A Hybrid Layered Video Encoding Technique for Mobile Internet-based Vision

Suchendra M. Bhandarkar[†], Siddhartha Chattopadhyay[‡], Shiva Sandeep Garlapati[†]
[†]Dept. of Computer Science, The University of Georgia, Athens, Georgia 30602-7404, USA
[‡]Google Inc., Mountain View, California 94043, USA

Abstract. The increasing deployment of broadband networks and simultaneous proliferation of low-cost video capturing and multimedia-enabled mobile devices have triggered a new wave of mobile Internet-based computer vision applications. However, mobile networked environments are typically resource constrained in terms of the available bandwidth and battery capacity on mobile devices. Computer vision applications that entail analysis, transmission, storage and rendering of video data are typically resource-intensive. Since the available bandwidth in the mobile Internet is constantly changing and the battery life of a mobile video capturing and rendering device decreases with time, it is desirable to have a video representation scheme that adapts dynamically to the available resources. A *Hybrid Layered Video* (HLV) encoding scheme is proposed, which comprises of content-aware, multi-layer encoding of the image texture and a generative sketch-based representation of the object outlines in the input video stream. Different combinations of the texture- and sketch-based representations result in distinct video states, each with a characteristic bandwidth and power consumption profile. The proposed HLV encoding scheme is shown to be effective for mobile Internet-based vision applications such as background subtraction, face detection, face recognition and face tracking on resource-constrained mobile devices.

1. INTRODUCTION

The increasing deployment of broadband networks and simultaneous proliferation of low-cost video capturing and multimedia-enabled mobile devices, such as pocket PCs, cell phones, PDA's and iPhones during the past decade have triggered a new wave of mobile Internet-based vision applications. Computer vision applications that entail analysis, transmission, storage and rendering of video data are typically resource-intensive. Mobile networked environments, on the other hand, are typically resource constrained in terms of the available bandwidth and battery capacity on mobile devices, and are also characterized by constantly fluctuating bandwidth and decreasing device battery life as a function of time. Consequently, it is desirable to have a hierarchical or layered video encoding scheme where distinct layers have different resource consumption characteristics that can be mapped onto the available resources at any given point in time [11]. Traditional layered video encoding methods, such as the MPEG-4 Fine Grained Scalability profile (MPEG-FGS), are based on the progressive truncation of DCT or wavelet coefficients [7]. There is a tradeoff between the bandwidth and power consumption requirements of each layer and the visual quality of the resulting video [7]. Since MPEG-FGS is based on the spectral characteristics of low-level pixel data, the quality of the lower layer videos may be unsuitable for high-level vision applications in the face of resource constraints. To enable high-level vision applications, it is imperative that the video streams corresponding to the lower encoding layers contain enough high-level information while simultaneously satisfying the resource constraints imposed by the mobile network environment.

2. OVERVIEW OF PROPOSED SCHEME

In this paper, we present the design and implementation of a novel *Hybrid Layered Video* (HLV) encoding scheme where the constituent video layers are a combination of standard

MPEG-based video encoding and a generative sketch-based video representation. The input video stream is divided into two components: a *sketch* component $\mathbf{V_{SKETCH}}$ and a *texture* component $\mathbf{V_{TEXTURE}}$. The $\mathbf{V_{SKETCH}}$ component is a *Generative Sketch-based Video* (GSV) representation, where the outlines of the objects of the video are represented by curves [5]. The evolution of these curves (termed as *pixel-threads*), across the video frames is explicitly modeled in order to reduce temporal redundancy. Since the proposed GSV representation uses a sparse set of parametric curves, instead of necessarily closed contours, to represent the outlines of objects in the video frames, the number of graphical objects that one needs to overlay is small. The $\mathbf{V_{TEXTURE}}$ component is represented by three layers; a base layer video V_{base}, an intermediate mid-layer video V_{mid}, and the original video V_{org}. V_{base} represents a very low bitrate video with very low visual quality whereas V_{org} denotes the original video. The visual quality of V_{base} is improved significantly by augmenting it with the object outlines resulting from the GSV representation mentioned above. The visual quality of V_{mid} is higher than that of V_{base}, but lower than that of V_{org}. The quality of V_{mid} is further enhanced via high-level object-based re-rendering of the video at multiple scales of resolution. The result is termed as a *Features, Motion and Object-Enhanced Multi-Resolution* (FMOE-MR) video [4] wherein semantically relevant portions of the frames in V_{mid} are highlighted by selectively rendering them at higher resolution. A schematic diagram depicting the proposed Hybrid Layered Video (HLV) representation is given in Figure 1.

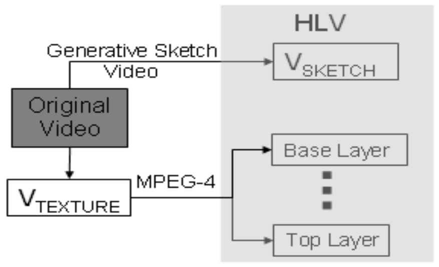


Fig 1: The Hybrid Layered Video (HLV) Scheme.

In the following sections, we elaborate upon the $\mathbf{V_{SKETCH}}$ and $\mathbf{V_{TEXTURE}}$ components of the proposed HLV representation. This is followed by a description of how to combine the various video layers comprising the aforementioned components to result in a resource-scalable video representation wherein the various video layers have different resource consumption characteristics. The contents of an HLV-encoded video can be decoded, analyzed and rendered at different levels of resource (i.e., power and bandwidth) consumption on the mobile device depending on the combination of layers used. We also show how the proposed HLV representation can be effectively used for some important vision tasks, i.e., background subtraction, face detection, face recognition and face tracking in a resource-constrained mobile Internet environment.

3. CREATING VIDEO COMPONENT $\mathbf{V_{SKETCH}}$

The video component $\mathbf{V_{SKETCH}}$ essentially represents the outlines or *sketches* of the objects in the video by a sparse set of parametric curves resulting in a *Generative Sketch-based Video* (GSV) representation. The video is first divided into a series of *Groups of Pictures* (GOPs), in a manner similar to standard MPEG video encoding. Each GOP consists of *N* frames (typically, $N = 15$ for standard MPEG/H.264 encoding) where each frame is encoded as follows:

1. The object outlines are extracted in each of the *N* frames. These outlines are represented by a sparse set of curves.
2. The curves in each of the *N* frames are converted to a suitable parametric representation.

3. Temporal consistency is used to remove spurious curves which occur intermittently in consecutive frames to remove an undesired flickering effect.
4. Finally, the parametric curves in the *N* frames of the GOP are encoded in a compact manner. The first frame of the GOP enumerates the curve parameters in a manner that is independent of their encoding, analogous to the I-frame in MPEG H.264 standard. The remaining *N*-1 frames in the GOP are encoded using motion information derived from previous frames, in a manner analogous to the P-frames in the MPEG H.264 standard.

Motion vectors are used in GSV encoding to reduce temporal redundancy. The error vector has the same form as the encoded representation of the moving object(s) in the video. The parametric curves used to represent the object outlines in each frame are termed as *pixel-threads*. A *pixel-thread* is derived from a polyline P:[0, N], which is a continuous and piecewise linear curve made of N connected straight-line segments. A polyline can be parameterized using a parameter $a \in \mathbf{R}$ (set of real numbers) such that $P(a)$ refers to a specific position on the polyline, with $P(0)$ referring to the first vertex of the polyline and $P(N)$ referring to its last vertex. Note that the *pixel-threads* contain information only about the vertices (or break points) of the polyline. Note that these break points can be joined by straight line segments (as in the case of a polyline), or by more complex spline-based functions to create smooth curves.

The *pixel-threads* essentially depict the outlines of objects in the underlying video stream. Each video frame is associated with its own collection of *pixel-threads* termed as a *Pixel-thread-Pool*. Thus, successive video frames are associated with successive *Pixel-thread-Pools*. Due to the temporal nature of the video, the *pixel-threads* and *Pixel-thread-Pool* are modeled as dynamic entities that evolve over time to generate the outlines of the moving objects in the video. The dynamic nature of the *pixel-threads* is modeled by the processes of *birth* and *evolution* of *pixel-threads* over time.

3.1. Birth of pixel-threads

For a given video frame, the *Pixel-thread-Pool* is created by first generating (i.e., sketching) the outlines of the objects in the video frame, and then representing these outlines parametrically in the form of *pixel-threads*. The edge pixels in a video frame are extracted using the Canny edge detector [3] and grouped to form one-pixel-wide edge segments or *edgels*. A curved edge segment is represented by a series of break points along the curve, determined using the algorithm of Rosin and West [10]. The break points are essentially points of significance (i.e., corners and high-curvature points) along the curve and are represented efficiently as a chain-coded vector. For each approximated curve i, one of the end points (first or last break point) is represented using absolute coordinates $\{x_0, y_0\}$ whereas the p^{th} break point, where $p > 0$, is represented by coordinates relative to those of the previous break point; i.e., $\{\delta x_p, \delta y_p\}$ where $\delta x_p = x_p - x_{p-1}$ and $\delta y_p = y_p - y_{p-1}$. The resulting chain-coded vectors constitute the *pixel-threads*. The fitting of straight line segments or cubic splines between the break points results in the rendering of an *approximate* version of the original curve.

3.2. Evolution of a pixel-thread

The temporal redundancy between *pixel-threads* in successive frames is exploited to *evolve* some of the *pixel-threads* in the current frame to constitute the *Pixel-thread-Pool* for the next frame. Motion modeling of *pixel-threads* is used to reduce the amount of information required to render the set of *pixel-threads* in successive frames, resulting in a compact representation of the dynamic *pixel-threads*. The evolution of *pixel-threads* between two successive *Pixel-thread-Pools,* say TP_1 *and* TP_2, involves two steps; (a) establishing the *pixel-thread* correspondence between the two *Pixel-thread-Pools*, and (b) estimating the motion parameters.

In order to determine the correspondence between *pixel-threads* in TP_1 and TP_2 one needs to determine for each *pixel-thread* in TP_1 its counterpart in TP_2. First, we need to predict a position to which a *pixel-thread* $\mathbf{T_1}$ in TP_1 is *expected* to move in the next frame. The predicted *pixel-thread*, $\mathbf{T'}$, is determined using a suitable optical flow function *OpF*(), such that

$$\mathbf{T'} = OpF(\mathbf{T_1}) \quad (1)$$

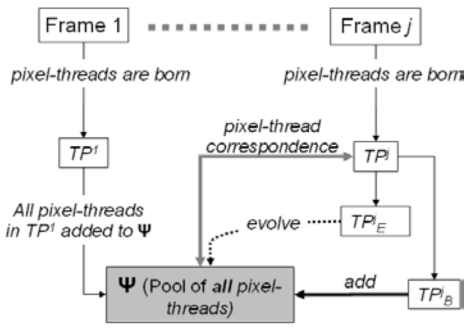

Frame 1
Frame j
pixel-threads are born
pixel-threads are born
pixel-thread correspondence
TP1
TPj
All pixel-threads in TP1 added to Ψ
evolve
TPjE
Ψ (Pool of all pixel-threads)
add
TPjB

Fig. 2. Establishing *pixel-thread* correspondence for a frame *j* and the current *All-Threads-Pool* Ψ.

The function $OpF()$ computes the coordinates of the break points of the *pixel-thread* $\mathbf{T'}$ in TP_2, given the coordinates of the break points of *pixel-thread* $\mathbf{T_1}$ in TP_1. The function $OpF()$ implements a sparse iterative version of the Lucas-Kanade optical flow algorithm designed for pyramidal (or multiscale) computation [2]. We hypothesize that if *pixel-thread* $\mathbf{T_1}$ in *Pixel-thread-Pool* TP_1 does indeed evolve to a corresponding *pixel-thread* $\mathbf{T_2}$ in TP_2, then $\mathbf{T'}$ and $\mathbf{T_2}$ should resemble each other in terms of shape and size. The correspondence between pixel threads $\mathbf{T'} = OpF(\mathbf{T_1})$ and $\mathbf{T_2}$ is determined using the *Hausdorff distance* [1] and is given by

$$\mathbf{T_2} = argmin\{ \delta_H(OpF(\mathbf{T_1}), \mathbf{T}): \mathbf{T} \in TP_2\} \quad (2)$$

We define a threshold $\varepsilon > 0$, such that if $\delta_H(OpF(\mathbf{T_1}), \mathbf{T_2}) < \varepsilon$, then we consider $\mathbf{T_2}$ to have evolved from $\mathbf{T_1}$; otherwise, $\mathbf{T_1}$ is deemed to have become *dormant*, and $\mathbf{T_2}$ in TP_2 is deemed to have been *born* in TP_2. The *pixel-threads* in TP_2 can be categorized as belonging to two mutually exclusive sets, TP^{Evolve} and TP^{born}. TP^{Evolve} is the set of all *pixel-threads* in TP_2 which are evolved from some *pixel-thread* in TP_1, and TP^{born} is the set of *pixel-threads* in TP_2 which are deemed to have been born in TP_2. Figure 2 provides a schematic description of this process.

To ensure compactness of the final representation, we assume that a linear transformation $L_{\mathbf{T}}$, specified by the translational parameters $\{t_x, t_y\}$, can be used for motion estimation. The *pixel-thread*, $\mathbf{T_2}^{\mathbf{estimated}}$ is computed from $\mathbf{T_1}$ by using a mapping function $L_{\mathbf{T2}}$, such that

$$\mathbf{T_2}^{\mathbf{estimated}} = L_{\mathbf{T2}}(\mathbf{T_1}) \quad (3)$$

The linear transformation coordinates in $L_{\mathbf{T2}}$ can be determined by computing the mean of the displacements of each break point, where the displacement of each break point is computed using the function $OpF()$ (eqn. (1)). Since $\mathbf{T_2}^{\mathbf{estimated}}$ may not align exactly point-by-point with $\mathbf{T_2}$, it is necessary to compute and encode the error $\mathbf{\Delta T_2}$ between $\mathbf{T_2}^{\mathbf{estimated}}$ and $\mathbf{T_2}$. Thus, the motion model required to evolve *pixel-thread* $\mathbf{T_1}$ into *pixel-thread* $\mathbf{T_2}$, is given by

$$\Theta_{\mathbf{T1}}(\mathbf{T_2}) = \{ t_x , t_y , \mathbf{\Delta T_2}\} \quad (4)$$

3.3. Evolution of a pixel-thread-pool

Given a video sequence of N frames, and the current frame j, let $\mathbf{\Psi}$ be the pool of all the *pixel-threads* which have been *born* or *evolved* thus far in frames 1 through j -1. $\mathbf{\Psi}$ may contain some dormant *pixel-threads*, i.e., ones which were not used to sketch a curve in the previous frame, j -1. The *pixel-threads* in $\mathbf{\Psi}$ belong to one of two disjoint subsets; $\mathbf{\Psi}_{\text{dormant}}$ or $\mathbf{\Psi}_{\text{active}}$ where $\mathbf{\Psi} = \mathbf{\Psi}_{\text{dormant}} \cup \mathbf{\Psi}_{\text{active}}$.

For the current frame j, the *pixel-threads* corresponding to frame j are first determined using the techniques discussed in Section 3.1. These recently acquired *pixel-threads* corresponding to frame j are grouped together in *Pixel-thread-Pool* TP_j. Assume that TP_j has n_j *pixel-threads*

$\{\mathbf{T}^1_\mathbf{j}, \mathbf{T}^2_\mathbf{j} \dots \mathbf{T}^{\mathbf{nj}}_\mathbf{j}\}$. Next, the correspondence between the *pixel-threads* in the set $\mathbf{\Psi}$ and the n_j *pixel-threads* in TP_j, is determined using the methods mentioned in Section 3.2. Note that, during the correspondence determination procedure, the dormant *pixel-threads* in $\mathbf{\Psi}$ are also considered.

Let $TP_j^{wereEvolved}$ be the subset of the *pixel-threads* in TP_j which have evolved from $\mathbf{\Psi}$. Let $\mathbf{\Psi}_{TPj}$ be the subset of *pixel-threads* in $\mathbf{\Psi}$ which evolve to a corresponding *pixel-thread* in $TP_j^{wereEvolved}$. The correspondence between $\mathbf{\Psi}_{\mathrm{TPj}}$ and $TP_j^{\mathrm{wereEvolved}}$ is determined in a manner such that a *pixel-thread* $\mathbf{T}^\mathbf{i}_\mathbf{j}$ belonging to $TP_j^{wereEvolved}$ and the corresponding *pixel-thread* in $\mathbf{\Psi}_{TPj}$ from which it has evolved are both assigned the same index *i*. Now, *pixel-threads* in $\mathbf{\Psi}_{\mathbf{TPj}}$ can be evolved to corresponding *pixel-threads* in $TP_j^{wereEvolved}$ via a set of motion models $\mathbf{\Theta}_{\mathbf{\Psi TPj}}$ (eqn. (4)). Since the remaining *pixel-threads* in TP^i_j cannot be evolved from any existing *pixel-thread* in $\mathbf{\Psi}$**,** these *pixel-threads* are considered to belong to the set TP_j^{born}; where $TP_j^{born} = TP_j$ - $TP_j^{\mathrm{wereEvolved}}$. Next, the set $\mathbf{\Psi}$ is updated in the following manner:
(a) *Pixel-threads* in $\mathbf{\Psi}_{\mathrm{TPj}}$ are *evolved* to corresponding *pixel-threads* in $TP_j^{\mathrm{wereEvolved}}$, using motion model parameters given by $\mathbf{\Theta}_{\mathbf{\Psi TPj}}$. The new *pixel-threads* in TP_j^{born} are included in $\mathbf{\Psi}$**.**
(b) The new set of active *pixel-threads* is given by $\mathbf{\Psi}_{\mathbf{active}} = \mathbf{\Psi} \cap TP_j$. These *pixel-threads* are used to generate the sketch-based representation of the new video frame. Naturally, the *pixel-threads* in this updated set $\mathbf{\Psi}$, that have no counterparts in TP_j, are deemed *dormant*; i.e.,

$$\mathbf{\Psi}_{\mathbf{dormant}} = \mathbf{\Psi} - \mathbf{\Psi}_{\mathbf{active}} \tag{5}$$

The data corresponding to frame *j* required to sketch the *j*th video frame are given by the motion model parameters denoted by $\mathbf{\Theta}_{\mathbf{\Psi TPj}}$. The newly born *pixel-threads* are included in TP_j^{born}. Thus, the entire process of evolution of all the *pixel-threads* across all the *N* frames of the video can be effectively represented as a GSV, given by

$$\mathrm{GSV} = \{< \mathbf{\Theta}_{\mathbf{\Psi TP1}}, TP_1^{born}>,.., < \mathbf{\Theta}_{\mathbf{\Psi TPN}}, TP_N^{born}>\} \tag{6}$$

The interested reader is referred to [5] for further details on GSV encoding.

4. ENCODING TEXTURE COMPONENT $\mathbf{V_{TEXTURE}}$

The video texture component $\mathbf{V_{TEXTURE}}$, consists of three sub-components termed as V_{org}, V_{mid} and V_{base} as described earlier. Each of the video sub-components is encoded using the MPEG H.264 standard. V_{org} is the original video which is encoded efficiently using a public domain state-of-the-art MPEG H.264 encoder [15]. The video layer V_{mid} represents an intermediate-level video which has a more compact representation than V_{org} albeit at the cost of lower visual quality. A lower-size video file leads to reduction in overall resource consumption during the decoding process. V_{mid} is generated using a novel multi-resolution video encoding technique termed as *Features, Motion and Object-Enhanced Multi-Resolution* (FMOE-MR) video encoding [4]. Instances of Features, Motion and Objects (FMOs) are detected in the video sequence using computer vision algorithms. A corresponding mask (FMO-Mask) is created to mark the regions corresponding to the presence of the FMOs. The regions containing FMOs are deemed to be of interest and displayed at high resolution whereas the rest of image frame is rendered at low resolution.

The FMO-Mask is essentially a combination of three individual masks; the Feature-Mask (F-Mask), Motion-Mask (M-Mask) and the Object-Mask (O-Mask). The F-Mask captures the important low-level spatial features in the form of edges in the video frame. We use the Canny edge detector [3] to detect and localize the edges in a video frame. The F-mask is essentially a weighting matrix created by assigning a value of 1 to regions in and around the edges, and a value of 0 elsewhere. An M-mask is created by identifying, via a process of background subtraction, the regions within the video frames that contain moving objects [8]. Further, certain foreground objects, such as human faces, are deemed to be special interest. In our current implementation, faces in a video sequence are detected using the algorithms described in [13] and tracked using the algorithm described in [9] to automatically create an O-mask based on human faces.

The three masks are superimposed to generate the final FMO mask. The original frame is re-encoded as a multi-resolution (MR) representation, guided by the FMO-mask such that regions corresponding to mask values close to 1 are rendered at higher resolution than regions corresponding to mask values close to 0. The original video frame V_O is used to render two video frames, V_H and V_L, representing a high resolution rendering and a low resolution rendering respectively. V_L and V_H are obtained by convolving V_O with Gaussian filters characterized by the smoothing parameters σ_L and σ_H respectively where $\sigma_L > \sigma_H$. If the FMO mask is represented as a matrix **W** whose elements lie in the range [0, 1], then the MR frame V_{MR} is obtained via a linear combination of the two frames V_H and V_L as follows:

$$V_{MR} = \mathbf{W}\bullet V_H + (\mathbf{I} - \mathbf{W})\bullet V_L \tag{7}$$

where **I** is a matrix all of whose elements are 1. Empirical observations have revealed that σ_L = 9 or 11, and σ_H = 3, can be used, in most cases, to yield videos of reasonable visual quality with significantly smaller file sizes than the original video.

The base video layer V_{base} is generated by first blurring each frame of the video using a Gaussian filter with smoothing parameter σ_{base} *prior* to MPEG H.264 encoding. Note that the Gaussian smoothing is performed uniformly over the entire video frame. V_{base} essentially serves to provide approximate color or texture information for the GSV representation described previously. We have observed empirically that values of σ_{base} in the range [19, 25] can be used to generate the base video layer V_{base}, resulting in a very small file size albeit at the cost of low resolution and low visual quality. However, approximate color or texture information is still retained in the video layer V_{base}, to the point that the visual quality of the resulting video improves significantly when the object outlines from the GSV representation are superimposed on the video layer V_{base}.

5. COMBINING V_{SKETCH} AND $V_{TEXTURE}$: VIDEO STATES

We assume that the $\mathbf{V_{TEXTURE}}$ component has *L* levels of resolution. In the current implementation, *L* = 4 which includes the three layers V_{org}, V_{mid} and V_{base} in decreasing order of visual quality and level 0 which denotes GSV-encoded video with complete absence of texture information. Let $\mathbf{V^j_{TEXTURE}}$ ($0 \leq j \leq L$-1) correspond to the MPEG H.264-based encoding of the video where $\mathbf{V^0_{TEXTURE}}$ denotes the complete absence of texture information, $\mathbf{V^1_{TEXTURE}}$ denotes the video layer of the least resolution (and correspondingly small video file size), and $\mathbf{V^{L-1}_{TEXTURE}}$ denotes the video layer of the highest resolution (i.e., the original video encoded using the MPEG H.264 standard with no deliberately induced loss in visual quality). Let the state of the HLV-encoded video be depicted as

$$\mathbf{\Gamma}(\textit{texture-level}, \textit{sketch-level}) = (\mathbf{V^{texture\text{-}level}_{TEXTURE}}, \mathbf{V^{sketch\text{-}level}_{SKETCH}}) \tag{8}$$

such that $0 \leq$ *texture-level* $\leq L–1$, and *sketch-level* Є {*no-sketch*, *polyline-sketch*, *spline-sketch*}. The above state-based representation allows for various resolutions of texture with superimposition of sketch-based representations of varying degrees of complexity. Under the above formalism, **Γ**(*L–1*, *no-sketch*) represents the original (i.e., best quality) video, and **Γ**(0, *polyline-sketch*) represents the video that contains no texture, but only the object outlines represented by polylines (presumably, the lowest quality video).

Furthermore, the states in the above representation are linearly ordered such that a “higher” video state is deemed to consume more resources (battery power and bandwidth) than a “lower” video state. Let *Resources*(**X**, *t*) be the resource (battery time, bandwidth, etc.) estimate provided by the operating system on the playback device *t* seconds after the video playback has been initiated, where **X** denotes the state of the video during playback. Let $\mathbf{\Gamma} = \{\mathbf{\Gamma_1}, \ldots, \mathbf{\Gamma_S}\}$ be the *S* distinct video states. We define a relation $\leq_p$ such that $\mathbf{\Gamma}_i \leq_p \mathbf{\Gamma}_j$ implies that

$$\textit{Resources}(\mathbf{\Gamma}_i, t) \leq \textit{Resources}(\mathbf{\Gamma}_j, t), \quad t > 0 \tag{9}$$

The value of *Resources*($\mathbf{\Gamma}_{current\text{-}state}$, *t*) is estimated using a simple operating systems call, which predicts the remaining battery time or the available bandwidth based on the current load.

6. HLV FOR MOBILE INTERNET VISION APPLICATIONS

The proposed HLV encoding technique is evaluated in the context of some important computer vision applications in a resource-constrained mobile Internet environment, i.e., background subtraction, face detection, face recognition and face tracking. As an objective comparison of HLV-encoded video quality, we used the videos corresponding to video states of the *Train Station* video, $\mathbf{\Gamma_3^{train}} = \mathbf{\Gamma}(V_{base}, \textit{polyline-sketch})$, and $\mathbf{\Gamma_6^{train}} = \mathbf{\Gamma}(V_{org}, \textit{null})$, to perform background subtraction. Note that $\mathbf{\Gamma_6^{train}} = (V_{org}, \textit{null})$ corresponds to the video with the highest visual quality. The background, once determined, is subtracted from each frame to extract the foreground, or moving regions within the video frames [8]. We hypothesize that the video states $\mathbf{\Gamma_3^{train}}$ and $\mathbf{\Gamma_6^{train}}$ yield videos of comparable quality if both videos result in similar foreground regions upon background subtraction. Figure 3 shows the resulting foreground masks after background subtraction has been performed on a $\mathbf{\Gamma_6^{train}}$ video frame and the corresponding $\mathbf{\Gamma_3^{train}}$ frame. As is evident from Figure 3, both videos were observed to yield similar foreground masks with an 85% overlap between them.

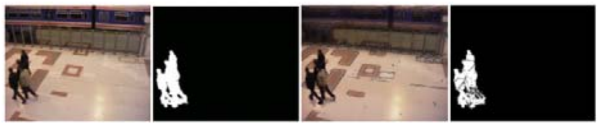
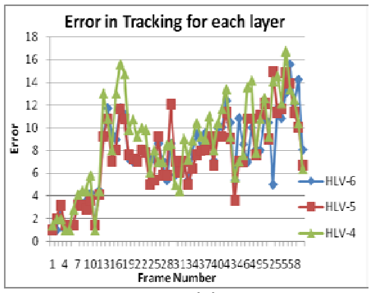
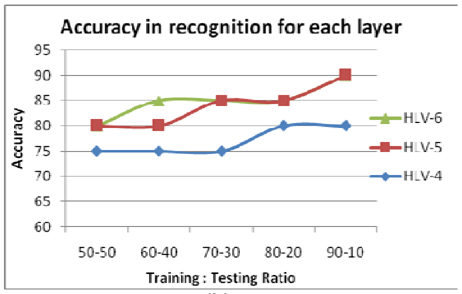


Fig. 3 (a) Video frame from Γ_6^{train} (b) Result of background subtraction on Γ_6^{train} (c) Video frame from Γ_3^{train} (d) Result of background subtraction on Γ_3^{train}

The proposed HLV encoding scheme was also evaluated in the context of face detection and tracking. Four videos of different people were taken and three different HLV layers were used, i.e., $\mathbf{\Gamma_6^{face}} = (V_{org}, \textit{null})$, $\mathbf{\Gamma_5^{face}} = (V_{mid}, \textit{spline-sketch})$ and $\mathbf{\Gamma_4^{face}} = (V_{mid}, \textit{polyline-sketch})$. Face detection was performed using a combination of skin detection [14] and the AdaBoost detection technique [13]. The detected faces were tracked using the Kernel-based tracker [6]. Figure 4(a) plots the face tracking error for the different HLV layers on the *Closeup Face* video. The error in the case of $\mathbf{\Gamma_6^{face}}$ (HLV-6) is indicative of the error introduced by the face detection and tracking algorithms and the basic MPEG encoding scheme. As can be seen in Figure 4(a), very little additional error is introduced by the proposed HLV encoding scheme. The HLV scheme was also evaluated in the context of face recognition using the *Eigenfaces* approach [12]. Figure 4(b) depicts the results of comparison in face recognition accuracy for each layer where the accuracy is measured for varying ratios of training images to testing images. All the training images were from $\mathbf{\Gamma_6^{face}}$ (HLV-6) whereas the testing for recognition was performed on all three layers $\mathbf{\Gamma_6^{face}}$, $\mathbf{\Gamma_5^{face}}$ and $\mathbf{\Gamma_4^{face}}$. In Figure 4(b) the face recognition accuracy is observed to decrease only slightly for $\mathbf{\Gamma_4^{face}}$ (HLV-4) whereas there is no significant difference in accuracy between $\mathbf{\Gamma_5^{face}}$ (HLV-5) and $\mathbf{\Gamma_6^{face}}$ (HLV-6).





(a) **(b)**

Fig. 4: Face tracking error and face recognition accuracy for different video layers

7. CONCLUSIONS

In this paper we have proposed and implemented a *Hybrid Layered Video* (HLV) encoding scheme, which comprises of content-aware, multi-layer encoding of texture and a generative sketch-based representation of the object outlines. Different combinations of the texture- and sketch-based representations result in distinct video states, each with a characteristic bandwidth and power consumption profile. The proposed HLV encoding scheme is shown to be well suited for computer vision applications such as background subtraction, face detection, face recognition and face tracking in a mobile Internet environment that is constrained by resources, in particular, the available bandwidth and available power on the mobile client device. Future work will consider more advanced computer vision-based applications in a mobile Internet environment such as intelligent video surveillance, distributed gaming and human activity understanding.

REFERENCES

[1] Atallah, M. J., Linear time algorithm for the Hausdorff distance between convex polygons, *Info. Processing Letters*, vol. 17, no. 4, pp. 207–209, 1983

[2] Bouguet, J.Y., *Pyramidal Implementation of the Lucas Kanade Feature Tracker*, Intel Corporation, Microprocessor Research Labs; included in the distribution of OpenCV.

[3] Canny, J., A computational approach to edge detection, *IEEE Trans. Pattern Analysis and Machine Intelligence,* Vol. 8, no. 6, pp. 679 –698, 1986

[4] Chattopadhyay, S., Luo, X., Bhandarkar, S. M. and Li, K., FMOE-MR: Content-driven Multi-resolution MPEG-4 Fine-grained Scalable Layered Video Encoding, *Proc. ACM Conf. Multimedia Computing and Networking (MMCN)*, Vol. 6504, San Jose, CA, January 2007, pp. 650404.1–650404.11.

[5] Chattopadhyay, S., Bhandarkar, S.M. and Li, K., Ligne-Claire Video Encoding for Power Constrained Mobile Environments, *Proc. ACM Conf. Multimedia*, Augsburg, Germany, Sept. 2007, pp. 1036–1045.

[6] Comaniciu, D., Ramesh, V., Meer, P., Kernel-based object tracking, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 25, No. 5, May 2003, pp. 564–577.

[7] Dai, M., Loguinov, D., Analysis and Modeling of MPEG-4 and H.264 Multi-Layer Video Traffic, *Proc. IEEE INFOCOM*, 2005.

[8] Luo, X., Bhandarkar, S.M., Hua, W., Gu, H. and Guo, C., Nonparametric Background Modeling Using the CONDENSATION Algorithm, *Proc. IEEE Intl. Conf. Advanced Video and Signal-based Surveillance (AVSS 2006)*, Sydney, Australia, pp. 13–18, November 2006.

[9] Luo, X. and Bhandarkar, S.M., Tracking of Multiple Objects Using Optical Flow-based Multiscale Elastic Matching, *Springer LNCS– ICCV Workshop on Dynamical Vision,* R.Vidal, A. Heyden and Y. Ma (eds.), Vol. 4358, pp. 203–217, 2007.

[10] Rosin, P.L. and West, G.A.W., Non-Parametric Segmentation of Curves into Various Representations, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 12, pp. 1140–1153, 1995.

[11] Sikora, T., Trends and perspectives in image and video coding, *Proc. IEEE*, vol. 93, no. 1, pp. 6–17, Jan. 2005.

[12] Turk, M. and Pentland, A., Eigenfaces for Recognition, *Journal of Cognitive Neuroscience*, vol. 3, no. 1, 1991, pp. 71–86.

[13] Viola, P. and Jones, M., Rapid Object Detection using a Boosted Cascade of Simple Features, *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, vol. 1, pp. 511–518, 2001.

[14] Zarit, B.D., Super, B.J. and Quek, F.K.H., Comparison of Five Color Models in Skin Pixel Classification, *Intl. Wkshp. Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, Washington DC, USA, September 1999, pp. 58–63.

[15] http://www.squared5.com.