國立交通大學

數學建模與科學計算研究所

碩士論文

利用置換邊之方法縮減平面三角網格

Using swap edges method for 2D coarse triangle mesh

研 究 生：王鈞澤

指導教授：陳福祥　教授

詹益政　教授

中華民國 一百零二年四月

摘要

Triangular Mesh 被廣泛的運用在各種方面，諸如模型的建立、電腦影像、在有限元素上做分析等等。在運算資料上 mesh 結構越緊密之處越能算出精確的數據，針對 mesh 上的結構如何增加或縮減並不破壞其結構是眾多人研究的問題。

本文針對 mesh 縮減上提出一種方法，以不破壞整體結構上做適度減少結點，使整個 mesh 結構達到縮減的目的。在處理過程中，可以針對區域規模做調整而不必在不需要縮減之處做計算。

關鍵字：Triangular Mesh、edge collapse、smoothing

致謝

這次的研究首先要感謝詹益政教授提出這個值得研究的方向給我，另外特別感謝一直給予我幫助的陳福祥老師，在兩位老師給予的指導下本論文才得以完成，最後感謝一直在背後支持我的家人，沒有大家的鼓勵與幫忙這次的研究真的難有結果，再次感謝。

目錄

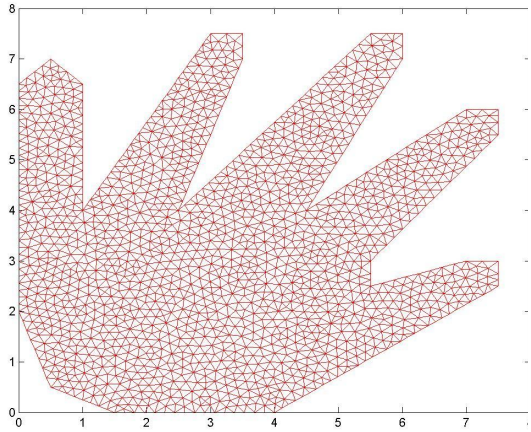
1.緒論

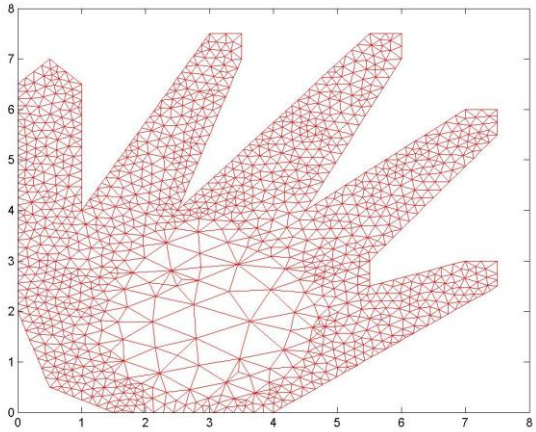
1.1 動機

Triangular Mesh 在科學計算上被運用了很久，從流體力學上來看，對一個區域做分析，常用的方式就是網格化然後代入值做計算進而達到想知道的近似結果。以左下圖爲例，將一區域三角網格化得到以 1100 多個節點組成的 Mesh，1100 多個節點組成約 2000 個小三角區域，在計算上須將 2000 多個小三角區域一一算過後得出結果再做分析，如果想得到更精細的結果則須將此區分化成更多更小的區域(ex.2000 以上節點 4000 以上三角網格)。

這樣的計算確實可以獲得此區域的近似解，不過並不是每個區域都需要如此精細的計算，假設一個區域以 1000 個三角網格爲基準，這表示必須做 1000 次的計算才能獲得結果做分析，在不需要很高的精準度的情況下這就造成計算上的浪費，因此減少區域內網格後再計算就成了一種方法，雖然精準程度會下降但是相對的減少的計算上成本。假設簡化後的網格剩餘約 10~20 個表示只需計算 10 次即可知道此區大概的情況，計算上減少了 100 倍。

正因爲科學計算上並不想在所有節點上做計算，因此我們希望僅在想縮減的區域縮減，而不是全部重構，因此如何選取區域簡化 Mesh 又不破壞原本結構(邊界組成)即成爲本次研究的問題。

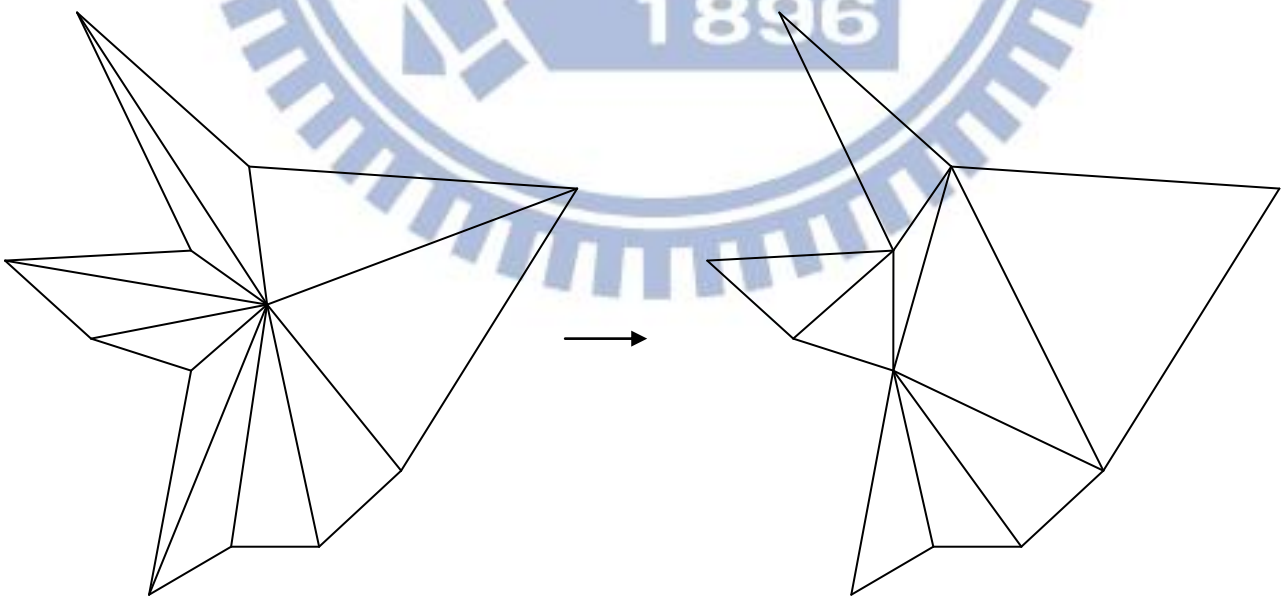




1.2 研究目的及方法

將一個原始的 Mesh 以簡單的方式進行有限步驟簡化，即爲本次研究的目的，雖然目前已存在將網格簡化的其他方法，但是各有缺點此點將在相關研究中說明。

在想法上將一區域的 Mesh 簡化不外乎實際的減少內部網格或節點，本論文的方向以節點爲準，藉由實際減少節點來簡化網格，做法上則是將以某個節點形成的小區域重新規劃後拿掉此點如下圖所示。在此方法下化簡的區域可以被限定，並且不會破壞外部結構，局部簡化後造成某些網格尖銳化則以 smoothing 技術和其他適用方法來處理。



2.相關研究

在 Hugues Hoppe 的研究中，Progressive Meshes 被利用在 Triangle Mesh 的處理上，是針對增加或減少 Mesh 的一種方法。圖 1.爲原論文中想法圖。向右的過程爲縮減 Mesh，向左則可看爲增加。不過此方法有條件限制，並非任意的點或邊皆可實行，必須找出適合的邊才能進行縮減。如果對任一邊縮減則可能導致 Mesh 結構不符如下圖 2.。在找尋適合縮減之邊上所花時間也不少。

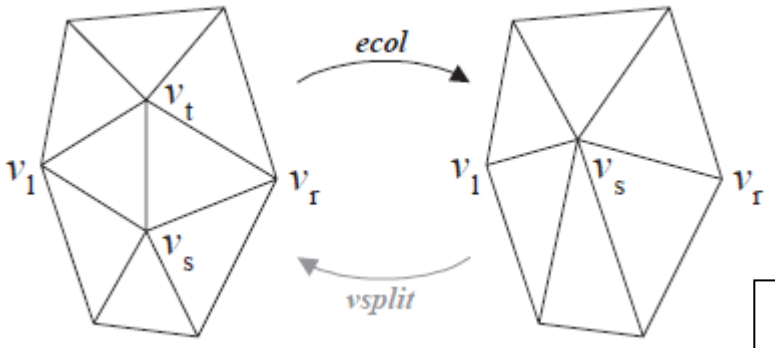


圖 1.

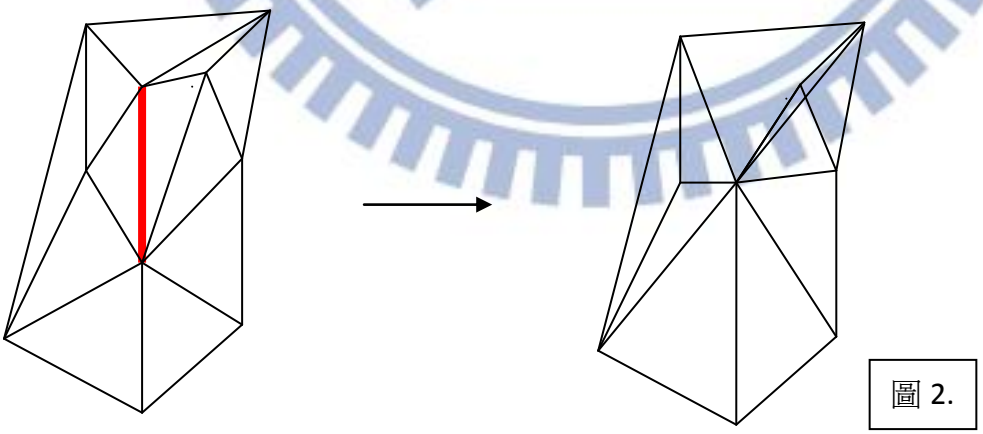


圖 2.

3.定義與理論

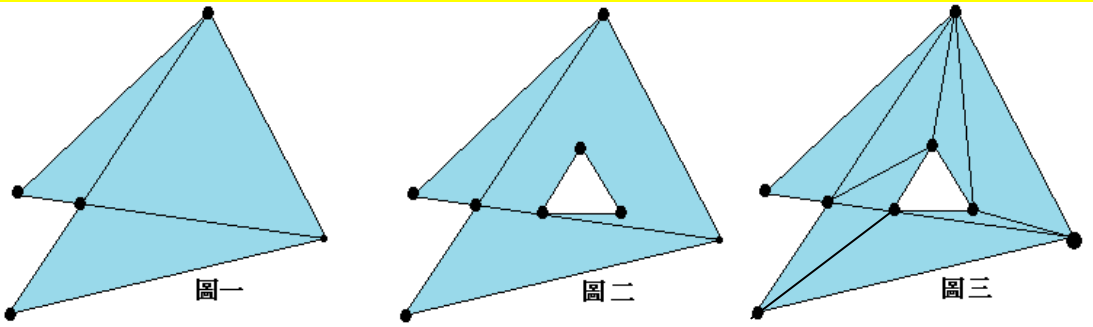
對於我們提出的方法這裡給一些定義及理論：

定義：**二維三角網格**

二維平面上一封閉區域集合 G 稱爲**三角網格**,是指 G 由有限個相連的元素(此時三角形稱爲元素)所組成的結構,其中任意元素之邊皆僅由 G 中的兩個節點相連. 這裡節點就是各元素的頂點. 當元素換成四邊形時, G 就稱爲四邊形網格.

我們可以仿此定義三維空間上的四面體網格,但本論文僅限於討論平面三角網格.

例如: 圖一,圖三 是三角網格, 圖二不是三角網格.



定義：**內部三角形**(元素)與**邊界三角形**(元素)

在二維三角網格 G 中因爲是封閉區域，所以有邊界形成，在 G 中所有的元素皆由三角形構成，因此只要三角形其中一邊爲邊界則定此三角形爲**邊界元素**，反之沒有任一邊爲邊界的元素則訂爲**內部三角形**。

例如: 圖三有 5 個內部三角形,4 個外部三角形

定義：**內部節點**與**外部(邊界)節點**

在二維三角網格 G 中因爲是封閉區域，所以有邊界形成，而所有的邊皆由兩節點組成，因此只要在邊界上的節點即稱爲外部節點，反之則爲內部節點。

例如: 圖一沒有內部節點,圖三有 3 個內部節點,5 個外部節點.

定義：**縮減網格**

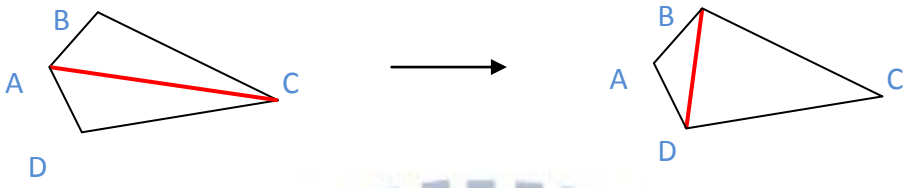
假設 G 由 N 個節點 M 個元素組成且邊界所構成區域爲 B，另一個 H 也是二維三角網格有相同的邊界區域 B，若 H 由 P 個節點 Q 個元素組成，且 P<N&Q<M 則稱 H 爲 G 縮減後之網格。在此所有代數字母皆爲正整數。

例如: 圖一是圖三的縮減網格

定義：置換邊(swap edge) (置換法)

由四點所形成之凸四邊形(ABCD)中，存在兩條對角線(AC、BD)，在三角網格結構上兩對角線只能取一，假設初始對角線爲 AC，將其改爲 BD 則稱爲置換邊。

例如: 下圖中紅線爲對角線



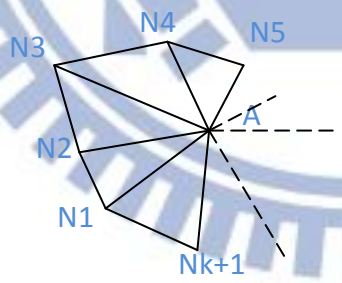
定理 1：四點形成的四邊形中只有凸四邊形可以有置換邊。

定理 2：任何二維三角網格 G 只要存在內部元素，皆存在縮減網格 H。

定理 3：二維三角網格 G 中經由任意指定內部範圍中的節點消除能達成我們想控制的縮減網格 H。

定理 4：二維三角網格中，以內部節點 A 爲中心所形成的內部區域，存在至少一個由 A 點與 3 連續節點的凸四邊形。

例如：下圖點 A 與 N2,N3,N4 所形成的四邊形即爲凸四邊形



定理 1：四點形成的四邊形中只有凸四邊形可以有置換邊。

證明：

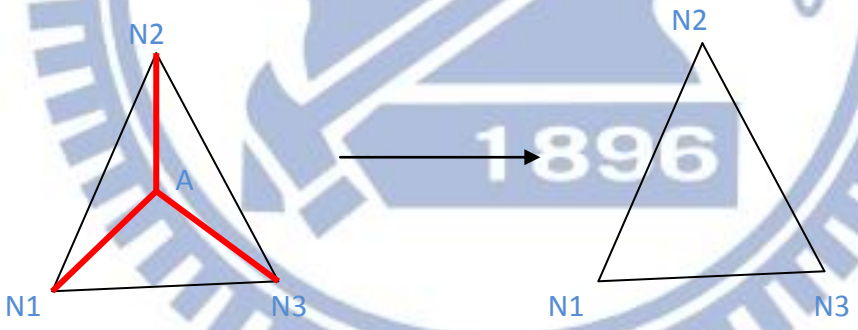
二維四邊形的情況只有兩種，凸四邊形或凹四邊形，置換邊的意思爲若四邊形依順時針或逆時針節點排序爲 ABCD，AC 或 BD 的連接邊即爲對角線，將原本是 AC 換爲 BD 稱爲置換，凸四邊形兩者互換並無問題，而凹四邊形在三角網格結構下其對角線只可能有一條因此不能置換，若是置換所形成的網格並不爲三角網格。

定理 2：任何二維三角網格 G 只要存在內部元素，皆存在縮減網格 H。

證明：

假設一節點 A 爲 G 之內部節點，所有與 A 相連的節點共 n 個，此 n 個節點{N1,N2,….Nn}也爲內部節點，以 A 爲中心所構成的三角網格即爲 G 中之內部元素。我們想藉由去除 A 點後將剩餘 n 點重新連接來達成縮減網格。利用數學歸納法由 n=3 組成的網格開始討論：

n=3

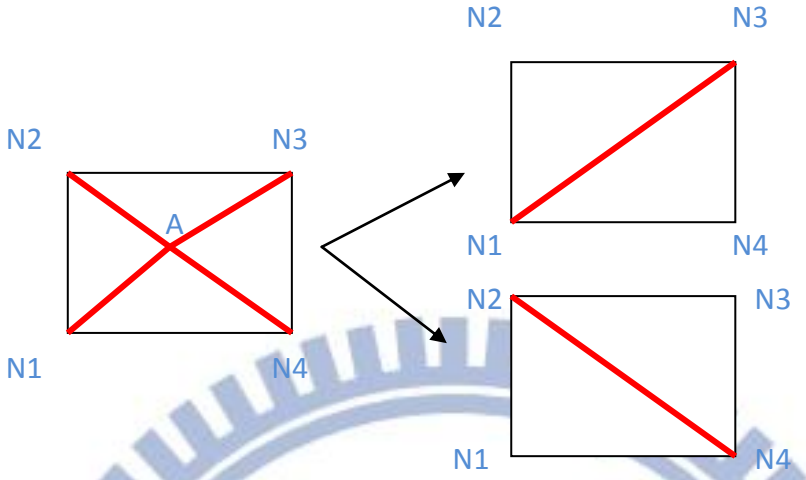


A 點去除的情況下與其相連邊亦拿掉，剩下的三個節點不需做任何處理直接爲三角網格，就網格縮減的定義來看節點與元素都比原本少(P=3<N=4 & Q=1<M=3)，因此成立。

n=4

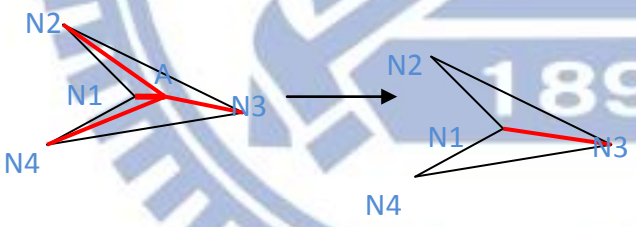
在此有兩種可能分別做討論：

1.凸四邊形



A 點去除相連邊拿掉後，剩下節點在不破壞三角網格結構下重連，其可能性只有兩種，任一種皆爲三角網格，而網格縮減的定義下(P=4<N=5 & Q=2<M=4)，因此成立。

2.凹四邊形



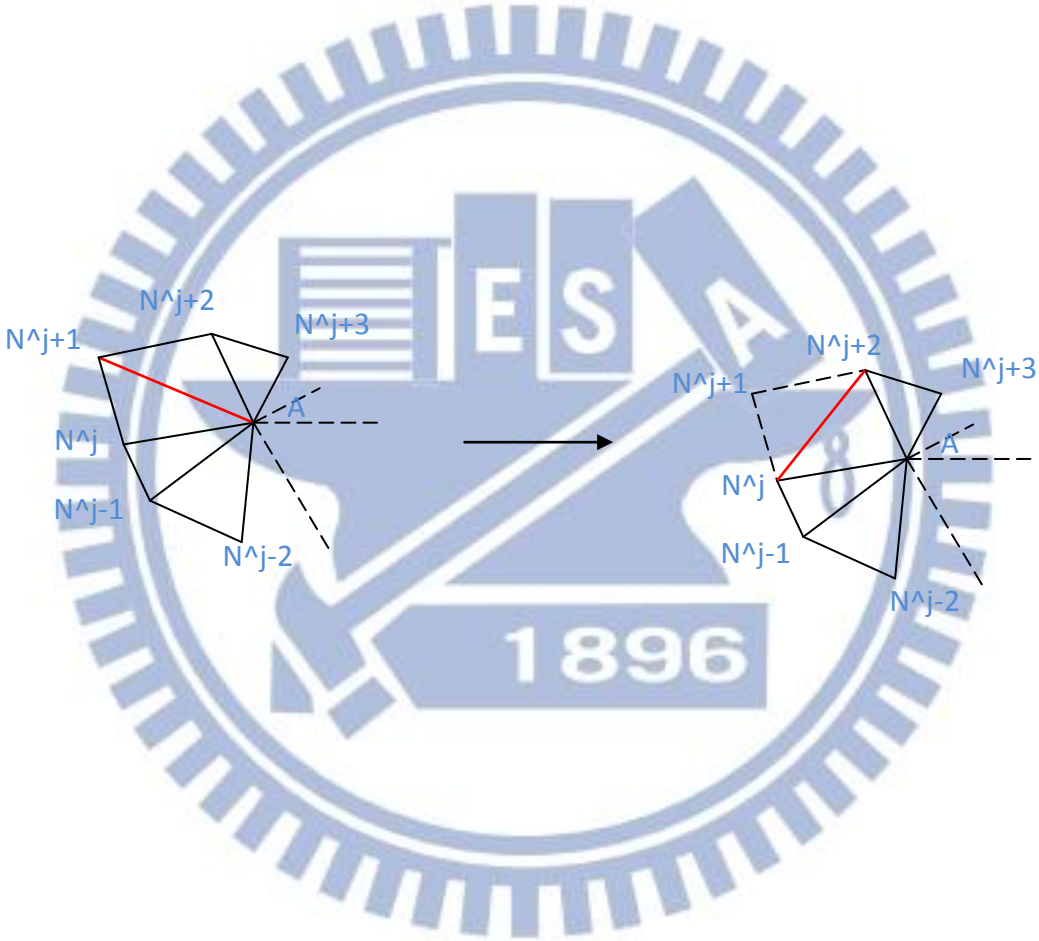
A 點去除相連邊拿掉後，剩下節點在不破壞三角網格結構下重連，其可能性只有一種，網格縮減的定義下(P=4<N=5 & Q=2<M=4)，因此成立。

假設 n=K 也有相同的結果

當 n=K+1

假設 A 爲中心內部節點，與其相連的節點有 K+1 個，我們的做法是要將這樣的網格處理至 n=K 的情況，因爲以假設 n=K 存在縮減網格，所以只要能將 A 點連接節點數 K+1 減至 K 即達成目的。

在與 A 相連之節點中我們先看是否可以找到 3 連續節點{N^j,N^j+1,N^j+2}，使得此四點形成凸四邊形,藉由定理 4 結果可之存在，將此凸四邊形情形作一個置換邊動作(A-N^j+1 ---> N^j-N^j+2)即可去除 N^j+1 點,如下圖.因此我們可以獲得由 n=k 所形成的網格結構至此用數學歸納法證明了此定理.

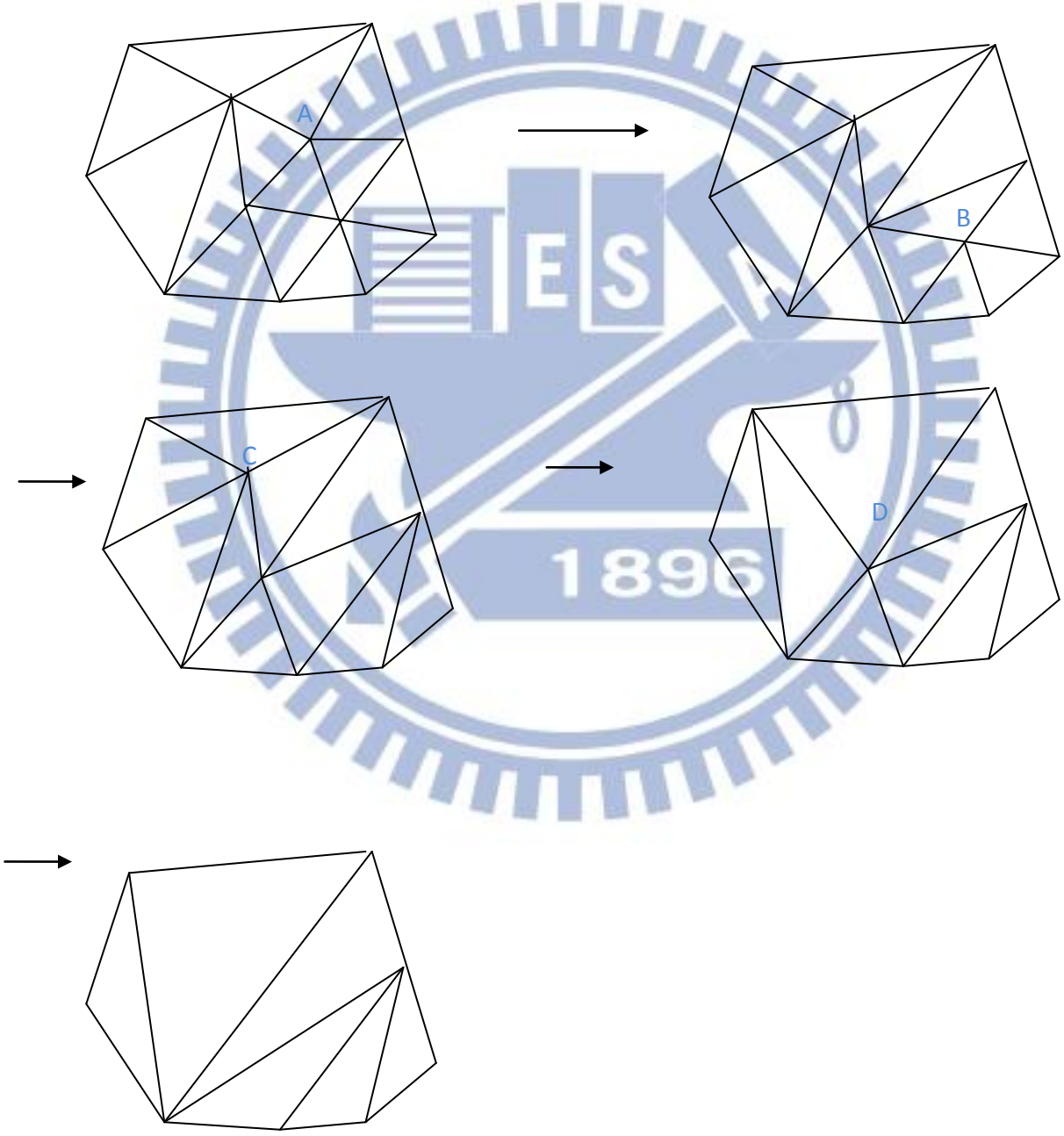


定理 3：二維三角網格 G 中經由任意指定內部範圍中的節點消除能達成我們想控制的縮減網格 H。

證明：

由定理 2 已知只要 G 中有內部區域(內部節點 A 爲中心形成)我們可以獲得縮減網格 H，且 H 中不存在 A 點，已 H 當作基礎我們可以選定內部節點進行消除來得到進一步縮減網格 H1，經此步驟我們可以將想刪除的內部節點一一消除最後獲得想要的縮減網格 F。

例如：下圖所有點皆爲內部節點，其邊界也在內部非邊界點



<u>定理 4</u>：二維三角網格中，以內部節點 A 爲中心所形成的內部區域，存在至少一個由 A 點與 3 連續節點的凸四邊形。

證明：

我們可以假設一以 A 爲中心的圓，在圓周上取數個點用直線相連即成爲已 A 爲中心之網格(圖 A)。網格上的凹四邊形可看爲初始圓向內凹缺所形成(圖 B)，因此我們可以知道，網格結構不可能全由凹缺弧所組成，兩凹缺弧的交點必爲凸四邊形(圖 C)。

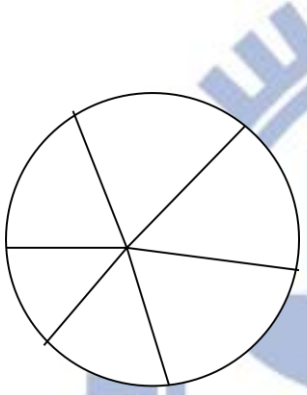


圖 A.

圖 B.

圖 C.

4.理論方法程式化

將上述定理以實際程式來完成，程式處理流程如下：

4.1 Triangle Mesh 結構

平面上 Triangle Mesh 是由一堆三角形組成的圖形，平面 Mesh 結構不一定是三角形的型態，不過就算是多邊形也可視爲幾個三角形組成，因此對於三邊以上的 Mesh 結構皆轉換爲三邊來處理。

基本的 Triangle Mesh 可用幾何學上 Euler's formula 得知點(node)、邊(edge)、面(elm)的關係：

點數目 — 邊數 ＋ 面個數(含邊界外也算一面) = **2**

⇨ 邊數(edge) = 點數目(node) ＋ 面個數(elm) — **1**(最外的面不算)

最基本的面必由三點組成(Figure 3)，不存在一或兩點形成的曲面(圖 3.)。

圖 3.

做爲程式處理需要，原始資料需給定節點(node)和面(elm)的組成方式，以下圖 4.爲例，共有 4 個節點(node)以及 2 個面(elm)。其組成方式可用矩陣儲存如下表 1.。

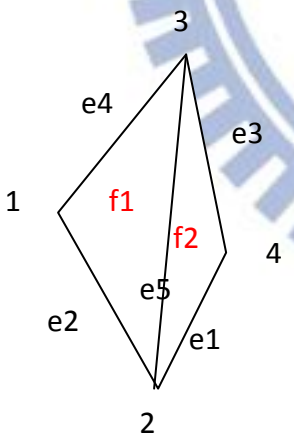


圖 4.

節點編號	X 座標	Y 座標
1	0	0
2	0.75	-2
3	1.5	2
4	2	-1

面編號	組成節點 1	組成節點 2	組成節點 3
f1	1	3	2
f2	3	2	4

表 1.

4.2 建立所需資料並調整 Mesh 結構

爲了保持 Mesh 結構，因此對於邊界上的節點和邊給予保留，縮減第一步只針對內部點。如何找出所有的邊界並建構輔助資料可由下方式進行，除了基本的節點(node)、面(elm)矩陣外，還需要三個輔助矩陣，分別是相鄰面關係(neib)、邊(edge)、連結點(connect)。

以上表 1.爲例，node 矩陣大小爲 4X2，elm 矩陣大小爲 2X3。以此爲基礎相鄰面關係(neib)中矩陣大小爲 elm 的擴充 2X(3+4)，當中記錄每一邊與哪一個面相鄰，如果沒有鄰面則以負值表示(表 2.)。這裡牽扯到排序的問題稍後再說。

邊(edge)矩陣大小則由 Euler's formula 簡單計算得知有幾邊，在此節點數 4 面數 2，邊數 = 4 + 2 -1 = 5。因此矩陣大小爲 5X11，當中紀錄此邊由哪兩點連成(n1,n2)、相對的頂點(n'1,n'2)、相鄰的面(F1,F2 此有排序的問題)、邊的長度(length)以及相對面的第幾邊(f1',f2')和連結點上的順序(c1,c2)。

連結點(connect)則以節點爲基礎擴充，大小可自行調整，基本矩陣大小爲 4X(3+k)，k 值爲節點連接數上限，以圖 6.爲例最大連結數可視爲 3，因此 k 值可設 3 以上，但是過多的預設只會浪費資源，因此這個情況可以設成 4，也就是 4X7。當中紀錄的有連接數(No.)、是否爲邊界點(bd)以及和哪些邊連接(Edge1~Edge4)。

在 neib、connect 中各留一行做爲重組時的準備。

neib				Edge1	Edge2	Edge3	
1	-1	f2	-2	e4	e5	e2	
2	f1	-3	-4	e5	e1	e3	

edge	n1	n2	F1	F2	length	n'1	n'2	f1'	f2'	c1	c2
e1	2	4	f2		1.6006	3		2		3	1
e2	2	1		f1	2.1359		3		3	2	2
e3	4	3	f2		3.0400	2		3		2	3
e4	1	3		f1	2.4998		2		1	1	1
e5	3	2	f2	f1	4.0689	4	1	1	2	2	1

connect		No.	bd.	Edge1	Edge2	Edge3	Edge4
1		2	1	e4	e2		
2		3	1	e5	e2	e1	
3		3	1	e4	e5	e3	
4		2	1	e1	e3		

表 2.

4.2-1 找出邊界

爲了不破壞結構，因此在縮減時邊界基本不動只選內部節點做處理。如何找出邊界並建立上述矩陣資料在此依據詹益政教授提出的想法爲基礎做擴充：先有基本的 node 和 elm 矩陣，找尋方式以 elm 爲主逐次搜尋節點編號，全部找尋過一次結束。以下圖 5.爲例：此圖總共有 10 個 elm(以紅色數字藍色底表示編號)以及 10 個 node(紅色數字黃底表示編號)，其中內部節點有 2，基本的資料則在右下表 3.顯示。

在此先定義網格組成排序，以第一個 elm 來說其資料爲(1,5,8)，因此對這個 elm 來說第一邊即爲節點 1 和 5 組成，第二邊爲 5 和 8 第三邊爲 8 和 1。依此由左至右的順序最後回到第一個節點完成一個三角形，依照順序爲順時針方向。

從第一個 elm 開始，固定第一邊(1,5)然後往下列 elm 尋找是否也有(1,5)或(5,1)找到第 10elm 其第一邊爲(1,5)因此這兩個 elm 爲相鄰面，也就表示節點(1,5)組成的邊不爲邊界，找尋過程中一旦找到符合的邊立即記錄資料並跳往下一個搜尋，因爲一個邊只可能由兩面相鄰而成，以 elm 1 的第三邊(8,1)來看，搜尋到最後的結果會沒有符合的邊，因此這一邊就是邊界了，從右圖 5.來看也是如此。

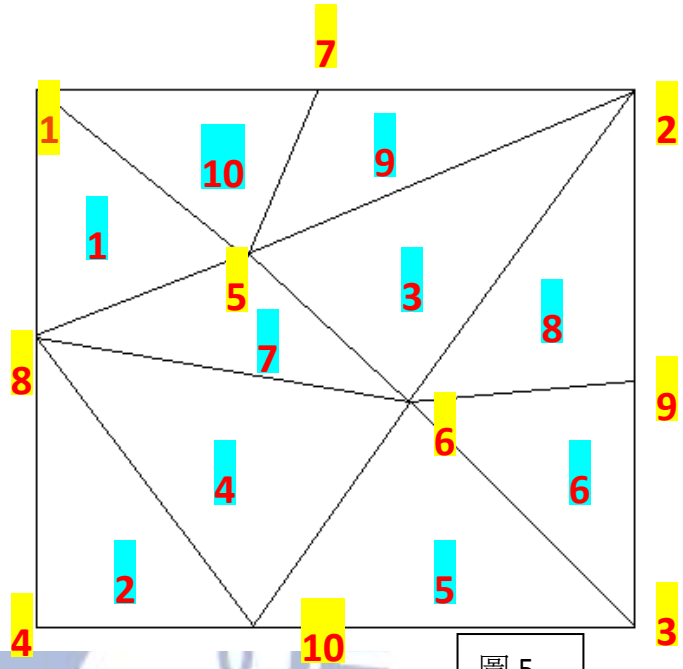


圖 5.

元素及其構成的節點(elm)				節點實際位置(x,y)(nod)		
1	1	5	8	1	0	1
2	4	8	10	2	1	1
3	2	5	6	3	1	0
4	8	6	10	4	0	0
5	3	6	10	5	0.4	0.7
6	6	9	3	6	0.6	0.4
7	5	6	8	7	0.5	1
8	9	2	6	8	0	0.5
9	7	5	2	9	1	0.45
10	1	5	7	10	0.36	0

表 3.

在記錄所需資料上以 neib 爲例，其矩陣大小爲 10X7，找尋 elm 1 第一邊(1,5)此時 neib 上直接先在〈1,4〉位置上填1表示這是第一個邊(此爲全部邊來看)，找尋結果在 elm 10 第一邊找到符合的結果，因此〈1,1〉位置填上 10 表示 elm 1 的第一邊相鄰的面是 elm 10，而〈10,1〉位置填上 1 表示相鄰面爲 elm 1，另外〈10,4〉位置也填上 1 表示 elm 10 第一邊在全部邊來說編號是 1。如右表 4.所示。

elm 1 第二邊也是如此方式，在找尋過程中一但 neib 上已有值則直接跳過，elm 1 第三邊爲邊界不會有與其相鄰的面，因此在〈1,3〉位置填上-1 表示沒有相鄰面也就是邊界(如左下表 5.)。至此 elm 1 搜尋完畢，接著 elm 2 依此類推直到全部找尋過完成表格。右下表 6. 爲完成表。

neib				Edge1	Edge2	Edge3	
1	10			1			
2							
3							
4							
5							
6							
7							
8							
9							
10	1			1			

表 4.

neib				Edge1	Edge2	Edge3	
1	10	-1	7	1	2		
2							
3							
4							
5							
6							
7			1			2	
8							
9							
10	1			1			

表 5.

neib				Edge1	Edge2	Edge3	
1	10	7	-1	1	2	3	
2	-2	4	-3	4	5	6	
3	9	7	8	7	8	9	
4	7	5	2	10	11	5	
5	6	4	-4	12	11	13	
6	8	-5	5	14	15	12	
7	3	4	1	8	10	2	
8	-6	3	6	16	9	14	
9	10	3	-7	17	7	18	
10	1	9	-8	1	17	19	

表 6.

4.2-2 調整 Mesh 結構

三角網格的組成根據上述的定義有順、逆時針兩種，而這種混編的網格對於本研究來說會造成某些問題，因此在建立資料時會簡單判斷一下使整個網格都爲順時針或逆時針排序，以下圖 6.爲例子，我們畫出三角形之外接圓，依照節點順序經圓取最短距離至下一點繞行一圈後即可知此排序爲順時針或逆時針。如何判斷一個三角形是順時針還是逆時針排序方式以下提出兩種方法：

方法一：座標轉換
方法二：向量外積

判別三角形組成爲順時針或逆時針如右圖 8.
逆時針 ：f(a,b,c) or f(b,c,a) or f(c,a,b)
順時針 ：f(a,c,b) or f(b,a,c) or f(c,a,b)

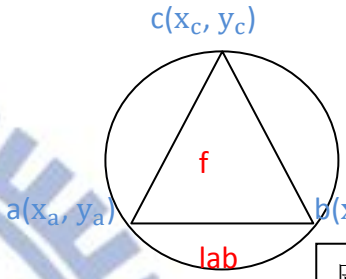


圖 6.

方法一

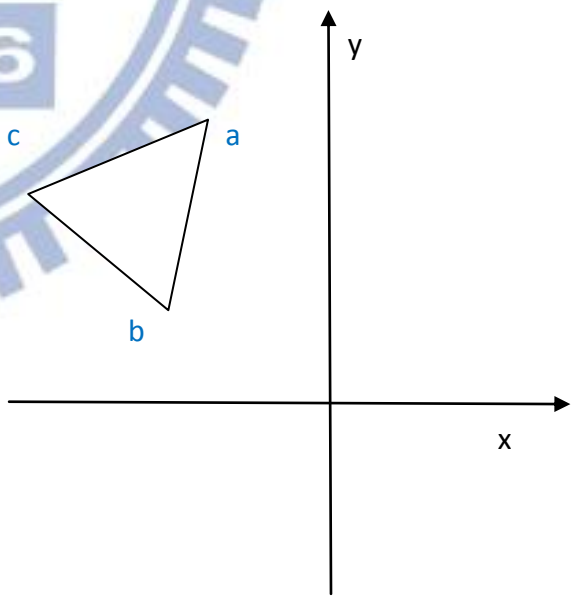
將三點位移，其中 a 作爲原點(0,0)，也就是說三點(x,y)座標值扣除 a 點(x,y)值。計算 ab 距離爲 lab，將(lab,0)作爲 b 旋轉後位置，算出旋轉角度，將位移後 c 點帶入算出新 c 點之 y 值，若 y>0 則爲逆時針，y<0 爲順時針。

a'(0,0)

b'$(x_b - x_a, y_b - y_a)$

c'$(x_c - x_a, y_c - y_a)$

lab=$\sqrt{(x_b - x_a)^2 + (y_b - y_a)^2} = \sqrt{(x_{b'})^2 + (y_{b'})^2}$



$$[x_{b'}, y_{b'}]\begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix} = [lab\ 0]$$

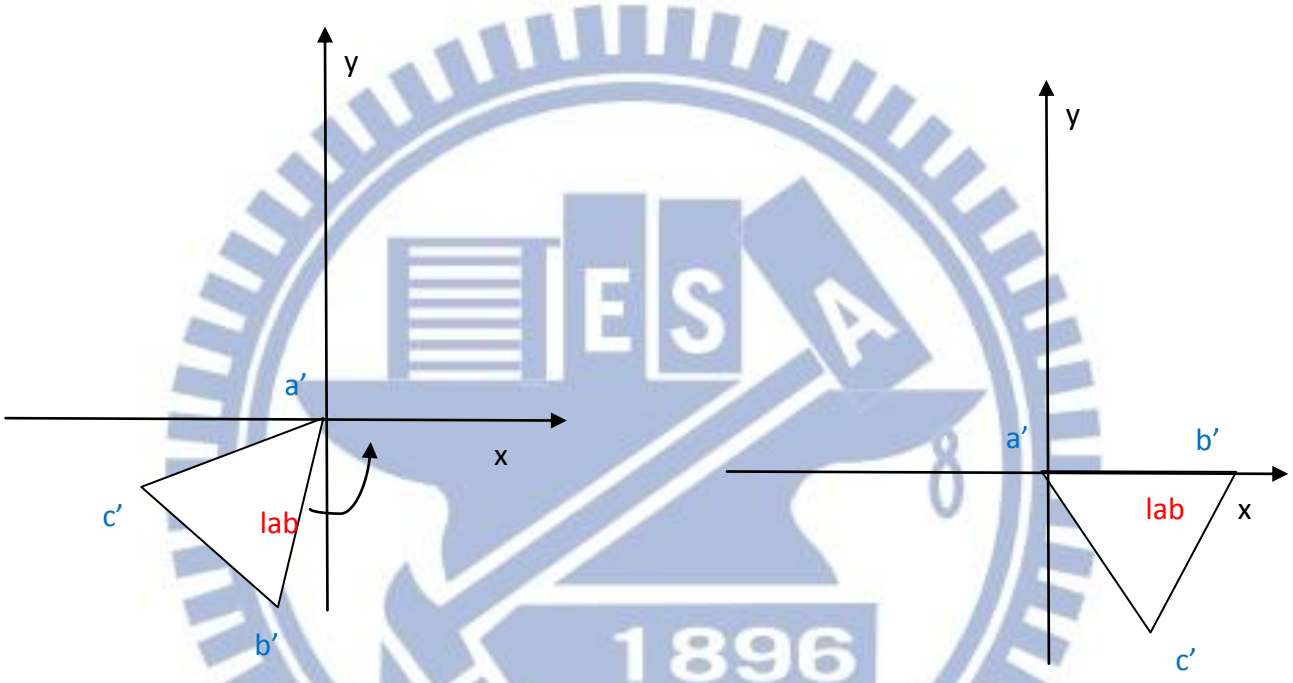
$$\begin{cases} x_{b'}\cos\theta + y_{b'}\sin\theta = lab \\ -x_{b'}\sin\theta + y_{b'}\cos\theta = 0 \end{cases}$$

$$\begin{cases} \sin\theta = \dfrac{y_{b'}}{lab} \\ \cos\theta = \dfrac{x_{b'}}{lab} \end{cases}$$

$$[x_{c'}\ y_{c'}]\begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix} = \frac{1}{lab}[x_{c'}\ y_{c'}]\begin{bmatrix}x_{b'} & -y_{b'} \\ y_{b'} & x_{b'}\end{bmatrix}$$

$$= \frac{1}{lab}\begin{pmatrix} x_{c'}\,x_{b'} + y_{c'}\,y_{b'} \\ -x_{c'}\,y_{b'} + y_{c'}\,x_{b'}\end{pmatrix}^{t}$$

$$\frac{-x_{c'}\,y_{b'} + y_{c'}\,x_{b'}}{lab} < 0 \implies \text{順時針排序}$$



方法二

我們依序將第 1 點與第 2 點當作第一向量，第 2 點與第 3 點當作第二向量，利用這兩向量做外積得到一正或負值，藉此判斷此爲順或逆時針排序。其計算結果與方法一相似不過原理上並不相同。

假設 f(a,b,c)，則向量 1 爲 ab 組成，向量 2 由 bc 組成：

V1 = b-a = $(x_b - x_a , y_b - y_a)$

V2 = c-b = $(x_c - x_b , y_c - y_b)$

(V1 X V2)=(0,0,k) ,k≠0

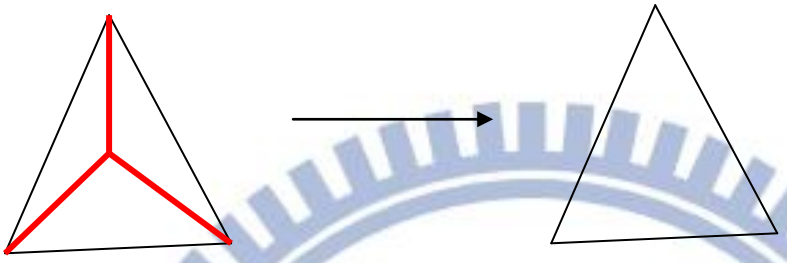
k = $(x_b - x_a)(y_c - y_b)-(y_b - y_a)(x_c - x_b)$

以此例子來說 k>0 所以爲逆時針排序

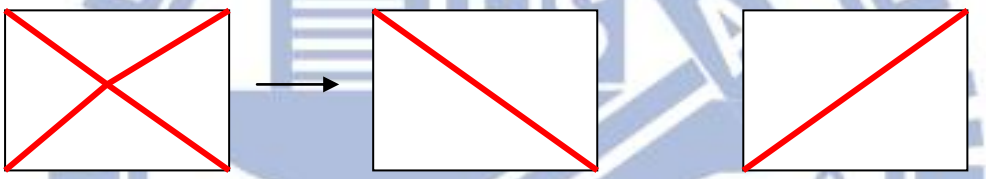
4.3 縮減 Mesh & Smoothing

本研究中簡化 Mesh 的方法是以一個內部節點爲基礎，藉由互換四邊形對角線的方式來消去此節點，具體做法由連接三點所形成的三角區域開始處理，然後是連接四點的四邊形區域，再來是五邊形依此類推至 N 邊形。

1. 內部節點只有連接 3 點(三角形)，直接縮成三角形，不論外邊點是否爲邊界

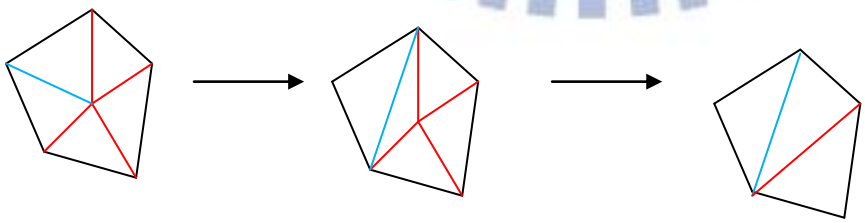


2. 內部節點連接 4 點(4 邊形)
   a. 凸 4 邊形，任意兩對頂點相連



   b. 凹 4 邊形，只能類似下圖連接法：

3.內部節點爲 5(以上)邊形，置換一邊(N 邊)後爲四邊形，然後以四邊形處理，方式依據置換邊的原則



4.3-1 四邊形對角線置換原則

以下圖 7.爲例，置換對角線時相鄰的兩個 elm 以順時針移動爲準。任意邊假設由 a 至 b，則左面爲 f1 右面爲 f2，置換後以 f1 來說 b 點爲其頂點，f2 來說 a 點爲其頂點。不論此邊是由 a 至 b 或者 b 至 a 皆用此原則置換，意即若由 b 至 a 則此時左面爲圖中的 f2 右面爲圖中的 f1，置換後 f1 頂點仍然是 b，f2 頂點仍然爲 a。另外如果 f1 的組成排序爲(c,a,b)、f2 爲(b,a,d)(此爲逆時針排序)，置換後仍然必須保持逆時針排序，也就是說置換後 f1 可以爲(b,c,d)、f2 可爲(c,a,d)。

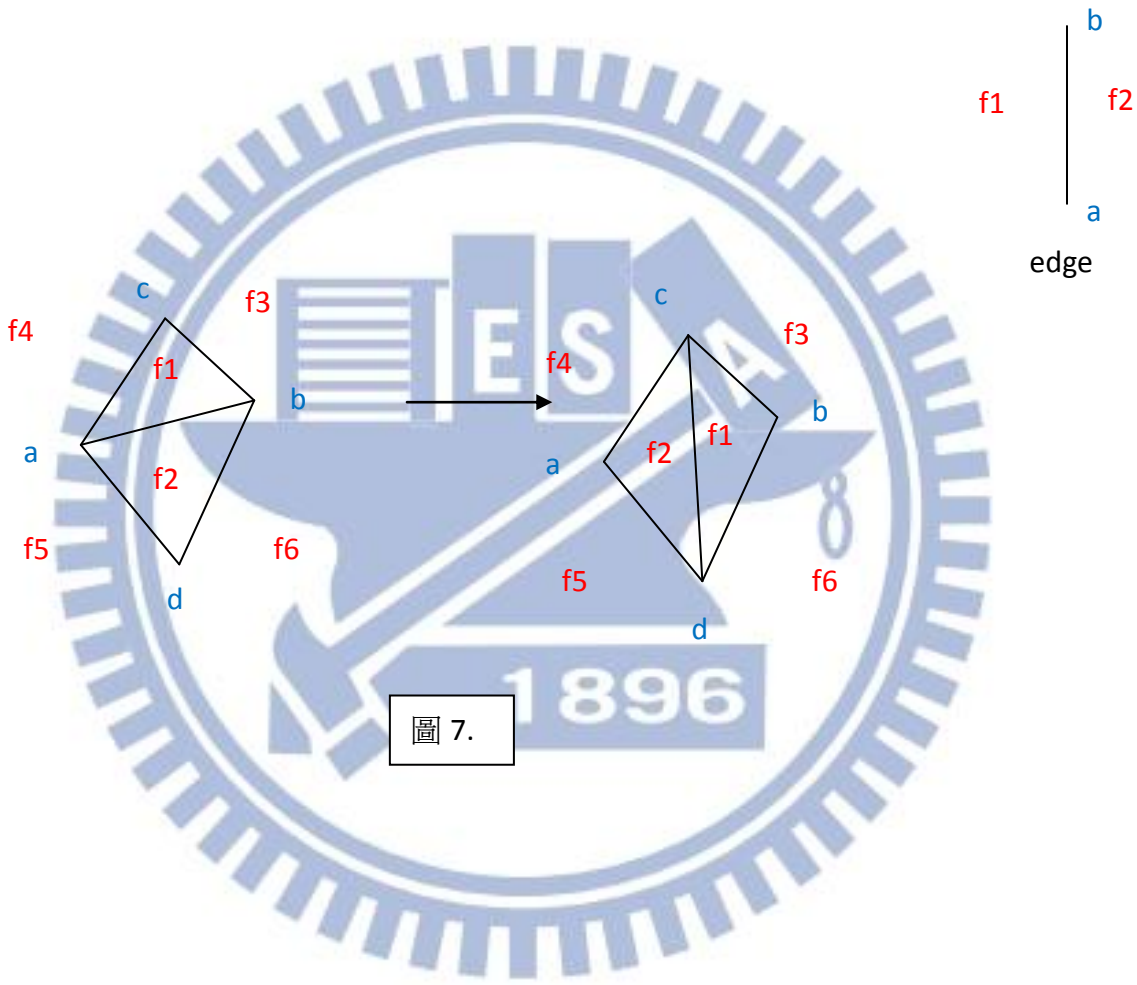


圖 7.

4.3-2 判斷凹或凸四邊形法

對於置換四邊形對角線上存在一個問題，只有凸四邊形才適合置換，凹的四邊形置換後會破壞三角網格結構，如下圖 8.所示：

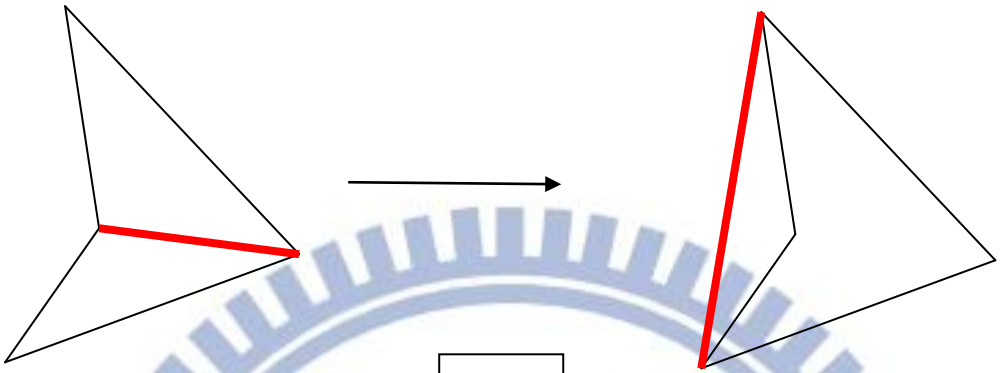


圖 8.

凸四邊形則沒有這種問題，如下圖 9.所示：

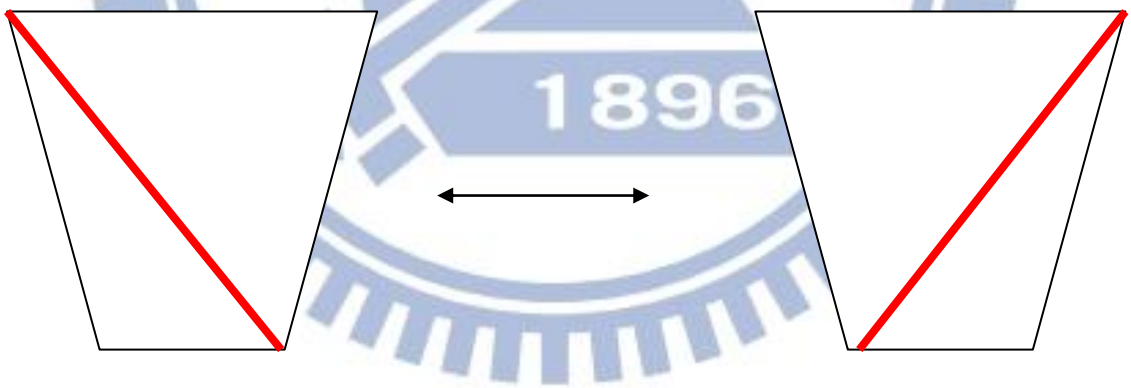


圖 9.

如何判斷是否爲凸四邊形則可以用平面直角坐標系上直線方程式，以下圖10.爲例將對角線之直線方程式 f(x,y)作爲標準，即以 A、B 兩點連成的直線方程式，然後帶入 B、D 兩點求值即 f(B)、f(D)，若 B、D 點在直線上則值 f(B)、f(D)=0，不在線上則會大於 0 或小於 0。以 Figure 10 來看如果 f(B)> 0 則 f(D)< 0 反之亦然。而圖 11.的情況則 f(B)、f(D)皆大於 0 或皆小於 0，因此只要 f(B)、f(D)兩值互爲異號則四邊形必爲凸四邊形，兩值同號則爲凹四邊形。

$y = ax+b$, $a= \frac{y_a-y_c}{x_a-x_c}$, $b = y_a - ax_a = y_c - ax_c$

⇨ $f(x,y) = y - ax - b$

⇨ $y = \frac{y_a-y_c}{x_a-x_c}x + y_a - ax_a = \frac{y_a-y_c}{x_a-x_c}(x - x_a) + y_a$

⇨ $f(x,y) = (y_a - y_c)(x - x_a) + (x_a - x_c)(y_a - y)$

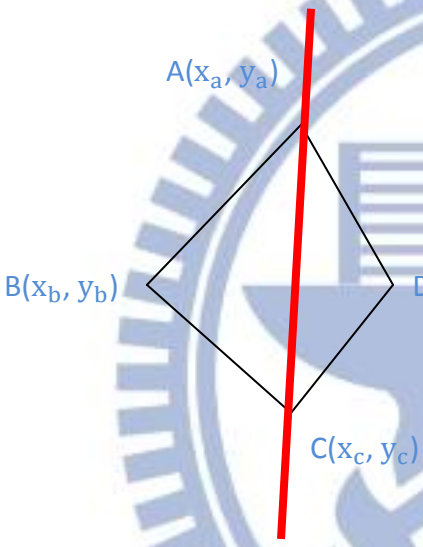


圖 10.

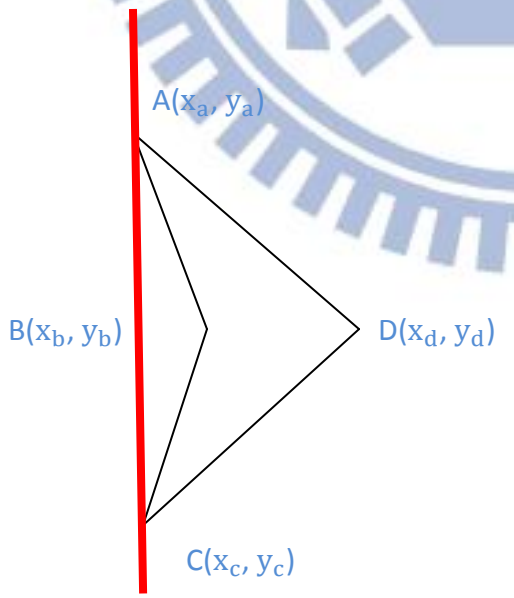


圖 11.

4.3-3 Smooth

經過縮減後的 MESH 在結構上會有些微變化，些許變化在整體結構上雖然影響不大，不過多次的縮減後則可能出現過於尖銳的三角網格如下圖 12.，過於尖銳的三角網格不僅不利於計算在網格縮減過程也會造成阻礙，因此有必要藉由平滑的技術來處理種問題。另外初始網格結構平滑程度也許不足，因此獲得初始網格時也可先 smooth 後再做接下來的處理。

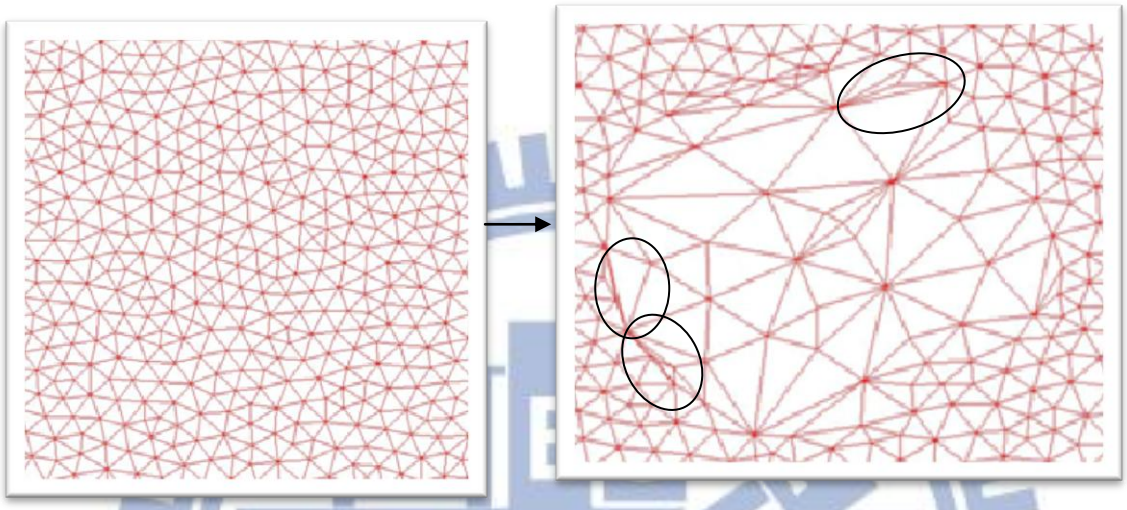


圖 12.

本文採用的平滑技術爲質心位移法，藉由每次將中心點向質心逐漸移動後達成平滑目的圖 13.。在此對於邊界點以及某些固定點則不做此動作，以避免整體網格結構破壞。

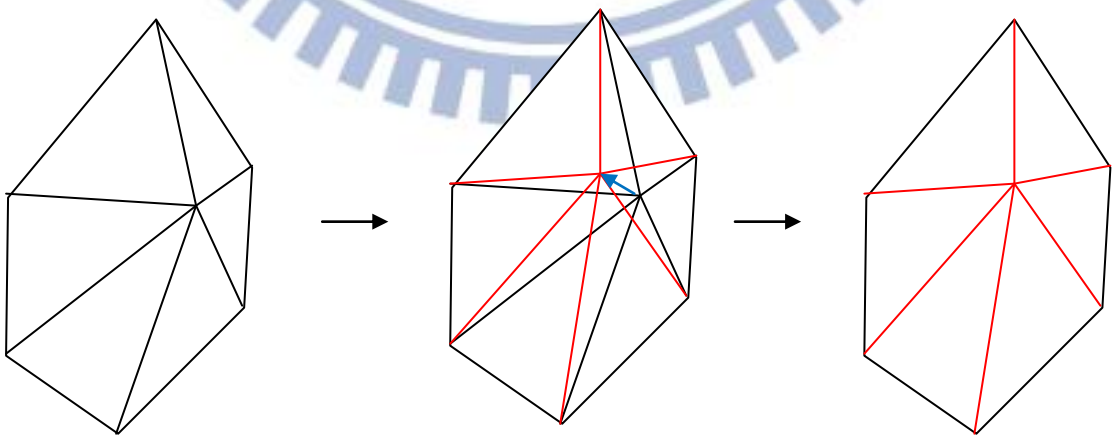


圖 13.

4.3-4 對角線置換法

除了使用 smooth 處理過於尖銳的網格外，對角線置換也可適用於這部分，如下圖 14.，左圖兩紅色圈部分爲尖銳網格，將其最長邊做置換後得到右圖，在此是否可置換皆依據之前提過之判斷法，兩圖比較下明顯右圖較符合所需，經過置換後之網格仍配合 smooth 一並處理，以便得到較平滑網格。

需要轉置的邊長條件設定爲三角網格中，最大邊長大於三邊平均長度 1.5 倍或最大邊長爲最小邊長 2 倍以上，符合此條件即進行處理，所有面都檢查過後再進行 smooth，以效率來看此方法比直接 smooth 造成的效果還要顯著。

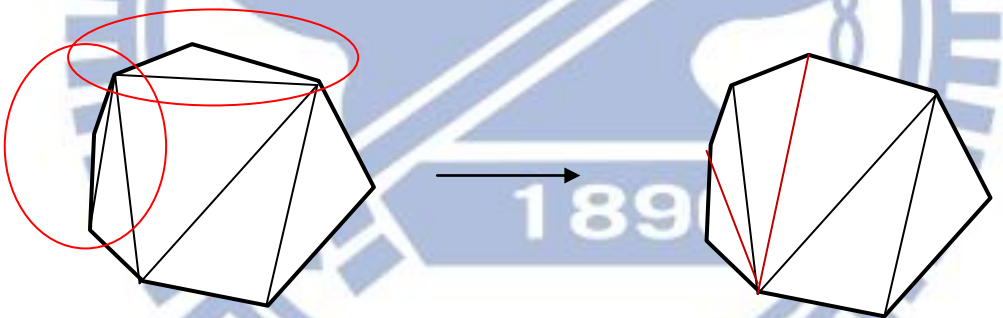


圖 14.

4.4 邊界點的縮減

在整體縮減過程中並沒有刪除或更改邊界點，經過縮減後的網格某些邊界點則變得不重要可以進行縮減，在此提出可縮減的邊界點判斷法，如下圖 15.只有邊界點的連接數爲 3 時才適合刪除，a、b、c 皆爲邊界上的節點，其中 a 爲主要目標點，與其相連的點有(b,c,d)三點，因此刪除 a 點(刪去 ad 邊)後整體來看仍爲三角形不會破壞網格結構。

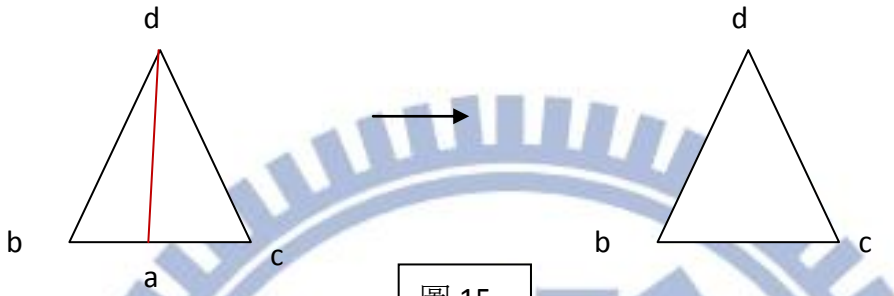


圖 15.

如果 a 點連接數超過 3 點如下圖 16.，則不處理。

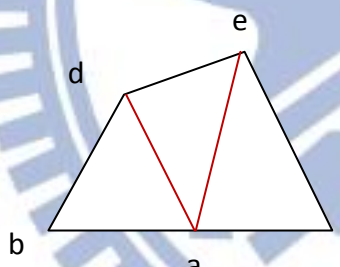


圖 16.

4.4-1 判斷基準點

邊界點中只有當(b,a,c)3 點爲同一直線時才適合刪減，當 3 點不共線時 a 點則視爲基準點(此以圖 17.爲例)，而基準點不論何時我們都不動它。如下圖 17.，(a,b,c,d,e)爲邊界上之 5 點，其中(b,d,c)、(c,a,e)分別共線，而 c 點即爲基準點。在此我們可將有轉折的點(不共線)視爲基準點。

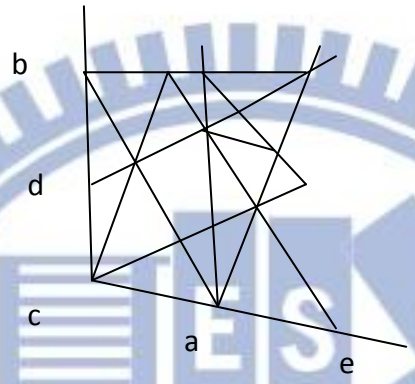
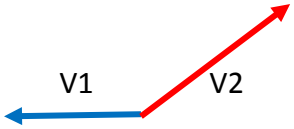


圖 17.

如何判斷基準點除了一開始給定之外可利用向量外積(corss product)的方式來判斷，若兩向量平行則外積值爲 0，即爲 3 點共線圖 18.，當外積值不爲 0 時則有基準點圖 19.。

(V1 X V2)=(0,0,0)

圖 18.



(V1 X V2)=(0,0,k) ,k≠0

圖 19.

4.5 重組 Mesh

縮減過程中，儲存節點和面資料的矩陣在編號上並無更動，若使用未變更的編號來產生縮減後的網格將會造成錯亂，針對這個問題有多種解決方法，在此採用後置編排法。因爲在縮減過程中並未更改編號，因此可以將所剩的節點也就是新結點另外填上新的編號後，再將留下的 elm 內的編號作替換，替換過程中同時造新結點矩陣以及新面矩陣，當替換完成時縮減後新的網格也完成。

實際做法如下：下表 7.爲初始節點矩陣，在建立所需相關資料時，neib、connect 矩陣分別留下一行作爲刪去之標記，因此縮減過程中如有節點或面被刪除，就在相對應的 connect、neib 矩陣上填入 -1 表示已被刪除。由 connect 矩陣開始，從數字 1 開始只要矩陣內之值不是 -1 就依序填上號碼，直到填完爲止，而最後填入之數字即爲新節點個數，進行過程中新節點矩陣也隨之產生。

原始編號	新編號
1	-1
2	-1
3	0
4	0
5	-1
6	-1
7	-1
8	0
9	0
10	-1
11	-1
12	0
13	-1
14	-1
15	-1

→

原始編號	新編號
1	-1
2	-1
3	1
4	2
5	-1
6	-1
7	-1
8	3
9	4
10	-1
11	-1
12	5
13	-1
14	-1
15	-1

表 7.

完成上述動作後，即可建立新面矩陣。作法與重牌節點相同，在此取 neib 矩陣一樣從 1 依序填入不是 -1 的矩陣內，而新面矩陣也隨之產生，不過原先組成面的三個號碼必須藉由新節點編號作替換，替換方式只需將原本的號碼換成後來給的新編號即可。

左下爲擷取一部分新節點之表格(表 8.)，右下(表 9.)爲初始面之矩陣，紅色部分表示被刪除面，黑色部分表示仍存在之面，需要替換的就是黑色的部分，根據節點對應編號替換，如原始的 3 替換爲 1，20 替換爲 7 如下表 8.所示。完成表爲表 10.。

新編號	原始編號由小至大
1	3
2	4
3	8
4	9
5	12
6	16
7	20
8	21
9	25
10	26
11	29
12	31

表 8.

1	3	4	20
2	3	2	5
3	4	20	8
4	21	20	16
5	31	3	25
6	4	5	8
7	9	2	11
8	10	23	16
9	18	19	20
10	20	16	26
11	29	21	16
12	10	21	18

表 9.

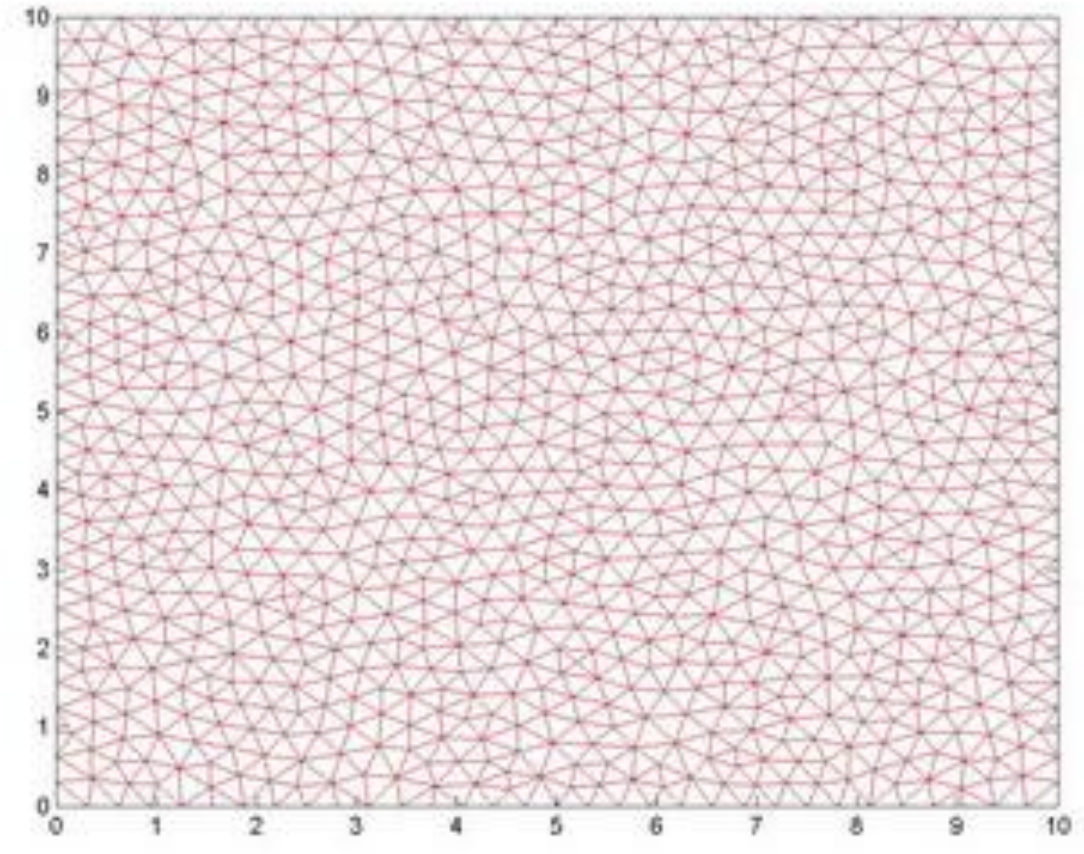
1	1	2	7
2	2	7	3
3	8	7	6
4	12	1	9
5	7	6	10
6	11	8	6
7			
8			
9			
10			

表 10.

5. 實際結果

最後我們對研究方法做測試，第一部分先對縮減後的結果做檢視，查看是否仍爲正常的 Mesh 結構，整體上 smoothing 結果是否理想。第二部分將未縮減前原始 Mesh 與縮減之後的新 Mesh 在數值計算上做比較。

以下圖 20.爲原始 Mesh 開始測試，此圖有 1101 個節點和 2072 個面，且所有面皆爲三角形組成，整體看來爲正規的三角網格。

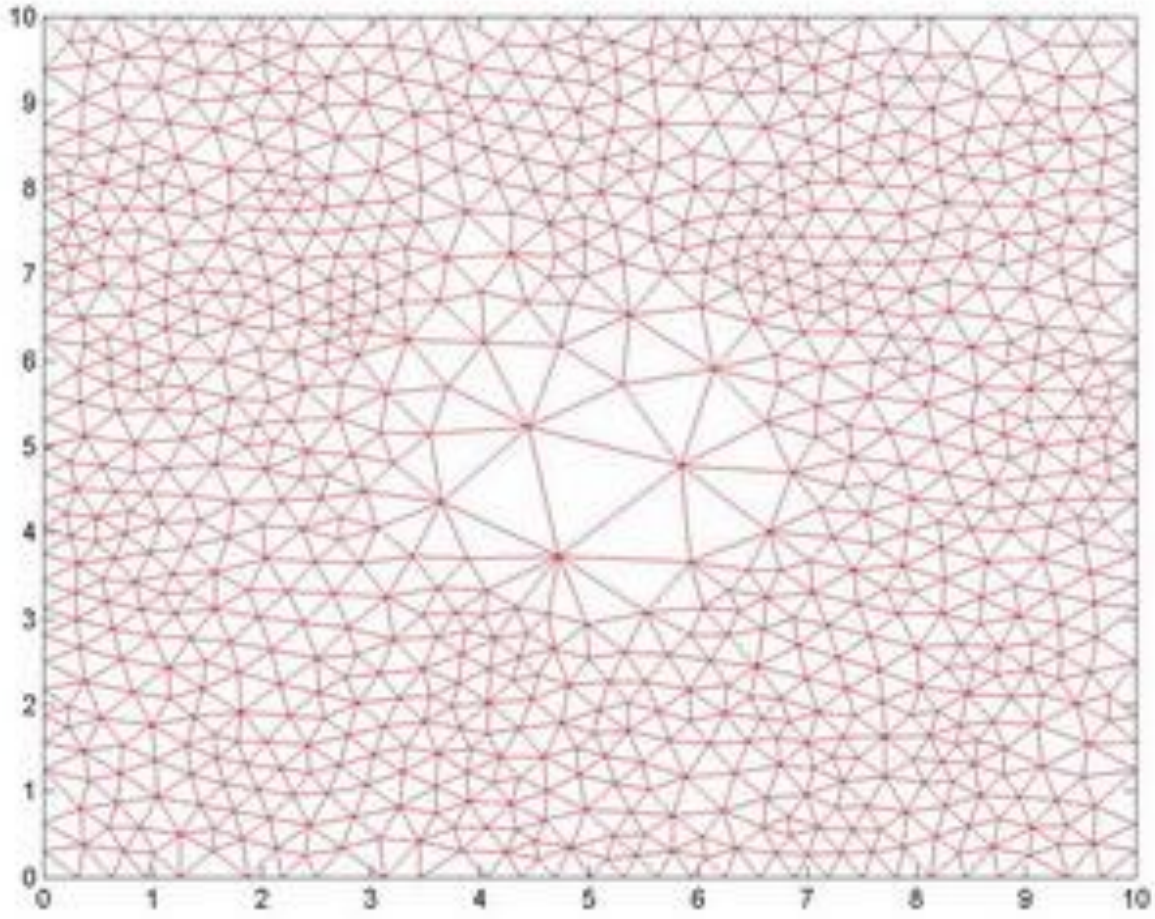


(1101,2072)

圖 20.

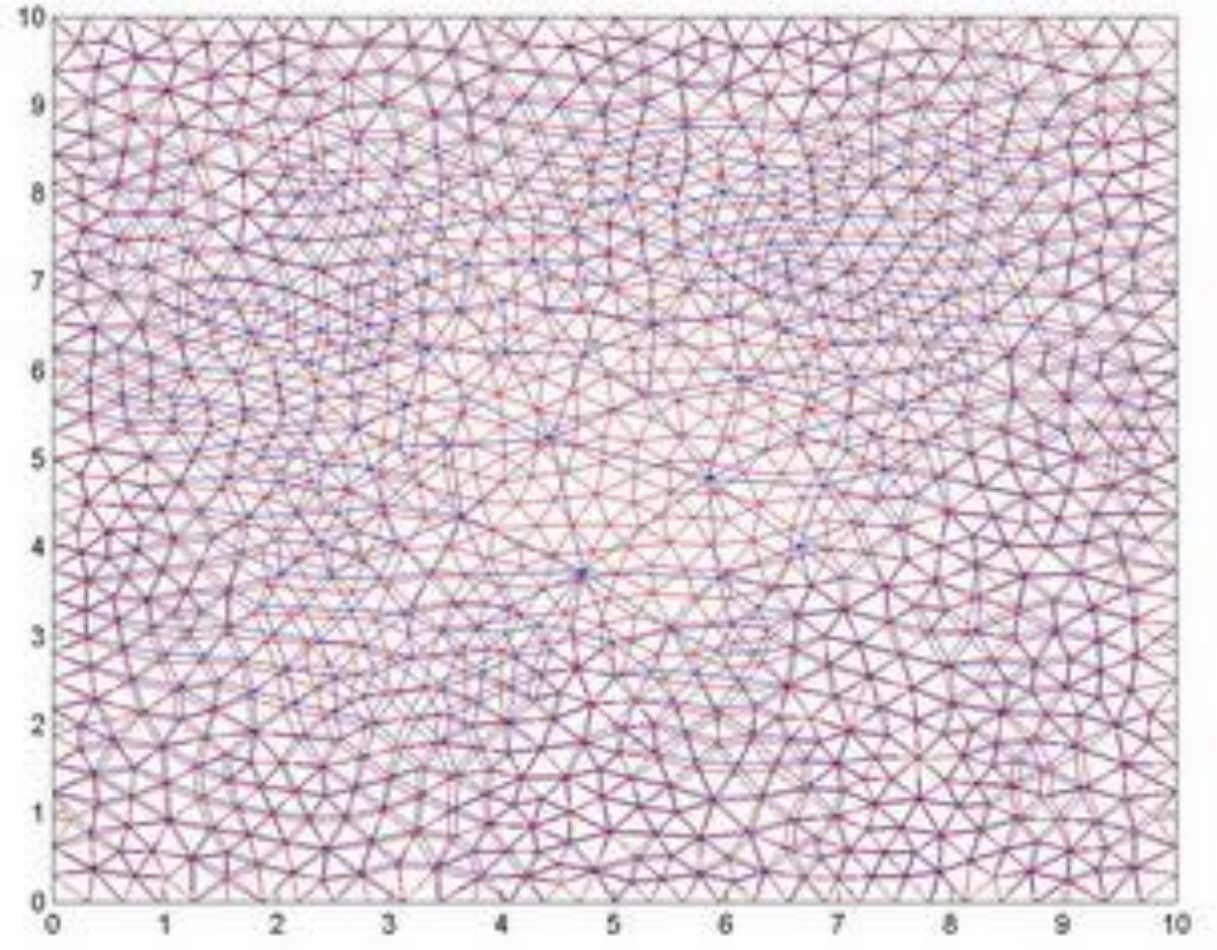
對於上圖 20.縮減的方式，因爲可以針對需要縮減的地區進行縮減，所以在此先對中心點(5,5)周圍區域進行縮減，範圍暫定爲 2，也就是(5,5,2)這個圓的部分進行測試。下圖 21.爲縮減後結果，總體上並無錯誤，平滑程度也可以接受，此結果節點數爲 900 而面個數爲 1694。

(5,5,2)=(x 座標 ,y 座標 ,圓半徑)



(900,1694)　　圖 21.

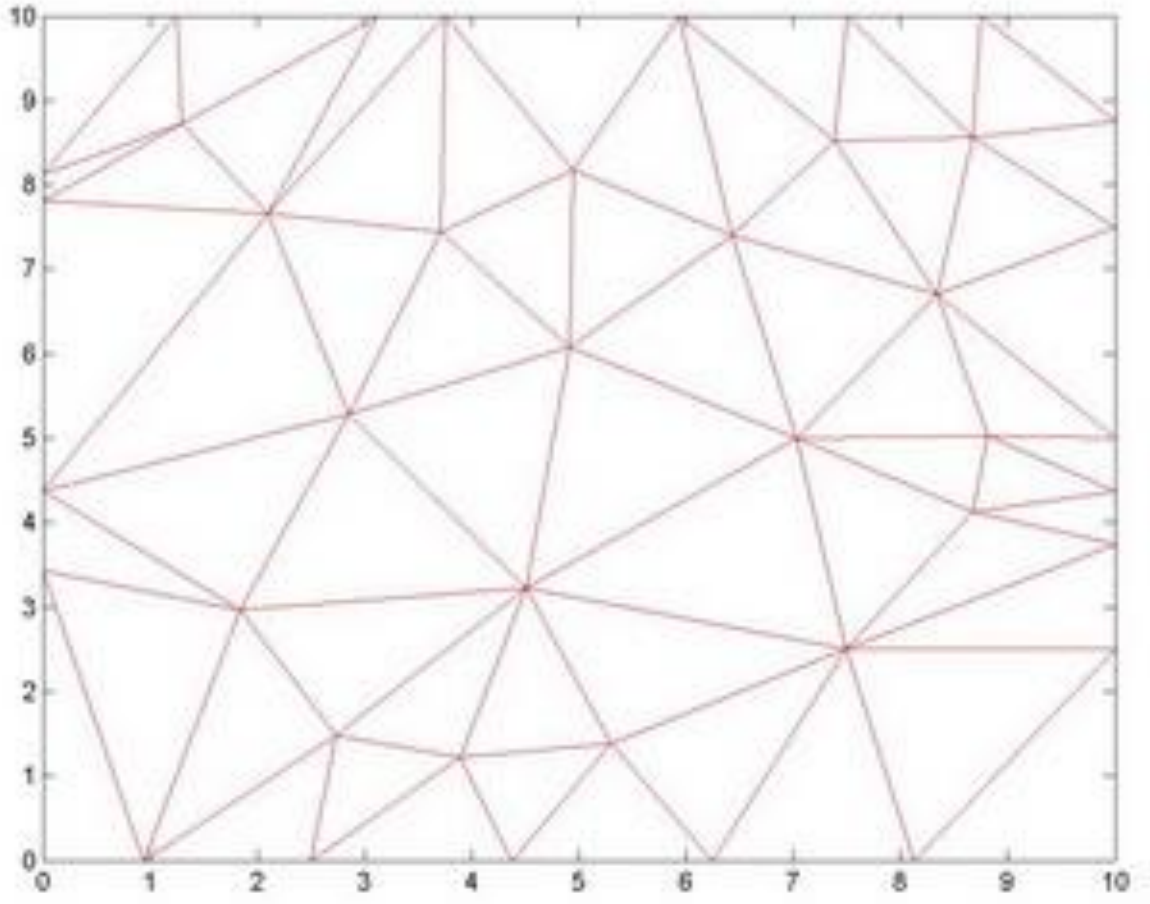
下圖 22.將原始網格和縮減後網格重疊做比較，紅色線爲原始網格藍色線爲縮減後網格。大致上可以看出已(5,5)爲中心周圍少了很多節點



Red(1101,2072) & Blue(900,1694)

圖 22.

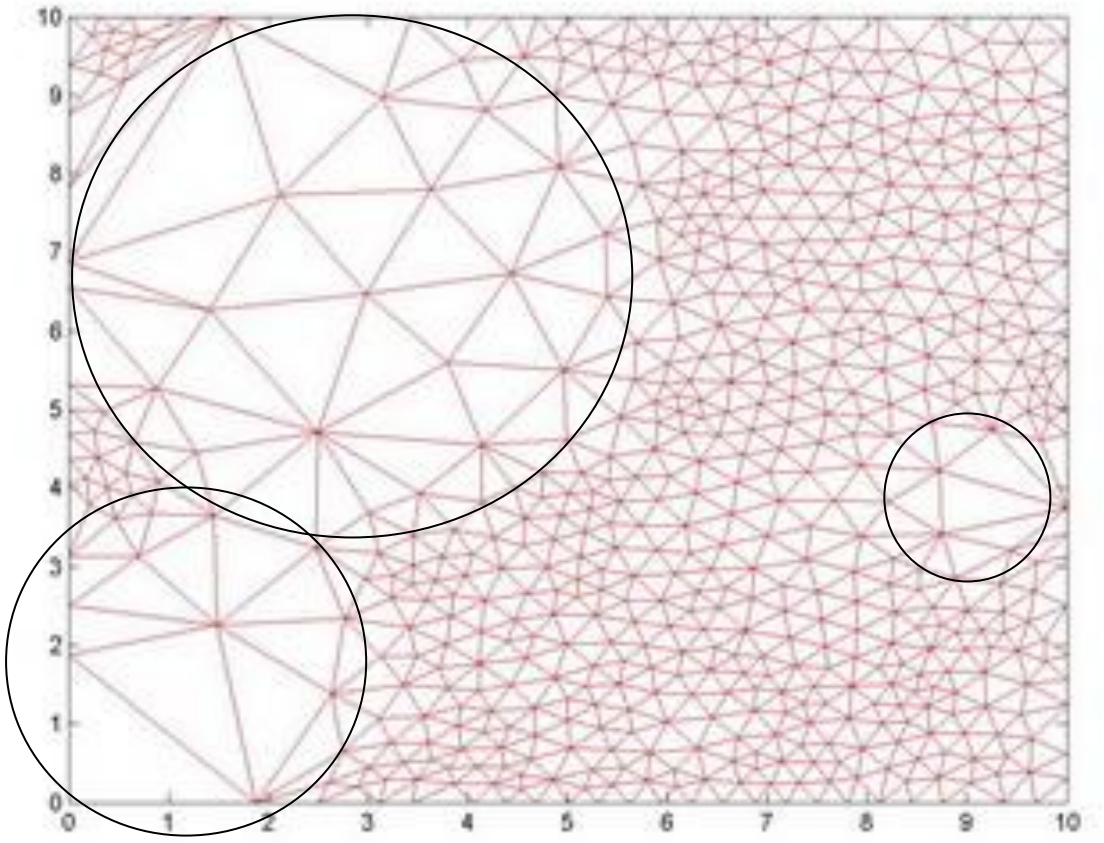
接下來對整個區域做縮減，預設剩餘節點數約 10%。下圖 23.爲處理後結果，此網格結構僅由 44 節點以及 61 個面組成，對於邊界點的處理上除了可拿掉的節點之外皆不改變其位置，因此會有部分邊界點很靠近的情況，基於邊界點不動的原則上對此不另作處理。



(44,61)

圖 23.

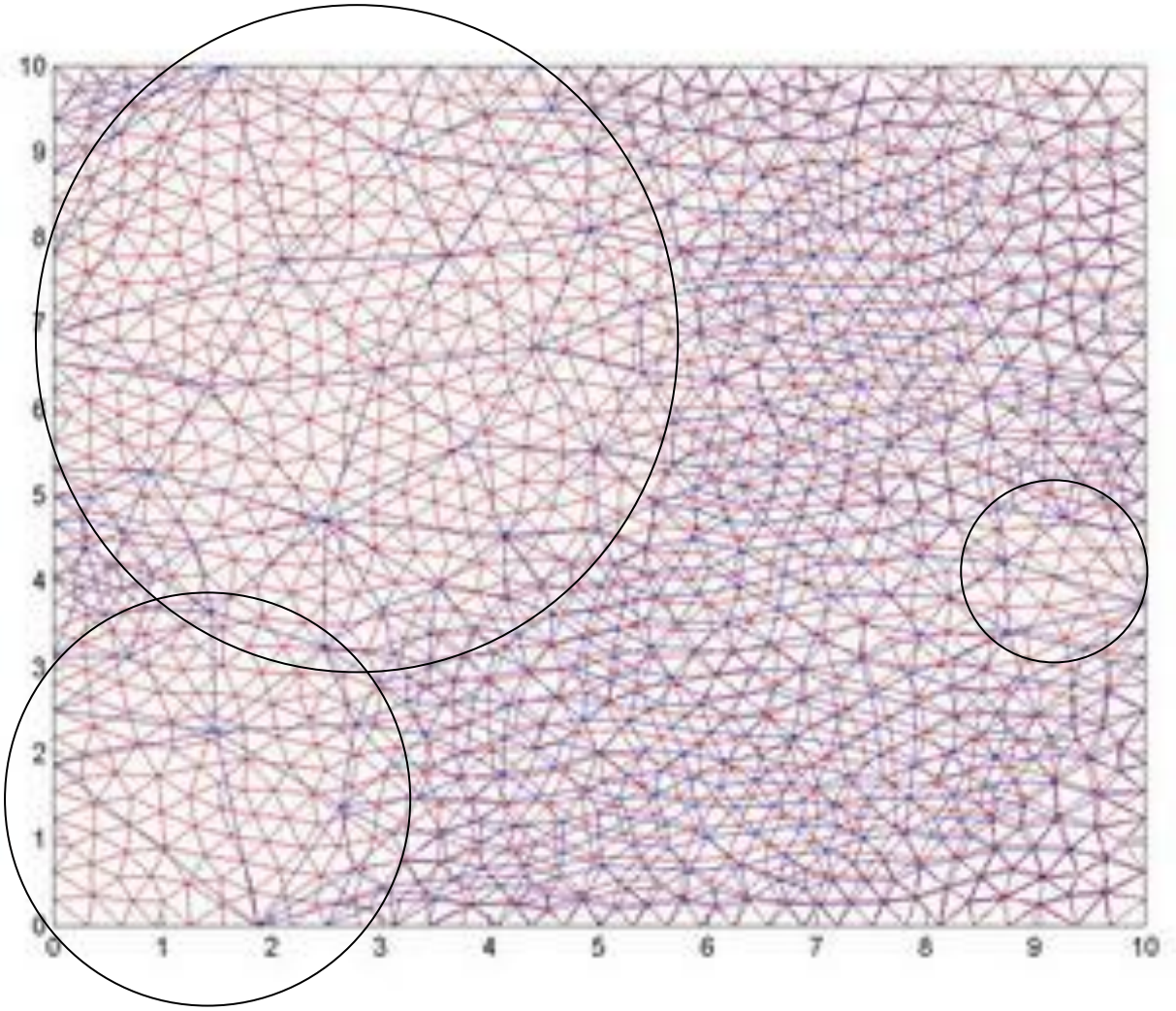
下圖 24.多增加幾個縮減區域測試，縮減區設定爲三，分別爲(1,2,2)、(3,7,3)、(9,4,1)，縮減範圍以黑圈標示，節點數 609 面個數 1137。就結果來說沒有網格結構上的錯誤，整體平滑程度也可以接受。



(609,1137)

圖 24.

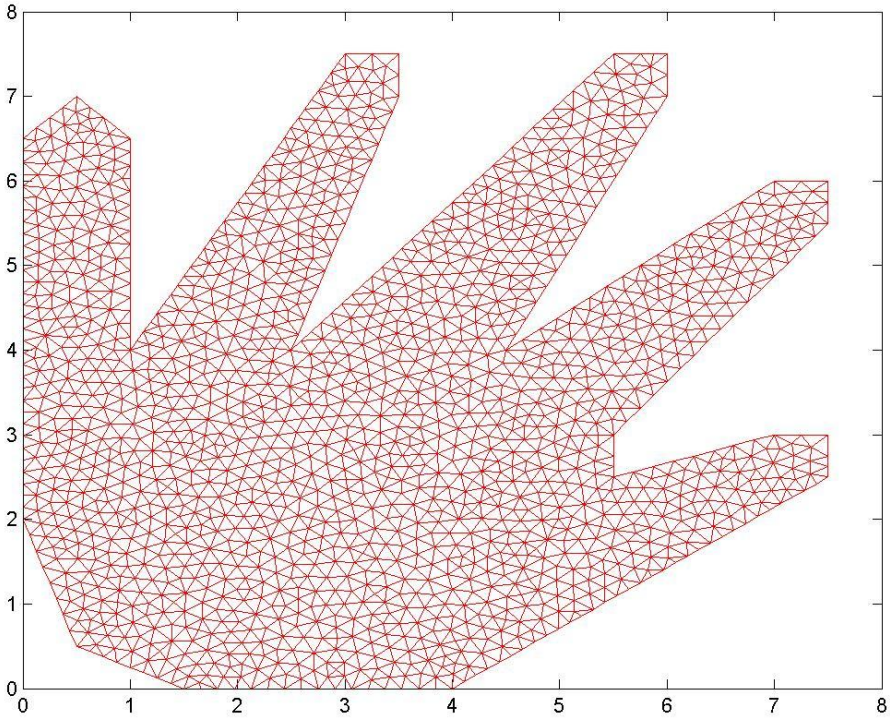
下圖 25.爲原始網格圖 20.和圖 24.重疊圖做比較，紅色線仍爲原始網格藍色線則是縮減後網格。



Red(1101,2072) & Blue(609,1137)

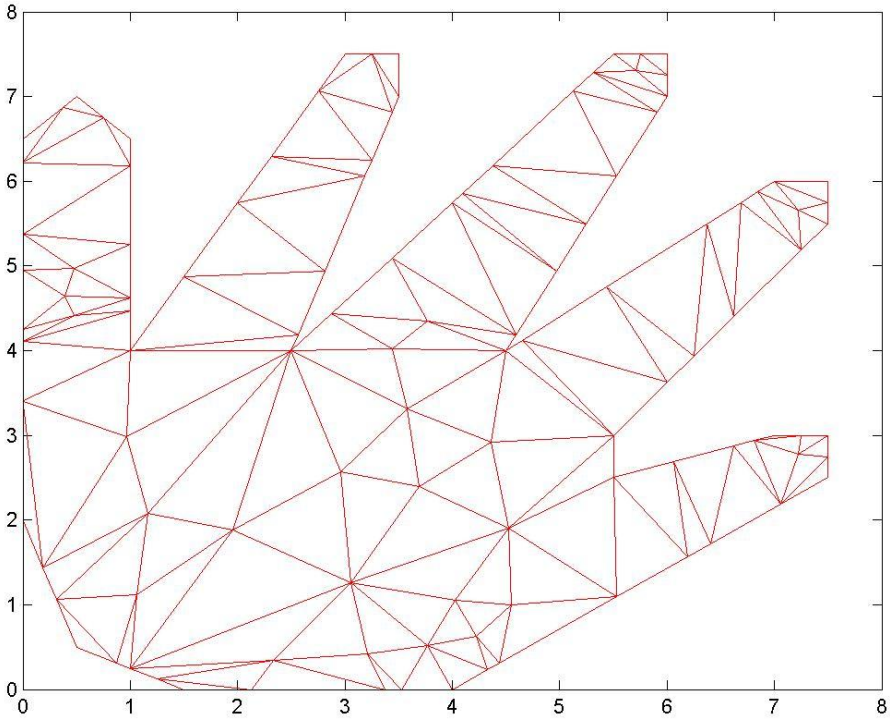
圖 25.

接下來我們對不規則的網格結構做測試，下圖 26.類似手掌的網格



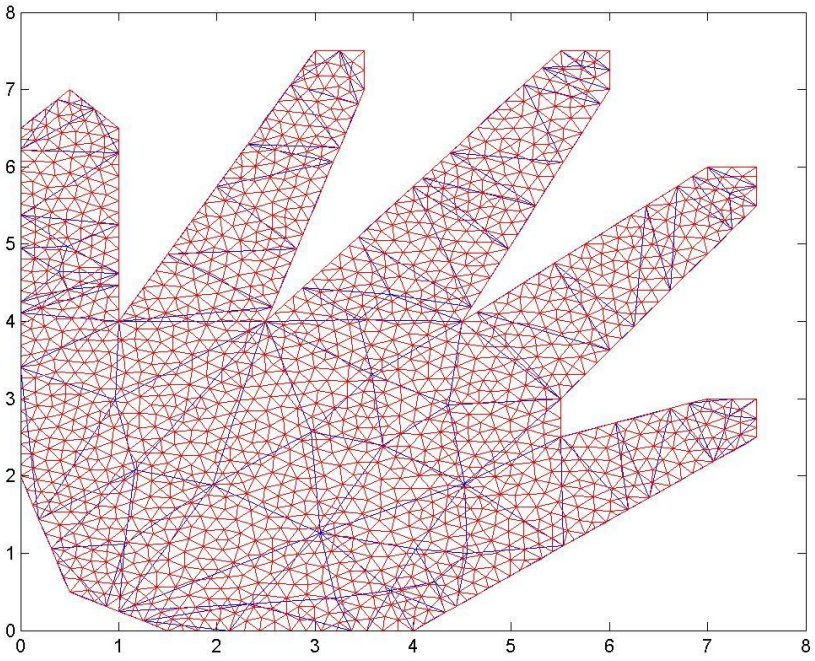
(1663,3046)　圖 26.

我們對其進行全域縮減得到以下的結果(圖 27.)：



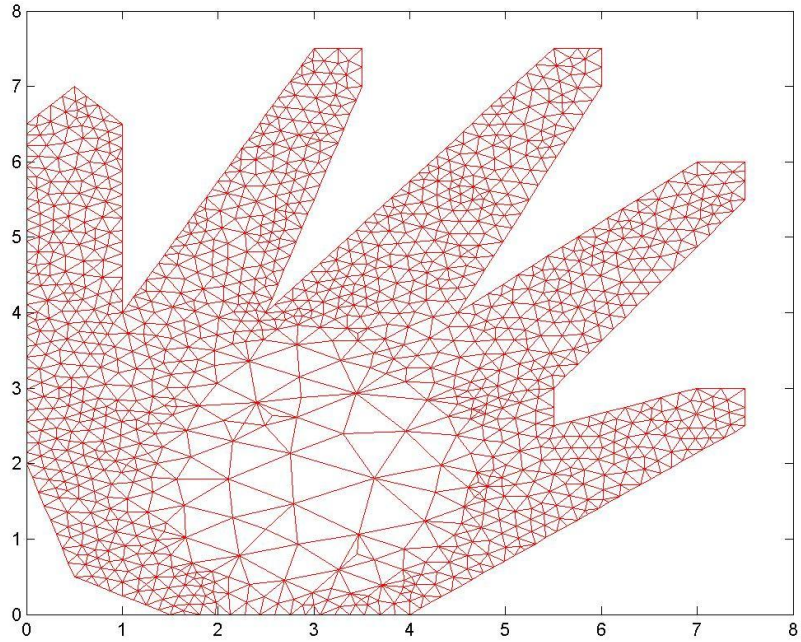
(113,135)　圖 27.

將上頁兩圖重疊做比較(圖 28.)：



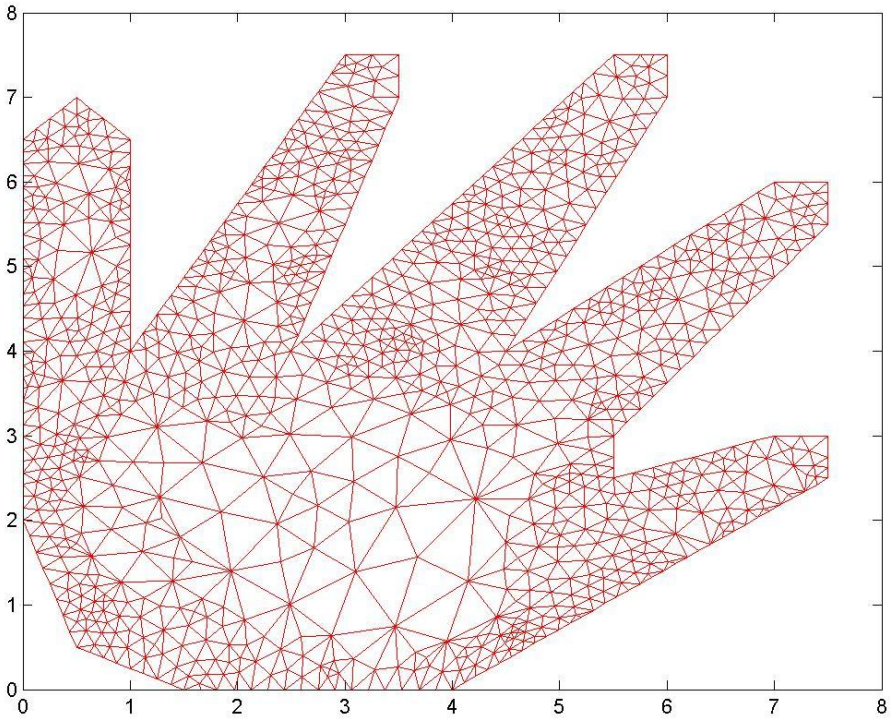
red(1663,3046), blue(113,135)　圖 28.

基於科學計算上的觀點來說，我們並不希望在邊界也進行消去，因此下圖 29.將保留所有的邊界點，在五根手指的部分因爲過於細長則不處理僅對掌心部分進行縮減：

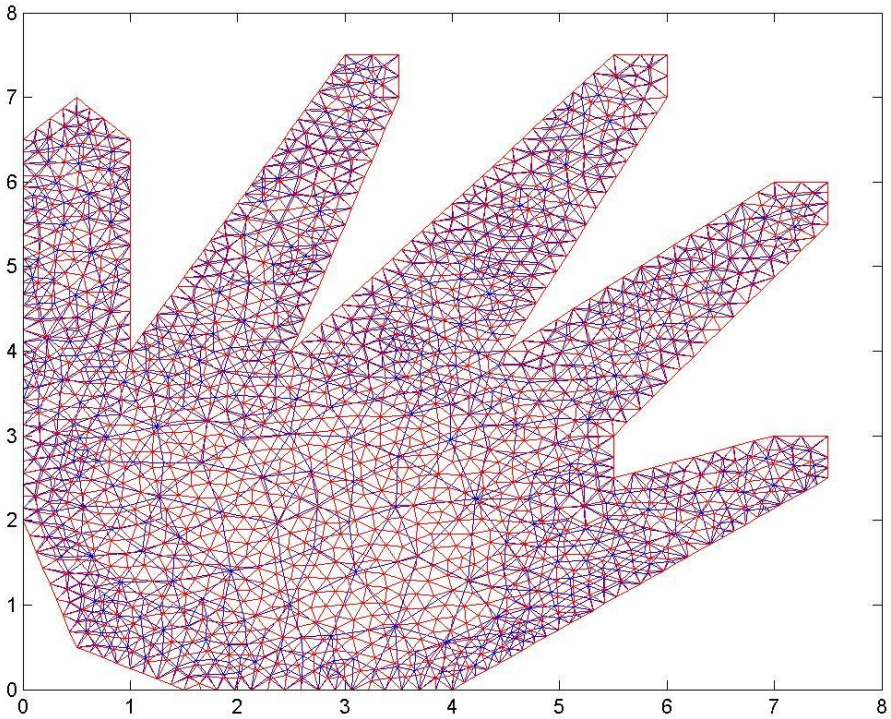


(1330,2380)　圖 29.

最後我們適當的取範圍得到下面的結果(圖 30. & 圖 31.)：



(1130,1980)　圖 30.



red(1663,3046),blue(1130,1980)　圖 31.

對第一部分測試的結果來看，並無破壞網格結構，對縮減區域也有不錯的結果。因此接下來我們將在數值計算上以及處理速度上做一些討論。

網格在數值計算上的應用常見的方式爲有限元素法，方法上將每個面當成一個元素來處理，經過計算後形成的矩陣求得數值解。第二部分中我們將用有限元素法解一般常見的偏微分方程例子，針對解的誤差值以及處理時間上做討論。

例子一：

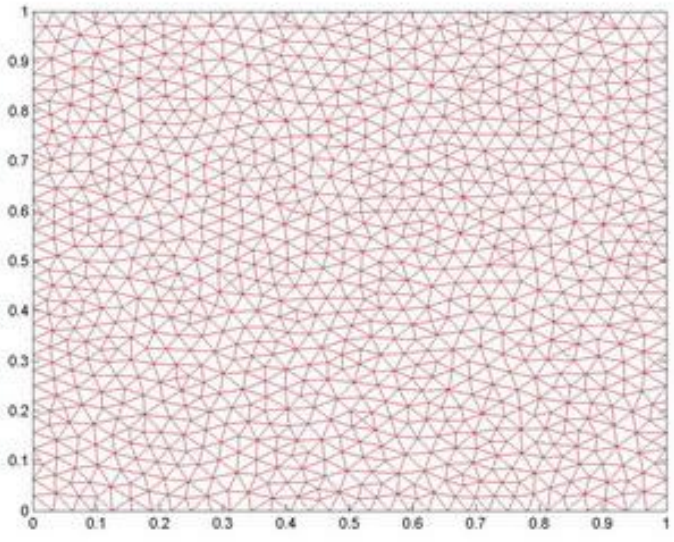
Laplace's equation

PDE : $\nabla^2 u = 0 \quad \Leftrightarrow \quad \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$

實際解設 u = xy，邊界條件預設爲實際解 u = xy

以有限元素法求數值解可知精準度與元素大小有關，當分割的越細計算出的精準度就會越高，而誤差估計則與元素劃分時最大三角形邊長 h 有關，一般來說如果實際解爲二階導數則函數誤差與h^2等價，一階導數誤差與 h 等價，因此可以預測 Laplace's equation 數值解誤差在h^2。

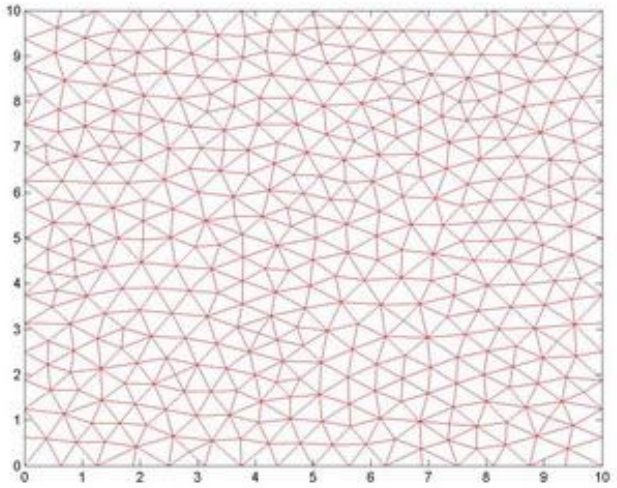
在此將以下列網格(圖 32.)做計算求數值解並比較：



(1101,2072) ,
[0,1]X[0,1]

圖 32.

此網格有 1101 節點 2072 三角元素(面)，最大邊長 h=0.0482，X、Y 介於[0,1]。實際最大誤差爲 8.5941×10^{-5}，$h^2=0.0482^2=0.00232324=2.32324 \times 10^{-3}$，誤差值在預估之內。

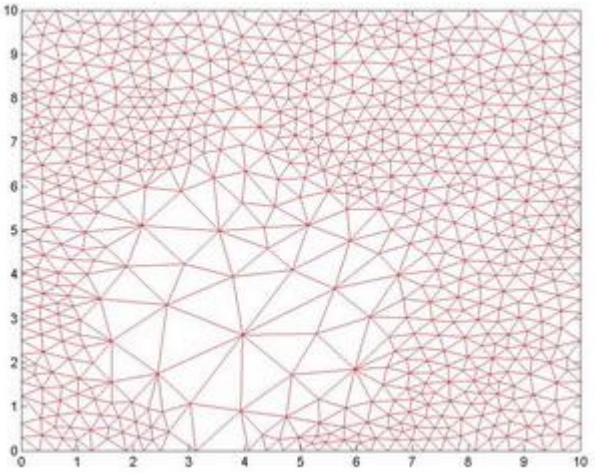


(432,798) ,
[0,10]X[0,10]

圖 33.

以元素較大的網格(圖 33.)測試，此網格中最大邊長 h=0.8046，X、Y 介於 [0,10]。實際最大誤差爲 $1.83x10^{-2}$，$h^2=6.4738116x10^{-1}$，誤差值仍在預估之內。因此我們以有限元素法來計算此問題是可行的。

接下來以局部縮減之網格(圖 34.)做測試：



(758,1411) ,
縮減區域爲 (4,3.5,3)
[0,10]X[0,10]

圖 34.

在此網格(圖 34.)中，經由誤差估計可以知道最大誤差會與縮減後的區域內最大元素邊長 h 有關，因此可以預想縮減區域的誤差會最大，而未縮減的區域與原網格之誤差相差不大。此網格最大邊長 h=1.956，實際最大誤差爲 $1.336\ x10^{-1}$，$h^2=3.8259$ 仍在誤差之內。

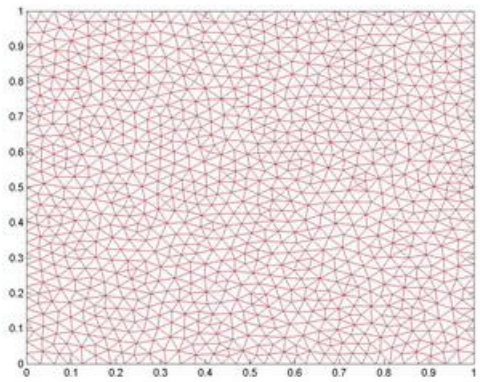
本研究的目的在於針對選定區域縮減網格，縮減比例依需求設定，在此預設皆爲 10%，也就是將區域內 90%左右之節點消除，在此之下誤差因部分元素變大而增加但是以誤差估計來看仍然在範圍之內，也就是說在數值處理過程中這區域因爲少了 90%個節點而省去 90%的時間，代價就是數值的精準度下降，不過這區域的誤差仍在預計的範圍。

在數值計算上除了有限元素法之外也有其他的方法，在此僅以常見的解法來比較，對於其他精準度較高的算法則不在本研究的討論之內。根據上面的例子結果可知在誤差估計上除了縮減區域精準度下降，未縮減之部份可說沒有影響。因此接下來針對處理時間上做討論。

因爲研究上的方便性，本研究使用 matlab6.5 編寫程式並執行所有結果，基於軟體本身的限制，節點、網格數目有上限，所以無法對 2 萬節點以上的網格做測試，因此僅對 1 萬以下節點做一些比較分析。

首先紀錄將原網格縮減所需時間，針對須處理節點數的多少所花的時間上也會因此增加或減少。因爲科學計算上的問題以下網格我們將保留所有邊界點。

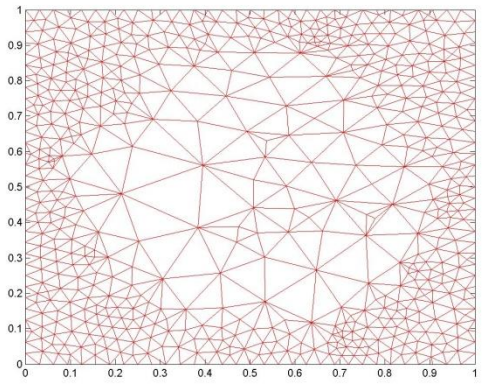
因爲程式內如何選擇刪除節點爲亂數決定，因此即使是同一區域進行縮減後的結果也略有不同。右圖 35.爲原網格圖，使用有限元素法所花時間平均 0.7 秒。



(1101,2072)

圖 35.

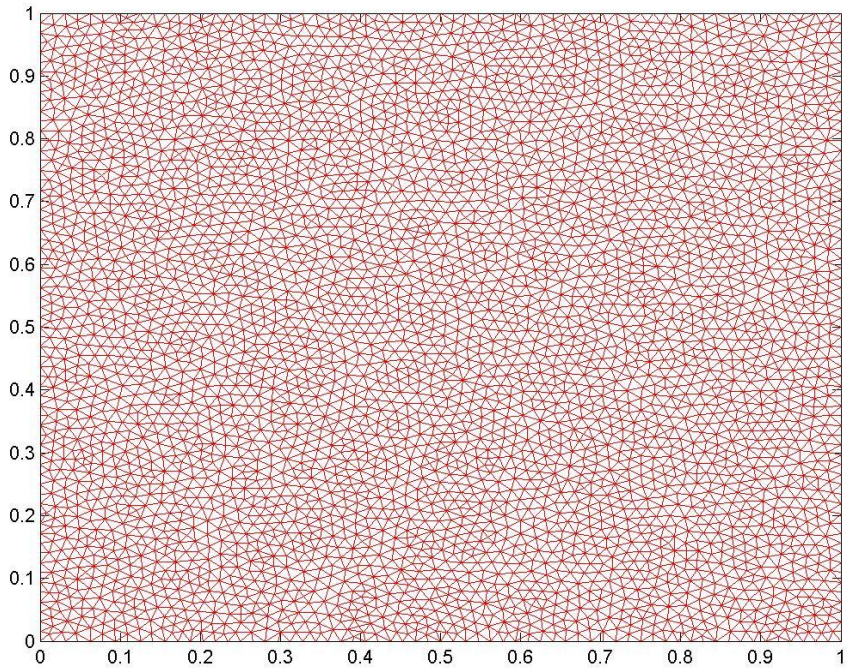
將原圖取中心範圍縮減後得到(638,1146)右下圖 36.，所花時間 6.2 秒。



(638,1146)

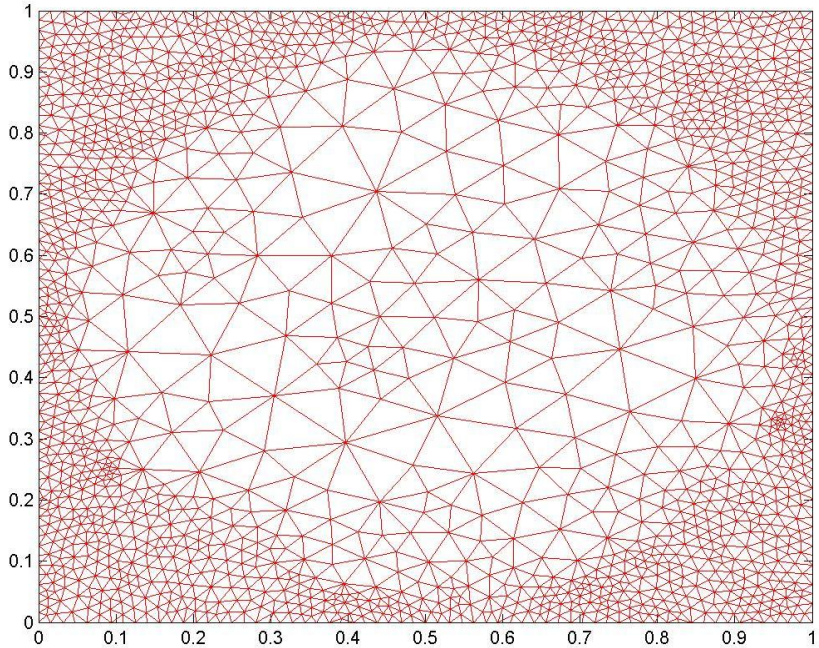
圖 36.

此問題最後以(4233,8208)圖 37.所構成的網格做測試，以有限元素法處理此網格所花平均時間爲 36 秒。將此網格進行中心區域縮減所花時間爲 80 秒(1729,3200)圖 38.。



(4233,8208)

圖 37..



(1729,3200)

圖 38.

根據上述結果可見縮減所花時間比直接計算還要多，對此進行程式內部分別討論所花時間，針對第一部分搜尋並建立資料上所花時間爲總時間之 1/2 以上，而在縮減部分以及優化網格部分則佔總時間約 1/3，因此若能有效建立資料將可節省很多時間。另外基於軟體處理速度上的問題也不容忽視。

在數值計算上求解的方法有很多，在此就不對數值方法所花時間進行討論。

例子二：

Stokes Equations

$$\text{PDE}: \begin{cases} -\nabla^2 u + \nabla P = f \\ \nabla \cdot u = 0 \end{cases}$$

$U = (u_1, u_2)$分別爲 X,Y 方向之流速，

∇P 爲壓力梯度，$f = (f_1, f_2)$爲應對力

$$\Leftrightarrow \begin{cases} -\dfrac{\partial^2 u_1}{\partial x^2} - \dfrac{\partial^2 u_1}{\partial y^2} + \dfrac{\partial P}{\partial x} = f_1 \\ -\dfrac{\partial^2 u_2}{\partial x^2} - \dfrac{\partial^2 u_2}{\partial y^2} + \dfrac{\partial P}{\partial y} = f_2 \\ \nabla \cdot u = 0 \end{cases}$$

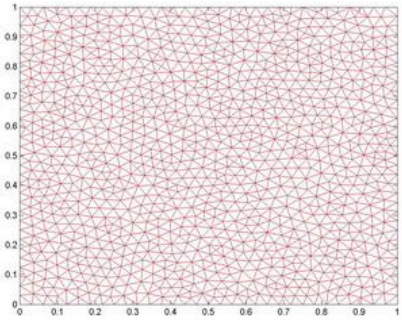
實際解設爲 $u = \begin{cases} u_1 = x^2(1-x)^2(2y - 6y^2 + 4y^3) \\ u_2 = y^2(1-y)^2(-2x + 6x^2 - 4x^3) \end{cases}$

$P = x^2 + y^2$

$f = \begin{cases} f_1 = 2x + (2y - 6y^2 + 4y^3)(12x^2 - 12x + 2) + (-12 + 24y)(x^4 - 2x^3 + x^2) \\ f_2 = 2y + (12 - 24x)(y^4 - 2y^3 + y^2) + (12y^2 - 12y + 2)(-2x + 6x^2 - 4x^3) \end{cases}$

對此問題同樣以有限元素法處哩，首先測試誤差是否與例一相同，也就是當最大邊長 h 時誤差在h^2。測試網格如右圖 39.：

此網格有 1101 節點 2072 三角元素(面)，最大邊長 h=0.0482，X、Y 介於[0,1]。實際最大誤差爲 $5.3407x10^{-5}$，$h^2=0.0482^2=0.00232324=2.32324x10^{-3}$，誤差值在預估之內。下圖 40.爲數值解之流速圖。花費平均時間 0.94 秒。



(1101,2072) , [0,1]X[0,1]

圖 39.

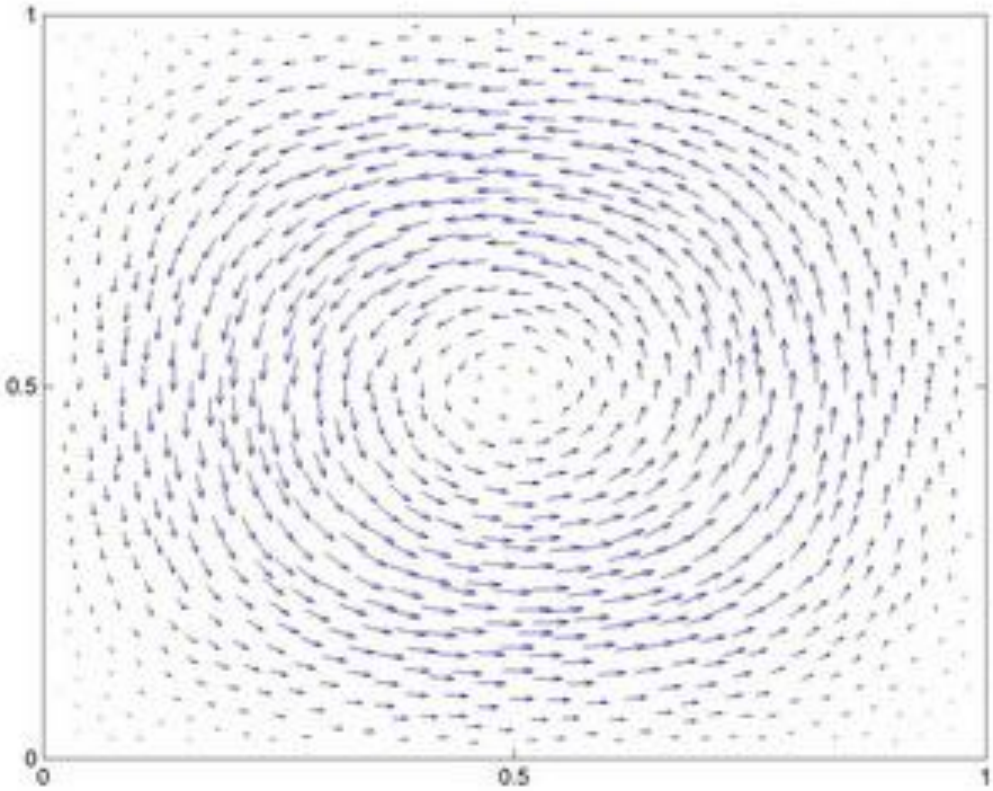


圖 40.

下圖 41.爲流速圖和網格重疊圖：

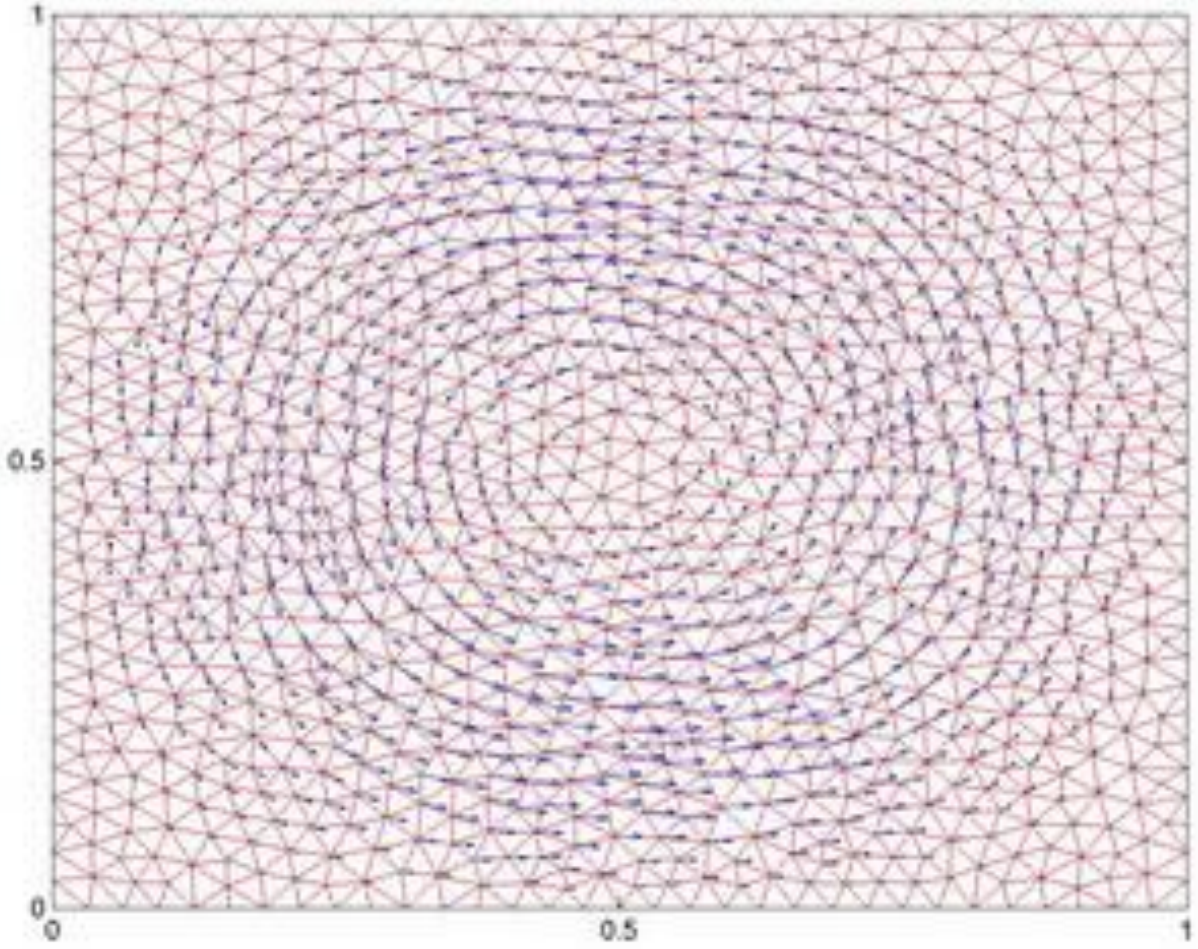
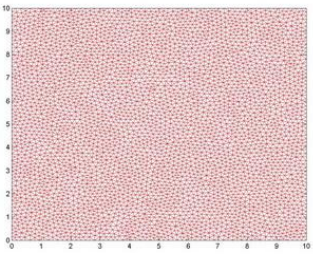


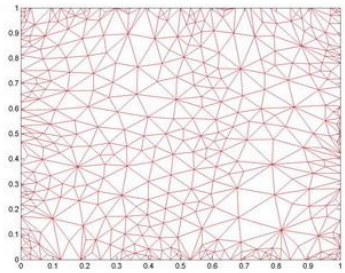
圖 41.

對此例題我們分別以更密之網格(圖 42.)進行全域縮減以及部分區域縮減進行測試。使用有限元素法所花平均時間 30.6 秒。將此網格進行縮減已於例一測試過，因此只將縮減後網格進行測試。



(4233,8208)　圖 42.

進行全域縮減後之網格爲(427,740)(圖 43.)，使用有限元素法所花平均時間 0.147 秒。數值解形成流速圖 44.如下：



(427,740)　圖 43.

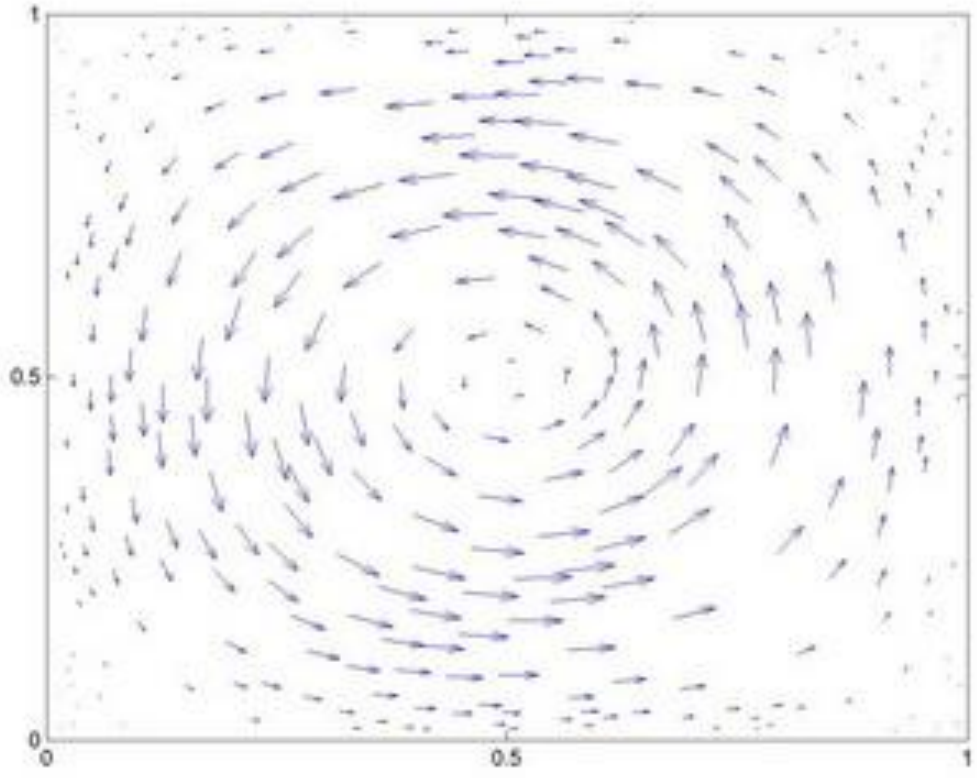


圖 44.

6 結論

對於本文所提出之方法確實可以在指定區域進行平面網格縮減，實驗結果顯示雖然縮減結果符合預期，但是程式執行時間上也不少，這方面可以在建立資料時只對縮減區域建立所需資料而不對整體建構，藉此可以省去建立不需處理的網格時間。從實驗結果來看建立資料與縮減過程花費時間大約各占一半，因此可預期只在縮減區域進行處理花費時間會大幅減少，這是一個改進的方向。

由於本文只在 1 萬點以下做測試並不能真正反映出此方法的速度，我們從程式處理上做分析，我們以每填入一格資料算一次處理動作，建構全部節點與網格(即輔助矩陣)所花的成本若節點數爲 N 網格數爲 M 則最多需花 15M+20N，而在處理網格縮減上每個節點最多進行 10 次的置換邊，因此若想縮減至 10%以下的節點數，最多花費時間爲 0.9 X 10 X N = 9N。也就是總共最多花費 15M+29N 的計算量可以獲得 0.1*N 的縮減網格，以這種理論上來看在節點數很大時因該可以看出其速度，如果不建構輔助矩陣而以其他方法代替，可以預見縮減至 10%的網格所需計算量最多僅需 9N。

對於未來除了程式寫法有待改進外仍然有很多發展的空間，例如將此方法與數值計算直接結合計算、推廣至三維空間進行處理，在不同的數值方法下做時間及誤差的分析討論仍不失爲很不錯的研究方向。

參考文獻

[1] Hugues Hoppe, “ Progressive Meshes”, . SIGGRAPH, 1996.

[2] Lori A. Freitag, “ON COMBINING LAPLACIAN AND OPTIMIZATION-BASED MESH SMOOTHING TECHNIQUES”,1997

[3] Howard C. Elman, David J. Silvester, Andrew J.Wathen ,“ finite elements and fast iterative solvers :with applications in incompressible fluid dynamics”, 2006

[4] Nicholas Bray, “Notes on Mesh Smoothing”, October 18, 2004

[5] Mari, Jo a.F., Saito, J.H., Poli, G., Zorzan, M.R., Levada, A.L.M. “ Improving the neural meshes algorithm for 3d surface reconstruction with edge swap operations”,In: SAC '08: Proceedings of the 2008 ACM symposium on Applied computing,New York, NY, USA, ACM (2008) 1236-1240