Real-Time Multi-Gigapixel Light Field Ray Traced Rendering with JPEG Compression

Nicholas Wells, Matthew Hamilton

Memorial University of Newfoundland, Department of Computer Science

VISUAL & ANALYTIC COMPUTING LAB

MEMORIAL UNIVERSITY

Introduction

Cinematic quality light field views can enable photo-realistic free viewpoint images in real time, although requiring billions to trillions of pixels per frame.

In this work, we propose a real-time ray-traced light field renderer which utilizes a real-time JPEG decoder to allow the viewing of very large light field images at interactive frame rates. We introduce a JPEG texture decompression method similar to Sodsong et al [1] providing real-time random access to JPEG images in a ray traced CUDA kernel. Our JPEG decoder is implemented through the generation of a lookup table correlating JPEG blocks with their position within JPEG's encoded bit stream.

We demonstrate the utility of our JPEG decoder by rendering a light field image of resolution 184,000 x 184,000 (~32 billion pixels, ~100GB) in real-time.

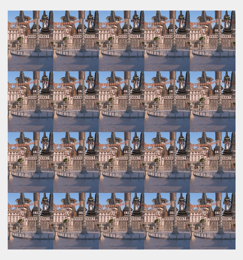
Objectives

- **Real-time Rendering of Large light field images (32 gigapixel)**
- **Compression of large multi gigapixel images**
- **Real-time JPEG random access decoder**

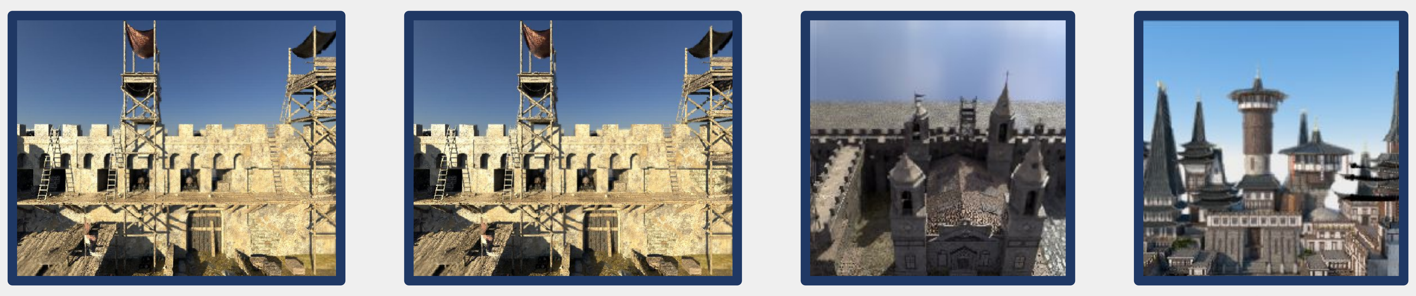
Light Field JPEG Production

Many existing light field images are publicly available, however even the largest images are relatively small.

- **We target gigapixel-scale light field images**
- **Rendered synthetic light field images**
- **sectioned into (64k x 64k) blocks,**
- **compressed with JPEG**



Light Field Image sub section



Gate_512 Gate_720 Church_512 Roofs_256

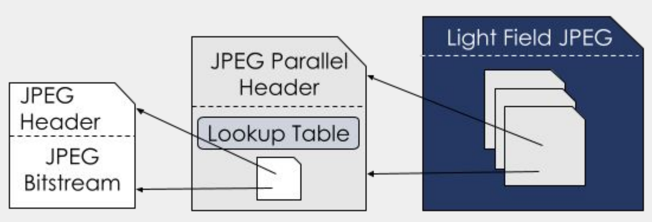
Scene Name	Spatial Resolution	Directional Resolution	FOV	GigaPixels	Size (GB)
Gate_512	512 x 512	174 x 174	120	~8	~23.8
Gate_720	720 x 720	256 x 256	160	~34	~102
Church_512	512 x 512	115 x 115	45	~3.5	~10.4
Roofs_256	256 x 256	115 x 115	45	~0.86	~2.6

Light Field JPEG Compression

We **compile a JPEG parallel header on the CPU** on start up of our viewer. The JPEG parallel header contains the **conventional JPEG header** and further contains a **lookup table for the JPEG bit stream**.

The look up table contains a reference to the **location** in the **encoded bitstream** of **each JPEG block**, along with the **decoded DC coefficient** of the block.

Load JPEG parallel header into **GPU memory** allowing real-time decoding within a **CUDA kernel.**



Real-Time Light Field Rendering

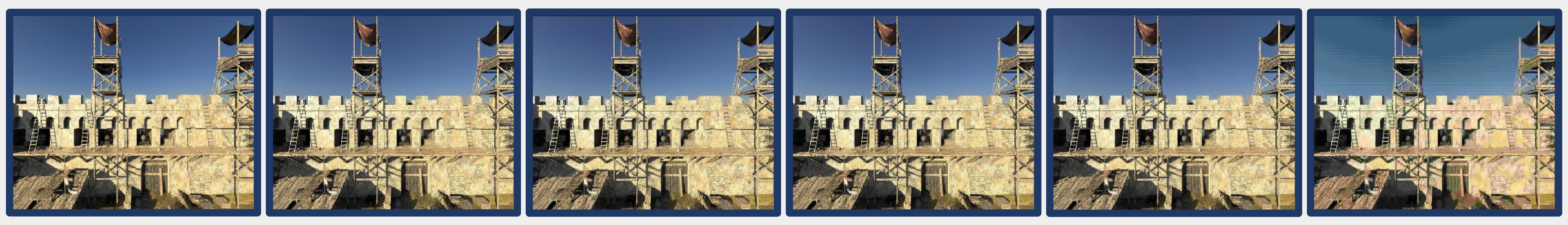
We consider a camera viewing a light field mapped onto a plane. We implement a ray-tracing approach based on NVIDIA's Optix ray tracing engine to render this view. Once a ray intersection occurs with the light field plane, we use the location and direction of the intersection to decode the required light field ray. We then decode the JPEG texture at the index constructed from the light field inference.

Contacts

Nicholas Wells: nwwells@mun.ca
Matthew Hamilton: mhamilton@mun.ca
http://www.vaclab.ca

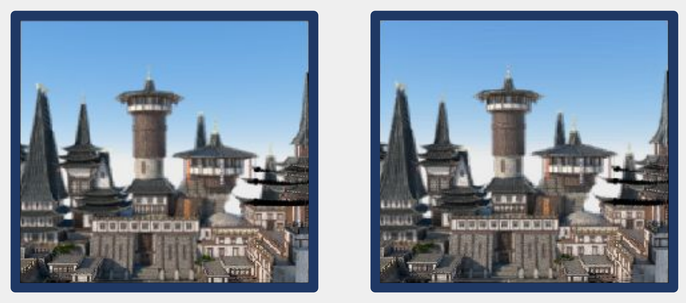
Rendering Experimental Results

Progressively Applying JPEG Compression on Gate_512 Scene



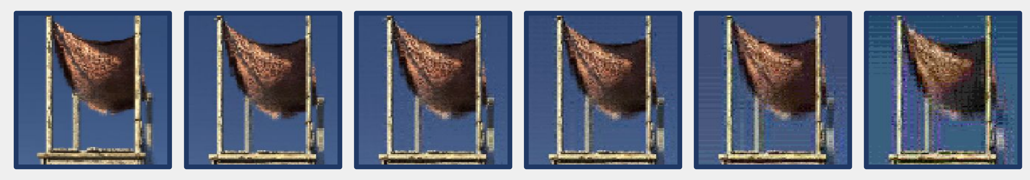
Raw JPEG 90 JPEG 70 JPEG 50 JPEG 30 JPEG 5

Raw JPEG



Raw JPEG

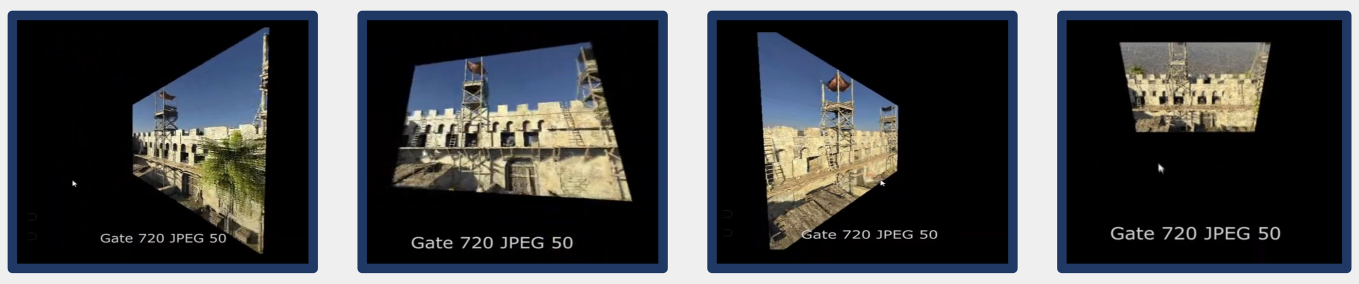
JPEG Level	90	70	50	30	10	5
PSNR	19.35	19.3	19.28	19.33	19.30	18.91
SSIM	0.67	0.67	0.66	0.65	0.60	0.57
Ratio	2.74:1	4.0:1	4.5:1	5.1:1	6.3:1	6.7:1



Raw JPEG 90 JPEG 50 JPEG 30 JPEG 10 JPEG 5

The time to load our JPEG implementation into VRAM required less than 3 minutes for all cases considered. We apply a 3x3 Gaussian based point spread sampling of the light field image. Camera configuration adds an additional 160 MB overhead to the rendering process.

Interactive Results



Looking Right Looking Forward Looking Left Looking Down

		RAW		JPEG		
Scene Name	Viewer Distance	FPS	VRAM	FPS	Size VRAM	JPEG level
Gate_512	1.0	~530	~23.8	~125	~9.8	90
Gate_512	3.0	~560	~23.8	~230	~9.14	90
Gate_720	1.0	NA	~102	~230	~22.8	50
Gate_720	3.0	NA	~102	~320	~22.8	50
Church_512	6.0	~500	~10.4	300	~3.2	90
Roofs_256	6.0	~560	~2.6	230	~0.8	50

- **Interactive viewer provides 6 degrees of freedom around the light field image**
- **Views rendered at a resolution of 600 x 600, with a FoV of 60°**

Experimentation conducted on a system with: CPU: AMD Ryzen 7 3700X 8-Core Processor, RAM: 64GB GPU: NVIDIA RTX 3090 with 24GB VRAM

JPEG Memory Analysis

We construct an equation to provide the total cost of memory storage for the JPEG header vs storing the raw image array. By the nature of JPEG's variable bit length and Huffman encoding, there is no way to determine the size of the bit stream until the image is compressed. As a result, we choose to ignore the bit stream size to construct a ratio formula.

We can then compare our JPEG formula with the raw image size equation and produce the ratio 1:0.14. Although 1:0.14 is significant, in practice we need to consider the bit stream data. This data will be encoded as efficiently as the JPEG compression scheme offers.

JPEG: 36 + (4534 x number of subImages) + (Pixels x 27/64) + BS
Raw: 36 + Pixels x 3

Parameter	Size (Bytes)
Light Field Parameters	36
Raw Image Size	Pixels x 3
Conventional JPEG Header	4534
JPEG Lookup Table	Pixels x 27/64
JPEG Bitstream	BS

Conclusion

In this work, we show how JPEG compression can enable use of gigapixel light field images for the representation of high quality light field images. The novel JPEG ray decoder implementation is capable of decoding 32 gigapixel light field images in real-time on consumer graphics hardware. Though we demonstrate with rendered light fields, live-captured scenes are naturally supported.

In order to scale to terapixel light field images, supporting also video, compression ratios achieved must increase by orders of magnitude. We suggest that to reach these extreme compression requirements, studying compression targeted at the redundancy found within light fields will be required, for example inter-hogel redundancy is not captured in this work's solution.

References

[1] Wasuwee Sodsong, Minyoung Jung, Jinwoo Park, and Bernd Burgstaller. 2016. JParEnt: Parallel entropy decoding for JPEG decompression on heterogeneous multicore architectures. Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores, PMAM 2016 (2016), 104–113.