Optimizing Next-Generation Cloud Gaming Platforms with Planar Map Streaming and Distributed Rendering

Pin-Chun Wang[1], Apollo I. Ellis[2], John C. Hart[2], and Cheng-Hsin Hsu[1]
[1]Department of Computer Science, National Tsing Hua University
[2]Department of Computer Science, University of Illinois at Urbana-Champaign

***Abstract*—We propose a new cloud gaming platform to address the limitations of the existing ones. We study the rendering pipeline of 2D planar maps, and convert it into the server and client pipelines. While doing so naturally gives us a distributed rendering platform, compressing 2D planar maps for transmission has never been studied in the literature. In this paper, we propose a compression component for 2D planar maps with several parametrized modules, where the optimal parameters are identified through real experiments. The resulting cloud gaming platform is evaluated through extensive experiments with diverse game scenes. The evaluation results are promising, compared to the state-of-the-art x265 codec, our platform: (i) achieves better perceptual video quality, by up to 0.14 in SSIM, (ii) runs fast, the client pipeline takes ≤ 0.83 ms to render each frame, and (iii) scales well for ultra-high-resolution displays, we observe no bitrate increase when moving from 720p to 1080p, 2K, and 4K displays. The study can be extended in several directions, e.g., we plan to leverage the temporal redundancy of the 2D planar maps, for even better performance.**

I. Introduction

Cloud gaming refers to: (i) running complex computer games on powerful servers in data centers, (ii) capturing, compressing, and streaming game scenes over the Internet, and (iii) interacting with gamers using thin clients on inexpensive computing devices [1]. In the past few years, we have witnessed strong interests in cloud gaming from both the industrial [2] and academic [3] sides. Existing commercial cloud gaming platforms are *video streaming* based, where the cloud servers perform all the rendering tasks, and the thin clients are merely video decoders. Such a design decision treats computer games as *black boxes* and allows cloud gaming service providers to trade *gaming experience* for *time-to-market.* Video streaming based cloud gaming, however, comes with several limitations, including *high bandwidth consumption, limited scalability, and little room for optimization*, and thus calls for *next-generation* cloud gaming platforms [4].

We make some observations on these three limitations:

- **High bandwidth consumption.** Although video streaming based cloud gaming imposes low computation requirements on thin clients, it incurs high networking bandwidth requirements. This is partially caused by *under-utilizing* the computing power of thin clients. Nowadays, even low-cost smartphones, come with GPUs (Graphics Processing Units), which are certainly more capable than video decoders. *Moving some rendering tasks from cloud servers to thin clients may reduce the network bandwidth consumption.*
- **Limited scalability.** Since all the rendering tasks are done on cloud servers, supporting more gamers not only leads to higher bandwidth cost, but also results in higher computational cost on cloud gaming service providers. This in turn makes the cloud gaming less profitable and not scalable to many gamers. *Distributing rendering tasks among thin clients may improve the scalability.*
- **Little room for optimization.** Treating computer games as black boxes prevents cloud gaming platforms from leveraging in-game contexts for performance optimization. *Extracting simplified forms of 3D scenes from computer games may open up a large room for optimization.*

We believe the crucial step of building next-generation cloud gaming platforms is to study the *rendering pipelines* of games for deeper integration between games and platforms.

We study the rendering pipeline of *planar maps* [5, 6, 7], to understand its potential for addressing the three limitations. The planar map is a vector image consisting of points and edges in the plane, in our case, representing the visible triangles of a 3D scene. In this paper, we apply planar maps in cloud gaming platforms. This is achieved in two major steps. As the first step, we propose a distributed pipeline for *planar map rendering.* The crux of the new pipeline is the *compressor*, as the compression of planar maps has never been investigated in the literature. In the second step, we design a compressor for planar maps, consisting of several parameterized modules. Using real game scenes, we systematically derive the optimal parameters for a compressor that is specifically designed for *planar map streaming.* Our experiment results are quite promising. Compared to video streaming based platforms, our planar map based platform: (i) achieves higher perceptual video quality at the same bitrate, (ii) supports complex scenes when considering perceptual video quality, (iii) runs fast, especially at the client side, and (iv) scales well to ultra-high-resolution displays.

II. Related Work

Cloud gaming platforms [3, 8, 9] can be roughly divided into three groups [10]: (i) video, (ii) graphics, and (iii) hybrid streaming. Video streaming refers to gaming platforms in which each frame is rendered completely on the server, and the frames are compressed into video streams for transmission to the client. Graphics streaming describes transmission of game scene data and/or rendering commands of each frame from server to client. Hybrid streaming as implied by the name refers to some clever combination of the above technologies. In this paper, we explore a new graphics streaming approach using planar maps for cloud gaming.

There are several compression algorithms proposed for graphics streaming. For example, Meilander et al. [11] propose: (i) a caching mechanism for rendering commands, (ii) a compression algorithm for rendering instructions, and (iii) multi-layer representation of 3D objects. Nan et al. [12] introduce a hybrid delivery approach, where the server progressively streams the encoded frames and the graphics information. Similarly, Chuah et al. [13] aim to fully leverage the computational power on the client by rendering the low-quality base layer locally, while the server transmits a high-quality enhancement layer. Compared to the existing cloud gaming platforms, our platform: (i) naturally achieves distributed rendering and brings cloud gaming to inexpensive computing devices with weak GPUs, (ii) produces and compresses concise data representations for clients with limited network bandwidth, and (iii) scales to high-resolution game scenes for large displays.

III. Rendering Pipeline of Planar Map

Planar map rendering pipelines, such as the one, proposed in Ellis et al. [5], have not been customized for distributed rendering in the literature. The input of the whole rendering pipeline is the gaming 3D scene and gamer's viewing information. The planar maps are generated in the following components: (i) silhouette detection, (ii) silhouette clipping, (iii) triangle-triangle occlusion, and (iv) vector rendering.

Silhouette detection. The term silhouette is used loosely here to refer to the visual edge or convex contour edge shared by both a frontfacing and backfacing triangle. We leverage the vertex geometry properties instead of mesh topologies to efficiently detect silhouette. In particular, we use a hash table to record all the edges in the 3D scene as entries. Once the hash table is constructed, we detect the silhouette edges by checking whether the corresponding face normals have opposite signs in the z component. Lastly, we mark an edge shared by more than two triangles as a silhouette.

Silhouette clipping. We clip each triangle according to those detected silhouettes. First, we remove the non-overlapping triangle-silhouette pairs using their geometric information. Secondly, we clip each triangle against its list of overlapping edges. We apply a version of Bernstein and Fussells' clipping algorithm [14] to handle clipping with reduced error. Lastly, we tessellate all output polygons from this phase back into triangles.

Triangle-triangle occlusion. The important property we use to remove occluded triangles is that no triangle is partially occluded. That is, we adopt a lightweight centroid test to determine visibility. We discard the triangles whose centroid is overlapped by any other triangle.

Vector rendering. The vector renderer works as follows. First, the vector renderer generates the vertex attribute information including 2D position coordinates, texture coordinates, and normal vectors by barycentric interpolation. Barycentric coordinates are the input and output of the compressor/decompressor discussed later, and they are the main primitives of transmission. We thus interpolate vertex attributes from their original positions in the quasi-static database to the clipped positions. We pass the vertex data, as well as the scene information, such as viewing matrix, to the GPU to render the scenes for gamers.

IV. Using Planar Map in Cloud Gaming

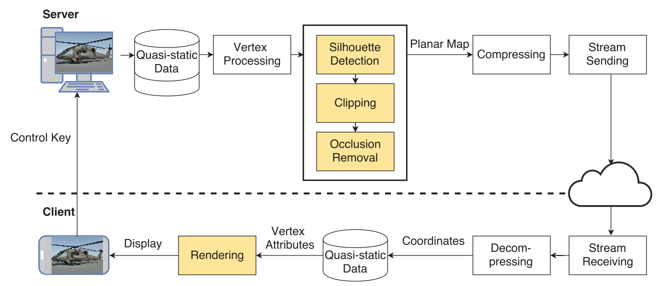


Fig. 1: The revised planar map rendering pipeline for our cloud gaming platform. The shaded boxes are from the ordinary planar map rendering pipeline.

To leverage planar maps in cloud games, we divide the ordinary planar map rendering pipeline into the server and client sides. The first three components are put at the server side, and the vector renderer is put on the client side. The design rationale is to have the light weight vector renderer on the client, so that the thin client can render frames *merely* with per-pixel texturing and shading. To connect the server and client pipelines, we add three additional components to our platform: (i) *compressing*, (ii) *decompressing*, and (iii) *streaming sending/receiving*. We also include two quasi-static databases of 3D models (including vertices, texture, and shaders) at the server and the client. The database at the server contains 3D models for all game scenes, while the one at the client stores a subset of 3D models in the current and nearby scenes.

Fig. 1 shows the distributed planar map rendering pipeline, which works as follows. Starting from the server side, game scenes can be viewed as a set of 3D models from the quasi-static database in the model space. More specifically, scenes are represented in 3D models in file formats, such as obj files. With the typical 3D rendering transformations, we can take 3D models from model space, to world space, to view space, and finally into screen space using vertex processing. Then through the silhouette detection, clipping, and occlusion processes (see Sec. III), we obtain 2D planar maps. The 2D planar maps are then compressed to further reduce the required network bandwidth and streamed to client side through the networks. At the client side, we decode the received data stream into coordinates in the decompression component. The quasi-static database at the client side is pre-populated offline, like most computer games. Note that all the quasi-static data are independent of the viewpoints and control inputs from gamers. Instead, they only depend on the game scenes determined by game states. With barycentric coordinate interpolation and the corresponding quasi-static data, we can render the game frames and display them to the gamer by GPU rendering.

The presented rendering pipelines have the heavier components at the resourceful cloud servers, and the lighter component at the thin clients with weak GPUs. Our key optimization problem is the *compression of planar maps* for mitigating high bandwidth consumption, which, to our best knowledge, has not been rigorously studied in the literature.

V. Compressing Planar Maps

A. Coordinate Systems and Data Format

While the planar maps in the ordinary rendering pipeline [5] are in barycentric coordinate system, the streamed planar maps can be represented using *Cartesian* or *barycentric* coordinate systems. Cartesian coordinates are relative to a single origin, and thus preserve the spatial property across vertices and among video frames. However, Cartesian coordinates do not leverage the (triangles of the) 3D models in the quasi-static database, which may result in unexploited redundancy. In contrast, barycentric coordinates describe each vertex information, including position, normal, and texture, within a triangle (in the quasi-static database) using three floating numbers in $[0, 1]$, say u, v, and $1 - u - v$. The merits of barycentric coordinates include: (i) shorter indexes for triangles, and (ii) common triangle patterns on unclipped triangles. However, barycentric coordinates are related to individual triangles, making the correlation among vertices harder to be leveraged. Since the Cartesian and barycentric coordinates both have pros and cons, we consider both coordinate systems.

Next we explain the format of the planar map data sent from the server to the client. Figures of detailed formats are omitted due to the space limitations. For each video frame, there are three headers: (i) model view matrix, (ii) draw call number, and (iii) draw call size. In particular, the model view matrix *transforms* the world space vertices into model space vertices; the draw call number and size facilitate multi-texture mapping. Moreover, each frame contains a set of *triangles*. With the Cartesian coordinates, each triangle is represented using vertex positions, vertex normals, world space coordinates, and texture coordinates. The vertex positions refer to the 2D positions in the screen space; the vertex numbers and world space coordinates are used for shading; and the texture coordinates are required by adding textures. With the barycentric coordinates, each triangle is represented by a triangle ID (in the quasi-static database), followed by 3 pairs of u, v of individual vertices.

B. Compression Modules

Quantization. Quantization could be classified into uniform and non-uniform quantization. Non-uniform quantization, including *scalar* and *vector* quantization, is the foundation of floating-point number compression. For scalar quantization, we apply Lloyd's algorithm [15] on individual dimensions sequentially. For vector quantization, *all dimensions* are jointly quantized using K-means algorithm [16], in which all inputs are clustered in several groups, and the centroid of each group is determined.

Delta prediction. To leverage the properties that close-by vertices share similar information, a prediction algorithm may use previous coordinates to predict current coordinates [17]. We adopt the delta prediction [18] approach. Only the first input is encoded into a symbol, say as a 32 bit floating number. Others are represented in the deltas compared to the previous input.

Entropy coding. Entropy coding further exploits the different symbol frequency to reduce transmission bandwidth. We choose two widely used entropy codecs: *Huffman* and *arithmetic* coding [19]. The Huffman coding builds a Huffman tree with the more frequent elements at lower levels of the tree. We then assign shorter codes to the symbols closer to the root. For arithmetic coding, it converts the whole symbol sequence into a floating point between 1 and 0. The procedure loops through the symbols and shrinks the interval based on the symbol probability. In addition, we also consider the Lempel Ziv Markov Chain Algorithm (LZMA) and use 7-zip as the implementation. LZMA is a lossless dictionary compressor, which encodes a stream with an adaptive binary range coder. We notice that the coordinates may not be byte-aligned, which may be difficult for the entropy coders to handle. We therefore expand the symbols to the next byte boundary before entropy coding, e.g., 7-bit symbols are padded with one extra highest zero bit. Some sanity checks show that the padding strategy reduces the bitrate with no penalty on video quality.

C. Module Parameter Selection

We record three game scenes in 720p resolution, in which we vary the number of the popular Bunny model, among 1, 2, and 8. In every scene, bunnies with different textures and standing positions are placed on a plane, and the viewing position and orientation changes according to the

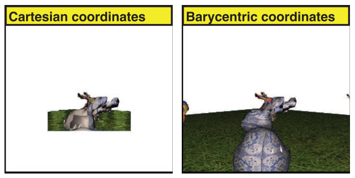
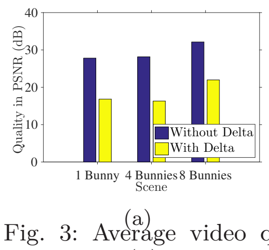
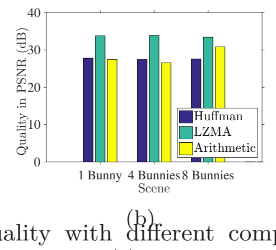


Fig. 2: Sample video frames using different coordinates.





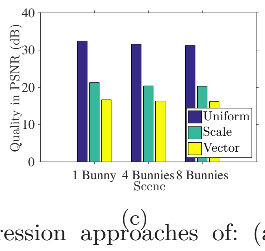


Fig. 3: Average video quality with different compression approaches of: (a) delta prediction, (b) entropy coding, and (c) quantization.

gamer's speed. Each scene lasts for 10 secs at 30 Hz frame rate. We consider the following performance metrics to quantitatively compare module parameters.

- **Video quality.** The rendered quality in PSNR, SSIM, and Perceptual Evaluation of Video Quality (PEVQ). PEVQ is a video quality metric described in ITU-T J.247 Annex B [20], and implemented in Skarseth et al. [21].
- **Compression ratio.** The compressed stream size over the uncompressed stream size.

We have sent the game scenes through a compressor with different quantization, delta prediction, and entropy coding [17] algorithms. We report four most important findings below.

Quantization with Cartesian coordinates causes huge distortion. Fig. 2 presents the same video frame rendered by scale-quantized 16-bit Cartesian and barycentric coordinates. The quantized Cartesian coordinates severely damage the video quality. A closer look indicates that this is because Cartesian coordinates have a very wide range: all real numbers are possible, making them more challenging to compress. On the other hand, all barycentric coordinates are between 0 and 1. On top of that, Cartesian coordinates contain 30-dimension vectors, which dictate more bits to quantize.

Delta prediction negatively impacts compression ratio as well as video quality. We measure the PSNR and compressed stream size with and without delta prediction. Fig. 3(a) plots the achieved average video quality of different scenes with and without the delta prediction at a bitrate of 2.3 Mbps. We find that the delta prediction makes more variance on the resulting symbols, which then leads to lower compression ratio. Moreover, because barycentric coordinates are all positive numbers, applying delta prediction requires one extra sign bit, which worsens the performance.

LZMA outperforms other entropy coding algorithms. We compress each scene multiple times with different entropy coders and plot the average PSNR at the same compression ratio using these entropy coders in Fig. 3(b). It is clear that LZMA outperforms the other two entropy coders.

Uniform quantization outperforms other quantization approaches. On the 2D plane, the vector quantization divides the space into quadrilaterals, and the scale quantization divides the space into variable-size rectangles. In contrast, uniform quantization crops the space into equal size blocks. We plot the average PSNR from the three quantization approaches at the same compression ratio in Fig. 3(c). This figure reveals that the uniform quantization outperforms others by far. We believe this is because the barycentric coordinates are between 0 and 1, and have weak clustering property.

Based on the above findings, we adopt *barycentric coordinates, uniform quantization, and LZMA coder* to compress the 2D planar maps. We note that while we report sample results in PSNR, results in SSIM and PEVQ (not shown due to the space limitations), also support the same findings.

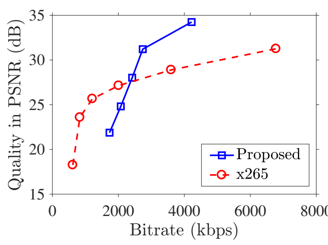
VI. Evaluations

A. Implementation and Setup

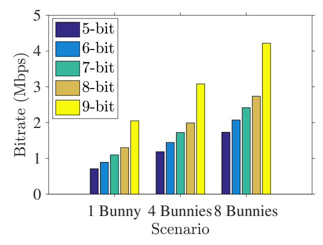
We have implemented the proposed server and client pipelines using a combination of C++ and Matlab. We compare our solution against the current cloud gaming platforms, in which all the rendering tasks are done at cloud servers. The rendered videos are compressed by the state-of-the-art video codec, such as x265 [22], and streamed to the clients. We do not compare against pure graphic streaming platforms, where all the rendering tasks are done at the clients. This is because for the (not-so-thin) clients that have enough horsepower to render game scenes, the benefits of cloud gaming are limited. We expand the game scenes used in the last section. In addition to the number of bunnies, we also consider diverse : (i) model complexity, basic or fine-grained bunnies, and (ii) moving speed, slow or fast. In particular, we adopt 12 game scenes in our experiments. We consider three performance metrics: video quality in PSNR, SSIM, and PEVQ, bitrate in kbps, and component-wise running time in ms. We run the experiments on an i7 3.4 GHz workstation with an NVidia Quadro M4000 card.

B. Results

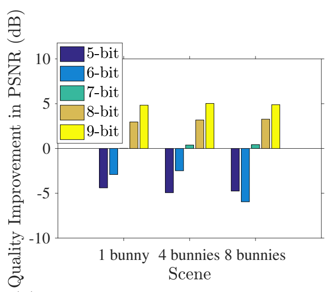
Potential of our proposed solution. We first present the results from the basic bunnies at the low speed. We configure x265 [22] with ultra-fast preset and compress each scene with 6 different Quantization Parameter (QPs) to get its Rate-Distortion (R-D) curve. For our proposed solution, we exercise the tradeoff between bitrate and quality using the bit-depth of the uniform quantization. By varying the bit-depth between 5- and 9-bit, the resulting



(a) curve result from scenario eight bunnies

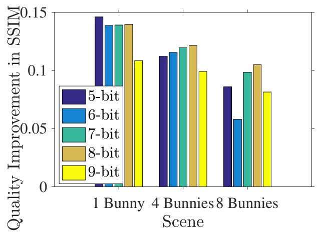


(b) The bitrate of all scenarios under the same compression level

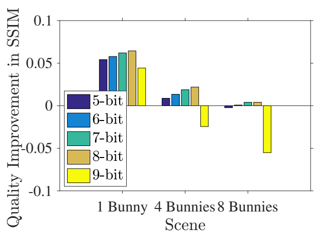


(c) The image quality result compare with x265 encoding in PSNR.

Fig. 4: Performance of our proposed solution in PSNR with basic bunnies: (a) sample R-D curves from slow scene with 8 bunnies, (b) bitrate of our proposed solution at a bit-depth of 8-bits, and (c) quality improvement of our proposed solution.



(a)



(b)

Fig. 5: Performance improvement of our proposed solution in SSIM, with: (a) basic bunnies and (b) fine-grained bunnies.

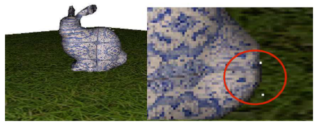


Fig. 6: Zoomed-in image frame of rendered scences.

stream has different bitrates and video quality, which lead to the R-D curves. We first plot sample R-D curves from the scene with 8 bunnies in Fig. 4(a). This figure shows that when bitrate is higher, the proposed solution outperforms x265 in terms of video quality in PSNR. The same observation holds in other scenes, although we cannot present all R-D curves due to the space limitations.

Video quality improvement of our proposed solution. We next compare the video quality in PSNR of our solution against that of x265. Fig. 4(b) presents the bitrate of our solution under different bit-depths. Upon we derive the bitrate of each game scene, we derive the expected video quality of x265 by performing linear interpolation on its R-D curve. Fig. 4(c) plots the quality improvement of our proposed solution over x265. This figure shows that up to 5 dB improvement is possible, and as long as the bit-depth is ≥ 7 bits, our proposed solution results in higher PSNR.

Implications of complex game scenes. Our proposed solution may be more vulnerable to complex game scenes, compared to the existing cloud gaming platforms that stream videos at a fixed resolution. Our game scenes contain either basic bunnies, each with 37,677 vertices on average, or fine-grained bunnies, each with 200,700 vertices. We find that our proposed solution leads to worse PSNR and SSIM when the scenes contain fine-grained bunnies. However, in cloud gaming graphics design, we can use simple 3D models with pre-rendered textures to achieve the same visual result as fine grained models.

Perceptual video quality metrics. Figs. 5(a) and 5(b) plot the video quality improvement of our proposed solution in SSIM with basic bunnies and fine-grained bunnies. The gap is as high as 0.14 in SSIM. However, while using PEVQ as an image quality metrics, x265 scores between 1 to 4 and our solution ranges between 0 to 1.5. The reason that our proposed method performs poorly may because that Human Visual System (HVS) used in PEVQ degraded significantly due to holes in scenes. Fig. 6 is a sample frame from rendered scene, while taking a closer look we may observe small holes between vertice.

Per-component running time. We report the average running time per video frame in Table I. We only consider the computationally intensive components, and with more complex game scenes (8 bunnies). This table shows that the running time of the client side component is much smaller compared to those of the server side components. Although the rendering time is measured on a workstation in our experiments, the negligible values of rendering component (≤ 0.83 ms on average) reveal that porting it to resource-constrained clients is possible.

TABLE I: Running Time (ms), Average/Maximum, 8 Bunnies

	Server			Client
	Detection	Clipping	Occlusion	Rendering
Basic	0.27/0.28	25.56/33.41	1.94/2.68	0.22/0.48
F.G.	3.66/4.73	60.55/88.14	17.43/24.02	0.83/3.13

Implications of diverse speed and resolution. Due to the space limitations, we only report some high-level numbers below. We observe virtually no difference in terms of bitrate of slow and fast game scenes. At a bit-depth of 8, the rate increase due to the speed is merely 0.04% on average. This can be attributed to the fact that we haven't leveraged the temporal redundancy, which is among our future tasks. We also compare the bitrate of our proposed solution and x265 at higher resolutions of 1080p, 2K, and 4K. We encode the game scenes using our proposed solution at 720p and 8-bit bit-depth, and get an average video quality of 31.66 dB in PSNR. We then gradually increase the resolution, and compute the rate increases of achieving 31.66 dB. We find that, for our proposed algorithm, higher resolutions lead to lower bitrates (at the same video quality). More specifically, reductions of 15% (1080p), 13% (2K), and 12% (4K) are observed. If we perform the same analysis on x265, higher resolutions lead to higher bitrate; increases of 18% (1080p), 22% (2K), and 30% (4K) are observed. This shows that our proposed solution has more potential in the future, where ultra-high-resolution displays become popular.

VII. Conclusion

We studied the feasibility of leverage the 2D planar map streaming and distributed rendering for cloud gaming. We first presented the server and client pipelines, based on the standalone 2D planar map rendering pipeline with additions of several components for compressing and transferring 2D planar maps. We then dived into the core challenge of the platform: the design of the compressor/decompresser of 2D planar maps, which has not been studied before. We designed a parameterized compression component, and derived the optimal parameters through real experiments. We then put up the rendering pipelines in our platform, and compared its performance against the state-of-the-art x265 [22]. Our evaluation results are quite promising. Although our platform is outperformed by x265 in PSNR at low bitrate, we significantly outperform it at high bitrates. In addition, our platform always outperforms x265 in terms of perceptual video quality, e.g., by up to 0.14 in SSIM. Other merits of the proposed platform include: (i) fast running time, especially at the client side and (ii) high scalability to ultra-high-resolutions without bitrate penalty.

The current work can be extended in several directions. First, to achieve higher perceptual video quality scores, we may use morphological antialiasing [23] to fill holes in rendered scenes based on precise silhouette information from planar map generation. Secondly, the platform can be extended to support foveated rendering, which renders a part of the region at full quality while others with degraded quality. Last, we can analyze gamers' mobility pattern to leverage temporal redundancy in our planar map streams. These enhancements will further improve the performance of our platform.

References

[1] P. Ross, "Cloud computing's killer app: gaming," *IEEE Spectrum*, vol. 46, no. 3, p. 14, 2009.

[2] "PlayStation Now web page," January 2015, http://www.playstation.com/en-us/explore/playstationnow/.

[3] C. Huang, K. Chen, D. Chen, H. Hsu, and C. Hsu, "GamingAnywhere: the first open source cloud gaming system," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 10, no. 1s, pp. 10:1–10:25, January 2014.

[4] W. Cai, R. Shea, C. Huang, K. Chen, J. Liu, V. Leung, and C. Hsu, "The future of cloud gaming," *Proceedings of the IEEE*, vol. 104, no. 4, pp. 687–691, 2016.

[5] A. Ellis, W. Hunt, and J. Hart, "Svgpu: real time 3d rendering to vector graphics formats," in *Proc. of High Performance Graphics (HPG'16)*, 2016, pp. 13–21.

[6] P. Baudelaire and M. Gangnet, "Planar maps: an interaction paradigm for graphic design," in *Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI'89)*, 1989, pp. 313–318.

[7] P. Asente, M. Schuster, and T. Pettit, "Dynamic planar map illustration," *ACM Transactions on Graphics*, vol. 26, no. 3, p. 30, 2007.

[8] M. Hemmati, A. Javadtalab, A. Shirehjini, S. Shirmohammadi, and T. Arici, "Game as video: bit rate reduction through adaptive object encoding," in *Proc. of ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'13)*, 2013, pp. 7–12.

[9] A. Jurgelionis, P. Fechteler, P. Eisert, F. Bellotti, H. David, J. Laulajainen, R. Carmichael, V. Poulopoulos, A. L. P. Perälä, A. Glora, and C. Bouras, "Platform for distributed 3d gaming," *International Journal of Computer Games Technology*, vol. 2009, pp. 1–15, 2009.

[10] W. Cai, R. Shea, C. Huang, K. Chen, J. Liu, V. Leung, and C. Hsu, "A survey on cloud gaming: future of computer games," *IEEE Access*, vol. 4, pp. 7605–7620, 2016.

[11] D. Meiländer, F. Glinka, S. Gorlatch, L. Lin, W. Zhang, and X. Liao, "Bringing mobile online games to clouds," in *Proc. of IEEE Computer Communications Workshops (INFOCOM WKSHPS'14)*, 2014, pp. 340–345.

[12] X. Nan, X. Guo, Y. Lu, Y. He, L. Guan, S. Li, and B. Guo, "A novel cloud gaming framework using joint video and graphics streaming," in *Proc. of IEEE International Conference on Multimedia and Expo (ICME'14)*, 2014, pp. 1–6.

[13] S. Chuah, N. Cheung, and C. Yuen, "Layered coding for mobile cloud gaming using scalable blinn-phong lighting," *IEEE Transactions on Image Processing*, vol. 25, no. 7, pp. 3112–3125, 2016.

[14] G. Bernstein and D. Fussell, "Fast, exact, linear booleans," *Computer Graphics Journal*, vol. 28, no. 5, pp. 1269–1278, 2009.

[15] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[16] D. Arthur and S.Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proc. of the ACM-SIAM Symposium on Discrete algorithms (SODA'07)*, 2007, pp. 1027–1035.

[17] J. Peng, C. Kim, and C. Kuo, "Technologies for 3d mesh compression: a survey," *Journal of Visual Communication and Image Representation*, vol. 16, no. 6, pp. 688–733, 2005.

[18] M. Deering, "Geometry compression," in *Proc. of Conference on Computer Graphics and Tnteractive Techniques (SIGGRAPH'95)*, 1995, pp. 13–20.

[19] T. Cover and J. Thomas, *Elements of information theory.* John Wiley & Sons, 2012.

[20] "Objective perceptual multimedia video quality measurement in the presence of a full reference," ITU Telecommunication Standardization Sector, Standard, 2008.

[21] K. Skarseth, H. Bjørlo, P. Halvorsen, M. Riegler, and C.Griwodz, "OpenVQ: a video quality assessment toolkit," in *Proc. of ACM International Conference on Multimedia (MM'16), OSSC paper*, 2016, pp. 1197–1200.

[22] February 2017, http://x265.org.

[23] A. Reshetov, "Morphological antialiasing," in *Proc. of Conference on High Performance Graphics (HPG'09).* ACM, 2009, pp. 109–116.