Towards Immersive Cinematic Video for Immersive Displays

Nicholas Wells
Department of Computer Science
Memorial University of Newfoundland
St. John's, Canada
nwwells@mun.ca

Mathew Hamilton
Department of Computer Science
Memorial University of Newfoundland
St. John's, Canada
mhamilton@mun.ca

***Abstract*— Rendering cinematic quality computer generated imagery (CGI) is too expensive for real-time. We show how offline pre-rendering can enable real-time rendering of a static scene at cinematic quality. We use OTOY Octane to pre-render static scenes offline into a light field image, using a custom Open Shading Language (OSL) camera shader. We show how to perform reconstruction of an arbitrary view of the static scene from the light field in real-time using NVIDIA's Optix API, using RTX hardware accelerated ray tracing. Within the created real time ray tracer, we use NVIDIA's CUDA API to accelerate ray reconstruction. The result is an ability to render real-time views of a static scene at cinematic quality. By pre-rendering a sequence of light field images we can extend to video. This can be further extended to view video on immersive displays, including virtual reality and holographic displays. Future work would include compressed light field video, to support longer, streaming video. (*Abstract*)**

Keywords—immersive video, VR/AR, light field visualization (*key words****)***

I. INTRODUCTION

In video designed for conventional 2D displays, a single camera view is required for each frame. For interactive video, this rendering must be performed in real-time. However, where video is non-interactive and designed for passive consumption, these frames can be pre-rendered using offline rendering techniques, resulting in higher quality images at the cost of a one-time upfront rendering time.

Immersive displays such as virtual reality (VR), augmented reality (AR) headsets and standalone 3D displays offer the promise of enhanced video for both interactive and passive consumption viewing. With immersive displays, the observer's viewing position is not fixed, opening up another dimension to video viewing. Immersive displays present the possibility for a passive consumption video experience where the viewer's vantage point shifts with view head and body movement.

The benefits of light field consumption can also be gained within more common 2D display interfaces as consumers can define the view of a scene they want to focus upon. This can include refocusing the camera onto interesting parts of a scene to decipher a deeper meaning or to retrieve better scale within the scene. Within the 2D display common interaction mechanics already exist within other real time rendering solutions and as such interaction can come naturally into video.

This presents a new challenge. To provide even passive video on immersive displays requires not just a single camera view, but multiple simultaneous views whose position varies dynamically with the viewers' head position. Providing video in this context now presents a greater challenge, as multiple viewpoints must be captured per video frame. These multiple viewpoints can further vary, depending on the movements of the viewer. Depending on the specifics of the immersive display, this can range from 2 viewpoints (left and right eye in Head Mounted Displays) to millions of viewpoints (3D light field displays). This poses a question of how to create and represent all the possible views of a scene that could be taken of a scene by a viewer.

This first begins with how we capture even a single frame of an immersive video with all the possible viewpoints captured. The idea of capturing all views from within a scene can also be formulated to be capturing all the rays of light within a scene. By using this definition we can use the concept of a light field to retrieve a single frame of all possible views or all possible light rays. A light field is the idea that from within an area in a 3D space we can calculate all the possible light rays (or colors for simplicity) which will pass through the area. We can think of the area within the 3D space as a window in which when looking through it, we can see everything outside even when moving our eyes while looking at the window. The concept of a light field is the same as it captures all the views from outside it.

Modern cinema-quality graphic rendering techniques have great computational cost which require large computational times outside the ability to render within real time. By creating a light field we can generate a movie within the same offline methods which provide extreme quality while the single images generated now contain multiple camera angles within them.

As one of the primary contributions of this paper, we show how from this pre-rendered, high-quality light field image many views can be dynamically generated in real-time, to

accommodate the requirements of immersive displays. The other main contribution is showing how to render light fields using OSL cameras in Otoy's Octane.

II. Related Work

The introduction of light fields and simple methods to render a single view are concepts discussed in many papers. Levoy and Hanrahan [1] introduced use of light fields and generation of novel views from light field image representation. They discuss the implementation of a single light field frame within their paper and do not mention the requirements or methods needed to view a light field video in real time. Their methods are also not designed for a pure ray-traced implementation and instead describe a mixed approach including projective rendering techniques.

Another paper discusses an approach to real time light field rendering display [2] This paper discusses an approach to simulation of light field simulators without the explanation of how to generate high quality light fields. The created simulator from this approach is also based upon a hybrid rasterization ray-tracing approach, which results in less rendering ray control than a pure ray-traced approach.

Another paper discusses the rendering of complex realistic images into a VR scene [3]. The method used within the paper describes an approach to rendering realistic images within VR from multiple viewpoints. The approach though does not capture view dependent complex lighting effects other than simple effects which can be added with post-processing effects at rendering time.

The Octane rendering engine from OTOY provides a fast and efficient approach to physically-based rendering. This approach is explicitly described by the book Physically Based Rendering: From theory To Implementation [4]. Octane as a result provides cinematic quality rendering for conventional 2D images. The program does not provide the ability to render light field images as a default function, however.

III. Background

Our project focuses on an area under-developed within current papers. Our project uses ray tracing technology which involves the ability to cast a ray (a line) from a starting point in a space into a direction based on another point in space. We then use the concept of light fields [1]. These are areas within a world where we can represent all light (or colors) which can be seen from the position of the area on screen. This is performed by tracing a given number of all the possible rays which pass through the area.

IV. Generating Lightfield Images.

A. Creation of Light Fields

To be able to generate an immersive video, we need to start much like any other video is created. We first need a scene so our cameras have something to capture. To do so within computer generated graphics requires a rendering program. We chose to use a rendering engine designed for large cinematic quality projects as this will allow our results to contain the highest cinematic quality outputs. Otoy Octane rendering engine, is designed exactly for the purposes above. Its rendering times are much slower than real-time as a result.

Along with the quality of the rendering engine we also needed the software to handle the technical requirements for generating light fields. These requirements created problems within other rendering engines we first began testing with. An important issue is the size of generated images. For our project to deliver a high quality light field for rendering, the size of even a single frame of a light field video would surpass any standard photo of the same output size. Since Otoy is designed with large projects in mind the program itself will still generate images of any size given until physical per system limits prevent rendering.

Light field rendering contains an unnatural camera state compared to other camera techniques in filming. The double frustum [5] of light field cameras behaves differently to standard single view cameras and is unnatural to think about within the real world. We call this a *light field camera.* Otoy's Octane renderer has the ability for cameras to be generated with Open Shading Language (OSL) scripted cameras known within Octane as OSL cameras. The OSL camera offers us the ability to generate custom cameras for different photographic needs. Within our project an OSL camera shader is written to generate a light field camera that contains a double frustum.

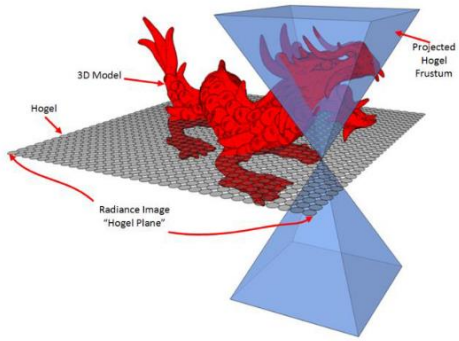


Fig 1: Double-Frustum Rendering [5].

B. Open Shading Language (OSL) Camera

The written shader first begins by splitting the actual output canvas into a grid of pixels. The width and height of this grid is not the size of the output canvas but will pertain to a single output view's resolution when the light field is rendered, these cells are known as *hogels.* Now within each hogel of this grid are a number of pixels which represent all the possible rays of the output pixel, starting from the hogel's location in the scene and pointing towards the scene based on the field of view (FoV).

C. Saving a light field

The light field camera created by the OSL-camera shader allows the light field to be saved as a single large PNG image and when viewed raw a quick observation can be made about the grid in which we can see the formation of hogels. This

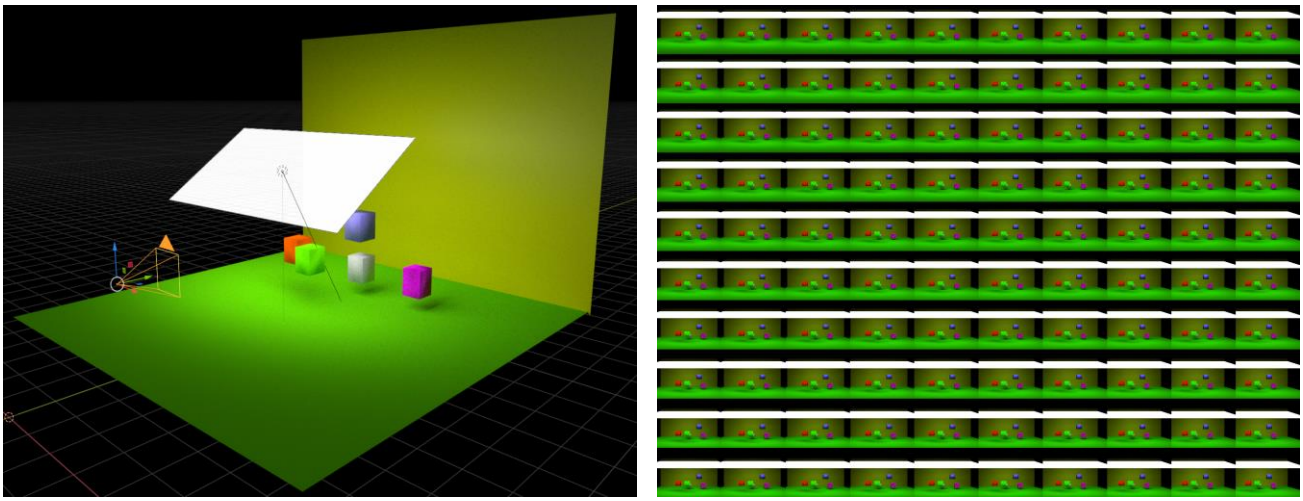


Fig 2: Comparing a Scene and The Light field Camera Output Generated from it.

format is also of benefit to rendering as each hogel will represent the same output pixel as its grid location and the entire image can be loaded onto hardware to increase performance.

To then generate an immersive video, we follow standard methods for generating movies, we now look at the generated light field not as a single image but instead as a single frame within a movie. So now we need to generate every frame within the scene, so we move the actors and light field camera into their new positions based on the frame and render.

D. Testing and Results

Having generated a light field image, we needed to develop a method to view the generated images. This began with generating 2D images from the light field based on orthographic projections of the light field (taking a single pixel from each hogel all corresponding the same light ray angle). Once this process was complete an output image would be saved and could be viewed. By manual inspection we concluded that the results were correct and moved onto a real-time renderer.

E. Scaling to Real-Time and Videos

After initial testing had been completed we began to formulate the design decision on how we would generate a final viewer for our immersive videos. The idea of rendering on a plane in a scene, then placing our immersive video onto the plane became our method of choice. This method offered great flexibility on our ability to test how an immersive video would perform under different settings and specifications to both the plane and camera viewing. This would include the ability to move around the plane to view our light field from multiple angles. As well, we can modify the size of the plane to represent different display sizes and resolutions.

V. Rendering LightField in Realtime

A. NVIDIA Optix 7.3 Ray Tracing Engine

To begin with a real-time render we needed to evaluate current real time programing APIs and software. Since we were not constrained by any program, we looked towards an API which did not restrain the use of our program. We then decided on the use of the NVIDIA Optix 7.3 Ray Tracing Engine (known hereto as Optix) as this allowed us to create our real time renderer in a very basic and efficient manner. This engine also requires the use of NVIDIA CUDA API which enables our program to interact with the graphics card directly.

Optix offered the ability to generate a ray tracing environment in a fast and efficient method. Optix also gave low-level control to the application developer allowing projects developed on the engine to also be highly efficient. The ability of memory control on both CPU and GPUs and their interactions was included by the low-level design and is now the responsibility of the user application through the use of the CUDA API. Being able to control memory within our application is essential for the large memory size of immersive videos.

The resources available within Optix for how to develop within the application was also of value as our program was created using code available within the sample projects contained within the installation of Optix. This enabled our renderer to begin with some basic features already included and enabled focus onto code which pertained to the rendering of immersive video and not to basic rendering techniques already understood.).

B. Building the Renderer

The creation of our render now begins with the basics of how Optix renders a scene. We begin this process by creating a basic Optix program. This entailed the exploration and modification of the sample code provided within the Optix API. This allowed our project to already contain the basic features of a ray tracing program. This included a method to define geometry within our scene, create a camera and a system to render from that camera in a window on the screen. The code also contained a system to generate rays from each pixel location. From this basic code we now needed to build our immersive video renderer. The first step within this process was to build a single light field renderer. This process begins with how our program accesses a light field.

Since our implementation of a single light field is based upon a large PNG file we need to load this into our program. This required the use of the lodePNG library to be able to process the compression of a PNG file. From this library we load the image as a raw array of unsigned characters in which the values follow a pattern of

$$R(i), G(i), B(i), R(i+1), G(i+1), B(i+1) \ldots \quad (1)$$

where i represents a pixel location. These raw values are stored onto the system memory (ram) and need to be moved onto the graphics card video memory to enable the high speed look up required for our light field. To do this we generate a 2D CUDA texture on the graphics card and copy our light field from the system memory onto the device using the CUDA memcpy() function.

We now attempt to render a conventional 2D view of the scene we have pre-rendered into a light field. This requires a viewer camera position which images a plane representing the light field image of our target scene.

This begins with sending rays from our camera position into the scene. The detection of a ray hitting onto the plane we intended to use as our light field generates a hit in Optix. From this hit we need to retrieve where on the plane the ray intersected, the direction the ray was going, and the specifics of the light field we have hit.

Based on the ray, we can reconstruct the light field from the sampled rays contained in the light field image, using a framework similar to that presented elsewhere [2].

VI. Results

The objective of rendering immersive video requires our results of all components of the project to behave according to their function. We can then begin focusing on the results of constructing light fields. After such we can examine the performance of our real-time rendering of such scenes.

Our light field creation method contained parameters to optimize the quality of a generated light field. This resulted in the ability of a wide range of rendering times based on the desired quality of a light field. During testing, generation of light field images of sizes 8000 pixels by 8000 pixels could be generated within 8 minutes under a direct lighting method with low sampling. Although this result indicates we can generate large light fields, time requirements would rise dramatically when rendering complex cinematic scenes.

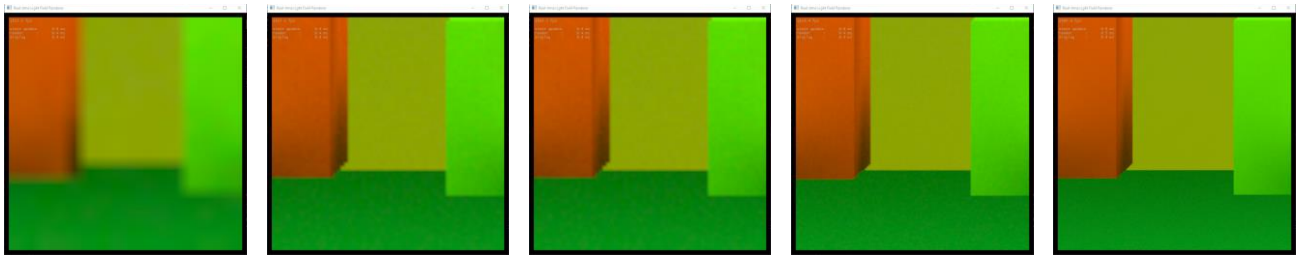
Fig 3: Program displaying entire light field Image loaded (Not single view)

Our real-time rendering engine performance needs to be within an interactive range. We use a frame rate measure to denounce speed, denoted as frames per second (fps). Movies and other such graphic entertainment average frame rates of 30-60 fps. As such our results need to be well within these limits to be in an interactive range.

We first then began testing with standard 2D images. When displaying these standard 2D images averaged frame rates ranged between 1200-1700 fps. A standard image of size 26754 X 26754 would have a rendered real time frame rate between the range 1450 - 1650 fps.

Once rendering of standard 2D images produced the good results we moved to testing rendering a view of a light field. Through our testing light field images would run within the same range of display as our simple 2D images. These results are much higher than the stated frame rate of movies. By taking an averaged frame rate within results we can subtract from the standard movie speed requirements of 60 fps to receive how many frames we have within our program to manage the time requirements of turning our single frame renderer into a multi frame.

The number of hogels used within a light field image had a high impact on the quality of the image produced. During testing the number of hogels directly represented the number of pixels a



16X16 Hogels 32X32 Hogels 64X64 Hogels 128X128 Hogels 256X256 Hogels

Fig 4: Increasing number of Hogels resulted in increased quality within a view during real time rendering.

view would have, as a result when using low hogel resolutions our displayed results would be of low resolution, but when using high number of hogels we would gain a high-resolution image. This effect then had to be balanced by the number of different views we would like to view our scene from to be able to handle the physical limits on test machines

VII. Conclusion

Our project generates an approach to rendering light field scenes in real time. We begin by creating light fields through the Otoy Octane rendering engine using OSL cameras. From there we tested the generated images to validate they were light field images. From the results we began the next phase which entailed the development of our real time rendering engine. We used Optix and CUDA to generate a real time ray tracer that displays any arbitrary view from the defined camera.

VIII. Future Work

The next step towards immersive video is expanding our rendering engine to display multiple frames of a light field scene over a time period. Once our rendering engine handles multiple frames a compression of light fields will provide our program with the ability to store enough light fields to generate entire immersive movies in real time.

References

[1] M. Levoy and P. Hanrahan, "Light field rendering," *Proc. 23rd Annu. Conf. Comput. Graph. Interact. Tech. - SIGGRAPH '96*, pp. 31–42, 1996, doi: 10.1145/237170.237199.

[2] M. Hamilton, C. Rumbolt, T. Butyn, D. Benoit, R. Lockyer, and M. Troke, "P91: Light Field Display Simulator for Experience and Quality Evaluation," *SID Symp. Dig. Tech. Pap.*, vol. 49, no. 1, pp. 1523–1526, 2018.

[3] P. Lall, S. Borac, D. Richardson, M. Pharr, and M. Ernst, "View-Region Optimized Image-Based Scene Simplification," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 1, no. 2, pp. 1–22, 2018, doi: 10.1145/3233311.

[4] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering:From Theory To Implementation*. 2018.

[5] T. L. Burnett, "Light-field Display Architecture and the Challenge of Synthetic Light-field Radiance Image Rendering," *SID Disp. Week*, pp. 899–902, 2017.