3D-Z^+-string：一個針對影片中物件連續消失出現時空知識表示法

3D-Z^+-string：A spatio-temporal knowledge representation for the object successive appears and disappears

簡永仁
靜宜大學資管系
Email: yrjean@pu.edu.tw

黃敬欣
靜宜大學資管系
Email:g9771028@pu.edu.tw

摘要

本篇文章中，我們以 3D Z-string為基礎，改善提出新的時空知識表示法(Spatio-temporal Knowledge Representation)稱為 3D Z^+-string。針對 3D Z-string無法表示物件多次消失再出現的情形，我們將物件的消失，視為物件的大小變為“零”，並增加一個新運算子“&”，來記錄影片中物件消失的影片片段。因而，物件多次消失再出現的情形可以被記錄，使得表示法更為完善。同時，我們提供兩種搜尋的方法，可尋找符合特定需求的影片片段。第一種為判斷在第N個影片片段時，兩物件之間的空間關係為何；第二種為尋找兩物件之間的特定空間關係是在第幾個影片片段時發生。這兩種方法，可以提供相關領域的軟體開發人員使用，以提供使用者更方便、強大的影片查詢功能。

關鍵詞：空間知識表示法，時空知識表示法，2D string，3D Z-string，3D Z^+-string。

ABSTRACT

In this paper, we propose a novel spatio-temporal knowledge representation called 3D Z^+-string that is based on 3D Z-string. As a result of 3D Z-string, it cannot deal with the condition that an object appears and then disappears for more than one time. To solve this problem, we set the disappearance of the object as that object' size is zero. And, we bring up the "&" to be a new operator. Therefore, it can record that an object appears and then disappears for more than one time. At the same time, we offer two search methods to find the specific frame. The first method, it can find out the spatial relationship between two objects in the Nth frame. The second method, it can find out the spatial relationship between objects that in which frame. These two methods can be offered to software developers. They can extend these methods for users to more convenient and powerful video search functions.

Keywords: Spatial knowledge representation, spatio-temporal knowledge representation, 2D string, 3D Z-string, 3D Z^+-string.

一、緒論

近幾年資訊科技的快速發展，影像資料庫的應用與日俱增，被廣泛地應用在電腦製圖、工業管理、地理資訊系統(GIS)…等方面。影像的資料量不斷增加，對於影像資料之管理也越

來越重要。所以，有效率儲存管理大量影像的影像資訊系統(Pictorial Information System)就變得非常重要。傳統的找尋圖片方式，只需輸入要搜尋的關鍵字或描述。但是影像資料庫裡的影像資料，為使用者主觀意識將資料作命名及關鍵字加註。建置的方式較為麻煩費時，且每個使用者的主觀意識都不相同。因此，使用者利用關鍵字搜尋資料庫的資料時，往往會因認知的不同，而無法搜尋到所尋找的目標。故使用者如何在這龐大的資料庫中有效且快速的搜尋圖片，就顯得非常重要。因此，如何透過以圖尋找圖(Query-By-Pictorial Example)[6]的方式，來快速找出目標影像是很重要的課題。

以影像內容為查尋基礎的影像搜尋技術(Content-Based Image Retrieval，CBIR)[4]因而被提出。CBIR由影片中擷取特徵，如：顏色(Color)，紋理(Textural)，空間關係(Spatial Relationships)等資訊作為影像比對的依據。因此，不會用到使用者自己主觀的建置定義及關鍵字。但CBIR搜尋機制並不成熟，所得的結果還不令人滿意，且也必須花費較多的運算時間。

在1987年，張教授等人首先提出了2D string空間知識表示法[8]，將影像檢索轉換成字串的比對。之後陸續有許多人也加入此研究領域，將2D string改善延伸。早期影像資料庫的搜尋，都是用來尋找圖片。但是近年來科技的發達，3C產品的普遍，不只是圖片的資料量增加，影片的數量也跟著暴增。因此，對於影片資料的管理也越來越重要。故許多影像相關的搜尋技術一一被提出，如3D C-string[2]及3D Z-string[3]…等。在本篇文章中，我們以3D Z-string[3]為概念基礎。但我們發現到某情況3D Z-string無法完整表示。如圖1所示，有一隻海豚在海中連續跳躍，因而有連續的消失又出現之情況。針對這樣的情況，3D Z-string只記錄物件有狀態的改變，卻未能將物件連續消失又出現的狀態記錄起來，圖2為海豚和船在時間軸上的關係變化。因此，我們針對3D Z-string進行改善。我們將物件的消失，視為物件大小為“零”。並增加一個新運算子“&”，以記錄影片中物件消失的片段(Frame)。因而，物件多次的消失再出現，可以被記錄下來，使得表示法更為完整。我們改善提出新的時空知識表示法，稱為3D Z^{+}-string。最後，我們為方便使用者搜尋符合需求的影片片段，提供兩種搜尋方式。第一種方法為搜尋在第N個片段時，兩物件之間的空間關係為何；第二種方法為搜尋兩物件之間的空間關係是在第幾個片段發生的。

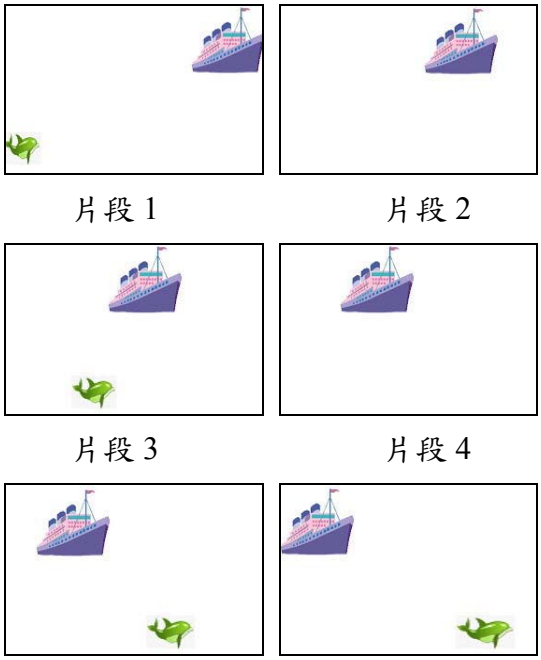


圖1. 影片的6個連續片段

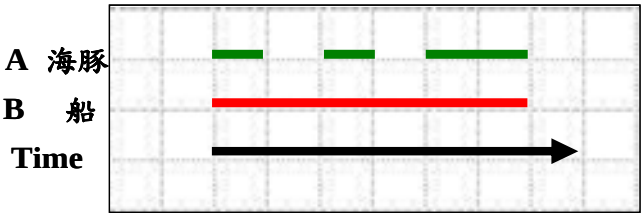


圖2. 海豚和船在時間軸上的關係變化

本文的主要架構，在第一章我們介紹影像資料庫搜尋的背景，談論影像資庫的應用和傳

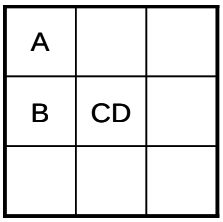
統的搜尋方式，進而帶出本研究的動機與主題。在第二章說明相關的文獻探討及背景。在第三章說明3D Z^+-string所做的修正與改善，並對李教授等人的演算法做修改，同時，提供兩種搜尋方式。最後，第四章為本篇文章的總結。

二、文獻探討

影像資料庫系技術不斷的進步，許多相關的影像搜尋技術蓬勃發展。而空間知識表示法(Spatial Knowledge Representation)就是其中的一種，可用來對影像建立搜尋用的索引機制。空間知識表示法大致上可分為卡笛生座標系統(Cartesian Coordinates System)與極座標系統(Polar Coordinates System)。在本篇文章中，我們所用的就是卡笛生座標系統。

(一) 2D string 的各種變形發展

在空間知識表示法研究領域中，張教授等人在1987年提出2D string空間知識表示法[8]。2D string有一項重要的概念，就是在擷取影像空間資訊之前，影像必須先經過前置的影像處理程序，並且以最小邊界矩形(Minimum Bounding Rectangle，MBR)的方式標示出影像中的每一個物件。原本的圖片經過影像處理後，會將影像中有用的資訊抽離出來，即形成所得的符號圖片(Symbolic Picture)，如圖3。符號圖片的物件會辨別成“零尺寸”(Zero-Size)的方式來表達。因此，對2D string[8]而言，並不包含物件大小的資訊。爾後有許多人陸續投入此領域，他們將2D string的方法改善延伸，發展提出了2D G-string[5]、2D C-string[9]、2D C^+-string[7]、2D Z-string[1]、3D C-string[2]，3D Z-string[3]等表示法。到現在仍然有新的空間知識表示法與時空知識表示法及相似度檢索方法被提出。



u-string:(A=B<C:D)

v-string:(B=C:D<A)

圖3.符號圖片，並以2D string來表示

在1990年時，李教授等人提出2D C-string空間知識表示法[9]，來解決2D G-string因切割而產生大量子物件的缺點。2D C-string透過重新定義切割的規則，來減少因物件過多的切割，而產生大量子物件的問題。並將運算子拓展到七個，如表格1.所示，來完整表達兩物件在一維空間的所有關係。除了“=”外，其餘都有反運算子(Inverse Operator)，故共有十三個空間運算子，如圖4。但是2D C-string[2]在字串的表達上，仍無記載物件的大小與距離資訊。在1994年，黃教授等人提出2D C^+-string[7]來補足上述的資訊。2D C^+-string增加了物件大小的資訊，使得2D C^+-string比2D C-string更加精確。不過2D C^+-string仍然須要切割，故仍會產生子物件，使得字串長度會過長。因此，為了避免產生過多的子物件及過長的字串。在2003年，李教授等人，以零切割機制(Zero-Cutting Mechanism)，提出2D Z-string[1]。因為2D Z-string不需要做切割(No-Cutting)。因此，不會產生子物件。

在2D C-string中，空間運算子“/”用來表示切割機制而使用，也是為了避免2D C-string因缺乏精確性所產生的不明確情況。然而，在2D C^+-string時，已經實際考量了物件的大小與距離資訊，能夠解決不明確的情形。但2D C^+-string並未利用計量資訊的優點，故2D Z-string將空間運算子“/”重新檢視，並將它列入計量資訊。

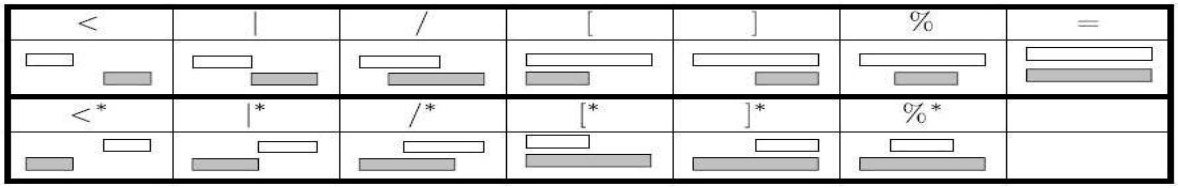


圖4. 兩物件在x軸(y軸)上的13種空間關係

表格1. 2D C-string空間運算子之定義

Notation	Condition	Meaning
A < B	end(A) < begin(B)	A disjoins B
A = B	begin(A) = begin(B), end(A) = end(B)	A is the same as B
A / B	begin(A) < begin(B) < end(A) < end(B)	A is partly overlap with B
A | B	end(A) = begin(B)	A is edge to edge with B
A] B	begin(A) < begin(B), end(A) = end(B)	A contains B and they have the same end bound
A % B	begin(A) < begin(B), end(A) > end(B)	A contains B and they do not have the same bound
A [B	begin(A) = begin(B), end(A) > end(B)	A contains B and they have the same begin bound

於是將 2D Z-string 的計量資訊(Metric Information)重作定義：

1. 物件的大小(size)：A_s表示物件A在x軸(或y軸)上的投影大小為s。s即End(A)-Begin(A)，其中End(A)為物件A投影在x軸(或y軸)上的結束點(end-bound)，Begin(A)為物件A投影在x軸(或y軸)上的起始點(begin-bound)。

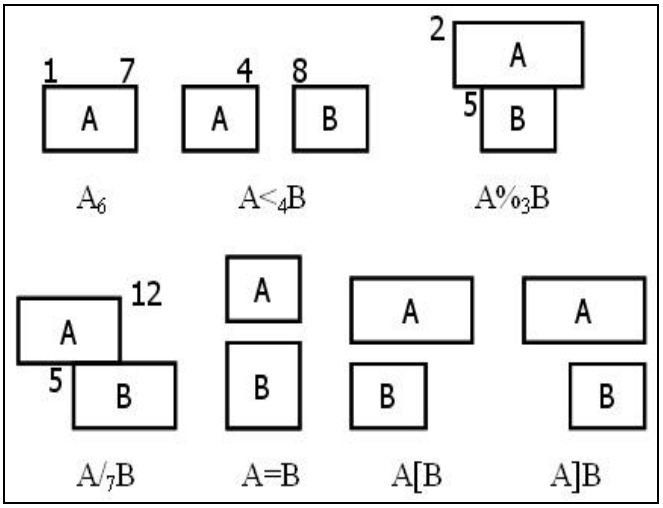


圖5. 物件大小及物件間距離

2. 有關於運算子"<"和"%"及"/"的距離d：表示A物件與B物件在x軸(或y軸)的距離，A $<_d$ B和A $\%_d$ B及A $/_d$ B中的d，分別定義為Begin(B)－End(A), Begin(B)－Begin(A), and End(A)－Begin(B)，如圖5。
3. 其他空間運算子{"[", "=", "]", "|"}的對應距離皆訂為零，如圖5。

(二) 3D Z-string

3D Z-string[3]和 2D Z-string[1]的差別，在於2D Z-string是記錄圖片的資訊，而 3D Z-string是記錄影片的資訊。其中 3D Z-string比 2D Z-string多記錄第三維度-時間的時間軸資訊。因為圖片是單張的，而影片是連續多張的圖片所組成，因此，需要將第三維度的時間軸資訊記錄起來。3D Z-string包含了 2D string的所有定義規範，不論是物件大小資訊、物件之間的距離，還是零切割機制的特性，都有含括在內。不過，3D Z-string比 2D string多了記錄物件狀態變化的移動運算子。3D Z-string的結構包含六個組合

元素(O, A, R_g, R_l, R_t, S)：

(1) O：影片中所包含的所有物件；

(2) A：在 O 所包含的物件中，A 用來描述屬性；

(3) R_g = {“<”, “|”}：為全域運算子，用來處理物件之間非重疊(non-overlap)的情形;

(4) R_l = {“=”, “[”, “]”, “%”, “/”}：為區域運算子，用來處理物件之間完全重疊(complete overlap)和部分重疊(partly overlap)的情形;

(5) R_t = {“↑”, “↓”, “#”}：而“↑”和“↓”是移動運算子，表示物件移動的方向，只使用在u-(v-)string上。“↑”表示物件往座標軸的右邊(x-axis)和上面(y-axis)移動，即往正的方向移動。“↓”是表示物件往座標軸的左邊(x-axis)和下面(y-axis)移動，即往負的方向移動。“#”是間隔運算子，用來表示物件移動速率改變的間隔或是物件大小改變的間隔，只使用在t-string。

(6) S = {“(”, “)”}：用以描述在同一空間或時間內的 template object。

它們的遵循規則如下：

1. 物件的大小(size)：A_s表示物件A在x軸(或y軸)上的投影大小為s。s即End(A)-Begin(A)，其中End(A)為物件A投影在x軸(或y軸)上的結束點(end-bound)，Begin(A)為物件A投影在x軸(或y軸)上的起始點(begin-bound)。
2. 關於運算子“<”和“%”及“/”的距離d：表示A物件與B物件在x軸(或y軸)的距離，A $<_d$ B和A $\%_d$ B及A $/_d$ B中的d，分別定義為Begin(B)－End(A), Begin(B)－Begin(A), and End(A)－Begin(B)。
3. 物件移動的速度和物件大小的改變與移動運算子$\uparrow_{v,r}$ 和$\downarrow_{v,r}$有關：運算子$\uparrow_{v,r}$ 和$\downarrow_{v,r}$中，“v”為物件移動的速度，而“r”為物件大小的改變。例如：A$\uparrow_{3,2}$表示物件的速度=3，表示物件從這個Frame到下個Frame的時候，往正的方向移動了3個單位。而物件大小的改變=2，表示物件從這個Frame到下個Frame的時候，變為上個Frame的物件大小的2倍(沒變表示為1，1倍)。在物件消失的時候，即把物件size視為0，以$\uparrow_{(0,0)}$表示。
4. 間隔運算子#：在$\#_i$中的“i”表示當物件移動的速度或物件大小的改變等狀態改變所間隔的時間長度。如果物件沒有速度或大小的改變時，t-string則不需要用到間隔運算子。

以圖6的3個連續片段為例，先以物件在影片中的起始位置(如圖7)為基點，做空間關係判斷，並利用A、B兩物件在時間軸上的關係(如圖8)做時間關係判斷，之後再將狀態的變化記錄起來。得到最後的3D Z-string，其中u-string為($A_2\uparrow_{(2,1)}\uparrow_{(3,1.2)} < B_1\uparrow_{(0,1)}$)，v-string為($A_2\uparrow_{(0,1)}\uparrow_{(0,1)} < B_1\uparrow_{(0,1)}$)，t-string為($B_3\#_3=(A_3\#_2\#_1)$)。

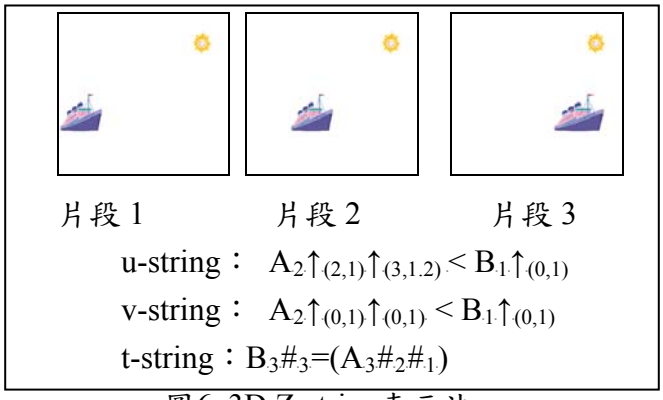


圖6. 3D Z-string表示法

由此例可知，3D Z-string[3]可以表現出，在連續影片片段中，物件都存在沒有消失的情況下，物件的移動和物件大小的變化(如圖6)，但3D Z-string卻無法完整表示出圖1的情況。這是因為3D Z-string只記錄物件有狀態的改變，但卻未將物件連續消失又出現的狀態記錄起來。

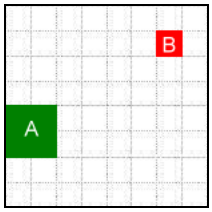


圖7. 為圖6之A、B兩物件的起始位置

圖8. A、B兩物件在時間軸上的關係

三、提出的新方法

為了解決，3D Z-string[3]，無法表示物件多次消失又出現的狀況之缺點。我們提出3D Z^+-string時空知識表示法。

(一) 研究探討

我們所提出的方法3D Z^+-string，主要是以3D Z-string為基礎進行改良。因此，如零切割機制和加入物件的大小...等優點，我們都延續保留下來，並增加新空間運算子和新定義，使表示法的記錄更為完善。為了改善3D Z-string的缺點，主要有兩點需著手修改：

1. 改進原本 3D Z-string的定義，將原本的移動運算子$\uparrow_{(v,r)}$和$\downarrow_{(v,r)}$中“r”做補充。將影片中物件的消失，當做物件的size變為 0，即“r”=0的時候，當做為物件消失，以$\uparrow_{(0,0)}$來表示。而“v”的定義維持不變。
2. 要使t-string能表示物件在時間軸上的消失狀況，我們將原本 3D Z-string的定義元素組合R_t，新增一個特殊運算子“&”。用來表示在影片連續片段中，物件多次出現再消失的狀態，所必須記錄的物件消失間隔。

經過上述兩點的修改補充定義，以及使用新增加的運算子後，以下就是我們所提出來的新時空知識表示法 3D Z^+-string。

(二) 時空知識表示法 3D Z^+-string

我們所提出的方法 3D Z^+-string，利用上述提到的兩點，改善先前的一些規則。在第五個元素組合R_t應該修正為R_t={“↑”, “↓”, “#”, “&”}，即新加入的特殊運算子“&”。用來區別在影片的連續片段中，物件一直出現沒有消失(使用“#”)和物件出現又消失(使用“&”)的情況。物件沒有狀態改變時，t-string不需用到運算子“#”。反之，若是物件沒有多次出現再消失的情形，t-string不需用到運算子“&”。

為方便 3D Z^+-string的字串產生，我們在執行程式之前，將要輸入的資訊先做一個特別的處理。即在做時間軸的空間關係時，若遇到在連續片段中，物件多次的消失再出現，時間軸物件會分成很多段，並會使它與其他物件之間的空間關係變得更亂更難判斷，故我們先做一個假設，先將被分成多段的時間軸物件看成一整段，再利用“&”這個運算子把它區別開來。雖然我們把物件的size設為 0 視為物件消失，而在這邊的“視為連續出現”，只是我們把這物件的size設為 0，但它仍然是存在的。這個方式有利於我們處理物件在時間軸的空間關係判斷。最後，再把剛剛所做的假設，即未記到的物件消失片段，以“&”補回到產生的字串中。

依循前述之說明後，我們需將原字串產生演算法string generation algorithm[3]稍做改變，修改後的演算法稱為3D Z^+-string generation，如圖9。以下為演算法的符號縮寫定義[3]：

- Bi/Ei：物件 i 的 begin-bound 和 end-bound。
- Si：物件i的字串。列出產生的u- (v- or t-) string。在x-(y-)axis，Si包含物件i的資訊，像是物件的位置大小、物件的速度及物件大小的改變。在time-axis，Si包含物件i的間隔長度、物件消失的間隔長度及移動運算子$\uparrow_{v,r}$ 和$\downarrow_{v,r}$等資訊。
- SQ：同值序列(Same-Value-List Sequence)，Bi 和 Ei 的 i = 1,2,. . .,n，在同值序列中至少包含一個同值點，而這些同值表都是由同值序列 SQ 而來。

- DO：dominating object (Lee and Hsu, 1990)。物件之間有相同的 end-bound，而這些 end-bound 分類在同值表中。而在這些物件中有最小的 begin-bound 就叫 dominating object。
- FO(i)：former object (Lee et al., 2002)。Former object 表示說，比物件 i 有更小的 begin-bound，或是跟物件 i 有相同大小的 begin-bound 而卻有最大的 end-bound。
- $FO_{Nest}(i)$：nearest former object (Lee et al., 2002)。在物件i之前的物件有最大的begin-bound則為nearest former object。若有一個以上，則選擇有最小的end-bound的物件來當作nearest former object。
- FO_{None}：表示在剩下的其他物件中不是最接近的former object。

```
Algorithm: 3D Z+-string generation
Input: an object list O1(B1,E1,S1), O2(B2,E2,S2),… , On(Bn,En,Sn) of the x-(y- or t-) projections,
where n is the number of objects.
Output: an u-(v- ot t-) string
1.    Sort Bi,Ei,i=1,2,… ,n in non-decreasing order and group the same value points into
    a same-value list. From a same-value-list sequence SQ.
2.    For each same-value-list L in SQ
3.    Begin
4.        If there is an end-bound in L)
5.            Find the dominating object DO from L.
6.            If there exist any objects partly overlapping with DO then
7.            Choose among them the object with the smallest end-bound.
                If the number of objects with the smallest end-bound is more than one,
                    choose among them the object with the smallest begin-bound.
                Let PO be the chosen object.
8.                Let TO1= templateObjectGeneration (the objects i with Bi≧BPO and Ei≦EPO)
                    and remove those Bi and Ei from SQ.
9.                Let TO2= templateObjectGeneration (the objects i with Bi≧BDO and Ei≦EDO)
                    and remove those Bi and Ei from SQ.
10.           Merge TO1 and TO2 into TO, where BTO=BTO2, ETO=ETO1, STO2=( STO2/d STO1) and
                d = ETO2－BTO1.
11.       Else
12.           Let TO = templateObjectGeneration (the objects i with Bi≧BDO and Ei≦EDO)
                and remove those Bi and Ei from SQ.
13.               EndIf
14.               Add BTO and ETO to SQ.
15.       EndIf
16.   EndFor
17.   TO = templateObjectGeneration (all the remaining objects)
18.   When objects met the state that the object disappear and appear more than once.
    Then assign the "&" to TO. /* used in t-string */
19.   Output STO.
```

圖 9. 3D Z^+-string generation algorithm

```
Function: templateObjectGeneration
Input: a list of objects W
Output: a template object
1.    While (more than one object in W).
2.    Begin
3.        For the objects with same begin-bound and end-bound, merge them into
            a new template object by operator "=".
4.        If there is only one object in W, exit the while-loop.
5.        For each object I in W, find its FONest(i).
6.        Find FONome. Let Q be FONome and N be FONest(Q).
7.        If (BN=BQ and EN>EQ) then merge Q and N into a new template object TO = (BN, EN, (SN[SQ)).
8.        ElseIf (EN=EQ and BN<BQ) then merge Q and N into a new template object
              TO = (BN, EN, (SN]SQ)).
```

```
9.        ElseIf (EN>BQ and BN<BQ) then merge Q and N into a new template object
                TO = (BN, EQ, (SN/d SQ))
            and d = EN—BQ.
10.       ElseIf (EN<BQ) then merge Q and N into a new template object TO = (BN, EQ, (SN<dSQ))
            and d = BQ—EN.
11.       ElseIf (EN=BQ) then merge Q and N into a new template object TO = (BN, EQ, (SN|SQ)).
12.       ElseIf (EN>BQ) then merge Q and N into a new template object TO = (BN, EN, (SN%dSQ))
            and d = BQ—BN.
13.       EndIf
14.       Remove Q and N from W and add TO to W.
15.   EndWhile
16.   Return the template object.
```

圖 10. TemplateObjectGeneration function

在 3D Z^+-string generation algorithm中，需要另一個演算法來做合併前的空間關係判斷，我們直接引用李教授等人的templateObjectGeneration function[3]，如圖 10。我們利用 3D Z^+-string generation algorithm及templateObjectGeneration function來產生 3D Z^+-string的字串，我們首先產生物件i的Si，與其他的資訊(B_i,E_i,S_i)，i=1,2,3,...n，n表示為影片中物件的數量。將這些輸入到 3D Z^+-string generation algorithm。然後，沿著x-(y- or time-)axis，尋找dominating object。如果沒有物件跟dominating object部分重疊，則找有涵蓋到dominating object的物件(包含dominating object)。換句話說，如果有任何物件跟dominating object部分重疊，則選擇這些物件中有最小的end-bound。如果這些物件不只一個有最小的end-bound時，就選擇這些物件中有最小的begin-bound。令所選擇的物件為former object。然後找出物件中dominating object的begin-bound到former object的end-bound之間所涵蓋的範圍。所涵蓋的物件，將它們合併為一個template object(TO)。然後重複依照這樣的方式去產生其他的dominating object。最後，將template object和其他的物件給合併起來。利用 3D Z^+-string generation Algorithm來產生字串。而利用templateObjectGeneration function將這些物件合併為template object。

現在，利用我們提出的演算法，產生圖 1 的 3D Z^+-string。在u-string的產生，將物件A和物件B的起始位置投影在x-(y-)axis上，成為符號圖片。接著依照x-axis的部份做字串產生解釋。$A_1{\uparrow}_{(0,1)}{\uparrow}_{(0,0)}{\uparrow}_{(2,1.2)}{\uparrow}_{(0,0)}{\uparrow}_{(2,1.1)}{\uparrow}_{(1,1.2)}$，$B_2{\downarrow}_{(1,1)}$。則 A(0,1, $A_1{\uparrow}_{(0,1)}{\uparrow}_{(0,0)}{\uparrow}_{(2,1.2)}{\uparrow}_{(0,0)}{\uparrow}_{(2,1.1)}{\uparrow}_{(1,1.2)}$)，B(5,7, $B_2{\downarrow}_{(1,1)}$)，將這些輸入到 3D Z^+-string generation algorithm。然後，依照所訂的規則在 3D Z^+-string generation algorithm中執行。沿著x-axis，找到dominating object為物件A後，再跟物件B做合併，判斷兩物件之間的空間關係再將之合併起來。這樣我們就可以得到 u-string：$(A_1{\uparrow}_{(0,1)}{\uparrow}_{(0,0)}{\uparrow}_{(2,1.2)}{\uparrow}_{(0,0)}{\uparrow}_{(2,1.1)}{\uparrow}_{(1,1.2)}<B_2{\downarrow}_{(1,1)})$。而在y-axis部份的一樣跟著上述步驟，就會得到 v-string：$(A_1{\uparrow}_{(0,1)}{\uparrow}_{(0,0)}{\uparrow}_{(0,1)}{\uparrow}_{(0,0)}{\uparrow}_{(0,1)}{\uparrow}_{(0,1)}<B_2{\uparrow}_{(0,1)})$。

最後在t-string的部分，依先前所說，為方便 3D Z^+-string的字串產生，依照之前所提的特別做法，先將物件消失再出現的時間軸物件，假裝視為都一直存在(只是size為0)。最後，再利用“&”運算子把它區別開來。所以，先將A物件多段的時間軸看成一整段，然後再與B物件的時間軸做空間關係的判斷。此時所產生的t-string還不完整，接著將“&”給放回時間軸上消失的段落裡。先產生的字串為$((A_6\#_6)=B_6\#_6)$，再將“&”放回 $((A_6\#_6)=B_6\#_6)$ 裡。則可得 t-string：$((A_6\#_1\&_1\#_1\&_1\#_1\#_1)=B_6\#_6)$。

```
Algorithm: video reconstruct
Input: an u-(v- or t-) string with n elements: string = (E1, E2, … , En)
Output: a list of video objects: ObjectList = (O1, O2, … , Om)
1.     Loc← 0; ObjectList← null; Stack← null; i← 1; j← 0;/*Initialization*/
2.     MoreOperators← False;
3.       While(more elements in the u-(v- or t-) string) /* process the u-(v- or t-) strings*/
4.         While(MoreOperators)
5.           i← i+1; /*next operator*/
6.           Case Ei.sym
7.             "%": Loc← Loc + Ei.size; i← i+1; MoreOperators← False;
8.             "<": Loc← Loc + PreviousObjectSize + Ei.size; i← i+1; MoreOperators← False;
9.             "/": Loc← Loc + PreviousObjectSize - Ei.size; i← i+1; MoreOperators← False;
10.            "|": Loc← Loc + PreviousObjectSize; i← i+1; MoreOperators← False;
11.            "]": If Ei+1.sym≠ "(" then TemplateSize← Ei+1.size;
                      Else TemplateSize← GetTemplateSize(i+1, string);
                           Loc← Loc + PreviousObjectSize- TemplateSize;
                           i← i+1; MoreOperators← False;
                      EndIf;
12.            "=" or "[": i← i+1; MoreOperators← False;
13.            "↑ ": Append (v,r) to Oj.motionList; MoreOperators← Ture;
14.            "↓ ": Append (-v,r) to Oj.motionList; MoreOperators← Ture;
15.            "#": Append Ei.size to Oj.intervalList; MoreOperators← Ture;
16.            "&": Append Ei.size to Oj.intervalList; MoreOperators← Ture;
17.            ")": Pop an element E from Stacj;
                    Loc← E.beginBound; /*E is a template object*/
               PreviousObjectSize← E.size; MoreOperators← Ture;
18.            EndCase
19.          EndWhile
20.          While(Ei.sym="(")
21.             Create a template object E;
22.             E.beginBound← Loc;
23.             E.size← GetTemplateSize (i, string);
24.             Push the template object E onto Stack;
25.             i← i+1;
26.          EndWhile
27.          If Ei is a string object then
28.             j← j+1;
29.             Create a new object Oj so that
30.                  Oj.sym← Ei.sym; Oj.size← Ei.size; Oj.beginBound← Loc;
31.             Append object Oj to ObjectList.
32.           EndIf
33.           PreviousObjectSize← Ei.size;
34.           MoreOperators← Ture;
35.    EndWhile
36.    Output the ObjectList.
```

圖 11. Video reconstruct algorithm

至於李教授等人的影像重建演算法 video reconstruction algorithm[3]也需修改，修改後的演算法為 video reconstruct algorithm，如圖 11。

(三) 提出的兩種搜尋方法

我們所提出的第一種方法為 QueryFrame algorithm，如圖 13。可以查詢在影片中的某個片段，某兩物件之間的空間關係為何。以下為演算法中的相關定義：

- B_A：A的begin-bound，E_A：A的end-bound；B_B：B的begin-bound，E_B：B的end-bound。
- B_{A0}：改變後A的begin-bound，E_{A0}：改變過後 A 的 end-bound 。 B_{B0} ： 改 變 後 B 的 begin-bound，E_{B0}：改變過後B的end-bound。
- TL：表示物件連續出現的總時間長度(total length)，第i個連續的時段為#x_i，第i個消失的時段為 &y_i ， 1< x_i ,y_i < TL ， i=1,2,3,…,TL。Ex：(A_5#$_2$&$_1$#$_2$)表示A的TL=5，x_i=2,2，則x_1=2, x_2=2，y_i=1 則y_1=1。

- CT：狀態改變的時間(change time)，CT_i為第i個狀態改變的時間，遇到"#"或"&"則記一次CT。$CT_{ALL}=x_{i=1,2,3\ldots}+y_{i=1,2,3\ldots}$，i=1,2,3,…。而$\uparrow_{(v,r)}=(v_{CTi},r_{Cti})$ and$\downarrow_{(v,r)}=(-v_{CTi},r_{CTi})$，i=1,2,3,…。Ex：$(A_9\#_2\&_1\#_2\&_1\#_3)$表示A的$CT_1=2$，$CT_2=1$，$CT_3=2$，$CT_4=1$，$CT_5=3$。
- k：尋找的N落在某CT時段上，卻只使用此CT的部份時段，所求出被用那部份之值。
- Mo(motion)：$\{[v_{CT1}*(CT_1-1)]+(v_{CT2}*CT_2)+(v_{CT3}*CT_3)+\ldots+(v_{CTi-1}*k)\}$。在"$(CT_1-1)$"中的-1是因要扣掉最初的那個片段，才表示為經過幾個片段，故要減1。若Mo得到的結果為負，則表示為倒退。
- Size：$\{(E_A-B_A)/2*[(r_{CT1}\hat{}CT_1)*(r_{CT2}\hat{}CT_2)*(r_{CT3}\hat{}CT_3)*...*(r_{CTi-1}\hat{}k)-1]$，若遇到$r_{CT}=0$則略過不計算}，在"$(r_{CTi-1}\hat{}k)-1$"中的-1是因要計算大小改變多少，故要扣掉1才能知道變多少。若Size得到的結果為負，表示變的比原本小。如變為1.5倍，則表示比原本多了0.5倍的自己。變為0.8倍，則表示比原本少了0.2倍的自己。Mo＋Size與Mo－Size的差別，在於物件的大小改變，是以物件的質心往x軸或y軸的兩邊改變。因此，往負的方向改變，begin-bound要減(－)。往正的方向改變，end-bound要加(＋)。

我們以圖1的6個連續片段為例，列出處理過後的符號圖片以及其3D Z^+-string，如圖12所示。假如我們想知道在第5個片段時，A、B兩物件之間的空間關係為何，即可利用QueryFrame algorithm查詢。以下為執行過程：

輸入x-axis上A的$(B_A, E_A)=(0,1)$，B的$(B_B, E_B)=(5,7)$，(y-axis作法相同)，再輸入u-string：$A_1\uparrow_{(0,1)}\uparrow_{(0,0)}\uparrow_{(2,1.2)}\uparrow_{(0,0)}\uparrow_{(2,1.1)}\uparrow_{(1,1.2)}<B_2\downarrow_{(1,1)}$，t-string：$(A_6\#_1\&_1\#_1\&_1\#_1\#_1)=B_6\#_6$。

1. 判斷，在片段5時，A、B都存在則執行。
2. 利用公式找尋，$k=n-(CT_1+CT_2+\ldots+CT_{i-1})=5-(1+1+1+1)=1$，在套近Mo及Size中計算。
3. A改變後的B_A為B_{A0}(到片段5時)
   $B_{A0}=0+\{[1*(1-1)]+(1*2)+(1*2)\}-\{(1-0)/2*[(1\hat{}1)*(1.2\hat{}1)*(1.1\hat{}1)-1]\}=3.84$。
4. A改變後的E_A即為E_{A0}(到片段5時)
   $E_{A0}=1+\{[1*(1-1)]+(1*2)+(1*2)\}+\{(1-0)/2*[(1\hat{}1)*(1.2\hat{}1)*(1.1\hat{}1)-1]\}=5.16$。
5. 得到結果，A的$(B_{A0}, E_{A0})=(3.84,5.16)$。
6. B改變後的B_B為B_{B0}(到片段5時)
   $B_{B0}=5+[1*-(5-1)]-\{(7-5)/2*[(1\hat{}5)-1]\}=1$。
7. B改變後的E_B為E_{B0}(到片段5時)
   $E_{B0}=7+[1*-(5-1)]+\{(7-5)/2*[(1\hat{}5)-1]\}=3$。
8. 得到結果，B的$(B_{B0}, E_{B0})=(1, 3)$。
9. 然後，在比對A的(B_{A0}, E_{A0})和B的(B_{B0}, E_{B0})，判斷後得知為$S_{B0}<_d S_{A0}$關系，且$B_{A0}-E_{B0}=3.84-3=0.84$。
10. 則可知道在片段5的時候，A、B兩objects之間的關係為$B_2<_{0.84} A_1$。

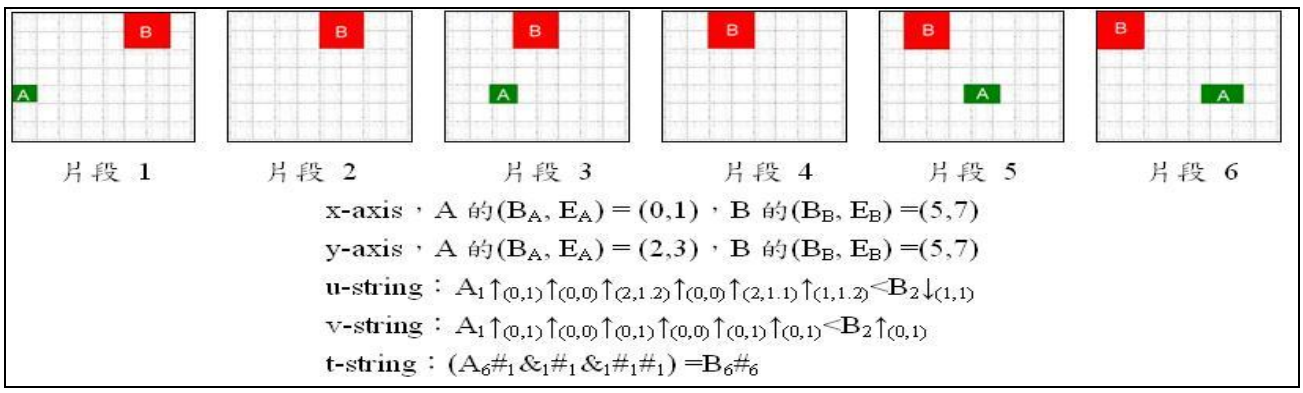


圖 12. 把圖 1 處理成符號圖片後的 6 個連續片段，並將圖 1 以 3D-Z^{+}-string表示

```
Algorithm: QueryFrame
Input: u-string(v-string), t-string，A, B, N
Output: Find out the spatial relationship between A object and B object that in number N frame.
1.    Use u-string (v-string) and t- string to construct the begin-bound and end-bound
      of A and B, when objects in number N frame.
2.    (
3.    While (more than one object in number N frame).
4.        To determine the objects relations of disappear and appear from A object and B object.
5.        If (N is fall in CTi of A, then the CTi  is corresponding rCT=0(&yi) of u-(v-)string,
            it mean the A is disappear. And N is fall in CTi of B, then the CTi
            is corresponding rCT≠ 0(#xi) of u-(v-)string, it mean the B is appear.)
6.            printf("A is disappear, we can't indicate the relations between A and B.");
7.        ElseIf (N is fall in CTi of B, then the CTi  is corresponding rCT=0(&yi) of u-(v-)string,
            it mean the B is disappear. And N is fall in CTi of A, then the CTi
            is corresponding rCT≠ 0(#xi) of u-(v-)string, it mean the A is appear.)
8.            printf("B is disappear, we can't indicate the relations between A and B.");
9.        ElseIf (N is fall in CTi of A, then the CTi  is corresponding rCT=0(&yi) of u-(v-)string,
            it mean the A is disappear. And N is fall in CTi of B, then the CTi
            is corresponding rCT=0(&yi) of u-(v-)string, it mean the B is disappear.)
10.           printf("A andB are disappear, we can't indicate the relations between A and B.");
11.       Else (N is fall in CTi of A, then the CTi  is corresponding rCT≠ 0(#xi) of u-(v-)string,
            it mean the A is appear. And N is fall in CTi of B, then the CTi
            is corresponding rCT≠ 0(#xi) of u-(v-)string, it mean the B is appear.)
12.           (
13.            for loop( If N>CT1, cumulative the next CT's value (CT2), and continue.
                 Then, N>CT1+CT2 cumulative the next CT's value (CT3), and continue.
                 Until , when N<CT1 + CT2+ … +CTi to be stop.
                 And k = N-(CT1 + CT2+ … +CTi-1))
14.           k is use of the above in Mo and Size./*from the interpretation of formula */
15.           BAO= BA+Mo-Size;
16.           EAO= EA+Mo+Size;
17.           BBO= BB+Mo-Size;
18.           EBO= EB+Mo+Size;
19.           Use the new begin-bound and new end-bound of A and B to determine.
20.           )
21.   Begin /* Begin to determine.*/
22.       For the objects with same begin-bound and end-bound, merge them into a new template object by operator "=".
23.       If there is only one object in number N frame, exit the while-loop.
24.       For each object I in number N fram, find its FONest(i).
25.       Find FONome. Let BO be FONome and AO be FONest(Q).
26.       If (BAO= BBO and EAO>EBO) then merge BO and AO into a new template object TO= (BAO, EAO, (SAO [SBO)).
27.       ElseIf(EAO= EBO and BAO<BBO) then merge BO and AO into a new template object TO= (BAO, EAO, (SAO]SBO)).
28.       ElseIf (EAO>BBO and BAO<BBO) then merge BO and AO into a new template object TO= (BAO, EBO, (SAO/dSBO))
            and d=EAO-BBO.
29.       ElseIf (EAO<BBO) then merge BO and AO into a new template object TO= (BAO, EBO, (SAO<dSBO))
            and d= BBO-EAO.
30.       ElseIf (EAO= BBO) then merge BO and AO into a new template object TO= (BAO, E BO, (SAO|SBO)).
31.       ElseIf (EAO>BBO) then merge BO and AO into a new template object TO= (BAO, EAO, (SAO%dSBO))
            and d= BBO-BAO.
32.       EndIf
33.   EndWhile
34.   )
35.   Return the answer.
```

圖 13. QueryFrame algorithm

我們所提出的第二種搜尋方法為QueryObjectsRelations algorithm，如圖 14。主要在尋找兩物件間為某種特定的空間關係，是在第幾個片段(或哪幾個片段)發生。以圖 12 為例，欲查詢A、B兩物件間的空間關係為$A_1/_{0.1}B_2$時，是在第幾個片段發生。執行方式如下：

1. 判斷 Input 投影長度 A 是否為 1，B 是否為 2。
2. 作六層迴圈，然後過濾物件消失的片段。因物件消失無法比對之間的空間關係。故在片段 2 和片段 4 去除後，剩下的片段繼續執行。
3. 算出片段 1，片段 3，片段 5，片段 6 的物件 A 和物件 B 的 begin-bound 及 end-bound。
4. 知道 begin-bound 及 end-bound，就能算出片段 1，片段 3，片段 5，片段 6 時，A、B 兩物件之間的空間關係。
5. 再用 4.得知的結果，與欲尋找的空間關係 $A_1/_{0.1}B_2$去做比對。比對後的結果為片段 3。

```
Algorithm: QueryObjectsRelations
Input: u-string(v-string), t-string, On(Bn,En,Sn), the relations between A and B
Output: Which number of the Frame
1.     While
2.     To determine the projection length of object A and object B
     If SA and SB the same with On(Sn), then continue.
3.     {
4.       To do the determine From Frame 1 to Frame N
5.       Begin
6.       For ( i = 1 ; i ≦ N ; i ++ )  /* i=1,2,3… … */
       {
7.       To determine the objects relations of disappear and appear from A object and B object.
8.     If(Frame i is fall in CTi of A, then the CTi  is corresponding rCT≠ 0(#xi) of u-(v-)string,
         it mean the A is appear. And Frame i is fall in CTi of B, then the CTi
         is corresponding rCT≠ 0(#xi) of u-(v-)string, it mean the B is appear.)
9.        Then continue to do
10.       {
11.        For loop(If i＞CT1, cumulative the next CT's value (CT2), and continue.
             Then, i＞CT1＋CT2 cumulative the next CT's value (CT3), and continue.
             Until , wheni＜CT1 + CT2+ … +CTi to be stop.
             And k = i－(CT1 + CT2+ … +CTi-1))
12.        k is use of the above in Mo and Size./*from the interpretation of formula */
13.        BAO= BA＋Mo－Size;
14.        EAO= EA＋Mo＋Size;
15.        BBO= BB＋Mo－Size;
16.        EBO= EB＋Mo＋Size;
17.        Use the new begin-bound and new end-bound of A and B to determine.
18.        If Frame i with same begin-bound and end-bound between A and B, and the operator
             that input the relations between A and B is "=", then return Frame i
19.        ElseIf Frame i with (BA= BB and EA＞EB) between A and B, and the operator
             that input the relations between A and B is "[", then return Frame i
20.        ElseIf Frame i with (EA= EB and BA＜BB) between A and B, and the operator
             that input the relations between A and B is "]", then return Frame i
21.        ElseIf Frame i with (EA＞BB and BA＜BB) between A and B, and the operator
             that input the relations between A and B is "/d" and d=EA－BB, then return Frame i
22.        ElseIf Frame i with ( BA＜BB) between A and B, and the operator
             that input the relations between A and B is "＜d" and d=BB－EA, then return Frame i
23.        ElseIf Frame i with (EA= BB) between A and B, and the operator
             that input the relations between A and B is "|", then return Frame i
24.        ElseIf Frame i with (EA＞BB) between A and B, and the operator
             that input the relations between A and B is "%d" and d= BB－BA, then return Frame i
25.     Else return Null is mean No Find.
26.        }
27.     Else (Stopped to do, it mean object A or object B that one of the both is disappear)
28.     }
29.    }
30.    EndWhile
31.    List all answers or all frames.
```

圖 14. QueryObjectsRelations algorithm

在此章節中，可知我們的新作法可以改善3D Z-string，無法記錄物件多次出現又消失的情形。同時，我們提出的兩種搜尋方法，可供使用者更方便、強大的影片查詢功能。

四、結論

3D Z-string[3]為李教授等人所提出，用來表示影片中物件之間的空間關係。因為 3D Z-string不需要用到切割機制，故可避免產生過多的子物件及字串過長的問題。且 3D Z-string使用運算子，記錄物件的大小變化和物件移動的變化，令字串可以記錄更詳細的影片資訊。因此，在字串的比對上可以增加搜尋的準確性。但 3D Z-string無法表示出在影片中，物件多次消失又出現的情況。使得影片的資訊記錄有遺漏，會使字串無法重建回原本的影片。在本文中，我們將物件的消失，視為物件大小為“零”，並增加一個新的運算子“&”，用以記錄影片中物件消失的片段。我們提出一個新的時空知識表示法-3D Z^{+}-string，它包含 3D Z-string的定義規範，並改善在影片連續片段中，無法表示出物件多次消失又出現的情形。除此之外，為了強化搜尋功能，我們提出兩種新的搜尋方式。第一種搜尋方式，為搜尋在影片的第N個片段中，兩物件之間的空間關係為何；第二種搜尋方式，為搜尋兩物件之間的空間關係為某種關係時，是發生在第幾個片段(或哪幾個片段)。這兩種方法，可以給相關領域的軟體開發人員去應用，以提供使用者更方便、強大的影片查詢功能。像是影片管理軟體中，使用者可以利用這兩種功能，在影像資料庫中的搜尋想要的特定影片片段及空間關係。有了這兩種功能，使用者就可以更方便地尋找所要的影片。

五、參考文獻

[1] Anthony J.T. Lee, H.P. Chiu, “2D Z-string: A new spatial knowledge representation for image databases,” *Pattern Recognition* Letters vol. 24, pp. 3015-3026, 2003.

[2] Anthony J.T. Lee, H.P. Chiu, “3D C-string as: A new spatio-temporal knowledge structure for video database systems,” *Pattern Recognition*, 35(11), pp.2521-2537, 2002.

[3] Anthony J.T. Lee, Ping Yu, Han-Pang Chiu, Ruey-Wen Hong, “3D Z-string: A new knowledge structure to represent spatio-temporal relations between objects in a video,” *Pattern Recognition Letters*, 26, pp. 2500–2508, 2005.

[4] C. L. Huang, D. H. Huang, ”A content-based image retrieval system,” *Image and Vision Computing*, 16, pp, 149-163, 1998.

[5] E. Jungert, “Extended Symbolic Projection used in a knowledge structure for Spatial reasoning,” *The 4th BPRA Conference on Pattern Recognition. Springer V*, Cambridge, pp.343-351, 1988.

[6] N.S. Chang, K.S. Fu, “Query-by-pictorial Example,” *IEEE Trans. Software Engineering*, 6, pp. 519-524, Nov, 1980.

[7] P.W. Huang and Y.R. Jean, “Using 2D C^{+}-string as spatial knowledge representation for image database systems,” *Pattern Recognition* 27, pp. 1249–1257,1994.

[8] S.K. Chang, Q.Y. Shi and C.W. Yan, “Iconic Indexing by 2D-strings,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 3, pp. 413-428, 1987.

[9] S. Y. Lee and F. J. Hsu, “2D C-string: A New Spatial Knowledge Representation for Image Database Systems,” *Pattern Recognition*, 23, pp. 1077-1088, 1990.