Available online at www.sciencedirect.com
SCIENCE DIRECT®
Pattern Recognition Letters 26 (2005) 2500–2508
Pattern Recognition Letters
www.elsevier.com/locate/patrec

3D Z-string: A new knowledge structure to represent spatio-temporal relations between objects in a video

Anthony J.T. Lee [a,*], Ping Yu [a], Han-Pang Chiu [b], Ruey-Wen Hong [a]

[a] Department of Information Management, No. 1, Section 4, Roosevelt Road, National Taiwan University, Taipei 10617, Taiwan, ROC
[b] Department of EECS, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA

Received 10 September 2004; received in revised form 10 January 2005
Available online 25 July 2005

Communicated by L. Goldfarb

Abstract

In this paper, we propose a new knowledge structure called 3D Z-string, extended from the 2D Z-string, to represent the spatial and temporal relations between objects in a video and to keep track of the motions and size changes of the objects. Since there are no cuttings between objects in the 3D Z-string, the integrity of objects is preserved. The string generation and video reconstruction algorithms for the 3D Z-string representation of video objects are also developed. The string generated by the string generation algorithm is unique for a given video and the video reconstructed from a given 3D Z-string is unique too. The experimental results show that the 3D Z-string is more compact and efficient than the 3D C-string in terms of storage space and execution time.

Keywords: Iconic indexing; Video databases; 3D Z-string; 3D C-string; 2D Z-string

1. Introduction

With the advances of information technologies, the amount and volume of multimedia data, such as images, audios, and videos, rapidly increases. There is an urgent need to develop effective video retrieval and summarization methods (Sebe et al., 2003). To represent spatial and temporal relations between objects in a video is one of the most important methods for retrieving the videos from a database.

To represent the spatial relations between the objects in a symbolic image, many iconic indexing approaches have been proposed, such as, 2D string (Chang and Jungert, 1986), 2D G-string (Jungert,

* Corresponding author. Tel.: +886 2 33661188; fax: +886 2 33661199.
E-mail addresses: jtlee@ntu.edu.tw (A.J.T. Lee), chiu@mit.edu (H.-P. Chiu).

0167-8655/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2005.04.018

1988), 2D C-string (Lee and Hsu, 1990), 2D C^+-string (Huang and Jean, 1994), unique-ID-based matrix (Chang et al., 2000), GPN matrix (Chang et al., 2001), virtual image (Petraglia et al., 2001), BP matrix (Chang et al., 2003) and 2D Z-string (Lee and Chiu, 2003).

To represent the spatial and temporal relations between the objects in a symbolic video, many iconic indexing approaches, extended from the notion of 2D string to represent the spatial and temporal relations between objects in a video, have been proposed, such as, 2D B-string (Shearer et al., 1996), 2D C-Tree (Hsu et al., 1998), 9DLT strings (Chan and Chang, 2001), 3D-list (Liu and Chen, 2002), and 3D C-string (Lee et al., 2002). The 3D-list approach (Liu and Chen, 2002) uses the projection of the objects to represent spatial and temporal relations between them. The basic idea is to project the objects onto the x-, y-, and time-axes to form three strings representing the relative positions of the projections in the x-, y-, and time-axes, respectively. However, the 3D-list approach only records the central point and starting frame number for an object. So the spatial overlapping relations and precise temporal relations between the objects cannot be realized. The information about the motions and size changes of objects was omitted in their work.

To resolve such a problem, Lee et al. (2002) proposed the 3D C-string approach, extended the concept of the 2D C^+-string, to represent the spatial and temporal relations between objects in a video and to keep track of the motions and size changes of the objects. By using the same cutting mechanism of the 2D C-string and 2D C^+-string, the 3D C-string approach may produce $O(n^2)$ cut sub-objects in the generated u-, v- or t-strings, where n is the number of objects in the video.

To reduce the number of cut sub-objects in the generated strings and preserve the integrity of objects, in this paper, we propose a new knowledge structure called 3D Z-string. The 3D Z-string, extended from the concept of the 2D Z-string, can represent the spatial and temporal relations between objects in a video and keep track of the motions and size changes of the objects. The string generation and video reconstruction algorithms are also developed. The string generated by the string generation algorithm is unique for a given video and the video reconstructed from a given 3D Z-string is unique too. Since there is no cutting between the objects in the video, the 3D Z-string is more compact and efficient than the 3D C-string approach in terms of storage space and execution time.

The rest of this paper is organized as follows. In Section 2, we present the new spatial knowledge representation of the 3D Z-string. The string generation algorithm of the 3D Z-string is described in Section 3. In Section 4, we propose the reconstruction algorithm based on the 3D Z-string representation. In Section 5, some experiments are conducted to compare our proposed approach with the 3D C-string approach. Finally, concluding remarks are made in Section 6.

2. 3D Z-string approach

In the 3D Z-string, the objects in a video are projected onto the x-, y-, and time-axes to form three strings representing the relations and relative positions of the projections in the x-, y-, and time-axes, respectively. These three strings are called u-, v-, and t-strings. The projections of an object onto the x-, y-, and time-axes are called x-, y-, and time-projections, respectively. In the 2D C-string, Lee and Hsu (1990) proposed 13 possible relations between the x- (or y-) projections as shown in Table 1, where Begin(A) and End(A) are the begin-bound (beginning point) and end-bound (ending point) of the x- (or y-) projection of object A, respectively. One of them is "equal" relation and six of them are symmetric relations of the others. Hence, those

Table 1
The definitions of spatial operators in 2D C-string

Notations	Conditions
$A < B$	End(A) < Begin(B)
$A = B$	Begin(A) = Begin(B), End(A) = End(B)
$A \mid B$	End(A) = Begin(B)
$A \,\%\, B$	Begin(A) < Begin(B), End(A) > End(B)
$A [B$	Begin(A) = Begin(B), End(A) > End(B)
$A] B$	Begin(A) < Begin(B), End(A) = End(B)
A / B	Begin(A) < Begin(B) < End(A) < End(B)

relations can be represented by seven spatial operators. In the time dimension, there are 13 temporal relations between the time-projections, too. So we use the temporal operators as the same as the spatial operators. For example, in the x- (or y-) dimension, "$A < B$" denotes that object A is before that of object B. In the time dimension, "$A < B$" denotes that object A disappears before object B appears.

In the 3D Z-string, like the 3D C-string, for each object, we keep track of the initial location and size of the object by a minimum bounding rectangle (MBR) whose sides are parallel to the x- or y-axes. We also record the information about their motions and size changes. To record the time points of motion and size changes, the 3D C-string introduces a temporal operator, "$|_t$". It denotes an object contains motion state changing. For example, $A_2 |_t A_6$ denotes that in the first 2 frames, object A is in the same motion state and from the third frame to the eighth frame, the motion state is changed into another. In the 3D Z-string, we introduce a new temporal operator "#" to represent the interval of motion and size changes. So, $A_2 |_t A_6$ in the 3D C-string can be represented as $A_8 \#_2 \#_6$ in the 3D Z-string.

The knowledge structure of 3D Z-string is defined as follows.

Definition 1. The 3D Z-string is a 6-tuple (O, A, R_g, R_l, R_t, S) where

(1) O is a set of objects in a video;
(2) A is a set of attributes to describe the objects in O;
(3) $R_g = \{$"<", "|"$\}$ is the set of global relation operators;
(4) $R_l = \{$" = ", "[", "]", "%", "/"$\}$ is the set of local relation operators;
(5) $R_t = \{$"↑", "↓", "#"$\}$, where "↑" and "↓" are the motion operators to denote the direction of the motion of an object, and they are only used in the u- and v-strings. Operator "↑" ("↓") denotes that the object moves along the positive (negative) direction of the x- (or y-) axis. "#" is the interval operator to denote the interval of a motion/size change. It is only used in the t-string.
(6) S = {"(", ")"} contains a pair of separators which are used to describe a set of objects as a spatial or temporal template object (see Section 3 for the detail).

Most metric measures in the 3D Z-string are the same as those in the 2D Z-string except that the 3D Z-string has the temporal metric measure of interval operator "#". Those metric measures are listed as follows:

1. The size of an object: A_s denotes the size of its x- (y- or time-) projection is equal to s, where $s = \mathrm{End}(A) - \mathrm{Begin}(A)$, $\mathrm{Begin}(A)$ and $\mathrm{End}(A)$ are the begin-bound and end-bound of the projection of object A in the x- (y- or time-) axis, respectively.
2. The distances associated with operator $<$, % and /: The distances d associated with $A <_d B$, $A \%_d B$, and $A /_d B$ are equal to $\mathrm{Begin}(B) - \mathrm{End}(A)$, $\mathrm{Begin}(B) - \mathrm{Begin}(A)$, and $\mathrm{End}(A) - \mathrm{Begin}(B)$, respectively.
3. The velocity and rate of size change associated with motion operators $\uparrow_{v,r}$ and $\downarrow_{v,r}$: Operators $\uparrow_{v,r}$ and $\downarrow_{v,r}$ have two subscripts (fields). "v" is the velocity of the motion and "r" is the rate of size change of the object. For example, an u-string: $A\uparrow_{2,1}$ denotes that object A with the velocity = 2 and the rate of size change = 1. That is, the velocity of object A moves along the positive direction with 2 units/frame and the size remains unchanged.
4. The interval associated with operator #: $\#_i$ denotes that the interval length associated with the motion/size change is equal to i. If an object has not motion or size changing, there is not an interval operator associated with the object in the t-string.

To see how the 3D Z-string works, let us consider the example as shown in Fig. 1. The projections of the initial locations of objects A, B, C, and D in the x- and y-axes are shown in Fig. 1(b). The corresponding 3D Z-string of the video is shown in Fig. 1(c).

From frame 1 to frame 2, object (car) A moves along the x-axis with the velocity of 4 units/frame. In frame 3, it changes its velocity into 3 units/frame.

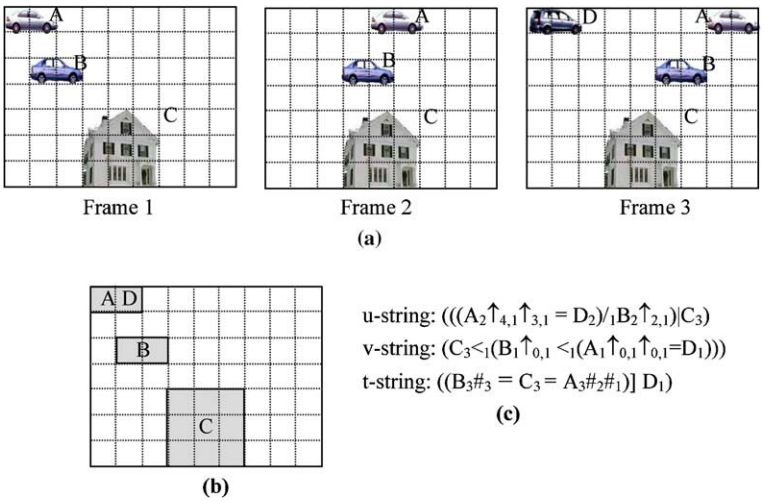

Fig. 1. An example video and the corresponding 3D Z-string. (a) A video contains three frames. (b) Projecting the initial locations of objects A, B, C and D onto the x–y plane. (c) The corresponding 3D Z-string.

From frame 1 to frame 3, object A has no motion in the y-axis. So A_2 is followed by operators $\uparrow_{4,1}\uparrow_{3,1}$ in the u-string, A_1 followed by operator $\uparrow_{0,1}\uparrow_{0,1}$ in the v-string and A_3 followed by operator $\#_2\#_1$ in the t-string. From frame 1 to frame 3, object (car) B moves along the x-axis with the velocity of 2 units/frame, but no motion along the y-axis. So, B_2 is followed by operators $\uparrow_{2,1}$ in the u-string, B_1 followed by operator $\uparrow_{0,1}$ in the v-string and B_3 followed by operator $\#_3$ in the t-string. From frame 1 to frame 3, there is no motion for object (house) C along the x- (or y-) axis. Object (car) D appears in frame 3 with the same initial location and size of object A. Therefore, the 3D Z-string provides an easy and efficient way to represent the spatio-temporal relations between objects in a video.

3. String generation algorithm

We first introduce the notations using in the *string generation algorithm*. Assume that there are n objects in the video.

B_i/E_i: the begin-bound/end-bound of object i.

S_i: the object string of object i. To generate the u- (v- or t-) string, we first generate a list of object strings. In the x (or y) dimension, S_i contains the information of the size, motions and size changes of object i. In the time dimension, S_i contains the interval length and a list of interval operators recording the intervals of motions/size changes for object i.

SQ: same-value-list sequence. Sort all B_i and E_i, $i = 1,2,\ldots,n$ in non-decreasing order and group the same value points into a same-value list which contains at least one point. Those same-value lists form a same-value-list sequence SQ.

DO: dominating object (Lee and Hsu, 1990). For the objects with the same end-bound, those end-bounds are grouped into a same-value list. Among them, an object with the smallest begin-bound is called the dominating object.

$FO(i)$: former object (Lee et al., 2002). The former object of object i is the object with smaller begin-bound than object i or with equal begin-bound and bigger end-bound than object i.

$FO_{Nest}(i)$: nearest former object (Lee et al., 2002). The nearest former object of object i is the former object with the biggest begin-bound. If the number of such objects is more than one, choose one with the smallest end-bound as the nearest former object.

FO_{None}: the object is not the nearest former object of any other objects.

To generate strings, for each object we first generate S_i for each object i and use (B_i, E_i, S_i), $i = 1, 2, \ldots, n$ as the input to the *string generation algorithm*, where n is the number of objects in the video. Second, find all dominating objects (*DO*s) by scanning from left to right along the x- (y- or time-) dimension. For each *DO*, if no object partly overlaps with the *DO*, find the objects covered by the *DO* (including the *DO*). Otherwise, if any objects partly overlap with the *DO*, 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 the object with the smallest begin-bound. Let the chosen object be *PO*. Then find the objects covered by the range from the begin-bound of *DO* to the end-bound of *PO*. The covered objects are merged into a template object (*TO*). Repeat the above processes for each *DO*.

Finally, we can merge together those template objects and the remaining objects. This is the main idea of the *string generation algorithm* as shown in Fig. 2. How to merge objects into a template object is described later in the *templateObjectGeneration function* as shown in Fig. 3.

Now, let us demonstrate how to apply the *string generation algorithm* to obtain the u-string for the example as shown in Fig. 1. To generate the u-string, the initial locations of objects A, B, C and D are projected onto the x–y plane as shown in Fig. 1(b). Next, we generate the object string in the x dimension for each object: $A_2\uparrow_{4,1}\uparrow_{3,1}$, $B_2\uparrow_{2,1}$, C_3, and D_2. Then, those objects $A(0, 2, A_2\uparrow_{4,1}\uparrow_{3,1})$,

Algorithm: string generation

Input: an object list $O_1(B_1, E_1, S_1)$, $O_2(B_2, E_2, S_2)$,…, $O_n(B_n, E_n, S_n)$ of the x- (y- or time-) projections, where n is the number of objects.

Output: an u- (v- or t-) string

1. Sort B_i, E_i, i=1,2, …, n in non-decreasing order and group the same value points into a same-value list. Form 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 TO_1 = *templateObjectGeneration*(the objects i with $B_i \geq B_{PO}$ and $E_i \leq E_{PO}$) and remove those B_i and E_i from *SQ*.
9. Let TO_2 = *templateObjectGeneration*(the objects i with $B_i \geq B_{DO}$ and $E_i \leq E_{DO}$) and remove those B_i and E_i from *SQ*.
10. Merge TO_1 and TO_2 into *TO*, where $B_{TO} = B_{TO2}$, $E_{TO} = E_{TO1}$, $S_{TO} = (S_{TO2} /_d S_{TO1})$ and $d = E_{TO2} - B_{TO1}$.
11. **Else**
12. Let *TO* = *templateObjectGeneration*(the objects i with $B_i \geq B_{DO}$ and $E_i \leq E_{DO}$) and remove those B_i and E_i from *SQ*.
13. **EndIf**
14. Add B_{TO} and E_{TO} to *SQ*.
15. **EndIf**
16. **EndFor**
17. *TO* = *templateObjectGeneration*(all the remaining objects)
18. Output S_{TO}.

Fig. 2. 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 $FO_{Nest}(i)$.
6. Find FO_{None}. Let Q be FO_{None} and N be $FO_{Nest}(Q)$.
7(a). **If** ($B_N = B_Q$ and $E_N > E_Q$) **then** merge Q and N into a new template object $TO = (B_N, E_N, (S_N[S_Q))$.
7(b). **ElseIf** ($E_N = E_Q$ and $B_N < B_Q$) **then** merge Q and N into a new template object $TO = (B_N, E_N, (S_N]S_Q))$.
7(c). **ElseIf** ($E_N > B_Q$ and $B_N < B_Q$) **then** merge Q and N into a new template object $TO = (B_N, E_Q, (S_N/_d S_Q))$ and $d = E_N - B_Q$.
7(d). **ElseIf** ($E_N < B_Q$) **then** merge Q and N into a new template object $TO = (B_N, E_Q, (S_N <_d S_Q))$ and $d = B_Q - E_N$.
7(e). **ElseIf** ($E_N = B_Q$) **then** merge Q and N into a new template object $TO = (B_N, E_Q, (S_N|S_Q))$.
7(f). **ElseIf** ($E_N > E_Q$) **then** merge Q and N into a new template object $TO = (B_N, E_N, (S_N \%_d S_Q))$ and $d = B_Q - B_N$.
8. **EndIf**
9. Remove Q and N from W and add TO to W.
10. **EndWhile**
11. Return the template object.

Fig. 3. templateObjectGeneration function.

$B(1,3,B_2\uparrow_{2,1})$, $C(3,6,C_3)$, and $D(0,2,D_2)$ are the input to the *string generation algorithm*.

In step 5, the first DO is A. Since objects A and D have the same end-bound and the begin-bound and object B partly overlaps with object A. In step 8, only one object is covered by object B (B itself). In step 9, the object covered by object A is object D. Call the *templateObjectGeneration function* with objects A and D as the input parameter. In the first while-loop of the *templateObjectGeneration function*, steps 3 and 4 are executed. The template object $TO_1(0,2,(A_2\uparrow_{4,1}\uparrow_{3,1} = D_2))$ is generated. In step 10, object B and template object $TO_1(0,2,(A_2\uparrow_{4,1}\uparrow_{3,1} = D_2))$ are merged into a template object $TO_2(0,3,((A_2\uparrow_{4,1}\uparrow_{3,1} = D_2)\ /_1\ B_2\uparrow_{2,1}))$.

In step 17, input all the remaining objects: template object $TO_2(0,3,((A_2\uparrow_{4,1}\uparrow_{3,1} = D_2)\ /_1\ B_2\uparrow_{2,1}))$ and object C to the *templateObjectGeneration function.* In the first while-loop, steps 4, 5, 6, 7(e), 8, 9, 10 and 11 are executed. The template object $TO_3(0,6,(((A_2\uparrow_{4,1}\uparrow_{3,1} = D_2)\ /_1\ B_2\uparrow_{2,1})\ |\ C_3))$ is generated. Since all the objects are merged together as a template object. So, the corresponding u-string of the video shown in Fig. 1 can be represented as $(((A_2\uparrow_{4,1}\uparrow_{3,1} = D_2)\ /_1\ B_2\uparrow_{2,1})|C_3)$.

Theorem 1. *For an input list containing n objects, the string generation algorithm generates a unique u- (v- or t-) string.*

4. Video reconstruction algorithm

In this section, we present the *video reconstruction algorithm* which converts an u- (v- or t-) string to construct the locations, motions and size changes for the objects in the x (or y) dimension or to construct the starting frame, duration and intervals of motions and size changes in the time dimension.

Assume that a given u- (or v- or t-) string consist of n elements, each of which may be a string object, or an operator of "<", "%", "/", "|", "]", "[", "↑", "↓", or "#". For each element E, $E.sym$ represents the symbol of a string object or an operator and $E.size$ represents the size associated with E. For a string object, $E.size$ represents the initial size in the x (or y) dimension, or the interval during which E appears in the time dimension. For a relation operator of "<", "%" or "/", $E.size$ represents the metric measure of the relation. For a relation operator of "|", "[", "]", $E.size$ is set to zero.

For a motion operator of "↑" or "↓", *E.size* has two fields, (v, r), to record the velocity and rate of size change for the associated string object. For an interval operator "#", *E.size* represents the interval during which the motion/size change of the associated string object lasts. Similarly, a video

```
Algorithm: video reconstruction
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; MoreOperatosr ←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 ←True;
14.         "↓": Append (-v, r) to Oj.motionList;  MoreOperators ←True;
15.         "#": Append Ei.size to Oj.intervalList;  MoreOperators ←True;
16.         ")": Pop an element E from Stack;
                 Loc ← E.beginBound;        /* E is a template object */
             PreviousObjectSize ← E.size;  MoreOperators ←True;
17.       EndCase
18.    EndWhile
19.    While (Ei.sym = "(")
20.       Create a template object E;
21.       E.beginBound ← Loc;
22.       E.size ← GetTemplateSize(i,string);
23.       Push the template object E onto Stack;
24.       i ← i +1;
25.    EndWhile
26.    If Ei is a string object then
27.       j ←j +1;
28.       Create a new object Oj so that
29.          Oj.sym ← Ei.sym; Oj.size ← Ei.size; Oj.beginBound ← Loc;
30.       Append object Oj to ObjectList.
31.    EndIf
32.    PreviousObjectSize ← Ei.size;
33.    MoreOperators ←True;
34. EndWhile
35. Output the ObjectList.
```

Fig. 4. Video reconstruction algorithm.

object O contains four fields: $O.sym$, $O.size$, $O.location$, and $O.motionList$ (or $O.intervalList$). For an u- (or v-) string, $O.sym$, $O.size$, and $O.location$, represent the symbol, size, and location of object O respectively. $O.motionList$ is used to record a list of velocities and rates of size changes of object O. For a t-string, $O.sym$, $O.size$, and $O.location$, represent the symbol, appearing interval, and starting frame of object O. $O.intervalList$ is used to record a list of intervals for each motion/size change state of object O.

Assume that there are n elements in a given u- (v- or t-) string and m string objects in the n elements. The *video reconstruction algorithm* converts the given u- (v- or t-) string into a sequence of m video objects. After all video objects are derived from the given u- (v- or t-) string, we have finished the video reconstruction. The *video reconstruction algorithm* is described in detail in Fig. 4.

The function GetTemplateSize calculates the size of the template object which is equal to the summation of:

1. the size of the first element after "(",
2. the size of the element after global operator "<",
3. the size of the element after global operator "|",
4. the distance associated with the operator "<",
5. the size of the element after the operator "/",
6. the negative distance with the operator "/".

Notice that it is not necessary to extend the calculation to the third or lower levels since the size of a template object at the second level has already included the sizes of all the template objects at the third or lower levels.

Theorem 2. *For a given u- (v- or t-) string, the list of video objects generated by the video reconstruction algorithm is unique.*

5. Performance analysis

To show the efficiency of our proposed approach, we perform two series of experiments to compare our proposed approach with the 3D C-string approach. All the algorithms are implemented on an IBM compatible personal computer of Pentium IV-2.0G with Windows 2000.

The first series of experiments is made on the synthesized videos with three freely set cost factors: the number of videos, the number of objects and the number of frames in a video. We compare our *string generation and video reconstruction algorithms* with those of the 3D C-string by using synthesized videos. The execution time of every experiment is measured by the average elapsed time of video processing. We generate 1000 videos, each of which contains 5000 frames. The experiment result shows that the execution time grows linearly as the number of videos increases from 200 to 1000 videos and the 3D C-string approach spends about 10–60% more time than the 3D Z-string approach to generate a string or to reconstruct a video. In the experiment result of string length versus the number of objects in a video, the string length grows sharply as the number of objects in a video increases from 5 to 160 objects for the 3D C-string approach. The 3D C-string approach requires about 10–200% more storage than the 3D Z-string approach.

The second series of experiments is made on the real videos. In our real video database contain two types of videos: 50 traffic videos and 50 news videos that contain objects of cars, people, buildings, etc. Since the performance of both algorithms depends on the number of objects, the average number of objects contained in a real video is 47.5 (traffic: 64, news: 31). All videos are around one minute long. In general, we index one to 10 objects from each frame. Typically, a video of one minute long contains 1800 frames. To represent the movements of the objects, at least a frame should be indexed for every 10 frames. The experiment result also shows that the execution time grows linearly as the number of videos increases from 10 to 50 videos. The execution time of processing traffic videos is larger than that of processing news videos since traffic videos contain about twice as many objects as news videos. It is shown that the different types of videos have different costs of generating and reconstructing a video in the real video database. Moreover, the longer the generated string is, the more execution time and storage is required. Generally speaking, the 3D C-string

approach spends about 40–200% more time to generate a string or to reconstruct a video than the 3D Z-string approach does.

6. Conclusions

In this paper, we propose a new spatio-temporal knowledge representation called 3D Z-string, which is extended from the concept of 2D Z-string, to represent spatial and temporal relations between objects in a video. The string generation and video reconstruction algorithms are also developed. The string generated by the string generation algorithm is unique for a given video and the video reconstructed from a given 3D Z-string is unique too. By inheriting the property of the 2D Z-string, there are no cuttings between objects in the 3D Z-string, so the integrity of objects is preserved. If we discard the motion and interval operators in the generated string, like the 2D Z-string (Lee and Chiu, 2003), the length of the generated 3D Z-string is bounded by $O(n)$ where n is the number of objects in the video. However, the length of the generated 3D C-string is bounded by $O(n^2)$ since the 3D C-string inherits the property of the 2D C^+-string (Huang and Jean, 1994). The shorter the generated string is, the less storage space and execution time are required for string generation and video reconstruction. So, the 3D Z-string approach is more compact and efficient than the 3D C-string approach in terms of storage requirement and execution time. Our proposed knowledge structure can be easily applied to a video database system to reason about spatio-temporal relations between objects in a video.

Acknowledgement

The authors are grateful to the referees for helpful comments and suggestions. This research was supported in part by Center for Information and Electronics Technologies (CIET), National Taiwan University.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.patrec.2005.04.018.

References

Chang, S.K., Jungert, E., 1986. A spatial knowledge structure for image information systems using symbolic projections. In: Proceedings of Fall Joint Computer Conference, pp. 79–86.

Chan, Y.K., Chang, C.C., 2001. Spatial similarity retrieval in video databases. J. Visual Commun. Image Represent. 12, 107–122.

Chang, Y.I., Ann, H.Y., Yeh, W.H., 2000. A unique-ID-based matrix strategy for efficient iconic indexing of symbolic pictures. Pattern Recognition 33, 1263–1276.

Chang, Y.I., Yang, B.Y., Yeh, W.H., 2001. A generalized prime-number-based matrix strategy for efficient iconic indexing of symbolic pictures. Pattern Recognition Lett. 22, 657–666.

Chang, Y.I., Yang, B.Y., Yeh, W.H., 2003. A bit-pattern-based matrix strategy for efficient iconic indexing of symbolic pictures. Pattern Recognition Lett. 24, 537–545.

Hsu, F.J., Lee, S.Y., Lin, B.S., 1998. Video data indexing by 2D C-trees. J. Visual Languages Comput. 9, 375–397.

Huang, P.W., Jean, Y.R., 1994. Using 2D C^+-string as spatial knowledge representation for image database systems. Pattern Recognition 27, 1249–1257.

Jungert, E., 1988. Extended symbolic projections as a knowledge structure for spatial reasoning. In: Proceedings of 4th BPRA Conference on Pattern Recognition, pp. 343–351.

Lee, A.J.T., Chiu, H.P., 2003. 2D Z-string: A new spatial knowledge representation for image databases. Pattern Recognition Lett. 24, 3015–3026.

Lee, A.J.T., Chiu, H.P., Yu, P., 2002. 3D C-string: A new spatio-temporal knowledge structure for video database systems. Pattern Recognition 35, 2521–2537.

Lee, S.Y., Hsu, F.J., 1990. 2D C-string: A new spatial knowledge representation for image database systems. Pattern Recognition 23, 1077–1087.

Liu, C.C., Chen, A.L.P., 2002. 3D-list: A data structure for efficient video query processing. IEEE Trans. Knowledge Data Eng. 14 (1), 106–122.

Petraglia, G., Sebillo, M., Tucci, M., Tortora, G., 2001. Virtual images for similarity retrieval in image databases. IEEE Trans. Knowledge Data Eng. 13 (6), 951–967.

Sebe, N., Lew, M.S., Smeulders, A.W.M., 2003. Video retrieval and summarization. Comput. Vision Image Understanding 92, 141–146.

Shearer, K., Venkatesh, S., Kieronska, D., 1996. Spatial indexing for video databases. J. Visual Commun. Image Representation 7 (4), 325–335.