Scene-Aware Video Modeling and Compression

Georgios Georgiadis, Stefano Soatto
Department of Computer Science,
University of California, Los Angeles,
Los Angeles, CA 90095

Abstract

We describe a video compression methodology that exploits the structure of the data formation process, whereby the "source" is the scene, and the "channel" includes scaling and occlusion phenomena that are critical elements of image formation. Thus our scheme involves occlusion detection, optical flow computation, texture/structure partition, and a notion of proper sampling. We show preliminary but promising results that exceed baseline compression performance, albeit at an increased computational cost.

1 Introduction

Video exhibits considerable redundancy in both space and time, and generic priors such as sparsity and regularity have been exploited with some success, but diminishing return. Pressure from the applications[1] calls for new approaches to break the "video compression wall." To do so, encoders ought to explicitly model and exploit the phenomenology of the data-formation process, rather than treating video as a generic bit-stream. In other words, rather than encoding the video, one should encode some representation of the *scene* that generated it. The representation (encoding) depends on the task: The best encoding for human fruition (e.g., compression of movies) is different from the best for surveillance, robotic navigation, or content-based video retrieval [18]. In this paper we focus on human fruition, like classical video compression, where performance is ideally computed with some perceptual score [21], but often in practice with the (perceptually irrelevant) RMS or PSNR scores.

Two phenomena of the optical data-formation process are critical to compression: *occlusion* and *scaling*. Occlusion implies that there are portions of the scene, or the light source, that are visible in one image but not another, and therefore must be encoded anew. *Scaling* implies that there is no "Nyquist frequency" in the scene, as moving closer to objects causes more and more structure to appear. This causes phase transitions (singular perturbations) in the representation, and forces the continuous limit to be part of the analysis.

Occlusion phenomena (occluding boundaries), together with material transitions (material boundaries) and illumination boundaries (cast shadows), are indirectly accounted for in existing video compression schemes: They all yield highly kurtotic gradient distributions that are captured by sparsity-inducing priors. However, of the three, only occlusion phenomena yield an "information increment" (innovation), where future data cannot be explained with past data in the presence of motion. Therefore, it is important to be able to *detect occlusions*, which we do in Sect. 2.1. This also relates to optical flow which addresses temporal redundancy. Scaling and quantization challenge the traditional sampling paradigm, and calls for a new notion of *"proper sampling"*, that we introduce in Sect. 2.2. This relates also to the notion of "texture" and addresses spatial redundancy.

In Sect. 3 we report some experimental results. Although our focus is on outlining a modeling framework, we show that even a non-optimized implementation of our pipeline beats baseline performance using the PSNR score. In Sect. 4 we point at limitations and challenges in our approach.

[1] CISCO projects that, by 2014, 90% of Internet traffic, and 64% of wireless traffic will be video. In 2010, video surpassed peer-to-peer traffic on the Internet [5].

1.1 Related work

At the level of generality adopted thus far, this work relates to the entirety of computer vision. However, our goal is not to infer geometric, photometric, or dynamic properties of the scene from video. Instead, we aim to infer just enough of the phenomenology of the scene to be useful for compression, specifically occlusion and scaling. Even so, relevant prior work goes beyond what we can cover in conference submission. We refer the reader to representative work on *optical flow* [10, 4, 15], *occlusion detection* [2, 3], *tracking* e.g. KLT and particle filter [19, 22, 17], *segmentation* e.g. gPb, mean shift [1, 6, 8], *texture analysis and modeling* [7, 14, 12], including texture/structure partitioning [9, 24]. Finally, all video compression standards that offer end-to-end solutions are related to our work e.g. H.262, H.263, H.264 [11].

1.2 Formalization

A (grayscale) video $I : D \subset \mathbb{Z}^2 \times \mathbb{N} \to \mathbb{N}; (x, t) \mapsto I(x, t)$ is a spatially and temporally quantized version of a hidden signal that we call *radiance* ρ. If we consider, for the sake of example, a planar scene parallel to the image, then a motion orthogonal to it induces a re-scaling of the image domain: $I(x,t) = \int \delta_\epsilon(x - \tilde{x})\rho(s(t)\tilde{x})s(t)d\tilde{x}$, where δ_ϵ is the quantization kernel (for instance an indicator function supported on the pixel). This shows that we cannot just discretize the radiance by defining ρ on a discrete domain, because by moving sufficiently far from the scene we can make $s(t)$ sufficiently small, and therefore the sampling of $s(t)\tilde{x}$ sufficiently high-rate. Instead, we must consider it as a (continuous) function from surfaces embedded in three-dimensional space to the positive reals. If the scene is not a fronto-parallel plane, but has arbitrary shape, it induces a deformation of the image domain that is an epipolar transformation [16], which can be shown to admit as closure the entire group of planar diffeomorphisms [20], so $I(x,t) = \int \delta_\epsilon(x - \tilde{x})\rho(w(\tilde{x}, t))|J_w(x,t)|d\tilde{x}$. Therefore, we model the radiance as a function $\rho : \mathbb{R}^2 \to \mathbb{R}^+$, and scene or viewer's motion as a domain diffeomorphism $w : \mathbb{R}^2 \to \mathbb{R}^2$. If we think of the radiance as the "representation" that we wish to infer, encode, store, and reconstruct, in addition to unknown domain deformations w, we have unknown transformations of the range of the data. The simplest are contrast transformations $\kappa : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^+; (I, t) \mapsto \kappa(I, t) \doteq \kappa(t) \circ I$, that are continuous monotonic transformations of the range space of the image. For simplicity, we will include the quantization kernel in the contrast transformation, and therefore consider models of the form:

$$I(x,t) = \kappa(t) \circ \rho \circ w(x,t) + n(x,t) \tag{1}$$

where n is the residual that lumps all unmodeled phenomena (mutual-illumination, inter-reflection, transparency, translucency, vignetting, etc.) as well as sensor noise and other more standard phenomena. The model (1) describes the temporal evolution of the data *in the co-visible portions of the scene*. These are portions of the scene that are visible in more than one image at a time. In the complement, i.e. the (multiply-connected) *occluded region* $\Omega(t) \subset D$, the image at a given time t can in principle take any value compatible with the marginal statistics of natural images. So, the residual $n(x,t)$ can be assumed to be small, spatially and temporally white and homoscedastic in the co-visible region $D \backslash \Omega(t)$, but it could be arbitrarily large in the occluded region $\Omega(t) \subset D$. However, such regions are typically sparse, in the sense that the occluded area $|\Omega(t)|$ is small relative to D.

Given a collection of images, $\{I(x,t)\}_{x \in D, t \in [0,\, T]}$, our goal is to infer a model $\Sigma = \{\hat{\kappa}, \hat{\rho}, \hat{w}\}$, that is as "simple" as possible, and that yields as "small" a residual $\sum_{t,x} |n(x,t)|$. This can be framed as an optimization problem, where we must declare what we mean by "simple", what we mean by "small", and define the class of models Σ. Unlike other tasks, such as decision and control, in reconstruction we do not care about model *identifiability*, so long as we can find *any* model that describes the data sufficiently well. In Sect. 2 we describe these ingredients and the ensuing algorithmic instantiations.

2 Encoder

2.1 Occlusions, optical flow, and temporal redundancy

Occlusion detection is a binary classification problem where each pixel is labeled as either *co-visible* ($x \in D \backslash \Omega(t)$) or occluded ($x \in \Omega(t)$). Note that this does not depend on how many "objects" there are in the scene, or occlusion layers: A point on the scene is either visible or not. In order to test for occlusion, we must invalidate the co-visibility hypothesis, that entails searching for the domain diffeomorphisms that maps one image onto another. If we restrict our attention to two temporally adjacent images, this problem is known as *optical flow*: $\rho(x) = I(x, t-1)$ and the residual $n(x,t)$ in (1) is the sum of two components: $e_2(x,t)$ that is *dense*, (defined for all $x \in D$) but statistically "simple" (spatially and temporally uncorrelated, isotropic, homoscedastic etc.) with a small variance. The other, $e_1(x,t)$, can be arbitrarily large, but it is only defined on the occluded domain $x \in \Omega(t)$. Thus the problem is to simultaneously infer optical flow $w(x,t)$ as well as the occluded region $\Omega(t)$, that is time-varying, multiply-connected, and in general can be rather complex (think of the occlusions induced by motion in front of a barren tree). Ayvaci et al. [2] have framed this problem as a convex variational optimization, and solved it with numerically efficient Augmented Lagrangian methods. We will therefore take this stage as a building block and assume that, at each time t, we have an estimate of the characteristic function of the occluded region, $\hat{\Omega}(t) \subset D$, as well motion field $\hat{w}(x,t), x \in D \backslash \hat{\Omega}(t)$ that solve

$$\hat{w}(x,t), \hat{\Omega}(t) = \arg\min_{w,\Omega} \|e_2\|_{\ell^2(D)} + \lambda \|e_1\|_{\ell^1(D\backslash\Omega)} \tag{2}$$
$$\text{s. t.} \quad I(x,t) = I(w(x,t), t-1) + e_1(x,t) + e_2(x,t)$$

plus some regularization of the motion field w; λ is a tradeoff factor (multiplier). Note that we have not included a range transformation $\kappa(t)$, since we do not expect significant contrast changes between adjacent images, and the small residual can be lumped into e_2. In Fig. 1, we show results on a few test sequences based on the algorithm used. While optical flow tells us what *regions* of the images are occluded, and therefore regions where we cannot exploit temporal consistency, not every *pixel* in the co-visible region has a unique correspondent. Those that do can be encoded once and then "tracked" through subsequent frames. Those that do not can be *sampled* or *filled in* independently in successive frames, as we discuss in Sect. 2.2.

Encoding the motion field $w(x,t)$ is, in principle, more than twice as costly as encoding the image, since $I(x,t) \in \mathbb{N}$, whereas $w(x,t) \in \mathbb{R}^2$. So, to achieve compression one has to exploit the spatial regularity of the motion field, that is assumed to be piecewise smooth everywhere away from occlusions (unlike the image that cannot be realistically assumed to be piecewise smooth). For any given tolerance ϵ, one can devise a partition of the co-visible region such that the cumulative approximation is within ϵ. This corresponds to a *motion segmentation* task. We break this process in two parts. The first involves detecting contours and the second part involves using those contours to segment the image frame. Breaking this process into these two components allows us to retrieve several different segmentations depending on the strength of the boundary. As we will see, this enables adaptive merging of neighboring regions to trade off complexity and fidelity. We use [1] for contour detection and segmentation. In Fig. 2 we show representative samples of this process.

2.2 Proper sampling

Co-visible regions can be put in correspondence, in the sense that there exists a deformation of the co-visible domain that maps one image onto another: $D \backslash \Omega(t) = \{w(x,t) \mid x \in D \backslash \Omega(t-1)\}$. However, individual pixels can have multiple correspondents, for instance when the co-visible region is homogeneous or self-similar. These regions will be spatially encoded as "textures" in Sect. 2.3. Everywhere else, we can establish a local reference frame via a process known as *canonization*

Figure 1: (COLOR) *Occlusion detection and optical flow. Original images (top) and optical flow (bottom), visualized according to the color scheme shown in the top left corner of each image. Black regions are occluded, so no motion estimate is available since there is no region in image* $I(x, t-1)$ *that, transported with* $w(x,t)$ *yields* $I(x,t)$.

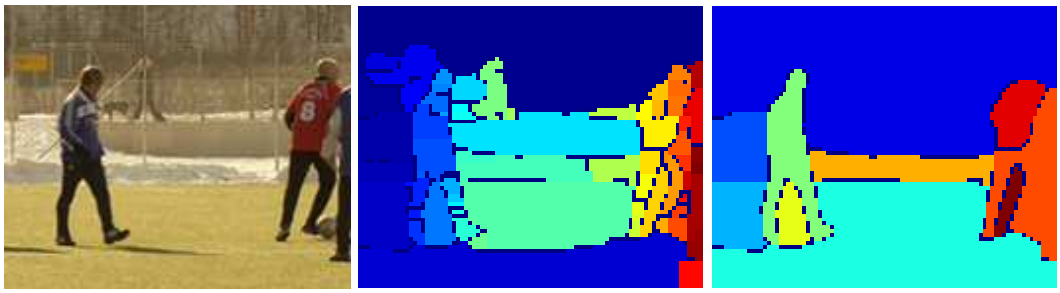

Figure 2: (COLOR) *Segmentation [1]. Varying a threshold yields structurally different segmentations. Original test frame (left). Segmentation using a low threshold (center). Segmentation with a higher threshold (right).*

[18]: Co-variant detector functionals [13] operate on a multi-scale representation of the data and are used to detect extrema that correspond to *"canonical"* local frames. However, we do not know whether such extrema are due to the "scene" (e.g. a material transition), or are "aliasing" phenomena (e.g. discontinuities due to spatial quantization or noise). Ideally, for the discretization to be a *proper sampling*, we would like for the response of co-variant detector functionals operating on the image $I(x,t)$ to be *topologically equivalent* to the response of the same functionals operating on the scene $\rho \circ w(x,t)^{-1}$. Unfortunately, we do not know the scene, so this hypothesis is not testable. However, under the assumptions of Lambertian reflection and co-visibility, proper sampling is equivalent to topological consistency between *different images of the same scene*, for instance $I(x,t), I(w(x,t), t-1)$. The portions of the regions that are properly sampled are then "trackable" [13], and can be encoded once. The aliased structures are instead lumped in the ensemble description that is treated by the texture module described in Sect. 2.3.2. Additional processing can be performed to aggregate tracks over multiple views [19]. Covering trajectories on sequences are shown in Fig. 3.

Figure 3: *Covering of trackable regions [19]. The rest is encoded by the texture module.*

2.3 Structure/texture partition

Once proper sampling is established, and therefore temporal redundancy is accounted for, we consider spatial regularity. This can be of two kinds: Spatial regularity of trackable regions and their description [13], and spatial stationarity of the (colored) noise process that is independently sampled in different temporal frames. This corresponds to the notions of "regular" and "stochastic" textures [23].

As discussed in Sect. 2.1, encoding the motion field point-wise in trackable regions would be costly. Given a segmentation, we can assign the same motion vector to the entire region, or a sub-partition. Unfortunately, segmentation does not usually respect structure/texture partition and can contain areas that are trackable, or not. Since scale composed with quantization is a *semi-group*, rather than a group, we have to consider multiple segmentations at different scales (hierarchical segmentation) and aggregate motion vectors only at the finest scales. We then merge regions in a greedy fashion to satisfy both motion constraints and structure/texture partition. We show representative results in Fig. 4.

Figure 4: *Original Test Sequences (top). Texture/Structure partition (bottom). Regions in green are labeled trackable regions and we can assign a motion vector to them. Textured regions are either homogenous regions or stochastic textures and it is preferable to spatially predict them since they cannot be tracked.*

2.3.1 Encoding trackable regions

For a trackable region in frame $I(x, t+1)$, the encoder can select either temporal or spatial prediction. For temporal prediction, we calculate the median optical flow within the region and use it to find the corresponding region in $I(x, t)$, together with the residual error. This is done for each region independently. This allows the encoder to select prediction rules (spatial or temporal) independently for each region. Note that at occluded regions, we do not perform motion estimation.

Spatial prediction can be performed in trackable regions provided that the motion, and the local description of the image around each co-variant frame, is spatially regular, so the image can be considered as a sample from a stationary and ergodic spatial process. Spatial prediction thus yields a form of texture segmentation. However, unlike the case described in the next section, the prediction has to be *temporally consistent*. To this end, we calculate the mean value of each region in an image and propagate it through its motion vectors, to temporally adjacent images. Again, this process is performed independently for each region and yields a (spatial) prediction, together with a residual.

2.3.2 Encoding non-trackable regions

Non-trackable texture regions can be encoded independently in each image, since by definition there is no temporal consistency. One can still exploit spatial redundancy and perform temporal prediction as done in Sect. 2.3.1. Rather than spatially extrapolating a realization we extrapolate statistics and sample independently at each image, thereby eliminating temporal consistency. This is equivalent to texture synthesis and is described next. In addition, the encoder has still the option of using temporal prediction, to enable overcoming errors in the texture/structure partition.

In the texture synthesis mode, we chose to use a variation of the algorithm of [12] because it is relatively efficient and gives results that are perceptually acceptable, even though we are aware that we will pay a price in terms of PSNR. This non-parametric sampling-based approach takes a subset of a texture region and expands it. This is accomplished by minimizing the following energy:

$$E_t(I_{|_B}; \{I_{|_{N(B)}}\}) = \sum_{p \in N(B)} \|I_p - \hat{I}_p\|^2 + \mu \sum_{p \in N(B)} \|DI_p - D\hat{I}_p\|^2 \tag{3}$$

where $I_{|_B}$ is a vectorized version of the textured region to be synthesized and $\{I_{|_{N(B)}}\}$ is the set of vectorized neighborhoods from the input texture. The vectors $\hat{I}_p$ and I_p are neighborhoods of the input and synthesized textures respectively, centered at pixel p, that are closest in appearance. D is the differentiation operator. In principle, $N(B)$ can be every pixel on the synthesized image grid. For computational complexity reasons, we only consider a subset of them.

The energy above is minimized using an EM-like alternating minimization. The first step minimizes the energy by direct differentiation with respect to I_p that yields a linear system of equations. The second step uses the calculated I_p to find the set $\{\hat{I}_p\}$, using nearest-neighbor (NN) search. The process is repeated until the decrease in energy is negligible, as customary. In addition, a re-weighting scheme is used to improve outlier rejection (similar to iteratively re-weighted Least Squares (IRLS)). The scheme is repeated over 3 neighborhood sizes: $w = [32, 16, 8]$. Finally, the minimization procedure is repeated over a number of different resolutions (i.e. image sizes). Therefore, the whole algorithm is performed in a multi-resolution multi-scale fashion. The relative weight parameter is set to $\mu = 10$.

The algorithm works well for fine-scale textures; however, textures that exhibit structure at coarse scale (e.g., the bricks in Fig. 5) require large region sizes to be captured. This is particularly severe when the regions are selected with a uniform prior. To minimize this effect, we replace M-Step with the following minimization problem for all I_p, $p \in N(B)$:

$$\hat{\alpha} = \arg\min_{\alpha_p} \|I_p - Y\alpha_p\|_2^2 + \lambda||\alpha_p||_1 \tag{4}$$

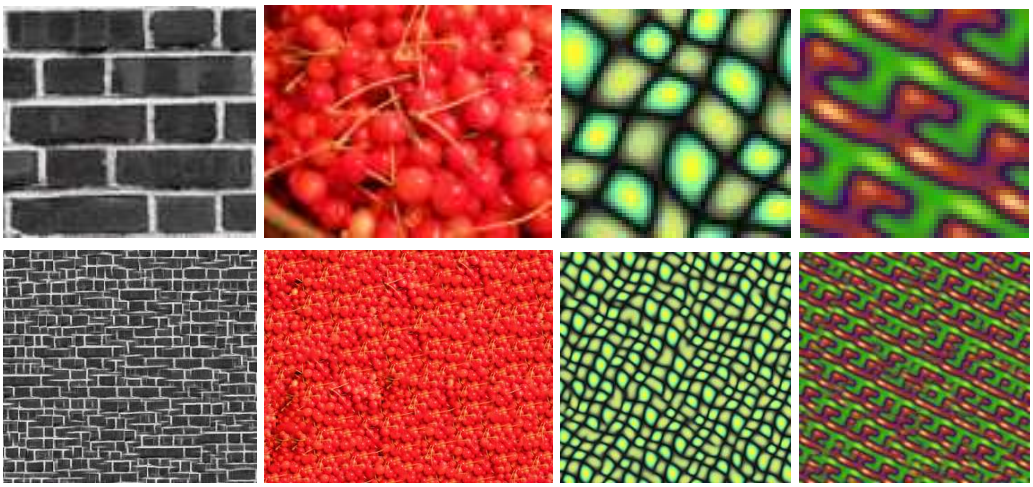

Figure 5: *Low resolution image patches (top). Image Patches of higher resolution synthesized with our algorithm (bottom).*

where $Y = [I_1\ I_2\ \ldots\ I_N]$ are the concatenated vectorized neighborhoods from the set $\{I_{|_{N(B)}}\}$ and N is the total number of neighborhoods in the original image that we consider. Thus, instead of selecting one NN, we choose a sparse linear combination. This improves the stability of the algorithm in degenerate cases. We show results of our texture synthesis algorithm in Fig. 5. Artifacts are still visible for the brick texture, but they are less severe than without this additional step. To synthesize textures, we always encode a quarter of the size of the region and synthesize the rest.

2.4 Encoding video frames

After calculating the predicted images using the spatial, temporal and synthesize modes, the encoder chooses for each region the prediction mode that yields the smallest residual. We maintain an index function for the prediction mode used for each region. The predicted and error images are then formed. If a region is predicted temporally, the predicted region is taken to be the one calculated with the temporal prediction. The same applies for the other two modes. Since each region has been predicted independently, there is no issue of inconsistency. The error image is transformed and quantized in the same way as in H.264 [11], zig-zag scanned and entropy encoded. We encode the boundary indicator function of the regions using run length encoding (RLE). We also use RLE for the indicator function of the occluded pixels. We use entropy coding on the output of the RLE. Motion vectors, mean values for spatial prediction and the small patches for textured regions are also entropy encoded. The predicted image plus the error image form the new image frame, which is then subsequently used as the previous frame for encoding the following frame. Hence the decoder is embedded in the encoder, similar to modern coding standards. All quantities are quantized at the quantization level specified by the user.

3 Experiments

To evaluate our encoder we compared its performance with four different encoding methods. In the baseline method used (blue curve in Fig. 7), we encoded the video sequence by quantizing and entropy coding each frame independently. In the second method (yellow curve) we calculated the difference image between the last encoded frame and the next frame. We quantized and entropy coded the difference. The other two methods we compared with was MJPEG (green curve) and H.264 JM ver.18.0 (orange curve)[2]. In the H.264 JM implementation, we used default parameters

[2]The H.264 JM implementation can be found at http://iphome.hhi.de/suehring/tml/

Figure 6: *Qualitative results of our encoder. Predicted image using spatial, temporal and texture synthesis prediction modes (left). Residual error: black corresponds to zero error, white to large (center). Reconstructed video frames (right).*

provided by the authors. We varied the performance of the encoder by changing the quantization parameters. In our encoder we fixed the parameters for all experiments and we also only varied the quantization parameter. For optical flow estimation and tracking we used the default parameters set by the authors of each paper. For the hierarchical segmentation we used 3 levels. In Fig.6 we show the prediction performance of our encoder. The left column corresponds to the predicted image. Small or zero error (middle column) can be achieved in most of the image domain. Where large deformations of the domain occur, the translational model fails (e.g. the face of foreman).

Quantitative results are shown in Fig. 7 for four video sequences of QCIF resolution (city, flower garden, mobile, coastguard). PSNR scores of the Y-channel are reported. We used 50 frames from each video sequence [3]. It can be seen that our encoder significantly outperforms the other methods at almost all bit rates. The biggest gain comes at high bit rates where the gain in PSNR in using our encoder compared to others becomes significantly large.

Following is a break-down of the contribution of each module to the overall computational cost. For VGA resolution video, the GPU implementations of the optical flow, tracking and segmentation algorithms take 10, 2 and 13 seconds respectively. Segmentation can be ran in parallel with the other two algorithms. Texture synthesis takes approximately 5 seconds per frame. It was found that the average encoding time per frame was 21 seconds. The reported computational time was recorded on a dual core 2.4 GHz desktop with an NVIDIA GeForce GTX 560 Ti GPU.

[3]The video sequences can be found at http://media.xiph.org/video/derf/

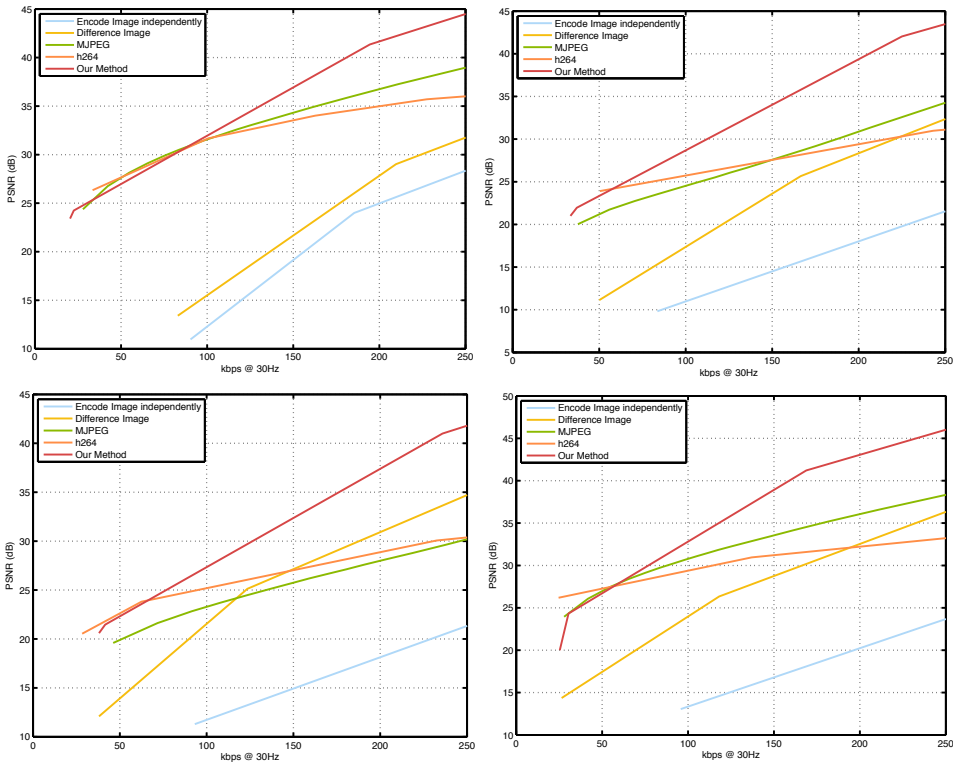

Figure 7: *Quantitative comparison of our encoder with other methods. Our method significantly outperforms all other methods (including H.264) in a wide range of kbps in terms of PSNR. Results shown are for the sequences city (upper left), flower garden (upper right), mobile (bottom left) and coastguard (bottom right). Comparisons were ran for a duration of 50 frames.*

4 Discussion

We have presented a modeling framework for video compression that exploits the basic phenomenology of optical data-formation, focusing on occlusion and scale. Temporal consistency is exploited by detecting *co-visible regions* (as the complement of occluded regions) and encoding them once, together with a compressed motion model. Spatial consistency is exploited by partitioning the data into structures (that are encoded by sparse bases) and "texture" (samples from a locally stationary distribution), that can be encoded in ensemble form, and sampled at decoding. The decision as to what is "properly sampled", and therefore can be tracked, as well as what is visible, can only be made by considering multiple images.

While the compression performance of our approach is demonstrably better than existing schemes, this is not surprising since the model class we consider is far larger than that implied in traditional approaches. The obvious downside is a significant computational complexity. However, we believe that rather than incrementally refining intrinsically linear superposition models with additive uncertainty, embracing the full non-linearity implied by occlusion and scaling is necessary to break the video compression wall and develop the next generation of video compression schemes. Also, the results we have displayed use standard PSNR as a performance score. Clearly this is inadequate for the task of compressing video for human fruition. A perceptual score is more appropriate, although the lack of a universally accepted model of perceptual performance prompted us to use the more conventional PSNR. It is our intention to further the development of perceptual scores, and integrate them into the spatio-temporal encoding of the scene radiance, so that albeit not observable (unique), at least the model can be *minimal* with respect to the specific task at hand. This is subject of on-going research.

Acknowledgments. This research was supported by DARPA KECoM N66001-11-C-4001 and DARPA MSEE.

References

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE TPAMI*, 2011.

[2] A. Ayvaci, M. Raptis, and S. Soatto. Sparse occlusion detection with optical flow. *IJCV*, 2011.

[3] M. Black and P. Anandan. The robust estimation of multiple motions: parametric and piecewise smooth flow fields. *Computer Vision and Image Understanding*, 1996.

[4] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. *ECCV*, 2004.

[5] CISCO. Entering the zettabyte era, visual networking index. CISCO VNI, 2011.

[6] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE TPAMI*, 2002.

[7] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. *Proc. of ACM SIGGRAPH*, 2001.

[8] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 2004.

[9] C. E. Guo, S. C. Zhu, and Y. N. Wu. Towards a mathematical theory of primal sketch and sketchability. *ICCV*, 2003.

[10] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 1981.

[11] ITUT-Recommendations. http://www.itu.int/itu-t/recommendations/.

[12] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. Texture optimization for example-based synthesis. *Proc. of ACM SIGGRAPH*, 2005.

[13] T. Lee and S. Soatto. Video-based descriptors for object recognition. *Image and Vision Computing*, 2011.

[14] L. Liang, C. Liu, Y. Xu, B. Guo, and H. yeung Shum. Real-time texture synthesis by patch-based sampling. *Proc. of ACM Transactions on Graphics*, 2001.

[15] C. Liu, J. Yuen, A. Torralba, J. Sivic, and W. T. Freeman. Sift flow: Dense correspondence across different scenes. *ECCV*, 2008.

[16] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry. *An invitation to 3D vision, from images to models*. Springer Verlag, 2003.

[17] P. Sand and S. Teller. Particle video: Long-range motion estimation using point trajectories. *CVPR*, 2006.

[18] S. Soatto. *Steps Toward a Theory of Visual Information*. ArXiv http://arxiv.org/abs/1110.2053, Technical Report UCLA-CSD100028, September 13, 2010.

[19] N. Sundaram, T. Brox, and K. Keutzer. Dense point trajectories by gpu-accelerated large displacement optical flow. *ECCV*, 2010.

[20] G. Sundaramoorthi, P. Petersen, V. S. Varadarajan, and S. Soatto. On the set of images modulo viewpoint and contrast changes. *CVPR*, 2009.

[21] P. Teo and D. Heeger. Perceptual image distortion. *ICIP*, 1994.

[22] C. Tomasi and J. Shi. Good features to track. *CVPR*, 1994.

[23] Y. N. Wu, C. Guo, and S. C. Zhu. From information scaling of natural images to regimes of statistical models. *Quarterly of Applied Mathematics*, 2008.

[24] H. Zhi, Z. Xu, and S.-C. Zhu. Video primal sketch: A generic middle-level representation of video. *ICCV*, 2011.