Event-driven Stitching for Tile-based Live 360 Video Streaming

Bo Chen
University of Illinois at Urbana-Champaign
Urbana, Illinois
boc2@illinois.edu

Zhisheng Yan
Georgia State University
Atlanta, Georgia
zyan@gsu.edu

Haiming Jin
Shanghai Jiao Tong University
Shanghai
zyan@gsu.edu

Klara Nahrstedt
University of Illinois at Urbana-Champaign
Urbana, Illinois
Klara@illinois.edu

ABSTRACT

360 video streaming is gaining popularity because of the new type of experience it creates. Tile-based approaches have been widely used in VoD 360 video streaming to save the network bandwidth. However, they cannot be extended to the case of live streaming because they assume the 360 videos stitched offline before streaming. Instead, stitching has to be done in real-time in live 360 video streaming. More importantly, the stitching speed as shown in our experiments is one order of magnitude lower than the network transmission speed, making stitching more of a deciding factor of the overall frame rate than the network transmission speed. In this paper, we design a stitching algorithm for tile-based live 360 video streaming that adapts stitching quality to make the best use of the timing budget. There are two main challenges. First, existing tile-based approaches do not consider various semantic information in different scenarios. Second, the decision of tiling schemes for tile-based stitching is non-trivial. To solve the above two challenges, we present an event-driven stitching algorithm for tile-based 360 video live streaming, which consists of such an event-driven model to abstract various semantic information as events and a tile actuator to make tiling scheme decisions. We implement a streaming system based on event-driven stitching called LiveTexture. To evaluate the proposed algorithm, we compare LiveTexture with other baseline systems and show that LiveTexture adapts well to various timing budgets by meeting 89.4% of the timing constraints. We also demonstrate that LiveTexture utilizes the timing budget more efficiently than others.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Human-centered computing** → **Virtual reality**; • **Theory of computation** → *Scheduling algorithms*; • **Computing methodologies** → *Optimization algorithms*;

MMSys '19, June 18–21, 2019, Amherst, MA, USA

ACM ISBN 978-1-4503-6297-9/19/06…$15.00
https://doi.org/10.1145/3304109.3306234

KEYWORDS

Live streaming, tile-based stitching, 360 video

ACM Reference Format:
Bo Chen, Zhisheng Yan, Haiming Jin, and Klara Nahrstedt. 2019. Event-driven Stitching for Tile-based Live 360 Video Streaming. In *MMSys '19: 10th ACM Multimedia Systems Conference, June 18–21, 2019, Amherst, MA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3304109.3306234

1 INTRODUCTION

360 video streaming is becoming more and more popular because of the new type of immersive environment it brings. 360 video streaming can be categorized into on-demand 360 video streaming and live 360 video streaming. For the on-demand case, web services like YouTube have already been able to provide on-demand 360 video streaming, where the viewer can watch a 360 video with a head mounted display (HMD) like the Google Cardboard [4] and virtually experience the feeling of being at a historical site or riding a roller coaster. The live case is also useful in scenarios like sports broadcasting to let remote viewers watch a game in a real-time and immersive manner.

Despite the extensive studies on the on-demand 360 video streaming systems, they cannot be directly extended to the live case. Many tile-based streaming techniques [3, 16, 21, 23] have been proposed as the default architecture in on-demand 360 video streaming system. In these systems, a 360 video is divided into chunks of the same duration and each chunk is further spatially divided into many tiles. Each of these tiles will be encoded into segments with different bitrate levels and stored at the server. When a client requests the video, the segments will be fetched and sent to the client based on the network condition using DASH protocol [15]. However, these approaches all make the assumption that the 360 video has been already prepared in an offline setting and cannot be scaled to the live case.

Across the system pipeline, the key difference between on-demand and live 360 video streaming systems is the need for real-time stitching. Unfortunately, we observe that, in a single-hop streaming link, stitching consumes much more time than other system components such as camera feeding, network transmission and video rendering (Section 2). This provides a unique opportunity to address stitching in the system design to achieve the vision of live 360 video streaming.

The objective of this paper is to design a stitching algorithm for tile-based live 360 video streaming systems that can logically divide

a panoramic frame into tiles for stitching and adjust the number of pixels to be processed in tiles in real-time. This stitching algorithm considers the semantic information indicating the region of interest where the viewer would be looking, and the stitching quality of each tile is chosen such that the tile which is of more interest to the viewer is stitched at a higher quality while the stitching time is restrained by the timing budget per frame [1] (timing constraint).

To achieve this goal, we are facing two challenges. First, the semantic information in live 360 video streaming is diverse depending on the scenarios, e.g., in the broadcasting of a basketball game, the viewer would be more interested in the players and the ball. Even though the existing architecture for the on-demand 360 video streaming has adopted a few general techniques to characterize the viewer's interests, e.g., maximizing the sum of quality in the saliency region [9] or maximize the quality in the region where the viewer will watch [3], they are not versatile enough to be generalized to various scenarios.

Second, given the semantic information, it is non-trivial to decide on an optimal tiling scheme [2]. Our experiment (Section 5) shows that the stitching time of a tile is a complex function of the tiling scheme that cannot be explicitly expressed, which makes it hard to model this decision as an optimization problem. A naive approach is to run the stitching module over every possible combination of the tiling scheme and semantic information to measure the stitching time and the objective (defined over the semantic information) under every possible setting. Thus, for any semantic information, we can pick an optimal tiling scheme that both satisfies the timing constraint as well as achieves the highest value of the objective. However, the number of combinations to explore would be countless.

To solve the first challenge, this paper presents a versatile event-driven model for tile-based 360 video live streaming. In this model, an event detector detects various types of semantic information and models them as events. The event is a 2D coordinate on the panoramic frame indicating the region where the viewer might be interested in. With this model, we can handle various scenarios by modeling useful semantic information as events. For example, in the broadcasting of a basketball game, the events can be defined as the location of the players' faces and the ball. We further define an objective function based on the events in a frame. By optimizing this objective function, we can decide on a tiling scheme for a frame.

For the second challenge, we design a stitching module comprised of a tile actuator and a basic stitcher. The tile actuator chooses the optimal tiling scheme via an offline profiling phase and an online greedy search phase. The basic stitcher uses the chosen tiling scheme to perform tile-based stitching.

Based on the proposed event-driven model and the stitching module, we implement an event-driven tile-based live 360 video streaming system named LiveTexture. To evaluate the proposed stitching algorithm, we compare several baseline live 360 streaming systems with LiveTexture over multiple combinations of events and timing budgets. The results show that LiveTexture is reliable, i.e., capable of meeting 89.4% of the timing constraints, which implies that the proposed algorithm can adapt well to various timing budgets. LiveTexture is also capable of providing a mean normalized score that is better than most baseline systems and shows that it can use the available timing budget more efficiently.

In summary, there are three main contributions of this paper.

- We propose a versatile event-driven model for tile-based live 360 video streaming that can handle different scenarios by modeling semantic information as events.
- We design a stitching module comprised of a basic stitcher and a tile actuator. The tile actuator finds the optimal tiling scheme via the offline profiling phase and the online greedy search phase.
- In the evaluation, we compare LiveTexture with various baseline systems over multiple combinations of sets of events and timing budgets. LiveTexture achieves a high reliability of 89.4% and a mean normalized score that is better than most of other baseline systems.

In the remainder of this paper, we first show the motivation of this paper by presenting a prototype for the live 360 video streaming and performing the latency analysis in Section 2. Based on the event-driven model and the tile-based 360 video stitching model, we present the event-driven tile-based live 360 video streaming framework in Section 3. In Section 4, we present the problem formulation for the stitching module of the proposed framework and identify the challenge. In Section 5, we design a tile actuator to tackle the aforementioned challenge. In Section 6, we present implementation details of the system, LiveTexture, built on top of the event-driven tile-based live 360 video streaming framework. The evaluation is conducted in Section 7. The related work is presented in Section 8. Finally, we conclude this paper in Section 9.

2 MOTIVATION

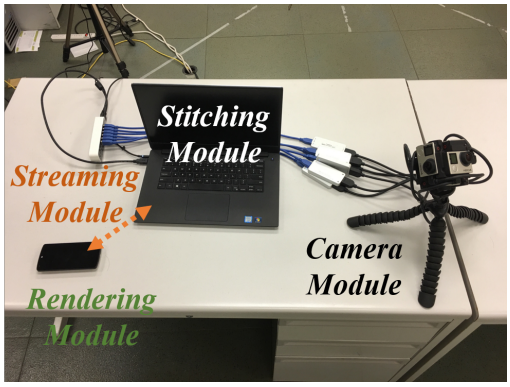


Figure 1: Live 360 video streaming prototype

Our motivation study shows that stitching is a primary component of bottleneck in the pipeline of live 360 video streaming and it is necessary to improve its performance. To study the latency performance of live 360 video streaming, we build a prototype as shown in Figure 1 to perform latency analysis. This system consists of a camera module (6 GoPro [6] cameras, 30 fps @ $960 \times 720 pixels$), a stitching module (implemented on a laptop with a 8-core CPU), a streaming module (implemented on both the laptop and the Android phone), and a rendering module (implemented on the Android

[1] The timing budget is a value determined by the required frame rate of a specific application, i.e., timing budget = 1/ frame rate.
[2] The tiling scheme describes how a panoramic frame is divided and how the stitching quality of each tile is assigned.

phone). We measure the processing latency required to render one panoramic frame in a single-hop setup including the camera feeding latency (CamFeed), the stitching latency (Stitching), the network transmission latency [3] (Net) and the rendering latency (Render), respectively. The stitched panoramic frame is of 2048 × 1024 pixels. The result is shown in Figure 2.

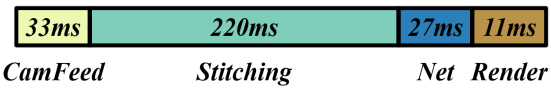


Figure 2: Time analysis of the prototype

It can be seen that the stitching latency is one order of magnitude higher than the latency of other modules. In other words, the rate of the stitching module to produce panoramic frames is too slow to catch up with that of the other modules to perform feeding of raw frames, network transmission of panoramic frames, or rendering of panoramic frames. This does not happen to on-demand 360 video streaming systems because, in their cases, all the video contents are stitched before requested and no real-time stitching is required as in live 360 video streaming. The condition may vary across different platforms with different processing power, e.g., replacing a quad-core Intel i7-3770 CPU @ 3.40 GHz with a GTX 980 GPU can improve the average frame rate to stitch a 2160 × 1080 panoramic frame from 7.30 fps to 23.76 fps [9]. However, it is unrealistic to embed GPUs into a 360-degree camera, e.g., GoPro Omni [6] and Ricoh Theta S [17], due to the large size, power consumption and the overheat problem of GPUs. For CPU-based 360 video live streaming systems, the stitching speed is crucial for the real-time performance of the whole live streaming system, whose latency requirement ranges from milliseconds (interactive applications such as remote medical assistance) to minutes (e.g., NFL broadcasting). Optimizing the stitching speed can help meet the latency requirement and make the system resilient to network jitters. Hence, we decide to focus on improving the stitching speed of 360 videos in this paper. Since tile-based approaches [20, 21] have been widely adopted for on-demand 360 streaming to save bandwidth in recent works, it is natural to consider tile-based approaches for live 360 video streaming. Motivated by this situation, the rest of this paper will answer the following three questions.

- How can we conduct tile-based live 360 video streaming (Section 3)?
- What are the new challenges in doing live 360 video stitching (Section 4) and how to solve it (Section 5) ?
- Is tile-based stitching good for live 360 video streaming (Section 7)?

3 EVENT-DRIVEN TILE-BASED LIVE 360 VIDEO STREAMING FRAMEWORK

In this section, we first introduce the event-driven model and the stitching model. Based on these two models, we propose the event-driven tile-based live 360 video streaming framework.

[3]Note that the network transmission latency depending on the network bandwidth is different from the propagation latency depending on the medium.

3.1 Event-driven model

In this subsection, we will first define the event in a tile-based 360 video streaming context and discuss how tile-based streaming is driven by the event.

3.1.1 Event. The event is a 2D point in the panoramic frame that carries semantic information. The semantic information indicates what region in the panoramic frame the viewer might be interested in, which can be extracted by image processing algorithms or recorded by sensors while the viewer is using a head-mounted display to view the video. Hence, the event can take many forms, e.g., the center of the viewer's viewport, the saliency point, or a point touched by the viewer on the screen (when the application is interactive).

3.1.2 Event-driven tile-based streaming. In event-driven tile-based streaming, the scaling ratios of tiles are selected such that tiles containing more events will have higher stitching quality with the constraints that the required resource does not exceed its limit.

The event-driven model for tile-based streaming can be generalized to many existing works. In 360 Video Foveated Stitching [9], the saliency information and the foveated region are used as events to reduce the data rate. In viewport-adaptive streaming [3], the viewport is used as the event. This model is also versatile to be applied to a specific scenario. For example, in a sport scenario where a basketball game is streamed in real-time, the event can be defined as locations where the players' face and the ball are detected. We show the example in Figure 3, where the location of players' faces and the ball, i.e., events, are marked by circles. In this way, the viewer can get more detailed image information of the players' face and the ball.

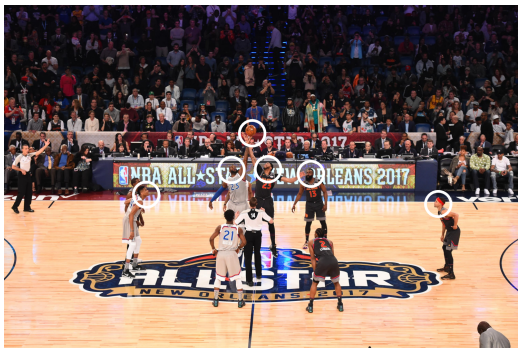


Figure 3: Events in a sport scenario (location of players' faces and the basketball are treated as events)

3.2 Stitching model

The stitching model we adopt is the tile-based 360 video stitching model. The panoramic frame is logically divided into N tiles indexed in raster-scan order. The stitching model is shown in Figure 4. For each tile to be stitched, the image content is first fetched from the corresponding camera(s) at its original size. Then, the fetched content will be down-sampled (scaled down) by a scaling ratio selected from M possible scaling ratios, i.e., the ratio of the size of a tile after scaling to that before scaling (scaling). Next, the scaled image content will be projected (warping) and blended (blending)

onto a tile on a scaled panoramic frame [4]. After all tiles are blended, one or multiple scaled panoramic frame will be created depending on the number of used scaling ratios (3 in the example of Figure 4). Finally, all scaled panoramic frames are up-sampled to the size of the output panoramic frame and then added together to create the output panoramic frame (stacking).

3.3 System Architecture

Based on the event-driven model and the stitching model, we propose an event-driven tile-based live 360 video streaming framework, which is shown in Figure 5. This architecture is comprised of a camera module, a stitching module, a streaming module, a rendering module and an event detector. The event detector detects semantic information and delivers the events to the stitching module [5]. The stitching module consists of a basic stitcher and a tile actuator. The basic stitcher fetches image contents from the camera module and the tiling scheme from the tile actuator, and create the output panoramic frame. Details on how the basic stitcher works are illustrated in Figure 4 The tile actuator makes tiling scheme decisions, i.e., decides on how a panoramic frame should be divided into tiles and how each tile should be scaled, based on events produced by the event detector. The streaming module sends the stitched panoramic frames over the Internet, which will then be rendered by the rendering module at the client side.

The key module for the proposed framework to adapt to various types of semantic information is the stitching module, where the tile actuator is the most important part. Next, we will present the problem formulation for the tile actuator and the challenge in Section 4. After that, we will present our design of the tile actuator in Section 5.

4 TILE ACTUATOR PROBLEM AND CHALLENGES

In this section, we first formulate the problem of making tiling scheme decisions based on the event-driven model. Then, we identify the challenge in solving this problem.

4.1 Problem formulation

We assume that a tile actuator has K division schemes indexed by $k \in \{1, ..., K\}$, where the k-th division scheme divides a panoramic frame into N_k equal-sized grids (tiles). Each tile is indexed by $i \in \{1, ..., N_k\}$. For clarity, in the rest of this paper, we will assume all notations are made under the k-th division scheme unless we point out otherwise, and omit the subscript of k. For example, we will use N to replace N_k.

We also define M different scaling ratios $s_j, j \in \{1, ..., M\}$, where $s_j \in (0, 1]$ is the ratio of number of pixels in a scaled tile to that in the corresponding original tile [6]. We can obtain a scaling scheme by assigning the scaling ratio to every tile in a panoramic frame. The set of all possible scaling schemes is denoted by $\{X_l\}$, where each scaling scheme X_l is indexed by $l \in \{1, ..., M^N\}$ and M^N is the number of combinations of scaling schemes. In the scaling scheme $X_l = \{x_{i,j}\}$ (the subscript l is omitted for clarity), $x_{i,j} = 1$ denotes that the i-th tile selects the j-th scaling ratio for stitching and $x_{i,j} = 0$ otherwise. (k, X_l) is defined as a *tiling scheme*. We further introduce the concept of the event e, which indicates the location in a panoramic frame that might be interesting to the viewer and is represented by (e_x, e_y), the x- and y- coordinates of e. The set of events is denoted by $E = \{(e_x, e_y)\}$.

In this problem, the tile actuator needs to find the optimal tiling scheme (k^*, X_{l^*}), i.e., the k^*-th division scheme and the l^*-th scaling scheme under this division scheme, for a specific set of events E.

We define the *event score* $u = \Phi(k, X_l, E)$ to represent the overall quality achieved for an event set E with the tiling scheme (k, X_l). By using n_i to denote the number of events within the i-th tile, the event score can be defined as shown in Equation 1.

$$\Phi(k, X_l, E) = \sum_{i=1}^{N} n_i \sum_{j=1}^{M} x_{i,j} s_j. \tag{1}$$

The objective to maximize the event score $\Phi(k, X_l, E)$ can be done in two steps.

In the first step, we fix k and find the optimal l^* for every division scheme indexed by k as shown in Equation 2.

$$\begin{aligned} l^* &= \arg\max_{l} \Phi(k, X_l, E) \\ s.t. &\sum_{i=1}^{N} \sum_{j=1}^{M} \tau_{i,j} x_{i,j} \le T, \\ &\sum_{j=1}^{M} x_{i,j} = 1, \forall i. \end{aligned} \tag{2}$$

The first constraint restricts the total time consumption of stitching a frame, where $\tau_{i,j}$ denotes the stitching latency of the i-th tile at the j-th scaling ratio, and T is the timing budget, which is determined by the required frame rate of the system. The second constraint restricts that there is only one scaling ratio for each tile.

In the second step, we find the index of the division scheme k^* producing the highest event score as shown in Equation 3:

$$k^* = \arg\max_{k} \Phi(k, X_{l^*}, E), \tag{3}$$

where l^* is obtained in the previous step. After this two-step optimization, we can obtain the optimal tiling scheme (k^*, X_{l^*}). However, we cannot directly solve the above optimization problem. The reason is that the modeling of $\tau_{i,j}$ is difficult, which will be discussed in the next subsection.

4.2 Challenges

To illustrate why modeling $\tau_{i,j}$ is difficult, we run the stitching module under different division schemes of a panoramic frame of 2048×1024 pixels, i.e., 1×1, 2×1, 2×2, 4×2, 4×4, and 8×4, where $a \times b$ means a panoramic frame is divided into equal-sized grids of a columns and b rows. Under each division scheme, we fix the scaling ratio of all tiles to the same value and measure the total stitching latency of a panoramic frame for different values of the

[4] All tiles with the same scaling ratio share one scaled panoramic frame.

[5] This paper does not discuss in detail how the semantic information are extracted which is beyond the scope of this paper. In fact, some events can be generated by performing computer vision (CV) algorithms, e.g., face detection, on the video. Some other events can be extracted by analyzing the sensor data while the viewer is watching, e.g., the gyrometer and the "touch" event of a touch screen. At the implementation level, we just simulate the behavior of the event detector by generating a random number of points at random positions on a panoramic frame.

[6] We sort the scaling ratios such that the smaller scaling ratio gets a larger index value.

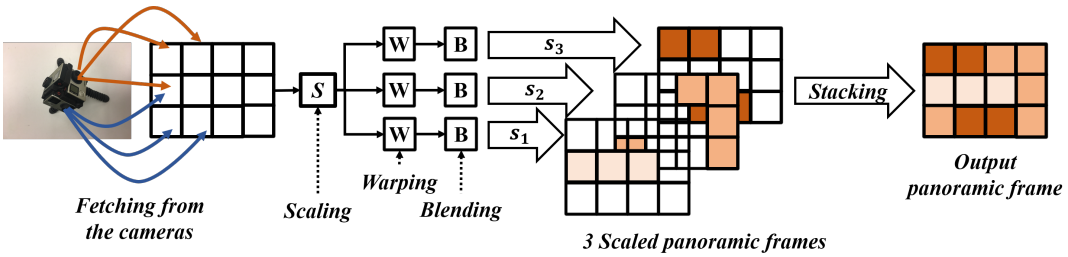


Figure 4: Stitching model (basic stitcher in Figure 5)

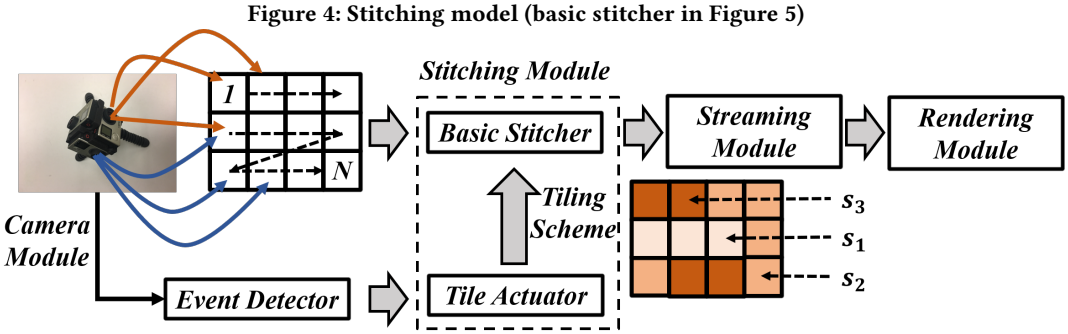
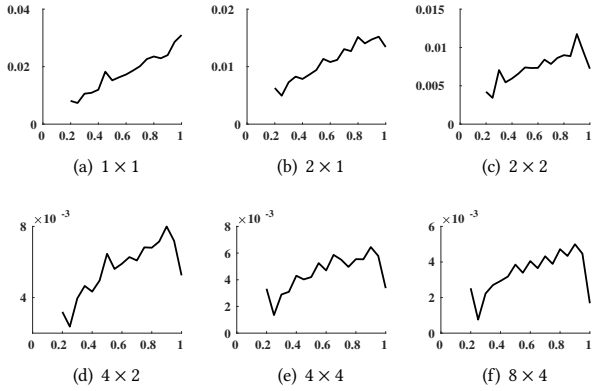


Figure 5: Event-driven tile-based live 360 video streaming framework

scaling ratio, i.e., 0.2, 0.25,...,1. The average stitching latency per tile for different scaling ratios is calculated by dividing the total stitching latency for different scaling ratios over the number of tiles and plotted in Figure 6. The x-axis in each figure represents the scaling ratio while the y-axis represents the processing latency for stitching. Obviously, the processing latency is neither linear to the scaling ratio nor monotonic!

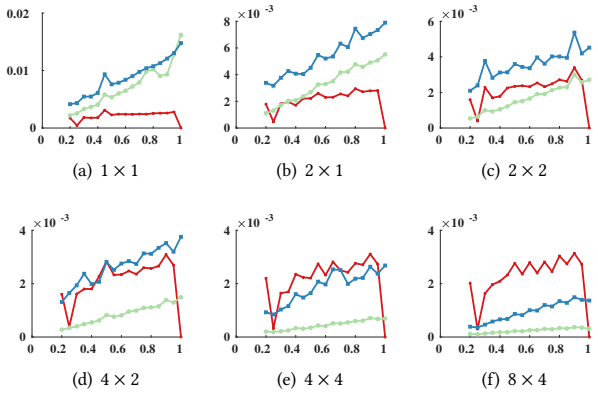


(a) 1×1 (b) 2×1 (c) 2×2

(d) 4×2 (e) 4×4 (f) 8×4

Figure 6: Average processing latency for stitching a tile ($a \times b$ means a 2048×1024 frame is divided into equal-sized grids of a columns and b rows). The x-axis represents the scaling ratio while the y-axis represents the processing latency (s).

In these figures, the higher scaling ratio leads to the higher processing latency in most cases, which is reasonable because more pixels are being processed. However, when the number of tiles in a division scheme increases, the latency is not linear to the scaling ratio. What is even more counter-intuitive is that at some scaling ratios, the processing latency for stitching a tile is even larger than that without scaling. For example, in Figure 6(e), the processing latency for stitching of a tile when the scaling ratio is 0.65 is larger than that when the scaling ratio is 1, i.e., no scaling.



(a) 1×1 (b) 2×1 (c) 2×2

(d) 4×2 (e) 4×4 (f) 8×4

Figure 7: Average processing latency for different operations ($a \times b$ means a 2048×1024 frame is divided into equal-sized grids of a columns and b rows). The x-axis represents the scaling ratio while the y-axis represents the processing latency (s). The red, green and blue lines represent the latency for scaling, feeding and warping, respectively.

To investigate the cause of this, we further conduct a thorough analysis of the total stitching latency of one panoramic frame where we analyze the latency of scaling, warping and blending operations performed on the image content from one camera to a tile [7]. The result is shown in Figure 7, where we can find that

i) the latency of warping and blending is nearly linear to the scaling ratio,
ii) the latency of scaling differs from the other two by first increasing then decreasing as the scaling ratio increases,

[7] We do not include the stacking time in Figure 7 since its effect is negligible in the pattern we observed

iii) when the size of divided tiles is large, e.g., 1×1 and 2×1, the latency of scaling is negligible compared with that of warping or blending, and
iv) when the size of divided tiles is small, e.g., 4×4 and 8×4, the latency of scaling is comparable to that of warping or blending.

These findings indicate that when the size of divided tiles is large, the latency of warping and blending dominates in each tile, thus the stitching latency appears to be nearly linear to the scaling ratio. However, when the size of divided tiles is small, the latency of scaling dominates in each tile, thus the stitching latency appears to be first increasing then decreasing as the scaling ratio increases.

We also find that the processing latency for warping differs from each other because of different positions of a tile in a panoramic frame, and the stacking time during stitching can be as high as $0.1s$ if we are stacking over 10 scaled panoramic frames together.

All these findings contribute to the unpredictability of the stitching latency. Confronted with this challenge, we propose a design of the tile actuator to derive the optimal tiling scheme based on an offline profiling phase and an online greedy search phase.

5 TILE ACTUATOR DESIGN

In this section, we present our design of the tile actuator to derive the optimal tiling scheme. This design consists of an offline profiling phase and an online greedy search phase. The offline profiling explore tiling schemes in an efficient way to find out the quasi-optimal tiling scheme for a given set of events in an offline manner. The quasi-optimal tiling scheme will be selected for different sets of events and further optimized by the online greedy search phase in real-time to obtain the final tiling scheme decision.

5.1 Offline Profiling

To find the quasi-optimal tiling scheme, we introduces the concept of the *Pareto Boundary* [24]. The Pareto Boundary in this paper is a small set of tiling schemes such that any tiling scheme not on the Pareto Boundary cannot outperform tiling schemes on the Pareto Boundary in terms of both the event score and processing latency for stitching at the same time. We find the Pareto Boundary for different set of events. In this way, when a set of events comes, the tile actuator can use the Pareto Boundary and select the tiling scheme that can maximize the event score with available CPU resources. However, there are two main problems

i) in contrast to the limited types of video queries, the number of different sets of events is countless and requires unlimited memory to hold the profile for every set of events, and
ii) in contrast to the limited selection of "knobs" while processing video queries, the complexity to explore different tiling schemes is about $O(M^{N_{max}})$ where M denotes the number of scaling ratios and N_{max} denotes the maximum number of tiles among all possible division schemes.

These two problems are fundamentally the "large state space" [8] problem. For the rest of this section, we will 1) present techniques to reduce the state space (Section 5.1.1), 2) explain how to explore the state space with aforementioned techniques (Section 5.1.2), and 3) describe how to find the *Pareto Boundary* for each set of events within the explored state space and use it to decide a quasi-optimal tiling scheme to be used in the online greedy search phase (Section 5.1.3).

5.1.1 State Space Reduction. We present four techniques named *Quantization*, *Hashing*, *Elimination* and *Offloading* to reduce the state space.

Quantization. The tile actuator needs to respond to different numbers of events in a tile differently, but instead of considering the actual number of events, we only consider the quantization level of events in a tile, denoted by q_i, where i is the index of a tile. Each of the quantization levels q_i is an integer, which can be linearly quantized from the number of events in a tile with a pre-defined total quantization levels. For example, suppose the number of events in four tiles are 0, 5, 9, 10 and the total quantization levels is 4, the quantization level of each tile is determined by finding out what range among $[0, 2.5)$, $[2.5, 5)$, $[5, 7.5)$ and $(7.5, 10]$ the number of events of a tile is in. Hence, the quantization level of the four tiles are 0, 2, 3, 3. The set of quantization levels of tiles in a panoramic frame is defined as the *quantization scheme*. The m-th set of quantization scheme is denoted by $Q_m = \{q_i | q_i \text{is the quantization level of the } i\text{-th tile}, i \in \{1, ..., N\}\}$. As a result, the event score defined in Equation 1 can be modified to Equation 4.

$$\Phi(k, X_l, Q_m) = \sum_{i=1}^{N} q_i \sum_{j=1}^{M} x_{i,j} s_j, \tag{4}$$

Hashing. Since the objective function in Equation 1 does not consider the spatial information of tiles, if we exchange the quantization levels of any two different tiles to produce a new quantization scheme (QS), we can easily obtain the corresponding optimal tiling scheme (OTS) by exchanging the scaling ratios in the original optimal tiling scheme at the positions previously exchanged in the quantization levels. For example, in Figure 8, QS #1 and QS #2 represent two quantization schemes while OTS #1 and OTS #2 represent the optimal tiling schemes for QS #1 and QS #2, respectively. Compared with QS #1, QS #2 only exchanges the quantization levels of two tiles, i.e., "1" and "3" in the figure. As a result, the optimal tiling scheme OTS #2 can be easily derived by exchanging the corresponding scaling ratios, i.e., "0.9" and "0.7", in OTS #1.

In other words, for any two quantization schemes with the same number of tiles for every quantization level, the optimal tiling scheme for them should be only considered once. We implement this technique by using a hash function $H(\cdot)$ that produces the same hash key for any two such quantization schemes, and quantization schemes with the same hash key will be only explored once.

Elimination. In Figure 6 we can find that for some division schemes, e.g., Figure 6(f), there exists a *critical scaling ratio* $s_c \in (0, 1)$ such that the stitching latency resulted from the scaling ratio $s \in [s_c, 1)$ is no better than that resulted from the scaling ratio $s = 1$, i.e., no scaling. Illustration of the critical scaling ratio is shown in Figure 9.

The critical scaling ratio implies that any tiling scheme containing a scaling ratio $s \in [s_c, 1)$ is no better than another tiling scheme with only that scaling ratio being replaced by 1. We term any scaling ratio $s \in [s_c, 1)$ as the *useless scaling ratio*. Any tiling

[8] The state space represents the combinations of tiling schemes and events in this paper.

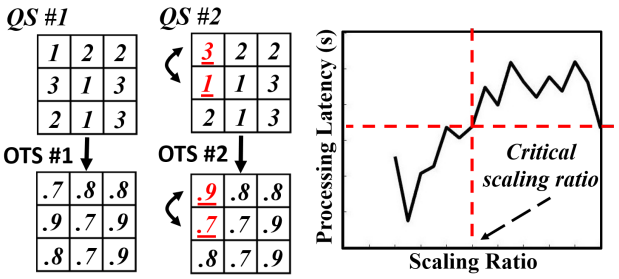


Figure 8: Optimal tiling schemes for quantization schemes with same hash

Figure 9: Critical scaling ratio

schemes having useless scaling ratios are eliminated and will not be explored in the offline profiling phase.

Offloading. For every quantization scheme, we enforce the scaling ratio of tiles with the same quantization level to be the same and let the offline profiling phase to do the fine-grained adjustment of the scaling ratio of tiles.

5.1.2 State Space Exploration. We explore the entire state space by exploring each division scheme sequentially. The pseudocode for the state space exploration of the k-th division scheme with aforementioned state space reduction techniques is shown in Algorithm 1.

We require all possible scaling schemes $\Omega = \{X_l\}$, all quantization schemes $\Psi = \{Q_m\}$, and the critical scaling ratio s_c. The outputs are all explored score-latency pairs $\{(u, t)\}$, where u and t are the event score and the latency, respectively. When the exploration starts, a hashmap will be first initialized to empty (line 1). Then, for each quantization scheme $Q_m \in \Psi$ (line 2), we obtain the hash key $key = H(Q_m)$ (line 3). If key exists in the hashmap, the exploration continues with the next scaling scheme. Otherwise, key will be inserted into the hashmap (line 4-7). After that, we iterate over all scaling schemes $X_l \in \Omega$ (line 8). We use the *Elimination* technique to check whether there exists a tile in the scaling scheme containing the *useless scaling ratio* and we continue with the next scaling scheme if it does (line 9-10). If we find two tiles, e.g., the i-th tile and the $i^{'}$-th tile, having the same quantization level $q_i = q_{i'}$ but with different scaling ratios, the exploration continues with the next scaling scheme (*Offloading*) (line 11-12). After all above checks to reduce the state space, based on the scaling scheme X_l, we finally run the basic stitcher to stitch 6 images of 960×720 pixels into one panoramic frame of 2048×1024 pixels, measure the latency, calculate the event score defined in Equation 4 and save the score-latency pair into a file (line 13-16).

By using the above techniques, the size of the explored state space is 643, 635 while that is over 68^{32} if none of the state reduction techniques is applied, which saves both profiling time, storage space and the hashing time for our system.

5.1.3 Quasi-optimal tiling scheme. After obtaining all the explored score-latency pairs $\Gamma = \{(u, t)\}$, we can determine the *Pareto Boundary* Γ_p by inserting $\{(u, t)\} \in \Gamma$ into Γ_p if the condition, $\forall (u, t) \in \Gamma, (u - u^{'}) \cdot (t^{'} - t) \leq 0$, is satisfied, which means none of the score-latency pairs in Γ is strictly better than any score-latency pair in Γ_p by achieving both higher event score and lower latency. Since the state space for a quantization scheme is not large, we implement the derivation of the *Pareto Boundary* in a brute-force way that compares every pair of explored score-latency pairs.

Algorithm 1 State Space Exploration (k-th division scheme)

Require: Scaling schemes: Ω; quantization schemes: Ψ; critical scaling ratio: s_c
Ensure: Explored score-latency pairs $\{(u, t)\}$
1: $hashmap \leftarrow \{\}$
2: **for** Q_m in Ψ **do**
3: $h \leftarrow H(Q_m)$
4: **if** h exists in $hashmap$ **then**
5: continue
6: **else**
7: $hashmap.insert(h)$
8: **for** X_l in Ω **do**
9: **if** $\exists x_{i,j} \in X_l$, s.t., $x_{i,j} = 1, s_j \in [s_c, 1)$ **then**
10: continue
11: **if** $\exists i, i^{'}, q_i = q_{i'}, i \neq i^{'}$, s.t., $x_{i,j} = 1, x_{i',j'} = 1, j \neq j^{'}$ **then**
12: continue
13: Run the basic stitcher based on X_l
14: Measure latency t
15: Calculate event score $u = \Phi(k, X_l, Q_m)$
16: Save (u, t) pair into a file

Now that we have the *Pareto Boundary*, we can follow the procedure illustrated in Figure 10 to find the quasi-optimal tiling scheme for a given set of events E. First, we will divide the panoramic frame as instructed by different division schemes, e.g., 1×1, 2×1, and 2×2. Then, we quantize the given set of events E into the quantization scheme based on different division schemes. The hash key of the quantization scheme will be computed by the hash function and used to query the corresponding *Pareto Boundary* in the memory. In the graph to the right of "Hashing", the x-axis represent the processing latency while the y-axis represent the event score. The red triangles represent score-latency pairs generated by different tiling schemes in a division scheme. Among these red triangles, those on the Pareto Boundary are marked by blue dots. On the *Pareto Boundary*, all scaling schemes with their processing latency greater than the timing budget ($T = 0.15$s in this example) will be discarded. Next, the tiling scheme with the highest latency is selected from the remaining scaling schemes, which is defined as the *Constrained Pareto Optimal* scaling scheme denoted by X_{lcpo} (the subscript of k is omitted for clarity). Since this scaling scheme is chosen from the *Pareto Boundary*, there will not be any scaling scheme achieving higher event score without violating the timing constraint. The corresponding tiling scheme (k, X_{lcpo}) is defined as the Constrained Pareto Optimal tiling scheme (referred to as the CPO tiling scheme). After obtaining the CPO tiling schemes for all division schemes, we can determine the quasi-optimal tiling scheme by selecting the CPO tiling scheme producing the highest event score (as defined in Equation 1) and denoting that tiling scheme by (k^{qo}, X_{lcpo}). The latency associated with it will also be saved for the online greedy search phase.

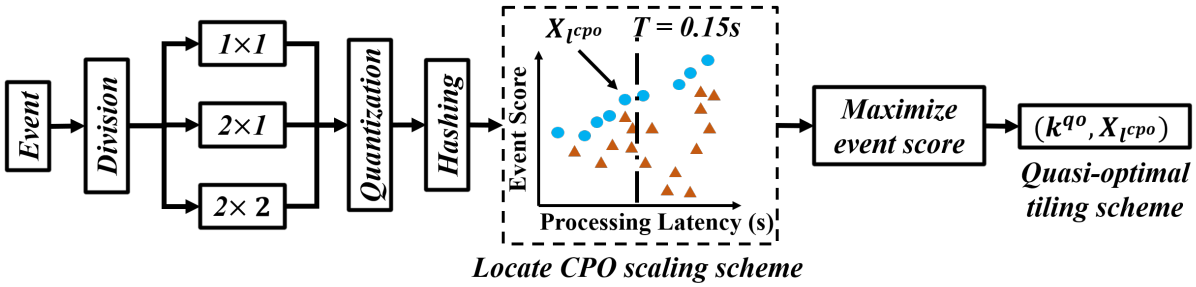


Figure 10: Find the quasi-optimal tiling scheme for a set of events, timing budget $T = 0.15$s

5.2 Online Greedy Search

In the offline profiling phase, we obtain a quasi-optimal tiling scheme for a given set of events. The online greedy search phase will fix the division scheme and fine-tune the scaling scheme through a greedy search approach to derive the optimal tiling scheme (k_*, X_{l^*}). The online greedy search phase proceeds in steps. Intuitively, in each step, it selects the tile that can maximize the gained event score per unit of incurred processing latency of the stitching module, and increases the scaling ratio in that tile. To achieve this, we need to solve two problems: 1) estimation of time consumption incurred by the increased scaling ratio, and 2) selection of a tile in each step.

5.2.1 Time consumption estimation. When the scaling ratio of a tile changes, the time consumption is mainly affected in two ways:

i) the sum of time for scaling, warping and feeding of that tile which changes as the number of pixels changes with the scaling ratio, and
ii) the time to stack scaled panoramic frames of different scaling ratios, which is increased when the scaling ratio of a tile is increased to a value not used in the previous step.

By denoting the scaling ratio of the i-th tile and the increment of the scaling ratio as s_i and δs, the total incremented time cost of the tile can be expressed as $\Delta\tau(s_i, \delta s) = \Delta\tau_1(s_i, \delta s) + \Delta\tau_2(s_i, \delta s)$, where $\Delta\tau_1(s_i, \delta s)$ is the incremented time cost due to scaling, warping and blending, and $\Delta\tau_2(s_i, \delta s)$ is the incremented time due to stacking (see Figure 4).

To find $\Delta\tau_1(s_i, \delta s)$, we run the basic stitcher for different division schemes with the same scaling ratio in one panoramic frame. The sum of latency for scaling, warping and blending (SWB latency) of a panoramic frame is measured and divided by the number of tiles to calculate the average SWB latency for a tile. The average SWB latency of a tile is thus modeled as a linear piece-wise function of the scaling ratio s_i denoted by $t(s)$. Thus, we have $\Delta\tau_1 = t(s_i+\delta s) - t(s_i)$.

To measure $\Delta\tau_2(s_i, \delta s)$, we run the stitching module on different numbers of different scaled panoramic frames and find that $\Delta\tau_2(s_i, \delta s)$ is nearly linear to the number of panoramic frames to be stacked together. The average time to stack one panoramic frame is measured to be $0.0042s$ [9]. By using S to represent the set of used scaling ratios, we have

$$\Delta t_2(s_i, \delta s) = \begin{cases} 0.0042 & s_i + \delta s \in S \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

[9] This value is dependent on the size of the output panoramic frame (2048×1024 pixels in our case) and the processing platform which is described in Section 7.

5.2.2 Tile selection. We denote the index of each step up to the step F of tile selection by $f = 1, 2, ..., F$. At each step, a tile indexed by i is selected whose scaling ratio, denoted by s_i will be incremented to $s_i + \delta s$, where δs is a constant. The quantization level of the i-th tile is denoted by q_i and the estimated increased latency due to selecting the i-th tile is denoted by $\Delta\tau_i = \Delta\tau(s_i, \delta s)$. The selection of the tile in each step can be modeled as a following optimization problem:

$$\begin{aligned} i^* &= \arg\max_i \frac{q_i \delta s}{\Delta\tau_i} \\ s.t. \tau^F &= \Delta\tau_i + \tau^{F-1} \le T, \end{aligned} \quad (6)$$

where we select the tile i^* that maximizes the gained event score per unit of incurred processing latency. The constraint restricts the total latency of stitching after F steps below the timing budget, where τ^F denotes the estimated processing latency after F greedy search steps and τ^0 is the estimated stitching time by applying quasi-optimal tiling scheme. This greedy search terminates when a step cannot be proceeded without violating the timing constraint.

6 IMPLEMENTATION

Based on the proposed framework, we implement a system named LiveTexture. The stitching module is implemented on Dell Precision Workstation 5510 with the processor of Xeon E3-1505M v5 @ 2.80GHz×8 and memory of 32GB. We employ off-the-shelf solutions for the other modules. Specifically, the camera module is comprised of 6 GoPro cameras (30 fps @ 960×720), which are connected to the workstation via a USB 3.0 hub. The size of the panoramic frame is 2048×1024 pixels. The event detector is implemented on the workstation and emulated by generating a random number of events at random locations on the panoramic frame. The streaming module is implemented on the workstation and the Google Nexus 5 with GStreamer [7]. The codec and protocol for streaming are x264 and UDP. The rendering module is realized with Google VR SDK [5] on the Google Nexus 5, which can produce 360 videos ready to be viewed by a Google Cardboard.

7 EVALUATION

In this section, we evaluate LiveTexture by comparing it with several baseline systems. We also investigate the performance of LiveTexture across different division schemes.

7.1 Evaluation Methodology

Evaluation settings. We consider 6 division schemes, i.e., 1×1(#1), 2 × 1(#2), 2 × 2(#3), 4 × 2(#4), 4 × 4(#5), and 8 × 4(#6), 17 scaling ratios, i.e. 0.2, 0.25, ...,1, and 4 quantization levels.

Definitions and Notations. We use each system to process frames from 6 Gopro Cameras with each camera generating $C_e = 100$ frames and the 6 frames generated at the same time are associated with a randomly generated set of events. In each set of events, a random number of events are generated at random positions on the panoramic frame, which emulates the behavior of the event detector. We repeat the above process for $C_t = 21$ different timing budgets, i.e., 0.05s, 0.06s,...,0.25s [10], resulting in $C_e \times C_t = 2100$ different combinations of events and timing budgets. The experiment for each combination of events and timing budgets are repeated for $C_r = 5$ times.

We use $i \in \{1, ..., C_e\}$ to denote the index of the events, and $j \in \{1, ..., C_t\}$ to denote the index of the timing budget. The estimated tiling scheme for the i-th set of events and the j-th timing budget is denoted by $TS_{i,j}$. The corresponding estimated processing latency and event score are denoted by $\tau_{i,j}$ and $u_{i,j}$, respectively. The maximal achievable event score for the i-th event is denoted by U_i, which is the event score of the i-th event when the scaling ratios of all tiles are set to the highest value. The real processing latency for $TS_{i,j}$ denoted by $\hat{\tau}_{i,j}$ is calculated by running the basic stitcher under this tiling scheme for C_r times and taking the average of the time measured during each repeated experiment. For the i-th event and the j-th timing budget, we use an indicator function to denote whether the timing budget is violated as shown in Equation 7.

$$\mathbb{1}_{i,j} = \begin{cases} 1 & \hat{\tau}_{i,j} \leq \tau_{i,j} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

7.2 Baselines

We implemented LiveTexture and 5 types of baseline systems with different implementations of the tile actuator.

1. LiveTexture. This is the proposed approach that runs on 6 division schemes defined earlier. We denote this approach as "LT".

2. Thresholding. The approach is used in [9] to assign stitching quality based on the KLT [18] feature density of tiles. Since we used randomly generated events instead of doing events detection, we simply perform thresholding on the density of events. Each tile will be assigned the highest or lowest stitching quality based on whether the density of events of it is above or below the threshold. We denote this approach as "Thresh".

3. LiveTexture without the online greedy search. This approach will use the quasi-optimal tiling scheme of LT as the final optimal tiling scheme without doing the online greedy search. We denote this approach as "LTW/OGS".

4. Greedy Search. This set of approaches employs the online greedy search phase described in Section 5.2. The difference is that its optimization starts the tiling scheme with all tiles the lowest scaling ratio. We implement 6 versions of this approach for different division schemes denoted as "GS #1, GS #2, ..., GS #6".

5. All high/low stitching quality. This set of approaches makes the stitching quality of all tiles to be the highest/lowest regardless of the events and timing budgets. It models the behavior of 360 video live streaming systems that do not change the stitching quality of different tiles. We implement 6 versions of all high stitching qualities and all low stitching qualities, respectively. They are denoted as "High #1, High #2,..., High #6" and "Low #1, Low #2,..., Low #6".

6. LiveTexture running on different division schemes. In contrast to LT, running with 6 division schemes, we also implement 6 variants of LT, each running a single division scheme denoted as "Div #1, Div #2,...,Div #6".

[10] By choosing this range, we can observe the behavior of the system when it does not or does have enough timing budgets.

7.3 Macro Analysis

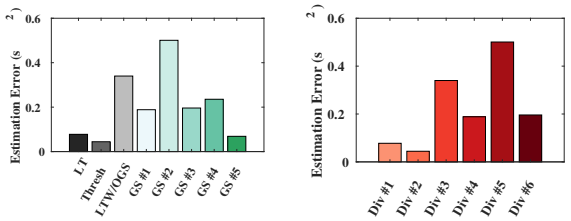
In this section, we compare the performance of LiveTexture with baseline systems.

7.3.1 Decision Time. Decision Time (*DT*) is the time of an approach to produce an optimization result, which measures how fast an approach can make a tiling scheme decision. The mean decision time for LT and LTW/OGS is around 0.0065*s*, which is higher than that of other approaches (approximately 0.0002*s*), but negligible for live 360 video streaming.

7.3.2 Estimation Error. Estimation Error (*EE*) evaluates the error of an approach to estimate the real processing latency of the stitching module. Estimation Error is measured as shown in Equation 8.

$$EE = \frac{1}{C_e \times C_t} \sum_{i=1}^{C_e} \sum_{j=1}^{C_t} \left(\frac{|\tau_{i,j} - \hat{\tau}_{i,j}|}{\hat{\tau}_{i,j}}\right)^2. \tag{8}$$

The result is shown in Figure 11(a). LTW/OGS has the lowest estimation error. The estimation error of LT and GS #6 are a little higher than LTW/OGS but within 0.1. The other baseline systems are not included since they do not estimate the stitching latency resulted from the tiling scheme.



(a) Different strategies (b) Different division schemes

Figure 11: Estimation Error

7.3.3 Reliability. Reliability (*R*) evaluates the capability of an optimization algorithm to provide a "valid" tiling scheme that enables the stitching module to meet various timing budgets. The calculated metric is shown in Equation 9. The result is shown in Figure 12.

$$R = \frac{1}{C_e \times C_t} \sum_{i=1}^{C_e} \sum_{j=1}^{C_t} \mathbb{1}_{i,j} \times 100\%. \tag{9}$$

We find that LT, LTW/OGS, and GS #1, 2, 3 can achieve high reliability of over 80% with that of LT being 89.4%. In addition, the reliability of the greedy search approaches, GS #1...6, decreases as the number tiles increases. The reliability of Thresh is bad (below 50%) because this approach cannot adapt the tiling scheme towards

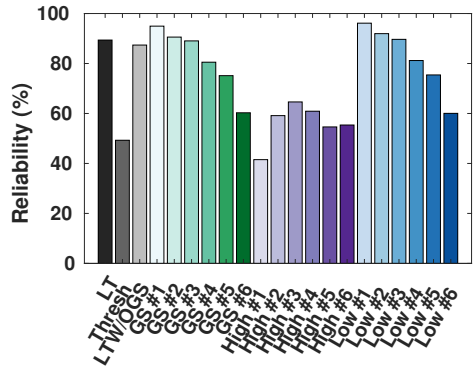


Figure 12: Reliability of different strategies

different timing budgets and fails to meet the timing constraint for more than half of the time. For High #1…6, the reliability is also bad because the stitching module violates the timing constraint for most of the time by adopting the highest stitching quality for all tiles. In contrast, the reliability for Low #1…6 is better than High #1…6 because the former are being more conservative by adopting the lowest stitching quality in all tiles.

7.3.4 Good-case Normalized Score (GNS). *GNS* evaluates how an approach maximizes the normalized event score (the ratio of the event score of an approach to the maximal achievable event score for a set of events) when the timing constraint is not violated, i.e., "good" cases. *GNS* is calculated via Equation 10. Figure 13 shows the result of *GNS*.

$$GNS = \frac{\sum_{i=1}^{C_e} \sum_{j=1}^{C_t} \mathbb{1}_{i,j} u_{i,j}/U_i}{\sum_{i=1}^{C_e} \sum_{j=1}^{C_t} \mathbb{1}_{i,j}}. \quad (10)$$

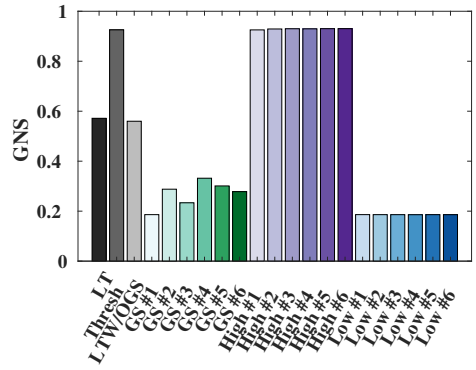


Figure 13: Good-case Normalized Score

High #1…6 achieve the highest *GNS* because they set the scaling ratio of all tiles to be the highest [11]. On the contrary, Low #1…6 achieve the lowest *GNS* because they set the scaling ratio of all tiles to be the lowest. LT achieves a $GNS = 0.572$, and LTW/OGS is close to LT in terms of *GNS*. Note that this value is dependent on both the platform the algorithm is running on and the range of the timing budget chosen for evaluation. These two approaches make conservative tiling scheme decisions to meet the timing constraints, making its *GNS* to be dragged down. *GNS* of Thresh is high because the chosen threshold value makes it tend to choose higher stitching quality than it should. We can also find that GS #1…6 do not perform well in terms of *GNS* implying that the online greedy search phase alone is not enough to make good use of the timing budget to achieve high event score.

7.3.5 Mean Normalized Score (MNS). *GNS* does not consider the cases where an approach violates the timing constraint, as a result of which, we use *MNS* to measure the event score of "good" cases under all cases regardless of whether the timing constraint is violated. *MNS* is calculated via Equation 11. The result is shown in Figure 14.

$$MNS = \frac{1}{C_e \times C_t} \sum_{i=1}^{C_e} \sum_{j=1}^{C_t} \mathbb{1}_{i,j} \frac{u_{i,j}}{U_i}. \quad (11)$$

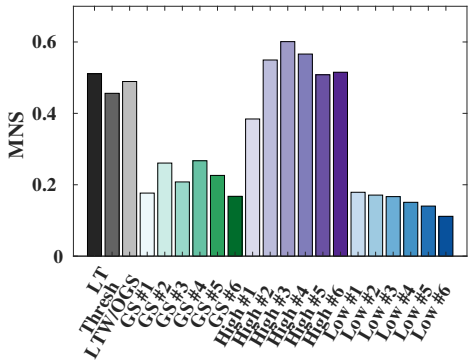


Figure 14: Mean Normalized Score

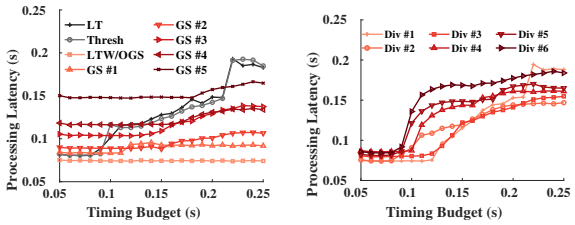
We can find that *MNS* is smaller than *GNS* for all approaches, but the reduction is not much for LT, LTW/OGS, and GS #1…6 because their reliability is high. LT achieves a $MNS = 0.511$, which is better than that of LTW/OGS and Thresh. The reason why using High #1…6 is comparable to our approach in terms of *MNS* is because the setup of our experiment range includes many timing budget that can be met by High #1…6 with the highest scaling ratios while our approach is conservative in these cases, which affects *MNS* of our approach. However, in real world, with the same *MNS*, it is better to meet 100% of the timing constraints with a moderate *GNS* (LT) than only meet 50% of the timing constraints with a high *GNS* (High #1…6).

7.3.6 Influence of the timing budget. To investigate how different approaches are affected by the timing budget, we plot the real processing latency as a function of the timing budget for LT, LTW/OGS, and GS #1…6 as shown in Figure 15(a). The real processing latency of LT, LTW/OGS and GS #1…6 gradually increases as the timing budget increases. The difference is that the real processing latency for each one of GS #1…6 can only be changed within a short range while LT and LTW/OGS can adapt the real processing latency in a wider range.

7.4 Micro Analysis

In this section we investigate how the division scheme affects the performance of LiveTexture.

[11] Note that $GNS = 1$ is sometimes not achievable because some sets of events is empty, making its event factor to be 0.



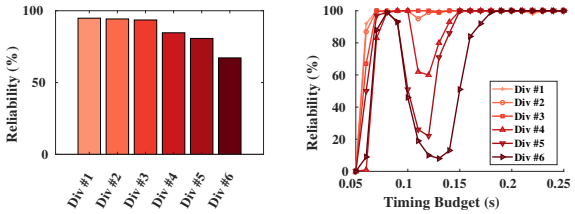
(a) Different strategies (b) Different division schemes

Figure 15: Real Processing Latency vs. Timing Budget

7.4.1 Estimation Error. We show the Estimation Error in Figure 11(b). The approaches of Div #1 and Div #2 perform better than the rest with less than 0.1 estimation error.

7.4.2 Reliability. The reliability for different division schemes is shown in Figure 16(a). In general, division schemes with fewer tiles are more reliable than those with more tiles. The main reason is that fewer tiles means less computation overhead of scaling and hence less mean processing latency, which enables the algorithm with fewer tiles to meet some timing constraints that cannot be met by those with with more tiles.

We also plot the reliability of different division schemes as a function of the timing budget in Figure 16(b). The reliability of division schemes with fewer tiles, e.g., Div #1, first increases as the timing budget increases, and then stabilizes at reliability 1. However, for division schemes with more tiles, e.g., Div #6, it takes more time for their reliability to stabilize at 1.



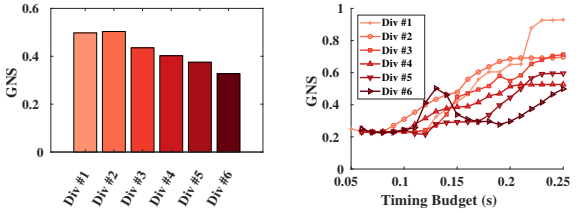
(a) Division Schemes (b) Relation to the timing budget

Figure 16: Reliability

7.4.3 Good-case Normalized Score. *GNS* of different division schemes are plotted in Figure 17(a). Generally, the division scheme with fewer tiles achieves higher *GNS*. The division scheme with more tiles does not perform well because the increased number of tiles causes its critical scaling ratio to be small. This causes the chosen scaling schemes to be conservative that leads to low event scores in more cases.

GNS of different division schemes as a function of the timing budget is shown in Figure 17(b). For all division schemes, *GNS* gradually increases as the timing budget increases while division schemes with fewer tiles have higher *GNS* values.

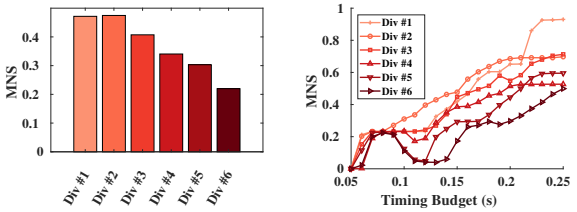
7.4.4 Mean Normalized Score. *MNS* for different division schemes are shown in Figure 18(a). Division schemes with more tiles have higher *MNS* values because of their high reliability and high *GNS* values. *MNS* as a function of the timing budget is shown in Figure 18(b). In general, *MNS* increases with the timing budget and



(a) Across division schemes (b) Relation to the timing budget

Figure 17: Good-case Normalized Score

the division scheme with fewer tiles perform better than those with more tiles.



(a) Across division schemes (b) Relation to the timing budget

Figure 18: Mean Normalized Score

7.5 Summary of evaluation

In this evaluation, we show that LiveTexture is reliable to meet various timing constraints by achieving a reliability of 89.4% and efficient to use available timing budgets by achieving a mean normalized score of 0.511 which is higher than that of most baseline systems. For systems that have comparable mean normalized scores to our system, they are more likely to violate the timing constraint and will suffer when the computation resource is not enough.

8 RELATED WORK

In this section, we present related work on the 360 video live streaming, 360 video stitching and rate adaptation techniques in 360 video streaming.

8.1 360 video live streaming

There have already been researchers working on 360 video live streaming. For instance, [9] has built a 360 video live streaming system using GoPro cameras and boosts the stitching speed by foveated stitching. [22] also builds an interactive 360 video telephony system over LTE using GoPro cameras carried by a drone. There are also consumer grade 360-degree cameras doing live streaming, e.g., GoPro Omni [6], Ricoh Theta S [17], and Samsung Gear 360 [13] that enable users to live stream the video captured by their device to a smartphone or PC application or the website.

8.2 360 video Stitching

360 video stitching consists the camera calibration and the image alignment.

Camera Calibration. Camera calibration is fundamentally the estimation of the camera model. A widely adopted camera model

is the one proposed by Jean-Yves Bouguet [2], where cameras are characterized by the extrinsic, intrinsic parameters and distortion coefficients. The estimation involves feature extraction [1, 11], image matching, rough estimation based on homography and fine adjustment through bundle adjustment [19].

Image Alignment. Image alignment is done by applying the projection map, the blending map and the compensation matrix sequentially to pixels in the input frames [9]. The projection map projects pixels on the input frame to the output panoramic frame. The blending map indicates how the pixel values in the overlapping image area projected from two cameras are decided. The compensation map compensates for the intensity in different cameras.

8.3 Rate adaptation Techniques

Tile-based Approach. For tile-based approaches like 360ProbDash [21] and OpTile [20], a raw panoramic video is divided into video chunks with the same duration. For each chunk, it is spatially divided into N tiles. Then, each tile is encoded into segments with M different bitrate levels. Therefore, $M \times N$ segments will be stored at the server for streaming. The client can pre-fetch the tiles based on the transmission bitrate budget.

Viewport-adaptive Approach. Viewport-adaptive approaches like those adopted by Facebook [8] and Pixvana [10] divide a raw panoramic frame into video chunks as tile-based approaches do. The difference is that each video chunk is encoded into different representations which have different distribution of quality. The client will request different representations based on the prediction of the viewport and the available bandwidth budget.

Projection-based Approach. Projection-based approaches map pixels from a spherical surface to different forms of surface instead of the standard approach of equirectangular projection that maps pixels to a 2D face. Typical examples are Cubic projection [12], offset cubic projection [25] and pyramid projection [14].

9 CONCLUSION

Tile-based approaches have been widely used in on-demand 360 video streaming. However, they cannot be directly applied to live 360 video streaming because they does not perform real-time stitching but instead assumes the video to be stitched before streaming.

In this work, we present an event-driven stitching algorithm for tile-based live 360 video streaming. This algorithm is featured by an event-driven model to abstract various types of semantic information as events and a tile actuator to decide on the optimal tiling scheme. The tile actuator decides on the optimal tiling scheme by employing an offline profiling phase and an online greedy search phase. The offline profiling phase efficiently explores combinations of tiling schemes and events by applying multiple state space reduction techniques and finds the quasi-optimal tiling scheme based on the *Pareto Boundary*. The online greedy search phase fine-tunes the quasi-optimal tiling scheme by modeling an optimization problem and repeatedly solving it.

In the evaluation, we compare our system, LiveTexture, with several baseline systems over different combinations of events and timing budgets. The result shows that the proposed algorithm adapts well to various timing budgets by meeting 89.4% of the timing constraints and uses the timing budgets more efficiently than other baseline algorithms by achieving a mean normalized score of 0.511.

REFERENCES

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. 2006. Surf: Speeded up robust features. In *European conference on computer vision*. Springer, 404–417.

[2] JY Bouguet. 2005. Camera calibration toolbox for matlab. California Institute of Technology. *Computational Vision at CALTECH* (2005).

[3] Xavier Corbillon, Alisa Devlic, Gwendal Simon, and Jacob Chakareski. 2017. Optimal set of 360-degree videos for viewport-adaptive streaming. In *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 943–951.

[4] Google. 2014. Google Cardboard. https://vr.google.com/cardboard/

[5] Google. 2018. Google VR API reference | Google VR | Google Developers. https://developers.google.com/vr/

[6] GoPro. 2002. GoPro. https://gopro.com/

[7] gstreamer. 2018. GStreamer: open source multimedia framework. https://gstreamer.freedesktop.org/

[8] Evgeny Kuzyakov and David Pio. 2016. Next-generation video encoding techniques for 360 video and VR.

[9] Wei-Tse Lee, Hsin-I Chen, Ming-Shiuan Chen, I-Chao Shen, and Bing-Yu Chen. 2017. High-resolution 360 Video Foveated Stitching for Real-time VR. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 115–123.

[10] Pixvana. 2018. An Intro to FOVAS: Field of View Adaptive Streaming for Virtual Reality. https://pixvana.com/intro-to-field-of-view-adaptive-streaming-for-vr/

[11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE international conference on*. IEEE, 2564–2571.

[12] David Salomon. 2007. *Transformations and projections in computer graphics*. Springer Science & Business Media.

[13] Samsung. 2017. Samsung Gear 360. https://www.samsung.com/global/galaxy/gear-360/

[14] Kashyap Kammachi Sreedhar, Alireza Aminlou, Miska M Hannuksela, and Moncef Gabbouj. 2016. Viewport-adaptive encoding and streaming of 360-degree video for virtual reality applications. In *Multimedia (ISM), 2016 IEEE International Symposium on*. IEEE, 583–586.

[15] Thomas Stockhammer. 2011. Dynamic adaptive streaming over HTTP–: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 133–144.

[16] Afshin Taghavi Taghavi Nasrabadi, Anahita Mahzari, Joseph D Beshay, and Ravi Prakash. 2017. Adaptive 360-degree video streaming using scalable video coding. In *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 1689–1697.

[17] Ricoh Theta. 2015. Product | RICOH THETA. https://theta360.com/en/about/theta/s.html

[18] Carlo Tomasi and Takeo Kanade. 1991. Detection and tracking of point features. (1991).

[19] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. 1999. Bundle adjustmentâĂŤa modern synthesis. In *International workshop on vision algorithms*. Springer, 298–372.

[20] Mengbai Xiao, Chao Zhou, Yao Liu, and Songqing Chen. 2017. OpTile: Toward Optimal Tiling in 360-degree Video Streaming. In *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 708–716.

[21] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. 2017. 360ProbDASH: Improving QoE of 360 Video Streaming Using Tile-based HTTP Adaptive Streaming. In *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 315–323.

[22] Xiufeng Xie and Xinyu Zhang. 2017. POI360: Panoramic Mobile Video Telephony over LTE Cellular Networks. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM, 336–349.

[23] Alireza Zare, Alireza Aminlou, Miska M Hannuksela, and Moncef Gabbouj. 2016. HEVC-compliant tile-based streaming of panoramic video for virtual reality applications. In *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 601–605.

[24] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance.. In *NSDI*, Vol. 9. 1.

[25] Chao Zhou, Zhenhua Li, and Yao Liu. 2017. A measurement study of oculus 360 degree video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 27–37.