MiniView Layout for Bandwidth-Efficient 360-Degree Video

Mengbai Xiao
George Mason University
mxiao3@gmu.edu

Shuoqian Wang
SUNY Binghamton
swang130@binghamton.edu

Chao Zhou
SUNY Binghamton
czhou5@binghamton.edu

Li Liu
George Mason University
lliu8@gmu.edu

Zhenhua Li
Tsinghua University
lizhenhua1983@tsinghua.edu.cn

Yao Liu
SUNY Binghamton
yaoliu@binghamton.edu

Songqing Chen
George Mason University
sqchen@gmu.edu

ABSTRACT

With the recent increase in popularity of VR devices, 360-degree video has become increasingly popular. As more users experience this new medium, it will likely see further increases in popularity as users experience its greater immersiveness compared to traditional video streams. 360-degree video streams must encode the omnidirectional view, and, with current encoding techniques, these views require significantly higher bandwidth than traditional video streams. These larger bandwidth requirements comprise the main barrier toward wider adoption by video streaming services.

To reduce bandwidth requirements of 360-degree streaming, we propose the *MiniView Layout*. Compared to the standard cube layout, with equal pixel densities, 360-degree videos encoded in the MiniView Layout can save 16% of the encoded video size while delivering similar visual qualities. In conjunction with the MiniView Layout, we make the following contributions toward improving the 360-degree video ecosystem: i) We create a "projection efficiency" metric that quantifies the efficiencies of sphere-to-2D projections. ii) We introduce the *ffmpeg360* tool. *ffmpeg360* transcodes 360-degree videos and measures comparative 360-degree video quality given user head movement traces. The tool performs these tasks efficiently, using OpenGL for GPU acceleration.

ACM Reference Format:
Mengbai Xiao, Shuoqian Wang, Chao Zhou, Li Liu, Zhenhua Li, Yao Liu, and Songqing Chen. 2018. MiniView Layout for Bandwidth-Efficient 360-Degree Video. In *2018 ACM Multimedia Conference (MM '18), October 22–26, 2018, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3240508.3240705

1 INTRODUCTION

Combined with head mounted display systems, 360-degree video streams provide greater levels of immersiveness than traditional video streams. This combination gives users intuitive control of the viewing experience, providing a sensation closer to "being in the scene". As devices for viewing 360-degree video decrease in price and become more-widely available, their share of video streaming traffic promises to increase as users see the advantages of the immersive 360-degree setting over traditional videos.

A significant bottleneck toward widespread adoption of 360-degree video streaming is its large bandwidth requirements. 360-degree videos encode the omnidirectional view, but users can only consume a limited field of view (FOV), e.g., a view spanning 100-degrees in both the horizontal and vertical directions. To render high quality views of the video, the underlying representation of the 360-degree video must be encoded in much higher quality. For example, to render 1080p 100×100-degree FOVs, the underlying 360 video stream must be 4K resolution or above using current 360-degree video representations. As a result, much of the video data is downloaded during streaming but never consumed (i.e., wasted).

Much existing research has focused on improving the bandwidth efficiency by reducing wasted data via tiling [10, 11, 14–16, 20–22] and offset projections [3, 23]. With tiling, only tiles that overlap with the user's predicted viewport need to be downloaded in high quality, while other tiles can be downloaded in low quality or not downloaded at all. Offset projections have also been proposed to reduce view-level waste. With offset projections, distortion is applied to the spherical surface so that in the resulting rectangular projection, more pixels are devoted to a specific direction on the sphere. Multiple versions of offset-projected videos are encoded, with each version concentrating pixels in a different direction on the sphere. During streaming, only the version whose pixel concentration direction best matches the user's predicted view is downloaded. To achieve the best bandwidth savings without compromising visual quality, both approaches require predicting the user's viewing direction at the time a segment is requested by the client [13]. Accurately predicting the user's head movement, however, is challenging [8, 17]. Specifically, if the predicted viewport significantly deviates from the actual user's viewport during streaming, the user's view will be rendered from low quality tiles or offset projections whose pixels are concentrated at a different spherical direction, resulting in bad visual quality of rendered views.

Besides bandwidth waste due to unviewed video data, a portion of bandwidth inefficiency results from an inability to directly encode

MM '18, October 22–26, 2018, Seoul, Republic of Korea

ACM ISBN 978-1-4503-5665-7/18/10…$15.00
https://doi.org/10.1145/3240508.3240705

and transmit the omnidirectional view 360-degree video provides. The omnidirectional view is most naturally represented as pixels on the surface of a sphere. Modern codecs encode pixels on a plane very efficiently but cannot directly consume spherical pixels. The typical solution to this incompatibility is to first map spherical pixels onto a 2D rectangular plane, then encode the plane as a frame in a standard video stream. This indirection is not without its cost; two types of inefficiency result: *First*, uniformly dense pixels on the sphere's surface cannot be mapped to uniformly dense pixels on a plane. Some portions of the sphere will be over-represented, resulting in inefficiency. For example, the popular equirectangular projection [4] for encoding 360-degree videos over-samples pixels around the poles of the sphere. *Second*, the distortions from these planar projections can affect encoding efficiency. For example, the curvature of lines in an image affects encoding efficiency, so if different projections produce planar representations of the spherical surface with different amounts of curvature, then these projections will likely be encoded with different compression ratios.

In this paper, we focus on improving bandwidth efficiency of 360-degree video streaming by reducing the quantity of over-represented pixels in the spherical-to-2D projection. To better understand how changing the projection can improve bandwidth efficiency, we introduce a method for quantifying the efficiency of any projection. Using this method of determining bandwidth efficiency, we observe that the efficiency of rectilinear projections [7] increases as the size of these projections decrease. For example, a 90×90 degree cube face from the standard cubic projection has a projection efficiency of about 52% while the 30×30 degree portion of the center of the cube face has an efficiency of about 93%.

Following this observation, we propose a new system of efficient projections of spherical pixels we call the **MiniView Layout**. A miniview encodes a small portion of the sphere by applying the rectilinear projection with a small FOV. To construct the MiniView Layout, we selected an efficient set of miniviews that fully cover the spherical surface. This construction must balance three types of inefficiencies: i) projection inefficiency due to rectilinear-projection-incurred redundant pixels, ii) compression inefficiency due to small-sized miniviews, and iii) overlap inefficiency caused by partially overlapping miniviews. To this end, we designed a dynamic programming algorithm for covering the spherical surface with an efficient set of miniviews. This set of miniviews is then laid out on a 2D rectangular plane – thus the name "MiniView Layout" – so that they can be encoded as a single video frame.

To evaluate the quality of these encodings, we created a tool called *ffmpeg360*. *ffmpeg360* both transcodes 360-degree videos into the MiniView Layout and renders these views from an input MiniView Layout. Using *ffmpeg360*, we compared the visual quality (peak signal-to-noise ratio (PSNR) and structural similarity (SSIM)) of views rendered by the MiniView Layout against views rendered by the standard and equi-angular cubic projections. Results show that the MiniView Layout can provide as good visual quality of rendered views while saving 16% storage and downloading bandwidth compared to the standard cubic projection.

This paper makes the following major contributions:

- We introduce a new metric "projection efficiency" to quantify the efficiency of various sphere-to-2D projections.
- We propose the "MiniView Layout" for projecting spherical pixels. The MiniView Layout balances projection efficiency and compression efficiency and can save 16% encoded video size and streaming bandwidth compared to the standard cube while providing similar visual qualities.
- We introduce the tool *ffmpeg360*. *ffmpeg360* transcodes 360-degree videos to and from various spherical projections. It uses OpenGL to accelerate geometry transformation and pixel sampling. Given a trace of user head movements, *ffmpeg360* can create a video of views rendered for the user, making it possible to perform visual quality analysis of the 360-degree video as experienced by the user. We have made the source code of this tool available online.

2 MOTIVATION

2.1 Projection Efficiency

360-degree videos encode information about every direction surrounding the camera location. As a result, pixels in 360-degree videos are most naturally represented as pixels on the surface of a sphere. To encode these spherical pixels, we typically first map them on to a rectangular surface and use standard video codec such as H.264, HEVC to efficiently compress them.

These mappings add redundant pixels and result in projection inefficiency. We characterize projection efficiency as follows:

$$\text{Projection Efficiency} = \frac{\text{area on the spherical surface}}{\text{area on the calibrated projection}}$$

For example, for a unit spherical surface with surface area 4π, a frame generated through the equirectangular projection [4] covers a corresponding rectangular area of $2\pi \times \pi = 2\pi^2$. Thus, if we were able to transmit the spherical surface directly, we would send $4\pi/2\pi^2 \approx 64\%$ of the pixels compared to the equirectangular image.

To compute projection inefficiency, it is necessary to ensure that the size of the projection is calibrated appropriately against the unit sphere. Here, we select the projection size where the lowest pixel density across the set of all generated views from the unit sphere matches the lowest pixel density from a projection. For example, since the pixel density at the center of the 2×2 cube face (in the limit of an infinitesimally small area) matches the pixel density on the unit sphere, we select the $2 \times 2 \times 2$ cube as our calibrated cubic projection.

While it is commonly believed that the cubic projection [2] is more efficient than the equirectangular projection, its overall projection efficiency is actually worse. To cover a unit sphere, six cube faces (from the calibrated cubic projection) each with area 2×2 are needed, requiring a rectangular area of 24. As a result, the projection efficiency of cubic projection is only 52%. Efficiency in the cubic projection decreases with distance from the center of the cube face; areas near the edge of a cube face correspond to smaller areas on the sphere than equal-sized areas at the center of the cube face.

The cubic projection is a special case of the more general rectilinear projection [7]. With the rectilinear projection, we place a plane tangent to the sphere at a single point. To fill a pixel on the plane, we project a ray from center of the sphere to the pixel. This ray also intersects with the surface of the sphere. The pixel value at this intersection point is used to fill the pixel on the plane.

Table 1: Projection efficiency of various schemes projecting an area on the sphere to 2D planes. Degrees by degrees indicates a rectilinear projection of a portion of the spherical surface.

	Area on the Sphere	Area on the Calibrated Projection	Projection Efficiency
Equirectangle	4π	$2\pi^2$	0.637
Equi-angular Cube	4π	$\frac{6}{4}\pi^2$	0.849
Standard Cube	4π	24	0.523
$100^\circ \times 100^\circ$	2.509	5.681	0.442
$90^\circ \times 90^\circ$ (cube face)	$\frac{4}{6}\pi$	4	0.523
$70^\circ \times 70^\circ$	1.341	1.961	0.684
$50^\circ \times 50^\circ$	0.718	0.870	0.826
$30^\circ \times 30^\circ$	0.268	0.287	0.934
$10^\circ \times 10^\circ$	0.030	0.031	0.992
MiniView Layout	4π	15.732	**0.799**

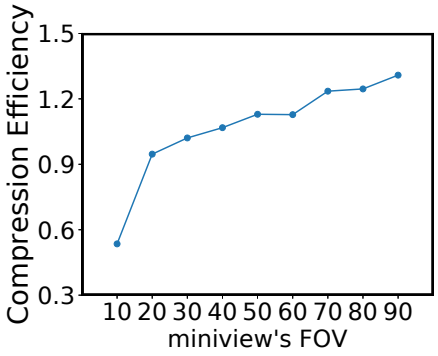


Figure 1: Compression efficiencies of rectilinear projections with different FOVs.

Table 1 shows the projection efficiency of rectilinear projections with different fields of view. To calculate the projection efficiency, we consider a unit sphere and compare the area on the projection with the area on the sphere that is covered by the projection[1]. For rectilinear projections, the projection efficiency increases as the FOV decreases.

2.2 Compression Efficiency

Despite their high projection efficiencies, encoding views with small fields of view suffers from another type of inefficiency: compression inefficiency. With the same pixel density as views with large FOVs, small FOV views encode fewer pixels. As a result, standard codecs typically do not compress segments with small FOV views as efficiently as large FOV views with more pixels.

To illustrate this problem, we selected five 360-degree videos and created 1-second long segments from these videos. We then rendered and encoded views with FOVs varying from 10 degrees to 90 degrees (with vertical FOV equals to horizontal FOV) at a total of 100 randomly selected orientations. For the 90 degrees view, the view's resolution is set to 640x640. For a view with $\theta \times \theta$ FOV, its resolution is set correspondingly to $640 \cdot \tan(\theta/2) \times 640 \cdot \tan(\theta/2)$. For each FOV value, we calculated the average size of encoded views and used this average size to obtain compression efficiency as follows:

$$\text{Compression Efficiency} = \frac{\text{\# of pixels on each frame}}{\text{encoded size in bytes}}$$

Figure 1 shows that as the FOV decreases, compression efficiency also decreases. As a result, if we hope to minimize the storage size of encoded 360-degree videos, we should consider both types of (in)efficiencies.

[1]The integral to calculate area on the sphere is

$$4 \times \int_0^{\frac{\text{v-fov}}{2}} \int_0^{\arctan\left(\tan\frac{\text{h-fov}}{2} \times \cos x\right)} \cos y \, dy \, dx$$

where h-fov is horizontal field of view and v-fov is vertical field of view.

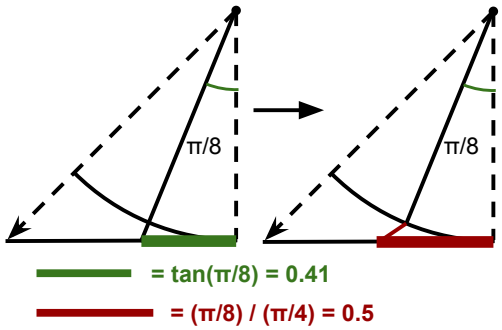


Figure 2: Comparison between standard cube and equi-angular cube (EAC).

2.3 Equi-angular Cube (EAC)

To address the projection inefficiency of cubic projections, Google recently proposed equi-angular cube (EAC) [1]. EAC attempts to address the problem of pixel inefficiency in cube faces by distorting the standard cubic projection to reduce projection inefficiency.

In the standard rectilinear projection, as pixels' distances from the tangent point increase, they require increasingly large amounts of area on the projected plane. The EAC projection distorts the standard rectilinear projection so that areas on the plane more closely match their corresponding areas on the sphere. To do so, pixels on the spherical surface are mapped to pixels on the plane through the following relation. Consider a pixel p on a standard cube face, the pixel's coordinates within the cube face is represented as $p_x \in [-1, 1]$ and $p_y \in [-1, 1]$. To create an equi-angular cube face, we take the same pixels, but re-arrange them so that the coordinates of a pixel q on the equi-angular cube face is represented as:

$$q_x = \frac{4}{\pi} \times \arctan(p_x), \quad q_y = \frac{4}{\pi} \times \arctan(p_y)$$

Figure 2 shows an example how pixels on the standard cube face are re-arranged. Pixels that are close to the center of the cube face are moved farther away from the center on the resulting EAC face.

When the size of the EAC face is $\frac{\pi}{2} \times \frac{\pi}{2}$, the density at the center horizontal and vertical lines on the EAC face matches the corresponding density on the spherical surface. Therefore, the calibrated EAC projection area is $6 \times \frac{\pi}{2} \times \frac{\pi}{2}$. (Recall that the calibrated size must match minimum pixel densities from any generated view generated from the sphere and the projection.)

Create a set of candidate miniviews with pitch varying from 0 to 90 degrees, FOV varying from 10 to 90 degrees, both in 1-degree increments. (Roll of these miniviews is always 0. The yaw direction is aligned with the latitude and therefore can be any value and does not affect our result. We choose yaw=0.)
Create an array S of cost (i.e., size) associated with these candidate miniviews.
procedure PATCH($L, \Delta L, miniview$)
    Calculate b, the maximum longitudinal extent fully covered by the $miniview$ between latitude L and $L - \Delta L$
    return b
procedure COMPUTEMINIVIEWS(S)
    for all valid miniviews **do**
        for all valid $(L, \Delta L)$ for a given miniview **do**
            b = PATCH($L, \Delta L, miniview$)
            Compute cost $c(L, \Delta L, miniview) = \lceil \frac{360}{b} \rceil \times S[miniview]$
    Initialize $R(L) = 0$ for $L \leq 0$; $R(L) = \text{inf}$ otherwise
    for all L **do**
        for all ΔL **do**
            $R(L) = \min\left(R(L), \min_{miniview}\left(c(L, \Delta L, miniview)\right) + R(L - \Delta L)\right)$

Figure 3: Computing an efficient miniview coverage using dynamic programming. $R(L)$ stores the minimum cost calculated to cover the sphere from latitude 0 to L. $c(L, \Delta L, miniview)$ represents the cost using miniview $miniview$ to cover the latitudinal band between L and $L - \Delta L$. The backtracking phase is omitted from the pseudocode.

3 DESIGN OF MINIVIEW LAYOUT

In this section, we propose a new layout designed to address the projection inefficiency issue and save storage size of encoded 360-degree videos without incurring visual quality loss. This new layout encodes 360-degree videos using a set of views created by the rectilinear projection. These views have smaller FOVs compared to cube faces. We thus refer to these views as miniviews.

Each miniview is parameterized by <*orientation, fov, and pixel density*>. Here, *orientation* is its Euler angle (yaw and pitch) relation to a fixed, base set of coordinate axes (we fix roll to 0). Horizontal and vertical fields of view are both set to the same *fov* value. To decide the best set of miniviews, we must balance three types of inefficiencies: projection inefficiency, compression inefficiency, and overlap inefficiency. While miniviews with small FOVs have better projection efficiency compared to cube faces, they can suffer from both compression and overlap inefficiencies. Overlap inefficiency occurs when the same pixels (on the spherical surface) occur more than once in overlapping miniviews used to encode 360-degree views. To efficiently encode 360-degree video, we thus need to select a set of that both fully cover the spherical surface and also incur small costs for both overlap and projection inefficiency.

3.1 Problem Formulation

It is possible to formulate the miniview selection problem as an optimization problem. The optimization problem constrains the selected set of miniviews to fully cover the sphere while minimizing the storage cost of miniview set. Assuming that compression of each miniview occurs independently, the miniview selection problem can be written as follows:

$$\begin{aligned} \text{minimize:} \quad & c^\top x \\ \text{subject to:} \quad & Mx \geq \mathbf{1} \\ & x_i \in \{0, 1\} \, \forall i \end{aligned}$$

To construct this optimization problem, we begin with a (finite) set of candidate miniviews. A vector c estimates the cost (e.g., size) of each candidate miniview. Solution to the optimization problem, x, is a binary vector that encodes the presence of candidate miniviews in the selected optimal set. We also need to represent the area on the sphere a miniview can cover and check if the selected set of miniviews can fully cover the entire sphere. To do so, we can decompose the spherical surface into discrete sub-areas and encode each miniview's coverage of these areas in a matrix M.

Regardless of how the sphere is discretized, the optimization problem formulated above is a weighted set cover problem [12] and is NP-hard. Optimal solutions to the problem are not feasible to compute exactly. While a greedy algorithm produces a polynomial time approximation to the weighted set cover problem, the solutions produced by the greedy approximation are ineffective for the miniview domain.

3.2 MiniView Selection using Dynamic Programming

To effectively solve this problem in polynomial time, we propose a structural heuristic based on a decomposition of the spherical geometry. We discretize a hemisphere (from the equator to the north pole) into horizontal bands, where each band is described by a latitude range. We then consider each band of latitude independently. This decomposition allows us to formulate dynamic program based on a cost function for each band.

Specifically, the dynamic programming algorithm computes the values of function $R(L)$, the minimum cost needed to fully cover portion of the sphere from latitude 0 to latitude L, where L is an integer between 1 and 90. Here, cost represents the total number of bytes of all miniviews selected to cover the corresponding area on the sphere. $R(L)$ can be calculated using the following recurrence:

$$R(L) = \min_{\Delta L}\big(R(L - \Delta L) + C(L, \Delta L)\big), \tag{1}$$

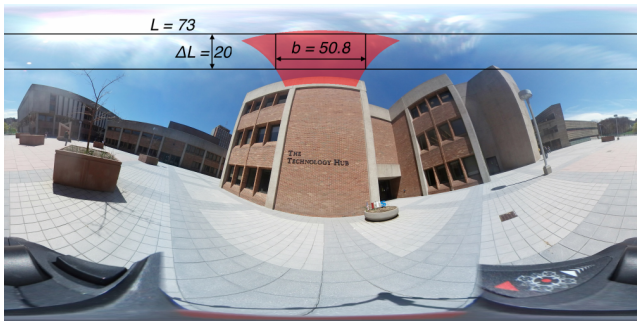


Figure 4: In this figure, we show an equirectangular image. The red shaded area represents the spherical area covered by a miniview oriented at $< pitch = 60, yaw = 0 >$ with horizontal and vertical FOV of 30 degrees. Given $L = 73, \Delta L = 20$, we calculate the maximum longitudinal extent b the miniview covers is 50.8. If we want to cover the entire band between latitude L and $L - \Delta L$ with miniviews with the same pitch and fov, we need $n = \lceil \frac{360}{b} \rceil = 8$ such miniviews.

where $C(L, \Delta L)$ is the cost needed to cover portion of the sphere from latitude $L - \Delta L$ to latitude L. Figure 3 shows the dynamic programming algorithm used to solve this recurrence. We first calculate $c(L, \Delta L, miniview_{(pitch,fov)}) = n \times S(miniview_{(pitch,fov)})$, representing the cost needed to cover the latitude band between L and $L-\Delta L$ using miniviews with pitch of $pitch$ and fov of fov. Here, $S(miniview)$ represents the size of $miniview$, and n represents the number of such miniviews needed. $C(L, \Delta L)$ can thus be calculated as $\min_{miniview} \left(c(L, \Delta L, miniview)\right)$. The complexity of this algorithm is $O(|L|^3 \cdot |F|)$, where $|L|$ is the number of latitude bands, and $|F|$ is the number of all available field of view (FOV) values, i.e., angular dimensions of each miniview.

To obtain n, we calculate the maximum longitudinal extent b the miniview covers between latitude L and $L - \Delta L$. Figure 4 illustrates how b is calculated. To fully cover the latitudinal band, $n = \lceil \frac{360}{b} \rceil$ $miniview_{(pitch,fov)}$s are needed.

To obtain the encoded sizes of miniviews, we interpolate a function $f(\cdot)$ using the average sizes of miniviews of different FOVs we obtained in Section 2.2. $f(\cdot)$ takes as input a miniview's area in pixels and outputs its encoded size in bytes. $S(miniview)$ is thus calculated as $f(area(miniview))$. Figure 5 shows our interpolation function's mapping of miniview pixel counts to encoded miniview sizes.

Note that the average sizes of these miniviews should be an estimate of the number of bytes used to encode the miniview within the layout image. In addition, depending on the set of videos encoded by the layout, the mapping from number of pixels to encoded size could differ. For instance, in the most extreme case, a different function, f, could be estimated for each encoded video, resulting in different miniview layouts for each video in a dataset.

We use the dynamic programming algorithm to calculate $R(90)$ and record the set of miniviews selected to cover each latitudinal band. Table 2 shows the parameters of all miniviews used to cover the sphere. A total of 82 miniviews are used. We then lay the miniviews out on a rectangular plane as in Figure 6.

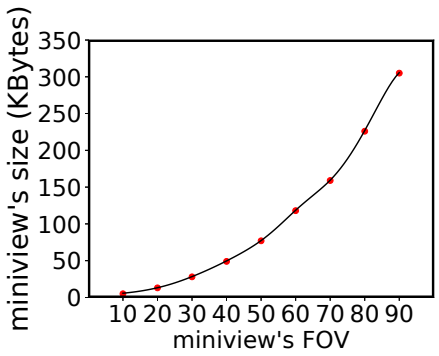


Figure 5: We fit a miniview's size as a function of the number of pixels to be encoded. $S(miniview) = f(area(miniview)))$, where $f(\cdot)$ is interpolated using points indicated in circle marker. All miniviews were created so that their pixel densities match those of a miniview with 90-degree horizontal and vertical FOV containing 640x640 pixels. That is, a miniview with $\theta \times \theta$ FOV was generated to comprise a $640 \cdot \tan(\theta/2) \times 640 \cdot \tan(\theta/2)$ image. (All miniviews are square.)

Table 2: Yaw and pitch values of miniview orientations (in degrees) in the MiniView layout. A total of 82 miniviews are used for encoding a 360-degree video in this layout. (This table omits roll, which is always 0.)

FOV	Pitch	Yaw
44	90	0
20	59	0, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300, 330
26	37	0, 27.7, 55.4, 83.1, 110.8, 138.5, 166.2, 193.8, 221.5, 249.2, 276.9, 304.6, 332.3
25	12	0, 24, 48, 72, 96, 120, 144, 168, 192, 216, 240, 264, 288, 312, 336
25	-12	0, 24, 48, 72, 96, 120, 144, 168, 192, 216, 240, 264, 288, 312, 336
26	-37	0, 27.7, 55.4, 83.1, 110.8, 138.5, 166.2, 193.8, 221.5, 249.2, 276.9, 304.6, 332.3
20	-59	0, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300, 330
44	-90	0

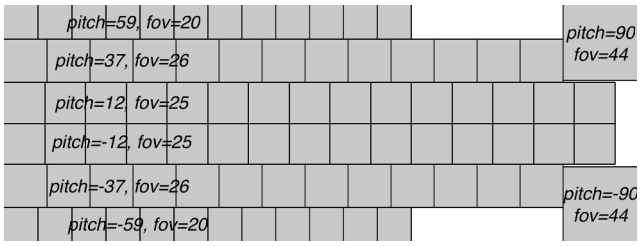


Figure 6: The final MiniView layout, placing 82 miniviews on a rectangular frame.

Projection efficiency. MiniView layout's projection efficiency is shown in the bottom row in Table 1. With **almost 80% projection efficiency**, the MiniView layout outperforms both the equirectangular projection and the cubic projection.

4 IMPLEMENTATION

To encode 360-degree videos in the MiniView Layout, we created a tool called *ffmpeg360*.[2] *ffmpeg360* extends the ffmpeg [5] source code and implements a new *video filter* called *360-project filter*. It uses OpenGL [6] to speed up geometry transformation and pixel sampling. We implemented different fragment shaders to perform pixel sampling required by different input and output projection types.

Input 360-degree video. *ffmpeg360* can take input 360-degree videos encoded using various projection and layout schemes including the equirectangular projection (`equi`), the standard cubic projection (`cube`), the equi-angular cubic projection (`eac`), and the miniview layout (`mvl`). Information about each face/miniview is provided in an *input layout* file. Each line in this file contains the following information about a face/miniview of the input 360 video:

`w:h:x-fov:y-fov:x-rotation:y-rotation:z-rotation:u:v`

Here, `w,h` are the width and height of this face/miniview normalized against the input frame's width and height. `x-fov,y-fov` are the horizontal and vertical FOVs of the corresponding face/miniview. For example, the horizontal and vertical FOVs of both `cube` and `eac` faces are 90 degrees. `*-rotation` represents the orientation of the equirectangular projection, face, or miniview around the x, y, or z axis. Finally, `u,v` are the coordinates of the upper left corner of the face/miniview on the input frame.

Frame-level processing. Given a 360-degree frame and the orientation and FOV of a view, our *360-project filter* can output the "filtered"/transformed frame that represents the view rendered from a 360-degree frame. Note that miniviews in `mvl` and cube faces in `cube` are all rendered views and can be generated using this filter. Using a different fragment shader, we can also use this filter to generate `eac` faces.

Output 360-degree video. After filtered frames are generated, we place these un-encoded frames on a single frame using ffmpeg's *overlay* filter. An *output layout* file is used to specify the position of each face/miniview on the final composite frame. These frames are then sent to the video encoder. Using this pipeline, we can transform 360-degree videos into various projection and layout schemes provided as long as input and output layout files are given.

Output video of rendered views. Given traces of a user's head movement during 360-degree video playback, we can also use *ffmpeg360* to generate views users observe over time and encode them into lossless videos. This allows us to create realistic videos mimicking a user's immersive watching experience. These videos can then be used for comparing the visual quality of views rendered from different projection and layout schemes.

5 EVALUATION

For evaluation, we compare our proposed *MiniView Layout* against two existing 360-degree video projection methods: the *Standard Cube* and the *Equi-angular Cube*. Figure 7 shows the resulting images when one 360-degree image is encoded using these three different methods. We focus our evaluation on the following three metrics: (i) compressed video size, (ii) visual quality of views rendered from 360-degree videos encoded by each method, and (iii) decoding and rendering times.

[2] https://github.com/bingsyslab/ffmpeg360

We use two public-available 360-degree video datasets [9, 19]. Dataset-1 [9] contains five 360-degree videos and 58 users' view orientations (i.e., head movement data) when viewing these videos in head-mounted displays. Dataset-2 [19] contains 18 360-degree videos and 48 users' head movement traces. These datasets allow us to extract a set of views representing realistic patterns of 360 video consumption. To avoid biasing our evaluation toward the results from longer videos, we divide each video into 1-second long segments and uniformly select 10 segments from each video for evaluation. In total, we consider 230 segments selected from all 23 videos from two datasets.

For each selected segment, we start with a high quality baseline video included in these datasets, encoded using the equirectangular projection. We refer to this baseline video segment as BASE. We then use this BASE video segment as input to our *ffmpeg360* tool to generate three videos: standard cube video segment in 1920x1280 resolution (CUBE), equi-angular video segment in 1920x1280 resolution (EAC), and MiniView Layout video segment in 2240x832 resolution (MVL). These generated CUBE and EAC video segments have the same number of pixels, and the CUBE and MVL video segments have the same pixel density. Here pixel density indicates the average number of pixels per unit area over the entire projected plane. Note that this pixel density value does not represent pixel density on the spherical surface. Pixel density on the spherical surface will vary depending on both the projection type and the pixel's position on the sphere.

We compare the sizes of these encoded CUBE, EAC, and MVL segments to gain insights on their compression efficiencies. We further use all users' head movement traces associated with a segment to generate rendered views and compare their visual quality against views rendered from a high quality baseline video. Finally, we compare the time needed to decode and render views from 360-degree videos encoded using different methods.

5.1 Pixel count and compressed size

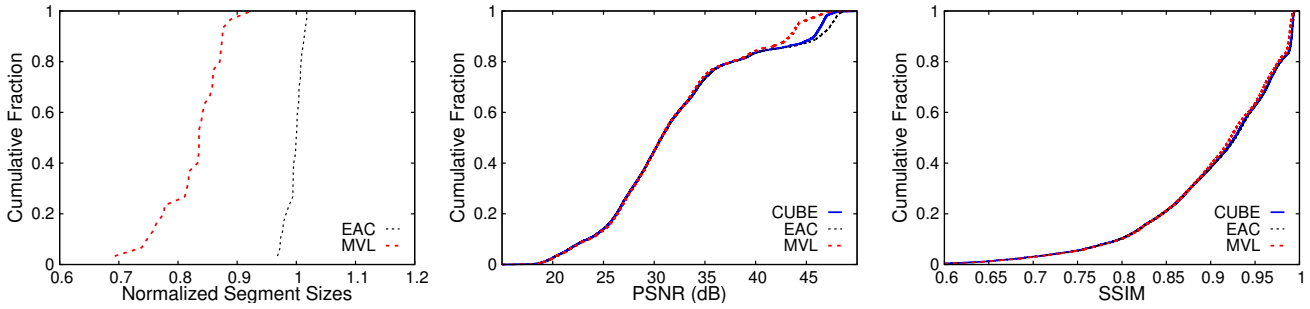
With the same pixel density, MVL video segments need 24.17% fewer pixels to encode the complete spherical view compared to CUBE. We further compare the compressed sizes of MVL segments with CUBE and EAC. For fair comparison, we use `x264` to encode these segments and configure x264 to use the same constant rate factor (`crf=23`). In this way, all three methods are subject to the same level of compression. Because the 23 videos in our datasets exhibit different characteristics, we categorize them into two types: *moving-camera videos* and *static-camera videos*.

For each segment, we normalize encoded EAC and MVL sizes against encoded CUBE size. Over all video segments, MVL segments consume on average 16% less space compared to CUBE segments. This indicates that miniviews have smaller compression efficiency compared to cube faces – 24% savings in pixel count only results in 16% savings in compressed size. However, the miniview layout has much higher projection efficiency compared to the standard cubic projection. As a result, the final compressed MVL segments are significantly smaller than CUBE segments.

The results for moving-camera videos and static-camera videos are shown in Figure 8(a) and Figure 9(a). For static-camera videos, EAC segments with the same number of pixels consume slightly

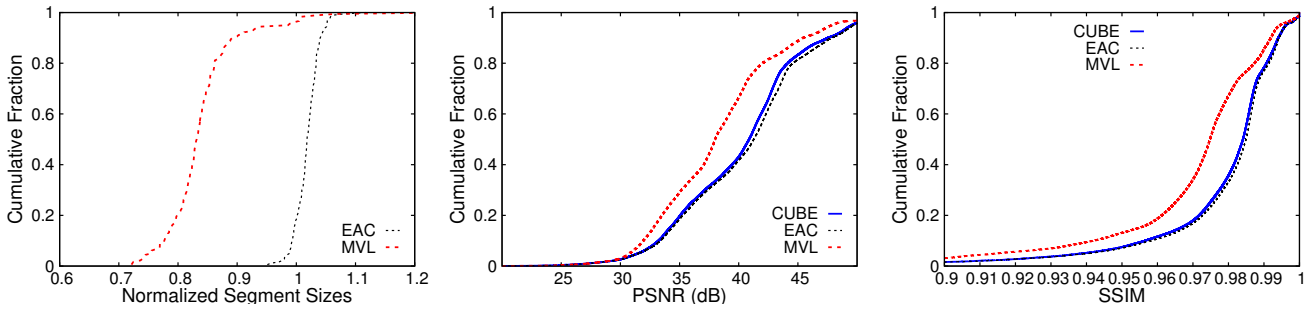
(a) Standard Cube (1920x1280) (b) Equi-angular Cube (1920x1280) (c) The MiniView Layout (2240x832)

Figure 7: A 360-degree image encoded in standard and equi-angular cube as well as our proposed MiniView layout.



(a) (b) (c)

Figure 8: Moving-camera videos: Figure (a) shows the distribution of encoded EAC and MVL segment sizes normalized against encoded CUBE segment sizes. MVL segments have the smallest sizes. Figures (b) and (c) show the distribution of PSNR and SSIM between views generated from CUBE/EAC/MVL segments and views generated from baseline segments. The three projection and layout schemes exhibit almost the same visual quality.



(a) (b) (c)

Figure 9: Static-camera videos: Figure (a) shows the distribution of encoded EAC and MVL segment sizes normalized against encoded CUBE segment sizes. MVL segments have the smallest sizes. EAC segment sizes are slightly bigger than CUBE segment sizes. Figures (b) and (c) show the distribution of PSNR and SSIM between views generated from CUBE/EAC/MVL segments and views generated from baseline segments. EAC segments perform the best.

more space compared to CUBE segments – both the median and mean ratio between EAC and CUBE is 1.02. MVL segments consume less space than CUBE segments – the median and mean ratios are 0.83 and 0.84, respectively. For moving-camera videos, both the median and mean ratios between EAC and CUBE are 1.00. MVL consistently saves storage space, with the median and mean ratios being 0.84 and 0.83, respectively.

5.2 Visual quality

Next, we compare the visual qualities of views rendered from MVL, EAC, and CUBE segments. In these experiments, segments are all encoded using x264 with the same `crf` setting of 23.

We consider user views to span 100-degree vertical and horizontal FOVs at a resolution of 800x800. For each video segment, we decode all its frames and select a user's head movement trace over the period of this segment's playback time from the dataset. For each frame, we render the views user observe based on the extracted view orientations and encode these rendered views using lossless H.264 encoding (this can be achieved by setting `crf=0` in x264). Note that as a user's head position may change over a very short period of time, view generation in a segment is frame-based.

For visual quality analysis, we compare views generated by CUBE, EAC, and MVL segments with views generated by their corresponding baseline BASE segments. We use objective visual quality

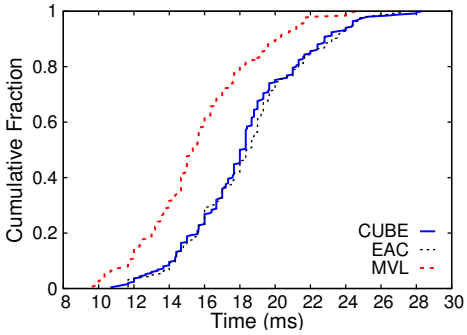


Figure 10: Distribution of CUBE/EAC/MVL per-frame decoding and rendering times.

metrics PSNR and SSIM [18]. Both metrics have been implemented by ffmpeg. Given a pair of videos for comparison, ffmpeg can output these visual quality metrics frame-by-frame. We thus report the PSNR and SSIM between CUBE-, EAC-, MVL-generated view videos and BASE-generated view videos using ffmpeg-calculated statistics directly.

Figure 8 show the PSNR and SSIM of views generated from moving-camera videos. We can observe that the three types of layouts exhibit almost the same visual quality. The median values of PSNR between BASE and CUBE/EAC/MVL are 30.72/30.74/30.67 dB, respectively. The median SSIM values are 0.9269 (CUBE), 0.9282 (EAC), 0.9223 (MVL), respectively. For static-camera videos, the results are shown in Figure 9. We can observe that the MVL views have slightly lower visual quality: the median PSNR (SSIM) values are 40.88 dB (0.98), 41.27 dB (0.98), and 38.01 dB (0.97) for CUBE, EAC, and MVL, respectively.

5.3 Decoding and rendering time

Finally, we focus on MVL's performance on the client side. Specifically, we investigate if it takes longer to render views from MVL segments compared to CUBE and EAC segments. To do so, for each video segment, we measure the time it takes to decode and render views with changing orientation. This emulates the decoding and rendering time during real 360-degree video playback. To measure this time, we use a similar setup using *ffmpeg360* on a Linux computer as our previous experiments. However, since we focus on the decoding and rendering time here, we do not encode or store views after they are rendered. Depending on different videos' frame rates, the number of frames in 1-second long segments may be different. We thus report the average decoding and rendering time per frame.

The results are shown in Figure 10. Decoding and rendering MVL-encoded video is faster than both CUBE and EAC. The median times to decode and render one frame in 1-second long CUBE, EAC, and MVL segments are 18 ms, 18.4 ms, and 15.3 ms, respectively.

6 DISCUSSION

A number of extensions to the MiniView Layout are possible that could improve performance under the right scenario. We have not evaluated these extensions directly as they do not affect the performance of core MiniView scheme.

The MiniView is naturally extensible to tiling strategies. In these approaches, rather than decomposing a source projection into equal-sized tiles, each miniview would be independently encoded into segments, and these segments would be requested independently by the streaming client as needed. As with other projection types, the MiniView Layout can also be encoded as offset representations by moving the camera perspective within the sphere. Applying the equi-angular transformation to miniviews is also possible. The mechanism is the same for each miniview as for each cube face. However, since miniviews are typically smaller than cube faces, the corresponding improvements in projection efficiency will not be as much. The idea of applying the equi-angular transform to the miniview can be extended to using other non-rectilinear projections as miniviews. The requirements for such application would be the same for the rectilinear transform: i) the set of miniviews must cover the sphere and ii) projection efficiency should increase as the size of the miniview decreases.

7 CONCLUSION

To develop better understandings of projections used in 360-degree video, this work introduces a "projection efficiency" metric. "Projection efficiency" quantifies the efficiencies of sphere-to-2D projections. Our analysis of different projections show that the degree of efficiency improvements of smaller rectilinear projections over the larger 90×90 projections used in the standard cube.

This analysis motivated the *MiniView Layout*. The MiniView layout encodes the 360-degree view using smaller-FOV projections than the standard cube, increasing projection efficiency. However, it must balance this increased efficiency against losses from both encoding efficiency, which decreases as the miniview size decreases, and overlap inefficiency, which is incurred when multiple miniviews cover the same set of spherical pixels.

To measure the MiniView Layout's performance, we developed the *ffmpeg360* tool. *ffmpeg360* transcodes 360-degree videos and measures comparative 360-degree video quality given user head movement traces, allowing us to compare MiniView performance against the Standard Cube and Equi-angular Cube. Experiments were run efficiently, as *ffmpeg360* accelerates encoding and rendering through OpenGL's GPU interface.

From these experiments, we found that at equal pixel densities, the MiniView layout saves 16% of the encoded video size while delivering similar visual qualities for high-motion videos compared to both the Standard Cube and Equi-angular Cube layouts.

The MiniView Layout is fast to decode and render, as it requires fewer pixels than other approaches. It is extensible to tile-based approaches, compatible with offset and transformation techniques (e.g., the equi-angular transform), and the layout can be adapted to specific videos or datasets potentially improving its efficiency under different video-specific conditions.

8 ACKNOWLEDGEMENT

We appreciate constructive comments from anonymous referees. This work is partially supported by National Key R&D Program of China under grant 2018YFB1004700, by NSF under grants CNS-1524462 and CNS-1618931, and by gift funding from Adobe Research.

REFERENCES

[1] Bringing pixels front and center in VR video. https://blog.google/products/google-vr/bringing-pixels-front-and-center-vr-video/.

[2] Cubic Projection. http://wiki.panotools.org/Cubic_Projection.

[3] End-to-end optimizations for dynamic streaming. https://code.facebook.com/posts/637561796428084/end-to-end-optimizations-for-dynamic-streaming/.

[4] Equirectangular Projection. http://mathworld.wolfram.com/EquirectangularProjection.html.

[5] FFmpeg. http://www.ffmpeg.org/.

[6] OpenGL. https://www.opengl.org/.

[7] Rectilinear Projection. https://wiki.panotools.org/Rectilinear_Projection.

[8] Yanan Bao, Huasen Wu, Tianxiao Zhang, Albara Ah Ramli, and Xin Liu. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1161–1170, 2016.

[9] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 360-degree video head movement dataset. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 199–204. ACM, 2017.

[10] Xavier Corbillon, Alisa Devlic, Gwendal Simon, and Jacob Chakareski. Optimal set of 360-degree videos for viewport-adaptive streaming. *in Proc. of ACM Multimedia (MM)*, 2017.

[11] Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. Viewport-adaptive navigable 360-degree video delivery. In *Communications (ICC), 2017 IEEE International Conference on*, pages 1–7. IEEE, 2017.

[12] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[13] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. Fixation prediction for 360 video streaming in head-mounted virtual reality. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 67–72. ACM, 2017.

[14] Mario Graf, Christian Timmerer, and Christopher Mueller. Towards bandwidth efficient adaptive streaming of omnidirectional video over http: Design, implementation, and evaluation. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 261–271. ACM, 2017.

[15] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. An http/2-based adaptive streaming framework for 360 virtual reality videos. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 306–314. ACM, 2017.

[16] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 1–6. ACM, 2016.

[17] Liyang Sun, Fanyi Duanmu, Yong Liu, Yao Wang, Yinghua Ye, Hang Shi, and David Dai. Multi-path multi-tier 360-degree video streaming in 5g networks. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 162–173. ACM, 2018.

[18] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.

[19] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. A dataset for exploring user behaviors in vr spherical video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 193–198. ACM, 2017.

[20] Mengbai Xiao, Chao Zhou, Yao Liu, and Songqing Chen. Optile: Toward optimal tiling in 360-degree video streaming. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 708–716. ACM, 2017.

[21] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. 360probdash: Improving qoe of 360 video streaming using tile-based http adaptive streaming. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 315–323. ACM, 2017.

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

[23] Chao Zhou, Zhenhua Li, and Yao Liu. A Measurement Study of Oculus 360 Degree Video Streaming. In *Proceedings of the 8th International Conference on Multimedia Systems*. ACM, 2017.