國 立 成 功 大 學
資 訊 工 程 研 究 所
碩 士 論 文

具深度學習的三維模型線條藝術生成

Line Art Generation from Model in Three-dimensional Space with Deep Learning

研究生：藍晧

指導老師：李同益博士

中 華 民 國 110 年 7 月

摘要

在本篇論文中，介紹一個利用深度學習將三維模型進行對稱分割(symmetric segmentation)，並且產生線條藝術(3D wire art)的方法。本研究的起源是在於藝術家製作線條藝術時，會根據實體的模型外圍來進行線條生成與組合，最後進而與原本模型實體體積(3D volume)相當。本篇整體概要是在於如何用同樣的方式基於深度學習的模型分割(segmentation)對 3D 模型來產生線條藝術。首先，我們利用深度學習的方法來對 3D 模型的網格(mesh)先進行分割，我們提的分割分法是先對模型本身來進行分類，之後才能達到對稱線條的結果。接著萃取網格上的邊來進行合併(merge)，並產生合併後的邊線段，這些線段大致上就可以來表示模型的外圍輪廓。由別於傳統模型分割，本研究利用 deep learning 進行分割，其優點是可以較準確得對模型執行部位分割，像是對每個部位去做分類，如此優勢可以有利於產生對稱性且具維持原模型 3D 體積之特性。產生線段後，為了使連接後的連續線條達到最少的數量，我們制訂了多旅行商問題(multiple traveling salesman problem, mTSP)來達成目的進而產生線條藝術。我們發現線條生成會產生不連續線段的斷成，因此也制定了根據使用者選取不同線段之間的節點(node)來進行合併，此方式可以讓最後的成品看起來較自然且較少條 wire 線。我們希望最後的線條能夠盡量達到平滑的效果，對此我們利用一個最佳化的方式使線段之間角度差別較大的地方進行平滑化，使得最後生成的線條不會看起來崎嶇不平，而較有連續感。另外、我們也提供了介面可以依照個人喜好來進行線段連接，產生符合使用者的預期結果。

關鍵字：深度學習、線條藝術、最佳化、網格

Abstract

In this paper, we introduce a method of using deep learning to symmetrically segment a three-dimensional model and generate line art. The origin of this work is that when the artist makes line art, it will generate and combine lines based on the periphery of the entity's model, and finally it will be equivalent to the original model entity. The overall summary of this article is how to use the same way to generate line art based on the model segmentation of deep learning on the 3D model. First, we use the deep learning method to segment the mesh of the model, that is, to classify the model itself before reaching the result of symmetrical lines. Then extract the edges on the mesh to merge, and generate merged edge line segments, which can roughly represent the outer contour of the model. After generating line segments, in order to minimize the number of connected continuous lines, we formulated the multiple traveling salesman problem (mTSP) to achieve the goal and produce line art. We found that line generation will produce discontinuous line segments. Therefore, we have also formulated a combination of nodes between different line segments selected by the user. This method can make the results look more natural. We hope that the results of line art can be as smooth as possible. For this, we use an optimized method to smooth the parts where the angle difference between the line segments is large, so that the final generated lines will not look rugged, but with smooth lines. In addition, we also provide an interface to connect line segments according to personal preferences, and produce results that meet the user's expectations.

Keywords: Deep learning, line arts, optimization, mesh

致謝

進本實驗室以後，經過各種學校課程磨練以及眾多人的幫助下才達到預期的論文成果。

首先，感謝父母讓我有這個機會到學校來做自己想要的研究，也在疫情這段期間給予我最大的生活協助以及心靈上的支柱，並且也適時地陪伴我度過這兩年。

接著，感謝指導老師李同益教授在此期間不斷地給我許多的建議和鼓勵。平時在學校也積極得跟我討論要如何使自己的作品能夠更好，也更希望自己的作品能夠跟其他人的結果相較之下更上一層樓。

感謝侯志穎學長，在投論文的期間教導我並且給我適當建議，之後讓我在線條藝術作品上有更深層的發想。

最後，感謝實驗室的各位同學們在一起研究的期間能給予陪伴以及精神上的支持，在此期間彼此有一起努力奮鬥以及歡樂時光。

目錄

圖目錄

Chapter 1

Introduction

線條藝術是由藝術家經過對實體模型及物品進行觀察，接著利用鋼線、銅線經過徒手或工具來曲折線段組合形成一個藝術作品。其中線條藝術又分為利用單條線來做整個模型的簡易外圍輪廓，如 Figure1.1，以及利用多條線繞行的方式銜接其他線段，如 Figure 1.2、Figure 1.3。

Figure 1.1: 藝術家 Theodore Huckins 用單一線段創作的線條藝術作品

Figure 1.2: 藝術家 Bud Bullivant 用多條線段創作的線條藝術作品

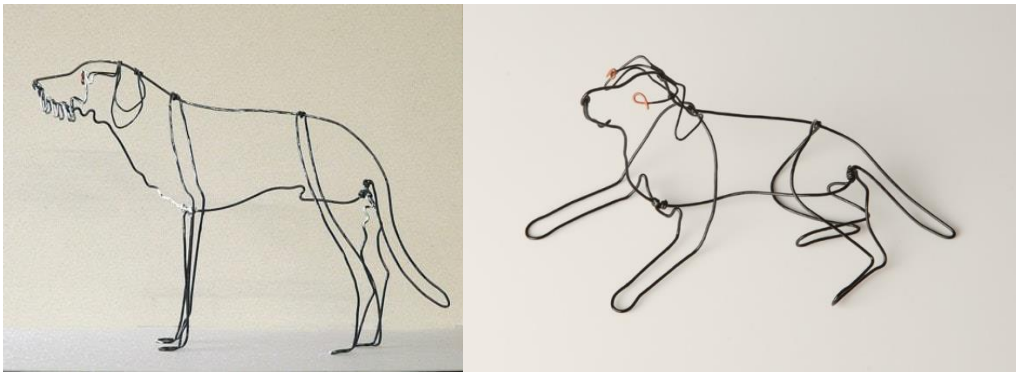


Figure 1.3: 藝術家 Bridget Baker 用多條線段創作的線條藝術作品

由 Figure 1 藝術家創作的作品，我們可以發現藝術家利用多種方式以及不同

的創意來呈現同樣的作品。其中又用不同多寡的線條和繞行方法使得最後的成品千變萬色。在 Figure 1.1，可以看到 Theodore Huckins 是利用單一線段來形成一個簡易的動物外圍輪廓，雖然某些視角會看起來並沒有動物的形狀，不過他是用最少的線條來組成這份作品。而在 Figure 1.2，可以看到 Bud Bullivant 利用多條線段扭曲繞行的方式組成的作品，但所有線段並非是連接在一起的。最後在 Figure 1.3，Bridget Baker 也是用多線段繞行的方法使得每個線段都是連接著，並且讓成品在某些角度有體積感以及不同的觀看姿勢。

在本篇論文中以這些作品為參考的基礎點，我們嘗試利用原本具有的 3D 模型來產生出線條藝術。當人在特定角度看 3D 物體時，可以容易將其感知到大致上的邊界，進而計算出整體輪廓，藝術家也是利用此方法來產生出線條藝術。不過要產生出每個視角都能夠辨別出物體本身的形狀是有難度的(如 Figure 1.1 右圖)，而且如果產生出來的線條在某些地方少了一些曲折或線段，也會因此使得某些視角的辨別度更加困難。我們的方法會盡量克服上述難題並希望有別於藝術家線條藝術，能盡量維持與原本模型實體體積以達 view-independent 且最少條線條之藝術設計。

首先，我們利用網格的資訊計算在深度學習的架構上來取得初步的模型分割且進行分類，使得產生出來的線段具有對稱性(symmetry)。接著制定出一個提取出線段的演算法，使線條結果讓人眼足以在不同的視角(view independent)下能夠辨認出物體本身的形狀。我們參考了 Liu[3]等人的多旅行商問題(multiple traveling salesman problem, mTSP)方法，並且加一些限制方程式來進行改進，用此方式來決定所有線段連接的組成方法(wire composition)。接著，我們提供使用者介面讓使用者可以選取看起來不自然的不連續線段，選取後能夠自動使兩條分開的線段連接起來，讓最後整體的線條看起來是連續且對人眼辨別較自然的結果。最後，為了使整體的線段看起來更像線條藝術，我們制定一個最佳化問題使這些線段可以平滑化，也更像一個線條藝術作品。用這樣的方法來模擬藝術家用鋼、銅線做成的作品，我們主要的貢獻包括：

1. 提出一個能將三維模型轉換成線條藝術的方法。
2. 利用深度學習的分割方法來結合在線條藝術上。
3. 制定一個使用者介面能夠使一些不連續線段有連續性。
4. 最後產生出的線條藝術能夠盡量具有對稱性及體積。

Chapter 2

Related Work

由二維線條產生出多視角三維線條藝術。Hsia[1]等人提出一個利用 visual hull 以及 voxelization 的概念，將三個二維線條影像從三個不同視角投影來獲得三維交點，並取得初步的 discrete visual hull。接著再利用影像正投影的方式來補齊三個不同影像在不同視角所需要的線條。最後將產生出的三維線條來形成線條藝術。在形成線條的過程有可能會產生線條不連續性，因此也需要使用者手動做修復的動作使線條連續且簡化，如 Figure 2.1。藉由不連續性以及線條藝術形成的啟發，讓本研究列入一些更多考量因素。Figure 2.2 是他們產生出來結果。他們的方法可以將較複雜的二維線條產生出三維線條藝術，但是反而在於較簡單結構的二維線條作為輸入產生出的效果較差，如 Figure 2.3。在於程式時間計算上，則是要看輸入影像的解析度。如果三個二維影像解析度為512^3就需要 510 分鐘；解析度為256^3可以減少七倍以上時間。

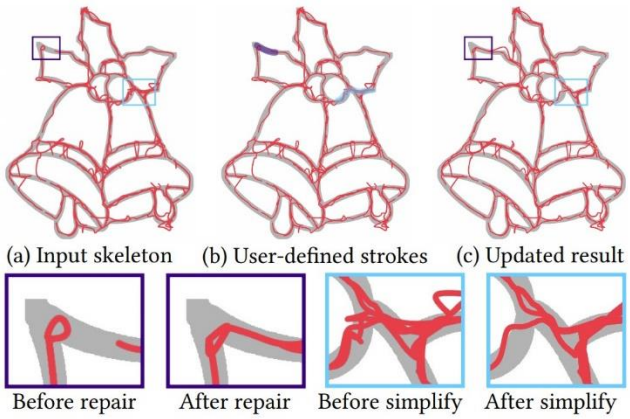


Figure 2.1: 修補線段的過程。

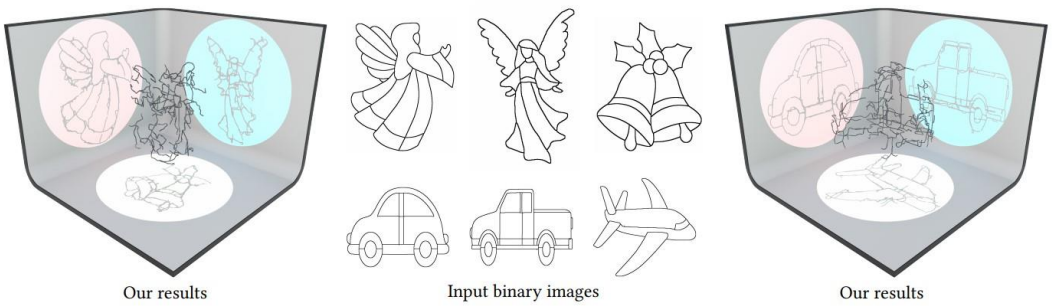


Figure 2.2: Hsia 等人的結果。

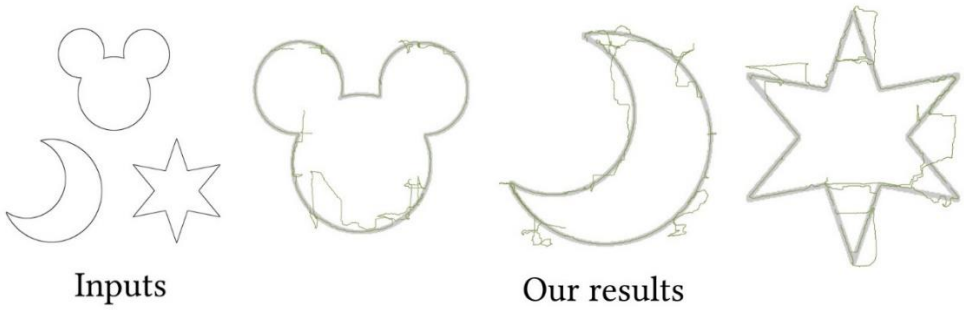


Figure 2.3: 較簡單的線條產生出的輪廓(contour)會產生出較差的結果，我們可以發現很多細線都在輪廓外。

基於神經網路的 3D 形狀分割。Wang[2]等人利用 Fully convolutional networks 的方法去做改良來對三維模型進行分割。他們的方法是將三維資料的網格(mesh)轉成 graph structure 來方便進行訓練，如 Figure 2.4。將這些 graph 資料經由改良過的神經網路來進行一系列的卷積(convolution)和池化(pooling)。最後他們的研究是可以將每個模型來進行對稱分割，如 Figure 2.5。他們利用深度學習的方法也讓我們的研究列入考量。此篇的應用是在於對 3D 模型進行部位分割(partial segmentation)及辨識(classification)。有別於一般傳統分割，可能同樣種類的模型沒辦法切割出一樣數量種類的部位分割，因此他們將模型轉換成圖表(graph)資料來去做訓練，也可以大幅縮短訓練時間。

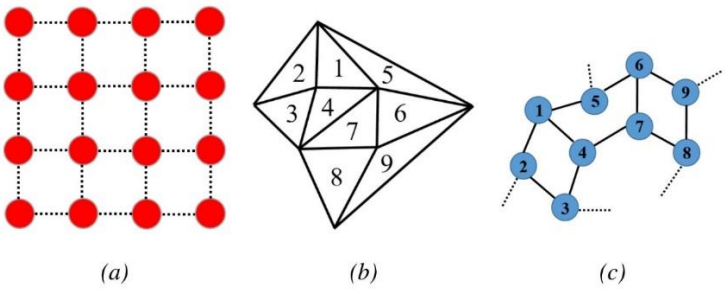


Figure 2.4: 由(a)來說明二維影像的資料意義，(b)是三圍模型的網格資訊，(c)是經由(b)轉換過來的 graph structure。

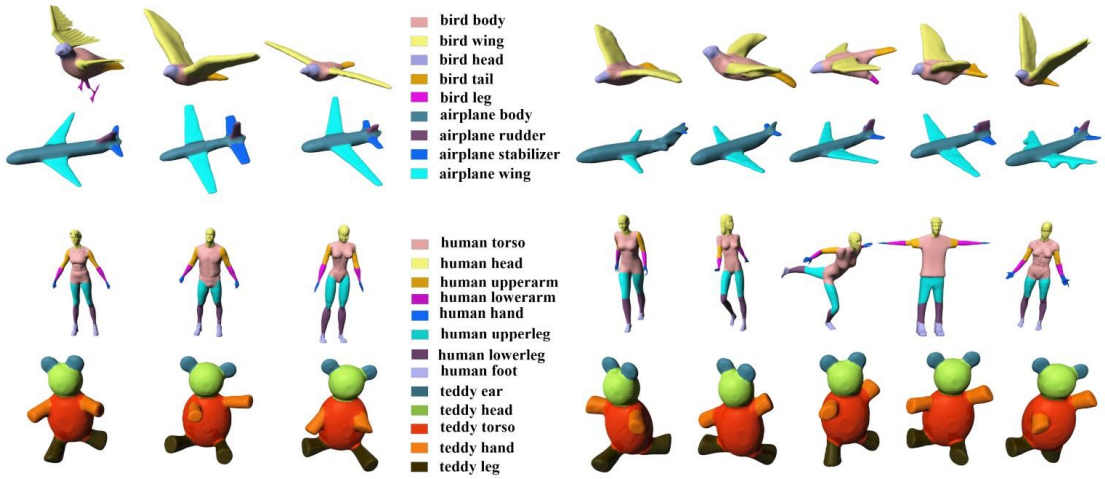
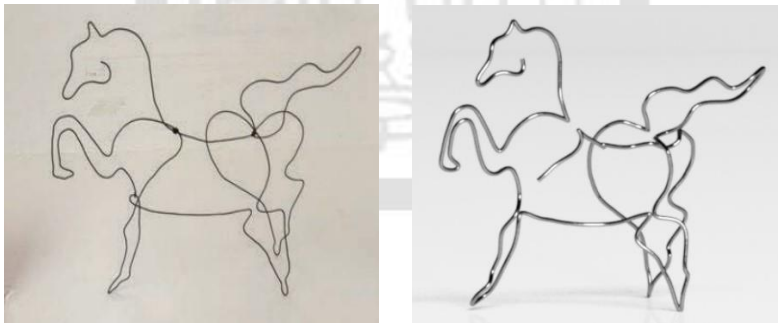


Figure 2.5: Wang 等人的結果。

基於影像的線條重建。Liu[3]等人提出一種從不同角度拍攝的二維線條藝術作品來產生三維的線段，並且利用多旅行商問題(mTSP)來連接不同的線段產生連續的線條，如 Figure 2.6。我們將多旅行商問題來進行改進，並且加一些限制條件來使最後的結果比較符合我們的預期。他們的方法會使某些路徑線條的節點會有重疊現象，使得最後連接的線條無法為最少組成線條，因此我們提出方法以及一些限制來解決這問題。

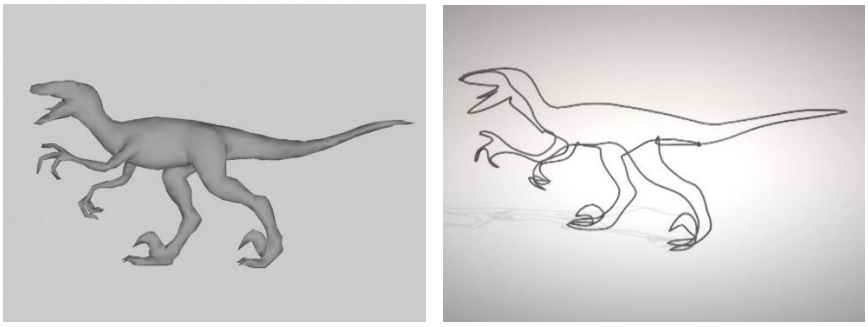


(a)輸入的二維影像　　(b)重建後的三維線條

Figure 2.6: Liu 等人的結果。

三維模型的線條藝術生成。Hou[4]等人提出能將三維模型為輸入資訊，而能將其自動產生出三維線條藝術。他們利用三維模型的網格資訊，並且利用合併線段的貪婪演算法來形成最後會呈現的三維線段。在這其中他們也將多旅行商問題進行改寫，進而使最後連接在一起的連續線段是最少的，如 Figure 2.7。在這其中，他們也設計一種方法能使線條平滑化，且能夠讓使用者自己制定出預想保留的線條。他們的方法雖然可以呈現最少線條來表示一個模型，但是在某些其他模型的外型

下較無法有效呈現，在本篇結果與討論章節會有比較結果。我們利用他的方法並且合併 deep learning 對稱分割來使最後呈現的線條能夠較有體積感及對稱感，使每個角度來看此線條藝術是較可以輕易辨識出來。



(a)輸入模型　　(b)輸出線條藝術

Figure 2.7: Hou 等人的結果。

由三維線條生成三維模型。Nealen[5]等人提出一種能將網格上生成線條的方法來產生出預期的三維模型。在此，他們提供一個使用者介面，能夠將使用者在網格上畫出的線條即時轉化成三維的控制線，並且是維持在網格上的曲線，如 Figure 2.8。根據這些曲線來定義出三維模型，如 Figure 2.9。

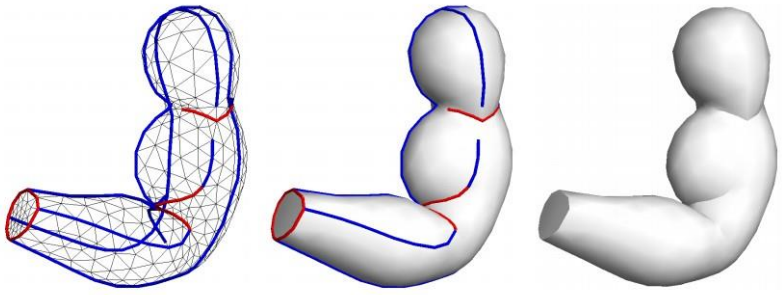


Figure 2.8: 在網格上畫出控制曲線。

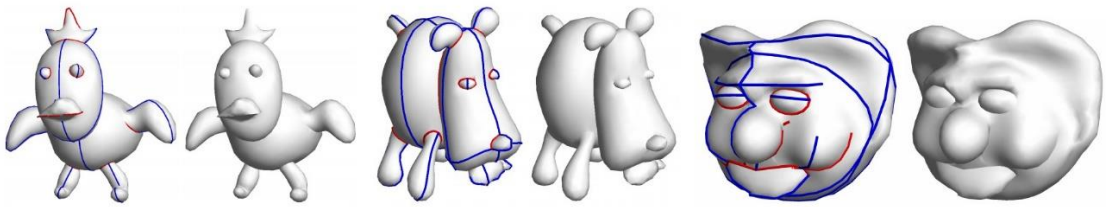


Figure 2.9: Nealean 等人的結果。

Chapter 3

System Overview

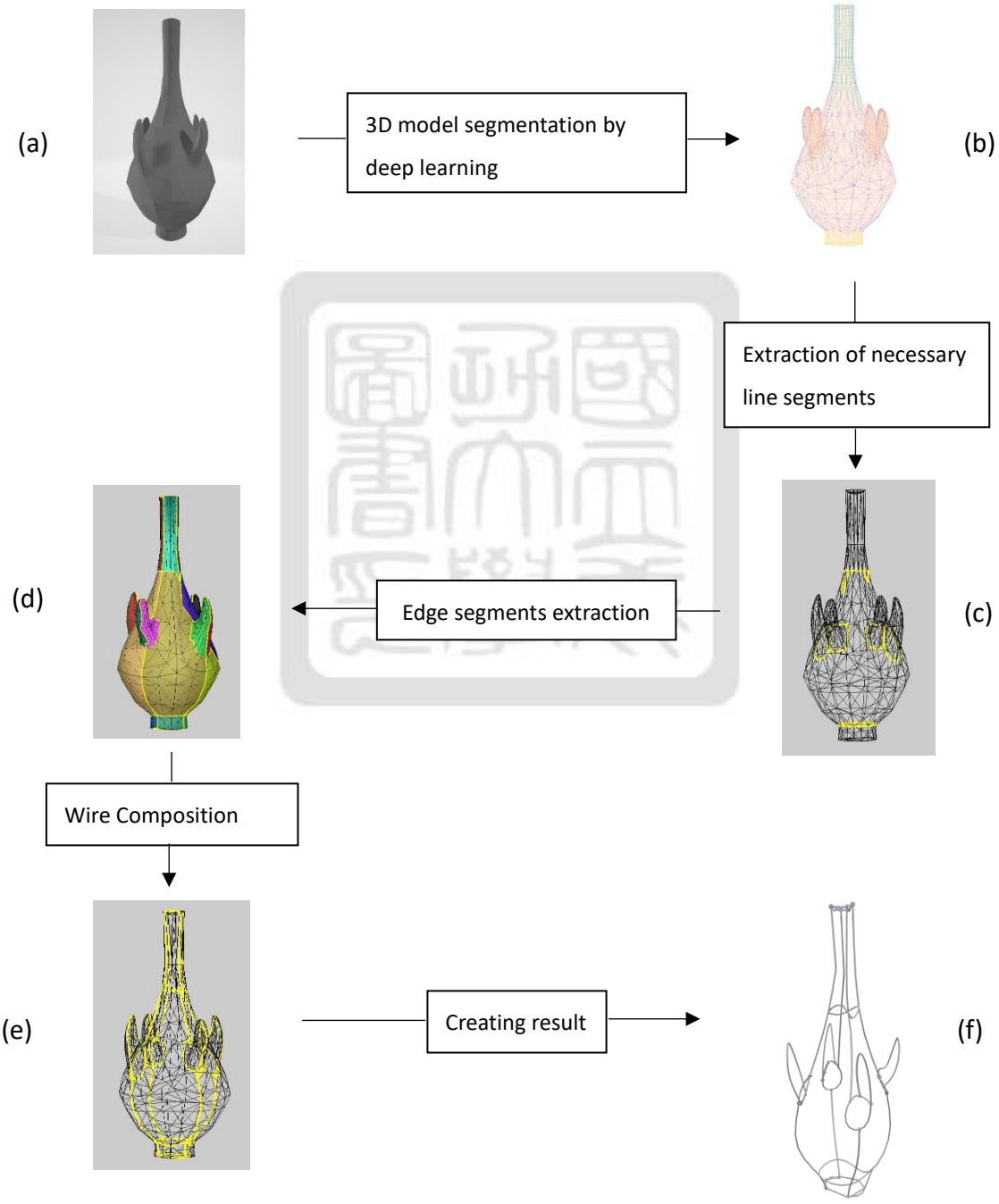


Figure 3.1: 系統步驟。

三維模型分割(3D model segmentation)。我們的系統是輸入三維模型的各式資訊，包含點、線、面等。利用 Hanocka[6]等人 MeshCNN 深度學習的一系列卷積(convolution)和池化(pooling)，得到最終模型簡化的結果，如 Figure 3.2。接著再經由監督式學習(supervised learning)的方式來去預測每個模型的每種區塊去做歸類，並且做分割處理。我們將分割以後不同顏色區塊之間的邊預先保留下來，這些邊將會是最後結果會呈現的對稱線條，如 Figure 3.1(c)。

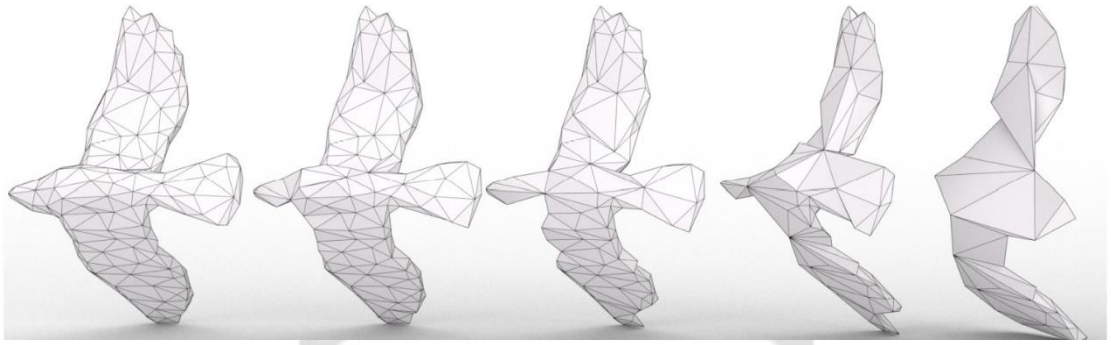


Figure 3.2: 模型的邊經由 MeshCNN 簡化後的結果。

邊線段之萃取(Edge segments extraction)。將需要保留的對稱切割邊提取出來後，接下來就要從網格上提取出模型裡的重要邊。這裡是參考 Garland[7]等人的貪婪演算法，他們是利用此演算法來做三角形面的簡化。我們利用此演算法來模型網格(mesh)上較平滑的邊移除掉，而剩下來的邊都是相對較多皺褶的部分。我們認為這樣的方法可以使保留下的線段更可以代表模型本身的重要邊，也比較能使辨別度提高。利用合併的方式，取得模型再一次的分割結果，並且產生線段初步的成果，如 Figure 3.1(d)。

連續線的組成(Wire composition)。前個步驟使所有的連續線段生成後，我們為了使產生的線段能夠是由最少數量的線條組成，我們參考了 Hou[4]等人提出的方法。他們是改良來自 Liu 等人多旅行商問題來連接線段，用端點的平滑程度以及切線向量來計算線段之間的代價。在這其中，為了使最後的線條能夠符合我們預期的連續線段結果，他們多設了幾個條件(constraints)，並且利用 CPLEX Optimizer 來解決此最佳化問題。

產生線段之連續性(Wire continuity)。當我們生成模型的線條後，我們發現有某些部分會產生斷層。因此我們利用手動的方式將斷層線段最近的兩個環面(torus)選取，將選取後的環面之兩線段合併，並且使環面移除。這樣的方法可以有效減少線條的數量，也可以使最後的線條藝術看起來較自然。

線條平滑化(Wire smoothing)。當整體連續線條產生後，為了使結果更真實也更像實際藝術家會呈現的作品，我們將較粗略且皺褶的線段加以平滑化。這邊我們參考 Hou[4]等人的方法，制定出一個最佳化問題來產生出平滑的線條。此方法我們會在第四章作更詳細的敘述。

Chapter 4

Methods

4.1 三維模型分割(3D model segmentation)

4.1.1 啟發(Inspiration)

Hanocka[6]等人利用深度學習的方法將模型直接套用在卷積神經網路(Convolutional Neural Network)來做應用。卷積神經網路是一種由一個或多個卷基層(convolution layer)以及池化層(pooling layer)所組成。這種深度學習訓練方法主要是應用在影像辨識、影像分析上。他們改編來自 U-Net 的架構，將模型做簡化(collapse)，最後也能在模型分割上做更多的應用。這個神經網路的架構大部分適用於二維影像的資料訓練，然而如何套用在三維模型上將是個很大的挑戰。他們的方法可以將三維模型直接當作輸入，利用網格上的三角形(triangle mesh)資訊來去做訓練，最後可以得到簡化(collapse)的結果。在應用上可以做模型辨別(classification)以及模型分割(segmentation)，如 Figure 4.1。

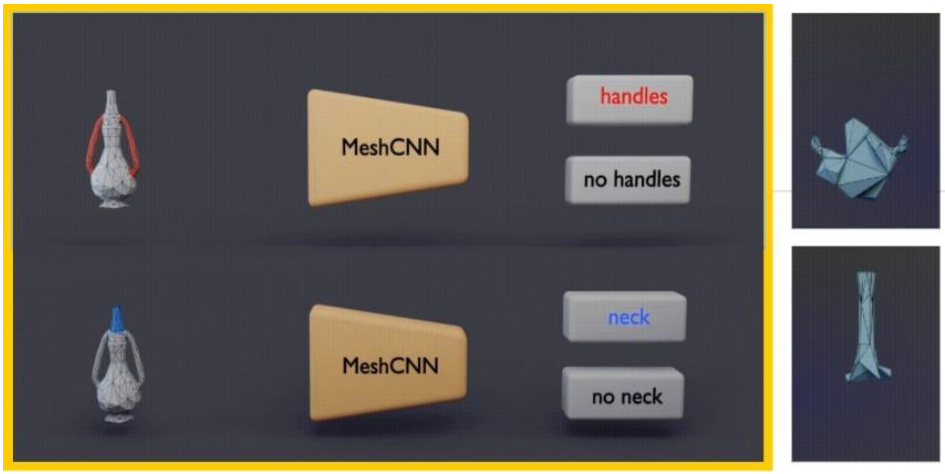


Figure 4.1: 此示意圖是來自 Ranocka[6]等人簡報，說明 3D 模型經過訓練簡化後可以做模型辨別(Model Classification)。

4.1.2 卷積 (Convolution)

首先，先對模型做卷積的步驟。此步驟處理在影像上並不困難，因為影像是基於像素(pixel-based)去做訓練，每一格像素都擁有固定的值和架構。這邊主要的挑戰是在於如何用這樣的架構來套用在不規則的模型網格(mesh)上。在此，我們的設計可以將網格三角形之間的關係去做卷積以及下一個步驟的池化層。

在模型網格的設定上，它們都要是流形網格(manifold mesh)，如 Figure 4.2。因此每兩個三角形網格之間必定會有一個邊(edge)，而這個邊必定會有兩個或四個鄰邊。利用這樣的關係，定義邊特徵 e、卷積核 k，並且藉由卷積原本的定義，制定出方程式：

$$e \cdot k_0 + \sum_{j=1}^{4} k_j \cdot e^j \tag{4.1}$$

在式子 4.1 中，四個鄰邊(e^1, e^2, e^3, e^4)可以是(a, b, c, d)或(c, d, a, b)兩個順序，例如可能會讓k_1先選 edge a 或 edge c 去做內積，如 Figure 4.3。因此制定出 symmetric functions 來解決不同順序選擇的問題，如此式子：

$$(e^1, e^2, e^3, e^4) = (|a - c|, a + c, |b - d|, b + d) \tag{4.2}$$

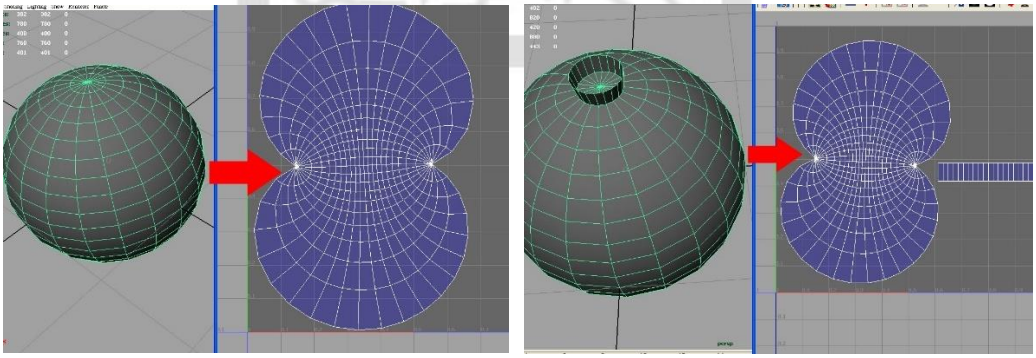


Figure 4.2: 左圖為流行網格，右圖為非流行網格。

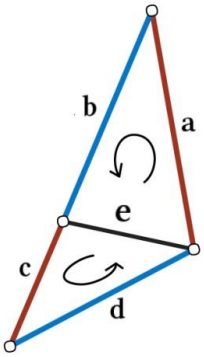


Figure 4.3: 從 edge e 做逆時針旋轉，可能會選取的順序(a,b,c,d)或(c,d,a,b)，會影響到卷積之後的平移不變特徵(translation invariance)

4.1.3 池化 (Pooling)

我們將傳統池化層的轉變為可以用在不規則的資訊。在此步驟有主要的三個核心：

1) 給訂鄰邊關係定義出池化區塊
2) 在池化區合併特徵線(merge features)
3) 重新定義鄰邊關係

在二維影像中，可能會出現類似 2×2 pooling。那麼在三維模型中，我們此步驟用來簡化邊線段，從兩個三角形中的五個邊合併為兩個邊，如 Figure 4.4。

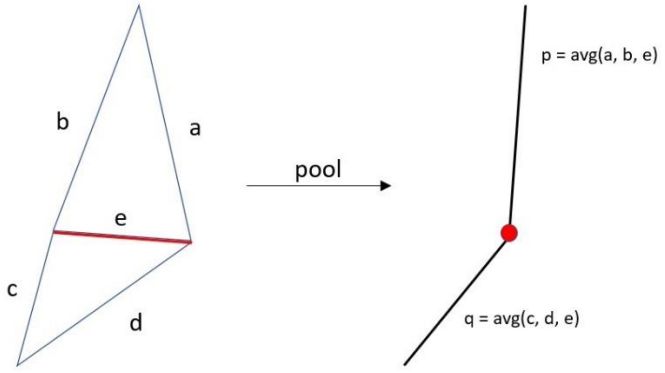


Figure 4.4: Mesh pooling 將五個線條合併為兩個線條。用三角形(a,b,e)及(c,d,e)做 average pooling 而得。

我們利用邊特徵的關係來讓網路去自行選擇哪些網格是適合於這樣的簡化方式，這樣的方法可以讓網路去計算重要邊，並且合併(merge)不重要邊，如 Figure 4.5 所示。在此步驟，我們會有兩個合併的過程，利用 average pooling 將三角形(a,b,e)及(c,d,e)做合併。我們定義邊特徵的 index 為 *i*，可以得以下池化公式：

$$p_i = avg(a_i, b_i, e_i),\ q_i = avg(c_i, d_i, e_i) \tag{4.3}$$

池化的過程，有一些條件是沒辦法被簡化的，例如非流行(non-manifold)的面無法被簡化合併、屬於頂點相關的邊線段。

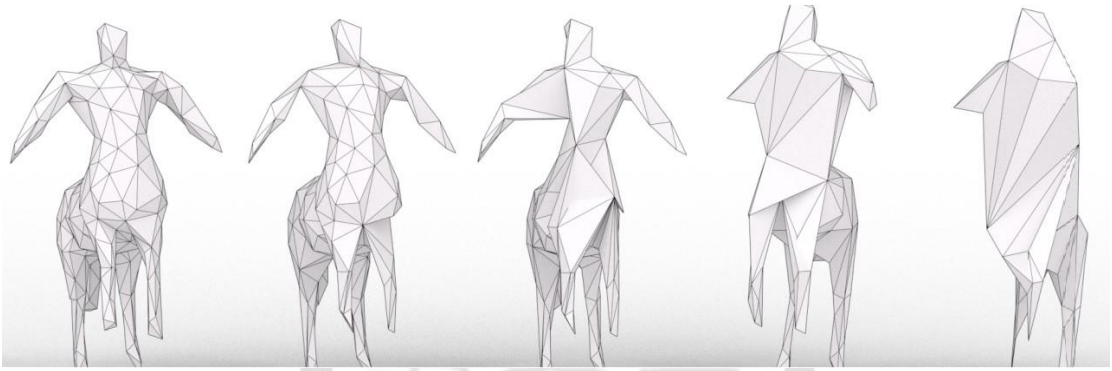


Figure 4.5: 人馬模型經過池化後合併呈現的過程。

4.1.4 上池化 (Unpooling)

此步驟是將池化後簡化的部分做還原，使原本壓縮的資訊還原以利後面的應用，因此這邊只是單純與卷積做結合來把失去的資訊取回來，如 Figure 4.6。每個池化層 (pooling layer) 與上池化層 (unpooling layer) 是配對在一起來進行上採樣 (upsample)。

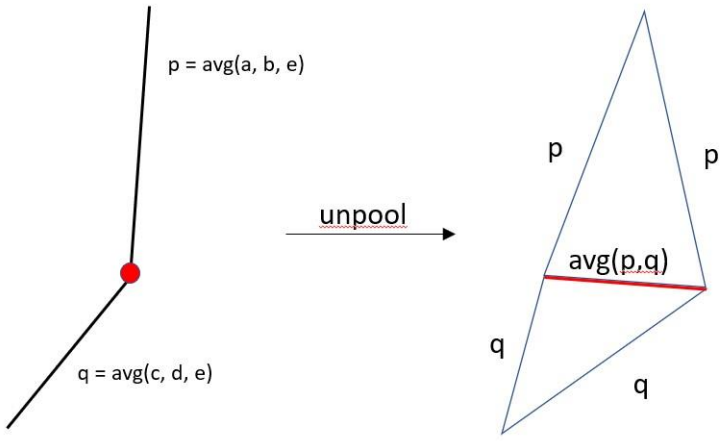


Figure 4.6: Mesh unpooling 使原本合併的線條做邊線段的還原。

4.1.5 網格分割(Mesh segmentation)

利用簡化的步驟取得邊特徵的資訊後，我們利用監督式學習(supervised learning)來在 MeshCNN 上做預測，讓網格自行去做分類。在這邊我們用了 COSEG [Wang et al. 2012]以及 Human Body Segmentation[Maron et al.2017]的資料庫來做訓練，因為這兩個資料庫是有提供分割後的標註數據(ground truth)。在訓練上，我們把訓練資練以及測試資料(train/test data)分成 85%及 15%。訓練模型改為適合訓練模型分割的參數，如 Figure 4.7。經過一系列的卷積跟池化後，我們得到模型分割的結果，如 Figure 4.8。

ResConv $f_{in} \times 32$
MeshPool $\rightarrow$ 1200
ResConv 32×64
MeshPool $\rightarrow$ 900
ResConv 64×128
MeshPool $\rightarrow$ 300
ResConv 128×256
MeshPool $\rightarrow$ 279
Segmentation

Figure 4.7: 模型分割網路架構來用於 COSEG 以及人形模型的資料庫。網路都是從 2250 個邊的模型去做訓練。此示意圖是訓練下半部的架構，上半部的部分是與此對稱(U-Net)。

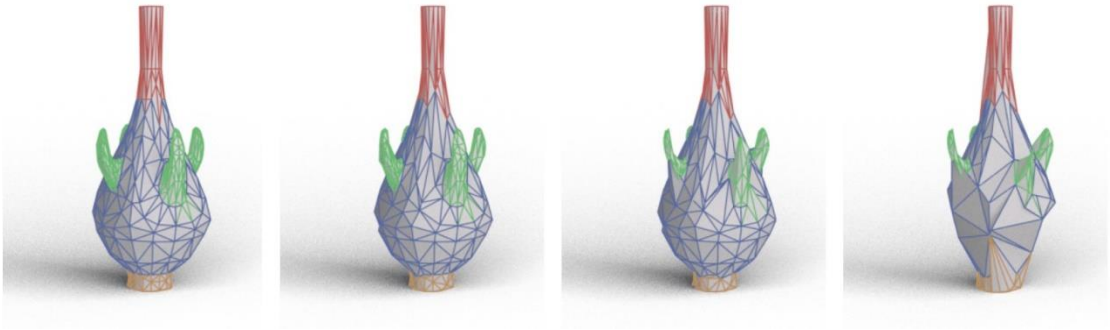


Figure 4.8: 模型分割其中一個花瓶的結果。

4.2 邊線段之萃取(Edge segments extraction)

4.2.1 前提概要(Abstract)

將模型的分割結果萃取來成為我們必定會經過的線段以後，如 Figure 4.9，接著我們利用 Garland[7]等人提出的三角形簡化的貪婪演算法去做為提取網格上重要的邊。他們是依序將網格上的邊進行收縮為一個頂點，利用此方式來移除原先相鄰三角形的面，如 Figure 4.10。再利用收縮的兩個端點相鄰的所有面做距離平方總和作為收縮邊的代價(cost)。計算完代價後依序由小到大進行合併與收縮，藉此達到簡化的效果，同時也可以看出原本模型的樣子。

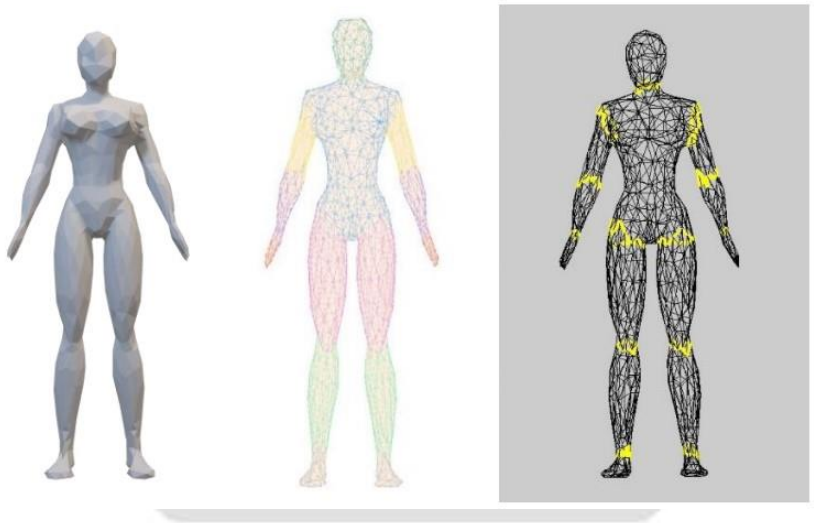


Figure 4.9: 左圖為原本模型樣式、中圖為分割結果、右圖為分割後將不同區塊顏色的交界萃取出必經過線段。

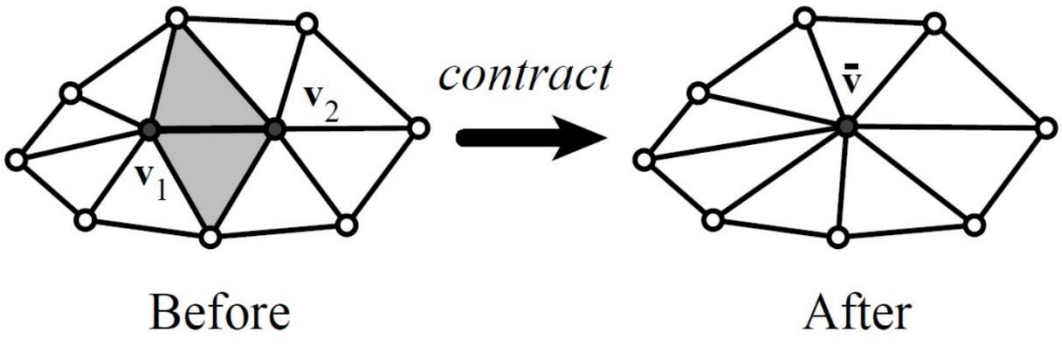


Figure 4.10: 左圖為收縮邊之前的形狀，灰色的區塊為即將被收縮的兩個面。右圖為被收縮後為一個頂點後的結果。

4.2.2 演算法流程(Algorithm process)

來自 Garland[7]等人的方法，我們提出一個相似的貪婪演算法(greedy algorithm)來取出三維模型網格中重要的邊。這樣的目的是在於希望可以移除網格上較不重要的邊，最後剩下的邊可以成為我們之後建立線條的基礎，我們稱為「邊線段(edge segments)」。這演算法主要是將較平滑的三角形網格區塊，並且利用共同邊的關係，將它們合併再一起，如 Figure 4.11。

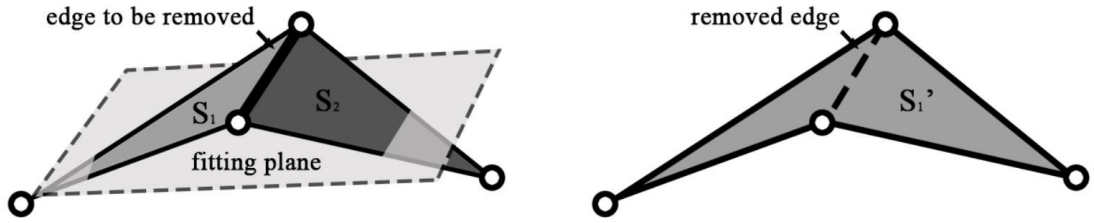


Figure 4.11: 左圖為利用算出來的擬合平面與S_1、S_2兩個三角形之間的平滑程度來作為兩集合之間的代價。若選擇合併則將共同邊消除，如右圖。

首先，我們對所有的三角形網格產生 *n* 個集合$\{S_1, S_2, S_3 \dots S_n\}$，並且讓每個集合$S_i$對應一個三角形面$F_i$，接著將這接鄰近的兩集合$S_i$、$S_j$作為一對。利用集合中三角形面所圍成的區塊的所有頂點為V_a，這些頂點可以算出擬合平面(fitting plane) P_f。我們用這些擬合平面以及三角形集合之間的差異程度來合併S_i、S_j，並且計算出代價$C_{i,j}$，每當有三角形需要被合併時，兩集合之共同邊就會被移除掉。利用$C_{i,j}$算出集合合併的代價後，我們將$C_{i,j}$由小到大依序移除共同邊，一直到保留的邊線段是符合指定的數量為止。在這邊我們也設定了一些限制來確保線段跟線段之間不會產生分離的狀況，也能確保使用者自己畫出來的邊能夠被保留下來。

4.2.3 計算擬合平面(Fitting plane implementation)

在三維空間中，我們計算出一個平面能夠擬合 *n* 個頂點，也就是利用這 *n* 個頂點的距離平方總和為最小。我們利用 Ernerfeldt[8]等人提出的線性回歸(linear regression)方法來算出擬合平面。

經過 Ernerfeldt 的預設平面方程式：

$$ax + by + d = -z \tag{4.4}$$

接下來將 *n* 個點帶入平面方程式中：

$$\begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ d \end{bmatrix} = \begin{bmatrix} -z_0 \\ -z_1 \\ \vdots \\ -z_n \end{bmatrix} \tag{4.5}$$

Ernerfeldt 在等號兩邊乘上轉置矩陣，用此為了找出最小平方法的平面方程式係數：

$$\begin{bmatrix} x_0 & x_1 & \cdots & x_n \\ y_0 & y_1 & \cdots & y_n \\ 1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ d \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & \cdots & x_n \\ y_0 & y_1 & \cdots & y_n \\ 1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} -z_0 \\ -z_1 \\ \vdots \\ -z_n \end{bmatrix} \tag{4.6}$$

簡化後得到以下方程式：

$$\begin{bmatrix} \sum x_i x_i & \sum x_i y_i & \sum x_i \\ \sum y_i x_i & \sum y_i y_i & \sum y_i \\ \sum x_i & \sum y_i & N \end{bmatrix} \begin{bmatrix} a \\ b \\ d \end{bmatrix} = -\begin{bmatrix} \sum x_i z_i \\ \sum y_i z_i \\ \sum z_i \end{bmatrix} \tag{4.7}$$

為了再進一步簡化，將所有頂點的x, y, z值相對於平均點的值，設$\sum x = \sum y = \sum z = 0$，經改寫後如下：

$$\begin{bmatrix} \sum x_i x_i & \sum x_i y_i & 0 \\ \sum y_i x_i & \sum y_i y_i & 0 \\ 0 & 0 & N \end{bmatrix} \begin{bmatrix} a \\ b \\ d \end{bmatrix} = -\begin{bmatrix} \sum x_i z_i \\ \sum y_i z_i \\ 0 \end{bmatrix} \tag{4.8}$$

再進一步作簡化：

$$\begin{bmatrix} \sum x_i x_i & \sum x_i y_i \\ \sum y_i x_i & \sum y_i y_i \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = -\begin{bmatrix} \sum x_i z_i \\ \sum y_i z_i \end{bmatrix} \tag{4.9}$$

最後用簡化後的結果再套入克萊姆法則(Cramer's rule)得：

$$\begin{aligned} Let\ D &= \sum x_i x_i \times \sum y_i y_i - \sum x_i y_i \times \sum y_i x_i \\ a &= (\sum y_i z_i \times \sum x_i y_i - \sum x_i z_i \sum y_i y_i) \div D \\ b &= (\sum y_i x_i \times \sum x_i z_i - \sum y_i z_i \sum x_i x_i) \div D \end{aligned} \tag{4.10}$$

在得到a, b值以後，再將平均點代入就能夠算出平面方程式d值。Figure 4.12 為擬合平面的例子，以便之後集合代價的計算。

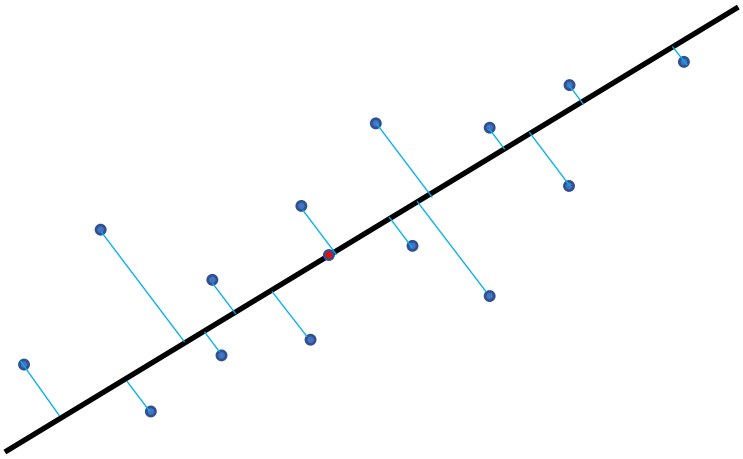


Figure 4.12: 圖為空間中 14 個頂點算出來的擬合平面側面圖。在線上的紅頂點為平均點，淺藍線代表深藍點距離平面的距離。

4.2.4 計算集合代價(Cost of pairs implementation)

利用 Hou[4]的方法，我們把所有三角形的面定義為$F = \{F_1, F_2, \dots, F_n\}$，$n$為面的數量，將所有面區塊裡的所有頂點$V_a$作為擬合平面的輸入。計算後的擬合平面與及合裡的三角形面必須為同向，因此而制訂出以下平滑化的方法：

$$C = \sum_{i=1}^{n} \frac{A_i}{A_{sum}} \sqrt{(a_f - a_i)^2 + (b_f - b_i)^2 + (c_f - c_i)^2 + (d_f - d_i)^2} \qquad (4.11)$$

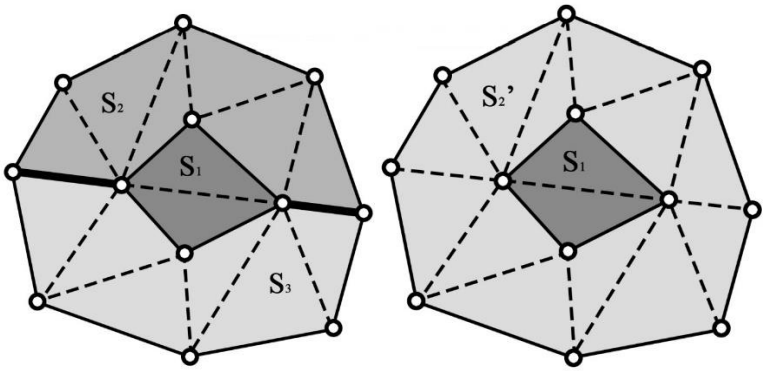
參數裡A_{sum}為所有三角形面的F總面積、A_i為三角形面F_i的面積，a_f、b_f、c_f、d_f為擬合平面的係數，a_i、b_i、c_i、d_i為三角形面的平面方程式係數，算出來的C為整個集合算出來的代價。在此，若算出來的代價C越大，代表越需要保留的邊；而代價C越小，代表邊越平滑也越可以被合併的線條。

4.2.5 集合合併之限制(Constraints for merging pairs)

在合併的過程中，我們會需要設定一些條件與限制，讓一些不能被合併的部分排除掉。此步驟是為了讓最後線條裡所有線段是連接在一起，不會被分隔開。

1. 集合之合併造成線段分離：

在合併集合時，有可能會發生線段分離的狀況，如 Figure 4.13。當S_2與S_3要合併時，如 Figure 4.13(a)，可以看到集合S_1會跟合併過的S_2'沒有線條連接而分離。因此在這邊我們 Figure 4.13(a)黑色的實線部分我們不會將它們做合併。

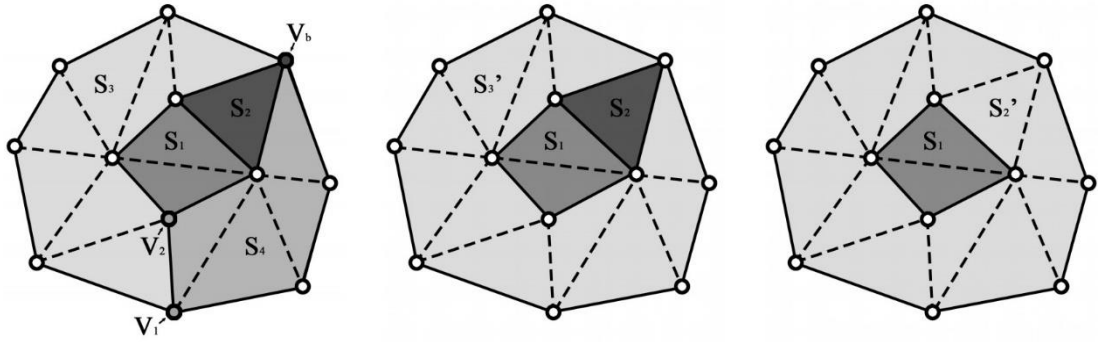


(a) 黑實線為S_2與S_3須被合併的線段。　　(b)此為S_2與S_3合併為S_2'集合。

Figure 4.13: 兩集合合併後使S_1與S_2'產生線段分離。

2. 合併之頂點不處於共同邊上：

若將三個處於共同集合的頂點的邊合併時，有可能使後續合併的結果跟 Figure 4.13(b)一樣，因此在這邊我們需要多幾個限制。如 Figure 4.14 所示，如果S_3與S_4為需要被合併的集合，會牽涉到三個頂點(V_1、V_2、V_b)。在這邊V_1與V_2屬於共同邊上，而V_b則不是。如果我們將S_3、S_4合併會形成如 Figure 4.14(b)的結果，設S_3'為合併的集合，若再將S_2與S_3'合併，會產生如 Figure 4.14(c)的結果。為了避免這樣的狀況，我們會先避免S_3、S_4之集合合併。



(a) S_4中V_1、V_2、V_b為相鄰頂點。　　(b)合併S_3、S_4形成S_3'的結果。　　(c)合併S_2、S_3'形成S_2'的結果。

Figure 4.14: 將不處於共同邊之頂點的集合合併後，可能也會產生集合分離。

4.2.6 結果(results)

完成合併後，我們將保留下來較重要的線段與深度學習產生的對稱線段做結合，如 Figure 4.15。

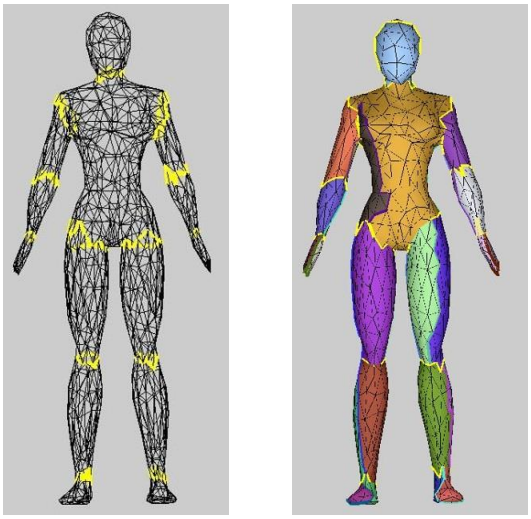


Figure 4.15: 左圖為從深度學習的分割獲得的線條結果。右圖為經過邊線段之萃取而得的模型分割結果。

4.3 連續線的組成(Wire composition)

4.3.1 啟發(Inspiration)

經由線段的形成以後，接下來要做的是如何將這些線段連起來組成連續線條。此方法我們參考 Liu[3]等人制定出的多旅行商問題來連接三維線段使線條數可以呈現最少，並且也使最近的線段能夠連接。

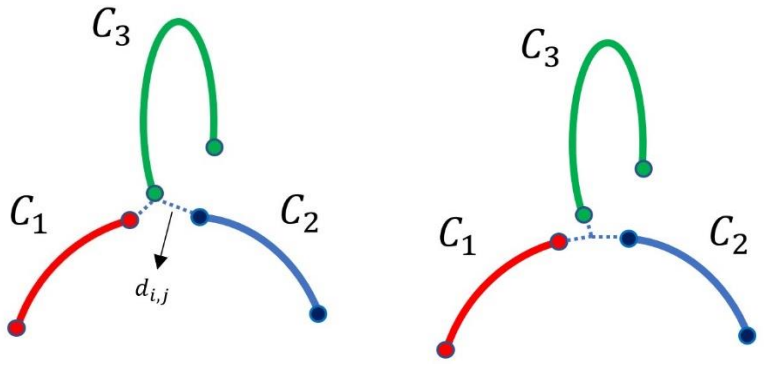
4.3.2 演算法(Algorithm)

Liu 等人將距離以及角度這兩項條件計算出來的代價來做為依據。他們假設線段之間最短的端點距離的條件為$d_{i,j}$，如 Figure 4.16(a)。角度則是計算端點之間平滑程度的條件，如 Figure 4.16(b)，用此來提取出線段的切線向量，並將角度條件定為：

$$a_{i,j} = (1 + cos\alpha)/2 \tag{4.12}$$

$a_{i,j}$為線段i與線段j之間的角度條件，α為最近兩端點與線條的切線向量之間的夾角。他們將線段之間的代價定義為：

$$w_{i,j} = d_{i,j} + \mu a_{i,j} \tag{4.13}$$



(a) 虛線$d_{i,j}$為C_1、C_2、C_3之間的最近端點距離，例如$d_{1,2}$為C_1、C_2線段之間的虛線。

(b)此虛線為各個線段端點之切線向量。

Figure 4.16: 此示意圖為線段之間的距離與切線向量。

Liu 等人因此制定出一個方法決定所有線段的連接，並且連接後的線條數量也是最少的。首先，他們將旅行商問題中線段的表示法分成節點之間的有向圖(directed graph)。接著先產生出一個虛擬起始點(dummy starting node)來作為線條連接後的起始點與終點，再來將計算後的路徑連接起來形成線條，如 Figure 4.17。最後再將決定後的順序以直線相連。

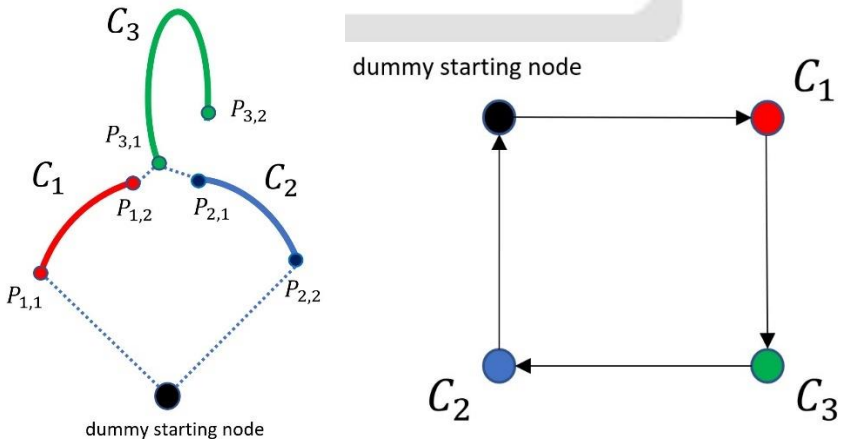


Figure 4.17: 假設C_1、C_2、C_3為要連接的線段，而黑點為虛擬起始點，$P_{i,j}$為端點位置。將所有線段最近的端點連接起來形成藍虛線的部分，如左圖。右圖是將線段表示為節點，箭頭方向為連接的順序。

利用多旅行商方法，他們制定出此方程式：

$$\{k^*, x_{i,j}{}^*\} \coloneqq argmin_{\{k,x_{i,j}\}} \sum_{i.j} x_{i,j}\, w_{i,j} + \delta k \qquad (4.14)$$

方程式中k為線段連接後的數量，亦為路徑的數量。設$x_{i,j}$為二元變數，若$x_{i,j} = 1$代表節點 i 到節點 j 是被包含再其中一條路徑；若$x_{i,j} = 1$代表節點之間並不包含在路徑中，$w_{i,j}$則為節點 i 與 j 之間的代價。另外，常數δ是會影響最後產生出線條的數量，例如：若δ越大，則 k 越小，代表產生出的路徑數量越少；若δ越小，則反之。

不過在他們的方法中會有缺陷，因為結果所產生出來的線條並不能保證是連續，也就是連接的過程可能會產生重疊的部分。如果線段計算後的順序為$C_1 \rightarrow C_3 \rightarrow C_2$，最後會使端點會有兩條與$C_1$、$C_2$連接的直線，產生出重疊，因為$C_3$與$C_1$、$C_2$的最近端點為同一個，如 Figure 4.18。

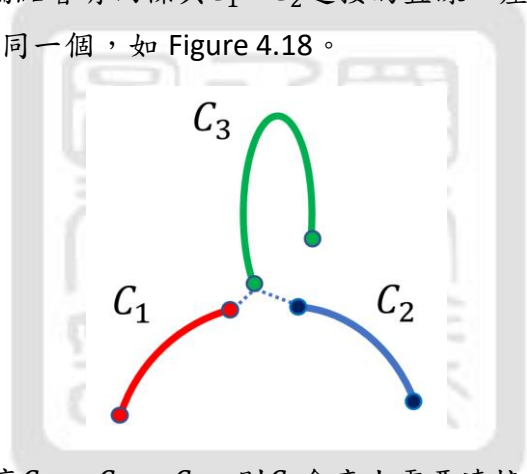


Figure 4.18: 連接順序$C_1 \rightarrow C_3 \rightarrow C_2$，則$C_3$會產生需要連接不同線段的兩條藍虛線，使線條會產生不連續性及端點重疊。

我們的改進。Liu 等人的方法把線段作為節點去做連接會產生重疊，因此我們改用線段中的端點作為我們的節點。利用一些約束的方式，讓此方法可以使經過的端點必定為一次，這樣可以使同個端點不會有重疊的關係，也能確保經過所有的線段路徑。將 Figure 4.18 為例，我們將線段中的端點作為節點，並且限制每個端點必定與線段相連，最後再用最佳化來相連所有路徑，如 Figure 4.19、Figure 4.20 所示。

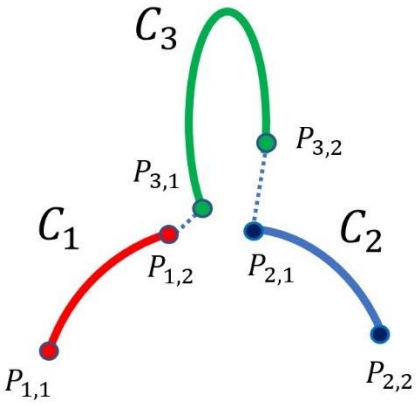


Figure 4.19: 我們將C_1、C_2、C_3中每個線段的兩端點作為節點$P_{i,j}$，並且做連接後的結果。這樣的方式可以避免同個端點會經過來自兩種不同線段。

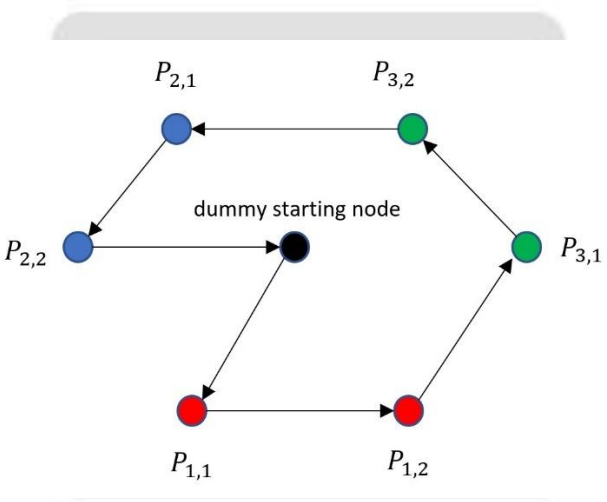


Figure 4.20: 假設圖中黑點為虛擬起始點，利用最佳化方法算出的路徑。可以看到所有線段與端點皆會被通過一次且不重疊。

Hou 等人的演算法。設線段的集合$C_{opt} = \{C_t\}, t = 1,2,\dots,n$，則所有線段兩端點的集合$P_{opt} = \{P_i\}, i = 1,2,\dots,2n$，這樣線段$C_t$中的端點可以表示為$P_{2t-1}$，$P_{2t}$。在此定義出虛擬起始點$V_0$作為所有線段路徑的起始點與終點，讓節點之間計算出$P_i$、$P_j$的代價，此代價的計算方式與 Liu 等人的相同，如方程式 4.13。為了確保線段與節點可以被包含在路徑中，我們設有向邊$e_{i,j}$一個二元變數$x_{i,j}$，若$x_{i,j} = 1$代表他是被包含在某一路徑中，而$x_{i,j} = 0$則不是。再來，我們讓經過的節點設一個整數變數u_i代表在路徑中被訪問的順序，例如虛擬起始點V_0的對應順序為$u_0 = 0$。最後，我們計算的最佳化問題與式子 4.14 一樣，希望最後算出來的代價以及路徑總數量為最少。在 4.14 方程式中，我們設定δ為最大數值$\max w_{i,j}$，讓路徑 k

為最小。

我們利用一些約束條件來計算方程式 4.14。我們為了讓每個節點剛好都被訪問過一次，我們限制節點中進入邊與離開邊必須被選到：

$$\forall i > 0 \ : \ \sum_{j,j\neq i} x_{j,i} = 1 \ ，\forall i > 0 : \ \sum_{j,j\neq i} x_{i,j} = 1 \tag{4.15}$$

限制 k 條路徑中，起始點與終點都是V_0：

$$\sum_{j,j>0} x_{j,0} = k \ ，\sum_{j,j>0} x_{0,j} = k \tag{4.16}$$

有些狀況可能會使最後的線段組成是不相連的環，為了避免這狀況，也用了 Kara[9]等人所提出的子路徑消除約束(subtour elimination constraints)：

$$u_i + (L-2)x_{0,i} - x_{i,0} \leq L-1，\forall i > 0 \tag{4.17}$$

$$u_i + x_{0,i} \geq 2，\forall i > 0 \tag{4.18}$$

$$u_i - u_j + Lx_{i,j} + (L-2)x_{j,i} \leq L-1，\forall i,j > 0, i \neq j \tag{4.19}$$

L 為在路徑中經過的節點最多數量，方程式 4.17、4.18 限制了每一條路徑所拜訪的節點數量之上限與下限，方程式 4.19 則是確保有向邊最後是被包含在路徑中。

為了確保線段上兩端點必須是相連，我們使端點其中一條有向邊一定會被選取，如 4.20 方程式。相反的，我們也希望彼此不相鄰的線段不會被相連，如 4.21 方程式。

$$x_{i,j} + x_{j,i} = 1 \tag{4.20}$$

$$x_{i,j} + x_{j,i} = 0 \tag{4.21}$$

經過 CPLEX 最佳化的計算以後，我們得到線條組成的成果，最後可以利用這樣的結果組成線條藝術。

Algorithm 1 Algorithm of our mTSP

Input: Set of 3D line segments $C_{opt} = \{C_t\}$
Collect endpoints (P_i) of each C_t; /* Each C_t consists of two endpoints */
$P_{opt} = \{P_i\}$;
$\mathcal{V} \leftarrow P_{opt}, \mathcal{E} \leftarrow \{w_{i,j}\}$;
Construct graph $\mathbf{G} = (\mathcal{V}, \mathcal{E})$;
$V_0 \leftarrow$dummy node;
for each path p_k **do**
    $V^k = \{V_t^k\}$; /* Set of nodes that are visited in path p_k */
    $V_0^k \leftarrow V_0$; /* Starting a path with dummy node */
    $V_1^k \leftarrow V_i$; /* V_i is randomly selected in $\mathcal{V}$ */
    while $V_c \neq V_0$ **do** /* V_c is the last visited node */
        for each node $V_j \in \mathbf{G}(\mathcal{V} - V^k)$ **do**
            if V_j is on the same segment with V_c **then**
                Add V_j to V^k;
                $V_c \leftarrow V_j$;
            else
                Calculate cost from V_j to the nodes on $\mathbf{G}(\mathcal{V} - V^k)$
            end if
        end for
        Find the node (V_{min}) that has the smallest cost to V_j on $\mathbf{G}(\mathcal{V} - V^k)$;
        Add V_{min} to V^k;
    end while
end for
Find minimum number of paths by solving Eq.(4.14) associated with the constraints in Eq.(4.15) to Eq.(4.21);
Output: minimum number of paths.

以上是我們的演算法 pseudocode，設定好 3D 線段以及定義端點與虛擬起始點，經過多旅行商問題以及我們設定的限制條件，我們可以得到最終連接線段的最少線條組成。

4.4 線條的連續性(Wire continuity)

在模型的建構上，我們採用球面、柱面、環面，如 Figure 4.21。我們利用這三種模型結構來建立出實際上的線條藝術。產生線條後，我們發現有某些部分會有不連續的線條形成斷層，也就是某些線條看起來是不連續且不自然，容易曲折。因此我們設想將不連續線條的部分手動連接起來。

(a)球面 (b)柱面 (c)環面

Figure 4.21: (a)球面作為模型上的頂點。(b)柱面作為模型上的邊。(c)環面作為模型上的連接處。

在 Figure 4.22，我們發現在人型線條模型中的腰與腿之間的交接處會產生出線條沒有連接在一起的狀況以至於有分離的現象，因此我們制定出使用者介面將交接處的環面 A(Torus A)與環面 B(Torus B)合併起來，並且移除兩環面，使最後產生的線條看起來較連續。

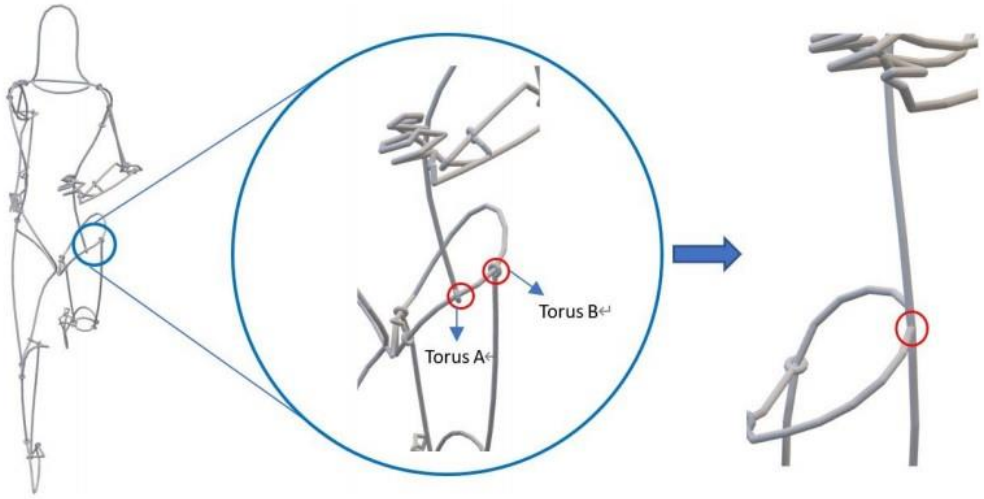


Figure 4.22: 人的模型產生線條後，選取兩個不自然線段形成的環面(如中圖)。最後合併兩線段，移除環面，形成單一連續線段(如右圖)。

4.5 線條平滑化(Wire smoothing)

經過模型分割、邊段的萃取、線條的組成以及連續線段以後，因為線條都是基於網格上的資訊來形成，我們發現在模型中還是會有一些線條看起來是較粗略且不平滑的狀況。因此利用 Hou 等人制定最佳化問題的方法將線條做平滑化，如 Figure 4.23。

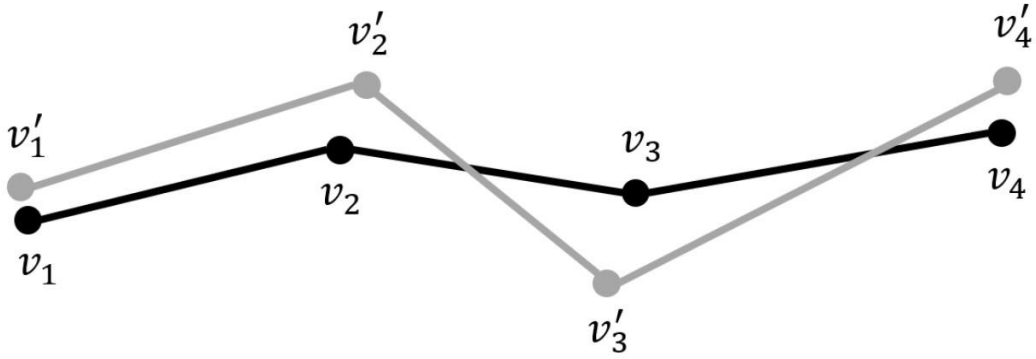


Figure 4.23: 假設原本線條如灰色線條，v_i'、v_i分別為平滑前及平滑後的頂點位置，黑色線段為平滑後的線條。

Hou 等人對粗略的線條做最佳化方程式計算：

$$argmin_{v_i} \sum_{i=3}^{n} \|(v_i - v_{i-1}) - (v_{i-1} - v_{i-2})\|^2 + \alpha \sum_{i=1}^{n} \|v_i - v_i'\|^2 \quad (4.22)$$

在方程式中v_i為線段上的頂點、n 為線條上頂點的總數量、α為衡量兩項式的係數。方程式第一項裡，我們希望線條邊與邊之間的向量盡量相同；第二項則是希望新算出來的頂點與原本的頂點越近越好，用這樣的方式能夠盡量使線條曲折度降低，並且也能盡量保持原本應有的形狀，如 Figure 4.25。

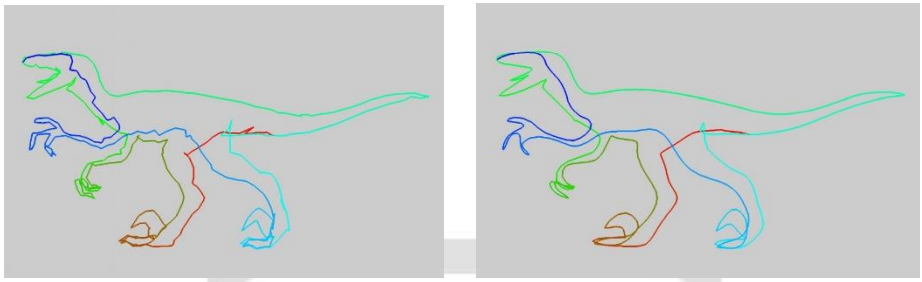


Figure 4.24: Hou 等人利用最佳化將較曲折的線條加以平滑化。

Chapter 5

Results and Discussion

5.1 設定(Settings)

在模型訓練上，我們用 Pytorch 來對 MeshCNN 架構去做訓練，主要有包含的模型有人型、花瓶、動物等。我們將每個類別拆成 85%/15%來去做訓練資料以及測試資料，而每個類別的資料都包含 300 個模型以上。在硬體設備上搭配了 i7-7700 CPU(3.60GHz)處理器以及顯示卡 GTX1080，在每個類別的模型訓練上需要兩到三小時。

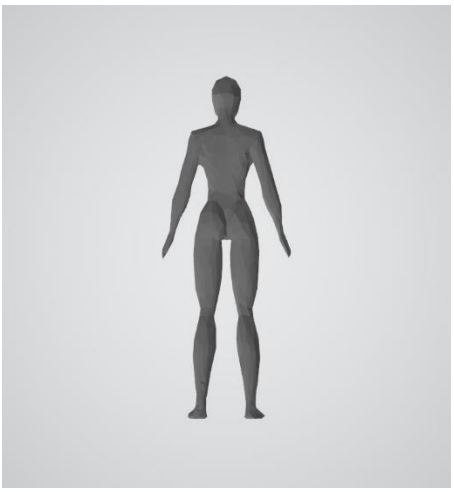
在產生路徑以及線條的介面上，我們運用本實驗室侯志穎等人在 Visual Studio 2013 的 Windows Form 上的專案，並且包括了 OpenGL 來做為視覺介面功能。在萃取線段以及計算多旅行商的最佳化問題，我們用 C++以及 CPLEX Optimizer (version 12.63)等程式語言及功能來計算。

5.2 結果與評估(Results and evaluations)

在結果中，我們經過訓練以及線條自動產生了 6 筆結果。以下將產生出的結果與 Hou 等人自動產生出的結果做比較，以及每個模型在不同視角的角度皆呈現，另外還有運算時間上的表格。

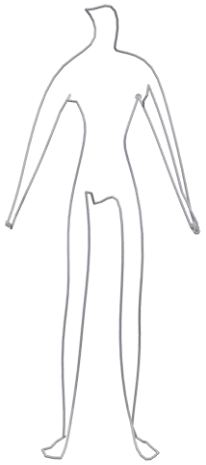
模型邊段萃取與線條組合		
模型	線條數量	執行時間(秒)
Human-1	20	4.234
Human-2	22	5.021
Bird	8	2.523
Cat	10	2.989
Vase-1	16	3.534
Vase-2	18	3.586

Figure 5.1: 此表之模型皆為三角型邊數量為 2250



(a)輸入模型

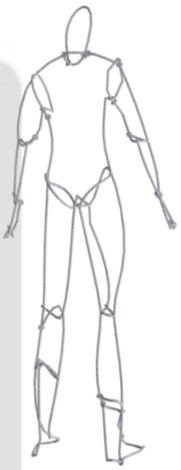
(b)我們的結果



(c)Hou 等人結果

(d)視角 1

(e)視角 2



(f)視角 3

(g)視角 4

(h)視角 5

(i)視角 6

Figure 5.2：模型 Human-1 結果

(a)輸入模型

(b)我們的結果

(c)Hou 等人結果

(d)視角 1

(e)視角 2

(f)視角 3

(g)視角 4

(h)視角 5

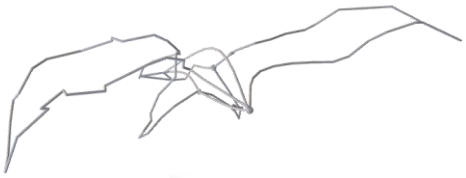
(i)視角 6

Figure 5.3：模型 Human-2 結果

(a)輸入模型

(b)我們的結果

(c)Hou 等人結果



(d)視角 1

(e)視角 2

(f)視角 3

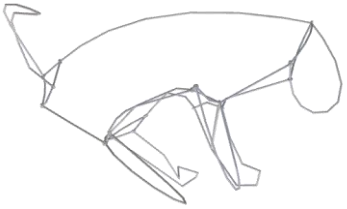
(g)視角 4

(h)視角 5

(i)視角 6

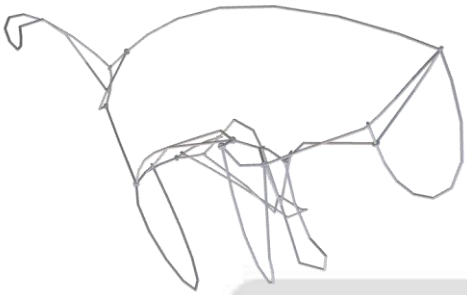
Figure 5.4：模型 Bird 結果

(a)輸入模型



(b)我們的結果

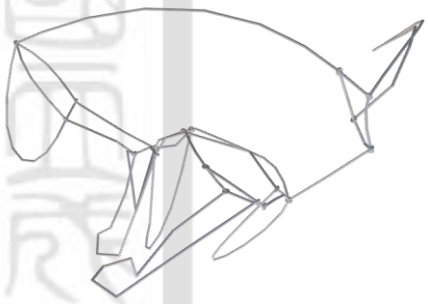
(c)Hou 等人結果



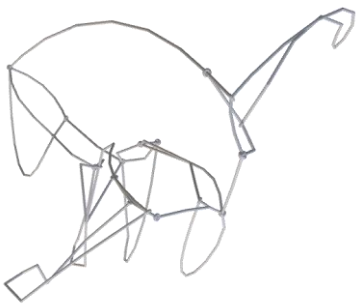
(d)視角 1

(e)視角 2

(f)視角 3



(g)視角 4



(h)視角 5

(i)視角 6

Figure 5.5：模型 Cat 結果

(a)輸入模型

(b)我們的結果

(c)Hou 等人結果

(d)視角 1

(e)視角 2

(f)視角 3

(g)視角 4

(h)視角 5

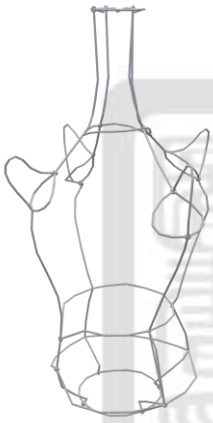
(i)視角 6

Figure 5.6：模型 Vase-1 結果

(a)輸入模型

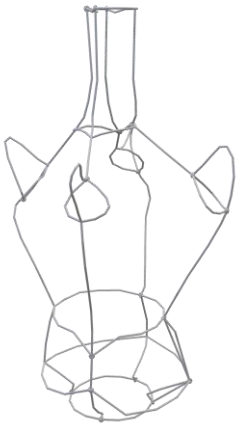
(b)我們的結果

(c)Hou 等人結果



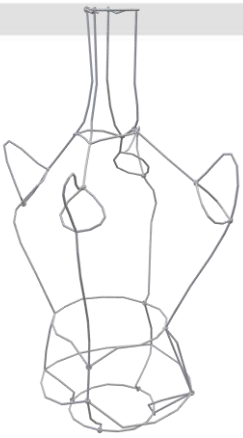
(d)視角 1

(e)視角 2

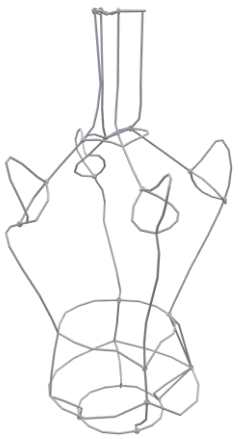


(f)視角 3

(g)視角 4



(h)視角 5



(i)視角 6

Figure 5.7：模型 Vase-2 結果

我們可以發現侯志穎等人的結果會出現部分沒有產生出線條，例如花瓶的例子 Figure 5.7，只有部分線條有產生出。造成這樣的結果可能是因為在計算代價時沒辦法在整體網格上有較均勻的分布。經過整合深度學習得到的網格分割會產生出對稱線條，在合併計算代價時，這些線條也必定會經過對稱線條，因此可以使最後的結果較均勻且較有體積感。在本實驗中，只有在連續性的步驟上需要讓使用者自己手動調整，使線條之間不會有斷層的感覺。

以下結果是經過 Huang[12]等人所提出的 Hu moments 演算法將原本模型轉成 2D 影像來進行 shape matching 比對以及結果的相似度評估，數字越低代表相似度越高。

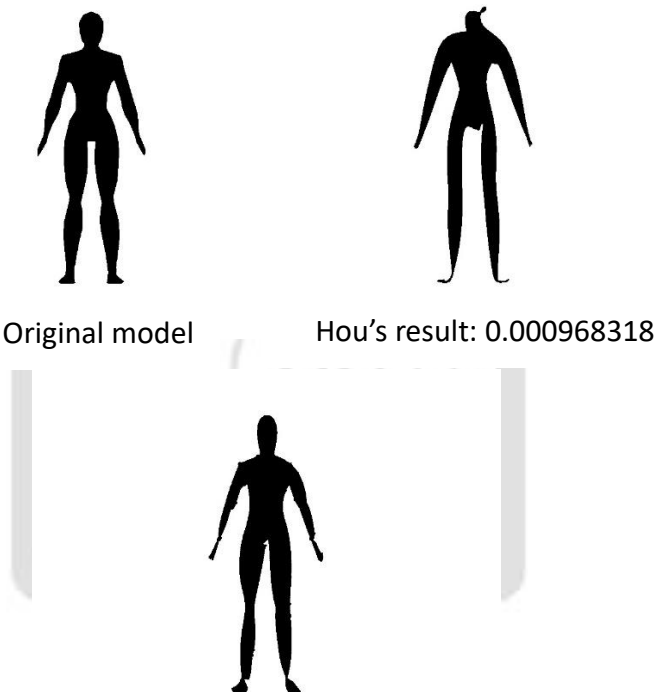


Figure 5.8: 模型 Human-1 經過 Hu moments 比較後的結果。

Original model Hou's result: 0.005876

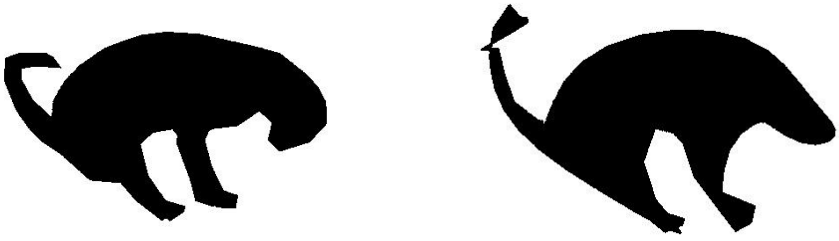
My result: 0.00162977

Figure 5.9: 模型 Human-2 經過 Hu moments 比較後的結果。

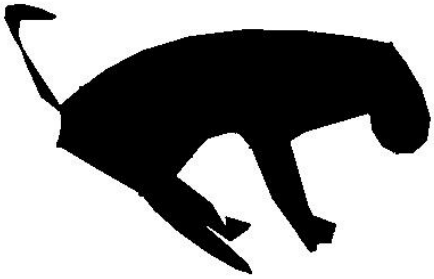
Original model Hou's result: 0.00349423

My result: 0.00033544

Figure 5.10: 模型 Bird 經過 Hu moments 比較後的結果。



Original model Hou’s result: 0.000296299



My result: 0.00018435

Figure 5.11: 模型 Cat 經過 Hu moments 比較後的結果。

Original model Hou’s result: 0.00222939

My result: 0.000309693

Figure 5.12: 模型 Vase-1 經過 Hu moments 比較後的結果。

Original model

Hou’s result: 0.00393699

My result: 0.00102312

Figure 5.13: 模型 Vase-2 經過 Hu moments 比較後的結果。

Chapter 6

Conclusion

在本論文中，利用 MeshCNN 訓練的方法得到對稱切割(segmentation)，再利用邊萃取來移除網格模型上較為不崎嶇的邊，來保留下重要的邊線段。接著再利用多旅行商問題來連接被保留的線段，最後將不連續線段做調整以及將較為崎嶇的線條做平滑化。

我們最後生成的結果做平滑化還是會使某些線條看起來較不平整，可能是因為我們是基於 mesh-based 的方法做平滑，使得一些崎嶇的網格雖然經過平滑計算，但還是看得出鋸齒的形狀。我們的方法可以盡量保持原有模型的體積性及對稱性，然而在一些模型的某些角度實際上是很難保持原有的體積，因此希望之後可以參考其他方法可以使線條看起來更有體積感，以及 Fu[11]等人的方法來代替多旅行商問題的最佳化方法來組成線條。在連續線的產生也希望能夠自動計算，進而取代掉手動調整的部分。

References

[1] Hsiao, Kai-Wen, Jia-Bin Huang, and Hung-Kuo Chu. "Multi-view wire art." *ACM Trans. Graph.* 2018.

[2] Wang, P., Gan, Y., Shui, P., Yu, F., Zhang, Y., Chen, S., & Sun, Z. 3D shape segmentation via shape fully convolutional networks. *Computers & Graphics*, *76*, 182-192. 2018

[3] Lingjie Liu, Duygu Ceylan, Cheng Lin, Wenping Wang, and Niloy J. Mitra. Image-based reconstruction of wire art. ACM Trans. Graph., 36(4):63:1–63:11, July 2017.

[4] YEH, CHIH-KUO, et al. "Generating Virtual Wire Sculptural Art from 3D Models." 2021.

[5] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: Designing freeform surfaces with 3d curves. In ACM SIGGRAPH 2007 Papers, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[6] Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., & Cohen-Or, D. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, *38*(4), 1-12. 2019.

[7] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '97, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[8] Emil Ernerfeldt. Fitting a plane to noisy points in 3d, September 2017.

[9] Imdat Kara and Tolga Bektas. Integer linear programming formulations of multiple salesman problems and its variations. European Journal of Operational Research, 174(3):1449 – 1458, 2006.

[10] Rusinkiewicz, S. Estimating curvatures and their derivatives on triangle meshes. In Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. (pp. 486-493). IEEE. 2004, September**.**

[11] Lira, W., Fu, C. W., & Zhang, H. Fabricable eulerian wires for 3D shape abstraction. *ACM Transactions on Graphics (TOG)*, *37*(6), 1-13. 2018.

[12] Huang, Zhihu, and Jinsong Leng. "Analysis of Hu's moment invariants on image scaling and rotation." *2010 2nd International Conference on Computer Engineering and Technology*. Vol. 7. IEEE, 2010.