國立彰化師範大學資訊工程學系

碩士論文

基於表面體素顏色限制的 Recursive Interlocking Puzzles

Recursive Interlocking Puzzles for Colored 3D Models

研究生：羅凱威

指導教授：林炳賢　博士

中華民國一○五年七月

基於表面體素顏色限制的 Recursive Interlocking Puzzles

研究生：羅凱威　指導教授：林炳賢

國立彰化師範大學資訊工程學系

中文摘要

3D 體素(voxel)模型近來在遊戲有越來越受歡迎的趨勢，一般的體素模型在解出表面的顏色之後單純的作為遊戲模型或者樂高積木的拼圖使用。

傳統的立體拼圖不依靠電腦的話必須耗費專業人員大量的時間，利用 Recursive Interlocking Puzzles 的方法，我們可以得到一個擁有唯一解題順序的立體拼圖，但是使用的是一組表面無著色的體素模型。

本文提出了一種新的體素模型拼圖的樣式:使用已經渲染顏色的體素模型，加以分解並使其在保持原本色塊的條件下成為一組立體的拼圖。

關鍵字: Voxel, 3D puzzles (3D 立體拼圖), interlocking

Recursive Interlocking Puzzles for Colored 3D Models

Auther： Kai-Wei Luo Advisor：Ping-Hsien Lin

Department of Computer Science and Information Engineering

National Changhua University of Education,

Changhua,Taiwan,R.O.C.

Abstract

3D voxelized data are more and more popular in computer graphics society recently. They are widely applied in video-game, automatic LEGO model generation, or art.

Traditionally, the generations of 3D puzzles by hand are very time-comsumming processes. In SIGGRAPH 2012, Peng et. al [6], they developed an automatic technique to generate recursive interlocking puzzles from 3D voxel data with no color limitation.

In this thesis, we develop a new technique for generating recursive interlocking puzzles with color limitation on the envelope of the 3D voxel data. With this color limitation, we can generate more colorful, vivid, and satisfactory 3D puzzle result.

Keywords: Voxel, 3D puzzles, interlocking

目錄

封面.....1

中文摘要.....2

Abstract.....3

第一章　緒論.....6

第一節　研究背景與動機.....9

第二章　相關文獻探討.....12

體素化(Voxelization)模型.....12

立體拼圖模型.....13

第三章　研究方法.....18

第一節　概觀.....18

第二節　產生 Recursive Interlocking Puzzle 的條件.....20

第三節　解出$\boldsymbol{P1}$的步驟.....21

計算 Accessibility:.....22

選擇 seed voxel:.....24

找出$\boldsymbol{P1}$的擴張範圍(Expand Range):.....26

確定$\boldsymbol{P1}$只能由單一方向抽出:.....27

尋找剩餘方向的 blocking voxel.....30

隨機擴張$\boldsymbol{P1}$，使其符合最低 voxel 數.....31

確定$\boldsymbol{P1}$符合 local interlocking.....33

第四節　解出$\boldsymbol{Pi}$的步驟.....33

選擇 seed voxel.....34

決定$\boldsymbol{Pi}$的抽出方向.....35

找到$\boldsymbol{Pi}$的擴張範圍、顏色、目標.....36

設法令 seed voxel 可沿著方向抽出.....39

找出剩餘的 5 個 blocking voxel.....40

隨機擴張$\boldsymbol{Pi}$，使其符合最低 voxel 數.....40

確定$\boldsymbol{Pi}$符合 local interlocking.....41

第五節　終止條件.....41

第四章　結論.....42

參考文獻.....45

圖目錄

圖 1 美國藝術家 Mauricio Buitrago 的作品[1] 9

圖 2 格拉斯·柯普蘭（*Douglas Coupland*）的體素藝術作品[2] 10

圖 3 知名電玩 minecraft，當個創世神以體素描繪所有的地形場景[3].....10

圖 4 經過八元樹分割的模型示意圖[4].....11

圖 5 基於面積比例的體素化顏色計算技術[7]，左方為體素化之前的一個地球.....12

圖 6 榫卯結構示意圖.....13

圖 7 孔明鎖的解題順序.....13

圖 8 如果產生的拼圖如同陷入死結一般互相卡死等於是一個無法解決的拼圖.....14

圖 9 以 2D 表示，只有中間的圖可稱之為 interlocking puzzle[6] 15

圖 10 上方為 recursive interlocking，抽取順序為 P > Q > R。下方為 interlocking，P 抽出後 Q 與 R 都可同時抽出[6].....16

圖 11 Recursive Interlocking Puzzles 一文中的抽出順序[6] 16

圖 12 演算法概觀.....19

圖 13 一開始的模型，無顏色表示 inside voxel，有顏色表示 shell voxel.....22

圖 14 Accessibility 的分布，可以看出越接近暖色系的部分相鄰的 voxel 數越多.....23

圖 15 選擇 candidate set 的方式.....24

圖 16 seed voxel 的選擇，上方箭頭為$\boldsymbol{vp1}$ 25

圖 17 擴張範圍的選擇.....27

圖 18 選擇最接近的 Blocking pair.....29

圖 19 試圖連通 seed voxel 與 destination voxel 29

圖 20 令 shortest path 可以抽出 30

圖 21 找出 blocking voxel.....31

圖 22 隨機擴張$\boldsymbol{P1}$ 32

圖 23 圖中顯示了$\boldsymbol{Pi}$與$\boldsymbol{Ri-1}$的關係，紅色為$\boldsymbol{Pi}$，往右方(+X)抽出，其餘顏色為$\boldsymbol{Ri-1}$，白色代表 inside voxel，藍色與綠色為 shell voxel 的顏色 34

圖 24 選擇 candidate set 的方式，選取$\boldsymbol{Pi-1}$(紅色區塊)相鄰的做 candidate set(淺藍色區塊)，而紫色的 voxel 因為接觸面只有平行，所以不會加入。 35

圖 25 決定$\boldsymbol{Pi}$抽出方向，褐色為 seed voxel 36

圖 26 下方黃框的 voxel 是 seed voxel，為了向+Y 方向抽出，上方的 mark voxel(黃框部分)也必須加入$\boldsymbol{Pi}$，考慮到本論文的限制，mark voxel 自身必須要是相同顏色，或是 inside voxel(無色)才能加入 37

圖 27 以藍框表示$\boldsymbol{Pi}$的 expand range，左下方綠色的 voxel 因為無法抽出所以不加入.....37

圖 28 以 inside voxel 為 seed voxel 的情況下，下方的黃框為 seed voxel，上方為 mark voxel 38

圖 29 以藍框標示在 inside voxel 為 seed voxel 的情況下的擴張範圍.....38

圖 30 淺藍色框的為擴張範圍，黃色框為連通 seed voxel 與 mark voxel 之後再沿著抽出方向的 voxel，都被加入***Pi***之中 39

圖 31 褐色區域代表了 blocking voxel.....40

圖 32 黃色框的為最終擴張後的**Pi**.....41

圖 33 原始體素模型 解析度為 16*16*16.....42

圖 34 目前的結果，由上到下為 P1 到 P4.....43

第一章　緒論

第一節　研究背景與動機

隨著電腦科技的發展，藝術也多了各種不同的表現方式，像素藝術(pixel art)就是一種表達形式，如圖 1，不使用單純的畫筆或攝影方式，採用壓克力或木塊來拼貼圖形。

圖 1 美國藝術家 Mauricio Buitrago 的作品[1]

而隨著電腦 3D 技術的進步，開始出現另一種新的表達形式:體素藝術(voxel art)，使用樂高模型來完成一座座有如雕塑般的作品(如圖 2)，同時在電玩遊戲界(例如 minecraft，見圖 3)也非常受到歡迎。

圖 2 格拉斯·柯普蘭（*Douglas Coupland*）的體素藝術作品[2]

圖 3 知名電玩 minecraft，當個創世神以體素描繪所有的地形場景[3]

體素(voxel)概念上類似像素，是 3D 空間中的最小單位，將 3D 模型不採用網格與貼圖，而是用一個個的體素的顏色來表達一個模型。

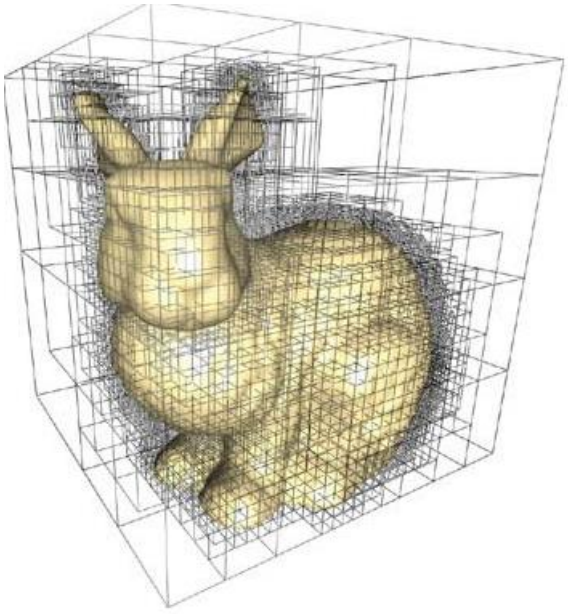

圖 4 經過八元樹分割的模型示意圖[4]

拼圖(Puzzle)在人類的歷史已經有數千年，三維的拼圖模型更是吸引我們的興趣，這個遊戲由來已久，但是卻很少有系統的提出方法來生成 3D 拼圖。

在 Coffin 的書中 [5]提到，即便是一名專業的設計人員，製作一個 Interlocking Puzzle 仍然需要花上數小時甚至數個月來完成。

2012 年推出的一篇論文 Recursive Interlocking Puzzles[6]提供了我們一個很好的方向去生成 3D 的體素拼圖，但是本身只能處理無表面顏色的體素模型，本文擴張了原本的方法，取已經渲染顏色的體素模型為基礎，進一步生成可以包含原先模型色彩的體素拼圖。

我們以此為發想，將原本已經著色的體素模型，連同 Recursive Interlocking Puzzles 提供的方法，進行研究並改良原本的立體拼圖。

第二章　相關文獻探討

體素化(Voxelization)模型

體素化是將幾何對象從連續的幾何表示形式轉換成一組最近似於連續對象的體素。體素化不只可將物體以體素展示，也可以應用在其他圖學研究，例如體積建模、虛擬醫療影像、觸覺渲染、碰撞測試以及三維空間分析等。

體素化(voxelization)的模型是我們再進行 puzzle 計算之前重要的資料，這裡我們採用前人[7]的體素化模型來進行，該篇文章以八元樹進行快速的體素化模型並可以在表面進行色彩的渲染，也因此只提供2^n解析度的體素模型。

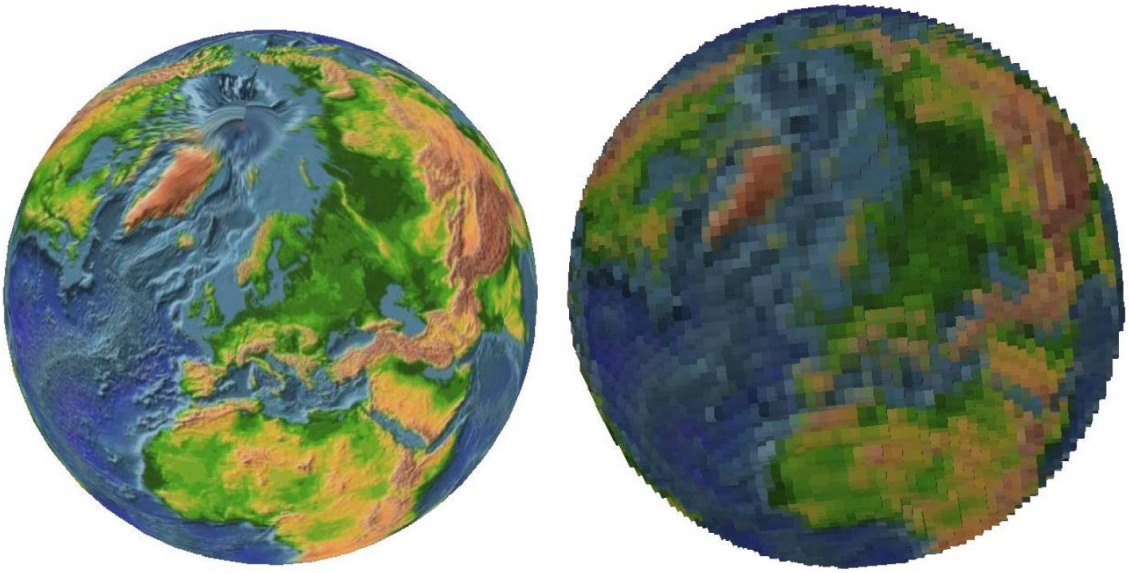

圖 5 基於面積比例的體素化顏色計算技術[7]，左方為體素化之前的一個地球

立體拼圖模型

在古時候的中國與日本，木造建築中時常見到榫卯結構的構造，這個構造的神奇之處在於不費任何接合劑或是釘子就能輕易的把兩塊建築構造接合再一起而不會輕易脫落。

圖 6 榫卯結構示意圖

使用這種結構所製造出來的立體拼圖相當的穩定，同時也出現在許多古玩上，比如孔明鎖、六子連方等。

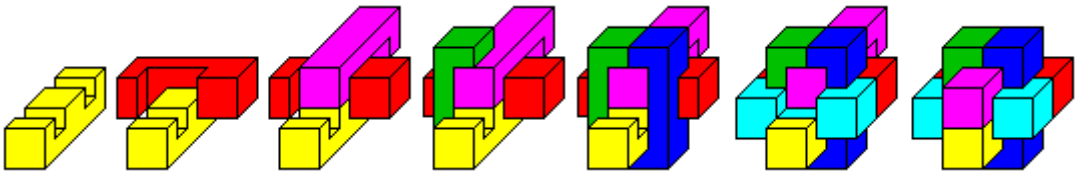

圖 7 孔明鎖的解題順序

目前以人腦目前的能力要設計出這些結構仍然相當困難[5]，這也是為什麼這類的玩具目前的種類不多，也因此，設計並自動隨機的產生出這些立體拼圖才會如此有趣。

由[4]所給予的 Interlocking Puzzles 的定義如下:

「任何一個可以組合的(assembly)立體拼圖(必須至少 3 塊以上)若要稱之為 interlocking，必須有而且只能有一個可以抽出的關鍵拼圖(key piece)，而其餘的拼圖(以及這些拼圖所有的子集合)則必須在關鍵拼圖抽出之前保持不會脫落的狀態(immobilize)」。

由此定義我們可以窺探這個問題的困難度:首先，單一拼圖不會脫落(immobilize)並無法確保此拼圖的母集合不會脫落，其次，每當生成一塊新的拼圖的時候都必須確保該拼圖在前一塊關鍵拼圖(key piece)抽出之後一定可抽出，同時又不會因為生成死結(deadlock)或循環等待(Circular waiting)的情況。

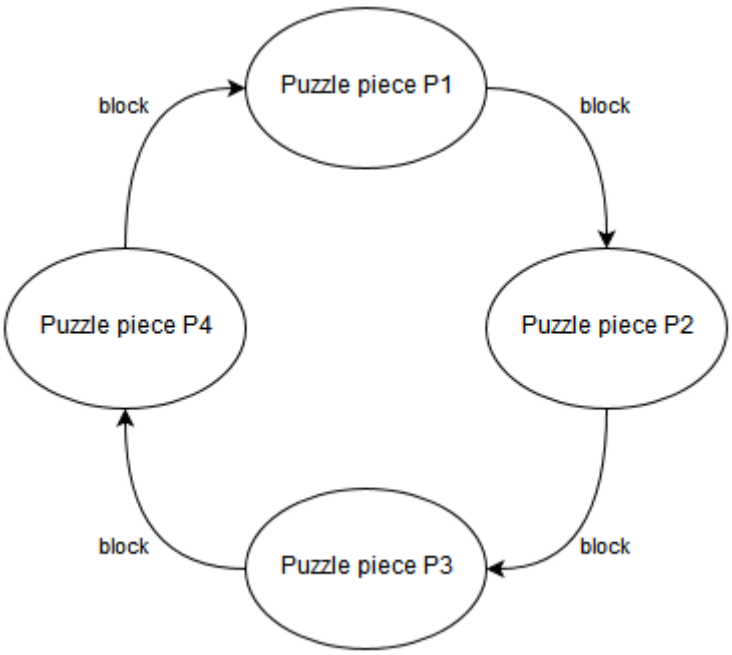

圖 8 如果產生的拼圖如同陷入死結一般互相卡死等於是一個無法解決的拼圖

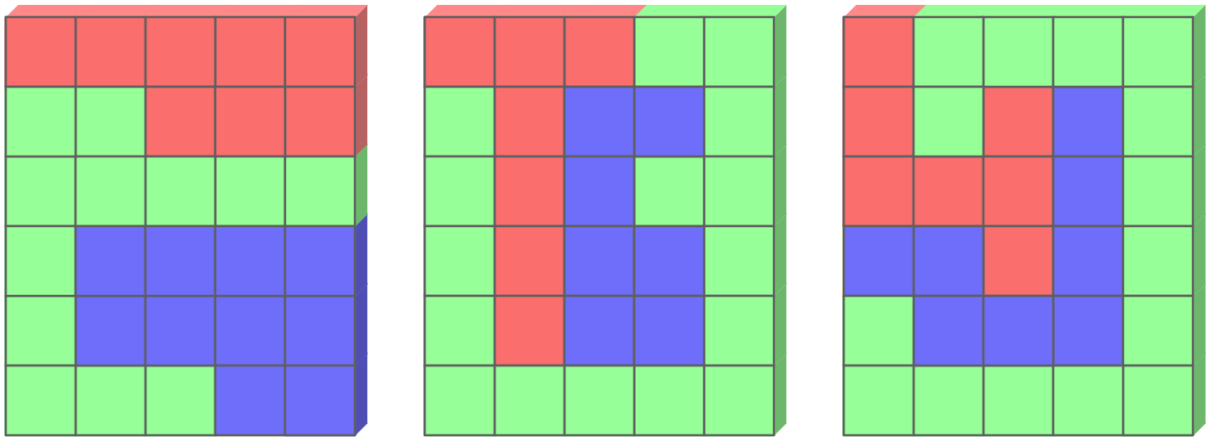

圖 9 以 2D 表示，只有中間的圖可稱之為 interlocking puzzle[6]

因此，為了減化問題，[4]提出了一個簡化後的題目:我們將 puzzle 想成只擁有單一解法，每次都只能抽出一塊拼圖作為關鍵拼圖(key piece)，而未被抽出的拼圖則必須保持不動。

如此一來，考慮一個以拼圖形成的有序集合$\{P_1, P_2, \dots, P_n\}$，其中P_i表示第 i 塊拼圖，其中 i 的大小表示抽出的順序。

只要我們能確定(1) P_1可以抽出，$\{P_2, \dots, P_n\}$在此之前保持不動(2)當P_i可以抽出，$\{P_i, P_{i+1}, \{P_{i+2}, \dots, P_n\}\}$之間形成 interlocking 的關係。我們就能確定$\{P_1, P_2, \dots, P_n\}$之間為 interlocking。

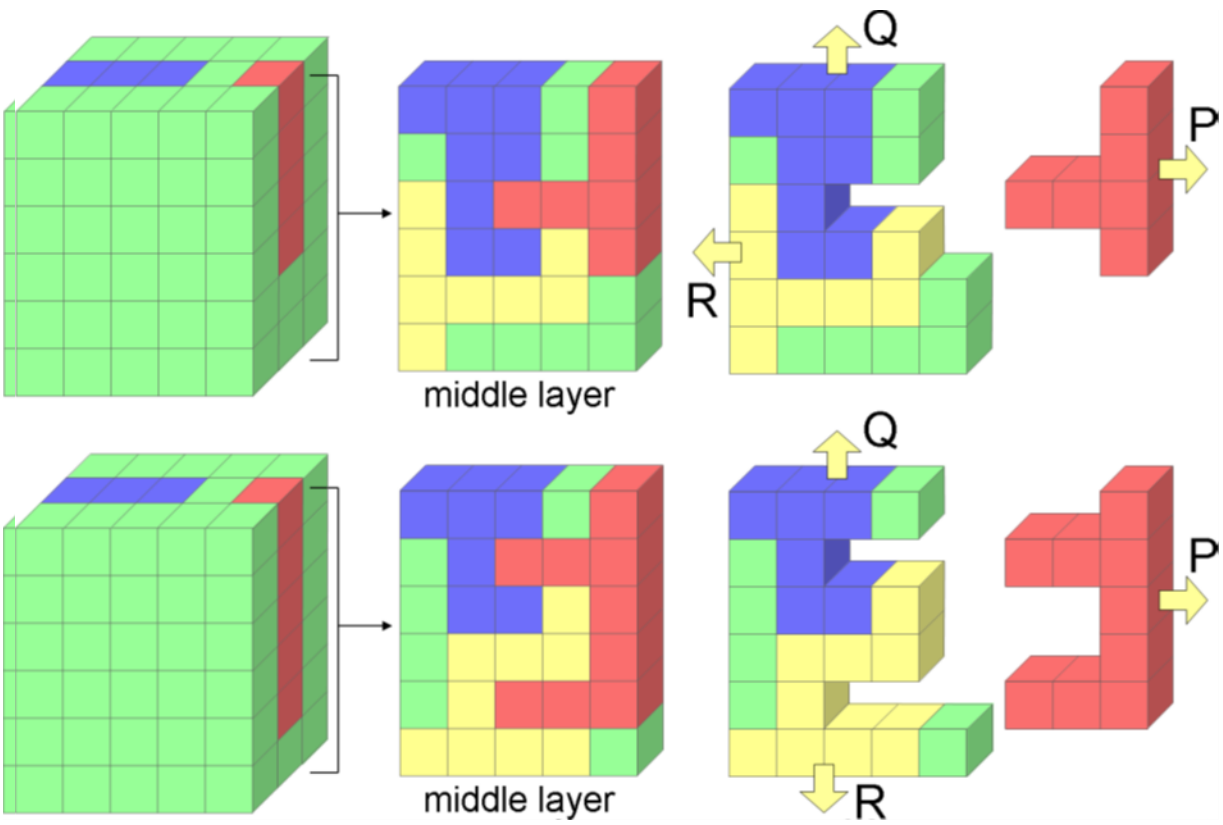

圖 10 上方為 recursive interlocking，抽取順序為 P > Q > R。下方為 interlocking，P 抽出後 Q 與 R 都可同時抽出[6]

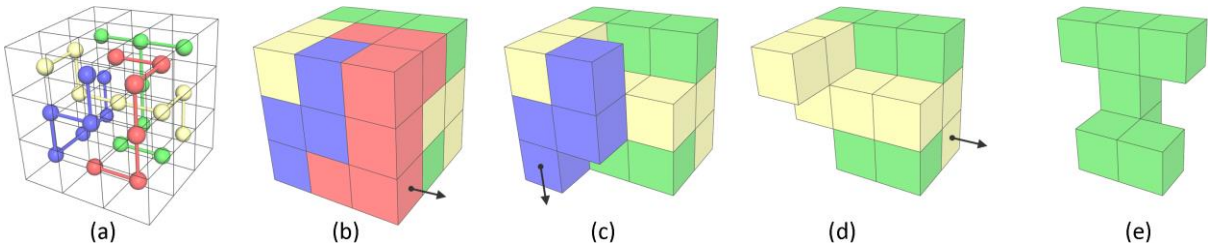

圖 11 Recursive Interlocking Puzzles 一文中的抽出順序[6]

有幾個性質我們會在接下來的內容中經常提到:

Shell voxel:

每個體素化的模型都有分為內部與外部，基本上體素化只會保存與網格有相交的體素，並記錄該體素的顏色。

Inside voxel:

位在模型內部的體素，一般體素化的過程都不會保留這部分的資訊，這部分的體素沒有顏色，而且在六鄰域內只會相鄰 inside voxel 或 shell voxel。

Connected set(連通集):

一個集合如果要稱為連通集，代表當中任意兩個元素都有一條路徑連通。

用在以體素所形成的集合也是一樣，一個由體素所形成的聯通集裡，任意兩塊體素都應該有路徑可以連通。

要判斷一個體素模型是否連通，我們只要以廣度優先搜尋(BFS)來判定是否有未曾拜訪過的體素即可。

第三章　研究方法

第一節　概觀

首先我們定義整個體素(voxel)模型的集合為 S，而每塊 puzzle piece 所包含的體素(voxel)集合稱為P_i $(i \in N)$，因為我們生成的 puzzle piece 屬於遞迴解法，因此按照生成的順序可以得到$P_1, P_2, P_3, \dots, P_i$。

每當我們生成P_i時，都會自 S 中取走一些剩餘的體素(voxel)，稱之為R_i 。

我們的目標就是由 S 開始，依照順序產生$P_1, P_2, P_3, \dots, P_i$，直到我們產生的 puzzle piece 數目達到目標，或者體素(voxel)都用完為止。

$$S \rightarrow [P_1, R_1] \rightarrow [P_1, P_2, R_2] \rightarrow \cdots \rightarrow [P_1, P_2, \dots, P_n, R_n]$$

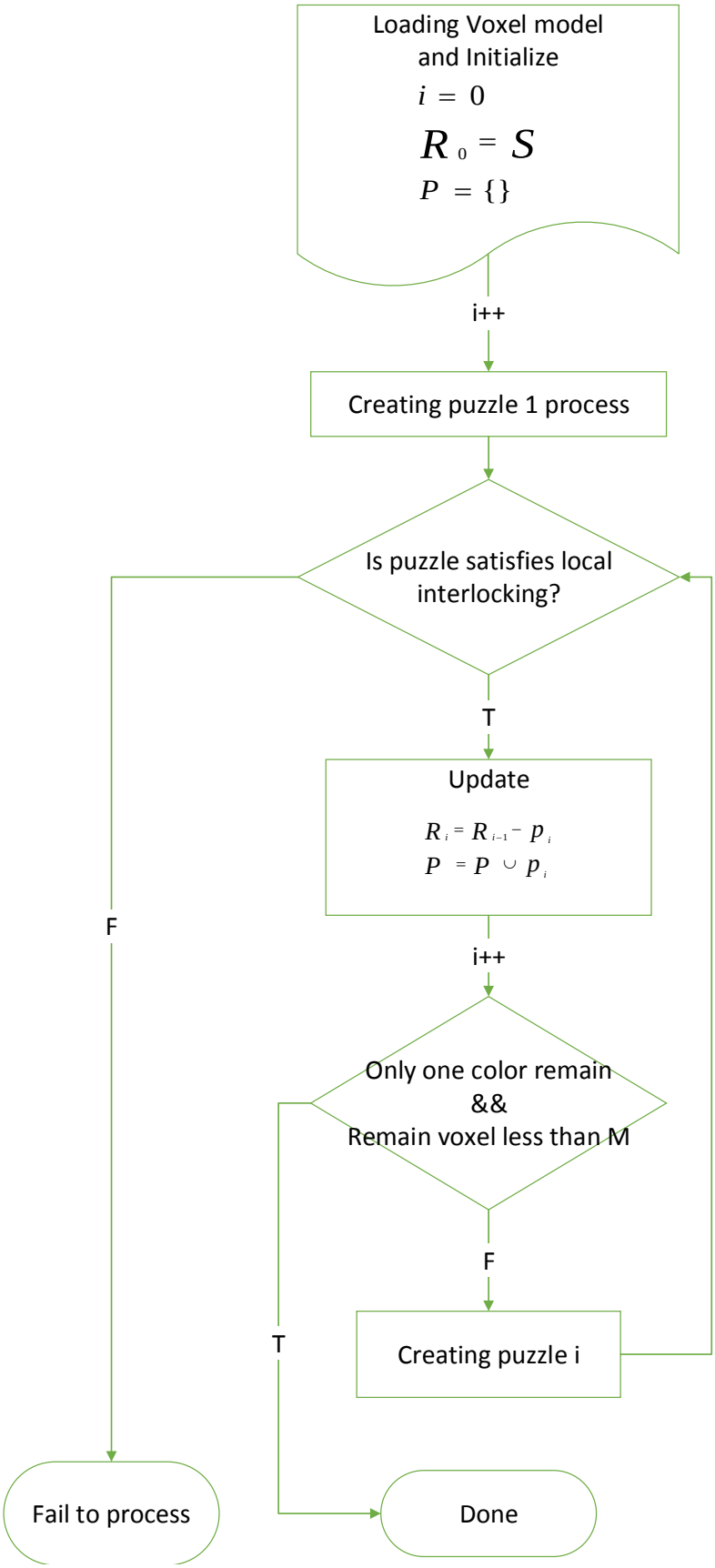
Loading Voxel model
and Initialize
$i = 0$
$R_0 = S$
$P = \{\}$
i++
Creating puzzle 1 process
Is puzzle satisfies local interlocking?
T
Update
$R_i = R_{i-1} - p_i$
$P = P \cup p_i$
i++
Only one color remain
&&
Remain voxel less than M
F
T
Creating puzzle i
F
Fail to process
Done

圖 12 演算法概觀

如圖 12 所示，我們的流程一開始先讀取體素模型，並同時初始化(也就是把剩餘的 voxel 集合指定為所有的體素)。接著，我們首先產生第一塊拼圖(詳見第三節　解出$\boldsymbol{P_1}$的步驟)。

由於P_1是隨機產生，因此我們會先檢查P_1是否符合我們的需求(詳見第二節　產生 Recursive Interlocking Puzzle 的條件，第一部分: 產生P_1的條件)。如果成功，則我們會更新剩餘的體素集合R_1，以及紀錄P_1的體素。

我們試圖讓每一塊拼圖盡量接近一樣的大小，因此我們設定每一塊拼圖的體素數目為 M，因此，當剩餘的體素小於 M，我們就終止產生新的拼圖並當作最後一塊拼圖(詳見第五節　終止條件終止條件)。

我們會再尚未滿足終止條件之前不斷的產生新的拼圖(詳見 3.4 解出P_i的步驟)，並且在每塊拼圖產生成功之後更新剩餘的體素。

第二節　產生 Recursive Interlocking Puzzle 的條件

每次產生一塊 puzzle 的時候，我們都要檢查此塊 puzzle piece 本身是否符合 local interlocking 的條件，當最後生成的所有 puzzle piece 都符合 local interlocking，我們就可以確定 puzzle 本身是有解並且成功。

- 產生$\boldsymbol{P_1}$的條件:

當我們得到一塊 3D 體素模型，從 S 中取出P_1並得到R_1的時候必須符合以下條件。

第一，我們可以將P_1從R_1中以單方向取出，並且$[P_1, R_1]$只有兩塊。這個條件確保了P_1一定能一開始成功取出。

第二，P_1一定只能從單方向單一步驟取出，如果P_1能往兩個方向以上取出，可能會造成整塊 puzzle 容易散落或者不穩。

第三，P_1與R_1皆為聯通集(connected)。如果我們不能確保聯通，當P_1取出後可能會造成 puzzle 本身散開，違反了 interlock 的特性。

- 產生$\boldsymbol{P_i}$的條件($\boldsymbol{i > 1}$):

當P_1產生之後，接下來的每塊 puzzle piece 都要符合以下條件。

第一，產生P_i之後，必須確認P_i本身在P_{i-1}抽出之前不可自$\{P_{i-1}, P_i, R_i\}$抽出。也就是說(i) P_i在$[P_{i-1}, P_i, R_i]$之中無法移動而且(ii) P_{i-1}本身不可和P_i同時抽出。

第一，在P_{i-1}抽出之後，P_i只能以單方向自R_i取出。

第二，P_i與R_i皆為聯通集。

第三節　解出$\boldsymbol{P_1}$的步驟

作為第一塊被抽出的 puzzle piece，P_1的作法擁有指標性的意義。在這裡所使用的觀念基本上在後面對P_i的產生方式都會使用到。

要產生一塊 puzzle piece，首先我們會先挑選一塊 voxel，接著以此塊 puzzle 為基底，陸續擴張直到我們符合第三章第二節所需的條件為止，最後利用隨機擴張此塊基底的方式，隨機產生出符合我們需求的 puzzle piece。

這裡我們定義對P_1的抽出方向為v_{p_1}，考慮到作為第一塊 puzzle piece 必須要容易抽出，我們令$v_{p_1} = +Y$，也就是從模型正上方(Y 軸)抽出。

圖 13 表示了我們的體素模型，這裡以 2D 方式表示，一個著色的體素模型只有最外側的 shell voxel 具有顏色，內部的 inside voxel 則以白色表示，inside voxel 並不具有顏色，只有在被納入 puzzle piece 的時候才會改變它的顏色。圖片的下方模型我們略過不表示。

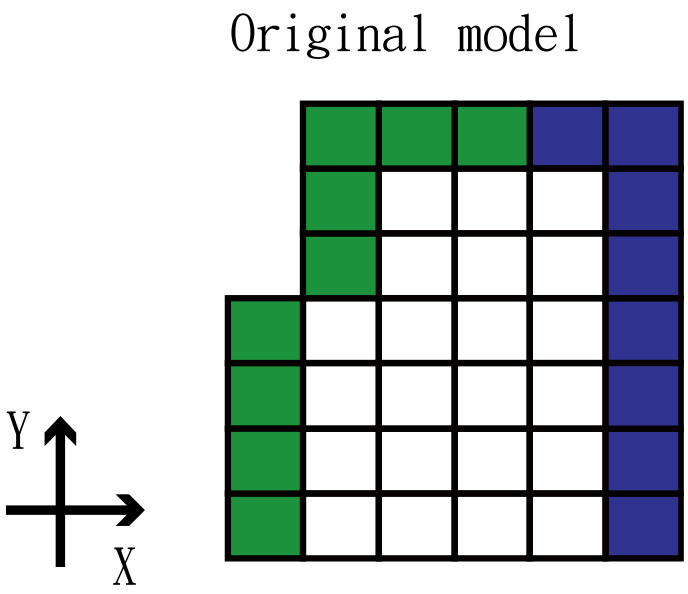

圖 13 一開始的模型，無顏色表示 inside voxel，有顏色表示 shell voxel.

計算 Accessibility:

自R_i取出 puzzle piece 的過程中有可能會導致R_{i+1}不連通(unconnected)，導致整個模型破碎化(fragmentation)。

因此，考慮一個 voxel 自R_i取走後是否會導致破碎化可以用 Accessibility 表示:

$$a_j(x) = \begin{cases} number\ of\ neighbors\ of\ x, for\ j = 0. \\ a_{j-1}(x) + 0.1^j \sum_i a_{j-1}(y_i(x)), for\ j > 0 \end{cases}$$

$a_j(x)$稱為體素(voxel) x 自R_i中取出的 accessibility。

Accessibility 與一個 voxel 周遭一定範圍內相鄰的 voxel 數目成正比，$a_j(x)$越低表示周圍的 voxel 越少，其中下標的 j 表示 x 的相鄰距離。

Accessibility 是一個遞迴運算所求得的值，這裡我們通常計算一個 voxel 周遭距離 3(也就是該 voxel 的鄰居的鄰居)以內的 voxel 數，越遠影響的越小，而 j 的最大值我們只取到 3。

當一個 voxel 自R_i中被選中，Accessibility 越低的 voxel 比起 Accessibility 高的 voxel 比較不會導致R_{i+1}的破碎化(fragmentation)。

我們也可以這麼說，當R_i的 Accessibility 總和越低，代表R_i的形狀越不平整。

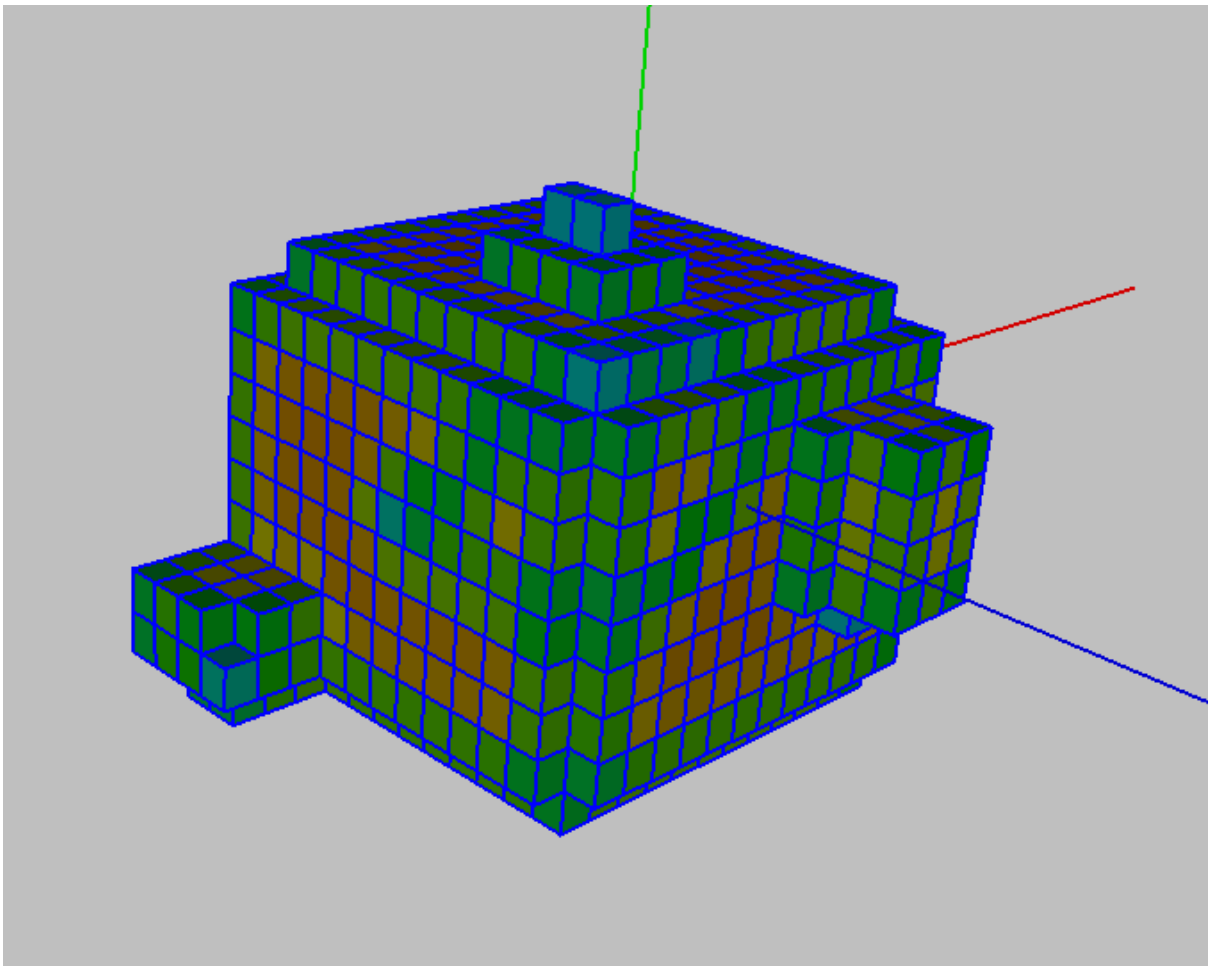

圖 14 Accessibility 的分布，可以看出越接近暖色系的部分相鄰的 voxel 數越多

選擇 seed voxel:

我們要生成 puzzle piece 的時候會選擇一個 voxel 作為生長的起點，稱之為 seed voxel，接著才會繼續沿著 seed voxel 擴張成一塊完整的 puzzle piece。

我們選擇 seed voxel 並非只選擇某一塊，而是選擇一組R_i中符合 seed voxel 條件的 voxel 做候選，稱之為 candidate set。

作為P_1的 seed voxel(假定座標為$\{x, y, z\}$)，因為我們設定自上方抽出，因此選擇模型中只有正上方(+Y)以及垂直正上方的某一面(±X, ±Z)對外的 voxel(只有兩面對外)。

作為第一塊的 puzzle piece，除了要可以往上抽出之外，考慮到至少應該要有一個面可以方便提供摩擦力，因此選擇有兩個面的作為 candidate set。

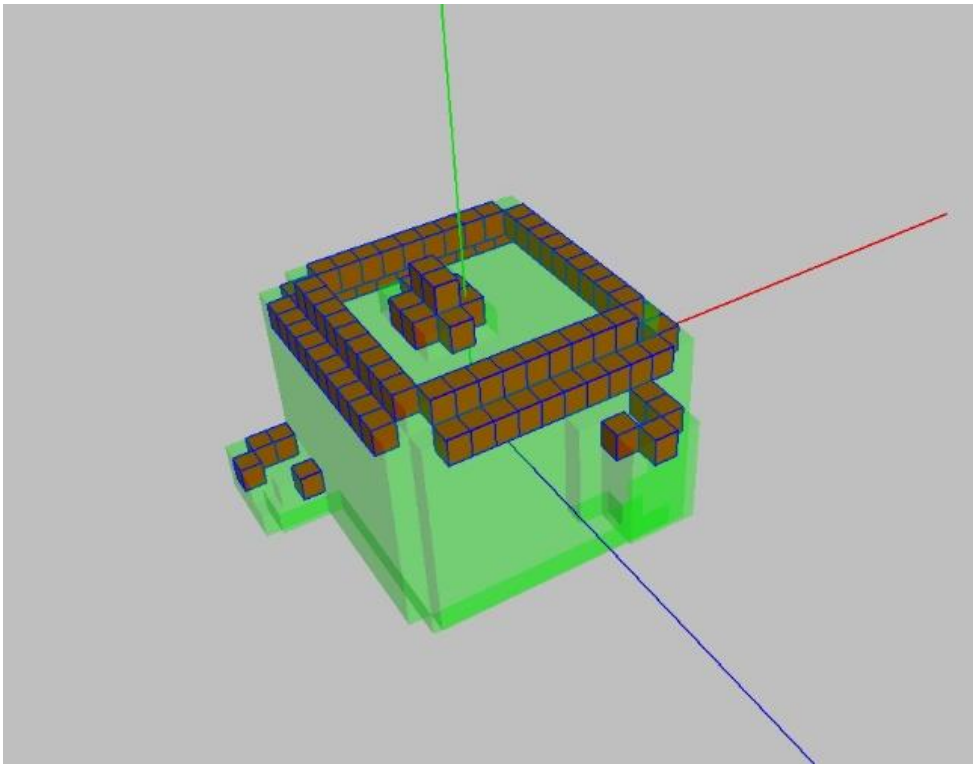

圖 15 選擇 candidate set 的方式

我們會持續的不斷選擇 candidate set 中其中一塊作為 seed voxel(生長的基點)直到該次 puzzle piece 成功為止。

因為P_1的 candidate set 都是從 shell voxel 之中取出，所以P_1的顏色定為 seed voxel 的顏色(圖 13 的例子，P_1為綠色)。

如圖 16 所示，左側紅色的 voxel 為 seed voxel，為了方便標示，被加入 puzzle piece 的 voxel 一律以紅色標示，關於 voxel 原始的顏色可以參考圖 13。因為P_1選擇+Y 為抽出方向，同時 seed voxel 有一個面向外側的面-X 作為抽出P_1時提供摩擦力的表面。

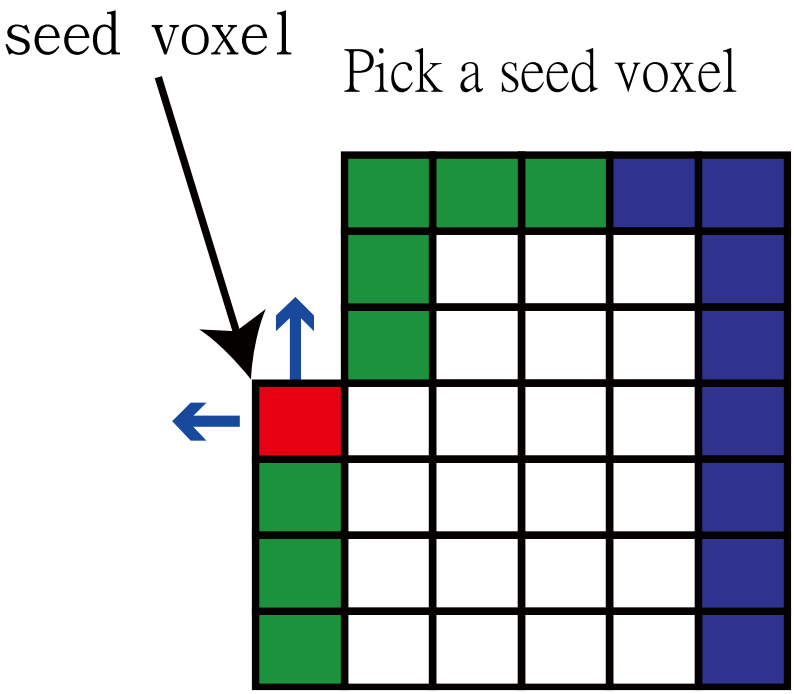

圖 16 seed voxel 的選擇，上方箭頭為v_{p_1}

找出P_1的擴張範圍(Expand Range):

考慮到我們的模型表面具有顏色的限制，並非所有R_i中的 voxel 都能夠選為 puzzle piece，因此在確認抽出的方向以及 puzzle piece 的顏色後要先標示出擴張範圍。

每塊擴張範圍都需符合以下條件: (i)本身必須為 connected set (ii)如果是 shell voxel 則必須與 puzzle piece 的顏色相同 (iii)每個 voxel 都必須確保能沿著 puzzle piece 的預訂方向抽出。

為了符合條件，我們取擴張範圍的時候先找出所有R_i中可沿著方向抽出的同色 voxel，然後以 seed voxel 為起點做 BFS 確保擴張範圍內每個 voxel 都聯通(如圖 17 所示)。

圖 17 表示了一塊P_1的擴張範圍，以綠色表示，可以看到如果每個 voxel 沿著抽出方向到達表面都會與P_1顏色相同(綠色)，而且同時擴張範圍本身為一個連通集(每個 voxel 都有路徑相通)。

Find expand range

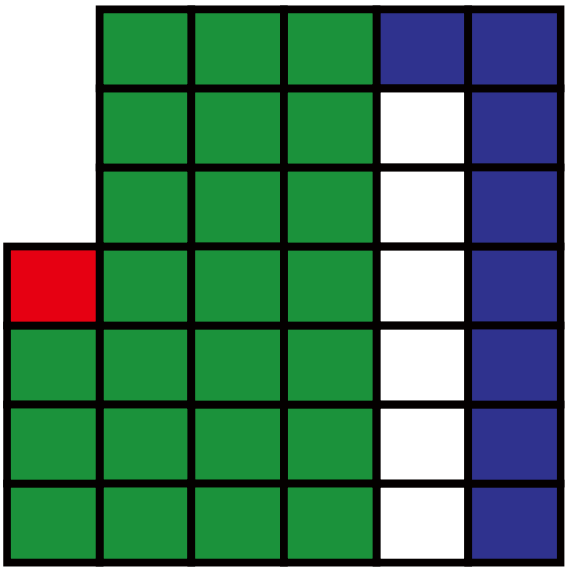

圖 17 擴張範圍的選擇

確定P_1只能由單一方向抽出:

到目前為止，我們的P_1只選擇了 seed voxel，確保了P_1的抽出方向v_{p_1}、顏色以及P_1的擴張範圍後，接下來要擴張P_1，並且確保可以使其往v_{p_1}抽出。

我們可以透過以下步驟來達成:

1. 有鑑於先前我們選擇 seed voxel 的時候選擇擁有兩個對外的面有兩個，除了v_{p_1}的方向稱之為v_n(見圖 16)。
2. Seed voxel 在v_n的方向上有被抽出的風險，因此我們沿著v_n的方向尋找 **blocking pair**，blocking pair 是兩個相鄰的 voxel，順著v_n方向的稱為 blocking voxel，逆向v_n的稱為 destination voxel。

   圖 18 左方顯示了一塊 blocking pair(黃色圈選的部分)，因為v_n是-X 方向，所以我們選擇 X 軸的 blocking pair。

要確保$\boldsymbol{P_1}$在v_n的方向上不能被抽出，我們必須確定$\boldsymbol{P_1}$的 voxel 中至少有一個 voxel 不能往v_n移動，也就是說該 voxel 在v_n的方向相鄰R_1。

Blocking pair 代表的就是一組即將加入$\boldsymbol{P_1}$的 voxel(destination voxel)以及作為阻擋$\boldsymbol{P_1}$往v_n前進的 voxel(blocking voxel)。

圖 18 右方表示了 blocking voxel(左)以及 destination voxel(右)的位置。

我們利用廣度優先搜尋(BFS)的方式，以 seed voxel 為起點，expand range 的範圍內搜尋最鄰近的 blocking pair。

3. 接下來，利用最短路徑的方式，我們將 seed voxel 連到 blocking pair 中的 destination voxel，把這條路徑上的所有 voxel 加入$\boldsymbol{P_1}$，而該路徑的 voxel 也都必須在 expand range 範圍內，同時也不能跨過 blocking voxel。

   圖 19 表示了我們連接 seed voxel 到 destination voxel 的路徑，被加入$\boldsymbol{P_1}$的 voxel 以紅色表示

4. 接著，為了確保目前路徑上的 voxel 都能沿著v_{p_1}抽出，路徑上的 voxel 沿著v_{p_1}方向的所有 voxel 都跟著加入$\boldsymbol{P_1}$。事實上，所有準備加入$\boldsymbol{P_1}$的 voxel 在確認以後上方(+Y)的所有 voxel 都必須加入$\boldsymbol{P_1}$才能確保該 voxel 可以抽出。(圖 20)

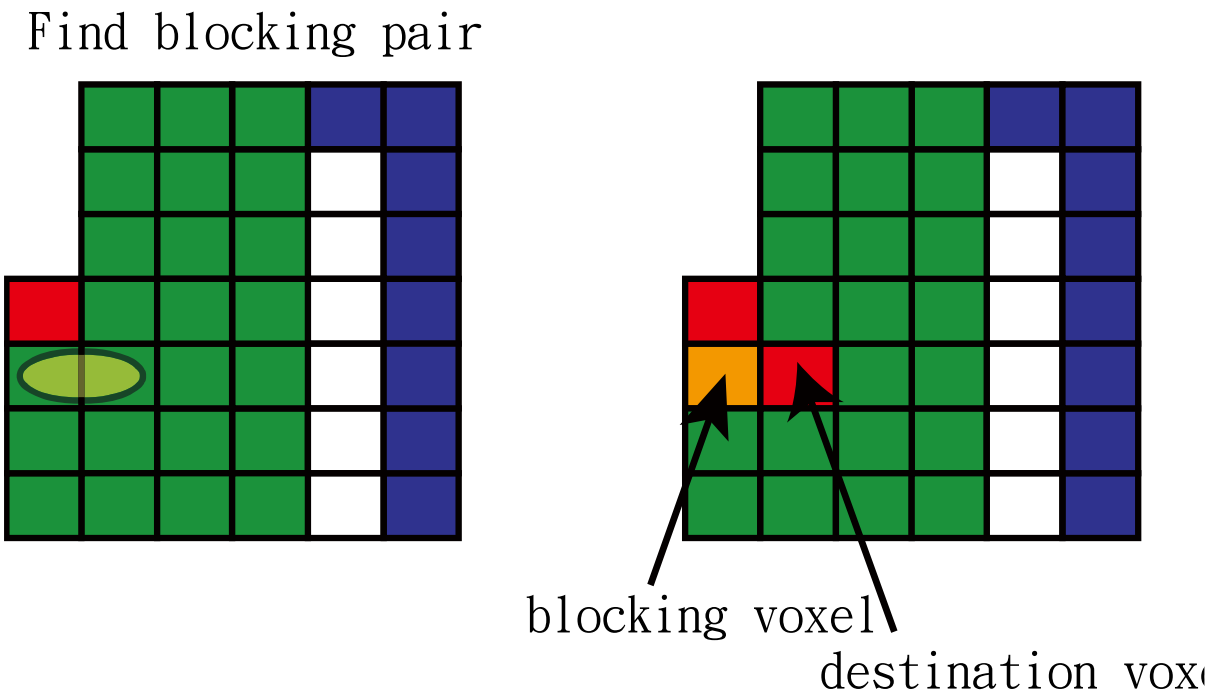

圖 18 選擇最接近的 Blocking pair

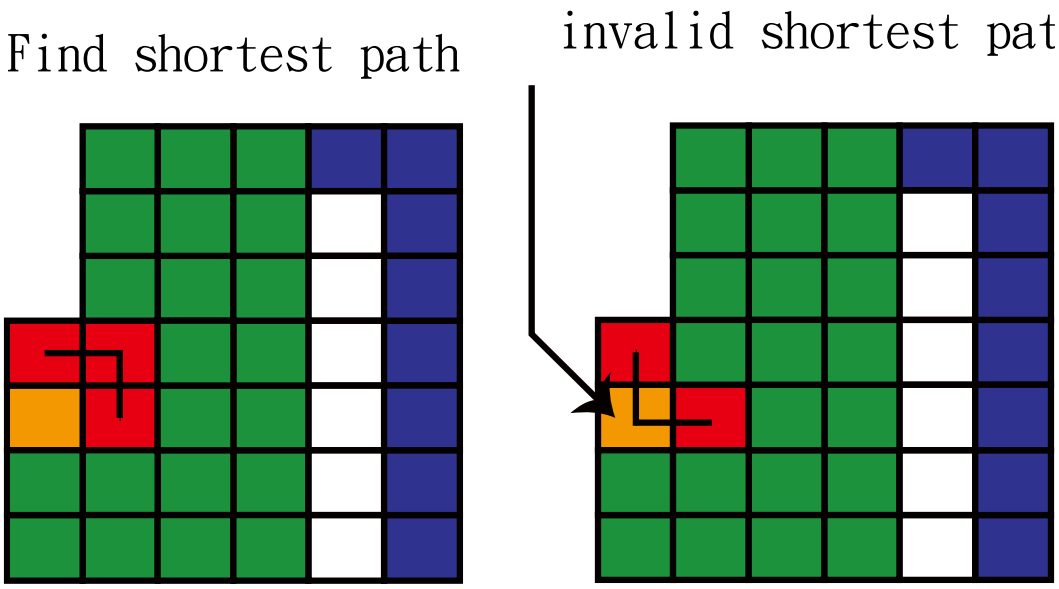

圖 19 試圖連通 seed voxel 與 destination voxel

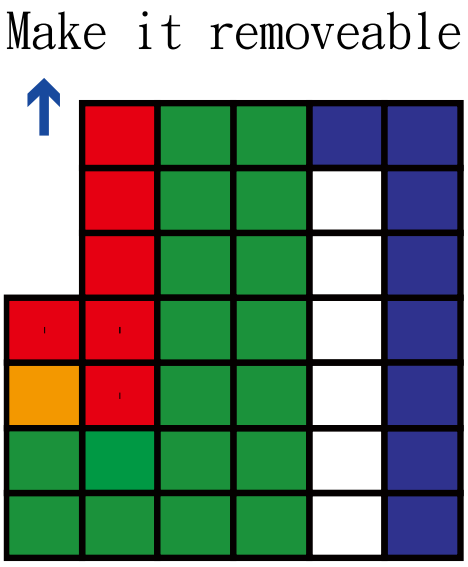

圖 20 令 shortest path 可以抽出

尋找剩餘方向的 blocking voxel

目前為止我們確保了P_1無法自v_n抽出，要確保P_1一定只能從v_{p_1}抽出，我們必須尋找剩餘 4 個方向的 blocking voxel。

blocking voxel 的概念在於:如果想要確保任一塊 puzzle piece 在某一方向不能移出，我們只要確保在該方向(v)至少要有一塊屬於R_i的 voxel 不會被加入P_i即可，該 voxel 因為不在P_i中，所以P_i要沿著方向前進的時候會被卡住。

選定了一塊 blocking voxel 之後，blocking voxel 以下(反向v_{p_i}的方向)的 voxel 便無法被抽出。

在選擇 blocking voxel 的時候要注意一點，如果無法確保 puzzle 與 blocking voxel 位在同一軸線上，就不能保證移動的時候會被 block 住。

如果要尋找P_1剩餘的 4 個 blocking voxel，我們可以簡單的尋找以 seed voxel 為中心，沿著要阻擋的方向的最遠方的 voxel 即可。

圖 21 標示了 blocking voxel，以黃色標示，可以看到在P_1應該要阻擋的方向中(+X,-X 及-Y)，以 seed voxel 為中心延伸出去都有 blocking voxel，-Y 方向與-X 方向的 blocking voxel 同一個。

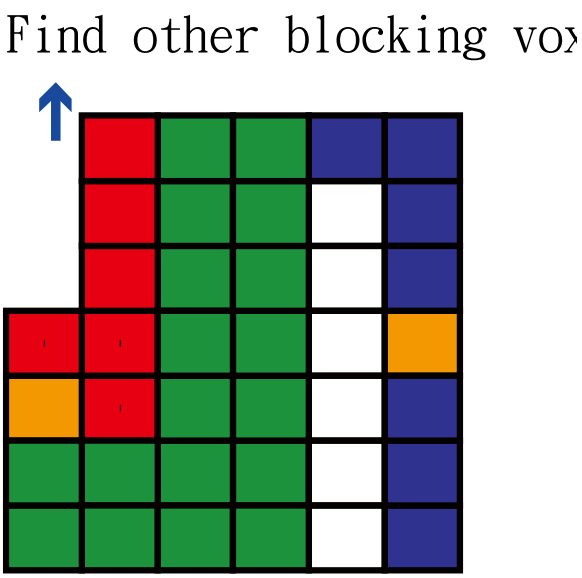

圖 21 找出 blocking voxel

隨機擴張P_1，使其符合最低 voxel 數

當我們完成前述的步驟，通常這時候 piece 的數目會小於規定的數量 m，因此我們為了盡可能的平均每塊 puzzle piece 的數目，必須盡可能擴張直到符合 m 為止。

首先，既然我們已經選出了所有的 blocking voxel(記住，選擇離 seed voxel 距離最遠 voxel)，在擴張P_1的時候要確保擴張的 voxel 不會選擇到 blocking voxel 以及 blocking voxel 之下(會被 blocking voxel 擋住而無法抽出)的 voxel 作為擴張的對象。

再來，判斷 voxel 是否可以擴張時要確保在P_1的擴張範圍內，如此才能確保該 voxel 可以抽出(表面顏色與P_1顏色相同)

接著，加入的 voxel 要確保跟P_1的 voxel 連通，關於這點我們可以在擴張的時候使用廣度優先搜尋法(BFS)來達成。

最後，每加入一塊 voxel 進P_1，就同等於把該 voxel 以上(沿著v_{p_1}方向)的所有 voxel 都加入。

為了保持模型的隨機性質，我們會隨機的選擇每塊 voxel 是否加入，直到我們的 puzzle piece 數目達到要求為止。

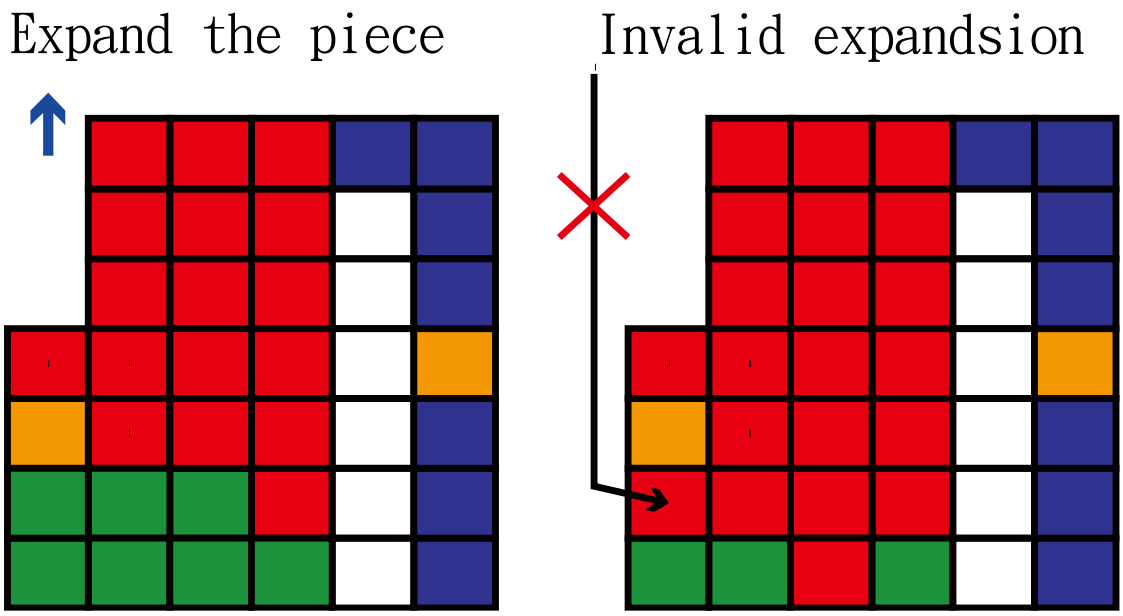

圖 22 隨機擴張P_1

如圖 22 所示，我們會再擴張範圍內以原來的圖 21 的範圍再往外隨機的擴張 voxel，直到數目符合我們的需求。

圖 22(右)是一個錯誤的例子:選擇 blocking voxel 以下的 voxel 會導致P_1無法抽出。

確定P_1符合 local interlocking

雖然我們盡力確保P_1的條件都滿足(見第三章第二節)，但還是有機會導致R_1不連通，從而使P_1失敗。

避免不連通的方式有幾種:

(i)重新回到 candidate set 的步驟，選擇下一個 seed voxel

(ii)透過我們之前介紹的 Accessibility 來避免破碎化(也就是不連通)。

如同第三節第一項所說，當R_1的形狀越破碎，代表 Accessibility 的總和越低，因此我們優先取出 Accessibility 越低的 voxel 來減低破碎的機率。

第四節　解出P_i的步驟

跟求出P_1相似，要找出P_i的過程我們一樣先求出 seed voxel，然後以此開始擴張，也就是從R_{i-1}中取出P_i。

不過，$\boldsymbol{P_i}$有額外的要求，我們要求$\boldsymbol{P_i}$在$\boldsymbol{P_{i-1}}$抽出之前不可以被抽出，所以我們的過程稍有不同。

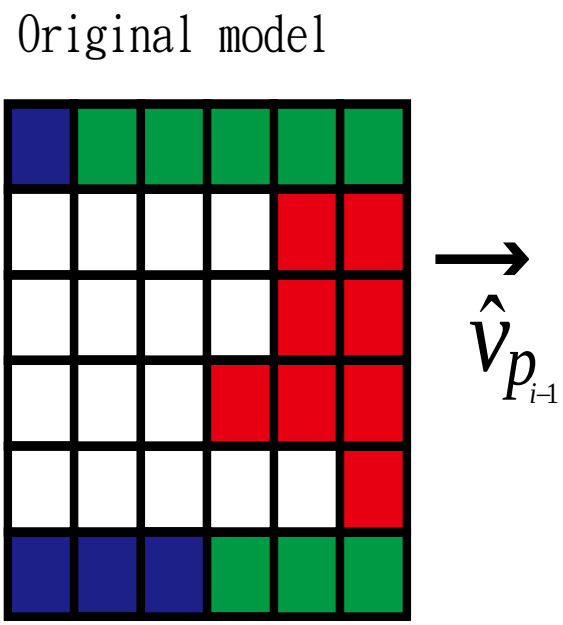

圖 23 圖中顯示了P_i與R_{i-1}的關係，紅色為P_i，往右方(+X)抽出，其餘顏色為R_{i-1}，白色代表 inside voxel，藍色與綠色為 shell voxel 的顏色

選擇 seed voxel

為了確保$\boldsymbol{P_i}$被$\boldsymbol{P_{i-1}}$阻擋住，並且在$\boldsymbol{P_{i-1}}$抽出後可以抽出，$\boldsymbol{P_i}$的 voxel 必須至少有一塊相鄰於$\boldsymbol{P_{i-1}}$。

這裡我們定義每塊 puzzle piece $\boldsymbol{P_i}$的抽出方向為$\hat{v}_{p_i}$，如果$\boldsymbol{P_i}$與$\boldsymbol{P_{i-1}}$的抽出方向平行，會使得$\boldsymbol{P_{i-1}}$會與$\boldsymbol{P_i}$可以同時抽出，因此每次選擇$\hat{v}_{p_i}$的時候都必須選擇$\hat{v}_{p_{i-1}}$垂直的方向。

因此，我們的策略如下:選擇R_{i-1}中相鄰$\boldsymbol{P_{i-1}}$的 voxel，並且要求這些 voxel 相鄰的方向一定要與$\boldsymbol{P_{i-1}}$的抽出方向垂直。

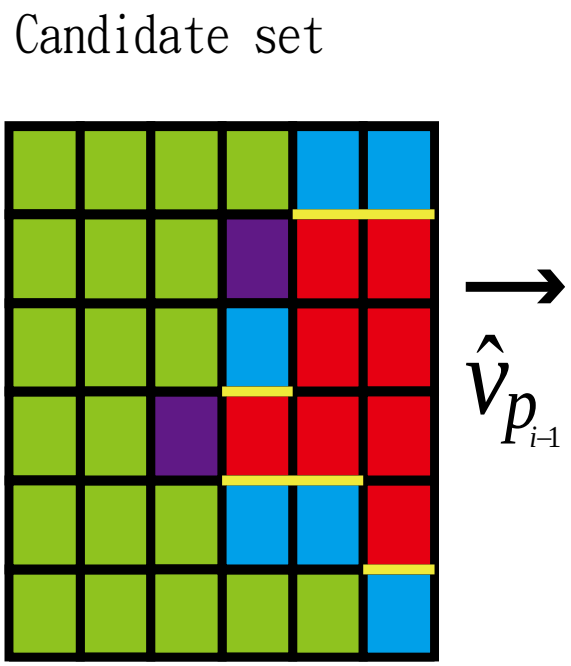

圖 24 選擇 candidate set 的方式，選取P_{i-1}(紅色區塊)相鄰的做 candidate set(淺藍色區塊)，而紫色的 voxel 因為接觸面只有平行，所以不會加入。

決定P_i的抽出方向

P_i的抽出方向決定於上一步 seed voxel 中與P_{i-1}的方向(見圖 25)，如前面所說，我們選擇的 seed voxel 本身與 P_{i-1}的接觸面必須與P_{i-1}的抽出方向垂直，否則會造成P_i與P_{i-1}一同抽出的情況。

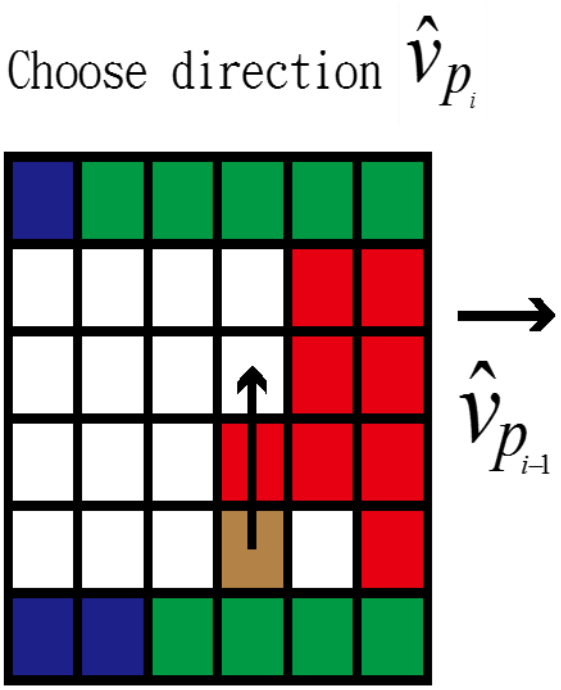

圖 25 決定P_i抽出方向，褐色為 seed voxel

找到P_i的擴張範圍、顏色、目標

我們的目標是令 seed voxel 可以抽出，跟據 seed voxel 的選擇，會影響到這一步的做法，我們分成以下幾種情況:

- **Seed voxel 為 shell voxel:**

第一塊加入P_i的 shell voxel 決定了P_i本身的顏色，因此我們的P_i的顏色就是 seed voxel 的顏色。(圖 26.)

為了令 seed voxel 可以往$\hat{v}_{p_i}$抽出，首先我們先考慮必須會跟隨著抽出方向加入的這些 voxel，稱為 seed voxel 的 mark voxels。 (圖 26.)

Seed voxel 的選擇仍然必須嚴格遵守$\boldsymbol{P_i}$的規則:都是同一個顏色，假如 mark voxels，也就是 seed voxel 抽出所影響的範圍會加入不同顏色的 shell voxel，我們就必須放棄，轉用其他的 seed voxel。

我們以 mark voxel 確保 seed voxel 可以抽出後，mark voxels 會被當作 destination voxel，接下來標記出P_i的擴張範圍。(圖 27)

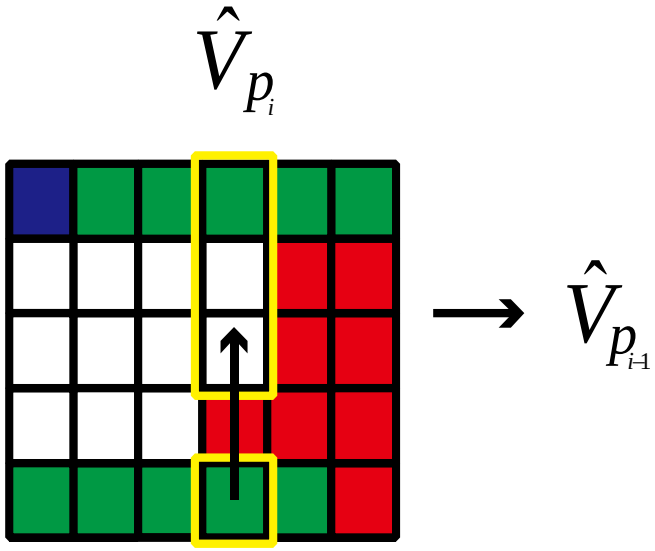

圖 26 下方黃框的 voxel 是 seed voxel，為了向+Y 方向抽出，上方的 mark voxel(黃框部分)也必須加入 P_i，考慮到本論文的限制，mark voxel 自身必須要是相

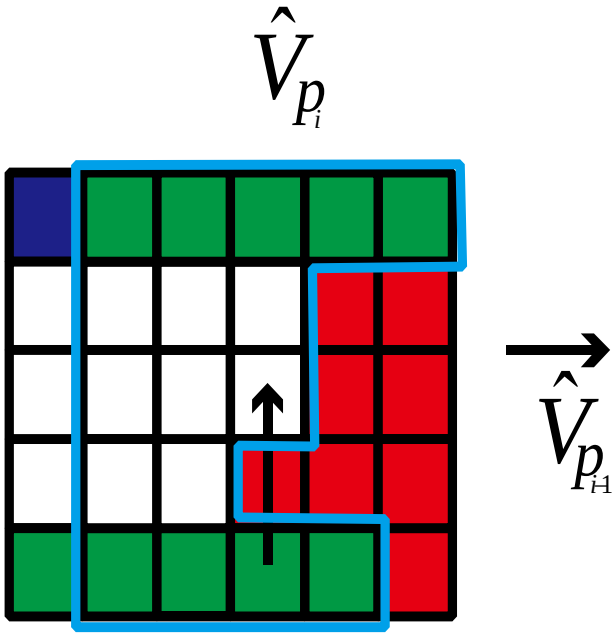

圖 27 以藍框表示P_i的 expand range，左下方綠色的 voxel 因為無法抽出所以不加入

- **Seed voxel 為 inside voxel:**

為了令 seed voxel 可抽出，如同上方的做法，我們考慮 seed voxel 的 mark voxels。

我們試著收集 mark voxel 的資訊，確認是否可以找到一塊 shell voxel 來做為P_i的顏色，如果沒有，則我們必須搜尋一塊距離 seed voxel 最近且連通的 shell voxel。

接著我們一樣標記出P_i的擴張範圍。

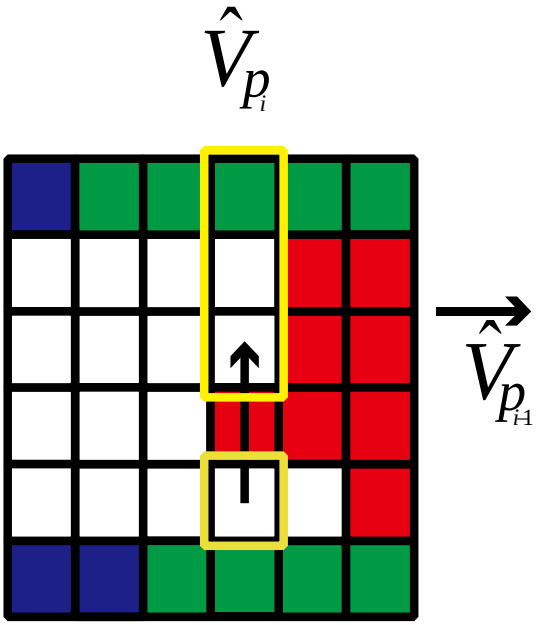

圖 28 以 inside voxel 為 seed voxel 的情況下，下方的黃框為 seed voxel，上方為 mark voxel

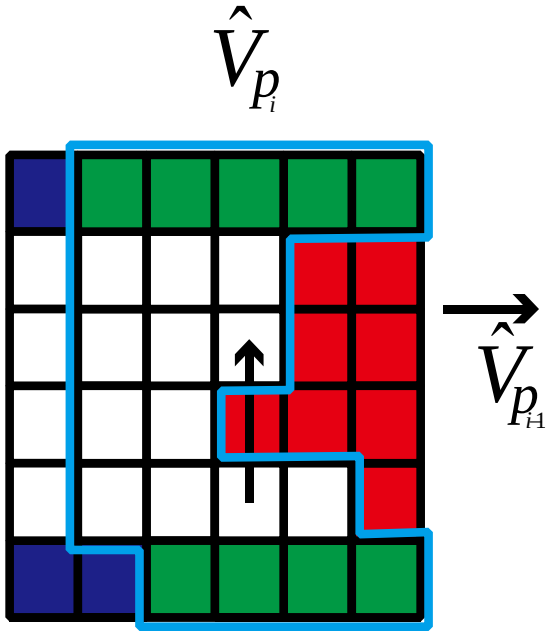

圖 29 以藍框標示在 inside voxel 為 seed voxel 的情況下的擴張範圍

設法令 seed voxel 可沿著方向抽出

經過上面的步驟，為了令 seed voxel 可以抽出，我們在擴張範圍內連通 seed voxel 與 destination voxel，並且令此條路徑可以抽出。

在這裡我們可以看到，目前的P_i被P_{i-1}所擋住，而且因為抽出方向互相垂直，我們的P_i確保不會跟著P_{i-1}同時抽出。(圖 30)

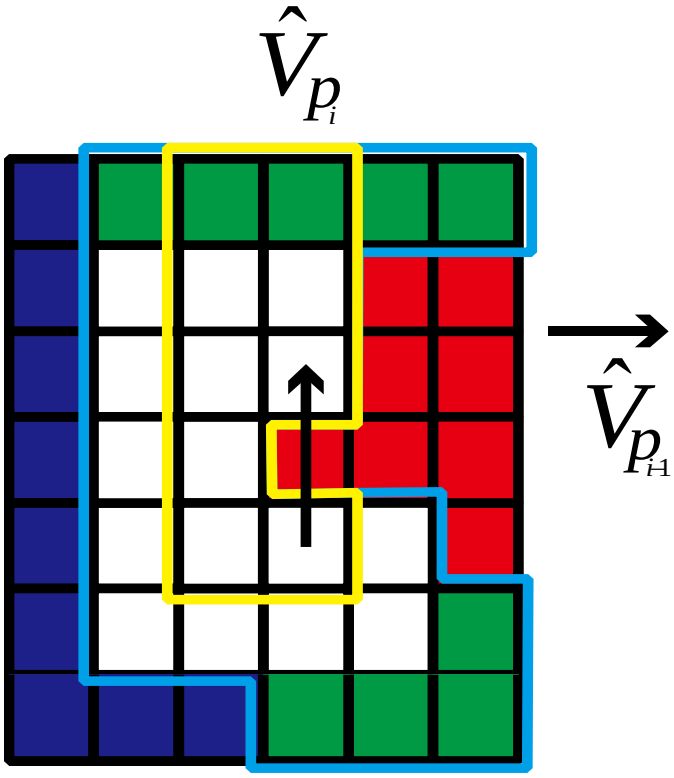

圖 30 淺藍色框的為擴張範圍，黃色框為連通 seed voxel 與 mark voxel 之後再沿著抽出方向的 voxel，都被加入P_i之中

找出剩餘的 5 個 blocking voxel

相似於第三節中"P_1尋找剩餘方向的 blocking voxel"的作法，現在我們有了一個可以抽出的 puzzle piece，接下來為了固定P_i不會因為擴張的緣故使得可以抽出$\hat{v}_{p_i}$以外的方向，我們必須找出剩下的 5 個方向的 blocking voxel。(圖 31.)

我們的策略很簡單:搜尋所有目前已經加入P_i的 voxel，然後試著找尋離這些 voxel 距離最遠的 voxel 作為 blocking voxel。

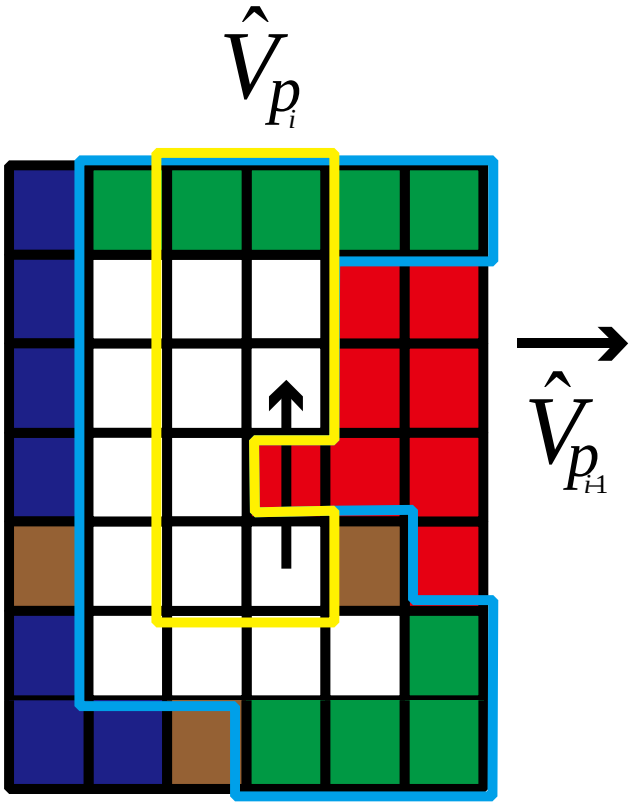

圖 31 褐色區域代表了 blocking voxel

隨機擴張P_i，使其符合最低 voxel 數

如同P_1的做法，我們以 BFS 隨機的擴張P_i並且使其符合最低的 voxel 數量。(圖 32.)

確定P_i符合 local interlocking

如同的做法，我們不能保證R_i為連通集，每次求出P_i後我們都必須確定R_i為連通。

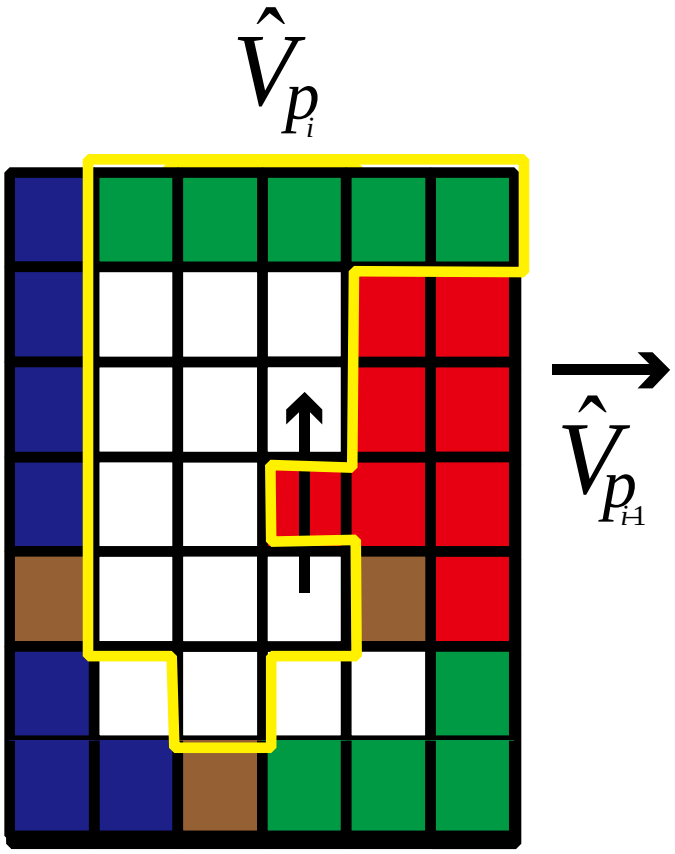

圖 32 黃色框的為最終擴張後的P_i

第五節　終止條件

我們的目標可以分為兩種: (i)簡單的把所有的顏色分配到數目不定 puzzle piece 就好 (ii)試圖平均分配所有的 puzzle 大小。

因此，結束 puzzle piece 的條件可粗略分為兩種: (i)當R_i的所有 shell voxel 只剩一種顏色或者(ii)剩下的R_i的 voxel 數量小於我們期望的 voxel 數。

第四章　結論

到目前為止我們已經可以處理一些簡單的模型並生成 puzzle，但是結果仍不理想，如圖所示。

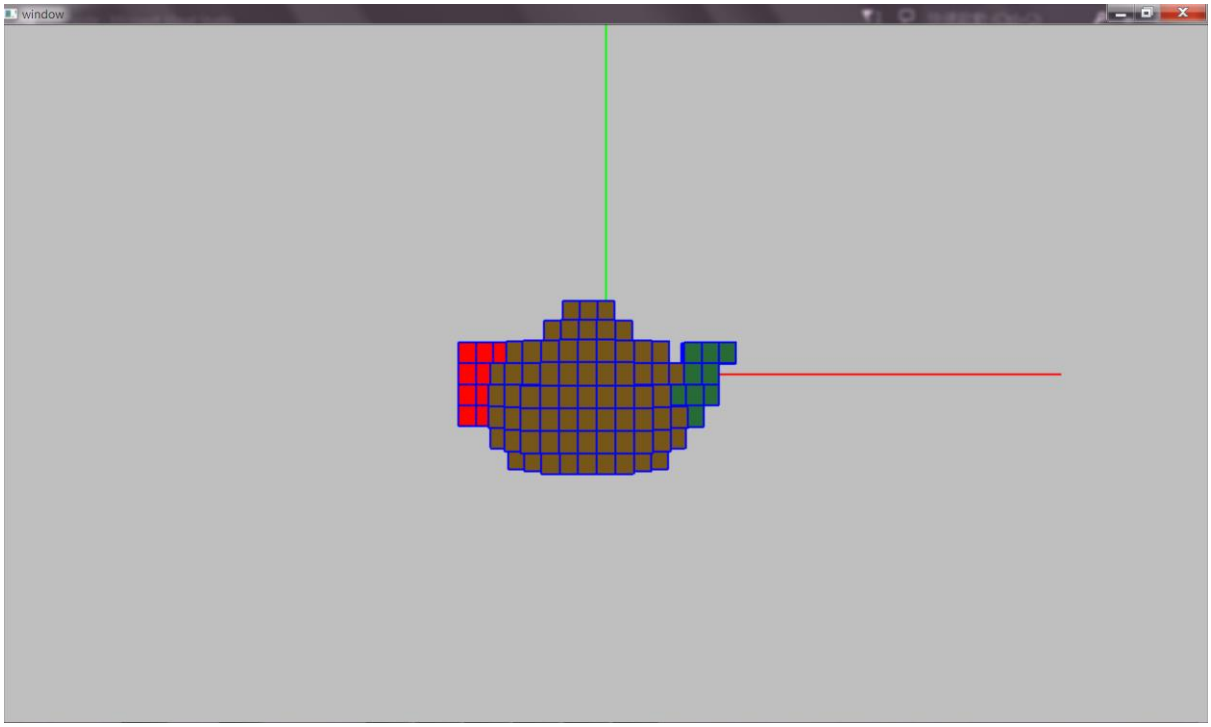

圖 33 原始體素模型 解析度為 16*16*16

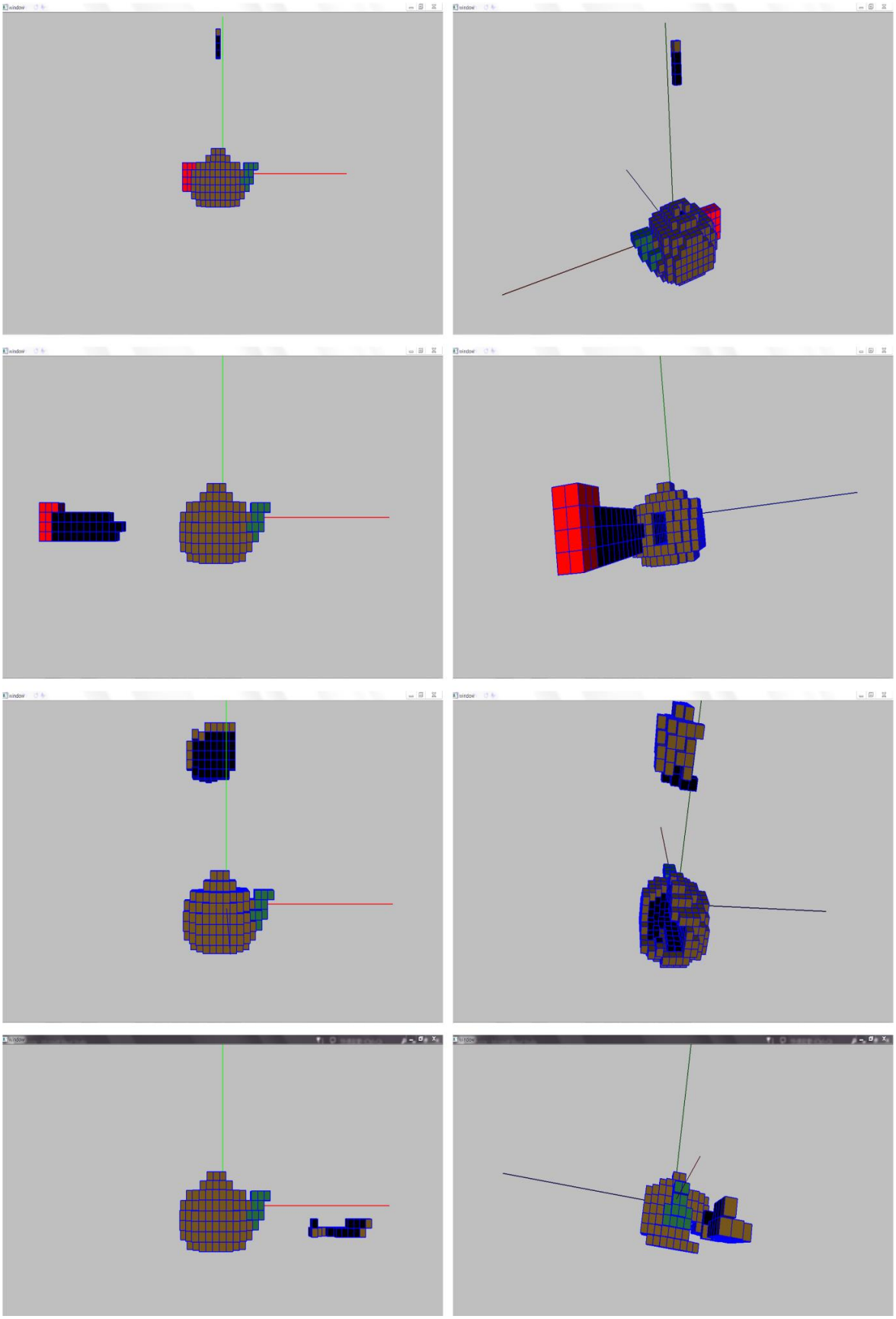

圖 34 目前的結果，由上到下為 P1 到 P4

儘管我們目前只有處理2^n解析度的體素模型，但是其實方法本身並不局限於體素的解析度大小。

儘管如此，方法本身還是有若干限制的情況會造成本身無法處理的情形:

如果模型本身的解析度過小(例如 2*2*2)或是過度狹長(例如 2*16*16)這些情況都會造成模型本身並沒有提供可以生長 puzzle piece 的空間，自然也無法做到滿足 local interlocking。

當模型本身的 shell voxel 不符合聯通的條件，也就是說該 voxel 本身與六鄰域的 shell voxel 顏色都不相同，同時又不與 inside voxel 相鄰的時候，我們將找不到任何將 shell voxel 擴張的可能，當然也會失敗。

參考文獻

[1] *MAURICIO BUITRAGO UNIQUE DIMENSIONAL ART 像素藝術* http://www.mauriciobuitrago.com/

[2] *Douglas Coupland 作品 體素藝術* http://www.vancouverconventioncentre.com/about-us/art-project/digital-orca

[3] *Game Minecraft 當個創世神遊戲官網.* https://minecraft.net/

[4] *GPU Gems 2 chapter 37. Octree Texture on the GPU 八元樹分割示意圖* http://http.developer.nvidia.com/GPUGems2/gpugems2chapter37.html

[5] *COFFIN, S. T. 1990. The PuzzlingWorld of Polyhedral Dissections.Oxford University Press. COFFIN 有關立體積木的著作*

[6] *Peng Song, Chi-Wing Fu and Daniel Cohen-Or 2012. Recursive Interlocking Puzzles ACM Trans. Graph. 31, 6 (Nov.), 128:1–128:10*

[7] *徐旻慶, 林炳賢,顏韶威,CGW 基於面積比例的體素化顏色計算技術*