Spider: A System for Finding 3D Video Copies

NAGHMEH KHODABAKHSHI, Simon Fraser University and Qatar Computing Research Institute
MOHAMED HEFEEDA, Qatar Computing Research Institute

This article presents a novel content-based copy detection system for 3D videos. The system creates compact and robust depth and visual signatures from the 3D videos. Then, signature of a query video is compared against an indexed database of reference videos' signatures. The system returns a score, using both spatial and temporal characteristics of videos, indicating whether the query video matches any video in the reference video database, and in case of matching, which portion of the reference video matches the query video. Analysis shows that the system is efficient, both computationally and storage-wise. The system can be used, for example, by video content owners, video hosting sites, and third-party companies to find illegally copied 3D videos. We implemented Spider, a complete realization of the proposed system, and conducted rigorous experiments on it. Our experimental results show that the proposed system can achieve high accuracy in terms of precision and recall even if the 3D videos are subjected to several transformations at the same time. For example, the proposed system yields 100% precision and recall when copied videos are parts of original videos, and more than 90% precision and recall when copied videos are subjected to different individual transformations.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering, Search process*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Video copy detection, 3D video, video fingerprinting, depth features, visual features

ACM Reference Format:

Khodabakhshi, N. and Hefeeda, M. 2013. Spider: A system for finding 3D video copies. ACM Trans. Multimedia Comput. Commun. Appl. 9, 1, Article 7 (February 2013), 20 pages.
DOI = 10.1145/2422956.2422963 http://doi.acm.org/10.1145/2422956.2422963

1. INTRODUCTION

Three-dimensional (3D) video is becoming popular, as its technology is getting mature. Now, content creators are producing more materials in 3D formats, cinemas are upgraded to 3D, 3DTV broadcasting has become reality, and even video sharing Web sites, such as YouTube, provide means to upload and watch 3D contents. Many experts are anticipating that the future of video is 3D. As 3D video technology progresses, protecting 3D videos against copyright infringement becomes an important issue,

This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada and in part by the British Columbia Innovation Council.
Authors' addresses: N. Khodabakhshi, School of Computing Science, Simon Fraser University, Surrey, BC, V3T 0A3, Canada; email: nkhodaba@sfu.ca; M. Hefeeda, Qatar Computing Research Institute, Qatar Foundation, PO BOX 5825, Doha, Qatar; email: mhefeeda@qf.org.qa.

DOI 10.1145/2422956.2422963 http://doi.acm.org/10.1145/2422956.2422963

especially since creation of 3D contents is more expensive than traditional 2D videos. Watermarking and content-based copy detection, which is the focus of this article, are two approaches to protect videos.

Detecting copies of traditional 2D video is a complex task. First, videos are composed of many frames (usually 25 or 30 frames per second), and comparing numerous frames from potential video copies against reference videos is computationally intensive. Detection of 2D video copies is also complicated by the fact that many edit effects occur on the copied videos. These edits, usually called video *transformations*, can be done intentionally to avoid detection or unintentionally because of the copying process. For example, a copied 2D video may be scaled, rotated, cropped, transcoded to a lower bit rate, or embedded into another video. The contrast, brightness, or colors of video can also be changed. In addition to mentioned spatial transformations, temporal transformations may also be applied to copied videos. Fast or slow motion, random replication or omission of frames, and changes in frame rate due to different video standards are typical temporal transformations.

Detecting copies of 3D videos is even more challenging. This is because 3D videos have many more transformations than 2D videos. First, each 3D video has at least two views, where each view is a 2D video. 2D traditional transformations can be applied on one, all, or subset of the views, resulting in many possibilities for transformations. Second, 3D videos can be encoded using different formats, including stereo, multiview, video plus depth, and multiview plus depth. Changing from one format to another complicates the detection process. For 3D formats that have depth signal, several transformations can be applied on the depth as well, such as depth blurring. Furthermore, a copied 3D video can contain a subset of the views in the original video. Finally, new views can be created (synthesized) using techniques such as the one in Shum et al. [2004]. Synthesized views display the scene from different angles, and thus reveal different information than in the original views. For example, an object occluded in one view could appear in another.

We present a novel content-based copy detection system for 3D videos. The system extracts compact and robust signatures from the 3D videos' content, both texture and depth. These signatures are compact and robust to many 3D video transformations, including the ones just described. Then, these signatures are used to compare videos. Signatures of query videos are created online and compared against the signatures of reference videos in the indexed database. We implemented Spider, a complete realization of the proposed system. 3D videos collected from different sources with diverse characteristics were used to evaluate the proposed system. Our experimental results show that the proposed system can achieve high accuracy in terms of precision and recall even if the 3D videos are subjected to several transformations at the same time. For example, the proposed system yields 100% precision and recall when copied videos are parts of original videos, and more than 90% precision and recall when copied videos are subjected to different individual transformations.

The rest of this article is organized as follows. In Section 2, we summarize the related works in the literature. In Section 3, we present the details of the proposed 3D video copy detection system, and Section 4 indicates how we implemented this system in practice. We present our extensive experimental evaluation in Section 5, and we conclude the work in Section 6.

2. RELATED WORK

There are different methods for video copy detection. One method called watermarking [Kahng et al. 1998] is to embed information which is both distinctive and invisible to the user into the content. Then, copy detection becomes a matter of searching the video content for this hidden information. Another method is to use content itself. This is known as Content-Based Copy Detection (CBCD). The underlying premise of content-based copy detection is that the content itself is the watermark. In other words, there is enough information in the content to create a unique fingerprint of the video.

These kind of methods involve extracting the fingerprint from the content and performing a distance measure to determining the similarity between the fingerprints of the query video and the original videos.

3D watermarking approaches in the literature can be classified into three groups [Smolic et al. 2007]: (i) 3D/3D: Watermark is embedded in the 3D model, and it is detected in the 3D model; (ii) 3D/2D: Watermark is embedded in the 3D model, and it is detected in the 2D rendering; and (iii) 2D/2D: Watermark is embedded in the 2D rendering, and it is detected in the 2D rendering. The first two groups try to protect the traditional representation of a 3D scene, which are geometry and texture. The third group tries to watermark the sequences of images which are the 2D projections of the same 3D scene. Consequently, the third group can be used for copy detection of 3D videos [Smolic et al. 2007]. While the first two groups have been studied quite widely, the third group emerged after image-based rendering techniques developed [Koz et al. 2010].

Koz et al. [2010] propose a watermarking scheme for multiview 3D videos. They embed the watermark into the main representation of a multiview 3D content and extract it after the content is transformed or a virtual view is generated. Their research is limited to static scenes consisting of one object or one depth layer. Also, this watermarking scheme only considers multiview 3D format, not depth-enhanced formats.

The content-based copy detection of 3D videos is a fairly new problem. The only work that we are aware of is by Ramachandra et al. [2008] where they propose a method to protect multiview 3D videos using a fingerprint based on Scale-Invariant Feature Transform (SIFT) [Lowe 2004], a local feature extractor and descriptor. They extract SIFT descriptors of each of the views of a multiview query video, and compare it to those of an original video. A problem with this work is that their evaluation is performed at the frame level, and the authors do not explain how they decide whether a video is a copy or not, nor do they identify the location of a copied clip in the reference video.

Although 3D copy detection methods are scarce in the literature, there are many methods available for 2D video copy detection. Hampapur et al. [2002] use the temporal features of the video as the fingerprint. They describe a signature based on motion of the video across frames. In a similar way, Tasdemir and Cetin [2010] use the motion vectors for the signature of a frame. Some other methods [Li and Chen 2010; Zhang and Zou 2010] use fingerprints which are obtained directly from compressed videos. Another group of methods use color histograms as videos' fingerprint. For instance, Hampapur et al. [2002] use YUV color space. It quantizes Y into 32 bins and each of U and V into 16 bins to produce a 64-bin histogram. The color histogram signature is the sequence of histograms at each frame. Matching is done by maximizing the histogram intersections between the test and the reference video. The color histogram signature is prone to global variations in color which are common when recoding video. Other group of methods use interest points of video frames as signature. Liu et al. [2010] use local features that are extracted by SIFT as the frame signature. Roth et al. [2010] take every frame of the source video and divide it into 16 regions. They then use Speeded-Up Robust Features (SURF) [Mahanti et al. 2008] to find local points of interest in the frame.

Although all of these methods can be used for 3D video copy detection, they are designed for 2D videos, and they ignore the information in different views and the depth of videos, which are important especially in the presence of 3D video transformations such as view synthesis. The importance of using depth and visual information together to increase the performance is shown in Appendix B.

A preliminary version of this work appeared in Khodabakhshi and Hefeeda [2012]. The current article presents the complete design of the Spider system, which was not presented before. It also proposes a new method for creating signatures, which are much more compact than the ones in Khodabakhshi and Hefeeda [2012]. In addition, this article substantially improves the evaluation of the system by adding twenty 3D videos from YouTube.

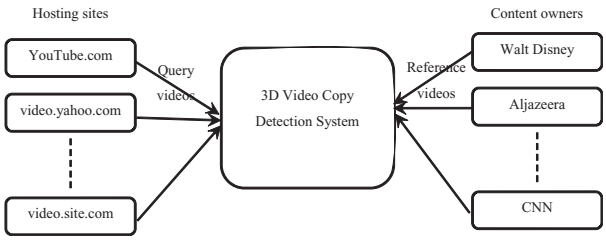


Fig. 1. High-level illustration of the video copy detection system.

3. PROPOSED 3D VIDEO COPY DETECTION SYSTEM

In this section, we start by presenting an overview of the proposed system and how it can be used. Then, we present the details of its different components. Then, we analyze its space and time complexity.

3.1 Overview

We propose a novel system to detect 3D video copies. Figure 1 shows a high-level illustration of the system. The system can be used in different scenarios, including the following.

—*Content Owners.* A copyright owner can deploy the copy detection system, which periodically crawls online sites and downloads recently posted videos. These videos are then compared against the owners' videos to find potential copies.

—*Hosting Sites.* A video hosting site can offer content owners a service of detecting copies of their copyrighted materials for possible actions. Copies of certain videos could even be prevented from being posted on the hosting Web site.

—*Third-party Offering Video Copy Detection Services.* A third-party company can deploy the video copy detection system on behalf of one or more content owners.

The 3D video copy detection system has two main components: processing reference videos and comparing query videos. We describe each of them in the following. Note that we refer to original videos as *reference* videos. Videos that we check against reference videos are called *query* videos. If a query video matches one of the reference videos, that query video is called a copied video.

3.2 Processing Reference Videos

The first component of the system is processing reference videos. Each reference video is processed once to create its signature, which is later used to detect potential copies. The signature is composed of depth information as well as visual features extracted from frames of the video. Signatures of reference videos are stored in a way that facilitates searching and comparison against query videos. Description of each component is presented next.

3.2.1 *Extract Depth.* The depth in a 3D video is emulated by presenting two slightly different views to the human eyes. The human brain fuses these two views to perceive the third dimension. Depending on the display technology, goggles may be needed to control the view seen by each eye and at what time. Different methods exist for preparing and coding 3D videos.

—*Stereo Video.* The video has two views. A view can be thought of as a separate 2D video stream.

—*Multiview Video.* The video has multiple views and a subset of them is displayed to the user depending on the angle of viewing.

—*Video plus Depth.* In this case, the video is encoded in 2D and a separate depth map is created for the 2D video. The depth map allows the creation of many virtual (synthesized) views, which adds flexibility and supports wider viewing angles for users. Creating of virtual views, however, is computationally expensive and could introduce some visual artifacts.

Combinations of the preceding methods are possible, as described in Kauff et al. [2007] and Merkle et al. [2010]. For example, a 3D video can be encoded in multiview plus depth, where a few views are used with the depth map to create more virtual views.

For 3D videos encoded in video plus depth format (or its variants), the depth information of each frame is usually represented as a gray-level image showing the depth of each pixel in that video frame.

For 3D videos encoded in stereo or multiview formats, where no depth information is explicitly given, a method for estimating the depth information is used, which is based on the following. Human eyes are horizontally separated by about 50–75 mm depending on each individual. Consequently, each eye has a slightly different view of the world. This difference between the points of projection in the two eyes is called binocular disparity. Disparity between a stereo pair of images can be used to extract the depth information, since the amount of disparity is inversely proportional to the distance from the observer. Generating disparity images is called stereo matching, which is the process of taking two or more images and estimating a 3D model of the scene by finding corresponding pixels in the images and converting their 2D positions into 3D depths. Szeliski [2013] provides taxonomy of methods available in the literature for correspondence matching. It is worth mentioning that there are both hardware-based and software-based approaches available to generate depth information in real time [Wang et al. 2006].

3.2.2 *Create Depth Signature.* After extracting the depth map which is a gray-level image, the depth signature is computed in two steps. First, the depth map is divided into a grid. The division can be uniform, that is, into equal size blocks, or nonuniform to account for different importance of the regions in the depth map. The number of blocks in the grid is a configurable parameter which trades off the computational complexity with the copy detection accuracy. We found out that a 20×20 uniform blocks grid gives us a good accuracy in an acceptable time.

In the second step of creating the depth signature, the blocks of the depth map grid are summarized into a vector, where each element of the vector represents one block. Various metrics can be used to summarize the depth information in each block. We use mean of the depth values in each block to summarize the whole block. More complex metrics that are composed of multiple components, for example, the mean and standard deviation, can also be used. The depth signature for a video frame takes the form $< d_1, d_2, \ldots, d_D >$, where D is the total number of blocks in the depth map grid, and d_i is the mean of depth values in block i.

The depth signature can be created for every frame in the video. It can also be created for only a subset of the frames in order to reduce the computational complexity. This subset of the frames can be chosen deterministically, for example, each 10th frame is chosen, or randomly. In addition, the subset of the frames can be the keyframes of the video, where a keyframe is a representative frame for a sequence of video frames containing similar visual information, which is referred to as a video shot. Shot boundary detection algorithms such as Liu et al. [2007] can be employed to identify when a shot starts and ends. Keyframe selection algorithms such as Gong and Liu [2000] can be used to select keyframes. The depth signature of a video is composed of the depth signatures of its frames, or the chosen subset of frames for which the depth signatures are created.

3.2.3 *Index Depth Signature.* Depth signatures are vectors with multiple dimensions. These vectors will need to be compared against depth vectors from other videos in order to find potential copies. We index depth signatures in order to facilitate these comparisons. In particular, given a depth vector from a query video, we are interested in finding the closest depth vectors from the reference video

database. Multiple methods such as randomized kd-tree, k-means, and Locality-Sensitive Hashing (LSH) can be used to achieve this nearest-neighbor search efficiently. Based on the requirements of the system, like the performance or the need to be dynamic, a method can be chosen. We propose to use kd-tree for nearest-neighbor search for a large-scale video copy detection system, because it is dynamic and can be distributed for scaling purposes. More explanation is provided in Section 4.3 for this choice.

Here we present a brief background on randomized kd-tree to find approximate nearest neighbors, and then show how it can be used in our system. In the standard version of kd-tree [Friedman et al. 1977], starting with N points in R^d, the data space is split on dimension i in which the data exhibits the greatest variance. An internal node is created to store dimension i and its median value, so an equal number of points fall to left and right subtrees. Then, the same process is repeated with both halves of the data, and finally the data are stored in the leaves of the tree. Since median values are used, the created tree is balanced with depth $\lceil log_2 N \rceil$. To find the nearest neighbor to a query point q, first the leaf that point falls in is found. The point in this leaf is a good candidate for query point nearest neighbor. Then a backtracking stage begins in which whole branches of the tree can be pruned if the region of space they represent is further from the query point than the closest neighbor seen so far. The search terminates when all unexplored branches have been pruned. However, this approach is not efficient in high-dimensional spaces, and also we are looking for approximate nearest neighbors, not the exact one. So we can limit the number of leaf nodes we are willing to examine, and return the best neighbors found up to that point.

Silpa-Anan and Hartley [2008] proposed an improved version of kd-tree in which multiple randomized kd-trees are created, and simultaneous searches are performed using several trees. Randomized kd-trees are created by choosing the split dimension randomly from the first D dimensions on which data has the greatest variance. They also use a priority queue across all trees to explore nodes based on their distances to the query, instead of backtracking based on the tree structure. Their experiments show that this technique leads to 3-times search speedup with the same performance. We used this improved version in our system, and for each signature we store the following fields: $< VideoID, FrameID, ViewID, DepthSignature >$, so that whenever a signature is returned, we know its corresponding video, frame, and view.

3.2.4 *Extract Visual Features.* We extract features from individual frames of videos. Visual features should be robust to, that is, do not change because of, various transformations such as scaling, rotation, change in viewpoint, and change in illumination. Different types of visual features can be used in our system, including but not limited to SURF [Mahanti et al. 2008] and SIFT [Lowe 2004]. In our implementation, we use SURF features, which outperform previous methods with respect to repeatability, distinctiveness, and robustness, and also can be computed much faster [Mahanti et al. 2008].

3.2.5 *Create Visual Signature.* In our previous work [Khodabakhshi and Hefeeda 2012], we used actual feature descriptors, SIFT descriptors, as visual signatures. This visual signature is large and hard to compare, since there are 200 signatures and each has 128 elements per frame. Using the number of visual features instead of their descriptors has been shown to provide good performance in Roth et al. [2010] and Harvey and Hefeeda [2012]. Thus, in the proposed system, we use a modified version of the signature introduced in Harvey and Hefeeda [2012] where only the count of visual features is used. This visual signature is spatio-temporal. Spatial information is obtained by dividing each video frame into regions. In each region, local SURF features are found and the number of features in each region is counted. To add the temporal information to the signature, the counts in each region are sorted along a time line and an ordinal value based on its temporal rank is assigned. We note that in Harvey and Hefeeda [2012], one signature is computed for the whole query video, which makes detecting copied videos embedded in other videos difficult. To overcome this, our proposed signature

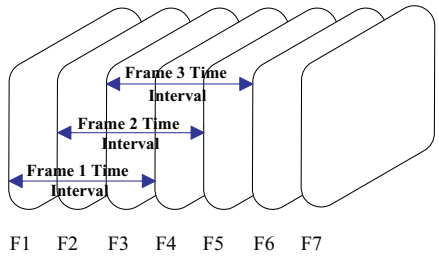


Fig. 2. Frames considered in visual signature computation of each frame.

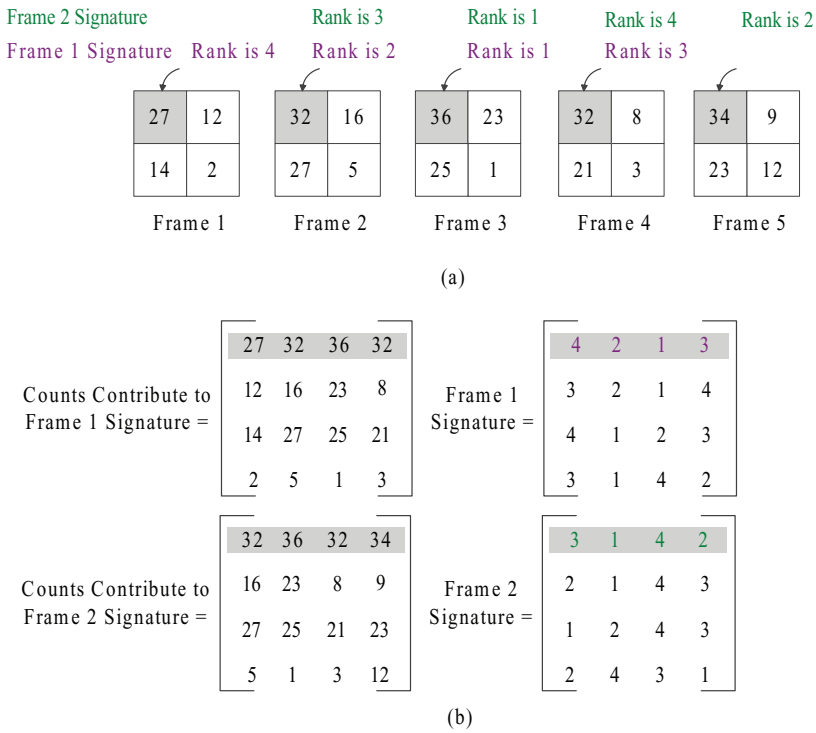


Fig. 3. An example of computing the visual signature.

generates a spatial-temporal signature for every frame, but we include a limited temporal information in each one. As shown in Figure 2, we generate the signature of a specific frame by considering a time interval that starts at that frame, and has a specific length which defines how much temporal information we want to include in each signature. For example, in our implementation, we used a 10-frame interval, and a 4×4 grid, which results in a $4 \times 4 \times 10 = 160$ element vector signature per frame. Compared to the old visual signature, it is much smaller, but as will be shown in the experiments section, is still distinctive and gives good results.

An example of extracting this signature is shown in Figure 3. To make it simpler to demonstrate, a 2×2 grid and a four-frame-wide interval are considered for signature extraction. In Figure 3(a), each frame is divided to four regions, and the feature count in each region is shown inside it. To extract each frame signature, the ordinal scores of each region in the frame time interval are computed. The

ordinal scores of the first region, which is shown in gray background, are shown above the frames for frames 1 and 2. In Figure 3(b), the same procedure is performed for other regions, and finally the frame signature is the ordinal scores.

The visual signature for a video frame takes the form $< v_1, v_2, \ldots, v_V >$, where V is number of blocks$\times$ number of frames in the time interval, and v_is are ordinal scores.

Similar to the depth signature, the visual signature can be created for every frame in the video, or a subset of the frames in order to reduce the computational complexity. The visual signature of a video is composed of the visual signatures of its frames, or the chosen subset of frames for which the visual signatures are computed.

3.2.6 *Index Visual Signature.* Visual signatures are vectors with multiple dimensions. These vectors will need to be compared against visual vectors from other videos in order to find potential copies. Like depth signatures, we index visual signatures in order to facilitate these comparisons. Again, we use kd-tree to index visual signature, and the points added with the following fields: $< VideoID, FrameID, ViewID, VisualSignature >$, so that whenever a signature is returned, we know its corresponding video, frame, and view.

3.3 Processing Query Videos

The second component of the proposed system is comparing query videos. The depth signature is first computed from the query video. As mentioned before, the signature can be computed for only a subset of video frames. Subsampling is more important in case of query video, since we want to respond to a query as fast as possible. The methods used to extract query signatures are the same as the ones used to process reference videos. Then, the depth and visual signature of the query video is compared against the depth and visual signatures in the reference video database. If there is no match, the query video is not considered for any further processing. If there is a match, a combined score is computed based on the depth signature and visual signature matching scores. Finally, the combined score is used to decide whether the query video is a copy of one of the videos in the reference video database. If a query video is found to be a potential copy of a reference video or part of it, the location of the copied part in the reference video is identified. More details are provided in the following.

3.3.1 *Compare Depth Signatures.* Finding the potential copied videos using depth signature takes place in two main steps: frame-level comparison and video-level comparison. The goal of the first step is to find the best matching frames in the entire database for each query frame and compute a score between each matched pair. The goal of the second step is to account for the temporal aspects of the video frames, and to compute a matching score between the query video and each reference video.

In the first step, for each depth signature of the query video, the depth signatures that are closest to it based on their Euclidean distance are found using the nearest-neighbor search method. Using kd-tree, for each depth signature a fixed number of its approximate nearest neighbors and their distances are found. Distances can be used as matching scores. Alternatively, a threshold can be used such that scores for distances exceeding the threshold can be set to zero and other scores are set to 1. This will reduce the computation needed to compare scores. It should be noticed that the frames found in this may belong to different videos.

In addition, 3D videos can have multiple views, and a signature from the query frame should be checked against frames from different views. Two frames are considered a match if at least one of their views matches. Finally, a score is computed for each matched pair of frames using the distance of their views. At the end of this step, the number of matched frames in each reference video is counted. Then, reference videos with the number of matched frames exceeding a threshold are considered in the next step. Other videos are no longer considered.

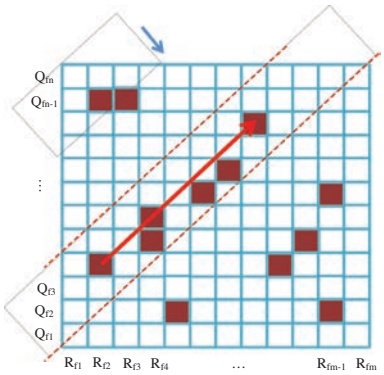


Fig. 4. Matching matrix for frames from query and reference videos considering their timing.

In the second step of the depth signature matching, the temporal characteristics of the videos are considered. Temporal characteristics mean the timing and order of the frames in the query and reference videos. For example, if frame x in the query video matches frame y in the reference video, we expect frame $x + 1$ in the query video matches frame $y + 1$ in the reference video. This is important to account for as copied videos are typically clips with contiguous frames taken from reference videos. Also, a copied video can be embedded in other videos.

In order to consider the temporal characteristics, a matching matrix is computed for each candidate reference video and the query video. The columns of this matrix represent reference video frames, and the rows represent the query video frames. Entries in the matrix are the relevance scores. Figure 4 shows an example where dark squares represent matched frames. Using this matrix, the longest sequence with the largest number of matching frames is considered as a potential copy. If the frame rate of the original video and its copy are the same, this sequence's gradient (slope) would be equal to 1. However, since change of frame rate transformation may be applied to the copied video, we also consider sequences with gradients 0.5, and 2. These gradients correspond to two extreme cases when the frame rate of the query video is changed to half and twice of its original, respectively. Other frame rate changes are between these three cases, and are still detected, as is examined in Appendix B.1.

It is worth mentioning that frame dropping and occasional frame mismatches caused by possible transformations must be taken into account. Thus, the sequences mentioned before are not strictly linear and gaps may exist. So, to find the longest sequence with the greatest score, instead of considering a line of frames, a band with a specific width is considered, as shown in Figure 4. This band, with one of the gradients mentioned earlier, starts sweeping the matrix from top leftmost position and moves one block each time. At each position, the temporal score of the longest sequence of matched frames inside the band is computed. After performing this process for all positions, the position with the greatest temporal score is considered the potential copied location, and its score is considered the depth matching temporal score of the reference video.

3.3.2 *Compare Visual Signatures.* Like depth signature comparison, visual signatures comparison takes place in two steps, frame level, and video level. First, for each query frame visual signature, the visual signatures that are closest to it based on their Euclidean distance are detected using the nearest-neighbor search method. Using kd-tree, for each visual signature a fixed number of its approximate nearest neighbors and their distances are found. These returned signatures may belong to different videos. To find the matched videos, the number of matched frames in each reference video is counted, and videos with the number of matched frames exceeding a threshold are considered a

match. At the video-level comparison, like the one explained for depth video-level matching, the temporal characteristics are taken into account, and a temporal score is computed between the query video and each potential reference video. Finally, the best matching videos based on their temporal scores are considered as potential copies.

3.3.3 *Identify Matching Clip.* Copied videos can be small clips of the reference videos. It is useful to automatically identify the location of a copied clip in the reference video. We use the matching matrix shown in Figure 4 to identify the location of the copied clip. Notice that we have two matching matrices: one from matching depth signatures and the other from matching visual signatures. We can either use one of them or both. We find the longest diagonal sequence with the greatest score in each case. The start and end of the longest sequence give the start and end location of the copied clip in the reference video. Using both of depth and visual matching matrices can yield more accurate locations of the copied clips. In this case, the intersection of the two sequences returned from comparing depth and visual signatures is used to mark the start and end location in the reference video.

3.4 Complexity Analysis

We analyze the space and time complexity of the proposed system. Space complexity refers to the storage needed to store the signatures of reference videos. Signatures of query videos are created online and compared against the signatures of reference videos in the database. We analyze the space complexity as a function of total number of frames in all reference videos. We use kd-tree to index signatures in order to facilitate nearest-neighbor search. To calculate the space needed to store the index, we note that we have to store signatures, their information, and the trees. To store the signatures themselves, in addition to signatures' feature vectors, we have to store video ID and frame ID of each signature. We assume that we use 4 bytes for video ID, and 4 bytes for frame ID of each signature, which is large enough for long videos and large video databases. To store the trees, we have to store internal nodes and leaves. In a kd-tree with n leaves, there are $n-1$ internal nodes. The internal nodes need to store their splitting dimension and the threshold values (assumed to store each of them in a single byte), and the leaves need to store the index of the signature, which can be stored in $log_2 n/8$ bytes. Thus, the space required to store a kd-tree index can be computed using Eq. (1). We have

$$\begin{aligned} Storage \;=\; & n(sd+4+4) && \text{Storing signatures and their information} \\ & +n(2T_{kd}+T_{kd}\tfrac{log_2 n}{8}) && \text{Storing the trees} \end{aligned} \tag{1}$$

where d is the signature dimension, s is number of bytes per feature, and T_{kd} is the number of kd-trees. Now we replace the general terms with the specific values of our system. Let the total number of frames in all reference videos be N_r. Also, in our implementation we used FLANN library, with 4 trees, so $T_{kd} = 4$. The dimensions of depth signature, and visual signature are 400 and 160, respectively. Each depth signature feature is between 0 and 255, and each visual signature feature is between 0 and 9, so for both features we have $s = 1$ byte. Using these parameters, the storage required to store our depth and visual indexes is $O(N_r log_2 N_r)$.

Next, we analyze the time complexity, which is the time needed to process a given query video of length N_q. In our analysis, we consider the worst-case scenario, which happens when the query video matches one of the reference videos. We assume that the time taken to compute the depth and visual signatures for a frame is T_d and T_v, respectively. T_d and T_v do not depend on N_q, but depend on the frame resolution, which is constant. Comparing signatures and identifying the location of a copied clip do depend on N_q.

For comparison of signatures, we search through the kd-tree to find approximate m nearest neighbors. This involves scaler comparison operations at each level of the tree, limited number of

backtracking (typically $B \sim 250$) to examine more leaves, computing the distance between query and each leaf signatures, and linear search in the final list of signatures B to find the best m matches in the list. We know that the height of the tree is $log_2 N_r$, and that computing the distance between query and leaf signatures can be performed with d multiplications and d additions. So the running time of a single search in the tree is $B(2d+log_2 N_r)+B$. Thus, for comparing the depth and visual signatures for a query video that has N_q frames, we need $250N_q(801 + log_2 N_r)$ and $250N_q(321 + log_2 N_r)$ time, respectively.

In the Spider system, we found out that using $m = 50$ suffices for good performance. So the $m = 50$ best matching frames for each frame are used to vote for corresponding reference videos, and K best matching videos are found in $mN_q + R$, where R is the number of reference videos in the database. Then, we compare these K reference videos against the query video, constructing the matching matrix, such as the one shown in Figure 4, between the query video and the reference video, which takes LKN_q steps, where L is a constant referring to the number of frames in the longest video. Finally the temporal score is computed using this matrix. To do so, as explained in Section 3.3.1, each matrix element is traversed once, and this takes LN_q addition, at most, to compute the temporal score and matched location. Adding all up, the worst-case running time to process a query video of length N_q is

$$\begin{aligned} Time &= 250N_q(801 + log_2^{N_r}) + 250N_q(321 + log_2^{N_r}) + 2(50N_q + R + LKN_q + LN_q) \\ &= 2R + N_q(280600 + LK + L) + 500N_q log_2 N_r \\ &= O(N_q log_2 N_r). \end{aligned} \tag{2}$$

The preceding equation considers the fact that L, K, and R are constant and much smaller than N_r.

4. SPIDER SYSTEM

We implemented all steps of the proposed 3D video copy detection system, which we call Spider. We modified and integrated several open-source libraries in our system. We provide some highlights of our implementation in the following.

4.1 Overview

We implemented Spider, including its graphical user interface, in Java. Figure 5 shows a snapshot of the Spider system. The user provides the location of the reference videos that need to be protected, and then the location of a query video that wants to match it against the reference videos. The Spider system compares the selected query video with the database, and returns the detected videos, if any, in a table sorted according to their scores.

The main components of the Spider system are Detection Engine, Index Storage, and Input/Output Management. We explain each of these components in the following.

4.2 Detection Engine

DetectionEngine is the main component that controls everything from signature extraction to matching. In the first run of the Spider, the user provides the location of the reference videos that need to be protected. DetectionEngine extracts their signature, indexes them, and stores the indexes on the disk. The user can also add other reference videos to the index later.

In processing the reference videos phase, DetectionEngine first extracts video frames, texture, and depth from videos, which have different formats. It uses the FFmpeg [ffm 2013] package to extract the frames in grayscale format, since the depth is grayscale itself, and SURF works with grayscale images, and it takes less space to save them as well. Then, these frames are read to the memory for processing. The depth signature and visual signature of each frame is computed, and indexed to facilitate nearest-neighbor search needed in the query processing phase. More explanation on indexing and storage is

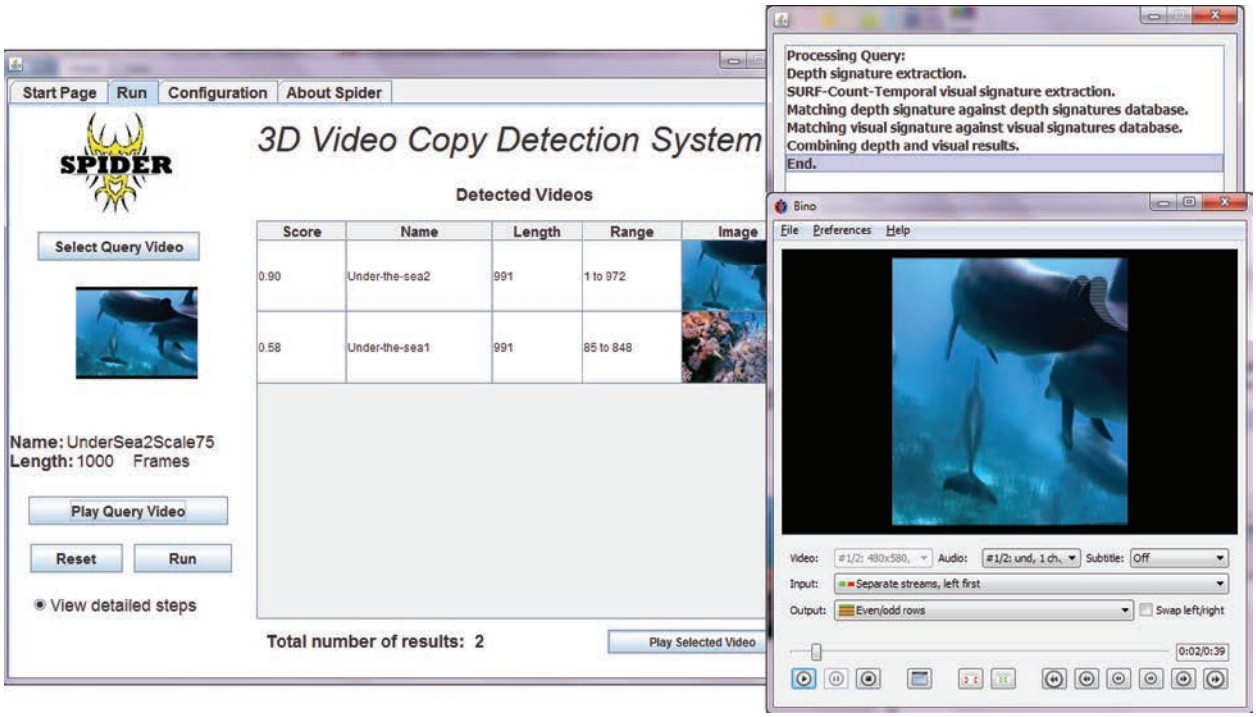


Fig. 5. A screen shot of the demo software. The query video, shown on left, which is scaled version of a reference video is detected to be a copy of the reference videos shown in the table.

provided in Section 4.3. For the visual signature, the system extracts SURF features using jopensurf [jop 2013] open-source library.

In the query processing phase, a query video is selected by the user. Then, the software extracts its frames and signatures, similar to steps taken for reference videos. The matching takes place both for depth and texture of the query. Each matching step returns the reference videos that are detected to be the original version of the query video, and a score between [0, 1] is assigned to each of them. Then, the scores of these two matching steps are combined. For each reference video returned, its score will be the weighted sum of their visual and depth scores. We used the same weight of 0.5 for both depth and texture matching scores. The returned reference videos are shown as the result to the user.

4.3 Index Storage

Multiple methods such as randomized kd-tree, k-means, and Locality Sensitive Hashing (LSH) can be used for indexing to achieve approximate nearest-neighbor search efficiently. Based on the requirements of the system, like the performance or the need to be dynamic, a method can be chosen. Our first choice was LSH [Datar et al. 2004] for nearest-neighbor search for large-scale video copy detection system, because it can be easily parallelized. However, we should consider the fact that our data is very dynamic. In other words, videos may be added to the index at any time, and any newly added video will add many points to the index. We have two choices to deal with this situation in LSH. First is to keep the table sizes fixed, which will put more and more points in the same bucket, and reduce the accuracy. Second is to resize the hash tables periodically, and rehash all the points again, which will reduce the performance. Thus, the number of hash functions or bin sizes parameters affects the trade-off between

runtime and accuracy, and they have to be tuned for every database size under consideration, and we should not use the same settings when enlarging the database. This made LSH less appealing for our software, since the reference videos will be available gradually.

Our next choice was randomized kd-tree [Silpa-Anan and Hartley 2008], since it is dynamic and has good performance. According to Aly et al. [2011b] which did a thorough comparison between different indexing approaches for high-dimensional data, kd-trees provide the best overall trade-off between performance and computational cost. In their experiments, they observed that its runtime grows very slowly with the number of points while giving excellent performance. Moreover, with smart parallelization schemes, like the one proposed and implemented in Aly et al. [2011a], it can significantly speed up when running on multiple machines, so it can be scaled to large video databases. The only drawback of kd-trees is larger storage requirements, which can be addressed by using more compact signatures.

In the Spider system, we use the randomized kd-tree implementation of FLANN library version 1.6.11 [fla 2011] which uses five dimensions to build the tree, and we use 4 randomized kd-trees.

4.4 Input/Output Management

As mentioned earlier, the user provides the location of the reference and query videos via the ConfigPage of the graphical user interface. The system uses the videos' location to extract and store the frames in a specific location. Then, these frames are read to the memory, one by one, by ImageClass as buffered images to speed the reading process.

The output of the system are instances, as shown in Figure 5, are the name, score, length, and a sample image of the detected reference videos. It also shows the location, start and end frames, in the reference video which is detected to be the copied portion of the query video. The user can also play the query and reference videos to compare them visually. When the user wants to play the videos, the system launches the Bino [Bin 2013] 3D video player. The Spider system has a verbose option, in which the user can see the detailed steps as the system is performing them.

5. EXPERIMENTAL EVALUATION

We conducted extensive analysis using real 3D videos to assess the Spider system performance. We start by describing how we prepared our video dataset in the following section. Then, we describe our experimental setup. Then, we present the results of our experiments to show the performance of the proposed system. We note that, due to space limitations we only present a representative sample of our results. Also, some of our results are presented in Appendix B. Specially the results showing: (i) Temporal Transformations, (ii) View Synthesis and Subset of Views Transformation, (iii) Importance of using Depth and Visual Signatures together, (iv) Sensitiveness of Depth signal to alternation, and (v) Depth Signature Grid Size are given in Appendix B.

5.1 Preparing 3D Video Dataset

We collected thirty-seven 3D videos with different characteristics from three resources: YouTube, Microsoft, and Mobile3DTV project. Two of the videos, Ballet and Break Dancers, have eight views each, which are placed along a 1D arc spanning about 30 degrees from one end to the other. These two videos are in bitmap format and their camera calibration parameters are generated and distributed by the Interactive Visual Group at Microsoft Research [MSR 2013]. Depth maps are computed using the method described in Zitnick et al. [2004]. The other 15 videos are obtained from the MOBILE3DTV project [mob 2013]. These are stereo videos, with only two views. These videos and their depth signals are provided in YUV format, which we converted to bitmap format using the FFmpeg package [ffm 2013]. The last 20 videos are stereo videos downloaded from the YouTube video sharing Web site. Then, their depth signals are computed using Triaxes DepthGate [Tra 2013] software.

To create our reference video database, we use one of the eight views (view0) and its associated depth signal from each of the Ballet and Break Dancers videos. And we use the left view and its associated depth from each of the remaining videos except video TU-Berlin. That is, we create 36 reference videos, which are listed in the first 36 rows in Table II (given in Appendix A). The TU-Berlin video is used in creating queries.

We create many different query videos to capture most realistic scenarios. Specifically, we first create three types of queries as described next and illustrated in Table II:

—Type 1: Query videos are segments or clips of reference videos. These are the first 36 rows in Table II.

—Type 2: Query videos are segments of reference videos embedded in other videos. These are rows 37 to 42 in Table II.

—Type 3: Query videos contain no parts of the reference videos. These are the last 6 rows in Table II.

Then, we apply different video transformations on the 48 videos shown in Table II. A video transformation means that the video has been modified either intentionally (to avoid detection) or unintentionally (as a result of the copying process). A transformed video is supposed to provide acceptable perceptual quality to viewers, that is, the transformation can be tolerated by viewers. Transformations of 3D videos can be applied on texture, depth, or both. In addition, transformations can be applied individually, that is, only one transformation is applied on the video, or combined, that is, multiple transformations are applied on the video at the same time.

We apply 4 transformations on each of the first 20 videos in Table II, which are the stereo videos downloaded from YouTube. These transformations are applied to both views of the stereo videos, and then the depth is extracted from the transformed stereo videos. This results in 80 query videos.

—Video Blurring: Both views are blurred using a radius of 3 for the blur disk.

—Video Scaling: Both views are scaled to 75% of their original size.

—Insertion of Logo: A logo is inserted into both views.

—Insertion of Text: Some text is inserted into both views.

We apply the following 9 transformations on each of the last 28 videos in Table II, which results in 252 query videos.

—Video Blurring: This transformation is applied to texture only. The radius of blur disk is chosen randomly from range [0.5 7].

—Video Gamma Correction: This transformation is applied to texture only. The gamma value is chosen randomly from range [0.2 4].

—Video Noise: This transformation is applied to texture only. The noise is chosen randomly from range [0 0.06].

—Crop: This transformation is applied to texture and depth in a way that the same number of pixels are cropped from both texture and its depth. The number of pixels cropped are chosen randomly from range [5 15].

—Logo Insertion: This transformation is applied to texture and depth in a way that the pixels covered by the logo in the video are set to minimum depth.

—Text Insertion: This transformation is applied to texture and depth in a way that the pixels covered by the text in the texture are set to minimum depth.

—Flip: The texture and its depth are flipped.

Table I. Generating Query Videos with Different Frame Rates

Decreased Frame Rate	Original Video Frame Rate	Increased Frame Rate
10	15	24
15	24	30
15	25	30
24	30	50

—Depth Blurring: This transformation is only applied to depth signal. Blurring the depth image smooths sharp horizontal changes in depth images, so fewer holes would appear in the warped views; however, the warped image quality reduces especially around the not edge areas. The radius of blur disk is chosen randomly from range [0.5 7].

—Depth Noise: This transformation is only applied to depth signal, which adds noise to depth frames and causes quality reduction in the warped images. So the noise cannot be very high, as the quality of the warped image may not be acceptable. The noise deviation is chosen randomly from range [0 0.06].

In addition, we apply two types of combined transformations on these 28 videos: 3 and 5 transformations. For the 3 transformations case, we choose 3 different transformations and apply all of them on each of the 28 videos in Table II. One transformation is applied on the texture, another one the depth, and the third is applied on both. Similarly, for the 5 transformations case, 5 different transformations are applied on each video: two on the texture, two on the depth, and one on both. These combined transformations added 2*28 = 56 videos to our query video database.

We also examine common temporal transformations. Fast or slow motion happen when the video's frame rate is changed, while the number of frames is kept fixed. Our system has immunity to this transformation, since it does not change the matching matrix. Random replication or omission of frames is also handled in the matching matrix by considering a band instead of a line of frames. Change of frame rate is the most challenging temporal transformation which keeps the video's total length by dropping, replicating, or merging frames. To test our system against this transformation, FFmpeg [ffm 2013] package is used to change the frame rate of our 48 query videos. We examine both increase and decrease in frame rate. Table I shows our approach to generate the transformed versions, which adds 48*2 = 96 videos to our query video dataset.

Finally, we apply two *new* transformations which are specific to 3D videos: view synthesis and copying a subset of views. In the view synthesis case, we create additional views from given views using view synthesis tools. Synthesized views present the visual content from different angles to viewers, which could reveal objects that were occluded or present objects with different depths or shades. This means that synthesized views can contain different visual and depth information than the original views. Synthesized views can be created and distributed to enrich users' experience or to evade the detection process. In our experiments, we synthesized 18 views using the VSRS reference software for depth estimation and view synthesis tool [VSR 2008] for the BreakDancers video. This creates 18 additional query videos. For the copying subset of the views case, we assume that original 3D videos can have multiple views, and only a subset of them are copied. This can be done to save storage or bandwidth of the hosting site or the viewers. In our experiments, we have two videos that have eight views each, which are Ballet and BreakDancers. For each one of them, we use view0 as the reference view, and we create 7 different queries, one for each of the remaining 7 views. Thus, we add 2*7 = 14 entries to our query videos database.

Including all transformations, the total number of query videos in our experiments is 460.

5.2 Experimental Setup

We conduct the following sets of experiments.

—No Transformations. Query videos are parts of some reference videos and they are not subjected to any transformations.
—Individual Transformations. Each query video is subjected to one transformation, either on the texture or depth.
—Temporal Transformations. Each query video is subjected to change in frame rate transformation, either decrease or increase.
—Combined Transformations. Each query video is subjected to multiple transformations, both on the texture and depth.
—View Synthesis and Subset of Views Transformation. A query video can contain synthesized views or a subset of the original views.
—Importance of using Depth and Visual Signatures together. Study the possibility of using the depth signature only or visual signature only in the copy detection process.
—Sensitiveness of Depth signal to alternation. Show how the depth signature and quality of the synthesized views change as the depth signal undergoes some transformations.
—Depth Signature Grid Size. Compare the results when different grid sizes are used for the depth signature.
—Subsampling. Use only a subset of frames in the copy detection to speed up the process.

We consider two important performance metrics: precision and recall, which are defined in the following two equations.

$$precision = \frac{number\ of\ correctly\ identified\ copies}{total\ number\ of\ reported\ copies} \tag{3}$$

$$recall = \frac{number\ of\ correctly\ identified\ copies}{actual\ number\ of\ copies} \tag{4}$$

5.3 No Transformations

In this experiment, we evaluate the performance of the proposed 3D video copy detection system using query videos that do not have any transformations. This scenario happens, for example, when a clip is directly taken from a digital video stored on a DVD or hard disk. We compare all 49 query videos against the reference video database. We vary the threshold, which determines whether a video is a copy or not based on its score, between 0.0 and 1.0 and compute the precision and recall for each case. We plot the results in Figure 6(a). The figure shows that the proposed system can achieve 100% precision and 100% recall when the threshold value is between 0.7 and 0.8. For a wide range of threshold values between (0.5 and 0.9), the system yields 100% recall and precision.

5.4 Individual Transformations

In this experiment, we apply individual transformations on query videos. This shows the robustness of the proposed copy detection system against video modifications that occur in practice when videos are copied. We apply all transformations described in Section 5.2. This means that we repeat the experiment for each transformation, and in each repetition, we vary the threshold between 0.0 and 1.0 and compute the precision and recall for each case. In all cases, the achieved precision and recall values are more than 90%, and these are obtained for a wide range of threshold values, which shows that our

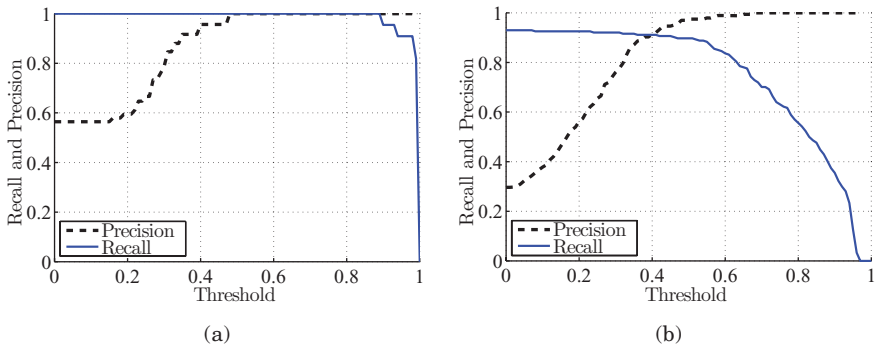


Fig. 6. Precision and recall of the proposed 3D video copy detection system (a) when videos do not have any transformations and (b) when videos are subjected to nine different transformations.

system does not require fine-tuning of the threshold parameter. We notice that some transformations, for example, texture blur, have more impact on the precision and recall than other transformations such as flip.

We plot the average results across all transformations in Figure 6(b), where we compute the average precision and recall values across all experiments for the corresponding values of the threshold. The figure shows that, on average, our system can result in 92% precision and recall at the same time (the intersection point). By using the threshold parameter, administrators can control the performance of 3D video copy detection systems based on the requirements of the system. For example, in some systems, 100% precision is desired even if some copies are not detected. For such systems, higher threshold values can be used.

In summary, the results in this section show that the proposed system yields high precision and recall values in presence of various video transformations.

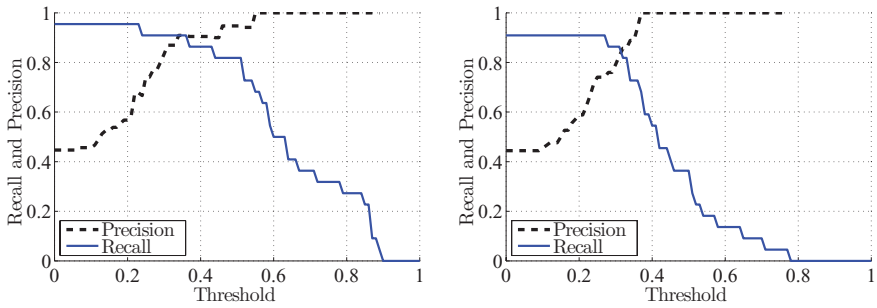
5.5 Multiple Transformations

We assess the performance of our system in quite challenging scenarios, where each query is subjected to multiple transformations at the same time. We experiment with two cases. First, when each video is subjected to 3 different transformations, one applied on texture, one on depth, and one on both. The 3 transformations are chosen randomly for each query video. In the second case, we apply 5 transformations, 2 on texture, 2 on depth, and one on both. We plot the average precision and recall values for these cases in Figures 7(a) and 7(b). The results in Figure 7(a) show that the proposed system achieves high precision and recall values around 90%, even when query videos are subjected to three different transformations. For extreme situations where query videos are subjected to 5 different transformations, the precision and recall values are still more than 80%. These experiments demonstrate the robustness of the proposed system to multiple video transformations.

5.6 Subsampling

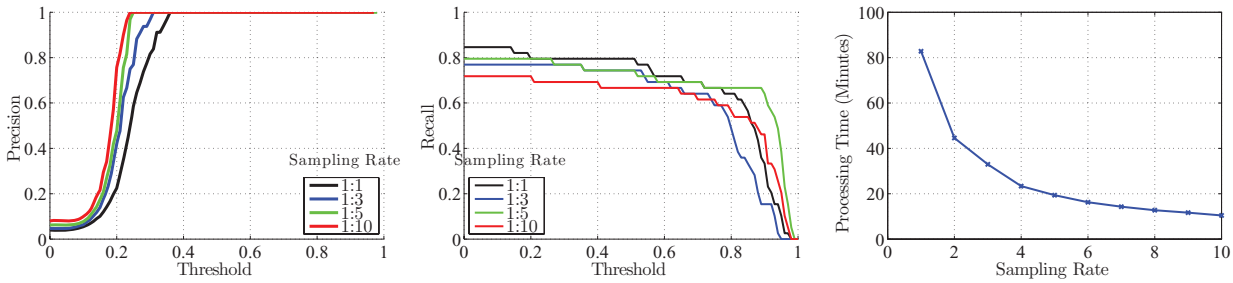
As mentioned in Section 3.2.2, subsampling in the copy detection context means that instead of extracting the signature for every frame in the video, we can extract the signature for only a subset of the frames in order to reduce the computational complexity. However, this may affect the performance of the system. So there is a trade-off between the processing time and accuracy of the system.

Using the Spider system, and the reference and query videos explained in Section 5.1, different sampling rates are examined to measure the performance. For each sampling rate, all 48 queries are processed and the total processing time, precision, and recall are used to evaluate the performance.



(a) 3 transformations (b) 5 transformations

Fig. 7. Precision and recall of the copy detection system when videos are subjected to multiple transformations.



(a) (b) (c)

Fig. 8. Precision,recall, and processing time of the copy detection system, called Spider, using different sampling rate.

Here a sampling ratio of $1 : X$ means we consider every Xth frame in the detection process. The total processing time and the precision and recall for a fixed threshold (0.4) are measured. The results are shown in Figures 8. As shown in these figures, the recall of the system decreases by a small amount as the sampling rate decreases, which causes the precision to increase, since some of the documents which are not declared as being a copy are actually false alarms. However, the change in precision and recall is not noticeable, while the processing time reduces significantly. For example, for a sampling rate of 1:6, the total processing time drops from over 80 minutes to less than 20 minutes. Therefore, subsampling can be used to increase the scalability of the system without reducing its performance significantly.

We note that the total query processing time of our system is spent in three main steps: depth signature extraction, visual signature extraction, and matching. From our experiment, the second step took 4.5 minutes, which is approximately 68% of the total processing time. The other stages took 1.6 and 0.5 minutes for first and third steps, respectively.

6. CONCLUSIONS

Three dimensional (3D) videos are getting quite popular, and creating 3D videos is expensive. Thus, protecting 3D videos against illegal copying is an important problem. We presented the detailed design of a 3D video copy detection system. The system has two main components: processing reference videos and processing query videos. In the first component, the system creates compact signatures of the depth and texture of the reference videos and stores them in a database. The second component creates

similar signatures for each query video and compares them against signatures in the database. If a match is found, the location of the copied part in the reference video is also identified.

We implemented the proposed system and evaluated its performance in terms of precision and recall using many 3D videos. Some of these videos have two views, where the others have eight different views. We created a large set of query videos, which has a total of 460 3D videos. We carefully customized the query videos to represent most practical scenarios for copying 3D videos. Specifically, our query videos represent the following scenarios: (i) query videos are segments of some reference videos, (ii) each query video is subjected to nine different transformations, either on the texture or depth, (iii) multiple combined transformations are applied on the texture and depth of each video, (iv) new views are synthesized from existing ones, and (v) query videos have only a subset of views of reference videos. Our experimental results show that the proposed system achieves high precision and recall values in all scenarios. Specifically, the proposed system results in 100% precision and recall when copied videos are unmodified parts of original videos, and it produces more than 90% precision and recall when copied videos are subjected to different individual transformations. These results are achieved using a unified threshold around 0.4. Even in the extreme cases where each video is subjected to five different transformations at the same time, our system yields precision and recall values more than 75%. Furthermore, the preceding results are obatained for a wide range of the threshold parameter used in the system, which means that our system does not need fine-tuning of that parameter.

REFERENCES

Bino Free 3D Player. 2013. http://bino3d.org/

FFmpeg. 2013. http://www.ffmpeg.org/

Jopensurf. 2013. http://code.google.com/p/jopensurf

Microsoft Research. 2013. MSR 3D video. http://research.microsoft.com/en-us/um/people/sbkang/3dvideodownload/

Mobile 3DTV. 2013. http://sp.cs.tut.fi/mobile3dtv/video-plus-depth/

Traixes DepthGate. 2013. http://doc.triaxes.tv/depthgate.

ISO. 2008. ISO/IEC JTC1/SC29/WG11. Reference softwares for depth estimation and view synthesis. Doc. M15377.

FLANN. 2011. FLANN - Fast library for approximate nearest neighbors. http://www.cs.ubc.ca/ mariusm/index.php/FLANN.

ALY, M., MUNICH, M., AND PERONA, P. 2011a. Distributed kd-trees for retrieval from very large image collections. In *Proceedings of the British Machine Vision Conference (BMVC)*.

ALY, M., MUNICH, M., AND PERONA, P. 2011b. Indexing in large scale image collections: Scaling properties and benchmark. In *Proceedings of the IEEE Workshop on Applications of Computer Vision (WACV'11)*. 418 –425.

CHEN, W.-Y., CHANG, Y.-L., LIN, S.-F., DING, L.-F., AND CHEN, L.-G. 2005. Efficient depth image based rendering with edge dependent depth filter and interpolation. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME'05)*. 1314 –1317.

DATAR, M., IMMORLICA, N., INDYK, P., AND MIRROKNI, V. S. 2004. Locality-Sensitive hashing scheme based on p-stable distributions. In *proceedings of the 20th Annual Symposium on Computational geometry (SCG'04)*.253–262.

FRIEDMAN, J. H., BENTLEY, J. L., AND FINKEL, R. A. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw. 3,* 3, 209–226.

GONG, Y. AND LIU, X. 2000. Video summarization using singular value decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'00)*. 174 –180.

HAMPAPUR, A., HYUN, K., AND BOLLE, R. M. 2002. Comparison of sequence matching techniques for video copy detection. In *Proceedings of the SPIE Conference on Storage and Retrieval for Media Databases (SPIE'02)*. 194–201.

HARVEY, R. C. AND HEFEEDA, M. 2012. Spatio-temporal video copy detection. In *proceedings of the ACM Multimedia Systems conference (MMSys'12)*. Chapel Hill, NC, USA.

KAHNG, A. B., LACH, J., MANGIONE-SMITH, W. H., MANTIK, S., MARKOV, I. L., POTKONJAK, M., TUCKER, P., WANG, H., AND WOLFE, G. 1998. Watermarking techniques for intellectual property protection. In *Proceedings of the 35th Annual Design Automation Conference (DAC'98)*. 776–781.

KAUFF, P., ATZPADIN, N., FEHN, C., MLLER, M., SCHREER, O., SMOLIC, A., AND TANGER, R. 2007. Depth map creation and image-based rendering for advanced 3DTV services providing interoperability and scalability. *Signal Process. Image Comm. 22,* 2, 217–234.

KHODABAKHSHI, N. AND HEFEEDA, M. 2012. Copy detection of 3d videos. In *Proceedings of the ACM Multimedia Systems Conference (MMSys'12)*. Chapel Hill, NC, USA.

KOZ, A., CIGLA, C., AND ALATAN, A. 2010. Watermarking of free-view video. *IEEE Trans. Image Process. 19,* 7, 1785 –1797.

LI, Z. AND CHEN, J. 2010. Efficient compressed domain video copy detection. In *Proceedings of the International Conference on Management and Service Science (MASS'10)*. 1–4.

LIU, Z., GIBBON, D., ZAVESKY, E., SHAHRARAY, B., AND HAFFNER, P. 2007. A fast, comprehensive shot boundary determination system. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME'07)*. 1487 –1490.

LIU, Z., LIU, T., GIBBON, D. C., AND SHAHRARAY, B. 2010. Effective and scalable video copy detection. In *Proceedings of the International Conference on Multimedia Information Retrieval (MIR'10)*. 119–128.

LOWE, D. G. 2004. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision 60,* 2, 91–110.

MAHANTI, A., EAGER, D., VERNON, M., AND SUNDARAM-STUKEL, D. 2008. Speeded-up robust features (SURF). *Comput. Vis. Image Understand. 110,* 3, 346–359.

MERKLE, P., MULLER, K., AND WIEGAND, T. 2010. 3D video: Acquisition, coding, and display. In *Proceedings of the International Conference on Consumer Electronics Digest of Technical Papers (ICCE'10)*. 127 –128.

RAMACHANDRA, V., ZWICKER, M., AND NGUYEN, T. 2008. 3D video fingerprinting. In *Proceedings of the 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV'08)*. 81 –84.

ROTH, G., LAGANIE ANDRE, R., LAMBERT, P., LAKHMIRI, I., AND JANATI, T. 2010. A simple but effective approach to video copy detection. In *Proceedings of the Canadian Conference on Computer and Robot Vision (CRV'10)*. 63 –70.

SHUM, H. Y., LI, Y., AND KANG, S. B. 2004. An introduction to image-based rendering. In *Integrated Image and Graphics Technologies*, D. Zhang, M. Kamel, and G. Baciu, Eds., The Kluwer International Series in Engineering and Computer Science, vol. 762, Springer.

SILPA-ANAN, C. AND HARTLEY, R. 2008. Optimised kd-trees for fast image descriptor matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*. USA, 1–8.

SMOLIC, A., MÜLLER, K., STEFANOSKI, N., OSTERMANN, J., GOTCHEV, A., AKAR, G. B., TRIANTAFYLLIDIS, G., AND KOZ, A. 2007. Coding algorithms for 3DTV-a survey. *IEEE Tran. Circ. and Syst. Video Technol. 17,* 11, 1606–1621.

SZELISKI, R.2013. Stereo correspondence. In *Computer Vision*, D. Gries and F. B. Schneider, Eds., Texts in Computer Science., Springer, 467–503.

TASDEMIR, K. AND CETIN, A. 2010. Motion vector based features for content based video copy detection. In *Proceedings of the International Conference on Pattern Recognition (ICPR'10)*. 3134 –3137.

WANG, L., LIAO, M., GONG, M., YANG, R., AND NISTER, D. 2006. High-quality real-time stereo using adaptive cost aggregation and dynamic programming. In *Proceedings of the 3rd International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*. 798 –805.

ZHANG, Z. AND ZOU, J. 2010. copy detection based on edge analysis. In *Proceedings of the IEEE International Conference on Information and Automation (ICIA'10)*. 2497 –2501.

ZITNICK, C. L., KANG, S. B., UYTTENDAELE, M., WINDER, S., AND SZELISKI, R. 2004. High-quality video view interpolation using a layered representation. *ACM Trans. Graph. 23,* 3, 600–608.

Received February 2012; revised May 2012; accepted May 2012