

Region-of-Interest Prediction for Interactively Streaming Regions of High Resolution Video

EE392J Group Project Report

by

Aditya Mavlankar and David Varodayan

Information Systems Laboratory, Stanford University, Stanford, CA 94305

{madya, varodayan}@stanford.edu

Abstract

This project considers region-of-interest (ROI) prediction strategies for a client-server system that interactively streams regions of high resolution video. The system operates in two modes. In manual mode, the user interacts actively to view select regions in each frame of video. In tracking mode, the user simply indicates an object to track and the system supplies a ROI trajectory without further interaction. In both cases, the client has a buffer of low resolution overview video frames available. We propose and study ROI prediction schemes that can take advantage of the motion information contained in these buffer frames. For the manual mode, prediction aids pre-fetching and aims to reduce distortion experienced by the viewer. For the tracking mode, the prediction aims to create a smooth and stable trajectory that satisfies the user's expectation of tracking.

1 Introduction

High resolution digital imaging sensors are becoming more widespread. In addition, high spatial resolution videos can also be stitched from views from multiple cameras, as implemented by Hewlett-Packard in their video conferencing product called Halo [1]. However, challenges in delivering this high resolution content to the client are posed by the limited resolution of display panels and/or limited bit-rate for communications. Imagine that there is a client who is limited by one of these factors and is requesting the server to stream a high spatial resolution video. One approach would be to stream a spatially downsampled version of the entire video scene to suit the client's display window resolution or bit-rate. However, with this approach, the client might not be able to watch a local region-of-interest (ROI) in the highest captured resolution. We propose a video delivery system which enables virtual pan/tilt/zoom functionality during the streaming session such that the server can adapt and stream only those regions of the video content that are desired at that time at the client's end.

We have developed a user interface with real-time interaction for ROI selection while watching the video sequence. This has been developed using OpenGL [2]. As shown in Fig. 1, the display screen at the client's side consists of two areas:

- The first area displays a downsampled version of the entire scene. We call this the overview display area. It is b_w pixels wide and b_h pixels tall.
- The second area displays the client's ROI. We call this the ROI display area. It is d_w pixels wide and d_h pixels tall.

The zoom factor can be controlled with the scroll of the mouse. For any zoom factor, the ROI can be moved around by keeping the left mouse-button pressed and moving the mouse. As shown in Fig. 1, the location of the ROI is depicted in the overview display area by overlaying a corresponding rectangle on the video. The color and size of the rectangle vary according to the zoom factor.

2 Problem Description

The overall system is shown in Fig. 2. Notice that in our system, the client indicates his ROI and the desired spatial resolution (zoom factor) real-time to the server. The server then reacts to this by



Figure 1: User interface: The display screen consists of the overview display area and the ROI display area. The effect of changing the zoom factor can be seen by comparing left and right hand sides of the figure.

sending relevant video data which is decoded and displayed at the client’s side. The server should be able to react to the client’s changing ROI with as little latency as possible. However, streaming over a best-effort packet-switched network implies delay, delay jitter as well as loss of packets. To keep the complexity low, we design the system to work for a known value of the worst-case delay, assuming that there is no packet loss on the network. Since the overview video always displays the entire scene, we allow some start-up delay and always send some number of frames of the overview video ahead of time. We can choose the size of the buffer such that it enables us to always deliver a minimum number of frames of the overview video in advance despite the worst-case delay of the network.

We assume that the client renders the requested ROI while displaying the immediately next frame after obtaining the ROI location information from the mouse; i.e., the worst-case latency for the interactive part of the system is equal to one frame-interval. Meeting the display deadline for the ROI display area is challenging since the ROI is not known beforehand and is being decided by the user real-time. As one possible solution, we propose to predict the ROI of the user beforehand and use this to pro-actively pre-fetch those regions from the server. A timeline is shown in Fig. 3. The lookahead is d , i.e., we pre-fetch the ROI d frame-intervals in advance. Specifically, we are trying to solve the following two problems in this project:

- Manual mode: Predict the user’s ROI d frame-intervals ahead of time. Note that the user continues to indicate his choice of the ROI in this mode.
- Tracking mode: The user right-clicks on an object in the ROI. The aim is to track this object automatically in order to render it within the ROI till the user switches this mode off.

2.1 Manual mode

Predicting the future ROI could be done by extrapolating the mouse moves observed until the current time instant. Ramanathan used a simple autoregressive moving average (ARMA) model for predicting the future viewpoint of the user in his work on interactive streaming of lightfields [3]. Kurutepe et al. used a more advanced linear predictor, namely, the Kalman filter to predict the future viewpoint for interactive 3DTV [4], [5]. However, all these approaches are agnostic of the video content itself. In this project, we investigate possible improvement of the ROI prediction by processing the frames from

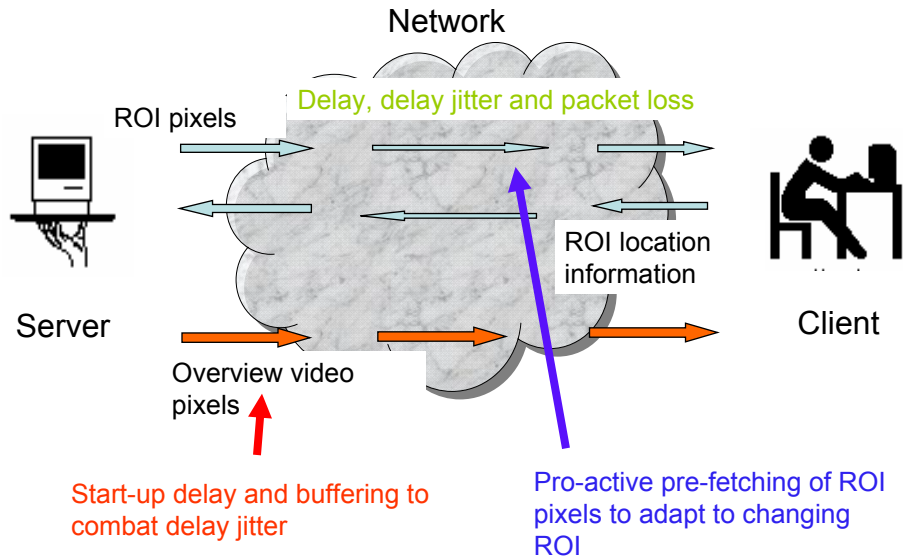


Figure 2: Problem description: Streaming over a realistic network entails delay, delay jitter and packet loss. Some number of overview video frames are always delivered beforehand to the client. The ROI is predicted at the client’s end at least one roundtrip time in advance and the appropriate ROI pixels are pro-actively pre-fetched.

the overview video which are already present in the client’s buffer. Notice that the motion estimated through the processing of the buffered overview frames can also be combined with the observations of mouse moves to predict the ROI better.

If the wrong regions are pre-fetched then the user’s desired ROI can still be rendered by interpolating the colocated ROI from the overview video. The quality of the rendered ROI would be lower in this case. Assuming this concealment scheme, we can evaluate the impact of the prediction objectively by computing the mean distortion in the rendered ROI with respect to the ROI rendered from the original video sequence. Note that in this project, we do not compress the high resolution layers of the video and also simplify the computation of the distortion even further; the distortion due to error concealment is computed by comparing against the ROI rendered in case of perfect ROI prediction as a reference. The overview video, also called as base layer, is compressed using H.264/AVC and the reconstructed signal is upsampled for error concealment.

2.2 Tracking mode

Note that in this mode, we are allowed to shape the ROI trajectory at the client’s side. Hence we do not need to calculate the distortion-based metric as described above. We generate videos which show the automatic tracking of objects. We also show videos in which a human operator tracks the same objects in the manual mode as described above.

3 Description of Algorithms

The common objective of our various algorithms is to predict the user’s ROI d frames ahead of the currently displayed frame n . Notice that this requires a 3D prediction in two spatial dimensions and one zoom dimension. As discussed in the previous section, there are two modes of operation. In the manual mode the algorithms may process the user’s ROI trajectory history up to the currently displayed frame n . In the tracking mode, this source of information does not exist. The novelty of our work is that in both modes the buffered overview frames (including n through $n + d$) are available at the client,

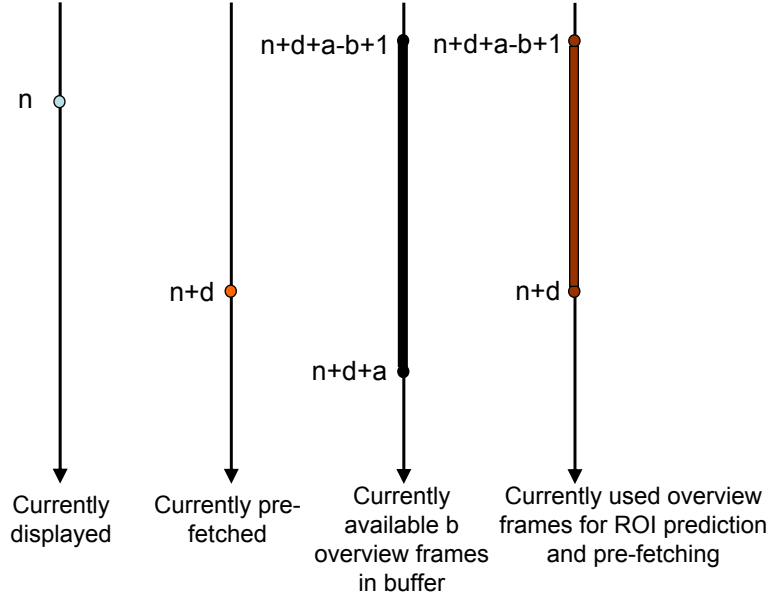


Figure 3: Timeline: The ROI is predicted d frame-intervals in advance. There are b buffered overview video frames up to frame $(n + d + a)$ available. A subset of these b frames are processed in order to better predict the user’s ROI. If the manual mode is switched on then the mouse translations are known till time instant n . In the tracking mode, the user does not indicate the ROI explicitly through the mouse.

as shown in Fig. 3. This means our algorithms may exploit the motion information in those frames to assist the ROI prediction.

3.1 Manual Mode Algorithms

3.1.1 Autoregressive Moving Average (ARMA) Model Predictor

We first adapt the straightforward ARMA trajectory prediction algorithm of [3] that is agnostic of the video content. This provides a framework for extrapolating the spatial co-ordinates of the ROI. Suppose, in the frame of reference of the overview frame, the spatial co-ordinates of the ROI trajectory are given by $p_t = (x_t, y_t)$ for $t = 0, 1 \dots, n$. Then, we recursively estimate the velocity v_n according to

$$v_t = \alpha(p_t - p_{t-1}) + (1 - \alpha)v_{t-1}. \quad (1)$$

The parameter α is used to trade off responsiveness to trajectory changes and smoothness of the overall trajectory. The spatial co-ordinates $p_{n+d} = (x_{n+d}, y_{n+d})$ of the ROI at frame $n + d$ are predicted as

$$p_{n+d} = p_n + dv_n, \quad (2)$$

suitably cropped if they happen to veer off the extent of the overview frame. The zoom co-ordinate of the ROI cannot be predicted in this way because our rendering system does not allow continuous zoom. Since there are only a small number of discrete zoom levels, we choose to predict the zoom z_{n+d} at frame $n + d$ as the observed zoom z_n at frame n .

3.1.2 Kanade-Lucas-Tomasi (KLT) Feature Tracker Based Predictor

The second algorithm we consider for the manual mode of operation does exploit the motion information in the buffered overview video frames. As shown in the processing flowchart in Fig. 4, we first apply the Kanade-Lucas-Tomasi (KLT) feature tracker [6] to perform optical flow estimation on the buffered overview video frames. This yields the trajectories of a large (but limited) number of feature points from

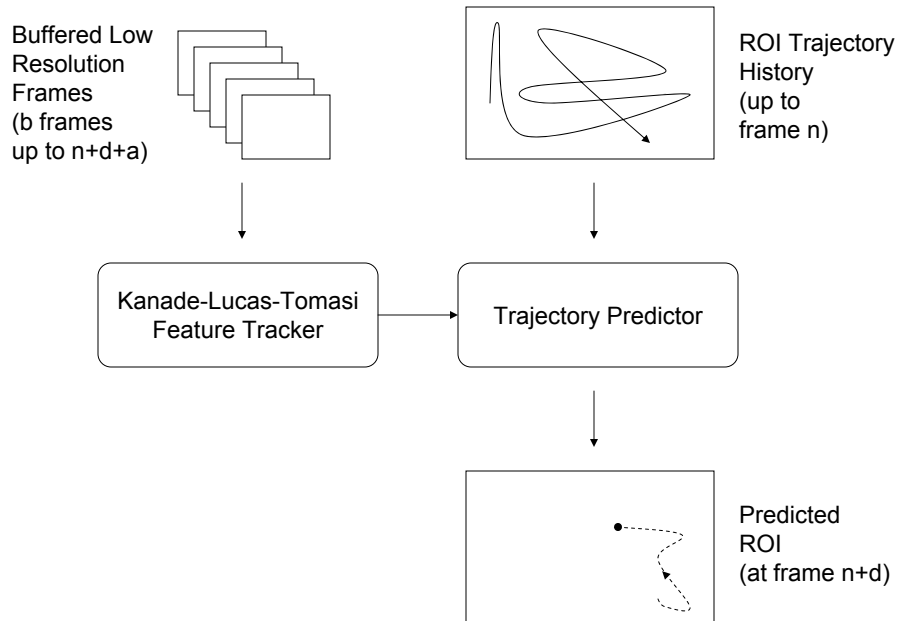


Figure 4: Flowchart of processing for the Kanade-Lucas-Tomasi Feature Tracker trajectory prediction algorithm.

frame n to frame $n + d$. The trajectory predictor then incorporates these feature trajectories into the ROI prediction for frame $n + d$.

We use an open implementation of the KLT feature tracker [7]. At its core, this software solves the Lucas-Kanade equation by inverting a so-called gradient coefficient matrix G for each feature window. For a window of pixels W in an image I , the gradient coefficient matrix is

$$G = \int_W (\mathbf{g}\mathbf{g}^T w) dx dy, \quad (3)$$

where gradient $\mathbf{g} = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$ and w is a weighting function over the window W . Among several feature windows, the Lucas-Kanade equation is solved with greatest confidence for those whose matrix G is farthest from singular. In particular, if both eigenvalues of G are large, the feature window contains variation in two directions, making it a corner or some other texture. This suggests the criterion for the ranking of feature windows in terms of their suitability for tracking: descending order of minimum eigenvalue of G .

In our application, the KLT feature tracker begins by analyzing frame n and selecting a specified number of the most suitable-to-track feature windows according to this condition. Then it solves the Lucas-Kanade equation for each selected window in each subsequent frame up to frame $n + d$. Each solution involves an iterative Newton-Raphson method performed in a multiresolution pyramid to search efficiently over relatively large displacements. Figs. 5 and 6 show the features tracked from frames 250 to 259 of the *Tractor* and *Sunflower* sequences, respectively. Note that most (but not all) feature trajectories are propagated to the end of the buffer.

It is these feature trajectories that are exploited to predict the user's ROI at frame $n + d$. Among the features that survive from frames n to $n + d$, the trajectory predictor finds the one nearest the center of the ROI in frame n . It then predicts spatial co-ordinates of the ROI to center that feature in frame $n + d$. As in the ARMA model predictor, the zoom z_{n+d} is predicted as the zoom z_n . In essence, this predictor follows and centers the KLT feature nearest the center of the ROI.

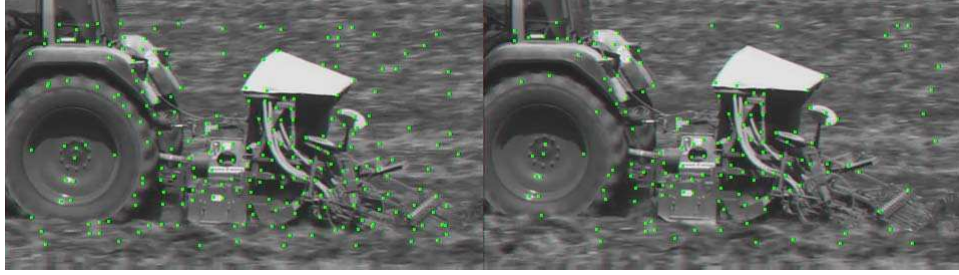


Figure 5: Features propagated from frame 250 (left) of the *Tractor* sequence to frame 259 (right)

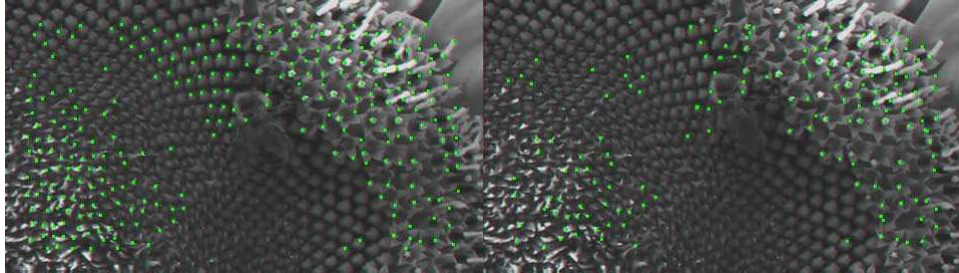


Figure 6: Features propagated from frame 250 (left) of the *Sunflower* sequence to frame 259 (right)

3.2 Tracking Mode Algorithms

Algorithms for the tracking mode differ from those for manual mode in two significant ways. Firstly, these algorithms cannot expect ongoing ROI information as input from the user. Instead a single click on a past frame indicates which object to track in the scene. This rules out predictors that are agnostic of the video content, like the one based on the ARMA model. Secondly, the predicted ROI trajectory in tracking mode is actually presented to the user. This imposes a smoothness requirement on the ROI for pleasant visual experience.

3.2.1 Kanade-Lucas-Tomasi (KLT) Feature Tracker Based Predictor

Just as for the manual mode, we employ the KLT feature tracker to extract motion information from the buffered overview frames. In the absence of ongoing ROI information from the user, we propose two distinct ROI trajectory predictors, a centering predictor and a stabilizing predictor. Then we combine them as a blended predictor. All these predictors begin by identifying the feature nearest the user’s initial click in the frame in which the click was made. Then they follow that feature trajectory into future frames.

The centering predictor chooses the spatial co-ordinates of the predicted ROI to center the feature trajectory. Whenever the feature being followed disappears during propagation, this predictor starts following the surviving feature nearest to the one that disappeared. It continues to center the new feature trajectory within the ROI window. This centering strategy introduces jerkiness into the predicted ROI trajectory each time the feature being followed disappears.

We seek to mitigate this effect with the stabilizing predictor. Once again the predictor follows a feature into future frames, but now it selects the spatial co-ordinates of the predicted ROI to keep the feature in the same location with respect to the ROI. As in the centering predictor, should a feature disappear, this predictor begins to follow the surviving feature nearest to the one that disappeared. But it now keeps the new feature where it was found in the ROI. The stabilizing predictor is designed to create very smooth trajectories but runs the risk of drifting away from the object selected by the user.

To combine the advantages of the centering and stabilizing predictors, we blend their predictions

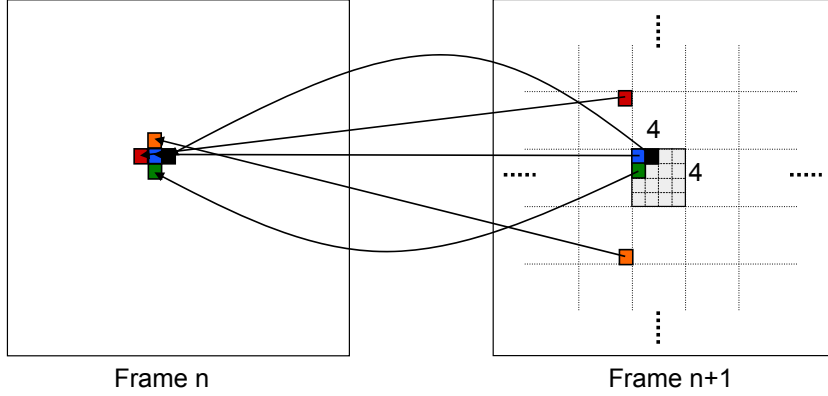


Figure 7: Propagation of chosen point from frame n to frame $n + 1$ by making use of motion vectors sent by the server.

$p_{n+d}^{centering}$ and $p_{n+d}^{stabilizing}$ according to a fixed parameter β :

$$p_{n+d}^{blended} = \beta p_{n+d}^{centering} + (1 - \beta) p_{n+d}^{stabilizing}. \quad (4)$$

Similar to the α for the ARMA model predictor, β trades off the blended predictor’s responsiveness to motion cues in the video and its trajectory smoothness.

For each of these three predictors, the predicted spatial co-ordinates of the ROI are cropped if they veer off the extent of the overview frame. Also, the zoom z_{n+d} is predicted as the observed zoom z_n .

3.2.2 H.264/AVC Motion Vectors Based Predictor

The client always receives some encoded frames of the overview video in advance. Included in the coded representation are the motion vectors which attempt to describe motion from frame to frame. In the tracking mode, we use these motion vectors to propagate the pixel at the point which the user indicates at the beginning of the tracking mode. The pre-fetched and rendered ROI is centered around this pixel and the user is still allowed to change the resolution through zoom operations.

Imagine there is a pixel in frame n , i.e., P_n , which we want to propagate to the future frames. The simple approach is to choose a pixel in frame $(n + 1)$, i.e., P_{n+1} , which is connected via its motion vector to the given pixel P_n . We refer to this as the first approach. This simple approach is not robust enough and the pixel might drift out of the object that needs to be tracked.

In order to make the above approach robust, we first determine the pixels in frame $(n + 1)$ connected to the four nearest neighbors of P_n . This is shown in Fig. 7. Out of these five pixels in frame $(n + 1)$, we choose one pixel which is then propagated forward in the future frames. For choosing one out of these five pixels we tried two metrics:

- Choose that pixel which minimizes the sum of the squared distances to the remaining four pixels in frame $n + 1$.
- Choose that pixel which has the minimum squared difference in intensity value compared to P_n .

We found that the second metric is more robust for the test trajectories. This algorithm is referred to as the second approach below.

4 Results

We captured ROI trajectories for three video sequences (*Tractor*, *Sunflower* and *Pedestrian Area*) with the highest resolution of 1920x1088 and 3 zoom factors and a fourth video sequence (*Card Game*) with the highest resolution of 3584x512 and 2 zoom factors. The ROI display area was 480x272 for the first

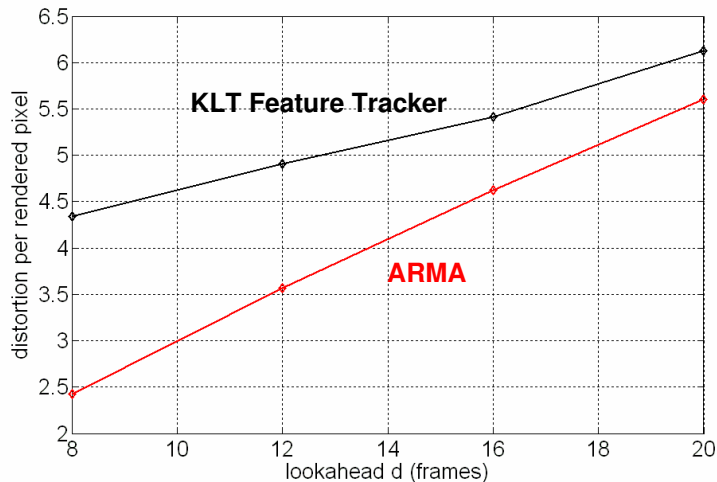


Figure 8: Distortion per rendered pixel for the ARMA and KLT feature tracker predictors for trajectory 1 over 600 frames of the *Tractor* sequence. Trajectory 1 can be roughly described as an attempt to track the entire tractor.

three sequences and 480x256 for the fourth. The overview videos with a resolution of 480x272 for the first three sequences and that for the fourth video sequence with a resolution of 896x128 were encoded using H.264/AVC.

4.1 Manual Mode Results

The manual mode predictors are evaluated based on the distortion they induce in the user’s selected ROI for a given ROI trajectory through a sequence. Whenever there is a mismatch between the predicted ROI and the user’s selected ROI, the samples unavailable at the client are concealed by upsampling the lower resolution overview frame, creating distortion. We have not used any compression scheme for the higher resolution video signals and we assume that if the ROI is perfectly predicted then the distortion is zero. Hence, the distortion shown in the results is calculated by comparing against the ROI rendered in case of perfect ROI prediction as a reference.

In our experiments, we compare the ARMA model predictor and the KLT feature tracker based predictor. For the ARMA model predictor, we follow [3] and choose parameter $\alpha = 0.5$. We set the KLT feature tracker to propagate the 300 most suitable-to-track features from frame n to frame $n + d$.

Figures 8 and 9 show the distortion per rendered pixel of the two trajectory prediction schemes for two different user ROI trajectories through the *Tractor* sequence as a function of d . Trajectory 1 can be roughly described as an attempt to track the entire tractor using the intermediate zoom level. Trajectory 2 can be described as an attempt to track the machinery attached to the back of the tractor using the deepest zoom level.

In Fig. 8, the ARMA model predictor outperforms the KLT feature tracker based predictor for the range of lookahead values d examined. This is because trajectory 1 is quite smooth. Fig. 9 is different in two respects. The distortion per rendered pixel for both predictors on trajectory 2 is much higher than on trajectory 1, since trajectory 2 is at the deepest zoom level. This makes the concealment from the overview frame less faithful. The deeper zoom and the unstable content make trajectory 2 jerkier than trajectory 1. This accounts for the second difference in Fig. 9: the KLT feature tracker outperforms the ARMA model predictor. For both trajectories, the ARMA model predictor degrades faster than the KLT feature tracker based predictor with increasing lookahead d .

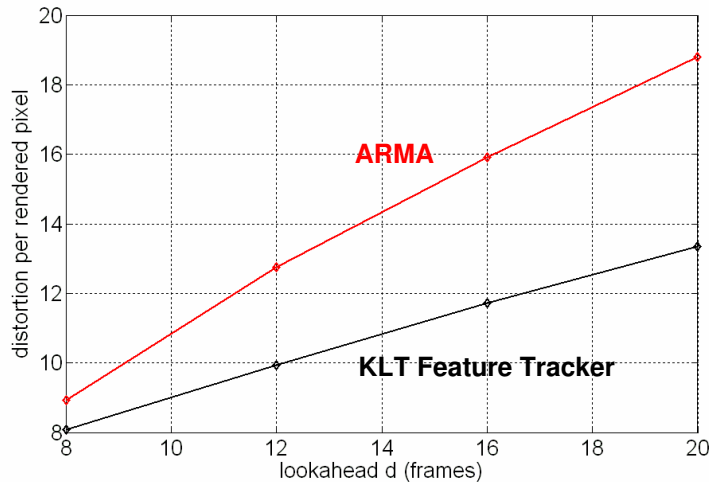


Figure 9: Distortion per rendered pixel for the ARMA and KLT feature tracker predictors for trajectory 2 over 600 frames of the *Tractor* sequence. Trajectory 2 can be described as an attempt to track the machinery attached to the back of the tractor.

4.2 Tracking Mode Results

In the tracking mode, the client displays the video pre-fetched according to the predicted ROI trajectory. So the evaluation is purely visual since there is no distortion due to error concealment.

Among the KLT feature tracker predictors applied to the trajectories on the *Tractor* sequence, the centering predictor is effective at tracking but creates visually-unappealing jerky trajectories. On the other hand, the stabilizing predictor produces smooth trajectories that diverge from the object to be tracked. Subjective experimentation suggests that a good compromise is achieved by the blended predictor with parameter $\beta = 0.25$.

This blended predictor also works well for the trajectory recorded for the *Card Game* sequence, but fails to track the bee in the *Sunflower* sequence. This failure indicates one of the drawbacks of using the KLT feature tracker for trajectory prediction. As seen in Fig. 6, the bulk of the most suitable-to-track features for the *Sunflower* sequence do not lie on the bee, the most semantically-interesting object. For sequences with this property, these KLT feature tracker predictors are not robust.

The results with the motion vectors based tracker are quite encouraging. The first approach is not robust enough and the tracked pixel can drift out of the object that needs to be tracked. The second approach, as described in section 3.2.2, is quite robust. The comparison of the two approaches is shown in Fig. 10. As can be seen on the *left* side of the figure, the tracked point fails to track the object till the end of the sequence with the first approach, whereas the second approach can track the object till the end. Figures 11 and 12 show more results with the second approach and the robustness can be seen.

5 Conclusions

In this project, we have implemented and evaluated several ROI predictors for interactively streaming regions of high resolution video. In the manual mode of operation, we demonstrate that exploiting the motion information in the buffered overview frames to assist pre-fetching can reduce the distortion experienced by the user. It is clear, however, that the best prediction strategy depends on both the trajectory and the video content. How best to combine information from these sources in the predictor remains an open question. The tracking mode of operation is central to system usability. We show that tracking objects is feasible, not only using sophisticated optical flow estimation on the buffer of overview frames, but simply by exploiting H.264/AVC motion vectors already available in the buffered bitstream. A more comprehensive study of these methods is necessary to improve their robustness.

Appendix: distribution of work

David

- Set parameters for and applied Kanade-Lucas-Tomasi feature tracker [7] to video sequences
- Wrote the ARMA model ROI prediction scheme for the manual mode
- Wrote the KLT feature tracker ROI prediction scheme for the manual mode
- Wrote the 3 KLT feature tracker ROI prediction schemes for the tracking mode: centering, stabilizing and blended predictors
- Performed distortion measurements for the two manual mode prediction schemes
- Generated video sequences of the tracking output for the 3 KLT feature tracker ROI prediction schemes for the tracking mode

Aditya

- Encoded overview video sequences using H.264/AVC
- Captured ROI trajectories which involve both manual mode and tracking mode
- Wrote program to evaluate the distortion per rendered pixel, given the original ROI trajectory and the predicted ROI trajectory and the sequence resolution and zoom factors
- Wrote code for generating video outputs in order to visualize the results of implemented algorithms
- Wrote code to pipe out motion vectors from the H.264/AVC bit-stream
- Implemented and optimized the H.264/AVC motion vectors based ROI predictor for the tracking mode

References

- [1] “Halo: Video Conferencing Product by Hewlett-Packard,” Website: <http://www.hp.com/halo/index.html>.
- [2] “OpenGL Application Programming Interface,” Website: <http://www.opengl.org>.
- [3] P. Ramanathan, “Compression and interactive streaming of lightfields,” March 2005, Doctoral Dissertation, Electrical Engg. Department, Stanford University, Stanford CA, USA.
- [4] E. Kurutepe, M. R. Civanlar, and A. M. Tekalp, “A receiver-driven multicasting framework for 3DTV transmission,” *Proc. of 13th European Signal Processing Conference (EUSIPCO)*, September 2005.
- [5] E. Kurutepe, M. R. Civanlar, and A. M. Tekalp, “Interactive transport of multi-view videos for 3DTV applications,” *Journal of Zhejiang University, SCIENCE A* 7(5), 2006.
- [6] C. Tomasi and T. Kanade, “Detection and tracking of point features,” April 1991, Carnegie Mellon University Technical Report CMU-CS-91-132.
- [7] “KLT: Kanade-Lucas-Tomasi Feature Tracker,” Website: <http://www.ces.clemson.edu/~stb/klt/index.html>.

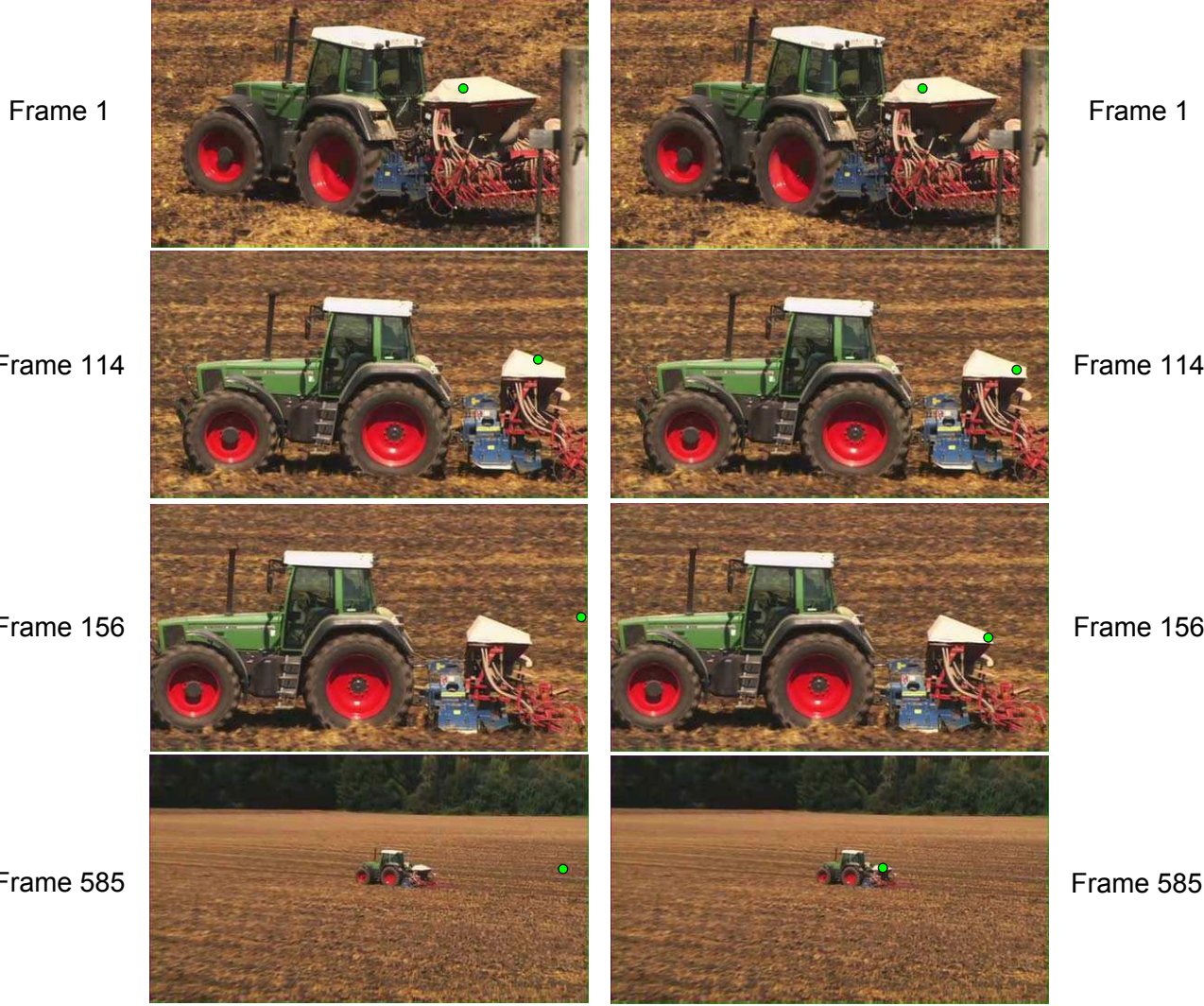


Figure 10: Tracking using motion vectors of the overview video for the *Tractor* sequence 600 frames. The tracked point is highlighted for better visibility. The result of the first approach is shown on the *left* and the result of the second approach is shown on the *right*.



Frame 1



Frame 232



Frame 271

Figure 11: Tracking using motion vectors of the overview video for the *Cardgame* sequence 300 frames. The tracked point is highlighted for better visibility. The second approach for tracking is employed.

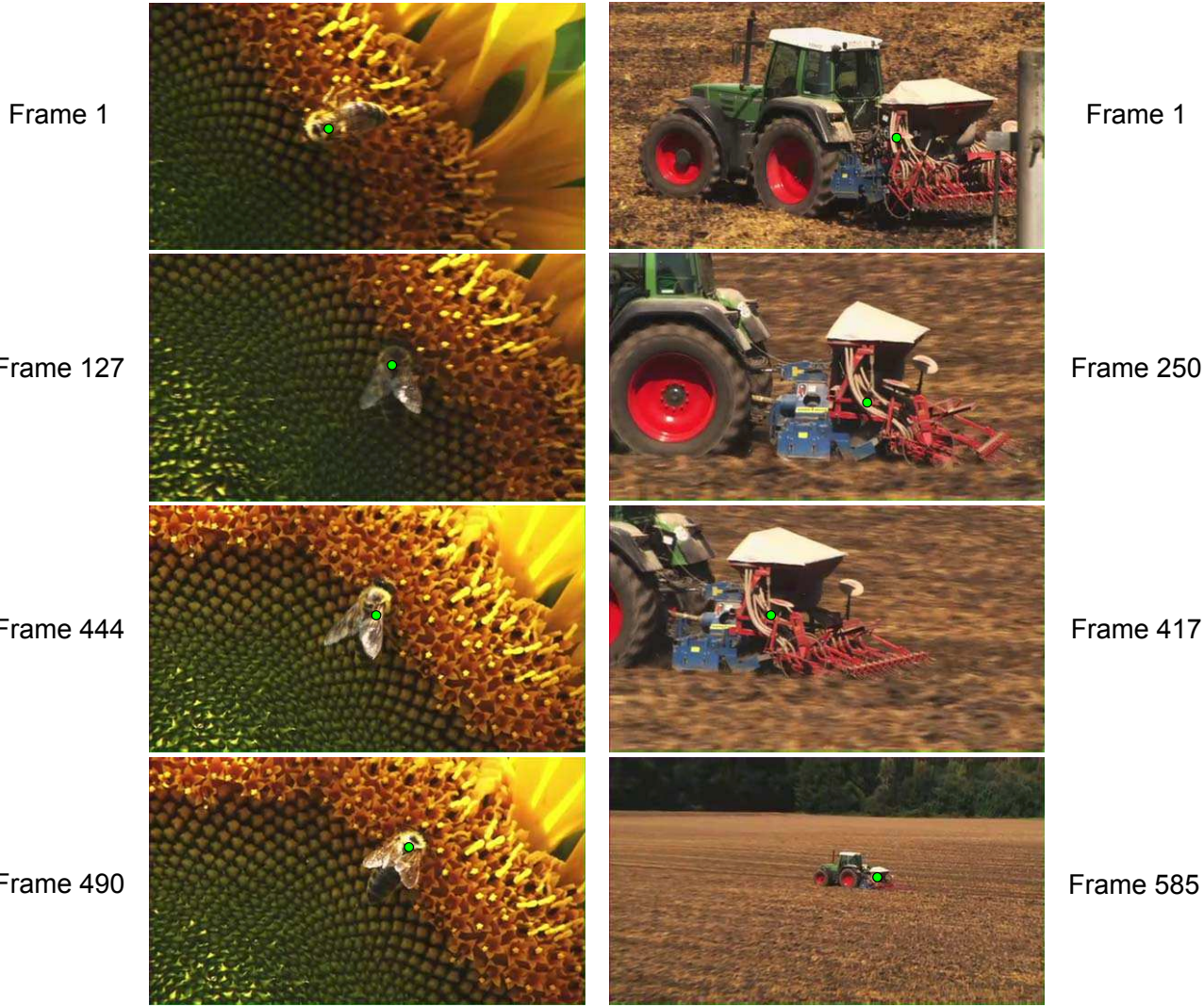


Figure 12: Tracking using motion vectors of the overview video for the *Sunflower* sequence 500 frames and the *Tractor* sequence 600 frames. The tracked point is highlighted for better visibility. The second approach for tracking is employed.