# PERCEPTUAL ANALYSIS OF MOTION BLUR TECHNIQUES IN DISTRIBUTION RAY TRACING

By

Raffi Agop Bedikian

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Computer Science

May, 2011

Nashville, Tennessee

Approved:

Prof. Robert E. Bodenheimer, Jr.

Prof. Julie A. Adams

TABLE OF CONTENTS

Page

# ACKNOWLEDGMENTS

I would like to thank my advisor Prof. Bobby Bodenheimer for his guidance throughout this project, and Prof. Julie Adams for her mentoring and feedback. Thank you to Pete Shirley for his insight onto sampling in ray-traced motion blur and the challenges inherent therein. Thank you to Seth Rosenthal of Tweak Software for use of the RV software package. And thank you to my family and friends who supported this work.

LIST OF TABLES

LIST OF FIGURES

# CHAPTER I

# INTRODUCTION

The goal of this work was to improve computer graphics by better understanding how internal rendering parameters affect the visual perception of animation. The scope of the work was constrained to the perception of object motion in animated sequences, specifically the motion blur effect that is generated to mimic the shutter and exposure of a film camera. Although this motion blur occurs as a natural consequence in real-world cameras due to the nonzero time needed for the image to be fully exposed, it is generated artificially in computer graphics in order to make the generated animation appear more realistic. There are a large number of techniques for artificially creating the motion blur effect, and the techniques greatly depend on the type of rendering that is being used to create the images. In this work, we limited the scope of our examination of motion blur to ray-traced rendering. We examine in detail the perceptual aspects of different techniques for generating ray-traced motion blur via experiments designed to isolate the various parameters in order to determine their effectiveness.

## Problem Motivation

The process of producing computer-rendered animations is composed of rendering individual frames, with each frame corresponding to a certain time in the animated sequence. In a real-world camera, each frame of a captured video is produced by opening the camera shutter for a short duration of time. During this duration, moving

objects will be blurred in the frame because their position within the frame changes as the film is exposed.

In computer graphics, the approach to rendering is to sample a mathematical description of a scene (containing objects, lights, materials, etc.) to yield a color for each pixel in the frame. In general, as the sampling density increases, the quality of the resultant image increases, including the appearance of the computer-generated motion blur. In this work, we set out to provide a foundation for ways to generate motion blur efficiently from a perceptual standpoint; in other words, we hope to provide insight into answering the following question: “Given a certain resource budget, what is the best way to produce ray-traced motion blur such that the perceptual quality is highest?”

Another motivation for this work arises from the scarcity of unbiased evaluation of the factors we were interested in testing, and this scarcity is more pronounced when approaching the evaluation from a perceptual standpoint, rather than a mathematical or analytical standpoint. We feel as though the field of computer graphics has had a long period of flourishing inventions and novel algorithms without enough unbiased comparisons between the known methods, and we hope that this work can act as one piece of the puzzle toward correcting this balance.

## Contributions

This work aims to contribute to the understanding of perception in computer graphics. Within the area of motion blur rendering, there are numerous challenges related to the process by which the sampling is performed, and some of these challenges have no clear optimal solutions. Our work’s main contribution is perceptual evaluation of various approaches to these challenges via user experiments.

In particular, we constrain the definition of perceptual quality to noise in the image (Figure 1). Although the amount of noise can be exactly determined via techniques that compare an image to its maximum-quality counterpart, these techniques do not adequately capture the human perception of that noise. Similarly, prior work has (Mitchell, 1991) approached the problem of finding an optimal sampling pattern from a signal processing point of view. Our work attempts to find the cutoff beyond which the noise is not readily detectable to a human, in a variety of different scenarios. By comparing those cutoffs across the various settings used to perform the rendering, we are able to determine which parameters have the largest impact on perceptual quality.

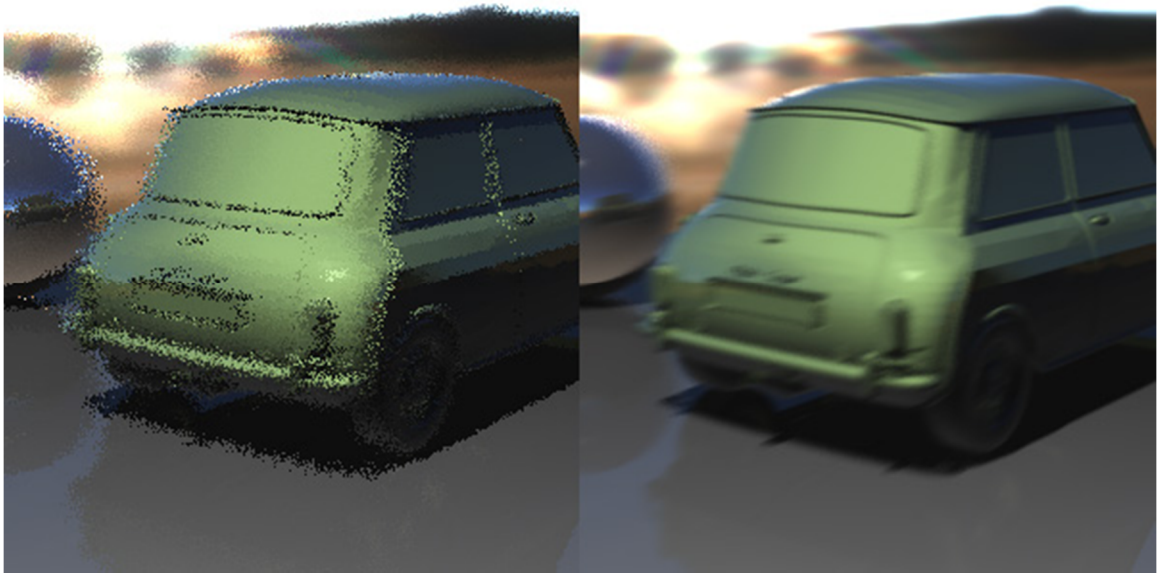


**Figure 1: The appearance of noise due to undersampling in a ray-traced scene.**

## Outline

In Chaprer 2, we provide background knowledge relevant to this work, which includes discussions of ray tracing, sampling, and motion blur. Chapter 3 presents a detailed description of the renderer used to produce the animations that were played during the user experiments. In Chapter 4, we will provide an overview of the design of

our experiments, followed by the results and analyses of those experiments in Chapter 5. Our concluding remarks are presented in Chapter 6.

# CHAPTER II

# BACKGROUND

This chapter discusses relevant topics that provide context for our experiments. We begin by broadly discussing rendering in computer graphics as well as various aspects of ray tracing. We then describe how the motion blur effect is produced in ray tracing through simulation of a camera shutter. Finally, we discuss details of the sampling strategies we investigated.

## Rendering and Ray Tracing

Fundamentally, the process of rendering in computer graphics is the procedure by which the mathematical description of the scene is converted into an image (Catmull, 1974). The scene description may contain many different attributes, such as:

- Geometric objects including their location, material properties, and size.
- Parameters for the virtual camera including orientation, field of view, and location.
- Descriptions of light sources, ranging from the type of light (e.g., point light, directional light, ambient light) to color and intensity.

The goal of the renderer is to take a given scene description and produce an image according to that description as efficiently and accurately as possible. In animated

sequences such as film, the rendering process is repeated for each image in the sequence due to the fact that changes in geometry (e.g., moving characters or objects) or the camera will alter the image to be produced.

The task of rendering has two approaches that are commonly used today, namely rasterization (Foley, 1982) and ray tracing (Whitted, 1980).

### Rasterization

In rasterization, the scene geometry is processed in bulk by iterating over the objects and projecting them onto the virtual camera film. The projection can be accomplished efficiently through the use of matrix multiplication with the geometric data. Modern graphics processing hardware has evolved to be incredibly fast at performing this task, making rasterization is suitable for interactive rendering applications (Foley, 1982) such as video games.

In order to render scene geometry to the screen, geometric objects are first pre-sorted by depth from the camera. The geometry is then rendered from furthest to closest from the point of view of the camera, projecting and shading each polygon onto the image space from the viewpoint of the virtual camera.

A major drawback of rasterization is the lack of more physically correct effects such as reflections and shadows. When a particular triangle of a mesh is being processed, these effects are difficult to perform because that triangle has no knowledge of its context in the environment. Therefore, such effects are often simulated with approximations that may not be physically correct. For example, reflections are often approximated by rendering an intermediate image from the point of view of the object, then mapping this image onto the object as a texture. However, the drawbacks of processing scene geometry independently are also what make rasterization so fast; with

a limited need to reference other information about the scene, a single geometric object can be fed through the pipeline very quickly without unexpected interruptions.

In a rasterizing renderer for a video game, in order to maintain high performance even on modern hardware, the sampling strategy for the pixels in the image is largely homogeneous from pixel to pixel. As such, the renderer often has limited control over the specific samples that are generated for the color calculation of the pixel. However, new approaches such as stochastic rasterization (Akenine, 2007; McGuire, 2010) have potential to allow for much more control over the samples, thereby enabling a broad range of effects seen in distribution ray tracing that were previously much more difficult to simulate.

## Ray Tracing

Ray tracing is a natural extension to an algorithm known as ray casting (Roth, 1982). In ray casting, the scene remains resident in memory and the algorithm instead iterates over the pixels in the image to be produced, casting a ray from the virtual camera into the scene to see what the ray intersects. The closest object to the camera that the ray intersects is the one that is used for calculating the color of that sample.

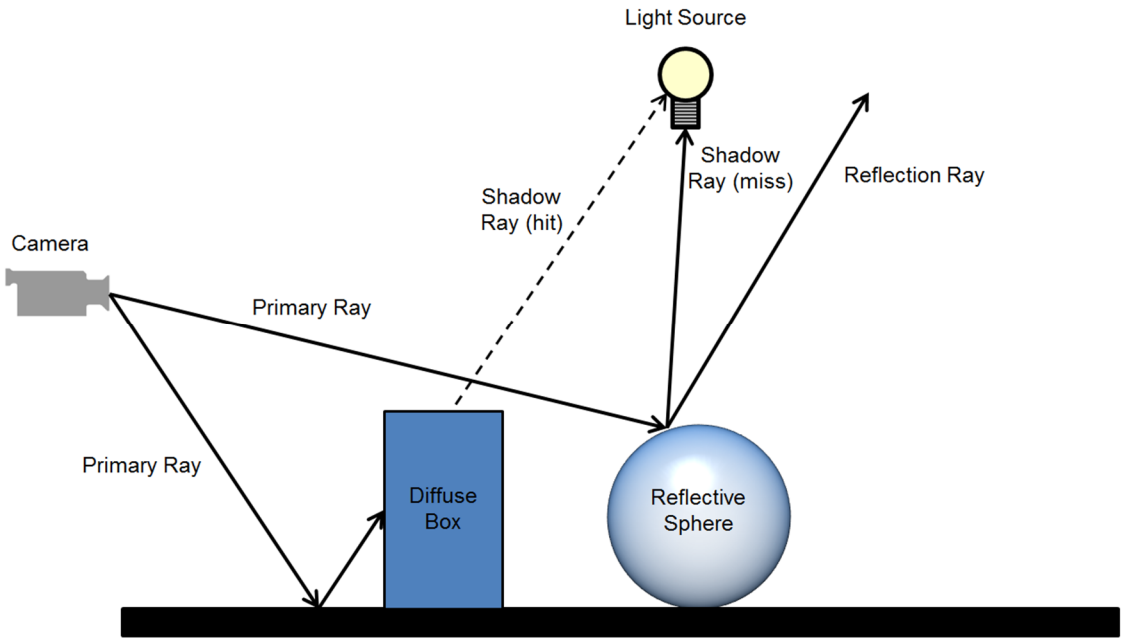



**Figure 2: Overview of the ray tracing algorithm. Shadow rays are cast from shading points toward light sources. If a shadow ray intersects another object along the way, then the point is in shadow.**

Ray tracing improves upon ray casting by allowing the above principle to be applied recursively (Whitted, 1980). Whereas effects such as reflections and shadows are difficult to implement in rasterization, they are trivial to implement in ray tracing (Figure 2). After the initial “primary ray” is cast from the camera into the scene and the closest intersection is found, a “secondary ray” is cast from the intersection point for each of these effects. For example, in order to determine whether the intersection point is in light or shadow, a secondary ray is cast toward the light source. If this secondary ray intersects any geometric object along the way, then the point is in shadow. Similarly, reflections can be simulated by casting a secondary ray whose direction is calculated by reflecting the incoming primary ray across the normal vector of the surface at the point of intersection. The returned color from the secondary reflection ray is used in calculating the shading of the initial intersection point. As such, ray tracing is an inherently recursive

procedure and this necessitates that the recursion depth should be limited in order to prevent performance pitfalls.

Fundamentally, one the biggest advantages of ray tracing over rasterization is the flexibility of the algorithm, but the reasons for that flexibility also affect the algorithm's speed in practice. Not only is it easier to implement physically correct effects such as those discussed above, but the algorithm yields much more control over the sampling strategy, since the sampling locations are chosen entirely in the high-level rendering code rather than as part of the graphics pipeline. This additional control proves to be very useful in the context of the sampling strategies we evaluated.

An additional consideration in implementing an efficient ray tracer is the traversal through the scene objects. For example, it does not make sense for a ray to perform an intersection check against every object in the scene, especially if those objects are not anywhere near the ray. For this reason, a number of acceleration structures have been developed that allow for a ray to perform this scene traversal in a much more intelligent manner (Kay, 1986; Amanatides, 1987). The acceleration structures fall into two main categories: enveloping the scene objects in bounding volumes, and subdividing the scene space into smaller partitions. We are primarily concerned with the first approach. In this approach, simple bounding volumes (e.g., boxes) enclose the scene geometry as well as other bounding volumes. A hierarchical tree of bounding volumes is formed, with larger bounding volumes at the root of the tree and objects as leaves. This tree is constructed such that nearby geometry objects are relatively close to one-another in the tree. The idea behind this layout is that a ray can quickly perform an intersection test against an entire potion of the scene simply by performing an intersection test against its respective bounding volume.

## Motion Blur

The notion of motion blur arises from simulating the shutter of a camera. However, this is not the only type of camera effect that can be artificially simulated. Representations of different camera effects, such as shutter and focal length fall under different camera models. A simple ray tracer employs a very basic camera model due to its computational simplicity. One of the simplest camera models, without any motion blur or focal length effects, is the instantaneous-exposure pinhole camera. This model is unrealistic because a physical camera may have a finite, but nonzero exposure length as well as a finite, but nonzero aperture.

Simulating the lens aperture of a physical camera in distribution ray tracing is achieved by modifying the start locations and directions of the primary rays, assuming the starting points occur on a virtual lens (rather than a single point, as in the pinhole model) and that the rays intersect a focal plane at some distance from the camera that is in maximum focus (Cook, 1984). Objects closer or further away than this focal length will be blurred, since the incoming rays no longer intersect at the same point, thereby creating a depth-of-field effect. This effect is outside the scope of this work, even though it is a necessary component of creating a physically accurate camera model.

In a pinhole camera model, each frame of an animated sequence will capture the appearance and positions of the scene objects at the precise time values for those frames, even though this is not possible with a physical camera due to the nonzero time needed for the incoming light to properly expose the film or digital sensor. This exposure time creates a blurring effect, since a scene object will have moved in the window of time during which the camera shutter is open. With instantaneous exposure, an animated sequence of images will suffer from temporal aliasing (Korein, 1983), in which objects

appear to jump from location to location, rather than smoothly moving between locations. An example of temporal aliasing is the wagon-wheel effect, in which a rotating object can appear to be stationary or rotating in the reverse direction if the sampling rate is too low.

The simple solution to this issue in ray tracing is to simulate the camera exposure time directly by using many samples per pixel that are distributed throughout time (Korein, 1983; Cook, 1984). The time for each ray is selected according to the virtual camera shutter window, and that time is saved in local data that is kept for each ray. During scene traversal, this saved ray time is used to perform intersection tests. When the scene is being traversed for a particular ray, objects are moved to their respective positions based on the time of the incoming ray. As will be discussed in Chapter 3, several optimizations take place in order to prevent severe performance degradation during this step. The ray time is also copied from the primary ray to any secondary rays that are generated by the primary ray intersection, such as shadow or reflection rays.

In this way, some fraction of the rays may intersect a moving object, while the remaining rays miss the object. Combining the returned sample colors into one estimate of the pixel color via the use of an accumulation buffer will cause the object to appear blurred in the final result (Haeberli, 1990).

In order to fully mitigate the effects of temporal aliasing, the samples are distributed across the shutter window randomly, rather than at regular intervals. Jittered sampling is an essential feature for implementing motion blur in ray tracing because of the very noticeable strobing effects that occur without its use (Figure 3).

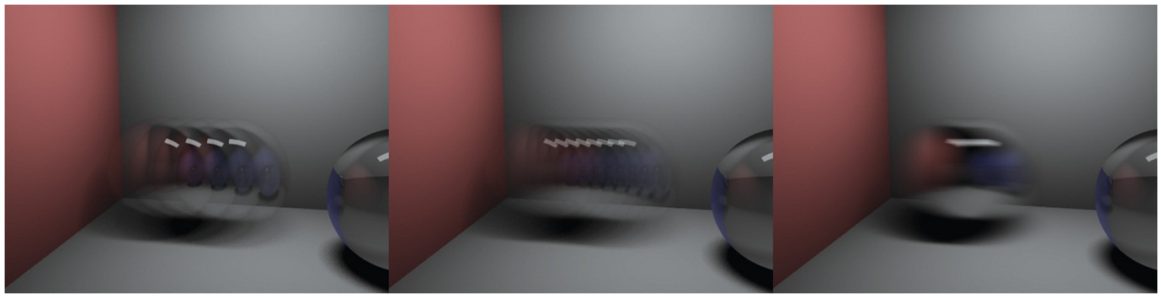


**Figure 3: From left to right, 4 non-jittered samples, 8 non-jittered samples, and jittered sampling.**

## Sampling

So far, we have only discussed concepts related to the mitigation of temporal aliasing. Another type of aliasing that has large impacts on the perception of image quality is spatial aliasing (Mitchell, 1987). Whereas in temporal aliasing the issue arises from poor sampling throughout the time axis, spatial aliasing is caused by poor sampling in the sub-pixel region. Similar to the approach for temporal aliasing, the effects of spatial aliasing can be mitigated by using more samples per pixel that are distributed throughout the sub-pixel region. A basic ray tracer without motion blur may fire one ray per pixel from the center of the pixel, but this choice is arbitrary; the top-left corner can be chosen, or the origin of the ray can be randomized for each pixel. Although this latter option may seem unintuitive at first, it is based on the fact that random noise is much less objectionable to the human visual system than aliasing (Cook, 1986).

Based on these three dimensions (two values *u*,*v* for the sub-pixel coordinate, and one value *t* for time), we initialize each primary ray with an origin and direction. The spatial coordinates are initialized in the range [0, 1) representing their position within the pixel, and the time value for the ray lies in the interval in which the virtual camera shutter is open for the frame being rendered. Given we are not simulating lens aperture in this

work, the origin for every ray will be the camera eye point adjusted by the time for that ray and the velocity of the camera. If the camera is not moving, then all rays will have the same origin, regardless of their respective time values. The direction of a ray is calculated by finding the vector between the camera eye point and the desired sampling location on the virtual image plane. This sampling location is determined by combining the row and column of the pixel with the individual ray's sub-pixel coordinate.

A core issue lies in how to distribute the samples spatially and temporally, and this is one of the key questions that this work seeks to address. In our case, the sampling region forms a cube, but more complex rendering schemes may require additional dimensions for sampling, such as distributing ray origins over the virtual camera lens to form depth of field. In stochastic sampling (Cook, 1986), it is suggested that the values for the different dimensions should be uncorrelated so as to ensure that the samples do not bunch up or leave large gaps unsampled. This idea has been studied from a variety of perspectives including in quasi-Monte Carlo sampling methods (Kollig, 2002).

### Point Sets for Sampling

We now discuss the approaches to solving the sampling problem that we explore in this work, namely stratified Monte Carlo sampling, Poisson disk sampling, and quasi-Monte Carlo sampling via the Hammersley sequence. All three approaches attempt to improve upon the standard Monte Carlo method, in which the high-dimensional illumination integral for rendering (Kajiya, 1986) is evaluated stochastically using point samples. By choosing the locations of those point samples more intelligently, these methods reduce the amount of noise in the rendered image, which makes rendering more efficient.

In stratified Monte Carlo sampling – also called jittered grid sampling (Cook, 1986) – the sampling region is partitioned into evenly-sized grid cells, with one grid cell for each sample. The samples are then randomly distributed within their respective grid cells. Since we consider three sampling dimensions in this work, the grid cells are small cubes.

We also explore Poisson Disk sampling, in which samples are distributed randomly across the entire region with the restriction that there is a guaranteed minimum distance between any two sampling points. This distribution approximates the distribution of photoreceptors in the retina (Yellott, 1983) and has very nice blue-noise properties due to its band-limited nature (Mitchell, 1987; Lagae 2006). However, without an intelligent algorithm (Dunbar, 2006) these sampling points can be computationally intensive to calculate for a large number of samples due to the fact that neighboring samples must be checked to ensure that the distance restriction is upheld.

The last sampling distribution we explore is the Hammersley sequence, which is a deterministic, quasi-Monte Carlo method designed to have low discrepancy (Pharr, 2004). Although there are varying precise definitions of discrepancy, the main idea is as follows. If one partitions the pixel space into sub-regions and then samples the pixel using a particular strategy, ideally the ratio of samples contained in a certain sub-region to the total number of samples should be very near the ratio of the volume of that sub-region to the total volume of the pixel (Shirley, 1991). Therefore, discrepancy refers to the maximum error over all possible ways to divide the region into sub-regions. The Hammersley sequence is defined using the radical inverse. Pharr and Humphreys (2004) cleanly describe the formulation of the Hammersley sequence using the radical inverse. The radical inverse construction is based on the fact that any positive integer $n$ can be expressed in a base $b$ with a sequence of digits $d_m \ldots d_2 d_1$ uniquely determined by:

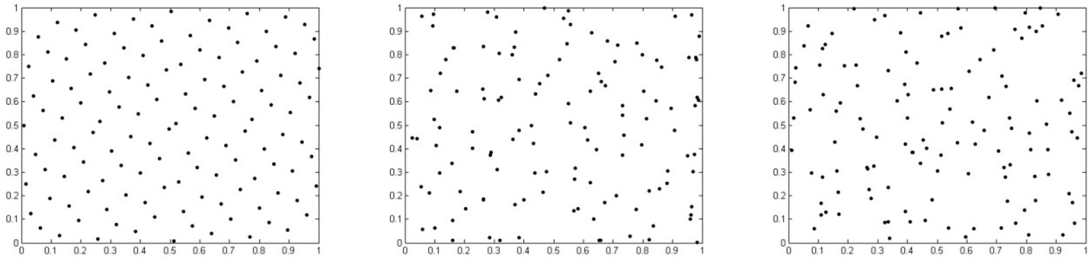
$$n = \sum_{i=1}^{\infty} d_i b^{i-1}.$$

The radical inverse function $\Phi_b$ in base *b* converts a nonnegative integer *n* to a floating-point value in [0, 1) by reflecting these digits about the decimal point:

$$\Phi_b(n) = 0.d_1 d_2 \ldots d_m.$$

The Hammersley sequence is defined for a set of *N* samples in an *n*-dimensional space by the following (note that the bases $b_i$ must all be relatively prime):

$$x_i = \left(\frac{i}{N}, \Phi_{b_1}(i), \Phi_{b_2}(i), \ldots, \Phi_{b_n}(i)\right).$$



Hammersley Poisson Disk Stratified Monte Carlo

**Figure 4: Projecting the three-dimensional sampling points onto two dimensions.**

One interesting aspect of these point sets is their behavior when projecting the three-dimensional points onto two dimensions. Figure 4 shows the appearance of the three point sets when the time dimension is ignored from a set of 125 samples. While the Hammersley point set remains well-distributed, both other point sets exhibit clumping behavior, since the third dimension was dropped. This behavior can alter the performance of the point sets in regards to strictly spatial anti-aliasing (e.g., in a scene with no motion, the time of each sample is irrelevant). Our work seeks to provide insight into whether this performance difference matters from a perceptual standpoint.

Figure 5 provides an overview of the appearance of the three different sampling methods at three sampling densities. Upon close inspection, the strong specular streaks have slightly less noise with the Hammersley set than the other two methods, with the effect most noticeable at 64 samples per pixel. The reflections appear less granulated, and we hypothesize that this will make the motion blur appear smoother. However, the difference is very subtle and our work intends to provide evidence into whether the difference is noticeable in animation rather than static images.

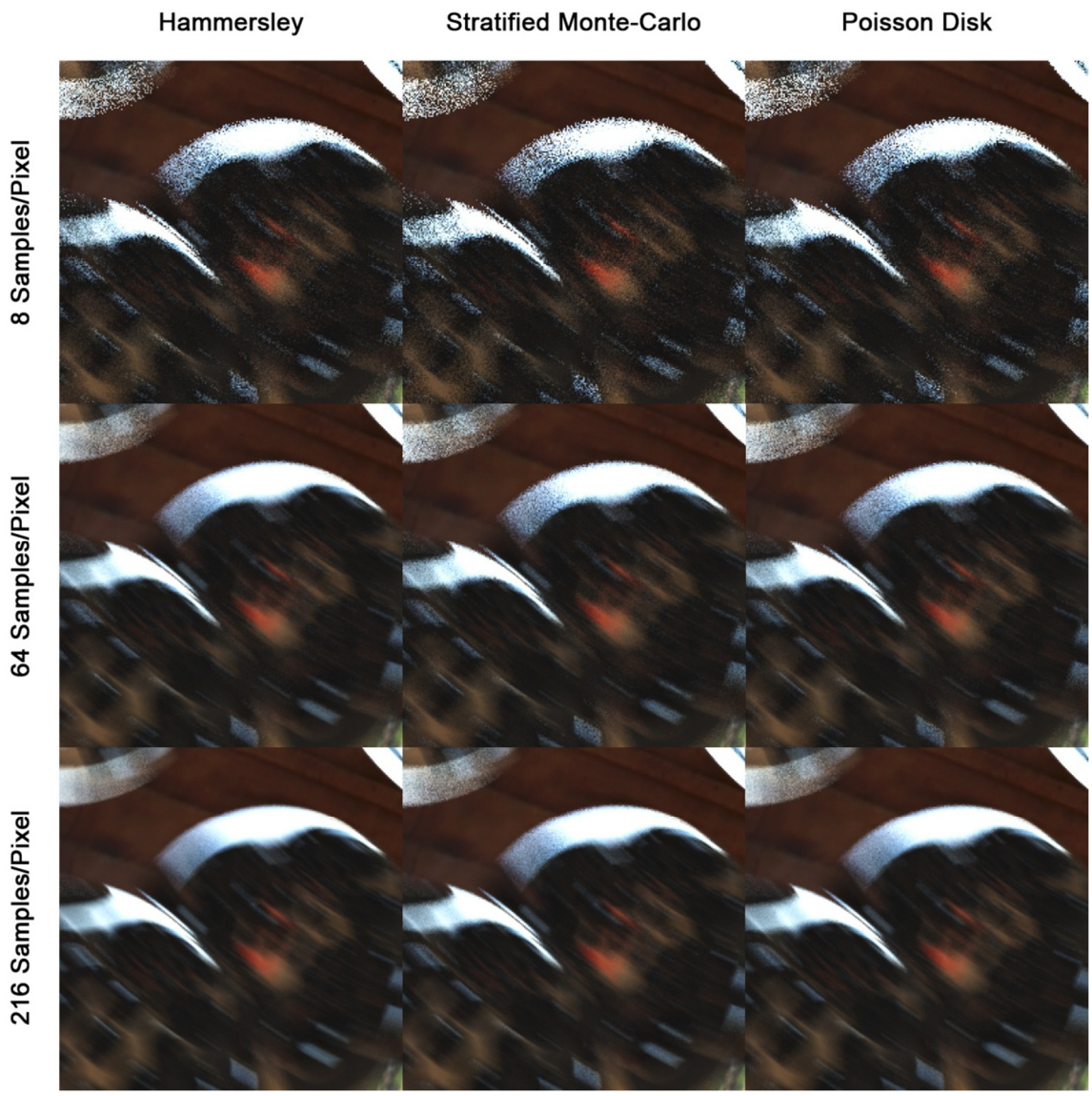



**Figure 5: The appearance of motion blur with three different point sets and three sampling levels.**

## Spatial Filtering and Reconstruction Techniques

From these samples, we must estimate a color for the pixel based on the colors returned by the samples during the ray tracing procedure, which involves the use of a reconstruction filter (Netravali, 1988). A simple approach is to simply average all of the colors (i.e., a box filter spanning the entire pixel), but this approach assumes that each

sample has equal importance to the final result, which may result in poor image quality. Another filter kernel is the two-dimensional Gaussian, which allows for samples closer to the center of the pixel to be weighted more highly, while the weights of all other samples decrease toward the edges. In practice it is often beneficial to use reconstruction filters larger than one pixel, which has two advantages. The first advantage is that sample values are able to be shared across multiple pixel color calculations, and the second advantage is that including a larger area in the color calculation makes edges appear slightly smoother (Figure 6).

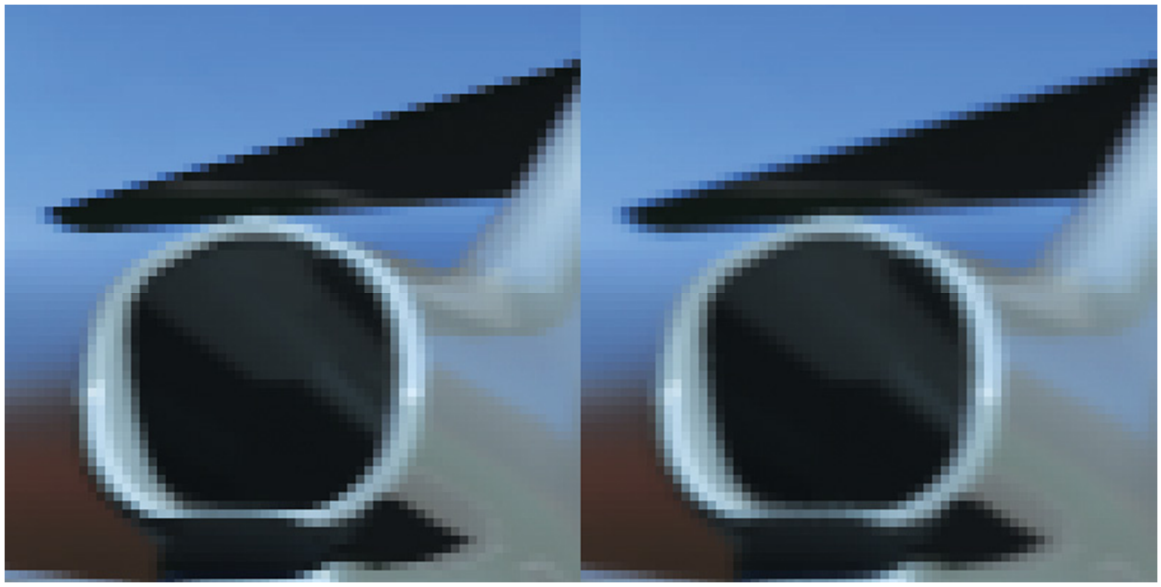


**Figure 6: Unmodified (left) and Gaussian-warped (right) sampling.**

In the work by Ernst (2006), the authors describe ways in which the “weighting” approach is unsatisfactory. Even when using a filter larger than one pixel and sharing samples between adjacent pixels, many of those samples will have a very low weight during reconstruction, making the overall use of the samples inefficient.

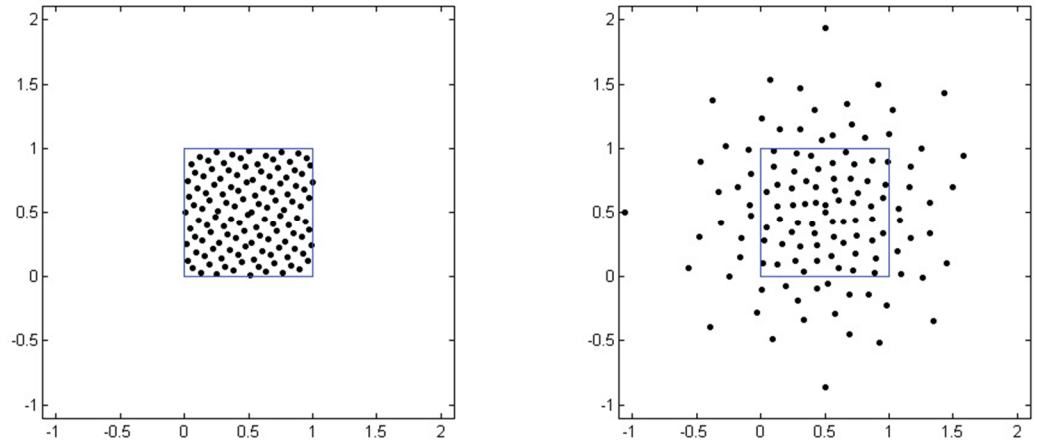



**Figure 7: 125-point Hammersley set unmodified (left) and Gaussian-warped (right)**

An alternative approach is filter importance sampling (Ernst, 2006), in which the sample locations are warped according to the reconstruction filter kernel prior to ray tracing. In doing so, samples are not shared between adjacent pixels and no weighting is applied during the color calculation. Figure 7 shows the effect of applying a two-dimensional Gaussian warping procedure to the Hammersley point set using the Box-Muller transform (Box, 1958).

Given two independent random variables $X_0$ and $Y_0$ uniformly distributed over (0, 1],

$$X_G = \sqrt{-2 \ln X_0} \cos(2\pi Y_0)$$

$$Y_G = \sqrt{-2 \ln X_0} \sin(2\pi Y_0),$$

where $X_G$ and $Y_G$ are independent random variables with a normal distribution of standard deviation 1.

Since the samples are altered in location rather than weighted, the contributions of all samples toward the pixel color calculation are equal.

Temporal Filtering Kernels

Similar to the spatial warping techniques, we also employ warping on the temporal axis of each time sample. We restrict the temporal warping to keep the time values within the same range of values as the unmodified distribution. We explore two time filters (Figure 8), namely the truncated box filter and the triangle filter. The result of applying each filter to a regularly-spaced set of points is shown in Figure 9.

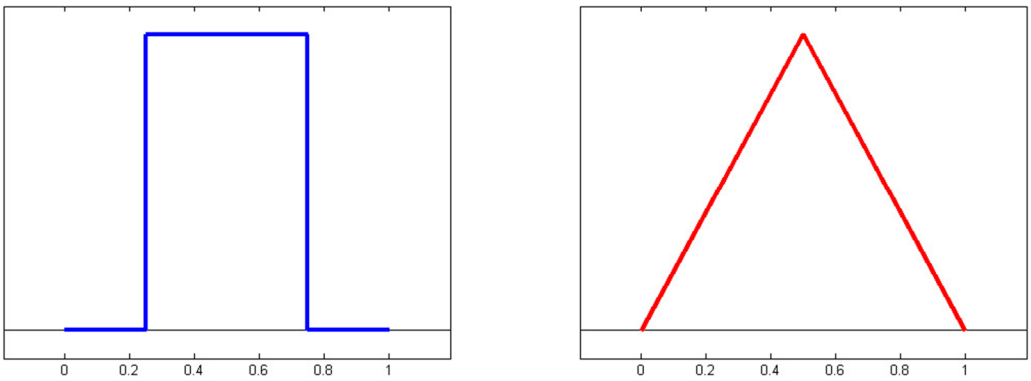



**Figure 8: Truncated box filter (left) and triangle filter (right)**

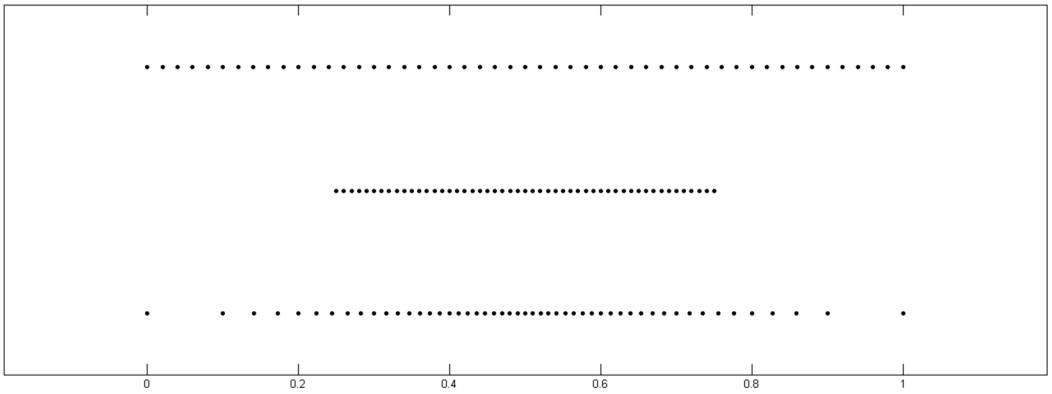



**Figure 9: The effect of warping regularly-spaced points in the interval [0, 1] with the truncated box filter (middle) and triangle filter (bottom)**

The truncated box filter is a commonly used filter for motion blur due to being very easy to implement. Our truncated box filter has a value of 0 for the first and last 25% of the shutter window. This compresses the motion blur to occur during the middle 50% of each frame's shutter time.

The triangle (or tent) filter is not as commonly used, but has the property that samples are more concentrated in the center of the shutter window, with a gradual decline to 0 toward the edges of the window. We explore this time filter because the resulting blur may appear more natural, since the blur edges (and specular highlights) appear smoother (Figure 10).

**Figure 10: The appearance of motion blur with a truncated box filter (left) and a triangle filter (right)**

## Related Work

There is prior work in sampling methods relevant to this work. In filter importance sampling (Ernst, 2006), a method of sampling is described in which the reconstruction filter is used to warp the sampling locations prior to ray tracing traversal for improved image quality. In quasi-Monte Carlo sampling (Keller, 1998; Kollig, 2002), the quality of numerical integration tasks such as sampling in ray tracing is improved via deterministic methods such as the Hammersley sequence. The development of stochastic rasterization (Akenine, 2007; McGuire, 2010) has also created new opportunities for more intelligent sampling in rasterization as well.

Analyzing sampling patterns has been approached in a variety of ways. The effects of aliasing on motion perception has been studied, (Colletta, 1990). The problem of determining optimal sampling patterns has been approached via blue-noise spectral analysis (Cook, 1986; Mitchell, 1991; Lagae 2006). Efficient tiling patterns have been explored (Cohen, 2003) that solve issues with aliasing artifacts between pixels. Lastly, the Poisson-disk sampling pattern was determined to correspond to the arrangement of photoreceptors on a retina (Yellott, 1983).

For motion blur, new efficient methods exist that require substantially fewer samples than standard distribution ray tracing (Hachisuka, 2008; Egan, 2009). There are also methods for generating motion blur under globally-illuminated conditions (Cammarano, 2002) and for computing shading and visibility separately to reduce noise (Sung, 2002). Motion blur can also be computed in the image space rather than using full distribution ray tracing, but with much faster computation times (Potmesil, 1983; Max, 1985). The application of distribution ray tracing in interactive settings has also been explored, including the use of ray-traced motion blur (Boulos, 2006).

# CHAPTER III

# RAY TRACER IMPLEMENTATION

This chapter presents the technical details of our ray tracer. We will discuss the high-level goals we set out to achieve in constructing this ray tracer, followed by details of certain aspects of our implementation.

## Motivation and Goals

The first step in addressing the issue of rendering for this project was to identify the primary constraints on which our choice of renderer would depend. These constraints are outlined below:

- Due to the high number of combinations of rendering parameters, there would be many videos to generate. Therefore the renderer needed to support batch rendering capabilities or be flexible enough to allow for a straightforward batch rendering implementation. Also, the renderer needed to be fast enough to allow the large number of videos to be generated in a reasonable amount of time.

- Given that additional more advanced rendering features (soft shadows, depth of field, etc.) would require additional dimensionality when generating the point sets for sampling, we decided that limiting the dimensionality to three (namely *u, v, t* in the pixel space) would allow for the effects of motion blur to be tested in a more

isolated and controlled fashion. Therefore, the renderer did not have to support advanced effects, such as global illumination.

- A high degree of control over the sampling procedures of the renderer would be required in order to implement the different rendering parameters we were interested in evaluating. Therefore, the renderer needed to support the use of multiple point sets, time filters, and sampling levels.

When evaluating a variety of options to satisfy these constraints, we concluded that traditional commercial rendering offerings either did not provide sufficient flexibility for the features we needed to implement or the flexibility was too cumbersome to use effectively. Conversely, many open-source implementations focused on different areas of ray tracing (such as photorealistic rendering) that were not necessarily relevant to our project, in addition to being too slow for being able to render a large number of videos in a reasonable time frame.

Given these conclusions, we decided to create our own ray tracer. We decided to write our ray tracer on top of the NVIDIA OptiX SDK (Parker, 2010).

OptiX is a general-purpose ray tracing engine that maps highly-parallel ray tracing tasks onto NVIDIA CUDA-based GPU architectures. NVIDIA CUDA is a parallel computing architecture that simplifies the task of writing highly-parallel code for NVIDIA GPUs by providing a high-level parallel programming API that is accessible through C/C++ as well as many other programming languages. OptiX provides an additional layer of abstraction on top of CUDA through the use of a just-in-time compiler that generates CUDA kernels based on user-supplied programs for ray generation, material shading, object intersection, and scene traversal.

Compared to traditional CPU architectures that may have 4-8 complex cores, GPUs take the opposite approach by using upwards of 400 very simple execution cores. Although CPUs may feature complex branch prediction, pipeline reordering, and other tasks that improve single-thread performance, GPUs sacrifice single-thread performance in favor of performance boosts for massively-parallel tasks. This approach is well-suited for tasks such as ray tracing.

Our reasons for choosing OptiX were as follows:

- GPU acceleration provided substantial rendering speed increases, which allows us to generate more videos in less time. This speed increase was more pronounced by the fact that our rendering needs did not involve more complex effects like indirect illumination or ambient occlusion, and therefore the thread execution divergence would be fairly minimal.
- We had prior experience writing code for OptiX.
- The OptiX SDK provided substantial pre-built functionality that we needed, such as high-performance acceleration structures and an .OBJ model loader.
- The flexibility of custom ray generation code in OptiX allowed us to implement all of the flexibility needed in terms of multiple point sets, time filters, and sampling densities.

## Implementation Walkthrough

Given that the OptiX SDK is a generalized ray tracing framework, rather than a complete renderer, it provided only a starting point for the remainder of our

implementation. Figure 11 shows an overview of the operational structure of our system. The OptiX SDK contains samples for performing simple ray tracing with no motion blur, but our ray tracer required additional functionality not present in the SDK by default. The ray generation program needed to be modified to distribute samples temporally and retrieve sampling locations from a pre-generated sampling buffer. The intersection and bounding-box calculation programs for geometric objects needed to alter the location of the geometry based on the time window. The Context and Scene Graph (including acceleration structures) are built-in OptiX classes for facilitating transfer between the CPU and GPU as well as allowing efficient representation of the scene on the GPU. All host code was written for this project except for the random number generator, which uses a Mersenne Twister implementation (Matsumoto, 1998).

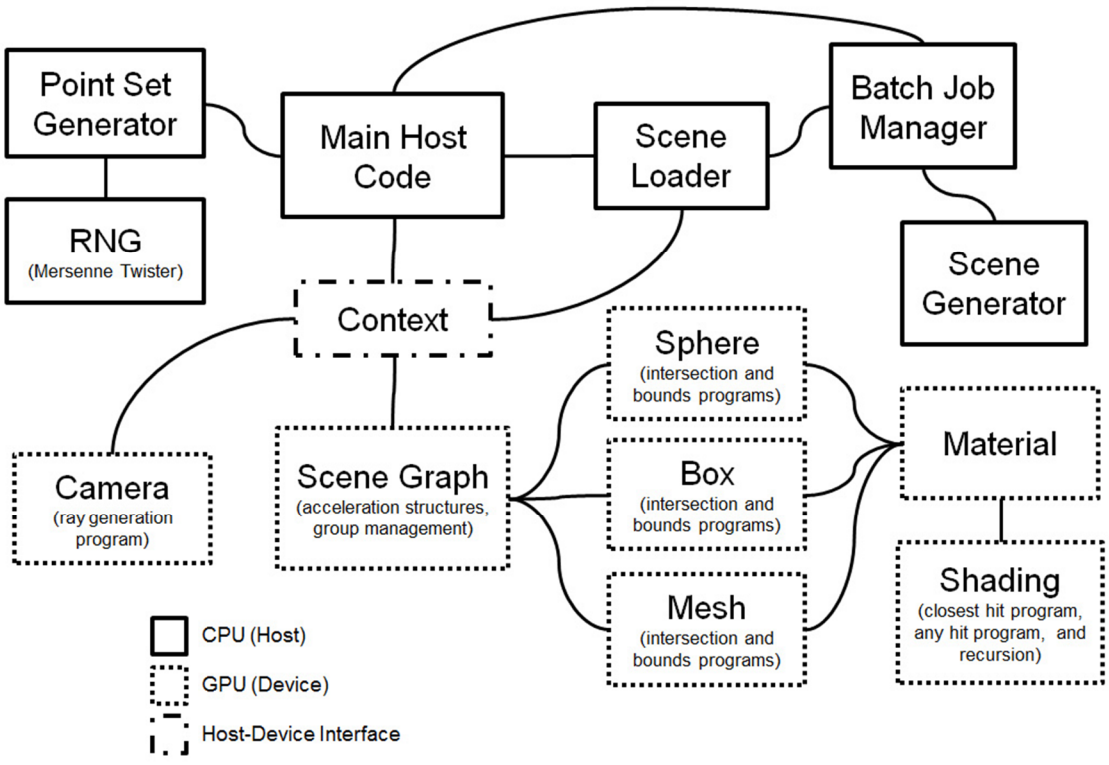



**Figure 11: Overview of system design**

We will now walk through a high-level overview of the execution of a batch rendering job by the system. The job description (Figure 12) contains seven lines.

```
Plane_scene
8
1 8 27 64 125 216 343 512
3
hammersley stratifiedmc poisson
2
box triangle
```

**Figure 12: An example job description for rendering the plane scene**

The first line gives the name of the job, for logging purposes. The next line gives the number of sampling levels to test, followed by a line containing those sampling levels. Similarly, the next two lines give the number of point sets followed by the point sets to test, and the two lines after that give the number of time filters followed by the time filters to test. In this way, the user has precise control over the batches of frame sequences that the renderer generates.

```
material
materialname plane_material
diffusecolor 0.6 0.85 1.0
specularcolor 0.3 0.3 0.3
ambientcolor 0.05 0.05 0.05
fresnelminimum 0.3
fresnelmaximum 1.0
fresnelexponent 2.0
end

object
type model
materialname plane_material
modelfilename models/B-727-200.obj
scale 13
location 0 0 420
velocity -1000 0 0
end

camera
location 400 375 800
locationvelocity -600 0 0
lookat 300 350 0
up 0 1 0
upvelocity 0 0 0.005
lookatvelocity -800 10 0
fov 50
end

settings
starttime -12.5
frametime 0.12
numframes 210
sizex 768
sizey 768
ambientlightcolor 0.7 0.9 1.0
maxdepth 4
pointset hammersley
numsamples 64
timefilter triangle
envmap data/CedarCity.hdr
scenename planescene
end
```

**Figure 13: A few blocks used in the description of the plane scene. Note that the full scene description contains blocks for lighting and other scene information.**

The batch job manager interprets the job description and creates the different combinations of rendering parameters that are to be fed into the renderer. For each different combination specified implicitly in the job description, the batch job manager

generates a scene description (Figure 13), which contains all of the information that the renderer needs in order to render a sequence of frames for a given scene and rendering parameters. The scene description is composed of blocks, which each contain information about a different part of the scene. There are seven types of blocks used in the scene description format for this system: Group, Light, Material, Object, Object instance, Camera, and Settings. There may be many groups, lights, materials, objects, and object instances in a scene, but a scene description can only have one Camera block and one Settings block.

After the scene description is generated, it is passed into the scene loader. This class creates an OptiX context and then reads the scene description, setting fields in the context appropriately. The scene graph is initially empty, and is populated based on the scene description during loading. Not every node created during loading refers to a geometry object. In addition to the intermediate nodes created for managing geometry groups, intermediate nodes are also created for performing ray transforms. These transform nodes and their use will be discussed in more detail in the following section.

After the scene loader creates the OptiX context appropriately, the context is passed to the main renderer host code. This code is responsible for creating and initializing necessary OptiX structures (such as buffers to transfer data between the host and device) as well as initiating the ray-tracing procedure for every frame. The host code also generates the appropriate point set based on the scene description. Given that generating a unique point set for each pixel is prohibitively slow (especially for the Poisson set), this point set is either generated once for each frame or once for the entire frame sequence, depending on the configuration. In addition to generating the point set, the host code also generates a buffer of random seeds that is used to shift/rotate the sample points for each pixel. This random shift means that each pixel uses a different

sampling pattern even though only one point set was generated. The random seeds are generated using a Mersenne Twister (Matsumoto, 1998).

With all the necessary support structures created, the first frame of the scene can be rendered. Our renderer utilizes an accumulation buffer approach to combining the sample values into a color estimate for the pixel. In this approach, each frame of the final rendered sequence is composed of *N* intermediate frames in a render process with *N* samples per pixel. Each intermediate frame is rendered using only one sample per pixel, and the results are accumulated over time until the necessary number of samples has been reached. At this point, the result buffer is written to disk as the final rendered frame and the process repeats for the subsequent frame.

## Implementation Details

### Transform Nodes in Scene Graph

As previously mentioned, we insert transform nodes in the scene graph for geometry objects and groups. These transform nodes are used in two ways, namely for geometry instancing and reducing the size of bounding boxes.

Geometry instancing refers to the ability for a renderer to create a virtual copy of a geometry object in a different location without duplicating the model information in memory. In order to create an instance of another object without recreating the object from scratch, a transform node (with the appropriate offsets specified) is created with a reference to the object as one of its children. During ray traversal both the transform node and the object itself are visited, creating two apparent versions of the same object.

The use of transform nodes make motion blur rendering more efficient in ray tracing based on the fact that a geometry object must have a bounding box that encompasses all possible locations that the object can have during rendering (otherwise, some of the rays will miss the object when they should have hit). Rather than having one large bounding box for the whole sequence, a smaller bounding box is used for each frame along with a transform node to place that bounding box in the correct location. The locations of these transform nodes are updated for each frame based on the velocity of their children. This additional cost is greatly outweighed by the speed increases afforded by smaller bounding boxes, due to the fact that a very large number of ray hit/miss calculations are performed for a single frame, and a ray hit or miss can be determined much more quickly when the bounding volume hierarchy can be traversed efficiently.

Sampling and Warping

The usage of the generated point sets was described at a high level in the previous section. Figure 14 shows the device code used to perform the following steps:

- Retrieve seeds generated in the host code based on the pixel location.
- Convert seeds into floating point numbers in the interval [0, 1).
- Retrieve the point set sampling location based on the current accumulation frame count. This count is reset for each rendered frame of the final sequence, and increments for each intermediate frame.
- Shift the three coordinates of the point set sample by the three floating point numbers, ensuring that they wrap around.

```
01      // need three random seeds to shift point set
02      unsigned int seed1 = rnd_seeds[ launch_index ].x;
03      unsigned int seed2 = rnd_seeds[ launch_index ].y;
04      unsigned int seed3 = rnd_seeds[ launch_index ].z;
05
06      // x y z = u v t respectively
07      float xShift = rnd(seed1);
08      float yShift = rnd(seed2);
09      float zShift = rnd(seed3);
10
11      float3 curPoint = point_set[accum_frame];
12      float x = curPoint.x;
13      float y = curPoint.y;
14      float z = curPoint.z;
15
16      x += xShift;
17      if (x >= 1.0f)
18        x -= 1.0f;
19      y += yShift;
20      if (y >= 1.0f)
21        y -= 1.0f;
22      z += zShift;
23     `if (z >= 1.0f)
24        z -= 1.0f;
```

**Figure 14: Device code used to retrieve and shift sampling locations for each pixel**

This process allows for one point set to be used for the entire rendering operation. Generating a unique point set for each pixel is too expensive (especially for the Poisson Disk set), and reusing the same set for all pixels without any modification will result in visible artifacts (Figure 15), and such aliasing artifacts are considered more perceptually objectionable than noise (Cook, 1986).

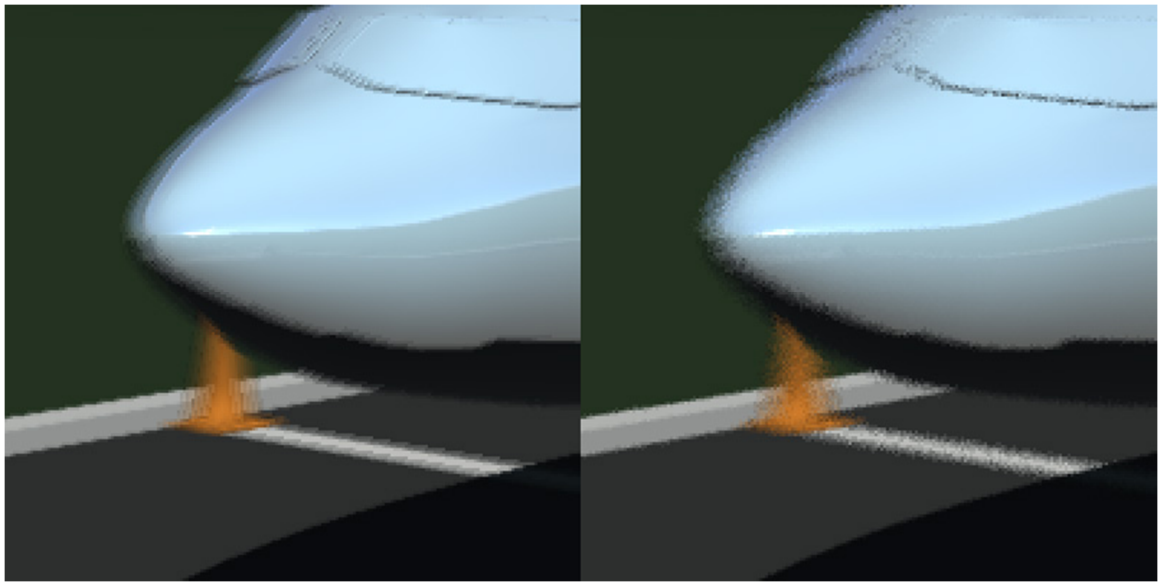


**Figure 15: Aliasing artifacts that result from reusing the same pattern for all pixels (left) versus randomly shifting the point set (right) at 8 samples per pixel.**

Other approaches exist to reducing aliasing with pre-generated point sets, such as using a Wang tiling pattern (Cohen, 2003). Our approach is simple to implement and requires very little additional computational overhead during run-time, while also ensuring that the tiling artifacts are not present.

We also apply a two-dimensional Gaussian warp to the spatial position of the sample, followed by another warp to the temporal position of the sample. Although the spatial warping remained constant throughout our work, we explored the use of two reconstruction filters for the temporal axis, namely a truncated box filter and a triangle filter, as discussed in Chapter 2. Under the sample weighting approach, the value from the filter can be multiplied directly by the value of the sample after scene traversal, but with the warping approach it is necessary to multiply the sample position by the inverse cumulative distribution function of the reconstruction filter prior to traversal so that the warped ray direction is used.

```
01     __device__ void inverseGaussian2D(float& x, float& y)
02    {
03      float tempx = x;
04      float tempy = y;
05      x = (sqrtf( -2.0*log(tempx) )*cos( 2.0*M_PIf*tempy )/2.0f)+0.5f;
06      y = (sqrtf( -2.0*log(tempx) )*sin( 2.0*M_PIf*tempy )/2.0f)+0.5f;
07    }
08
09    __device__ void inverseTruncatedBox(float& x)
10    {
11      x = (0.5f*x) + 0.25f;
12    }
13
14    __device__ void inverseTriangle(float& x)
15    {
16      if (x <= 0.5f)
17        x = 0.5f * sqrt(2.0f*x);
18      else
19        x = 0.5f * (-sqrt(-(2.0f*x)+2) + 2);
20    }
```

**Figure 16: Device code used to perform warping operations on an input parameter.**

Figure 16 shows our implementation of the two-dimensional Gaussian warp based on the Box-Muller transform (Box, 1958), as well as functions to warp a parameter value according to the truncated box and triangle filters.

Discussion

Many of the methods described in this chapter, such as sample set generation and warping, would have been too cumbersome to implement into a commercial renderer, and too constrained for many open source implementations. Since this ray tracer's primary objective was rendering a large number of ray-traced sequences with motion blur, there were a few key goals that the ray tracer needed to meet, including fast batch-controlled rendering and sufficient control to implement the various sampling strategies. A custom ray tracer built using OptiX proved to be a wise choice due to the degree to which it satisfied these goals.

# CHAPTER IV

## EXPERIMENTAL DESIGN

This chapter discusses the goals of this work in terms of experimental measurement. We also discuss our experimental methodology, as well as practical details of our experimental design.

Our primary experimental goal was to determine the relationship between three different variables (scene type, sampling level, and sampling method) and the perception of noise. The reasoning for this goal is to use the insights gleaned from experimental data to make recommendations on perceptually efficient motion blur generation. Although there is prior work in the area of optimal sampling patterns (Mitchell, 1987) and in the area motion perception as a whole (Coletta, 1990), this issue is a challenging issue for multiple reasons:

- The perception of noise changes from person to person based on visual acuity and other factors.
- The design of sampling patterns, reconstruction filters, etc. is often considered more of an art due to the fact that purely mathematical techniques to analyze their optimality do not necessarily indicate perceptual quality.

Due to the relative lack of analysis from the perceptual standpoint in this area, we decided that we would approach this problem using experimental results from user trials. A number of problems in computer graphics such as the design of piecewise cubic reconstruction filters (Netravali, 1998) have been approached in this way.

## Methodology

Our primary analytical tool for measuring perceptual noise is the just-noticeable difference. The just-noticeable difference (hereafter abbreviated as JND) refers to the point at which the subject is able to reliably distinguish two stimuli as being different (Green, 1966). We measure the JND across two parameters, namely scene type and the sampling method. In this way, we are able to determine the effect of those parameters on the perception of noise in the final result.

Each stimulus presented to the subject was a pair of two videos shown side by side. One pair contained videos that were identical, rendered at the highest sampling density (512 samples per pixel) to act as a ground truth. The other pair contained one video at ground truth and another video with the test sampling density. The subjects were then queried to determine which of the two pairs contained the different videos, and told to guess if they did not know the correct answer. As such, subjects are more readily able to answer correctly when the test sampling density is low, since it appears to have much more noise compared to the ground truth version directly next to it. As the test sampling density increases, this difference becomes more difficult to determine.

Measuring the JND requires a large number of queries to the subject in order to adequately estimate the psychometric function, and there are a variety of methods for performing this task. The simple approach used in this work is the method of constant stimuli (Green, 1966). For each combination of scene and sampling method, we query the subject at every test sampling level equally, but in a randomized order. We used eight total sampling densities for testing, namely {1, 8, 27, 64, 125, 216, 343, 512} samples per pixel. These values correspond to the first eight perfect cubes. We chose perfect cubes as the interval due to limiting the dimensionality of our sampling region to

three, and the Stratified Monte Carlo method in particular requires that the number of samples per dimension be a whole number in order to properly subdivide the pixel region into sub-cubes of equal size. These intervals correspond to an equal sampling density in all three dimensions *(u,v,t)* of the pixel. Future work may address varying the sampling density between spatial and temporal sampling to determine their effects on the perceptual quality of motion blur individually, or sampling spatial dimensions individually to determine the relationship of perceptual quality to direction of motion.

The three sampling methods, discussed in detail in Chapter 2, are the Hammersley deterministic point set, the Stratified Monte Carlo point set, and the Poisson Disk point set. Since our work uses a three-dimensional sampling region, the Poisson Disk set is more accurately termed a Poisson Sphere set due to the minimum distance restriction creating a sphere around a given point in which no other samples may appear.

### Scenes

During an initial pilot study phase using four scenes, we noticed that three of the scenes exhibited similar subject responses, while the fourth scene displayed very different behavior. This clustering was due to the fact that the three scenes with similar responses had a lack of strong specular highlights, which were highly prevalent in the fourth scene. As such, we restricted our choice to only two scenes, one with strong specular highlights and one without. The Plane scene (Figure 17) contains mostly diffuse surfaces with some specular reflections, while the Spheres scene (Figure 18) has many more fast-moving reflective surfaces closer to the camera, thereby creating a large number of strong specular highlights.

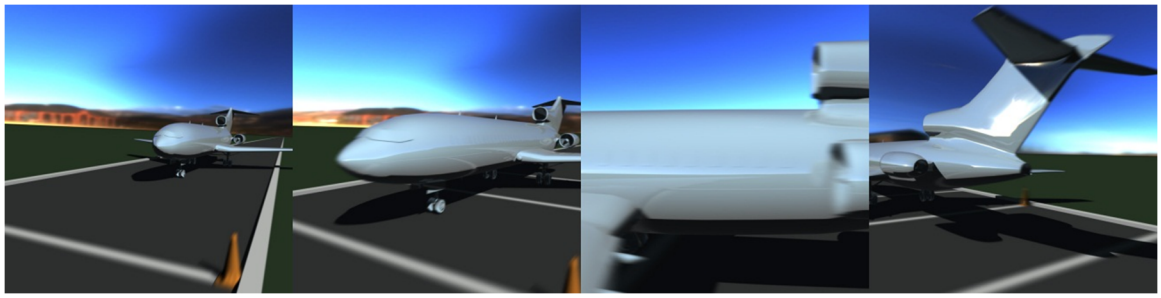


**Figure 17: Four frames from the plane scene animation.**

**Figure 18: Four frames from the spheres scene animation**

Both scenes use high dynamic range (HDR) environment maps for the background texture. Specular highlights are more pronounced under high dynamic range rendering conditions, since the environment map may contain any floating point value, and a large discrepancy between adjacent samples may result. One factor we were interested in testing is the magnitude by which this difference has an impact on the perceptual appearance of noise.

The experimental model follows a repeated measures within-subject design. The order of the different individual stimuli is randomized for each subject in terms of the scenes, point sets, and sampling levels tested. This is done to make it difficult for the subject to predict or find patterns in the stimuli.

## Practical Details

Twelve subjects were shown stimuli corresponding to each combination of two scenes, eight sampling levels, and three sampling methods. The truncated box time filter was used for these stimuli. This yields a total of 48 different combinations per subject. Each combination was tested four times. As a result, the value for each combination for a particular subject will either be 0, 0.25, 0.5, 0.75, or 1, and this value represents the percentage of time that the subject was able to correctly discern the difference in the stimuli for that combination. Therefore we expect to see values close to 1 for low sampling levels, and values close to 0.5 for high sampling levels as the correctness of their answers approaches random chance. Across all 12 subjects, a particular combination is tested a total of 48 times.

We choose the value of 0.75 as our JND threshold, since it corresponds to the halfway point between being able to decipher the difference 100% of the time and random chance.

The total experiment runtime per subject was limited to two hours, and on average the subjects completed all trials within an hour and 45 minutes. The subjects were allowed to take short breaks at regular intervals throughout this time to reduce fatigue.

The experiment code, written in MATLAB, performed the ordering randomization and data gathering procedures along with launching the video player for each stimulus. The video player needed to be advanced enough to play two videos side by side in a synchronized fashion, while also being configurable enough to initiate playback automatically and hide window decorations. We used RV (Tweak Software, San Francisco, CA) since it satisfied these goals.

The monitor was a Dell U2311H, which features a 23-inch diagonal display size with resolution of 1920x1080 along with an IPS display panel for improved color reproduction, wider viewing angles, and reduced ghosting. Ghosting in particular was an important aspect of the monitor for this project, since it blurs nearby frames of animated video, which conflicts with the very factors we were attempting to test. Ghosting is a byproduct of high display response time, which is a measurement of how quickly the panel pixels can change from one color to another. Compared to the IPS panel, other panel types (TN and PVA) did not provide as adequate of a balance between color reproduction, viewing angles, and response time.

The videos themselves were 768x768 pixels, and were displayed in the RV video player at 1:1 pixel scaling to avoid the impacts of any resampling. The videos were preloaded into memory for each stimulus and shown in uncompressed format to avoid any color banding or compression artifacts.

# CHAPTER V

# EXPERIMENTAL RESULTS

This chapter presents the results of our experiments. The relationship between the JND and the scene will be shown, followed by the relationship between the JND and the point set used for sampling. In both cases, the JND is determined by the point at which the psychometric function intersects the horizontal line for the threshold, which is $p = 0.75$. Additionally, for each graph error bars represent the standard error of the mean to indicate the range of variation of each calculated data point. We then present an analysis of the results.

## Scene Dependence

This section shows how the JND depends on the scene shown. We previously discussed the two scenes used for testing, and the rationale for their selection.

We will now present graphs for each scene showing the probability of correctly identifying the different videos as a function of the sampling level. Figure 19 is the graph for the plane scene, and Figure 20 is the graph for the spheres scene. The dashed red line shows our JND threshold ($p = 0.75$) and the dashed green line shows the random chance level ($p = 0.5$). The error bars shown represent the standard error of the mean for $N = 12$ trials.

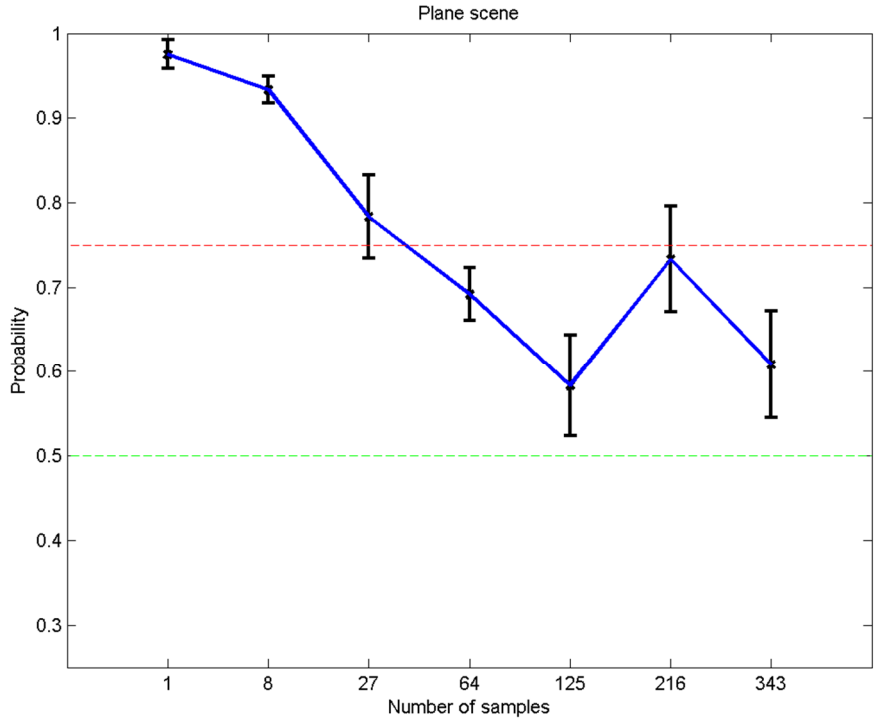



**Figure 19: Psychometric function for plane scene. Error bars show standard error of the mean.**

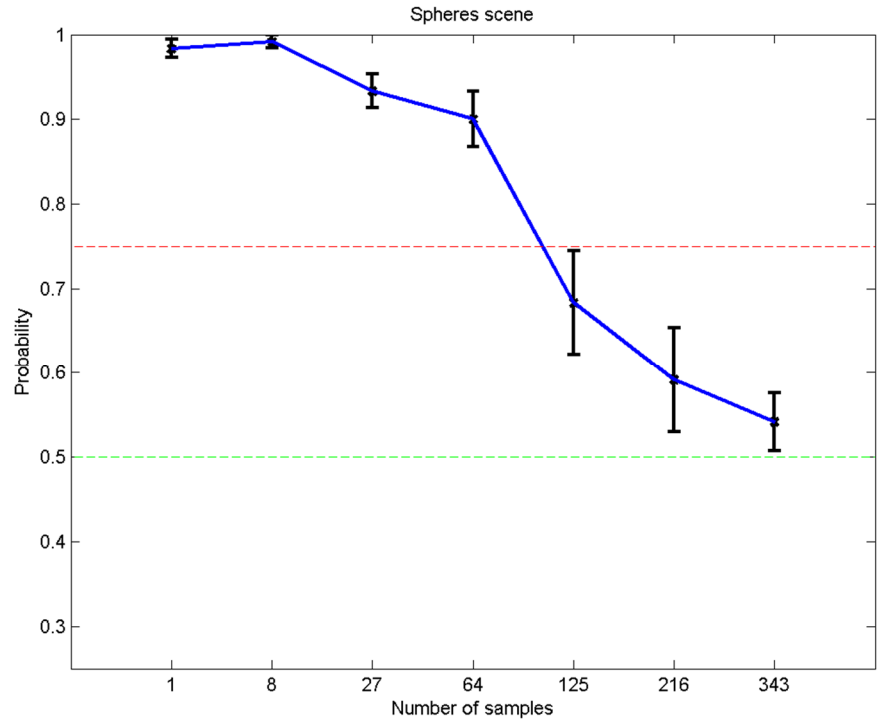



**Figure 20: Psychometric function for spheres scene. Error bars show standard error of the mean.**

It is immediately visible that the JND appears higher in the spheres scene than the plane scene, based on the intersection of the psychometric function with our JND threshold. This intersection occurs at roughly 40 samples per pixel in the Plane scene, and at roughly 100 samples per pixel in the Spheres scene. We will perform an analysis of this difference later in this chapter.

## Point Set Dependence

This section shows how the JND depends on the point set used for sampling. In Chapter 2, we discussed the three point sets tested, and the rationale for their selection.

We will now present graphs for each point set showing the probability of correctly identifying the different videos as a function of the sampling level. Figure 21 presents the Hammersley point set, Figure 22 presents the Stratified Monte Carlo point set, and Figure 23 presents the Poisson point set. The error bars shown represent the standard error of the mean for N=12 trials.

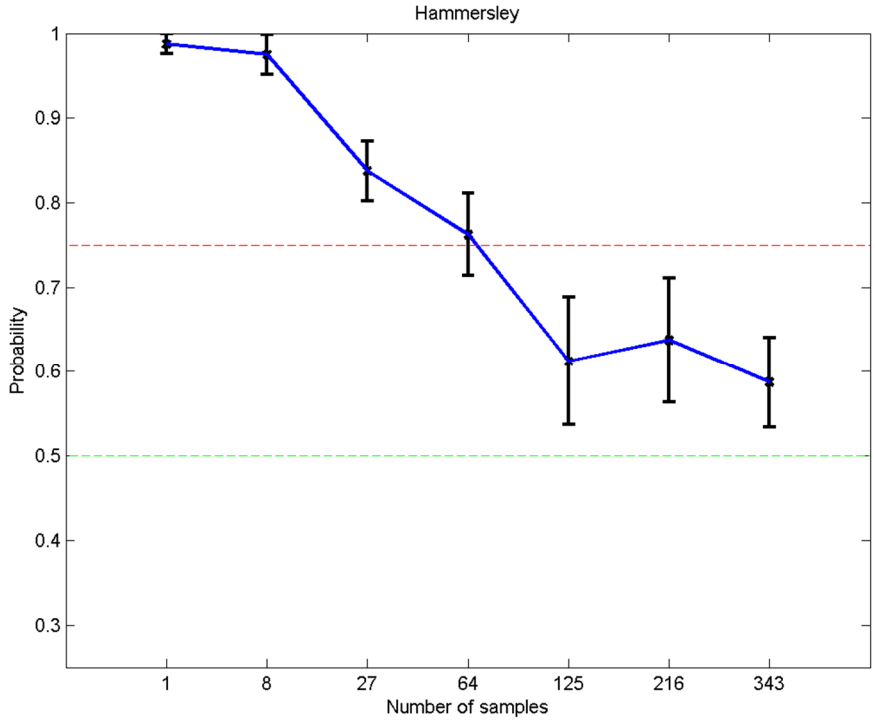



**Figure 21: Psychometric function for Hammersley point set.**

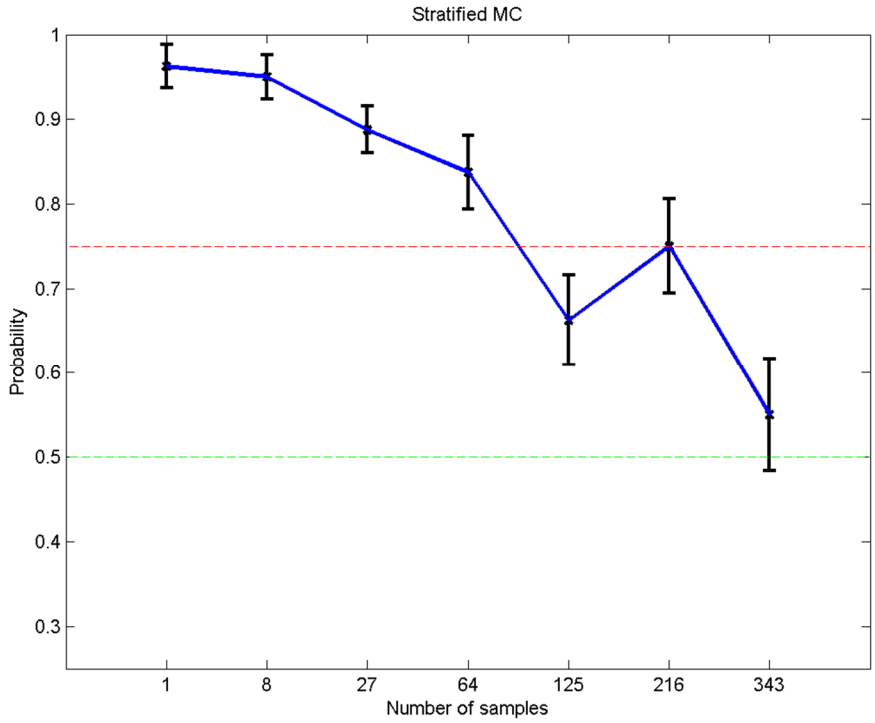



**Figure 22: Psychometric function for Stratified Monte Carlo point set.**

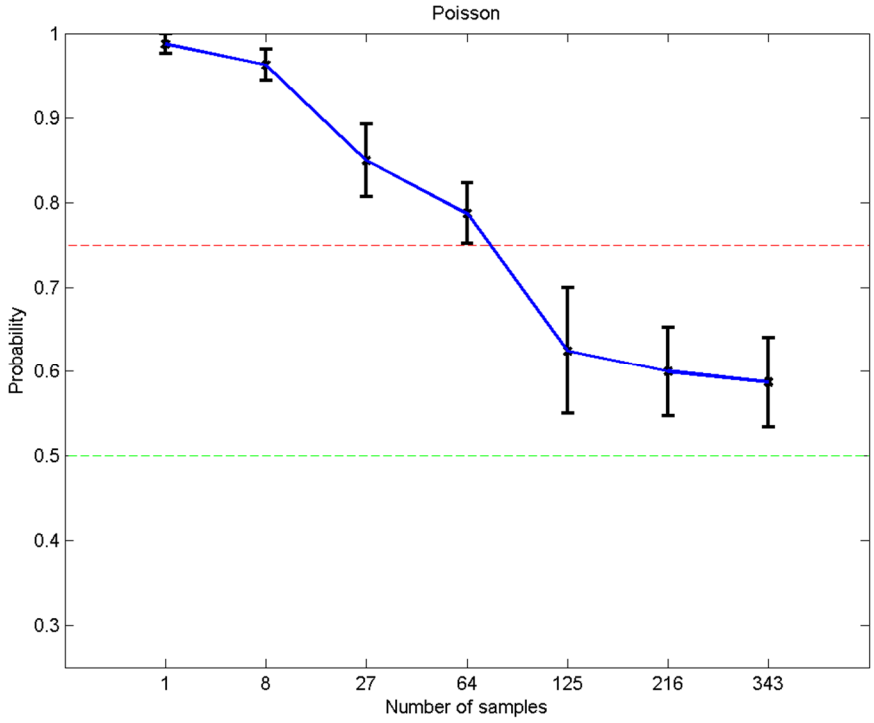



**Figure 23: Psychometric function for Poisson Disk point set.**

Unlike the scene dependence, the relationship between point set and JND appears weak. We will discuss this more fully later in the following section. Additionally, there is an anomaly with the Stratified Monte Carlo point set at 216 samples per pixel, and upon further investigation we were unable to discern any reason for this either in the MATLAB experiment code or in the video rendering process. Therefore we are unsure whether the anomaly would decrease with additional trials or if there is an underlying cause for its existence. In either case, the anomaly appears at a higher sampling density than the area containing the JND for all scenes and point sets, so our analysis will not be substantially impacted by its presence.

We now present the above results in an overlaid fashion in order to see the graphs of the different techniques at once. Figure 24 shows all of the scenes, while Figure 25 shows all of the point sets.

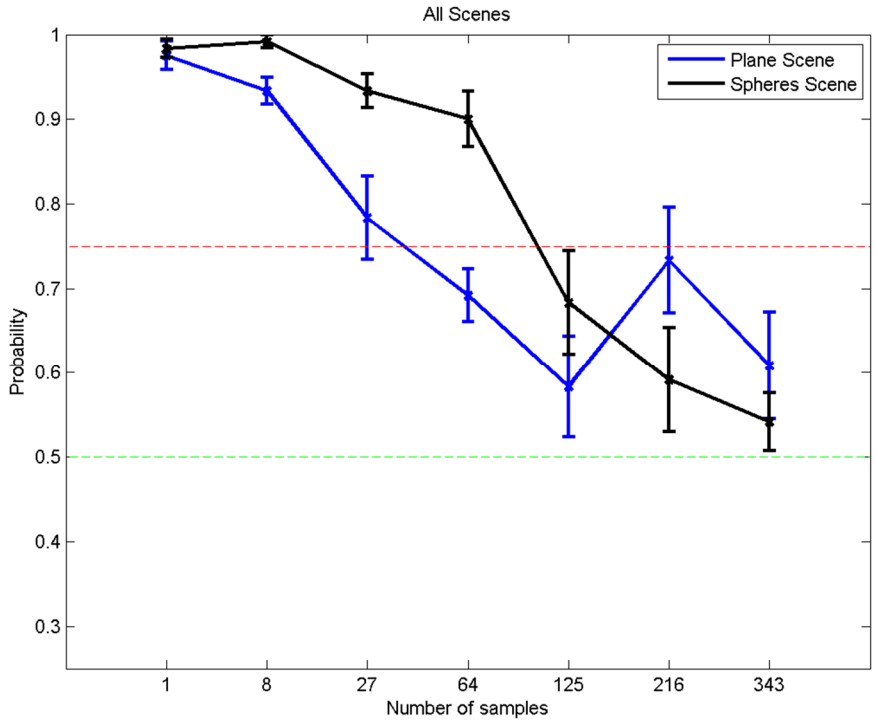



**Figure 24: Psychometric functions for both scenes.**

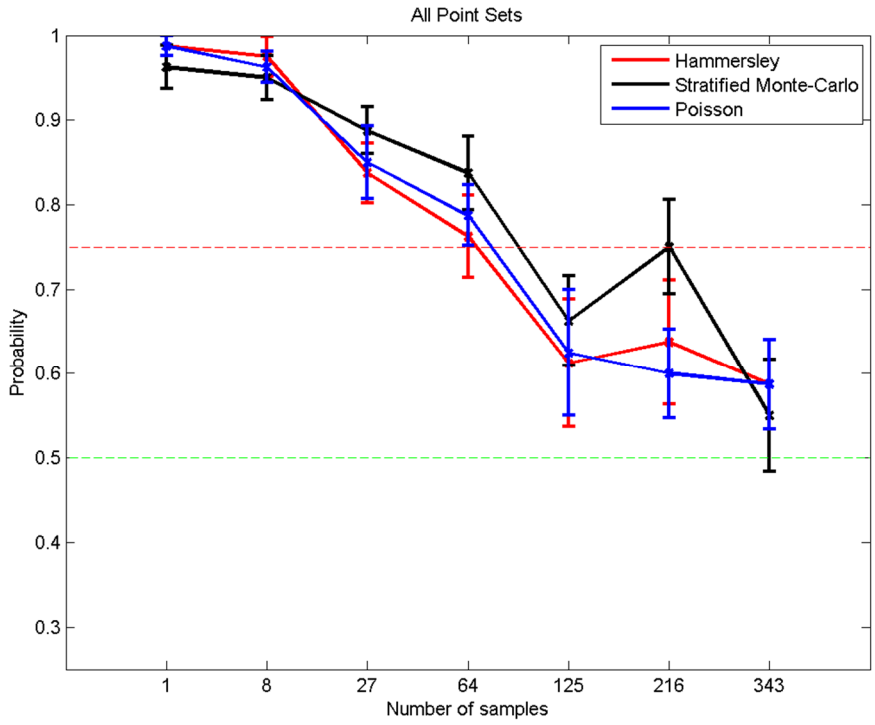



**Figure 25: Psychometric functions for all three point sets.**

We perform t-tests in order to calculate statistical significance of the data, both across scenes (Table 1) and across point sets (Table 2). We use the three data points near the JND level, namely 27, 64, and 125 samples per pixel (abbreviated as spp). We also include 216 samples per pixel in Table 1 to test the anomaly described in the previous section.

**Table 1: Results of t-tests for determining statistical significance between scenes**

| Test (Degrees of Freedom = 29) | Results of t test | | |
|---|---|---|---|
| | p value | t value | Significant (p = 0.05) |
| Plane Scene vs. Spheres Scene, 27 spp | 0.0027 | -3.275 | Yes |
| Plane Scene vs. Spheres Scene, 64 spp | 0.000025 | -5.000 | Yes |
| Plane Scene vs. Spheres Scene, 125 spp | 0.1296 | -1.560 | No |
| Plane Scene vs. Spheres Scene, 216 spp | 0.0477 | 2.067 | Yes |

**Table 2: Results of t-tests for determining statistical significance between point sets**

| Test (Degrees of Freedom = 19) | Results of t test | | |
|---|---|---|---|
| | p value | t value | Significant (p = 0.05) |
| Hammersley vs. Stratified Monte Carlo, 27 spp | 0.2141 | -1.285 | No |
| Hammersley vs. Stratified Monte Carlo, 64 spp | 0.2492 | -1.189 | No |
| Hammersley vs. Stratified Monte Carlo, 125 spp | 0.5190 | -0.6571 | No |
| Hammersley vs. Poisson, 27 spp | 0.8252 | -0.2239 | No |
| Hammersley vs. Poisson, 64 spp | 0.6810 | -0.4175 | No |
| Hammersley vs. Poisson, 125 spp | 0.8805 | -0.1523 | No |
| Stratified Monte Carlo vs. Poisson, 27 spp | 0.4194 | 0.8254 | No |
| Stratified Monte Carlo vs. Poisson, 64 spp | 0.2967 | 1.073 | No |
| Stratified Monte Carlo vs. Poisson, 125 spp | 0.6513 | 0.4592 | No |

The statistical tests indicate that the scene has a greater impact on JND than the point set. The JND for the plane scene occurs close to halfway between 27 and 64 samples per pixel, while the JND for the spheres scene occurs close to 125 samples per pixel. We found no statistically significant difference between point sets, with the JND for all three point sets occurring between 64 and 125 samples per pixel. In order for a particular point set to be more perceptually efficient, the probability of distinguishability would have to decrease more quickly as sampling density increases, thereby indicating that it performs more closely to ground truth at a particular sampling density than other point sets. The p-values for the statistical tests indicate that the Hammersley and Poisson sampling methods exhibited the most similar performance out of any two of the three point sets. However, the differences between the sampling methods are small enough that a larger number of user trials would be needed to prove any statistical significance.

# CHAPTER VI

# CONCLUSION

This work explored the perception of motion blur in distribution ray tracing through the use of experimental trials designed to isolate various techniques for generating motion blur in order to determine their effectiveness. Our experimental trials tested the perception of motion blur under two scenes and three sampling patterns used for sub-pixel sampling. Our results indicate that the choice of scene, and in particular the presence of strong specular highlights, has a larger impact on the visibility of noise than the choice of sampling pattern. The just-noticeable difference of noise is determined to be roughly 40 samples per pixel for primarily diffuse scenes, and roughly 100 samples per pixel for scenes with strong specular highlights and HDR rendering. Although pre-examination of the rendered images yielded a hypothesis that the Hammersley point set produces the least motion blur noise, we were unable to find a statistically significant difference beyond a slight correlation to confirm this hypothesis.

## Contributions and Future Work

This work's main contribution is a perceptual evaluation of various approaches to ray-traced motion blur. We performed this evaluation by finding the just-noticeable difference of noise detectable by humans in various settings, with different scenes and point sets used for rendering. This work contributes to the understanding of human

perception of object motion by giving estimates of the sampling densities required to achieve low perceptual noise under these different situations.

There is a variety of future work related to this project, such as testing additional sampling point sets (especially in the realm of quasi-Monte Carlo sampling). Although we implemented the box and triangle time filters into our renderer, the experiments discussed in this work did not seek to identify which time filter produced more perceptually pleasing motion blur. Additionally, we assumed in this work that the (*u*,*v*,*t*) sub-pixel dimensions should be sampled with equal density. Future work should address whether spatial or temporal anti-aliasing has a larger impact on the perceptual quality of the result. Furthermore, although we constrained the dimensionality of our point set to three, there is a large potential in exploring adding additional dimensions (for depth-of-field, global illumination, area lights, etc.) and performing similar experiments to determine perceptually optimal sampling techniques under such conditions.

# REFERENCES


Akenine-Möller, T., Munkberg, J., Hasselgren, J. Stochastic Rasterization using Time-Continuous Triangles. Graphics Hardware 2007, 7–16.

Amanatides, J., Woo, A. A fast voxel traversal algorithm for ray tracing. Proceedings of Eurographics 1987, 3-10, 1987.

Boulos, S., Edwards, D., Lacewell, J. D., Kniss, J., Kautz, J., Shirley, P., Wald, I. Interactive Distribution Ray Tracing. Technical Report, SCI Institute, University of Utah, No. UUSCI-2006-022, 2006.

Box, G. E. P, Muller, M. E. A note on the generation of random normal deviates. Annals Math. Stat, V. 29. 610-611. 1958.

Cammarano, M., And Jensen, H. W. Time Dependent Photon Mapping. Eurographics Symposium on Rendering, 135–144, 2002.

Catmull, E. Computer Display of Curved Surfaces. Proc. IEEE Conference on Computer Graphics, Pattern Recognition and Data Structures. 1974.

Cohen, M., Shade, J., Hiller, S., Deussen, O. Wang Tiles for Image and Texture Generation. ACM Transaction on Graphics (Proceedings of SIGGRAPH 2003), 22, 3, 287-294, 2003.

Coletta, N. J., Williams, D. R., Tiana, C. L. M. Consequences of spatial sampling for human motion perception. Vision Research, 30, 1631-1648, 1990.

Cook R. L., Porter T., Carpenter L. Distributed Ray Tracing. Computer Graphics (Proceedings of SIGGRAPH 1984) 18, 3, 137–144. 1984.

Cook, R. L. Stochastic sampling in computer graphics. ACM Transactions on Graphics 5, 1, 51–72, 1986.

Dunbar, D., Humphreys, G. A spatial data structure for fast poisson-disk sample generation. ACM Transactions on Graphics 25, 3, 503-508. 2006.

Egan, K., Tseng, Y., Holzschuch, N., Durand, F., Ramamoorthi, R. Frequency analysis and sheared reconstruction for rendering motion blur. ACM Transactions on Graphics 28, 3, 2009.

Ernst, M., Stamminger, M., Greiner, G. Filter Importance Sampling. IEEE Symposium on Interactive Ray Tracing, 125-132, 2006.

Foley, J., Van Dam, A. Fundamentals of Interactive Computer Graphics. Addison-Wesley, 1982.

Green, D. M., Swets, J. A. Signal Detection Theory and Psychophysics. John Wiley and Sons. 1966.

Hachisuka, T., Jarosz, W., Weistroffer, R., Dale, K., Humphreys, G., Zwicker, M., Jensen, H. Multidimensional Adaptive Sampling and Reconstruction for Ray Tracing. ACM Transactions on Graphics (SIGGRAPH) 27, 3, 33:1–33:10. 2008.

Haeberli, P., Akeley, K. The Accumulation Buffer: Hardware Support for High-Quality Rendering. Computer Graphics (Proceedings of SIGGRAPH 1990), 24, 309–318. 1990.

Kajiya, J. T. The Rendering Equation. In Computer Graphics (Proceedings of ACM SIGGRAPH), 143–150. 1986.

Kay T. L., Kajiya, J. T. Ray tracing complex scenes. Computer Graphics (Proceedings of SIGGRAPH 1986) 20, 4, 269-278. 1986.

Keller, A. Quasi-Monte Carlo Methods for Photorealistic Image Synthesis. Ph. D. thesis, Shaker Verlag Aachen, 1998.

Kollig T., Keller A. Efficient Multidimensional Sampling. Computer Graphics Forum (Proceedings of Eurographics 2002) 21, 3, 557–56, 2002.

Korein, J., And Badler, N. Temporal Anti-Aliasing in Computer Generated Animation. In Computer Graphics (Proceedings of SIGGRAPH 1983), 17, 377–388, 1983.

Lagae, A., Dutré, P. A comparison of Methods for Generating Poisson Disk Distributions. Report CW 459, Department of Computer Science, K.U.Leuven. 2006.

Matsumoto, M., Nishimura, T. Mersenne Twister: a 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator. ACM Transactions on Modeling and Computer Simulation 8 (1): 3-30, 1998.

Max, N. L., Lerner, D. M. A Two-and-a-Half-D Motion-Blur Algorithm. Computer Graphics (Proceedings of SIGGRAPH 1985), 19, 85–93, 1985.

McGuire, M., Enderton, E., Shirley, P., Luebke, D. Real-time stochastic rasterization on conventional GPU architectures. High Performance Graphics 2010: 173-182.

Mitchell, D. P. Generating antialiased images at low sampling densities. Computer Graphics (Proceedings of SIGGRAPH 1987), 65–72, 1987.

Mitchell, D. P. Spectrally optimal sampling for distributed ray tracing. Computer Graphics (Proceedings of SIGGRAPH 1991), 157–164.

Mitchell, D. P., Netravali, A. Reconstruction Filters in Computer Graphics. Computer Graphics (Proceedings of SIGGRAPH 1988), 22, 4, 221-228, 1988.

Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, R., Robison, A., Stich, M. OptiX: A General Purpose Ray Tracing Engine. ACM Transactions on Graphics 29, 4, 2010.

Pharr, M., Humphreys, G. Physically Based Rendering: From Theory to Implementation. Morgan Kaufmann. 2004.

Potmesil, M., Chakravarty, I. Modeling Motion Blur in Computer-Generated Images. Computer Graphics (Proceedings of SIGGRAPH 1983), ACM, vol. 17, 389–399, 1983.

Roth, S. Ray Casting for Modeling Solids. Computer Graphics and Image Processing 18, 109-144. 1982.

RV, Tweak Software, San Francisco, CA. www.tweaksoftware.com.

Shirley, P. Discrepancy as a Quality Measure for Sample Distributions. Proceedings of Eurographics 1991, 183-193, 1991.

Sung, K., Pearce, A., Wang, C. Spatial-Temporal Antialiasing. IEEE Transactions on Visualization and Computer Graphics 8, 2, 144–153, 2002.

Whitted, T. An improved illumination model for shaded display. Comm. ACM, 23, 6, 1980, 343-349.

Yellott, J. I. Spectral consequences of photoreceptor sampling in the rhesus retina. Science 221, 382–385, 1983.