The Dissertation Committee for Peter C. Djeu
certifies that this is the approved version of the following dissertation:

High Quality, High Performance Rendering Using Shadow Ray Acceleration and Aggressive Micropolygon Tessellation Rates

Committee:

Donald S. Fussell, Supervisor

William R. Mark

Keshav Pingali

C. Greg Plaxton

Peter Shirley

High Quality, High Performance Rendering Using Shadow Ray Acceleration and Aggressive Micropolygon Tessellation Rates

by

Peter C. Djeu, B.A., M.S.C.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2011

Dedication

To Hong, Tuck, and Jack.

A million thanks for being my loving family.

Acknowledgments

This work would not have been possible without the hard work of the following people and organizations.

Don Fussell, Bill Mark, Gordon Stoll

Warren Hunt, Sean Keely, Rui Wang, Ikrima Elhassan, Jeffery A. Williams

Brent Burley, Dylan Lacewell, Keenan Crane, Anjul Patney

Don Porter, Manny Ko, Raven Starsmore

Techland Corporation, Valve Corporation, headus (metamorphosis) Pty Ltd, Blender Foundation, Stanford 3D Scanning Repository

Graphics and Parallel Systems Group at UT Austin

Intel Corporation

High Quality, High Performance Rendering Using Shadow Ray Acceleration and Aggressive Micropolygon Tessellation Rates

Publication No. _____

Peter C. Djeu, Ph.D.
The University of Texas at Austin, 2011

Supervisor: Donald S. Fussell

Rendering in computer graphics is the process of converting a three dimensional scene description into a two dimensional image. In this work we focus on high quality rendering, which has numerous applications in entertainment and visualization. Many films today are created either entirely or in concert with computationally generated imagery and serve as a vivid example of the benefits of high quality rendering.

This dissertation consists of two parts, each presenting novel work in the field of high quality, high performance rendering. The first part proposes the use of *volumetric occluders*, or a collection of axis-aligned boxes placed within a polygonal

model, to accelerate the rendering of shadows cast by the model while producing images identical to the unaccelerated baseline. We show that our approach performs well on single object scenes and extend our approach for use with scenes from a professional open source movie. Although the technique has not yet proven itself on these multi-object scenes, we identify the scene characteristics which are hampering the approach and show that in some cases it is still possible to achieve an improvement in performance.

The second part of the dissertation presents a new way to determine micropolygon tessellation rate within a Reyes style renderer. Our new scheme, called *final approach tessellation*, evaluates the tessellation rate close to a Reyes surface rather than upon entry into its bounding box. Our determination of the tessellation rate is more aggressive than previous approaches, producing a more compact tessellation which in turn is faster to compute and requires less memory. Our evaluation shows that although final approach tessellation is promising in theory, it ultimately fails to improve performance on actual test scenes.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Rendering, the computational process which converts a three dimensional scene description into a two dimensional image, is a fundamental algorithm in computer graphics. Rendering is the computational analog to classic photography. Photography involves the creation of images from a scene in the physical world while rendering involves the creation of images from a scene described within a computer. Rendering is used for a variety of purposes in scientific visualization, interactive media, and entertainment. This dissertation will focus on improving the rendering process for the entertainment medium, particularly with regard to computer generated (CG) movies with cinematic quality.

Since rendering algorithms are computational processes, they must often make a tradeoff between high performance and high quality. For example, certain renderers prioritize their frame rate (i.e. high performance) and give up the ability to support visual effects that would exceed the frame time budget (high quality). These renderers are suited for tasks that require immediate visual response, and are called real-time or interactive renderers. Other renderers, in contrast, prioritize support for a rich set of visual effects (i.e. high quality) first and foremost, with high performance as a secondary, but still desired, goal. These renderers are suited

for tasks that require high-end visual fidelity, such as state-of-the-art CG movie production where images (also called *frames*) are generated at non-interactive speeds and then played back at a smooth frame rate. These renderers are called offline or batch renderers.

In this work, we focus primarily on techniques that will improve offline renderers, and as such our algorithmic priorities are high quality *and then* high performance. We want to discover new algorithms that first and foremost preserve or improve image quality, and as a secondary concern improve performance. We explicitly state high performance as a concern to rule out rendering algorithms that solely produce a complicated visual result without regard for computational expense. Although it is not an exact minimization process, we constantly keep in mind the visual quality that is gained by our expenditure of computation, and try to be economical in our spending.

1.1 Ray Tracing as a Means of Rendering

Ray tracing is a simple, elegant algorithm for generating high quality images in almost any rendering situation [App68, Whi80, Gla89, PH10]. In situations where other techniques fail to produce the correct result, ray tracing simply requires the evaluation of more ray queries. Ray tracing is one of two options for rendering, while the other option is known as *rasterization*. We will not discuss rasterization in detail, but various references are available [Wat00]. This dissertation will instead use ray tracing as the sole means of rendering and our work will focus on improving the process of ray tracing.

A *ray* is the fundamental building block for ray tracing. Rays are computational constructs that represent a *visibility query*, which is a query about the nature or number of objects that lie between two points A and B. Rays emanate from one part of the scene and go to another. A ray queries a database of objects in the scene

and can ask one of two questions 1) what is the first object on the path from A to B? or 2) are there any objects between A and B? We refer to these queries, respectively, as *first hit* and *any hit* visibility queries, which are shown in Figure 1.1.

Like photography, which uses a real world camera, ray tracing places a virtual camera within the digital scene description. When rays are used to determine things directly visible to this camera, they are called *primary rays*. These rays emanate from the lens of the camera. A ray that determines whether a point in the scene is in shadow or not is called a *shadow ray*. One of the two endpoints from these rays touches a light source in the scene. Ray tracing also makes use of reflection and refraction rays, although these ray types are not studied in this dissertation. Figure 1.1 provides an overview of the ray types.

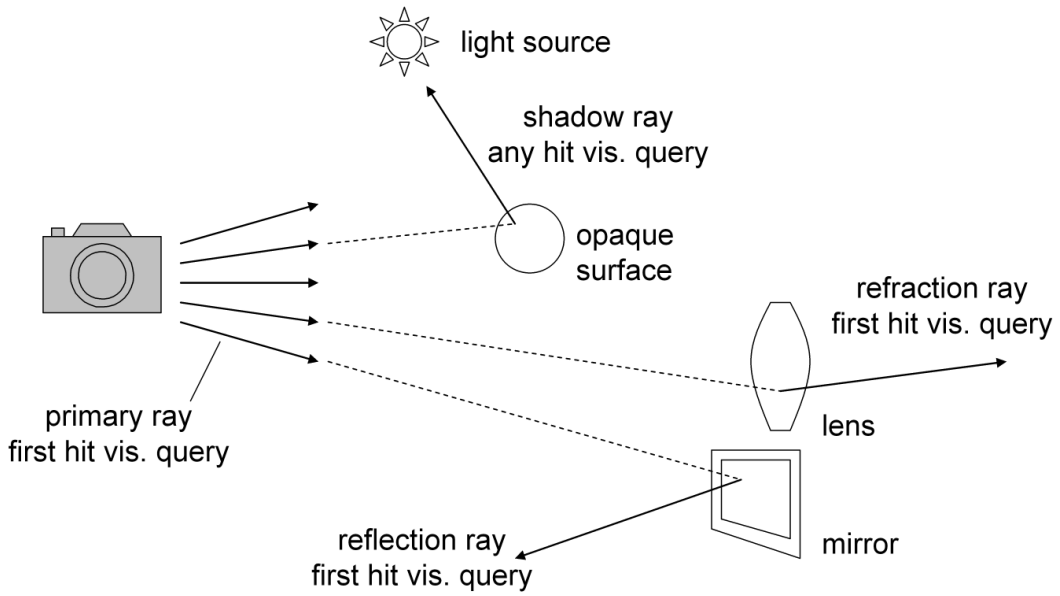


Figure 1.1: Primary rays emanate from the virtual camera and encounter three different types of objects in the virtual scene. Shadow, reflection, and refraction rays are spawned from primary rays and are collectively known as secondary rays.

1.2 Evaluation Methodology

The overall experimental methodology of this dissertation is to measure run-time performance while providing an understanding of the underlying causes of the improvement. We will measure final run-time but we will also decompose the computation into its component operations. When possible, we will analyze the decomposition to provide a greater understanding of when our proposed algorithmic changes improve performance and for what reason.

As a general rule, we will only explore algorithmic changes that preserve the quality of the generated image while improving final run-time and operation count. We choose this overall approach in order to avoid advocating algorithmic changes that increase performance at the cost of image quality.

1.3 Goals of this Two Part Dissertation

This thesis is divided into two parts. The first part is an in-depth study of accelerating shadow rays using a new approach we have developed called *volumetric occluders*. Our approach is exact, meaning there is no loss in image quality to our generated shadows compared to the ground truth, and we show empirically that rendering times decrease from previous best practice. This approach, which is exact and also faster, fits well with our overall goals of high quality, high performance rendering. In this work we also measure both operation counts and runtimes in real scenes and show that this approach is promising both in theory and in practice. We show that the idea is sound and can show very satisfactory performance improvement on simple test scenes. We move on to substantially more realistic test scenes taken from an actual CG movie production and show that performance improvement is less impressive in these scenes, and in some cases, there is performance degradation. We discuss the reasons for this degradation.

The second part of this thesis describes a proposed improvement to a micropolygon renderer that uses rays and ray differentials. We propose a more aggressive method of determining the tessellation rate of the surfaces encountered by the rays being traced. We call our approach *final approach tessellation*. We tessellate according to the Reyes quality threshold [CCC87], which is the standard for high quality CG movie rendering. Our proposed improvement seeks to improve the rendering time while adhering to the Reyes quality threshold, which means this work represents a second venture into the world of high quality, high performance rendering. We provide operation counts that represent the performance implications of final approach tessellation and show that it does provide the improvement that we suspected it would. However, we discover over the course of this work that there is not enough compelling evidence to continue pursuit of our approach because of a lack of runtime improvement.

Since the two domains that we are studying are quite fundamental to image generation, we believe that this dissertation presents further knowledge and insight within the important field of high quality, high performance rendering.

1.4 Summary of Results

Overall, this dissertation presents results for two new approaches to high quality, high performance rendering. Volumetric occluders, the first of these two approaches, has shown itself to work in ideal conditions, to work somewhat on a handful of off-the-shelf production-caliber movie scenes, and is poised to become even more relevant in the future due to recent trends in modeling and rendering.

The second new algorithm, final approach tessellation, does not show as much promise. Our work provides a detailed study into a theoretically interesting improvement on an existing standard. We quantify this improvement by measuring operation counts. We show, however, that the reduction in operation count does not

translate into final runtime improvement and recommend against using our approach in the future. Our study, although conveying a mostly negative result, does provide an important capstone to a line of research that advocated use of final approach tessellation.

1.5 Thesis Organization

The rest of this dissertation is organized as follows. Chapter 2 covers necessary background for understanding volumetric occluders. Chapter 3 describes our proposed volumetric occluder technique in detail and also presents an initial proof of concept study. We further explore the applicability of volumetric occluders by working with scenes from an actual CG movie in Chapter 4. Chapter 5 presents the background for our work on final approach tessellation, while Chapter 6 presents a detailed description of our approach as well as empirical results exploring its viability. Finally, Chapter 7 provides our closing thoughts on the usefulness and potential future applicability of our work.

Chapter 2

Background for Volumetric Occluders

This chapter describes the background and motivation for using volumetric occluders. We first cover the basics of rendering and visualizing shadows, then cover previous approaches to shadow rendering, and finally conclude with a discussion of how volumetric occluders compare to previous work. The goal of this chapter is to describe volumetric occluders within the greater context of research in high quality, high performance offline rendering.

2.1 Rendering, Rasterization, and Ray Tracing

There are many variations of computational *rendering*, or converting a three dimensional scene description into an two dimensional image. We choose to look at ray tracing [App68, Whi80, Gla89, PH10] in a form that allows for the most general type of rendering known today, as expressed in Kajiya's seminal rendering equation work [Kaj86]. This form of rendering is elegant in its simplicity (more ray samples leads to better images) and the algorithm converges to a ground truth consistent

with the model of light used in computer graphics.

Ray tracing is typically considered expensive compared to the other methods of rendering, particularly rasterization, which is frequently used for graphics applications that require real-time responsiveness. However, with increasing computational power, many professional quality *renderers* have switched from being rasterizer based to being ray tracer based, including Blue Sky [Stu11], DreamWorks [TL04], Pixar [CFLB06], and Sony Imageworks [HF10]. Ray tracing is quickly become the method of choice for offline, high quality rendering.

2.2 Rays in Ray Tracing

The fundamental operation in ray tracing is the evaluation of geometric objects called *rays*. The entire approach is in fact named after these geometric objects, which consist of an origin (a point), a direction vector, and a maximum distance. When the maximum distance is set to a finite value the ray represents a geometric line segment, with endpoints A and B. We will deal exclusively with these type of finite length rays. Even when rays technically are semi-infinite, a maximum distance can be imposed on them based on the overall bounds of the scene specification. When we refer to *bounds* (or *bounding boxes*), we refer to a three-dimensional rectangular box whose sides are all aligned to the three axes of the world coordinate space.

For the purposes of ray tracing, a ray also represents a *visibility query*, which comes in two forms. The first form asks, "what is the first occluder along the straight line from point A, the ray origin, to point B, the ray destination?" These queries are used for *primary visibility* (i.e. objects directly visible from the camera in zero optical bounces), and for assembling color information for certain *secondary visibility* effects like reflection and refraction. We refer to the first form of visibility as a *first hit visibility query*.

The second form of the visibility queries asks a simpler yes or no question,

"are there any occluders along the straight line from point A to point B?" These queries are commonly used for evaluating lighting conditions such as which points in the scene are in shadow. We refer to the collective location and brightness (or dimness) of shadows as *shadowing effects*. Our work in this chapter will focus on the second type of query, the *any hit visibility query*, which is expressed in ray form as a *shadow ray*.

A shadow ray is used to determine if any objects lie along the line segment connecting two arbitrary points in the scene. One endpoint of the shadow ray is typically located at a light source and the other endpoint is located at the surface whose lighting intensity is actively being determined. The objects that lie on the shadow ray are called *occluders* since they block the transmission of light between the two endpoints. A shadow ray is the fundamental query that is used in the generation of the hard and soft shadows that appear in a ray traced image.

The shadow ray is typically cast from the shading point to the light source to determine if the shading point is illuminated by the light (Figure 1.1). The shadow ray can also be cast in the reverse direction due to the symmetric nature of evaluating visibility.

Tracing shadow rays is an extremely fundamental operation in ray tracing. Shadow rays produce the *shadowing effects* in a scene which are nearly the most important visual feature in the scene, second only to primary visibility. Shadow-based cues are extremely important for understanding the structure of a scene (Figure 2.1).

Shadow rays require significant computational effort to evaluate, in particular due to their abundance, and our work focuses on reducing the effort. We differ from previous techniques in that we preserve the visual quality of the final image, which to our knowledge is unique among techniques for accelerating shadow ray evaluation.

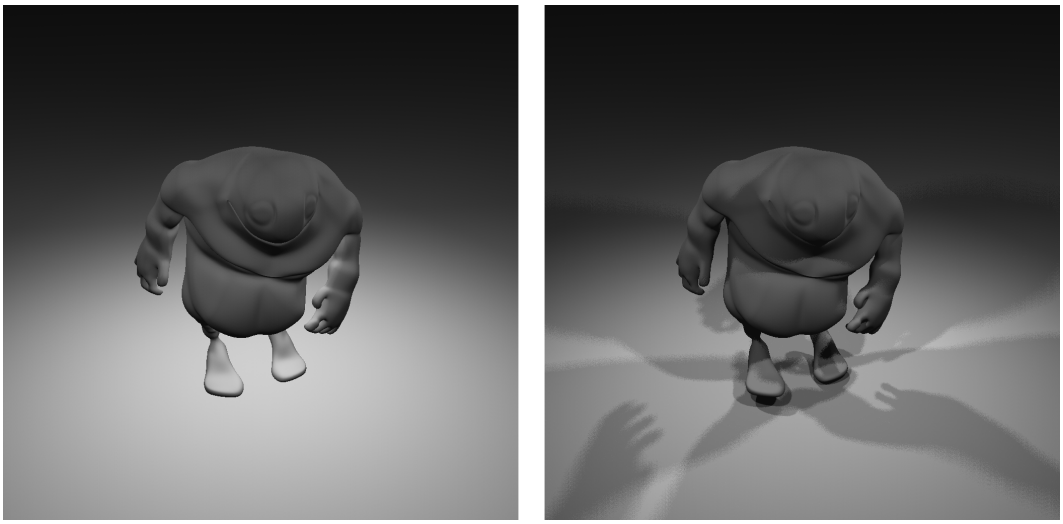


Figure 2.1: Shadows provide important information about shape that would otherwise be unavailable, such as detail around the character's fingers.

2.3 Monte Carlo Ray Tracing

We generate the shadows in our images using Monte Carlo sampling, which is a robust sampling-based technique for determining the shadows cast by area lights. Points on the light are randomly sampled from the surface point whose shading color needs to be determined. This surface point is called the *shading point*. At each shading point, the color is governed by the percentage of the area light that is visible to that point, which is approximated by the percentage of shadow rays that strike the light rather than an occluder. As the number of shadow ray samples increases, the computed lighting color *converges* to the actual analytic result.

2.4 Motivation

The visual effects that can be generated by shadow rays, such as hard and soft shadows, are among the most important in rendering. Images that lack appropriate shadows are immediately recognized as artificial. Shadow rays are also used

in other parts of rendering, such as during hemisphere-sampling in an ambient occlusion shader [Lan02] and during the path joining phase of bi-directional path tracing [LW93]. When rendering an image for a CG movie, it is not unusual to use as many as 100 shadow rays per pixel to reach convergence in shadow quality. In this case there are 100 times as many shadow rays as primary rays, so any run-time improvement in shadow ray evaluation will have around a 100 fold greater impact on performance than a corresponding improvement in primary rays.

Our goal is to accelerate the evaluation of shadow ray queries in a general manner. We do so by improving shadow ray performance when the polygon mesh casting the shadow satisfies certain properties, specifically when it is *well behaved* (Section 3.1). Our results, which will be presented in the next chapter, show that we can use this property to improve shadow render time by as much as 51% while producing final images that are identical to those produced by the unaccelerated baseline renderer. It is conceivable that total rendering time could be reduced by nearly this amount in cases where shadow ray evaluation comprises most of the work per frame.

One of the consequences of our work is that shadows that are *simple* in appearance become cheap to generate. We will use the term simple to refer to a shadow with a non-intricate silhouette that encloses a large region of contiguous screen space. We believe that our work represents an important step forward in shadow algorithms by bringing the cost of shadows more in line with their complexity. In other words, our approach makes the generation of simple shadows cheaper while leaving the expense of generating a complex shadow unmodified. By bringing the computation invested more in line with the complexity of the resulting output, we believe we are eliminating a source of algorithmic inefficiency in the shadow generation phase.

Finally, one of the advantages of our algorithm is that it produces results

that are identical to those produced by the unaccelerated baseline. As such, it can be used seamlessly in a brute-force distribution ray tracer used to generate reference images. Our adherence to an exact solution allows us to ensure we are meeting our high quality, high performance goals. In other words, we hope that by restricting our work to an exact approach, we can discover some fundamental algorithmic improvements to shadow ray tracing while preserving the quality of the final image.

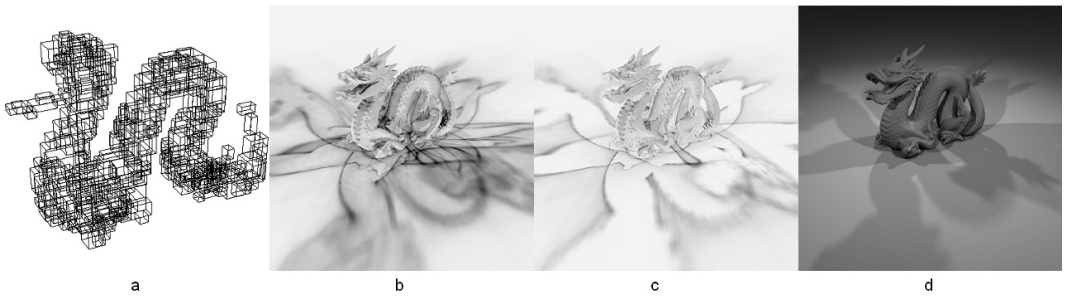


Figure 2.2: (a) Volumetric occluders created from the Dragon model. (b) Operation count per pixel without our approach. Darker pixels require more operations. (c) Operation count per pixel with our approach. Volumetric occluders noticeably accelerate completely shadowed regions. (d) Final image of soft shadows generated by Monte Carlo sampling of five area light sources.

2.5 Approach

An overview of our approach, known as *volumetric occluders*, is given in Figure 2.2. Suppose we start with a polygonal mesh that is casting a shadow. Normally, we would evaluate shadow ray queries by intersecting rays with the polygons from the mesh. We propose accelerating this process by intersecting shadow rays with polygons as well as volumetric occluders, which are axis-aligned rectangular boxes that lie completely within the mesh. The volumetric occluders for the Dragon model are shown in Figure 2.2.a.

The algorithm involves a creation phase and a usage phase. The creation phase includes building a kd-tree over the mesh and marking which kd-nodes are

internal to the polygonal mesh, which must be well behaved (Section 3.1). The usage phase involves modifying the order in which nodes are visited as a ray traverses the kd-tree so that volumetric occluders appear in the traversal order. Once such a modification is in place, volumetric occluders can be used to perform early termination on the shadow rays. Further performance improvement is provided by using a software-managed cache to reuse volumetric occluders from one shadow ray to the next.

Our technique accelerates generic shadow ray queries which can be used to generate a variety of shadows, such as the soft shadows seen in Figure 2.2.d. Compare Figure 2.2.b with Figure 2.2.c for a visualization of the pixels in which our approach reduces computational expense.

2.6 Contributions

We propose a new way to accelerate the generation of shadowing effects in rendered scenes, which we will refer to broadly as *shadow ray acceleration*. We describe two novel ways of modifying kd-tree traversal so that volumetric occluders are encountered during traversal. We present an empirical study of single object scenes that validates the soundness of our approach. We also study scenes containing multiple objects from a professional quality CG movie production and find that slight speedup is sometimes possible.

2.7 Acceleration Structures

An acceleration structure is used in all practical ray tracing to reduce the computational complexity of evaluating a ray's intersection with the objects in the scene. The reduction is typically from an O(N) brute force search through the scene geometry (consisting of N primitives) to a simpler O(log N) search.

Unaccelerated ray tracing requires that a ray be tested for intersection with all triangles in the scene whenever a first hit visibility query needs to be evaluated. An any hit visibility query is slightly simpler. Should an intersection be found at any point during testing, the answer can be returned immediately. In the worst case however, it is still necessary to check all triangles in the scene.

Unaccelerated intersection testing is conceptually simple but extremely inefficient. Conventional ray tracing instead relies on building an *acceleration structure* over the scene geometry to speed up this testing for intersections. Acceleration structures come in two general forms, *bounding volume hierarchies* and *space partitioning structures*. We will briefly discuss object partitioning acceleration structures. We will then discuss space partitioning acceleration structures at length because they are used in our volumetric occluders approach.

2.7.1 Bounding Volume Hierarchies

A *bounding volume* is a simple geometric shape that encloses the spatial extent of a *geometric primitive*. Geometric primitives are the building blocks that comprise the actual objects present in the scene, while bounding volumes are ultimately invisible to the rays. A bounding volume can also be used to enclose a geometric aggregate, such as a polygon mesh which is composed of many geometric primitives. The only requirement of a bounding volume is that the geometric primitive, or geometric aggregate, lies completely within the bounding volume.

Because a bounding volumes is just a simple geometric shape, popular choices include rectangular boxes and spheres. When the bounding volume is a box, we refer to the bounds as a *bounding box*, and in this work all bounding boxes are assumed to be *axis-aligned*, which means each face of the box is parallel to one of the three coordinate planes.

If we allow bounding volumes to enclose other bounding volumes, we can cre-

ate a hierarchy, which is naturally called a *bounding volume hierarchy* (BVH) [Cla76, RW80]. The BVH is a tree where the leaves, or *terminal nodes*, consist of bounding volumes that enclose geometric primitives. The remaining nodes in the tree are called *internal nodes*, which are bounding volumes that enclose other bounding volumes. Due to the invariant that each node in the BVH enclose the combined extent of its children, the BVH has a single node at its root that is the bounding volume for all objects in the scene.

When determining which geometric primitive a ray intersects, a *ray-scene intersector* can use a BVH to quickly cull large parts of the scene from consideration. If a ray does not overlap a volume in the hierarchy, then it cannot overlap any of the children volumes so they no longer need to be considered. The hierarchical nature of the tree means that if the ray does not intersect a volume high up in the tree, an exponentially large number of geometric primitives can be removed from consideration. On the other hand, if a ray does overlap a node in the hierarchy, all of the node's children need to be recursively examined for possible intersection.

A BVH accelerates the ray tracing process by ensuring that the ray-scene intersector only tests rays against volumes, and eventually geometric primitives, that lie close to the path of the ray. Volumes that lie sufficiently far away from the ray will be removed from consideration in an efficient, hierarchical fashion.

2.7.2 Space Partitioning Structures

A space partitioning structure divides the entire bounds of the scene into disjoint regions of space, called *nodes*, where each point in the scene is represented in exactly one terminal node (there may also exist interior nodes for the purposes of forming a hierarchy). More formally, the geometric union of all terminal nodes in the structure is identical to the original scene bounds, while the intersection of any two distinct nodes is the empty set.

A ray-scene intersector uses a space partitioning structure to leverage the fact that a ray only occupies a certain region of space. If the nodes that the ray overlaps can be identified, only the geometry from those nodes needs to be tested for intersection. All geometry in the remaining nodes can be ignored.

During construction of the acceleration structure, the scene's geometric primitives are placed within the terminal nodes with which they overlap. The space partitioning structure is now ready to accelerate ray tracing. The ray-primitive intersector first creates a list of terminal nodes that the ray overlaps. If an intersection occurs, it must occur within one of these terminal nodes, which means the intersector only needs to test the ray against the geometry that lies within these nodes. The upshot is that each ray now tests against only nearby geometry, rather than all geometry.

Space partitioning structures come in many flavors, with differences primarily in how the shape of terminal nodes is determined. A *uniform grid* [FTI86] places a *splitting plane* at regular intervals along all three coordinate axes, creating a uniform subdivision of space with no consideration for the density of geometry. One grid cell can contain a disproportionate number of primitives. An *octree* [Gla84] is more sophisticated, taking an existing node and recursively subdividing it into eight equally sized child nodes, also called octants. An octree has an advantage over a grid in that it is adaptive to the density of geometry, although its choice of placing the splitting plane is still constrained. A *kd-tree* [Ben75] is even more flexible because the location of the splitting plane, which is still axis-aligned, is computed based on the location of the geometry. We make heavy use of kd-trees in our work and will discuss them in more depth shortly. Finally, a *BSP tree* [FKN80] removes the axis-aligned restriction, allowing splitting planes to be placed at any orientation within the current node. BSP trees are the most general of the four spatial partitioning structures.

Building Kd-Trees

Kd-trees are built top-down. All geometric primitives start in an encompassing root node, and then a single axis-aligned splitting plane is selected in that node, creating two child nodes. The primitives are then sorted into the child nodes with which they overlap. If a primitive crosses the splitting plane then it is added to both children. Because a two-way split is used in the creation of children nodes, kd-trees are binary trees.

The different variants of kd-trees come about based on how the splitting plane is selected. The object median and spatial median [MB90] are simple versions of splitting plane selection. A more sophisticated approach, which is considered state of the art, is to use the surface area heuristic (SAH) [GS87, Hav00, WH06] to select the splitting plane. The SAH is based on computational geometry and provides two useful pieces of information: it indicates to the kd-tree builder where to place the splitting plane that will minimize the expected cost of using the tree, and it indicates when it is no longer beneficial to continue building the tree.

Figure 2.3 shows an example of a two dimensional kd-tree built over the geometry of a simple oval mesh. In three dimensions, each terminal node (i.e. *leaf node*) corresponds to an axis-aligned bounding box.

In our volumetric occluder work we assume the kd-tree is already built. We follow the basic approach of building kd-trees outlined in Wald and Havran [WH06].

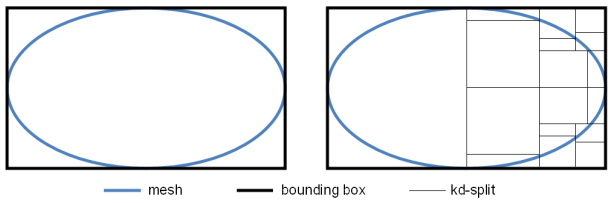


Figure 2.3: A kd-tree is built from an initial polygonal mesh in this 2D example. Kd-splits determine the extent of the kd-nodes. For clarity, only the right half of the kd-tree is shown.

Traversing Kd-Trees

Traversal is the process of accessing the contents of a kd-tree for determining ray-scene intersection. Traversal entails finding a list of all kd-nodes that overlap a ray, in the order that they lie along the ray. This sorted order is useful when evaluating primary rays (first hit visibility queries) because the intersector can terminate the intersection process early, on the granularity of a kd-node.

The fast, modern approach to traversing kd-trees is outlined in Arvo and Kirk [AK89]. As we previously discussed, each kd-tree is binary, so there are two children nodes for every interior node. The *closer node* is the child that precedes the other along the direction of the ray. The other child is the *further node*.

The ray is first intersected with the bounds of the entire kd-tree to find a t-min and t-max, which forms a *ray interval*. From here, the ray interval is incrementally shrunk to fit within the bounds of the children. A value known as t-split is computed by intersecting the ray with the splitting plane in the current node. If the t-split lies within the ray interval, the ray overlaps both children. The ray interval is broken along t-split into two child intervals and traversal recursively proceeds down both children, with care being taken to visit the closer node before the further node. If t-split lies outside of the ray interval and is greater than the interval's upper bound, the ray only overlaps the closer node so the closer node is visited using the existing ray interval. If instead t-split is less than the interval's lower bound, the ray only overlaps the further node so only the further node is visited, again using the existing ray interval.

Note that this algorithm ensures that we never visit nodes that the ray does not pass through. We only visit kd-nodes that overlap some portion of the ray, and the handling of closer and further nodes means that the nodes we visit are encountered in the order in which they occur along the ray.

2.8 Related Work

Our overall goal is to improve shadow ray performance while maintaining the high quality threshold of convergent Monte Carlo ray tracing. There is a rich literature on accelerating the generation of shadowing effects, although most of these approaches necessitate making a compromise in image quality in exchange for better performance. These approaches are ill suited for use in offline rendering, where visual artifacts in shadows (e.g. flickering, darkening, or unnatural spatial or temporal variation) would be considered unacceptable. Our algorithm lies in the remaining much smaller space of exact shadow acceleration algorithms.

2.8.1 Augmented Acceleration Structures

The possibility of improving performance by adding additional information to a ray tracing acceleration structure has been suggested by many researchers. We call such a structure an *augmented acceleration structure* because the structure has been augmented with information that is secondary to its primary purpose of finding the exact object that a ray hits.

In most cases, the extra information comes in the form of computing a stand-in, or proxy, for the geometry within an acceleration node. The idea is that the stand-in is in some way cheaper to utilize than the original geometry, although using the stand-in often produces a different result than the original geometry. There is a smaller body of work that proposes the use of augmented acceleration structures to achieve exact results. Our work falls into the second category.

2.8.2 Inexact Shadow Algorithms

Yoon et al. [YLM06] present a level-of-detail system that fits a plane to the geometry in each node of their acceleration structure using principal component analysis. These stand-ins can be used for both primary and secondary visibility. Wald et

al. [WDS04] construct a system for rendering massive models. When the data for a particular acceleration structure node is not readily available due to the memory hierarchy, the stand-in for the current node is used in place of the actual geometry. The stand-in is equivalent to a shaded cube and can be used for either primary or secondary visibility. Lacewell et al. [LBBS08] precompute a cube map within each acceleration structure node to represent the overall transmissivity of the aggregate geometry within that node. The authors generate soft shadows using these cube maps. Djeu and Volchenok [DV08] generate approximate hard shadows by computing the average geometric normal within a node to selectively enable the node as an occluder.

We do not compare to these approaches in our results because they do not satisfy our goal of preserving high quality. While many of these approaches do offer compelling results most of the time, their approximate nature means that the error is potentially unbounded given a pathological situation, which is the main reason why they have not seen widespread adoption in professional offline rendering. These approximate approaches are susceptible to light leaks [YLM06], darkening [LBBS08], and jagged silhouettes [WDS04, DV08], none of which are acceptable in offline rendering. There are also a variety of inexact shadow algorithms that fall outside the scope of this work, particularly those used in rasterization. Johnson provides a survey of these approaches [Joh08].

2.8.3 Exact Shadow Algorithms

Using augmented acceleration structures for exact rather than approximate results has also been proposed. Reshetov et al. [RSH05] mention doing so, although the authors do not provide further details. The initial phase of Schaufler et al. [SDDS00]'s construction of volumetric information is nearly identical to our work. We discuss these similarities in Section 3.3. In contrast to our work, Schaufler et al. use their

volumetric information to precompute potentially visible sets for the known light positions, requiring an update phase if light positions move or are introduced outside of a predefined region of the scene. Our technique, on the other hand, generates volumetric information that can be used for any lighting configuration, requiring rebuild only when the mesh, rather than the lighting, changes. We also share a number of similarities with Woo and Amanatides [WA90], although their work performs a more expensive precomputation step that uses per-object shadow volumes.

Our work falls into this relatively sparsely populated subfield of shadow algorithms. Our augmented acceleration structure (Section 2.8.1) improves on the work of Schaufler et al. [SDDS00] because it does not have to be rebuilt when the lighting configuration changes. This same advantage holds in comparison to Woo and Amanatides [WA90], whose shadow volumes are created based on the lighting configuration.

2.8.4 Other Approaches to Shadow Acceleration

Shadow acceleration that does not use an augmented acceleration structure have also been explored. Spherical proxies are particularly popular [SSK04, RWS+06]. Other researchers have used two-dimensional proxies, such as billboards and depth maps [PMDS06]. To our knowledge these approaches may produce inexact results with potentially unbounded error, so they are not studied in comparison to our work.

2.8.5 Modifying Traversal Order

A key component of our work is modifying kd-tree traversal order to improve performance. Haines [Hai91] proposes similar modifications to bounding volume hierarchy traversal, observing that it is a good idea to prioritize intersection tests with large objects and cheap-to-intersect objects. Our work generalizes his modifications to

high performance kd-tree traversal.

2.8.6 Caching for Ray Tracing

Caching is a very fundamental optimization that has seen many uses throughout computer science. With respect to ray tracing, Haines and Greenberg [HG86] propose caching the most recently used object to test against subsequent rays while Pearce and Jevans [PJ91] propose doing the same with the most recently used voxel in the acceleration structure. We use the same observation that caching can exploit ray-to-ray coherence and save unnecessary work.

2.9 Visibility vs Shading

Placing this work within the context of a full rendering system is difficult because the two primary components of a rendering, visibility and shading, disproportionately account for the time taken to render an image. The common wisdom is that shading consumes the bulk of the render time [CFLB06]. In this work we choose just to focus on the visibility portion of the renderer's workload, and our improvements are restricted to improving only this portion.

While this focus on visibility might be viewed as a limitation to the scope of our work, it should be noted that evaluating a multitude of shadow rays to determine color can easily be considered shading, rather than visibility. It remains to be seen whether our improvements affect the overall rendering time in a full featured production renderer.

Chapter 3

Volumetric Occluders

We propose a new approach that uses volumetric occluders to accelerate shadow rays. This approach accelerates ray-scene intersection by using volumetric occluders when possible, and falls back to using the original polygon mesh when necessary. Our results are ultimately identical to the reference image produced when our approach is not used.

This chapter provides detail on how volumetric occluders can be used to accelerate rendering. Table 3.1 shows that volumetric occluders only affect the performance of shadow rays. Primary ray performance is unchanged.

3.1 Preconditions

To use volumetric occluders, we start with a polygonal mesh. The polygonal faces can have any number of edges, but we will simplify our nomenclature, and without

query	visibility	expressed as	effect of vol occ
first occluder?	first hit	primary ray	unchanged
any occluders?	any hit	shadow ray	improved

Table 3.1: Overview of the effect of volumetric occluders.

loss of generality, by referring to all polygons as triangles.

The following conditions need to be met.

- The mesh is closed, and all edges belong to exactly two faces.
- The triangle normals have consistent orientation.
- All triangle normals point outwards.
- No two triangles intersect.

We will use the term *well behaved* to refer to meshes that satisfy all of the preceding properties. If a mesh is well behaved, it will work correctly with volumetric occluders such that the resulting image matches the image that would have been generated when volumetric occluders are not in use.

3.2 Procedurally Checking the Preconditions

The necessary conditions for using volumetric occluder can be detected in an automated fashion. We do not implement them in this work because we check that meshes are compatible with our technique by human inspection, but we present the procedural checks due to their usefulness in a more general setting.

- Check each edge of the mesh. Each edge must belong to exactly two faces. This ensures that the mesh is closed, i.e. it does not have a boundary.
- If an edge appears as (v0, v1) in one face, it appears as (v1, v0) in the neighboring face. This ensures normals are consistently orientated.
- Take any point outside of the bounds of the mesh and cast a ray from that point to the center of any triangle in the mesh. Determine if the first face hit by the ray, which may not be the face that was picked, is front-facing or

back-facing. If front-facing do nothing, otherwise reverse the orientation of all triangles to ensure that all triangle normals point outwards.

- Check for triangle-triangle intersection using a standard collision detection algorithm [PML97]. If a self-intersection is found, report that the mesh is not compatible with volumetric occluders.

A mesh that has been checked for compatibility can be used with volumetric occluders without introducing visual artifacts.

3.3 Creating Volumetric Occluders

Our approach to creating volumetric occluders is almost identical to the initial preprocess phase of Schaufler et al. [SDDS00], although our work differs from theirs in all later steps. We describe our preprocess phase in this section.

Our technique requires that the input model be specified as a watertight polygonal mesh. The mesh should contain no holes and no self-intersections. Also, all polygons within the mesh should have consistent orientation. In other words, the mesh should be well behaved (Section 3.1).

The first step is to build a kd-tree over the mesh. We build a kd-tree using the surface-area metric [GS87, MB90, HB02] and the empty space bonus [HKRS02]. We use the empty space bonus because it encourages the formation of kd leaf nodes that contain no geometry, which benefits our approach because it increases the number of candidate nodes which can become volumetric occluders. We implement the empty space bonus by reducing the cost of splits that produce empty space by 15%.

We now extend the notion of a boundary representation to the kd-tree. Our input mesh can be viewed as partitioning all points in space into three categories: inside, outside, and boundary. We extend this categorization into the discretized space of a kd-tree. We will refer to a node as being *opaque* if it is completely

within the boundary representation, as being *clear* if it lies completely outside of the boundary representation, and as being *boundary* if it contains any polygons from the mesh.

Of the three types of nodes, the opaque nodes are the ones most relevant to this work. These nodes are the volumetric occluders and can be treated as intersectable objects when evaluating shadow ray queries. For a given shadow ray, consider all leaf nodes that overlap the extent of the ray. If any of these nodes are opaque, and if the ray starts or ends on the exterior of the mesh, then we know by the watertightness property that the ray must intersect some polygon from the mesh. In other words, an intersection with an opaque node implies an intersection with the mesh geometry. We can use the opaque nodes in this manner to terminate shadow rays.

During a preprocess phase, we classify all nodes within the kd-tree. Each node is labeled as either opaque, clear, or boundary, which are the three types of *volumetric state*. To label each node, we iterate through all leaf nodes in the kd-tree. If the leaf node contains any mesh geometry, it is marked as a boundary node. For all remaining leaf nodes, we cast a test ray to determine whether the volumetric state is opaque or clear. The only requirement for the test ray is that its origin lies within the node bounds; its direction does not matter. In our implementation the test ray originates from the center of the node and the direction is chosen randomly. We send the test ray through the kd-tree using our primary ray intersector in order to determine the closest intersection. If the test ray hits a back-facing polygon, the current node is marked as an opaque node. If it hits a front-facing polygon or misses all polygons in the mesh, the node is marked as a clear node. If the test ray hits a polygon edge-on, we cast a new test ray. Figure 3.1 shows the result of volumetric occluder creation for a simple input mesh.

All nodes marked with a volumetric state of opaque can act as volumetric

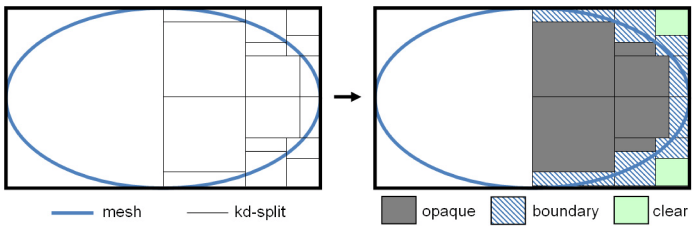


Figure 3.1: Initial kd-tree and the volumetric occluders after creation. For clarity only the right half of the kd-tree is shown.

occluders. These opaque nodes are desirable for intersection for two reasons: a) they are cheap to intersect against and b) they are often be larger, and therefore better occluders, than the polygons in the mesh. We will discuss the second benefit in more detail in Section 3.4.3.

To store the volumetric state within a kd-node, we require only two flag bits to represent all three volumetric states. We store the two bits in the node's child pointer, which is not used at the leaves. Our two bit representation fits within our 64 bit kd-node representation, thereby preserving efficient and memory-friendly kd-tree traversal.

3.4 Modifying Traversal Order

3.4.1 Kd-Tree Traversal and Standard Ray Order

In order to use volumetric occluders to terminate shadow rays, we must change the order that rays visit nodes when they traverse the kd-tree. Before we propose such a change, we first review the node order produced by standard kd-tree traversal.

Recall that fast kd-tree traversal is performed by incrementally shrinking the ray interval to compute the region of the ray that overlaps the current node (Section 2.7.2). At each interior (non-leaf) node that is visited, the traversal algorithm intersects the ray with the node's split plane. If the ray crosses the split plane, two new ray intervals are created, one for each side of the split plane. Otherwise, the

current ray interval is preserved. The result of this intersection test also determines whether the ray should visit the node's near child, far child, or both children. In the case where the ray must visit both children, traversal visits the near child and then the far child. For each child that is visited, the corresponding ray interval is passed along.

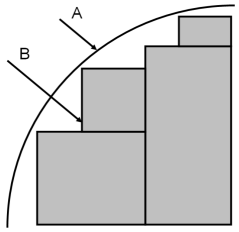


Figure 3.2: Volumetric occluders cannot be accessed under standard ray order due to an outer layer of mesh geometry (ray A). Our proposed modifications to traversal order allows rays to access volumetric occluders which lie beyond the mesh geometry (ray B).

Kd-nodes are visited during this traversal in the order in which they occur along the ray, where nodes closer to the ray origin appear before further nodes. We refer to this traversal order as *standard ray order*, which we will often shorten to *ray order*. Ray order is very useful for rays which require the closest hit to their origin, such as primary rays. However, this order is not necessary for shadow rays, which only require that some intersection be found between the ray origin and endpoint. In fact, ray order is detrimental to our task of intersecting shadow rays with volumetric occluders because it guarantees that shadow rays will not encounter any volumetric occluders during traversal (Figure 3.2). Instead, shadow rays will always encounter a terminal boundary node before an volumetric occluders, rendering volumetric occluders useless (Figure 3.3, ray order).

We propose two modifications to traversal order to correct this problem, in effect producing two novel ways of traversing kd-trees which allow for early termination using volumetric occluders.

The first variant extends ray order to include the nodes that would have

appeared in ray order had the ray not been terminated at its terminal boundary node. This extension allows for volumetric occluders that lie immediately beyond the terminal boundary node to show up during traversal.

The second variant is based on breadth-first search that prioritizes visiting shallow nodes along the ray before deeper nodes. This policy tends to quickly find the largest volumetric occluder that lies along the ray, should it exist. In our test scenes, this variant finds volumetric occluders that are up to 73x larger by surface area than the first variant.

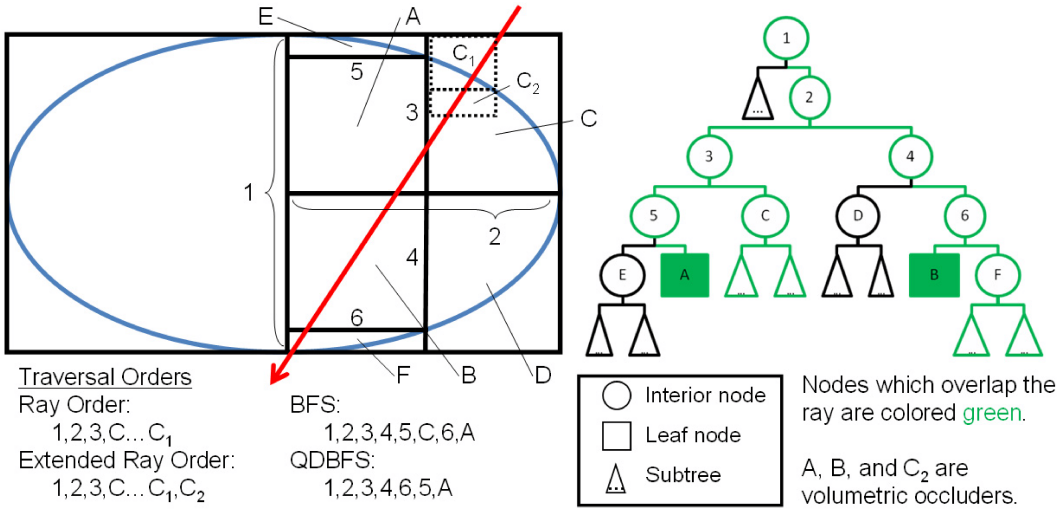


Figure 3.3: Different traversal orders for the given ray. The digits 1 through 6 represent a kd-tree split (left) in addition to the node that contains the split (right). Letters are used to represent a region of space (left) and the corresponding node (right). Not shown on the right, C_1 and C_2 are descendants of C.

3.4.2 Extended Ray Order

Our first modification to traversal order is to extend ray order so that volumetric occluders that lie just past the terminal boundary node can be encountered during traversal. We accomplish this by deferring the intersection of the ray with the terminal boundary node and speculatively taking extra traversal steps. This is

a promising idea because the cost of a traversal step is significantly lower than geometry intersection and the final intersection with a volumetric occluder, should it exist along the ray, is similarly inexpensive. In this way, we are speculatively taking extra traversal steps in the hope of eliminating intersection tests against mesh geometry.

The new traversal order differs from standard ray order. A ray now visits the same nodes as it would in standard ray order, except it continues on to nodes that are further along the ray than the boundary node that would normally terminate the ray. Standard ray order is a prefix of this new traversal order, as such we call the new order *extended ray order*. The extra nodes that are part of extended ray order have the potential to be volumetric occluders (Figure 3.3, extended ray order).

To implement extended ray order, we use a FIFO (first in, first out) *deferment list* which stores the ray-node intersection tests have been postponed. When a ray enters a kd-leaf node, the volumetric state is first checked. If the volumetric state is opaque, the ray is terminated as a hit. If the state is clear, no hit occurs. If the state is boundary, we add the current node to the deferment list if there is room. Should the list be full when we try to add, we intersect the ray with the nodes on the deferment list, returning a hit if an intersection is found. If no intersection is found, the the list is flushed and the node that we previously did not have room for is added to the list.

Notice that testing for intersection with a volumetric occluder is an extremely lightweight operation that only requires the checking of the volumetric state of the node, which has already been fetched into the system memory cache along with the node. These intersections are considerably cheaper than ray-triangle intersections and provide a strong motivation for using volumetric occluders.

Another thing to note is that two separate pieces of data are saved when an intersection is deferred. Recall that during fast kd-tree traversal [AK89], each visit

by a ray to a node is accompanied by both a node identifier and the ray interval representing the region of overlap between the ray and the node. In order to defer the intersection, both the node identifier and ray interval must be saved. We save both values in our implementation to support the full generality of deferring any type of ray, although the ray interval that we store is not strictly necessary for shadow rays. For these rays, the interval associated with the ray will work just as well as the local interval.

One final thing to remember when implementing deferred intersection is that intersections may be pending even after tree traversal completes. We need to check and flush the pending intersections on the deferment list at this time, but only if the shadow ray has not yet found a hit.

In this work we use a deferment list of size 512, meaning we postpone intersection with up to 512 boundary nodes before the deferment list must be processed. A list size of 512 guarantees that all rays in our test scenes postpone polygon intersection until after they have traversed completely through the kd-tree. Interestingly, we found that this policy provides the best performance in our test scenes. We theorize that this policy is most beneficial because it maximizes the number of rays that undergo early termination using volumetric occluders.

3.4.3 Breadth-First Search Order

If the input mesh has a large interior region, there is a tendency for our algorithm to create large volumetric occluders within that region. These large volumetric occluders provide many opportunities for performance improvement. First, we will describe these opportunities and then describe how to give preference to finding large volumetric occluders during traversal.

Two nice properties of large volumetric occluders are increased likelihood of occluding future shadow rays and favorable positioning in the kd-tree. The largeness

of large volumetric occluders means that future shadow rays are more likely to hit the volumetric occluder if the occluder were somehow cached for reuse. Additionally, large volumetric occluders are often the shallowest nodes in the tree. If a large volumetric occluder is used to terminate a ray under a traversal order that visits shallow nodes before deeper nodes, it is possible that fewer nodes need to be fetched and traversed, with obvious performance benefits.

We will now describe how to modify traversal order so that large volumetric occluders can indeed be found during traversal. We simply use breadth-first search (BFS). BFS guarantees that shallow nodes are visited before deep nodes. This modification biases the traversal to visit large kd-nodes first, which in turn increases the chances of encountering a large volumetric occluder. A traversal that has such a bias can theoretically take fewer overall traversal steps than even standard ray order, should it encounter a large volumetric occluder in the upper levels of the tree. Under the traversal order induced by BFS, which we call *BFS order*, it is possible to decrease both the number of traversal steps and number of ray-triangle intersections. In contrast, extended ray order always increases the number of traversal steps.

To implement breadth-first search, we require a FIFO queue to store the list of nodes that need to be visited. Every time a ray visits an interior node in the kd-tree, it clips its interval to the two children to determine whether it needs to traverse one or both of them. Any child that needs to be visited is added to the end of the queue, along with the overlapping ray interval, in the exact same manner as kd-nodes and intervals were added to the deferment list. The front element is then removed from the queue and processed as the next node. This process iterates until the queue is empty (Figure 3.3, BFS order).

While at first glance adding a FIFO queue may seem like extra overhead for traversal, recall that standard ray order requires using either the system call stack or a self-managed stack to keep track of pending nodes during depth-first search

(DFS) through the kd-tree. When switching between DFS and BFS, we no longer need a stack, replacing it instead with a queue [RN02].

3.4.4 Quick Descent Breadth-First Search Order

One limitation with breadth-first search is the extra work performed when only one of two children needs to be visited, in which case the relevant child is added to the queue. In contrast, when only one child needs to be visited in depth-first search, the ray immediately descends into the child with no need to add the child to the stack.

We can remove this inefficiency in BFS by slightly modifying the breadth-first search algorithm. Any time there is only one child to traverse, we descend immediately into the child, exactly as depth-first search does. If instead there are two children to traverse, we fall back to breadth-first search and add both children to the end of the queue. We call this variant quick descent breadth-first search (QDBFS) to indicate the preference of the search. QDBFS has two nice features derived from depth-first search. Extra bookkeeping is only needed when the near and far children both need to be visited. Also, each quick descent removes the need to enqueue a node, reducing memory traffic and queue size.

We notice a decrease in runtime by 15.0% to 20.0% in our test scenes when we switch from pure BFS to QDBFS. The drawback of QDBFS is that the traversal order, which we call *QDBFS order*, is now more complicated than either ray order or standard BFS order, and it is much harder to predict how it will behave. However, the order also has the possibility of being shorter than any other traversal order while still discovering large volumetric occluders (Figure 3.3, QDBFS order).

We use QDBFS rather than breadth-first search for the remainder of this paper due to its more favorable characteristics. In our test scenes, the volumetric occluders found under QDBFS were up to 73x larger by surface area than volumetric occluders found under extended ray order. This results indicate that although

QDBFS has taken on some of the characteristics of DFS, it still exhibits the BFS property that we care about, namely that shallow nodes are visited before deep nodes.

3.5 Volumetric Occluder Cache

We store and reuse volumetric occluders using a software-managed *volumetric occluders cache*, which will be abbreviated VC. We use the VC as a systems level optimization to further improve performance. Before rays are sent through the kd-tree, they are tested against the members of the VC. If a hit is found, the ray can be terminated immediately, which eliminates all kd-tree traversal steps and intersection tests with mesh geometry. If a hit is not found, the ray is traversed through the kd-tree like before. In this case, using the VC requires extra ray-bounding box intersection tests that do not help terminate the ray. In our experience, however, using the VC always improves system performance.

Caching is not a new idea to ray tracing. We would like to mention, however, that the volumetric occluders that we cache are much simpler to intersect against than other items that tend to be cached. Intersecting a ray with a volumetric occluder (i.e. an axis-aligned bounding box) can be performed much faster than intersecting a ray with an entire object [HG86], the contents of an acceleration structure node [PJ91], or even a triangle.

The VC is populated by adding a volumetric occluder every time an occluder is encountered during shadow ray traversal. If the occluder is already present in the cache, no change occurs. Otherwise, the occluder is added to the cache using an LRU (least recently used) replacement policy.

The VC is accessed by all shadow rays that intersect the overall bounds of the kd-tree. These rays are tested for intersection against each member of the cache until either a hit is found or the end of the cache is reached. If no hit is found, the

ray traverses through the kd-tree as before.

One difficulty with using the VC is that intersecting a ray with a kd-node within the VC requires that the bounding box be known for that kd-node. A ray testing against the VC is not in the middle of kd-tree traversal so it must perform a full ray-bounding box test to determine if it indeed hits the node. However, at the time when items are being added to the VC, that is, during kd-tree traversal time, these bounding boxes are not readily available.

We solve this problem by computing a node-to-bounds lookup table. The bounding box for each kd-node is computed and stored in this lookup table during our preprocess phase. When a volumetric occluder is hit during traversal, its bounds are looked up from the table and added to the VC, along with the node ID.

One useful way to think of a software-managed volumetric occluder cache is that it speculatively performs bounding box intersection in order to find an occluder that terminates the ray. The aggressiveness of the speculation is proportional to the size of the cache. We reserve space in the VC for 4 bounding boxes per light, which works well for our test scenes.

3.6 Triangle Intersection and Volumetric Occluders

We assume that the triangle intersector is leak-free for this work, meaning that it will not permit rays to *leak* through regions of the polygon mesh which are solid. In practice there are numerical issues with all floating point based ray-triangle intersectors due to the imprecise nature of the numerical representation. In other words, there will be leaks which will affect the claim that our approach produces images that are exact.

A leaky ray-triangle intersector can interact with our approach during the classification phase of the kd-tree nodes. The intersection results of these rays may lead to two types of incorrect classification of kd-tree nodes, each of which has its

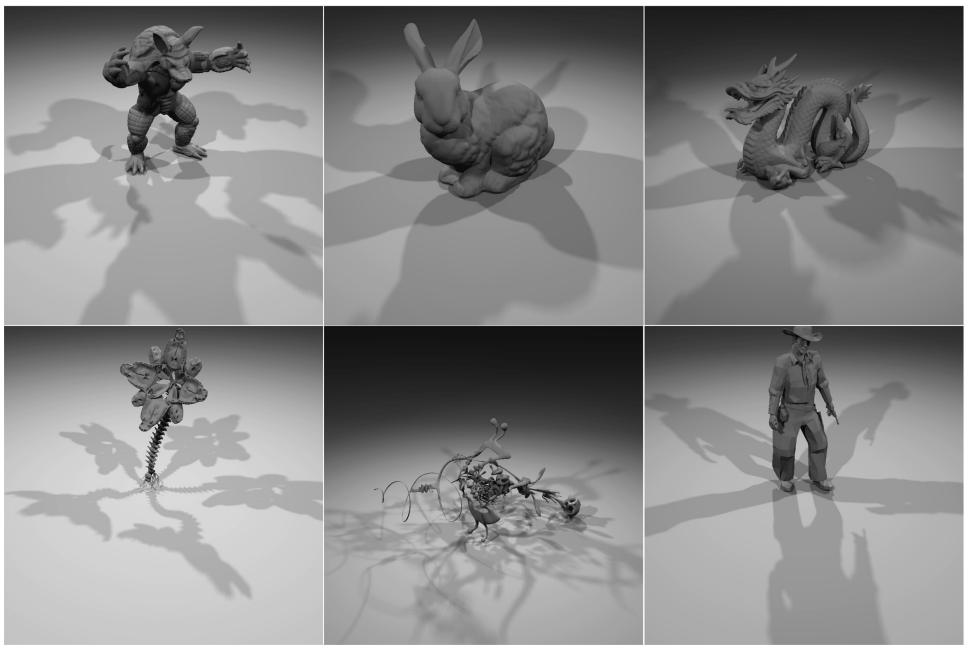


Figure 3.4: Scene setup. Top row: Armadillo, Bunny, Dragon. Bottom row: Rose, Yeah Right, Cowboy.

own repercussions.

- Node incorrectly labeled as opaque. If a classification ray leaks through a mesh into its interior and then hits a backface on the far side of the mesh, a node outside of the mesh, which should be labeled clear, will instead be marked as opaque. When this occurs the resulting image can have noticeable visual artifacts because an exterior node is now marked as opaque, and will cast shadows. The visual result is that the scene may contain extraneous shadows.

- Node incorrectly labeled as clear. If a classification ray leaks through a mesh into the exterior space and hits a frontfacing polygon, the node will be mistakenly identified as clear. When this occurs there will be no visual artifacts, however performance may decrease because the interior of the mesh will now be missing a volumetric occluder that could otherwise be providing useful

occlusion.

3.6.1 Possible Visual Artifacts

Interestingly, volumetric occluders does not alter the shadow rays being traced in the slightest from the baseline approach, either in position, direction, or length. If a ray intersects a triangle in the baseline approach, this intersection will also occur in our volumetric occluder version (although it may take the form of a ray-volumetric occluder intersection). Even if a volumetric occluder fails to occlude something that it should occlude (due to, say, numerical issues), the original ray-triangle intersection will still be detected and the ray with be marked as terminated. The one numerical precision weakness here is that volumetric occluders may occlude rays that they should not, such as when a ray passes very close to, but not exactly through, the corners or edges of an occluder. This false occlusion can of course lead to visual artifacts.

3.7 Results

We have currently built a proof of concept system that demonstrates that our approach provides a savings of up to 51% in models that are appropriate for use with our technique.

The primary goals of this part of the work are to study runtime improvement when using volumetric occluders and to verify that the number of traversal steps and geometric intersection are changing as we hypothesized earlier. We compare extended ray order and QDBFS order to standard ray order, both with and without the VC. We also present results for situations in which our approach does not provide performance improvement. In Section 3.8, we offer some rules of thumb for determining whether a model is appropriate for our approach.

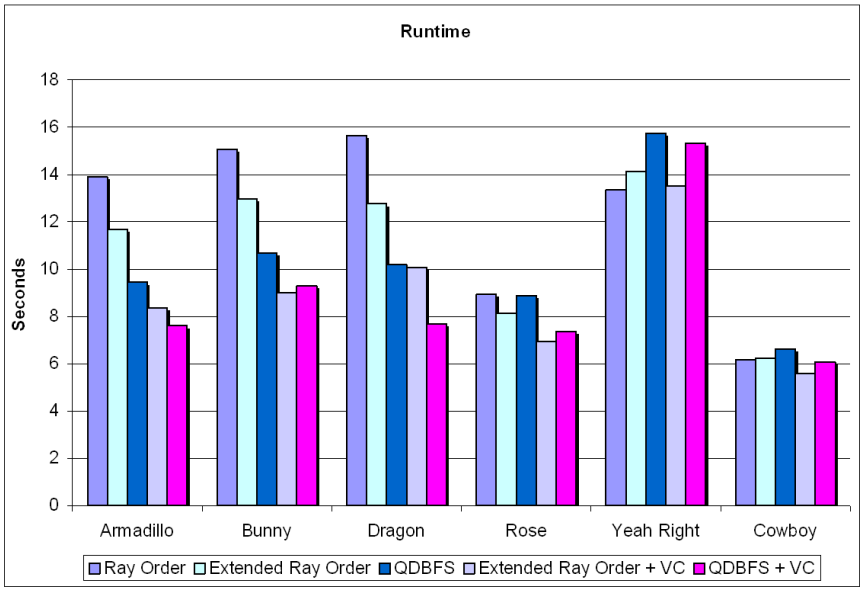


Figure 3.5: Using volumetric occluders and modified kd-tree traversal in a 4-wide packet tracer. Runtimes listed are for the time spent during shadow ray evaluation only.

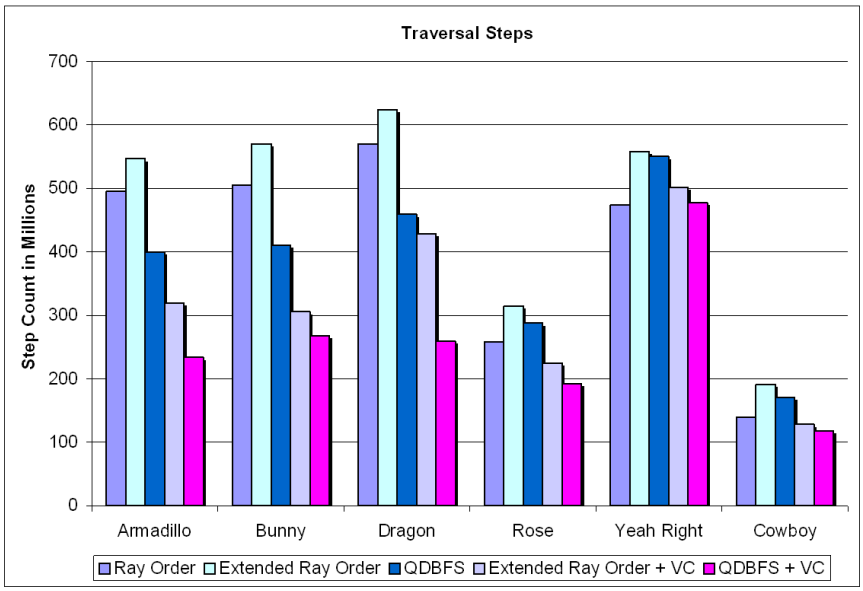


Figure 3.6: Kd-tree traversal steps for shadow rays.

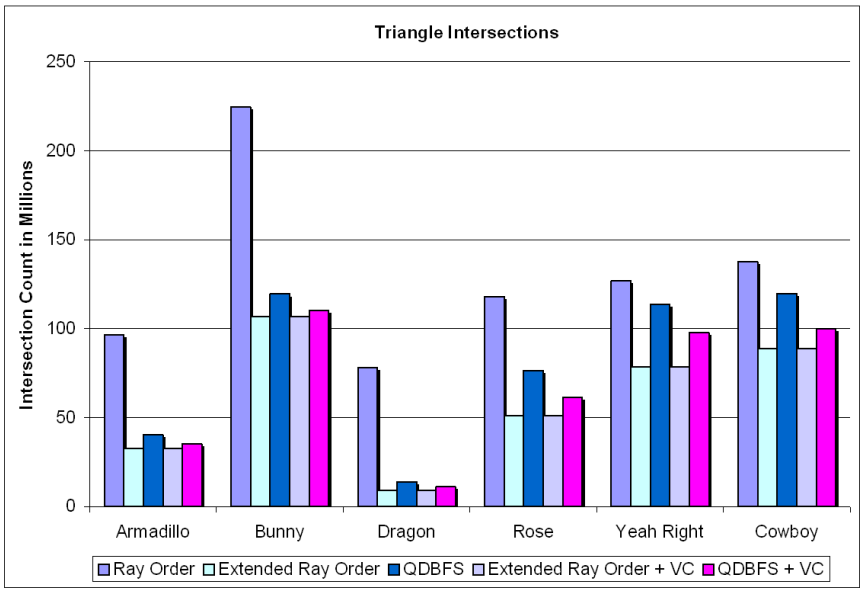


Figure 3.7: Triangle intersection tests for shadow rays.

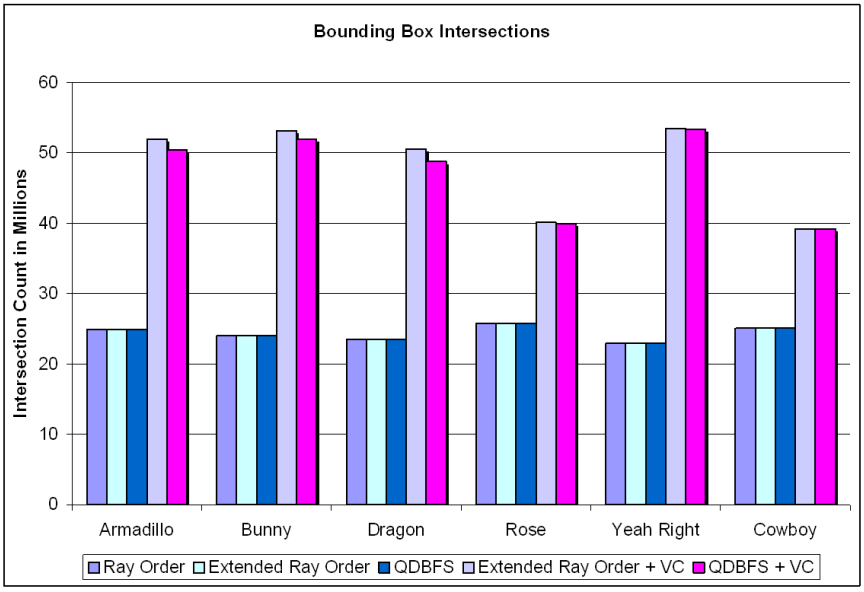


Figure 3.8: Total number of bounding box intersection tests for shadow rays.

scene	tri	v.occ	s.rays	pre (s)	base (s)	best (s)	save
Armadillo	346	80	99.7	0.26	13.9	7.6	45%
Bunny	70	35	96.0	0.09	15.1	9.0	40%
Dragon	871	231	93.7	0.73	15.7	7.7	51%
Rose	326	109	103.2	0.38	8.9	6.9	22%
Yeah Right	119	54	91.7	0.27	13.4	13.5	-1%
Cowboy	7	0.9	100.3	0.004	6.2	5.6	10%

Table 3.2: Tri is the number of triangles (thousands). V.occ is the number of volumetric occluders (thousands), S.rays is the number of shadow rays cast (millions). Pre is the time needed to create volumetric occluders and build the node-to-bounding box lookup table. Base is the shadow ray time from the unaccelerated baseline. Best is the shadow ray time from our fastest run, see Figure 3.5 for which run was fastest. Save is the percentage savings of best compared to base.

3.7.1 Ray Tracing Implementation

Although we have explained our proposed modifications within the context of a ray tracer that traces only one ray at a time, our ideas generalize to packet tracing [WBWS01], which we use in our primary results. Our packet tracer implementation uses a fixed shadow packet size of 4 and performs operations in a 4-wide manner throughout traversal, triangle intersection, and bounding box intersection. We emphasize our results for packet tracing to show that the two techniques are compatible.

3.7.2 Experimental Methodology

All results were gathered using a single core of a 2.66 GHz Intel Core 2 Q6700 processor, 1066 MHz FSB. All images are generated at 1024x1024 using 5 area lights and 20 shadow rays per light, per pixel. A shadow ray is not cast if the surface from which it originates is back-facing to the light (s.rays in Table 3.2).

We consider a variety of models (Figure 3.4), ranging from a low-polygon game model (Cowboy) to highly tessellated offline rendering models (Dragon, Rose). We choose to restrict our study to scenes consisting of a single mesh to understand which properties of the mesh affect the applicability of our approach.

3.7.3 Runtime, Traversal, and Intersect

The total runtime under various approaches is shown in Figure 3.5. The left bar for each scene is the baseline rendering using standard ray order. We also test extended ray order, QDBFS order, extended ray order with the VC, and QDBFS order with the VC.

Figure 3.5 contains the time spent evaluating shadow rays for each of our images. Our results show that significant gains can be had by utilizing volumetric occluders. When the model in question is compatible with our technique, we see a runtime improvement of up to 2.0x. Overall QDBFS with the VC performs best, providing good improvement for the Dragon model in particular. While we do well on 4 of our test scenes, we fail to improve performance for the Yeah Right and Cowboy models. We discuss why these models do not do well in Section 3.8.2.

In addition to runtime improvement, we also provide the number of traversal steps and number of triangle intersections for each run (Figures 3.6 and 3.7). As expected, we see a decrease in triangle intersections whenever we apply a modified traversal order. These savings are accompanied by an increase in traversal steps when using extended ray order. In contrast, QDBFS order reduces the number of triangle intersections *and* the number of traversal steps in almost all of the scenes where we improve performance. As expected, the VC reduces the number of traversal steps regardless of the accompanying traversal order.

The final cost associated with our approach is bounding box intersection tests for the shadow rays (Figure 3.8). Although an explicit intersection test is not needed when a ray encounters a volumetric occluder within the kd-tree (recall that this test only requires checking the state bits stored in the node), such a test is needed when testing a ray against the members of the volumetric occluder cache. We only see an increase in number of tests when the volumetric occluder cache is being used. Note that the baseline, standard ray order, uses a significant number of bounding

scene	orig leaf	orig all	best leaf	best all
Dragon	52.6%	59.6%	63.8%	67.0%
Rose	61.8%	68.9%	67.1%	68.6%
Yeah Right	62.2%	68.2%	62.5%	66.1%
Cowboy	71.9%	79.5%	71.0%	76.1%

Table 3.3: Packet utilization for three models. Orig is the baseline, best is the result from the fastest of our runs. Leaf means utilization only at kd leaf nodes. All means overall utilization across all nodes in the tree.

box intersection tests in order to determine if a shadow ray hits the kd-tree at all.

3.7.4 Packet Utilization

Table 3.3 describes packet utilization when we apply our technique. When we are using an appropriate model, such as Dragon, packet utilization increases by around 10%, both overall and at the leaf nodes. Utilization at a kd leaf node is indicative of how much useful work is accomplished by each packet-vs-triangle intersection, as well as how many rays will early exit should the leaf node be a volumetric occluder. Overall packet utilization remains constant for the Rose model, where we see only marginal gains in final runtime. For our two failure cases, overall packet utilization decreases by a few percentage points.

3.7.5 Using a Single-Ray Tracer

Up to this point, we have presented our results within the context of a packet tracer. We also applied our approach to a single-ray tracer and found that the percentage savings are almost identical to those reported in Table 3.2, differing by only about 2%. This result provides evidence that using volumetric occluders are a very general approach that can be applied to both packet tracing and single-ray tracing with similar gains.

scene	b.occl (s)	q.occl (s)	b.unoccl (s)	q.unoccl (s)
Armadillo	10.1	2.7	5.4	6.1
Bunny	9.7	2.7	6.9	7.7
Dragon	12.2	3.2	5.1	5.7
Rose	6.1	3.7	4.4	5.0
Yeah Right	7.3	7.6	8.4	9.4
Cowboy	3.9	3.2	3.8	4.3

Table 3.4: Total time spent for each type of shadow ray. B.occl means occluded rays for the baseline, q.occl means occluded rays for QDBFS+VC, b.unoccl means unoccluded rays for the baseline, and q.unoccl means unoccluded rays for QDBFS+VC.

3.7.6 Occluded vs Unoccluded Rays

Our method, an early termination technique, can only accelerate shadow rays which intersect some object in the scene. We refer to these shadow rays as *occluded rays*. All remaining shadow rays, the *unoccluded rays*, do not see any performance benefit from our approach and in fact must incur some overhead. We provide a breakdown of the time spent tracing both occluded and unoccluded rays (Table 3.4).

Notice that our technique removes up to 74% of the time spent tracing occluded rays. Previously, the performance bottleneck for all scenes except Yeah Right was the evaluation of occluded rays (Figure 3.9). Now, the bottleneck for all scenes is the tracing of unoccluded rays. This bottleneck holds regardless of whether one considers the extra runtime incurred by unoccluded rays when using our approach. Refer to the b.unoccl column in Table 3.4 for the no-overhead runtimes.

3.7.7 Shadow Ray Distribution

Figure 3.10 shows the distribution of shadow rays for the entire scene. For all scenes that perform well, the majority of occluded shadow rays happen to hit a volumetric occluder. Scenes that do not perform well are ones where a significant portion of the occluded shadow rays require falling back to triangle intersection.

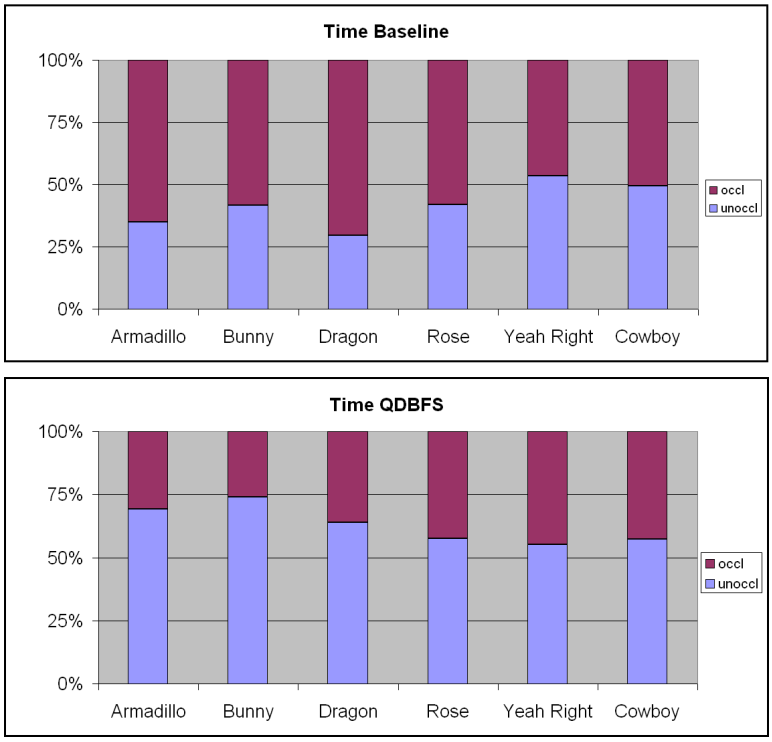


Figure 3.9: When volumetric occluders are not used (baseline), more time is spent ray tracing occluded shadow rays than unoccluded shadow rays in all scenes except Yeah Right. When volumetric occluders are used (QDBFS), the majority of shadow ray evaluation time is now spent on unoccluded shadow rays.

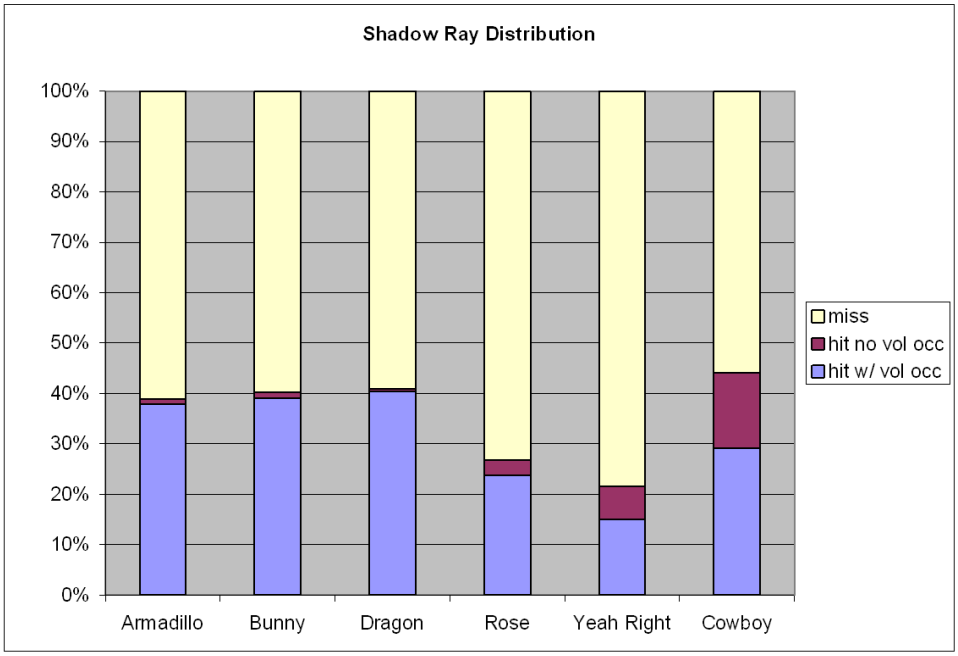


Figure 3.10: The shadow ray distribution when using QDBFS with the VC. Three shadow ray types are shown, shadow rays that do not hit anything (miss), shadow rays that hit a triangle (hit no vol occ), and shadow rays that hit a volumetric occluder (hit w/ vol occ).

3.8 Discussion

3.8.1 Limitations

Our approach only works on well behaved meshes (Section 3.1). If the mesh is not well behaved, the result of our volumetric occlusion creation step is undefined. We also require that all geometry within the mesh be opaque. If any part of the mesh is transmissive, a technique such as Lacewell et al.'s [LBBS08] would be more appropriate.

This technique is designed to accelerate shadow rays. Rendering systems that do not spend a significant portion of their time performing shadow ray evaluation will fail to benefit from this approach accordingly.

3.8.2 Rules of Thumb for Applicability

Our results indicate that this technique, while offering up to a 2.0x speedup on compatible models, is not always appropriate. We propose two rules of thumb for determining the compatibility of a model, expressed as simple tests that require inspecting the model and lighting conditions. Both tests should be considered when determining applicability.

Shadow Shape Test

Figures 2.2 and 3.11 show false color images which indicate the number of operations required to evaluate shadow rays at each pixel. The false coloring is computed as the sum of the number of traversal steps and the number of ray-triangle intersections. Pixels with higher sums receive a darker color. While not a completely accurate measure of time spent per pixel, this tally still provides insight into which regions of the image benefit from our approach. Figure 2.2 shows the Dragon model, a case in which our technique does well, while Figure 3.11 shows the Yeah Right model, a

case in which our technique fails to improve performance.

One reason why we do not do well on Yeah Right is because its shadow region contains a long, intricate silhouette. Rays cast near the silhouette are, for the most part, unaccelerated by our technique. Instead, these rays must intersect with mesh geometry in order to correctly distinguish the shadow boundary. Other models that cast shadows with intricate silhouettes, such as trees, are similarly inappropriate. In contrast, the Dragon model casts a shadow with a simple silhouette and a large interior, meaning it will generate a significant number of shadow rays that are likely to hit volumetric occluders.

We observe that the shape of the shadow is very important in determining the effectiveness of volumetric occluders. Models and lighting configurations that produce expansive shadows with simple silhouettes tend to perform well with our approach. If the shape of the shadow can be predicted before image generation, a quick analysis of the shape will provide a sense of the potential benefit.

Figure 3.11: Operations per pixel for a failure case. The unaccelerated baseline is on the left and QDBFS+VC is on the right.

Internal Volume vs Mesh Tessellation Test

Our technique works well when there is sufficient mesh tessellation with respect to a model's internal volume. We present a two-step procedure for determining if this is the case. The first step is to estimate the internal volume by imagining how well axis-aligned boxes would fit within the mesh interior. The next step is to look at the mesh tessellation rate and compare the size of the imagined boxes with the size of the polygons in the mesh. If the boxes do not have significantly larger surface area than the mesh polygons, then it is unlikely that our approach will perform well.

The Yeah Right model represents a case where not enough space exists within the interior of the model. The thin, serpentine shape of the model makes axis-aligned boxes hard to fit within the interior, meaning fewer volumetric occluders. The Cowboy model also exhibits degraded performance but for a different reason. While Cowboy contains an expansive interior, it also has a very low tessellation rate. Even though reasonable volumetric occluders are constructed, the large polygons from the mesh already provide a cheap and effective way of terminating shadow rays. This balance between polygon occlusion and volumetric occlusion is why we advocate using a ratio test.

Using this rule of thumb, we predict that most architectural models are inappropriate for our approach. Even though volumetric occluders would nicely fill out the interior of the model, the mesh tessellation rate is too low. On the other hand, well-performing models from our study, such as Dragon, Bunny, and Armadillo, have good internal volumes and a high surface tessellation rate. We feel that our approach will become increasingly useful in the future as model complexity continues to increase.

3.8.3 Symmetry of Empty Space and Filled Space

Normal kd-trees (which do not have volumetric occluders) provide acceleration through a spatial volume, an in particular they make ray traversal through empty space fast. These empty space regions are often represented as large, singular nodes within the tree which can be traversed quickly. Volumetric occluders represent a way to provide similar acceleration for rays that pass through completely filled space, that is, the interior volume of an object. The one remaining region type is the mesh boundary (boundary nodes), which we believe will remain slow because the geometry in this region must be checked to correctly resolve visibility.

3.8.4 Adaptivity to Shape

One advantage of volumetric occluders is that they can be created for all well behaved meshes regardless of shape. Although on some scenes a different type of proxy occluder within the mesh may perform better, such as enclosed spheres, a problem is that these occluders may become hard to fit within the narrower regions of the objects, such as the limbs.

Volumetric occluders, on the other hand, leverage the adaptive nature of kd-trees and can be placed within parts of the object that would otherwise be problematic for proxy occluders with fixed shape. We observe that volumetric occluders are generated even in the far reaches of our objects, such as the toes of the Armadillo (Figure 3.12).

3.9 Conclusions and Future Work

Volumetric occluders provide a new way to accelerate shadow rays. We study this technique within a ray tracer that generates Monte Carlo shadows. Compatible models exhibit up to a 2.0x increase in performance and the resulting image matches the

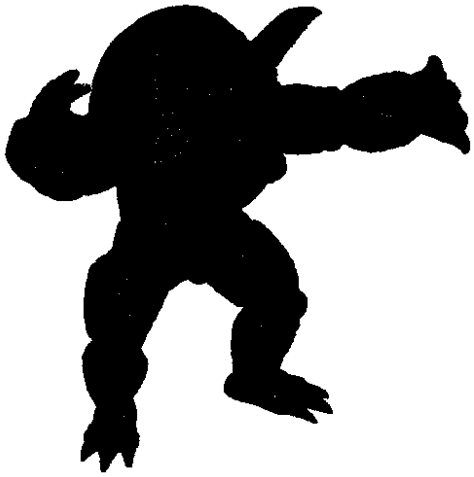


Figure 3.12: Volumetric occluders generated for the Armadillo scene.

unaccelerated baseline. However, this work only examines single-object scenes. In the next chapter, we will examine our approach in more practical scenes containing multiple objects.

Our technique generalizes to all algorithms that use shadow rays. For example, volumetric occluders could be applied to hemisphere-sampled ambient occlusion [Lan02] without any further modification. Bi-directional path tracing [LW93], which uses numerous shadow rays to determine whether light travels between paths from the viewpoint and paths from the light, also stands to benefit from our approach.

Using volumetric occluders extends to other types of space partitioning structures (Section 2.7.2) besides kd-trees. For example, marking acceleration nodes as volumetric occluders can also be performed on uniform grids and octrees. Extended ray order could then be used with either acceleration structure. Octrees, BSP trees, and other hierarchical representations could also use QDBFS. Note that volumetric occluders do not directly apply to bounding volume hierarchies (Section 2.7.1) because there are no empty nodes in a bounding volume hierarchy.

We discovered that using volumetric occluders shifts the majority of the computation effort in shadow ray evaluation from one type of shadow ray to another type. Instead of spending the majority of the time processing occluded shadow rays, the shadow rendering phase is now bottlenecked on evaluating *unoccluded* shadow rays that do not intersect any object in the scene (Figure 3.9). One avenue of future work is to accelerate the now prominent unoccluded shadow rays, whose performance may be improved by applying geometric or interval-based frusta to encapsulate many such rays into an aggregate query [RSH05, WBS07].

Finally, our work proposes the use of two new traversal orders for kd-trees. Extended ray order and QDBFS order represent two ways of efficiently finding volumetric occluders during traversal. Different traversal orders are entirely compatible with achieving this goal. More broadly, we would like to emphasize that this work provides a proof-of-concept that kd-tree traversal order *can* be modified in ways that improve overall performance. The notion that ray order is not a sacred part of kd-tree traversal may pave the way for brand new traversal orders suited for other tasks in high-quality, high-performance rendering.

Chapter 4

Case Study: Volumetric Occluders in *Big Buck Bunny*

This chapter builds on the foundation laid by the previous chapter's study by extending volumetric occluders to multi-object scenes. Multi-object scenes are representative of scenes used in practice, such as in movies generated either entirely or in part by computer graphics. We use scenes from an actual computer generated movie production called *Big Buck Bunny*, which is an open movie project sponsored by the Blender Foundation [Ble08]. As an open movie, the visual assets of *Big Buck Bunny* are freely available, allowing us access to the exact art assets that were used to create the final film. *Big Buck Bunny*, like many other movies, features widespread use of shadowing effects to convey artistic direction and spatial relations, as shown in Figure 4.1.

Our goal is to understand whether the promising results from our previous single object study extend to scenes like the ones used in practice in high quality offline rendering projects. Our use of *Big Buck Bunny* provides us with full access to the rendering workloads that were used to produce this professional, offline movie. Unlike our previous study, which analyzed volumetric occluders in ideal laboratory

conditions, this part of our work puts volumetric occluders through its paces in a far more realistic setting.

Figure 4.1: (Left) A scene from the movie *Big Buck Bunny*. The movie includes forest scenes that emphasize shadows cast by the environment. (Right) A scene from the movie as seen in Blender. The scenes are open-source and available for browsing, modification, and conversion into other formats.

4.1 Overview

A few modifications to the renderer are needed in order to support multi-object scenes. Not all objects in the scene are suited for generating volumetric occluders, so we must account for this distinction in our preprocessing phase. We also detail some minor changes to our approach to accommodate certain aspects of our new environment.

With these modifications in place, we run experiments to first determine the viability of volumetric occluder acceleration in our *Big Buck Bunny* multi-object scenes. We use only the Buck Bunny character model over a ground floor and a single point light. We reduce the shading workload from 100 shadow rays per pixel to just 1 shadow ray per pixel, and even under this dramatic reduction we observe that the scene is amenable to volumetric occluder acceleration. This result implies that there is a possibility for volumetric occluders to do well, even if we are using point lights and object models with lower polygon count than previously. We repeat

this study with 17 point lights to match the number of point lights that we will use later and find the results unchanged.

Next, we use scenes in their entirety from *Big Buck Bunny* and find that these scenes do not perform well. In fact, overall ray tracing time sometimes increases when using volumetric occluders. We present evidence that suggests that the underlying cause is a severe lack of candidate meshes for generating volumetric occluders. We conclude with a discussion of promising trends that suggest that volumetric occluders will have increased applicability in the future.

4.2 Volumetric Occluder Generators

There are a handful of technical issues that need to be addressed when applying volumetric occluders to multi-object scenes. The first is identifying which objects in our scene can be used to generate volumetric occluders. We do this identification manually and on an object level.

Recall that volumetric occluders are marked in a preprocess phase. We call this phase *volumetric occluder generation*, and refer to the well-behaved meshes (Section 3.1) used within this phase as *volumetric occluder generators*. Not all objects are suited for the role of generating volumetric occluders. For example, an object that has a hole in it should not be used because it does not have a well defined interior or exterior.

We modify our renderer to support loading a scene with multiple objects. The scene itself is formatted as a collection of triangles that are grouped into objects (we use a regular Wavefront OBJ file). This object-level granularity is determined by the modelers of the original scene and we use the objects as they are specified in the *Big Buck Bunny* scene files. We have found that a character model is typically a single object, as is the trunk of a tree. However, the leaves of that same tree are considered another object since they are designed to be shaded with a leaf shader

rather than the tree's trunk shader.

Once the scene is partitioned into objects, we manually identify which objects in the scene are well-behaved, although more automated methods are certainly possible (Section 3.2). These well-behaved objects are marked as the volumetric occluder generators, which are the only objects visible to the classification rays during our volumetric occluder generation phase (Section 3.3). All other triangles are ignored during this phase. The generators are therefore the only objects in the scene that can create the volumetric occluders.

We will discuss on a scene by scene basis which objects are used as the generators. We typically use the characters in the scene and the closed tree trunks.

4.3 Dynamic Use of Volumetric Occluders

Another modification to the algorithm is the use of volumetric occluders only when necessary. Since volumetric occluders are now far more sparsely distributed in the scene, we implement a mechanism that tests if a shadow ray is even anywhere near a volumetric occluder before using the more expensive modified kd-tree traversal (Section 3.4).

From the volumetric occluder generation phase, we keep around a single bounding box, which is the bounding box that encloses all generator objects. If dynamic use of volumetric occluders is enabled, each shadow ray is tested for whether it overlaps with the overall bounds of all generators. If it does, then the more expensive modified kd-tree traversal order is used (either QDBFS or deferred intersection). Otherwise, we default to normal kd-tree traversal, i.e. standard ray order, which is the most efficient approach when we know there are no volumetric occluders along the ray's path.

All runs in this chapter assume that dynamic use of volumetric occluders is enabled.

4.4 Aggressive Flush of Deferred Intersection List

When using the deferred intersection list in a scene that is sparsely populated with volumetric occluders, it is possible that shadow rays that skim the surface of an object may push geometry onto their deferred intersection list that would immediately terminate the ray if it were examined, but instead of performing the intersections (i.e flushing the list) the ray continues traversal through an expanse of empty space, postponing the discovery of an intersection. To address this problem, we use *aggressive flush* of the deferred intersection list, which means that every time we enter a clear node, we examine the deferred intersection list for intersections.

This policy is not always the best because it could prevent the finding of volumetric occluders that would otherwise be found under our old non-aggressive policy (which speculatively explores more kd-tree nodes), but we have found in scenes with sparse volumetric occluders this policy offers a slight performance improvement. As such, this policy is used throughout the remainder of this chapter.

4.5 Exporting the Light Configuration from Blender

Our goal is to simulate a real workload, and in addition to using the object models from a real scene, we put forth a best effort within the constraints of our system to use the same lighting configuration. We take all lights from the original scene and add them as point lights in our scene.

Our export pathway captures the macrostructure of the lighting configuration used in the original scene, although there are some problems due to the reduced feature set in our renderer. One problem is that our renderer ignores the fact that in Blender certain lights only affect certain parts of the scene. For example, compare the leaf shadows on the floor in the original Devious Trio scene versus the version in our renderer (Figure 4.18 vs Figure 4.26). In the original scene the leaves are

marked as being invisible to the overhead light so they do not cast a shadow onto the ground. Our renderer lacks such fine-grained per-object lighting support, so the shadow appears on the ground.

Another problem is that the original lighting configuration in Blender uses area lights and spot lights in addition to point lights to illuminate the scene. Our renderer, however, only supports point lights. Area lights were studied in the previous chapter, so we know they can work quite well with volumetric occluder acceleration, although it would have been nice to support them in these results as well.

With these disclaimers in mind, the results presented in this section do faithfully reproduce the location of the lights from the original scene if not the exact lighting configuration. Although this is a shortcoming, our goal with this work is to get reasonably close to the original lighting configuration.

4.6 Shortcomings of Our Approach

Before detailing our results, we will first discuss shortcomings that are directly a result of our experimental methodology.

We do not have support for fur, which is prominently featured in Larry's tail (Figure 4.40). Fur is often quite an expensive feature to support in a renderer, and this expense directly impacts the shadow rays that we are studying. However, due to limited implementation time we do not support fur shaders. It is hard to tell *a priori* how this omission affects our results, since fur shading will make shadow rays that we can accelerate more expensive (shadow rays that generate the umbra region of a shadow), as well as shadow rays that we cannot accelerate (ones that generate the penumbra region).

We also do not support subdivision surfaces. It is obvious that we are rendering the control mesh rather than the final mesh in certain parts of the scene, such as the tree root in the Devious Trio image (compare Figure 4.18 vs Figure 4.26). Subdi-

vision surfaces would, like fur shading, increase the time spent for both acceleratable and unacceleratable shadow rays.

As mentioned in the Section 4.5, the differences in lighting configuration may add some extra imprecision to our results.

4.7 Results

We will first describe our methodology. Next, we will describe our first experiment which explores the use of volumetric occluders with only the Buck Bunny model and a floor plane. We choose to study this simplified subset of a full scene in order to isolate the suitability of the individual objects in the scene for use with volumetric occluders.

We then discuss our experiments that use entire scenes from *Big Buck Bunny*. These scenes capture the full effect of using multi-object scenes and also of using scenes representative of an actual professional offline rendering project.

4.7.1 Experimental Methodology

Rather than modifying the Blender renderer to include the use of volumetric occluders, our overall approach is to export the scenes from Blender into our custom renderer, which does support volumetric occluders. This approach was easier to develop than modifying the Blender renderer, but it comes with an important drawback: it is unclear how much our work ultimately contributes to the final cost of rendering a Blender scene. For example, it may be possible for the benefit of accelerating visibility to be negligible compared to the cost of shading and features that we do not support in our *Big Buck Bunny* renderings, including fur shading (see squirrel tail in Figure 4.40), model subdivision (compare tree root in Figures 4.18 and 4.26), and grass shading (compare presence of background grass in Figures 4.28 and 4.34). For more discussion on the cost of visibility versus shading, see Section 2.9.

All results were gathered using a single core of a 2.66 GHz Intel Core 2 Q6700 processor, 1066 MHz FSB. All images are generated at 1024x1024 resolution. At each shading point a single shadow ray is cast to each light and, due to the lack of coherent shadow rays, tracing is done via a single ray tracer. Only point lights are used in these scenes. Finally, a shadow ray is not cast at all if the surface from which it originates is back-facing to the light.

The volumetric occluder cache is always enabled, and each light has a light-specific cache that holds 4 occluders. Also, for all runs that use the deferred intersection list, a list size of 8 is used.

Figure 4.2: The protagonist from *Big Buck Bunny* in his default animation pose (T pose). A single point light is used.

4.7.2 Rendering the Buck Model with 1 Light

We first use volumetric occluders with just the primary protagonist model, whom we will refer to as Buck after the title of the movie. Buck is placed over an infinite

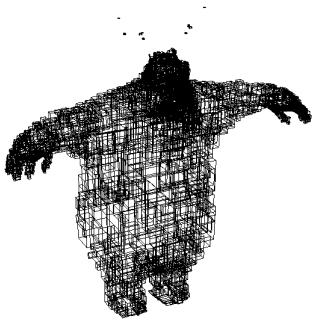


Figure 4.3: The volumetric occluders that are created from the Buck model.

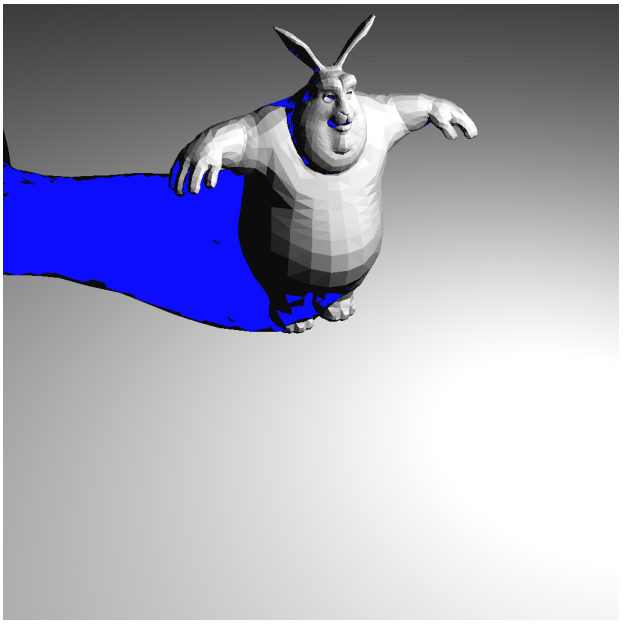


Figure 4.4: A blue pixel means that the shadows ray for that pixel was terminated by a volumetric occluder.

settings	time (s)	% reduction
baseline	0.29	0.0%
bfs	0.27	6.9%
di	0.26	10.3%

Table 4.1: Shadow rendering times for the Buck model with a single point light. *bfs* is quick descent breadth-first search, *di* is deferred intersection.

floor plane (Figure 4.2). We expect this simple test scene to provide insight into the suitability of the models from a real production scene for use with volumetric occluders.

We decide to test Buck because he is a closed model that is prominently featured throughout the shots in the movie. Figure 4.3 shows the volumetric occluders generated from Buck. His interior volume, with the exception of his ears, is nicely filled in by the volumetric occluders. Volumetric occluders also see frequent use within the shadowed pixels, as shown in Figure 4.4. A pixel with a blue hue means that shadow rays cast from that pixel were terminated by a volumetric occluder.

The one drawback of using the Buck model as a volumetric occluder generator is that its polygon count is quite low. Consisting of 8136 triangles, this model is only slightly more complex than the simplest model in our previous test suite, the Cowboy model which has 6752 triangles. Models which perform the best with volumetric occluders contain triangle counts as high as 871k (Dragon model). See Section 3.8.2 for more discussion on the effects of polygon count on volumetric occluder effectiveness.

Nevertheless, we still see an improvement in run-time when using volumetric occluders (Table 4.1). The runtime reduction ranges from 7% - 10%, which overcomes an important initial hurdle: we now know that volumetric occluders can accelerate the shadows cast by the object models from *Big Buck Bunny*. This runtime improvement is less than the 50% reduction we observed in our single object tests, although it is worth noting that this object model is much simpler than the

ones used before.

4.7.3 Rendering the Buck Model with 17 Lights

We increase the number of point lights to 17, to match the number of lights used in a typical scene from *Big Buck Bunny*. The object model being used is identical to before. There are now far more shadow rays cast and far more shadowing effects present in the final image (Figure 4.5). Volumetric occluders are useful throughout most of the entire shadow region (Figure 4.6), with the exception of Buck's ears (Figure 4.3 shows the lack of volumetric occluders in this region).

Figure 4.5: The protagonist from *Big Buck Bunny* in his default animation pose (T pose). The number of point lights has been increased to 17.

We observe that when rendering the Buck model with only one light, volumetric occluders provided some runtime acceleration. When rendering with many lights, the results are even better, as shown in Table 4.2. We now reduce runtime from the baseline by nearly 22%.

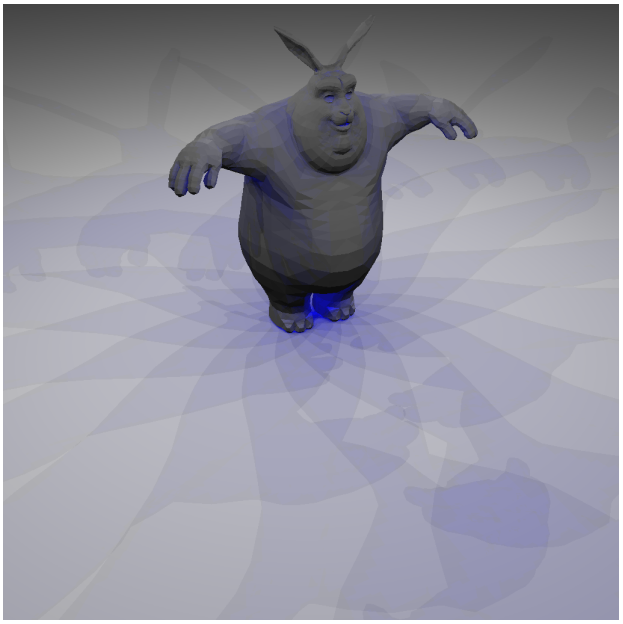


Figure 4.6: A darker blue hue means that more shadows rays from that pixel are terminated by a volumetric occluder. Since there are 17 point lights, the darker blue hue appears in areas where a significant number of shadows which use occluders overlap, such as the area between the bunny's feet.

settings	time (s)	% reduction
baseline	5.65	0.0%
bfs	4.75	15.9%
di	4.41	21.9%

Table 4.2: Shadow rendering times for the Buck model with 17 point lights. *bfs* is quick descent breadth-first search, *di* is deferred intersection.

scene	cache hit rate	overall use
1 light	7.3%	23.9%
17 lights	10.5%	35.7%

Table 4.3: Important differences in volumetric occluder effectiveness as the number of lights increase. *cache* refers to the volumetric occluder cache. *overall use* refers to the total number of shadow rays terminated by volumetric occluders as a percentage of all shadow rays sent to the kd-tree. Results are given for the runs using the deferred intersection list.

There are at least two contributing factors to this increased acceleration. These factors are listed in Table 4.3. The first factor is the hit rate within the volumetric occluder cache. Recall that a hit in the volumetric occluder cache provides the most runtime reduction because it eliminates the need for ray-triangle intersections *and* kd-tree traversal steps, replacing those with a small number of ray-bounding box tests.

The cache hit rate is greater in the 17 light scene because the shadows on the floor are bigger. Consider the volumetric occluders within Buck. Project these occluders onto the floor. A bigger shadow means the projected occluders are also bigger, which means that the likelihood that neighboring pixels on the floor overlap the same occluder also increases. This trend manifests as a higher hit rate in the occluder cache.

Additionally, Table 4.3 indicates the overall use of volumetric occluders has increased for the 17 light scene. This statistic refers to the number of shadow rays that are terminated due to the use of volumetric occluders, normalized by the number of shadow rays that are sent to the kd-tree. Shadow rays are not sent to the kd-tree if they are back-facing to the shading normal or if they miss the bounds of the kd-tree. The overall use of volumetric occluders also increases for the same reason that was discussed earlier, specifically the shadows introduced by the extra lights are bigger, meaning they occupy more pixels and thus provide an increase opportunity for volumetric occluders to provide acceleration.

Overall, the observed increase in volumetric occluder acceleration when using

scene	lights	tri	vol occ tri
Bunny Hole	17	383	8
Devious Trio	26	455	8
Pushing the Trunk	15	2,570	50

Table 4.4: The *tri* column is the total number of triangles (thousands). The *vol occ tri* column is the number of triangles that are part of a volumetric occluder generator (thousands). There is a scarcity of objects in the scene that can serve as generators.

17 lights is promising because the full scenes from Big Buck Bunny that we are targeting have light counts that are in this range.

4.7.4 Testing Full Scenes

We now expand our study to full test scenes from *Big Buck Bunny*. Details for our three test scenes are given in Table 4.4. We use the polygonal models as they are specified in Blender. Each scene consists of 15 or more point lights scattered throughout the scene, according to the location of the original lights in Blender. Our conversion is far from perfect, however. Section 4.6 describes the various caveats that are involved in our export and rendering pathways.

Not all models within the Blender files are as amenable as the Buck model for use with the volumetric occluders approach. For example, many of the trees in the scene are not closed models. The danger of such open models is that kd-nodes within the interior of the tree may be mistakenly identified as outside, which hurts performance, or kd-nodes on the exterior of the tree may be mistakenly identified as inside, which would lead to unacceptable visual artifacts. Figures 4.7, 4.8, and 4.9 show trees from our test scenes that were identified as being unsuitable for volumetric occluder generation.

Overall, we've found that only a very small portion of the meshes in the scenes meet the prerequisites for volumetric occluders (Section 3.1). These meshes are typically the protagonist Buck or two of his three antagonists, which we have named *Larry*, *Curly*, and *Moe*, together forming the *Devious Trio* (Figure 4.10).

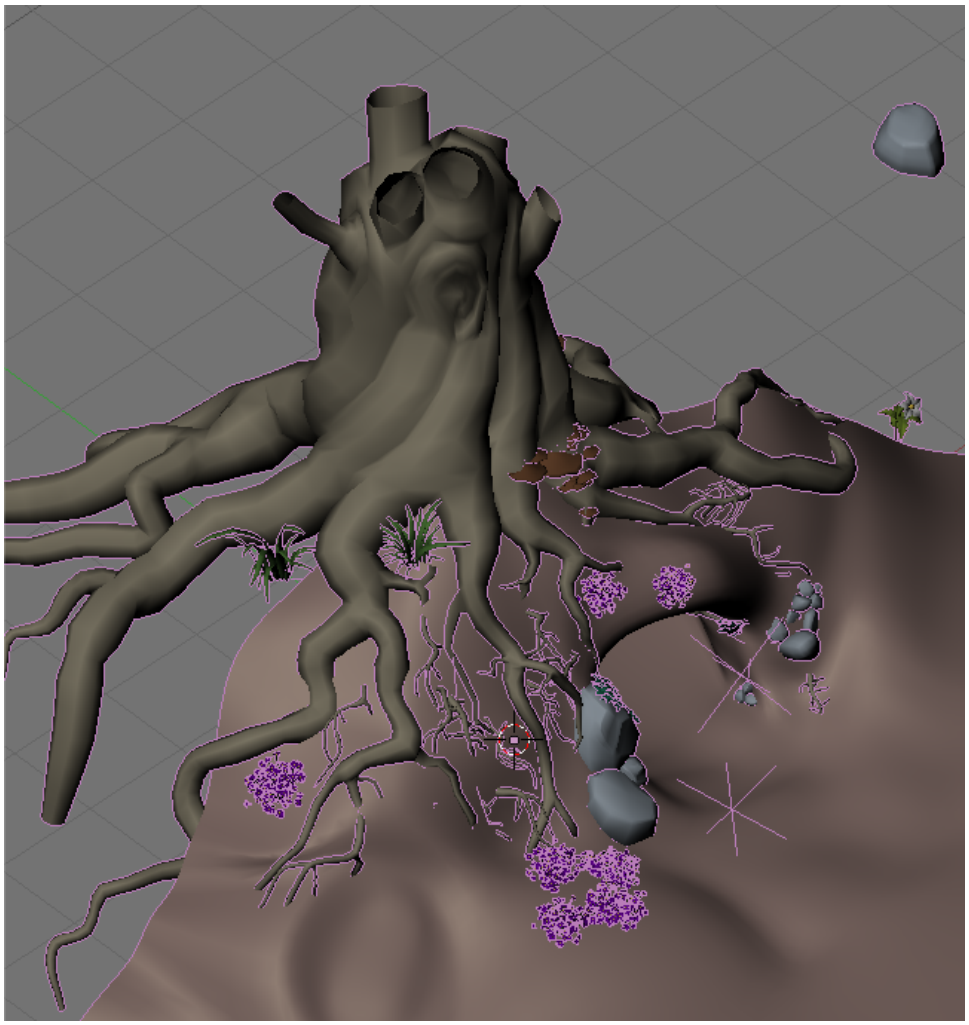


Figure 4.7: Top view. Tree trunk in our first test scene that is open on the top.

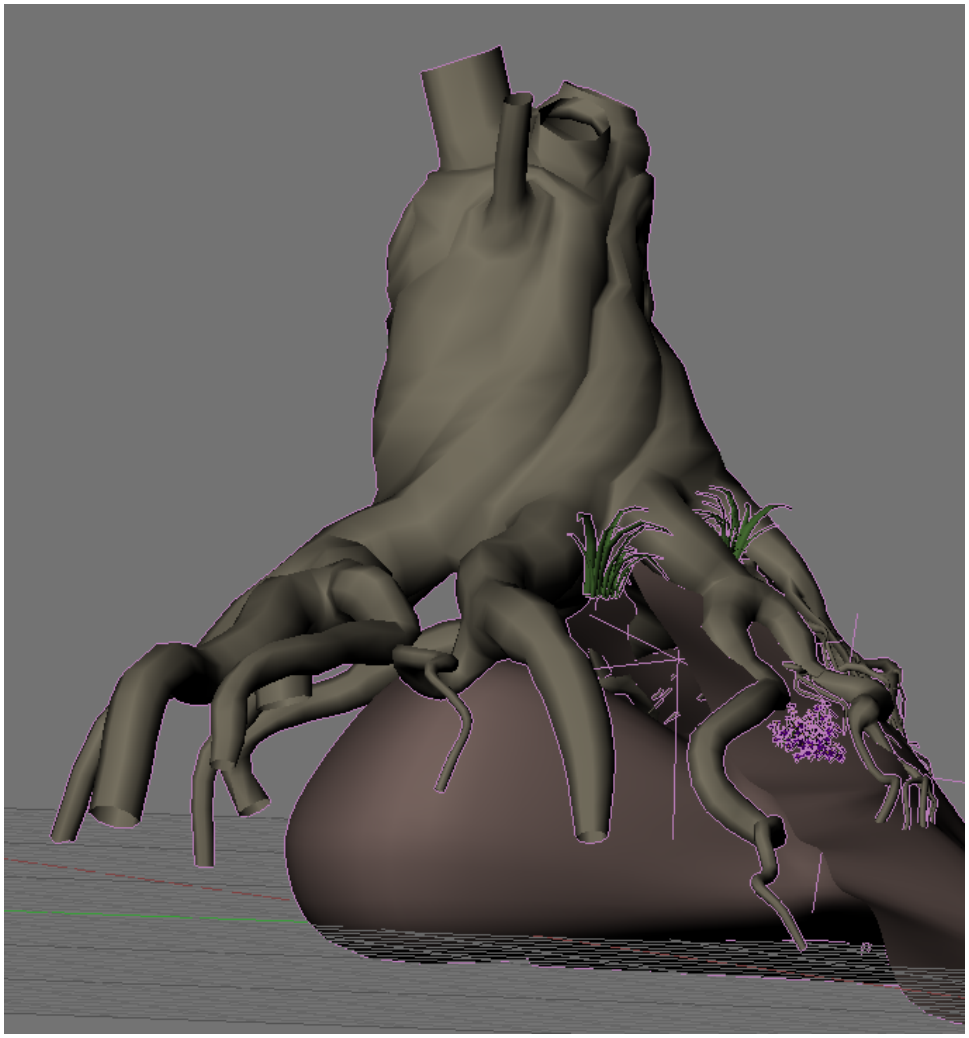


Figure 4.8: Side view. The tree trunk is also open at the roots.

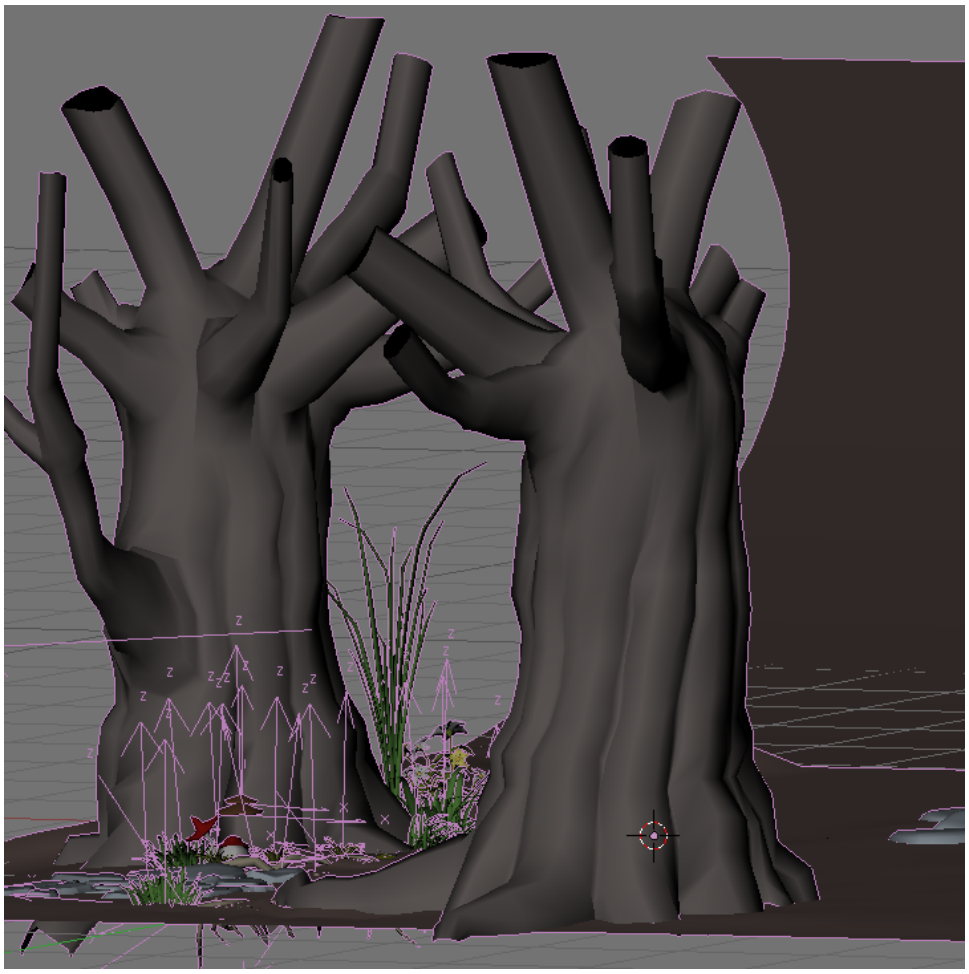


Figure 4.9: The trees in our second test scene are also open.

Figure 4.10: The Devious Trio. From left to right, the members are Curly, Larry, and Moe.

4.7.5 Visual Discrepancies

There are various differences between our rendered images and Blender's rendered images , some of which we have discussed before (see Section 4.6). There is a significant visual difference between our renderings and the final frames used in *Big Buck Bunny*.

We take every light in the Blender file description and treat them as point lights, while Blender often treats these lights as a spotlight instead. More importantly, we consider each light to have global effect, meaning they cast light on all objects as one would expect in a classical ray tracing environment. However, Blender often labels lights as affecting only certain objects, while the other objects are invisible to the light. They receive no lighting contribution from the light. This is the main reason why the lighting in our images is different from that in the final frames used in *Big Buck Bunny*, particularly in areas of the scene that are brighter than the reference image. While proper handling of lights is important, we can still study the effectiveness of volumetric occluders without fully implementing these details.

Additionally, we do not perform normal interpolation or mesh refinement. Our lack of normal interpolation means our faces will be more faceted in appearance. The lack of mesh refinement also contributes to the faceted look, and also produces coarser, piecewise linear silhouettes. These deficiencies, which should be fixed for obtaining high image quality in a production renderer, have little to no impact on the effectiveness of volumetric occluders on improving performance, which is our focus.

4.7.6 Scene 1: Bunny Hole

This scene is near the opening of the movie. Buck is emerging from his home, which is a hole in the ground (Figure 4.11). The scene in Blender is shown in Figure 4.12.

The only good volumetric occluder generator in this scene is Buck himself (Figure 4.13). The rabbit hole, although being the primary shadow caster in the scene, is an incomplete thin shell which only models the visible interior of the bunny hole and has no enclosed volume. The backside of this mesh is visible as the bulbous brown mesh right below the tree in Figure 4.8.

With our generator mesh selected, we are ready to create the volumetric occluders from it (Figure 4.14). Note that Buck's ears are particularly thin so few empty nodes are created there during the initial kd-tree build, which explains the sparsity of opaque nodes in this region.

Our renderer produces the image seen in Figure 4.15. Note that a big difference in visual quality is the interior of the bunny hole. In our rendering, we treat all lights from the scene as classical point lights within a ray tracer, so the area within the bunny hole is illuminated. We suspect that the darkened bunny hole in the original rendering is due to lights that are annotated with a list of geometry that they do not affect. We do not support this feature in our renderer, although we do not believe this omission impacts our results.

setting	preproc	primaries	secondaries	secondaries % reduction
baseline	0.0	0.9	18.8	0.0%
bfs	0.6	0.9	17.9	4.8%
di	0.6	0.9	17.0	9.6%

Table 4.5: The preprocessing time and time taken for tracing primary and secondary rays (seconds) for the Bunny Hole scene. The deferred intersection list has 8 elements.

We visualize which pixels benefit from the use of volumetric occluders in Figure 4.16. Note that some lights are below Buck, which is why the volumetric occluders cast shadows onto the ceiling of the bunny hole.

We originally chose to explore this scene due to the extensive shadows in the bunny hole. These shadows turned out to be unacceleratable due to the construction of the hole as an open thin shell. The run-times for this scene are given in Table 4.5. Even factoring in the preprocess time, volumetric occluders provide a modest improvement in overall render time, reducing the time taken on secondary rays by up to 1.8 seconds, while requiring an additional 0.6 seconds for preprocessing time.

One reason why we see a lack of improvement is because the number of volumetric occluders, relative to the number of kd-tree nodes, is quite low, in fact much lower than in our previous single object scenes (Section 3.7). This difference will be a recurring theme in our *Big Buck Bunny* results (Section 4.8.2). We see this reduction reflected in the operation counts for this scene (Figure 4.17). A lower number of volumetric occluders means that fewer shadow rays will be terminated by them, as reflected in a less pronounced reduction in the number of ray triangle intersections compared to the previous results for single object scenes.

4.7.7 Scene 2: Devious Trio

Since using Buck was not giving particularly good results, we decided to try a scene that used the other characters, the Devious Trio of Larry, Curly, and Moe. We try one scene of many which features all three characters and which also has some

Figure 4.11: Final frame render of the Bunny Hole opening scene from the original movie.

shade from other objects in the scene (Figure 4.18). The original scene in Blender, including the haphazard lighting conditions, can be seen in Figures 4.19 and 4.20.

We mark the trio as the volumetric occluder generators in the scene. While some Trio members are fine (Figure 4.21), one Trio member is actually an open mesh. Curly (the red squirrel) is discontinuous in the area around his jaw (Figure 4.22), which causes bad volumetric occluder formation. In particular, a node that is on the exterior of all three generator meshes is marked as opaque because the identification ray for that node happened to strike a backface on Curly's discontinuous mesh segment, erroneously indicating that the node was inside of Curly (Figure 4.23). So instead of three volumetric occluder generator meshes, we only use two for this scene, Larry the front squirrel and Moe the chinchilla. (Figure 4.24).

Another problem is that all of the trees are composed of a collection of leaves which are not viable candidates for volumetric occluder generation. Had the shadows in the scene been cast by the trunk, they might have benefited from volumetric occluders. However, as it stands the leaves, which are a collection of

Figure 4.12: The full scene as modeled in Blender.

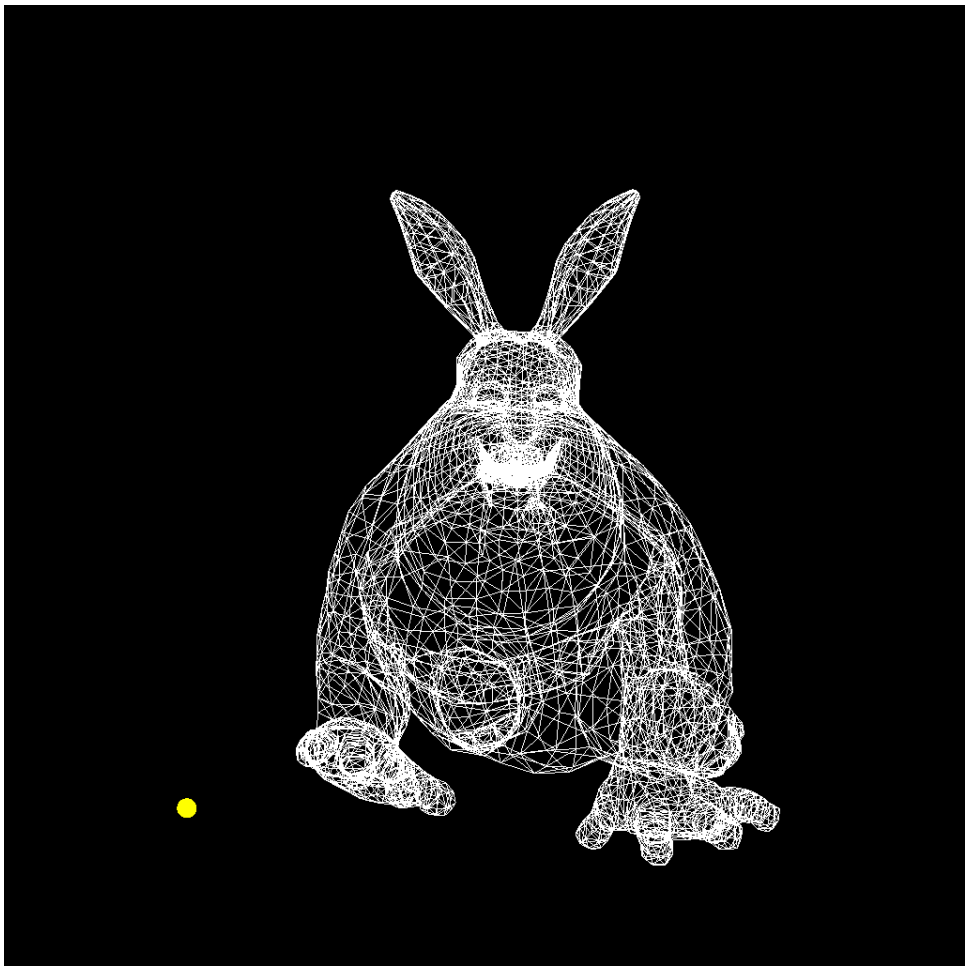


Figure 4.13: Buck is being used as the generator mesh for volumetric occluders.

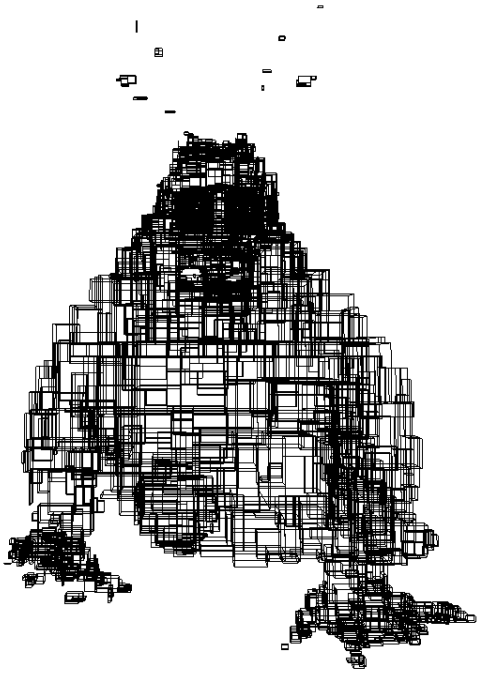


Figure 4.14: Volumetric occluders created from the Buck mesh.

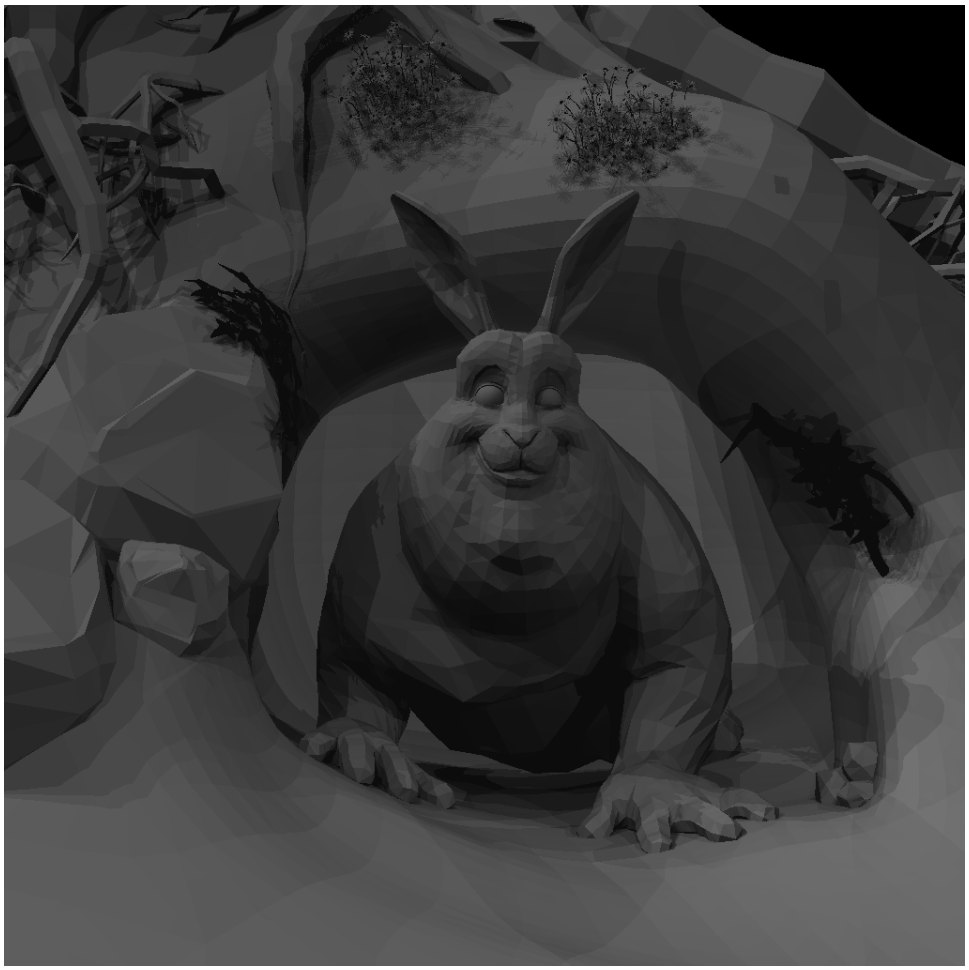


Figure 4.15: The final rendering of the Bunny Hole scene in our renderer. Notice the extensive shadows throughout the scene due to the scattered placement of the 17 lights that illuminate the scene.

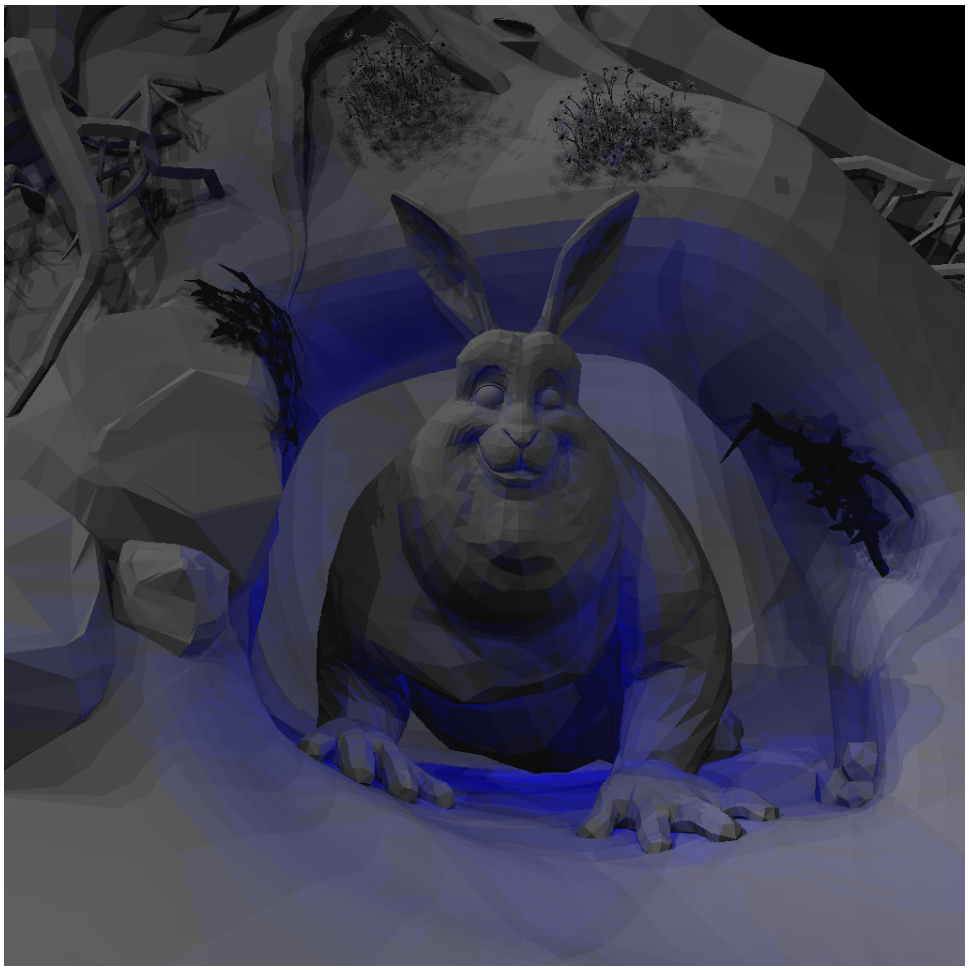


Figure 4.16: A darker blue hue means that more shadows rays from that pixel are terminated by a volumetric occluder.

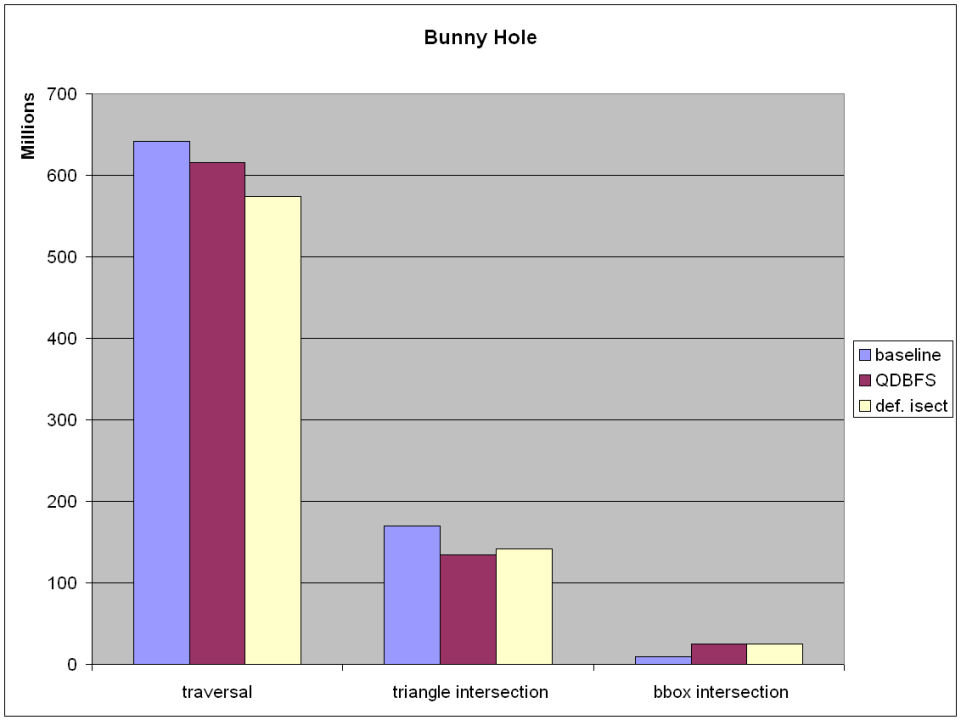


Figure 4.17: Operation counts for rendering the Bunny Hole scene. The decrease in triangle intersection tests is less pronounced than in the single object scenes because the volumetric occluder occurrence rate is lower.

setting	preproc	primaries	secondaries	secondaries % reduction
baseline	0.0	0.8	28.5	0.0%
bfs	1.1	0.8	29.3	-2.8%
di	1.1	0.8	28.9	-1.4%

Table 4.6: The preprocessing time and time taken for tracing primary and secondary rays (seconds) for the Devious Trio scene. The deferred intersection list has 8 elements.

simple meshes with a very small interior volume compared to their mesh complexity, are not amenable to our approach.

Along these lines, we again see the problem of not having enough volumetric occluders generated for the total number of nodes in the kd-tree. The Devious Trio scene has the highest percentage, although it is about 9x worse than the average rate from our original test scenes (more detail in Section 4.8.2).

Table 4.7 shows the runtimes for the Pushing the Trunk scene (the rendering from our system is shown in Figure 4.34). We observe the worst performance yet in this scene. Both versions of volumetric occluders (QDBFS and deferred intersection) are unable to improve the trace time of secondaries and they require an extra 1.1 seconds of preprocessing time. We believe the underlying cause is due to this scene having a significantly lower ratio of opaque nodes to total kd-nodes. Opaque nodes comprise only seven-tenths of a percent of the overall leaf nodes in the kd-tree, the lowest percentage from even other *Big Buck Bunny* scenes by a factor of 2 (Section 4.8.2).

We visualize which pixels benefit from the use of volumetric occluders in Figure 4.25. Table 4.6 summarizes our inability to improve the runtime of secondary rays in the Devious Trio scene (our final render is shown in Figure 4.26). Measuring operation counts (Figure 4.27) indicates that volumetric occluders do not provide a reduction in either traversal steps or ray-triangle intersection tests for this scene. We believe this trend is due to how few volumetric occluders are utilized in the scene compared to our experiments on the single object scenes.

Figure 4.18: Final frame render of the Devious Trio, looking for trouble, from the original movie.

4.7.8 Scene 3: Pushing the Trunk

We finally resort to looking for a scene with a volumetric occluder generator other than the characters. Recall that an ideal mesh for occluder generation should be composed of many triangles, while having a large, contiguous interior volume. Additionally, they should be positioned relative to the lights so that they cast expansive shadows on the screen (an otherwise good occluder which happens to cast no shadows is not helpful).

Fortunately, the Pushing the Trunk scene has a polygon mesh, the tree trunk (center of Figure 4.28), which does a better job of meeting these criteria than our previous character meshes. The tree trunk is more complex than the characters, specifically it is composed of 42k triangles in contrast to the Buck model which only has 8k triangles. Also, it is positioned so that it casts several expansive shadows on the ground. The volumetric occluders created from Buck and the tree trunk are shown in Figure 4.29.

This scene has 15 lights (Figure 4.30). A close up view of the model in

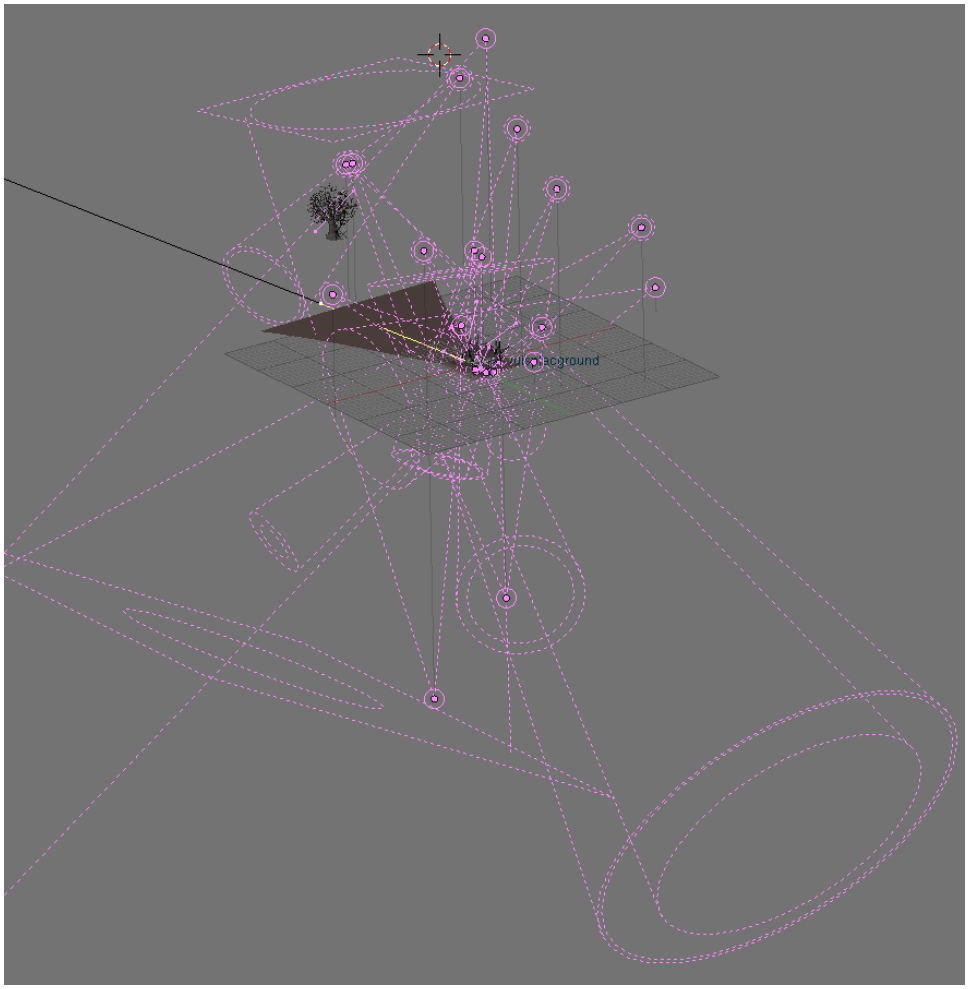


Figure 4.19: The scene is lit by 26 lights.

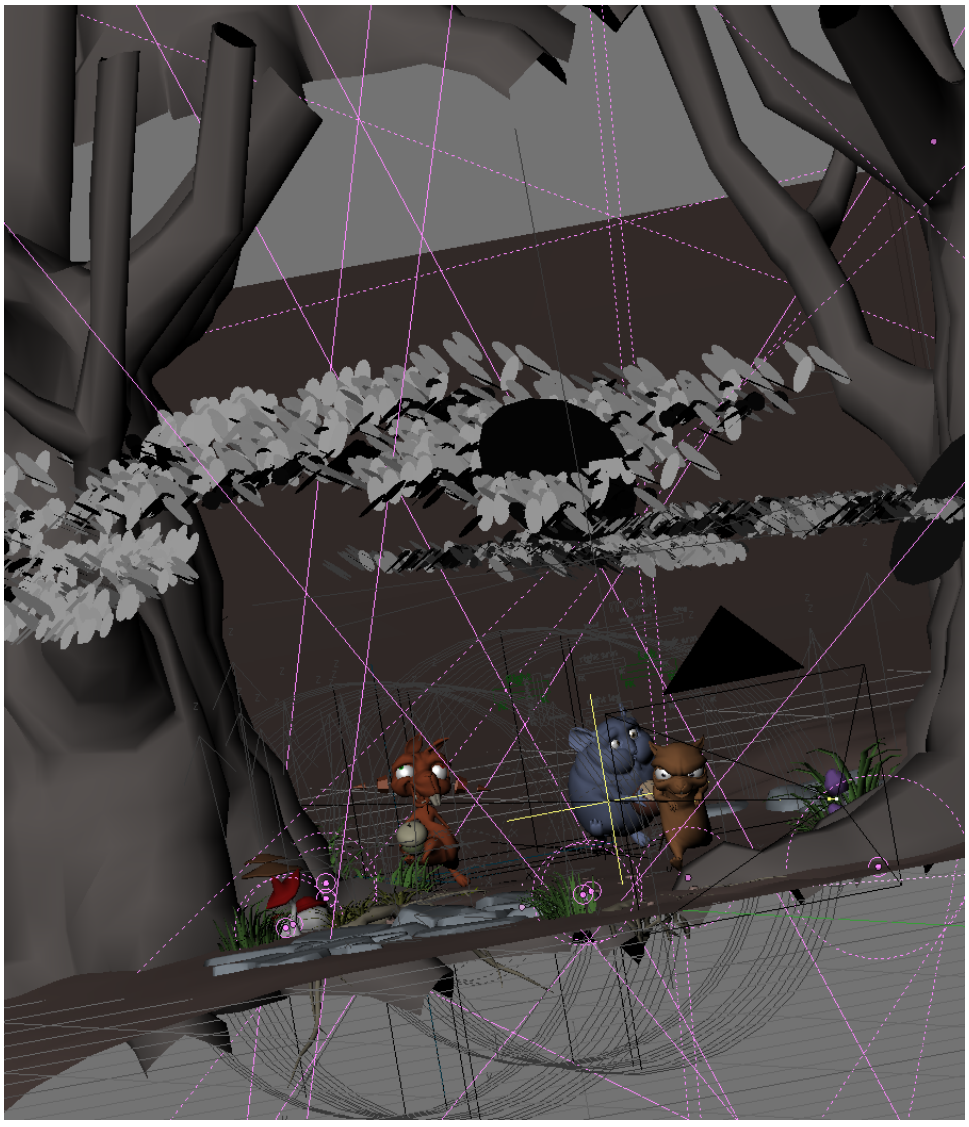


Figure 4.20: A close-up of the lighting configuration. Circled dots are point lights, while the pink lines represent the extent of Blender spotlights.

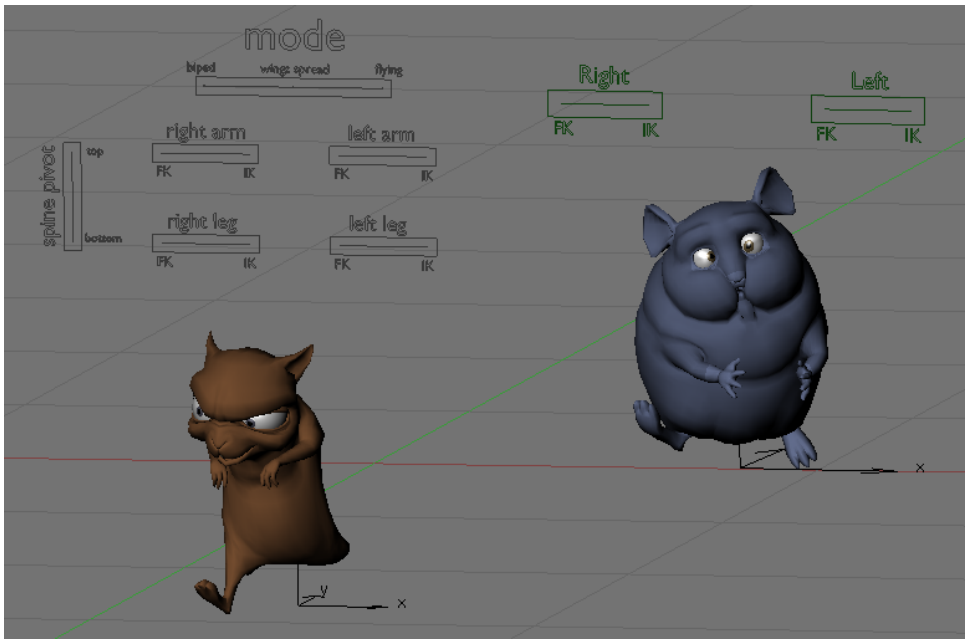


Figure 4.21: Larry and Moe are the volumetric occluder generators used in this scene.

Blender can be seen in Figure 4.31.

Note that only 2% of triangles from the full scene can be used to build volumetric occluders (these triangles are shown in Figure 4.32). This means that we can improve intersection tests for these triangles only. For 98% of the triangles in the scene, we can do nothing to accelerate their intersection tests. In other words, there are too many open models in the scene for which volumetric occluders can provide no improvement. The percentage of pixels that show improvement is much lower when the entire scene is taken into account (Figure 4.33).

Table 4.7 shows the runtimes for the Pushing the Trunk scene (the rendering from our system is shown in Figure 4.34). We marginally improve the render time but only when using deferred intersection, with a final reduction time of less than a second. The run with QDBFS on the other hand does not lead to performance improvement. The operation counts in Figure 4.35 suggest that QDBFS does worse

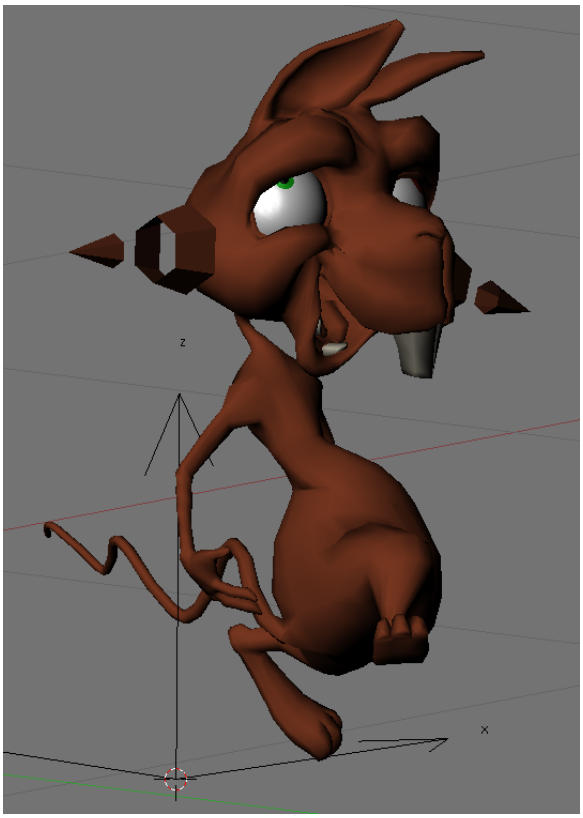


Figure 4.22: Curly cannot be used to generate volumetric occluders because his mesh is open. Note that Curly's elongated cheeks are modeled as disjoint segments from the rest of his head.

Figure 4.23: Volumetric occluders when using the leftmost member of the Trio (Curly), which is represented as an open mesh. Notice the incorrectly identified volumetric occluder on the left side of the image.

because of the extra traversal steps that it requires, which is to be expected when a traversal search inspired by breadth-first search, which may visit unimportant branches of the tree, is used on a kd-tree as large as this one. While normally these potentially diversionary visits to other nodes is made up for with the discovery of useful volumetric occluders, this scene, like others from *Big Buck Bunny*, has a very low volumetric occluder occurrence rate.

The largeness of the kd-tree also has other implications. The preprocessing time is now a considerable amount of the overall time because of the number of nodes that need to be identified. Note, however, that the preprocessing time is a computational investment that is amortized over many invocations of the renderer. If the currently high preprocessing time is a problem, it can be mitigated by doing a simple bounding box check before casting the classification ray. The kd-node that contains the ray origin must be within the bounding box of at least one of the mesh generator objects. If it is not, the node is marked as a clear node with no ray cast

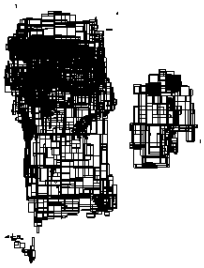


Figure 4.24: Volumetric occluders currently being used. Two of the three members of the Trio, Larry and Moe, are used as the volumetric occluder generators.

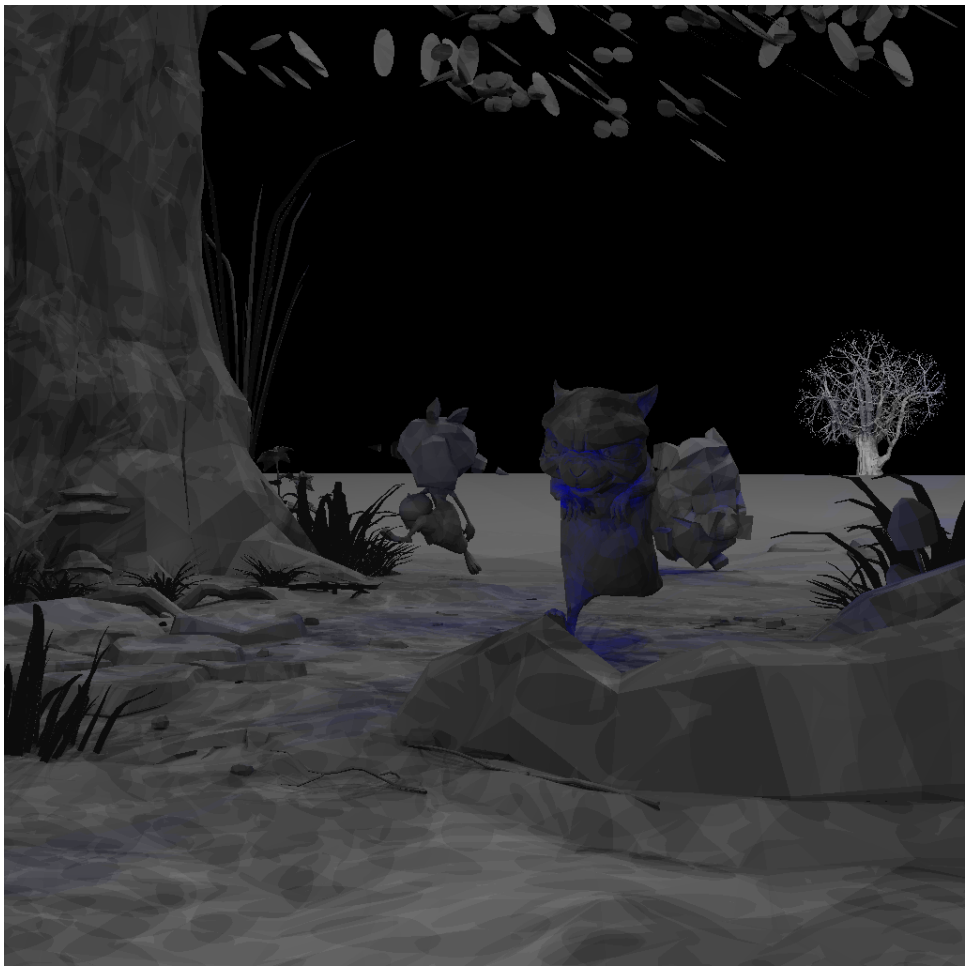


Figure 4.25: A darker blue hue means that more shadows rays from that pixel are terminated by a volumetric occluder. Although there are a lot of shadows in the scene, most of them are being cast by objects which do not generate occluders, such as the leaves.

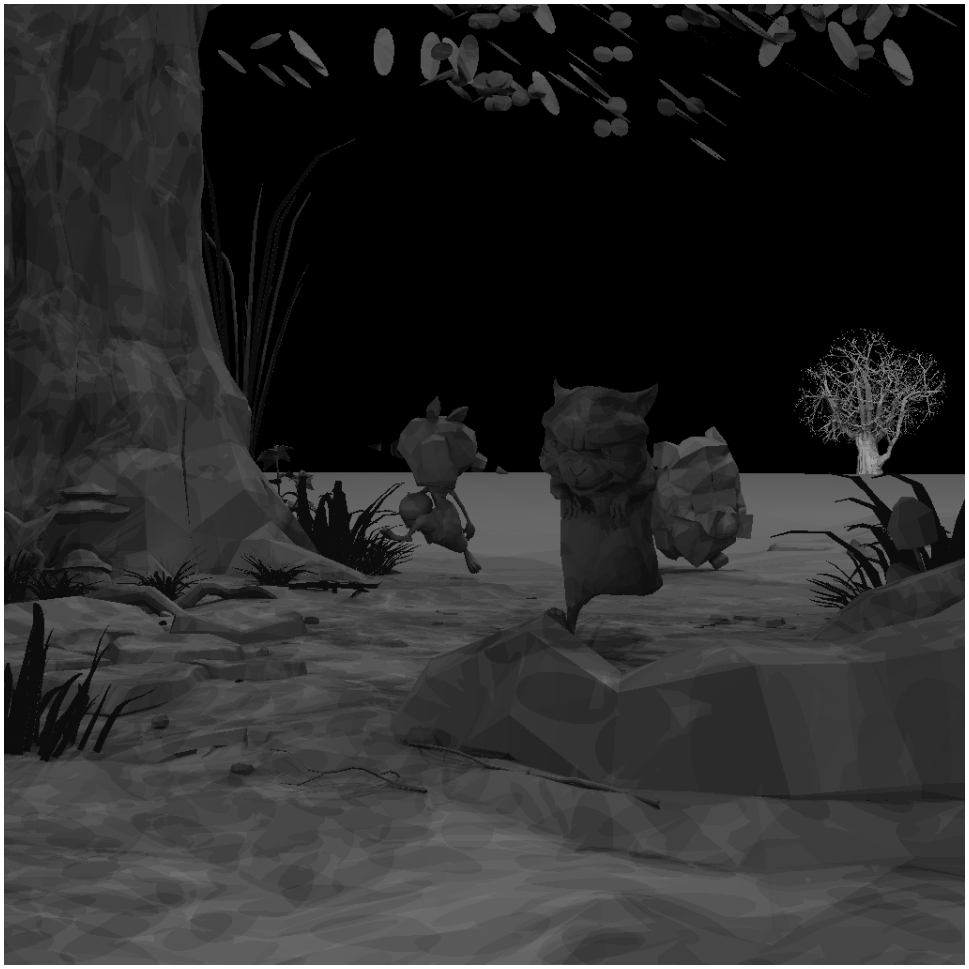


Figure 4.26: Our renderer's output. Note that the characters lack their fur shader and the tree models are noticeably not as smooth as in the final frame used in the movie due to insufficient subdivision. The leaves overhead also cast shadows onto the ground which are not visible in the movie final frame due to annotated lights, which we do not support.

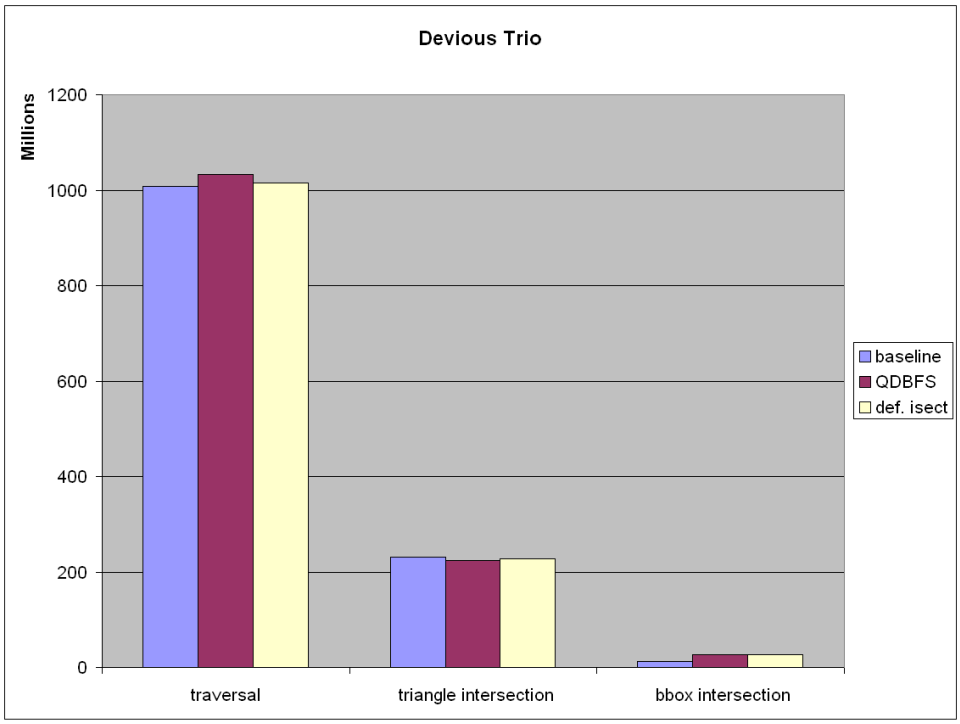


Figure 4.27: Operation counts for rendering the Devious Trio scene. Volumetric occluders provide almost no reduction in operation count in this scene while adding a bit of overhead in the number of bounding box intersection tests.

setting	preproc	primaries	secondaries	secondaries % reduction
baseline	0.0	1.4	24.8	0.0%
bfs	7.0	1.4	25.7	-3.6%
di	7.0	1.4	24.0	3.2%

Table 4.7: The preprocessing time and time taken for tracing primary and secondary rays (seconds) for the Pushing the Trunk scene. The deferred intersection list has 8 elements.

necessary.

We suspect once again that the underlying cause of the underwhelming performance improvement is due to this scene having a significantly lower ratio of opaque nodes to total kd-nodes. Opaque nodes comprise only seven-tenths of a percent of the overall leaf nodes in the kd-tree, the lowest percentage from the *Big Buck Bunny* scenes by a factor of 2 (Section 4.8.2).

Figure 4.28: Final frame render of Buck pushing the trunk from the original movie.

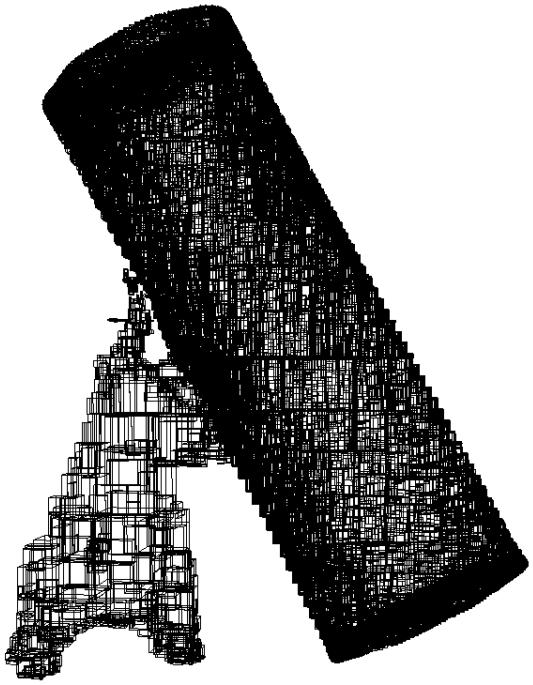


Figure 4.29: Example of volumetric occluders. They represent the interior of the objects from which they were generated.

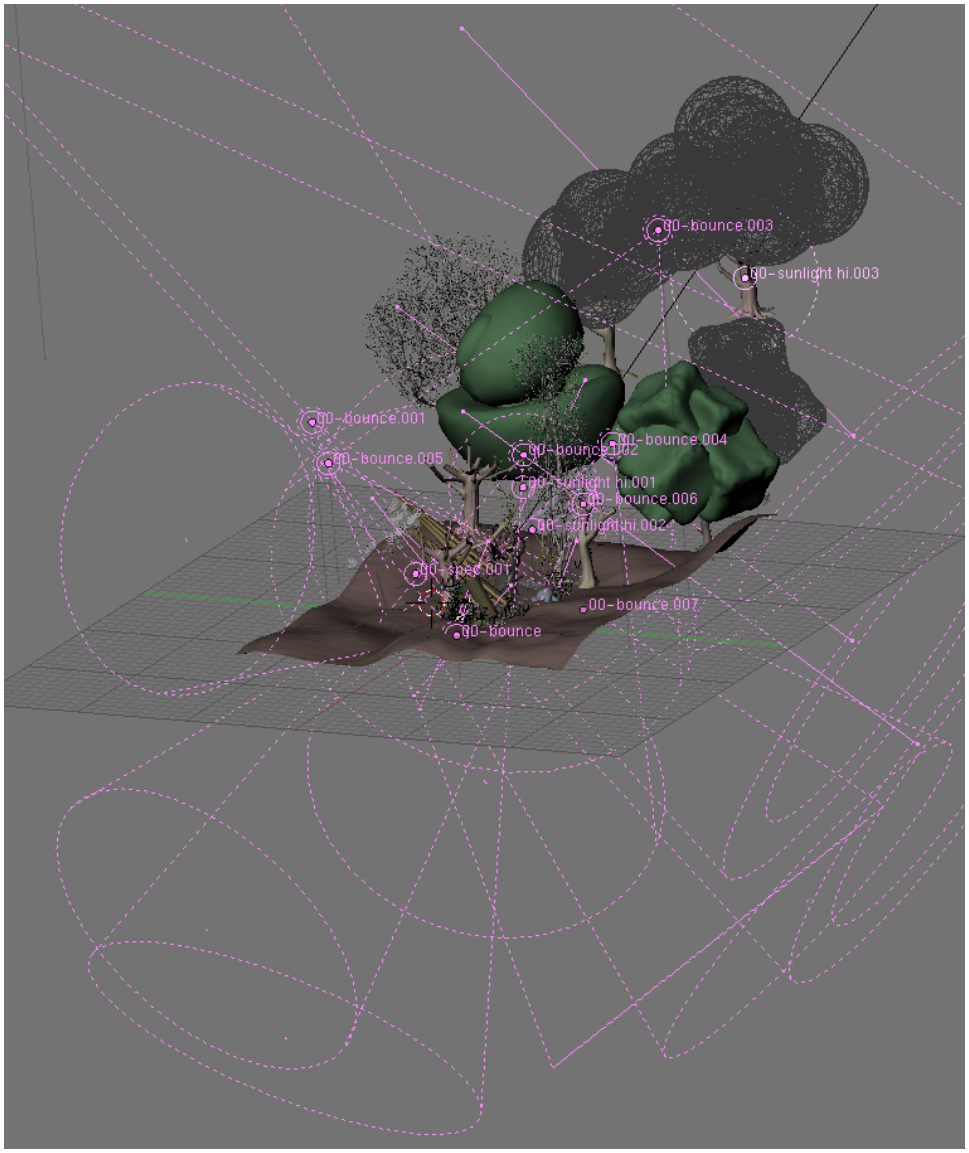


Figure 4.30: All 15 lights in the scene. Circled dots are point lights, while the pink lines represent the extent of Blender spotlights.

Figure 4.31: Close-up of the original scene in Blender.

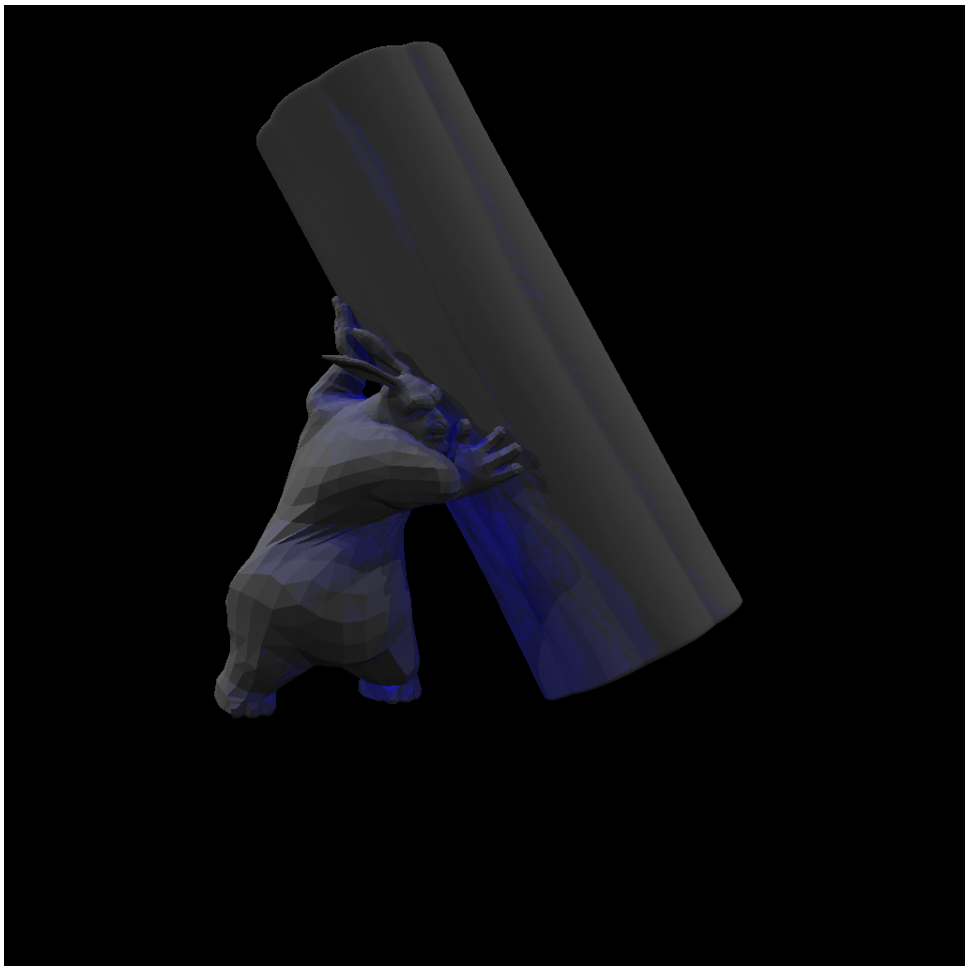


Figure 4.32: A darker blue hue means that more shadows rays from that pixel are terminated by a volumetric occluder. When rendering this image (just the bunny and tree trunk), volumetric occluders improve the runtime by 18%.

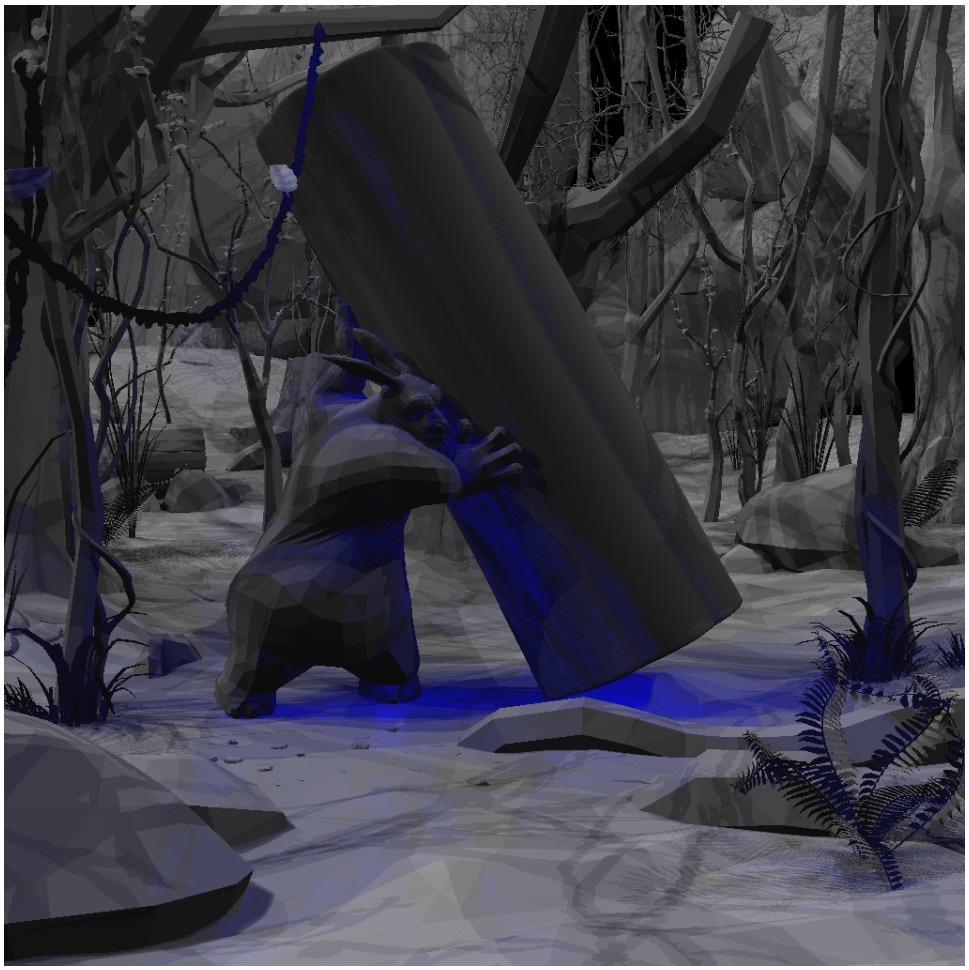


Figure 4.33: A darker blue hue means that more shadows rays from that pixel are terminated by a volumetric occluder. Large regions of the scene do not benefit from volumetric occluders.

Figure 4.34: The rendered image produced by our renderer.

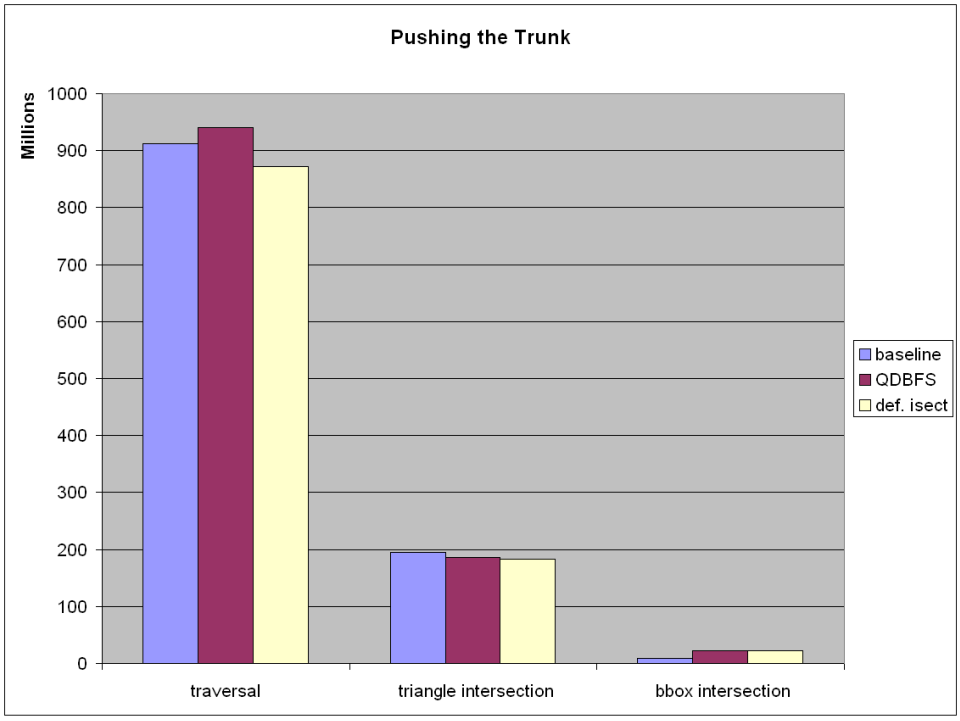


Figure 4.35: Operation counts for rendering the Pushing the Trunk scene. Volumetric occluders provide almost no reduction in operation count in this scene while adding a bit of overhead in the number of bounding box intersection tests.

4.8 Discussion

This section provides discussion on various issues related to volumetric occluders. We will discuss one way of predicting when volumetric occluders will be less than effective at providing shadow ray acceleration. Our predictive method is based on the occurrence rate of occluders within the kd-tree.

4.8.1 Shadow Ray Distribution

Figure 4.36 shows the distribution of shadow rays for the three *Big Buck Bunny* scenes. In contrast to the shadow ray distribution from our single object test scenes (Figure 3.10), the majority of occluded shadow rays no longer hit a volumetric occluder. Instead, these rays must fall back to using triangle intersection. The improvement in shadow ray evaluation time corresponds almost directly to the percentage of shadow rays that encounter a volumetric occluder (blue bar).

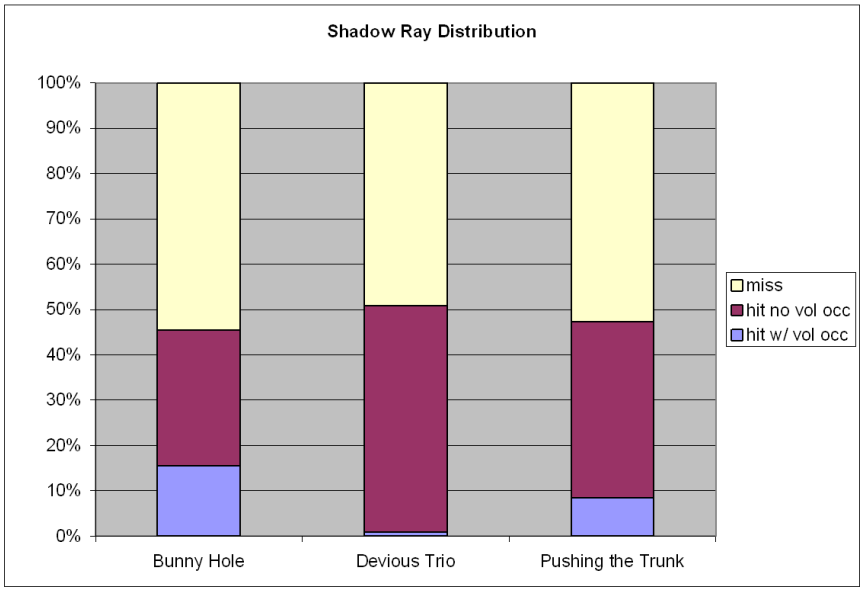


Figure 4.36: The shadow ray distribution when using deferred intersection with the volumetric cache. Three shadow ray types are shown, shadow rays that do not hit anything (miss), shadow rays that hit a triangle (hit no vol occ), and shadow rays that hit a volumetric occluder (hit w/ vol occ).

4.8.2 Volumetric Occluder Occurrence Rate

There is no shortage of shadow rays that are occluded (Figure 4.36), and in fact the shadow ray occlusion rate, just under 50%, is even higher than the 40% rate in our single object test scenes (Figure 3.10). The main reason why shadow rays have trouble hitting volumetric occluders is because there are far fewer volumetric occluders in the kd-tree.

We now describe a simple predictive measure which we call the *volumetric occluder occurrence rate*. This rate is the number of volumetric occluders as a proportion of the total number of kd-tree leaves (recall that the candidates for becoming volumetric occluders are the kd-tree leaves, whose union is a nonoverlapping partition of the entire space represented by the kd-tree).

Our original single object scenes are shown in Figure 4.37. On average, about 18% of the kd-tree leaves are marked as volumetric occluders, which we have observed is a plentiful occurrence rate of occluders for terminating shadow rays. Notice that the scenes that do not do particularly well, such as Yeah Right and Cowboy, have lower volumetric occluder occurrence rates. This relation matches our intuition. If there are fewer occluders in the scene, it is less likely a shadow ray will encounter an occluder during its modified kd-tree traversal, and overall acceleration will be lessened.

However, in our *Big Buck Bunny* scenes we observe that the volumetric occluder occurrence rate has dropped significantly to about 1% (Figure 4.38). Very few objects in the scene can actually be used as a volumetric occluder generator, so very few kd-tree leaves actually become marked as volumetric occluders. Shadow ray acceleration is greatly reduced as a result.

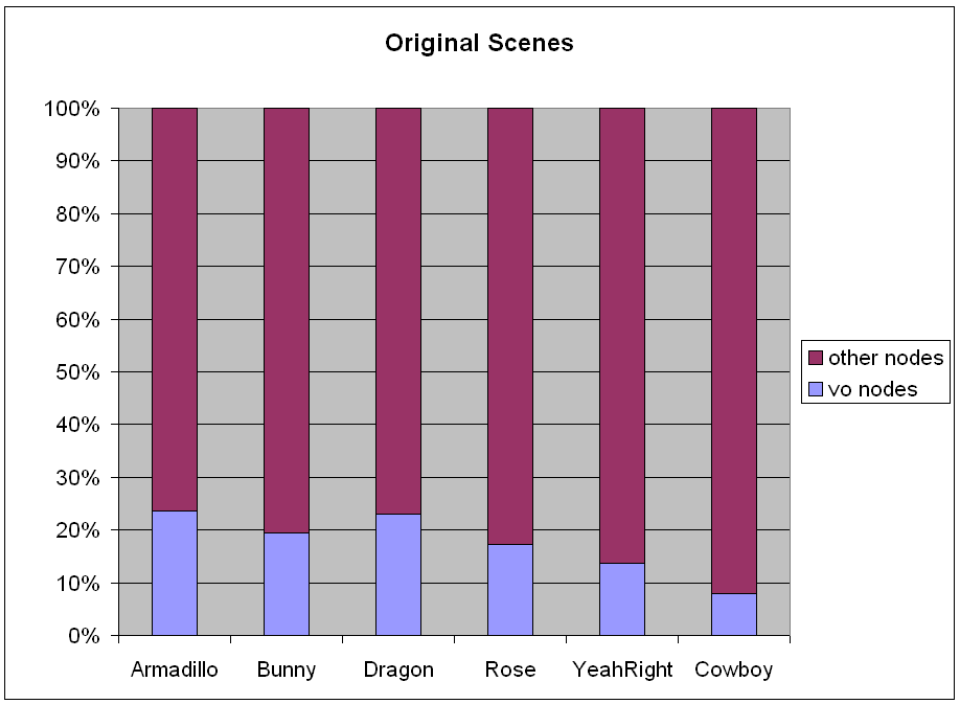


Figure 4.37: The original test scenes, which contain only a single mesh, contain volumetric occluders at roughly an 18% rate. Scenes that do not perform as well with volumetric occluders, like Yeah Right and Cowboy, have lower volumetric occluder rates.

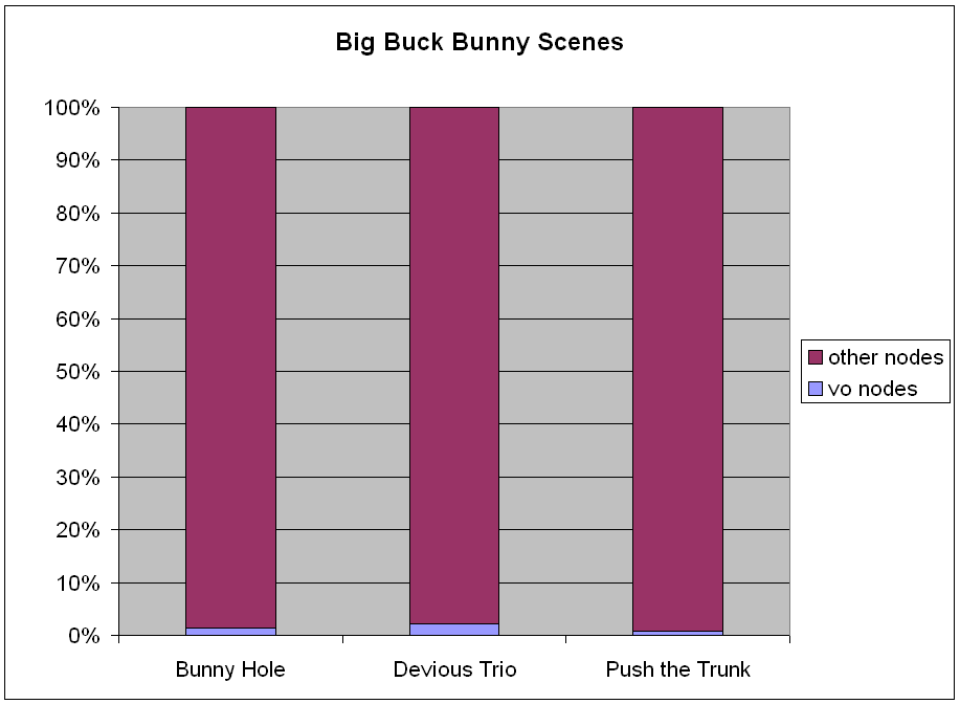


Figure 4.38: Kd-tree leaf nodes become volumetric occluders at a significantly lower rate in the *Big Buck Bunny* scenes, averaging around 1%.

4.8.3 Types of Shadows

While visually inspecting the movie for scenes that would benefit from volumetric occluders, we found that there were a variety of shadow types, some more amenable to volumetric occluders than others.

Leaf shadows are used heavily throughout the movie and are not good candidates for generating volumetric occluders. Leaves are small, are comprised of simple geometry, and contain minimal interior volume (Figures 4.39, 4.40, 4.41, 4.42).

Figure 4.39: Shadows cast by leaves are used throughout the movie. In this example we see leaf shadows on Buck.

Tree trunk shadows occur fairly frequently, however they are only sometimes useful, depending on whether the trunk is closed. An example of a closed tree trunk is shown in Figure 4.43. See Section 4.7.4 for examples of open tree trunks which cannot be used with volumetric occluders.

Rock shadows occur infrequently, which is unfortunate because the rocks themselves are often ideal volumetric occluder generators (Figure 4.44).

Figure 4.40: Larry clings to a tree as more leaf shadows are used to great artistic effect.

Figure 4.41: Buck gives a determined look to the screen in another scene where he is standing under leaves.

Figure 4.42: Forest scenes are not complete without leaf shadows.

4.8.4 QDBFS vs Deferred Intersection

We noticed a slightly different trend in our *Big Buck Bunny* results compared to the single-object test scenes, where the QDBFS traversal order (Section 3.4.3) performed the best overall. In our multi-object test scenes, we noticed the opposite trend, where the extended ray order provided by deferred intersection (Section 3.4.2) performed best.

We observe that QDBFS performs very well for our single object test scenes, beating deferred intersection in the majority of the scenes. However, it is the slower of the two traversal algorithms in all three scenes from *Big Buck Bunny*. We believe that QDBFS is especially well suited for environments with a high concentration of volumetric occluders, but is hampered by unnecessary detours in volumetric occluder sparse environments.

For example, imagine a scene with two objects, neither of which contain volumetric occluders, and a ray whose extended line intersects both objects. QDBFS will have a preference to explore the large interior nodes of both objects before

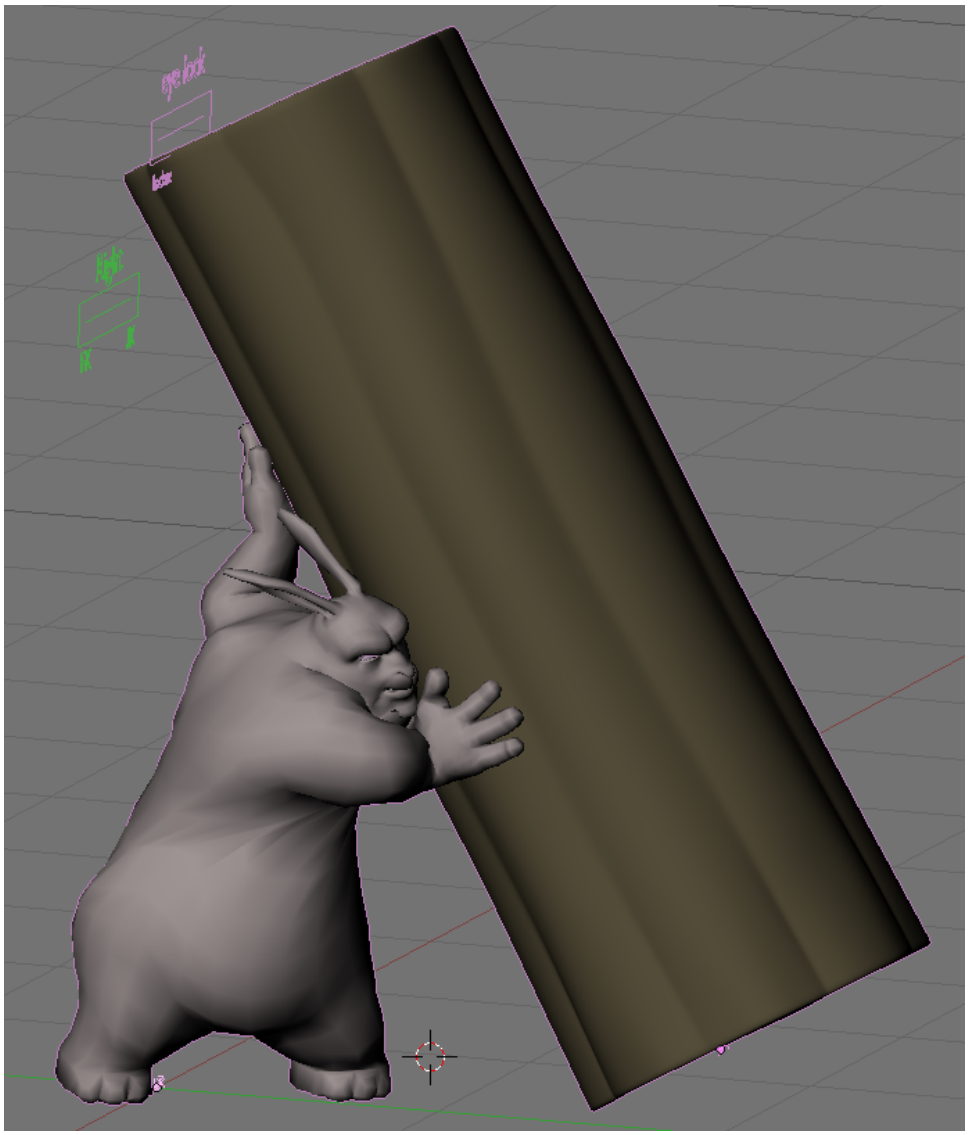


Figure 4.43: Although leaves are not compatible with our volumetric occluder approach, tree trunks are, as long as they are closed like this one.

Figure 4.44: Rocks are prime candidates for being volumetric occluder generators. While they have good internal volume, they often lack the geometric complexity to benefit from this technique.

finding a leaf node that terminates the ray. On the other hand, deferred intersection follows standard ray order and terminates the ray shortly after finding the leaf node that would normally have terminated the ray. Either the deferred intersection list fills up and is flushed (we use a rather small list of size 4 for the *Big Buck Bunny* work), a volumetric occluder is found, or a clear node is encountered triggering an aggressive flush (Section 4.4).

4.8.5 Additional Concerns Regarding Performance

Additionally, previously mentioned factors, particularly the shortcomings of this study (Section 4.6), may color our results in an overly positive or negative light.

The following issues suggest that volumetric occluders could in fact pose a greater benefit moving forward than what has been explicitly shown in this work. Fur shaders are often used to procedurally generate geometry which is used to make objects look furry both directly and within their shadows. This means that shadow rays must encounter the expensive fur geometry. With our approach, we can defer

the instantiation of fur geometry until we conclude there is no intersection with a volumetric occluder. Only then should we run the fur shader. This means that all shadow rays that originate in the region of the shadow that is covered by a volumetric occluder will no longer need the results of the fur shader, so volumetric occluders are poised to produce a performance boost in this case that is even more pronounced than our current results.

The same argument can be made for objects that are represented as a subdivision surface, such as the tree trunk in Figure 4.26. Normally shadow rays will need to intersect the procedurally subdivided version of the mesh to avoid coarseness in the shadow. However, we can conservatively build the volumetric occluders using the bounding boxes of the surface patches and then use these volumetric occluders to resolve as many shadow ray queries as possible. Only shadow rays near the shadow boundary need to test against the actual subdivided geometry.

However, certain parts of the renderer that we have omitted may actually show volumetric occluders in a less favorable overall light. For example, any unaccounted for expense will only increase the time taken in unacceleratable parts of the rendering, and make any gains from volumetric occluders more marginal with regards to their percentage of overall render time. For example, we do not implement a grass geometry shader (compare behind the trunk in Figures 4.28 and 4.34) nor do we implement a leaf geometry shader. Additionally, any use of subdivision surfaces in objects that must remain open or otherwise incompatible with being a volumetric occluder generator (e.g. very thin or serpentine objects) also counts towards the unacceleratable computations that are not accounted for in this work.

Once the object count in the scene increases, the actual creation of volumetric occluders may become prohibitively expensive since it is a per-object cost. Should this problem arise, we advocate using a demand driven system where kd-trees and volumetric occluders are constructed lazily on an object granularity.

Overall, it is difficult to determine the extent to which these additional concerns affect the conclusions we draw regarding this work. We believe, however, that this list of concerns can serve as a road map for others interested in extending our work.

4.9 Future Work

We will now discuss various directions for future work.

4.9.1 Promising Trends

We believe that the volumetric occluder occurrence rate is an excellent way to determine if volumetric occluders can be fruitfully applied to a test scene. Moving forward, we have reason to believe that the occurrence rate in production scenes will increasingly resemble that of our single object test scenes. As discussed in a SIGGRAPH 2010 talk [CJ10], the computer generated characters from the recent *Avatar* CG movie are represented with incredibly complex polygonal models. These models were also watertight due to the requirements of the physics simulation, making them the perfect candidates for volumetric occluders (humanoid, large contiguous interior, complex polygonal representation). Shadows cast from these models stand to benefit the most from volumetric occluder use and the exact nature of our technique is suited for a movie like *Avatar* where visual artifacts are rarely, if ever, acceptable. Although we do not have access to the scenes from such movie productions, future work should explore applying volumetric occluders to similar models. We suspect that volumetric occluders would do very well in this environment.

There are also other signs that shadow rays will continue to benefit from the use of volumetric occluders. Boulos [Bou10] shows that across 6 different benchmark scenes used in the production of a variety of commercial movies (*Beowulf*, *2012*, *Cloudy with a Chance of Meatballs*, and *Alice in Wonderland*), the overall oc-

clusion rate for shadow rays was 52%. Boulos also relates anecdotally that it is not uncommon for shadow rays to constitute over 90% of all rays traced over the course of rendering a typical scene, meaning shadow rays are very much worth accelerating, at least within the domain of resolving visibility.

4.9.2 Accelerating Shadows from Catmull-Clark Surfaces

One possible extension to this work is to combine volumetric occluders and ray-tracing with rendering Catmull-Clark surfaces. The work necessary to refine a Catmull-Clark surface is often quite high even though in many places in the shadow, such as the fully shadowed umbra, the refinement rate is irrelevant to determining the extent of the region. Volumetric occluders provide one way to take advantage of these observations.

4.9.3 Connection to Level of Detail

Volumetric occluders are similar to level of detail approaches [LRC+03] in that volumetric occluders act as simplified proxies for the original geometry. Although it is outside the scope of this work, future work could look at volumetric occluders in this light. For example, we believe work that compares volumetric occluders to level of detail approaches for shadow ray acceleration would lead to useful cross pollination.

In their current form, volumetric occluders are different from most other level of detail approaches in that they generate final images that are exact rather than approximate. It may be possible to explore the level of detail literature to gain insight into how to relax this condition, for example under the right conditions it may be possible to completely discard the original mesh and use only the volumetric occluders.

4.10 Summary

We now summarize the first half of this dissertation which focused on using volumetric occluders to accelerate shadow rays. In the previous chapters, we covered necessary background and then discussed how volumetric occluders work and why they ought to provide runtime improvement. We explained the building blocks of volumetric occluders, including modified kd-tree traversal and the volumetric occlude cache. We concluded our initial study by using volumetric occluders with a handful of single-object scenes to demonstrate that volumetric occluders provide actual runtime improvement.

In this chapter we studied the effectiveness of volumetric occluders using three scenes from *Big Buck Bunny*. We used a very direct application of our approach with minimal accommodations made for the vast differences between our single object test scenes and the multi-object scenes from a professional CG movie. Even so, we were still able to produce a noticeable speedup in one scene. We found this result to be very encouraging in light of the fact that our approach is being tested under conditions that are as realistic as our research constraints will allow.

Overall we found that volumetric occluders are not yet ready for use in production quality offline rendering. However, there are trends in this domain that indicate that our approach will be increasingly relevant, particularly as the frequency of compatible meshes increases. We look forward to future movie productions in the vein of *Avatar* which will contain the complicated, well behaved meshes that have the most to gain from our approach.

Chapter 5

Background for Final Approach Tessellation

This chapter sets the stage for the second part of this dissertation by providing background on *final approach tessellation*, which is the use of ray differentials to aggressively set the tessellation rate of micropolygon surfaces.

5.1 Overview

Micropolygon rendering is a well understood approach to generating high quality images. It is also a time-tested industry standard, most notably featured in Pixar's RenderMan, a commercial renderer which is recognized as setting the bar in state of the art computer generated movies. RenderMan has been featured in computer generated movies such as *Toy Story*, *Finding Nemo*, and *Cars*. A lesser known fact about RenderMan is that it has also been used in quite a few live action movies such as *Terminator II* and *Jurassic Park* [Ren11], which is a testament to the the high quality of its resulting images. RenderMan is the evolution of the original Reyes system pioneered by Cook et al [CCC87].

One of the key strengths of micropolygon rendering is the ability to render objects at different *resolutions*. Objects rendered in this way can be made to have boundable errors in surface shape and color, avoiding the problems of fixed resolution representations where the underlying polygon mesh becomes visible. Resolution, when referring to surfaces, represents the amount of times a surface has been *refined* so that the errors in the surface are bounded. The amount of refinement required varies, for example, based on the distance the surface is to the camera. Distant objects look good with little refinement while nearby objects require more refinement to avoid appearing faceted. The task for a micropolygon renderer is to produce high quality images by finding the level of refinement that is "just right" for each surface in the scene. Too much refinement is a waste of computation and memory while too little refinement results in visual artifacts. Finding a barely sufficient level is the key to both high quality and high performance rendering.

A standard scene representation consisting of triangles is not sufficient for micropolygon renderers. Instead, scenes specified with objects as *higher-order surfaces* can leverage the micropolygon refinement process, particularly the guarantee that surfaces will appear smooth and unfaceted due to using the appropriate amount of refinement. In this work, we use Catmull-Clark subdivision surfaces [CC78] as our higher-order representation for modeling the scene objects. Catmull-Clark surfaces are popular and used in various digital modeling programs due to their robustness and flexibility, which we will describe later. Once a micropolygon renderer determines the required resolution for a higher-order surface, it converts the surface into a grid of polygons where each polygon is about one pixel in size (others have also used one-half pixel polygons [BFM10]). These resulting polygons are called *micropolygons* and the grid that they form is called a *micropolygon grid*, or grid for short. We refer to the density of the triangles in the resulting grid as the *tessellation rate*.

5.2 Motivation for Our Work

To make the conversion of higher-order surfaces to micropolygon grids efficient, the micropolygon renderer must compute an appropriate tessellation rate. If the tessellation rate is too low, geometry errors will be visible, usually in the form of faceting. If the rate is too high, the resulting surface will look good but will require extraneous computation. Our work will focus on how to improve existing methods for finding an appropriate rate. We believe that current approaches for finding the tessellation rate are too conservative, meaning their tessellation rates are higher than necessary which results in extraneous computation.

We propose a new approach to finding a tessellation rate that is more aggressive, which we call *final approach tessellation*. We will describe our approach in detail in Chapter 6. The remainder of this chapter is devoted to explaining the background necessary to understand final approach tessellation.

5.3 Background

Final approach tessellation is built upon a number of components, in particular Reyes, ray differentials, and Catmull-Clark surfaces. We will now describe these components in detail.

5.4 Reyes Quality Threshold

Reyes is the rendering system that first pioneered micropolygon rendering [CCC87]. Reyes features higher order surfaces whose tessellation rates were determined by projecting the bounding box of the surface into screen coordinates and deciding a rate that would ensure each resulting micropolygon would be "small enough."

We will refer to the specific size of the micropolygons that constitutes being "small enough" as the *Reyes quality threshold*. The original Reyes implementation

advocates a micropolygon size of about a single pixel in area (once projected onto the screen) [CCC87], although as reported in Burns et al. [BFM10] other implementations use thresholds as small as one-half pixel in area. One of the key reasons why micropolygon renderers produce high quality images is because the Reyes quality threshold ensures that the objects in the scene are rendered with bounded error in their shape and color. Errors are constrained to within the size and extent of a single pixel. We hold our renderer to the Reyes quality threshold throughout our work in Chapter 6.

The three fundamental operations in a Reyes system are bound, split, and dice, which operate on the higher order surfaces within the scene specification. We refer to these three fundamental operations as *Reyes operations*. *Bound* is used to obtain the bounding box of a surface. *Split* is used to turn a surface into two or more sub-surfaces which themselves should either be splittable or diceable. Finally, *dice* is used to convert a surface into a micropolygon grid which can then be rasterized. With this interface, all surfaces in the scene are either directly diceable (i.e. can become a micropolygon grid) or are diceable after some number of splits. We have implemented this interface in our work.

After micropolygon grids are created, visibility is resolved through standard rasterization of the micropolygons into a color and depth buffer. The performance of this final phase can be improved by using a bucketed approach to reduce the size of the working set [CCC87].

5.5 Shortcomings of Reyes

The original Reyes system is based on rasterization, which is great for performance but runs into stumbling blocks in image quality. Rasterization is limited in that object tessellation rate is determined solely by distance from the center of projection.

For example, consider a far away object that is visible through a magnifying

lens. When Reyes determines the tessellation rate of this object, only its distance from the camera is considered, which would ignore the presence of the lens and the magnification effect on the object. The object would appear at a lower resolution than the one warranted by the lens, and visual artifacts would result. The problem becomes trickier when one considers that this far away object could be visible in the same scene through multiple optic paths, each one requiring a different tessellation rate to meet the Reyes threshold.

The Reyes approach to determining the tessellation rate works quite well for directly visible objects (recall that these objects are in a direct line of sight with the camera) but it runs into serious issues for objects visible via reflection or refraction. One solution to this problem is to keep track of the pixel size along every optical path that matters during rendering, which brings us to ray differentials.

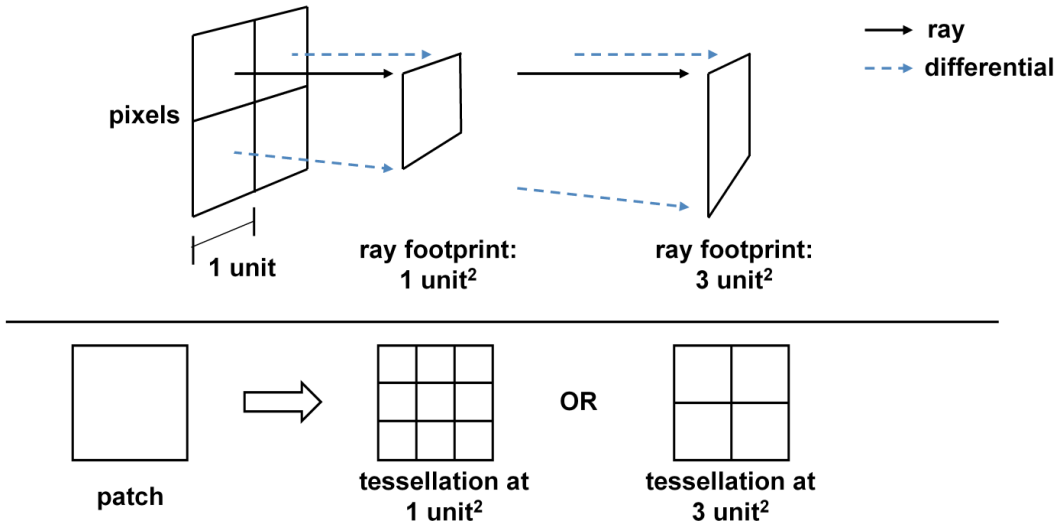


Figure 5.1: Ray differentials track the change in ray origin and ray direction between a given ray and its neighbors. Ray differentials are used to determine the size of a footprint which will be used to set the tessellation rate.

5.6 Ray Differentials

In contrast to rasterization, ray tracing allows for the independent tracking of each visibility query in the scene. Neighboring visibility queries can have completely different parameters, such as the world space size of a pixel, while the bulk processing constraint of rasterization means that visibility queries within a raster have to share parameters (recall that these visibility queries are fixed within the view frustum and are hence constrained). Specifically, ray tracing allows for the independent tracking of the pixel size associated with each reflection and refraction ray used over the course of rendering.

Ray differentials are a compact and efficient bookkeeping system for annotating each ray with the local change in ray direction and origin between neighboring rays in screen space, thereby tracking the changing world space size of a pixel as a ray travels through the scene (Figure 5.1). Ray differentials can be propagated from parent rays to child rays when a reflection or refraction event occurs by using a simple set of formulae based on basic calculus [Ige99].

Previous approaches determine surface tessellation rate using ad hoc approaches such as the distance the ray has traveled [BBLW07] or a combination of heuristics [MTF03] and cannot detect that the optical path has transitioned from decreasing focus to increasing focus, or vice versa. As a result, the object visible by way of complicated optic paths may be insufficiently tessellated, or overly tessellated. Ray differentials address this problem by tracking the world space size of a projected pixel along the entirety of an optic path.

5.6.1 Beam Tracing

Ray tracing with ray differentials is in many ways analogous to beam tracing [HH84], where instead of formulating a renderer's visibility queries as point samples, they are formulated as area samples, called beams. Beams are typically represented as a

geometric frustum that describes the exact extent of a region in screen space as it is projected into the scene. Beam tracing's strength is that its results are exact and do not require sampling until convergence. However, it can become prohibitively expensive due to excessive splitting of beams as they encounter objects in the scene.

Ray differentials provide a compromise. Ray differentials are not as expensive as beams because they do not require beam splitting, however the drawback is that many rays may need to be traced before an image reaches convergence. A renderer that uses rays and ray differentials is still based on point samples and comes with all of the limitations therein.

Additionally, a ray differential is only a first order approximation of the true shape of a projected pixel, so it is correct only up to first order [Ige99]. A beam is a full, exact representation of the projected pixel and has no error. However, the ray differential still captures the necessary information of the beam in the common case, such as when neighbor rays are converging or diverging. As such, effects such as reflection and refraction can be fully supported up to first order. Igehy presents a compelling proof of concept for using ray differentials to do proper texture filtering under these optical effects [Ige99].

Recent work by Overbeck et al. demonstrates that beam tracing can be made to perform well [ORM07]. Fast beam tracing is complementary to our approach and could be used as a replacement for our use of ray differentials.

For this work, we ultimately choose to integrate ray differentials with Reyes with the understanding that they represent a reasonable compromise between quality and performance, as shown by previous work which we now discuss.

5.7 Ray Tracing in Reyes

Ray differentials integrated into Reyes provides a mechanism for determining micropolygon tessellation rate under complex optical effects, such as reflection and

refraction. Such a system has already been used in Pixar's own extensions to RenderMan. Citing a difficulty in achieving certain rendering effects using only scan-line rendering, Christensen et al. [CLF+03] employ ray tracing and ray differentials to generate effects such as color bleeding, soft shadows from large area lights, and multi-bounce reflections. Our work follows in their footsteps, using ray differentials to determine tessellation rates for scene objects encountered by area lights.

In follow-up work, Christensen et al. [CFLB06] extend their approach to a movie production which required convincing images of cars. Visual effects that need to be handled include one-bounce and inter-reflection induced by car paint. This work goes into detail over the various requirements and implementation concerns when implementing a ray tracer within a micropolygon renderer.

We follow the overall system design of Christensen et al. by using ray tracing and ray differentials within a Reyes system. Specifically, our work is a proposal to improve the tessellation mechanism within such a system.

5.8 Multiresolution Surfaces

Now that we have described the overall layout of the renderer, we move on to the multiresolution surfaces that are given as input to the renderer. We will use the term *multiresolution surface* to refer to a surface representation that can represent an object at different tessellation rates.

There are two key advantages of multiresolution surfaces in a Reyes framework. When a detailed representation is not needed, a lower resolution version of the surface can be used, saving computation and storage. When a detailed representation is needed, the higher cost representation can be used to ensure the resulting image has high quality. Improving multiresolution surfaces is thus perfectly in line with our goal of improving high quality, high performance rendering.

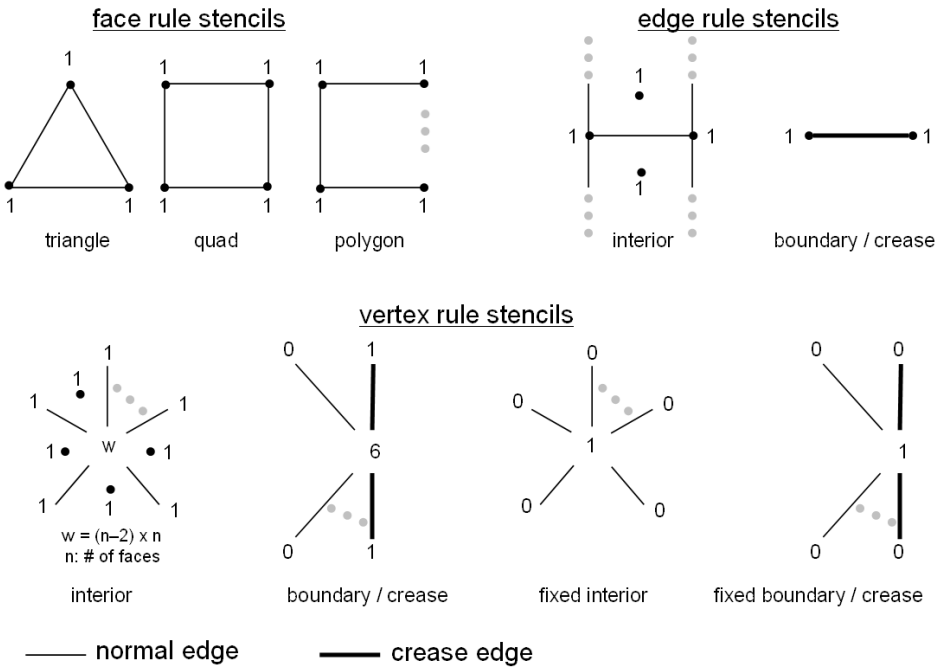


Figure 5.2: Catmull-Clark iteration stencils.

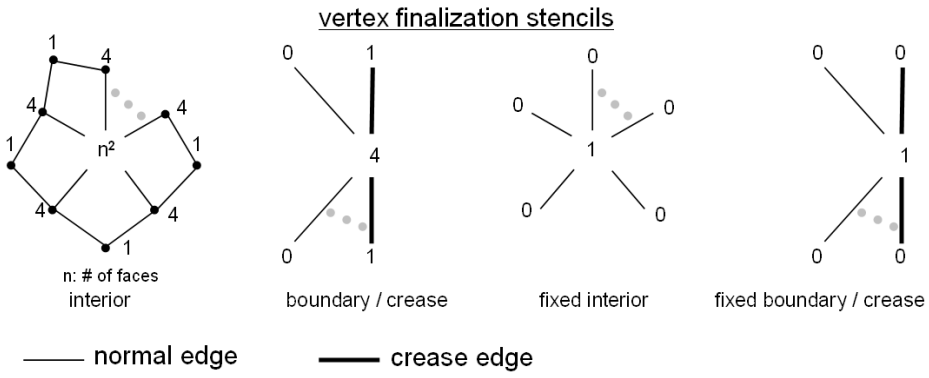


Figure 5.3: Catmull-Clark finalization stencils.

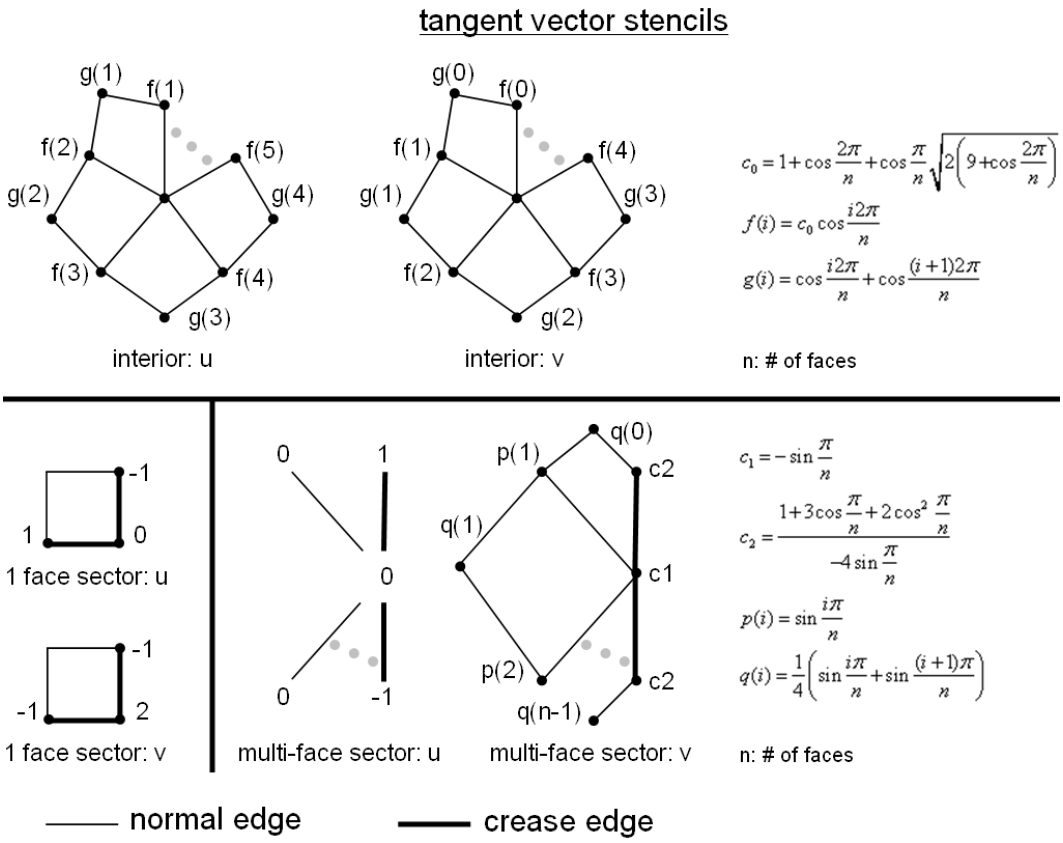


Figure 5.4: Catmull-Clark tangent vector stencils. The vertex normal is computed as the cross product of both tangent vectors.

5.8.1 Catmull-Clark Surfaces

Catmull-Clark surfaces [CC78] are the particular type of multiresolution surface that we use in our work. Specifically, they are a type of subdivision surface that use a control mesh composed of quadrilaterals, which is a very natural modeling primitive.

A subdivision surface consists of a *control mesh*, or *base mesh*, and a set of subdivision rules. The actual 3D surface is the infinite set of points generated by iteratively applying the subdivision rules an infinite number of times on the base mesh; this set of points is known as the *limit surface*. There is a continuum of intermediate meshes between the base mesh and the limit surface and the continuum can be used to represent the surface at different resolutions. In effect, Catmull-Clark surfaces are a type of multiresolution surface.

Catmull-Clark surfaces are also C2 continuous almost everywhere, meaning smoothness in surface properties like the surface normal. C2 continuity means that optical effects such as reflection and refraction will appear continuous on a Catmull-Clark surface. The only places where this continuity does not hold is along *boundaries* and at a finite number of points over the surface known as *extraordinary points*. Boundaries are areas of the base mesh where a face does not have a neighbor and areas where the modeler has asked for a discontinuous normal. Extraordinary points correspond to locations in the original base mesh where a vertex has a valence that is not equal to four.

Another advantage of Catmull-Clark surfaces is that they begin with a normal polygonal base mesh. This base mesh can be arbitrary, meaning that it can be created with a variety of tools designed to build and modify polygonal meshes. The low requirements on the base mesh stand in contrast to other higher-order surfaces, such as NURBS, which require a global parametrization.

Catmull-Clark surfaces also serve as a succinct yet precise representation of a complex surface. Using this representation efficiently can lead to economical use

of limited resources such as memory and bandwidth.

Each polygonal face in the base mesh is converted to a Catmull-Clark *subdivision patch*, which includes the original polygonal face and all neighboring faces (the *one-ring*). A *patch* is a self-contained part of the surface that can be refined independently from all other patches. *Refinement* is the process of turning patches into increasingly smaller sub-patches. Each refinement step generates more points on the limit surface of the Catmull-Clark surface, thereby increasing the fidelity with which the patch represents the surface. Catmull-Clark surfaces are often tessellated by refining the patches up to a certain refinement threshold and then converting the patches into polygons.

Our implementation of Catmull-Clark has support for mesh boundaries, textures, and geometric and texture crease edges [HKD93, BLZ00]. Following the design used by DeRose et al. [DKT98], the Catmull-Clark patches are represented using a generic topology representation (i.e. vertices, edges, faces) in regions of the surface that contain extraordinary points or boundaries and as uniform bicubic B-splines in regions that are completely regular [Pet94]. The latter representation is preferable because 1) it can be split into two smaller bicubics and 2) it can be tessellated anisotropically along each of its two parametric directions. Support for anisotropic tessellation is particularly useful since it mitigates the possibility of overtessellation when refining anisotropically shaped patches. However, the results we present in the next chapter are within the context of a simplified system in which we only use the generic topology representation.

We have gathered all of the non-bicubic refinement rules used in this work in Figure 5.2 (iteration), Figure 5.3 (finalization), and Figure 5.4 (tangent vectors).

5.8.2 Other Multiresolution Surfaces

While Catmull-Clark surfaces are used in this work, there are certainly other types of multiresolution surfaces that are or can be used in Reyes. Rather than being based on quadrilateral control meshes, Loop surfaces [Loo87] are subdivision surfaces for triangle control meshes. Progressive meshes [Hop96] present yet another form of multiresolution. New vertices of a progressive mesh are introduced at the location of existing vertices and change position continuously during the refinement step, enabling support for fractional refinement. The end result is a geomorphing of one shape to the next, in contrast to the discrete transitions of subdivision surface refinement. Even more types of multiresolution surfaces are discussed in the level-of-detail literature [LRC+03].

5.8.3 Ray Tracing Subdivision Surfaces

Different approaches to ray tracing of subdivision surfaces have been used in other systems. This work is typically focused on rendering the subdivision surface without necessarily conforming to the Reyes specification.

The ShaoLin system [MTF03] uses a complex adaptive tessellation scheme, with bounding box tests based on projection of the ray onto a test plane. This system relies on additional stitching polygons within gaps to prevent cracks (for our discussion on cracks, see Section 5.9.1), and neither stores nor caches any results. Benthin et al.'s interactive system tessellates on-the-fly (no caching is used) and avoids the complications of adaptive subdivision by always subdividing patches a constant number of times [BWS04]. Similar to the goals of our work, Lai and Cheng study the tessellation rate needed by a Catmull-Clark patches in order to bound the patch error. However, their approach bounds absolute deviation from the limit surface and is neither view nor ray dependent [LC05].

In subsequent work, Benthin et al. present promising results indicating that

a ray-driven renderer for Catmull-Clark surfaces produces a smaller number of refinement steps than using a full-scene tessellation governed only by an edge length criterion [BBLW07]. However, this work selects a ray-specific tessellation from only three levels of refinement over the entire scene and does not discuss how to handle the requirements of reflection and refraction rays.

More tangential to this work is ongoing research in directly intersecting rays with higher-order surfaces. Direct intersection completely sidesteps the creation of a micropolygon grid by intersecting the ray with the mathematical representation of the surface rather than a tessellated representation. Recent interactive work in this domain performs direct ray intersection with NURBS [PSS+06, AGM06] and piecewise quadratic surfaces [SGS06].

5.8.4 Reyes and Rendering Platforms

Another distant area of interest is mapping Reyes onto high performance rendering platforms. Such platforms that have been targeted are the Imagine stream processor [OKTD02], PC clusters [LSZL02], and GPUs [PO08]. The main focus in these works is discovering an effective way to implement Reyes on the target platform. In contrast, we place our focus on algorithmic improvement using commodity CPU hardware.

Fatahalian et al. [FLB+09] explore how to shape a new rasterization pipeline to support micropolygon rendering in future hardware. This work studies the efficiency of rendering effects such as motion blur and object defocus. Their work is more comprehensive than ours from a Reyes features standpoint due to their inclusion of these advanced effects. However, their work is primarily concerned with improving the sampling methods for handling motion blur and object defocus, while we explore improvements in tessellation rate.

Burns et al. [BFM10] address a more related issue of reducing the amount of

shading performed on micropolygon grids in work that also targets future hardware. Rather than invoke the shader for every vertex in micropolygon grid at the time of grid creation, Burns et al. propose doing so on demand, once regions of the grid are determined to be visible, and at a smaller granularity, such as on 2x2 blocks of vertices. They are able to perform on demand shading be splitting the micropolygon grid into two subgrids, a shape grid and a shading grid, which can be at different resolutions. Burns et al. discuss how to ensure high quality shading by instantiating vertices in the shading grid on demand from the parts of the shape grid that pass the rasterization visibility test. How to determine the resolution of the shape grid, which is closely related to our work, is left unspecified.

5.9 Additional Previous Work

This section describes previous work that is not as closely related to our work, although it may be of more general interest.

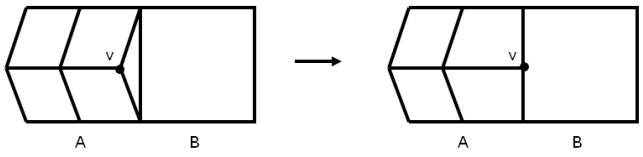


Figure 5.5: (Left) Cracking occurs when two neighboring patches are refined at different tessellation rates. (Right) By moving vertices from the more tessellated side to lie on the edges from the less tessellated side, the crack is repaired. Another solution is to insert additional polygons which fill the crack.

5.9.1 Cracks in Catmull-Clark Surfaces

This work uses adaptive refinement of Catmull-Clark surfaces. Adaptive refinement occurs when different regions of a surface are refined to different granularities, usually because different regions of the surface need different tessellation rates to satisfy the

refinement threshold (such as projected size in screen space). One problem that must be addressed in any adaptive refinement scheme is fixing T-junctions, also called hanging vertices, that result from adaptive refinement. These junctions can produce visible holes, or cracks, in the mesh. We refer to the process of removing these holes as *crack fixing*.

Owens et al. [OKTD02] propose a technique for fixing cracks that uses a tree of necessary patch refinements to produce final patches that satisfy the termination threshold. In a top-down process, edge equations are used to determine which edge splits are needed and which are not, the end result being a set of patches with no cracks. This process removes the T-junctions by picking the set of edges that just satisfies the edge length criteria, preventing vertices introduced by unnecessary splits from showing up in the crack-free output patches.

There is a rich literature on crack fixing that describes various ways to address this problem. These different techniques include binary dicing and stitch geometry [AG00], triangle fans [MH00], breadth-first subdivision with incremental correction [PEO09], and splitting along diagonal lines in parameter space [FFB+09].

In this work we do not address crack fixing, but rather allow cracks to form where they may. This lack of attention does not result in particularly noticeable artifacts, in large part due to our conformance to the Reyes quality threshold. Section 6.14 has more discussion on this aspect of our resulting images.

Chapter 6

Final Approach Tessellation

6.1 Overview

In this chapter we present our work exploring a new approach to determining micropolygon tessellation rate called *final approach tessellation*. *Tessellation* is the process of fitting triangles to a surface in order to represent that surface. Our work belongs to the field of determining object tessellation rate in a way that satisfied two constraints: the tessellation rate should be set high enough so that the visual artifacts associated with mesh faceting are avoided, while at the same time the rate should not be set so high as to incur unnecessary computation which does not improve visual quality. Figure 6.1 illustrates the need to find an appropriate tessellation rate.

Previous work by this researcher (Djeu et al. [DHW+07]) proposes final approach tessellation as a better way to find surface tessellation rates, but does not explore the technique beyond providing an overview of the algorithm. There have been multiple requests from the community to look into this subject in more detail. The focus of this work is to answer the question of whether final approach tessellation is indeed a source of performance improvement.

We first describe how our new approach works and the reasons why it should theoretically provide an improvement over existing techniques. We then discuss empirical results that suggest that the technique is ultimately not worth pursuing. Although it does lower the operation count of generating the images, as expected, it eliminates so few operations (only about 10% to 20%) that the resulting runtime improvement is either nonexistent or negligible.

6.2 Objects and Patches

The scene objects used in this work consist of a collection of *patches*, which are self-contained subdivision surfaces which can all be tessellated at their own rate (Figure 6.2). In this work all patches are Catmull-Clark surfaces [CC78]. The tessellation rate is set for each patch independent of all other patches.

6.3 Aggressive vs Conservative Tessellation

Recall that the Reyes quality threshold (Section 5.4) mandates that micropolygons be about one pixel or one-half pixel in size in screen space. The goal of our approach is to maintain this Reyes quality threshold while performing less tessellation on the multiresolution surfaces (Section 5.8). Satisfying these constraints will ensure that we preserve image quality while improving performance, in line with our overall goal of exploring high quality, high performance rendering.

Although tessellations can refer to the fitting of arbitrary polygons to a surface, in this work we will use tessellation to refer to the process of fitting triangles to a surface. We classify tessellations into two different categories, aggressive and conservative.

Aggressive tessellation is a fitting of triangles to a surface that is just what is necessary to meet the Reyes quality threshold. It consists of as *few* triangles

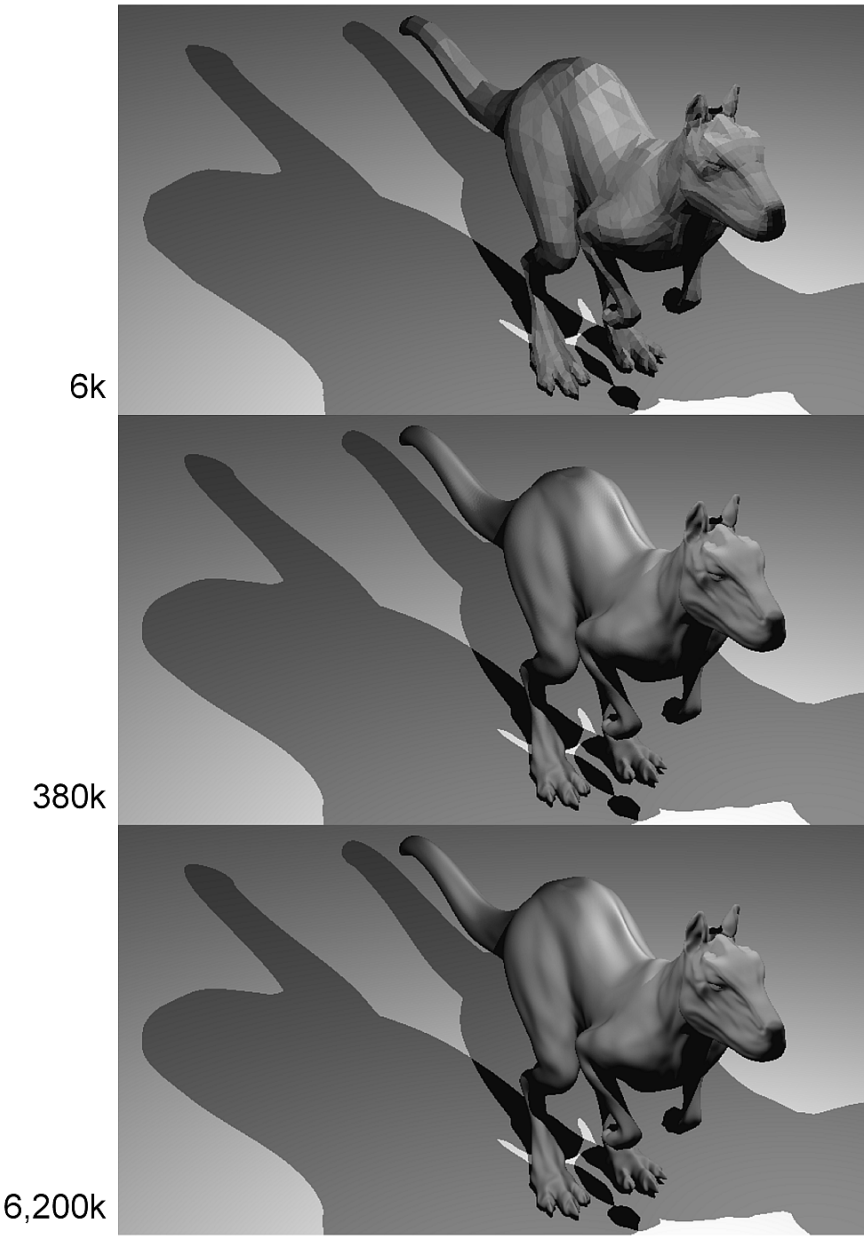


Figure 6.1: Tessellating the killeroo model using only 6k triangles produces noticeable faceting artifacts (top). We wish to tessellate enough so that there are no visual artifacts (middle) while avoiding excessive tessellation which offers no improvement in visual quality (bottom).

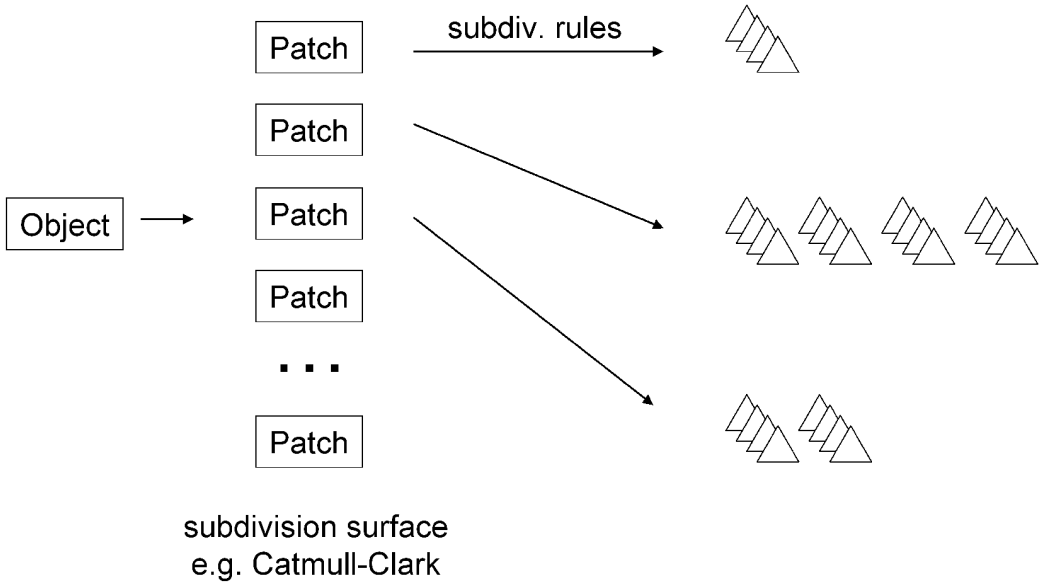


Figure 6.2: An object is composed of a collection of patches. Under our adaptive tessellation scheme, each patch is tessellated at its own independent so that the resulting micropolygons are about one pixel in size.

as possible to meet this threshold, which also means the triangles are as large as possible, or in other words, they are close to the size threshold without exceeding it. We would like to find *tessellation rates* that are as aggressive as possible.

Conservative tessellation is a fitting of triangles to a surface that uses more triangles, or *overtessellates*, to meet the quality threshold. It consists of *more* triangles than necessary and as such the triangles are also smaller than the size threshold.

Ideally we would always want to use an aggressive tessellation if it were cheap to compute, although this is not always the case. Often, determining a conservative tessellation rate is much faster and simpler because it only requires loosely bounding the tessellation rate within a region of space. Computing the exact rate requires a more involved computation. Additionally, when tessellation rate varies too rapidly within a region of space, issues arise regarding how to reconcile two parts of the surface that have been converted into micropolygon grids at different rates. In fact, this

is exactly the surface cracking issue that was previously mentioned (Section 5.9.1).

Final approach tessellation is a type of aggressive tessellation. Our goal in this work is to determine where it lies on the spectrum of tessellation techniques (Figure 6.3).

The tessellation rates in the Reyes system are set using the bounding boxes of the objects in the scene, which we will now discuss.

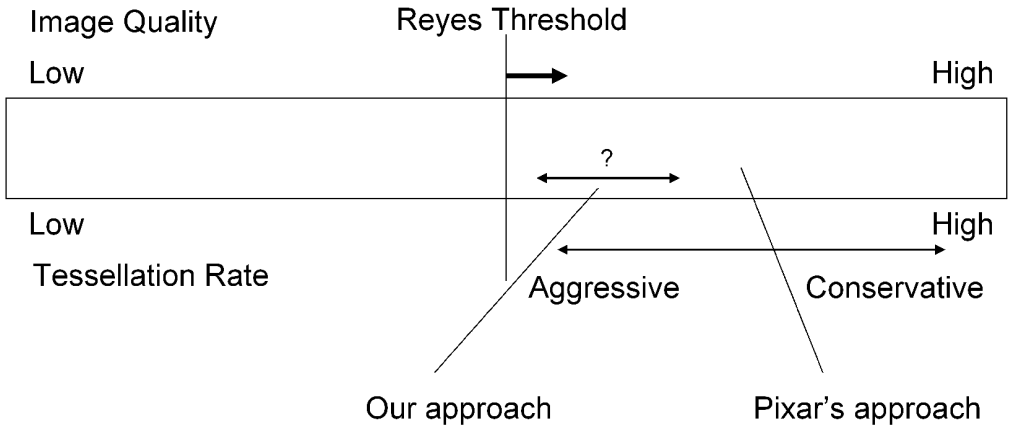


Figure 6.3: Tessellation spectrum. Aggressive and conservative tessellations both satisfy the Reyes quality threshold, although aggressive tessellations produce meshes with a lower tessellation rate. Our approach is designed to produce more aggressive tessellations than the Pixar approach, although it is not clear where our approach lies in the spectrum. In this work we will determine whether there is a meaningful distinction between our approach and the Pixar baseline.

6.4 Bounding Catmull-Clark Patches

A Catmull-Clark patch represents a local surface, or collection of points, which is C2 continuous almost everywhere (Section 5.8.1). A necessary prerequisite to using Catmull-Clark surfaces in Reyes is that they must support a *bound* operation (Section 5.4), which returns a bounding box for the entire surface. This is a hard contract within the Reyes system, the bounding box returned by *bound* must enclose all points that lie on the surface.

Our implementation of this bounding box is to use the bounding box of the 16 control points that form a regular Catmull-Clark patch (if the patch is irregular more control points are involved but the idea is the same). Since all points on the Catmull-Clark surface can be expressed as a linear combination of these control points, we know that the entire surface must lie within the convex hull of the control points, which lies within the bounding box we compute. Our bounds are therefore guaranteed to bound the entire surface.

We note that these patch bounding boxes are loose fitting with respect to the actual points on the surface. The surface typically lies well within the convex hull of the control mesh, and often appears as a smaller version of the center of the control mesh. The exception to this rule is in the degenerate case when the control mesh lies completely within a plane, in which case the bounding box is tightly fitting rather than loose.

The bounds we use for Catmull-Clark patches are not best practice. Various methods have been explored for computing better bounds for subdivision surfaces. Of particular note is decomposing a subdivision surface into the appropriate set of basis functions which are then bounded [Wu05].

6.5 Previous Work

The best known approach for selecting micropolygon tessellation rates that meet the Reyes quality threshold is found in Christensen et al.'s work incorporating ray differentials into the Reyes system [CLF+03, CFLB06]. Their approach sets the tessellation rate the moment the ray enters the bounding box of the multiresolution surface and uses this fixed tessellation rate for the entire distance that the ray remains within the bounding box. This approach guarantees that the Reyes quality threshold is met and comes with nice properties, such as no surface cracks within a patch. We discuss the advantages and disadvantages of this approach in more detail

in Section 6.7.

We consider the work of Christensen et al. to be a direct predecessor to our proposed tessellation scheme. Chapter 5 contains a thorough discussion of other work that is also foundational or related to our work.

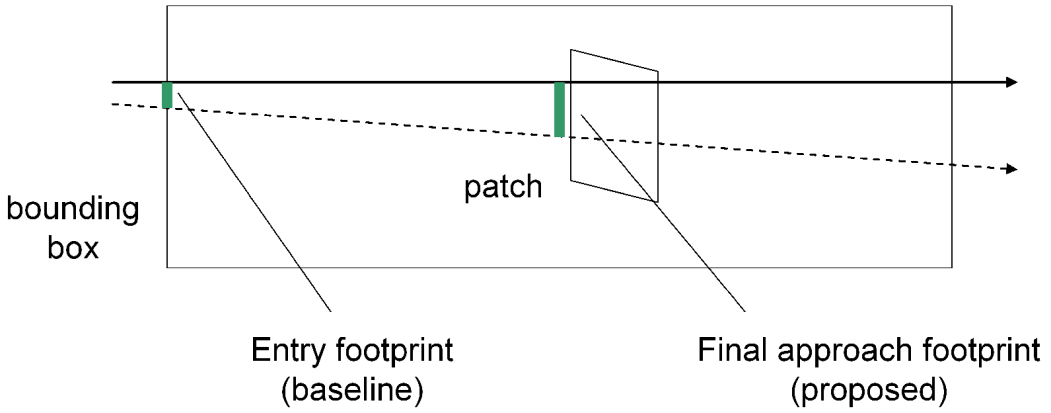


Figure 6.4: The entry footprint is located at the point where the ray enters the patch bounding box. The entry footprint are used in the Pixar baseline approach. The final approach footprint is the footprint located immediately before any part of the patch is encountered. For rays with diverging differentials (2-D example shown) the final approach footprint is larger than the entry footprint. This size difference results in a lower tessellation rate.

6.6 Final Approach Tessellation

While Christensen et al.'s approach has been shown to be sound and effective within Pixar's RenderMan system, we believe we can find a more aggressive tessellation technique. Instead of determining the tessellation rate upon entry into the bounding box (see Figure 6.4), we propose setting the rate based on the size of the ray footprint when the ray is on closest approach to the surface. The ray footprint has had time to grow since it entered the bounding box and a larger footprint means that the tessellation rate can be set lower while still meeting the Reyes quality threshold (as shown in Figure 5.1).

We refer to our proposed scheme as *final approach tessellation* because the tessellation rate is determined right at the point where the ray reaches the surface, i.e. where it is on final approach to the surface. Our scheme should produce a more aggressive tessellation than the previous method used by Christensen et al. We expect that computation and memory usage for generating and storing the tessellation will be lower than that of Christensen et al. but the quality of the rendered images will be just as high because we are still meeting the Reyes quality threshold.

6.7 Taxonomy of Ray Footprints

Ray differentials determine the location of a ray relative to its two neighboring rays in screen space (one in the screen space x direction and the other in the y direction). As such, given a distance from the ray origin the ray and its differentials define a quadrilateral in world space (see Figure 5.1), which we refer to as the *ray footprint*. A ray footprint represents the size of the screen space pixel had it been projected into world space.

If a ray footprint is large, the underlying ray differentials are far apart and indicate that a *coarse tessellation* (i.e. low resolution micropolygon grid) is sufficient for meeting the Reyes quality threshold. In contrast, small ray footprints indicate that the ray differentials are close together and a *fine tessellation* (high resolution micropolygon grid) must be used to meet the quality threshold. If a ray footprint is growing in size as the distance from the ray origin increases, we refer to the ray and its neighbors as *diverging rays*. If on the other hand a ray footprint shrinks as distance from the origin increases, we refer to the ray and its neighbors as *converging rays*.

If we ensure that the triangles in the tessellation are no larger than the ray footprint, then we guarantee that the Reyes quality threshold is met. In our work,

we ensure that each edge in the micropolygon grid is no longer than the shorter side of the ray footprint. However we are left with the task of deciding where along the ray to measure the footprint. The different footprints along the ray produce tessellations with different properties. An illustration of these footprints is shown in Figure 6.4.

6.7.1 Entry Footprint

The *entry footprint* is the ray footprint used in the tessellation technique of Christensen et al. [CFLB06], which is the best known previous approach to determining surface tessellation rate. The entry footprint (left side of Figure 6.4) is the size of the ray differential when the ray enters the bounding box of the patch. Since the entry footprint is the smallest footprint throughout the length of the box, it produces the finest tessellation (i.e. the tessellation with the most micropolygons). The resulting tessellation definitely meets the Reyes quality threshold, but the downside is that the tessellation is conservative.

One significant advantage of this approach, first observed by Christensen et al. [CFLB06], is that cracks cannot occur within a patch. Once the ray enters the bounding box, the entire patch is tessellated at a fixed rate, so it is contiguous by merit of this once-and-for-all tessellation.

- Pro: Meets Reyes quality threshold
- Pro: No cracks within a patch
- Con: Conservative tessellation, it tessellates more than necessary
- Note: Used in previous best approach

6.7.2 Final Approach Footprint

Our proposed technique, called final approach tessellation, uses a footprint that lies between the bounding box entry point and the patch. We can use this *final approach footprint* (center of Figure 6.4) to set the tessellation rate. The final approach footprint is the ray footprint right before the ray encounters any part of the patch, or, to use an air travel analogy, when the ray is on final approach to the patch.

The primary reason to set the tessellation rate using the final approach footprint is because the footprint has had a chance to grow since the ray entered the bounding box. A larger footprint means the tessellation will be coarser (i.e. contain fewer triangles), and coarser tessellations are preferable because they are faster to generate and require less storage. Note that this approach meets the Reyes quality threshold because the resulting tessellation will have triangles that are bounded by the size of the final approach footprint. This footprint size is exactly the size necessary so that the resulting triangles are only one pixel in size in screen space. Using the final approach footprint is thus completely in line with our overarching goal of improving high quality, high performance rendering. We preserve image quality by meeting the Reyes quality threshold, while improving performance by reducing the micropolygon tessellation rate.

Note that using the final approach footprint results in a more aggressive tessellation (Section 6.3) than the use of the entry footprint pioneered by Christensen et al [CFLB06]. In fact, the final approach footprint produces the most aggressive tessellation possible. A tessellation that uses triangles that are any larger in size would fail to meet the Reyes quality threshold.

Although previous work [DHW$^+$07] warns about *tunneling* artifacts when using variations of this approach (see Figure 6.5), in this work final approach tessellation does not have this problem. Similar to tessellating with the entry footprint,

the final approach tessellation is determined for the entire patch based on a single footprint, and the ray is traced *from the point of entry into the bounding box* when testing for micropolygon intersection. The single footprint tessellation criteria means that there are no cracks within a patch, and starting the ray from the bounding box entry point means that it is not possible for the ray to miss the surface if it was supposed to hit it. These two properties guarantee that tunneling will not happen. In contrast, there may still be visual artifacts due to cracks which form between patches (Section 5.9.1), which are also present in entry footprint tessellations.

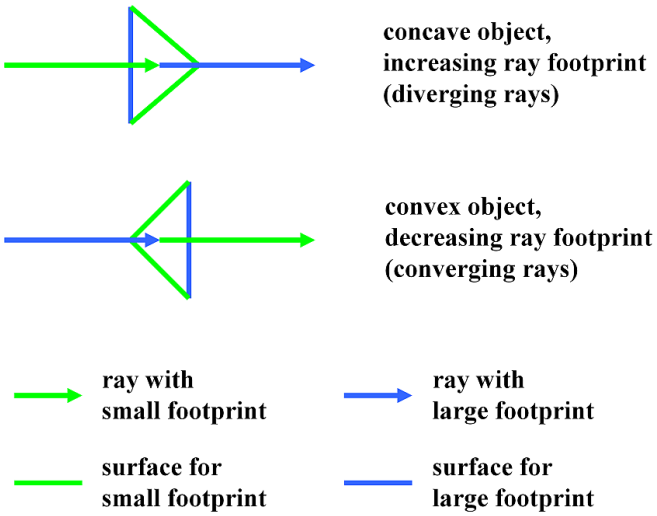


Figure 6.5: Tunneling can occur when surfaces meant for particular ray footprint lie just outside of that footprint's domain.

- Pro: Meets Reyes quality threshold
- Pro: No cracks within a patch
- Pro: Tessellation is as aggressive as possible
- Con: Requires extra computation to determine footprint size
- Note: Our proposed approach

6.8 Rays with Decreasing Footprints

All of the previous discussion applies to rays with increasing footprints (i.e. diverging rays). We have yet to discuss the behavior of rays with decreasing footprints (i.e. converging rays). The situation is very similar, and in fact symmetric along the ray (Figure 6.6). The exit footprint, which is the ray footprint at the point where the ray exits the bounding box, is now used to set the appropriate tessellation rate under the Pixar approach. The final approach footprint is now on the other side of the patch, closer to the exit footprint. It is still defined as the footprint upon final approach to the surface, only now the direction of approach is reversed.

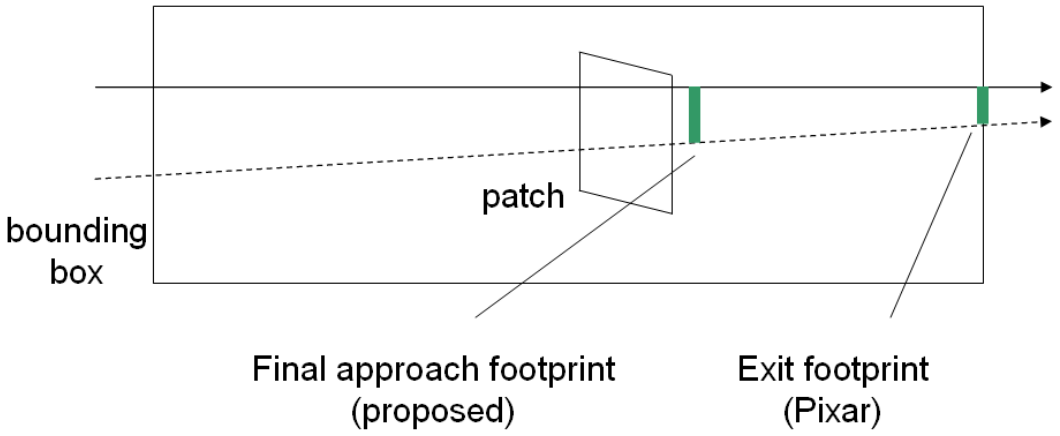


Figure 6.6: For rays with converging differentials, the exit footprint is used by the Reyes system to perform conservative tessellation of the micropolygon surface. Extending this idea to our work, the final approach footprint is now on the far side of the patch, which ensures that the Reyes quality threshold is still met.

6.9 Exact Final Approach Tessellation

Although we do not use exact final approach tessellation in this work, it is implementable by iteratively refining a patch while repeatedly checking a new entry footprint [Kee11]. This method relies on the fact that bounding boxes get tighter for *subpatches*, which are produced when a patch is refined according to the subdivision

refinement rules. A subpatch is itself a patch which can undergo further refinement.

To perform exact final approach tessellation, the tessellation rate is first determined for the initial patch using the patch bounding box and entry footprint. This tessellation rate sets an upper bound on the required tessellation. The patch is then subdivided into subpatches using the Reyes split operation, as long as using this operation does not exceed the previously determined upper bound on tessellation rate, and we record that we have performed one level of subdivision for future reference. We now simply iterate this process on all subpatches. Each subpatch is bounded and the subpatch's entry footprint determines a new tessellation rate. If this tessellation rate, plus the number of split operations already performed, is lower than the old upper bound on tessellation rate, it becomes the new upper bound. Otherwise, the old upper bound is used to tessellate and the iteration stops.

By iteratively refining a patch, we access tighter and tighter bounding boxes and thus get a better value for the size of final approach footprint without ever encountering the patch surface. The final approach footprint found by this algorithm is exact in that it guarantees that the Reyes quality threshold is met, which may not be the case when using an approximate version of the footprint, which we will describe next. We choose to use the approximate version because it is easier to implement and we assert the results of the two approaches will not differ enough to change our final conclusions.

6.10 Approximate Final Approach Tessellation

We use an approximate version of final approach tessellation. Rather than compute the size of the final approach footprint exactly, we use a simple approximation. The Reyes dice operation is used to turn the initial patch into two triangles, which represent the coarsest tessellation of that patch. The ray is intersected with these two triangles and the ray footprint is evaluated at the intersection point. We refer to

the footprint so computed as the *approximate final approach footprint* (Figure 6.7, top).

The error associated with using the approximate final approach footprint is related to the curvature of the patch and may be arbitrarily large. The error is also bidirectional. When the patch is convex, the actual final approach footprint is smaller than the approximate footprint, while the opposite is true when the patch is concave (Figure 6.7, bottom). In practice, patch curvature is reasonable in at least the majority of the patches, so we assert that using the approximate footprint does not noticeably differ from using the actual footprint.

The only extra work involved with computing the final approach footprint, compared to the entry footprint, is performing the intersection of the ray with the coarsest tessellation. Note that the generation of this coarsest tessellation is not extra work because under standard entry footprint tessellation, this tessellation is produced as a by-product when computing the entry footprint tessellation rate.

In the event that the ray does not strike the coarsest version of the patch, we must still assume that there could be a hit (it is possible to strike the surface but miss the initial two triangles). We cannot compute the approximate final approach footprint in this case so we instead fall back to using entry footprint tessellation. This overall approach means we will tessellate no more than entry footprint tessellation.

We use approximate final approach tessellation throughout the rest of this chapter.

6.11 Shading Performance

Our work is completely constrained to the visibility portion of ray tracing, and involves creating the micropolygon grids. It does not concern itself with shading. In classic Reyes, however, the resolution of the grid (the number of vertices it contains) is directly proportional to how much shading is done. Since shading is quite often

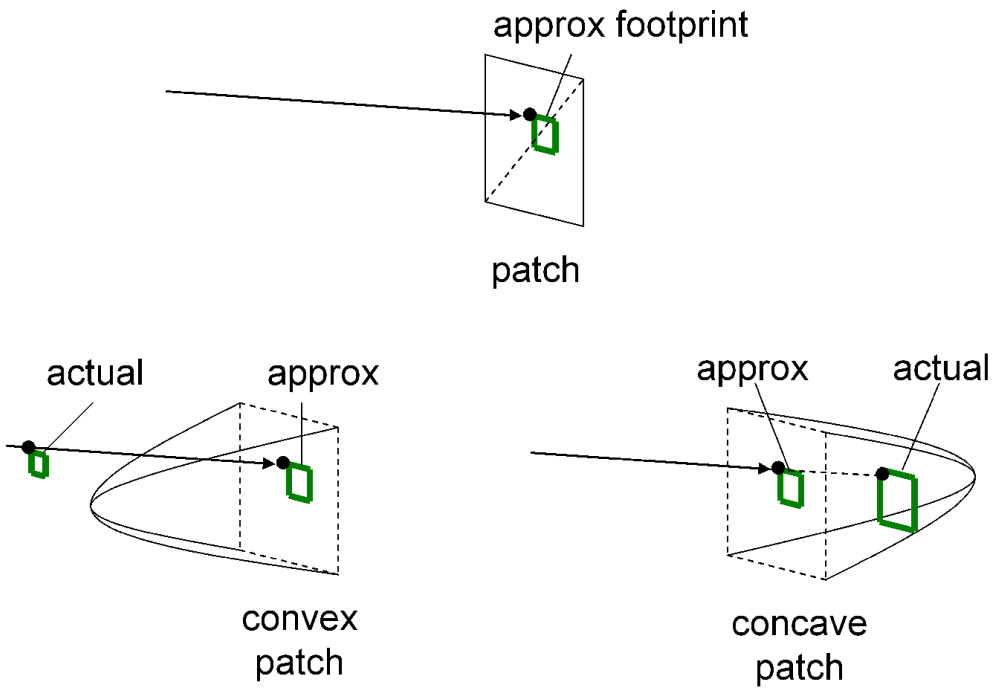


Figure 6.7: The approximate final approach footprint is the ray footprint when the ray intersects the coarsest version of the patch (top). This footprint may be smaller or larger than the actual final approach footprint, depending on the curvature of the patch (bottom).

cited as the bottleneck in rendering performance [CFLB06], the grid resolution directly affects overall renderer performance. By minimizing the grid resolution, we directly address the primary performance bottleneck while preserving the Reyes image quality threshold. We are therefor pursuing with this work our overall goal of high quality, high performance rendering.

More recent work on decoupling shading from the micropolygon grid resolution is found in Burns et al [BFM10]. The authors propose a system that allows for the shading of grids on demand, after visibility has been resolved, rather than at grid creation time. Their system allows for shading to occur roughly once per pixel, and also means that the parts of the grids that are off screen or occluded will not get shaded. The authors also provide more discussion on other approaches that decouple shading from visibility.

If these decoupling approaches are applied to this work, it would decrease the overall performance impact of final approach tessellation because the shading rate becomes independent of the mesh tessellation rate. Our more aggressive tessellation would only affect the time spent during grid creation and visibility resolution.

Under the classic Reyes approach, where shading is performed at grid vertices, the cost of shading is accounted for in the invocation of the Reyes dice operation (Section 5.4), which creates and shades the micropolygon grids. We will measure the frequency of the dice operation in our results.

6.12 Edge Based Tessellations

The entry footprint tessellation that we have discussed is based on the patch bounding box. There is an entirely different class of techniques known as *edge based methods*, which instead use the edges in the object to determine the tessellation rate. Owens et al. [OKTD02] use an edge based method to ensure that neighboring patches agree on the vertex placement along their edges. In addition to meeting

footprint	Reyes qual	op count	op count reduction
entry	yes	normal	baseline
final approach	yes	lower	under study

Table 6.1: *Reyes qual* refers to whether the Reyes quality threshold is met. *op count* is the Reyes operation count (split, dice, bound).

the Reyes quality threshold, no cracks (Section 5.9.1) will form in their tessellation. The DiagSplit approach of Fisher et al. [FFB+09] offers an even more robust set of features. In addition to producing a crack-free tessellation, their framework is able to split patches along diagonal lines in parameter space to produce an even more compact tessellation.

Edge based tessellations seek to achieve the same goals as our work, that is, to produce an image that meets the Reyes quality threshold, but they approach the problem from a different direction, using edges projected into screen space. This decision means that an edge based tessellation may fail to generate critical features, like the sharp curvature of a patch, in the case where projection produces extremely foreshortened edges. In contrast, bound box approaches use distances in world space rather than projected space to guarantee a certain minimal triangle occurrence rate in world space, avoiding the foreshortening problem.

For the time being, we consider edge based tessellations to be outside the scope of our work. However, there is the very real possibility that edge based tessellations are superior to bounding box based tessellations in terms of final quality of the tessellation for the computational expenditure. To our knowledge, the research community has yet to conduct a head-to-head study.

6.13 Results

We will now quantify the effectiveness of final approach tessellation by counting the number of *Reyes operations* needed to generate each scene. Section 5.4 describes

the three Reyes operations (split, dice, and bound) in more detail. The number of Reyes operations will be used as the measurement of how expensive a scene is to render. We will make use of the term *Reyes operation count reduction* to refer to the improvement (i.e. reduction) in operation count when final approach tessellation is used rather than entry footprint tessellation. Our goal is to determine whether the Reyes operation count reduction justifies further pursuit of final approach tessellation.

Our study will make use of the ray footprints mentioned in Section 6.7. We will be examining scenes with primary rays (which use diverging ray differentials) and shadow rays cast towards area light sources (which also use diverging ray differentials), so we will make use of the formulation of final approach tessellation for diverging rays, where the entry footprint is the baseline (Section 6.7). The final approach footprint represents a potential improvement over the entry footprint. The number of Reyes operations is expected to go down due to its use because surfaces will not be tessellated as much. Table 6.1 summarizes the two footprint types that we are used in our study and their effect on the Reyes operation count.

We first discuss our methodology and details of how our rendering framework is implemented. We then discuss a simple proof of concept experiment that shows a promising reduction in Reyes operation count. We delve deeper by studying the effect of camera zoom and light sampling on operation count reduction. We then discuss the effects of using tighter bounding boxes on our test scenes. Finally, we show the runtime improvement of our approach side-by-side with the operation count reduction.

6.13.1 Methodology, Implementation, and Test Scenes

We mainly measure operation counts in this study. All runtimes are gathered on a single core of a 2.83 GHz Xeon X5440 with 16 GB of main memory. We use a ray

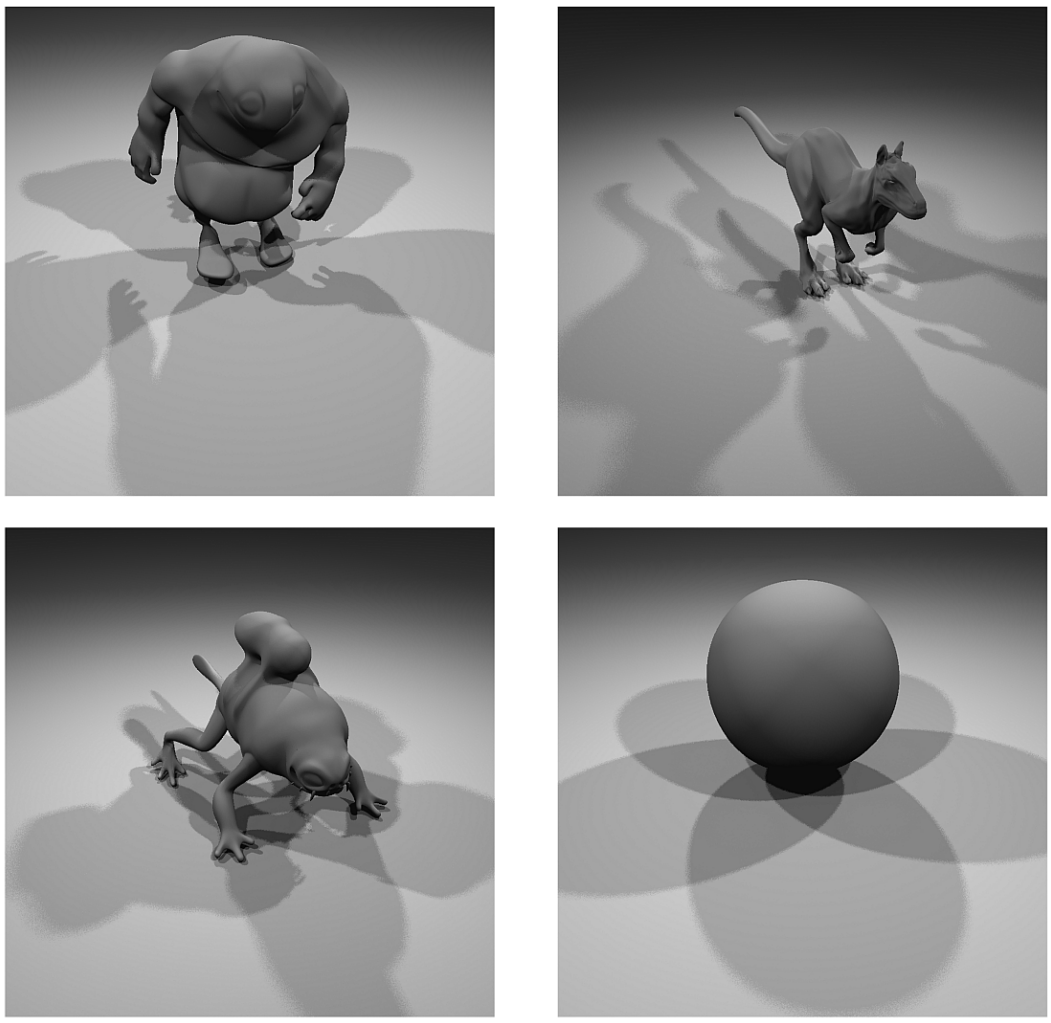


Figure 6.8: Top row: Big Guy, Killeroo. Bottom row: Monster Frog, Sphere.

scene	patches
monster frog	1292
big guy	1450
killeroo	3042
sphere	6

Table 6.2: The four scenes used in our viability study. The patch count represents the complexity of the Catmull-Clark control mesh. The sphere scene is a simple test scene of our own design that is not widely used.

tracer that supports ray differentials (Section 5.6) and Catmull-Clark subdivision (Section 5.8.1).

Meeting the Reyes quality threshold is done through the comparison of edge lengths. The longest edge in the tessellation is constrained to be no longer than the shortest edge in the ray footprint, which ensures that the resulting micropolygons are smaller than one-half pixel in area.

We select our scenes by using three standard test scenes for Catmull-Clark surfaces [PO08]. These test scenes consist of a single object represented as a Catmull-Clark control mesh. Our fourth and final test scene is an extremely simple test scene, a sphere, which we designed to sanity check our approach. Figure 6.8 shows our four test scenes while Table 6.2 lists their complexity.

This study is limited to images rendered with direct visibility (primary rays) and soft shadows (shadow rays) in order to work with the two most common and important components of a rendered image. We study the effect of final approach tessellation for these rays only.

Each scene is rendered at 1024x1024 resolution with one primary ray per pixel. If lighting is used, 5 area lights are procedurally placed at regular intervals along a ring above the object.

6.13.2 Normal Camera and Lighting Settings

Our first test is verifying that final approach tessellation lowers Reyes operation count when the camera and lighting settings are set to default values for our renderer. The camera is placed so that it comfortably frames the object and we use a small but reasonable number of lights, in this case 5 area lights with 4 samples taken per light.

Figure 6.9 shows that for the first three test scenes, Reyes operation count decreases by about 10% to 20% . This result is obtained by comparing the Reyes operation counts when using entry footprint tessellation (the baseline) and final approach tessellation. The difference between these runs is then computed as a percentage reduction when using final approach tessellation rather than the baseline.

Notice that there is an even greater operation count reduction for the Sphere scene. This scene has the most potential for improvement when using final approach tessellation because each of its patches include all 8 control points in the scene, meaning the bounding box for each patch encompasses the entire sphere object. Such large bounding boxes are ideal for our approach because they maximize the size difference between the entry and final approach footprints, which in turn leads to the greatest difference in tessellation rate.

6.13.3 Camera Zoom

We now investigate how final approach tessellation changes under different camera positions. We factor out lighting as a concern by disabling all shadow rays for this set of tests. Figures 6.10 and 6.11 show the image pairs that are used. The first member of each pair is the scene rendered with no lights and normal zoom, while the second member is the scene rendered with no lights and close zoom.

We expect that using close zoom will cause final approach tessellation to have a more pronounced effect. Figure 6.12 shows a ray, its differential, and a patch

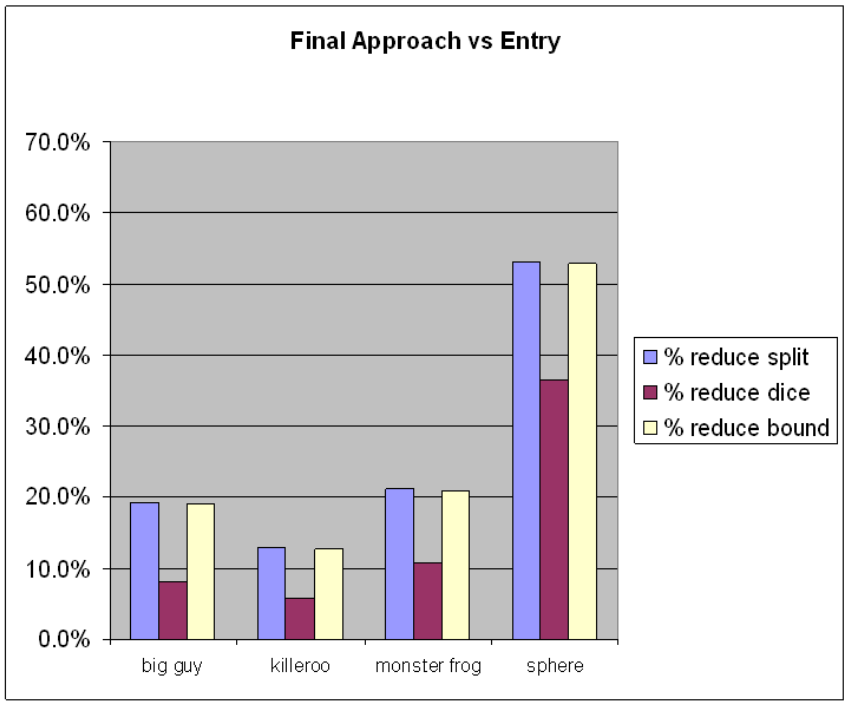


Figure 6.9: We verify that Reyes operation count decreases when final approach tessellation is used in place of entry footprint tessellation. The scenes were rendered under normal zoom and 5 area lights, each with 4 light samples.

that lies along the ray. The ray is labeled with the regions along its extent where a constant number of subdivision iterations is needed to tessellate the patch to meet the Reyes quality threshold. We refer to these regions where a constant number of subdivisions is needed as *subdivision bins*. Subdivision bins with higher values are located closer to the ray origin. The bins themselves are spaced apart according to a power series.

When the camera is far away (Figure 6.12, top), the patch bounding box tends to fall within the same subdivision bin, meaning there is no difference between entry footprint tessellation and final approach tessellation. When the camera is moved closer to the patch (i.e. closer zoom), the bins become thinner and there is a greater chance that the two footprints will fall in different bins (Figure 6.12, bottom). Therefore, we expect that final approach tessellation will have a more pronounced effect when close zoom is used.

We see that this behavior is indeed the case in our test scenes (Figure 6.13). The operation count reduction of final approach tessellation is close to zero under normal zoom, and is in fact zero for the Sphere scene. At this zoom level, all patch bounding boxes in Sphere fall into subdivision bin 9, so there was no difference between the two tessellation techniques. However, under close zoom this was no longer the case, and the Sphere scene exhibited the best improvement under final approach tessellation. All three remaining scenes also improved as a result of close zoom.

We conclude that final approach tessellation performs better for scenes featuring close-ups of Reyes surfaces.

6.13.4 Light Sampling

We now re-enable our 5 area lights and increase their sampling rate from 4 samples per light to 8 (Figure 6.14). Although analytically we expect the percentage reduc-

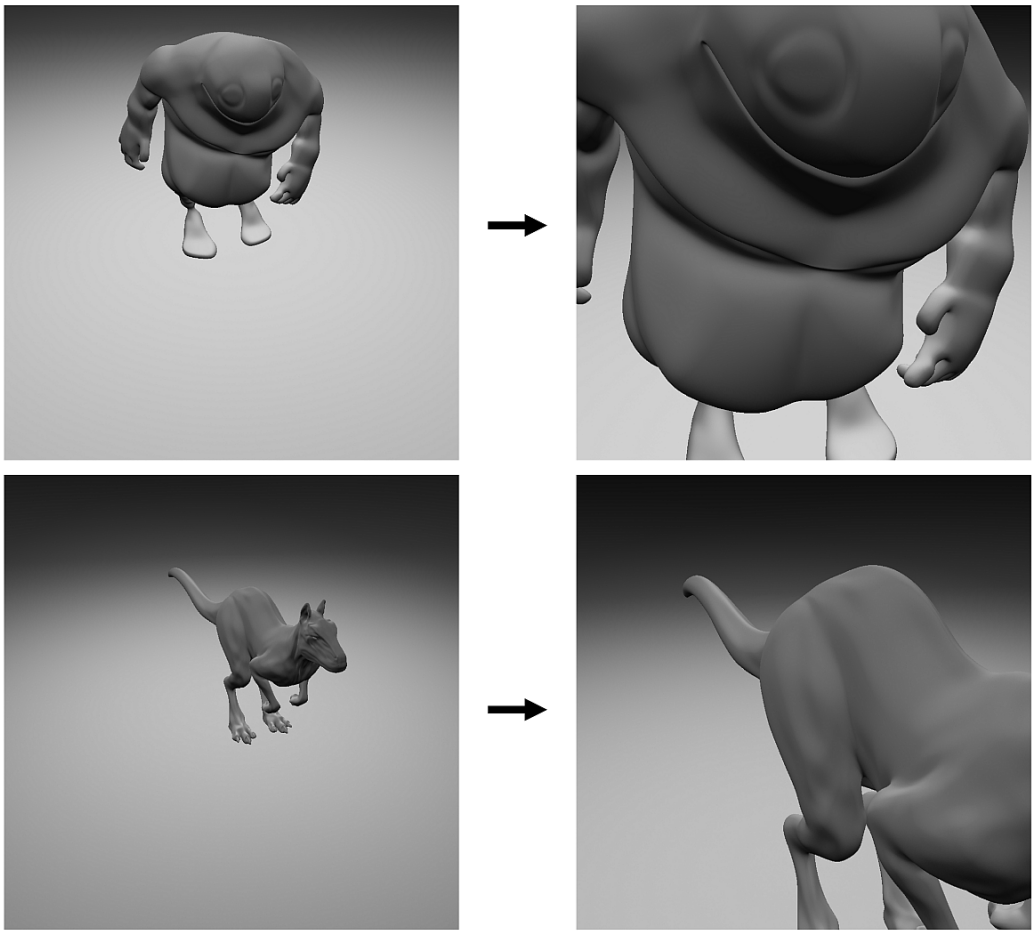


Figure 6.10: Zooming in for the Big Guy and Killeroo scenes. Lighting has been disabled in both the normal zoom and close zoom images.

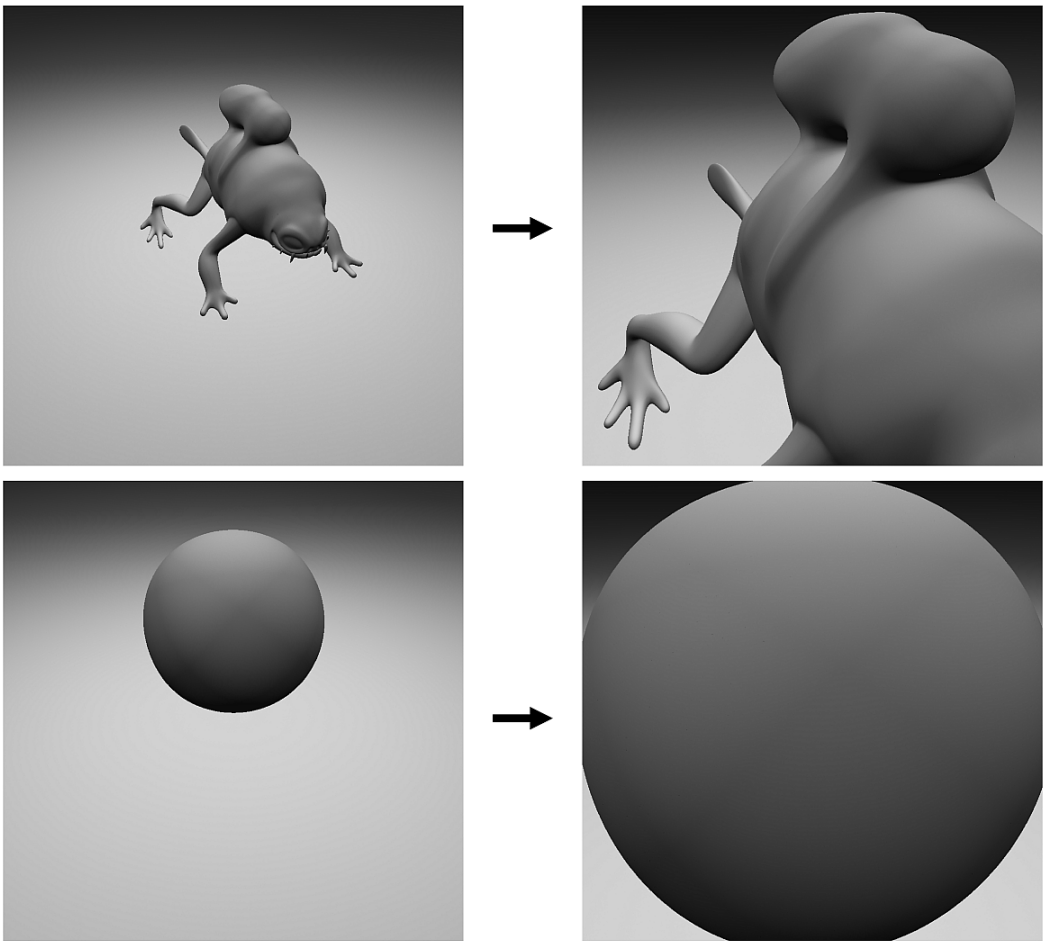


Figure 6.11: Zooming in for the Monster Frog and Sphere scenes. Lighting has been disabled in both the normal zoom and close zoom images.

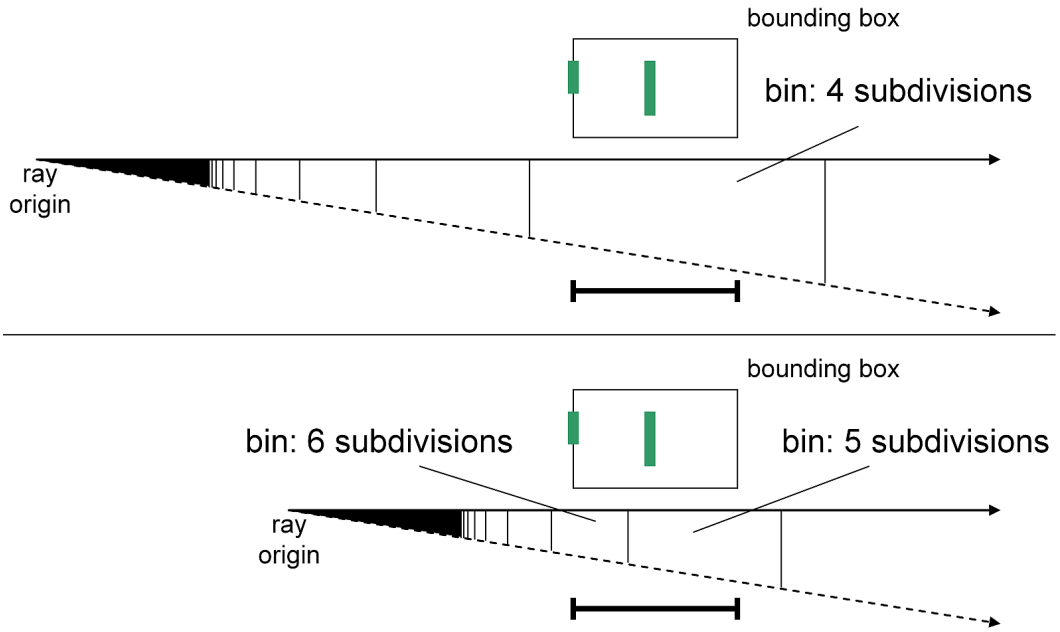


Figure 6.12: When patches are far away from the camera, there is a greater chance that the entry and final approach footprints will fall into the same subdivision bin, leading to no difference in the two approaches. The opposite is true when patches are closer to the camera.

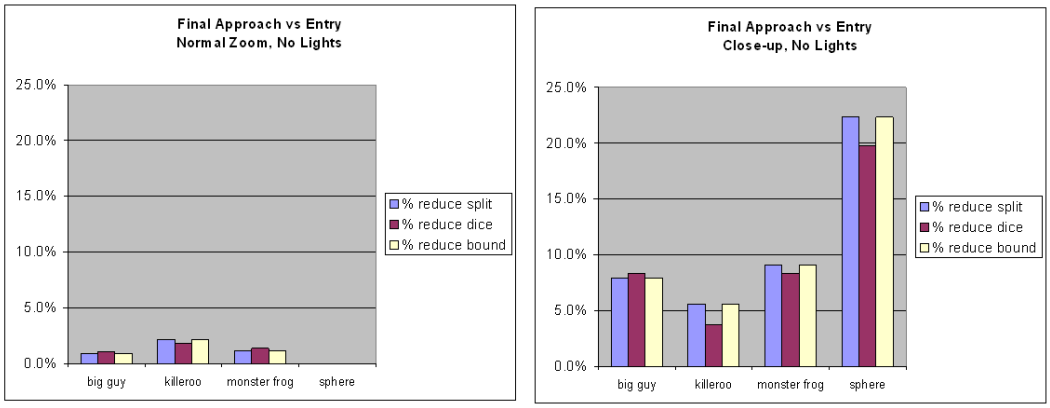


Figure 6.13: Zooming in the camera increases the effectiveness of final approach tessellation. The Sphere scene exhibits no difference between the two tessellation approaches under normal zoom, while exhibiting the most difference under close zoom.

tion in Reyes operations to remain about the same, we notice empirically there is a slight improvement across all four test scenes (Figure 6.15). We conclude that final approach tessellation provides somewhat greater benefit as the light sampling becomes denser. This correlation is nice because high quality movie renderers tend to sample lights more densely.

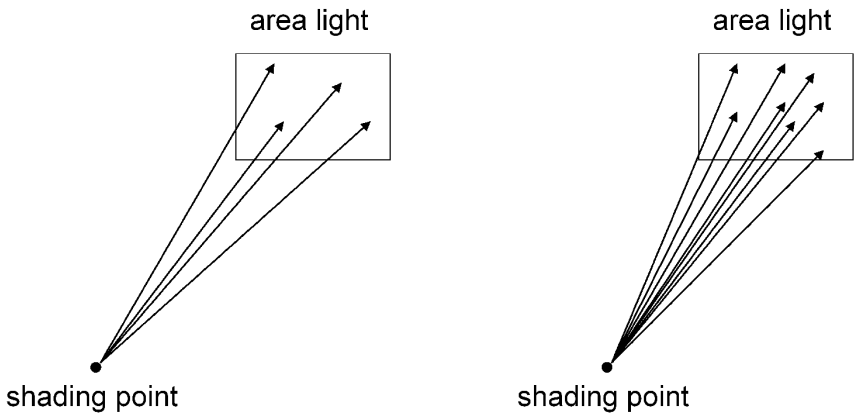


Figure 6.14: Light sampling is increased from 4 samples per area light to 8 samples per area light.

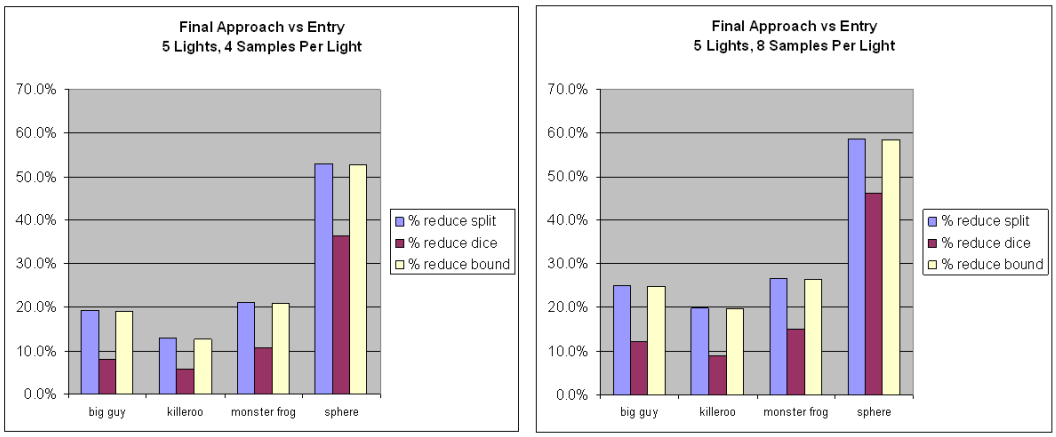


Figure 6.15: Denser light sampling improves the effectiveness of final approach tessellation in reducing Reyes operation count.

6.13.5 Tighter Bounds

We use simple, loose bounds when computing our patch bounding boxes (Section 6.4). We suspect that tighter bounds are used in practice. Our next set of experiments hint at the effectiveness of final approach tessellation when these tighter bounds are used.

We compute our tighter bounds by performing *preemptive subdivision*, which is illustrated in Figure 6.16. When the overall bounds of a patch are required, rather than computing the bounding box of the control mesh, the patch is first subdivided into subpatches using N levels of subdivision. These subpatches are bounded using our original bounding technique, and the subpatch bounds are unioned together to produce a bounding box for the initial patch. Since patch bounds get tighter and tighter as the subpatches get smaller, the final bounding box gets tighter and tighter as N, the levels of subdivision, increases.

The effectiveness of final approach tessellation depends heavily on the size of the patch bounding box. As the bounding box becomes tighter, the size of the entry footprint and final approach footprint converge (Figure 6.17). In the limit, they become identical. We therefore expect that as the bounding boxes become tighter, the differences between the two tessellation methods will also decrease.

Figure 6.18 shows the effect of tightening the bounding box. Tight bounds 0 is the baseline and corresponds to the loose bounds we previously used. Tight bounds 1 corresponds to performing 1 level of preemptive subdivision, and similarly for tight bounds 2 and 3. We notice that the operation count reduction consistently shrinks as the bounds tighten, as expected. In particular, the extremely loose, object-encompassing bounding boxes of the sphere become reasonably tight and the Sphere's operation count reduction decreases to the level of the other test scenes. We also note that although the operation count reduction decreases across all four test scenes, it does not go away entirely and seems to stabilize by tight bounds 3,

meaning there is still some benefit to using final approach tessellation even when bounds are tight.

We conclude that the effectiveness of final approach tessellation, relative to entry footprint tessellation, decreases with the tightness of the patch bounding boxes. However, the reduction in operation count does not disappear entirely.

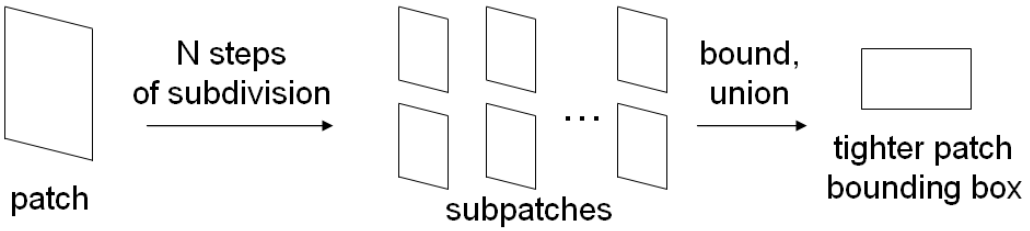


Figure 6.16: Preemptive subdivision of patches produces subpatches with better bounding boxes. These bounding boxes can be unioned together to produce a tighter overall bounding box for the initial patch. The more levels of preemptive subdivision performed, the tighter the resulting bounding box.

6.13.6 Runtimes

We now examine the runtime improvement when using final approach tessellation. Figure 6.19 shows the change in runtime along with the corresponding Reyes operation count reduction when using normal lighting, normal zoom, and our original bounding box strategy (tight bounds 0). We note that although final approach tessellation lowers the operation count in the first three scenes, the corresponding runtimes exhibit no improvement and sometimes increase. We attribute the increase to the additional expense of intersecting the ray with the coarsest patch tessellation to determine the size of the final approach footprint. However, in the Sphere scene, enough Reyes operations have been eliminated where final approach tessellation begins to provide a performance improvement, despite its extra cost.

We conclude that on practical scenes the Reyes operations eliminated by final approach tessellation do not outweigh the extra computation needed for our

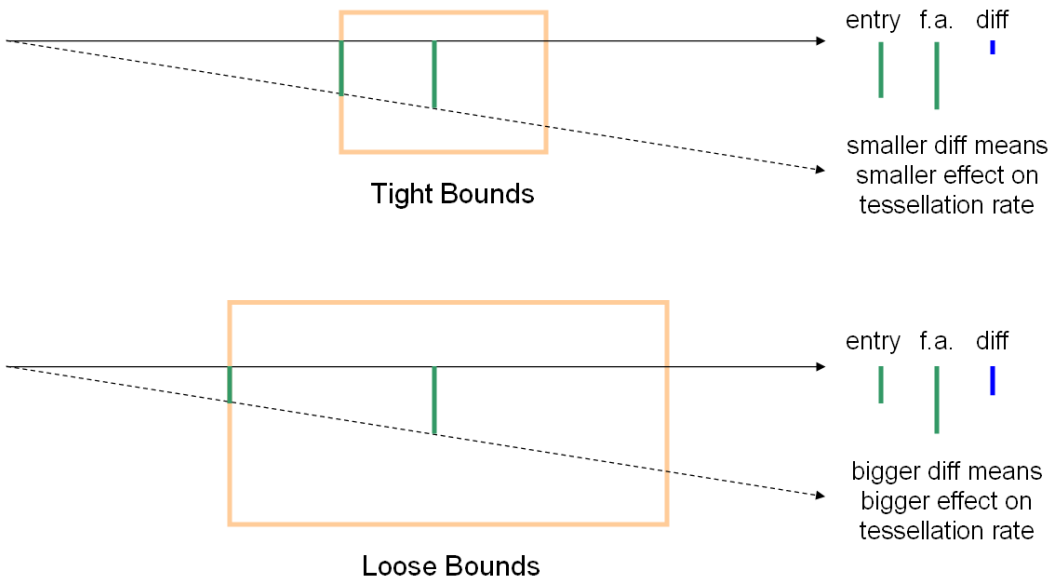


Figure 6.17: Tight bounds produce a smaller difference between the entry and final approach footprints (*f.a.* is short for final approach). This difference (in blue) is the source of improvement in final approach tessellation, so we expect final approach tessellation to exhibit a smaller benefit when tight bounds are used.

approach, and hence final approach tessellation is not a generally useful technique.

6.14 Crack Occurrence Rate

Although cracks are not a central part of our work, we have over the course of this work generated supplementary crack-related results which may be of interest. Cracks (Figure 6.20) are often presented in the literature as an incredibly prevalent problem which must be addressed to produce high quality images [AG00, CFLB06, DHW+11]. We observe that if the Reyes quality threshold is followed through use of the entry footprint, cracks are a rare occurrence, at least within our test scenes.

Over the course of over 4 million pixels rendered, cracks occurred in only 7 pixels, producing an overall error rate of 0.00017% (Figures 6.21, 6.22, 6.23, and 6.24). We acknowledge that cracks are still a concern for offline rendering, where even this error rate may be unacceptable. However, the rate is low enough

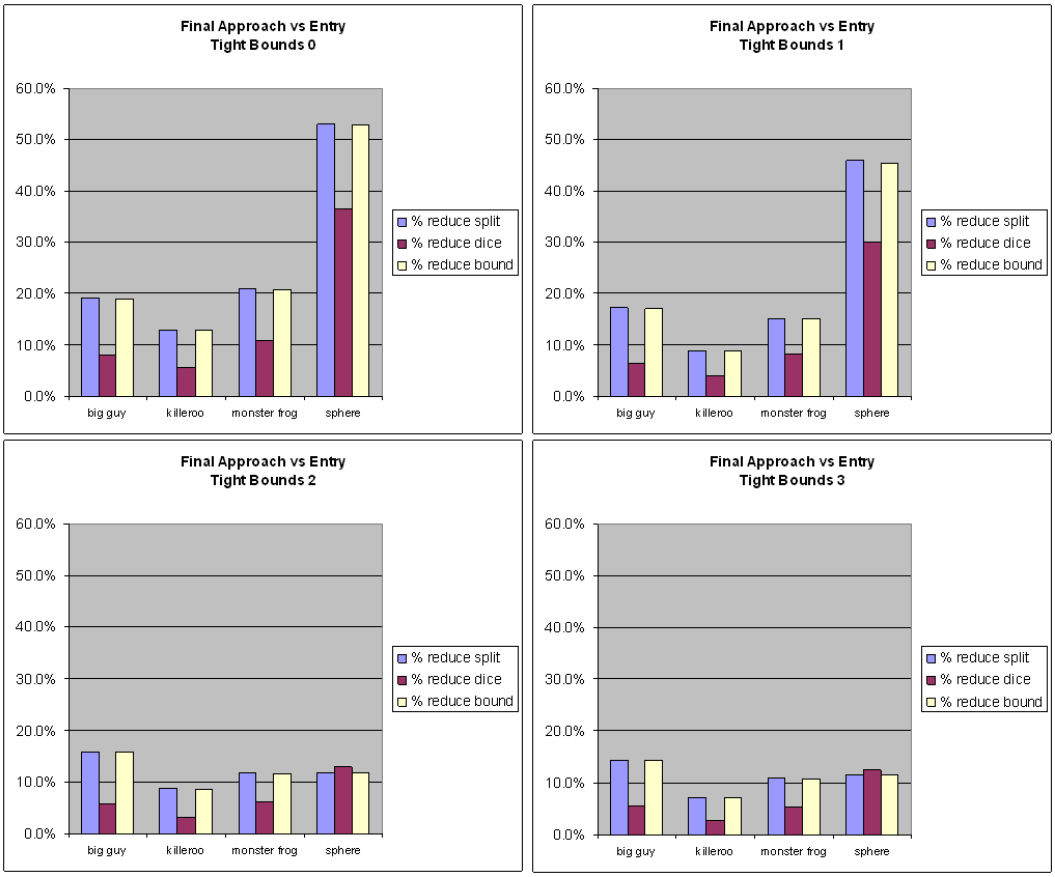
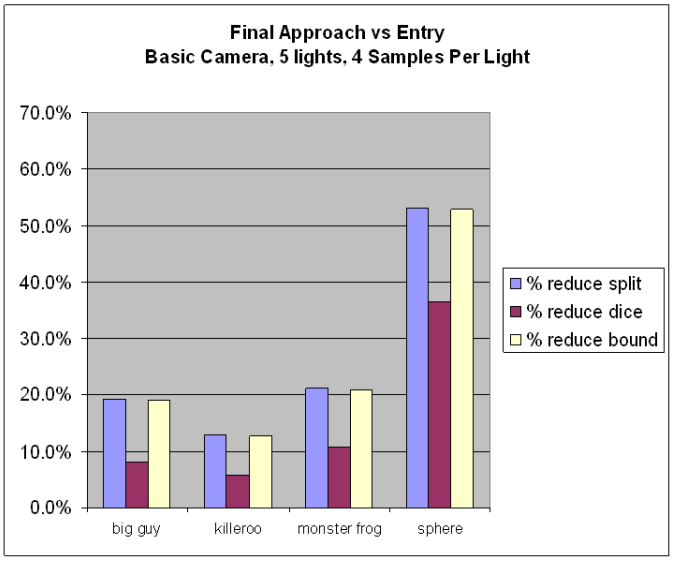


Figure 6.18: The result of performing 0, 1, 2, and 3 levels of preemptive subdivision. Use of tighter bounds results in less improvement in Reyes operation count, as expected. The Sphere scene shows a marked decline in Reyes operation count reduction because using tighter bounds decreases the looseness of its bounding boxes to that of the remaining 3 scenes.



scene	entry (sec)	f.a. (sec)
big guy	155	157
killeroo	92	94
monster frog	136	136
sphere	91	80

Figure 6.19: The Reyes operation count reduction is not significant enough to lower final runtimes. In some cases the runtime increases due to the extra expense of computing the final approach footprint. *f.a.* is short for final approach.

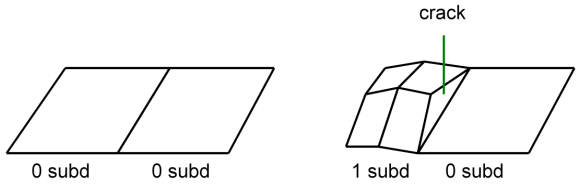


Figure 6.20: Cracks form between neighboring patches that are tessellated at different rates. Here, the left patch is tessellated at a higher rate (1 subdivision step) while the right patch is tessellated at a lower rate (0 subdivision steps). A crack can form on the boundary between these two patches.

where perhaps real-time and interactive variants of Reyes [PO08, ZHR+09] could shoulder the artifact occurrence rate in exchange for better performance and simpler rendering pathways.

In contrast to our bounding box approach, edge based approaches [OKTD02, FFB+09] produce crack-free tessellations while meeting the Reyes quality threshold. These approaches explicitly avoid creating cracks during tessellation by enforcing agreement along the shared edges between patches. Fisher et al. also suggest that the presence of displacement maps may significantly increase the crack occurrence rate. We did not include displacement maps in our study although we agree that the crack occurrence rate in our test scenes may increase.

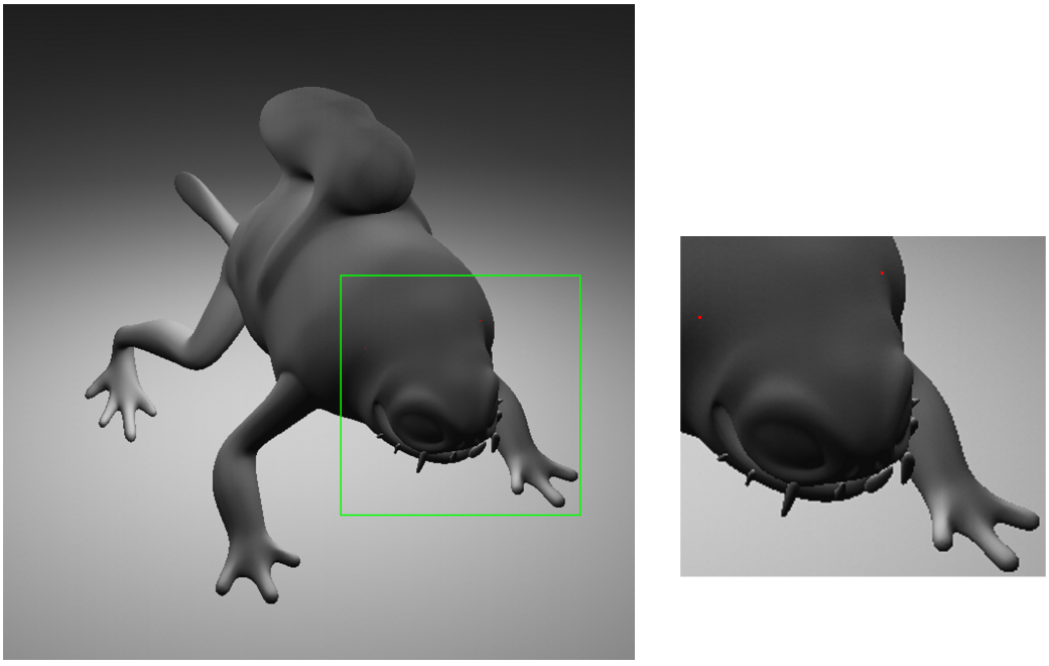


Figure 6.21: Two pixels (shown in red) contain holes due to cracks.

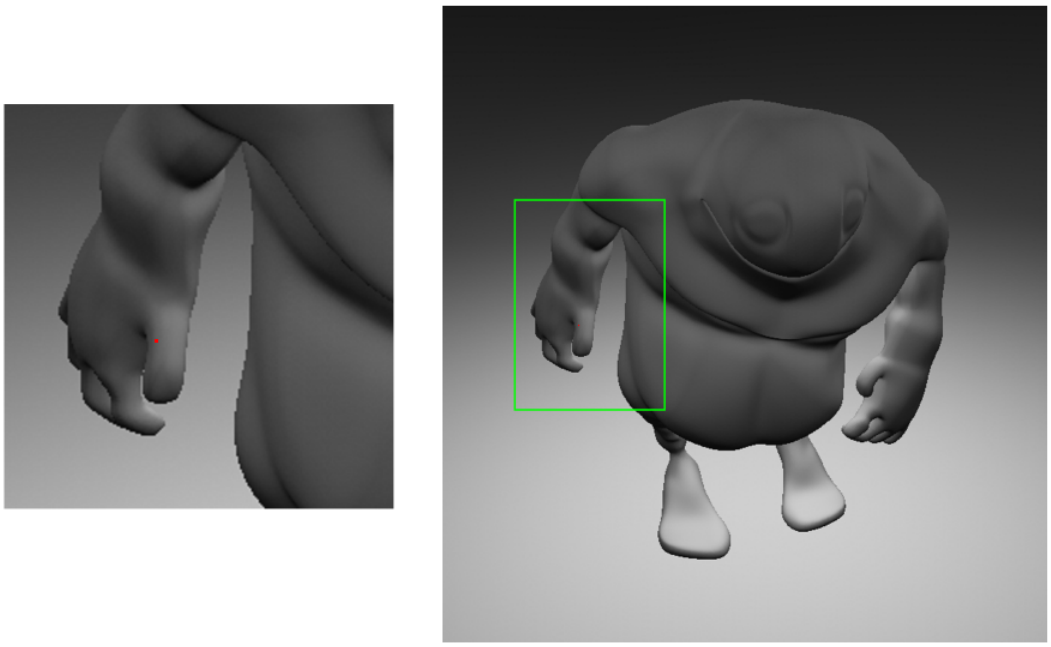


Figure 6.22: A single pixel (shown in red) contains holes due to cracks.

6.15 Future Work

We discovered that the primary expense of generating our images is in the tessellation rates needed by shadow rays, which very frequently encounter patches near their ray origin. As shown in Figure 6.12, subdivision bins are tightly clustered around the origin and the number of subdivisions required for these bins is quite high. One idea for reducing the amount of tessellation required, while producing identical images, is to trace shadow rays backwards, from the light to the surface. The ray differential would then be converging (Section 6.8). If the only patch along the path of the ray is the original occluder, then no extra work is performed. However, if there are multiple occluders, the amount of tessellation required is reduced because occluders close to the light require less tessellation.

For rendering systems that do not support converging ray differentials, a

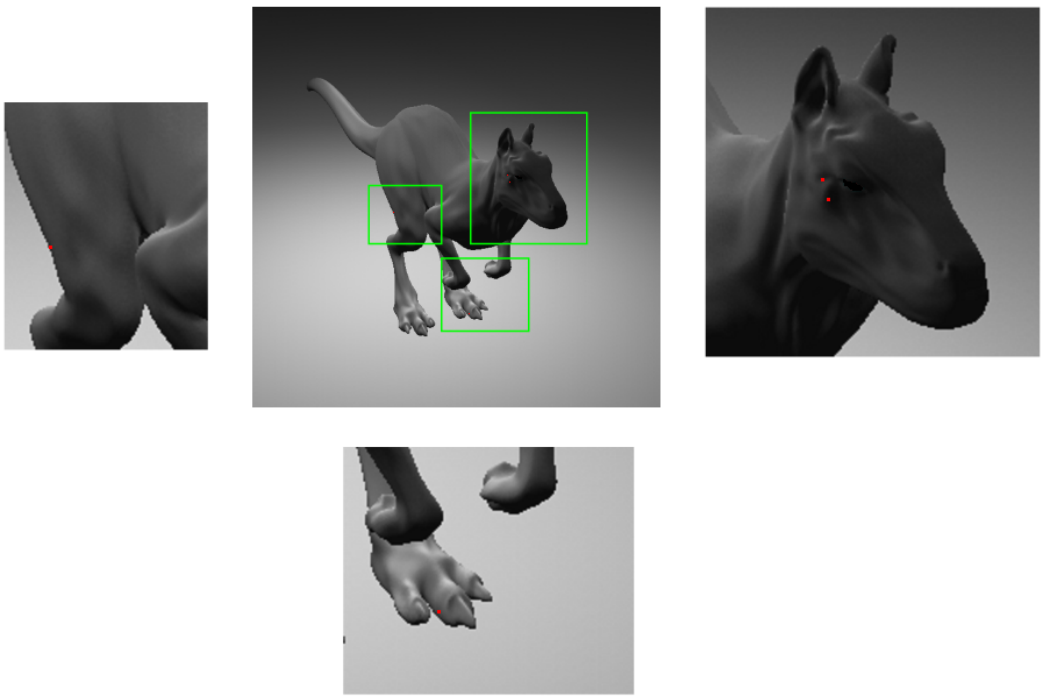


Figure 6.23: Four pixels (shown in red) contain holes due to cracks.

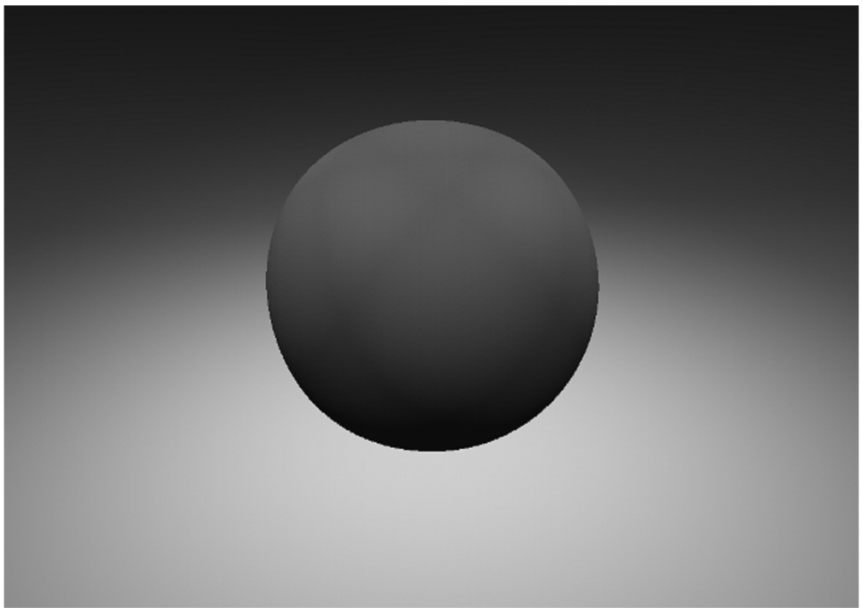


Figure 6.24: No holes are present in the image.

similar effect could be achieved through the use of a deferment list, like the one used in our volumetric occluder work. The deferment list would contain the patches encountered along the extent of the ray and when they were finally needed to resolve a shadow ray query, they would be read from the list in LIFO (last in, first out) order, so that patches closer to the light were tested for intersection first.

6.16 Summary

We propose final approach tessellation, a new method of generating aggressive tessellations while meeting the Reyes quality threshold. We develop a theoretic basis for why final approach tessellation should lower tessellation rates before moving on to our empirical validation.

Our experiments demonstrate that final approach tessellation does reduce the number of Reyes operations needed to render our test scenes. We show that final approach tessellation works better when the camera is close to the object and when light sampling is dense. We also show that its benefits are diminished when tighter bounding boxes are used. Finally, we demonstrate that the runtime improvement is not compelling enough to justify further exploration of this tessellation strategy.

Although our work presents a mostly negative result, we provide closure to an important loose end from previous work that proposes final approach tessellation within the context of a much larger system [SMD+06, DHW+07, GDS+08, DHW+11]. Our work examines final approach tessellation in much greater depth than that afforded by previous work and we ultimately do not recommend further pursuit of this tessellation method, at least from the standpoint of performance improvement. Our conclusion provides an answer to a heretofore open question in the research community.

Chapter 7

Conclusion

7.1 Final Thoughts

This thesis has presented detailed studies on two new approaches to improving high quality, high performance rendering.

Our first study looked at using volumetric occluders for accelerating shadow rays. We first presented two new algorithms for modified kd-tree traversal to allow for the use of volumetric occluders. We followed up with a study of volumetric occluders in very controlled scenes, examining their impact on single object scenes. We found the results there extremely promising, showing a reduction in shadow rendering times by up to 51%. Finally, we concluded our volumetric occluder work with another set of experiments, this time on actual scenes from a CG movie production. We found the extension to actual multi-object scenes was not as promising as expected, with the primary contributors being the low number of volumetric occluder generators.

We believe that our volumetric occluder work will see more use in the future. There are encouraging trends in movie production, particularly with reports of the frequency of occluded shadow rays in other multi-object scenes and the increasing

prevalence of watertight meshes, particularly in complex main character models that will be featured throughout a movie. Our work presents what we believe to be a promising new approach for shadow ray acceleration that will influence the way high quality, high performance renderers are implemented of the future.

Our second study explored finding a more aggressive tessellation rate for micropolygon surfaces using a new technique we developed called final approach tessellation. We described the details of this algorithm and explained the reasons why we believed it would improve performance.

While final approach tessellation held a lot of promise in theory, the empirical evidence strongly indicates that it does not provide an overall practical benefit. We verified that final approach tessellation reduces operation count, as expected, but also that these reductions did not translate into improvements in runtime. Our work serves as an important capstone to a line of previous work that proposed the use of final approach tessellation but did not evaluate its performance with respect to the existing standard for tessellating micropolygon surfaces.

Bibliography

[AG00] Anthony A. Apodaca and Larry Gritz. *Advanced RenderMan: Creating CGI for Motion Pictures*. Morgan Kaufmann, 2000.

[AGM06] Oliver Abert, Markus Geimer, and Stefan Müller. Direct and fast ray tracing of NURBS surfaces. In *IEEE Symposium on Interactive Ray Tracing 2006*, pages 161–168, September 2006.

[AK89] James Arvo and David Kirk. A survey of ray tracing acceleration techniques. In Andrew S. Glassner, editor, *An Introduction to Ray Tracing*, pages 201–262. Academic Press, 1989.

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS '68 (Spring)*, pages 37–45, April 1968.

[BBLW07] Carsten Benthin, Solomon Boulos, J Dylan Lacewell, and Ingo Wald. Packet-based ray tracing of Catmull-Clark subdivision surfaces. Technical Report UUSCI-2007-011, SCI Institute, The University of Utah, July 2007.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[BFM10] Christopher A. Burns, Kayvon Fatahalian, and William R. Mark. A lazy object-space shading architecture with decoupled sampling. In *High Performance Graphics 2010*, pages 19–28, June 2010.

[Ble08] Blender. Blender Foundation, Big Buck Bunny open movie project. `http://www.bigbuckbunny.org`, 2008. Retrieved August 2009.

[BLZ00] Henning Biermann, Adi Levin, and Denis Zorin. Piecewise smooth subdivision surfaces with normal control. In *SIGGRAPH 2000*, pages 113–120, July 2000.

[Bou10] Solomon Boulos. How often do shadow rays hit? *Ray Tracing News*, 23(1), July 2010.

[BWS04] Carsten Benthin, Ingo Wald, and Philipp Slusallek. Interactive ray tracing of free-form surfaces. In *Afrigraph 2004*, pages 99–106, November 2004.

[CC78] Edwin Catmull and Jim Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, November 1978.

[CCC87] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The Reyes image rendering architecture. *SIGGRAPH '87*, pages 95–102, July 1987.

[CFLB06] Per H. Christensen, Julian Fong, David M. Laur, and Dana Batali. Ray tracing for the movie 'Cars'. In *IEEE Symposium on Interactive Ray Tracing 2006*, pages 1–6, September 2006.

[CJ10] Simon Clutterbuck and James Jacobs. A physically based approach to virtual character deformations. In *"Avatar in Depth" Talk Session, SIGGRAPH 2010*, July 2010.

[Cla76] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, October 1976.

[CLF+03] Per H. Christensen, David M. Laur, Julian Fong, Wayne L. Wooten, and Dana Batali. Ray differentials and multiresolution geometry caching for distribution ray tracing in complex scenes. In *Eurographics 2003*, pages 543–552, September 2003.

[DHW+07] Peter Djeu, Warren Hunt, Rui Wang, Ikrima Elhassan, Gordon Stoll, and William R. Mark. Razor: An architecture for dynamic multiresolution ray tracing. Technical Report TR-07-52, Department of Computer Science, The University of Texas at Austin, January 2007.

[DHW+11] Peter Djeu, Warren Hunt, Rui Wang, Ikrima Elhassan, Gordon Stoll, and William R. Mark. Razor: An architecture for dynamic multiresolution ray tracing. *ACM Transactions on Graphics*, 2011. To appear.

[DKT98] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. In *SIGGRAPH '98*, pages 85–94, July 1998.

[DV08] Peter Djeu and Stan Volchenok. Using annotated kd-trees to accelerate shadow ray queries. Technical Report TR-08-36, Department of Computer Science, The University of Texas at Austin, August 2008.

[FFB+09] Matthew Fisher, Kayvon Fatahalian, Solomon Boulos, Kurt Akeley, William R. Mark, and Pat Hanrahan. DiagSplit: Parallel, crack-free, adaptive tessellation for micropolygon rendering. In *SIGGRAPH Asia 2009*, pages 150:1–150:10, December 2009.

[FKN80] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *SIGGRAPH '80*, pages 124–133, July 1980.

[FLB+09] Kayvon Fatahalian, Edward Luong, Solomon Boulos, Kurt Akeley, William R. Mark, and Pat Hanrahan. Data-parallel rasterization of micropolygons with defocus and motion blur. In *High Performance Graphics 2009*, pages 59–68, August 2009.

[FTI86] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, April 1986.

[GDS+08] Venkatraman Govindaraju, Peter Djeu, Karthikeyan Sankaralingam, Mary Vernon, and William R. Mark. Toward a multicore architecture for real-time ray-tracing. In *IEEE/ACM International Symposium on Microarchitecture 2008 (MICRO-41)*, pages 176–187, November 2008.

[Gla84] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[Gla89] Andrew S. Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989.

[GS87] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.

[Hai91] Eric Haines. Efficiency improvements for hierarchy traversal in ray tracing. In James Arvo, editor, *Graphics Gems II*, pages 267–272. Academic Press, 1991.

[Hav00] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[HB02] Vlastimil Havran and Jiri Bittner. On improving kd-trees for ray shooting. *Journal of WSCG*, 10(1):209–216, February 2002.

[HF10] Eric Haines and Marcos Fajardo. Marcos and Arnold. *Ray Tracing News*, 23(1), July 2010.

[HG86] Eric Haines and Donald Greenberg. The Light Buffer: A shadow-testing accelerator. *IEEE Computer Graphics and Applications*, 6(9):6–16, September 1986.

[HH84] Paul Heckbert and Pat Hanrahan. Beam tracing polygonal objects. *SIGGRAPH '84*, pages 119–127, July 1984.

[HKD93] Mark Halstead, Michael Kass, and Tony DeRose. Efficient, fair interpolation using Catmull-Clark surfaces. In *SIGGRAPH '93*, pages 35–44, August 1993.

[HKRS02] Jim Hurley, Alexander Kapustin, Alexander Reshetov, and Alexei Soupikov. Fast ray tracing for modern general purpose CPU. In *GraphiCon 2002*, September 2002.

[Hop96] Hugues Hoppe. Progressive meshes. In *SIGGRAPH '96*, pages 99–108, August 1996.

[Ige99] Homan Igehy. Tracing ray differentials. In *SIGGRAPH '99*, pages 179–186, August 1999.

[Joh08] Gregory S. Johnson. *A Hybrid Real-Time Visible Surface Solution for Rays With a Common Origin and Arbitrary Directions.* Ph.d. thesis, Department of Computer Science, The University of Texas at Austin, December 2008.

[Kaj86] James T. Kajiya. The rendering equation. In *SIGGRAPH '86*, pages 143–150, August 1986.

[Kee11] Sean Keely. Personal communication, April 2011.

[Lan02] Hayden Landis. Production-ready global illumination. In *Course 16 notes, SIGGRAPH 2002*, pages 87–101, July 2002.

[LBBS08] Dylan Lacewell, Brent Burley, Solomon Boulos, and Peter Shirley. Raytracing prefiltered occlusion for aggregate geometry. In *IEEE Symposium on Interactive Ray Tracing 2008*, pages 19–26, August 2008.

[LC05] Shuhua Lai and Fuhua (Frank) Cheng. Adaptive rendering of Catmull-Clark subdivision surfaces. In *Computer Aided Design and Computer Graphics (CAD/CG 2005)*, pages 125–130, December 2005.

[Loo87] Charles T. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, The University of Utah, August 1987.

[LRC+03] David Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2003.

[LSZL02] Oscar Lazzarino, Andrea Sanna, Claudio Zunino, and Fabrizio Lamberti. A PVM-based parallel implementation of the REYES image rendering architecture. In *PVM/MPI 2002*, pages 165–173, September 2002.

[LW93] Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *Compugraphics '93*, pages 145–153, December 1993.

[MB90] David J. MacDonald and Kellogg S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3):153–166, June 1990.

[MH00] Kerstin Müller and Sven Havemann. Subdivision surface tesselation on the fly using a versatile mesh data structure. In *Eurographics 2000*, pages 151–159, September 2000.

[MTF03] Kerstin Müller, Torsten Techmann, and Dieter Fellner. Adaptive ray tracing of subdivision surfaces. In *Eurographics 2003*, pages 553–562, September 2003.

[OKTD02] John D. Owens, Brucek Khailany, Brian Towles, and William J. Dally. Comparing Reyes and OpenGL on a stream architecture. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware 2002*, pages 47–56, September 2002.

[ORM07] Ryan Overbeck, Ravi Ramamoorthi, and William R. Mark. A real-time beam tracer with application to exact soft shadows. In *Eurographics Symposium on Rendering 2007*, pages 85–98, June 2007.

[PEO09] Anjul Patney, Mohamed S. Ebeida, and John D. Owens. Parallel view-dependent tessellation of Catmull-Clark subdivision surfaces. In *High Performance Graphics 2009*, pages 99–108, August 2009.

[Pet94] John W. Peterson. Tessellation of NURB surfaces. In Paul S. Heckbert, editor, *Graphics Gems IV*, pages 286–320. Academic Press, 1994.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2nd edition, 2010.

[PJ91] Andrew Pearce and David Jevans. Exploiting shadow coherence in ray tracing. In *Graphics Interface '91*, pages 109–116, June 1991.

[PMDS06] Voicu Popescu, Chunhui Mei, Jordan Dauble, and Elisha Sacks. Reflected-scene impostors for realistic reflections at interactive rates. In *Eurographics 2006*, pages 313–322, September 2006.

[PML97] Madhav K. Ponamgi, Dinesh Manocha, and Ming C. Lin. Incremental algorithms for collision detection between polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):51–64, January 1997.

[PO08] Anjul Patney and John D. Owens. Real-time Reyes-style adaptive surface subdivision. *SIGGRAPH Asia 2008*, pages 143:1–143:8, December 2008.

[PSS+06] Hans-Friedrich Pabst, Jan P. Springer, André Schollmeyer, Robert Lenhardt, Christian Lessig, and Bernd Froehlich. Ray casting of trimmed NURBS surfaces on the GPU. In *IEEE Symposium on Interactive Ray Tracing 2006*, pages 151–160, September 2006.

[Ren11] RenderMan. Renderman movies. `http://renderman.pixar.com/products/whats_renderman/movies.html`, 2011. Retrieved February 2011.

[RN02] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[RSH05] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. In *SIGGRAPH 2005*, pages 1176–1185, July 2005.

[RW80] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *SIGGRAPH '80*, pages 110–116, July 1980.

[RWS+06] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. In *SIGGRAPH 2006*, pages 977–986, July 2006.

[SDDS00] Gernot Schaufler, Julie Dorsey, Xavier Decoret, and François X. Sillion. Conservative volumetric visibility with occluder fusion. In *SIGGRAPH 2000*, pages 229–238, July 2000.

[SGS06] Carsten Stoll, Stefan Gumhold, and Hans-Peter Seidel. Incremental raycasting of piecewise quadratic surfaces on the GPU. In *IEEE Symposium on Interactive Ray Tracing 2006*, pages 141–150, September 2006.

[SMD+06] Gordon Stoll, William R. Mark, Peter Djeu, Rui Wang, and Ikrima Elhassan. Razor: An architecture for dynamic multiresolution ray tracing. Technical Report TR-06-21, Department of Computer Science, The University of Texas at Austin, April 2006.

[SSK04] László Szécsi and László Szirmay-Kalos. Efficient approximate visibility testing using occluding spheres. *Journal of WSCG*, 12(1–3):435–442, February 2004.

[Stu11] Blue Sky Studios. Tools description. `http://www.blueskystudios.com/content/process-tools.php`, 2011. Retrieved February 2011.

[TL04] Eric Tabellion and Arnauld Lamorlette. An approximate global illumination system for computer generated films. In *SIGGRAPH 2004*, pages 469–476, August 2004.

[WA90] Andrew Woo and John Amanatides. Voxel occlusion testing: A shadow determination accelerator for ray tracing. In *Graphics Interface '90*, pages 213–220, May 1990.

[Wat00] Alan Watt. *3D Computer Graphics*. Addison-Wesley, 3rd edition, 2000.

[WBS07] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray tracing deformable

scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics*, 26(1):6:1–6:28, January 2007.

[WBWS01] Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. Interactive rendering with coherent ray tracing. In *Eurographics 2001*, pages 153–164, September 2001.

[WDS04] Ingo Wald, Andreas Dietrich, and Philipp Slusallek. An interactive out-of-core rendering framework for visualizing massively complex models. In *Eurographics Symposium on Rendering 2004*, pages 81–92, June 2004.

[WH06] Ingo Wald and Vlastimil Havran. On building fast kd-trees for ray tracing, and on doing that in O(N log N). In *IEEE Symposium on Interactive Ray Tracing 2006*, pages 61–69, September 2006.

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[Wu05] Xiaobin Wu. *Efficient, Tight Bounding Volumes for Subdivision Surfaces*. PhD thesis, University of Florida, August 2005.

[YLM06] Sung-Eui Yoon, Christian Lauterbach, and Dinesh Manocha. R-LODs: Fast LOD-based ray tracing of massive models. *The Visual Computer*, 22(9–11):772–784, September 2006.

[ZHR+09] Kun Zhou, Qiming Hou, Zhong Ren, Minmin Gong, Xin Sun, and Baining Guo. RenderAnts: Interactive Reyes rendering on GPUs. In *SIGGRAPH Asia 2009*, pages 155:1–155:11, December 2009.

Vita

Peter Djeu grew up in Melbourne, Florida and graduated from Palm Bay High School in 1998. He then did his undergraduate studies at Harvard University where he braved the snow and received his Bachelor of Arts in Computer Science in 2002. Afterwards he undertook the interesting life decision of becoming a graduate student, pursuing his PhD in computer science at the University of Texas at Austin. He graduated with his PhD in 2011.

During his years in graduate school, he acquired a taste for country music and started developing his own hobbyist video games due to a lifelong love of the medium. In addition to a preexisting affinity for philosophy, he hopes to make these interests an integral part of his life after graduate school, although not necessarily all at the same time.

Permanent Address: djeu@cs.utexas.edu

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab.