行政院國家科學委員會專題研究計畫　成果報告

3D 模型包絡球快速演算法

計畫類別：個別型計畫
計畫編號：NSC94-2623-7-007-007-
執行期間：94 年 01 月 01 日至 94 年 12 月 31 日
執行單位：國立清華大學數學系(所)

計畫主持人：王偉成

報告類型：完整報告

處理方式：本計畫可公開查詢

中　華　民　國 95 年 3 月 9 日

多層次結構體包絡球的快速演算法

王偉成　簡龍昇

國立清華大學數學系

摘　　要

我們針對具有樹狀結構的資料點，發展一套用來計算其主結構及各層次結構所有最小包絡球的快速演算法。此算法在計算各個次結構的最小包絡球的同時，也對資料點作分析，來找出這群資料點的凸包或近似凸包，並將這些凸包頂點或近似凸包外部的資料點上傳至上層結構以減少上層結構的資料點總數以及計算量。經過詳細分析與實際測試，凸包頂點篩檢法與近似凸包頂點篩檢法可以有效的降低整體計算量，提高工作效率。

關鍵字：凸包、多層次結構、包絡球

A Fast Algorithm for Enveloping Balls of Hierarchical Structures

Lung-Sheng Chien and Wei-Cheng Wang

Department of Mathematics
National TsingHua University

ABSTRACT

We propose a convex-hull selection algorithm to compute the enveloping balls of an object and sub-objects of all levels for data points arranged in a hierarchical tree structure. When computing the enveloping ball of a sub-object, we locate the vertices of its convex hull or approximate convex hull at the same time. Then we pass these convex hull vertices, (or points outside the approximate convex hull, respectively) instead of the whole substructure, into the parent node of the tree in order to compute the enveloping balls for the sub-objects in the parent level. The computation for the parent level enveloping balls will then benefit from this selection since the total number of data points is reduced significantly. From our analysis and numerical experiments, the convex hull (or approximate convex hull) selection algorithm are capable of reducing the total computational cost and improve the efficiency significantly.

Key Words: convex hull, hierarchical structure, enveloping balls.

一、前言

在視效影像模擬時，視效資料庫中存有許多三維結構體。在動態影像繪製時，必須先判別各個結構體是否出現於視窗中，完全看不到的結構體不必再計算，以節省運算時間，提高繪圖效率。

一個結構體的包絡球是包含此結構體的最小圓球。只需記錄此包絡球的中心點位置及球半徑，便可以非常快速的判別此結構體是否出現於視窗中，藉此決定是否繼續處理此結構體的視效影像計算。

一個三維結構體通常包含幾個較小的次結構體。而這些次結構體也由更小的次結構體所組成。最小的結構體是單一三角形，由三個座標點組成。

在視效影像前處理時，每一層級的結構體都要計算包絡球。最上層的結構體至少包含數千個座標點，在本文中，我們提供一種快速的演算法，希望能以較短的時間計算出每一層級結構體的包絡球。

二、凸包頂點篩檢法

當資料點不具有樹狀結構時（單層結構，只計算一個包絡球）最佳演算法的計算量為$O(N)$。當資料點有樹狀結構，並且需要計算每一層次結構所有的大小包絡球時，直接的演算法是將次結構的小包絡球計算出來之後，將所有次結構的資料點集合到上一層的結構中，用來計算上一層結構較大的包絡球，如此重覆到最上層為止。

因為計算包絡球的計算量與資料點的數目成正比，因此每一層的總計算量是一樣的(正比於資料點的總數)。所以最終的總計算量正比於（資料點數目）乘以（層數），亦即$O(N\log N)$的計算量。

以此演算法為根據加以改進，我們把資料點由次結構上傳至上一層結構時，先過濾掉一部分的資料點，只保留有用的資料點上傳至上一層結構，從而減少上一層結構包絡球的總計算量。

這個篩選部分資料點的原則，是根據下列的定理：

定理一[1]：

一組資料點$S=\{p_1,p_2,\cdots,p_n\}$的包絡球與這組資料點的凸包（Convex Hull）的包絡球相同。

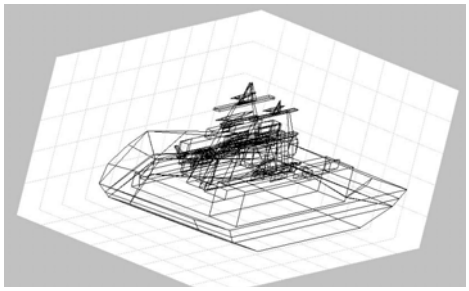
換句話說，這個改進的演算法，是在每一層結構，先行計算各組資料點的凸包，（亦即保留凸包的頂點），據此算出各組的包絡球，再將這些凸包的頂點（亦即一部分的資料點），上傳至上一層結構中，重覆此步驟直至最上層為止。

這個新算法的效率，取決於經篩選過後的凸包頂點數目是否夠少。平均而言，如果一個結構體原先所含資料點的數目是$|S|=k$個，其凸包頂點（ $CHV(S)$ ）的數目大約是$O(\sqrt{k})$ [2]：

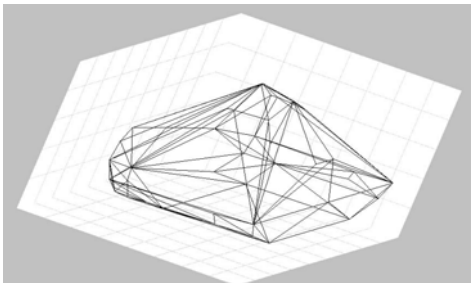
$$|\ CHV(S)\ | \sim C|\ S\ |^{\frac{1}{2}} \qquad (1)$$

因為每一層的資料點數目比起原始方法（每一層全部打散重算）少了許多，因此計算量也跟著減少了。

範例一：以船艦模型為例(見圖一)，此模型由 552 個資料點及 375 個面所構成，但其凸包(見圖二) 只餘下 42 個資料點及 80 個面，約為原資料點數目之十分之一。根據定理一，此二者之最小包絡球是相同的。

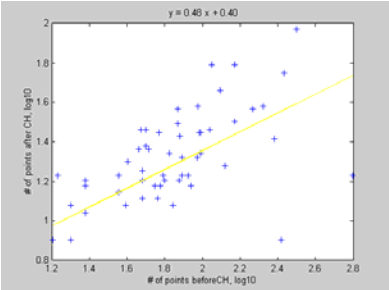


圖一：模型船之三維結構，此結構體由 552 個資料點及 375 個面所構成。



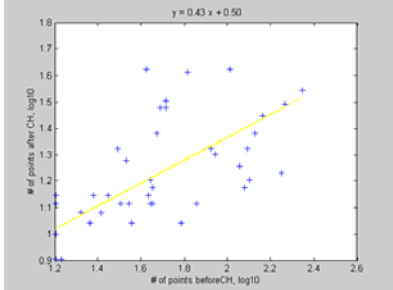
圖二：模型船之凸包結果，經過凸包計算，只餘下 42 個資料點及 80 個面。

範例二：以靶場為例(見圖三)，以最小方差 (Least-Square) 作最適曲線 (curve fitting) 其凸包頂點總數約為 $O(\sqrt{k})$，k = 原資料點數目。同樣的船艦模型(見圖四)亦有此特性。



圖三：靶場 field.txt

$|CHV(S)| \approx 2.5|S|^{0.48}$

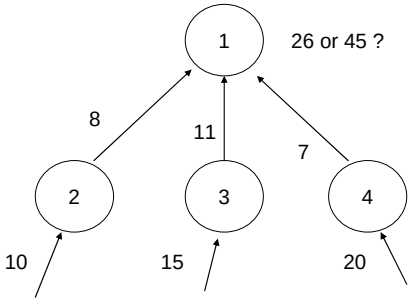


圖四：模型船 ship.txt

$|CHV(S)| \approx 3.16|S|^{0.43}$

以下我們針對上述方法的計算量做較為詳盡的分析。首先是結構體凸包頂點的篩檢比例。

定義一：在待測結構體之樹狀結構下，每一節點有兩個資料，匯入點(incoming data points)及匯出點(outgoing data points)。令每一節點所需計算的資料點數為及子節點(subtree) 之資料點總數，可為匯入流或是匯出流，且允許重複資料點出現，如圖一：



圖一：在此例中節點 1 是父節點(parent)，有 3 個子節點，分別是節點 2, 3 和 4。節點 2 有 10 個匯入點，節點 3 有 15 個而節點 4 有 20 個。除此之外節點 2 有 8 個匯出點，節點 3 有 11 個而節點 4 有 7 個。如果採用子節點之匯入點累加則節點 1 有 45=10+15+20 個匯入點，另一方面，如果採用子節點之匯出點累加則節點 1 有 26=8+11+7 個匯入點。

表一 ： 凸包與非凸包資料點數目分析

欄位一 ： 測試檔包含靶場地形，船艦及坦克模型，數個地形地物結構。

在往後的討論中我們用以下的測試檔 ：

(1) 模型坦克 ：T63　(2) 模型船 ： Ship　(3) 靶場 ： field

(4) 地形地物 ： 包含 18_27, 19_27, 21_22, 22_22, 18_26, 19_26, 21_21, 22_21, L5, L5_2，其中 L5_2 是由兩個 L5 合併而成

欄位二 ： 用 *QuickHull* 做前處理，累加所有節點之匯入點。匯入點的計算方式包含(1)如果子節點已做凸包則累加子節點之匯出點。相反的，如果子節點未做凸包則累加子節點之匯入點。

欄位三 ： 累加所有匯出點（凸包頂點總數）。

欄位四 ： 資料點篩檢比例 = 匯出點總和除以匯入點總合。

欄位五 ： 累加所有節點之匯入點（執行 *EB* 前各層資料點總數），匯入點為所有子節點之匯出點累加。*EB* (Enveloping Ball)為計算包絡球的副程式。

欄位六 ： 累加所有節點之匯出點（各層資料點執行 *EB* 後剩餘點數）。

欄位七 ： 資料點篩檢比例 = 匯出點總和除以匯入點總合。

(1) Benchmark (測試檔)	(2) 匯入 *QuickHull* 的各層資料點總數	(3) *QuickHull* 匯出的各層凸包頂點總數	(4) 資料點篩檢比例 (3)/(2)
T63	2244	812	36.19 %
ship	3020	967	32.02 %
field	5929	1522	25.67 %
18_27	15608	2738	17.54 %
19_27	17323	3076	17.76 %
21_22	16974	3121	18.39 %
22_22	17160	3165	18.44 %
18_26	17297	3154	18.23 %
19_26	17168	3029	17.64 %
21_21	16901	3068	18.15 %
22_21	17062	3253	19.07 %
L5	276191	50490	18.28 %
L5_2	552612	101107	18.3 %

Benchmark (測試檔)	(5) 執行 *EB* 前各層資料點總數	(6) 各層資料點執行 *EB* 後剩餘點數總和	(7) 資料點篩檢比例 (6)/(5)
T63	10612	10433	98.31 %
ship	15797	15652	99.08 %
field	23879	23682	99.18 %
18_27	104814	104406	99.61 %

19_27	114891	114207	99.40 %
21_22	109270	108056	98.89 %
22_22	106857	104197	97.51 %
18_26	115893	115464	99.63 %
19_26	108512	105772	97.47 %
21_21	105726	103913	98.29 %
22_21	105629	103218	97.72 %
L5	2467534	2429801	98.47 %
L5_2	5311449	5236001	98.58 %

平均而言，各層凸包頂點總數約為各層資料點總數的 18%。所以將凸包頂點篩檢作為 *EB* 之前處理（preprocessing）是一個合理的構想。但是實際操作上則需考慮計算凸包之負載(overhead）是否可接受，此處「可接受」意指其執行時間不可超過 *EB* 過多，我們會緊接著就此作討論。

定義二：凸包頂點篩檢法 *CHVS* (Convex Hull Vertices Selection)：在待測結構體之樹狀結構下，在上傳資料點至父節點（parent)前，先選擇凸包頂點。此法可將資料點總數由 $|S|$ 降低至 $|CHV(S)|$，從而提昇副程式 *EB* (Enclosing Ball)的執行效能。

假設待測結構體之樹狀結構是完全二元樹(fully binary tree)，我們可對 *CHVS* 方法之運算量(Operation count) 做如下的估計。我們定義

N ：測試檔之資料點總數（二元樹之深度為 $\log N$）。

Q_k ：第 k 層所有節點的總數。

R_k ：第 k 層每個節點平均凸包頂點數

P_k ：由第 $k+1$ 層所上傳至第 k 層之平均節點數。

再假設常數 C_1 與 C_2 分別為平均在每個節點上用來計算包絡球與凸包頂點所需時間，而 C 則為式 (1) 中的比值常數。此三者的確切數值需由實驗決定。此常數隨著不同的作業系統，編譯器，甚至是硬體快取(cache)大小而有差異。我們有如下之關係式

$$Q_k = 2^k，\quad P_k = 2C\sqrt{\frac{N}{Q_{k+1}}}，\quad R_k = C\sqrt{\frac{N}{Q_k}}，\quad EB_k = C_1 R_k Q_k，CHVS_k = C_2 P_k Q_k$$

以此關係式，遞迴使用可形成表二

表二：*CHVS* 之理論運算量分析

層(Level)	P_k	R_k	EB_k	$CHVS_k$
0	$2C\sqrt{\frac{N}{2}}$	$C\sqrt{N}$	$C_1C\sqrt{N}$	$C_2C\sqrt{2N}$
1	$2C\sqrt{\frac{N}{4}}$	$C\sqrt{\frac{N}{2}}$	$C_1C\sqrt{2N}$	$C_2C\sqrt{4N}$
k	$2C\sqrt{\frac{N}{2^{k+1}}}$	$C\sqrt{\frac{N}{2^k}}$	$C_1C\sqrt{2^k N}$	$C_2C\sqrt{2^{k+1}N}$

凸包頂點篩檢法的總運算量 = $\sum_{k=0}^{\log N}\left(EB_k + CHVS_k\right) \le \frac{C}{\sqrt{2}-1}\left(\sqrt{2}C_2 + C_1\right)\sqrt{N}\left(2\sqrt{N}-1\right)$

總結我們可以算出凸包頂點篩檢法(包括「篩選凸包頂點」以及「計算凸包頂點的包絡球」)，由最下層至最上層的總計算量是$O(N)$。而直接法則是$C_1 N \log N$的總運算量。

對於形狀較為複雜的地形地物或飛行器、炮艦等，資料點的樹狀結構較長(亦即層數較多)，凸包頂點篩檢法相對於直接重新計算包絡球即佔有較大的優勢。實際的均衡點(Break-even point)當由資料點的結構與數目決定。為了驗證以上的理論，我們也對視效資料庫中典型的結構體，實際計算「直接演算法」以及「凸包頂點篩檢法」需要用來計算每一層次大小包絡球所牽涉的資料點總數加以比較。

三、凸包頂點子程式的比較

「凸包頂點篩檢法」的效率取決於計算凸包頂點子程式的效率，因此我們檢視了幾個已知的凸包頂點子程式。首先是 Chand and Kapur 在 1970 年提出的 Gift-Wrapping 方法[3]，利用三角形的凸包表面逐次將整個凸包像包禮物般的包裹而成，一如其名。此法之複雜度為$O(fk)$，k 為資料點的個數而 f 為凸包表面的面數，在資料點本身接近凸包時，效率最差為$O(k^2)$。接者是 Preparata 和 Hong 在 1977 年提出的 Merge Hull [4]，此法的動機 (motivation) 是為了將兩個凸包合併 (Merge) 成一個大凸包，作者利用特殊資料結構 DCEL (double-connected-edge-list) 作出$O(k)$的複雜度，為階層式 (Hierarchical divide and conquer) 的凸包演算法開啟$O(N \log N)$之新世紀。再來是 Kalley 在 1981 年提出的 Beneath and Beyond [5]，一次增加一個點 p 並計算 p 在現有之凸包所形成之支持椎 (supporting cone)，移除在支持椎以下的部份近而形成新的包含 p 點的凸包，可以說是點態的包裹法，和 Gift-Wrapping 的面包裹法大異其趣，但其低效能$O(k^2)$確是缺陷。最後要提的是 C.Barber 等在 1996 年所提出的 *QuickHull* [6]，此法為任意維度的凸包演算法，結合了二維的 *QuickHull* 概念與 Beneath and Beyond 的點態包裹技巧所形成，在效率上有重大的突破，運算量為$O(k \log V)$，V 為凸包頂點的數目。雖然在資料點本身接近凸包效能最差，但也比 Gift-Wrapping 快$O(k / \log k)$倍。

其次 Merge Hull 需要兩個互斥(disjoint)的凸包做合併(merge)，在我們的 3D 模型中，結構已經事先切割完畢並且存成樹狀結構，此結構無法保證兩群由下層上傳的子節點的凸包是互斥的，所以 Merge Hull 並不適用。另外 *QuickHull* 已經內建 Beneath and Beyond 的想法，且其執行效能遠優於 Beneath and Beyond，因而我們捨 Beneath and Beyond 而就 *QuickHull*。

我們將自己實作(implement)的Gift-Wrapping與採用公開原始碼(open source) 的 *QuickHull*演算法(可由http://www.qhull.org下載)做比較。結果如表三：

表三：Gift-Wrapping 和 *QuickHull* 效能之比較

項目 \ 測試檔	Field	18_27	19_27	21_22
(1) 資料點數目	1862	2065	2390	2068
(2) Gift-Wrapping 所需時間	188.8 ms	230.5 ms	336.3 ms	281.7 ms
(3) *QuickHull* 所需時間	2.78 ms	3.52 ms	5.03 ms	4.29 ms
(4) Gift-Wrapping : *QuickHull*	67.9 : 1	65.5 : 1	66.9 : 1	65.7 : 1

項目 \ 測試檔	T63	Ship	L5	L5_2
(1) 資料點數目	471	552	28491	56982
(2) Gift-Wrapping 所需時間	55.13 ms	32.82 ms	8.01 s	22.8 s
(3) *QuickHull* 所需時間	1.39 ms	1.228 ms	34.07 ms	68.27 ms
(4) Gift-Wrapping : *QuickHull*	39.7 : 1	26.7 : 1	235.1 : 1	334 : 1

上表顯示 *QuickHull* 優於 Gift-Wrapping，在資料點超過二萬點時，雙方速度差距上百倍。

四、「凸包頂點篩檢法」與「直接演算法」的比較

我們選擇 *QuickHull* 作為前處理器(preprocessor)，在樹狀結構由底部往上（bottom-up）以後序排列(post-order）執行以下的動作：

1. 如果是終端節點 (leaf node)，不作凸包頂點篩檢，直接計算包絡球。
2. 如果是第 k 層內節點 (internal node)，先匯集子節點之匯出點，然後用 *QuickHull* 篩檢凸包頂點，再由此凸包頂點計算包絡球。

在上述的動作中不難發現我們需要兩種資料結構的轉換，一是形成 *QuickHull* 之輸入結構，二是形成 *EB* 之輸入結構。因為 *QuickHull* 與 *EB* 為不同資料結構，此轉換勢將不可避免，縱使資料轉換的運算量是 $O(N)$，量測資料轉換之負載 (overhead) 還是有其必要性。為了找出凸包頂點篩檢法的優劣(pros & cons)，在表四中我們做以下的效能評比

1. 比較 *QuickHull* 與 *EB* 之效能，此評比排除資料轉換的時間，單純是演算法的較量。
2. 比較「凸包頂點篩檢法」與「直接演算法」的資料型態轉換負載，主要目的是評估 *QuickHull* 之資料轉換負載。
3. 比較「*EB* 在凸包頂點篩檢法之負載」與「*EB* 在直接演算法之負載」，此結果將反映凸包頂點少於原始資料點的優勢。

表四 ：「凸包頂點篩檢法」與「直接演算法」的比較

列一 ： 測試檔之資料點數目，<u>重複點只算一次</u>。

列二 ： 凸包頂點篩檢法在整個樹狀結構所需時間，包含列三~五。

列三 ： *QuickHull* 在整個樹狀結構所需時間。

列四 ： 最小包絡球在整個樹狀結構所需時間。

列五 ： 在每一個子節點，我們需計算凸包及包絡球，需轉換兩種不同之資料型態，其一為 *QuickHull* 之輸入，其二為 *EB* 之輸入，此二者總和即為資料轉換時間。

列六 ： 直接演算法在整個樹狀結構所需時間，包含列七~八。

列七 ： 最小包絡球在整個樹狀結構所需時間。

列八 ： 資料轉換時間，直接演算法只有單一資料結構。

列九 ： 比較「凸包頂點篩檢法之資料型態轉換時間」與「直接演算法之資料型態轉換時間」。

列十 ： 比較「*QuickHull* 在凸包頂點篩檢法之負載」與「*EB* 在直接演算法之負載」。

列十一 ： 比較「*EB* 在凸包頂點篩檢法之負載」與「*EB* 在直接演算法之負載」，因為 *QuickHull* 已經拔除凸包內部資料點，所以「*EB* 在凸包頂點篩檢法」所過濾的點數較少，可視為 *EB* 之增益比。

列十二 ： 比較「凸包頂點篩檢法」與「直接演算法」之效能。

項目 \ 測試檔	Field	18_27	19_27	21_22
(1) 資料點數目	1862	2065	2390	2068
凸包頂點篩檢法				
(2) 凸包頂點篩檢法所需時間	48.15 ms	107.14 ms	115.19 ms	115.97 ms
(3) *QuickHull* 所需時間	38.70 ms	87.64 ms	96.55 ms	98.41 ms
(4) 包絡球所需時間	3.86 ms	1.24 ms	1.35 ms	1.4 ms
(5) 資料型態轉換所需時間	2.26 ms	7.62 ms	8.01 ms	7.4 ms
直接演算法				
(6) 直接演算法所需時間	13.46 ms	71.37 ms	72.97 ms	64.38 ms
(7) 包絡球所需時間	10.59 ms	61.13 ms	62.35 ms	54.74 ms
(8) 資料型態轉換	1.58 ms	5.51 ms	5.92 ms	5.56 ms
數據比較				
(9) 資料型態轉換比	1.43	1.38	1.35	1.33
(10) *QuickHull* 負載比	3.65	1.43	1.55	1.80
(11) *EB* 增益比	36.45 %	2 %	2.2 %	2.6 %
(12) 凸包頂點篩檢法 ： 直接演算法(時間比)	3.56 : 1	1.5 : 1	1.58 : 1	1.8 : 1

項目 \ 測試檔	T63	Ship	L5	L5_2
(1) 資料點數目	471	552	28491	56982
凸包頂點篩檢法				
(2) 凸包頂點篩檢法所需時間	26.23 ms	27 ms	1922.1 ms	3920.2 ms
(3) *QuickHull* 所需時間	22.29 ms	21.36 ms	1612 ms	3302.2 ms
(4) 包絡球所需時間	1.62 ms	2.81 ms	24.9 ms	41 ms
(5) 資料型態轉換	0.79 ms	1.19 ms	120.5 ms	241.7 ms
直接演算法				
(6) 直接演算法所需時間	5.6 ms	8.64 ms	2390.3 ms	5262.7 ms
(7) 包絡球所需時間	4.86 ms	7.58 ms	2201.1 ms	4885.6 ms
(8) 資料型態轉換	0.47 ms	0.65 ms	91.4 ms	184.9 ms
數據比較				
(9) 資料型態轉換比	1.68	1.83	1.32	1.31
(10) *QuickHull* 負載比	4.59	2.82	0.73	0.68
(11) *EB* 增益比	33.3 %	37.1 %	1.1 %	0.8 %
(12) 凸包頂點篩檢法 ： 直接演算法(時間比)	4.68 : 1	3.13 : 1	0.8 : 1	0.745 : 1

結論 ：

1. 由列九可知，*QuickHull* 的資料型態轉換時間約為直接演算法之資料型態轉換時間的 1.5 倍，原因是資料轉換至 *QuickHull* 的時間並不低，但因凸包頂點數較少，資料轉換至 *EB* 的時間反而較少(約為原 1/50 倍)，所以 1.5 倍的負載並不會是主要的瓶頸(bottleneck)。
2. 由列十一可知，經過凸包篩檢頂點後，*EB* 的計算量只需原來的 2% 左右，可見凸包搜尋的重要性。
3. 由列十可知，在資料點數目少於三千時，凸包頂點篩檢法並無法降低執行時間，因為 *QuickHull* 之負載遠超過 *EB* 之執行時間。但是在 L5 及 L5_2，凸包頂點篩檢法可以有效降低執行時間。實驗顯示資料點數超過二萬點時用凸包頂點篩檢法才有成效。

四、「近似凸包頂點篩檢法」

由表四的結果，我們發現當點數小於二萬點左右時，「凸包頂點篩檢法」並無優勢存在，原因是要精確的找出所有凸包頂點的工作量太大，因此我們也針對資料點數較少的結構體，發展了「近似凸包頂點篩檢法」。

定理二：

若點集 $P=\{p_1,p_2,\cdots,p_m\}$ 為點集 $Q=\{p_1,p_2,\cdots,p_n\}$ 的子集，則 $CH(P)\subseteq CH(Q)$.

在二維的例子中，一群資料點的最小包絡球有兩種可能，如圖 2-1： 有三點落在圓周上和圖 2-2：只有兩點落在圓周上，我們只考慮圖 2-1 的例子，相同的想法可套用在圖 2-2 的例子身上。在圖 2-1 中包絡球有一個內接三角形的頂點是由三個資料點 A, B 和 C 所構成。由定理二得知此三角形內部的資料點可以不必上傳至上一層(圖 3-1)，故此內接三角形即為第一階的近似凸包。

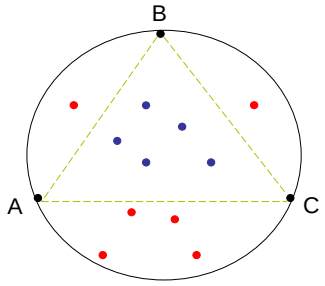


圖 2-1： 三角形 ABC 為圓內接三角形。

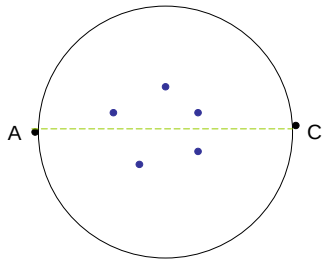


圖 2-2： 最小圓只通過 A 和 C 兩點所以 AC 為直徑。

而第二階的近似凸包則由三角形的大邊外側垂直距離最遠的資料點加入，由兩個三角形疊成一個四邊形所構成(圖 3-2 和圖 3-3)，以此類推，我們可以在其他方向加入最遠的資料點，得到更高階的近似凸包(圖 3-4)，一直到得到真正凸包為止。

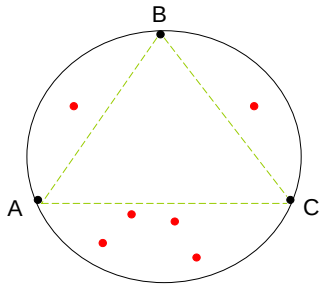


圖 3-1:三角形 ABC 為一階近似凸包。其內部資料點可移除，不必上傳

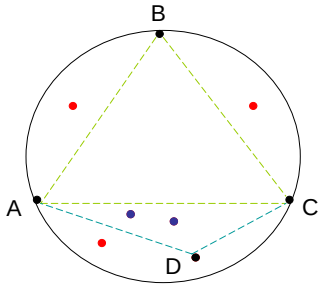


圖 3-2： 新增三角形 ADC，和三角形 ABC 合併為凸四邊形 ABCD。

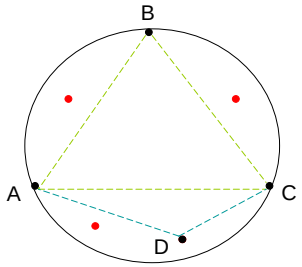


圖 3-3： 移除三角形 ADC 內部點。四邊形 ABCD 為二階近似凸包。

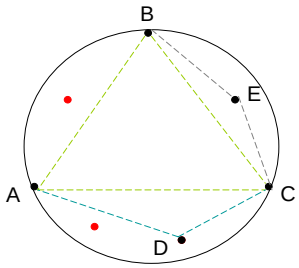


圖 3-4： 加入三角形 BCE 構成三階近似凸包。

而三維的對應情形最低階的近似凸包為球的內接四面體(圖 4-1)，第二階的近似凸包則由兩個四面體疊成的六面體所構成(圖 4-2)。

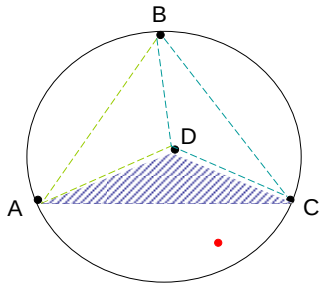


圖 4-1 ： 最小包絡球包含內接四面體 ABCD，一階近似凸包。

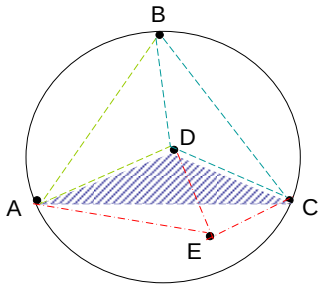


圖 4-2 ： 加入 E 點形成新四面體 ADCE，此為二階近似凸包。

一階與二階近似凸包的計算量遠較真正凸包為低，而所過濾的內部頂點也相對較少，因此可以降低過濾資料點這部份的計算量，但計算包絡球的工作量則介於「凸包頂點篩檢法」與「直接演算法」之間，見表五

表五：「二階近似凸包頂點篩檢法」與「直接演算法」之效能分析

列一：測試檔之資料點數目，重複點只算一次。

列二：二階近似凸包頂點篩檢法在整個樹狀結構所需時間，包含列三~四。

列三：最小包絡球在整個樹狀結構所需時間。

列四：資料轉換時間，「近似凸包頂點篩檢法」只有單一資料結構

列五：直接演算法在整個樹狀結構所需時間，包含列六~七。

列六：最小包絡球在整個樹狀結構所需時間。

列七：資料轉換時間，直接演算法只有單一資料結構。

列八：比較「二階近似凸包頂點篩檢法之資料型態轉換時間」與「直接演算法之資料型態轉換時間」。

列九：「*EB* 在二階近似凸包頂點篩檢法之負載」與「*EB* 在直接演算法之負載」之比較。因為二階近似凸包已經拔除凸包內部資料點，所以「*EB* 在凸包頂點篩檢法」所過濾的點數較少，可視為 *EB* 之增益比。

列十：比較「二階近似凸包頂點篩檢法」與「直接演算法」之效能。

項目 \ 測試檔	Field	18_27	19_27	21_22
(1) 資料點數目	1862	2065	2390	2068
近似凸包頂點篩檢法				
(2) 二階近似凸包頂點篩檢法所需時間	14.66 ms	54.93 ms	56.29 ms	53.83 ms
(3) 包絡球所需時間	12.24 ms	45.14 ms	46.28 ms	44.1 ms
(4) 資料型態轉換	1.38 ms	5.46 ms	5.9 ms	5.9 ms
直接演算法				
(5) 直接演算法所需時間	13.46 ms	71.37 ms	72.97 ms	64.38 ms
(6) 包絡球所需時間	10.59 ms	61.13 ms	62.35 ms	54.74 ms
(7) 資料型態轉換	1.58 ms	5.51 ms	5.92 ms	5.56 ms
數據比較				
(8) 資料型態轉換比	0.8734	0.9909	0.9966	1.0612
(9) *EB* 增益比	115.58 %	77 %	77 %	83.6 %
(10) 二階近似凸包篩檢法：直接演算法	1.09 : 1	0.77 : 1	0.77 : 1	0.84 : 1

項目 \ 測試檔	T63	Ship	L5	L5_2
(1) 資料點數目	471	552	28491	56982
近似凸包頂點篩檢法				
(2) 近似凸包頂點篩檢法	5.41 ms	7.69 ms	1756.1 ms	3858.5 ms
(3) 包絡球所需時間	4.83 ms	6.68 ms	1586.3 ms	3521 ms
(4) 資料型態轉換	0.37 ms	0.63 ms	96.6 ms	182.7 ms

直接演算法				
(5) 直接演算法所需時間	5.6 ms	8.64 ms	2390.3 ms	5262.7 ms
(6) 包絡球所需時間	4.86 ms	7.58 ms	2201.1 ms	4885.6 ms
(7) 資料型態轉換	0.47 ms	0.65 ms	91.4 ms	184.9 ms
數據比較				
(8) 資料型態轉換比	0.7872	0.9692	1.0569	0.9881
(9) *EB* 增益比	99.4 %	88.1 %	72.1 %	72.1 %
(10) 二階近似凸包篩檢法 : 直接演算法	0.97 : 1	0.89 : 1	0.73 : 1	0.73 : 1

結論 :

1. 「近似凸包頂點篩檢法」與「直接演算法」皆使用單一資料結構，資料型態轉換時間並無顯著差距，可由列八的數據支持。
2. 縱然如此，近似凸包畢竟可移除較多資料點 (見表六)，使得 *EB* 有顯著的進展，由列九觀察得知約為 30 % 改善。
3. 綜合上述兩點，假以列十的數據，「二階近似凸包頂點篩檢法」在資料點數超過二千點時皆可改善 30 %的效能。由是觀之，在資料點數等級為數千時，可採用「二階近似凸包頂點篩檢法」; 當資料點數等級為數萬甚至更高時，可考慮使用「凸包頂點篩檢法」。

表六 : 「二階近似凸包頂點篩檢法」與「直接演算法」資料點數目分析

欄位一 : 測試檔包含靶場地形，船艦及坦克模型，數個地形地物結構。

欄位二 : 累加所有節點之匯入點 (所有樹狀節點之 *EB* 前資料點總數)，匯入點為所有子節點之匯出點累加。

欄位三 : 累加所有節點之匯出點 (所有樹狀節點之 *EB* 後資料點總數) 。

欄位四 : 資料點篩檢比例 = 匯出點總和除以匯入點總合。

(1) Benchmark (測試檔)	(2) 所有樹狀節點 *EB* 前資料點總數	(3) 所有樹狀節點之 *EB* 後資料點總數	(4) 資料點篩檢比例= (3)/(2)
field	20385	17497	85.83 %
18_27	63018	54611	86.66%
19_27	68566	59580	86.89 %
21_22	69600	60895	87.49 %
22_22	72950	65185	89.36 %
T63	8292	7390	89.12 %
18_26	73079	64738	88.59 %
19_26	70278	61868	88.03 %
21_21	71624	63773	89.04 %
22_21	69785	61422	88.02 %
ship	11420	10132	88.72 %
L5	1343557	1176517	87.57 %
L5_2	2804568	2454996	87.54 %

五、結論

我們針對具有樹狀結構的資料點，發展一套用來計算其主結構及各層次結構所有最小包絡球的快速演算法。此算法在計算各個次結構的最小包絡球的同時，也對資料點作分析，來找出這群資料點的凸包或近似凸包，並將這些凸包頂點或近似凸包外部的資料點上傳至上層結構以減少上層結構的資料點總數以及計算量。

凸包頂點篩檢法的核心為凸包頂點搜尋法，我們比較各種凸包頂點搜尋法後，將 *QuickHull* 的原始碼嵌於階層式凸包頂點篩檢法中，並且在大量資料點的測試檔中展現預期的效率。但在少量資料點的測試檔中卻因 *QuickHull* 的負載遠高過 *EB* 負載而使凸包頂點篩檢法效能降低許多。

為此我們提出近似凸包的概念，不需精確的算出所有凸包頂點，而是引進”近似階數”，視情況需要而採用一階近似凸包或二階近似凸包，甚至是高階近似凸包。實驗顯示我們採用的二階近似凸包頂點篩檢法的計算量約只有直接法的 70 %。

因此我們建議

（1）點數少於二千點的結構體使用直接演算法。

（2）點數介於二千點至二萬點的結構體使用二階近似凸包頂點篩檢法。

（3）點數超過二萬點的結構體則使用凸包頂點篩檢法。

實驗環境與軟體

本實驗是架構在在 Intel i686 的機器，4G 記憶體，作業系統是 Linux Redhat 9.0 核心 2.4.20-31.9bigmem，使用 GNU g++ 編譯器，並且用 GNU gprof 作時序數據(timing profile)。至於 GUI 部份，則是採用 Qt 函式庫，所以程式碼可跨平台，已在 Linux Redhat 9.0 及 WinXP 上測試過。

參考文獻

[1] J.G. Hocking and G.S. Young, Topology, Addison-Wesley, Reading, MA, 1961..

[2] H. Raynaud, Sur l'enveloppe convexe des nuagesdes points aleatoires dans R^N, I, J. Appl. Prob. , 35-48 (1970).

[3] D. R. Chand and S. S. Kapur, An algorithm for convex polytopes, *JACM* 17(1), 78-86 (Jan. 1970)

[4] F. P. Preparata and S. J. Hong, Convex hulls of finite sets of points in two and three dimensions, *Comm. ACM* 2(20), 87-93 (Feb. 1977)

[5] M. Kallay, Convex hull algorithms in higher dimensions, unpublished manuscript, Dept. Mathematics, Univ. of Oklahoma, Norman, Oklahoma, 1981

[6] Barber, C.B., Dobkin, D.P., and Huhdanpaa, H.T., "The Quickhull algorithm for convex hulls," ACM Trans. on Mathematical Software, Dec 1996. http://www.qhull.org