STEREOSCOPIC RAY TRACING ON GRAPHICS PROCESSORS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ALPER DAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

NOVEMBER 2012

Approval of the thesis:

STEREOSCOPIC RAY TRACING ON GRAPHICS PROCESSORS

submitted by **ALPER DAŞ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Veysi İşler _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Veysi İşler
Computer Engineering Dept., METU _____

Asst. Prof. Dr. Tolga Çapın
Computer Engineering Dept., Bilkent University _____

Asst. Prof. Dr. Murat Manguoğlu
Computer Engineering Dept., METU _____

Asst. Prof. Dr. Sinan Kalkan
Computer Engineering Dept., METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ALPER DAŞ

Signature :

ABSTRACT

STEREOSCOPIC RAY TRACING ON GRAPHICS PROCESSORS

Daş, Alper
M.S., Department of Computer Engineering
Supervisor : Assoc. Prof. Dr. Veysi İşler

November 2012, 56 pages

Stereoscopic rendering methods generate two images for binocular viewing. Exploiting the spatial coherence present in the surfaces rendered, it is possible to reduce the rendering time of a stereoscopic image pair significantly. Ray tracing is a frequently used rendering method for realistic image synthesis applications. With the tremendous increase in computational power of Graphics Processing Units and their availability for general purpose programming, ray tracing methods suitable for Graphics Processing Units have been developed.

In this thesis, a parallel stereoscopic ray tracing method that reuses view independent information in one of the stereoscopic image pairs for generating the other image is presented. The results of the method are analyzed using a perception based error detection mechanism. The main contribution lies in the new parallel algorithm that takes advantage of the massive parallel processing power of the modern Graphics Processing Units efficiently. Results show that about 20 times speed-up is achieved over

a previous work which is based on a sequential algorithm in the worst case among the experiments carried out. According to the perception based quality analysis performed, lower than only 2% of the pixels in high resolution images is expected to be perceived to be in error.

Keywords: graphics processing units, ray tracing, CUDA, stereoscopic rendering

ÖZ

GRAFİK İŞLEMCİLERİ ÜZERİNDE STEREOSKOPİK IŞIN İZLEME

Daş, Alper
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Veysi İşler

Kasım 2012, 56 sayfa

Stereoskopik görüntü oluşturma sistemlerinde binoküler izlemeye uygun iki adet resim oluşturulmaktadır. Çizilen yüzeylerin uzaysal tutarlılığından faydalanarak bir stereoskopik görüntü ikilisinin oluşturulma süresinin önemli ölçüde azaltılması mümkündür. Işın izleme gerçekçi görüntü üretme uygulamalarında yaygın olarak kullanılan bir yöntemdir. Grafik İşlem Birimleri'nin işlem gücündeki muazzam artış ve bunların genel amaçlı programlamaya açılması ile birlikte Grafik İşlem Birimleri'nde çalışmaya uygun ışın izleme yöntemleri geliştirilmistir.

Bu çalışmada stereoskopik resim eşlerinin birindeki bakış açısından bağımsız bilgiyi ikincisinin oluşturulmasında kullanan bir paralel stereoskopik ışın izleme yöntemi sunulmaktadır. Sunulan yöntemin sonuçları algı bazlı bir hata denetleme mekanizması ile analiz edilmektedir. Çalışmadaki asıl katkı Grafik İşlem Birimleri'nin paralel işlem gücünü verimli olarak kullanan yeni bir paralel algoritma geliştirmede yatmaktadır. Sonuçlar, paralel olmayan bir algoritma üzerine kurulu olan önceki bir

çalışmaya kıyasla, yapılan deneyler arasında en kötü 20 kat hızlanma sağlandığını göstermektedir. Gerçekleştirilen algı bazlı kalite analizine göre, yüksek çözünürlüklü görüntülerde piksellerin sadece yüzde 2'sinden azının hatalı olarak algılanması beklenmektedir.

Anahtar Kelimeler: grafik işlem birimi, ışın izleme, CUDA, stereoskopik görüntü oluşturma

To Those Who Make This Possible

ACKNOWLEDGMENTS

The author wishes to express his deepest gratitude to his supervisor Assoc. Prof. Dr. Veysi İşler and Asst. Prof. Dr. Ahmet Oğuz Akyüz for their guidance, advice and encouragements.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Stereoscopic visualization systems exploit the depth perception mechanism of human visual system by providing stereoscopic image pairs for binocular vision. Perceived depth presented via stereoscopic viewing leads to more immersive visual experience. Although the idea dates back to early nineteenth century [18], recent technological advancements, mostly in the availability of the presentation medium, has made stereoscopic content popular again.

Along with the great acceptance gained by stereoscopic viewing equipments, 3D content generation in multimedia platforms has increased tremendously in the recent years. In addition to being a rising trend in motion picture industry, many major computer game titles support 3D vision natively. Even limited automatic stereoscopic support can be added with minimal effort using underlying graphics system built-in functionality [3]. It should be mentioned that due to extra immersion created by stereoscopy, it is a frequently used method in virtual reality applications. Leaving the details of stereoscopic rendering to its own section in this chapter, it needs to be mentioned that the application of stereoscopic rendering with a naive approach brings along the cost of rendering a second image from scratch, which amounts to nearly twice the cost of monocular rendering in practical cases. However, due to strong spatial coherence between the stereo image pairs, it is possible to reduce the computational requirements greatly [9].

Stereoscopic image synthesis methods differ from their monocular counter-parts in the adoption of optimization methods to increase rendering performance. Therefore, a brief overview of rendering techniques seems to beneficial. Rendering algorithms can be broadly classified into two distinct groups based on their adoption of either local or global illumination models. An overview of global illumination can be found in the work by Möller[21, Ch. 9] and Shirley[27, Ch. 8] gives information about the local illumination based rendering of the mainstream graphics pipeline .

Renderers that use local illumination model consider each object independent of the others in the scene. OpenGL and DirectX are two popular rasterizer implementations that adopt this model. The advantage of this model is its ability to render higher number of polygons in the same time period compared to global illumination models. Due to its speed advantage, this model is dominantly used in real-time applications. However, the features of images generated by this method lack some visually significant phenomena present in real-life, which are desirable in realistic rendering applications. Some of these are reflection, refraction and subsurface scattering. Although these mentioned phenomena are not present in the core of the model, there are methods to approximate them to some degree. Generally these approaches are far from being realistic and efficient when practical cases are considered.

Contrary to local illumination methods, global illumination algorithms take light interaction between surfaces into consideration and produce more realistic results at higher computational costs. Global illumination algorithms more closely simulate the underlying physical phenomena. In an attempt to create a computationally tractable model, a rendering equation (which has later become *the rendering equation*) that captures most of the features required for realistic image synthesis is proposed by Kajiya [19]. Although it is a simplified model of reality, it is still computationally quite complex for most applications. Therefore, nearly all practical methods try to approximate a solution for this equation in some way, leading to distinct sets of global illumination algorithms.

What type of interaction is modeled gives us a group of rendering methods that excel in its own ways and fall short in others. Radiosity, for instance, is a rendering method that is based on calculation of radiance energy transfer in the scene among diffuse surfaces [15]. Soft shadows, color bleeding are the phenomena that are naturally provided by this method. However, non-diffuse surfaces are not modeled in this method. Although methods exist to incorporate specular properties to surfaces, they provide solutions up to a certain degree [25]. Another frequently used rendering method that provides global illumination is ray tracing, in which light ray traversal paths are simulated and surface radiance values are calculated [29]. The advantage of this method is its ability to provide shadow, reflection and refraction effects easily. Color bleeding, caustics phenomenon are beyond what is provided by this method.

1.1 Stereoscopy

Depth cues perceivable by human visual system can be divided into two groups based on the number of image sources. Visual perception from a single image source is called *monocular vision* and using two images for viewing is called *stereoscopic vision*. Although there is no depth information in monocular vision, various cues are used by the visual system. Lipton [20, Ch. 2] gives detailed description about the following seven major monocular depth cues: light and shade, relative size, interposition, textural gradient, aerial perspective, motion parallax and linear perspective.

With the inclusion of binocular vision, depth information becomes available. Two stereoscopic cues, vergence and disparity, leads to depth perception. Vergence is related to the direction of the line of sight of the eyes with respect to each other. This cue loses its effectiveness as distance increases due to decreasing angle change per unit of distance. Disparity is the distance between the projections of the same surface point onto the retinas of the eyes. Disparity occurs in the horizontal axis, since both eyes are separated horizontally.

In stereoscopic image generation systems, target scene is rendered from two cameras which are configured to produce easily perceivable image pairs. Resultant image pair is presented to the user via one of many presentation means. Time multiplexed, polarized or color filtered solutions are some of the well known methods, each with its own advantages and disadvantages. In the anaglyph method which is based on color filtering, each of the stereoscopic images are encoded into a single image. The viewer uses a color filtering equipment(e.g. red-cyan glasses) to extract the encoded stereoscopic images from the displayed image. Figure 1.1 visualizes the anaglyph method that uses color filtering to extract image pairs from color coded anaglyph image.

There are three major geometric configurations in a stereoscopic camera system as depicted in Figure 1.2, namely off-axis, parallel and toe-in camera systems. Off-axis camera systems consist of two horizontally displaced cameras with parallel optical axes and superimposed image planes. Due to interocular distance and superimposed image planes, viewing volume of each camera forms a horizontally asymmetric frustum. In parallel camera configuration, cameras are horizontally shifted from each other similar to off-axis system but they preserve their symmetrical viewing frusta. In toe-in camera system, similar to other two camera systems, horizontal offset between cameras is present but the optical axes of the cameras are not constrained to be parallel to each other.

1.2 Ray Tracing

Ray tracing is a rendering technique that simulates the behavior of light rays as a means of generating synthesized images [29]. As opposed to local illumination techniques, which only capture local visibility information by projecting geometry onto a specified plane, ray tracing offers a global illumination solution by using a simplified light ray model. The main advantage of a ray tracer is its ability to simulate reflection, refraction and shadow phenomena without special treatment. Ray tracing

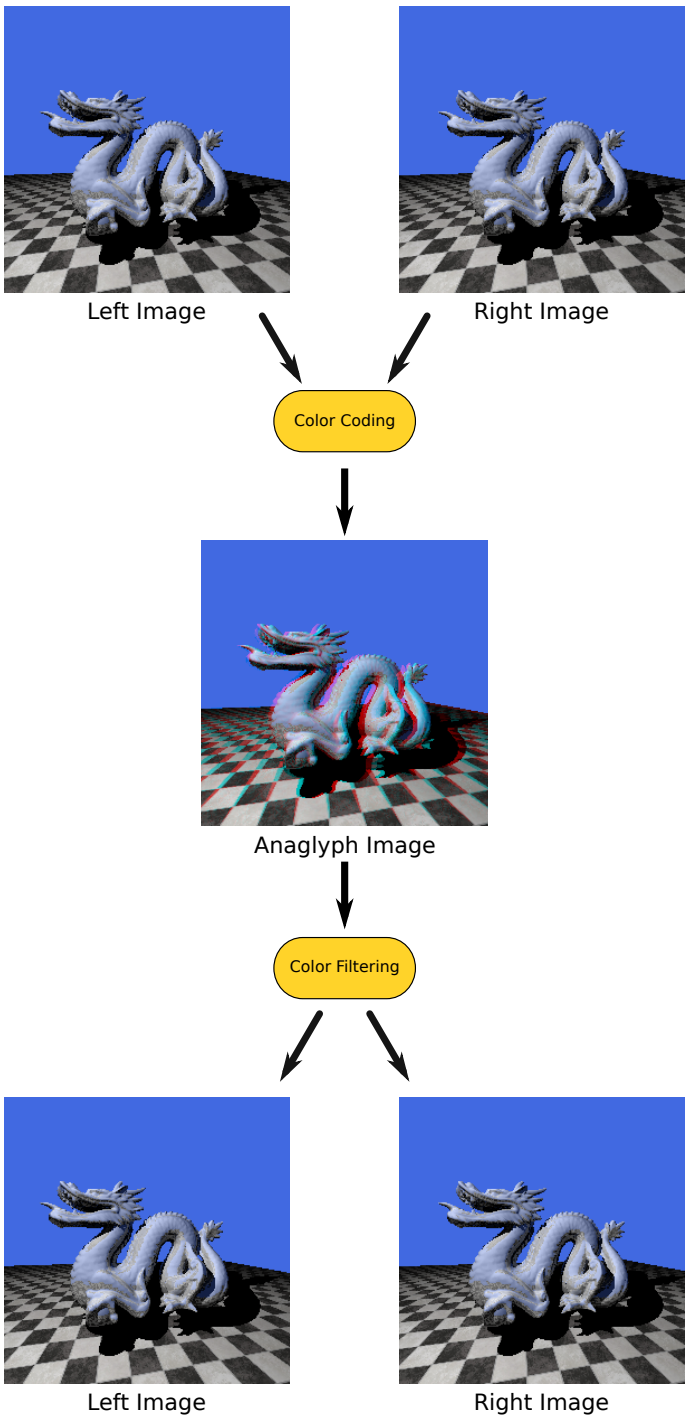


Figure 1.1: Illustration of the anaglyph method.

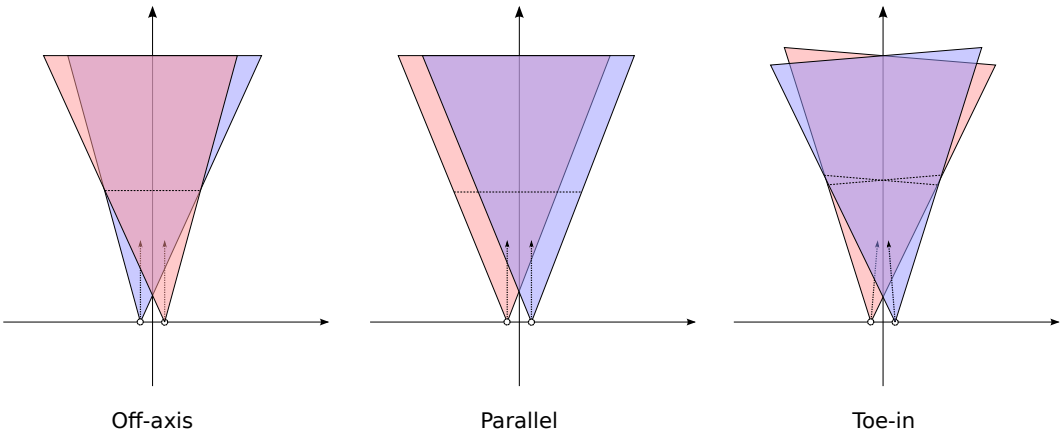


Figure 1.2: From left to right, off-axis, parallel and toe-in camera configurations is illustrated.

achieves the global illumination effects by means of its recursive nature. Figure 1.3 illustrates the procedure. The independence of the computation for each ray makes ray tracing a promising application area for the current graphics hardware where data parallel operations are carried out efficiently. With the availability of high performance graphics hardware computation platforms, namely Compute Unified Device Architecture (CUDA) [1] and OpenCL [4], more efficient methods that exploit the new possibilities are proposed.

1.3 GPU Computation

Graphics hardware performance has increased tremendously in the last two decades to meet the demands of the industry. Current graphics hardware is designed to operate efficiently on highly parallel arithmetic intensive problem domains, with less die space for control logic structures and more on arithmetic operations. This design decision enables the Graphics Processing Unit (GPU) to outperform the Central Processing Unit (CPU) with a great margin in certain application domains.

Early generations of GPUs were designed to provide a fast way of performing certain

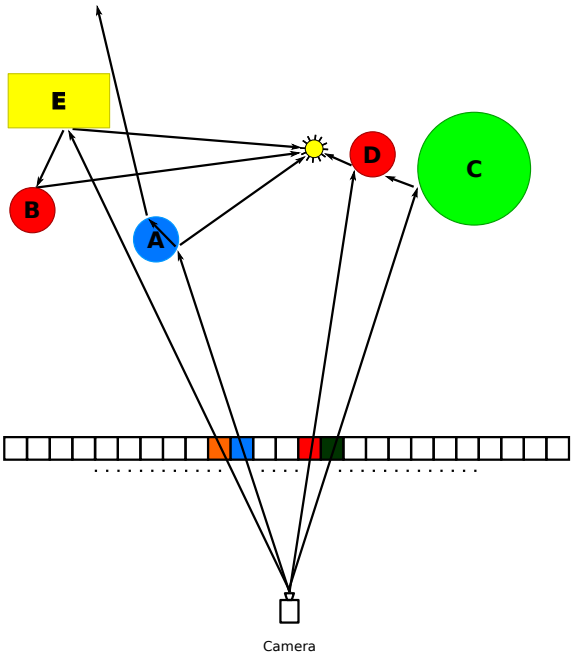


Figure 1.3: Illustration of ray tracing method. B, C, D are diffuse surfaces. A is transparent and E is reflective. Arrows symbolize the rays used in ray tracing.

calculations that appear in rasterization. The flexibility was limited to only changing a small number of states of the machine that defines the fixed functionality pipeline. In the subsequent generations some parts of the pipeline became programmable. This advancement has lead to a way of exploiting this functionality in order to perform general purpose arithmetic operations on the GPU with increased performance compared to the CPUs. The main goal of this advancement was to provide flexibility for the rasterization techniques. However, a high number of problems from distinct domains could be solved with increased performance compared to their CPU counterpart solutions. The main idea was to model the computational problem in terms of the primitive structures provided by the graphics APIs. The inputs to the systems were transformed to fit into the input mechanisms of the graphics APIs (textures and per-vertex parameters) and the functional parts were provided as shading programs.

In the recent years , GPUs have turned into a general purpose parallel computation unit with a general programming interface. CUDA provides a platform that enables

to develop applications that perform efficiently data parallel, compute-intensive operations by using a C-like programming language as the interface.

1.4 Motivation

As previously mentioned, stereoscopic rendering has become quite popular over the last decade in parallel with the demand for stereoscopic content. The more computation is spent on rendering stereoscopic content, the more resources can be saved by providing more efficient methods. With recent developments in the GPU computing platforms, there is a need for a solution to stereoscopic rendering in the context of ray tracing that utilizes the potential of modern GPUs.

1.5 Contribution

There are various methods that exploit the spatial coherence in radiance of surfaces visible in both of the stereoscopic images. However, none of these works utilize the full potential of the GPUs. In this work, a fully GPU-based reprojection method that speeds up the stereoscopic ray tracing process is presented. Furthermore, quality implications of the proposed method is analyzed using a perception based error detection mechanism.

1.6 Outline

In Chapter 2, previous work related to stereoscopic rendering and how the method proposed in this work differs from previous work are discussed. In Chapter 3, the proposed method is explained in detail along with implementation implications and limitations. In Chapter 4, experiments and results is presented. Finally, Chapter 5 summarizes the presented work and discusses possible future work areas.

CHAPTER 2

PREVIOUS WORK

In stereoscopic image synthesis, there is strong spatial coherence between generated image pairs. This coherence is noticible in surfaces with diffuse reflectance properties visible in both images. Various methods have been proposed to exploit this coherence to speed up the image generation procedure. In this chapter, previous approaches related to ray tracing and stereoscopic reprojection are presented.

2.1 Ray Tracing

As mentioned in the previous chapter, ray tracing is based on intersecting rays with geometrical objects in order to simulate light rays. Therefore, most of the computational resources used in ray tracing is spent on ray-object intersection tests [29]. The majority of ray tracing acceleration algorithms either reduce the number of ray-object intersections or provide more efficient intersection tests. A comprehensive classification of ray tracing acceleration techniques can be found in the work by Arvo and Kirk [10].

Space partitioning and Bounding Volume Hierarchies (BVH) are two of the techniques widely used in computer graphics applications to exploit the object coherence in a scene to reduce the number of ray-object intersections [16]. In space partitioning method, the scene is divided into hierarchical volumes, whereas, in BVH, objects

in the scene are grouped hierarchically considering their spatial proximity. In both approaches, the main goal is to form a spatial structure suitable for certain geometric queries. In the ray tracing case, space partitioning or BVH structures reduces the asymptotic complexity of a single ray-object intersection from $O(n)$ to $O(\log n)$ in optimal cases. GPU-based methods are proposed to apply acceleration structures to GPU-based ray tracers. Popov et. al [24] show the efficient application of KDTree traversal structure on the GPU. Gunther et. al [17] propose a GPU based packet traversal method with BVH. In this work, the GPU-based stereoscopic ray tracer uses a BVH acceleration structure.

OptiX [5] is a GPU based ray tracing engine developed by NVIDIA. Although it is close to manually optimized ray tracing applications in performance, it allows wide range of ray tracing algorithms to be implemented without delving into details that are irrelevant to the problem at hand. Certain parts of the pipeline are provided to the OptiX Engine via CUDA programs. To spawn first level rays a *ray generation* program is provided. To define the behavior when rays hit or miss objects in the scene *nearest hit*, *any hit* and *miss* programs are used. To introduce primitive geometry to the engine *intersection* and *bounding box* programs are used. To take action on exceptional cases *exception* programs can be provided. And finally, to gain control over the traversal behavior on the scene nodes *selector* programs can be used. As mentioned by Parker et. al [23], OptiX is designed to be used in any kind of ray tracing application. As a result, there is no imposed dependency to concepts that are specific to computer graphics applications. The stereoscopic ray tracer developed in this work is based on OptiX.

2.2 Stereoscopic Reprojection

Badt [11] proposes a technique to exploit spatio-temporal coherence in ray traced animation sequences to reuse visible surface information from previous frames. Due to the possibility of occlusion by some unseen surfaces in the source image, direct trans-

formation of surface projections leads to problematic and missing regions that need to be processed. A filtering method is used to find the problematic pixels. However, some valid pixels are marked as problematic by this method and some problematic cases are not revealed. In the tests presented by Badt, about 62% of pixels is reused without any further calculation.

Ezell and Hodges [14] adapt the method presented by Badt [11] to stereoscopic ray tracing by treating the rendering of stereoscopic image pairs as two consecutive animation frames where only the camera is moved along its horizontal axis. Since they use the same method as proposed by Badt [11], it suffers from the same problems due to the filter based error detection mechanism. They report that in their tests, 50-75% of the second image is inferred from the first one.

Adelson and Hodges [9] propose a technique for stereoscopic rendering similar to that of Ezell and Hodges [14] but with an improved error detection mechanism. Instead of applying a filter on the reprojected pixels, they provide a pixel classification method that reveals exact validity of reprojections. In this work, pixels are grouped into four cases based on the analysis performed on their reprojection and original positions. These cases are *good*, *missing*, *overlapping* and *bad* pixels.

The good pixel case is observed when the reprojection can be used without further calculation. For the reprojections classified as good pixels, color values are directly copied from the source image to the target image without further need for processing. The missing pixel case is encountered when there is no reprojection onto a pixel location, thus no information is available for that pixel. In this case, this pixel location is ray traced from scratch. The overlapping pixel case occurs when there are more than one pixel that reproject to a pixel location. Adelson and Hodges [9] show that if two pixels reproject to the same location, the rightmost one occludes the other(when calculating the right image from the left one). By imposing a processing order on the pixels in a row, it is guaranteed that valid reprojections overwrite the occluded ones. Bad pixel case occurs when two consecutive pixels reproject to locations with a gap

among them. The validity of reprojection of other pixels in this gap becomes questionable due to the possibility of projection of some surface in this region. Therefore, the gap between these reprojections are marked as bad pixels and ray traced from scratch.

In reprojection methods, due to the possibility that some pixels in the source image do not reproject to exact sampling positions in the target image, aliasing problems arise. Adelson and Hodges [7] report that for diffuse surfaces, this distortion is quite insignificant. However with the introduction of reflection and refraction rays, situation becomes more problematic. In reflection and refraction ray calculations, surface positions are also obtained from reprojection, hence any error in reprojection manifests itself in surface normal values used in reflection and refraction calculations. Considering that small changes in surface position may result in significant changes in surface normal(depending on surface roughness frequency), for highly reflective or refractive surfaces, aliasing problem becomes more problematic. In addition to that, even small difference in color, contour or lightness information presented to the same region on the retina can cause binocular rivalry and may break the illusion intended. As a solution, using higher resolution images is proposed as this will decrease the aliasing error.

Adelson and Hodges [7] also adapt their method for multisampled ray tracers where 9 uniform samples are used for each pixel (4 at the corners, 4 at the center of the edges and 1 at the center). For reprojection, only the center ray is used. It is reported that the cost was only 4 times the single sample case owing to the reuse of corner and edge sampling points among neighbor rows.

The sequential order requirement for row processing makes this approach not suitable (without modification) for parallel processing of pixels in a row. However since each row is independent of each other, row-wise parallelization is viable.

Later Adelson and Hodges [8] propose a method for rendering animation sequences by using reprojection techniques. The work is based on their previous work on stereo-

scopic ray tracing using reprojection [7]. In this work, animation sequences with dynamic camera and objects are supported. Animation frames are calculated from previous ones with reprojection calculations.

Es and İşler [13] provide a GPU adaptation of the method presented by Adelson and Hodges [9] for calculating stereoscopic image pairs using shading programs of the programmable graphics pipeline. Since shading programs can not perform write operations to arbitrary number of memory locations but have no problem reading from multiple memory locations via texture fetches, whole procedure is divided into two consecutive phases. In the first phase, they calculate the reprojection coordinates by rendering a screen sized quad to a scatter table using a shading program performing the reprojection calculation. In the second phase, using the scatter table created in the first phase, point or line primitives are sent to the GPU for each pixel and a gather table is created as a result. Gather table stores for each pixel the source location in the source image of the reprojection and the validity of reprojection. The primitives involved in the second phase, are ordered respecting a fixed direction along a row by the necessity of the algorithm. Three different methods are proposed based on the utilization of the GPU. Among these, completely GPU based method is reported to perform better than the others. Although the algorithm runs sequentially for a row, it provides parallel execution among rows since their calculation is independent of each other.

Nehab et al. [22] propose a method that reuses shading calculation result for animation sequences and stereoscopic rendering suitable for hardware rasterizers. Instead of reprojecting the calculated pixels to the target image, they render the geometry of the scene from the second camera using special fragment programs(instead of the actual expensive shading calculation) and apply a back reprojection transformation and validation mechanism similar to shadow map comparison to find where these pixels are in the first image. Assuming the cost of reprojection calculation is cheaper than the actual per pixel shading operations, they show that this method provides significant performance gains. Although the memory consumption is constant(proportional

to framebuffer dimensions); due to the rendering of the scene geometry from the second camera, time performance is dependent on the geometric complexity(in terms of primitive count). They mention an increase of about 57% in frame rate when this technique is applied to stereoscopic rendering.

For stereoscopic rendering of voxel based terrain data, Wan et al. [28] propose a method similar to the work of Adelson and Hodges [7] on stereoscopic ray tracing. In this work, terrain data is represented in voxel entries with opaque color values. They propose a four phase solution. First, one of the images in the stereoscopic pair is calculated. In the second phase, reprojection coordinates are calculated for the second image from the first one. Then problems are detected and finally missing and problematic pixels are calculated from scratch. In addition to previous work, an A-buffer [12] based error reduction mechanism is employed. To reduce the error introduced by non-integer reprojection positions, all the reprojection and gap calculations are performed on an A-buffer that provides a 1x8 area mask for each pixel and color values are summed over this extended buffer. Also a user defined gap threshold is provided to adjust the speed-quality trade-off.

CHAPTER 3

GPU-BASED STEREOSCOPIC REPROJECTION

In this chapter, a GPU-based solution to stereoscopic reprojection is presented. The method is based on forward reprojection procedure used by Adelson and Hodges [9] and Es and İşler [13]. However, in contrast to previous works which require a sequential processing order in a row of pixels, a parallel algorithm suitable for modern GPU architectures is proposed. Following the detailed explanation of the algorithm, implementation performed in order to realize the technique presented is described.

3.1 Stereoscopic Reprojection

The method described in this work assumes an off-axis camera system. However, any other horizontally displaced camera system can be used with minor modifications to the procedure.

In camera configurations where the optical axes of the cameras are not parallel, the reprojection transformation presented in this work becomes invalid as the optical axes are assumed to be parallel to each other. However, the one-to-one correspondance among the projections of a surface point on the image planes of the cameras still remains. Therefore, using a transformation considering the convergence of the optical axes, the proposed method can be adapted to camera configurations with varying vergence angle.

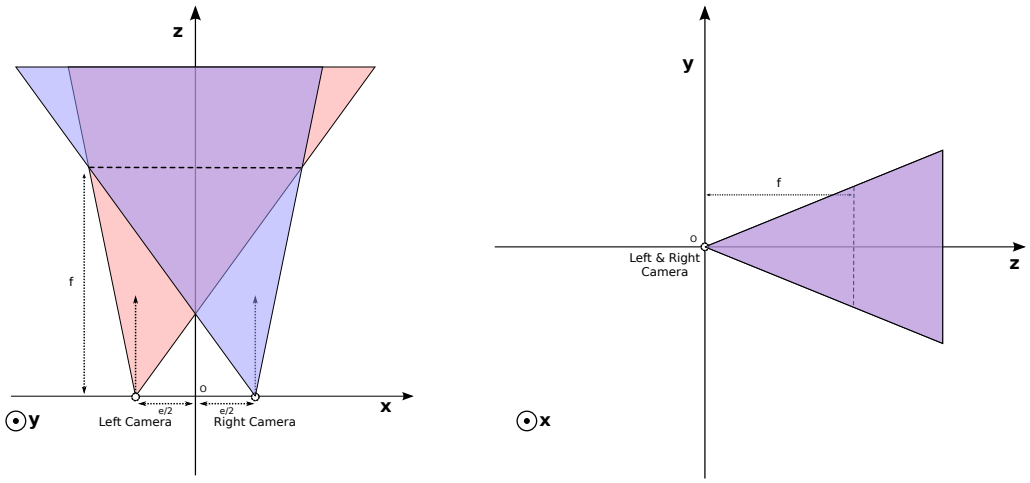


Figure 3.1: Camera configuration. Top view on the left and the side view on the right

In the following discussion about the derivation of reprojection transformation, a scene consisting of geometrical entities and an off-axis camera is assumed. For simplicity, all the spatial quantities are with respect to a left handed coordinate system, the origin of which is at the midpoint of the line connecting the optical centers of the cameras. The depiction of the configuration is shown in Figure 3.1. The distance between the cameras is called the *interocular distance* and symbolized as e. The distance from the optical centers to the image planes is expressed as the *focal distance*, f for short. In off-axis camera systems, parallax is zero at this distance, in other words, projection of a surface falls on the same location for left and right image planes. As it can be interpreted from the Figure 3.1, the z axis extends along the forward direction and the y axis advances in the upper direction of the cameras. Hereafter, the direction along which x coordinate values are increasing is treated as the right direction. Therefore, the camera the optical axis of which has smaller x coordinate is called the *left camera* and the other is called the *right camera*. Although similar derivations of the reprojection transformation used for stereoscopic rendering is available in [9] and [14], for completeness, it is presented in this section.

For a point P in the scene configuration mentioned at the beginning of this section, projection of P on the image plane is (X_l, Y_l, f) and (X_r, Y_r, f) when viewed from left

and right optical centers respectively. As illustrated in Figure 3.2, there is a geometric relation between the projection coordinates of a surface for two optical centers. Using the similarity property of triangles the difference between x coordinates of these two projections of point P, symbolized as d, is shown to be satisfying Equation 3.1.

$$d = \frac{e(P_z - f)}{P_z} \tag{3.1}$$

There is a one-to-one relationship between projections of a surface onto the image plane when viewed from right and left cameras if that surface is visible from both cameras. It is also noticed that the y coordinates and the z coordinates of the projections are the same, differing only in the x axis. Using this geometric relationship, projection of a surface point can be used in calculating the position of projection of that surface point in the other image, if that surface is visible in both images. However, not every surface visible in one image is necessarily visible in the other image due to occlusion of other surfaces and difference in the view frusta of the cameras. A surface can be visible in one camera but it can be blocked by some other surface in the other camera or a visible surface point in one camera may fall out of the view frustum of the other camera.

In stereoscopic rendering, this observation translates into the opportunity of reusing the view independent information in one image to infer some portions of the other image. This process of finding the projection position of a surface using the projection of that surface in the other image is called *reprojection*. The information about a surface point in one image should be independent of the changing parameters across the cameras in the stereoscopic system. These parameters are view direction and center of projection. Among these, view direction is frequently used in shading calculations. In materials where the final color is a function of the view direction, changes in that parameter should be not be ignored during the reprojection process. Despite the unlimited possibilities of shading calculations defining surface materials, most shading programs can be broken down into smaller subroutines so that parts that depend on

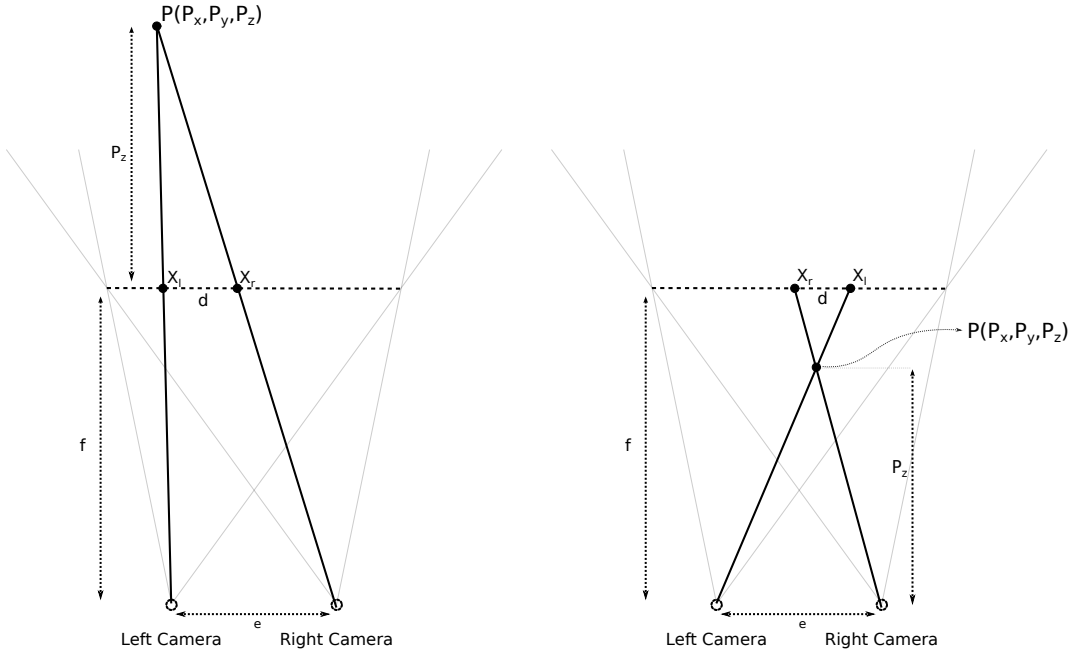


Figure 3.2: Depiction of reprojection of point P resulting in positive and negative parallax

view direction can be recalculated while other parts are reused from the source image used in reprojection.

In this work, surface materials has diffuse, reflective and refractive components the linear combination of which gives the final surface color. This model is quite similar to the model proposed by Whitted and Turner [29] but does not include specular highlights. In calculating the reflective and refractive components, view direction of the active camera is used, whereas diffuse component is independent view direction. Therefore, reflective and refractive components are calculated from scratch but the diffuse component is reused from the source image. The need for the recalculation of view dependent components makes the computation savings dependent on the properties of the surfaces. If the scene is composed of reflective and refractive surfaces, rate of reuse will be less than the case where all the surfaces have only diffuse components.

For brevity, from now on left image is treated as the source image and the right image as the inferred image, implying a left-to-right rendering order between stereoscopic image pairs. With trivial modifications, a right-to-left ordered version can be realized. Although the reprojection transformation provides a means of information reuse, it is not enough to just calculate the reprojection positions and copying color values from one image to the other. Two distinct sources of error do exist. One is due to asymmetric visibility of a surface for two cameras and the other stems from sampling errors introduced by reprojection. While details of the issues concerning sampling error is discussed in its own section later in this chapter, visibility problem is explained here with the solution proposed in a separate section.

Some of the information encoded at the pixel values in the left image may be redundant for the right image, therefore these pixels should be discarded. In addition to redundant pixels, the left image may not contain information about some of the pixels of the right image, leading to the need for calculating these pixels from scratch. These two cases are caused by the same reason, asymmetric visibility of a surface point for two distinct viewpoints. Redundant information in the left image means that the pixels in question are visible in the left image but not visible in the right image. Similarly, the missing information in the left image for some pixels in the right image means that some surface points are visible in the right image but not visible in the left image. The variation in visibility can be due to occlusion observed in one view but not in the other one or it can be caused by the difference between the viewing frusta of the cameras. The asymmetric visibility of surfaces is observed in binocular viewing systems, independent of the rendering method used.

3.2 Reprojection Validation

While the reprojection transformation does not affect the y coordinate of the source projection positions, x coordinates differ between two corresponding surface projections. Therefore, visibility problems arise only along the x axis. Therefore each row

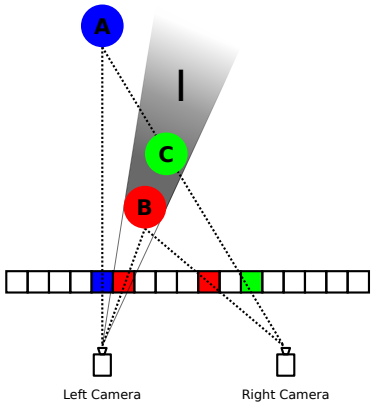


Figure 3.3: Left image has no information about the region behind B, leading to uncertainty about the visibility of A

can be treated independent of the other rows, simplifying the procedure and allowing row-wise parallelization.

The related work of Adelson and Hodges [7], as mentioned in Chapter 2, classify pixels in the source image according to their relative reprojection positions in the inferred image and by executing a sequential procedure on rows, they provide an exact visibility determination mechanism. Although this approach allows parallel execution on distinct rows, it imposes sequential processing in a single row, limiting further parallelization. Attempting to directly apply this method on parallel platforms, leads to synchronization problems among parallel execution units that need to write to the same locations.

In this work, a similar but different validation mechanism that is suitable for parallel execution(even in a row) is proposed. In the remaining of this chapter, uniform sampling with one sample per pixel is assumed. The implications of stochastic sampling with multiple samples on the quality of the method is discussed in a later section.

Given a pixel P_t in the left image, where t represents the x coordinate of the projection and its reprojection position $Rep(P_t)$; any two pixels P_t and P_{t+m}, where $m > 0$, can be classified in three distinct cases as follows:

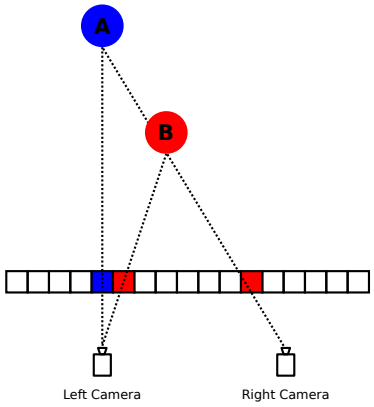


Figure 3.4: A and B reprojecting to the same pixel location.

In the first case, where $Rep(P_t) > Rep(P_{t+m})$, $Rep(P_t)$ is to the left of $Rep(P_{t+m})$ in the right image. As illustrated in Figure 3.3; in the left image, there is no information about the region behind surface projected to P_{t+m}(indicated by shaded region). Therefore, there may exist an object that project to pixel locations w, where $w > Rep(P_{t+m})$. And if there is no occluder closer to the right camera than this object, it should be visible occluding other surfaces behind it.

In the second case, where $Rep(P_t) = Rep(P_{t+m})$, two pixels in the left image reproject to the same pixel location as depicted in Figure 3.4. As shown in [9], since P_{t+m} is to the right of P_t, the surface point projected to P_{t+m} is closer to the center of right camera than the surface point projected to P_t. Therefore, reprojection of P_{t+m} should be valid and the reprojection of P_t should be discarded.

The final case is observed when $Rep(P_t) < Rep(P_{t+m})$ as visualized in Figure 3.5. Considering only these two pixels, both of the reprojections are valid. The conclusion reached when these three cases are analyzed is that for a reprojection $Rep(P_{t+m})$ of P_{t+m}, where $m > 0$, to be valid there should be no P_t, with reprojection $Rep(P_t)$ such that $Rep(P_{t+m}) < Rep(P_t)$. This observation can be equally interpretted as follows: For a reprojection of a pixel to be valid, the reprojection of that pixel should be minimum among reprojections of all the pixels to its right in a row.

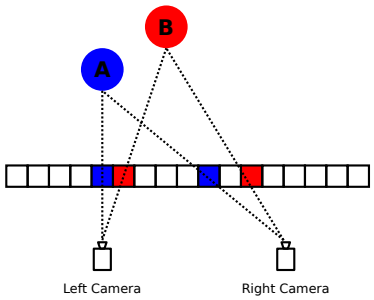


Figure 3.5: B reprojects to the right of A

3.2.1 Parallel Scan Algorithm

An inclusive scan operation is defined to take a binary associative operator $\triangle$, a list of n elements $[a_1, a_2, \ldots, a_n]$ and return a list of the form $[a_1, a_1 \triangle a_2, \ldots, a_1 \triangle a_2 \triangle \cdots \triangle a_n]$. The exclusive version, which is equivalent to the inclusive one in time and space complexity, returns a slightly different list $[I, a_1, a_1 \triangle a_2, \ldots, a_1 \triangle a_2 \triangle \cdots \triangle a_n]$, where I is the identity element of $\triangle$.

As explained by Sengupta et. al [26], a parallel version of the scan operation is available in $O(n)$, where n is the number of elements in the input list. This algorithm is also suitable for GPU architectures. The algorithm is composed of two phases, namely *up-sweep* and *down-sweep*, each of which takes $log(n)$ steps. In each successive step, the amount of work done is halved, leading to $O(n)$ operations in total. Algorithm 1 and Algorithm 2 present the pseudocode for up-sweep and down-sweep phases of the scan algorithm respectively. The CUDA implementation of the GPU based exclusive scan algorithm is explained briefly as follows:

In the up-sweep phase, a block of $n/2$ threads is run. Each thread performs a single application of the binary associative operator. At each subsequent iteration the number of running threads are halved as well as the operation count. For a list of $n = 2^k$ elements, $2^{k-1} + 2^{k-2} + \ldots + 2^0 = 2^k - 1 = n - 1$ operations are performed. Figure 3.6 illustrates the up-sweep phase of the exclusive scan operation.

In the down-sweep phase, a block of $n/2$ threads is run. $2^0 + 2^1 + \ldots + 2^{k-1} = 2^k - 1 = n - 1$ operations and $n - 1$ swaps are performed. Figure 3.7 illustrates the down-sweep phase of the exclusive scan operation. For up-sweep and down-sweep phases $3(n-1)$ operations are performed in total, leading to $O(n)$ operations.

Algorithm 1 Exclusive Scan Up-sweep

for $d = 0$ to $\log_2 n - 1$ **do**
  for $k = 0$ to $n - 1$ by 2^{d+1} in parallel **do**
    $x[k + 2^{d+1} - 1] \leftarrow x[k + 2^{d+1} - 1] + x[k + 2^d - 1]$
  end for
end for

Algorithm 2 Exclusive Scan Down-sweep

$x[n - 1] \leftarrow 0$
for $d = \log_2 n - 1$ to 0 **do**
  for $k = 0$ to $n - 1$ by 2^{d+1} in parallel **do**
    $tmp \leftarrow x[k + 2^d - 1]$
    $x[k + 2^d - 1] \leftarrow x[k + 2^{d+1} - 1]$
    $x[k + 2^{d+1} - 1] \leftarrow tmp + x[k + 2^{d+1} - 1]$
  end for
end for

Since the minimum operation is binary associative, it can be used in a parallel scan operation. Selection of a large enough value for I, the parallel exclusive minimum scan operation can be used in calculating minimum of reprojection positions to the right of each pixel. As the scan operation is required to run from right to left, the algorithm is slightly modified to serve this need. After the application of the scan operation, performing an equality check of each reprojection position with the corresponding element in the result of the scan operation yields whether that reprojection position is minimum among reprojections of all other pixels to the right of it in the source image. Both the scan operation and the equality check steps run in parallel for each pixel.

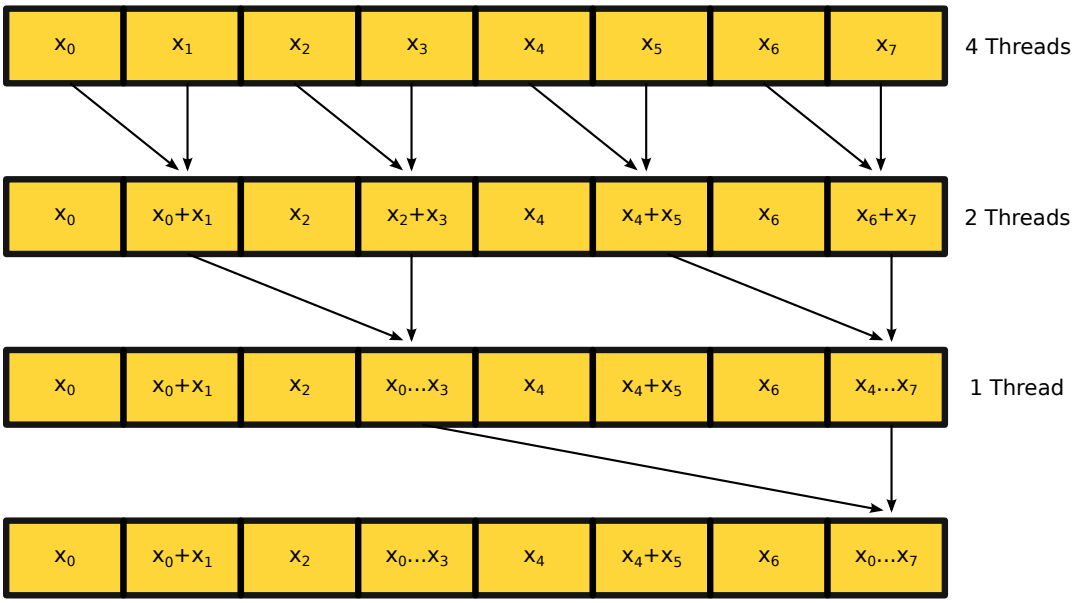


Figure 3.6: Illustration of up-sweep phase of the exclusive scan operation for '+' function for input list x with 8 elements. $x_k...x_l$ represents application of '+' operation for element $x_k + x_{k+1} + \cdots + x_l$

3.3 Error in Reprojection

Discrete representation of images leads to information sampling at points that are distributed according to a sampling method. Among deterministic sampling methods, uniform sampling is frequently preferred due to its minimal computational demands. In uniform sampling, sample points are uniform for each pixel. In stochastic sampling methods, to better represent the population sampled, unbiased random distributions are favored.

Careful inspection of the reprojection transformation reveals that reprojection offset is a function of only depth in a camera configuration as the interocular and focal distances are constant for reprojection transformation during the generation of a single stereoscopic image pair. Furthermore, as depth increases, reprojection offset grows nonlinearly until it converges to e at infinity. Figure 3.8 shows the relationship of deviations of reprojection offset from uniform sampling points with respect to depth

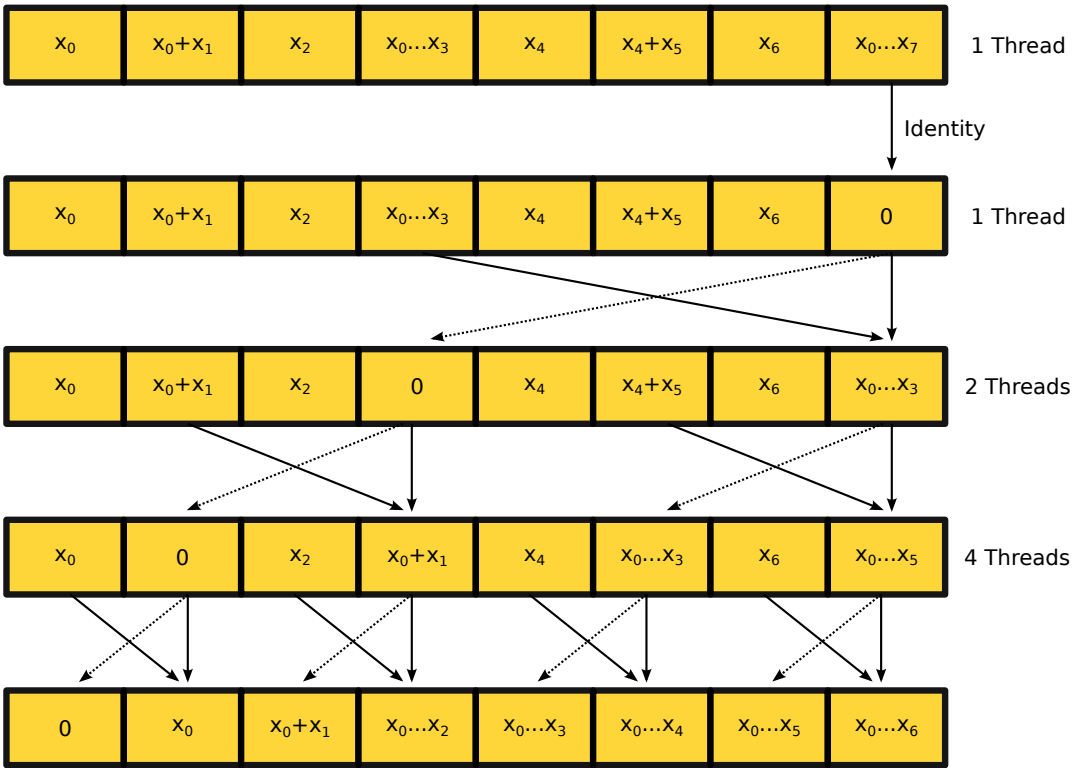


Figure 3.7: Illustration of down-sweep phase of the exclusive scan operation for '+' function for input list x with 8 elements. $x_k...x_l$ represents application of '+' operation for element $x_k + x_{k+1} + \cdots + x_l$

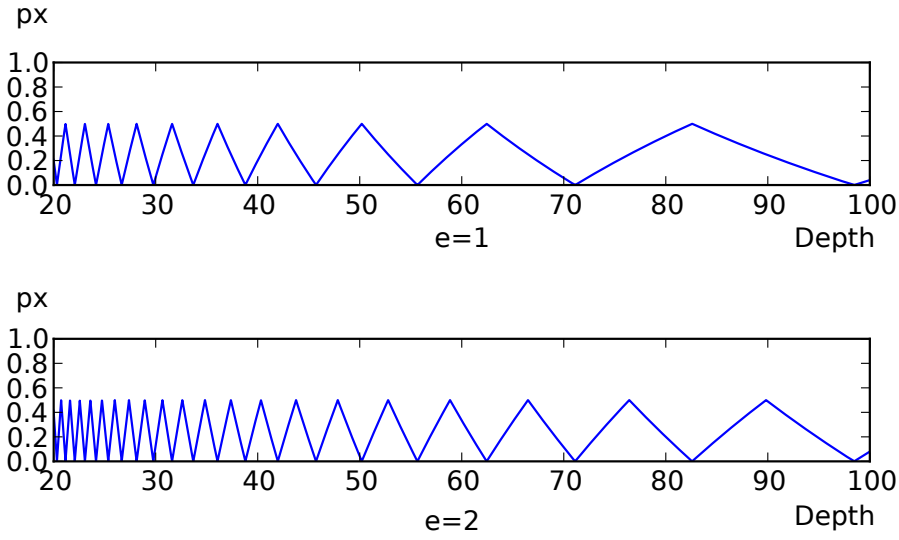


Figure 3.8: Deviation from uniform sampling points(pixels center in this case) with respect to depth for e=1 and e=2

for fixed values of e and f. The observed deviations are encountered due to the fact that the reprojection transformation is independent of the underlying sampling procedure. The distribution of the deviations varies depending on the sampling method employed by the renderer. For the uniform sampling case, deviation from a sampling point is observed to be an oscillating function of depth. In stochastic cases, a bias is expected to be introduced since the transformation function is deterministic in contrast to the sampling function. Figure 3.9 illustrates the observed deviations from uniform sampling points during the reprojection procedure.

Adelson and Hodges [7] report that even though sampling errors are insignificant for diffuse surfaces, they are more problematic for reflective and refractive surfaces. Inference of surface normals via reprojection introduces errors in normal vectors. But on contrary to diffuse color values, normal values are used in further ray tracing calculations where small changes in direction can dramatically change the surface intersected. To reduce errors due to the deviation from sampling points Wan et. al [28] propose a 8-by-1 A-buffer. By incorporating an eight times more resolution, values reprojecting to the same pixel contribute with more chance to the final pixel color as

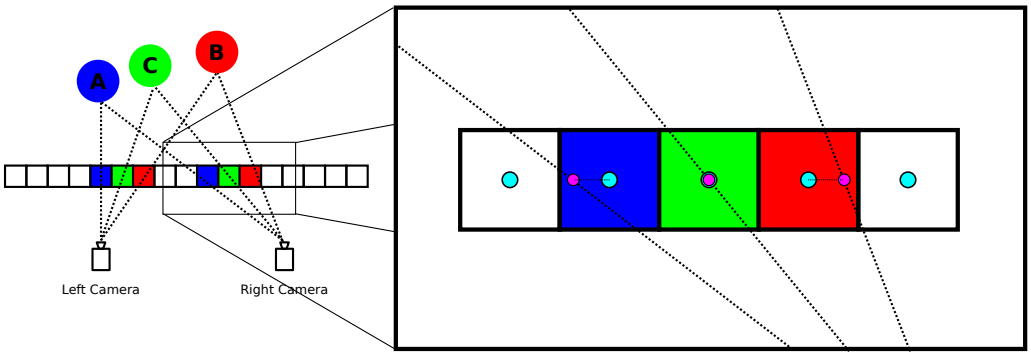


Figure 3.9: Reprojection of surfaces projected on sampling points may deviate from the sampling points of the target image. On the right of the figure a close-up portion of the image plane is presented to emphasize deviations. Cyan circles indicate the actual sampling points and magenta circles indicate the reprojection positions.

they have less chance of being occluded by other surfaces that reproject to the same pixel. However, in pixels that take only one reprojection, error due to sampling is still in effect at the same magnitude.

In uniform sampling case where pixels centers are selected as the sampling points, reprojection coordinates deviate from the sampling points only in the x axis as y axis is not affected. However, in stochastic sampling cases, the sampling positions of corresponding pixels(according to reprojection transformation) may be different depending on the sampling distribution. Since the reprojection transformation assumes the y coordinate does not change, deviations from the sampling points in the y axis is expected, leading to degradations in the image quality. In this work, only uniform sampling case is analyzed.

3.4 Implementation

To realize the method explained in this chapter, a GPU-based stereoscopic ray tracing system is developed. To see the result integrated in a generic GPU-based ray tracer, the OptiX is used in ray tracing calculations. In other parts of the algorithm, CUDA

is selected to perform GPU calculations. Remaining parts of the implementation that run on the CPU is written in C++.

In the rendering system, every frame is independent of each other. In other words, no information from previous frames are transferred to the current frame rendered. Although that is possible via GPU adaptations of the method described in [8], it is out of the scope of this work. In a single frame, two images from the stereoscopic cameras are rendered. In this implementation, first the left image is rendered from scratch and then parts of the right image is reprojected from the left image and finally the missing parts in the right image are filled via ray tracing only the necessary parts.

In calculation of the left image, in addition to the final color information; diffuse color, depth, normal values and material types for each surface projected on a pixel are stored for later use. Diffuse color buffer is needed since final color and diffuse components of surfaces may differ if the scene contains reflective or refractive surfaces. Considering the existence of view dependent surface properties; if diffuse color is not stored separate from the final color buffer, there remains no way to extract the diffuse component from the final color. For scenes without any view dependent surface properties, the need for a separate buffer vanishes.

Depth values are used in reprojection transformation to determine which projections reproject to which locations. Normal values are stored to be used in calculation of view dependent components of surface materials. Material type values are used in finding which material a projected surface has as surfaces may have different reflective and refractive properties. Depth values are also used in determination of the positions of projected surface points in the scene which will be the origin of the rays for secondary rays used in reflection and refraction calculations. To summarize, four distinct 2D buffers, each with the same resolution as the color buffer is stored in the GPU memory, so that information at an index (x, y) in each buffer corresponds to the related information about the same pixel. Using the depth values and camera parameters a CUDA kernel that performs reprojection transformation is run on the

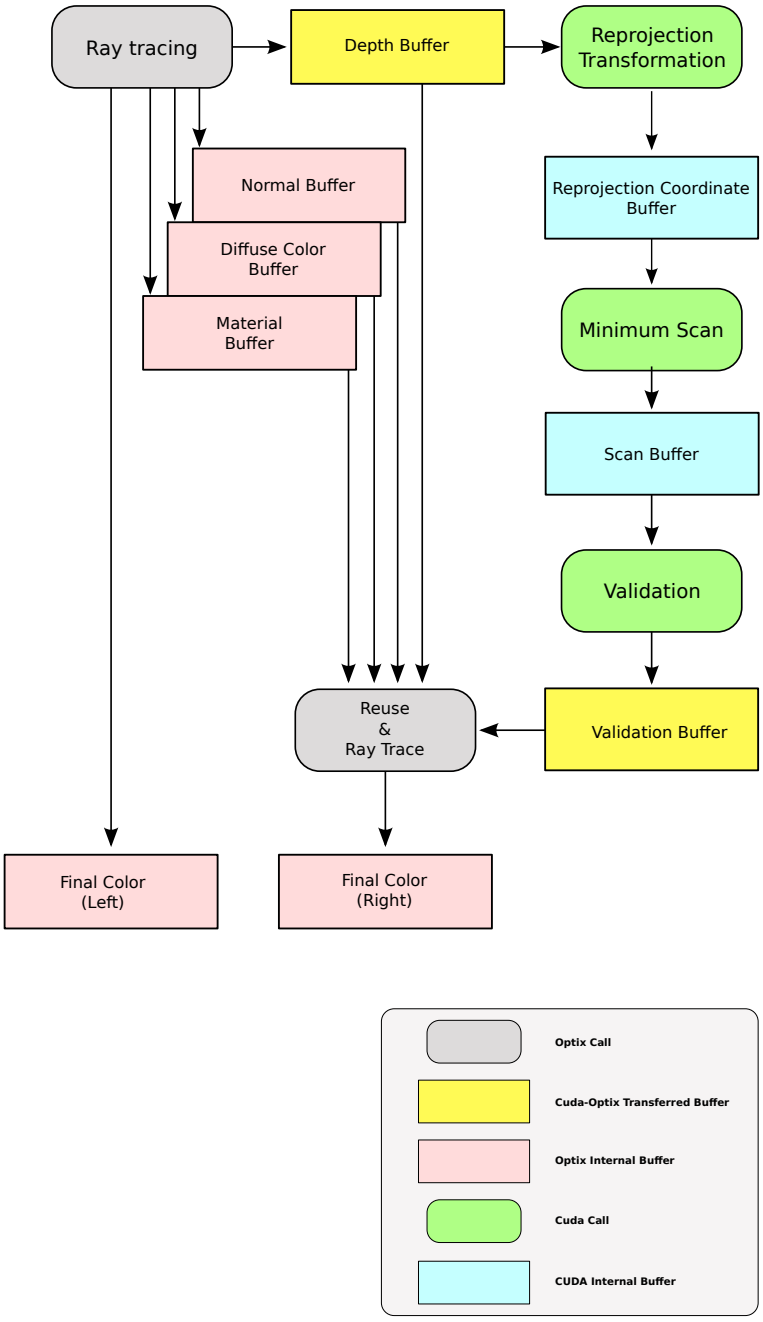


Figure 3.10: Major data and execution components of the system that reside on the GPU memory

depth buffer. Results of reprojection transformation are stored in reprojection position buffer which is of the same resolution as the depth buffer. The entry at (x, y) index of the reprojection position buffer stores how much pixel locations the projection of the pixel at (x, y) should shift in the horizontal axis of the buffer. Figure 3.10 shows the relationship among buffers and procedures performed throughout the whole process from starting the left image to finishing the right image.

As previously mentioned, the values resulting from the reprojection transformation may contain visibility errors, therefore validation step is performed on the reprojection position buffer. For each row of the buffer, a modified parallel minimum scan operation is executed resulting in a separate buffer called *scan result buffer*. An entry at (x, y) contains the minimum values among all the values among $(x + 1, y)$ and (x_{max}, y), where x_{max} is the horizontal size of the buffer. In the case of $x + 1 = x_{max}$, it holds the identity value, I, for the minimum scan operation, which is an arbitrarily selected constant value greater than x_{max}.

Subsequent to the scan operation, a separate CUDA kernel that compares values at (x, y) index of reprojection positions with result of the scan operation at the same index is executed. If the reprojection position in question is less than the corresponding result of the scan operation, that means that reprojection position value is the minimum among all the other reprojection position values to the right of it, hence it is visible from the right camera. To store the result of this validation step, a separate *validity buffer* is used, entries of which contain source x coordinates(-1 for invalid reprojections) to indicate the validity of the corresponding reprojection transformation.

After determination of the validity of reprojections, a slighly different version of the ray tracer used for calculating the left image is executed. First by consulting the *validity buffer* calculated previously in CUDA, validity of the current pixel location is decided. If reprojection is invalid then ray tracing is performed as if no reprojection is performed. However, if the reprojection in question is valid, diffuse color information,

normal and depth values along with material type information are retrieved from the *diffuse color buffer*, *normal buffer*, *depth buffer* and *material type buffer* respectively and stored locally to be used by subsequent instructions. If the material identified by the material type contains reflective or refractive properties, further ray tracing is performed. However, surface position and normal are required for reflection and refraction calculations. Even though normal is available, there is no explicit surface position. Therefore using the depth value and current sampling point, the surface position in the scene is reconstructed. Having all the required information for the calculation of reflective and refractive components of the surface point, ray tracing is performed for secondary rays and the resulting material components are combined with the diffuse component reused from the left image. The final color result is stored in a separate buffer for visualization. It should be noted that in this step no output other than final color buffer is produced.

OpenGL buffer objects are used to map buffers among OptiX and CUDA. Only depth and validity buffers are exchanged between OptiX and CUDA. Reprojection position and scan buffers are internal to the CUDA context, similarly normal, material and diffuse color buffers are internal to the OptiX context.

CHAPTER 4

RESULTS AND DISCUSSION

This chapter presents the results of the test procedures performed to analyze performance and quality aspects of the method proposed in Chapter 3 .

All the tests are performed on the same hardware configuration. The CPU used is an Intel Core i7-3930K with 6 cores each running at 3.20GHz. The GPU used is an NVIDIA GeForce GTX 570 with 480 cores. All the CPU-based algorithms are run on a single core of the CPU.

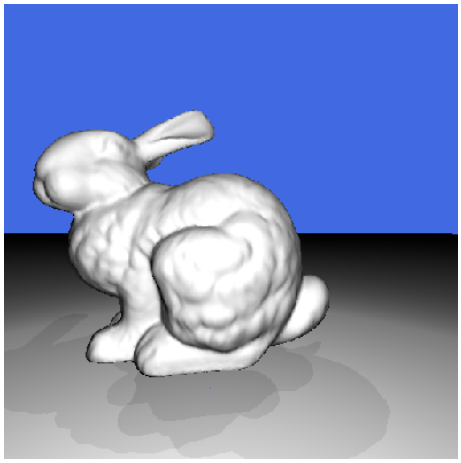
Five scene configurations with different characteristics related to the values measured are used in the experiments. The first scene consists of a single Stanford Bunny [6] polygonal mesh positioned on a quadrilateral. All the materials in this scene are perfectly diffuse surface with no texture. The second scene is the same as the first scene but with reflective materials with no texture. The reflectivity of the surfaces is 30%. The screenshots from the first and the second scenes are shown in Figure 4.1(a) and Figure 4.1(b) respectively. The third scene consists of four Dragon models [6] and a quadrilateral. Similar to the first scene, all the models in this scene have a perfectly diffuse material with no texture. The fourth scene is the same as the third one but instead of a diffuse material, a 30% reflective material is used. Figure 4.2(a) shows a screenshot from the third scene and Figure 4.2(b) provides a screenshot from the fourth scene. The final scene consists of a texture mapped diffuse sphere, a 30% reflective cube model with no texture map and a diffuse ground mesh with a checker-

board texture. This scene is only used in quality analysis tests since reprojection validation is independent of the surface color features. Figure 4.3 shows a view from this last scene. In all the scenes except for the last one, all the polygonal models are elevated from the ground. From now on, the scenes from the first to the last are called Bunny Scene, Reflective Bunny Scene, Four Dragons Scene, Reflective Four Dragons Scene and Textured Primitives Scene respectively. In all the scenes, the camera configuration is fixed to have focal length of 5 units and interocular distance of 0.4833 units. The scenes with the same geometry are rendered from cameras with the same coordinate frames to better compare the effect of the varying scene property among scenes. All the scenes are populated with four point lights that are positioned at the same height above the models and separated from each other , satisfying the contraint of lying on a circle with a radius of 5 units.

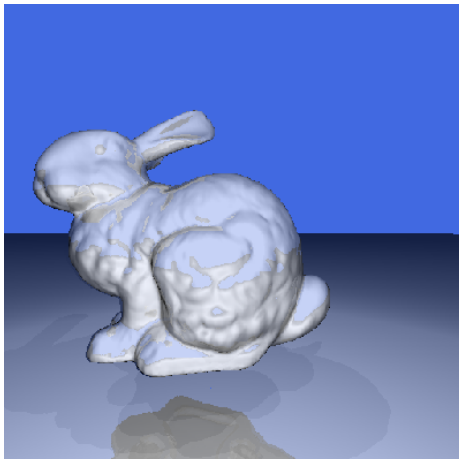
Each scene is selected according to its certain properties. The Bunny Scene has little variation in its depth compared to the Four Dragons Scene but has the same diffuse materials. The Reflective Bunny Scene has reflective properties which Bunny Scene does not have but has the same polygonal models. The same relation holds between Reflective Four Dragons Scene and Four Dragons Scene. The Textured Primitives Scene has texture mapped models.

4.1 Performance Tests

The runtime performance of the algorithm presented is compared with the CPU-based reprojection algorithm of Adelson and Hodges [7] for different resolutions for every scene mentioned previously. In both running of the algorithms, the same OptiX-based ray tracer is used. Since the CPU-based algorithm is not suitable for the GPU architecture and the ray tracer is GPU-based, in contrast to the original method, which reprojects while ray tracing, a three phased approach is adopted. First, the ray tracer calculates the left image extracting the required information(depth, diffuse color, normal values), then the reprojection algorithm runs on the CPU producing a reprojection

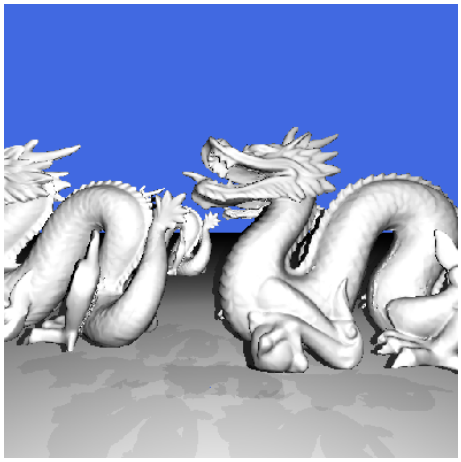


(a) Bunny Scene

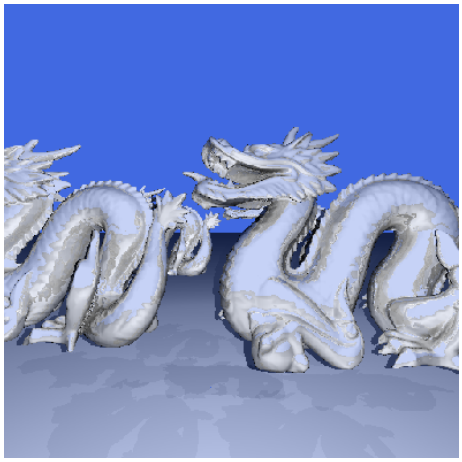


(b) Reflective Bunny Scene

Figure 4.1: Two scenes with the same Stanford Bunny model with different materials



(a) Four Dragons Scene



(b) Reflective Four Dragons Scene

Figure 4.2: Two scenes with the same Dragon model with different materials

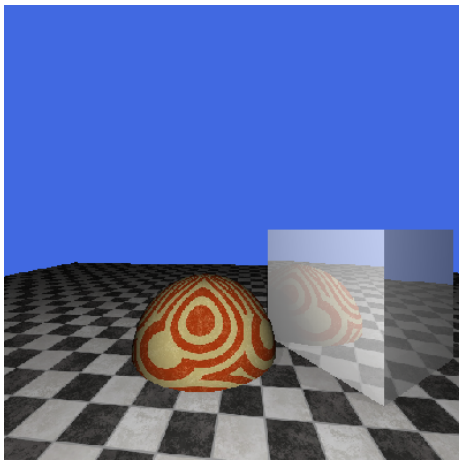


Figure 4.3: Textured Primitives Scene

validity buffer and finally the ray tracer uses the reprojection buffer output of the reprojection algorithm and calculates the right image. Transfering data between the CPU and the GPU address spaces incurs an overhead which the original fully CPU-based algorithm avoids. Therefore, during the comparison between the CPU method and the proposed method, which runs fully on the GPU, only the reprojection and validation calculations are compared.

4.1.1 Reprojection Speed-up

The amount of time reprojection transformation and validation calculations take is measured for the previous CPU method and the GPU-based proposed method. The measurement results are summarized in Table 4.1.

The test results show that, the CPU-based method is slightly affected by the scene content as the number of memory write operations(when marking bad pixels) is dependent on the depth of the left image. For the GPU-based reprojection method, scene content affects the performance less since there is less divergence in the flow based on the depth of the left image.

Table 4.1: CPU vs GPU Reprojection Performance

Scene	Resolution	CPU (ms)	GPU (ms)	Speed-up
Bunny	512x512	6.3333	0.1713	36.9823
	1024x1024	26.0000	0.7540	34.4835
	2048 x2048	103.3330	4.8603	21.2604
Reflective Bunny	512x512	6.0000	0.1716	34.9683
	1024x1024	26.0000	0.7537	34.4952
	2048 x2048	103.3330	4.8584	21.2690
Four Dragons	512x512	7.0000	0.1713	40.8573
	1024x1024	27.0000	0.7542	35.8012
	2048 x2048	106.6670	4.8565	21.9637
Reflective Four Dragons	512x512	7.0000	0.1716	40.7861
	1024x1024	27.3333	0.7540	36.2528
	2048 x2048	106.6670	4.8586	21.9544

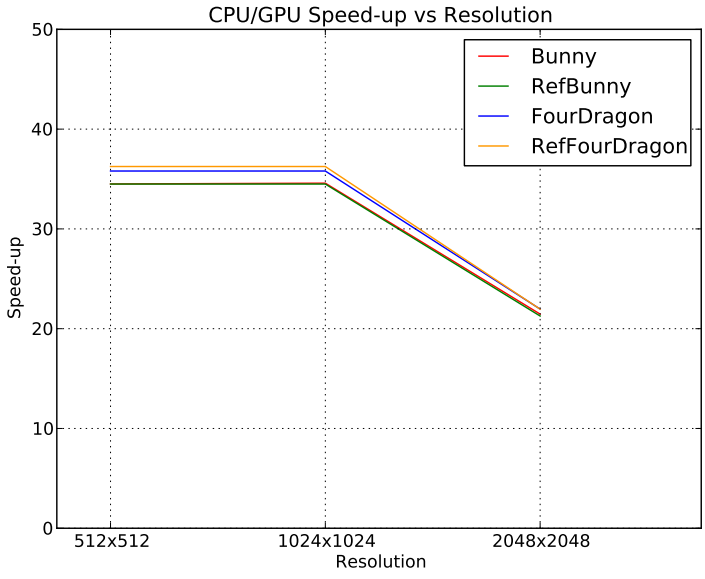


Figure 4.4: GPU vs CPU Speed-up

The speed-up achieved by the GPU-based method is more than 20 times in all of the tests. Figure 4.4 visualizes this observation. It is noticible that in 2048x2048 resolution the speed-up falls about 33% although it is nearly the same for 512x512 and 1024x1024 resolutions. In 2048x2048 resolution, CUDA blocks each with 1024 threads run to perform the exclusive scan operation on 2048 entries in the rows of the reprojection position buffer. Since a single block runs for a single row, 2048 blocks each with 1024 threads run in total. The compute capability of the GPU(which is 2.0) used in the tests can at most utilize about 67% of its processing power for thread blocks with 1024 threads but it can fully utilize its potential for blocks with 256 and 512 threads. In 512x512 and 1024x1024 resolutions, blocks with 256 and 512 threads for each row are executed respectively. Therefore, the decrease in the speed-up seems to be due to this limitation. In GPUs with compute capability 3.0 or higher, 100% utilization for the same configuration is expected as it is indicated by the CUDA Occupancy Calculator that blocks with 1024 threads can run with full utilization [2].

4.1.2 Overall Performance Gain

To analyze the performance gain achieved by the proposed method, the test scenes are first rendered for left and right cameras from scratch and then compared with the reprojection based proposed method which first renders the left image and calculates the right image by using the information extracted from the left image. The results and the performance increase is summarized in Table 4.2. *Scratch* field shows the time spent on ray tracing from scratch and *reuse* field expresses the time spent on ray tracing with the proposed GPU-based method. Measured time results for each scene are provided in Figure 4.6, Figure 4.7, Figure 4.8 and Figure 4.9.

The performance increase with reprojection when rendering stereoscopic image pairs with respect to rendering without reprojection is visualized in Figure 4.5 based on the data provided in Table 4.2. As shown in Figure 4.5, in Bunny and Four Dragons Scenes, ray tracing with GPU-based reprojection achieves more than 34% time reduc-

Table 4.2: Performance Gain

Scene	Resolution	Scratch(ms)	Reuse(ms)	Gain(%)
Bunny	512x512	24.400	16.000	34.426
	1024x1024	72.000	43.000	40.277
	2048 x2048	239.000	141.667	40.725
Reflective Bunny	512x512	46.400	37.000	20.258
	1024x1024	138.800	107.600	22.478
	2048 x2048	440.333	343.667	21.952
Four Dragons	512x512	47.400	30.600	35.443
	1024x1024	147.000	90.000	38.775
	2048 x2048	462.333	286.000	38.139
Reflective Four Dragons	512 x512	134.400	117.200	12.797
	1024x1024	421.000	365.800	13.111
	2048x2048	1354.000	1181.333	12.752

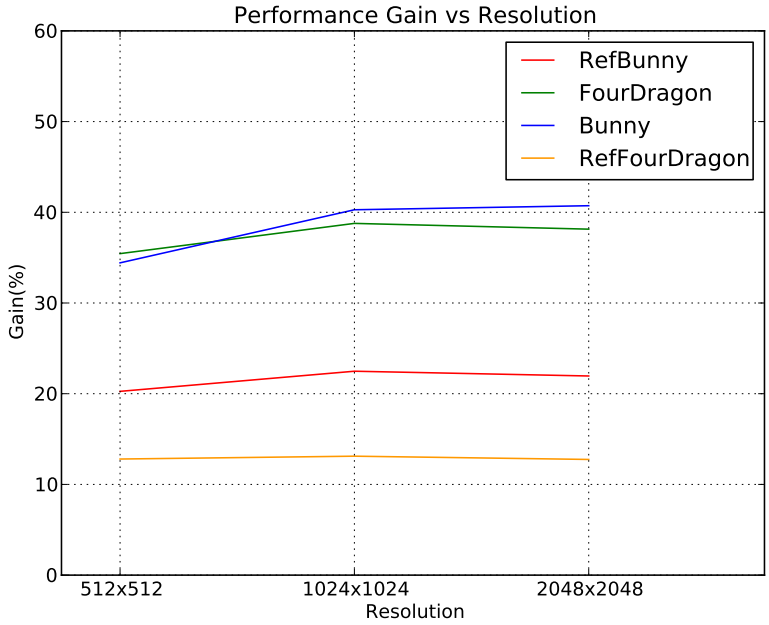


Figure 4.5: Performance Gain when GPU-based Reprojection is used

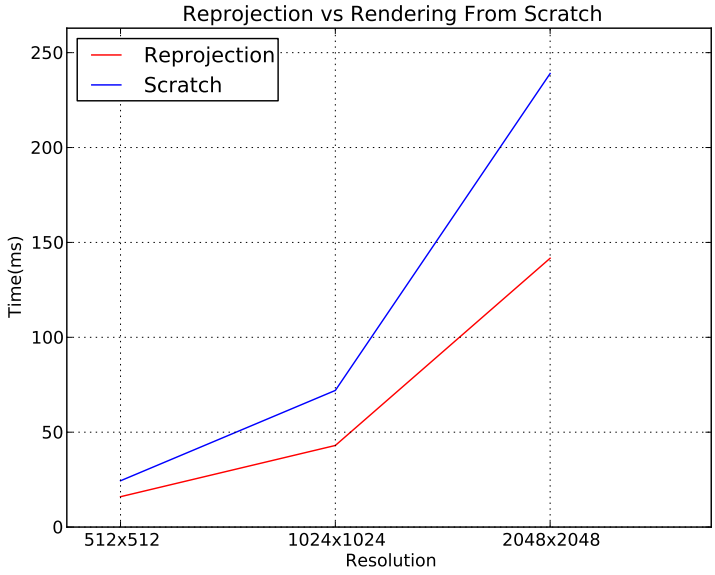


Figure 4.6: Ray Tracing from Scratch vs Ray Tracing with GPU-based Reprojection in Bunny Scene

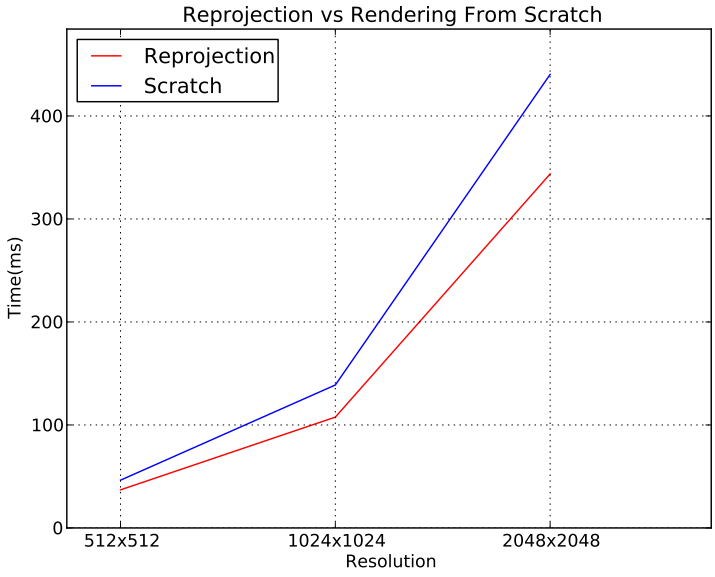


Figure 4.7: Ray Tracing from Scratch vs Ray Tracing with GPU-based Reprojection in Reflective Bunny Scene

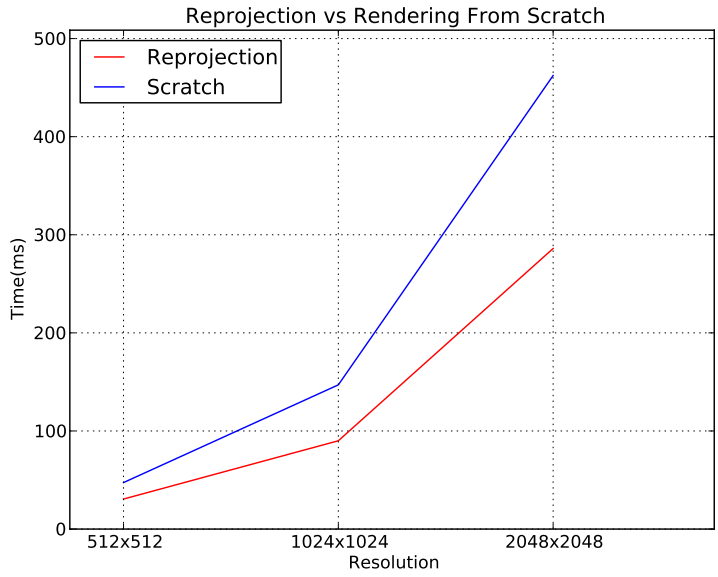


Figure 4.8: Ray Tracing from Scratch vs Ray Tracing with GPU-based Reprojection in Four Dragons Scene

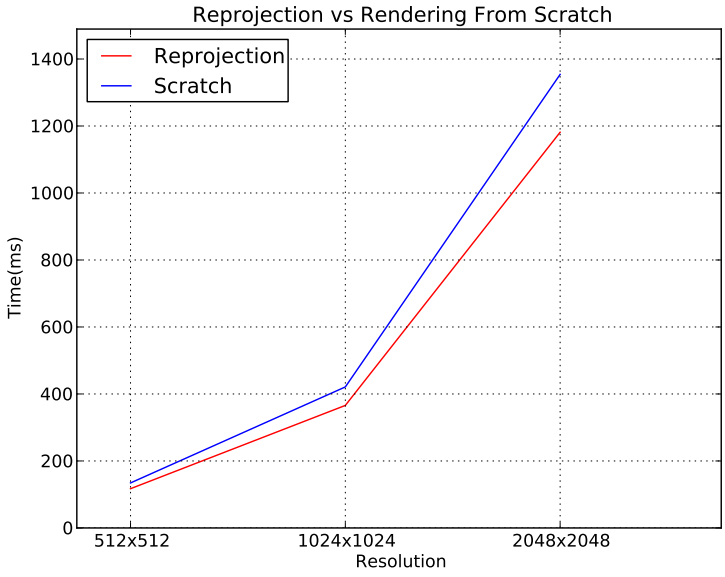


Figure 4.9: Ray Tracing from Scratch vs Ray Tracing with GPU-based Reprojection in Reflective Four Dragons Scene

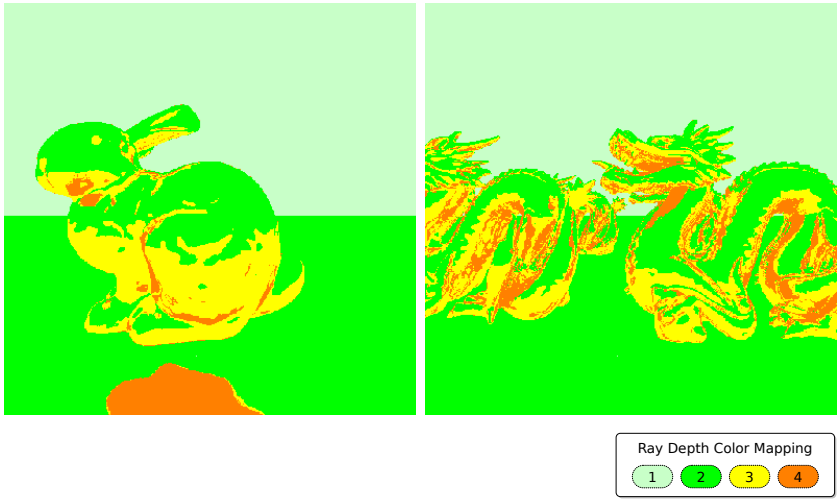
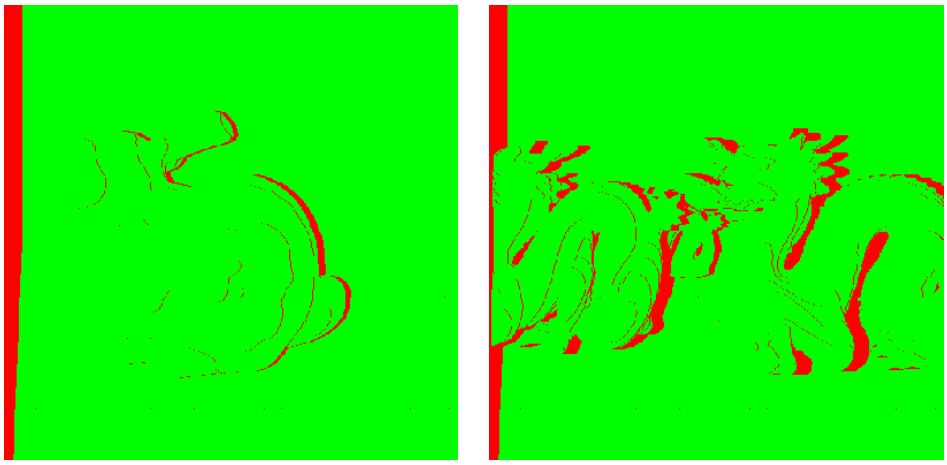


Figure 4.10: Ray depth visualization for Reflective Bunny and Reflective Four Dragons Scenes. The number of rays(excluding shadow rays) shot for each pixel are indicated by color coding which is provided in the lower right corner.

tion when compared to ray tracing from scratch. However in the other scenes, time reduction is lower than 23% for all resolutions. One of the main factors affecting the reprojection time performance is the number of reflection and refraction rays since only the first level ray intersections are used in the reprojection method and the rays that are spawned after the first level rays are calculated from scratch. In Bunny and Four Dragons Scenes, all the surfaces are perfectly diffuse whereas other scenes have reflective surfaces. Figure 4.10 visualizes the total number of rays spawned for each pixel in Reflective Bunny and Reflective Four Dragon Scenes. Since scenes with only diffuse surfaces have the same number of rays(excluding shadow rays) spawned for each pixel, they are not visualized in Figure 4.10.

The rate of successful reprojections to the total number of rays spawned is provided in Table 4.3. *Reprojection ratio* field of Table 4.3 indicates the percentage of successful reprojections when compared with the total number of rays(excluding shadow rays) and *max ratio* field indicates the maximum possible reprojection ratio for that image. The cases where max ratio is lower than 100% are the scenes with reflective surfaces as new rays need to be spawned for reflective surfaces. As visualized in Figure 4.12,



(a) Reprojection validity image for Bunny and Reflective Bunny Scenes

(b) Reprojection validity image for Four Dragons and Reflective Four Dragons Scenes

Figure 4.11: Reprojection validity images for Bunny and Four Dragons Scenes. Red pixels represent invalid or missing reprojections whereas green pixels represent successful reprojections.

Bunny and Dragon Scenes have high rate of successful reprojections due to their diffuse-only reflectance properties, whereas the reflective versions of these scenes have lower rate of successful reprojections since the total number of rays spawned is higher.

The reprojection validity buffer is visualized in *validity images* to see which pixels are successfully reprojected. Since only first-hit ray intersections are used in reprojection calculations, validity images of reflective and diffuse-only versions of the same scenes are the same. Therefore, only one image is presented for the scenes differing only in materials. Figure 4.11(a) and Figure 4.11(b) show validity images for Bunny and Four Dragons Scenes respectively. Green pixels represent successful reprojections and red pixels indicate pixels the color value of which need to be calculated from scratch.

During ray tracing with reprojection, percentage of time spent for each major component is analyzed in Figure 4.13. This components are ray tracing the left image while extracting extra information for reprojection calculation, reprojection calculation and ray tracing the right image using the results of reprojection calculation. Inspecting

Table 4.3: Successful Reprojection Rate

Scene	Resolution	Reprojection Ratio(%)	Max Ratio(%)
Bunny	512x512	94.4748	100.0000
	1024x1024	94.5627	100.0000
	2048x2048	94.5183	100.0000
Reflective Bunny	512x512	53.0628	56.1661
	1024x1024	53.1174	56.1716
	2048x2048	53.0910	56.1701
Four Dragons	512x512	91.0980	100.0000
	1024x1024	91.1456	100.0000
	2048x2048	91.1115	100.0000
Reflective Four Dragons	512x512	47.5941	52.2449
	1024x1024	47.6233	52.2497
	2048x2048	47.6015	52.2453

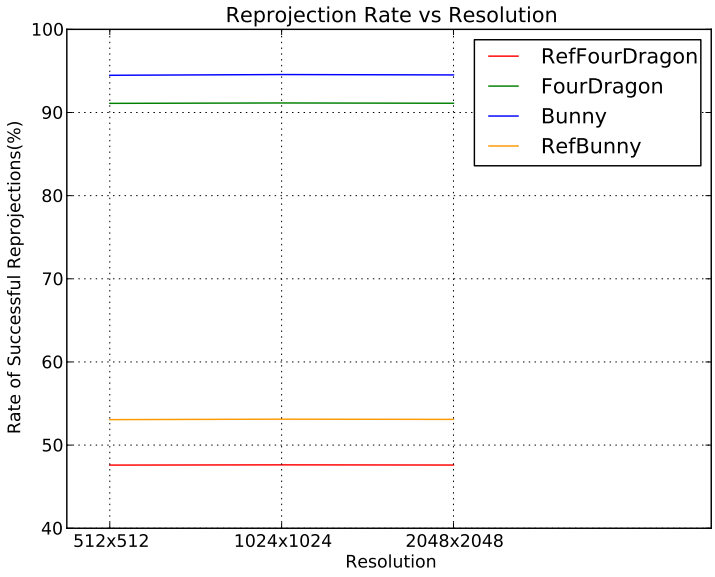


Figure 4.12: Successful Reprojection Rates for test scenes

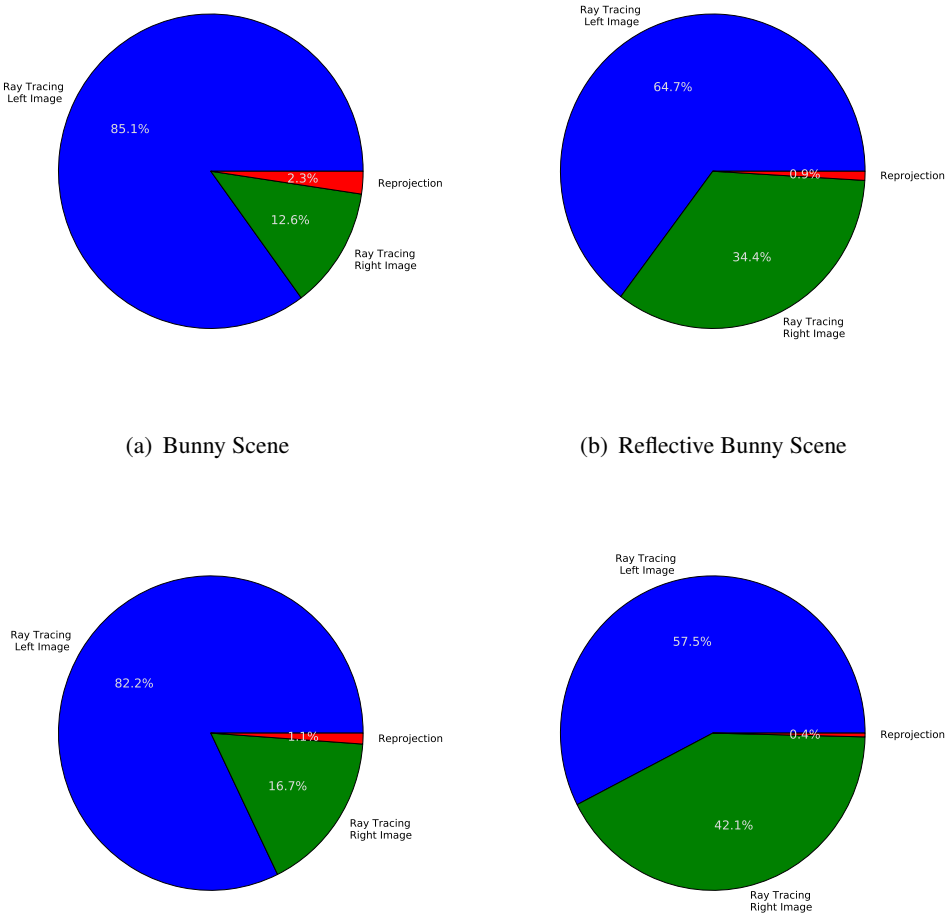
Figure 4.13 , it is observed that as the rendering cost(based on the geometric scene complexity and the materials) increases, the percentage of time spent on the reprojection calculation with respect to the rest of the process decreases.

4.2 Quality Tests

As previously discussed, the reprojection method introduces errors due to deviation from sampling points. Since the ray tracer used in this work is based on uniform single point sampling for each pixel location, the output of the ray tracing with reprojection is expected to be the same as the second stereoscopic image(the right image) calculated from scratch if no error is introduced. Therefore, the right image calculated without reprojection is selected as the ground truth reference for the result of ray tracing with reprojection. The error in pixel color values is calculated by measuring their Euclidean distance from the corresponding color value in the reference image in the RGB color space. The sum of the error values is divided by the total number of pixels in the final image to give the average error for comparison among different image resolutions and scenes. The comparison of average error introduced by reprojection is given in Figure 4.14.

It is observed that the average error decreases as the resolution of the image increases for all the scenes. This observation results from reduced deviation from sampling points as a pixel may reproject to a location away from the nearest sampling point by at most half of a pixel width. As resolution increases, pixel width decreases along with the average error introduced by the reprojection method.

It is also noticible that at the same resolution, reprojection method applied to the reflective versions of scenes results in more error compared to the application to their diffuse-only counter-parts. For reflective and refractive surfaces, normal and depth values are also transferred to the right image in addition to diffuse color. As discussed in earlier chapters, small variations in normal value may result in large variations in



(a) Bunny Scene

(b) Reflective Bunny Scene

(c) Four Dragons Scene

(d) Reflective Four Dragons Scene

Figure 4.13: Distribution of Time Spent in Ray Tracing with GPU-based Reprojection

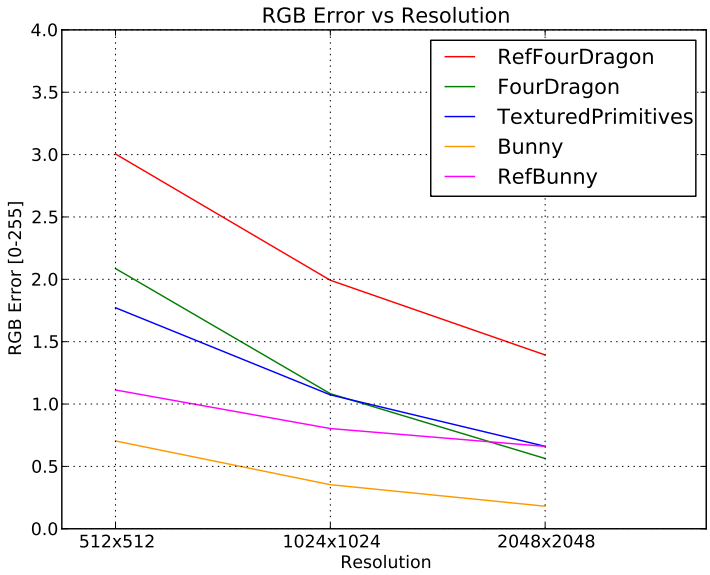


Figure 4.14: RGB error introduced by reprojection for all scenes

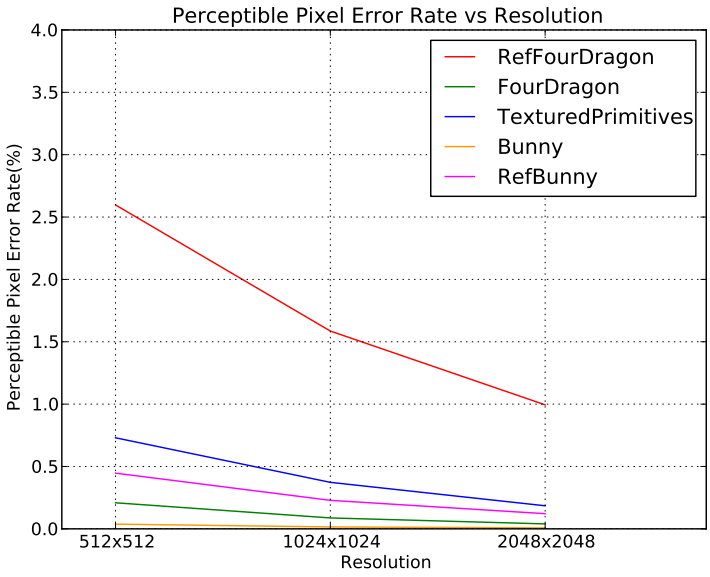
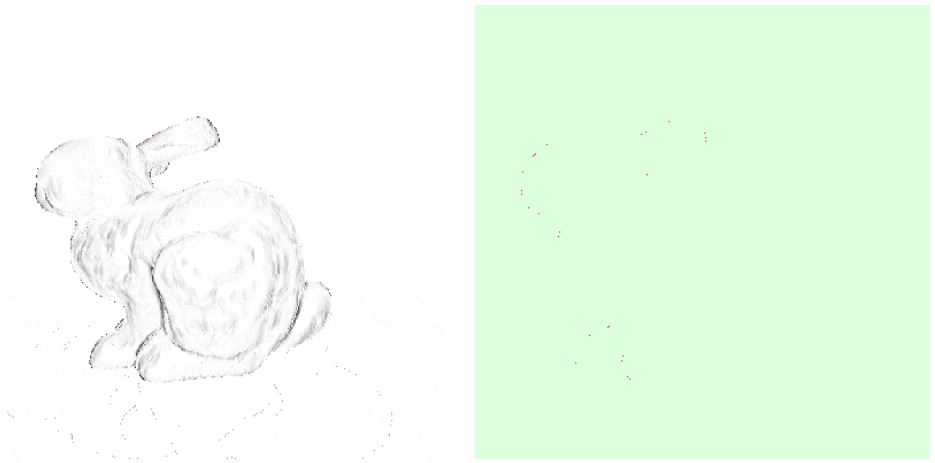


Figure 4.15: Percentage of pixels which are perceptible as being in error by reprojection for all scenes

color values. Therefore, it is not surprising to see that reflective surfaces introduce extra error in addition to the error in diffuse color. To compare the error variation between reflective and diffuse versions of Bunny and Dragon Scenes, difference between the final color images calculated by reprojection and calculated from scratch are stored in *color difference images*, each pixel of which has three color channels for the error in each channel. To make it more suitable for viewing and error inspection, the images are inverted and color values are remapped. The darker a pixel is, the more error is present.

Although the difference in the RGB color space gives information about the similarity between two images, that color space is not perceptually uniform. In other words, the response of the human visual system to color differences are not the same across the RGB color space. To evaluate the significance of the difference among images better, a perceptual difference analysis tool proposed by Yee and Newman [30] is used. As Figure 4.15 visualizes, the percentage of pixels perceived as being different from the reference image is lower than 3% in all the scenes for all the resolutions. The Reflective Four Dragons Scene shows a higher rate of perceptible pixel errors. This situation seems to result from the high variation in surface normal values and reflectivity of the objects. After the Reflective Four Dragons Scene, the Textured Primitives Scene exhibits a relatively higher rate of perceptible pixel errors which is believed to be due to the high amount of sharp color changes present in the scene. As expected, the scenes with only diffuse surfaces without any texture mapping has the least rate of perceptible pixel errors. This mainly stems from the smoothly varying radiance values of the objects in the scene. Similar to the relationship of the RGB space average error to image resolution, the rate perceptible pixel errors decreases as the resolution increases.

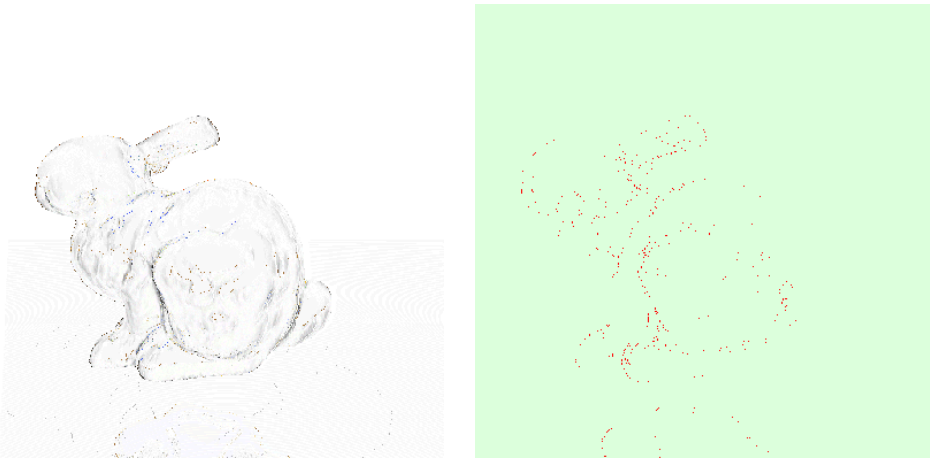
The color difference images for Bunny Scenes are presented in Figure 4.16(a)and Figure 4.17(a), and images for Four Dragons Scenes are given in Figure 4.18(a) and Figure 4.19(a). Along with the color difference images in the RGB color space, perceptibly different parts are marked in *perceptible difference images*. The pixels that



(a) Bunny Scene RGB Error

(b) Bunny Scene Perceptible Pixel Errors

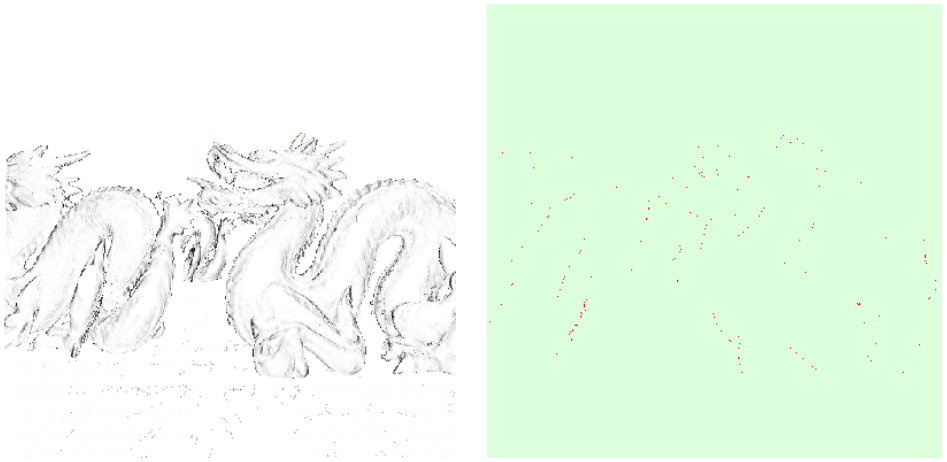
Figure 4.16: RGB error and perceptible error visualization of Bunny Scene



(a) Reflective Bunny Scene RGB Error

(b) Reflective Bunny Scene Perceptible Pixel Errors

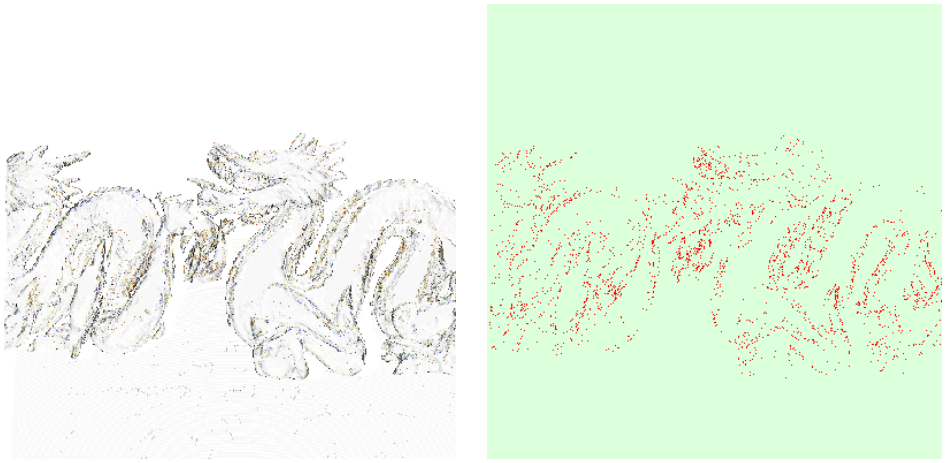
Figure 4.17: RGB error and perceptible error visualization of Reflective Bunny Scene



(a) Four Dragons Scene RGB Error

(b) Four Dragons Scene Perceptible Pixel Errors

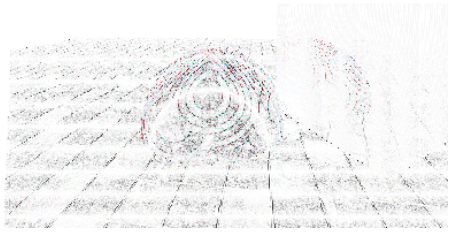
Figure 4.18: RGB error and perceptible error visualization of Four Dragons Scene



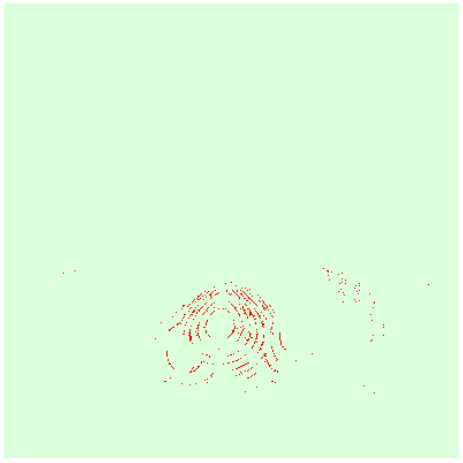
(a) Reflective Four Dragons Scene RGB Error

(b) Reflective Four Dragons Scene Perceptible Pixel Errors

Figure 4.19: RGB error and perceptible error visualization of Reflective Four Dragons Scene



(a) Texture Primitives Scene RGB Error



(b) Texture Primitives Scene Perceptible Pixel Errors

Figure 4.20: RGB error and perceptible error visualization of Texture Primitives Scene

are expected to be perceived as being in error are marked by red color, the other pixels are marked by pale green color. Perceptible difference images for Bunny, Reflective Bunny, Four Dragons, Reflective Four Dragons and Texture Primitives Scenes are visualized in Figure 4.16(b), Figure 4.17(b), Figure 4.18(b), Figure 4.19(b) and Figure 4.20(b) respectively.

To see the effect of depth on deviation of reprojection positions from sampling points, the color difference image for the Textured Primitives Scene is presented in Figure 4.20(a). As deviation from sampling points is an alternating function of depth, the effect of depth on the error introduced by reprojection is expected to behave similarly. It is apparent in Figure 4.20(a) that the error introduced by reprojection is dependent on the depth as the error alternates in a pattern similar to that of the deviation from the sampling points. Another observation is the effect of color frequency on the error patterns. Comparing the color image of the Textured Primitives Scene with its error image, the regions where the change in color values is high exhibit high rates of error.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this thesis, a GPU-based stereoscopic rendering method that exploits the spatial coherence in the stereoscopic image pairs is presented. One of the stereoscopic image pairs is first rendered via ray tracing while exporting extra surface information in addition to color values. Then all the corresponding reprojection position of each pixel is calculated in parallel. The reprojections that are invalidated due to visibility from the second camera are determined and discarded. The remaining usable surface information is used in ray tracing of the second image. All the computations steps are carried out on the GPU in parallel. On the contrary to previous approaches, parallelism is achieved in performing reprojections among pixels in a row. The presented method is adapted to run with a fully GPU-based OptiX ray tracing engine.

Over 90% successful reprojections are observed in fully diffuse surfaces leading to significant speed-ups in the calculation of the second image. Compared to the CPU-based method about 20 times speed-up is achieved.

The net performance gain achieved using the GPU-based method proposed is shown to be greater than 30% in scenes with diffuse-only materials. The effect of the rate of reflective and refractive surfaces on the amount of performance gain is examined.

The errors introduced by reprojection is examined in detail. The relation of deviation of reprojection coordinates from sampling points to the depth of the surface is analyzed. The implications of deviation from sampling points in uniform and stochastic sampling mechanisms is mentioned. The quality degradation introduced by reprojection is examined by using a perception based difference method and it is shown that in all the test scenes the rate of perceptible errors is less than 3%.

5.2 Future Work

As mentioned previously, projections of surfaces visible in the first image in the stereoscopic image pair reproject to positions in the second image independent of the sampling mechanism used in the second image, leading to degradations in the quality of the reprojected image. In addition to diffuse color values, normal and depth values reprojected are in error due this sampling error. The effect of the error in the normal values effect the final color significantly if the surface in question is highly reflective or refractive.

To reduce the error introduced by the deviation from sampling points, values at actual sampling points in the second image can be interpolated from neighbor reprojections based on their distance to the actual sampling point. In surfaces with smoothly varying color and normal values, the error is expected to reduce when this interpolation based method is applied.

In surfaces where the variation of the normal and color values is too high for the image resolution, no increase in quality should be expected as interpolation can not reconstruct the unknown and frequently changing surface property correctly. Furthermore, instead of linear interpolation, parametric surface values can be stored to be used in calculating the unknown surface properties. For instance, center point and radius of a sphere primitive can be used in calculating the surface property at a certain depth if it is known that this sphere projects to the pixel location in question. However, this

calculations should not be too computationally demanding as this may negate performance gains achieved by the reprojection method. This interpolation based approach is considered for the uniform sampling case. The application of that method can be extended to multisampling and stochastic sampling cases. In stochastic sampling case, variation of sampling points in the y axis poses an additional challenge as that may call for taking samples from neighbor rows to properly interpolate the surface properties.

The effects of more complex shading models can be explored to see the effect of different shading methods on the success rate of the reprojection method. Representing the shading calculation as a shade tree and reusing the nodes that are independent of the changing parameters between cameras(e.g. view direction) can be a starting point.

REFERENCES

[1] Compute Unified Device Architecture (CUDA). http://www.nvidia.com/object/cuda_home.html. Accessed: 07/10/2012.

[2] CUDA C Best Practices Guide. http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html. Accessed: 07/10/2012.

[3] NVIDIA 3D Vision Automatic : Best practices guide. http://developer.download.nvidia.com/whitepapers/2010/3D_Vision_Best_Practices_Guide.pdf. Accessed: 07/10/2012.

[4] OpenCL. http://www.khronos.org/opencl. Accessed: 07/10/2012.

[5] Optix. http://www.nvidia.com/object/optix.html. Accessed: 07/10/2012.

[6] Stanford 3D Scanning Repository. http://graphics.stanford.edu/data/3Dscanrep/. Accessed: 14/11/2012.

[7] Stephan J. Adelson and Larry F. Hodges. Stereoscopic ray-tracing. *The Visual Computer*, 10(3):127–144, 1993.

[8] Stephan J. Adelson and Larry F. Hodges. Generating exact ray-traced animation frames by reprojection. *IEEE Computer Graphics and Applications*, 15(3):43–52, 1995.

[9] Stephen J. Adelson and Larry F. Hodges. Visible surface ray-tracing of stereoscopic images. In *Proceedings of the 30th annual Southeast regional conference*, ACM-SE 30, pages 148–156, New York, NY, USA, 1992. ACM.

[10] James Arvo and David Kirk. An introduction to ray tracing. chapter A survey of ray tracing acceleration techniques, pages 201–262. Academic Press Ltd., London, UK, UK, 1989.

[11] S. Badt. Two algorithms for taking advantage of temporal coherence in ray tracing. *The Visual Computer*, 4:55–64, 1988.

[12] Loren Carpenter. The a-buffer, an antialiased hidden surface method. *SIGGRAPH Comput. Graph.*, 18(3):103–108, January 1984.

[13] A. Es and V. İşler. Gpu based real time stereoscopic ray tracing. In *22nd International symposium on computer and information sciences, ISCIS*, pages 1 –7, November 2007.

[14] John D. Ezell and Larry F. Hodges. Some preliminary results on using spatial locality to speed up ray tracing of stereoscopic images. pages 298–306, 1990.

[15] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '84, pages 213–222, New York, NY, USA, 1984. ACM.

[16] E. Gröller and W. Purgathofer. Coherence in computer graphics. Technical report, 1992.

[17] Johannes Gunther, Stefan Popov, Hans-Peter Seidel, and Philipp Slusallek. Realtime ray tracing on gpu with bvh-based packet traversal. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, RT '07, pages 113–118, Washington, DC, USA, 2007. IEEE Computer Society.

[18] R.M. Hayes. *3-D movies: a history and filmography of stereoscopic cinema*. McFarland classics. McFarland, 1998.

[19] James T. Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM.

[20] L. Lipton. *Foundations of the Stereoscopic Cinema*. Van Nostrand Reinhold, New York, 1982.

[21] T. Möller, E. Haines, and N. Hoffman. *Real-Time Rendering*. Ak Peters Series. Taylor & Francis Group, 2008.

[22] Diego Nehab, Pedro V. Sander, and John R. Isidoro. The real-time reprojection cache. In *ACM SIGGRAPH 2006 Sketches*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[23] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: a general purpose ray tracing engine. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 66:1–66:13, New York, NY, USA, 2010. ACM.

[24] Stefan Popov, Johannes Gunther, Hans-Peter Seidel, and Philipp Slusallek. Stackless kd-tree traversal for high performance GPU ray tracing. *Computer Graphics Forum*, 26(3):415–424, September 2007. (Proceedings of Eurographics).

[25] Holly E. Rushmeier and Kenneth E. Torrance. Extending the radiosity method to include specularly reflecting and translucent materials. *ACM Trans. Graph.*, 9(1):1–27, January 1990.

[26] Shubhabrata Sengupta, Mark Harris, Yao Zhang, and John D. Owens. Scan primitives for gpu computing. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, GH '07, pages 97–106, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[27] Peter Shirley, Michael Ashikhmin, Michael Gleicher, Stephen Marschner, Erik Reinhard, Kelvin Sung, William Thompson, and Peter Willemsen. *Fundamentals of Computer Graphics, Second Ed.* A. K. Peters, Ltd., Natick, MA, USA, 2005.

[28] Ming Wan, Nan Zhang, Arie Kaufman, and Huamin Qu. Interactive stereoscopic rendering of voxel-based terrain. In *Proceedings of the IEEE Virtual Reality 2000 Conference*, VR '00, pages 197–, Washington, DC, USA, 2000. IEEE Computer Society.

[29] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, June 1980.

[30] Yangli Hector Yee and Anna Newman. A perceptual metric for production testing. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, pages 121–, New York, NY, USA, 2004. ACM.