非線性最小平方法擬合來提高圓柱體擬合的精度。SimpleTree 亦會對幾何擬合方法進行統計上的改進，以解決在較細分支中被高估的問題。基於檢測到的分支和樹幹，構建完整的樹木結構模型，並按照拓撲順序組織。

SimpleTree 適用於多種樹種，包括常綠樹、針葉樹等。經驗證在針葉樹上的效果最佳。SimpleTree 的研究意義在於能夠生成高精度的樹木模型，可用於估算森林參數，如地上生物量（AGB）、樹冠形狀和間隙率等生態特性。

2.2.2 AdTree

AdTree[5] 是一種使用激光掃描取得點雲數據，並以此進行樹木建模的技術。可以從點雲數據中提取骨架結構，生成詳細的樹木模型。該方法利用點雲數據的空間分佈特徵，結合樹木的拓撲結構，實現主要樹木骨架的提取。AdTree 的主要優勢在於其高精度與其效率，特別適用於處理高密度的 LiDAR 點雲數據。然而，該方法對於稀疏點雲或樹冠茂密的點雲數據，可能存在精度下降的問題。此外，AdTree 的演算法是專注於處理 LiDAR 的數據，對於基於 SFM[13] 技術生成的點雲，可能需要進一步調整。

AdTree 的處理流程可分為以下三個主要階段：

1. 骨架初始化 (Skeleton initialization)

   首先對輸入點雲進行 Delaunay Triangulate[6] 建立初始圖形。根據邊 (Edge) 在三維空間中的實際長度作為權重。接著應用 Dijkstra 最短路徑演算法 [3] 在初始圖形上計算出 MST(最小生成樹)，並以此作為初始樹木骨架。為了改善後續提取的骨架品質，會使用 mean-shift 演算法來集中主幹點，以計算後續節點之權重。

2. 骨架簡化 (Skeleton simplification)

   初始樹木骨架通常包含大量冗餘的頂點和邊，因此需要簡化。根據節點相對於子頂點與其邊的長度，來評估其重要性。使用 Douglas-Peucker 演算法 [4] 合併距離在特定閾值內的相鄰頂點。對於擁有多個子節點的頂點，也會根據其子節點之間的角度相似性，計算出新的合併位置。

3. 樹枝擬和 (Branch fitting)

簡化後的骨架會經過圓柱體擬合的過程。通過將一系列圓柱體擬合到圖形節點上，來近似樹的幾何形狀。首先分割和識別樹的不同部分，然後通過非線性最小平方法來獲得準確的樹幹半徑。其餘樹枝的半徑則從主幹推導而來。

2.3 L-System

L-system[12]（Lindenmayer system 是一種形式語法（formal grammar）系統，由匈牙利生物學家 Aristid Lindenmayer 所提出。它可以用來模擬植物或其他有機結構的生長過程。透過簡單的字串替換規則並以遞迴方式擴展，能夠生成複雜的碎形結構。L-system 特別適合描述具有自相似性與層次性的幾何形狀，因此被廣泛應用於植物建模與程序式圖形生成。

- V (字母表)：可替換的變數（如 F、X）
- ω（axiom）：初始字串，作為成長的起始狀態
- P（rules）：重寫規則，用於描述每一代的生長方式

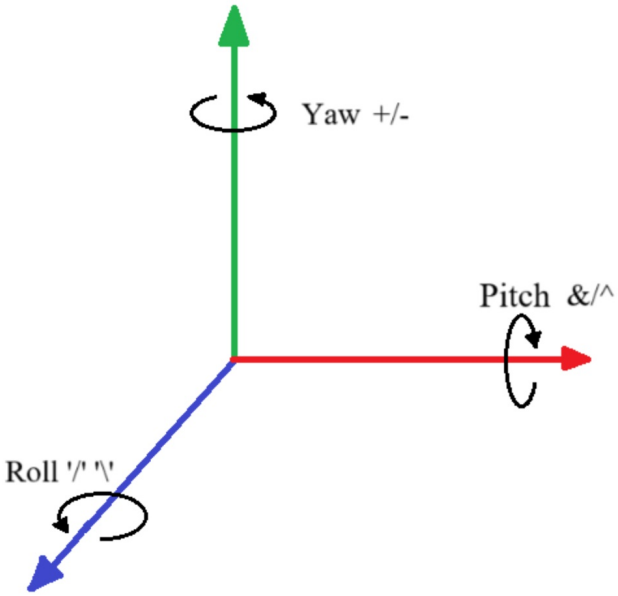

Figure 2.1: 三維空間的旋轉

Table 2.1: Lsystem 常用符號表

符號	操作
F	Forward
+	Turn left by a specific angle
-	Turn right by a specific angle
&	Pitch up by a specific angle
^	Pitch down by a specific angle
\	Roll left by a specific angle
/	Roll right by a specific angle
[	push current state(position and angle)
]	retreive prvious state

L-system 語法迭代範例，符號參考表2.1，圖2.2為每次迭代生成的結構

設 angle:30, axiom : X, rules : X=F[-X][+X]

iteration 1 : F[-X][+X]

iteration 2 : F[-F[-X][+X]][+F[-X][+X]]

iteration 3 : F[-F[-F[-X][+X]][+F[-X][+X]]][+F[-F[-X][+X]][+F[-X][+X]]]

iteration 4 : 略

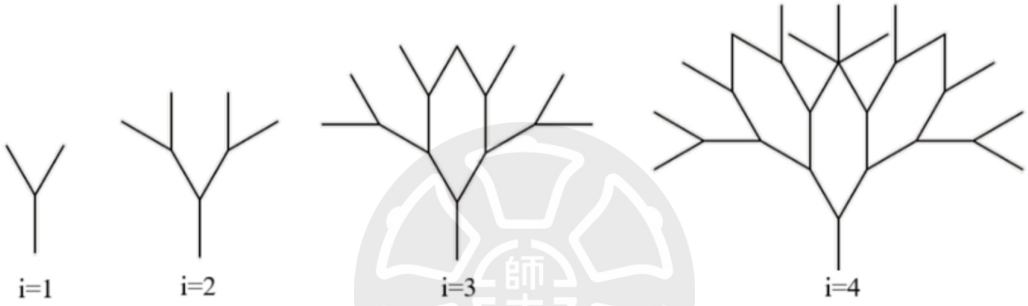

Figure 2.2: Lsystem 迭代範例

第三章　研究議題與方法

3.1　問題描述

AdTree 演算法原本設計用於高密度且準確的 LiDAR 點雲資料，但在應用於由 SFM 還原的點雲時，可能誤將來自樹葉的點雲判斷為分枝。而樹枝、樹葉搖晃產生的錯誤匹配點雲，有可能形成過度密集的分支結構。此外，若視角覆蓋不足亦可能產生點雲缺失，進一步影響骨架重建的正確性。本研究旨在克服上述限制，重建出結構合理、外觀自然的樹木模型。

- 輸入: 由多視角相片經 SFM 還原之點雲結構
- 輸出: 符合相片樹木風格之模型網格

3.2　方法與步驟

為了支援使用者輸入檔案並執行特定函數，我們設計了一個可互動的程式介面 (圖3.1)，詳細操作手冊可見於附錄 A，其系統運作如圖3.2所示，啟動時會先完成環境設定與資源載入，接著進入主循環，依序處理滑鼠與鍵盤輸入、更新 GUI 狀態、根據使用者操作及 GUI 互動 (如透過檔案對話方塊載入檔案)，系統會即時執行指定函數，並將處理結果渲染至場景視窗。

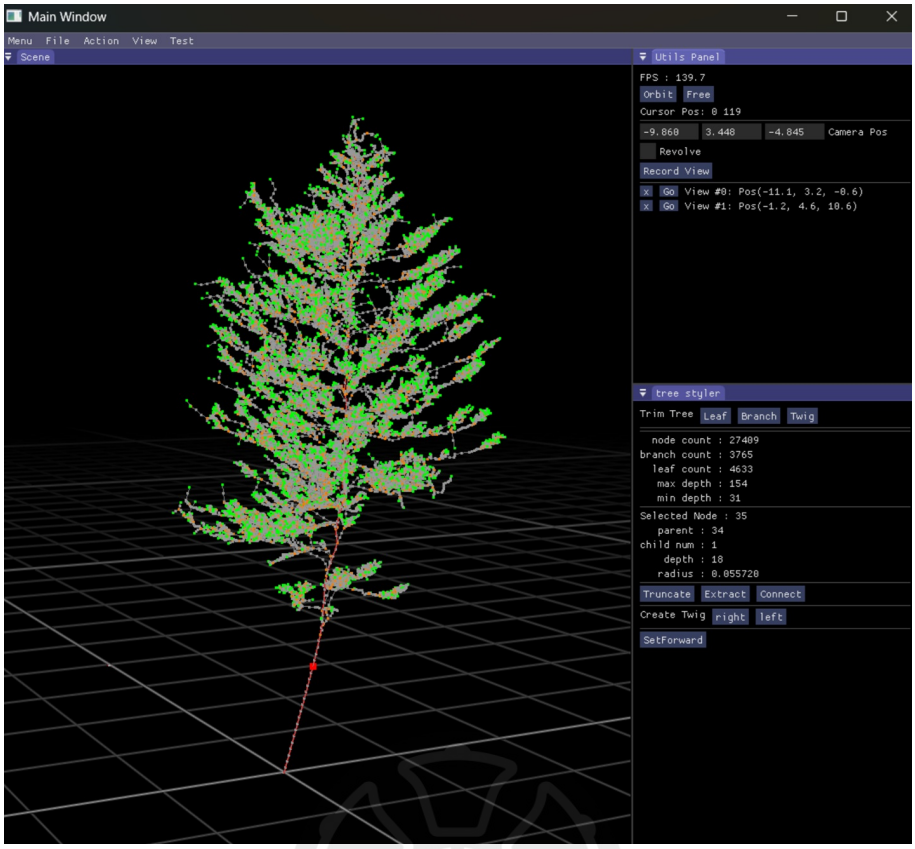

Figure 3.1: 主程式介面

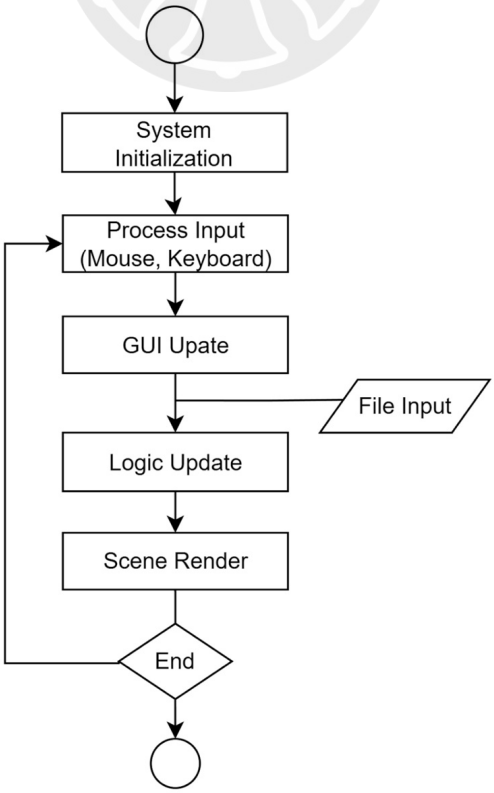

Figure 3.2: 程式執行迴圈

首先簡要說明整個重建流程，再於後續章節講述細部實作:

1. 讀取目標樹木相片組由 SFM 技術 [13] 還原之三維點雲資料，並對其進行降噪、移除離群值等預處理。
2. 利用 AdTree[5] 從點雲中提取樹木結構，作為初步骨架。
3. 對初步骨架以程序規則進行多餘分支的自動修剪，同時也支援手動指定節點進行編輯。
4. 從骨架節點擷取出子樹語法後，指定節點連接。
5. 結合 L-system 模擬樹木末端的葉片樣式與分布，豐富指定樹葉區域 (末端節點) 風格，提升整體模型的完整性。
6. 將重建完成的樹木轉換為可編輯的網格模型（mesh）格式，便於後續應用與編輯作業。整體重建流程如圖3.3所示:

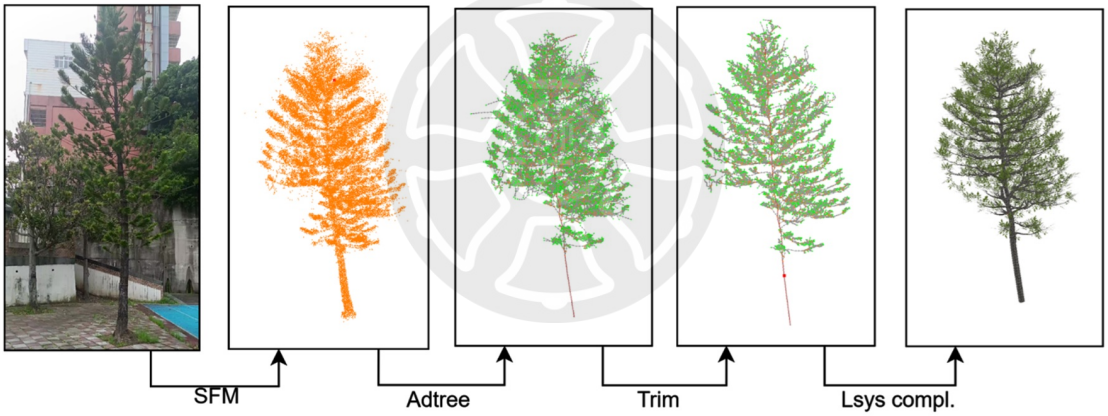

Figure 3.3: 重建流程簡圖

3.2.1 SFM 產生之點雲處理

對目標樹木進行環繞拍攝後，將相片組傳入 Colmap 程式 [14](SFM 實作) 處理，得到包含環境物件之點雲，其中每個點雲包含以下資訊 (x,y,z,nx,ny,nz,r,g,b)。如圖3.5所示。目前我們需要手動將屬於目標樹木的部分挑選出來以獲得初始點雲。

Figure 3.4: 由相片組還原相機視角與位置

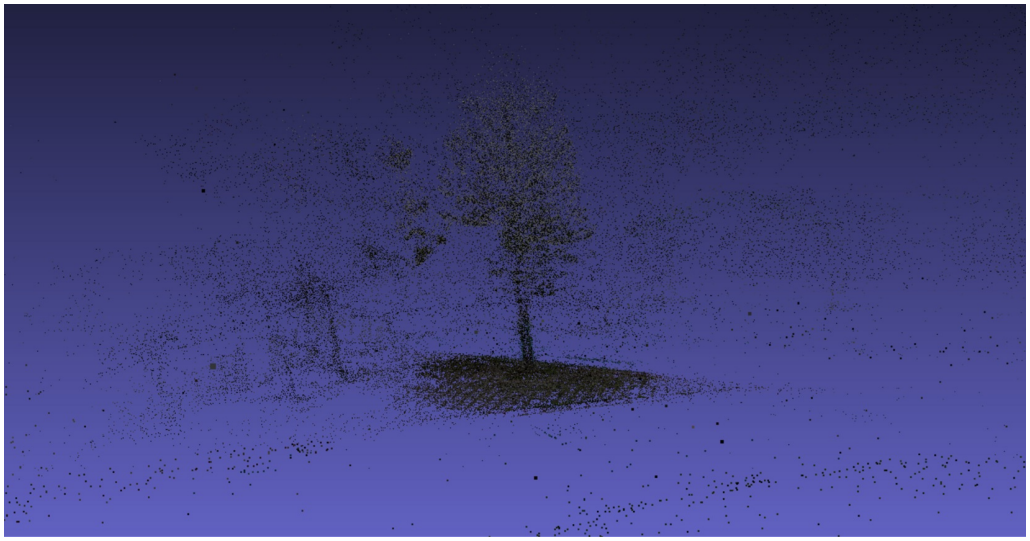

Figure 3.5: 經 colmap 重建後之環境點雲

針對目標樹木的點雲資料，首先進行過濾操作，刪除雜訊與離群點，避免影響後續骨架提取的品質。另外將顏色接近全白或全黑的點視為雜訊點予以剔除；離群點雲部分則透過 KNN 演算法 [2] (由 MeshLab[1] 編輯器提供) 進行偵測，計算每個點與其鄰近點之間的平均距離，判定是否為離散點予以移除，以提升點雲結構的整體性。

3.2.2 提取樹木骨架與解析

將初始樹木點雲經由 AdTree 程式處理後，會得到一 PLY 檔，經程式處理後可以提取出一個無環圖（No-Cycle）的結構，即樹狀圖。該結構中的每個節點包含父子關係與半徑資訊。

Figure 3.6: 由 AdTree 取得之骨架 (此點雲資料來自 Adtree 資料集)

本研究設計了自訂的 Tree 與 TreeNode 資料結構，用來儲存從 AdTree 所擷取的骨架，並支援多項操作，如樹木修剪、轉換為 L-system 語法、子樹連接、節點編輯與統計分析等（詳細內容見後續章節）。每個節點記錄其空間位置、半徑、深度、父子關係以及是否為主幹節點，儲存完整的結構資訊以支援後續幾何建模操作與可視化應用。

```
struct TreeNode {
    int id;
    int depth;
    vector3 position;
    float radius;
    bool isTrunk;
    TreeNode* parent;
    vector<TreeNode*>  children;
};
```

Table 3.1: TreeNode 結構說明

欄位	類型	說明
id	Integer	由系統自動遞增分配節點的唯一識別碼，利用 id 與節點的對應 (map) 可以在全域中快速引用
depth	Integer	該節點在整棵樹中的深度（從 root 為 1，往下遞增）。有助於分析樹的層次結構
position	Vector3	表示節點在三維空間的位置
radius	Float	代表該節點（連接段）的粗細，用於視覺化樹幹的寬度
isTrunk	Boolean	是否作為主幹的一部分，區分主要枝幹與分枝
parent	TreeNode Pointer	指向其父節點的指標。若為根節點則為 nullptr
children	vector	指向所有子節點的指標容器，允許多個子節點

```
struct Tree {
    TreeNode* root;
    vector<TreeNode*> Nodes;
    vector<TreeNode*> LeafNodes;
    vector<TreeNode*> BranchNodes;
    vector<TreeNode*> TrunkNodes;
    vector<TreeNode*> CommonNodes;
};
```

Table 3.2: Tree 結構說明

欄位	類型	說明
root	TreeNode Pointer	指向整棵樹的根節點指標
Nodes	vector	儲存樹中所有節點的指標，用於快速遍歷與統計
LeafNodes	vector	儲存所有葉節點，即沒有子節點的終端節點
BranchNodes	vector	儲存所有分叉點，即擁有兩個以上子節點的節點
TrunkNodes	vector	儲存被判定為主幹（trunk）的節點。
CommonNodes	vector	儲存僅有一個子節點且未被標記為主幹的節點，在結構簡化、樣式轉換通常是可以合併的候選對象。

為提升資料處理效率與支援即時互動編輯，本系統於樹木建構過程中即執行節點分類，將節點依其結構特性標記為葉節點、分叉節點或為主幹節點，並透過演算法1建立對應的索引結構。此設計可避免每次操作時皆需自根節點進行完整遞迴遍歷，特別是在將節點資料上傳至 OpenGL context 進行即時渲染時，能顯著提升系統效能與互動響應速度。此外，系統亦支援以不同顏色區分節點類型，協助使用者更直觀掌握樹木結構與編輯變化。如圖3.7所示，綠色點為葉節點、橘色點為分支節點、紅色線段為主幹節點及其對應的連接邊。

接續於結構建構階段，透過演算法2根據每個節點的子節點半徑，標記是否為主幹節點。此機制為後續套用 L-system 預設參數進行樹木結構擴展與建模之基礎。

Figure 3.7: 節點分類視覺化

Algorithm 1 TreeNode Classification and Depth Assignment

Require: root (TreeNode)
Ensure: AllNodes, LeafNodes, BranchNodes, CommonNodes

```
if root is null then
    return
end if
AllNodes, LeafNodes, BranchNodes, CommonNodes ← ∅
root.depth ← 1
queue ← {root}
while queue ≠ ∅ do
    current ← queue.dequeue()
    current.isTrunk ← False
    AllNodes ← AllNodes ∪ current
    depth ← current.depth
    childCount ← current.children.size()
    if childCount == 0 then
        LeafNodes ← LeafNodes ∪ current
    else if childCount == 1 then
        CommonNodes ← CommonNodes ∪ current
    else
        BranchNodes ← BranchNodes ∪ current
    end if
    for all child in current.children do
        child.depth ← current.depth +1
        queue.enqueue(child)
    end for
end while
```

演算法 1 負責分析整棵樹的結構，進行節點分類。演算法採用廣度優先搜尋（BFS），從根節點開始逐層遍歷。每個節點被處理時，會先標記為非主幹節點，加入至節點集合 AllNodes。根據子節點數目分類為葉節點、一般節點或分叉節點。接著將子節點深度設為父節點深度加一，推入佇列繼續處理。

Algorithm 2 Trunk Identification Based on Radius and Direction Thresholds

Require: root (TreeNode), radiusThreshold, degreeThreshold
Ensure: TrunkNodes

if root is null **then**
  return
end if
stack $\leftarrow$ {root}
TrunkNodes $\leftarrow$ {root}
while stack $\neq \emptyset$ **do**
  node $\leftarrow$ stack.pop()
  if node has no children **then**
    continue
  end if
  if node has only one child **then**
    TrunkNodes $\leftarrow$ TrunkNodes $\cup$ root
    stack.push(child)
    continue
  end if
  baseChild $\leftarrow$ child with largest radius
  maxRadius $\leftarrow$ baseChild.radius
  baseDir $\leftarrow$ normalize(baseChild.position $-$ node.position)
  trunkCount $\leftarrow 0$
  for all child in node.children **do**
    dir $\leftarrow$ normalize(child.position $-$ node.position)
    cosAngle $\leftarrow$ dot(baseDir, dir)
    rRatio $\leftarrow$ child.radius / maxRadius
    if $rRatio \geq$ radiusThreshold $\wedge$ $cosAngle \geq$ cos(degreeThreshold) **then**
      child.isTrunk $\leftarrow$ True
      TrunkNodes $\leftarrow$ TrunkNodes $\cup$ child
      stack.push(child)
      trunkCount++
    else
      child.isTrunk $\leftarrow$ False
    end if
  end for
  if trunkCount == 0 and baseChild exists **then**
    baseChild.isTrunk $\leftarrow$ True
    TrunkNodes $\leftarrow$ TrunkNodes $\cup$ baseChild
    stack.push(baseChild)
  end if
end while

演算法2用於標記骨架中的主幹節點，從根節點（root）出發，首先將 root 標記為主幹並放入堆疊中，接著以深度優先方式 (DFS) 遍歷整棵樹。對每個節點，若僅有一個子節點，則視為主幹延伸，繼續往下標記。若有多個子節點，則選擇半徑最大的節點作為主幹（baseChild），並以其方向向量作為參考基準（baseDir）。其他子節點若同時滿足以下兩項條件，則視為主幹分支：

- 半徑條件：子節點半徑大於 baseChild 的半徑乘上比例閾值 radiusThreshold。
- 方向條件：子節點與 baseDir 的夾角餘弦值需大於 cos(degreeThreshold)。

若有子節點同時符合上述條件，則標記為主幹節點並加入堆疊進行處理；若無候選主幹節點則退回選擇 baseChild 作為主幹節點。

3.2.3 樹木修剪

AdTree 在三維點雲資料中擷取樹木骨架的過程中，仍有部分離群點雲可能被誤判為細長枝條並納入結構之中，導致重建結果出現不自然、延伸過長的細枝（twig）。為了解決此問題，我們設計了自動修剪機制 (演算法3)，移除因離群點誤判而產生的非預期枝條。提升樹木結構的生長合理性與視覺平衡，同時消除多餘節點所帶來的運算負擔。

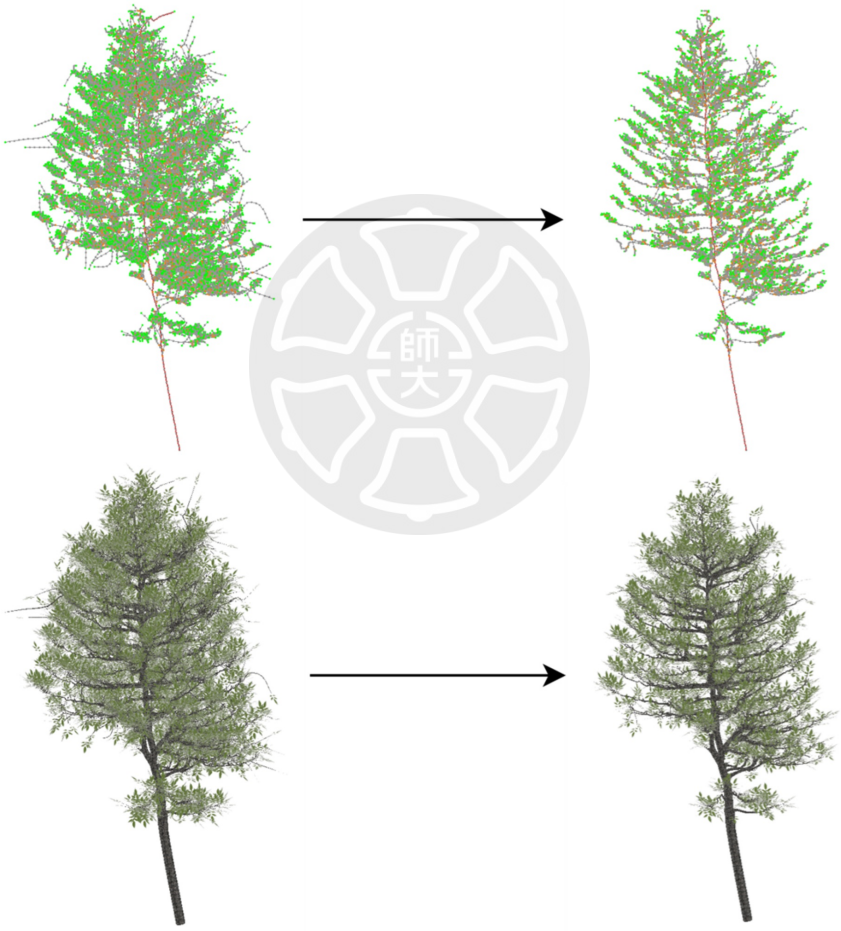

Figure 3.8: 經修剪後的樹木結構

Algorithm 3 Twig Pruning Based on Average Leaf Depth

Require: LeafNodes, TreeNode structure with parent/children
Ensure: Truncate all leaf paths significantly deeper than average

```
Initialize DepthMap ← ∅ {// map<TreeNode*, int>}
totalDepth, validCount ← 0
for all leaf in LeafNodes do
    current ← leaf
    depth ← 0
    while current.parent exists do
        depth ← depth +1
        current ← current.parent
        if current.children.size() ≥ 2 then
            DepthMap[leaf] ← depth
            totalDepth ← totalDepth + depth
            validCount ← validCount + 1
            break
        end if
    end while
end for
if validCount == 0 then
    return
end if
avgDepth ← ⌊ totalDepth / validCount ⌋
TruncateTargets ← ∅
for all (leaf, depth) in DepthMap do
    if depth > avgDepth then
        overshoot ← depth − avgDepth
        current ← leaf
        for i = 0 to overshoot−1 do
            if current has parent then
                current ← current.parent
            end if
        end for
        TruncateTargets ← TruncateTargets ∪ current
    end if
end for
for all node in TruncateTargets do
    Truncate(node)
end for
```

演算法3負責修剪過長的末端子樹。首先遍歷所有葉節點，從葉節點向上回溯，直到第一個分叉節點（即擁有兩個以上子節點者），並依據經過的節點數得出 twig depth。同時累加所有 twig depth 的總和與節點數量，以計算平均深度。接著對於 twig depth 超過平均值的葉節點，根據其超出部分（overshoot）向上回溯指定層數，定位出應修剪的節點加入至修剪清單。最後統一執行 Truncate 操作，移除這些不自然延伸的枝條，並更新整體樹結構資訊。

3.2.4 子樹擷取與連接

在實際應用中，目標樹木周圍常因遮蔽物阻擋，導致無法進行完整環繞拍攝，使重建出的點雲資料出現大範圍缺失。為補足這些區域，本研究提出一套以「子樹結構重用」為核心的修補流程：

1. 從現有樹結構中選取一個完整子樹並轉換為 L-system 字串 (演算法 4)，以保留其分支風格。使用者可手動指定缺失位置，由演算法 5 產生新節點供後續子樹連接，其方向根據相機視角自動推估。
2. 接著將導出的 L-string 還原成子樹並接回原樹木 (演算法 7)。為避免新生成的子樹與原始結構過於相似，引入隨機擾動機制，對分支長度與角度進行適度變異以提升自然感 (演算法 6)。

期望由此機制可補全點雲區域缺失，同時維持整體風格一致性與結構多樣性。具體流程如下: [點雲資料中存在缺口]→[指定節點預覽子樹結構]→[轉換為 L-system 字串]→ [指定位置生成分支作為連接點]→[還原為有差異性之子樹並連接至主樹]→ [完成結構補全，更新整體樹型]

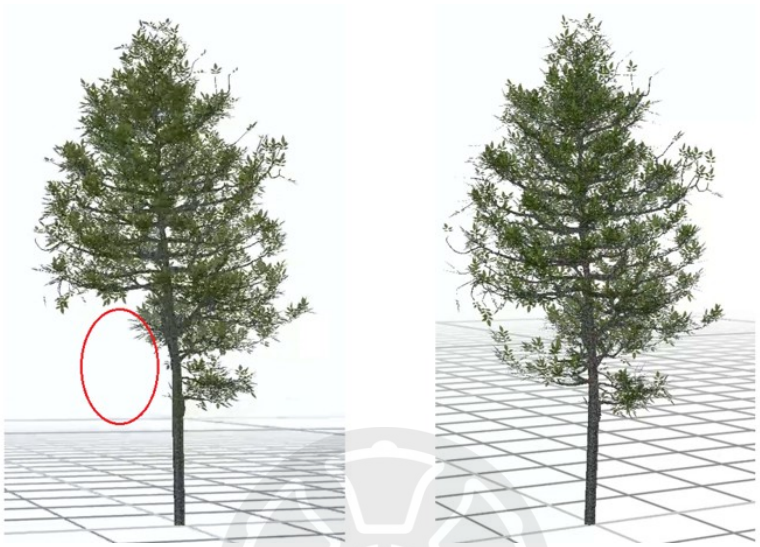

Figure 3.9: 重用子樹結構補全缺失區域

Algorithm 4 L-System String Conversion from Tree Structure

Require: TreeNode $node$
Ensure: Return L-system formatted string from tree traversal

if node is null **then**
  return empty string
end if
result ← empty string
for all child in node.children **do**
  direction ← normalize(child.position − node.position)
  distance ← length(child.position, node.position)
  yaw ← atan2(direction.z, direction.x)
  pitch ← atan2(direction.y, $\sqrt{direction.x^2 + direction.z^2}$)
  Convert yaw, pitch from radians to degrees
  orientation ← empty string
  if yaw $>$ 0 **then**
    orientation ← orientation + ” + ” + yaw
  else if yaw $<$ 0 **then**
    orientation ← orientation + ” − ” + $|yaw|$
  end if
  if pitch $>$ 0 **then**
    orientation ← orientation + ”&” + $pitch$
  else if pitch $<$ 0 **then**
    orientation ← orientation + ”^” + $|pitch|$
  end if
  forward ← ”F(” + distance + ”)”
  subTreeStr ← **WriteToLstring**(child)
  result ← result + [orientation + forward + subTreeStr]
end for
return result

演算法4從給定的節點出發，遞迴地處理其所有子節點，並依據每個子節點與父節點之間的位置關係，產生一組描述旋轉角度與前進距離的 L-system 指令。對於每一對父子節點，可以得出其前進方向 d 及兩個旋轉角度：

- Yaw（偏航角）：節點在水平面上的旋轉，由 atan2(d.z, d.x) 計算得出
- Pitch（俯仰角）：節點在垂直方向的旋轉，由 atan2(y, sqrt($d.x^2 + d.z^2$)) 得出

這些角度會被轉換成 L-system 指定角度的旋轉指令:

- +(angle)：向右轉
- -(angle)：向左轉
- &(angle)：向下傾斜
- ^(angle)：向上傾斜

依據父子節點之間的距離建立前進指令：F(distance)：往前延伸一段距離，其中'[' 和']' 為分支控制符號，用於保存與恢復當前的狀態（包含位置與方向）:

- [：將當前狀態推入堆疊（開始一個新的分支）
-]：從堆疊中恢復先前狀態（結束當前分支，回到分叉點）

透過這些指令組合，可精確描述樹木的階層結構與空間方向，產生對應整棵子樹的 L-system 字串。

Algorithm 5 Create Twig for SubTree Connection

Require: TreeNode $node$, boolean $rightSide$
Ensure: Append a small side twig to the node for L-system connection

if node is null ∨ node has no children **then**
  return
end if
next ← node.children[0]
trunkDir ← normalize(next.position − node.position)
twigLength ← distance(node.position, next.position)
nodeToCam ← normalize(cameraPosition − node.position)
if rightSide is true **then**
  sideDir ← normalize(cross(nodeToCam, trunkDir))
else
  sideDir ← normalize(cross(trunkDir, nodeToCam))
end if
if length(sideDir) is nearly zero **then**
  return
end if
newTwigDir ← normalize(sideDir + trunkDir × 0.3)
newPos ← node.position + newTwigDir × twigLength × 0.6
newRadius ← node.radius × 0.8
newTwig ← **new TreeNode**(newPos, newRadius)
node.AddChild(newTwig)
newTwig.parent ← node

演算法5 的目的是在指定節點上延伸出一小段側枝，作為後續子樹的連接點。使用者可指定側枝生成於主幹的左側或右側。函數會先確認該節點存在且具有子節點，接著根據攝影機位置，透過主幹方向與節點至攝影機的方向向量進行外積，計算出側枝的生成方向。若方向有效，則將主幹方向與側向方向加權混合，據此推算出新枝節點的方向與位置，完成側枝擴展。

Algorithm 6 Build Tree from L-System String

Require: Lstr, degreeDiff, distDiffPercent
Ensure: Reconstruct tree from L-system with randomness

randAngle ← uniform($-degreeDiff$, $+degreeDiff$)
randScale ← uniform($1 - distDiffPercent$, $1 + distDiffPercent$)
Initialize empty stack for TurtleState
Create root node at origin with default radius
currentNode ← root
currentPos ← origin
yaw, pitch ← 0
$i \leftarrow 0$
while $i <$ length(Lstr) **do**
    $c \leftarrow$ Lstr[i]
    if $c ==$ 'F' **then**
        if Lstr[$i + 1$] == '(' **then**
            dist ← ParseFloatFromParen(Lstr, i)
        end if
        length ← dist × randScale()
        yawRad ← radians(yaw + randAngle())
        pitchRad ← radians(pitch + randAngle())
        $x \leftarrow \cos(pitchRad) \times \cos(yawRad)$
        $y \leftarrow \sin(pitchRad)$
        $z \leftarrow \cos(pitchRad) \times \sin(yawRad)$
        dir ← normalize(x, y, z)
        newPos ← currentPos + dir × length
        newNode ← new TreeNode(newPos)
        currentNode.AddChild(newNode)
        currentNode ← newNode
        currentPos ← newPos
    else if $c ==$ '+' **then**
        yaw ← ParseFloatFromParen(Lstr, i)
    else if $c ==$ '-' **then**
        yaw ← $-$ ParseFloatFromParen(Lstr, i)
    else if $c ==$ '&' **then**

```
        pitch ← ParseFloatFromParen(Lstr, i)
    else if c == '^' then
        pitch ← − ParseFloatFromParen(Lstr, i)
    else if c == '[' then
        Push (currentNode, currentPos, yaw, pitch) to stack
    else if c == ']' then
        Pop (currentNode, currentPos, yaw, pitch) from stack
    end if
    i ← i + 1
end while
return root
```

演算法6實現了一個具有隨機性擴展的 L-system 解釋器 (Parser)。將 L-string 轉換成由 TreeNode 構成的樹結構，同時加入適度的隨機變異，以模擬自然中樹木生長的不規則性。使用「烏龜繪圖」（Turtle Graphics）的原理來進行解析，用一個 TurtleState 堆疊來支援分支回溯機制。每次解析 F 指令時，會根據目前設定的旋轉角度（yaw, pitch）與前近距離 (dist) 計算出新的節點位置，並依據參數 degreeDiff 和 distDiffPercent 引入角度和長度上的隨機擾動，避免擷取出的子樹結構完全相同。

Algorithm 7 Connect L-System Subtree to Main Tree

```
Require: node, subtreeNode
Ensure: Attach subtree to node with correct orientation and offset
if node is null ∨ node.parent is null ∨ subtreeNode is null then
    return
end if
direction ← Normalize(node.position − node.parent.position)
subtree.SetToLocalSpaceOrigin()
subtree.SetForward(direction)
offset ← node.position
queue ← empty queue
Push subtreeNode into queue
while queue is not empty do
    current ← queue.pop()
    current.position ← current.position + offset × 1.001
    for all child in current.children do
        queue.push(child)
    end for
end while
node.AddChild(subtree)
```

```
18: subtree.parent ← node
19: subtree ← null
20: SmoothRadius(node, 0.8)
```

演算法7，用於將子樹（subtree）接合至現有樹結構的葉節點。操作前會檢查指定葉節點、其父節點與子樹的有效性，確保資料完整。接著計算葉節點的延伸方向，對齊並重設子樹，使其方向一致並消除誤差。透過廣度優先搜尋（BFS），將屬於該子樹的節點平移至以指定葉節點為基準的位置。完成子樹與指定節點的連接。為避免接合處出現視覺上的不和諧，系統亦進行節點半徑的平滑處理 (演算法8)。

Algorithm 8 Smooth Branch Radius from a Node

```
Require: startNode, ratio
 1: if startNode is null then
 2:     return
 3: end if
 4: baseRadius ← startNode.radius
 5: Call RecursiveSmooth(startNode, ratio, baseRadius, 0)
 6:
 7: Function RecursiveSmooth(currentNode, ratio, baseRadius, depth):
 8: if currentNode is null then
 9:     return
10: end if
11: currentNode.radius ← baseRadius × ratio^depth
12: for all child in currentNode.children do
13:     Call RecursiveSmooth(child, ratio, baseRadius, depth+1)
14: end for
```

演算法8將節點進行半徑平滑處理，演算法會從指定的節點出發，遞迴拜訪所有子節點，並將每個節點的半徑依據深度乘上遞減比例，達到節點深度越深，半徑越細的自然效果。

3.2.5 定義樹葉分布

本研究進一步將樹木骨架中的葉節點接合上特定的 L-system 子樹，以模擬末端枝條的葉片分布結構。這些子樹可針對目標樹種進行 L-system 建模，根據實際植物的葉序特性（如對生、互生、輪生）生成對應的枝葉排列，進一步提升樹木模型的視覺表現。

以下為常見葉序類型之 L-system 語法範例：

- 對生（Opposite）：F[+F][-F]F[+F][-F]F[+F][-F]F

  葉片成對生長於枝條兩側，間隔均勻，常見於楓樹等植物。

- 互生（Alternate）：F[+F]F[-F]F[+F]F[-F]F

  葉片沿枝條交錯排列，左右輪替，見於櫻花等樹種。

透過此方式，每個葉節點可被視為連接起點，生長符合樹種特徵的末端結構，實現可擴展、具生態多樣性的程序式植物生成系統。

Figure 3.10: 樹葉節點連接不同 Lstring 葉子子樹

3.2.6 建立網格

為了將目標樹木進行可視化，渲染於場景視窗中。必須建立包含位置、法向量、UV 位置的頂點網格。生成樹木結構的網格時，從根節點出發，對樹結構進行深度優先搜尋（DFS），以取得從各分支至每個葉節點的路徑。這些節點路徑可視為空間中的曲線，依此進行樹木分支的網格構建。若直接沿此曲線建立圓柱截面並連接，容易在曲線彎折或切線變化劇烈時，截面方向可能產生扭轉，造成幾何結構出現明顯的割裂與貼圖扭曲。

為了解決此問題，本研究採用 Parallel Transport Frames (PTF) 方法建立網格 [9]，在曲線上的每一點構建穩定且平滑變化的局部座標系，即包含切線（tangent）、法線（normal）與副法線（bitangent）互相垂直的三個向量。在任意曲線（甚至近似折線）上穩定運作，有效避免截面因切線旋轉導致的扭轉問題。透過此方法，每段樹枝管狀結構在節點間皆可取得一致且平滑的截面對齊，提升網格的品質與視覺連貫性，便於後續貼圖與模型導出。

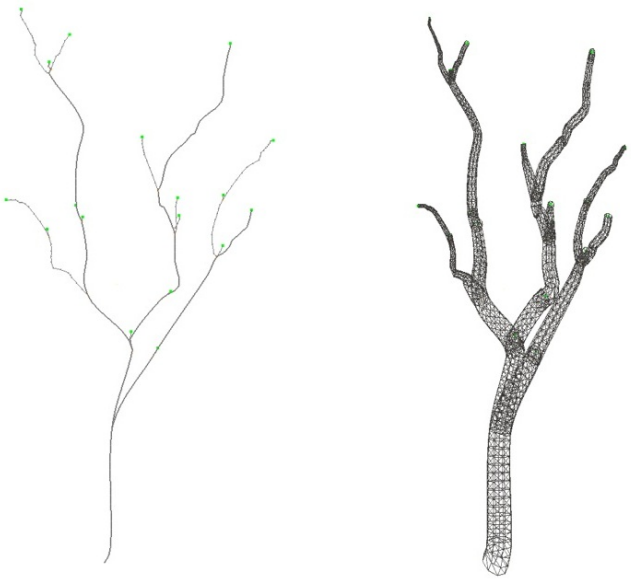

Figure 3.11: 由 PTF 建立之管狀網格

Algorithm 9 Generate Tree Mesh using Parallel Transport Frames

Require: path: a list of TreeNode, each with position and radius
Require: segments: number of sides per circular cross-section
Ensure: mesh: a triangle mesh representing the branch

if length(path) < 2 **then**
  return an empty mesh
end if
Initialize an array `frames` of size equal to path length
for $i = 0$ **to** length(path) -2 **do**
  $p_0 \leftarrow$ path[i].position, $p_1 \leftarrow$ path[$i+1$].position
  $T_1 \leftarrow$ normalize$(p_1 - p_0)$
  if $i = 0$ **then**
    Choose reference vector ref: $(0, 1, 0)$ if not parallel to T_1, otherwise $(1, 0, 0)$
    $N \leftarrow$ normalize$(T_1 \times ref)$
    $B \leftarrow$ normalize$(T_1 \times N)$
    frames[0] $\leftarrow (T_1, N, B)$
  end if
  $T_0 \leftarrow$ frames[i].tangent, $N_0 \leftarrow$ frames[i].normal
  $v \leftarrow T_0 \times T_1$, $c \leftarrow T_0 \cdot T_1$, $s \leftarrow \|v\|$
  if $s < \varepsilon$ **then**
    $N_1 \leftarrow N_0$
  else
    $axis \leftarrow$ normalize(v)
    $\theta \leftarrow \arctan 2(s, c)$
    $N_1 \leftarrow$ Rodrigues(N_0, $axis$, θ)
  end if
  $B_1 \leftarrow$ normalize$(T_1 \times N_1)$
  frames[$i+1$] $\leftarrow (T_1, N_1, B_1)$
end for
Initialize empty list of circles
for $i = 0$ **to** length(path) -1 **do**
  $center \leftarrow$ path[i].position, $r \leftarrow$ path[i].radius
  Retrieve frame at i: (T, N, B)
  for $j = 0$ **to** segments -1 **do**
    $\theta \leftarrow 2\pi \cdot j / segments$
    $offset \leftarrow r \cdot (\cos\theta \cdot N + \sin\theta \cdot B)$
    Add $(center + offset)$ to circle
  end for
  Append circle to list of cross-sections
end for
for $i = 0$ **to** length(path) -2 **do**
  $bot \leftarrow$ circle[i], $top \leftarrow$ circle[$i+1$]

$p_0 \leftarrow$ path[i].position, $p_1 \leftarrow$ path[$i+1$].position
for $j = 0$ **to** segments -1 **do**
  $n \leftarrow (j+1) \mod segments$
  Construct quad: $v_0 = top[j], v_1 = top[n], v_2 = bot[n], v_3 = bot[j]$
  Add triangles (v_0, v_1, v_2) and (v_0, v_2, v_3) to mesh.positions
  Compute normals as directions from p_1 or p_0 to each vertex
  $u_0 \leftarrow$ TexRepeat $\cdot\, j/segments$
  $u_1 \leftarrow (n == 0)?$TexRepeat : TexRepeat $\cdot\, n/segments$
  Add texture coordinates (UVs) for the 6 vertices of the two triangles
end for
end for
return mesh

第四章　實驗結果

4.1　實驗環境

為支援本研究中點雲資料的處理流程與重建結果的可視化，系統以 C++ 作為主要開發語言，並採用 OpenGL 作為圖形渲染 API，建構整體開發與執行環境。其硬體平台與開發工具配置如下所列。

硬體與系統平台

- 中央處理器: AMD Ryzen 5 3600 6-Core
- 顯示卡: NVIDIA GeForce RTX 2060 Super (Driver 576.40)
- 記憶體: DDR4-3200 16 GB
- 作業系統: Windows 11 64bit

程式語言與開發環境

- IDE : Visual Studio 2022
- C++ 20
- OpenGL/GLSL 4.6

函式庫

- SDL3 (https://www.libsdl.org/)
  用於顯示圖形化視窗，處理鍵盤滑鼠輸入。

- glad (https://glad.dav1d.de/)
  為主程式載入 Modern OpenGL 函數位址。
- GLM (https://github.com/g-truc/glm)
  提供電腦圖學常用的數學工具，如向量計算、變換矩陣等。
- ImGui (https://github.com/ocornut/imgui)
  輕量且快速建立使用者互動控制項，適用各種圖形 API。
- Boost Graph Library (https://www.boost.org/doc/libs/release/libs/graph/)
  用於快速處理圖結構與演算法操作。

4.2 重建結果展示

展示重建流程中間結果與其最終網格模型，前四張圖（由左至右）分別為：

1. 參考相片：對目標樹木進行環繞拍攝之相片，作為 SFM 重建之輸入。
2. SFM 還原點雲：利用 Structure-from-Motion 技術還原之稀疏點雲。
3. 初始骨架：由點雲資料中提取之初步樹木骨架結構。
4. 經修剪後之骨架：經過節點分析與程序修剪後之骨架，去除多餘分枝並保留主要結構。

右側黑框內分別呈現從 +X、+Z、−X、−Z 軸方向觀察的視角，展示重建樹木模型在不同角度下的外觀。

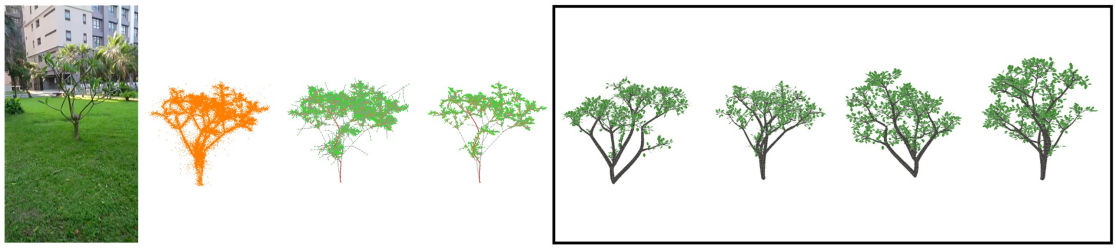

Figure 4.1: 重建案例一

Figure 4.2: 重建案例二

Figure 4.3: 重建案例三

Figure 4.4: 重建案例四

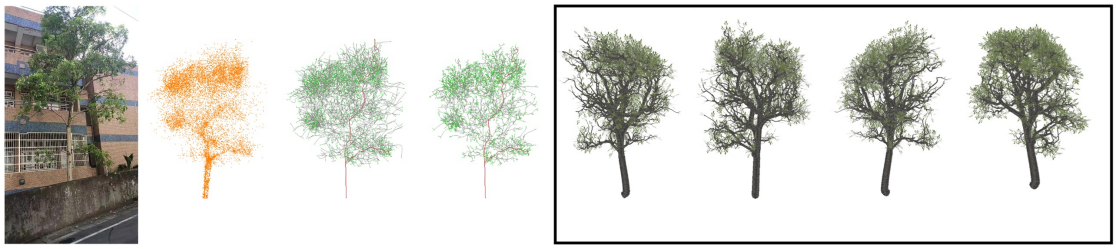

Figure 4.5: 重建案例五

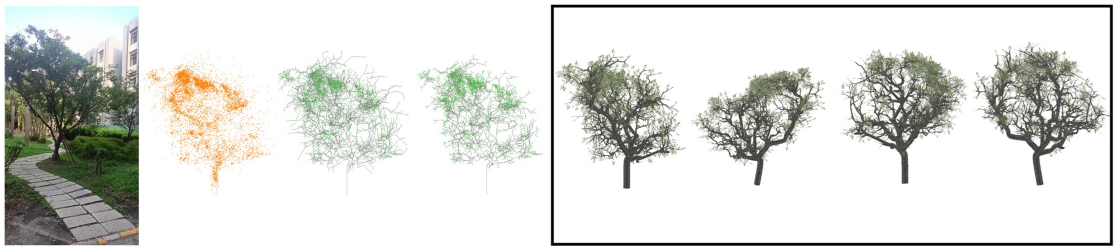

Figure 4.6: 重建案例六

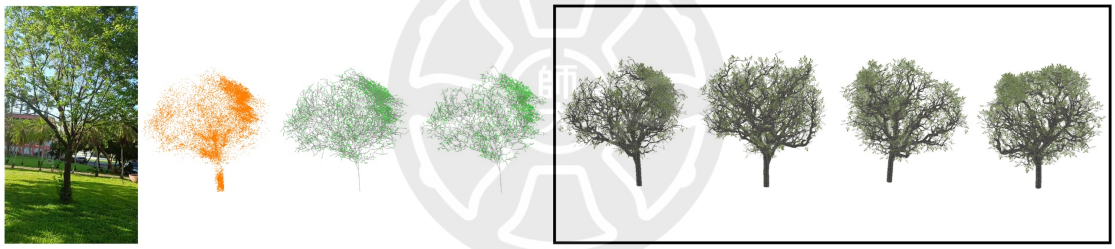

Figure 4.7: 重建案例七

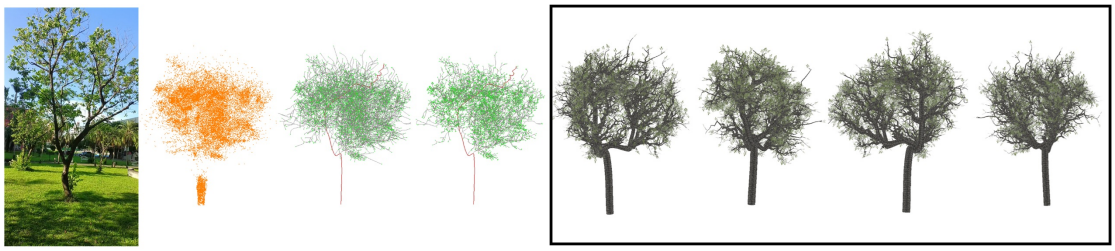

Figure 4.8: 重建案例八

4.3 重建結果討論

針對各項重建結果與實際照片進行對照分析與討論:

4.3.1 案例一

Figure 4.9: 各階段結果與輸入資訊 (案例一)

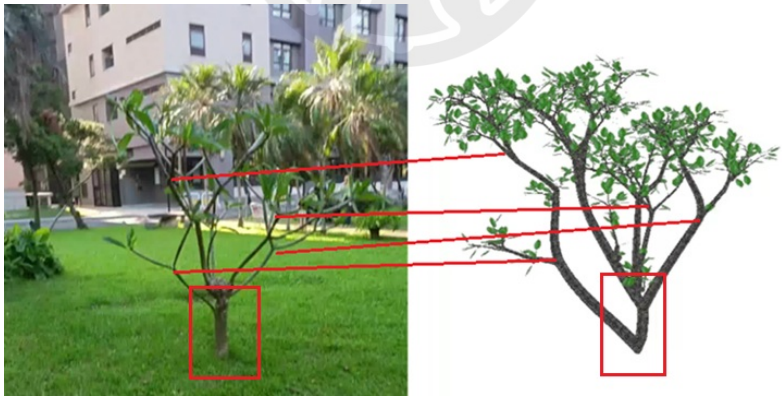

Figure 4.10: 相片與重建結果對照 (案例一)

案例一為缺乏明顯主幹的矮小樹木，從上圖紅框處可觀察到，系統將樹木根部誤判為具有兩個分支的結構。當主幹較短或不明顯時，骨架提取過程容易產生誤判，導致根部結構不正確。儘管原始樹木的主幹結構不明顯，仍成功捕捉到主要分枝結構，顯示在骨架還原上具有一定準確性。

4.3.2 案例三

Figure 4.11: 各階段結果與輸入資訊 (案例三)

Figure 4.12: 相片與重建結果對照 (案例三)

案例三樹木具有筆直且明確的主幹，以及成層水平分布的側枝，整體結構清晰有序且層次分明。由於骨架特徵明顯，分支規律性高，系統在進行骨架提取時能更容易辨識各層結構，有效還原出該樹木的三維空間資訊。

4.3.3 案例四

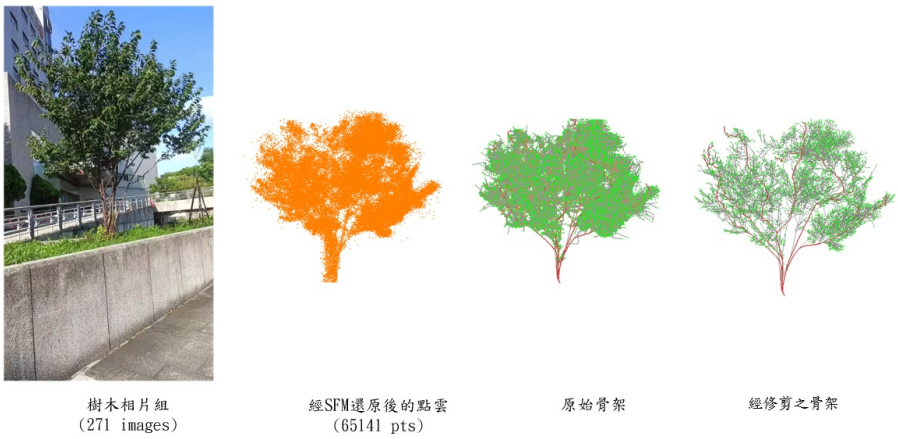

Figure 4.13: 各階段結果與輸入資訊 (案例四)

Figure 4.14: 相片與重建結果對照 (案例四)

案例四的重建結果在樹冠部分表現良好，成功還原了上層分枝與樹葉的分布，整體外形與實際照片一致。然而該樹木根部由多個細小主幹構成，結構較為複雜。對主幹半徑產生較大誤判，導致根部結構失真，進而影響後續對分支粗細的推估。

4.3.4 案例八

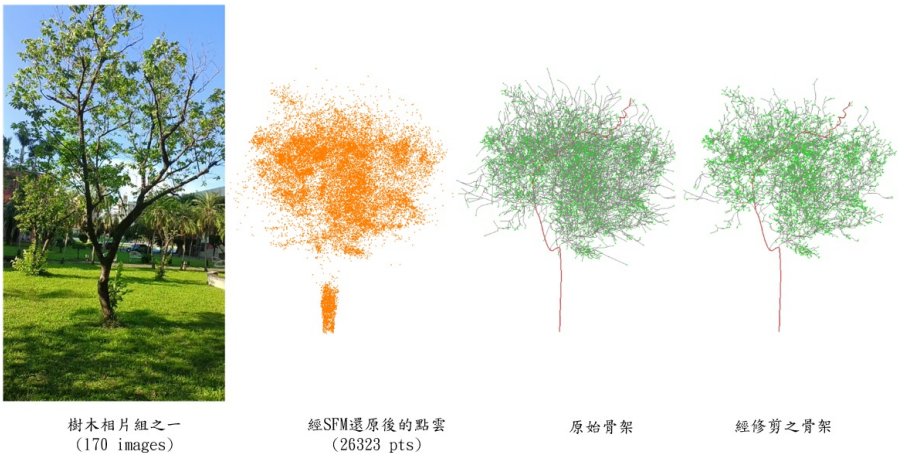

Figure 4.15: 各階段結果與輸入資訊 (案例八)

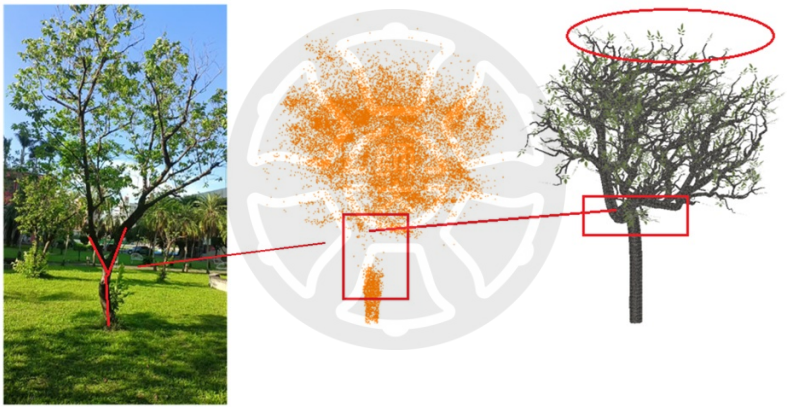

Figure 4.16: 相片與重建結果對照 (案例八)

案例八中的樹木在根部具有明顯的 Y 字型分支，但對應的點雲輸入在該區域（紅框處）出現大量缺失，導致主幹與分支連接錯誤。此外，樹冠部分的點雲也無法有效反映原樹木的真實結構，造成重建結果中產生過度複雜的枝條分布。當點雲資料在主幹等關鍵區域出現大範圍或缺失時，容易導致骨架重建產生明顯誤差。若樹冠點雲本身缺乏結構性，也容易被系統過度擬合，生成不符合實際的複雜結構。

第五章　結果與未來展望

5.1　結論

本研究結合運動回復結構 SFM 技術、AdTree 演算法，以及自行設計的程序，提出一套能從多視角影像中自動重建樹木結構的建模流程。針對 SFM 所產生結構不完備的稀疏點雲進行預處理、改善 AdTree 於不精確點雲中提取骨架時所產生的不自然細長分枝，透過程序進行修剪，最終生成符合相片自然風格的樹木結構，並輸出可供編輯的模型檔案。此外，本研究亦實作了一套具備圖形化使用者介面的互動式編輯器，能直觀且視覺化地呈現各階段的處理結果（點雲、骨架與網格），提供節點選取與增刪、子樹擷取與子樹連接等互動功能，有效協助使用者進行樹木模型的編輯，大幅提升整體系統的實用性與靈活性。

從實驗結果可觀察到，部分案例的重建品質與原始影像存在顯著差異。本研究的重建方法在實作上有以下限制與挑戰：

1. 重建效果高度依賴初始點雲的完整性
2. 當樹木本身缺乏明確主幹時，容易造成提取之骨架與實際結構不符
3. 未能完全去除的離群雜訊點雲，會使重建結果出現大量細碎且零散的樹葉分布
4. 樹冠過於茂密時，會遮蔽樹幹與分支，也干擾骨架的擷取，將樹葉點雲誤判為骨架節點產生多餘的分支

5. 為產生足夠量的初始點雲以構成較為完整的樹木基本結構，目前仍須進行環繞拍攝，並取得至少 50 張影像

5.2 未來展望

針對目前重建結果的完整度與準確性，未來可從以下幾個方向進行改進：點雲中的雜訊與缺失是造成重建誤差的重大因素。對於小範圍的缺失，可採用根據鄰近點的插值方式進行補全；而對於大區域的點雲缺失，則可考慮使用 PCN（Point Completion Network）等深度學習模型進行結構推測與修補。

現階段樹木分支與樹葉的外觀皆採用預先設定的材質貼圖進行統一繪製，樹葉樣式亦未區分樹種。未來若能從原始相片中分析其樹幹紋理特徵，進而製作對應的紋理貼圖，將可提高重建模型的視覺擬真度，有助於針對不同樹種進行更精細的建模。

若能在點雲中導入 Point Segmentation，針對樹幹、樹枝與樹葉進行語意分類，期望能有效區分出樹冠與樹枝區域，進一步提升骨架擷取的準確性。這種方式可避免將大量樹冠上的葉片點雲誤判為分支節點，減少結構上的錯誤與多餘的分支生成。此外分離出的樹冠點雲亦可作為樹葉分布的依據，用於控制葉片擺放與數量配置，提升模型在結構與視覺上的整體合理性。

參考文獻

[1] Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. MeshLab: an Open-Source Mesh Processing Tool. In Eurographics Italian Chapter Conference (2008), V. Scarano, R. D. Chiara, and U. Erra, Eds., The Eurographics Association.

[2] Corsini, M., Cignoni, P., and Scopigno, R. Efficient and flexible sampling with blue noise properties of triangular meshes. IEEE Transaction on Visualization and Computer Graphics 18, 6 (2012), 914–924. http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.34.

[3] Dijkstra, E. W. A note on two problems in connexion with graphs. Numerische Mathematik 1, 1 (1959), 269–271.

[4] Douglas, D. H., and Peucker, T. K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer 10, 2 (1973), 112–122.

[5] Du, S., Lindenbergh, R., Ledoux, H., Stoter, J., and Nan, L. Adtree: Accurate, detailed, and automatic modelling of laser-scanned trees. Remote Sensing 11, 18 (2019), 2074.

[6] Elshakhs, Y. S., Deliparaschos, K. M., Charalambous, T., Oliva, G., and Zolotas, A. A comprehensive survey on delaunay triangulation: Applications, algorithms, and implementations over cpus, gpus, and fpgas. IEEE Access 12 (2024), 12562–12585.

[7] Frahm, J.-M., and Pollefeys, M. Ransac for (quasi-)degenerate data (qdegsac). CVPR '06, IEEE Computer Society, p. 453–460.

[8] Hackenberg, J., Spiecker, H., Calders, K., Disney, M., and Raumonen, P. Simpletree —an efficient open source tool to build tree models from tls clouds. Forests 6, 11 (2015), 4245–4294.

[9] Hanson, A. J., and Ma, H. Parallel transport approach to curve framing. Indiana University, Techreports-TR425 11 (1995), 3–7.

[10] Lowe, D. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision 60 (11 2004), 91–.

[11] Mapillary Team. OpenSfM: Open source structure from motion library. https://github.com/mapillary/OpenSfM, 2017. Accessed: 2025-07-30.

[12] Prusinkiewicz, P., and Lindenmayer, A. The Algorithmic Beauty of Plants. Springer-Verlag, New York, NY, USA, 1990.

[13] Schönberger, J. L., and Frahm, J.-M. Structure-from-motion revisited. In Conference on Computer Vision and Pattern Recognition (CVPR) (2016).

[14] Schönberger, J. L., Zheng, E., Pollefeys, M., and Frahm, J.-M. Pixelwise view selection for unstructured multi-view stereo. In European Conference on Computer Vision (ECCV) (2016).

[15] Snavely, N., Seitz, S. M., and Szeliski, R. Photo tourism: exploring photo collections in 3d. ACM Trans. Graph. 25, 3 (July 2006), 835–846.

[16] Wu, C. VisualSFM: A visual structure from motion system. In Proceedings of the 2011 IEEE International Conference on 3D Vision (3DV) (2011). Available at: http://ccwu.me/vsfm/.

附錄 A — 程式介面與操作說明

A.1 程式介面與操作說明

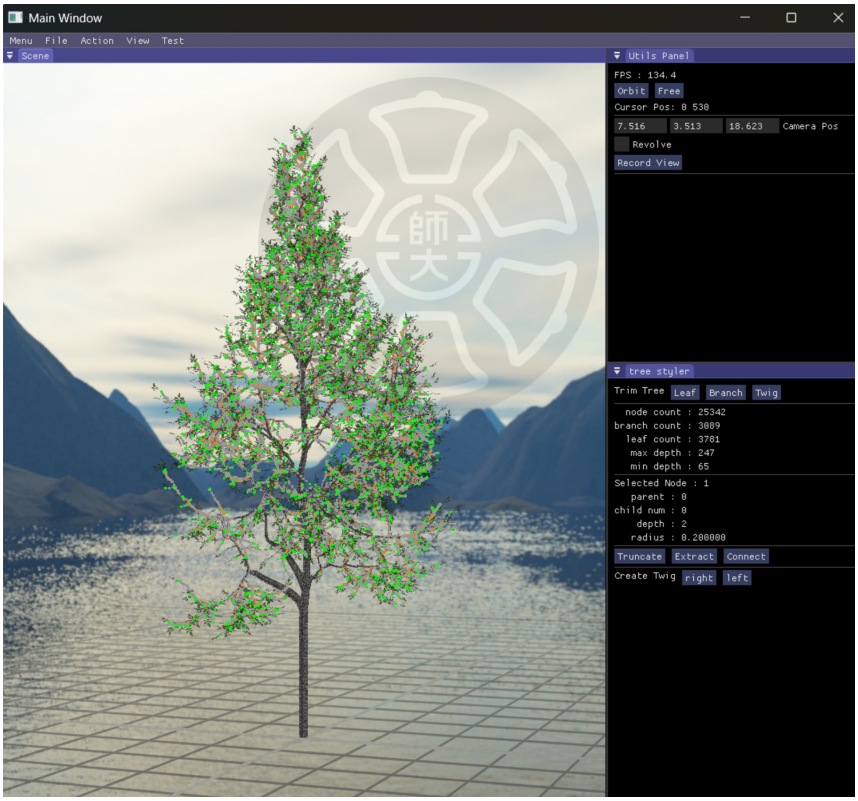

Figure A.1: 主程式介面

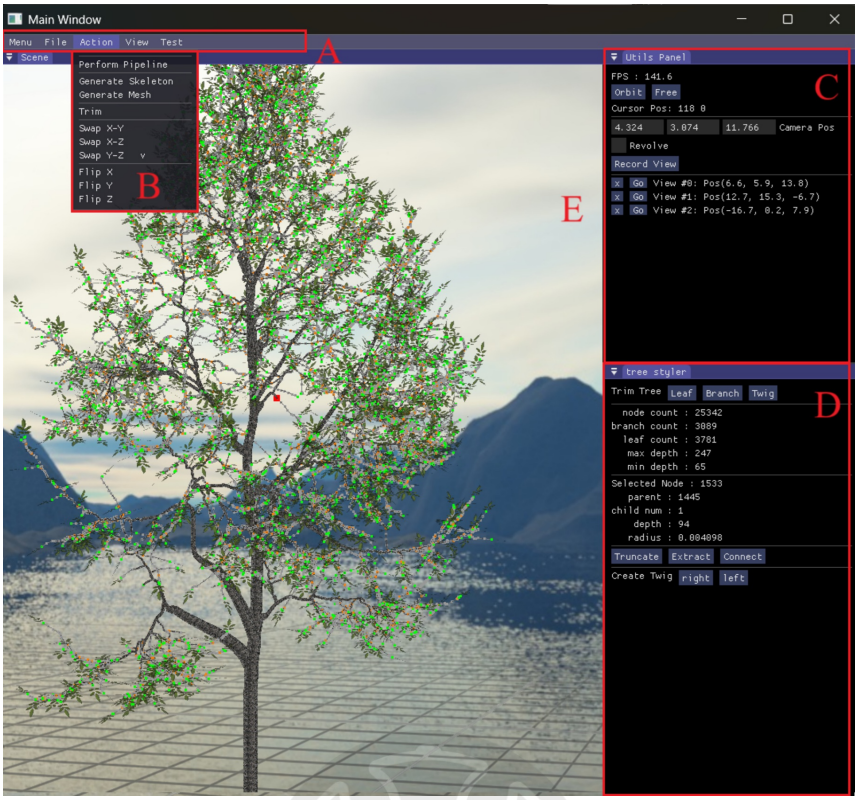

Figure A.2: 介面說明

本系統提供直覺化的圖形介面與操作功能，協助使用者完成點雲載入、樹木重建與結構編輯等任務，介面組成說明如下：

- 主選單 (A)：提供基本檔案與功能操作。
  - File：讀入或儲存點雲檔案（支援.ply 與.xyz 格式）
  - Action：包含各項處理操作，詳見下方「操作選單」說明。
  - View：控制各視覺物件的顯示與隱藏，亦可透過快捷鍵操作（見表 A.1）。
- 操作選單 (B)：供使用者對點雲資料進行多項操作，如骨架提取、座標翻轉、點雲轉置等。
- 相機面板 (C)：可儲存當前相機位置與視角參數，並切換相機運動模式（自由/軌道模式）、啟用環繞檢視等功能，方便使用者從多角度觀察樹木模型。

- 樹木編輯面板 (D)：提供節點級的互動控制，例如修剪、連接、子樹擷取與屬性修改等功能。

- 場景檢視 (E)：視窗中即時渲染點雲、骨架與網格模型，讓使用者即時查看編輯與分析結果。

本系統的相機運動模式分為「自由模式」與「軌道模式」兩種。自由模式模擬第一人稱視角，可使用 WASD 鍵與滑鼠移動視角；軌道模式則以目標為中心進行環繞觀察，更適合用於樹木結構編輯。兩者可透過 Tab 鍵進行切換，靈活適應不同操作需求。

Table A.1: 按鍵操作說明

按鍵	**功能**
W/A/S/D	相機 XZ 平面移動 (自由模式)
Space/Left-Shift	相機上升/下降 (自由模式)
滑鼠移動	相機視角轉動 (自由模式)
滑鼠右鍵拖動	相機平移 (軌道模式)
滑鼠中鍵拖動	相機視角轉動 (軌道模式)
滑鼠左鍵點選	選擇樹木節點
Tab	切換相機運動模式
PrtScr	場景截圖
R	相機繞模型旋轉
P	顯示/關閉樹木點雲
S	顯示/關閉樹木骨架
T	顯示/關閉樹木網格
E	開啟/關閉風場效果
X	顯示/關閉座標軸
G	顯示/關閉座標網格

A.2 程式使用指引

操作流程建議如下：

1. 使用者可先對目標樹木進行環繞拍攝，並將拍攝影像匯入 COLMAP，以還原對應的三維點雲。

2. 開啟程式後，透過主選單「File」讀入生成的點雲（.ply 或.xyz 檔案），系統會自動將資料對齊並置中顯示於場景視窗。

3. 若輸入點雲出現上下顛倒或座標軸方向錯誤，可經由「Action → Adjust Point Cloud」進行修正。

4. 調整完畢後，點選「Action → Run Pipeline」即可執行完整的樹木重建流程，包含骨架擷取與結構初始化，結果將即時呈現於視窗中央。

5. 接著可於樹木節點編輯面板執行各項編輯指令（如修剪、擷取、連接、子樹等），編輯結果會即時渲染至場景中，使用者可視需求進行多次互動調整。