On the Effectiveness of Offset Projections for 360-Degree Video Streaming

CHAO ZHOU, SUNY Binghamton
ZHENHUA LI, Tsinghua University
JOE OSGOOD and YAO LIU, SUNY Binghamton

A new generation of video streaming technology, 360-degree video, promises greater immersiveness than standard video streams. This level of immersiveness is similar to that produced by virtual reality devices—users can control the field of view using head movements rather than needing to manipulate external devices. Although 360-degree video could revolutionize the streaming experience, its large-scale adoption is hindered by a number of factors: 360-degree video streams have larger bandwidth requirements and require faster responsiveness to user inputs, and users may be more sensitive to lower quality streams.

In this article, we review standard approaches toward 360-degree video encoding and compare these to families of approaches that distort the spherical surface to allow oriented concentrations of the 360-degree view. We refer to these distorted projections as offset projections. Our measurement studies show that most types of offset projections produce rendered views with better quality than their nonoffset equivalents when view orientations are within 40 or 50 degrees of the offset orientation. Offset projections complicate adaptive 360-degree video streaming because they require a combination of bitrate and view orientation adaptations. We estimate that this combination of streaming adaptation in two dimensions can cause over 57% extra segments to be downloaded compared to an ideal downloading strategy, wasting 20% of the total downloading bandwidth.

CCS Concepts: • **Information systems** → **Multimedia streaming**; • **Computing methodologies** → **Virtual reality**;

Additional Key Words and Phrases: 360-degree video streaming, offset projection, visual quality, adaptive streaming

ACM Reference format:
Chao Zhou, Zhenhua Li, Joe Osgood, and Yao Liu. 2018. On the Effectiveness of Offset Projections for 360-Degree Video Streaming. *ACM Trans. Multimedia Comput. Commun. Appl.* 14, 3s, Article 62 (June 2018), 24 pages.
https://doi.org/10.1145/3209660

1 INTRODUCTION

As hardware platforms have matured and network capabilities have increased, video streaming has evolved from low-resolution, desktop-displayed videos to higher-resolution videos that are often played on mobile devices. We are now at the cusp of another transition in video streaming. A larger number of streaming providers are making 360-degree videos available. These 360-degree videos enable users to view content from a full spherical panorama rather than from a

Authors' addresses: C. Zhou, J. Osgood, and Y. Liu, Department of Computer Science, State University of New York at Binghamton, P.O. Box 6000 Binghamton, NY 13902; emails: {czhou5, josgood2, yaoliu}@binghamton.edu; Z. Li, East Main Building - Area 11, Room 212, Tsinghua University, Beijing, China 100084; email: lizhenhua1983@tsinghua.edu.cn.

https://doi.org/10.1145/3209660

fixed viewpoint. These 360-degree videos are viewable in browsers, on mobile devices, and, most recently, within virtual reality (VR) devices such as Samsung's GearVR. These viewing venues represent a continuum on the spectrum of immersiveness—control of the viewport can be actuated through standard peripheral devices such as the mouse and keyboard or sensors on mobile devices, either by manually moving the device or moving the device in sync with one's head movements, simulating real-life view changes.

This greater level of immersiveness does not come without cost. 360-degree videos encode the full omnidirectional views in high quality, requiring more storage space and more bandwidth to transmit over the network. Nowadays, many 360-degree videos are available in 4K quality. That is, each frame encodes 360-degree views using approximately 4,000 horizontal pixels and 2,000 vertical pixels. To stream 4K videos, Netflix recommends the Internet speed be at least 25Mbps [6]. On the other hand, according to Akamai's most recent report, the average broadband connection speed in the United States is only 15.3Mbps [1]. When the high-bandwidth requirement cannot be met, users may experience nonsmooth playback and degraded streaming quality.

To address this bandwidth scarcity problem, many 360-degree video streaming service providers have been actively working to address the concerns in encoding and transmitting 360-degree videos. Much of this effort has gone into encoding schemes that reduce the amount of information transmitted over the network during streaming. Because people do not generally view the entire spherical surface in a single video frame, much of the transmission bandwidth (the unviewed portions of the 360-degree view) is wasted. These schemes typically encode 360-degree views so that more information is devoted toward a particular direction matching the users' viewing sections, wasting less information on unviewed areas.

Facebook recently introduced what we refer to in this article as the "offset cubic projection" in its Oculus system. With this scheme, an offset is applied to distort the spherical surface. The distorted spherical surface is then projected to six cube faces, creating the offset-cubic-projected frame. In the resulting frame, more pixels in the encoding are mapped to views in one pixel-concentrated direction of the offset cube than in other directions. For each frame, Oculus encodes separate 360-degree views centered at a 22-pixel-concentrated direction on the sphere. It also encodes four quality levels of these sets of 22 images (for a total of 88 encoded images per frame) to allow for streaming at different available network bandwidths.

The idea of applying offset to distort the spherical surface to concentrate pixels toward a certain orientation can be applied to other basic projection schemes as well, including the equirectangular projection and the barrel projection. Facebook recently also proposed a new variety of the offset projection scheme—applying the offset to the horizontal plane only.

In this article, we study the offset cubic projection used in Oculus as well as eight other projection schemes. These nine schemes are the result of combining three basic projections: equirectangular, cubic, and barrel, with three offset options: no offset, offset, and offset on the horizontal plane only. We compared views rendered from various projection schemes against views rendered from a high-quality reference image to generate visual quality measures (PSNR and SSIM). Results show that most types of offset projections produce rendered views with better quality than their nonoffset equivalents when view orientations are within 40 or 50 degrees of the offset orientation.

Encoding 360-degree videos using oriented projections brings a new challenge to adaptive streaming: the streaming client not only needs to perform bitrate adaptation but also needs to select the streaming segment whose offset orientation best matches the user's view orientation. We refer to this new adaptation as "view orientation adaptation."

To this end, we attempt to gauge the performance of the Oculus streaming client's adaptation algorithm. We perform two types of experiments. First, we measure streaming performance during human viewing of the Oculus video stream. Next, we perform a more controlled set of experiments

using a mechanical testbed where a VR device is mounted on a rotating platform. We rotate the device at different rates while streaming the 360-degree video to understand how Oculus selects (and discards) segments during streaming. We found that the number of abrupt head movements has a significant impact on the number of downloaded-but-not-played (i.e., wasted) segments. Our findings lead to the conclusion that Oculus's segment selection strategy can cause over 57% extra segments to be downloaded compared to an ideal downloading strategy, wasting 20% of the total downloading bandwidth.

This article extends our preliminary conference publication in proceedings of ACM MMSys 2017 [29]. In this article, we conduct a more thorough study of offset projections. We conducted experiments on a new set of 3,000 randomly selected views produced by these offset projections. We also increased our experimental coverage over the conference paper by (1) evaluating the effectiveness of different offset values and (2) applying offsets to not only the cubic projection but also the equirectangular and barrel projections. We also considered multiple offset types—applying the offset both according to the standard procedure and on the horizontal plane only. Our measurements are accompanied by descriptions of these six combinations of offset projections and procedures for generating projected frames.

The remainder of this article is organized as follows: Section 2 discusses the background and related work. We describe the details of the offset projections in Section 3 and Oculus's use of offset cubic projection in Section 4. We then investigate the visual quality produced by offset projections in Section 5. Section 6 focuses on Oculus's streaming adaptation of offset cubic projections over both bitrate and offset cube orientation. Finally, we make concluding remarks in Section 7.

2 BACKGROUND AND RELATED WORK

2.1 Spherical Projections

Historically, 360-degree video applications have resorted to first mapping spherical pixels to a rectangular surface, then encoding these rectangular images as they would standard planar videos.

These mappings from the sphere to rectangular images have developed from simpler mappings that are easy to encode and render to more complicated mappings that more efficiently represent the sphere at a given viewing angle.

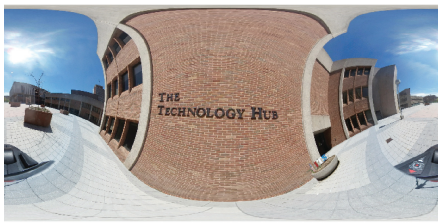
The earliest attempt at encoding the sphere was the **equirectangular projection** [5, 25]. This projection is similar to projections used to display maps of the world. To construct the equirectangular projection, angles on the sphere given by yaw and pitch values[1] (in degrees) are discretized and mapped to pixels on a rectangular image with $x = (\text{yaw} + 180)/360 \times \text{width}$ and $y = (90 - \text{pitch})/180 \times \text{height}$. Here we consider the size of the equirectangular image to be width × height, the center of the equirectangular image is at $<\text{yaw} = 0, \text{pitch} = 0>$, and the pitch angle increases in the upward direction.

Many 360-degree video streaming services today encode videos using the equirectangular projection including YouTube and Jaunt VR. Figure 1(a) shows a frame that uses the equirectangular projection. A significant disadvantage of the equirectangular projection is that it encodes the poles of the sphere with many more pixels than the equator, potentially wasting storage space.

To address the inefficiency in the equirectangular projection, Facebook proposed the **barrel projection** [4]. The barrel projection encodes the top and bottom quarter of the equirectangular projection as two circles, using a much smaller number of pixels for these top and bottom caps. The barrel caps are created by generating views orientated at the poles on the sphere with a

[1]The equirectangular projection is typically parameterized in terms of longitude and latitude. For consistency, we use yaw and pitch here. These are equivalent to longitude and latitude as long as the rotation in the yaw direction is performed before the rotation in the pitch direction and we consider latitudes in the southern hemisphere as negative.

(a) The standard equirectangular projection.



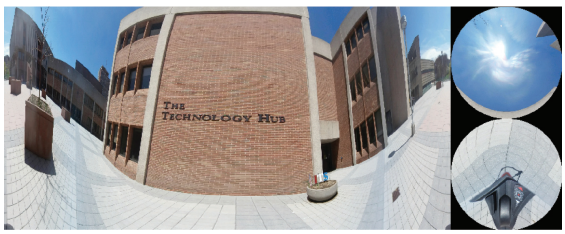
(b) The offset equirectangular projection.

(c) The offset equirectangular projection with horizontal offset only.

Fig. 1. A 360-degree image encoded in various equirectangular projections.

(a) The standard barrel projection.

(b) The offset barrel projection.

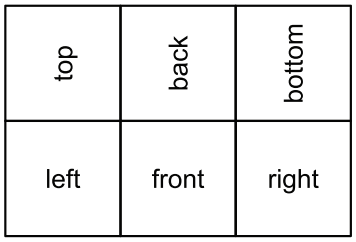


(c) The offset barrel projection with horizontal offset only.

Fig. 2. A 360-degree image encoded in various barrel projections.

90-degree-by-90-degree field of view (FOV) and then setting all pixels outside of a circular radius to black. The middle portion of the equirectangular image is then stretched to occupy the left 80% of the barrel layout. Figure 2(a) shows an example frame that is projected using the barrel projection.

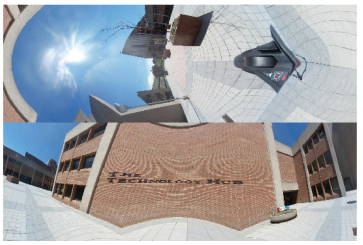
Another type of projection is the **cubic projection** [3, 25]. In this projection, a cube is constructed around the sphere. Rays are projected outward from the center of the sphere, and each ray intersects with both a location on the spherical surface and a location on a cube face. Pixels on the spherical surface are mapped to corresponding pixels on the cube through the mapping produced by these projected rays. For example, a pixel at <yaw = 0, pitch = 0> is pointed to the center of the cube's front face. For encoding, cube faces are arranged on a planar image and compressed using standard video compression techniques. The standard cubic projection is more space efficient than the equirectangular image—Facebook claims that the cubic representation saves about 25% of the video size compared to the original equirectangular representation [8, 14]. Based on our measurement findings, Facebook uses the standard cubic projection to encode 360-degree videos for streaming on web browsers and mobile devices. Figure 3(b) shows the same video frame as Figure 1(a) but encoded using the standard cubic projection for serving 360-degree videos in web browsers. Figure 3(a) shows Facebook's arrangement of the six faces of the cube onto a rectangular surface for video compression.



(a) Arrangement of the six cube faces.

(b) The standard cubic projection.

(c) The offset cubic projection.



(d) The offset cubic projection with horizontal offset only.

Fig. 3. A 360-degree image encoded in various cubic projections.

2.2 View Inefficiency in 360-Degree Video Streaming

A potential inefficiency of the spherical projections described above is that during 360-degree video playback, the FOV rendered and presented to end-users is much smaller than the encoded 360-degree image. Streaming the entire 360-degree view typically leads to large numbers of unrendered pixels, wasting the associated transmitted data. For example, to render a view oriented at $<\text{pitch} = 0>$ with 100-degree vertical and horizontal FOV, only fewer than 15% pixels on an equirectangular-projected frame are needed. As a result, over 85% pixels are not viewed and wasted.

Minimizing the amount of bandwidth used for unviewed pixels is especially important for streaming to devices that can render views at high resolution but have limited network bandwidth. If we can transmit higher-resolution data for the viewed portion of the video and lower resolutions for the unviewed portion, the view efficiency can potentially be improved.

To address the view efficiency problem, many groups have proposed strategies to increase pixel densities within the viewport and reduce or eliminate pixels in unviewed areas of the sphere.

While these strategies differ, their high-level ideas are similar: encode discretized areas on the sphere in different qualities. In this way, the streaming client can download portions of the sphere that overlap the user's viewport in high quality and the rest in lower quality to save wasted bandwidth.

For example, Ochi et al. developed a tile-based omnidirectional video live streaming system [21]. In this system, an equirectangular frame is divided into seven vertically overlapping tiles, and these tiles are encoded at a high bitrate. In addition, the entire omnidirectional video is also encoded at a low bitrate. During live streaming, the client requests two video streams: a low-bitrate stream that encodes the full omnidirectional view and a high-bitrate stream that encodes the area the user is currently watching. Corbillon et al. proposed to prepare predefined sets of tiles [16]. Each of these sets is characterized by the quality emphasis center (QEC): tiles around the QEC are encoded with higher quality, while tiles far away from the QEC are encoded with lower quality.

To allow the streaming server to select the best versions of a 360-degree video (e.g., tile quality) to encode, Corbillon et al. proposed an optimization problem that leverages historical user view distribution [15]. For tiles to be independently decoded, they need to be encoded independent of each other. As a result, tiles may suffer from less efficient video compression. In an attempt to address this problem, Zare et al. [28] proposed to use a motion-constrained tile set (MCTS), allowing tiles to be encoded using macroblocks in other tiles within the same MCTS. To realize the bandwidth savings of tile-based approaches, it is important that accurate predictions can be made regarding users' future view directions. Qian et al. proposed to leverage head movement prediction and download only the portion of video in the predicted viewport [24]. While this strategy can minimize wasted data transmission, it may severely affect the user experience as viewers of the video may see blank screen portions if head movement prediction is wrong. Petrangeli et al. proposed a streaming adaptation scheme, selecting quality levels of tiles based on if a tile is within the predicted future viewport or not [23].

Facebook recently proposed a **pyramid projection** for streaming 360-degree videos to virtual reality devices [8]. Here, a pyramid is constructed around the sphere, with the bottom of the pyramid facing in the viewing direction. Pixels on the spherical surface are then projected onto pyramid faces in a similar manner to the standard cubic projection. The pyramid projection is intended to devote more of the projection's surface in the direction of viewer attention than in other directions.

According to our measurements of Oculus 360-degree videos, however, this pyramidal projection is not currently in use. We also cannot find any other publicly available information of how Oculus encodes 360-degree views using the pyramidal projection. We suspect Oculus no longer uses the pyramidal projection because it devotes most pixels to the pyramid bottom in the viewing direction while severely underencoding areas of the sphere in nonviewing directions. This underencoding could cause users who turn their heads to experience suboptimal viewing quality.

Instead, through reverse engineering, we discovered that Facebook is using a new encoding scheme for encoding Oculus 360-degree videos. In this scheme, spherical pixels are first distorted before they are projected onto rectangular planes. As a result of this distortion, more pixels in the rectangular plane are devoted to pixels in a certain direction of the sphere than others. Next, we describe details of various offset projections, including how distortions are applied.

3 OFFSET PROJECTIONS

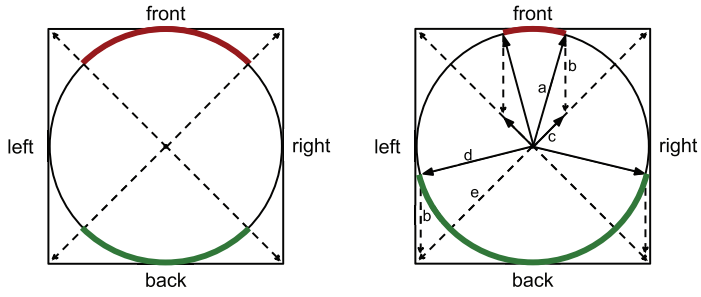
Offset projections operate by first distorting the spherical surface, then mapping the distorted sphere into 2D rectangular planes using the cubic, equirectangular, or barrel projections.

Every offset projection has an orientation. We refer to this orientation as θ_{offset}. The projection distorts spherical angles so that angles near θ_{offset} are mapped to wider angles on a distorted spherical surface.

To convert from a standard spherical angle to an angle with the offset distortion, we can take a unit vector, a, pointing to a pixel at a standard spherical angle and add a vector, b, in the direction opposite to θ_{offset}. The resulting vector, $c = a + b$, points to the pixel in a distorted spherical surface.

3.1 The Offset Cubic Projection

We start our discussion with the **offset cubic projection** as it is used by Oculus 360-degree video streaming. The offset cubic projection is similar to the standard cubic projection, mapping the pixels from a spherical surface to six cube faces. Unlike the standard cubic projection, the spherical surface is first **distorted** before being projected to six faces. This distortion produces cube faces where more pixels are provisioned for angles nearer to θ_{offset} and fewer pixels are given to angles farther from θ_{offset}.



(a) The standard cubic projection. (b) The offset cubic projection.

Fig. 4. In the offset cubic projection, vector a is a unit vector pointing to a pixel in the standard sphere. Vector b points in the opposite direction of the offset cube's orientation. Vector c is $a + b$. The intersection of vector c and the surface of the cube is the projection destination of the pixel at vector a. The red portion of the circle indicates the portion of the sphere mapped to the offset cube's front face. The green portion of the circle indicates the portion of the sphere mapped to the offset cube's back face.

Figure 4 depicts the offset cubic projection's transformation of spherical angles in two dimensions. For a unit vector a pointing to a pixel in the nondistorted sphere, an offset in the opposite direction of the offset cube's orientation, represented as vector b, is applied, resulting in a new vector c. This construction causes the offset cube's front face (the face oriented toward θ_{offset}) to encompass a smaller area on the sphere compared to the offset cube's back face (the face oriented opposite to θ_{offset}). We can use the following equation to calculate the horizontal and vertical extent encoded in the front face:

$$90^\circ - 2 \times \beta = 90^\circ - 2 \times \arcsin \frac{\|b\| \times \sin 135^\circ}{\|a\|} = 90^\circ - 2 \times \arcsin(\|b\| \times \sin 135^\circ),$$

where β is the angle between vector a and vector c, the angle between vector $-b$ and vector $-c$ is 135°, and $\|a\| = 1$.

We can also calculate the horizontal and vertical extent encoded in the back face as follows:

$$90^\circ + 2 \times \alpha = 90^\circ + 2 \times \arcsin \frac{\|b\| \times \sin 45^\circ}{\|d\|} = 90^\circ + 2 \times \arcsin(\|b\| \times \sin 45^\circ),$$

where α is the angle between vector d and vector e, and $\|d\| = 1$.

For example, in Oculus's implementation of the offset cubic projection, the magnitude of vector b is 0.7. This parameter causes the cube's front face to encompass 30.66 degrees. The cube's back face encompasses 149.33 degrees. Both measures occur in both the horizontal and vertical dimensions. Because these angular dimensions are represented by cube faces with the same number of pixels, areas on the sphere covered by the front face are encoded in much higher quality than areas covered by the back face.

Figure 3(c) shows the resulting image if we project the same image in Figure 1(a) and Figure 3(b) using the offset cubic projection with θ_{offset} set to a unit vector in the direction $<yaw = 0, pitch = 0>$ and $\|b\| = 0.7$.

3.2 The Offset Equirectangular Projection

The **offset equirectangular projection** can be created by projecting the **distorted** spherical surface onto an equirectangular image. Computing the distortion uses the same procedure as used for the offset cubic projection: we start with a pixel at a standard spherical angle represented by vector a, then add a vector, b, in the direction of $-\theta_{\text{offset}}$, resulting in a new vector $c = a + b$.

Unlike the offset cubic projection, the pixel's destination position in the offset equirectangular image is obtained by calculating the yaw and pitch angles of vector c. Suppose vector $c = (c_x, c_y, c_z)$, and we can then calculate the yaw and pitch values of c as follows:

$$c_{\text{yaw}} = \arctan \frac{c_x}{c_z}, \quad c_{\text{pitch}} = \arcsin \frac{c_y}{\|c\|}. \tag{1}$$

The corresponding pixel is thus mapped to the equirectangular image with position $x = (c_{\text{yaw}} + 180)/360 \times \text{width}$ and $y = (90 - c_{\text{pitch}})/180 \times \text{height}$. For example, Figure 1(b) shows the resulting image if we project the same image in Figure 1(a) using the offset equirectangular projection with θ_{offset} set to a unit vector in the direction $<yaw = 0, pitch = 0>$ and $\|b\| = 0.7$.

3.3 The Offset Barrel Projection

To create the **offset barrel projection**, we repeat the procedure for the offset equirectangular projection. For every pixel on the standard sphere, we use a vector a to represent its spherical angle. We then create a new vector $c = a + b$ for every vector a and extracting the yaw and pitch values for vector c using Equation (1). Pixels whose c_{pitch} angles are between -45 and 45 degrees are mapped to pixels on a rectangular image with $x = (c_{\text{yaw}} + 180)/360 \times \text{width}$ and $y = (90 - 2 \times c_{\text{pitch}})/180 \times \text{height}$. Other pixels are mapped to two circles placed next to the rectangle in the barrel layout.

Figure 2(b) shows the barrel projection of the equirectangular frame in Figure 1(a). Figure 2(a) shows the offset barrel projection of the same frame with θ_{offset} set to a unit vector in the direction $<yaw = 0, pitch = 0>$ and $\|b\| = 0.7$.

3.4 Offset Applied to the Horizontal Plane Only

To describe the horizontal offset projection, we begin with an orientation θ_{offset} at $<yaw = 0, pitch = 0>$. The horizontal offset requires an additional vector to define what we call the horizontal plane. We denote the horizontal plane normal by the vector ϕ. For an orientation at $<yaw = 0, pitch = 0>$, we select the XZ plane as the horizontal plane with the horizontal plane normal, $\phi = \vec{y}$.

Given a point on the surface of a sphere represented by vector a, we decompose a into a component a_v in the direction of the plane normal, and a component a_h perpendicular to a_v. We construct a vector c by first letting $c_h = \frac{a_h}{\|a_h\|} + b$, where b is a vector in the direction opposite to θ_{offset}, and then $c = \frac{c_h}{\|c_h\|} \times \|a_h\| + a_v$. This construction allows the offset transformation to occur only on the horizontal plane without affecting vertical components. The orientation of vector c is the orientation of a pixel corresponding to the pixel at the original vector a on the distorted sphere.

Note that ϕ can be set to any orientation perpendicular to θ_{offset}. In practice, ϕ is set by rotating $\vec{y}$ by the yaw and pitch of θ_{offset}.

The resulting sphere with horizontal offset can then be projected into equirectangular, barrel, and cubic projections. Figures 1(c), 2(c), and 3(d) show the resulting images when offset with the same magnitude; i.e., $\|b\| = 0.7$ is only applied to the horizontal plane.

4 OFFSET CUBIC PROJECTION IN OCULUS

As more pixels on the projected rectangular plane are devoted to spherical pixels near the offset direction, θ_{offset}, we can expect that high visual qualities are produced when the user's view orientation matches θ_{offset}. But if a user's view is in the opposite direction of θ_{offset}, then the visual quality of rendered views can be much worse than views rendered by standard cubes.

Table 1. Yaw and Pitch Values of Center of the Front Faces in 22 Offset Cubes

Pitch	Yaw
90	0
45	15, 105, 195, 285
0	0, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300, 330
−45	15, 105, 195, 285
−90	0

For every row in the table, an Oculus offset cube encoding exists for the single pitch value combined with each value in the Yaw column.

Table 2. Resolution and Bitrate of 4 Quality Levels of a Sample Video [10]

Quality	Frame Resolution	Bitrate (bps) Range
272w	1088 × 816	1,789,736 to 2,648,917
400w	1600 × 1200	6,290,250 to 9,613,871
528w	2112 × 1584	9,556,146 to 15,291,141
656w	2624 × 1968	13,512,541 to 22,261,091

Note that this sample video is stereoscopic. Therefore, two images are encoded in each frame for the left and right eyes, respectively. A frame encodes 12 cube faces in total.

In practice, when watching 360-degree videos, users can view in different orientations, and a user's view can change over time. To account for this variance in a user's view orientations, Oculus encodes 360-degree videos using 22 offset orientations. In Oculus 360-degree videos, offset cube orientations are specified by yaw and pitch angles of the center of the cube's front face. Table 1 lists the yaw and pitch values of these 22 orientations. Note that since each offset cubic projection's front face can cover only a roughly 30-by-30-degree field of view, the combination of 22 front faces cannot cover the full spherical surface at front-face quality levels.

To accommodate clients with heterogeneous downlink bandwidth, Oculus also encodes 360-degree videos into four quality levels with different frame resolution and range of bitrates. Table 2 shows the bitrate ranges from all four quality levels of one sample video we found on Oculus [10]. These quality levels are labeled by the resolution of cube faces. For example, the quality level of `400w` indicates that the resolution of each cube face is 400×400. If a 360-degree video is monoscopic, each frame encodes six cube faces. A monoscopic 360-degree video frame encoded into a `400w` offset cube therefore has a resolution of 1200×800. If a 360-degree video is stereoscopic, two images are encoded in each frame. These two images are used to render views for the left eye and right eye, respectively. As a result, each frame encodes 12 cube faces in total.

Overall, Oculus encodes each 360-degree video into **88** versions, storing all combinations of the 22 different offset cube orientations and four quality levels.

A disadvantage of the offset cube is that more storage is required on the server side to account for possible viewport orientations. The total storage size of all 88 versions of our 4-minute, 43-second sample video is 31GB (33,350,813,552 bytes). A further disadvantage lies in the complication of the segment selection algorithm over the standard cubic projection. To effectively stream offset-cube-projected videos, segments must be selected across two dimensions, orientation and quality, rather than only over the quality dimension.

5 VISUAL QUALITY

In this section, we investigate the visual quality of views rendered from offset projections. To do so, we compare views generated from the offset projection representations against views generated from high-quality equirectangular video frames.

These reference images consist of 12 frames in 8K resolution and include scenes with different characteristics, including an indoor scene, an outdoor city scene, an outdoor natural scene, and a scene from VR gaming. The high-quality frames are extracted from five 360-degree videos[2] from YouTube and are encoded using the equirectangular projection at 8K resolution. We extracted frames from these high-quality videos into `.bmp` format to minimize noise due to image compression.

5.1 Offset Cube versus Equirectangular

To describe our method of offset cube view quality comparison, we first define a set of notations used to describe different offset cube representations and the views rendered from these representations. As in the previous section, we use the angle θ_{offset} to describe the offset cube's orientation, where θ_{offset} is the coordinate of the center of the offset cube's front face and the direction of greatest pixel density. We then use q to represent the quality level of the offset cube. Views in any orientation on the sphere can be rendered from any offset cube. We represent the view orientation by the variable θ_{view}.

In our comparison against reference images, we compute the visual quality between the displayed frames generated in the orientation θ_{view} from the offset cube $(\theta_{\text{offset}}, q)$ against reference images generated in the θ_{view} orientations from an original, high-quality equirectangular image.

We have created a tool that can generate offset cubes at a desired $(\theta_{\text{offset}}, q)$ given an equirectangular image. The tool can also render a view at θ_{view} and a desired resolution given either an offset cube or an equirectangular representation of the 360-degree spherical surface.[3]

For each of the high-quality equirectangular frames, we generated 88 offset cubic projections in the same 22 orientations and four quality levels as used in Oculus. For each of the 88 offset cubes, we render views at a given orientation, with 96-degree horizontal and vertical fields of view, at a resolution of 1000×1000. These values roughly correspond to the viewport configuration in GearVR. We calculate the peak signal-to-noise ratio (PSNR) and global structural similarity index (SSIM)[4] [26] between views rendered from an offset cubic projection source and views rendered from the original 8K equirectangular image. Because human eyes are most sensitive to luminance, we calculate PSNR and SSIM based on the Y-component of the YUV representation of views. For each offset cube quality and orientation, we consider a total of 3,000 view orientations. These view orientations are randomly selected from a list of the yaw and pitch angles of 655,362 points on the sphere [13].

For each of these rendered views, we compute the angle between θ_{offset} and θ_{view}. The maximum angle on the sphere is 180 degrees. We group PSNR and SSIM results into 5-degree angular distance buckets. This bucketing step removes noise that occurs at smaller bucketing levels if some of these smaller buckets receive only a small number of quality samples. We then take the average of the PSNR and SSIM values for every combination of quality level and angular distance between θ_{offset} and θ_{view}.

[2] https://www.youtube.com/watch?v=DSvoUiYRrOc, https://www.youtube.com/watch?v=If5TkBmxsW0, https://www.youtube.com/watch?v=cwQGQWrOg_Y, https://www.youtube.com/watch?v=70g9qBJWSP4, https://www.youtube.com/watch?v=RNdHaeBhT9Q.

[3] https://github.com/bingsyslab/360projection.

[4] We calculate the global SSIM index in this article. Other varieties of SSIM can be used to compare visual quality. These varieties include the mean SSIM index (MSSIM) and multiscale SSIM index (MS-SSIM). We plan to investigate whether these other metrics reveal differences in visual quality that were not detected by the current measurement methods.

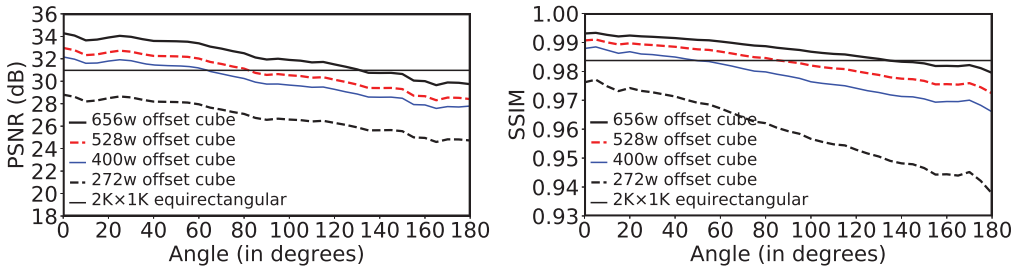


Fig. 5. PSNR and SSIM between views rendered by offset cubes of different quality and views rendered from the original 8K equirectangular frame. The x-axis in the graph shows the angle between the view orientation and the offset cube's orientation. Offset cubes in this figure are created with the offset magnitude $\|b\| = 0.7$. The dark, flat line indicates a baseline comparison between views generated from the original 8K frame and views generated from a subsampled version of the original frame subsampled to 2K×1K.

For comparison, we downsize the original 8K equirectangular image to 2K×1K resolution using OpenCV resize [11] with `INTER_AREA` interpolation method. We then calculate the PSNR and SSIM between views rendered by the 2K × 1K version and views rendered by the original 8K version. We calculate PSNRs and SSIMs of the 2K × 1K version for all 12 representative frames and all 3,000 view orientations. We then use the average of these values as a baseline for visual quality. The 2K × 1K equirectangular representation has a similar number of pixels as the highest-quality offset cube representation, so the PSNRs and SSIMs across this lower-quality equirectangular image and offset cube form a reasonable basis for understanding the capabilities of the offset cube encoding.

Figure 5 shows a plot of angles between θ_{offset} and θ_{view} against PSNR and SSIM. Offset cubes in this figure are created with the offset magnitude set to 0.7, which is the value used by Oculus for encoding 360-degree videos. As expected, both PSNR and SSIM decrease as the angle increases. That is, image quality gets worse the farther we get from the center of the offset cube. For example, at the highest-quality level, `656w`, the PSNR decreases from 34.8dB when the angle between θ_{offset} and θ_{view} is 0 degrees to 30.4dB when the angle is 180 degrees (i.e., the maximum). The SSIM value for `656w` also decreases from 0.9935 to 0.9814 as the angle increases. Views rendered by offset cubes of the higher-quality level give better visual quality as well. On average, the PSNR values of the `656w` quality level are 1.36dB better compared to the `528w` quality level and 2.71dB better compared to the `400w` quality, and 5.34dB better than the lowest, `272w` quality level.

Figure 5 also shows that when the angle is smaller than 50 degrees, the 2K × 1K equirectangular quality is below the `400w` quality, measured in terms of both PSNR and SSIM. Given that the 2K × 1K equirectangular representation is encoded using 2,000,000 pixels and the `400w` offset cube image is encoded using only 960,000 pixels (with a resolution of 1200 × 800), we can conclude that it is possible to obtain better or similar visual quality with the offset cubic projection using less than half the pixels of the equirectangular representation. This comparable quality is only possible, of course, if the streaming algorithm used in conjunction with the offset cube projection is capable of consistently delivering segments with θ_{offset} within 50 degrees of θ_{view}.

5.2 Offset with Different Magnitudes

We next investigate how offset with different magnitudes affects the visual quality of rendered views. We consider a total of nine offset magnitudes from 0.1, to 0.2, to 0.9. For each offset magnitude setting, we create 88 offset cubes with four different quality levels and 22 offset orientations. For reference, we also consider the case where no offset is applied. That is, the offset magnitude

is 0. In this case, we simply created 22 cubes whose front faces are oriented toward 22 different orientations.

We repeat our visual quality comparison procedure in the previous section, comparing the visual quality of views rendered by these offset cubes with views rendered by the 8K equirectangular image. The results are shown in Figure 6.

When θ_{view} is aligned with θ_{offset} (shown in Figure 6(a)), views rendered by offset cubes with the offset magnitude between 0.2 and 0.7 have roughly the same high visual quality compared to other magnitudes. When $\|b\| = 0.4$, offset cubes at the 656w quality yield the best visual quality with PSNR = 35.78dB and SSIM = 0.9947.

Although the difference is small, our results show that Oculus's choice of setting $\|b\| = 0.7$ is not necessarily the best even when the view perfectly matches the offset cube's orientation.

When θ_{view} deviates from θ_{offset}, the benefit of the offset projection starts to diminish. When their angle is between 40 and 45 degrees, offset cubes with the offset magnitude between 0.1 and 0.8 all generate roughly the same visual quality. For larger angles, e.g., 90 and 180 degrees, offset cubes with smaller-offset magnitudes produce better-rendered view visual qualities than cubes with large magnitude offsets. This quality reduction is expected, as large offset magnitudes cause much more spherical area to be encoded in the cube's back face.

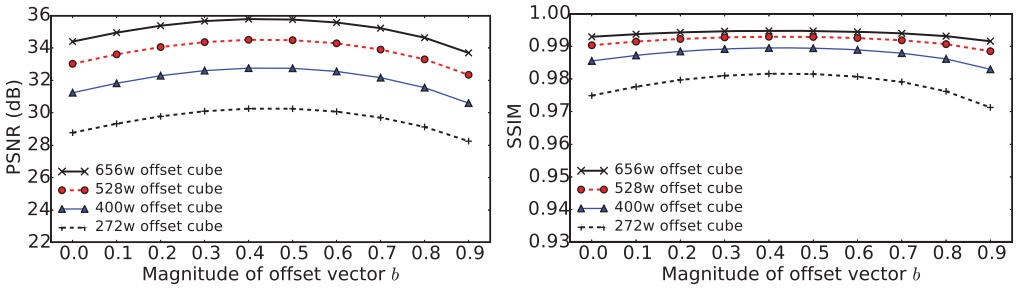
Although we have shown that the choice of $\|b\| = 0.4$ outperforms the $\|b\| = 0.7$ offset for full rendered views near the offset orientation, Oculus's use of the $\|b\| = 0.7$ offset may be motivated by other concerns. For example, past studies [18, 22] have shown that the human visual system is more sensitive to image quality toward the center of the visual field. At view orientations close to the offset orientation, the 0.7 offset likely delivers higher qualities toward the center of the view at the expense of lower qualities at the edges compared to the 0.4 offset. Care must still be taken when attempting to deliver views with higher quality at a central portion of the image. Effectively delivering these views would likely require smaller tolerances for deviation between the view orientation and the offset orientation. That is, although $\|b\| = 0.7$ might outperform $\|b\| = 0.4$ in this adjusted measure of quality when offset orientations are very close to the view orientation, the quality of the $\|b\| = 0.7$ offset would decrease more quickly as offset orientations move farther from the view orientation.

5.3 Offset versus Horizontal Offset Only versus No Offset

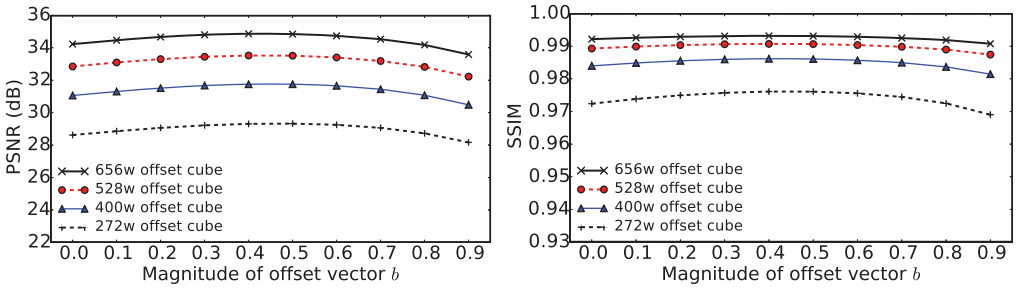
In this section, we investigate how offset strategies affect visual quality. We compare each offset scheme against its nonoffset equivalent with the same resolution. Here we expect offset strategies to have better visual quality than their nonoffset equivalents when θ_{offset} is close to θ_{view}, and as the angle between θ_{offset} and θ_{view} increases, we expect the visual quality to decrease, eventually worse than their nonoffset equivalents.

To compare the visual quality of views rendered from the offset equirectangular and nonoffset equirectangular projections, we generated oriented equirectangular projections at each of the offset equirectangular orientations. By "oriented equirectangular" projection, we mean an equirectangular projected image whose central yaw and pitch values are zeroed at the offset equirectangular projection's orientation. The oriented equirectangular projection is thus equivalent to an offset equirectangular projection with the offset = 0. At each of the 22 offset orientations listed in Table 1, we then compared the PSNR and SSIM of views generated at the same 3,000 view orientations described in Section 5.1.

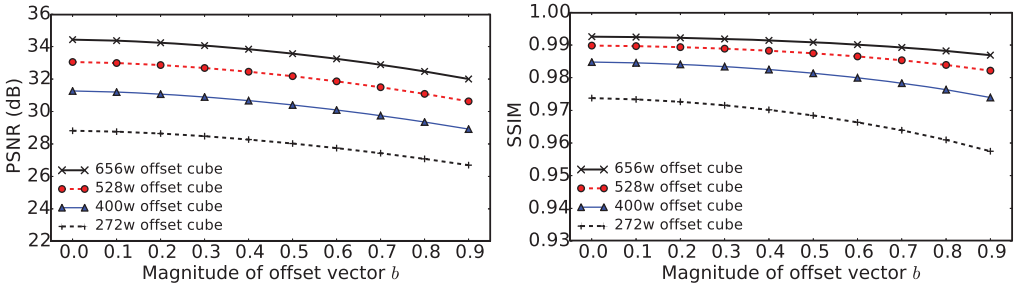
Figure 7(a) shows that at angles less than 50 degrees from its orientation, at a resolution of $2{,}000 \times 1{,}000$, the offset equirectangular projection achieves better visual quality than the standard equirectangular projection at the same resolution. However, if the offset is only applied to the



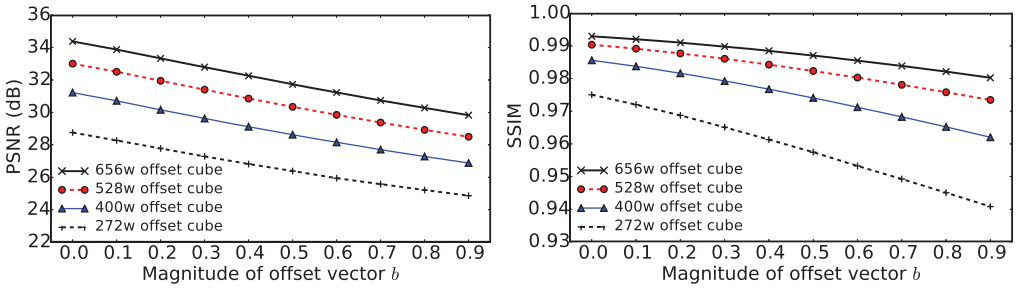
(a) $\angle(\theta_{\text{view}}, \theta_{\text{offset}})$ is between 0° and 5°



(b) $\angle(\theta_{\text{view}}, \theta_{\text{offset}})$ is between 40° and 45°

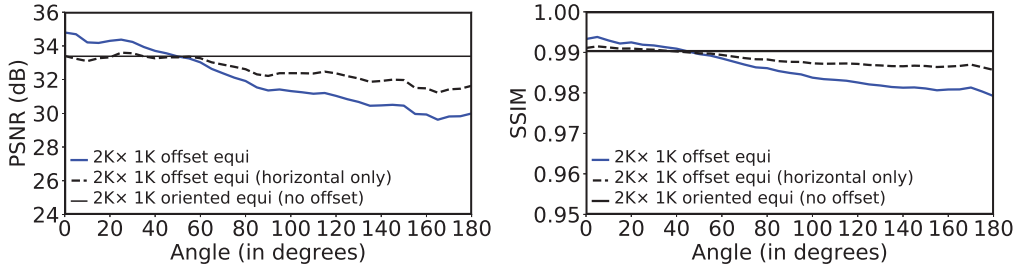


(c) $\angle(\theta_{\text{view}}, \theta_{\text{offset}})$ is between 90° and 95°

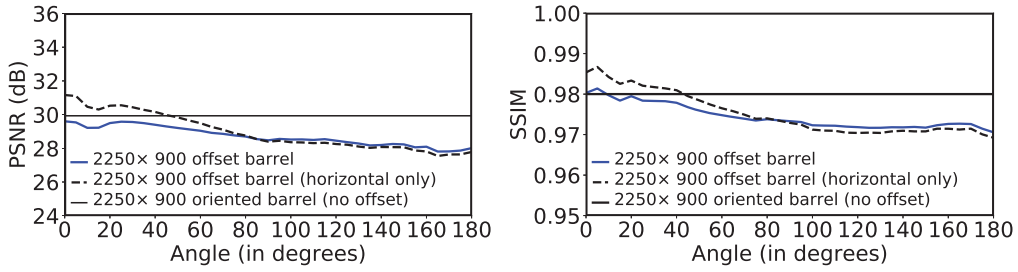


(d) $\angle(\theta_{\text{view}}, \theta_{\text{offset}})$ is between 175° and 180°

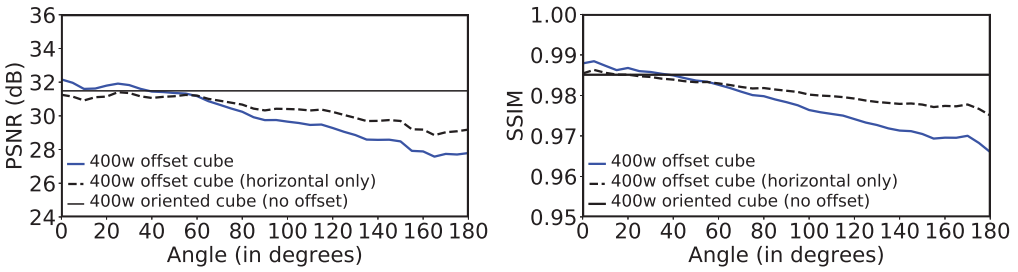
Fig. 6. PSNR and SSIM between views rendered by the **offset cubic** projection with different offset magnitudes and views rendered from the original 8K frame.



(a) Visual quality of views rendered by the **equirectangular** projection with different offset settings



(b) Visual quality of views rendered by the **barrel** projection with different offset settings



(c) Visual quality of views rendered by the **cubic** projection with different offset settings

Fig. 7. PSNR and SSIM between views rendered by projections with different offset settings and views rendered from the original 8K frame. Offset projections in this figure are created with the offset magnitude set to 0.7. That is, $\|b\| = 0.7$.

horizontal plane, the visual quality of rendered views is only as good as oriented equirectangular without any offset even when the angle is within 50 degrees.

Note that the 2K × 1K equirectangular image in Figure 5 differs from the oriented equirectangular projections used for Figure 7(a) in two ways: first, the previous equirectangular image has only a single orientation, and second, the equirectangular image was scaled by an averaging interpolation, whereas these oriented equirectangular images were regenerated at lower resolutions by projecting back to the spherical surface and retrieving the pixel from the higher-quality equirectangular image. These differences in generating lower-resolution equirectangular projections have a 2dB effect on visual quality. This difference in visual quality indicates that the choice of scaling method is important when deciding how to transform 360-degree projected images and videos.

Figure 7(b) shows the result of barrel projection under different offset settings. In this figure, "oriented barrel" is similar to "oriented equirectangular" but with the top and bottom quarter

encoded as two small circles. At a resolution of 2250×900, the barrel projection contains roughly the same number of pixels as the $2K \times 1K$ equirectangular projection. However, the visual quality of views produced by the barrel projection is much worse. For example, when no offset is applied, barrel projection is more than 3dB worse compared to the equirectangular projection. When θ_{view} is within 40 degrees of θ_{offset}, the horizontal-only offset barrel can render views with better visual quality compared to the nonoffset barrel. The final portion of our comparison of the barrel projection shows that views rendered from standard offset barrel have worse quality than the nonoffset barrel.

The poor performance of the barrel projection is likely caused by using too few pixels in the circular image portion of the projection to encode the upper and lower portions of the sphere. As the central, equirectangular portion of the barrel projection covers only a 90-degree pitch, to render a 96-degree-by-96-degree view, spherical pixels encoded in these circular images must be used, thus the lower quality compared to the standard and horizontal-only offset equirectangular projections.

The poor visual quality problem becomes even worse for offset barrel projections: due to the offset applied to spherical pixels, the central, equirectangular portion covers only a 30-degree pitch (when the magnitude of offset vector is 0.7). As a result, many of the pixels required for rendering a view are encoded in the circular portion in the barrel image.

Finally, Figure 7(c) shows the PSNR and SSIM results of views rendered by cubic projections with different offset settings. Similar to the equirectangular results, we find that when θ_{offset} is within 40 degrees of θ_{view}, the offset cube can render views with better visual quality compared to the nonoffset cube. The horizontal-only offset cube, however, performs worse than the offset cube and can even produce lower visual qualities than the nonoffset cube when θ_{offset} is within 40 degrees of θ_{view}.

5.4 Summary of Visual Quality Evaluation

We performed three types of comparisons covering offset projection quality: (1) we evaluated the visual quality of rendered views from offset projections as a function of rendered view orientation compared to offset orientation, (2) we evaluated the effect of the offset parameter, $\|b\|$, and (3) we evaluated applying offsets to different types of projections. Each of these evaluations yielded notable findings.

First, our comparisons between rendered view orientation and offset orientation reveals the efficiency of the offset cube projection. Compared to a downsampled equirectangular, the offset cube with $\|b\| = 0.7$ produces similar or better visual quality using fewer than 50% of the pixels when the rendered view orientation is within 50 degrees of the offset orientation.

When evaluating the visual quality of different settings of offset magnitudes, we discovered that a value of $\|b\| = 0.4$ produces the best visual quality of all offset values for a 96-degree rendered view when views are within 45 degrees of the offset orientation. As view orientations move farther away from the offset orientation, smaller offsets (closer to the standard cube) produce better visual qualities.

Finally, when evaluating the effectiveness of applying an offset to different types of projections, we discovered that for rendered views within 40 or 50 degrees of the offset orientation, the offset equirectangular and cubic projections perform better than their nonoffset equivalents. However, the barrel projection's quality decreases when a standard offset is applied. This decrease in quality is caused by portions of the high-quality central view being rendered in the lower-quality "barrel cap" portions of the projection. This poor performance can be corrected by applying only a horizontal offset. With the horizontal offset, no stretching occurred in the vertical direction, so the high-quality portions of the view do not migrate to the caps.

```
<ns0:MPD xmlns:ns0="urn:mpeg:dash:schema:mpd:2011" maxSegmentDuration="PT0H0M4.992S"
    mediaPresentationDuration="PT0H4M43.115S" minBufferTime="PT1.500S" profiles="urn:mpeg:dash:
    profile:isoff-on-demand:2011,http://dashif.org/guidelines/dash264" type="static">
<ns0:Period duration="PT0H4M43.115S">
<ns0:AdaptationSet FBProjection="offset_cubemap" lang="und" maxFrameRate="30" maxHeight="1968"
    maxWidth="2624" par="4:3" segmentAlignment="true" subsegmentAlignment="true"
    subsegmentStartsWithSAP="1">
<ns0:Representation FBExpand_coef="1.025" FBIs_stereoscopic="true" FBOffcenter_x="0"
    FBOffcenter_y="0" FBOffcenter_z="-0.7" FBPitch="0" FBQualityClass="uhd" FBQualityLabel="2160
    p" FBRoll="0" FBYaw="30" bandwidth="20592721" codecs="avc1.640033" frameRate="30" height="
    1968" id="dash_sve360_qf_656w_crf_18_high_5.1_p13_30yaw_0pitch_frag_1_videod" mimeType="
    video/mp4" sar="1:1" startWithSAP="1" width="2624">
<ns0:BaseURL>https://video.xx.fbcdn.net/...</ns0:BaseURL>
<ns0:SegmentBase FBFirstSegmentRange="4338-10514" indexRange="922-4337" indexRangeExact="true">
<ns0:Initialization range="0-921" />
</ns0:SegmentBase>
</ns0:Representation>
...
</ns0:AdaptationSet>
</ns0:Period>
</ns0:MPD>
```

Fig. 8. MPD document of an Oculus 360-degree video on Facebook.

6 STREAMING ADAPTATION OF OFFSET CUBIC PROJECTION IN OCULUS

Offset projections complicate adaptive 360-degree video streaming because they require a combination of bitrate adaptation (matching selected segments' bitrate with streaming client's available bandwidth) and view orientation adaptation (matching the selected offset cube's orientation, θ_{offset}, with the user's view orientation, θ_{view}).

In this section, we investigate Oculus's use of adaptive streaming with the offset cubic projection. Specifically, we attempt to understand what effect the extra orientation dimension in the offset cubic encoding has on dynamic streaming efficiency.

6.1 Adaptive Streaming in Oculus

Many video streaming services today implement HTTP adaptive streaming. With adaptive streaming, videos are divided into segments, typically a few seconds long. Each segment is then encoded multiple times at a selection of quality levels with progressively higher bitrates. These varying bitrate segments allow streaming clients to adapt to network bandwidth changes by selecting the video quality level that produces the best user experience. Many adaptive streaming protocols are available today. Among them, MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [19] is a widely used standard, adopted by popular services like YouTube. In MPEG-DASH, each quality level is referred to as a Representation. Information about each Representation such as its URL, bitrate, and resolution is described in a media presentation description (MPD) document in XML format.

Oculus extends the time-centric adaptation to allow streaming algorithms to not only select differing bitrates over time, as in standard DASH, but also select higher or lower bitrates for different areas of the 360-degree view. Figure 8 shows a snippet of an MPD document we downloaded from Facebook. This MPD document of an Oculus 360-degree video contains one Period, which further contains two Adaptation Sets, one for video, the other for audio. The audio Adaptation Set contains only one Representation. The video Adaptation Set, on the other hand, contains **88** Representations. There is one Representation for each of the 22 offset cube orientations at four separate

quality levels. In the example MPD document shown in Figure 8, Representation attributes `FBYaw` and `FBPitch` represent θ_{offset}, the orientation of the offset cube. `FBOffcenter_z` is used for deriving vector b in Figure 4. For example, we can construct a cube around a unit sphere so that the center of the cube's front face is oriented at <yaw = 0, pitch = 0> and located at (0, 0, 1) in the Cartesian coordinate system. In this case, vector b is `FBOffcenter_z`×(0, 0, 1) = (0, 0, `FBOffcenter_z`).

During 360-degree video streaming, the Oculus player has two decisions to make: (1) which of the 22 differently oriented offset cubes will perform best for the user's view, and (2) which quality level among `272w`, `400w`, `528w`, and `656w` will produce the best performance for the network conditions.

Each Representation in the video Adaptation Set in the MPD document can be downloaded as a single `.mp4` file, containing all segments. These `.mp4` files are encoded using H.264 AVC. The bitrate (i.e., bandwidth) of each Representation is calculated by dividing the `.mp4` file size by the total length (in seconds) of the video. Since each quality level has 22 different Representations encoded using different offset cube orientations, the bitrates of these Representations are also different. For example, Table 2 shows that the bitrates of our test video in the highest-quality level (i.e., `656w`) vary between 13Mbps and 22Mbps.

We have downloaded all 88 `.mp4` files in the video Adaptation Set of our testing video. After inspecting these files, we discovered that the Segment Index (`sidx`) is located at the beginning of each of these `.mp4` files. This encoding differs from the approach used by many other DASH implementations where the segment index is provided to the client as a separate file. To request a segment for a specific Representation, the streaming player can analyze the `sidx` box to determine the byte range of the segment within the `.mp4` file and request this byte range through an HTTP range request. We inspected the `sidx` box of these files and found that the Oculus segment duration is shorter than durations typically used in standard video streaming. Each segment contains 27 or 31 frames, which can play for roughly 1 second in a 30-frame-per-second video. The benefit of short segment duration is straightforward: the video player can switch to a different Representation more rapidly.

6.2 Experiment Setup

To test the performance of immersive 360-degree video streaming, we use the Samsung GearVR and the Samsung Galaxy S7 (S7 for short). We use the native Samsung ROM and Android 6.0.1. The Oculus application comes preinstalled on S7 and supports immersive video streaming. In a typical personal virtual reality setup, the S7 is placed inside the GearVR and acts as the display hardware, while the GearVR provides head-mounted goggles and additional sensor input.

VR testbed. When conducting measurements, it is nearly impossible for a human user to precisely repeat the same sequence of head movements. To address this problem, we designed a mechanical testbed that can be used to move a VR repeatedly through a sequence of motions, allowing repeatable experiments. Figure 9 shows the setup of this testbed.

In this testbed, we emulate a user's head motion during video playback using a customized, controllable Pan/Tilt mount [7]. This mount combines two separate Hitec HS-422 servo motors that can simulate yaw and pitch motions, respectively. With torque power of 57oz-in, these motors are capable of holding the test VR device. When operated with no load, at maximum speed, these motors can rotate 60 degrees in 0.16 seconds. We use a Raspberry Pi 2 Model B+ combined with an Adafruit 16-channel PWM servo board to precisely control these two servo motors.

We attach the VR device to the Pan/Tilt mount. To secure the mount onto a stable base, we 3D-printed a holder for our mount and installed it on a standard tripod. This testbed is compatible with many portable VR devices. We can repeat the current set of experiments or conduct future, multidevice experiments with controllable, emulated head motion.

Fig. 9. Our VR testbed consists of a GearVR, an S7, a Pan/Tilt mount, a 3D-printed holder for the mount, a Raspberry Pi to accurately control the mount's motion, and a tripod for stabilizing the test instruments.

Network measurement. The S7 connects to the Internet through a wireless 802.11n router using the OpenWRT open-source system. The Oculus application transmits all traffic through HTTPS. The HTTPS protocol complicates our test setup, making it impossible to inspect requests and responses through simple packet capture. To decrypt and inspect the HTTPS requests and responses, we set up a man-in-the-middle (MITM) proxy called Charles proxy [2] on another laptop machine that is connected to the same wireless router. This proxy allows us to log all HTTPS requests sent by and responses received at the Oculus application on S7.

During our measurement, we noticed that even after playback ends, the Oculus application may still keep some video content in its cache. This caching could affect results because the Oculus application could display these segments without our test setup detecting that they were downloaded. To overcome this problem, we powered off the S7 and restarted it between consecutive experiments. We confirmed through repeated testing that this strategy allows new playback start with an empty cache.

Testing video. We focus on one testing video: "Kids" [10]. This video was shot with Nokia OZO [9]. In total, this video is 4 minutes 43.115 seconds long and has 282 segments.

6.3 Oculus Streaming

Metadata downloading. To initialize the video decoder, the Oculus player first downloads the video metadata, including the initialization segment and the sidx segment. The initialization segment is placed in the beginning of the `.mp4` file and is required by DASH to initialize the video decoder. The `sidx` segment is placed after the initialization segment but before the first video segment. The byte range of these two segments in the `.mp4` files is available in the MPD document. For our test video, the metadata for 88 Representations is either 4,331 bytes or 4,337 bytes.

Instead of downloading the metadata of each Representation on demand (i.e., when the player decides to switch to the desired Representation), the Oculus player downloads the metadata for all 88 Representations before the video playback and before downloading any video segment. Based on our experiments, on average, it takes 1.345 seconds for the Oculus player to download the metadata of all 88 Representations via 88 HTTPS requests through multiple TCP connections. Because metadata download is a required initialization step for the streaming client, the video startup delay must be at least 1.345 seconds. Much of this startup delay is spent on TCP connection setup. To reduce video startup delay, Oculus could bundle all 88 metadata requests together into a single HTTPS request.

Video segment downloading. Oculus reuses a single TCP connection to download all segments during video streaming. This connection is one of the TCP connections that was set up during

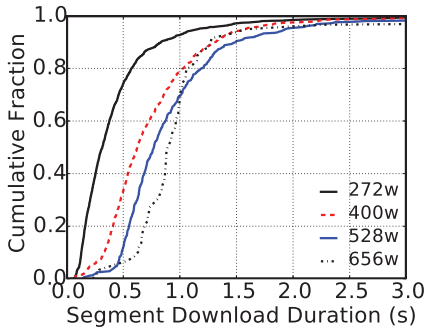


Fig. 10. Segment downloading duration distribution.

the metadata downloading step. HTTPS requests for segments with different quality levels, offset cube orientations, and segment indices are sent through this single TCP connection.

Since each segment contains roughly 1 second of streaming content, segment downloading duration must be smaller than 1 second for smooth playback. Figure 10 shows the distribution of segment downloading duration from eight experiments. In these eight experiments, no bandwidth throttling is enabled. These measurements indicated that segments in the lowest-quality level, `272w`, took the least time to download: 92.7% of `272w` segments downloaded in our experiments were downloaded within 1 second. This ratio was reduced to 78.9% for `400w` segments, 69.9% for `528w` segments, and 67.8% for `656w` segments.

Playback buffer filling. We connected the S7 to a wireless router and powered off the router after playing the video for 1 minute. After the network connection was cut, we found that the video playback could continue for another 5 seconds. We therefore estimate that the playback buffer contains roughly 5 seconds of video content.

6.4 Streaming Adaptation and Wasted Segments

An offset cube can produce the best visual quality when its orientation is exactly the same as the user's head orientation. When the user's head moves, predownloaded video segments in the playback buffer may no longer produce the best visual quality. In this case, the Oculus player may request new segments of new Representations at orientations that better match the user's current head orientation to replace existing segments of nonideal Representations in the playback buffer.

These replaced segments are never shown to the user. We therefore consider the bandwidth used to download these segments as having been wasted. In this section, we report our analysis on how segment redownloading occurs and the results of experiments measuring wasted segments under three settings: (1) fixed quality level with motion emulated by our testbed, (2) fixed orientation where quality levels are adapted by the Oculus player in response to varying network conditions, and (3) real user experiments.

6.4.1 Segment Redownloading. Ideally, to render the best visual quality, the Oculus player should redownload the next segment in a new Representation encoded with a differently oriented offset cube immediately after an orientation change is detected. However, considering that the playback deadline of the immediate subsequent segment is less than 1 second away, the pending segment may not arrive in time. Instead, the Oculus player (re)downloads segments further ahead in the future in the selected new Representation. For example, suppose the Oculus player is playing segment 10, containing video data from 10 to 11 seconds, in the best Representation, X, when head movement occurs. The playback buffer contains five predownloaded segments 11 to

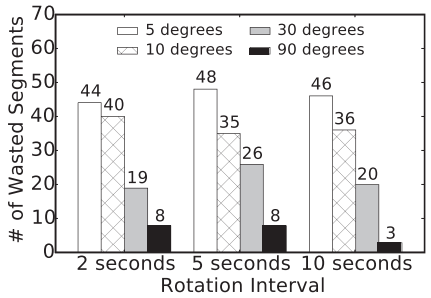


Fig. 11. Number of wasted segments under different emulated test settings. For all emulated tests reported in this figure, the motor rotates a total of 180 degrees in yaw motion. "5 degrees" in the legend means the motor rotated 5 degrees a time, 36 times in total. The total number of segments in our test video is 282.

15, in Representation X. Instead of immediately downloading a new Representation Y for the next segment 11, the Oculus player starts to download segment 13 in Representation Y. According to our extensive analysis, depending on the network bandwidth and playback buffer state, it takes 2 to 4 seconds for the Oculus player to start playback segments in a new Representation after head movements.

6.4.2 Emulated Tests: View Orientation Adaptation Only, No Quality Level Adaptation. In these tests, we evaluate the quantity of wasted segments due to view orientation adaptation. To do so, we make sure the Oculus player will always select only segments at the `272w` quality level by throttling the downlink bandwidth of the S7 to 4.5Mbps. This 4.5Mbps cap is greater than the maximum bitrate of the lowest-quality level, `272w`, but much smaller than the minimum bitrate of the second-lowest-quality level, `400w`. This bandwidth restriction thus forces the Oculus player to select the lowest-quality level and eliminates the degree of freedom for quality-level adaptation, isolating Oculus's view orientation adaptation.

We use our VR testbed for these tests so that we can accurately control the motion and conduct multiple trials of the same movement pattern. We simplify the test further by eliminating a degree of freedom in the pitch dimension. To do so, we fix one servo motor in our testbed so that the pitch value of GearVR is always zero. We then use the other motor to rotate in the yaw dimension.

We set up the motor to rotate periodically, every 2, 5, or 10 seconds. During each period, we rotate the motor 5, 10, 30, or 90 degrees. When loaded with GearVR and S7, it takes the motor about 50ms to rotate 5 degrees and 800ms to rotate 90 degrees. In all tests, the motor starts from the left-most yaw orientation (from the motor's perspective) <yaw = 0, pitch = 0> and rotates to the rightmost position <yaw = 180, pitch = 0>, a total of 180 degrees. For example, if the motor rotates 30 degrees every 5 seconds, it would take six rotations and a total of 30 seconds to finish 180-degree rotation. If the motor rotates 5 degrees every 5 seconds, it would take 36 rotations to finish the rotation in 180 seconds. When the motor rotates 5 degrees every 10 seconds, it would take 360 seconds to finish the rotation. This 360-second duration is longer than our 283-second test video. To adjust for this discrepancy, we use the number of wasted segments in the 283-second-long test to estimate how many segments would have been wasted if the motor rotated for 360 seconds. Once rotation finishes, Oculus continues the rest of video playback with fixed orientation at <yaw = 180, pitch = 0>. For each test setup, we conduct three trials and report the average.

The results are shown in Figure 11. The number of wasted segments is positively correlated with the number of rotations, while rotation interval does not have a significant impact on the number of wasted segments. When the motor rotates in 5-degree increments for a total of 36 times,

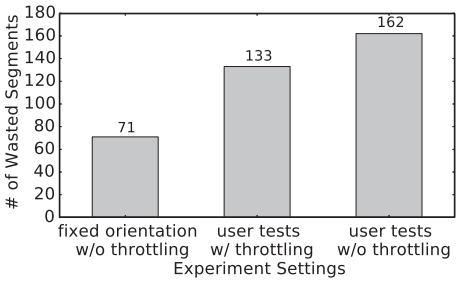


Fig. 12. Number of wasted segments under fixed orientation tests and real user tests. The total number of segments in our test video is 282.

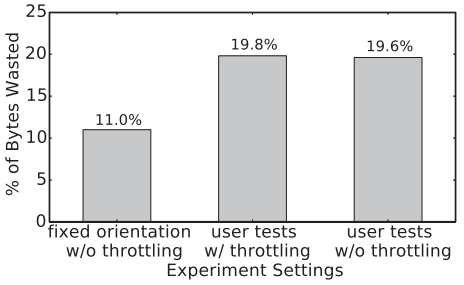


Fig. 13. Percentage of downloaded bytes wasted under fixed orientation tests and real user tests.

46 (extrapolated from 16.3% of the total 282 segments) downloaded segments are wasted on average. The number of wasted segments is reduced to 37 (13.1% of the total 282 segments) when the motor rotates 18 times in 10-degree increments, 22 (7.8% of the total 282 segments) wasted segments when the motor rotates 30 degrees at a time for a total of six times, and six (2.1% of the total 282 segments) wasted segments when the motor rotates 90 degrees at a time for a total of two times.

6.4.3 Fixed Orientation Tests: Quality-Level Adaptation Only, No View Orientation Adaptation. In fixed orientation tests, we fix the GearVR's orientation to <yaw = 0, pitch = 0> so that it will download offset cubes in only one single orientation. We also disable bandwidth throttling so that adaptation occurs over quality levels only. Before each test, we use "speedtest.net" [12] to estimate the downlink bandwidth of the S7 and to ensure that the network bandwidth is greater than 20Mbps. We conducted three fixed orientation tests. The results are shown in Figure 12 and Figure 13. In these tests, Oculus downloaded a total of 353 segments on average. The tests show that 71 (25.2% of the total 282 segments) downloaded segments were wasted. On average, 11% of bytes downloaded during these tests are wasted.

Figure 14 shows the percentage of segments in each quality level that were downloaded. The majority of segments were downloaded in low-quality levels: 42.7% in `272w` and 36.9% in `400w`.

6.4.4 Real User Tests: Both View Orientation Adaptation and Quality-Level Adaptation. We also asked five volunteers to watch our test video in GearVR. When downlink bandwidth throttling is enabled, Oculus downloaded 133 (47.2% of the total 282 segments) wasted segments on average (shown in Figure 12). Note that our test video contains 282 segments. This means 32% of all downloaded segments during video playback are wasted due to view orientation adaptation. Figure 13 shows that these wasted segments constitute close to 20% downloaded bytes.

We then disabled downlink bandwidth throttling to study the impact of both view orientation and quality-level adaptation. As in previous tests, we use speedtest.net to ensure that the S7's downlink bandwidth is greater than 20Mbps before each test starts. The results are shown in Figure 12 and Figure 13. On average, 162 (57.4% of the total 282 segments) downloaded segments were wasted, more than 36% of all downloaded segments during video playback. As a result, on average, 20% of bytes downloaded during these tests are wasted.

In these real user tests without bandwidth throttling, more segments in lower-quality levels are downloaded compared to the fixed orientation test configurations. Figure 15 shows the results. While only 42.7% segments were downloaded in the lowest-quality level of `272w` in fixed orientation tests, 60.5% of the segments were downloaded in the lowest-quality level during real user tests. We suspect that the pattern of users' head movements caused more downloaded segments

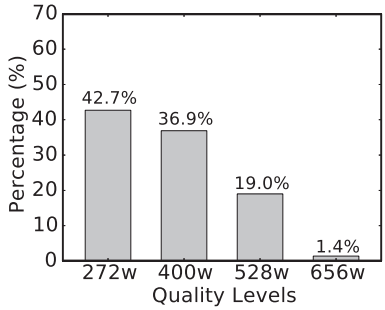


Fig. 14. Percentage of segments downloaded at each quality level in fixed orientation tests.

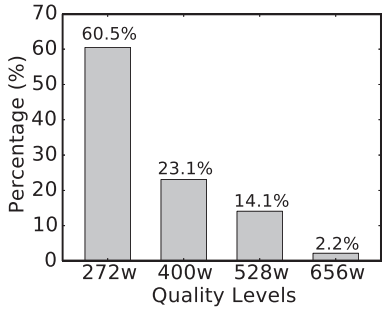


Fig. 15. Percentage of segments downloaded at each quality level in real user tests.

to be wasted. When switching to a new orientation, Oculus prefers to download segments in low quality to avoid missing the timely playback deadline.

6.5 Discussion

Streaming adaptation for offset projections can be generalized as a multidimensional adaptation problem. In the past, researchers have proposed solutions for the multidimensional adaptation problem in different contexts, such as multicamera 3D tele-immersive systems [20, 27] and image-based rendering [17].

When designing an effective adaptation algorithm for a 360-degree video streaming system that uses offset projections, many variables must be taken into consideration when selecting the best offset segment from an offset projection to download.

First, we must consider how well it is possible to predict a user's view orientation. Under perfect prediction, it is clear that offset projections will outperform standard projections. In addition, since perfect predictions cannot improve as the view deadline approaches, it never makes sense to redownload different versions of segments in the buffer. Under imperfect prediction, redownloading strategies may make sense, but we must be able to improve predictions, and possibly understand how much these predictions improve, as the view deadline approaches.

Next, we must consider bandwidth variation. For example, with constant bandwidth rates, we would be able to maintain a buffer of a single segment and, assuming better predictions closer to the view deadline, we could download the offset segment whose orientation best matches the predicted user view immediately before it was displayed to the user. With high bandwidth variability, different concerns come into play. Here, redownloading segments may make sense if segment orientations near the front of the buffer are likely to be bad. In this case, redownloading a previously downloaded segment may reduce the effective available bandwidth used to download future segments.

Finally, different offset projections experience different changes in visual quality as the difference between segment orientation and view orientation increases. Offset projections where this quality drop is large will require correspondingly accurate user view predictions. For known view prediction accuracies, it may be possible to select a corresponding offset size to optimize view quality.

7 CONCLUSION

In this article, we investigated the performance of a collection of offset spherical projections for encoding 360-degree videos. A key finding from our analysis of offset projections lies in

understanding how these methods devote more information to a direction on the 360-degree view. This directional concentration is accomplished through a simple yet novel method of distorting spherical angles. This direction-specific distortion allows the offset projection to transmit information about a user's view more efficiently than nonoffset projections when a streaming device understands which portion of the image is being viewed.

We quantified the effectiveness of these offset projections compared to their nonoffset equivalents by generating a large sample of rendered views from the offset projections in different orientations. As expected, we found that the farther away the view is from the offset projection's orientation, the lower the quality of the view. In general, offset projections can produce better visual quality than their nonoffset equivalents as long as the user's view orientation is within 40 or 50 degrees of the offset orientation. The only exception is the offset barrel projection, where the offset barrel projection consistently performs worse than the standard barrel projection. Instead, better visual quality can be achieved by applying the offset on the horizontal plane only.

Streaming 360-degree videos encoded in offset projections requires a combination of bitrate and view orientation adaptations. We evaluated these adaptations in the context of Oculus streaming client for the offset cubic projection. To do so, we both emulated user head movement using a mechanical test harness and conducted tests where human users controlled the device. We found that the Oculus player (re)downloads a segment with a new offset cube orientation whenever an abrupt orientation change occurs. As a result, over 57% extra segments are downloaded compared to an ideal downloading strategy, wasting about 20% of the total downloading bandwidth.

As 360-degree video streaming increases in popularity and begins to consume a greater share of Internet bandwidth, work to develop better encodings will become increasingly important. Our analysis of offset projection has demonstrated that significant improvements over the status quo are possible, but also that there is still room for improvement, especially in achieving better performance in segment selection during 360-degree video adaptive streaming.

REFERENCES

[1] Akamai. 2016. Akamai's state of the Internet q1 2016 report. Retrieved from https://www.akamai.com/uk/en/multimedia/documents/state-of-the-internet/akamai-state-of-the-internet-report-q1-2016.pdf.

[2] Charles Proxy. Retrieved from https://www.charlesproxy.com/.

[3] PanoTools wiki. Cubic Projection. Retrieved from http://wiki.panotools.org/Cubic_Projection.

[4] Evgeny Kuzyakov, Shannon Chen, and Renbin Peng. 2017. Enhancing high-resolution 360 streaming with view prediction. Retrieved from https://code.facebook.com/posts/118926451990297/enhancing-high-resolution-360-streaming-with-view-prediction/.

[5] Eric W. Weisstein. Equirectangular projection. Retrieved from http://mathworld.wolfram.com/EquirectangularProjection.html.

[6] Netflix. Internet connection speed recommendations. Retrieved from https://help.netflix.com/en/node/306.

[7] RobotShop. Lynxmotion pan and tilt kit/aluminium. Retrieved from http://www.robotshop.com/en/lynxmotion-pan-and-tilt-kit-aluminium2.html.

[8] Evgeny Kuzyakov and David Pio. 2016. Next-generation video encoding techniques for 360 video and VR. Retrieved from https://code.facebook.com/posts/1126354007399553/next-generation-video-encoding-techniques-for-360-video-and-vr/.

[9] Nokia OZO. Retrieved from http://ozo.nokia.com.

[10] OneRepublic. 2016. OneRepublic - Kids (360 version). Retrieved from https://www.facebook.com/OneRepublic/videos/10154946797263912/.

[11] OpenCV. OpenCV resize. Retrieved from http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html.

[12] SPEEDTEST. Retrieved from http://www.speedtest.net/.

[13] Matt C. Yu. 2015. sphere 655362. Retrieved from https://github.com/mattcyu1/omnieval/blob/master/compsph/sphere_655362.txt.

[14] Evgeny Kuzyakov and David Pio. 2015. Under the hood: Building 360 video. Retrieved from https://code.facebook.com/posts/1638767863078802/under-the-hood-building-360-video/.

[15] Xavier Corbillon, Alisa Devlic, Gwendal Simon, and Jacob Chakareski. 2017. Optimal set of 360-degree videos for viewport-adaptive streaming. In *Proceedings of ACM Multimedia (MM'17)*.

[16] Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. 2017. Viewport-adaptive navigable 360-degree video delivery. In *2017 IEEE International Conference on Communications (ICC'17)*. IEEE, 1–7.

[17] David Gotz and Ketan Mayer-Patel. 2004. A general framework for multidimensional adaptation. In *Proceedings of the 12th Annual ACM International Conference on Multimedia.*. ACM, 612–619.

[18] Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. 2012. Foveated 3d graphics. *ACM Transactions on Graphics (TOG)*, 31, 6 (2012), 164.

[19] ISO/IEC. 2014. ISO/IEC 23009-1:2014 Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats. Standard, International Organization for Standardization.

[20] Srinivas Krishnan and Ketan Mayer-Patel. 2007. A utility-driven framework for loss and encoding aware video adaptation. In *Proceedings of the 15th ACM International Conference on Multimedia*. ACM, 1026–1035.

[21] Daisuke Ochi, Yutaka Kunita, Akio Kameda, Akira Kojima, and Shinnosuke Iwaki. 2015. Live streaming system for omnidirectional video. In *2015 IEEE Virtual Reality (VR'15)*. IEEE, 349–350.

[22] Anjul Patney, Joohwan Kim, Marco Salvi, Anton Kaplanyan, Chris Wyman, Nir Benty, Aaron Lefohn, and David Luebke. 2016. Perceptually-based foveated virtual reality. In *ACM SIGGRAPH 2016 Emerging Technologies*. ACM, 17.

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

[24] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. 2007. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*. ACM, 1–6.

[25] David Salomon. 2007. *Transformations and Projections in Computer Graphics*. Springer Science & Business Media.

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

[27] Zhenyu Yang, Bin Yu, Klara Nahrstedt, and Ruzena Bajscy. 2006. A multi-stream adaptation framework for bandwidth management in 3d tele-immersion. In *Proceedings of the 2006 International Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 14.

[28] 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.

[29] Chao Zhou, Zhenhua Li, and Yao Liu. 2017. A measurement study of Oculus 360 degree video streaming. In *Proceedings of the 8th International Conference on Multimedia Systems*. ACM.

Received October 2017; revised March 2018; accepted April 2018