CSC 499 Report
Nicolaus Karamanian - V00819384

Virtual Reality Viewpoint Prediction

Introduction

In the past half decade, virtual reality (VR) has exponentially grown more popular. Much of this is attributed to the popularity of gaming, a market of which VR has penetrated. The Oculus Rift CV1, the first at-home consumer VR head-mounted device (HMD), saw its release in March 2016. Since then, competing companies such as Valve, Facebook, HTC, and Microsoft have pushed the innovation of VR technology in order to appeal to the consumer driven market.

VR attempts to emulate the real world to the user by immersing their sense of sight and sound (and sometimes touch) into a virtual world. High quality 360-degree video streams through the lenses of the HMD to the user. 360 video is either recorded by an omnidirectional camera or created virtually, and subsequently projected onto a sphere. The user's frame of reference is inside the video sphere where the video is played on all sides. VR is an interactive multimedia experience because 360 video allows for users to shift their perspective by turning their head.

The competition as described above has led to rapid development. New HMDs adding more capabilities such as arm and finger tracking (seen in figure 1) has led to VR getting much closer to a lifelike simulation. This has led to VR being adopted by other disciplines such as education and training for occupations like surgeons and pilots. However, these innovations add more computation to an already demanding system.

Figure 1: Valve Index arm & finger tracking in Boneworks, courtesy of Node

Problem Description

As new advancements are made for VR, the challenges and demands imposed by it are also increasing. 360-degree video is the primary cause, as it consumes 4 to 6 times the computation and bandwidth of regular 2D video [1]. This is a major problem for both consumers and producers. The main consumer problem is the cost of entry to VR; it is far too large currently for mainstream adoption as a result of the high computation power required. While prices for HMDs have dropped in recent years, the price of powerful PC hardware required to run even basic VR applications have not. On the producer end, the bandwidth required to deliver 360 video streams is high enough to alienate many potential consumers as well.

Another issue is the quality requirements of VR. Because it is trying to simulate how the real world feels, consistent high framerates and frame quality must be preserved. Oculus set the standard for VR to maintain 90 frames per second at all times, with a resolution of 1080x1200 for each eye lens. Any low frame rates, stutters, or screen tears will induce motion sickness in the viewer, completely dropping the usability of the technology. More

recent HMDs have only increased these requirements, which are now far higher than the minimum requirements of any other consumer multimedia system. Computation power will continue to be a barrier to entry until optimizations for VR video are made.

Viewpoint Prediction

The key solution to reducing computation for VR is to optimize 360-degree video. It uses 4-6 times the computation of 2D video because of the increased space of having the video span the sphere surrounding the user's head. Current practices render the entire spherical video to be streamed, despite users only viewing a small portion at any given time (typically 20%). Figure 2 illustrates the user's Field of View (FOV) on the sphere, which is the highlighted rectangle. The coordinate representing the center of the user's FOV is the *viewpoint*.

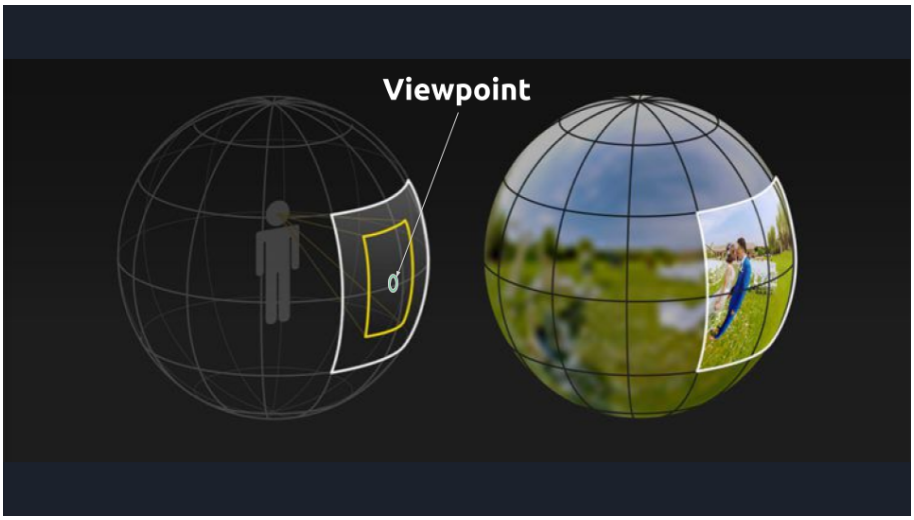


Figure 2: 360-degree video sphere and user FOV & viewpoint

Clearly, rendering 80% of the sphere that the user is not currently viewing is computing a large amount of redundant information. The current render cycle for VR is: render the entire 360 frame, obtain the viewpoint from the user's HMD, then stream the corresponding FOV of the rendered 360 frame back to the user's HMD screen. Early propositions were

to obtain the user's viewpoint first, then render only the corresponding FOV on the 360 frame to be streamed to the user, then stream it back. However this new model imposes a significant issue. It introduces too much of a delay between the HMD sending the sensed viewpoint, and the computer streaming back the corresponding FOV video. Rendering is the most computationally intensive operation in the cycle. By the time the FOV on the 360 video sphere is streamed back to the user, the user's viewpoint will likely have changed. When immersed in VR, the user's brain will expect the user's FOV to change while moving their head, as in real life. If there is a large delay to this due high latency, it will likely induce extreme motion sickness. This problem is far more significant than low framerates and is a worst case scenerio, so this model cannot be used. Figure 3 provides a visual for the models discussed thus far.

(a) Current model (b) Realtime FOV model (too much latency)

Figure 3: VR Render Models

Rendering only the current FOV is not a usable model, however, the same idea can be applied to the current model. If the current FOV can be predicted before the viewpoint is provided, then as in the realtime FOV model, only the corresponding portion of the 360 video sphere represented by the viewpoint will be rendered. This is the model for *viewpoint prediction*. If the prediction is perfect, then this new model acts the exact same as the Realtime FOV model above, but with the complete removal of latency. Perfect prediction

results in 80% saved computation and bandwidth [1], however in practice this is impossible due to the random nature of human viewers changing their viewpoints.

Proposed Methods

The following subsections are various methods, implementations, and results for various viewpoint prediction models. However, to help understand how much computation can be saved, we will run through a quick thought experiment. Imagine that the 360 video sphere is cut into two halves respective to the front and back of the user's current FOV, as shown in figure 4. The previous viewpoint of the user is known when rendering the current frame. If we assume that a human user cannot turn their head to view the back half of the frame within 1/90 of a second (due to 90 frames per second), then we can omit rendering the back half of the sphere for the next frame. This simple model is already able to attain a 50% render reduction. The following described methods aim to further optimize this reduction.

Figure 4: Front and back halves of a 360 video frame

Head Movement Prediction

The user's current viewpoint, as provided by the VR HMD, is the combination of pitch, roll, and yaw (X, Y, and Z) angles representing their head orientation (shown in figure 5). The X angle represents looking up and down, Y is side to side, and Z is tilting left and right.

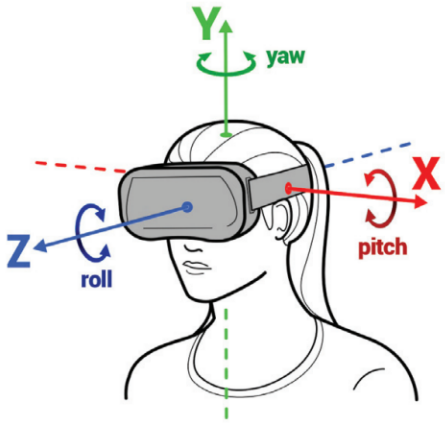


Figure 5: Pitch, roll, and yaw angles

Head Movement Prediction aims to generate the next viewpoint by specifically predicting the X and Y for the next frame based on previous orientation data. Yanan Bao et al propose to train a neural network on previous angle data to create a prediction model [1]. Their implementation and results are a good summary of this method, so it will be used as an example to describe it.

The X and Y angles are split into separate data for separate predictions. A prediction lookahead (sliding window) for the prediction models is selected. The sliding window is the amount of time or number of frames in the future that the angles will be predicted for. It is also the amount of time in the past that the model will have access to prediction input data. The frame lookahead is calculated by multiplying the lookahead by the framerate. Yanan Bao et al tested sliding windows ranging from 0.1 to 1.0 seconds. Lower sliding windows are more accurate (it is easier to predict closer in the future), however calculating the prediction

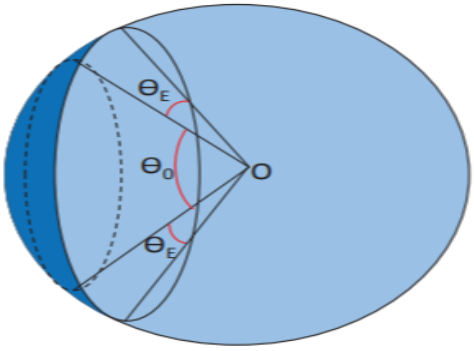
itself is intensive, so higher sliding windows will save more bandwidth. For example, a 0.1 second lookahead decrease was found to decrease the failure ratio by 10x at the cost of 15 to 20 percent higher computation. They suggest that 0.2 seconds (or $0.2 * 90 = 18$ frames) is a good balance and obtains best results.

The failure ratio is the correctly predicted pixels withing the user's current FOV divided by the total FOV pixels, and represents the prediction accuracy. For example, correctly predicting 99/100 pixels within the FOV is a failure ratio of 1%. The suggested failure ratio to maintain to balance between computation and accuracy is 0.1%, which should make the predicted render area indistinguishable from the current model to users.

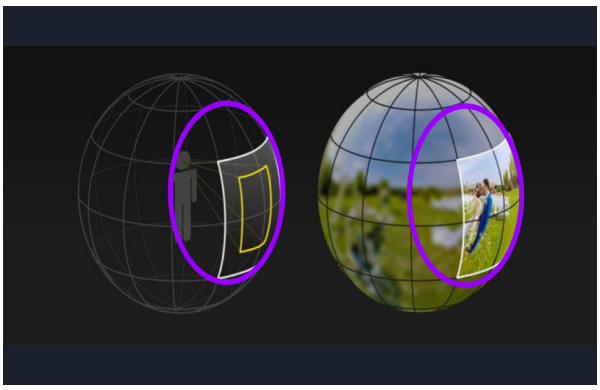
To minimize computation and failure ratio, three models are compared. The first model is a naive model used for control, where the next predicted viewpoint is the same as the current viewpoint. Surprisingly, this basic model performs better than expected, likely due to users looking in the same direction much of the time. A linear regression model is also trained, and while it performs about twice as well as the naive model, it is not the best option. A neural network (NN) trained to minimize the sum of squared error performs best. The suggested model is 3 layers and 5 hidden neurons with 50% data trained and 50% tested, however as will be mentioned later, the chosen numbers have little impact and likely these gave best results to this study.

In order to maintain the 0.1% failure ratio, the deviation of the original prediction (error) is also predicted. This deviation prediction is very important because it will act as an error net to account for all small misses by the original model. Like the original prediction, a NN is also used for this, and trained to calculate errors between the first and final frames of the sliding window. Many similar studies do not include deviation prediction, which leads to less efficient error handling.

Once the viewpoint angles are predicted, an FOV on the 360 video sphere is generated. The proposed model constructs the total FOV size around the predicted viewpoint coordinate, then adds/extends the FOV by the deviation prediction amounts. The constructed predicted FOV is the transmission area to be streamed to the HMD; figure 6 shows the model and an example of what portion of the sphere it would correspond to.



(a) Modelled original and deviation angles



(b) Example of predicted area on video sphere

Figure 6: VR FOV transmission area

With the suggested sliding window of 0.2 seconds and a failure ratio of 0.1%, the proposed NN model was able to attain a render reduction of 75%. When adding the error of X and Y predictions, this model is about 5% short of consistent perfect FOV predictions.

A different weighted linear regression model to predict angles was proposed by Feng Qian et al [2]. It is linear regression where angles from frames more recent to the current sample are considered more important, and given a higher weighting. The model performs slightly better than regular linear regression, however it performs worse than a NN in the long term. Instead of deviation prediction, they elect to grow the predicted FOV transmission area gradually as more errors are reduced to account for outliers. However, this can lead to either noticeable stutters if the FOV has not grown enough yet, or redundant bandwidth consumption if it has grown too much. The previous NN model remains the best performing model.

As an aside, Yanan Bao et al later propose to extend their head prediction model to optimize VR multicasting [3]. Streaming a single 360-degree video is already very bandwidth intensive, so multicasting multiple 360 videos, each showing a seperate FOV for each individual viewer, is impossible for the typical consumer. The proposed model predicts each viewers' viewpoint with an enhanced version of the above NN model, which oversamples cases with large errors. Locations on the 360 sphere of the multicasted video which are predicted to have a single viewer's FOV have the corresponding transmission area unicasted to that specific user. Locations predicted to have multiple viewers has the transmission area either multicasted or unicasted to each specific user, according to a cost efficiency comparison. The model results in a total bandwidth reduction of over 50% when compared to fully multicasting the entire sphere to each user.

Tiling Prediction

If head movement prediction directly predicts human actions, tiling prediction represents the non-human element. The 360 video sphere cut into separate segments, or "tiles", that detect whether they are within the user's FOV. Instead of predicting the next viewpoint based on exact angles, the tiles within the next FOV are predicted. The amount of segmented tiles will determine how accurate the prediction is, and also how much extra computation is required. In general, this method is less computationally intensive than head motion prediction, at the cost of a lower accuracy. Xueshi Hou et al propose a model that leads to good results [4], so their implementation will be used as an example.

The proposed model cuts the 360 video sphere into 72 tiles. Each tile is one hot encoded: tiles within the user's FOV are encoded as 1 and tiles outside the FOV are encoded 0; figure 7 illustrates this. A sliding window of 2.0 seconds is suggested, which is evidently higher than the chosen one for head motion prediction. Predicting 72 tiles is less complicated

than predicting exact angles, so this method benefits from the lower computation of a high prediction lookahead without a large expense of accuracy.

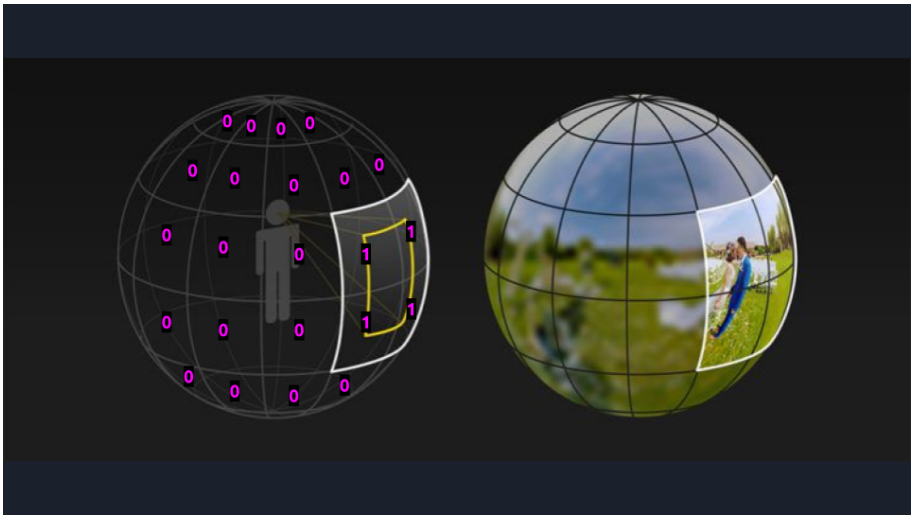


Figure 7: One-hot tiling encoding

Each one-hot encoded frame within the sliding window adds to create a 72 by n one-hot matrix (n frames lookahead) for input data. The prediction model is another neural network, implemented as a multi-layered Long Short-Term Memory (LSTM) model (128 LSTM units, 72 nodes). The NN is trained separately on each tested application.

The output of the NN model is each tile's respective probability of being within the next FOV. The size of the entire FOV is generated around the selected tile with the highest probability, which is subsequently rendered and streamed to the HMD. Selecting more than one tile will increase accuracy at the expense of more rendering; figure 8 demonstrates selecting two tiles. A FOV accuracy of at least 95% is suggested to maintain the original quality, and various applications each may require a different number of selected tiles to attain it.

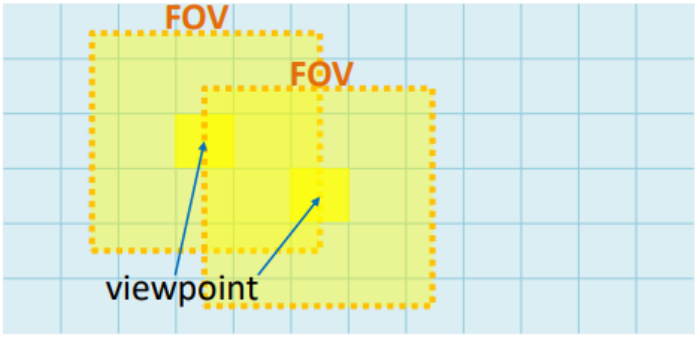


Figure 8: FOV generation around two high probability tiles

The proposed model was able to attain 70% render reduction for regular 360-video applications while using 4 to 5 selected tiles. However, they also tested the model on a much more intensive VR experience (Kong VR) and separated medium motion sequences from high motion sequences, such as quick 180 degree turns. The resulting render reduction dropped, as expected, to 55.7% for medium and 43.7% for high. While performing considerably worse than the average for head motion prediction, the computation cost efficiency of this method is not to be overlooked.

Optimization Attempts & Comparisons

Individual Methods

Before considering how the above mentioned viewpoint prediction methods can build on each other, a few attempts to optimize the existing models are shown below.

Head Motion Prediction

Prior to looking for optimizations, the same model proposed by Yanan Bao et al [1] was applied to larger dataset. The dataset they used for prediction was the temporal angle data for 153 new VR users. The dataset the model is applied to is the same as the one used for

the tiling prediction study above [4], 36 000 random VR users. The application the data is trained and tested on is the intensive application used in the tiling study (Kong VR). High and medium motion sequences were not separated, so a total average render reduction would be obtained. The application is chosen to simulate a worst case analysis.

The model used is a basic neural network with 3 layers and 5 hidden neurons with a sliding window of 0.2 seconds (50% training, 50% testing). The resulting mean render reduction lowered to 56%, a sharp drop from their reported average of 75%, but also higher than the tiling equivalent.

Next, optimizations within the neural network implementation were explored. The implementation was done and altered using Keras in Python, so the libraries for various NN implementations were easily available. First, a large variety of the number of layers and hidden neurons were altered and tested for the basic NN implementation (where the original was 3 and 5 respectively). Various other models were also tested with various parameters including the following: Long Short-Term Memory (128 LSTM units), Stacked Sparse Autoencoders (18 timestamps for 18 lookahead frames), Bootstrap-Aggregated Decision Trees (18 timestamps, 30 bagged decision trees), Weighted k-Nearest Neighbors (18 timestamps, 100 nearest neighbors).

Unfortunately, no variety of implementations and parameters improved the original resulting average render reduction of 56% by more than 1%. For the sake of consistency, the implementation of choice moving forward will be the originally proposed basic NN with 3 layers and 5 hidden neurons.

After the attempts above, it is evident that many variables contribute to the lowered render reduction. The foremost reason is likely that the applications the original model were

trained and tested on caused more predictable user movements than the intensive application used here. Another reason could be that all of the 153 users from the original dataset were new VR users, while many contained in the new 36 000 are likely experienced. Likely, it is much easier to predict how a collection of new VR users orientate versus veteran VR users.

To draw a consistent paradigm between models, the new model was tested on 153 random users from the 36 000, and surprisingly, the resulting average render reduction rose to 62%. Noting this sharp increase, 10 batches of varying quantities of randomly selected users from the 36 000 were trained and tested using the same model. Table 1 below displays the results.

User Batch Count	Render Reduction %
36 000	56.3
10 000	56.3
1 000	59.2
153	62.4
100	61.7
10	65.0
3	67.9
1	68.1

Table 1: Mean render reduction for various user batch sizes

While the above table displays an interesting trend, a general prediction model cannot be built from a low amount of users. The results may imply though that perhaps some sort of an individual prediction algorithm can be placed in each HMD, which creates a prediction model tailored to each specific user. The predictions would improve over time as the user uses the HMD more.

Tiling Prediction

Many studies have been published for tiling, so it is tough to consider methods to optimize tiling prediction that have not already been tried. Access to the same dataset as study by

Xueshi Hou et al [4] described above is available, so the results can be verified.

In order to compare tiling prediction to head motion prediction fairly, the model was retrained on the intensive application, with this time not separating high and medium motion sequences. The resulting mean render reduction (pixel savings) is 48%, still much lower than 56% as achieved with head motion prediction.

To optimize tiling prediction, the sliding window is reduced to 0.2 seconds (the same as the head motion prediction model) and the number of tiles are increased. At first, extremely large numbers such as 1 000 000 tiles were considered for comparison, however due to computation limits of hardware available for this report, the max tiles tested is 1000. Table 2 below shows the varying results.

Tile Count	Sliding Window	Render Reduction %
72	2.0	48.0
72	0.2	49.2
500	2.0	51.8
500	0.2	53.1
1 000	2.0	51.9
1 000	0.2	53.4

Table 2: Mean render reduction for various tile counts and sliding windows

Before considering the optimized results, it should be noted that running the prediction algorithm for 500 tiles took well over 10 times the amount of time that 72 tiles did, and 1000 tiles over 50 times. It is safe to conclude on this fact alone that the tile numbers should not be increased further, 72 is a good balance. Also, a 1% increase in render reduction is not enough to justify decreasing the sliding window lower than 2.0 seconds at the expense of higher computation or bandwidth.

Combined Method

Naive Model

After attempting to optimize the individual methods separately, a way of combining the methods for an increase in accuracy and performance is considered. The motivation to combine is the fact that head motion prediction predicts a human element, and tile prediction predicts a non-human element; perhaps combining both will lead to interesting results.

The first attempt is a naive control method. Both methods are considered equally and the FOVs generated by both method's viewpoint prediction are added together. The implementations are unaltered from their original studies due to reasons discussed in the previous section. The application trained and tested on is the same intensive application as the last section.

The resulting mean render reduction is 40%, considerably lower than each method's results individually (56% for head motion prediction, 48% for tiling prediction). This result is as expected, the generated FOVs likely increase accuracy by adding together, but the errors of both methods also add. The larger sum of errors results in a larger render transmission area.

Weighted Model

A approach is considered next. The idea is to assign weights to each method, representing how much to "rely" on each method. The weights represent how much the allowed failure ratio is divided by. For this model, a total failure ratio of 0.1% (or 99% accuracy) must be maintained. However, each method's individual allowed failure ratios can be altered by the weight. For example, assigning a weight of 0.1 to the tiling prediction method and 1.0 to head motion prediction means the head motion prediction model performs as normal, but the

tiling prediction model's allowed failure ratio is 10 times higher, meaning that the maintained accuracy can be 10 times lower. Lower accuracy requirements decrease the generated FOV size, so in this case, the generated FOV of tiling prediction has a requirement 10 times smaller than normal and will minimize its size, while the head motion predicted FOV is the same size. This means that head motion prediction is more "relied" upon, a higher accuracy requirement resulted in less error; or in other words the generated FOV is larger because the errors are smaller. If both methods were assigned weights of 1, the weighted model becomes the naive model. Table 3 shows various weights.

Head Motion	Tiling	Render Reduction %
0.01	1.0	48.0
0.1	1.0	48.0
0.5	1.0	47.7
0.8	1.0	43.8
0.9	0.6	52.9
1.0	1.0	40.2
1.0	0.8	49.2
1.0	0.5	54.6
1.0	0.1	56.3
1.0	0.01	56.3

Table 3: Mean render reduction for various prediction model weights

As expected, "relying" on methods individually provides better results. The trend shown also proves that tiling prediction is clearly the weak link and major source of error. Relying less on tiling prediction led to much higher render reduction. It should be noted that even slightly increasing the failure rate vastly reduces computation and bandwidth, which was the motivation to try to find a combination where both failure rates are increased, but adding the methods together still results in a 0.1% total failure rate. The interesting result is the weights of 0.9 and 0.6 for head motion and tiling respectively; the computation for predicting is lower than 1.0 and 0.8, but the render reduction is almost 5% higher. A weight of 0.6 was

the lowest optimal number for tiling while limiting head motion to a weight of 0.9. If either weights are any lower, a failure rate of 0.1% is unattainable.

Realtime Methods

One thing to consider unmentioned thus far is that in order for viewpoint prediction to be a valid method, there are two fundamental requirements. First, the HMD must have a good, fast connection to the machine rendering the 360-degree video. Second, the device rendering the 360 video must be able to calculate the prediction quick enough to be unnoticed by users.

Neither of these requirements are met by low quality (typically mobile) VR products such as smart phones. Typically, they are not used for most VR purposes, but purely for streaming 360 video events. The quality requirement is not nearly as high as the specifications for regular/high end VR as mentioned early in this report, but that does not change the large amount of bandwidth required to stream 360 videos.

Viewpoint prediction is not a feasible method for such low end devices, however many models that rely on current viewpoints have been proposed. Despite the latency such methods introduce, they are sometimes required for a smoother experience, and are acceptable for the low requirements.

Mohammed Hosseini et al propose a dynamic view-aware bandwidth efficient adaption technique [5]. A 3D hexaface sphere mesh is projected onto the 360 video sphere, which cuts the video into six segments (with four sections emphasizing the middle of the video). The MPEG-DASH SRD specification encodes and defines the reference space for each tile. The HMDs viewport X,Y and Z axis is tracked and normalized direction vectors are calculated to determine the user's FOV. High quality bits are sent to segments within the FOV, and

low quality to segments outside. This method saves upwards of 70% of bandwidth for high quality videos, however the trade off is that the switch from low quality segments to high quality takes a noticeable amount of time, and users will clearly see low quality segment edges if changing viewpoints quickly.

A similar model is proposed by Mario Graf et al that focuses much more on bandwidth savings at a higher cost to quality [6]. Instead of six, the model uses 6x4 tiles on the sphere, and elects to send high quality bits to the tiles within the current FOV. Tiles outside the FOV are suggested to have either very low quality bits, or no bits at all (blank tiles). Users will constantly notice low quality (or blank) tiles if changing viewpoints too quickly, but bandwidth savings are between 70 and 80 percent.

NTT Media Intelligence Laboratories and SWANGO Co., Ltd. successfully developed and tested a live streaming 360-degree video service [7], which has been replicated by large companies since. The sphere is divided into seven overlapping tiles, where all tiles within the user's FOV (including overlaps) are requested at a high bitrate. Tiles outside are, again, a low bitrate. Previous high quality tiles outside the current FOV remain high quality until new high quality tiles are delivered. They sucessfully streamed 185 minutes to over 10 000 viewers with no bandwidth problems. However, again, the usability suffered on the consumer end due to a 5 second latency between tile quality changes.

Again, the above mentioned methods are purely meant for low end devices currently. However, if rendering times and latency vastly speed up for future machines, then these methods become optimal render reduction solutions and would completely replace any viewpoint prediction methods.

Conclusion

The viewpoint prediction methods discussed in this report introduce only benefits to virtual reality. The rendered area of a 360-degree video sphere is reduced by as much as 75% at no cost perceived by the user. If a computer has the necessary requirements to render 360-degree video and run VR applications, then also dedicating the relatively low computation to predicting the next viewpoint should be trivial. As shown when attempting to combine methods, head motion prediction seemingly is the current best model accuracy-wise, with a render reduction of between 56 and 75 percent. However, if a low computation alternative is desired, tiling prediction saves between 48 to 70 percent and is a viable alternative.

References

[1] Y. Bao, H. Wu, T. Zhang, A. A. Ramli, and X. Liu, "Shooting a moving target: Motion-prediction-based transmission for 360-degree videos," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 1161–1170.

[2] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 video delivery over cellular networks," in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, 2016, pp. 1–6.

[3] Y. Bao, T. Zhang, A. Pande, H. Wu, and X. Liu, "Motion-prediction-based multicast for 360-degree video transmissions," in *2017 14th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2017, pp. 1–9.

[4] X. Hou, S. Dey, J. Zhang, and M. Budagavi, "Predictive view generation to enable mobile 360-degree and vr experiences," in *Proceedings of the 2018 Morning Workshop on Virtual Reality and Augmented Reality Network*, 2018, pp. 20–26.

[5] M. Hosseini and V. Swaminathan, "Adaptive 360 vr video streaming: Divide and conquer," in *2016 IEEE International Symposium on Multimedia (ISM)*. IEEE, 2016, pp. 107–110.

[6] M. Graf, C. Timmerer, and C. Mueller, "Towards bandwidth efficient adaptive streaming of omnidirectional video over http: Design, implementation, and evaluation," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, 2017, pp. 261–271.

[7] D. Ochi, Y. Kunita, A. Kameda, A. Kojima, and S. Iwaki, "Live streaming system for omnidirectional video," in *2015 IEEE Virtual Reality (VR)*. IEEE, 2015, pp. 349–350.