A Measurement Study of Oculus 360 Degree Video Streaming

Chao Zhou
SUNY Binghamton
czhou5@binghamton.edu

Zhenhua Li
Tsinghua University
lizhenhua1983@tsinghua.edu.cn

Yao Liu
SUNY Binghamton
yaoliu@binghamton.edu

ABSTRACT

360 degree video is a new generation of video streaming technology that 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 streaming technology, large scale adoption is hindered by a number of factors. 360 degree video streams have larger bandwidth requirements, require faster responsiveness to user inputs, and users may be more sensitive to lower quality streams.

In this paper, we review standard approaches toward 360 degree video encoding and compare these to a new, as yet unpublished, approach by Oculus which we refer to as the offset cubic projection. Compared to the standard cubic encoding, the offset cube encodes a distorted version of the spherical surface, devoting more information (i.e., pixels) to the view in a chosen direction. We estimate that the offset cube representation can produce better or similar visual quality while using less than 50% pixels under reasonable assumptions about user behavior, resulting in 5.6% to 16.4% average savings in video bitrate. During 360 degree video streaming, Oculus uses a combination of quality level adaptation and view orientation adaptation. 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**;

KEYWORDS

360 degree video streaming, offset cubic projection, visual quality, adaptive streaming

ACM Reference format:
Chao Zhou, Zhenhua Li, and Yao Liu. 2017. A Measurement Study of Oculus 360 Degree Video Streaming. In *Proceedings of MMSys'17, Taipei, Taiwan, June 20-23, 2017,* 11 pages.
DOI: http://dx.doi.org/10.1145/3083187.3083190

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. 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 fixed viewpoint. These 360 degree videos are viewable in browsers, mobile devices, and, most recently, within virtual reality (VR) devices such as Google's Cardboard and 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 four thousand horizontal pixels and two thousand vertical pixels. To stream 4K videos, Netflix recommends the Internet speed to be at least 25 Mbps [8]. On the other hand, according to Akamai's most recent report, the average broadband connection speed in the USA is only 15.3 Mbps [1]. When the high bandwidth requirement cannot be met, users may experience non-smooth playback, degraded visual quality, and reduced responsiveness of the viewport controls. These streaming degradations can have larger effects on the user experience in 360 degree video streaming than in the standard streaming setting. Specifically, in the VR setting, mismatches between head movements and display changes can cause motion sickness [20], making videos effectively unviewable.

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. In these encoding schemes, care must also be taken so that unexpected control inputs do not lead to poor quality or unrendered video. For instance, if a user moves his or her head in an unexpected direction, the device should not render a black screen because no segment for the viewed area was present in the playback buffer.

Given the recent popularity of Facebook's Oculus[1], in this paper, we focus on understanding its encoding scheme as well as its adaptive streaming mechanism. Specifically, we have reverse-engineered Oculus' 360 degree video encoding scheme. We refer to this scheme as the "offset cubic projection". The offset cubic projection distorts the encoded frames so that more pixels in the encoding are mapped to views in the front-facing direction of the offset cube than in other directions. For each frame, Oculus encodes separate 360 degree views centered at 22 orientations 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 network bandwidths. We believe that we are the first group to publish a description of this method.

We perform measurement studies of the offset cubic projection. We compared views rendered from Oculus' offset cubic orientations and quality levels against views rendered from a high-quality reference image to generate visual quality measures (PSNR and SSIM). We then compared these visual quality measures against quality measures produced by comparing lower resolution reference images against the high-quality reference image. Results show that when rendering views near the offset cube's front orientation, using an offset cube encoding can save half the pixels of the widely-used equirectangular representation while maintaining better or similar visual quality. This leads to 5.6% to 16.4% average savings in video bitrate.

Finally, we attempt to gauge the performance of the Oculus streaming algorithm. Oculus extends adaptive streaming by adding a new dimension of adaptation: view orientation adaptation. 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' 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.

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

2 BACKGROUND AND RELATED WORK

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

[1]Oculus was acquired by Facebook in 2014 [5].

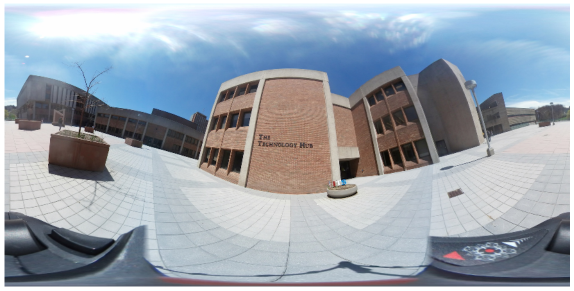


Figure 1: A 360 degree image encoded using the equirectangular projection.

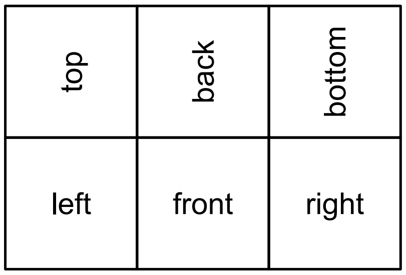
These mappings from the sphere to rectangular images have developed from simpler mappings that are easy to encode and render to more complicated mappings that more efficiently represent the sphere at a given viewing angle.

The earliest attempt at encoding the sphere was the **equirectangular projection** [4, 19]. 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[2] (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.

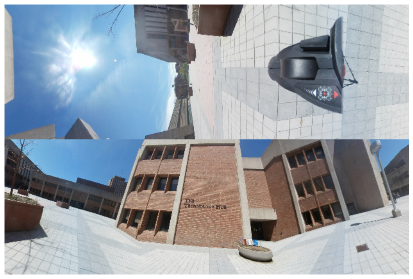
Many 360 degree video streaming services today encode videos using the equirectangular projection including YouTube and Jaunt VR. Figure 1 shows a frame extracted from a YouTube 360 degree video that uses the equirectangular projection. A significant disadvantage of the equirectangular projection is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

Another type of projection is the **standard cubic projection** [3, 19]. 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 $< \text{yaw} = 0, \text{pitch} = 0 >$ 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 [10, 14]. Based on our measurement findings, Facebook uses the standard cubic projection to encode 360 videos for streaming on web browsers and mobile devices. Figure 2(b) shows the same video frame as Figure 1 but encoded using the standard cubic projection

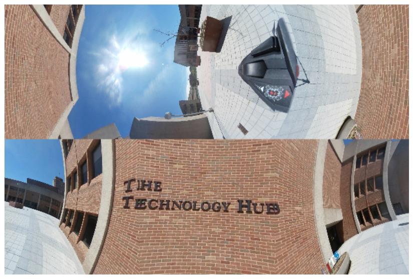
[2]The equirectangular projection is typically parameterized in terms of longitude and latitude. For consistency, we use yaw and pitch here. These are equivalent to longitude and latitude as long as the rotation in the yaw direction is performed before the rotation in the pitch direction and we consider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



(a) Arrangement of the six cube faces.



(b) The standard cubic projection.



(c) The offset cubic projection.

Figure 2: A 360 degree image encoded in standard cubic and offset cubic projections.

for serving 360 videos in web browsers. Figure 2(a) shows Facebook's arrangement of the six faces of the cube onto a rectangular surface for video compression.

A potential inefficiency of both the equirectangular projection and the standard cubic projection is that the field of view (FOV) rendered and presented to end users, typically smaller than 150 degrees, is much smaller than the encoded 360 degree image. Streaming the entire 360 degree view typically leads large numbers of unrendered pixels, wasting the associated transmitted data. 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, transmission efficiency can potentially be improved.

To address the transmission 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. For example, Ochi et al. developed a tile-based omnidirectional video live streaming system [17]. In this system, an equirectangular frame is divided into seven vertically overlapping tiles, and these tiles are encoded at high bitrate. In addition, the entire omnidirectional video is also encoded at 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 pre-defined sets of tiles [15]. 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. While both the Ochi et al. and Corbillon et al. approaches allow the video streaming client to download low quality representations of non-viewport portions of the video, thereby saving bandwidth, they suffer from less efficient video compression since tiles are encoded independent of each other. Independently encoded tiles may also produce artifacts at tile boundaries. If these artifacts are observed by viewers, they could significantly detract from the viewing experience. Qian et al. proposed to leverage head movement prediction and download only the portion of video in the predicted viewport [18]. 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.

Facebook recently proposed a **pyramid projection** for streaming 360 degree videos to virtual reality devices such as GearVR [10]. 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.

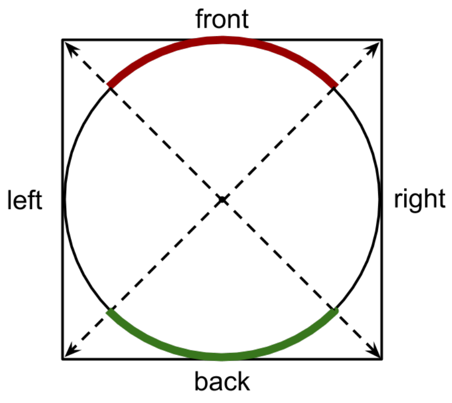
According to our measurements of Oculus 360 degree videos, however, this pyramidal projection is not currently in use. We also cannot find any other publicly-available information of how Oculus encodes 360 degree views using the pyramidal projection. We suspect Oculus no longer uses the pyramidal projection because it devotes most pixels to the pyramid bottom in the viewing direction while severely under-encoding areas of the sphere in non-viewing directions. This under-encoding 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. We refer to this scheme as the **offset cubic projection**. This offset cubic projection uses a novel strategy to distort encoded images so that more pixels in the encoding are mapped to views in the front-facing direction of the offset cube than in other directions. Figure 2(c) shows the offset cubic projection of the same video frame as the equirectangular projection in Figure 1 and the standard cubic projection in Figure 2(b). Next, we describe details of the offset cubic projection, including how distortion is applied when generating offset cubes.

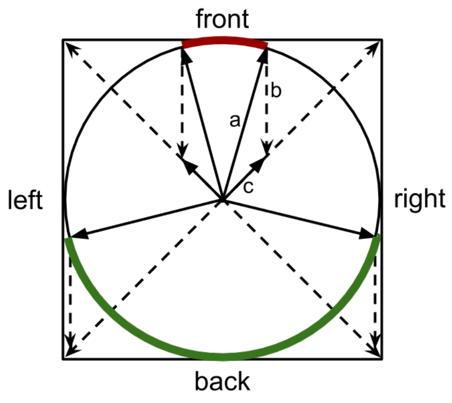
3 THE OFFSET CUBIC PROJECTION

In this section, we describe the offset cubic projection. Our knowledge of the offset cubic projection was gained by reverse engineering frames downloaded from Oculus video streams. To the best of our knowledge, we are the first to publish a description of this new projection scheme.

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 offset cubic 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 the cube face. This distortion produces



(a) The standard cubic projection.



(b) The offset cubic projection.

Figure 3: 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.

Table 1: Yaw and pitch values of center of the front faces in 22 offset cubes. 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.

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

cube faces where more pixels are provisioned for angles nearer to θ_{offset} and fewer pixels are given to angles farther from θ_{offset}.

To convert from a standard spherical angle to an angle with the offset cubic 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, $a + b$, points to the pixel in the offset cubic projection. Figure 3 depicts the offset cube's transformation of spherical angles in two dimensions. In Oculus' implementation of the offset cubic projection, the magnitude of vector b is 0.7. This parameter causes the cube face oriented towards θ_{offset} to encompass roughly 30 degrees. We refer to this face as the "front face" of the cube. The "back face" of the cube encompasses roughly 150 degrees. These 30 and 150 degrees 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.

3.1 Offset Cube in Oculus

Higher visual qualities are produced when the user's view orientation matches the orientation of the offset cubic projection, θ_{offset}.

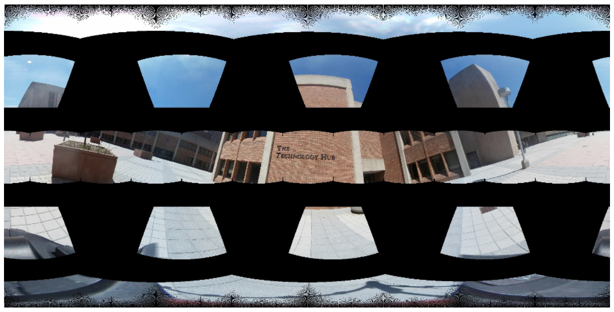


Figure 4: Drawing the front faces of all 22 offset cubes on to an equirectangular image.

To account for different user's view orientations, Oculus encodes 360 degree videos using 22 offset cube orientations. In Oculus 360 degree videos, offset cube orientations are specified by yaw and pitch angles of the center of the front face. Table 1 lists the yaw and pitch values of these 22 orientations.

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. Figure 4 shows the resulting equirectangular image when we apply this transformation to the front faces of all the 22 Oculus offset cubic projections.

To accommodate clients with heterogeneous downlink bandwidth, Oculus also uses 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 [12]. These quality levels are labeled by the resolution of cube faces. For example, the quality level of `400w` indicates the resolution of each cube face is 400×400. If a 360 video is monoscopic, each

Table 2: Resolution and bitrate of 4 quality levels of a sample video [12]. 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.

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

frame encodes six cube faces. A monoscopic 360 video frame encoded into a `400w` offset cube therefore has a resolution of 1200×800. If a 360 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 minutes 43 seconds sample video is 31 GB (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.

4 VISUAL QUALITY OF VIEWS RENDERED BY OFFSET CUBE

In this section, we investigate the visual quality of views rendered from the offset cubic projection. To do so, we compare views generated from the offset cube representations at standard resolutions used by Oculus 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[3] 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.

For comparison purposes, it is important that these high quality equirectangular frames can generate views of higher quality than any views generated from the highest quality Oculus offset cubic projections, otherwise visual quality metrics for front facing offset cube views could produce inaccurate values, as the reference image would be lower quality than views produced from offset cubic projections. To ensure that our reference images had a quality high enough to facilitate this comparison, we mapped the front faces of the offset cube to an equirectangular image (Figure 4). We can see, roughly, that an 8K resolution equirectangular image is large enough to capture the front face of the offset cube by combining the following pieces of information: (1) Each offset cube front face encompasses 30 degree vertical and horizontal field of views. (2) Offset cubes with the highest quality level, `656w`, have faces with 656×656 resolution. (3) To generate the front face of this offset cube from an equirectangular frame, the equirectangular frame would therefore need a horizontal resolution of at least $656\times360/30 = 7872$, indicating that 8K resolution reference frames are sufficient for this comparison.

To describe our method of offset cube view quality comparison, we first define a set of notation 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.

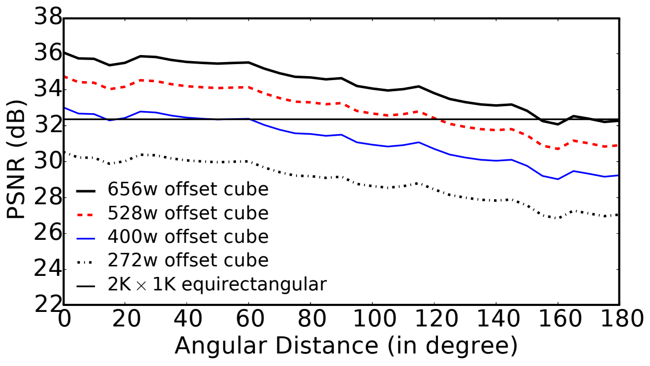
We have created a tool that can generate offset cubes at a desired $(\theta_{\text{offset}}, q)$ given an equirectangular image. The tool can also render a view at θ_{view} and a desired resolution given either an offset cube or an equirectangular representation of the 360 degree spherical surface.[4]

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 a 96 degree horizontal and vertical fields of view, at a resolution of 2000×2000. These values roughly corresponds to the viewport configuration in GearVR. We calculate the peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) [21] 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 2664 view orientations, with the pitch value in the range $[-90, 90]$ in five degree increments and the yaw value in the range $[0, 360)$ also in five degree increments. (Recall that pitch and yaw are equivalent to latitude and longitude respectively.)

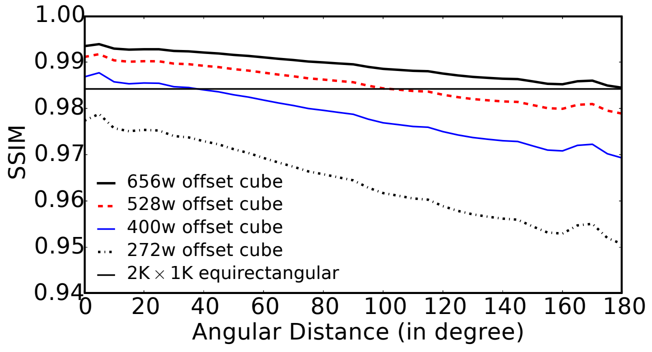
For each of these rendered views, we compute the angular distance between θ_{offset} and θ_{view}. Angular distance is the angle subtended by lines drawn from the centre of the sphere to each of the two points on the surface of a sphere [7]. In this paper, we compute angular distance in degrees. The maximum angular distance on

[3] 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

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



(a) PSNR



(b) SSIM

Figure 5: PSNR and SSIM between views rendered by offset cubes and views rendered from the original 8K frame. The X-axis in the graph shows the angular distance between the view orientation and the offset cube's orientation. 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.

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}.

For comparison, we downsize the original 8K equirectangular image to 2K×1K resolution and calculated 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 2664 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 angular distance between θ_{offset} and θ_{view} against PSNR and SSIM. As expected, both PSNR and SSIM decrease as the angular distance 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 36.1 dB when the angular distance between θ_{offset} and θ_{view} is 0 degrees to 32.3 dB when the angular distance is 180 degrees (i.e., the maximum). The SSIM value for `656w` also decreases from 0.9934 to 0.9845 as angular distance increases. Views rendered by offset cubes of higher quality level give better visual quality as well. On average, the PSNR values of the `656w` quality level are 1.37 dB better compared to the `528w` quality level and 5.40 dB better than the lowest, `272w` quality level.

Figure 5 also shows that when the angular distance is smaller than 40 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 40 degrees of θ_{view}.

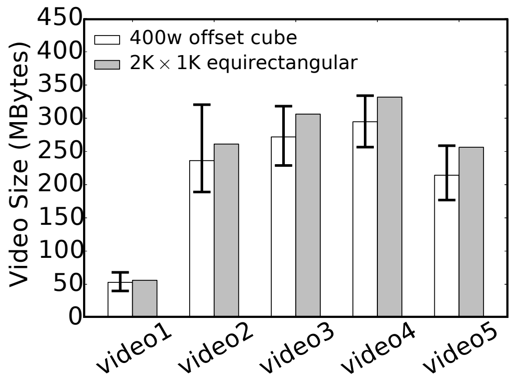


Figure 6: Average sizes of 22 offset cubes encoded at 400w quality compared to equirectangular at 2K×1K resolution. The error bars show the minimum and maximum sizes of all offset cubes at 400w quality of a specific video.

4.1 Video Size Reduction

While offset cube images can render similar or better visual quality using only less than half of the pixels compared to the equirectangular projection, we find that the reduction in video file sizes is not as significant.

We first removed the audio tracks from videos and then transoded the original videos from 8K quality into both a 2K×1K equirectangular version as well as into offset cubes in 22 different orientations with quality level `400w` using FFmpeg [6] with the x264 encoder. For all transcodings we used the following x264 settings: `keyint=30:min-keyint=30:no-scenecut -profile:v "high" -crf 18`. Final transcoded file sizes are shown in Figure 6. The average file sizes of 22 offset cubes are only 5.6%, 9.5%, 11.2%, 11.2%, and 16.4% smaller than their corresponding 2K×1K equirectangular versions, respectively. For the second video, the offset cube's representation was larger, by 22.7%, than the more-pixel-dense equirectangular version.

We speculate that the transformation used by the offset cube reduces the effectiveness of video compression by introducing distortions in the projected image that are not easily processed by existing AVC encoders. For example, changes in scale or curvature could impede motion estimation or not be well-represented by DCT coefficients.

5 ADAPTATION OVER BOTH QUALITY LEVEL AND VIEW ORIENTATION

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

5.1 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 bitrate. 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) [16] 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, 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 to select higher or lower bitrates for different areas of the 360 degree view. Figure 7 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 7, Representation attributes `FBYaw` and `FBPitch` repesent θ_{offset}, orientation of the offset cube. `FBOffcenter_z` is used for deriving vector b in Figure 3. For example, we can construct a cube around a unit sphere so that the center of the cube's front face is oriented at $< \text{yaw} = 0, \text{pitch} = 0 >$ and located at $(0, 0, 1)$ in the cartesian coordinate system. In this case, vector b is `FBOffcenter_z`$\times(0, 0, 1) = (0, 0,$ `FBOffcenter_z`$)$.

During 360 video streaming, the Oculus player has two decisions to make: (i) which of the 22 differently-oriented offset cubes will perform best for the user's view, and (ii) 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 13 Mbps and 22 Mbps.

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.

5.2 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). GearVR is one of the most affordable and popular portable VR device available on the market. We use the native Samsung ROM and update it to the latest version of Android 6.0.1. The Oculus application comes pre-installed on S7 and supports immersive video streaming. In a typical personal virtual reality setup, the S7 is installed inside the GearVR and provides 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 precisely repeat the same sequence of head movements. To address this problem, we designed a mechanical testbed that can be used move a VR repeatedly through a sequence of motions, allowing repeatable experiments. Figure 8 shows the setup of this testbed.

In this testbed, we emulate user's head motion during video playback using a customized, controllable Pan/Tilt mount [9]. This mount combines two separate Hitec HS-422 servo motors that can simulate yaw and pitch motions, respectively. With torque power of 57 oz-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

```
<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="2160p" 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>
```

Figure 7: MPD document of an Oculus 360-degree video on Facebook.

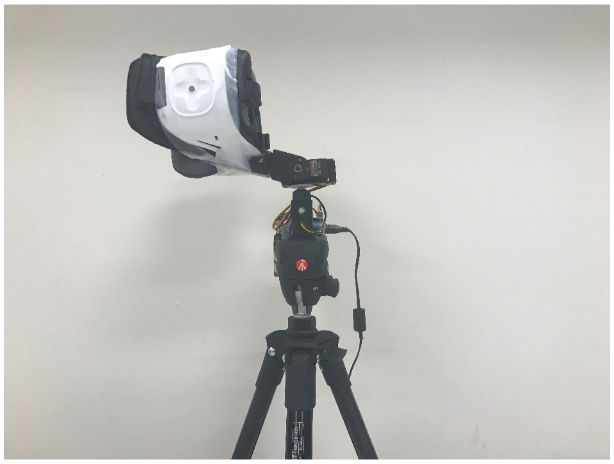


Figure 8: Our VR testbed consists of a GearVR, an S7, a Pan/Tilt mount, a 3D-printed holder for the mount, a Raspberry Pi to accurately control the mount's motion, and a tripod for stabilizing the test instruments.

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, multi-device experiments with controllable, emulated head motion.

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" posted by OneRepublic [12]. This video was shot with Nokia OZO [11]. In total, this video is 4 minutes 43.115 seconds long and has 282 segments.

5.3 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 are 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

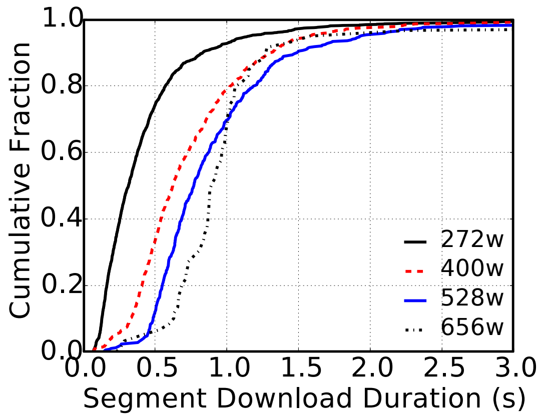


Figure 9: Segment Downloading Duration Distribution.

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 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 one second of streaming content, segment downloading duration must be smaller than one second for smooth playback. Figure 9 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 one 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 one minute. After the network connection was cut, we found that the video playback could continue for another five seconds. We therefore estimate that the playback buffer contains roughly five seconds of video content.

5.4 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, pre-downloaded 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 non-ideal 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 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.

5.4.1 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 4.5 Mbps. This 4.5 Mbps 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' 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 setup 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 50 ms to rotate 5 degrees and 800 ms 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 6 rotations, 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 3 trials and report the average.

The results are shown in Figure 10. 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, 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 a time for a total of 6 times, and 6 (2.1% of the total 282 segments) wasted segments when the motor rotates 90 degrees a time for a total of 2 times.

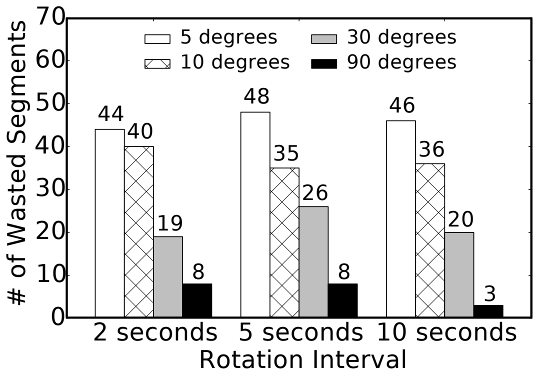


Figure 10: Number of wasted segments under different emulated test settings. For all emulated tests reported in this figure, the motor rotates a total of 180 degrees in yaw motion. "5 degrees" in the legend means the motor rotated 5 degrees a time, 36 times in total. The total number of segments in our test video is 282.

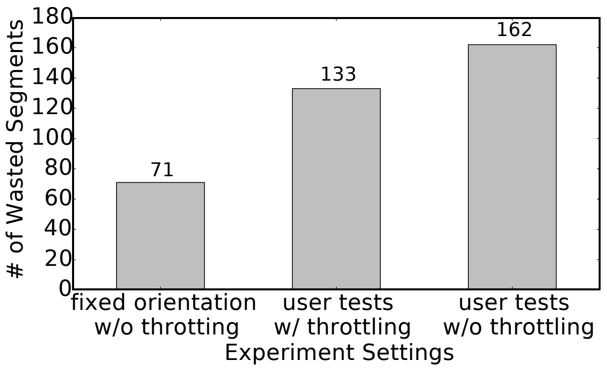


Figure 11: Number of wasted segments under fixed orientation tests and real user tests. The total number of segments in our test video is 282.

5.4.2 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" [13] to estimate the downlink bandwidth of the S7 and to ensure that the network bandwidth is greater than 20 Mbps. We conducted 3 fixed orientation tests. The results are shown in Figure 11 and Figure 12. 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 13 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`.

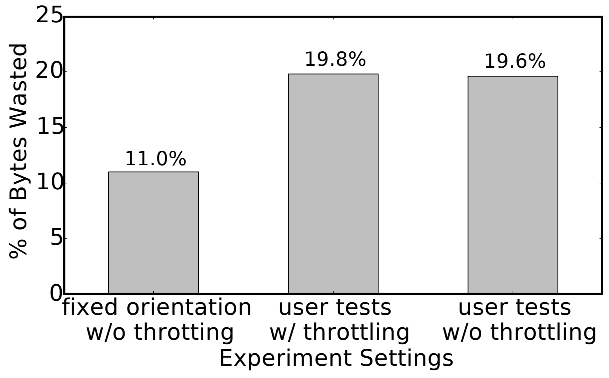


Figure 12: Percentage of downloaded bytes wasted under fixed orientation tests and real user tests.

5.4.3 Real user tests: both view orientation adaptation and quality level adaptation. We also asked 5 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 11). 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 12 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 20 Mbps before each test starts. The results are shown in Figure 11 and Figure 12. 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 14 shows the results. While only 42.7% segments were downloaded at 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 to be wasted. When switching to a new orientation, Oculus prefers to download segments in low quality to avoid missing the timely playback deadline.

Factors such as the frequency, speed, and distance of user head movements can affect the number of wasted segments. Users' head movements not only depend on user behavior but also on the characteristics of video being watched. We plan to conduct future, more comprehensive studies with real user input to more precisely characterize 360 degree video adaptive streaming algorithm behavior.

6 CONCLUSION

In this paper, we analyzed the current state of Oculus 360 degree video streaming. A key finding from our analysis of Oculus' offset cubic projection lies understanding its method of devoting more

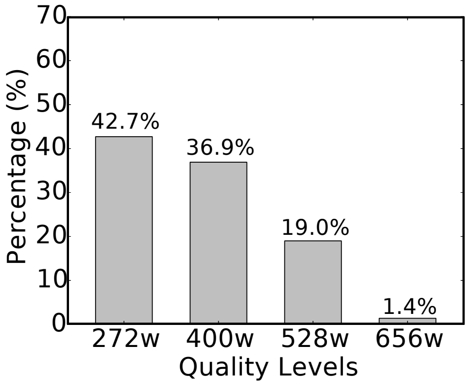


Figure 13: Percentage of segments downloaded at each quality level in fixed orientation tests.

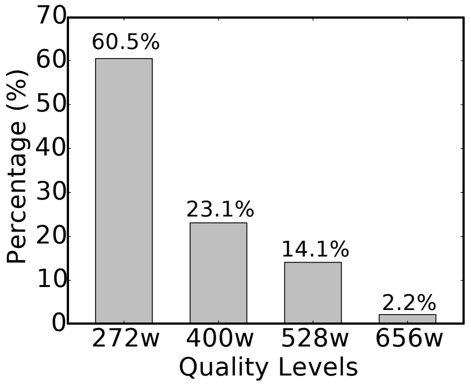


Figure 14: Percentage of segments downloaded at each quality level in real user tests.

information to a direction on the 360 degree view. It does so, through a simple, yet novel method of distorting spherical angles. This direction-specific distortion allows the offset cubic projection to transmit information about a user's view more efficiently than unoriented projections when a streaming device understands which portion of the image is being viewed.

We quantified the effectiveness of the offset cube encoding compared to the standard equirectangular representation by generating a large sample of views from offset cube projections in different orientations. As expected, we found that the farther away the view is from the offset cube's orientation, the lower the quality of the view. However, view quality remains high when the user is oriented within 40 degrees of the offset cube's direction. Within this region, qualities are better or comparable to the equirectangular view but use less than half of the pixels.

Other central findings in our measurements occurred through evaluation of Oculus' dynamic adaptive streaming algorithm. When streaming segments encoded using the offset cubic projection, the streaming client must select segments that can vary in both quality level and orientation. To evaluate Oculus' streaming strategies, we both emulated 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, like Oculus' work on the offset cube, will become increasingly important. Our analysis of the offset cube 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.

7 ACKNOWLEDGEMENT

We appreciate constructive comments from our shepherd Gwendal Simon and anonymous referees. This work is partially supported by High-Tech Research and Development Program of China ("863-China Cloud" Major Program) under grant 2015AA01A201, by National Natural Science Foundation of China (NSFC) under grants 61471217, 61432002 and 61632020, by CCF-Tencent Open Fund under grant RAGR20160105, and by NSF under grant CNS-1618931.

REFERENCES

[1] Akamai's [state of the internet] q1 2016 report. https://www.akamai.com/uk/en/multimedia/documents/state-of-the-internet/akamai-state-of-the-internet-report-q1-2016.pdf.
[2] Charles Proxy. https://www.charlesproxy.com/.
[3] Cubic Projection. http://wiki.panotools.org/Cubic_Projection.
[4] Equirectangular Projection. http://mathworld.wolfram.com/EquirectangularProjection.html.
[5] Facebook's 2 Billion Acquisition Of Oculus Closes, Now Official. https://techcrunch.com/2014/07/21/facebooks-acquisition-of-oculus-closes-now-official/.
[6] FFmpeg. http://www.ffmpeg.org/.
[7] Great Cicle. http://mathworld.wolfram.com/GreatCircle.html.
[8] Internet Connection Speed Recommendations. https://help.netflix.com/en/node/306.
[9] Lynxmotion Pan and Tilt Kit / Aluminium. http://www.robotshop.com/en/lynxmotion-pan-and-tilt-kit-aluminium2.html.
[10] Next-generation video encoding techniques for 360 video and VR. https://code.facebook.com/posts/1126354007399553/next-generation-video-encoding-techniques-for-360-video-and-vr/.
[11] Nokia OZO. http://ozo.nokia.com.
[12] OneRepublic - Kids (360 version). https://www.facebook.com/OneRepublic/videos/10154946797263912/.
[13] SPEEDTEST. http://www.speedtest.net/.
[14] Under the hood: Building 360 video. https://code.facebook.com/posts/1638767863078802/under-the-hood-building-360-video/.
[15] Xavier Corbillon, Alisa Devlic, Gwendal Simon, and Jacob Chakareski. Viewport-adaptive navigable 360-degree video delivery. *arXiv preprint arXiv:1609.08042*, 2016.
[16] 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, May 2014.
[17] Daisuke Ochi, Yutaka Kunita, Akio Kameda, Akira Kojima, and Shinnosuke Iwaki. Live streaming system for omnidirectional video. In *2015 IEEE Virtual Reality (VR)*, pages 349–350. IEEE, 2015.
[18] 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.
[19] David Salomon. *Transformations and projections in computer graphics.* Springer Science & Business Media, 2007.
[20] Kay M Stanney, Ronald R Mourant, and Robert S Kennedy. Human factors issues in virtual environments: A review of the literature. *Presence*, 7(4):327–351, 1998.
[21] 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.