LAMP-TR-018
CAR-TR-889
CS-TR-3916

MDA 9049-6C-1250
July 1998

Video Summarization by Curve Simplification

Daniel DeMenthon, Vikrant Kobla and
David Doermann

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUL 1998		**00-07-1998 to 00-07-1998**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Video Summarization by Curve Simplification	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Language and Media Processing Laboratory,Institute for Advanced Computer Studies,University of Maryland,College Park,MD,20742-3275**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**21**	

Video Summarization by Curve Simplification

Daniel DeMenthon, Vikrant Kobla and David Doermann

Language and Media Processing Laboratory (LAMP)
University of Maryland
College Park, MD 20742-3275

Abstract

A video sequence can be represented as a trajectory curve in a high dimensional feature space. This *video curve* can be analyzed by tools similar to those developed for planar curves. In particular, the classic binary curve splitting algorithm has been found to be a useful tool for video analysis. With a splitting condition that checks the dimensionality of the curve segment being split, the video curve can be recursively simplified and represented as a tree structure, and the frames that are found to be junctions between curve segments at different levels of the tree can be used as keyframes to summarize the video sequences at different levels of detail. These keyframes can be combined in various spatial and temporal configurations for browsing purposes. We describe a simple video player that displays the keyframes sequentially and lets the user change the summarization level on the fly with a slider. We also describe an approach to automatically selecting a summarization level that provides a concise and representative set of keyframes.

The support of this research by the Department of Defense under contract MDA 9049-6C-1250 is gratefully acknowledged.

1 Introduction

1.1 Significance of the Problem

Recent advances in digital technology have promoted video as a valuable information resource. We can now access selected clips from archives of thousands of hours of video footage almost instantly. This new resource is exciting, yet the sheer volume of data makes any retrieval task overwhelming and its efficient usage impossible. Browsing tools that would allow the user to quickly get an idea of the content of video footage are still important missing components in these video database systems. Fortunately, the development of browsing tools is a very active area of research [16, 19, 20], and powerful solutions are on the horizon. Browsers use as building blocks sets of frames called keyframes, selected because they summarize the video content better than other frames. Obviously, selecting one keyframe per shot does not adequately summarize the complex information content of a long shot in which camera pan and zoom as well as object motion progressively unveil entirely new situations. A shot should be sampled by a higher or lower density of keyframes according to its activity level.

Sampling techniques that attempt to detect significant information changes simply by looking at pairs of frames or even several consecutive frames are bound to lack robustness in the presence of noise, such as jitter occurring during camera motion or sudden illumination changes due to fluorescent light flicker, glare, and photographic flash. Interestingly, methods developed to detect *perceptually significant* points and discontinuities on *noisy 2D curves* have successfully addressed this type of problem, and can be extended to the multidimensional curves that represent video sequences.

In this paper, we describe an algorithm that can decompose a curve originally defined in a high-dimensional space into curve segments of low dimension. In particular, a video sequence can be mapped to a high-dimensional polygonal trajectory curve by mapping each frame to a *time-dependent feature vector*, and representing these feature vectors as points. We can apply this algorithm to segment the curve of the video sequence into low-dimensional curve segments or even straight line segments. These segments correspond to video footage where activity is low and frames are redundant. The idea is to *detect the constituent segments of the video curve rather than attempt to locate the junctions between these segments directly.* In such a dual approach, the curve is decomposed into segments which exhibit linearity or low dimensionality. Curvature discontinuities are then assigned to the junctions between these segments. Detecting general structure in the video curves to derive frame locations of features such as cuts and scene transitions, rather than attempting to locate the features themselves by local analysis of frame changes, ensures that the detected positions of these features are more stable in the presence of noise, which is effectively filtered out.

In addition, the proposed technique builds a binary tree representation of a video sequence where branches contain frames corresponding to more detailed representations of the sequence. The user can view the video sequence at coarse or fine levels of detail, zooming in by displaying keyframes corresponding to the leaves of the tree, or zooming out by displaying keyframes near the root of the tree.

The general operation of decomposing a high-dimensional video curve into low-dimensional segments has other advantages besides detecting cuts or summarizing movies to controllable

levels of detail. For example, low-dimensional segments can be visualized as 3D plots more faithfully than the original curve, because distances are better preserved when projecting, say, 15D points to 3D than when projecting 1800D points to 3D. Furthermore, data analyses such as those we developed in previous work (*Video Trail* analysis [10]) require less computation for the low-dimensional curve segments than for the original curve because a much smaller number of coordinates is involved.

1.2 Approach and Road Map

In the classic curve splitting algorithm developed by Ramer and others in the early 70's, a curve is recursively split into curve segments until these curve segments can be replaced by line segments. This replacement can occur if the distance from the curve to the line is small, i.e. if the local second dimension is small. A useful contribution of this paper is to show that this algorithm can naturally be extended to solve the problem of dividing a curve of dimension N into curve segments of any dimension between 1 and N (Section 2).

The choice of the proper feature vectors to characterize individual frames contributes to the dimensionality and general shape of the video curve. If the curve segment for a camera *shot* remains in a very low-dimensional space even when camera motions and illumination variations occur during the shot, the curve splitting algorithm will segment the video sequences into shots at the first steps of the recursion. We examine three types of feature vectors for the frames, "raw vectors" derived from the chrominance and luminance of MPEG DC coefficients, and two types of "centroid vectors" whose coordinates are the centroids of pixels in various luminance and chrominance ranges, together with the counts of pixels in different luminance and chrominance ranges, because such feature coordinates evolve linearly with the camera motion, and are more discriminative than color histograms. We can either use all pixels, or only connected pixels from the largest blobs in each luminance and chrominance range (Section 3).

For each frame, a tree depth level as well as a coarseness parameter are computed (Section 4). The result of this analysis is used to produce a hierarchical presentation of video clips in which the user can browse the clips at different tree levels or at different coarseness levels (Section 5). Finally, we address the issue of automatically extracting from a video sequence a concise set of keyframes that best summarizes the sequence (Sections 6 and 7).

2 Recursive Multidimensional Curve Splitting Algorithm

2.1 Background

Techniques for the approximation or simplification of curves have received considerable attention when these curves are planar. The goal of these techniques is to summarize the general shape of the curve with a few perceptually significant points. The variation of the curve between these points is small and includes noise. The curve is often approximated by a straight line between the perceptually significant points.

Many of the ideas developed to summarize planar curves can be extended to produce methods able to summarize high-dimensional video curves. Of particular interest and worth generalizing to video curves are curve approximations that produce tree representations of the curves. One of the

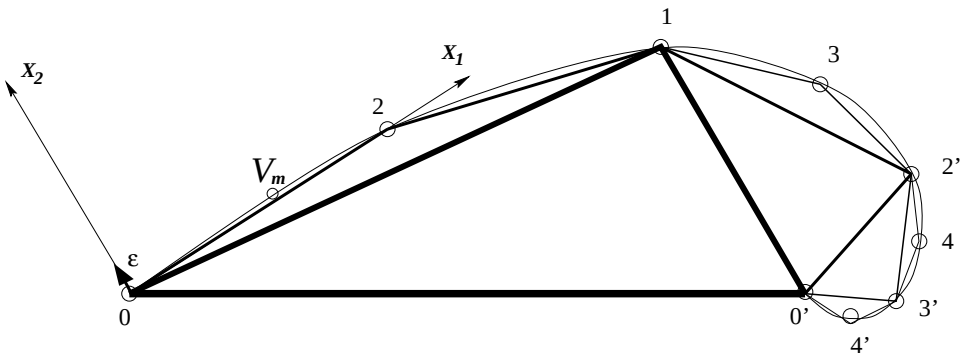

Figure 1: Example of polygon generation by binary curve splitting for a planar curve. Vertex numbers correspond to the depth of the binary tree generated by the splitting recursion.

best known curve simplification methods for planar curves is the recursive binary curve splitting algorithm. It was proposed by Ramer [13] in 1972 for image processing, and also reported by Douglas and Peucker [3] in 1973 for cartographic applications. According to Hershberger and Snoeyink [7], this algorithm was also shown to be "mathematically superior" [9] and to provide the best perceptual representation of the original curves according to 86 percent of sample subjects [17]. It is usually presented as follows [7] (Fig. 1 and Fig. 2):

To approximate the chain of points from V_i to V_j, start with the line segment V_iV_j. If the furthest point V_m from this line segment has a distance d_m *smaller* than a given coarseness parameter ϵ, then accept this line segment as the approximation of the chain of points. Otherwise, split the chain V_iV_j at this vertex V_m, and recursively approximate the two curve pieces V_iV_m and V_mV_j.

For planar curves, this can be also expressed as follows. To approximate the 2D curve V_iV_j with a chain of 1D segments, construct a 2D frame of reference by taking the first axis as V_iV_j, and the second axis perpendicular to it. Find the point V_m with the second coordinate of the largest magnitude. If this value is smaller than ϵ, then the second dimension of the 2D curve V_iV_j can be neglected, and the 2D curve can be approximated by its projection on the 1D space of the line segment V_iV_j. This is illustrated along segment (0, 2) of the curve of Fig. 1. A local Cartesian coordinate system (Ox_1, Ox_2) is constructed, and the point V_m with the largest x_2 coordinate is found to have an x_2 coordinate smaller than ϵ, so no splitting is needed at V_m.

This second description of the algorithm shows that what is performed along the initial curve is a sequence of local dimension reductions from 2D to 1D.

Ramer mentioned that his algorithm could reduce not only planar curves but also curves in an ND space [13]. However, a more powerful and more interesting generalization for multidimensional curves consists of simplifying the curve into curve segments of lower dimension n, such that the magnitudes of the coordinates for the dimensions that are being neglected are smaller than those for the dimensions that are kept, and smaller than the coarseness parameter ϵ. This approach is of special interest for the simplification of a video curve if the curve sections corresponding to shots with small camera pans and zooms evolve in low-dimensional spaces and fit better in nD local spaces than in the 1D spaces of line segments.

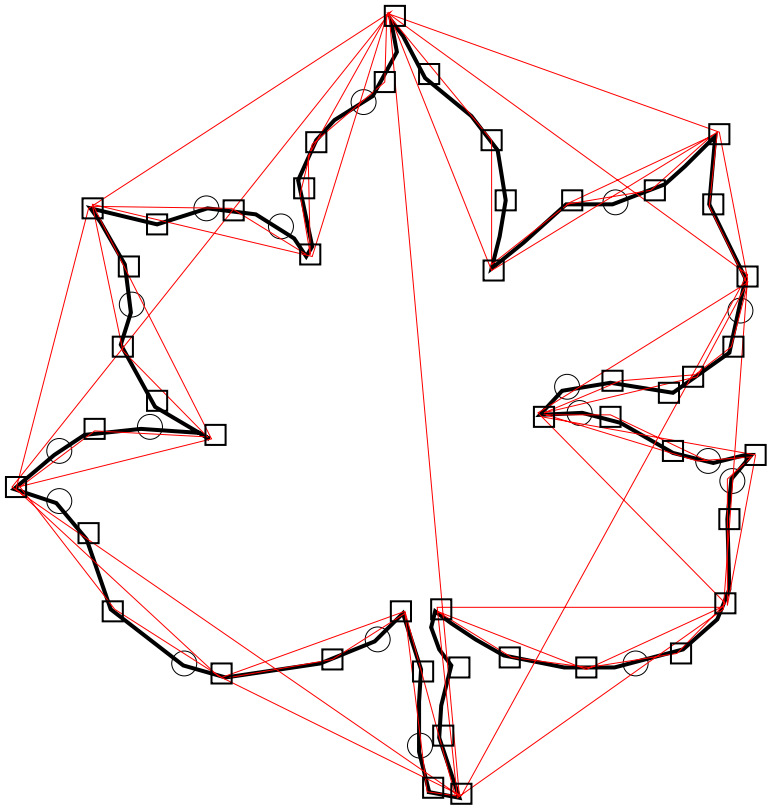

Figure 2: Curve simplification by binary curve splitting at a prescribed coarseness level (squares) and vertices added at the next (finer) coarseness level (circles). Vertices naturally fall at perceptually significant locations

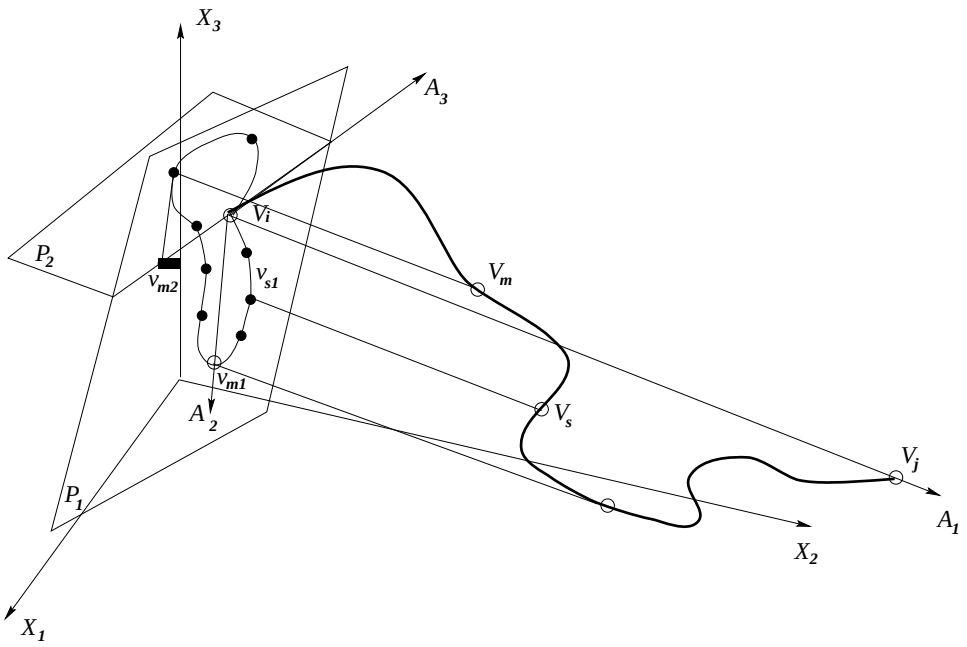

Figure 3: Binary curve splitting with splitting condition based on local third dimension

2.2 Algorithm Description

In this curve splitting algorithm, a local Cartesian coordinate system with n orthogonal axes is built for the curve segment V_iV_j. The origin is taken to be V_i. The first axis A_1 points from V_i to V_j (Fig. 3). As each successive axis A_k is constructed ($k = 1, \ldots, n$), we compute its unit vector a_k, and compute the coordinate x_{sk} of each point V_s of the curve ($s = i, \ldots, j$) by taking the dot product $V_iV_s \cdot a_k$. To compute a new axis or decide whether to split the curve segment when all n axes have been found, we compute the squared distance $V_iv_{sk}^2$ from V_i to the projection v_{sk} of each curve point V_s on the hyperplane perpendicular to the k^{th} axis. From Pythagoras' theorem, this squared projected distance is equal to the total squared distance minus the sum of the squares of the coordinates of the point for axes 1 to k:

$$V_iv_{sk}^2 = V_iV_s^2 - \sum_{k'=1}^{k} x_{sk'}^2$$

We select the largest distance V_iv_{mk} (call it d_m), the corresponding curve point V_m, and the corresponding projection v_{mk}. If the index k has not reached the value n (the prescribed number of coordinates for the local coordinate system), we build a new axis from V_i to v_{mk}, and we proceed as above for this new axis.

If k is equal to n, we compare d_m to ϵ. If d_m is smaller than ϵ, we are sure that not only is the coordinate $k+1$ smaller than ϵ and can be neglected, but so are all the coordinates beyond $k+1$. Indeed, all the other coordinate axes that remain to be constructed would be contained in the hyperplane perpendicular to axis A_k since they are themselves perpendicular to A_k. These coordinates are projections of the vectors V_iv_{sk} on these axes. Projections are smaller in magnitude than the original vectors V_iv_{sk}, which themselves have a maximum magnitude

smaller than d_m (since d_m is the maximum), and d_m is smaller than ϵ. Therefore these coordinates can also be neglected. The curve segment V_iV_j then "fits" in the space of dimension n, and the frames corresponding to end points V_i and V_j are taken as summaries of the video clip sequence between these frames. On the other hand, if d_m is larger than ϵ, the curve segment is split into two curve segments V_iV_m and V_mV_j, and the simplification procedure is applied recursively to each of the two curve segments.

2.3 Pseudocode

The generalization of the Ramer algorithm to the simplification of a curve of dimensionality N (an ND curve) into curve segments of dimensionality n can be summarized as follows:

- Express the coordinates of the points V_s of the curve segment V_iV_j in a local frame of reference, in which:
  the line segment V_iV_j is the first axis A_1,
  the second axis A_2 is obtained by projecting the points V_s on a hyperplane perpendicular to the first axis A_1 through V_i and joining V_i to the *projected point* v_{s1} *of* V_s *that is furthest from* V_i,
  the k^{th} axis A_k is obtained by projecting the points V_s at $v_{s(k-1)}$ onto the hyperplane perpendicular to the $(k-1)^{st}$ axis through V_i, and joining V_i to the projected point $v_{s(k-1)}$ that is furthest from V_i, with k from 1 to n.

- If the projection v_{sn} furthest from V_i on the hyperplane perpendicular to the n^{th} axis A_n is at a distance from V_i smaller than ϵ, then the curve segment V_iV_j fits in this local nD space and is not split.

- Otherwise, split the curve segment at the point V_m that produced that furthest projection v_{sn}, and recursively approximate the two curve pieces V_iV_m and V_mV_j.

3 Construction of Feature Vector

MPEG formats are rapidly becoming the most popular video compression formats. They significantly reduce the required amount of storage and transmission bandwidth without sacrificing much quality [15]. In generating summaries of MPEG video clips, the most responsive algorithms are those that keep the required amount of decompression to a minimum. This goal can be achieved if the DC coefficients are used to generate the feature vectors for each frame and the frame points of the video curves. We can use the 1800 DC coefficients provided by the MPEG encoding of 320 $\times$ 240 frames as coordinates of feature vectors and frame points, as we did in previous work [10]. However, it may be useful to consider an alternative feature vector that remains within the lowest possible dimensional space as frames evolve within a shot of a video clip, as long as not much novelty is introduced within the shots.

As an illustration of these issues, consider a shot in which a white car, with luminance 1, moves diagonally on a dark background (luminance 0), from the top left of the picture to the lower right (Fig. 4, top). The car occupies about two horizontal blocks of the 4×3 block image.

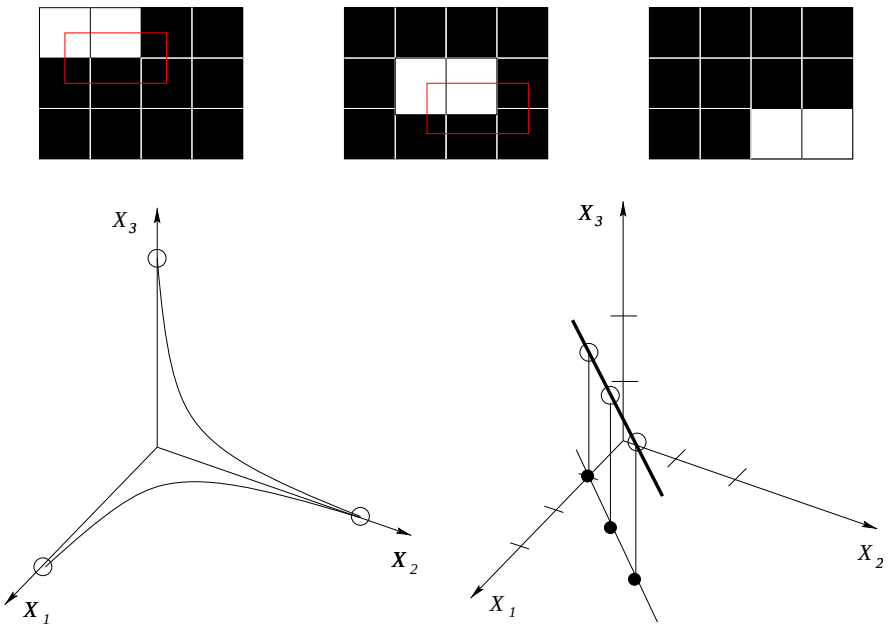

Figure 4: 3D projection of video curve produced by "raw feature vector" (left) and by feature vector using centroids and pixel counts in color ranges (right) for simple image sequence

The viewer will visually track the car as it passes through the image, and will not consider that there is enough significant change in content to require that several frames be used to summarize such a shot.

With a feature vector that uses colors in image blocks, the white car produces a feature vector $(1, 1, 0, \ldots, 0)$ in the first frame of the shot; for an intermediate frame the feature vector is $(0, \ldots, 0, 1, 1, 0, \ldots, 0)$, and for the last frame, it is $(0, 0, \ldots, 0, 1, 1)$.

Clearly, these feature vectors are perpendicular to each other, since their dot products are zero. They can be used as a basis for a 3D projection of the video curve. The actual curve is 10-dimensional (only 2 of the 12 coordinates of the feature vector remain equal to zero as the car moves across the image). It is much smoother than its 3D projection of Fig. 4, but it is highly nonlinear; otherwise its projection would be a line.

It is possible to construct feature vectors that span a smaller set of dimensions and are more stable within shots, even in the presence of camera motion, so that cuts and dissolves create jumps in totally new dimensions, generate changes of curvature that are much larger than curvature changes within shots, and are more easily detected in the curve simplification operation.

Staying with the previous car shot example, consider for instance the feature vector whose components are the coordinates of the largest blobs in different intervals of luminance and chrominance. The coordinates of a blob are the (x, y) image position of its centroid, and its area (number of pixels) can be considered to be its z coordinate. (Additional coordinates describing blob shape such as second and third moments with respect to the centroid could also be considered.) These blobs are detected by a blob coloring algorithm [2] within each luminance and chrominance in-

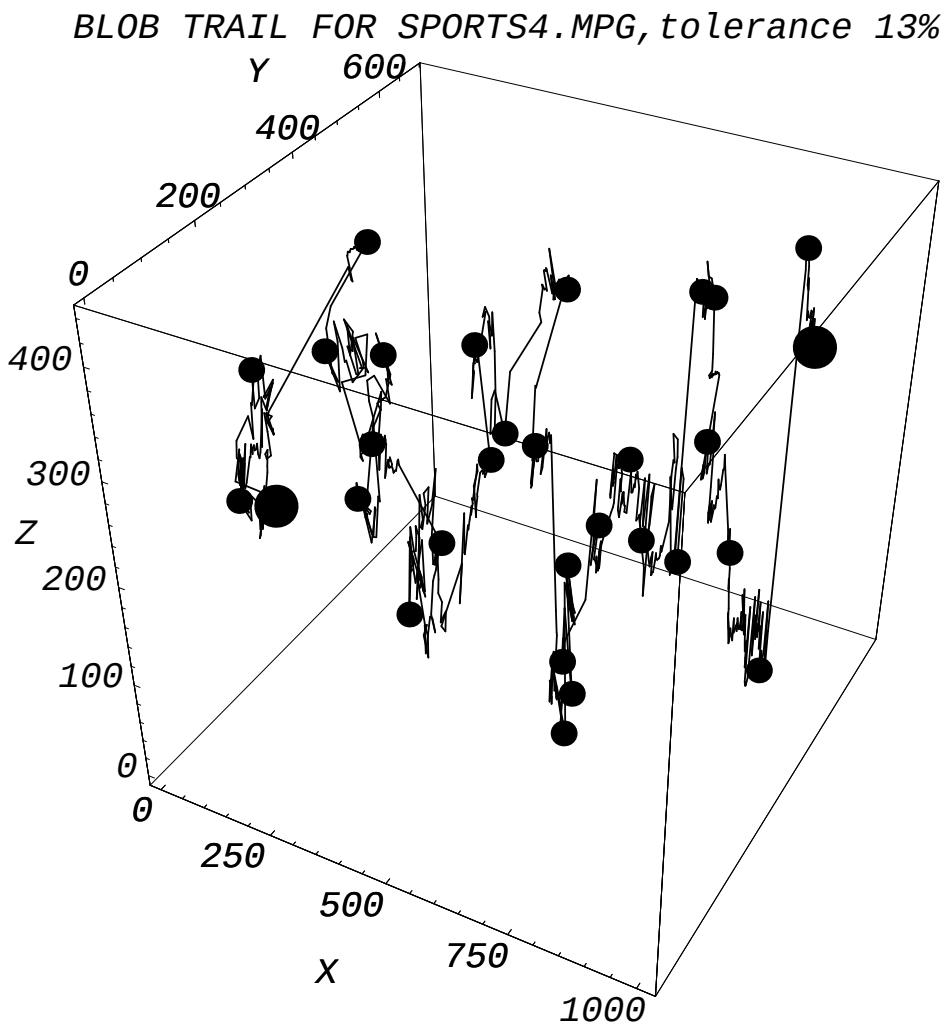

Figure 5: Video curve and sampled keyframes (black dots) for a baseball sequence with long shots and large camera motions.

terval, then sorted by size, with only the largest or few largest being used for each interval. For the car shot, there is a white blob of area 2 pixels with a position that varies from $(x = 1, y = 0)$ for the first frame to $(x = 3, y = 2)$ for the last frame. There is also the dark blob whose area remains 10 and whose centroid remains around the center ($x = 2$, $y = 1$) of the image. The feature vector for the car shot varies from $(1, 0, 2, 2, 1, 10)$ for the first frame to $(3, 2, 2, 2, 1, 10)$ for the last frame. It evolves within a 1D space (a straight line) depicted at the bottom right of Fig. 4.

In our experiments, we have constructed feature vectors of 37 dimensions by considering a time coordinate together with the three coordinates of the largest blobs in four intervals for each luminance and chrominance channel ($1 + 3 \times 4 \times 3$). A 3D projection of the video trajectory for a 937 frame baseball video sequence using this type of feature vector is illustrated in Fig. 5. The end points are the larger black spots. The smaller black spots along the curve correspond to the

keyframes obtained by curve simplification using line segments for a coarseness parameter equal to 13% of the largest inter-point distance. With this coarseness, all 21 shots are represented in the 29 frames found by the curve simplification. The few shots that are represented by more than one frame are fly balls and base runs where the camera smoothly unveils completely different backgrounds as it tracks the ball or a player. In these regions of the video stream, the shots were summarized by two or three keyframes. This example illustrates the fact that the summarization scheme is well-behaved in the sense that the summarization lets all the shots contribute at least one frame each without oversampling the shots with high activity content.

A problem with this approach is that we must sort the blobs by sizes and consider only the largest or the few largest in order to maintain a constant number of feature vector coordinates. A previously discarded blob, as it grows larger, may suddenly take the place of a previously considered blob, generating a discontinuity on the video curve. Such a discontinuity is not totally artificial since changes of respective blob sizes occur mainly in shots of higher activity and may be considered noteworthy events. However these observations led us to also experiment with feature vectors in which the centroids of pixels in chrominance and luminance bins were computed without connectedness consideration. We found these to provide slightly terser video summarizations in the video player described in the next section. Note that feature vectors can also be crafted to amplify video trajectory curvatures according to subjects of interest defined by the user.

4 Data Representation

Assume that we compute the vertices of a curve simplification for a coarseness parameter ϵ_1, and then perform another curve simplification for a *smaller* parameter ϵ_2. The second operation will start with the same binary curve splitting as the first one, and then split some curve segments further to reach the desired smaller tolerance.

To avoid repeating the curve splitting operation each time a new tolerance is desired, it is preferable to perform the curve simplification using a coarseness parameter ϵ close to zero, keeping relevant distance information along the way. (The curve splitting decomposition is then not a quite a curve "simplification", since all or most[1] vertices of the polygonal video curve end up being selected). During this decomposition at the finest possible tolerance level, we generate a *tolerance list* that stores for each selected vertex its index along with a tolerance that is the distance d_m being compared to ϵ as a curve splitting condition (see Section 2). For a curve simplification using line segments, the tolerance of each vertex is simply its distance to the line segment joining the two parent vertices (Fig. 1). Then, to find the vertices of the curve simplification for coarseness ϵ', very little computation is required: one simply scans the tolerance list and pulls out the vertex indices for which the tolerance is larger than ϵ'.

[1] A polygonal curve segment with $n+1$ vertices fits into an nD plane with tolerance 0, thus the curve simplification of a polynomial curve with M vertices into nD curve segments with tolerance 0 could contain as few as $1+(M-1)/n$ vertices in the absence of computation noise. For $n=1$, this expression is equal to M, i.e. all vertices are found.

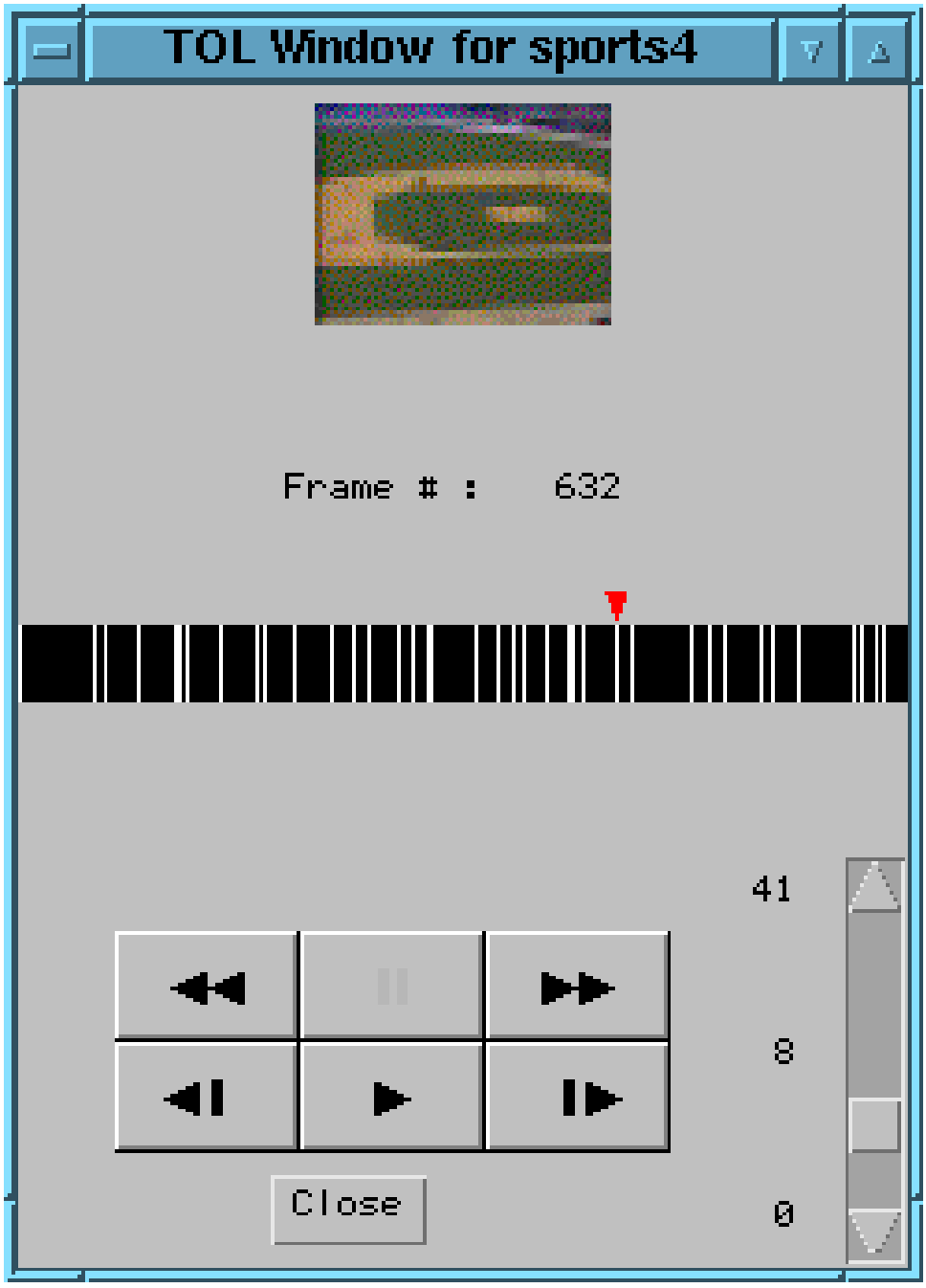

Figure 6: Video player with *altitude slider* for control of summarization level

5 Video Player with Altitude Slider

We have developed a video player that uses the tolerance list described in Section 4 to let a user view movies at levels of summarization that he can control (Fig. 6). A Java Applet prototype of this video player can be viewed and tested at [1]. This player offers regular controls that let the user navigate at will along the temporal dimension (play, fast forward and backward, pause, and mouse-controlled frame viewing by click-dragging the triangle slider above the black horizontal stripe). The video player provides an additional degree of freedom, an "altitude" control, that lets the user control the altitude at which he wishes to fly over the video footage. This control is implemented as a vertical slider. Internally, the slider position defines the level of coarseness at which the video curve is simplified. Only the frames for which the tolerances are larger than the level of coarseness in the tolerance list are displayed. When the slider is pushed all the way down (lowest altitude), all the frames are played in sequence (at the standard 30 frames/second speed in play mode). When the slider is pulled all the way up, the first and last frame and one keyframe are played in a loop. At some intermediary positions, a single frame is typically displayed for static shot sequences, and several frames are displayed for dynamic shots with large object action or significant camera motion. As the slider is pushed down, more frames are selected from dynamic shots.

The latency between two successively displayed frames is increased (up to a point) in proportion to the jump in frame index, to give the user more time to observe single frames from static shots that are less related to each other, while still perceiving the motion of more closely sampled dynamic shots. Thus the pace of the display of this video player varies from a fast slide show for successions of static shots to short bursts of animation when action shots are encountered and sampled. The user can change the level of summarization at any time by moving the altitude slider, and this change is instantly reflected in the movie display. Therefore the user can immediately obtain more detail with a finer frame sampling on any given portion of a movie, if for example the level of summarization was too coarse to allow enough understanding of the flow of the movie at that time.

A *sampling stripe* provides the user with a view of the level of sampling performed by the curve simplification process (Fig. 6). It is a long black stripe with one white vertical tick mark for each displayed frame at the requested coarseness. A triangular frame marker above the sampling stripe slides above the stripe as the video clip is being played and indicates which frame is being displayed. As mentioned, the user can also navigate through the video sequence by sliding this triangular frame marker with a direct click-and-drag mouse action. When all frames are to be played, the sampling stripe is completely white. At intermediary altitude slider positions, only a few white frame tick marks appear (Fig. 6).

There is an interesting level of summarization in which every shot is already represented by at least one frame, and very active shots are sampled by two or three shots. This summarization level is useful for the user who does not want to miss any shot. This level can be easily detected by slowly pushing the altitude slider down toward finer detail. As this level is reached, isolated tick marks stop appearing on the sampling stripe and tick marks start clustering within active shots.

A user interested in active shots can see them as regions of clustered tick marks in the sampling

stripe. He can jump to these shots by dragging the frame marker of the sampling stripe to these clusters.

6 Visualization and Feature Vectors

The sampling density of an active shot for a given coarseness parameter depends on choices made in the implementation of the curve simplification process discussed above. One of the choices is the way frames are mapped into points in a high-dimensional space. This is the feature vector issue addressed in Section 3. "Raw vectors" whose coordinates are DC luminance and chrominance of MPEG macroblocks produce curves with relatively high curviness in active shots, when compared to "centroid vectors" whose coordinates are centroids and counts of pixels in color ranges. Consequently, the summarization is more sensitive to the activity levels of shots and less sensitive to discontinuities between successive shots with raw vectors than with centroid vectors. Another choice concerns the splitting criterion used by the curve simplification process. Splitting can be performed when a curve segment cannot be approximated by a line segment, as in the original Ramer algorithm, or when the curve segment cannot be approximated by a plane or a 3D hyperplane, as allowed by the generalized algorithm described in Section 3.

In preliminary experiments, we have found that using line segments with a raw vector mapping tends to oversample very active shots before all static shots have been sampled by a single keyframe. Using planes instead of line segments with a raw vector mapping reduces this oversampling effect.

With a centroid vector mapping, we find that using line segments is a good way to sample all the shots with at least one keyframe and with a minimum number of keyframes. With centroid vectors, using planes seems to be less effective than using lines, selecting a larger number of keyframes in order to sample all the shots at least once.

Whether video curves are better approximated by a chain of line segments (a polyline) or by a chain of planes or hyperplanes thus depends on the selection of the feature vector mapping and on what is meant by "better" – e.g., detection of scene transitions vs. dense sampling of active shots as in the case discussed above. One problem we are presently studying is how to craft the feature vector mapping and the curve splitting criterion once the user has interactively defined what is better for him/her. The user can pose a query asking for human faces, outdoor scenes, independently moving objects, or colliding blobs. The query system would define a feature vector mapping so as to produce a video trajectory with a high curvature only when those features that interest the user show up in the video stream. The system would then select a curve splitting criterion and present to the user the frames corresponding to the vertices that the curve simplification method generates in these high-curvature regions.

7 Concise Summarization of Video Sequences

We now address the issue of automatically selecting the coarseness level that extracts from a video sequence a set of keyframes that is concise and yet completely summarizes the sequence. Ideally, the user should be able to view the set of keyframes and completely understand the message of the video sequence, without wasting time watching a large number of redundant

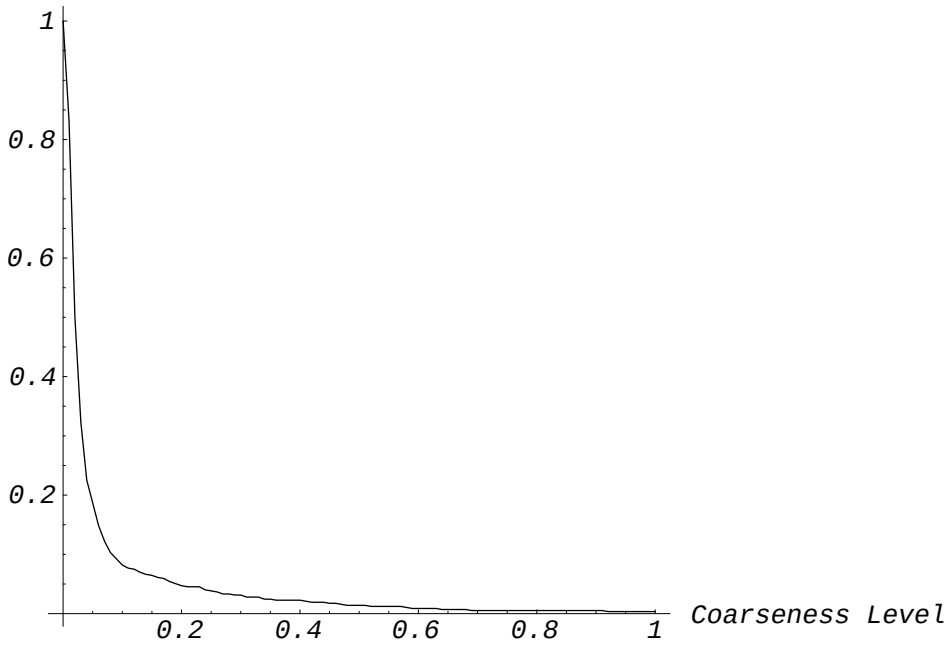

Figure 7: Typical histogram relating the selected coarseness level to the number of frames returned by the video summarization.

frames. One of the requirements for achieving this goal is that there should be at least one keyframe for each shot of the sequence. Indeed, since we do not analyze the semantic content of sequences, we cannot assess the relative importance of the shots and cannot discard any shot as useless. We have to assume that when the editor of the video included a shot, he/she believed that the shot contributes to the message of the sequence. Therefore we should summarize each shot for the user by at least one keyframe. This requirement has to be achieved while minimizing the total number of keyframes.

As stated above, curve approximation by a chain of line segments on a trajectory obtained by a mapping using centroid vectors appears to provide more concise and more representative keyframe selections than the other approximations we considered. Therefore we focus our attention on this mode of summarization. For this choice we find that the following method of choosing the coarseness level provides satisfactory results.

1. Compute the histogram relating the coarseness level (from 0 when all frames are returned as keyframes, to 100% when only the two end frames and one intermediate frame are returned as keyframes) to the proportion of the total number of frames returned by the video summarization at that coarseness level. To accomplish this, for each coarseness level, scan the tolerance list (Section 4), count the vertices for which the tolerance is larger than the coarseness level, and normalize this number with the number of vertices on the list.

2. Find the abscissa of the histogram for which the histogram slope is -0.5; select this abscissa as the coarseness level likely to provide a summarization that is concise and does not miss

shots (this coarseness level is typically different from clip to clip).

The justification for this approach is as follows. The clip histograms generally have similar shapes (Fig. 7), with two distinct slopes, a steep slope at small coarseness levels, and a gentle slope for large coarseness levels. In other words, as we slide the coarseness slider in the video browser to coarseness levels close to 0 (the region of steep slope in the histogram), we notice large variations in the number of selected keyframes. At this level of coarseness, the curve simplification is able to follow not only large-scale variations, but also small-scale variations that occur mainly within shots. On the other hand, for coarseness levels corresponding to the gentle slope, the curve simplification ignores the intra-shot variations and returns vertices that summarize large-scale variations in the video curve, corresponding mainly to shot transitions. But as we push toward larger coarseness levels, we start missing more and more of the keyframes characteristic of these large-scale variations. If we want to avoid detecting intra-shot variations and still detect the maximum number of large scale variations, we have to position the coarseness level somewhere around the break between the two slope regimes, in the region where the slope of the histogram is on the order of -1 (Fig. 7).

To refine this slope estimate, we examined the number of missed shots as a function of the slope of the histogram for a number of video sequences. We processed a total of 46 video sequences, mostly very active 15- and 30-second commercial clips, with a few more static documentary sequences of 1 to 2 minutes. The total number of frames was 40038, and the number of shots was 537. There were 94 shot transitions that were dissolves and fades. (Distances between frames during dissolves can be smaller than during camera motion; therefore methods selecting keyframes from inter-frame distances have difficulty noticing such shot changes.) We generated by hand groundtruth files listing for each clip the locations of shot transitions.

We computed for each video sequence a histogram from its tolerance list, found the coarseness levels for a range of slopes of the histogram, found the list of keyframes corresponding to tolerances larger than these coarseness levels, and used the ground truth lists of shot boundaries to count the numbers of missed shots. The missed shots were counted and normalized for the whole set of clips. We also computed the average number of keyframes retrieved in each shot and the proportion of keyframes to total number of frames (which measures the average summarization level of each coarseness level). The results are plotted in Fig. 8, Fig. 9 and Fig. 10.

Fig. 8 shows the proportion of shots that are not sampled by any keyframe (missed shots) in relation to histogram slope. This figure shows that as we select coarseness levels corresponding to histogram slopes of larger magnitude, the percentage of missed shots decreases, until it levels off to 0.4% (2 shots missed out of 537 shots) for a slope of -0.5. On the other hand, the number of keyframes extracted per shot (Fig. 9) and for the whole set of clips (Fig. 10) keeps increasing almost linearly with the histogram slope. Therefore the optimal choice seems to be for coarseness levels corresponding to a histogram slope around -0.5: with a gentler slope, more shots are missed; with a steeper slope, more keyframes are returned, but the number of missed shots no longer decreases. For coarseness levels corresponding to the -0.5 slope, the curve simplification extracted an average of four keyframes per shot, and these keyframes represented about 5% of the total number of frames. An average of four keyframes per shot seems to be a satisfactory summarization level if it guarantees that virtually no shot will be missed. In fact it would not be desirable to have only one keyframe per shot; many of the shots had very dynamic

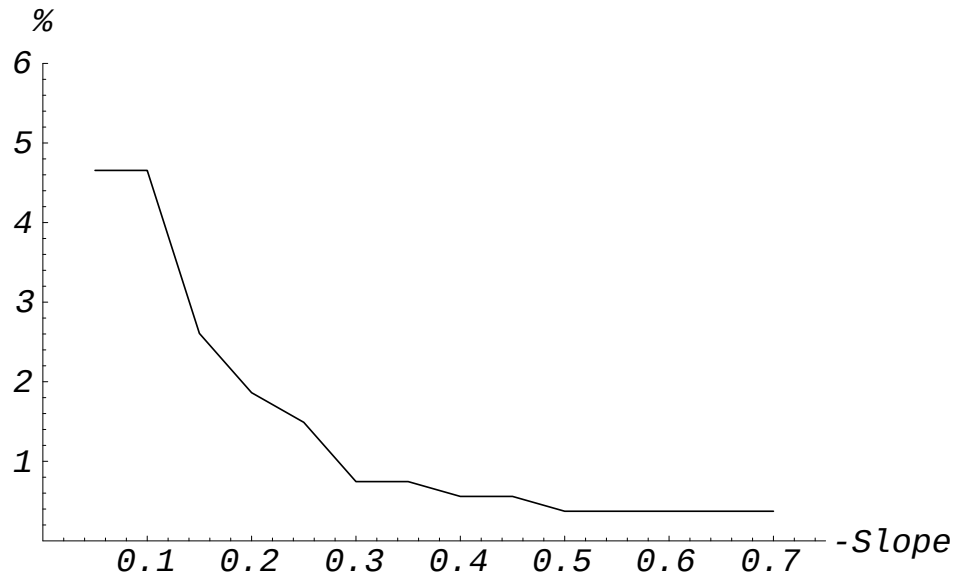

Figure 8: Percentage of missed shots vs. total number of shots in relation to histogram slope.

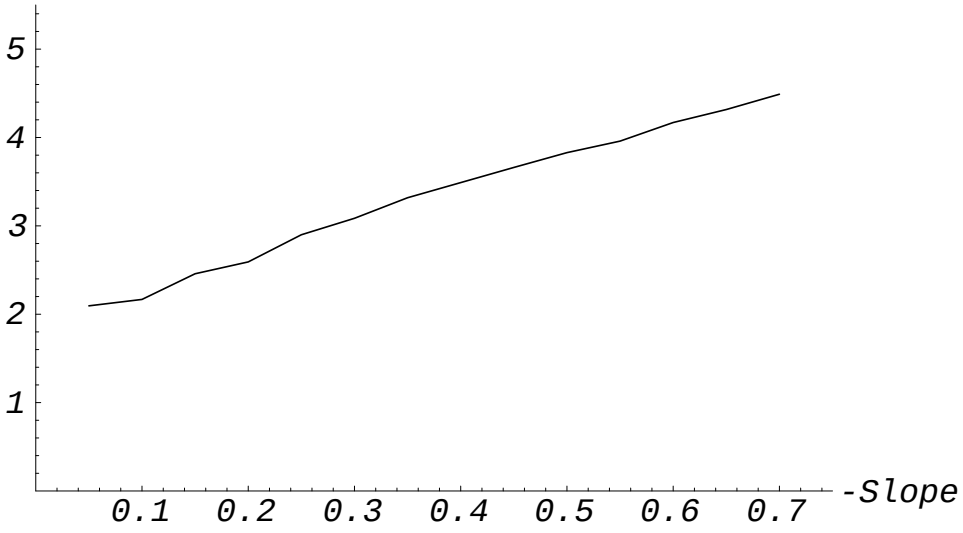

Figure 9: Average number of keyframes found in each shot in relation to histogram slope.

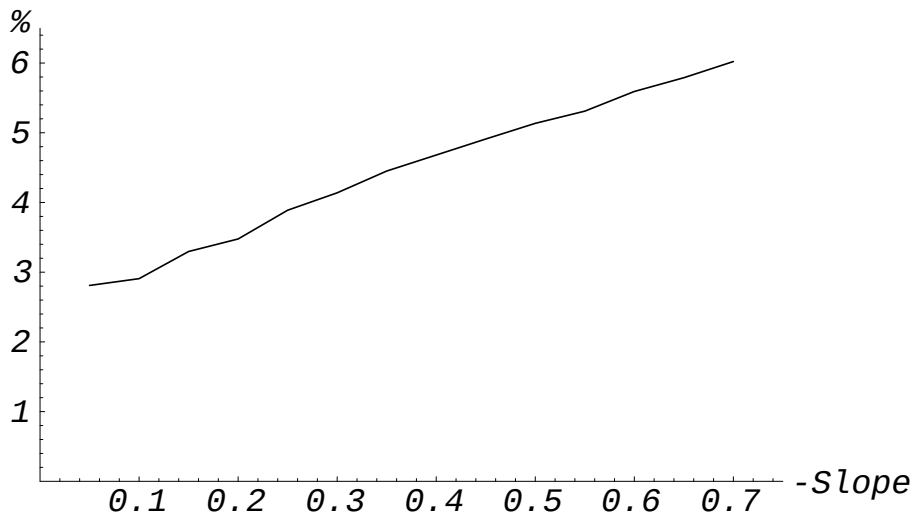

Figure 10: Percentage of frames returned as keyframes vs. total number of frames in relation to histogram slope.

content to attract attention to the commercial message, and the user would not understand what is going on from a single keyframe.

An examination of the two missed shots provides insight into some of the difficulties of the task. One missed shot followed a very similar shot, differing by a position switch between two dialoguing actors created by a different camera angle. The other missed shot had only two frames.

8 Related Work

Our method of constructing local Cartesian coordinate systems was inspired by a method called FastMap [5]. FastMap is a dimensionality reduction method that constructs each new basis axis along pairs of points at maximum distances among the projections on the hyperplane normal to the previous axis. By contrast, our new basis axis is constructed from the origin V_i to the furthest projection. FastMap's axes are better in the sense that fewer may be needed to reach a prescribed tolerance level of coordinate magnitude. FastMap could be used in the hierarchical curve splitting scheme. However, it provides a pair of vertices at each step. If both are used, they divide the curve segments into three subsegments, leading to a more complex curve splitting by trisection. If only the vertex of the pair with the furthest projected distance to the origin V_i is picked, this vertex is likely to be the vertex found by our simpler method. Other dimensionality reduction methods could be used to construct the local Cartesian coordinate systems, such as Principal Component Analysis [4], Singular Value Decomposition [12], or Eigenvector analysis using approximations by Ritz vectors [14]. Experiments could tell whether the advantage of detecting more representative summary frames would outweigh the burden of added complexity.

In related previous work [10], we used FastMap for dimensionality reduction of feature vectors

representing video clip frames, and for representing the data as Video Trails. We segmented regions of the trails by placing bounding rectangles around points of constant activity level, then classified these segmented regions as either transitions or shots by looking at the shape and point content of the bounding rectangles. For long video sequences, the curve splitting technique presented in this paper is required to divide the sequence into pieces in which the dimensionality reduction hypothesis is legitimate.

Schweitzer [14] uses Ritz vectors as a basis for constructing visual representations of video clips and for cluster analysis of related scenes. He also shows how a basis of Ritz vectors for a large distributed database can be efficiently computed by combining the vector bases of local databases.

Yeung and Yeo [18] recognize story structure in video clips by clustering the feature vectors of frames using a test based on distance and a temporal threshold, then building Scene Transition Graphs between the clusters. Their work provides insights into how the keyframes detected by low-level techniques such as those proposed in this paper can be combined to provide information about both the structure and content of video sequences in high-level video browsers [19]. Other browsing techniques are explored in [16] and [20]. Our goals were more modest and mainly aimed at improving the robustness of keyframe detection.

Color histograms are used as feature vectors in many applications. They have the advantage of being insensitive to small changes in camera motion. However, they lack spatial information, so images with very different appearances can have similar histograms. To address this problem, Pass et al. [11] define a color histogram in which each color has two buckets, one counting pixels of large blobs, the other counting the remaining pixels. One possible drawback is that a minimum size must be defined for what constitutes a large blob, and when a blob crosses this size, say during a camera zoom, the corresponding number of pixels is taken out of one bucket and into the other, which generates a feature vector discontinuity. A similar issue was discussed above for one of our feature vector generation methods. Huang et al. [8] review other schemes that incorporate spatial information into histogram methods, and describe a promising color correlogram method similar to the cooccurrence matrix approach used in texture discrimination [6]. Color correlograms may be a good alternative as candidate feature vectors for generating the trajectories required by our video summarization method, and we plan to explore their use in future work.

9 Conclusions

We have proposed a recursive multidimensional curve splitting algorithm in which the criterion for splitting a curve segment is its ability to fit into an nD hyperplane. Therefore the splitting condition is the magnitude of the coordinates of the points along a curve segment, with respect to the $n+1^{st}$ unit vector of a local orthonormal vector basis.

We have illustrated an application of this algorithm to the summarization of video sequences by small numbers of keyframes. A video sequence is mapped to a curve trajectory in a high-dimensional space, using a feature vector carefully crafted so that the appearance of significant new information produces discontinuities or regions of high curvature in the trajectory. Then the algorithm is applied to recursively simplify the trajectory into constituent segments of low dimension. The keyframes correspond to the junction vertices between the constituent segments.

This method detects keyframes that simplify the general trends of the video sequence and is less sensitive to local variations between consecutive frames than methods using frame differences or local filters.

We have described a video player that gives the user a second degree of freedom, the "altitude" at which he/she wishes to fly over the video stream. The user pushes down the altitude slider to instantly obtain more keyframes during the play of a summarized video. Internally, this slider controls the coarseness of the approximation of the video trajectory.

Finally, we have proposed a method for automatically extracting from a video sequence a set of keyframes that is concise yet contains sample frames from all the shots of the sequence. The method consists of computing a histogram representing the number of summarizing frames at different coarseness levels, and using the coarseness level corresponding to a specific slope of the histogram selected from experiments.

The proposed curve splitting algorithm is also useful for splitting a high-dimensional sorted dataset or time series into several low-dimensional subsets. Substantial gains in computational time can thus be obtained in each of the subsets for data retrieval, structure analysis, similarity searching and data mining, due to the lower dimensionality of each data point in each subset. Finally, the subsets can be more easily visualized than the global dataset.

Acknowledgements

The support of this research by the Department of Defense under contract MDA 9049-6C-1250 is gratefully acknowledged.

The authors would also like to thank Felix Sukhenko for writing the Java interface for the video player with summarization control.

References

[1] Online video player applet available at
http://documents.cfar.umd.edu/LAMP/Media/Projects/VideoContent/JavaApplet/index.html

[2] Ballard, D.H., and Brown, C.M., *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[3] Douglas, D.H., and Peucker, T.K.,"Algorithms for the Reduction of the Number of Points Required to Represent a Line or its Caricature", The Canadian Cartographer, 10(2), pp. 112–122, 1973.

[4] Duda, R.O, and Hart, P.E., *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[5] Faloutsos, C., and Lin, K., "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets", Proc. of ACM SIGMOD, pp. 163–174, 1995.

[6] Haralick, R.M., "Statistical and Structural Approaches to Texture", *Proceedings of the IEEE*, 67, pp. 786–804, 1979.

[7] Hershberger, J., and Snoeyink, J. "Speeding up the Douglas-Peucker Line-Simplification Algorithm", http://www.cs.ubc.ca/cgi-bin/tr/1992/TR-92-07.

[8] Huang, J., Kumar, S.R., and Mitra, M., "Combining Supervised Learning with Color Correlograms for Content-Based Image Retrieval", Proc. of ACM Multimedia, pp. 325–334, 1997.

[9] McMaster, R.B., "A Statistical Analysis of Mathematical Measures for Linear Simplification", *American Cartographer*, 13, pp. 103–116, 1986.

[10] Kobla, V., Doermann, D. and Faloutsos, C., "*VideoTrails*: Representing and Visualizing Structure in Video Sequences", Proc. of ACM Multimedia, pp. 335–346, 1997.

[11] Pass, G., Zabih, R., and Miller, J., "Comparing Images Using Color Coherence Vectors", Proc. of ACM Multimedia, pp. 65–73, 1996.

[12] Press, W.H., Teukolsky, S.A., Vettering, W.T., and Flannery, B.P., *Numerical Recipes in C*, Second Edition, Cambridge University Press, 1992.

[13] Ramer, U., "An Iterative Procedure for the Polygonal Approximation of Plane Curves", *Computer Graphics and Image Processing*, 1, pp. 244–256, 1972.

[14] Schweitzer, H., "A Distributed Algorithm for Content Based Indexing of Images by Projections on Ritz Primary Images", *Journal of Data Mining and Knowledge Discovery*, in press. Shorter version in Proc. of Int. Conf. on Computer Vision, 1998.

[15] Sikora, T., "MPEG Digital Video-Coding Standards", *IEEE Signal Processing Magazine*, pp. 82–99, Sept. 1997.

[16] Smith, M.A., and Kanade, T., "Video Skimming for Quick Browsing Based on Audio and Image Characterization", Proc. of Computer Vision and Pattern Recognition, 1997.

[17] White, E.R., "Assessment of Line-Generalization Algorithms Using Characteristic Points", *American Cartographer*, 12, pp. 17–27, 1985.

[18] Yeung, M.M, and Yeo, B.L., "Time-Constrained Clustering for Segmentation of Video into Story Units", Proc. of Int. Conf. on Pattern Recognition, 1996.

[19] Yeung, M.M., Yeo, B-L., Wolf, W. and Liu, B.,"Video Browsing using Clustering and Scene Transitions on Compressed Sequences", Proc. SPIE Conf. on Multimedia Computing and Networking, vol. 2417, pp. 399–413, 1995.

[20] Zhang, H.J., Low, C.Y., Smoliar, S.W., and Wu, J.H., "Video Parsing, Retrieval and Browsing: An Integrated and Content–Based Solution", Proc. of ACM Multimedia, 1995.