Zoomable Video Playback on Mobile Devices by Selective Decoding

Feipeng Liu and Wei Tsang Ooi

Department of Computer Science,
National University of Singapore,
13 Computing Drive Singapore 117417,
{liuf0005, ooiwt}@comp.nus.edu.sg

Abstract. Many mobile devices are not capable of playing high definition video smoothly. Moreover video playback on mobile devices drains the battery quickly. We propose a method named selective decoding to increase video playback frame rate and reduce power consumption. Selective decoding is based on the idea that users are usually interested in only part of a video scene. It will be more efficient if only the user's Region of Interest (ROI) is decoded. To enable selective decoding, we analyzed various dependency relationships among macroblocks in MPEG4 Part 2 Simple Profile codec, traced the macroblocks that are needed to decode the ROI, and modified standard decoding process to decode macroblocks selectively based on the trace. Our experiments on Android platform show that selective decoding can improve playback frame rate by up to 193.3% and reduce energy consumption by up to 64.5%.

Keywords: selective decoding, zoomable video, energy saving, Region of Interest playback

1 Introduction

Two issues of mobile video playback remain unaddressed. Firstly, High Definition (HD) video playback is not supported on many mobile devices due to its heavy processing requirement. Secondly, video playback is energy demanding, which makes the mobile devices unusable quickly.

The screen on mobile devices is usually not large, despite its high resolution. Zoom and pan gestures have been widely adopted to help users to view photos, maps and web pages. Recent research indicates zoom and pan are helpful for users when watching videos[1]. When a video is scaled up, only part of the video scene is displayed. The visible part is referred as Region-of-Interest (ROI). The simple scale and display approach is inefficient since the entire video scene is decoded but only part of it is displayed. This observation presents an opportunity to achieve more efficient video playback on mobile devices.

We propose a software approach named selective decoding that improves the efficiency of zoomable video playback. As its name suggests, this approach decodes video selectively based on the ROI captured from user interactions. With the improved efficiency, it is hoped this approach can help to address the two issues mentioned previously.

Our work is based on MPEG4 Part 2 Simple Profile (SP) codec. However this approach should be applicable for other Discrete Cosine Transform (DCT) based video codec. We focus our work on the decoder side of the codec in the local playback context, but the principles and techniques could be extended to encoder and video streaming. A typical MPEG4 SP decoder is illustrated in Fig 1.

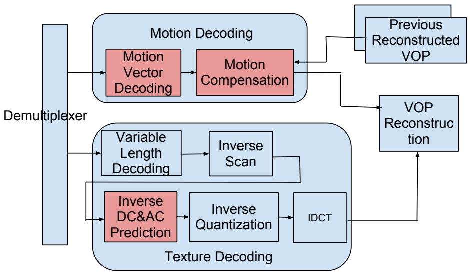


Fig. 1. A Typical MPEG4 Part 2 Decoder

The encoded bitstream is demultiplexed. The coded texture goes through the texture decoding and coded motion is processed by motion decoding. The decoder finally reconstructs the Video Object Plane (VOP). Note that the figure only depicts the core idea of MPEG4 SP decoder, while lots of complexities are not covered.

Various dependencies exist among macroblocks of a MPEG4 SP coded video. From the decoder's perspective, Inverse DC&AC Prediction phase at Texture Decoding, and Motion Vector (MV) Decoding and Motion Compensation at Motion Decoding of a macroblock are carried out with reference to other macroblocks. Because of these dependencies, decoding the MBs of a ROI requires macroblocks outside of ROI to be decoded.

The selective decoding approach pre-processes the video to obtain the dependencies. At decoding, it computes a bitmask based on the dependencies for every frame on the fly, where the selected macroblocks are marked as '1' and others are marked as '0'. The modified decoder then decodes the selected macroblocks accordingly to the bitmask. In this manner, the macroblocks that are necessary and sufficient to present a clear scene at ROI are decoded.

The key contributions in our research are as follows. First, we designed a selective decoding process that decodes user specified ROI efficiently. Secondly, we implemented the selective decoding process as a zoomable video player on Android to demonstrate its practical usage. Finally, we experimentally evaluated the higher frame rate and lower energy consumption benefits of selective decoding over standard decoding.

In the rest of this paper, Section 2 reviews related works. Section 3 analyzes the dependencies in detail and discusses the offline computation of selective decoding. Online computation is covered in Section 4, including the selective mask generation and modified decoder. The proposed approach is implemented for Android and described in Section 5. We evaluate selective decoding for both playback frame rate and energy consumption in Section 6 and finally conclude at Section 7.

2 Related Work

H.264/MPEG4 Advanced Video Coding (AVC) standard includes an extension supporting Scalable Video Coding (SVC)[2]. It enables transmission and decoding of partial bit streams to provide video of lower resolution and/or lower frame rate. SVC requires changes to both encoder and decoder and is not widely adopted yet.

Supporting zoomable video through encoders has been explored by several research groups. Mavlankar etc. studied the optimal slice size for zoomable video in a network streaming context[3]. Feng etc. presented how to produce a video stream with ROI cropping support by constraining the video compression process[4].

More works exist on zoomable video in network context, each with its own focus. Utilizing zoomable video to save bandwidth by refining the encoding process is studied[5]; Zoomable video on peer-to-peer streaming is explored[6]; ROI prediction and tracking for streaming zoomable video is examined[7][8][9]; multiple ROIs support is investigated[10].

In summary, extensive research has been done to enable zoomable video in network streaming context, with focus on video encoding process. To the best of our knowledge, no material has been found on enabling zoomable video from decoder in local playback context through the decoding process.

3 Offline Computation

The offline computation of selective decoding retrieves the dependencies by partially decoding a video. It is essential to understand various dependencies in order to comprehend offline computation. For all subsequent discussions in this paper, we assume the video is in YCbCr420 color space, which means a macroblock contains four luminance blocks and two chrominance blocks.

3.1 Dependencies

Two categories of dependencies can be identified, namely intra-frame dependency and inter-frame dependency. Dependencies are the reason why some macroblocks outside of ROI need to be decoded. By saving certain information, we can reduce the dependencies and improve the efficiency of selective decoding. Below we analyze the dependencies and introduce the methods to reduce them.

Intra-frame Dependency Intra-frame dependency refers to the dependencies among macroblocks within a single frame. There are two sources of intra-frame dependency, including DC&AC Prediction and MV coding.

DC&AC Prediction is performed for I-macroblock when the header field short_video_header is set to '0'. It consists of two steps, namely reference block selection and prediction decoding. The reference block selection step can be illustrated by Fig 2.

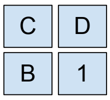


Fig. 2. DC&AC Prediction Reference Block Selection

There are two candidate reference blocks for each block, the immediate left block and the immediate upper block. In addition, the upper left block is also needed in order to determine the reference block. To determine the reference block for block 1 in Fig 2, the following rule is applied,

```
if (|F(B)[0][0] - F(C)[0][0]| < |F(C)[0][0] - F(D)[0][0]|)
   predict from block D
else
   predict from block B
```

F(B)[0][0], F(C)[0][0] and F(D)[0][0] refer to inverse quantized DC value of block B, C and D respectively. The selection rule indicates one block is dependent on three neighboring blocks. In the actual prediction decoding, the decoding block depends only on the selected reference block.

Two out of three blocks are only needed to determine the prediction direction, which is either up or left. A single bit is enough to record this information, with '0' indicating up and '1' referring to left. This reduces the dependency for a block from three to one.

MV of P-frame is differentially coded, which is the other source of intra-frame dependency. At motion decoding, the decoder recovers the MV values based on the decoded base values and residue values obtained from neighboring macroblocks. In selective decoding, the MVs are recorded to trace the Inter-frame Dependency, which is discussed next. Therefore the MVs can be read directly by decoder and no MV prediction decoding is needed. Thus the dependency due to MV prediction decoding is eliminated completely.

Inter-frame Dependency Inter-frame dependency refers to the dependencies among macroblocks at different frames, which is caused by motion compensation coding. In MPEG4 SP, motion compensation decoding only occurs at P-macroblock of P-frame. The inter-frame dependency is illustrated as Fig 3.

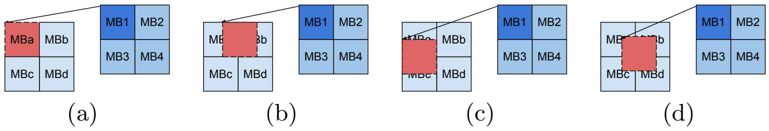


Fig. 3. Different Cases of Motion Compensation Decoding

Motion compensation decoding is performed on MB1 of the current frame, with reference to a region of a previous frame. The number of dependent macroblocks depends on whether the reference region aligns with the macroblock boundary. As shown in Fig 3, MB 1 depends on one macroblock at (a), two macroblocks at (b) and (c), and four macroblocks at (d).

Khiem etc. proposed an approach to reduce dependency due to motion compensation[5]. We adopted their technique in this research. Using the case in Fig 3(d) as an example, the approach is illustrated Fig 4.

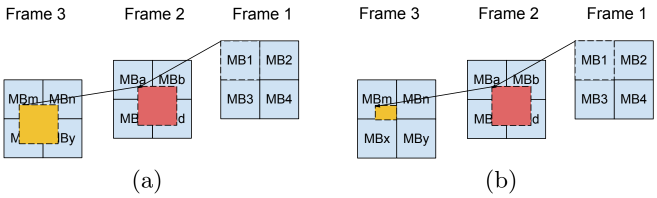


Fig. 4. Dependency Analysis for Motion Compensation

MB1 at Frame 1 depends on four macroblocks at Frame 2. MBa at Frame 2 depends on another four macroblocks at Frame 3. Dependency can be reduced by tracing it at pixel level. The dependency between Frame 1 and Frame 2 remains the same. But we do not really need all pixels at Frame 2 MBa. The region needed at MBa depends on only part of MBm at Frame 3. Therefore, the dependency for MBa between Frame 2 and 3 are reduced from four to one in this example.

3.2 Dependency Files

The offline computation partially decodes a video and records down the dependency information into a set of files named dependency files. The dependency files are generated for each Group of VOP (GOP). For every GOP, the dependency files include the following,

1. GOP record file: this file contains the start and end frame numbers of a GOP.
2. MB start and end position file: this file stores the macroblock start and end bit positions in the video bitstream for every MB of all frames in the GOP. The start and end positions are needed for the decoder to seek the bits for a macroblock.
3. DC&AC Prediction direction file: this file contains the DC&AC Prediction direction. A single bit is used to store the direction for each block. The direction is read directly by the decoder to avoid decoding the macroblocks used in DC&AC Prediction reference selection but not in actual prediction decoding. The direction is also used to trace the intra frame dependency.
4. MV file: this file records the MV values for every macroblock of each P-frame in the GOP and the number of bits for MVs. The selective decoder reads the MV from this file and skip the encoded bits. This file does not only eliminate the MV decoding dependency, but also allows the online computation to trace the inter-frame dependency.

We modified the standard MPEG4 SP decoder to partially decode a video in order to generate the above files.

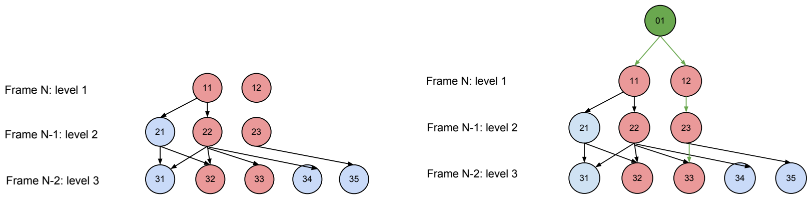
4 Online Computation

Offline computation is carried out once and the dependency files are saved. Every time the video is played, the online computation loads the dependency files, computes a selective mask for each frame and decodes according to the mask.

4.1 Selective Mask Computation

Selective mask indicates the macroblocks that the decoder needs to decode as '1' and the rest as '0'. It considers both inter-frame and intra-frame dependencies. Note that the inter-frame dependency has to be computed first. If intra-frame dependency is computed first, when computing inter-frame dependency, the computation will select some new P-macroblocks and I-macroblocks. Since the intra-frame dependency for the newly selected I-macroblocks is not computed, this leads to decoding errors at those I-macroblocks. The error will subsequently affect the motion compensation decoding at other macroblocks using those I-macroblocks as reference. By contrast, if inter-frame dependency is computed first, the intra-frame computation will select only I-macroblocks because DC&AC prediction coding only applies to I-macroblock. Since inter-frame dependency does not apply to I-macroblocks, the newly selected I-macroblocks won't introduce errors.

Inter-frame Dependency Computation Inter-frame dependency is caused by motion compensation decoding. A MPEG4 SP GOP consists of an I frame followed by a sequence of P frames. The P-macroblocks of every P frame are motion compensated with reference to macroblocks of its previous frame. This means every P frame is dependent on its previous frame. Therefore we compute the inter-frame dependency from last frame back to the first frame of the GOP. The dependencies are shown as Fig 5(a).

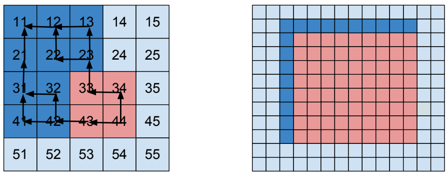


(a) Inter-frame Graph (b) Modified Inter-frame Graph

Fig. 5. Inter-frame Dependency Abstraction

Suppose frame N is the last frame of the GOP, the figure shows the dependencies in three frames. The macroblocks and the dependencies form a graph. By adding a pseudo root node and edges connecting the ROI macroblocks, the graph is transformed to a weakly connected directed acyclic graph shown as Fig 5(b). With this modification, the graph traversal algorithm Depth-First Traversal (DFT) or Breadth-First Traversal (BFT) can be applied. The macroblocks that are visited by the graph traversal are selected, while the rest are not needed.

Intra-frame Dependency Computation Similar to Inter-frame dependency, Intra-frame dependency computation can be abstracted to graph traversal problems.



(a) Graph (b) Optimization

Fig. 6. Intra-frame Dependency and the Optimization

In Fig 7(a), macroblock 33, 34, 43 and 44 are ROI macroblocks. The dependency graph due to DC&AC prediction coding is depicted. Because the dependency direction is always pointing up or left, the graph rooted at a macroblock is always formed by one of the graphs rooted at the first row and first column macroblocks of the ROI and some macroblocks within the ROI. Therefore, an optimization is to apply graph traversal algorithms only on the macroblocks at upper and left edges of the ROI, and select all macroblocks within ROI. This is illustrated as Fig 7(b).

4.2 Select the Bits

In order to decode selectively, a mechanism is needed for the decoder to select the bits for selected macroblocks. Two approaches can be used to select the bits, namely bitstream reconstruction and bit seeking. In bitstream reconstruction, a new video bitstream is constructed according to the selective masks and the macroblock start and end positions. The newly constructed bitstream consists of only the macroblocks selected in selective masks. At bit seeking approach, we instruct the decoder to seek to the start position of next selected macroblock at decoding. The second approach avoids the additional memory allocation for the new bitstream therefore it is the preferred approach in our research. However, for applications in the next streaming context, the first approach may provide better bandwidth utilization as the shorter reconstructed bitstream can be transmitted.

5 Implementation

We implemented the techniques and processes described in previous sections on Android platform[11] as a zoomable video player. The architecture of the player is depicted as figure below.

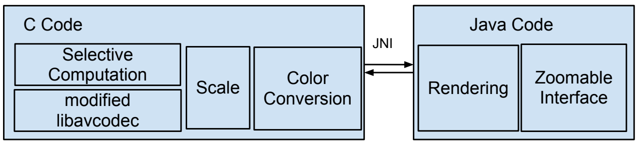


Fig. 7. Zoomable Video Player Implementation

Based on the open source MPEG4 SP codec from libavcodec of ffmpeg 0.7[12], we added the selective decoding functions. At the time of implementation, the ffmpeg codec was not optimized to run on Android platform, especially the scale and color conversion process. We adopted the code from another two open source projects, with scale from libyuv[13] and color conversion from Google Chromimum project[14].

We implemented the rendering and zoomable interface in Java with Android SDK. The gesture detector detects a user's pinch or pan gesture, and translates the gesture detected as zoom scale and pan scale. We then compute a ROI from these scales based on what user can see on the phone screen.

6 Evaluation

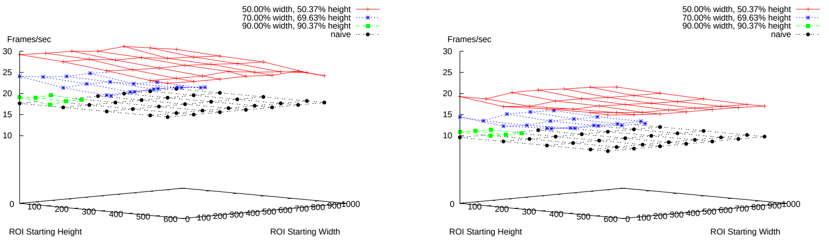
The purpose of selective decoding is to achieve more efficient zoomable video playback. We evaluate the efficiency of selective decoding from two aspects, video playback frame rate and energy consumption.

We used a Samsung Galaxy S2 phone for experiment. The device has a dual core processor of 1200 MHz clock rate each and 1024 MB RAM, with Android 2.3.6 running. Two videos of 1080p are used for evaluation, one having little motion while the other containing constant object motion. We refer them as video A and video B respectively.

6.1 Frame Rate

Selective video playback frame rate is affected by both ROI size and position. We designed experiments to examine the influence of each.

Different ROI Positions Different regions of a video frame may contain different amount of motions and dependencies, therefore the ROI position can affect the amount of processing at decoding and subsequently the video playback frame rate. We fix the ROI size, and then move the ROI from the upper left corner to the lower right of the video frame. The tests are repeated for three different ROI sizes, with the ROI width and height set as 50%, 70%, and 90% of the original video's width and height.



(a) Video A

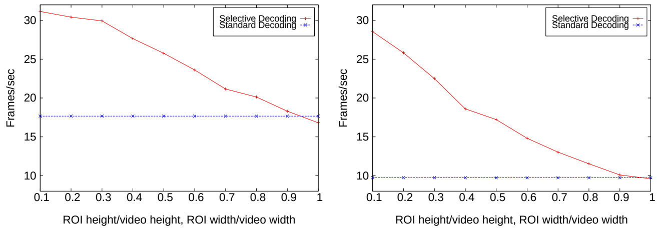
(b) Video B

Fig. 8. Frame Rate with 90%, 70%, and 50% of ROI at Different Positions

In Fig 8, each 3D interface indicates the frame rate for a ROI size. The intersection in a surface indicates the start position of a particular ROI size.

Looking at each 3D interface, the frame rate tends to decrease when ROI starting width and/or starting height increases. This is because the intra-frame dependencies increase towards the lower right, and more dependencies cause more macroblocks to be selected and decoded. However, there are exceptions to this decrease trend, which are probably caused by different amount of motions at different ROI positions. Compare different 3D interfaces at a single figure, it is clear that the ROI size has a more significant effect than ROI position and selective decoding outperforms standard decoding. Compare Fig 8(a) with (b), the frame rate for video A is higher than video B. This is expected because video A has less amount of motion than video B.

Different ROI Size Previous experiment already reveals that ROI size has significant influence on video playback frame rate. This experiment examines the affection of ROI size further. We position the ROI at the center of the video frame and vary the ROI height and width from 10% to 100% of original video height and width.



(a) Video A

(b) Video B

Fig. 9. Frame Rate with 10% to 100% ROI Centered

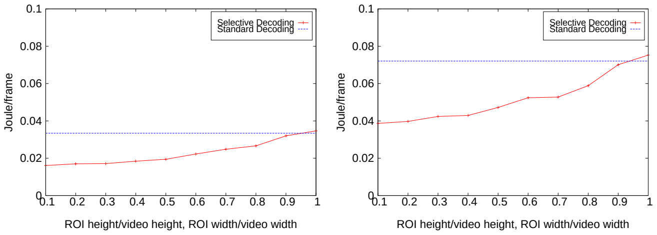
Looking at either Fig 9(a) or (b), frame rate decreases as ROI size increases. Selective decoding achieves higher frame rate at ROI size smaller than 90%. At 10% ROI, the frame rate is improved by 76.3% and 193.3% for video A and B respectively. The curves at different figures differ due to different amount of motion at center of the video frame.

6.2 Energy Consumption

Energy consumption is the other important aspect of our evaluation. Two experiments are done with different focuses.

PowerTutor Measurements Battery energy is mainly consumed by display and CPU at video playback. PowerTutor[15], an Android power measurement tool, is capable of measuring power consumed by different hardware components for a user specified app. In this experiment, we vary ROI size from 10% to 100%. The frame rate is controlled so that both selective decoding and standard decoding always play at same rate. This is essential for a fair comparison because the CPU display power is dependent on display time.

For display measurements, we found selective decoding and standard decoding consume almost same amount of energy. However, this is not the case for CPU power consumption, which is shown as Fig 10.

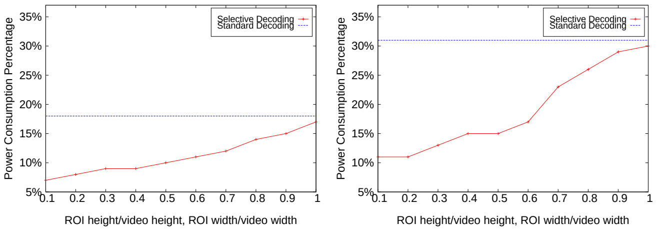


(a) Video A (b) Video B

Fig. 10. CPU Power Consumption Per Frame

Looking at each figure individually, CPU power consumption per frame at selective decoding increases when ROI size increases. At ROI size 90% or less, selective decoding consumes less CPU energy than standard decoding. Compare Fig 10(a) with (b), video playback for heavy motion video tends to consume more power because of more motion compensation decoding.

Power Drain Experiment PowerTutor measurements show selective decoding can save energy, mainly by reducing CPU power consumption. However, PowerTutor measurement is obtained through offline-training models[15] and may not be accurate for all phones. We designed another experiment to measure the power consumption.



(a) Video A (b) Video B

Fig. 11. Percentage of Power Drained

In this experiment, we place ROI at the video frame center and play the video repeatedly with constant frame rate. Based on the processing power and video, we set the frame rate for two videos as 15 FPS and 8FPS respectively.

The percentage of power drained is recorded for comparison. Before each test, we fully charge the phone battery to 100% and disable all background activities including Wi-Fi, Bluetooth, GPS, etc.

Looking at each figure individually, selective decoding consumes less power. At 10% ROI size, the battery consumption is reduced by 61.1% and 64.5%. Because we set the frame rate differently for videos, the comparison for different videos is not fair and therefore avoided.

In summary, selective decoding proves to be efficient in terms of playback frame rate and battery energy consumption when ROI size is below certain threshold.

7 Conclusion and Future Work

Based on the fact that users are only interested in part of a video scene at some cases, we designed a software approach named selective decoding to reduce the battery power consumption and increase the frame rate for HD video playback on mobile devices. Selective decoding is based on analyzing and tracing various intra- and inter-frame dependencies among macroblocks. By doing so, we compute a selective mask which indicates the macroblocks needed to present a clear scene in a user requested ROI and the modified decoder can then decode selectively according to the mask.

Selective decoding presented illustrates the idea of achieving more efficient zoomable video playback by tracing the dependencies among macroblocks. There are many possible future work can be done. Firstly, the dependency file storage overhead is large. We store the dependency information as plain values in binary format. Better storage scheme and compression can be applied to reduce storage overhead. Secondly, the dependency file generation could be done online, which requires optimizing the process and integrating it with selective mask computation. Thirdly, expanding selective decoding to encoder may bring some benefits. The encoder can generate dependency files and control the amount of dependencies among macroblocks. Lastly, selective decoding at decoder may help to improve the existing research on zoomable in network streaming context, which deals with encoders mostly.

References

1. Khiem, N.Q.M., Ravindra, G., Ooi, W.T.: Towards understanding user tolerance to network latency in zoomable video streaming. In: Proceedings of the 19th ACM international conference on Multimedia. MM '11, New York, NY, USA, ACM (2011) 977–980
2. Schwarz, H., Marpe, D., Wieg, T.: Overview of the scalable video coding extension of the h.264/avc standard. In: IEEE Transactions on Circuits and Systems for Video Technology In Circuits and Systems for Video Technology. (2007) 1103–1120

3. Mavlankar, A., Baccichet, P., Varodayan, D., Girod, B.: Optimal slice size for streaming regions of high resolution video with virtual pan/tilt/zoom functionality. In: Proc. of 15th European Signal Processing Conference (EUSIPCO. (2007)
4. Feng, W.C., Dang, T., Kassebaum, J., Bauman, T.: Supporting region-of-interest cropping through constrained compression. ACM Trans. Multimedia Comput. Commun. Appl. **7**(3) (September 2011) 17:1–17:16
5. Ngo, K.Q.M., Guntur, R., Ooi, W.T.: Adaptive encoding of zoomable video streams based on user access pattern. In: Proceedings of the second annual ACM conference on Multimedia systems. MMSys '11, New York, NY, USA, ACM (2011) 211–222
6. Mavlankar, A., Noh, J., Baccichet, P., Girod, B.: Peer-to-peer multicast live video streaming with interactive virtual pan/tilt/zoom functionality. In: in Proc. of IEEE International Conference on Image Processing (ICIP
7. Mavlankar, A., Varodayan, D., Girod, B.: Region-of-interest prediction for interactively streaming regions of high resolution video. In: In Proc. International Packet Video Workshop. (2007)
8. Shimoga, K.B.: Region-of-interest based video image transcoding for heterogenous client displays. In: Packet Video 2002. (2002)
9. Fan, X., Xie, X., qin Zhou, H., ying Ma, W.: Looking into video frames on small displays. In: In Proc. of ACM Multimedia 2003, Press (2003) 247–250
10. Bae, T.M., Thang, T.C., Kim, D.Y., Ro, Y.M., Kang, J.W., Kim, J.G.: Multiple region-of-interest support in scalable video coding. ETRI Journal (2006) 239–242
11. Android: Android official website. http://www.android.com/ (05 2012)
12. ffmpeg: Ffmpeg. http://ffmpeg.org/index.html (05 2012)
13. libyuv: libyuv. http://code.google.com/p/libyuv/ (5 2012)
14. chromium: chromium. http://code.google.com/p/chromium/issues/detail?id=71403 (5 2012)
15. Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R.P., Mao, Z.M., Yang, L.: Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. CODES/ISSS '10, New York, NY, USA, ACM (2010) 105–114