Method for Efficient CPU-GPU Streaming for Walkthrough of Full Motion Lightfield Video

Floyd M. Chitalu
University of Edinburgh
floyd.m.chitalu@ed.ac.uk

Babis Koniaris
Disney Research
ckoniaris@disneyresearch.com

Kenny Mitchell
Disney Research
Edinburgh Napier University
kenny.mitchell@disneyresearch.com

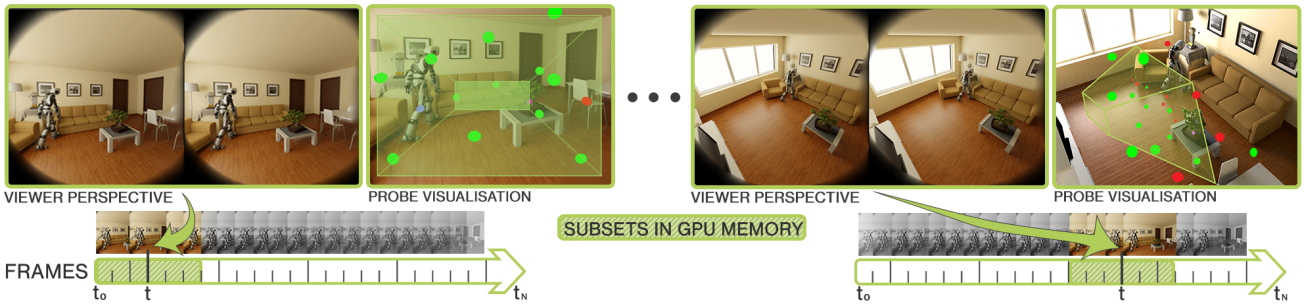


Figure 1: Real-time streaming of encoded subsets of colour and depth streams belonging to a dynamic set of probes (green) in the vicinity of the viewer, which are used for intermediate-view reconstruction in a lightfield rendering pipeline. Our method uses *clairvoyant inference* to determine the spatial-temporal subsets to upload in time for view-dependent decompression, intermediate-view reconstruction, and rendering.

ABSTRACT

Lightfield video, as a high-dimensional function, is very demanding in terms of storage. As such, lightfield video data, even in a compressed form, do not typically fit in GPU or main memory unless the capture area, resolution or duration is sufficiently small. Additionally, latency minimization–critical for viewer comfort in use-cases such as virtual reality–places further constraints in many compression schemes. In this paper, we propose a scalable method for streaming lightfield video, parameterized on viewer location and time, that efficiently handles RAM-to-GPU memory transfers of lightfield video in a compressed form, utilizing the GPU architecture for reduction of latency. We demonstrate the effectiveness of our method in a variety of compressed animated lightfield datasets.

CCS CONCEPTS

•**Computing methodologies** → **Virtual reality;** Graphics processors;

KEYWORDS

GPU, Streaming, Animated Lightfields, VR, Codec, Real-time

CVMP 2017, London, United Kingdom

978-1-4503-5329-8/17/12…$15.00
DOI: 10.1145/3150165.3150173

ACM Reference format:
Floyd M. Chitalu, Babis Koniaris, and Kenny Mitchell. 2017. Method for Efficient CPU-GPU Streaming for Walkthrough of Full Motion Lightfield Video. In *Proceedings of 14th European Conference on Visual Media Production (CVMP 2017), London, United Kingdom, December 11–13, 2017 (CVMP 2017),* 10 pages.
DOI: 10.1145/3150165.3150173

1 INTRODUCTION

Continuous or frequent movement of data streams between devices or even memory on the same device is a core feature of many real-time applications, such as video processing and 3D graphics rendering. An effective streaming method improves the scalability of an application by operating only on a subset of the data that gets continuously updated, where the update process incurs minimal overhead in memory, computations and latency, and is primarily limited by the bandwidth of the memory bus versus the required throughput.

The recent resurgence of Virtual Reality (VR) technologies has brought forth demand for high fidelity and immersive content for viewers, while still maintaining real-time performance. One such type of content is *lightfields* [3, 11]: datasets suitable for image-based rendering (IBR), that allow photo-realistic image quality while allowing six degrees of freedom. Such datasets, when animated, can typically be too large to fit in most storage media, and streaming becomes a necessity.

A particular framework of interest is IRiDiuM [7, 8], which is an end-to-end system bridging the gap between high quality rendering and the immersion factor provided by free head and body motion

in VR. IRiDiuM transforms off-line rendered movie content into a sparse, compressed lightfield representation that can be efficiently decoded in real-time, allowing six degrees of freedom exploration of the rendered content. In this framework, the video is rendered from a number of 360° viewpoints (probes), generating colour and depths streams that are compressed separately. The method trades off higher compression ratios for high decoding speed and probe data independence, so that the probe subset that is used for decoding and rendering can be dynamically adjusted based on conditions such as user location and orientation. The decoded data are then used for IBR for each eye using ray-casting. Individual probe datasets can be specified with different lengths, and potentially starting at different times relative to a common time-frame of reference.

IRiDiuM lightfield video datasets, even when compressed, typically exceed the storage capacity of GPUs, and therefore have to be streamed from RAM. Streaming performance for lightfield video for use with VR is paramount, as latency of view-dependent video leads to user discomfort and *simulator sickness* [1]. Due to the independence of the per-probe videos and their spatial nature, a streaming solution would ideally consider the locality of the user in addition to the current time as the main factors for selecting the streamable dataset. Due to the asynchronous nature of graphics and rendering APIs, the streaming solution needs to operate asynchronously to reduce stalls in order to achieve minimum latency of the PCI-e bus between the CPU and GPU.

We present a modular solution for streaming large immersive multiview video content to support an arbitrary number of probes for reconstructing 3D scenes accurately from any point of view. Our proposed method takes into consideration viewer locality in addition to time, as the IRiDiuM system supports an arbitrary number of probes. So, the streamer should work on streamed probe data that may not be known in advance. We ensure support for arbitrary video lengths by effectively caching only temporal and spatial subsets of per-probe datasets ahead of time on the GPU. To do this, we develop a novel scheme to identify and upload a *clairvoyance window*: a span of frames corresponding to a set of probes that are near a viewer at run-time, similar in concept to potentially visible sets (PVS) but in the context of animated view dependent lightfields. In our results, we demonstrate the effectiveness of our method in a variety of compressed animated lightfield datasets, with each having unique characteristics that affect performance and storage requirements.

Contributions. The contributions form a modular streaming solution, from using time and the viewer location to infer the subsets uploaded, to real-time asynchronous data transfers to minimise the cost of frequent and regular memory updates:

- A real-time streaming method for high-bandwidth full-motion lightfield video that supports a wide range of colour and depth compression formats, organised as modular streams for a dynamic set of probes that are used for intermediate-view reconstruction.
- A spatial-temporal scheme to efficiently determine the subsets of per probe datasets that are to be on the GPU ahead of time parametrised by viewer locality and a common time-frame of reference.
- We show that the method is a tractable solution for high-bandwidth lightfield renderers that bridge the gap between cinematic quality graphics and real-time interactivity, since it can accommodate lightfield solutions integrated with raw or block-compressed data of any spatial pixel format while still allowing for runtime optimizations such as view-dependent decompression to exploit precomputed visibility.

2 RELATED WORK

Recent advances in video coding standards have lead to a flourishing of a multitude of video coding technologies that extend upon conventional HD video. Many of these are 3D video solutions are categorized by the representation format, such as stereo and multi-view and those additionally based on depth image based rendering (DIBR) [13]. They are a direct result of the stereo and multi-view video coding extensions of the H.264/AVC [18], and the high efficiency video coding (HEVC) [15] standards. In multiview video plus depth (MVD) each frame of a captured view includes an associated depth map which is used for intermediate view synthesis.

However, the immersive quality improvements have also come at a cost of larger video datasets. In response, various works have proposed solutions to minimise the associated memory footprint. Of these, Müller et al. [2013] propose an extension to HEVC for application instances that involve video and depth information with a 3D video coding framework for depth-enhanced multi-view formats to overcome challenges in network streaming. Similarly, Hirota et al. [2016] propose an Ultra-HD multi-view video distribution system in order to realize high quality play-back and rapid view switch. The ReMA framework presented by Lee et al. [2017] is another solution for the challenge of streaming multi-view content. They propose an architecture for transmitter, receiver, and a distribution system to broadcast and generate 3D videos for mobile devices. While these solutions demonstrate the nature of the datasets of concern, they are aimed towards network distributed streaming. We propose a solution targeted at a system level by bridging the inherently heterogeneous execution model to move data between the CPU and GPU.

Bridging the gap between multi-view video coding extensions and 3D computer graphics has been another interesting challenge involving efficient coding schemes that utilize depth maps or polygon meshes [12]. While there is only a few available end-to-end systems, notable progress has been made. Collet et al. [2015] propose transmission of an animated texture mesh as an alternative to pure image data, encoded as an MPEG video stream, to achieve an end-to-end pipeline for free-viewpoint video. The encoding is efficient and delivers good results, but the captured space is limited and static, and the resulting meshes are pre-lit without view-dependent information, limiting the type of datasets that can be used. Koniaris et al. [2017] propose the IRiDiuM framework: an end-to-end solution using a modular lightfield video format with support for view-dependent decoding for real-time rendering with six degrees of freedom. The datasets used are small enough to fit in the GPU, limiting them in the number of probes that can be used, and video length for each.

To date, there is no streaming solution for multi-view video whereby both time and viewer locality are considered to infer which datasets must be streamed to the GPU at a given time, allowing unlimited freedom in either of these axes (time and 3D location).

Most modern GPUs support concurrent inter-device memory transfers between the CPU and GPU alongside the massively parallel execution. As a result, a number of works have researched the performance enhancements that can benefit applications by efficiently hiding the cost of data transfer with asynchronous copies. Sunitha et al [2017] investigate this problem from a general purpose GPU (GPGPU) computing perspective and experiment using the CUDA framework with different levels of concurrency supported by CUDA streams (GPU work queues). Similarly, Huang et al. [2012] propose to reduce transmission overhead by sharing computation between the CPU and GPU. The scheme tackles the common scenario wherein the CPU thread explicitly stalls until the GPU has completed work before it can proceed. Instead of sending all the data to the GPU, a subset is processed on the CPU, thereby leading to a more efficient utilization of all the available processing resources and a reduction in the data transfer overhead.

In general, the specific approaches for overlapping data transfer are dependent on the capability of the hardware because overlapping computation with communication requires fine-grained control [17]. The approach of explicit memory copies is the most common and can be performed either synchronously or asynchronously with regards to the CPU. Mapped memory approaches on the other hand do not rely on explicit copies, but map part of the GPU memory into CPU address space (or vice versa) using *pinned memory*. The potential limitation is its dependence on the memory access patterns to the mapped region by the GPU since they could effectively involve a transfer operation over the PCI-e bus. Finally, GPUs with support for full-duplex PCI-e transfers allow for a threaded approach to data transfers, which is based on having two copy engines on the GPU for concurrent to-and-from communication between the CPU and GPU. *Dual copy engine* and CUDA streams by NVidia are exemplars of this approach, which is also available via modern graphics APIs [5], and is commonly used in video processing.

3 OVERVIEW

We represent streaming as a two-step process of identifying the subsets of encoded 360° frames of a set of probes in the vicinity of the viewer, that will be used for intermediate-view reconstruction ahead of time, and uploading them to the GPU. During the identification step, we use runtime information such as viewer locality and the current video time-frame of reference to infer the spatio-temporal subsets of the active probe datasets to upstream to the GPU. We enqueue streaming operations in batches defined at a granularity of probes to ensure seamless uploading of subsets that lay within a specific frame-span while rendering. Each probe dataset contains the 360° cubemap frames of colour and depth streams which are compressed in unique and separate modular formats and are optimized for pure decoding purposes. Any encoding format for colour and depth images in a 360° format (either equi-rectangular or cubemap) can be incorporated since we do not impose any constraints. For example, the colour information of a frame can be uploaded to the GPU as a collection of key-frames that lay in a span

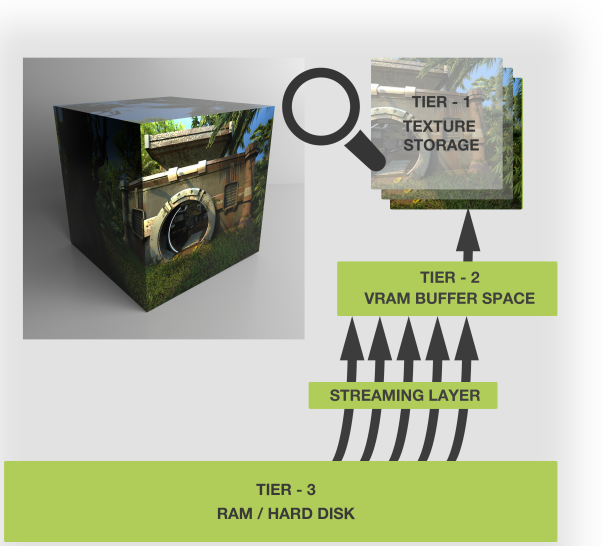


Figure 2: A representation of the hierarchical data storage scheme inherent in our approach, where we stream and cache the spatial-temporal subsets of probe datasets on the GPU. The streamer moves the data belonging to a PVS, that is parameterised by time and space, from a lower Tier-3 memory region to Tier-2 memory that is resident on the GPU.

of frames at the granularity of colour compression blocks. Depth compression can likewise assume any format including difference rectangles in the form of axis-aligned bounding boxes [8] or intra-coding using geometric primitives [12]. Our data layout scheme in memory assumes a 3-tier memory hierarchy where we maintain three representations of the video data in memory which includes GPU VRAM, CPU RAM, or even Hard-disk and cloud sources.

4 FRAMEWORK

In this section, we describe our approach to stream high-bandwidth lightfield video by ensuring that only a spatial-temporal subset of encoded 360° cubemap frames are on the GPU ahead of time. subsection 4.1 first describes our hierarchical memory abstraction for temporarily storing data on the GPU based on prioritisation, which is determined by the lightfield rendering runtime. subsection 4.2 then describes how we identify the subsets of per-probe datasets that need to be on the GPU. We then describe the process of streaming the identified subsets to the GPU in subsection 4.3.

4.1 3-Tier Storage

A hierarchical ordering of memory into three tiers is used for storing probe datasets based on temporal and spatial prioritization: compressed data in *Tier-3* RAM, compressed data in *Tier-2* GPU buffer memory and unpacked cubemaps in *Tier-1* GPU texture memory. The compressed video (colour and depth data) is first copied from RAM to GPU memory at application runtime. Texture

decompression is then performed by the runtime using separate GPU-to-GPU operations with very little CPU-to-GPU communication that amounts to scheduling GPU-GPU *Tier-2* → *Tier-1* memory copies.

Figure 2 provides an illustrative example of this ordering which is analogous to a three-level cache system for storing datasets based on temporal and spatial locality with the respect to viewer motion. For a given scene dataset (or set of datasets), we assume that all encoded frames reside by default in Tier-3 memory, which represents any system-accessible memory outside of of the GPU, including hard-disk or even cloud sources. At any given time, a subset of probe datasets identified using clairvoyance inference are stored in Tier-2 memory. Tier-1 memory is the set of cubemap textures (two colour and one depth per probe) that are necessary for reconstruction of a single frame. The streaming layer concerns itself with identifying and transferring the subsets of compressed from Tier-3 to Tier-2 memory in time for view reconstruction.

4.2 Clairvoyance Inference

We now describe our scheme to identify the subsets of our datasets that need to be in Tier-2 memory ahead of time for dependent rendering operations.

Seamless streaming of per-probe temporal datasets requires that the next frames are on the GPU in time for dependent rendering operations. This avoids potential stalls in the rendering pipeline due to unfinished memory transfers which can lead to video stutter and an overall degraded experience. To determine our temporal subsets needed for future rendering operations, we compute a *clairvoyance window*, C, representing a temporal span of frames, F, for the set probes that are used for view reconstruction, $\mathcal{P}$. The clairvoyance window defines the subset of encoded probe frames that will reside on the GPU for a given time-range and dataset.

We compute the clairvoyance window of each probe, C_{ρ_i}, to ensure that its correct subset is in GPU buffer memory ahead of time for rendering operations (see Figure 1): Given a probe $\rho_i \in \mathcal{P}$ of a particular dataset, we compute C_{ρ_i} using the common time-frame of reference ϕ, $1 \leq \phi \leq \Phi$, where Φ denotes the time-frames of the video sequence of the respective scene dataset. The variable ϕ is analogous the frame-reference for the next reconstructed view-frame that is displayed to the viewer. We estimate the start of C_{ρ_i} over the course of Φ given ϕ as $c_\phi = \phi - (\phi \bmod F) + 1$. The last frame in the span of C_{ρ_i} is computed as $c_\Delta = c_\phi + \min\left(F,\ \Phi - c_\phi\right)$ which we compute if the frames spanning C_{ρ_i} have elapsed. From this, the collective subset of frames in C for the set of probes whose data is streamed to the GPU at a given instance is defined by

$$C = C_{\rho_1} + C_{\rho_2} + \cdots + C_{\rho_N}$$

where N is the number of streamed probes that are potentially used for reconstruction, and $C_{\rho_i} \equiv \int_{c_\phi}^{c_\Delta} \Phi$. This representation for C is essential for supporting animated lightfield video features such as dynamic branching, as it permits us to decouple the process of streaming multiple datasets of a given collection whereby each dataset has its own list of probes.

It follows that the frequency of streaming for a given dataset is directly dependent on

$$\lambda = \begin{cases} F & \Phi > F \\ \Phi & \Phi \leq F \end{cases}$$

since the frames spanning C_{ρ_i} elapse specifically when $c_\phi = \phi$. This ability to control the frequency of data transfers via λ has an additional benefit with regard to performance tuning, as it ensures our ability to control the sizes of data transfer by tuning for the optimal value of F subject to storage constraints. An example use-case is in scenarios where a configuration optimized for high-end desktop platforms may not be suitable for lower-powered mobile platforms. Furthermore, λ can be chosen based on the compression rates of a given dataset to control the sizes of subsets being streamed to the GPU. This follows the fact that datasets with little animation will have higher compression ratios than those with more animation.

4.3 Streaming

We provide seamless integration of the process of identifying the subsets that lay within the span of C (as described in subsection 4.2) and performing asynchronous memory transfer operations to the GPU. A multi-buffered approach is used to prevent stalls in the pipeline that are caused by enforced waits for copy operations to complete before subsequent rendering operations can proceed. Our Tier-2 data storage scheme encompasses the use of cyclic buffering to ensure that we have asynchronous Tier-3 → Tier-2 transfers. When the frames spanning C have elapsed, we switch between multiple GPU memory instances using round-robin scheduling while ensuring that any previous operations that are dependent on a new *write-to* instance have finished.

An important constraint we must satisfy is probe data dependency in order to permit changes to $\mathcal{P}$ to give $\mathcal{P}'$ i.e. the updated set of probes that are used for reconstruction. These dependencies are determined by runtime probe selection heuristics of the rendering system. Such data dependencies must be resolved due to changes in probe-to-viewer locality, which could depend on viewer position and orientation in the virtual space and time. Since this may happen at any frame, we immediately transfer the complete new subsets of C if $\mathcal{P}' \neq \mathcal{P}$. (Note that in the case that the changing of probes occurs when $c_\phi = \phi$, streaming occurs normally for all $\rho_i \in \mathcal{P}'$).

5 ARCHITECTURE & IMPLEMENTATION

Our model for streaming is fundamentally based on the method of asynchronous buffer transfers [5], which is widely applied in cases for frequently uploading data to the GPU. This approach readily unifies the compression formats of full motion lightfield video (FMLV) with the readily available Direct Memory Access (DMA) capabilities of modern GPU architectures to allow asynchronous transfer operations between the CPU and GPU.

Since the streaming framework interacts with a FMLV runtime solution via a dynamic set of probes and a common time-frame of reference, we further extend the concept of asynchronous buffer transfers. For instance, encoded frames are uploaded at a frequency defined by the clairvoyance window as well as probe changes. Therefore, the key elements of our streaming framework address the performance and scalability impact of identifying and uploading the encoded subset to GPU buffer space for decompression and rendering.

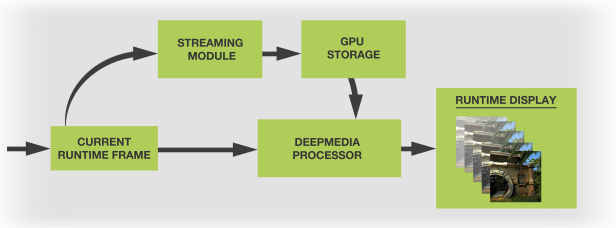


Figure 3: Schematic outline of streaming integration: The streamer is incorporated as a modular component into the lightfield rendering pipeline (IRiDiuM Deep Media processor), which then uses time, such as the current frame, and the current set of probe to ensure that the subset of encoded frames of each probe are on the GPU prior to any dependent rendering operations.

5.1 Schematic Outline

Streaming is represented as a modular runtime component that ensures the correct data subsets are in GPU buffer memory *ahead* of time. The main input is a set of probes which are determined by the FMLV runtime, and their state with respect to the clairvoyance window time span in order to infer the subsets to the streamed. In practice, we represent each probe as a simple lightweight attribute structure that defines the locations of data storage in Tier-3 and Tier-2 memory. These attributes also include synchronisation primitives e.g. `GLsync` objects, that the FMLV runtime may use to enqueue dependencies for subsequent decompression and rendering tasks.

Figure 3 provides an example outline of the streaming framework and how it integrates with the IRiDiuM pipeline. Given a target frame that is to be displayed next, the streaming module first ensures that the corresponding encoded subsets (colour and depth frames) are in GPU buffer space by loading the clairvoyance window of each probe in the active set if they are not loaded. The IRiDiuM runtime proceeds to perform decoding and view reconstruction, while ensuring that all necessary data for the next frame to be displayed is on the GPU by enqueueing necessary dependencies for data availability on the GPU. The result of the pipeline is the fully reconstructed frame that is seen by the viewer.

5.2 Uploading Datasets

Encoded subsets are transferred at periodic frame intervals determined by the size of the clairvoyance window, or when probe changes have occurred. First, we infer the range of frames spanning the current clairvoyance window span by computing c_ϕ and c_Δ. We then determine and copy the encoded frames C_{ρ_i} by scanning iteratively through the encoded colour and depth data of each frame of each probe. The streaming processor is agnostic to the organisation of the compressed data, and only requires an interface to access all data necessary to decode a given range of frames. We simultaneously copy data to the GPU while iterating through colour and depth data by using pinned-memory transfers and provide synchronisation primitives that the FMLV runtime can use to enqueue dependencies for the uploaded subsets of a particular probe.

5.3 Dependency Management of Probe Datasets

The active set of probes whose subsets are streamed to the GPU is directly dependent on the probe selection heuristics of the FMLV runtime, as they determine when new probes are required to reconstruct a novel viewer perspective subject to viewer location. We represent probe *eviction* and *replacement* for streaming as a process of identifying the probes which are are no longer part of the runtime set and the new subset to replace them. At each frame, we check for any discrepancies between the active set determined by the selection heuristics in the FMLV runtime and the representation held by the streamer.

The selection heuristics of the FMLV runtime can likewise be extended to the case of multiple datasets in order to support non-linear video playback which can be triggered on an arbitrary basis. In this case, streaming is performed at normal regular frame intervals for a given active dataset of the collection. We ensure to upload the first clairvoyance windows of a subset of probes from the inactive datasets to minimise the overhead that can be caused by switching between datasets. The set of 'default' probes that are preloaded for the inactive datasets is determined by the FMLV runtime.

6 EVALUATION

Our results are obtained on a system with an Intel® Core™ i7-5930K CPU @ 3.50GHz with 32GB RAM and an NVIDIA GeForce GTX 970 GPU with 4GB of VRAM. We test our solution using the IRiDiuM system [8] implemented using the OpenGL API. We use the *Robot*, *Sponza*, and *Swamp* datasets for evaluation which are created using Pixar RenderMan (see Figure 4). We use three datasets to show the flexibility of our method, each of which have characteristics that uniquely affect performance. The *Robot* and *Sponza* datasets demonstrate environments that are focused on a particular area. Viewer motion is generally constrained to a small volume. The Robot dataset has a set of 16 densely-placed probes per-unit area with 78 frames each. Consequently, the changing of probes due to runtime prioritization is likely to induce a lot of streaming operations subject to user motion. The Sponza dataset, while having just 9 probes, is the longest video sequence of all datasets with each probe having 600 frames. The dataset represents a large explorable space with a sparse set of probes with larger storage requirements as a result of having more animations than the Robot dataset. In addition, the captured frames are highly dynamic, therefore colour and depth frame compression schemes are less effective. The third dataset we use for demonstration is the *Swamp* which encodes *dynamic video branching* to support non-linear video playback [9]. The non-linearity of the video is represented as a collection of a number of sub-datasets of (potentially) related video sequences that may be switched arbitrary based on runtime conditions using a state machine. Each sub-dataset may contain a single or a range of encoded 360° frames. The Swamp is also the largest dataset with a size that readily exceeds the capacity of many conventional GPUs (21.8GB).

In each evaluated dataset, the encoded frames are primarily optimized for pure decoding performance due to the large number of pixels that must be decoded in runtime. Each 360° cubemap frame

Figure 4: Dataset snapshots: Robot (left), Swamp (middle), and Sponza (right).

Dataset	Col' Format	Dep' Format	Probes	Frames	Size
Robot	BC1	W16UI	16	78	519MB
Sponza	BC1	BC5	9	600	3.2GB
Swamp	BC1	R16UI	40	555	21.8GB

Table 1: Dataset characteristics: Colour data compression for all datasets is based on the BC1 (block compression) format, with depth compression formats using either 1- or 2-bytes per depth pixel.

(consisting of a colour and depth frame representation) has faces of dimensions 1024 × 1024 pixels. The colour frames have been compressed using the BC1 block-compression scheme with 256 cell blocks per face (64 × 64 pixels per cell block). The compression scheme for depths is not very aggressive since the quality and resolution of depth maps are critical for reconstruction (moreso than colour data). A pre-calculated stream of axis-aligned bounding boxes (AABB) is used for each depth-frame (see Koniaris et al. [2017] for more information). Table 1 provides further details on each of the evaluated datasets.

We evaluate the changing probes at runtime by emulating viewer motion using cubic-bezier interpolation through a set of probe positions. Motion occurs over a fixed period of elapsed frames, which is determined by the number of frames in a dataset multiplied by a number of video sequence repetitions. For each test, we repeat the video sequence at least 4 times to minimise variance in the observed measurements of execution time.

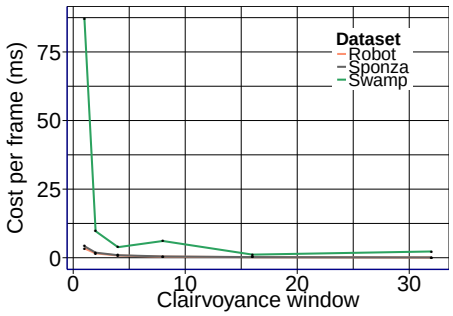
6.1 Streaming Performance

The average streaming performance in all our tests is real-time making it a suitable solution for lightfield rendering pipelines involving large datasets. The bottleneck of our streaming algorithm is the scanning, where individual colour and depth key-frames required to reconstruct each frame in a clairvoyance window must be updated with new references to regions in GPU buffer space. This is a necessary step during view-dependent reconstruction since texture update calls require a source address of respective data in GPU memory. These source addresses are organised at the a granularity of colour compression blocks and difference-rectangles for depth data, as per IRiDiuM design. As such, we reduce the cost of this overhead by controlling the size of clairvoyance window size. Even with this option however, choosing the right clairvoyance window size remains dependent on the dataset used since it is the compression scheme that determines the amount of iterations through the data required to update the necessary memory references. When compression is less effective for frames with a lot of animations, we reduce the clairvoyance window size, which has a side effect of increasing the frequency of CPU-to-GPU transfers (see subsection 4.2). The benefit however is that generally a smaller clairvoyance window size decreases the amount of data transferred per frame interval, and is likely to reduce the cost of changing probes at runtime.

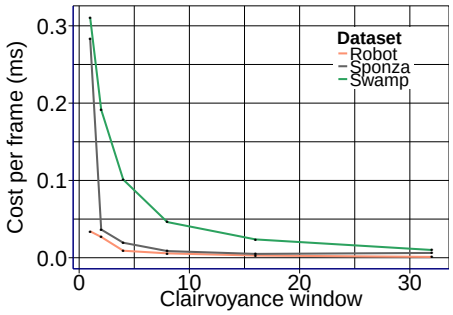
Figures 5a and 5b show the timings for the cost of streaming per-frame as a function of the clairvoyance window size on the three datasets. We measure the cost on the CPU (5a) as well as the time taken to transfer the subsets to the GPU (5b). Clearly, streaming on a per-frame basis does not benefit performance since the rate of memory upload requests is higher that rate of DMA transfers combined with rendering. Therefore, implicit dependencies between GPU memory instances and rendering tasks leads to stalls in the execution pipeline. Moreover, the choice of clairvoyance window size must allow for sufficient time between GPU execution and CPU requests for write-access to potentially-in-use GPU buffer memory. The results shown in Figures 5a and 5b, also reveal that the cost of moving data is generally low compared to the task of scanning the subsets of colour and depth streams to be copied from the CPU to GPU. In general, the CPU cost is dependent on the nature of a given dataset because inherent characteristics such as, the effectiveness of compression due to the level of animation, determines the amount of scanning and data copied for a range of frames. We found that streaming performance is not directly dependent on the number of probes but more so on the amount of data, in terms of colour and depth streams, each probes has.

6.2 Runtime Performance

The potential overhead that can be induced by the logic to stream subsets to the GPU from CPU RAM is largely dependent on the

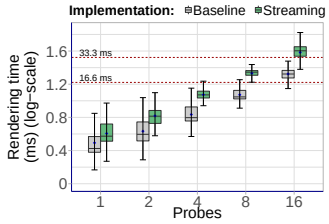


(a) CPU-side scanning.

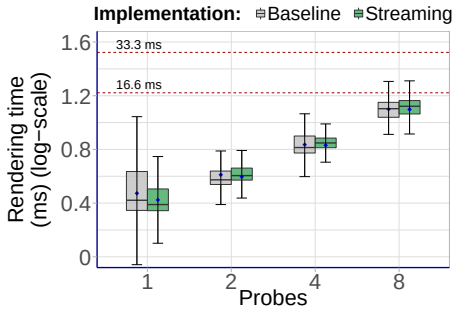


(b) RAM-to-GPU memory transfer.

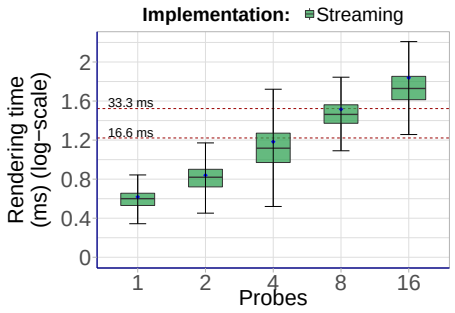
Figure 5: Performance cost of streaming subsets of a one probe relative to the clairvoyance window size.



(a) Robot dataset.



(b) Sponza dataset.



(c) Swamp dataset.

Figure 6: Summary plots of the achieved rendering times (in milliseconds) on the three evaluation datasets

number of probes used for reconstruction and the rate of animation in a dataset. A challenging case therefore is one of finding the optimal combination between the number of probes used for reconstruction and the clairvoyance window size. To evaluate the effect of the number of streamed probes on performance, we test the change in rendering time as a function of the number of probes on each dataset. We likewise, replicate this tests using a pre-existing implementation that attempts to cache all probe datasets on the GPU i.e. without any streaming. We analyse our results by using this as our baseline implementation to understand the potential overhead of streaming and how it affects performance.

Our method is real-time and maintains comparable average frame-rates against the baseline implementation, including for the typical-use-case of 8 probes. Figure 6 shows our performance results on the three datasets, which are compared to the baseline implementation that stores a whole dataset in GPU buffer memory for the duration of the application. On the Robot dataset however (6a), the performance at 8 probes falls below the 16.6ms threshold and achieves 22.03ms per frame, compared to the baseline implementation at approximately 12.7ms. Given that the Robot dataset is only 519MB, it is likely that this performance slow-down is caused by the changing of probes at runtime which incurs the overhead of replacing regions in GPU memory required to decompress and render immediate frames that follow. In contrast, the results of Figure 6b for the Sponza dataset reveals average frame-rates which are actually greater than the baseline performance with a lower dispersion in recorded frame times. On average, Streaming incurs a minor slow-down of approximately 3.2% compared to the baseline implementation. A possible explanation could be that storing a smaller amount of data of the GPU may lead to the advantage of reduced access latencies during view-decompression since only a (smaller) subset of encoded colour and depth streams reside on the GPU at a given time, unlike the case of storing a whole dataset in GPU VRAM. The results of Figure 6c for the Swamp dataset demonstrate the capability of our approach to handle scenarios in which a dataset exceeds the capacity of a GPU and where the dataset is highly dynamic with animations. The high levels of animation in the dataset combined with frequent changes in probes results a real-time average that just manages close to 33.3ms per frame at 8 probes and just lower than 16.6ms at 4 probes. The transmission latency combined with the changing of probes at runtime dominates performance. These results show the sensitivity of our

approach to datasets that are large due to the level of depth and colour compression. The swamp scene dataset has many animations with a relatively long video sequence. Given the results of Figures 6a and 6b, we expect the performance of a theoretical baseline implementation to have similar trends but with some differences in performance.

We briefly note that the quality of image reconstruction is also dependent on the number of cameras in the animated scene. However, our streaming solution primarily affects rendering performance more so than quality as measured via objective quality measures. Koniaris et al. [2017] have previously evaluated the effects of the number of probes on quality using peak-signal-to-noise ratio (PSNR) as well as the structural similarity index measure (SSIM) (see Figure 7 in referenced work). They have evaluated quality using a variable number of active cameras, selected via the angle-based prioritisation heuristic to conclude that it is the number and placement of cameras that affect rendering quality. Although their evaluations have used a different dataset, the reached conclusions would likely be the same on the datasets we use in our evaluations.

6.3 Implementation Analysis

Asynchronous Streaming. Single buffered pinned-memory transfers are now a common use case in modern applications. While this is convenient for having persistently mapped memory between CPU and GPU, such a solution alone for streaming has a major disadvantage: The overhead of synchronising buffer write operations by the CPU is inherently dependent on the rate at which the GPU can finish using the subsets currently contained in the buffer. In our particular case, such an approach would result in potential stalls in the rendering pipeline as an explicit wait operation has to occur before, for example, the contents of a previous clairvoyance window are used to reconstruct a preceeding number of frames submitted for rendering by the CPU. GPUs generally operate asynchronously to the CPU unless explicit synchronisation is enforced, which is not ideal for streamlined performance. It remains a limitation however, that the subsets of new probes as a result of changes due to selection heuristics must be uploaded to the GPU immediately which limits the scalability. The results of Figure 6c demonstrate this, as the rendering time increases above the minimum ideal threshold of 33.3ms, which is due to the increased size of the data streams and the number of probes in the dataset.

Compression Schemes. It has been discussed that the performance of streaming is inherently dependent on the data layout of the individual probe key-frames of a dataset. There is no globally ideal approach to traversing the representative data structures; this is dependent on the adopted compression scheme and runtime implementation. Since our implementation was based on the IRiDiuM system, we assume that a given runtime provides a standard representation for the encoded frames which we then traverse in order to update necessary memory references for the encoded data streamed to GPU buffer memory. As such, further optimizations like optimizing for cache misses when iterating through the encoded colour and depth frames etc. remain dependent on the host FMLV system's implementation choices.

7 CONCLUSION

We have presented a method for enabling efficient streaming of high-bandwidth lightfield video in compressed form to overcome memory capacity constraints faced by GPUs. The method is parametrised by viewer location in addition to time in order to handle RAM-to-GPU memory transfers ahead of time for dependent operations using our clairvoyant inference scheme. Our three-tier memory representation based on asynchronous CPU-to-GPU memory transfers provides a light-weight model to minimise the potential overhead of synchronisation. It is based on cyclic buffer storage on the GPU to enable asynchronous uploads for minimum transfer overheads thereby minimising stalls that may otherwise incur a significant cost on the rendering time budget. The method is general and can accommodate encoded lightfield video data layouts of any type since we do not constrain the approach to a particular compression scheme. It is simple and interacts well with probe-selection heuristics that may be used by a given FMLV system. The end result is a modular component for FMLV rendering pipelines that ensures that encoded frames of a dynamic set of probes are on the GPU in time for view-dependent decoding, reconstruction, and rendering operations.

7.1 Limitation and Future Work

While our method can minimise runtime overhead to achieve real-time frame rates, it remains susceptible to incurring performance cost due to a number of factors, such as the amount of data transferred at a given instance or the frequency of data transfer subject to the amount transferred. In our current solution, we upload several probe datasets at fixed time intervals defined by the clairvoyance window, which has the disadvantage that the load of copying a range of encoded frames is experienced in one frame rather than being distributed in time across different frames. With highly animated datasets, or a large clairvoyance window, we sometimes observe artefacts in the form of stuttering frames which can lead to simulator sickness. In future work, we plan to incorporate support for *distributed time spans*, where the clairvoyance window may be specified per probe to minimise the transmission latency and the cost of CPU-side scanning that is associated with large datasets. We would also like to extend our method to consider the case for view-dependent streaming since we currently makes no assumption about viewer direction, which could be beneficial for prioritizing streaming based on viewer orientation and direction. Another direction for further work is CPU and GPU bandwidth pre-computation which can be used to infer a predetermined timing of uploads according to temporal data since it is known beforehand.

ACKNOWLEDGMENTS

This work is funded by InnovateUK project Grant No. 102684 and is supported by grant EP/L01503X/1 for the Centre for Doctoral Training in Pervasive Parallelism (http://pervasiveparallelism.inf.ed.ac.uk/) from the UK Engineering and Physical Sciences Research Council. We share a special thanks Sophie North, Desislava Markova and Fraser Rothnie for their artwork, and thank Maggie Kosek and David Sinclair for their assistance in asset production. We also thank Abertay University artists' contributions (http://bit.ly/IRIDiuM)

REFERENCES

[1] Timothy J. Buker, Dennis A. Vincenzi, and John E. Deaton. 2012. The Effect of Apparent Latency on Simulator Sickness While Using a See-Through Helmet-Mounted Display. *Human Factors* 54, 2 (2012), 235–249. https://doi.org/10.1177/0018720811428734 arXiv:http://dx.doi.org/10.1177/0018720811428734 PMID: 22624290.

[2] Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. 2015. High-quality Streamable Free-viewpoint Video. *ACM Trans. Graph.* 34, 4, Article 69 (July 2015), 13 pages. https://doi.org/10.1145/2766945

[3] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. 1996. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 43–54.

[4] Y. Hirota, K. Takahashi, H. Tode, and K. Murakami. 2016. P2P-based Ultra High Definition multi-view video distribution system with best-effort and bandwidth guaranteed networks. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 774–775. https://doi.org/10.1109/CCNC.2016.7444876

[5] Ladislav Hrabcak and Arnaud Masserann. 2012. Asynchronous Buffer Transfers. In *OpenGL Insights*, Patrick Cozzi and Christophe Riccio (Eds.). CRC Press, 391–414.

[6] W. Huang, L. Yu, M. Ye, T. Chen, and T. Hu. 2012. A CPU-GPGPU Scheduler Based on Data Transmission Bandwidth of Workload. In *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*. 610–613. https://doi.org/10.1109/PDCAT.2012.15

[7] Babis Koniaris, Ivan Huerta, Maggie Kosek, Karen Darragh, Charles Malleson, Joanna Jamrozy, Nick Swafford, Jose Guitian, Bochang Moon, Ali Israr, and Kenny Mitchell. 2016. IRIDiuM: Immersive Rendered Interactive Deep Media. In *ACM SIGGRAPH 2016 VR Village (SIGGRAPH '16)*. ACM, New York, NY, USA, Article 11, 2 pages. https://doi.org/10.1145/2929490.2929496

[8] Babis Koniaris, Maggie Kosek, David Sinclair, and Kenny Mitchell. 2017. Real-time rendering with compressed animated light fields. *Computers and Graphics* (2017). http://researchrepository.napier.ac.uk/Output/951667

[9] Maggie Kosek, Babis Koniaris, David Sinclair, Desislava Markova, Fraser Rothnie, Lanny Smoot, and Kenny Mitchell. 2017. IRIDiuM+: Deep Media Storytelling with Non-linear Light Field Video. In *ACM SIGGRAPH 2017 VR Village (SIGGRAPH '17)*. ACM, New York, NY, USA, Article 10, 2 pages. https://doi.org/10.1145/3089269.3089277

[10] Suk Kyu Lee, Hyunsoon Kim, Albert Yongjoon Chung, and Hwangnam Kim. 2017. Integrated approach of streaming 3d multimedia contents in real-time for mobile devices. *Multimedia Tools and Applications* (23 Jan 2017). https://doi.org/10.1007/s11042-016-4339-5

[11] Marc Levoy and Pat Hanrahan. 1996. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 31–42.

[12] P. Merkle, K. Mller, D. Marpe, and T. Wiegand. 2016. Depth Intra Coding for 3D Video Based on Geometric Primitives. *IEEE Transactions on Circuits and Systems for Video Technology* 26, 3 (March 2016), 570–582. https://doi.org/10.1109/TCSVT.2015.2407791

[13] K. Muller, P. Merkle, and T. Wiegand. 2011. 3-D Video Representation Using Depth Maps. *Proc. IEEE* 99, 4 (April 2011), 643–656. https://doi.org/10.1109/JPROC.2010.2091090

[14] K. Mller, H. Schwarz, D. Marpe, C. Bartnik, S. Bosse, H. Brust, T. Hinz, H. Lakshman, P. Merkle, F. H. Rhee, G. Tech, M. Winken, and T. Wiegand. 2013. 3D High-Efficiency Video Coding for Multi-View Video and Depth Data. *IEEE Transactions on Image Processing* 22, 9 (Sept 2013), 3366–3378. https://doi.org/10.1109/TIP.2013.2264820

[15] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology* 22, 12 (Dec 2012), 1649–1668. https://doi.org/10.1109/TCSVT.2012.2221191

[16] N. V. Sunitha, K. Raju, and N. N. Chiplunkar. 2017. Performance improvement of CUDA applications by reducing CPU-GPU data transfer overhead. In *2017 International Conference on Inventive Communication and Computational Technologies (ICICCT)*. 211–215. https://doi.org/10.1109/ICICCT.2017.7975190

[17] B. v. Werkhoven, J. Maassen, F. J. Seinstra, and H. E. Bal. 2014. Performance Models for CPU-GPU Data Transfers. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 11–20. https://doi.org/10.1109/CCGrid.2014.16

[18] A. Vetro, T. Wiegand, and G. J. Sullivan. 2011. Overview of the Stereo and Multiview Video Coding Extensions of the H.264/MPEG-4 AVC Standard. *Proc. IEEE* 99, 4 (April 2011), 626–642. https://doi.org/10.1109/JPROC.2010.2098830