# 分散式階層式輻射法之研究

# Distributed Hierarchical Radiosity Methods in Complex Environments

林永智　　鄭進和
Yeong-Jyh Lin　　Chin-Ho Cheng

輔仁大學資訊工程系
Department of Computer Science and Information Engineering
Fu Jen Catholic University,Hsinchuang, Taipei 24205, Taiwan,　R.O.C.
E-mail:{lin85, chcheng}@csie.fju.edu.tw

摘要

*在電腦圖學中，輻射法(radiosity)為全域照明(Global Illumination)的重要方法之一，可模擬相當真實的影像，但其充分考慮各物體彼此之間能量平衡的關係，因此必須消耗相當長的計算時間，近年來平行的架構非常普及，對於需要大量運算的問題，提供了一個相當不錯的解決途徑。在本篇論文中，我們利用 PVM (Parallel Virtual Machine) 的程式環境下，提出一個分散式的階層式輻射演算法，並以不同的資料分配方式，對於各主機之運算能力的不同所產生的影響加以討論。*

*關鍵字： 電腦圖學、輻射法、分散式處理、資料分配*

## 1. 簡介

在電腦圖學中，全域照明 (Global Illumination) 的方法中主要有兩種，一為光線追蹤法 (ray tracing)，另一為輻射法 (radiosity) [1, 6]。這兩種方法依照不同的性質而產生影像。光線追蹤法，是以模擬光線的進行而呈現影像。這些過程都與觀測者有密切的關係，也就是說每一次改變觀測點的位置，光線必須從眼睛的位置開始，重新再追蹤一次。光線追蹤法最大的缺點就是它沒有辦法在模擬環境中互動式的漫遊提供有效率的計算。

輻射法是根據實際的物理理論，從能量平衡的觀點呈現更加真實的影像，輻射法充分考慮整個環境之中各物體彼此之間能量相互傳遞、平衡的關係。而在繪出圖像之前完全與觀測者的位置、方向無關，只有當要顯示於螢幕時，才必須給定觀測點的參數。也因此，對於在模擬環境中的互動式漫遊，輻射法可以提供很有效率的處理。由於輻射法考慮到實際能量的傳遞、平衡，必須消耗相當長的計算時間，近年來平行的架構非常普及，對於需要大量運算的問題，提供了一個相當不錯的解決途徑[3]，在本篇論文中，我們以分散式處理的方式，減低階層式輻射法的計算時間。在過去，分散式階層式輻射法曾被研究過[2, 5]，所處理的場景資料具有規律性。在本篇論文中，將處理不具任何規律性的場景資料，因此，資料分配的方式，將會嚴重影響分散式輻射法的計算時間。在本篇論文中，我們的分散式演算法是 Hanrahan[4]的階層式輻射法為基礎，所考慮的分散式處理架構下，各主機之間的負載平衡會影響到一個平行演算法的執行效率，尤其是在各主機的運算量不同的狀況下，再加上階層式輻射法在執行的過程[4]，都是以動態的方式進行，資料的運算量也是無法掌握，因此，資料的適當分配就顯得十分重要。我們以靜態資料分配方式與動態資料分配方式加以討論，並考慮階層式輻射中，資料片的面積與運算量的關係加以分配資料，以期使各主機之間的負載平衡，讓平行演算法達到最好的效果。

## 2. 分散式階層式輻射法

### 2.1 分散式架構

在執行分散式階層式輻射法時，首先主處理器 (master)先將整個場景資料，包括原始的資料片 (patches)，四元樹 (quadtree)，送到每個子處理器 (slave) 中，主處理器再告訴子處理器，它需要對那兩個資料片做精鍊運算 (refinement operation)，子處理器則針對需要精鍊的兩顆四

元樹做運算。

依上述的演算法，子處理器所分配到是一對一對的四元樹，而不是單一的四元樹，若子處理器分配到的是單一的四元樹，由於四元樹在精鍊過程中是動態切割，但另一顆四元樹卻在其它的子處理器中，若要兩邊同時進行動態切割，會造成資料不一致，及溝通上的困難。故在[2,5]的論文中，將四元樹改為單一方向切割，並將雙向的能量交換，改為單一方向的能量傳輸，但如此一來，任一顆四元樹必須與其它 *n-1* 顆四元樹做運算，其運算個數為 *n(n-1)*。若子處理器以需要做精鍊運算的一對四元樹為單位分配，這一對四元樹可以在同一個子處理器中做動態切割，不會有不同處理器造成資料不一致、及溝通上的問題，且其總運算個數，由演算法得知為 *(n-1)+(n-2)+.....+1*，也就是 *n(n-1)/2*，運算個數較少。

我們知道任何兩顆四元樹在精鍊過程，是以動態切割的方式進行，運算結束後，這兩顆四元樹的關係會相當複雜，若要將兩顆四元樹及其之間的關係完整的傳回主處理器並不是很容易，而且大量的資料會造成網路的負擔，而降低執行效率。而我們利用四元樹的特性，可以在四元樹中的任何一個節點，給定唯一的編碼(見附錄 A)，在子處理器中只記錄兩四元樹中節點的位置，所屬的原始的資料片的編號，及其彼此間形式因子(form factor)的值。在精鍊運算全部結束之後，才將這些比較少量的資料傳回主處理器。主處理器可以利用簡單的運算找到兩顆四元樹節點正確的位置，再將彼此加入自己的相互影響的串列（interaction list）中。

## 2.2 資料分配

資料分配在平行演算法中佔了很重要的部份，由於我們無法事先得知資料的特性，且階層式輻射法的資料都是動態切割，所產生的資料也無法非常準確的預估，再加上各主機的運算能力、負載程度也都不盡相同，若無法將資料作適當的分配，勢必會造成某些主機負擔很重，某些主機閒置，這種負載不平衡（load imbalance）的情況發生。在本篇論文中，使用以下三種資料分配方式：

### 2.2.1靜態資料分配（一）

在階層式輻射法的環境中，各主機的運算能力不盡相同，資料的運算量也不同。所以我們必須把基本的靜態資料分配加以調整，首先根據處理器的運算能力，依比例分配各子處理器的資料量。

在資料運算量不同的方面，則是先將原始資料隨機重新排列一次，避免某些運算量大的資料集中在一起，再以輪流分配（Round-Robin）的方式送到各子處理器，希望使每一個子處理器中的資料運算量不會相差太大，以減低負載不平衡的情況。

### 2.2.2靜態資料分配（二）

在靜態資料分配方面，依上述的方式，雖然可以讓運算能力強的主機分配到較多對的資料片去處理，但資料本身是動態的切割，其運算的量不一定跟資料量成正比，所以我們也試著以資料本身的特性加以觀察，希望找到一個比較好的方式去分配資料，使各子處理器的使用效率達到最高。我們發現兩片資料之間所產生的能量交換的相互影響的連結愈多，其運算的時間愈長，而產生能量交換的相互影響的連結的數目與兩資料片的面積有關，彼此的面積愈大，所產生的連結數目愈多。因此，我們對能量交換的相互影響的連結數目加以評估，並考慮這個因素將資料分配。假設表面 $i$ 與表面 $j$，面積分別為 $A_i$ 、$A_j$，面積臨界值為 $A_{eps}$，能量交換的相互影響的連結會在兩四元樹的葉子節點產生，在最糟的情況下，四元樹所產生的葉子節點個數為 $4^{k1}$ 及 $4^{k2}$（$k1=\left\lfloor \log_4 \frac{A_i}{A_{eps}} \right\rfloor$，$k2=\left\lfloor \log_4 \frac{A_j}{A_{eps}} \right\rfloor$），因此，能量交換的相互影響的連結數目為 $4^{k1}*4^{k2}=4^{k1+k2}$，根據這項評估，希望使各子處理器所分配到的資料運算量能大致相同，使子處理器之間的負載平衡。

### 2.2.3動態資料分配

動態資料分配是由主處理器負責監控，當子處理器處理完目前的工作後，向主處理器發出要求，表示已經呈閒置狀態，主處理器再分派資料給子處理器。這種方式會使各處理器的使用率達到最高，也可以使運算能量較強的主機處理較多的資料，也不會因為資料的運算量不同，而使某些處理器負載過重，某些處理器閒置，這些負載不平衡的情況發生。

PVM 的環境為分散式的記憶體環境，主處理器與各子處理器都有自己的記憶體空間，故無法在程式中以存取共同變數，作為動態控制的依據，達到動態分配的結果。另外一

個方式是以訊息傳遞（message passing）作為彼此之間動態的控制，但 PVM 的訊息溝通模式為非同步阻斷傳送、接收（asynchronous blocking send & receive），也不適合以訊息傳遞的方式作為動態控制的依據，達到動態分配的結果。

而我們採取另一個方式，以共同磁碟空間（shared disk）中的檔案作為動態的控制，每一個子處理器有一個屬於自己的動態控制檔案。在主處理器方面，對於每一個子處理器的動態控制檔案只能讀取，不能寫入，以確保檔案內資料的一致性。當主處理器想要把資料傳給某一個子處理器時，主處理器要檢查子處理器的動態控制檔案內的值與自己記錄的值，判斷是否能送資料給該子處理器，若條件成立，則送資料給該子處理器，反之，則對下一個子處理器做相同的檢查。重複執行上述步驟到資料全部送完為止。在子處理器方面，可以對自己的動態控制檔案寫入，當子處理器完成一筆資料的運算後，將自己的動態控制檔案內的值加 1，當主處理器讀到該子處理器的動態控制檔案的值再與記錄值比較後，就可以再送一筆資料給該子處理器。

## 3. 實驗結果與分析

我們的實驗主機，主要以八台 SUN 的工作站為主，各主機的硬體狀況如下表所示，PVM 系統則是架構在這八台主機之上，以 FDDI 網路相互連接。

| 主機名稱 | 機型 | 記憶體大小 | CPU |
|---|---|---|---|
| Athena | Sun Ultra 2 | 262144 K | 200 MHz |
| Zeus | Sun Ultra 2 | 262144 K | 200 MHz |
| Muses | Sun Ultra 1 | 196608 K | 167 MHz |
| Ares | Sun Ultra 1 | 131072 K | 167 MHz |
| Iris | SUNW, SPARCStation-5 | 131072 K | 110 MHz |
| Bellona | SUNW, SPARCStation-5 | 131072 K | 110 MHz |
| Diana | SUNW, SPARCStation-5 | 131072 K | 110 MHz |
| Apollo | SUNW, SPARCStation-5 | 131072 K | 110 MHz |

在實驗中，我們以三種不同的資料分配方法，使用不同的處理器個數，對五個測試檔案(分別以 cube, simple, office, fulloffice1, fulloffice2 表示，請見附錄 B)測試它們的執行時間。方法一，只將資料隨機排序，以輪流的方式方配給各處理器。方法二，我們考慮每筆資料，評估它們可能會產生的能量交換的相互影響關係的數目，再將資料加以分配。方法一與方法二同屬於靜態資料分配。方法三，以動態資料分配的方式，去分配資料。

實驗結果顯示，當整個場景小時，也就是資料量少的時候，包含 cube、simple、office 三個場景，方法三的效果比方法一和方法二為佳（圖 1, 2, 3），但整個場景變大時，包含 fulloffice1、fulloffice2 兩個場景，方法三的效率明顯下降（圖 4, 5），當處理器愈多方法三的執行時間必沒有跟著減少。原因為方法三中以檔案來做為動態的控制，當場景小時，檔案的存取次數少，的確使各處理器的使用效率增加，效果最佳。但是當場景大時，檔案的存取次數急遽增加，雖然各處理器的使用效率仍然相當高，可是過多的 I/O 動作，卻使得整個執行時間增加，方法一和方法二的執行效果反而比方法三為佳。

圖 1: cube 執行時間圖

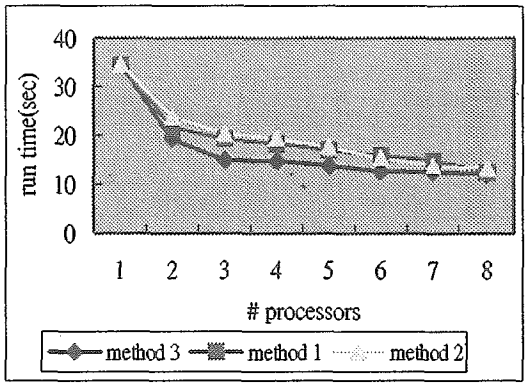
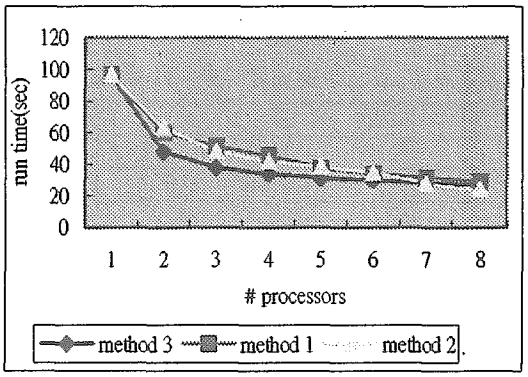



圖 2: simple 執行時間圖



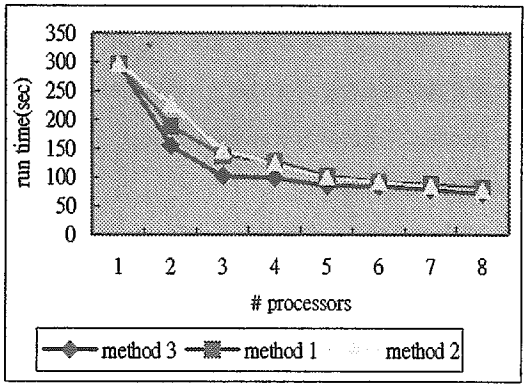



圖 3: office 執行時間圖

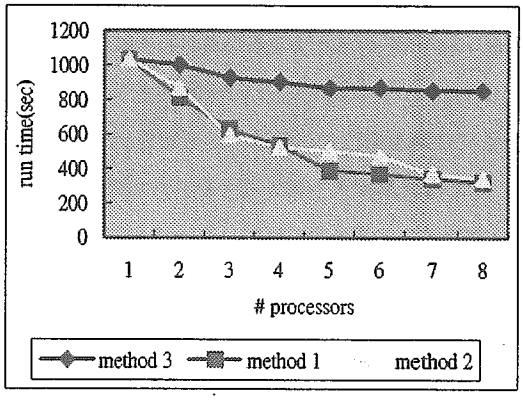



圖 4: fulloffice1 執行時間圖

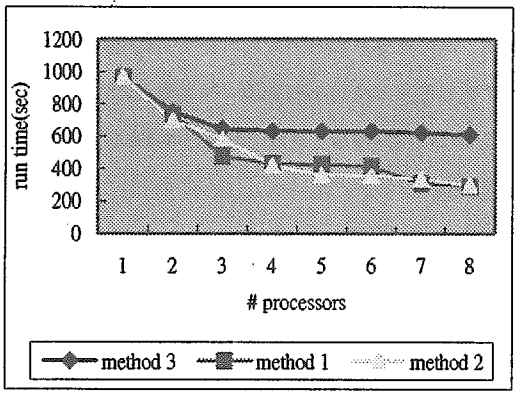



圖 5: fulloffice2 執行時間圖

我們從資料運算量的觀點去觀察，就方法一來看（表 1），我們以相同的資料量分配各處理器，但資料在處理的過程卻是動態產生，相同的資料量並不能保證其運算量會相同。就方法二來看（表 2），我們以面積評估了資料運算量，但實際上運算的量除了與面積有關外，也與整個場景的複雜度有關，所以產生的資料量不一定會與評估量成比例。方法三（表 3）為最有效能掌握資料的運算量。

| | Cube | | Simple | | Office | | Fulloffice1 | | Fulloffice2 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Pairs | Links | pairs | links | pairs | links | pairs | links | pairs | links |
| Athena | 13 | 18361 | 54 | 28146 | 186 | 31342 | 2694 | 36963 | 3846 | 65598 |
| Zeus | 12 | 15663 | 51 | 19204 | 186 | 17248 | 2694 | 24745 | 3845 | 56091 |
| Muses | 12 | 14455 | 51 | 35232 | 186 | 36605 | 2694 | 94321 | 3843 | 78558 |
| Ares | 12 | 3866 | 51 | 10949 | 184 | 35667 | 2693 | 70012 | 3843 | 38250 |
| Iris | 5 | 4650 | 18 | 7703 | 62 | 10314 | 898 | 28002 | 1282 | 34741 |
| Bellona | 4 | 3718 | 17 | 5289 | 62 | 12769 | 898 | 16445 | 1282 | 37999 |
| Diana | 4 | 250 | 17 | 5056 | 62 | 7402 | 898 | 17703 | 1281 | 31464 |
| Apollo | 4 | 5769 | 17 | 1080 | 62 | 10646 | 898 | 20365 | 1281 | 36952 |

表 1:方法一（pairs 分配的資料對、links 產生的連結數目）

| | Cube | | | Simple | | | Office | | |
|---|---|---|---|---|---|---|---|---|---|
| | Pairs | Link1 | Link2 | pairs | Link1 | Link2 | pairs | Link1 | Link2 |
| Athena | 4 | 196763 | 10521 | 72 | 280928 | 12821 | 279 | 255321 | 33719 |
| Zeus | 45 | 159986 | 9828 | 95 | 282848 | 18925 | 114 | 266433 | 32290 |
| Muses | 3 | 196608 | 13515 | 27 | 284928 | 23312 | 236 | 303464 | 28736 |
| Ares | 7 | 200704 | 13570 | 42 | 324960 | 29088 | 76 | 291712 | 29587 |
| Iris | 1 | 65536 | 4460 | 3 | 98304 | 6755 | 111 | 85107 | 11751 |
| Bellona | 1 | 35536 | 3430 | 2 | 131072 | 8713 | 92 | 85109 | 12785 |
| Diana | 4 | 68608 | 7183 | 19 | 91696 | 4239 | 28 | 85504 | 7345 |
| Apollo | 1 | 65536 | 5629 | 11 | 97280 | 8805 | 54 | 86562 | 5466 |

| | Fulloffice1 | | | Fulloffice2 | | |
|---|---|---|---|---|---|---|
| | Pairs | Link1 | Link2 | pairs | Link1 | Link2 |
| Athena | 3431 | 363104 | 53049 | 4959 | 409716 | 62624 |

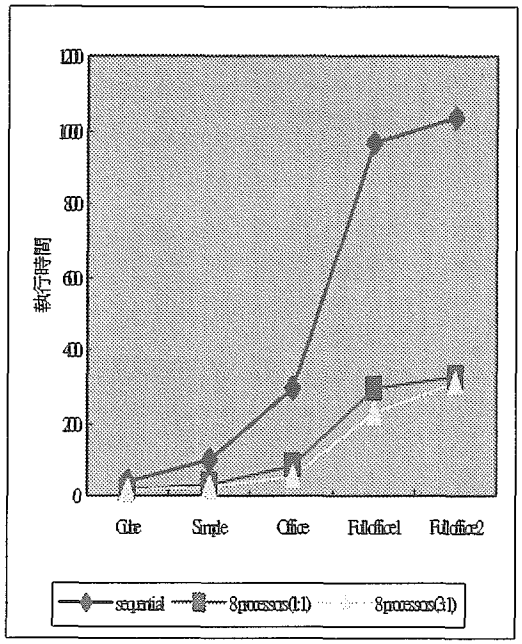
| | | | | | | |
|---|---|---|---|---|---|---|
| Zeus | 3101 | 363086 | 60550 | 3006 | 411033 | 72759 |
| Muses | 2191 | 374590 | 52902 | 3222 | 428484 | 61034 |
| Ares | 2414 | 374600 | 53028 | 4767 | 409737 | 64764 |
| Iris | 1303 | 121024 | 29776 | 1390 | 136555 | 33768 |
| Bellona | 734 | 121025 | 16686 | 12 | 137856 | 29451 |
| Diana | 1144 | 121024 | 15322 | 1598 | 136565 | 18752 |
| Apollo | 47 | 148880 | 28841 | 1549 | 136807 | 27154 |

表 2:方法二（pairs 分配的資料對、link1 預估會產生的連結數目、link2 產生的連結數目）

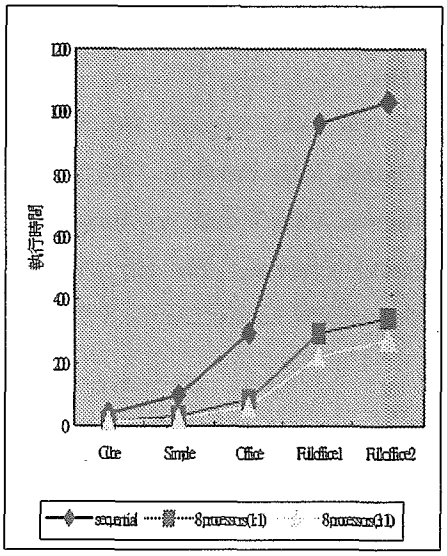
| | Cube | | Simple | | Office | | Fullofficel | | Fulloffice2 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Pairs | Links | pairs | links | pairs | links | pairs | links | pairs | links |
| Athena | 15 | 4516 | 54 | 19001 | 197 | 18735 | 1906 | 42873 | 2758 | 47582 |
| Zeus | 13 | 13644 | 49 | 23718 | 232 | 37563 | 2048 | 62877 | 2921 | 77322 |
| Muses | 6 | 16858 | 37 | 21801 | 145 | 28979 | 1799 | 43819 | 2672 | 67061 |
| Ares | 16 | 4367 | 34 | 13849 | 178 | 23520 | 2212 | 55323 | 3021 | 63073 |
| Iris | 1 | 4460 | 10 | 6755 | 36 | 8766 | 986 | 22868 | 1547 | 26968 |
| Bellona | 1 | 6607 | 17 | 9857 | 71 | 15456 | 1691 | 28751 | 2392 | 30980 |
| Diana | 12 | 7657 | 25 | 13885 | 90 | 10281 | 1934 | 26530 | 2418 | 35562 |
| Apollo | 2 | 5425 | 50 | 4756 | 41 | 17238 | 1789 | 28109 | 2774 | 24997 |

表 3:方法三（pairs 分配的資料對、links 產生的連結數目）

同時，我們又做另外一個實驗。將主機分為兩個群組 A 和 B， 群組 A 包含兩部 Sun Ultra 2 與兩部 Sun Ultra 1 的主機，群組 B 包含四部 SPARCStation-5 的主機，分別給這兩個群組的主機不同的資料量，測量其計算時間。當我們將 A、B 兩個群組分配不同的資料量 (3:1) 的時候，方法一以原始的資料對為分配單位，方法二以預估產生的連結為單位。




(1)




(2)

圖 6: (1)方法一 與 (2)方法二分配不同的資料量的執行時間圖(時間單位：秒)

不同的資料分配量，實驗結果，的確可以縮短執行的時間（圖 6）。

## 4. 結論與未來展望

在運算能力不同的主機下執行分散式演算法，影響

其效率的主要原因，仍然是負載的問題，雖然我們考慮了兩個表面的面積與其運算的關係，但是實際的運算量除了與面積有關外，也與兩表面的距離、位置、角度，兩表面之間是否有被阻隔，等幾何條件有關。若要評估整個幾何環境，勢必要花費相當長的時間去計算，也會影響整體的執行效率。希望將來能找到更好的評估方式，又不會影響執行效率。

我們的分散式演算法是以 Hanrahan[4]的階層式輻射法為基礎，但是階層式輻射法仍然需要 $O(n^2)$ 的時間和空間複雜度。若能將能量相互傳遞的單位從一片一片的表面，變成一些表面的集合。也就是說，將某一些比較小、比較近的表面群聚起來(clustering)，可以降低時間和空間複雜度，就有增加執行效率的可能。

參考文獻


[1] Michael F. Cohen and John R. Wallace. Radiosity and Realistic Image Synthesis. *AcademicPress*, Boston, 1993.

[2] C. C. Feng, Distrisbuted Hierarchical Radiosity with Large and Complex Scenes. *Master Thesis, Department of Computer Science, National Tsing Hua University*, Jun. 1996.

[3] Al Geist, Adam Beguelin, Jack Dongarra, Robert Manchek, Vaidy Sunderam. PVM : Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing. The *MIT Press*. Cambridge, Massachusetts. London, England. 1994.

[4] P. Hanrahan, Salzman, D., and Aupperle, L. A rapid hierarchical radiosity algorithm. *Computer Graphics (SIGGRAPH '91 Proceedings)* 25:4 (July 1991), pp. 197-206.

[5] Y. C. Pu, A Study on Distributed Hierarchical Radiosity. *Master Thesis, Department of Computer Science, National Tsing Hua University*, Jun. 1995.

[6] F rancois Sillion and Claude Puech. Radiosity and Global Illumination. *Morgan Kaufmann publishers*, San Francisco, 1994.


附錄

A. 四元樹的編碼

四元樹中的任何一個節點的編碼，可以用下列式子求得，關係如圖 A.1 所示：

position = my start position + 4 * (par position – par start position) + quadrant

其中

position ：代表該節點的位置編號。

my start position ：與該節點同一層的第一個節點的位置編號。

par position ：該節點的父親節點的位置編號。

par start position ： 與該節點的父親節點同一層的第一個節點的位置編號。

quadrant ：代表該節點是屬於那一個方向的節點(四個方向分別為東南、東北、西南、西北，以 0、1、2、3 表示)。

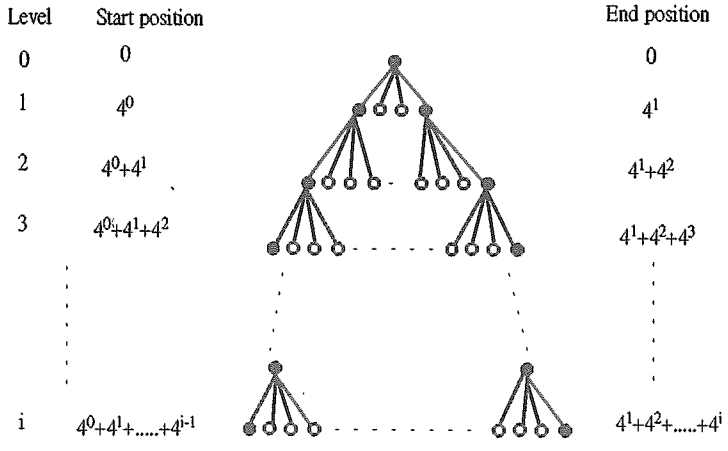



圖 A.1 ： 四元樹

B. 測試檔案資料

表 B.1 中為五個測試檔案的基本資料，包含資料片（patches）個數，及所產生的資料對個數（$n$ 個資料片，

會產生 $n(n-1)/2$ 筆資料對），預估會產生的連結（interaction link）數目，實際產生的連結數目。圖 B.1、圖 B.2、圖 B.3、圖 B.4、圖 B.5 為五個測試檔案繪出之圖形結果。

| | Cube | Simple | Office | Fullofficel | fulloffice2 |
|---|---|---|---|---|---|
| 資料片（patches） | 12 | 24 | 45 | 170 | 203 |
| 資料對（pairs） | 66 | 276 | 990 | 14365 | 20503 |
| 預估會產生的連結數目 | 1020144 | 1592016 | 1459212 | 1987333 | 2206753 |
| 實際產生的連結數目 | 68136 | 112658 | 161679 | 310154 | 371561 |

表 B.1：測試檔案的基本資料

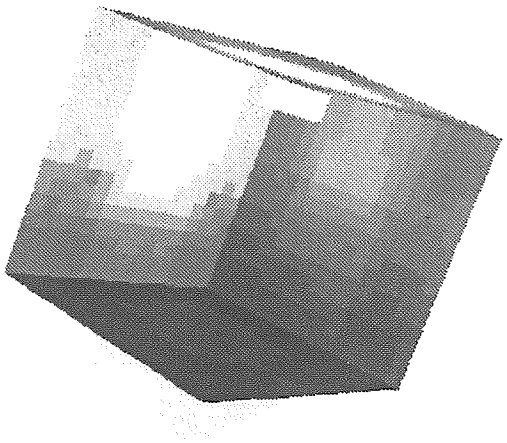


圖 B.1：cube

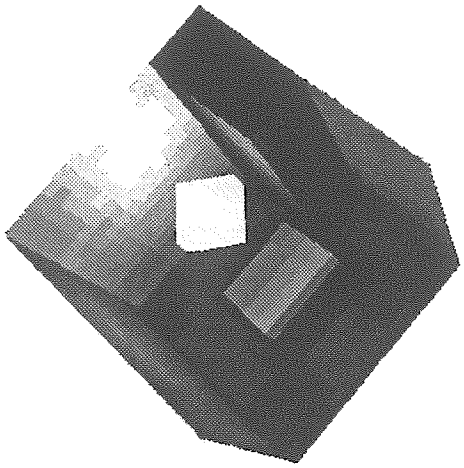


圖 B.2： simple

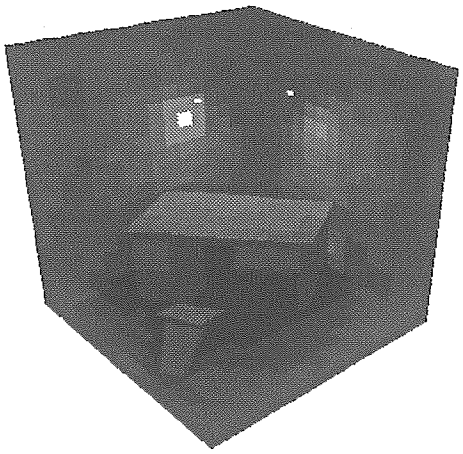


圖 B.3： office

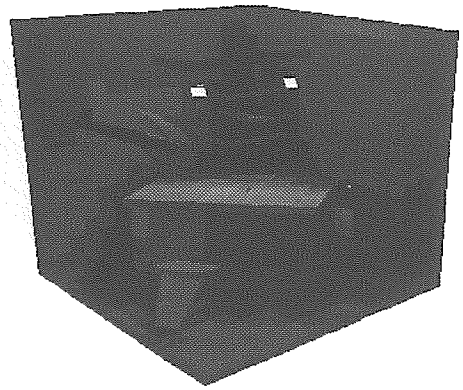


圖 B.4： fullofficel

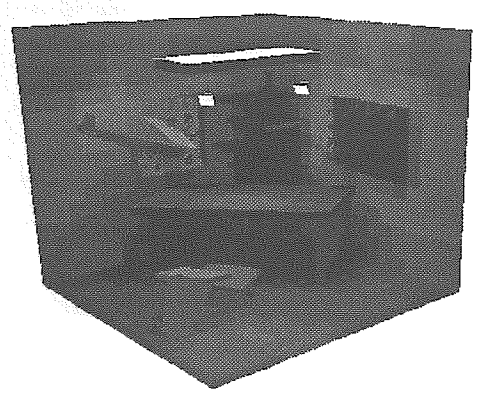


圖 B.5： fulloffice2