RATS: Adaptive 360-degree Live Streaming

Trevor Ballard
University of Central Florida
ballardt@knights.ucf.edu

Carsten Griwodz
University of Oslo
griff@ifi.uio.no

Ralf Steinmetz
Technische Universität Darmstadt
ralf.steinmetz@kom.tu-darmstadt.de

Amr Rizk
Technische Universität Darmstadt
amr.rizk@kom.tu-darmstadt.de

ABSTRACT

Recent approaches to tiled 360° adaptive bitrate video streaming present significant bandwidth savings at little risk of stalling when only parts of the video, e.g., the current and predicted viewport, are transferred in high quality while the rest of the 360° video tiles are transferred in a lower quality. While this is currently feasible for video on demand scenarios, it poses a difficult problem for 360° live streaming as naive methods produce a considerable overhead owing to the lack of tiling support in existing hardware encoders.

In this demo, we show Real-time Adaptive Three-sixty Streaming, or RATS, where we utilize GPU-based HEVC encoding to tile, encode, and stitch 360° video at different qualities in real-time. We show measurement results for the encoding speed, amont of output data, and output quality for different tiling configurations. While we observe an increase in both encoding time and output file size with the number of desired tile columns, we also see that real-time encoding is ensured for all considered tiling configurations.

CCS CONCEPTS

• **Information systems → Multimedia streaming**; **Multimedia content creation**; • **Computing methodologies → Graphics processors**;

KEYWORDS

GPU video encoding, HEVC, 360° video, live streaming

ACM Reference Format:
Trevor Ballard, Carsten Griwodz, Ralf Steinmetz, and Amr Rizk. 2019. RATS: Adaptive 360-degree Live Streaming. In *MMSys '19: ACM Multimedia Systems Conference - Demo Track, June 18–21, 2019, Amherst, MA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3304109.3323837

1 INTRODUCTION

Hardware HEVC encoders have recently appeared on consumer-grade GPUs, opening the door to mass live 360° video streaming. Traditional methods of adaptive bitrate (ABR) streaming, however, are ill-suited to these ultra-high resolution videos, wasting enormous amounts of bandwidth on portions of the video outside the viewport. A body of work on tile-based ABR streaming for UHD video has recently appeared in which the quality requested for each section of a video corresponds to the current or predicted viewport, allowing end-users to view the most important parts of a video in high-quality without stalling [2, 5, 7]. This is straightforward for pre-rendered videos, but is more complicated in live streaming because each quality requires a different encode, introducing some delay in the availability of each new segment. We are not aware of any commercially available hardware encoder that supports user-configurable tile sizes. One may feed each tile to the encoder as a separate video and stitch them afterwards, but the cropping required on each frame introduces considerable overhead.

To combat these challenges, we introduce RATS, a GPU-based HEVC encoding platform to tile, encode, and stitch 360 video at multiple qualities in real-time. The source video stream is divided into columns which are stacked on top of one another. The rearranged video image is then fed to the hardware encoder with slice boundaries set at the edges of the source video. This process occurs twice for each frame, once with a low average bitrate and once with a high one. When the video is reconstructed during playback, each tile is an independent region, allowing us to pick where every tile (belonging to either bitstream) should appear in the final video. For this demo, we implement stitching on the server side, with the expectation of deploying the code in 5G Edge Computing. We showed our work on client-side stitching earlier [4].

The rest of this paper is organized as follows. In Section 2, we provide background on HEVC and the NVENC hardware encoder. In Section 3, we describe the proposed system in detail. In Section 4, we sketch out a full web-based video streaming platform centered around RATS. In Section 5, we provide an evaluation of the system. Finally, Section 6 concludes the work.

2 HEVC AND NVENC

HEVC is the state-of-the-art video compression standard; however, the encoding process is computationally expensive and not easily parallelized. Instead, encoders divide each frame of the video into independent regions and encode them separately, joining their edges during playback. Two functionally identical versions of this concept, slices and tiles, exist within the HEVC standard, but slices are limited to horizontal strips while tiles are rectangular CTU-based areas of any size. Tiles are supported by most decoders, but only a few available software encoders support tile-based encoding with user-defined tile sizes. These encoders are far too slow for live streaming.

HEVC bitstreams consist of sequential NAL (Network Abstraction Layer) units with header and data components. Each slice or tile in the frame is encoded as a single NAL unit, where the encoder

MMSys'19, July 2019, Amherst, Massachusetts USA

ACM ISBN 978-1-4503-6297-9/19/06.
https://doi.org/10.1145/3304109.3323837

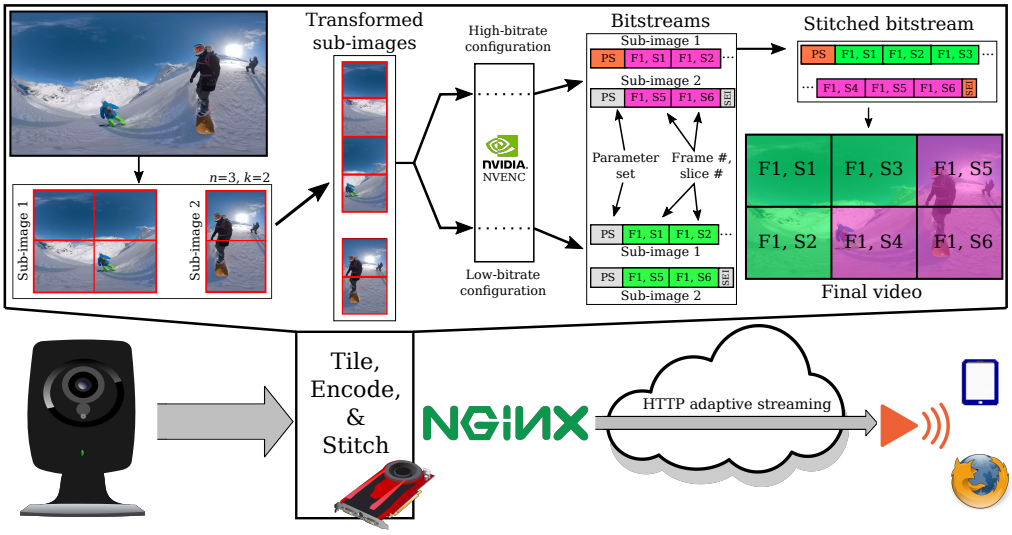


Fig. 1: Adaptive 360° Live Streaming. In this demo we show hardware encoding that allows us to stitch single tiles of the 360° video in different quality levels (see expanded pipeline). The lower part of the figure shows the demo setup in a streaming infrastructure that makes use of HTTP adaptive streaming.

does never create motion vectors that cross slice or tile boundaries. Visible boundaries are prevented by a de-blocking filter. Because tiles are independently-decodable, we can rearrange a set of given tiles by making few NAL header modifications. This process that is known as stitching, is widely used in video conferencing [1, 3, 8].

NVENC is a state-of-the-art hardware HEVC encoder present on newer Nvidia GPUs. Interactions with NVENC occur through the NVENCODE API[1]. It is possible to swap out encoder configurations, which contain parameters and previous image data, between frames, allowing one to encode multiple videos simultaneously.

3 RATS IMPLEMENTATION

To get around the lack of tiling in NVENC, we divide the RATS pipeline into an encoding component, in which a raw video frame is manipulated and converted into low- and high-quality bitstreams; and a stitching component, in which slices from these bitstreams are converted to tiles and arranged in the desired manner.

Details on the installation and usage of RATS, as well as an overview of the source code, can be found in the git repository[2].

3.1 Encoding

NVENC cannot perform tiling, but it is capable of slicing videos in one of several ways based on the `sliceMode` parameter. When `sliceMode=3`, each frame is cut into `sliceModeData=`n equal slices. Slices do not permit vertical boundaries to be specified, but the edges of the image act as natural slice boundaries, allowing us to effectively create tile columns by placing the desired column edges at the image borders. We therefore manipulate the input images by vertically cutting them into n chunks of equal size and stacking these chunks on top of one another, then setting `sliceModeData=`$k \cdot n$ for some $k \in \mathbb{N}$, so that the slice boundaries align with the top and bottom boundaries of the original image. This process is demonstrated for $n = 3$ and $k = 2$ in Fig. 1. Note that k corresponds to the desired number of tile rows, and the CTU height of the image must be divisible by k. Similarly, n corresponds to the desired number of tile columns, and the CTU width of the image must be divisible by n to keep stacked columns the same width. Here, we crop the input video if necessary, but one could, e.g., resize the input video.

In a YUV420p video, each frame consists of a luminance component followed by two chrominance components. The image transformation is performed on each component via memcpy. For each desired tile column, we begin at the first pixel row and move down, calling memcpy with a length equal to the width of a tile column on each pixel row and placing the contents in a new array, before moving on to the next tile column once we hit the bottom of the image. The result of this process will be an image in which the desired tile columns are stacked on top of one another (see Fig. 1).

To interact with NVENC, we use a custom version of the libavcodec API provided by FFmpeg [3]. The high- and low-bitrate configurations are distinct instances of the AVCodecContext struct, which are initialized before the encoding process begins. During the process, each transformed frame is passed to NVENC twice, once for either bitrate, to obtain the low- and high-quality bitstreams.

3.2 Stitching

The bitstreams from the encoding step will contain $(k \cdot n) + 1$ NAL units per frame, one for each tile and a single SEI at the end, but these will still be arranged in a vertical stack. To pick tiles from either quality and place them in the final image, we must modify the image dimensions in the SPS, the tile configuration in the PPS, and the tile CTU address within each tile header. Furthermore, a bit sequence is a byte-aligned sequence (BAS) if its first bit occurs

[1] https://developer.nvidia.com/nvenc-programming-guide
[2] https://github.com/ballardt/nvenc-live
[3] https://www.ffmpeg.org

on a byte boundary and its length is a multiple of 8. NAL borders are represented by the BAS `0x000001`. To prevent this sequence from occurring by chance within a NAL, the BAS `0x03`, referred to as the emulation prevention byte (EPB), may be inserted after any BAS `0x0000` which is not part of a NAL border. The EPB does not otherwise impact the NAL parsing process, and decoders will simply skip over it.

If the stitching process modifies the number of bits in the NAL header, e.g by inserting a new field or changing an unsigned Exponential Golomb coded value, any EPBs appearing after the point of change will likely cease to be BASes. The decoder will then consider these bits to have semantic value, resulting in a corrupt or incorrect header. We must also ensure that the stitching process does not introduce any new BAS `0x0000` without a trailing EPB. To tackle both of these issues at once, we discard all EPBs in the original NAL header before making any modifications, then check for any BAS `0x0000` afterwards, inserting EPBs as necessary. This procedure is not performed on NAL data because it is not modified, and byte alignment will be performed at the end of the header.

After the last semantic bit in a NAL header or body, byte alignment is performed by appending a `1` followed by as many `0`s necessary to complete the byte. As is the case with EPBs, if the size of the NAL header changes during modification, we will likely have an incorrect number of trailing `0`s. To redo the byte alignment, we find the last `1` in the original NAL component and remove it and everything after it, effectively undoing the previous byte alignment. We then perform byte alignment after our modifications have been made.

The stitching process uses the Boost C++ dynamic_bitstream API [4], which allows one to read bytes into a vector of bits, manipulate these bits, then convert the bit vector back into raw bytes. The high- and low-bitrate configurations are identical except for the specified bitrate, so the output from NVENC is highly predictable. Navigating to the right spot in the bitstream and making our changes is thus a straightforward process.

We are creating entire videos with GOP lengths of 16. This implies an average quality change duration of 8 seconds, which can be reduced by reducing the GOP length. While adapting much slower than our earlier work [4], it can work with arbitrary clients. NVenc does not generate SI or SP frames for faster transitions.

3.3 Hardware Limitations

In this work, we use the Nvidia GTX 1080Ti, a powerful consumer-grade GPU. As demonstrated in Section 5, this GPU meets our real-time encoding speed requirements; however, there are two limitations that we must consider. First, the maximum input image size is 8192×8192 pixels. Because we stack tile columns before encoding, the height limit is quickly reached. To get around this, we divide the image into sub-images whose stacked columns do not exceed 8192 pixels, and encode each of these sub-images independently at either quality (see Fig. 1). More tile columns thus require additional encodes, but NVENC is fast enough to keep the total encoding time well below our target.

Additionally, we must change the stitching process to handle sub-images. Note that the position of each tile within a sub-image may change in the final image. For example, consider a video with 6 tile columns. From left to right, if the first 4 columns constitute one sub-image and the final 2 columns constitute another, the CTU offsets of all tiles except those in the first row of the first sub-image will change, and the number of required bits to record the offset will change as well. Furthermore, the first tile in the second sub-image will not contain a CTU offset, so this value must be inserted.

A second limitation is that the GPU driver does not allow more than 2 simultaneous context configurations during the encoding process, which would normally prevent the use of sub-images. Fortunately, this is a limitation imposed by the driver, and it is possible to remove this limit by sending a particular bytestream to the GPU. A script to do so is provided in nvidia-patch [5], which we used here.

4 RATS DEMONSTRATION

We will demonstrate RATS using a single laptop on the server side with a built-in consumer-grade GPU. We will use stored video and live recording from a directly attached camera. Our demo will focus on the visual differences that are incurred when the number of tiles, as well as the tile pattern of encoding profiles, changes.

The visitor of the demo will be able to observe the changes on an arbitrary computer using any browser that supports JavaScript Media Source Extensions and HEVC decoding. The infrastructure used to deliver the encoded video via HTTP adaptive streaming is comprised of open-source software: Nginx[6], Kaltura nginx-vod-module[7] and MediaElement.js[8].

The solution that we present in this demo using MediaElement.js relies on tile stitching before delivery of the video to an end-system. This is not the only possible approach, as DASH provides the extension DASH-SRD (spatial relationship description), which allows a client to individually access tile streams, synchronize them, and make independent quality choices for each of them [6]. However, existing DASH-SRD players make dynamic adaptation choices internally, whereas our goal is to fully control tile configurations and demonstrate visual differences between them.

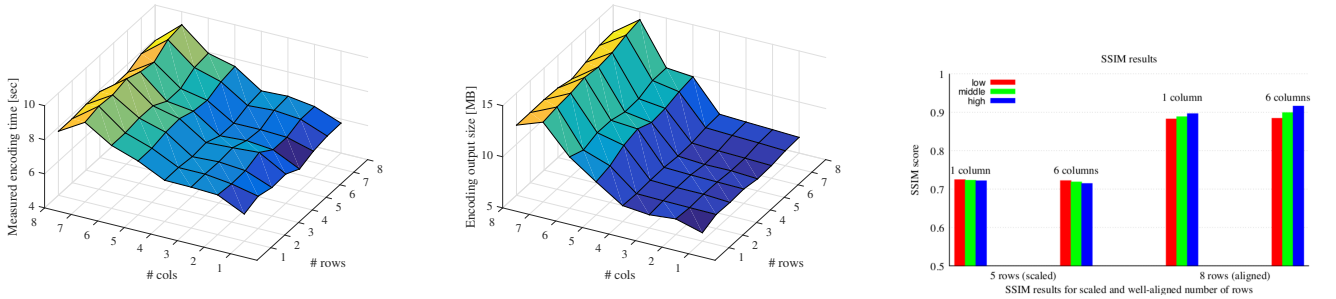
5 EVALUATION

To evaluate RATS, we analyze its performance across multiple tile configurations. In particular, we consider the encoding speed, output file size, and the quality of the final image. Examining Figs. 2(a) and 2(b), we see that the number of tile columns is the primary factor affecting encoder performance. Note that the number of columns dictates how many sub-images are necessary, each of which requires two more NVENC encodes per frame, multiple stitching operations, and column stacking of the source image. In contrast, the hardware encoder supports tile rows using slice mode 3. Slice mode 3 incurs additional computation costs only when alignment rules force us to crop the source image.

Fig. 2(a) confirms that RATS easily satisfies our real-time requirement, finishing the encode and stitch process in one-third of the total video time in the worst case tested. The small jump from 1 to 2 columns is due to a required source image transformation, both

[4] https://www.boost.org

[5] https://github.com/keylase/nvidia-patch
[6] https://github.com/nginx/nginx
[7] https://github.com/kaltura/nginx-vod-module
[8] https://www.mediaelementjs.com



(a) Real time required to encode a tiled configuration for 30 seconds of 3840x2048 video. The encoding process is CPU-bound because tiles a rearranged by the CPU.

(b) Sizes of the encoded tiled videos. The video size grows rapidly when additional encoding contexts are required.

(c) SSIM scores between input and tiles video. "Middle" quality uses a checkerboard pattern. Hardware alignment rules require padding for 3, 5, 6 and 7 rows; 5 and 8 rows are given as representative examples.

Fig. 2: Evaluation results on a 30 second video with a resolution of 3840×2048 pixels. High and low bitrates are 1.6Mbps and 0.8Mbps, corresponding to 480p and 360p video, respectively.[9]

before encoding and afterwards in stitching. The larger jump from 4 to 5 columns happens when the stacked height reaches the hardware limit, and rearrangements are more severe because another hardware encoder instance is required. The rapid increase thereafter happens within NVENC, obfuscating its origin. It is likely a byproduct of the in-loop deblocking filter, which crosses tile boundaries but cannot do this efficiently in the increasingly tall and slim configurations. Configurations of 7 columns incur the worst case padding overhead, making it a particularly challenging case.

In Fig. 2(b), we see that the number of columns has a drastic effect on the output file size once sub-images are introduced. Decrease in coding efficiency is to be expected with smaller tiles due to intra-picture prediction and entropy encoding, but it is interesting that this only occurs with sub-images and is not affected by the number of tile rows. This may also result from the in-loop deblocking filter. Here too, the spike at 7 columns is likely due to inferior alignment.

In addition to having sufficient speed and coding efficiency, we must ensure that the video quality is good enough to achieve a high quality of experience for the end user. In a tile-based ABR streaming scenario, this is especially true in high-quality tiles since these will likely be at the current position of the viewport. Furthermore, NVENC is a new technology not yet capable of matching the output quality of software encoders, and we can expect quality to degrade with more tiles because intra-prediction and motion estimation cannot pass tile boundaries, so choosing an appropriate tile granularity is essential. We use SSIM to express the deviation between the original and the tiled encoded frames, showing the results in Fig. 2(c). Well-aligned tile configurations (1, 2, 4 and 8 rows, 1 through 6 colums) provide an intuitive quality degradation from high to low quality. In Fig. 2(c) we only show the representative cases of 5 and 8 rows. For the middle quality we use a checkerboard pattern of alternating low and high qualities per tile showing the maximum number of quality transitions possible. Note that forced alignment of the tile height by cropping requires interpolation before SSIM, with the result that lower quality images with higher blur scores better in SSIM. Configurations with 7 and 8 columns require horizontal padding, resulting in very low SSIM scores.

6 CONCLUSION

In this work, we demonstrated an adaptive live video platform capable of encoding tiled 360 ° videos at multiple bitrates using an accessible consumer-grade GPU. By working closely with NVENCODE and the HEVC standard, we overcame limitations that had previously rendered hardware encoders useless in generating adaptive tile-based live video streaming. Furthermore, we have made our code open source and provided a demonstration of an HTTP video streaming service based on RATS.

REFERENCES

[1] Peter Amon, Madhurani Sapre, and Andreas Hutter. 2012. Compressed domain stitching of HEVC streams for video conferencing applications. In *Proc. of 19th International Packet Video Workshop (PV)*.

[2] Xavier Corbillon, Alisa Devlic, Gwendal Simon, and Jacob Chakareski. 2017. Viewport-Adaptive Navigable 360-Degree Video Delivery. In *Proc. of IEEE International Conference on Communications (ICC 2017)*.

[3] Christian Feldmann, Christopher Bulla, and Bastian Cellarius. 2013. Efficient stream-reassembling for video conferencing applications using tiles in HEVC. In *Proc. of International Conferences on Advances in Multimedia (MMEDIA)*.

[4] Carsten Griwodz, Mattis Jeppsson, Håvard Espeland, Tomas Kupka, Ragnar Langseth, Andreas Petlund, Peng Qiaoqiao, Chuansong Xue, Konstantin Pogorelov, Michael Riegler, Dag Johansen, and Pål Halvorsen. 2018. Efficient Live and on-Demand Tiled HEVC 360 VR Video Streaming. In *2018 IEEE International Symposium on Multimedia, ISM 2018, Taichung, Taiwan, December 10-12, 2018*. IEEE Computer Society, 81–88. https://doi.org/10.1109/ISM.2018.00022

[5] Jean Le Feuvre and Cyril Concolato. 2016. Tiled-based Adaptive Streaming Using MPEG-DASH. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16)*.

[6] Omar A. Niamut, Emmanuel Thomas, Lucia D'Acunto, Cyril Concolato, Franck Denoual, and Seong Yong Lim. 2016. MPEG DASH SRD: Spatial Relationship Description. In *Proc. of the 7th International Conference on Multimedia Systems (MMSys '16)*.

[7] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. 2017. An HTTP/2-Based Adaptive Streaming Framework for 360° Virtual Reality Videos. In *Proc. of the ACM Conference on Multimedia*. 306–314.

[8] Y. Sánchez de la Fuente, R. Skupin, and T. Schierl. 2017. Video processing for panoramic streaming using HEVC and its scalable extensions. *Multimedia Tools and Applications* (2017).

[9] https://support.google.com/youtube/answer/2853702