A video codec analyzer enables users to quickly and easily check the conformance and encoding algorithm performance. Commercial analyzers are usually available after the standards are mature or finalized. They decode the bitstream and visualize the encoding parameters through a graphical user interface. Nowadays, MPEG and VCEG are actively developing the next-generation video coding standard that will outperform H.264/AVC. There exist many competing solutions and the syntax elements evolve rapidly. To assist the standard development, we design an analyzer that accepts the statistical data as input and make it independent of the syntax design. The analyzer is flexible enough to be used by different solutions. In this paper, we illustrate our design principles using the Key Technology Area as an example and provide a few screenshots.

Index Terms— video coding, software tools

1. INTRODUCTION

A video codec analyzer is a powerful tool in designing video coding algorithms. Instead of taking time to read the log files containing the intermediate coding parameters, such as mode and motion vectors of every macroblock, video codec developers can gain more insights by looking at the graphic display of this information with the analyzer. This enables the users to quickly and easily check the conformance and encoding algorithm performance. Moreover, it can be very helpful in assisting and guiding the encoding algorithm design.

A typical video codec analyzer takes the encoded bitstreams as input and decodes it before visualizing the encoding parameters through a graphical user interface (GUI). Commercial video analyzers have been developed in the past, such as Sencore CMA1820 [1], Elecard Stream Analyzer [2], etc. These analyzers are designed for syntax analysis and presentation of the encoding parameters in a visual form. They support MPEG-2, H.264/AVC [3], and VC-1 video formats along with other audio formats.

While various analyzers offer different GUI, they are all capable of displaying the decoded video, high-level syntax, and encoding parameters. For syntax conformance checking, the syntax information is often visualized in a tree structure, which enables the developers to easily verify new encoder compliance before deployment. In addition, an analyzer allows users to quickly identify the potential problems in the encoder algorithms because well-tuned encoding parameters are often highly correlated with video content. It becomes easier to identify inconsistencies and confirm the effectiveness of the algorithm one is interested in by displaying the encoding parameters and the video content in the same window. For example, a smooth area should choose a smaller QP than a textured area to improve visual quality. When we overlay the QP for each macroblock (MB) on top of the picture, we can clearly see whether the QPs is adaptive to the texture of the MB.

Fig. 1 illustrates a typical video bitstream analyzer for H.264 [1, 2]. It requires a H.264 standard-compatible bitstream as its only input. Conformance check is performed and warning or error message is provided if there exists any problem in the syntax. The bitstream is then partially or fully decoded to obtain coding parameters such as the motion vectors, mode types, QP values, and/or residue sample values. Upon the user’s request from the GUI, the analyzer displays the corresponding data on the monitor. For example, a user can request through the GUI to display the motion vectors for a frame.

As shown in Fig. 1, these analyzers contain an embedded video decoder that decodes the bitstream before visualizing it. They require the input bitstream to be completely compliant with the decoder syntax specifications. This is appropriate for designing standard-compatible encoders after the standard has been finalized, but it is not desirable at the standard development stage. During the standardization process, many proposals compete for adoption, so the coding tools and syntax definitions change frequently. An ideal analyzer should be robust and flexible enough to accommodate different solutions. In this paper, we propose a novel video coding analyzer that decouples from the embedded decoder and instead takes the coding parameters as the input. Then, the analyzer only parses the coding parameters and is syntax-insensitive. Thus, compared to the existing stream analyzers, our proposed an-
The paper is organized as follows. In Section 2, we explain how to generate the coding parameter files at the encoder (or decoder), and how the analyzer parses the data and provides a graphical display. Example snapshots of the analyzer are presented in Section 3 with Key Technology Area (KTA) [4] for H.265. We conclude our work in Section 4.

2. THE PROPOSED ANALYZER

Fig. 2 illustrates the framework of the proposed analyzer. It takes the coding parameter files and the reconstructed YUV video sequence as inputs before displaying them on the GUI. Since the coding parameter files and the YUV can be generated at either the encoder or the decoder without any knowledge of the syntax definitions, the presence of a decoder becomes optional in our analyzer (dashed box in Fig. 2), while it is mandatory for the H.264 bistream analyzer (Fig. 1).

In the following, we will explain how to generate the coding parameter files at the encoder (or the decoder) before describing the functionalities of each module of the analyzer.

2.1. Coding Parameter Files

The coding parameter files from the encoder/decoder are taken as inputs of our analyzer. Given the different proposals [5] for next-generation video coding standards, we use the KTA software as our base codec and consider others to be similar. The essential parameters include coding modes, motion vectors, reference indexes, block partitioning for each coding unit (e.g., a macroblock), etc. Other parameters are new and may provide higher coding efficiency, such as the extended MB size and the quad-tree adaptive loop filter (QALF) [6] for KTA. The exact parameter values may still differ among competing solutions. For example, the largest extended block size is 64 in KTA, and it may be 128 in another platform. Our analyzer is flexible enough to accommodate such differences.

<table>
<thead>
<tr>
<th>Frame-level parameter</th>
<th>frame type, ( Q_P ), resolution, extended MB size, QALF size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended MB-level parameter</td>
<td>MB mode, MB-level ( Q_P ), CBP, filter flag</td>
</tr>
<tr>
<td>MB partition</td>
<td>extended MB partition, filter block partition</td>
</tr>
<tr>
<td>Sub-block level parameter</td>
<td>motion vector, reference index, intra prediction mode</td>
</tr>
</tbody>
</table>

Table 1. Coding parameters are classified into four categories.

The analyzer offers more flexibility and can accommodate the constantly changing syntax definitions at the development stage of a new video coding algorithm or standard.

To design an analyzer that is easily extendable, we study the properties of all these parameters and classify them into four categories, which are summarized in Table 1. Parameters of the same category are output at the encoder and parsed at the analyzer following the same methodology, which is explained in the following subsections. When a new coding tool is introduced, we can classify the related parameters into the categories and output them in the coding parameter file in a similar way.

2.1.1. Frame-level parameter

The frame-level information includes the frame type, \( Q_P \), resolution, extended MB size, QALF block size, etc. Such information is global to the frame and has a large impact on the visual quality. These data are written into a text file.

2.1.2. Extended MB-level parameter

It appears that the next-generation video coding standard will still use the block as the basic coding unit within a frame. In the case of KTA, the extended MB is the basic coding unit. There are a few coding parameters at the extended MB level. For example, the INTRA/INTER coding mode is distinctive for each MB (in B frames it supports sub-MB direct modes, thus the minimum unit is an 8x8 block). We store the coding mode index in the original scanning order. It can be easily visualized for every MB or sub-MB, either through different colors or displaying the indexes. Other parameters in this category include the MB-level \( Q_P \), the coded block pattern (CBP), the block-adaptive filter flag, etc.

2.1.3. Partition pattern within extended MB

In KTA, the partition pattern within each super block can be different and needs to be input to the coding analyzer. Since the number of possible partitions is very large and it varies with the super block size, it is not realistic to represent the partition for a super block by an index. Instead, we use a hierarchical representation. Let “1” stand for the current block having four equal-sized sub-partitions and “0” for the current block not being further partitioned. Assuming we use 64x64 extended MBs in KTA, the partition in Fig. 3(a) is represented by a binary string “1 1 0 0 0 0 0 0 0 0”, and Fig. 3(b) by “0”.

Another example is the QALF, which uses a quadtree-based partition over each MB (the size of which needs to be signaled at a frame level) and the partition pattern needs to be signaled in the hierarchical fashion similarly.
In our work, the data in this category are saved in a text file. Each line in the text file represents partition parameters for an extended block.

2.1.4. Sub-block level parameter

After the partition pattern within an extended MB is signaled, the sub partition-level coding parameters (e.g., the motion vectors and reference indexes for each sub-block) have to be represented. Take the motion vectors in H.264 and KTA as an example, each sub block has two sets of motion vectors: the list 0 and list 1 motion vectors. The vectors are stored for each block and can be easily visualized. Other parameters that belong to this category include the INTRA prediction mode.

In our work, the data in this category are saved block by block in a binary file.

2.2. Analyzer Modules

As shown in Fig. 2, our analyzer consists of two modules: a data parser and a GUI. In this section, we explain the functionality of each module.

2.2.1. Data parser

The data parser reads the coding parameter data files and the YUV sequence. As the coding parameter data falls into different categories as defined in Section 2.1, we have corresponding parsing functions to extract coding parameters from the formatted data files, which are then organized into a matrix or a bitmap that is ready to be displayed. For instance, the partition patterns for the extended MBs are organized into a logical bitmap of the same size as the video frame; the lines where partition happens are indicated by logic value TRUE (otherwise, FALSE is indicated). For the coding mode parameter, different modes correspond to different intensity values in the bitmap.

Another functionality of the data parser is to compute the distribution/statistics of the coding parameter data. For instance, it can collect statistics such as the frequency of each mode being selected and the percentage of different MB partition patterns within a frame. Such knowledge is greatly appreciated during the algorithm design. Take the mode distribution as example, the user can design a more efficient codebook based on the mode distribution with the frequently selected mode using a short codeword to save bit rate.

2.2.2. GUI

The GUI displays graphical objects based on the extracted data from the data parser. It also generates statistic plots and offers a set of interactive tools for users. In our work, the GUI presents the following information:

- coding parameters in Table 1
- statistical distribution of coding parameters
- the MB grid
- the Y/U/V channels

Fig. 4. An example GUI screenshot.
The “on” and “off” of each information display can be controlled independently, and the information can be overlayed to form a comprehensive display, which can be continuously played as a video. The users can access an arbitrary frame by entering the frame number, get the detail data by clicking on a particular MB, or take a snapshot of the analyzed results for record. All these iterative functionalities make the proposed analyzer a convenient and effective tool in assisting and guiding the design of video coding algorithms.

3. EXAMPLES

We have output the coding parameter files and the decoded YUV from the KTA software and performed analysis with the proposed analyzer. Please note that our analyzer can be easily extended to analyze other competing video coding softwares. In this section, we illustrate how to use our analyzer by a few screenshots.

3.1. Main GUI window

Fig. 4 shows a snapshot of the main GUI window, which is an example of the KTA mode/partition overlaying on the YUV. The control bar at the top provides easy access to all functionalities. The preview window at the right not only allows easy navigation among frames but also gives a simple preview of the neighboring pictures. The text window at the bottom shows the frame-level information together with the mode distribution. When a block is selected, its pertinent motion information is detailed in the text window.

3.2. Parameter display

Fig. 5 shows an example of the KTA QALF flag. Note that the overlay transparency can be adjusted from opaque to transparent. Fig. 6 illustrates the mode statistical distribution plot generated by our analyzer. The distribution is helpful in designing the codewords.

Fig. 6. An example mode distribution plot.

4. CONCLUSIONS

MPEG and VCEG are now actively developing the next-generation video coding standard. A comprehensive video analyzer that visualizes the coding parameters is highly attractive to the developers. In this paper, we have introduced a video coding analyzer that excludes an embedded decoder and does not rely on syntax definitions. It can display the video as well as the coding parameters, including motion vectors, modes, partitions, filter regions, QPs, etc. It can also calculate and display the statistics of the input parameters. By integrating the video content and the coding parameters into one window, the analyzer provides a comprehensive review of performance of the coding tools. Moreover, since it is insensitive to syntax elements, the analyzer can be easily extended for other video coding softwares. Therefore, this is a convenient and powerful tool for the development of the next-generation video coding standards.

5. REFERENCES