UNIVERSITAT JAUME I
Department of Languages and Computer Systems

Techniques for improving the visualization of natural scenes

Ph. D. dissertation
Jesús Gumbau Portalés

Advisor: Dr. Miguel Chover Sellés
Castelló, February 2011

UNIVERSITAT JAUME I
Departament de Llenguatges i Sistemes Informàtics

Técnicas para mejorar la visualización de escenas naturales en tiempo real

Tesis doctoral
Jesús Gumbau Portalés
Director: Dr. Miguel Chover Sellés

Castelló, Febrero 2011

Resumen

La representación interactiva de escenarios naturales en tiempo real presenta problemas debido a la gran cantidad de información que debe ser procesada. Un bosque puede estar compuesto por decenas de miles de especies vegetales, las cuales a su vez están compuestas por decenas de miles de elementos (hojas, tallos y ramas). Una descripción precisa de una escena de este tipo implica el procesamiento de una cantidad de información intratable para su uso en aplicaciones en tiempo real. Durante los últimos 20 años se han publicado muchos trabajos de carácter científico para resolver este problema, aportando diferentes estrategias con particulares ventajas e inconvenientes.

El objetivo principal de esta tesis es la presentación de un conjunto de técnicas para mejorar la eficiencia en la representación de escenas naturales en tiempo real. Para ello se han realizado aportaciones en diferentes ámbitos del área de los gráficos por computador. Por una parte, se ha propuesto un modelo de nivel de detalle multirresolución especialmente diseñado para la vegetación, basado en nivel de detalle de resolución variable, el cual permite la adaptación de la complejidad poligonal de especies vegetales en tiempo real. Esta propuesta se acompaña de una nueva aportación diseñada para el manejo de escenas masivamente pobladas, es decir, con miles de modelos multiresolución. Además se han propuesto nuevas técnicas para mejorar la visualización en el ámbito de la iluminación, tanto en cuanto a modelar los efectos de iluminación en árboles como a la representación eficiente de sombras visualmente realistas.

Objeto y objetivos de la investigación

El área de estudio de los gráficos por computador se centra en la visualización y manipulación de información visual y geométrica para la generación de imágenes sintéticas. En este área se proponen algoritmos y métodos que, a partir de la descripción de una escena (objetos, materiales e iluminación), generan una imagen que representa la escena en dos dimensiones. Dentro de este área, el objetivo de los gráficos por computador en tiempo real es la generación de imágenes lo más rápidamente posible, en ocasiones en detrimento de la calidad visual, con el objetivo de ofrecer aplicaciones interactivas, esto supone que el tiempo máximo permitido para la generación de cada imagen sea de 66 milisegundos, lo que permite generar animaciones a 15 Hz (fotogramas por segundo).

En el área de los gráficos en tiempo real existen dos temas de gran importancia que intervienen en gran medida en el realismo y en la calidad de las imágenes generadas: el modelado geométrico y la iluminación. Cuanto más preciso y detallados sean los cálculos invertidos en cada proceso mayor es la calidad obtenida, pero mayor también es el tiempo requerido para obtener los resultados. Sin embargo, durante le última década la potencia de los procesadores gráficos (GPU) ha aumentado de forma exponencial, permitiendo el desarrollo de técnicas interactivas realistas que antes se relegaban al ámbito de la representación no interactiva.

Modelado Multirresolución de especies vegetales

Como se ha mencionado anteriormente, cuanto más detallado sea el modelado geométrico de un objeto mayor es el realismo que aporta, sin embargo también aumenta el coste de representación. Para paliar este problema se han realizado muchos estudios relativos al concepto de nivel de detalle. Éste tiene en cuenta factores como el tamaño proyectado del objeto en pantalla o su visibilidad como criterio para reducir la complejidad geométrica del mismo, de forma que no se desperdicien recursos procesando detalles que están demasiado lejos para poder ser apreciados. El modelado multirresolución contribuye a disimular el impacto visual entre transiciones de diferentes niveles de detalle, de lo contrario se producen efectos que resultan molestos a la visualización y restan solidez a la escena. A éstos efectos se les denomina efectos de “popping”.

Aunque a lo largo de los años se ha publicado un gran número de estudios entorno al modelado multirresolución y al nivel de detalle, éstos han sido enfocados en la mayoría de los a mallas formadas por superficies continuas. Sobre este campo se han desarrollado multitud de métodos y métricas de simplificación que permiten la reducción de la complejidad poligonal de

mallas de triángulos a una fracción de la original con un impacto mínimo sobre el aspecto original, lo que aumenta enormemente su eficiencia en la representación. Sin embargo, las representaciones poligonales de especies vegetales no están formadas por densas mallas continuas de triángulos sino por multitud de pequeños elementos separados que forman la copa del árbol y las ramas. En esta situación las métricas de simplificación previamente mencionadas no se comportan correctamente y generalmente no producen aproximaciones válidas. Sin embargo, durante los últimos años han surgido métodos de simplificación especialmente diseñados para elementos disconexos como la copa de los árboles. Esta tesis aporta soluciones en este campo proponiendo un modelo multirresolución de resolución variable especialmente diseñado para especies vegetales que está ideado para ser implementado en la GPU, lo que permite explotar el poder computacional de estos dispositivos para maximizar el rendimiento.

El modelo multirresolución propuesto en esta tesis permite la visualización eficiente de árboles y plantas de gran densidad poligonal. No obstante, no hay que perder la perspectiva de la representación de bosques con miles de elementos en escena. En este caso, debido al tiempo de cálculo necesario para cambiar el nivel de detalle de cada objeto en la escena, el manejo del modelado multirresolución con una gran cantidad de objetos puede repercutir negativamente en el rendimiento, creando un cuello de botella que puede colapsar el sistema. Por esa razón se propone un sistema de manejo de escenas de nivel de detalle que permite la utilización de miles de elementos sin que se pierda velocidad en la extracción del nivel de detalle.

Iluminación de árboles y plantas

Tal como sucede con el modelado geométrico, cuanto más precisa y realista sea la técnica de iluminación utilizada más costosos son los cálculos asociados. Realizar una simulación de forma físicamente correcta de la iluminación en escenas complejas en tiempo real dista mucho de ser posible a día de hoy. Existen técnicas, ampliamente utilizadas en entornos de producción de animaciones sintéticas, que permiten la re-presentación de estas escenas de forma realista. Sin embargo, debido a la complejidad de los cálculos implicados en la simulación de la luz, el tiempo requerido para calcular cada fotograma de la animación es muy elevado, pudiendo incluso requerir varias horas por fotograma. Por lo tanto estas técnicas no son aplicables en aplicaciones interactivas.

Gracias a la fuerte evolución que han sufrido los procesadores gráficos durante los últimos años, han surgido técnicas que ofrecen aproximaciones realistas de la iluminación en entornos controlados. La clave de éstas técnicas es que se realizan estrictas asunciones que permiten simplificar los cálculos

requeridos con la consiguiente pérdida de flexibi-lidad y realismo. Sin embargo, éstas técnicas permiten la generación de imágenes sintéticas de calidad de forma interactiva. Un ejemplo de esto son las técnicas basadas en la simulación de la iluminación ambiente en el espacio de imagen. Esta tesis presenta técnicas para la aproximación en tiempo real de la iluminación especialmente diseñada para simular la interacción de la luz sobre la copa de los árboles.

Sombras

Aunque las sombras son un efecto generado por la falta de iluminación y pueden ser generadas indirectamente y de forma realista me-diante algoritmos de simulación de iluminación físicamente correctos, en el campo de los gráficos en tiempo real siempre se han calculado como un fenómeno independiente. Por este motivo existe una corriente de investigación dedicada al estudio de la generación eficiente de sombras realistas en tiempo real. Las técnicas más utilizadas hoy en día para la generación de sombras en aplicaciones interactivas son las basadas en "shadow maps". El problema de esta técnica es que, a parte de los problemas de discretización a la que está sujeta que produce pixelización o dentado, sólo es capaz de crear sombras "cortadas a cuchillo". Sin embargo, las luces del mundo real tienen un volumen que, dependiendo de su tamaño entre otros factores, genera una zona de transición suave entre los puntos que están en sombra y los que no. Esta transición se denomina penumbra y contribuye en gran medida al realismo en escenas generadas por ordenador. Las técnicas que estudian la generación de este tipo las denominan sombras suaves o "soft shadows". Éstas son capaces de reproducir este tipo de efectos en tiempo real, aunque éstas tienen un coste elevado dependiendo de la calidad (o suavidad) de la penumbra.

Esta tesis aporta dos tipos de mejoras en la representación de sombras en tiempo real. La primera contribución propone una nueva técnica de antidentado de sombras en tiempo real. Esta técnica se basa en técnicas conocidas y las mejoras reduciendo las situaciones en las que presentan problemas. La segunda contribución presenta una nueva técnica de generación de sombras suaves que mejora el rendimiento de otras técnicas mediante la utilización de filtrados Gaussianos de tamaño variable en el espacio de la imagen.

Planteamiento y metodología utilizados

Esta tesis tiene como objetivo la presentación de técnicas que mejoren el rendimiento en la representación de escenas naturales en tiempo real. Dichas técnicas pueden ser utilizadas en aplicaciones interactivas tales como videojuegos o aplicaciones de realidad virtual. Para ello se propone realizar

las tareas siguientes:

Estado del arte

Antes de definir una línea de trabajo conviene hacer un estudio en profundidad de las técnicas existentes que aporten soluciones a los objetivos propuestos en la tesis. Este análisis nos ofrecerá una visión de las ventajas e inconvenientes de los métodos existentes en la literatura y por consiguiente revelará las oportunidades de mejora apropiadas. En este apartado se pretende analizar el estado actual de las siguientes áreas dentro del campo de estudio de los gráficos en tiempo real: modelado multirresolución, manejo eficiente de escenas muy pobladas, modelado y representación de especies vegetales, técnicas de iluminación y de métodos de generación de sombras, incluyendo métodos de filtrado y representación de sombras suaves. El Capítulo 2 de esta tesis presenta un análisis detallado de las técnicas estudiadas durante esta fase.

Propuesta y desarrollo de técnicas para mejorar el rendimiento de escenas naturales.

Basándose en el estado del arte, se han estudiado las oportunidades de mejora y los inconvenientes de las técnicas existentes y se han propuesto nuevas técnicas para paliar estos problemas. Se han desarrollado técnicas originales en los siguientes campos: modelado multirresolución, manejo de escenas con nivel de detalle, iluminación de especies vegetales, filtrado de sombras y generación de sombras suaves en tiempo real.

Evaluación y comparación de los resultados

Para la validación y comprobación de las ideas desarrolladas se han utilizado las herramientas necesarias como motores de juegos o aplicaciones de edición 3D. Alternativamente, en los casos apropiados se han creado nuevas plataformas de software para la correcta validación de las propuestas. Los resultados obtenidos han sido analizados y comparados cuantitativa y cualitativamente para su validación.

Aportaciones originales

Esta tesis propone diversos métodos y técnicas para la mejora del rendimiento de aplicaciones interactivas que necesiten visualizar entornos naturales en

tiempo real. Las aportaciones realizadas en este marco son de distinta índole y se describen a continuación.

Inicialmente, el Capítulo 2 presenta el estado del arte de los temas tratados en esta tesis. En este capítulo se hace un estudio de las técnicas existentes en los campos de interés, analizando sus ventajas y desventajas, de forma que se muestren claramente las oportunidades de mejora que presenta cada una. Los temas de interés tratados en esta tesis son: modelado de multirresolución de especies vegetales, manejo de escenas con nivel de detalle, iluminación de árboles y plantas, filtrado de sombras y generación de sombras suaves en tiempo real. Los siguientes capítulos describen las técnicas desarrolladas para mejorar los algoritmos analizados en este capítulo.

La representación realista de bosques masivamente poblados aporta muchos problemas de eficiencia que dificultan la utilización de este tipo de escenas en aplicaciones en tiempo real. Para ayudar a resolver este problema en el Capítulo 3 se propone el desarrollo de una técnica de representación de este tipo de modelos basada en nivel de detalle de resolución variable, que permite reducir considerablemente la cantidad de datos a procesar en cada fotograma, minimizando el impacto en el aspecto visual y maximizando la eficiencia de la etapa de dibujado.

Como se ha comentado anteriormente, la utilización de modelos multirresolución en escenas masivamente pobladas puede provocar una caída del rendimiento que anule por completo las ventajas de la utilización de modelos de este tipo. Para resolver este problema esta tesis propone el desarrollo de un gestor de escenas de nivel de detalle multirresolución. Este gestor se basa en la alta reutilización de niveles de detalle similares en escenas de este tipo. La idea detrás de esta técnica reside en que no es realmente necesario calcular el nivel de detalle por cada objeto de la escena por separado, sino que es posible reutilizar niveles de detalle calculados en etapas anteriores para aplicarlos en otros objetos del mismo tipo. Esta observación permite implementar este tipo de modelos en escenas masivas sin la penalización anteriormente mencionada. El Capítulo 4 detalla esta técnica y los resultados obtenidos de forma más extensiva.

A continuación, el Capítulo 5 presenta una técnica para el cálculo de iluminación en árboles y plantas en tiempo real mediante la utilización de mapas de distancia. La clave de este método reside en la utilización de mapas de profundidad para capturar el volumen que ocupa la copa del árbol en el espacio de forma que es posible hacer una estimación rápida de la cantidad de iluminación que recibe cada hoja. Con esta estimación obtenemos una aproximación de la iluminación de baja frecuencia. Así pues, se ofrecen dos técnicas alternativas para el cálculo de la iluminación de alta frecuencia que aporta información de interacción local de la iluminación a nuestra solución.

El Capítulo 6 presenta una aportación original para la generación de som-

bras suaves con penumbra de tamaño variable en tiempo real. Este método se basa en la utilización de un filtrado Gaussiano de tamaño variable que difumina las sombras en espacio de imagen para generar la transición suave entre la luz y la sombra. Para determinar el tamaño del filtro Gaussiano y por consiguiente del tamaño de la penumbra se evalúan varios parámetros como la distancia relativa de los objetos que proyectan la sombra y los que la reciben, el tamaño de la fuente de luz y sus distancias relativas con la posición de la luz. La ventaja de este método respecto a otros existentes es que al usar un filtro Gaussiano para generar la penumbra se reduce un orden de magnitud en el coste del algoritmo, pasando de un coste $O(n^2)$ a un coste $O(n+n)$.

Finalmente y continuando con el tema de las sombras, el Capítulo 7 introduce una nueva técnica de filtrado de sombras para eliminar los efectos de dentado presentes en las técnicas basadas en "shadow mapping". Básicamente esta técnica es una extensión de técnicas conocidas, como "variance shadow maps", que mejora la evaluación de las sombras de forma que se reducen drásticamente los casos en los que las técnicas existentes presentan errores de representación.

Conclusiones obtenidas y futuras líneas de investigación

Del estudio planteado en el Capítulo 2 podemos concluir que, aunque existen multitud de técnicas que aportan mejoras en diferentes aspectos, también tienen desventajas de algún tipo que sugieren oportunidades de mejora. Esta tesis tiene como objetivo proponer nuevas técnicas para resolver los problemas tratados maximizando aún más las soluciones intentando minimizar las desventajas.

En el Capítulo 3 se presenta una técnica multirresolución que permite adaptar el nivel de detalle de especies vegetales y plantas modeladas poligonalmente con el fin de optimizar su representación en pantalla. Esto permite aumentar la cantidad de vegetación a usar en aplicaciones en tiempo real, donde el tiempo de dibujado es crítico para ofrecer interactividad. La solución propuesta está diseñada para ofrecer un modelo de nivel de detalle de resolución variable, lo que permite ajustar a la complejidad del modelo dependiendo del punto de vista del observador. Esto es importante para modelos de árboles ya que, al estar compuestos por un conjunto denso de elementos (las hojas), su representación en pantalla está sujeta a un gran porcentaje de auto-oclusión que depende del punto de vista. Esto contribuye a optimizar más la geometría del modelo afectando menos al aspecto visual del objeto en pantalla, ya que permite eliminar más geometría de las partes ocultas del modelo. Además, esta solución está especialmente diseñada para arquitecturas paralelas, por lo que se consigue un gran ben-

eficio implementándolo en la GPU. Como se puede ver en los resultados del Capítulo 3, este algoritmo rinde mejor que otras soluciones no basadas en la GPU. Además, en los test de calidad visual se puede apreciar que el algoritmo mantiene la calidad visual del modelo original aun usando grandes factores de simplificación.

Como se ha comentado anteriormente, los modelos multirresolución tienen asociado un tiempo de extracción, que es el tiempo necesario para "preparar" la malla para ajustarla al nivel de detalle deseado. Por muy rápido que se realice este paso, éste se acumula en escenas con gran cantidad de objetos y puede repercutir negativamente en el rendimiento de la aplicación. Para paliar este problema, la técnica presentada en el Capítulo 4 permite minimizar el número de operaciones de extracción de nivel de detalle, de forma que es posible eliminar este cuello de botella. Los resultados obtenidos confirman que esta técnica permite manejar escenas compuestas por grandes cantidades de modelos multirresolución sin que esto repercuta negativamente en el rendimiento. Además esta solución es fácilmente integrable en herramientas de "software" existentes o motores 3D.

El cálculo de iluminación de forma realista es un problema en aplicaciones en tiempo real, ya que realizar una simulación físicamente correcta del comportamiento de la luz sería inviable. La solución presentada en el Capítulo 5 realiza una estimación del volumen de la copa de los árboles y plantas involucrados en la visualización para aproximar la cantidad de iluminación que recibe cada hoja. Esta solución es fácilmente integrable en aplicaciones en tiempo real y permite el cálculo de la iluminación de baja frecuencia sobre este tipo de modelos. Para completar el cálculo de la iluminación, esta técnica se complementa con otra técnica basada en la evaluación del árbol en espacio de pantalla para determinar, de forma aproximada, la iluminación de alta frecuencia (altamente detallada). Las imágenes obtenidas en este capítulo muestran que los resultados son visualmente satisfactorios, incluso siendo generados en tiempo real.

El Capítulo 6 introduce una nueva técnica que permite calcular sombras con penumbra físicamente realista en tiempo real ("soft shadows"). La ventaja de esta técnica radica en que el cálculo de la penumbra se realiza mediante la aplicación de un filtro Gaussiano de tamaño variable en espacio de pantalla. Dado que este tipo de filtros es separable (se puede descomponer en dos pasadas independientes), su complejidad computacional resulta de un orden de magnitud inferior a soluciones existentes, como PCSS [Fer05], que tienen un coste de $O(n^2)$. Sin embargo, aunque esta técnica rinde mejor que otras soluciones, al ser aplicada como un filtro de post-proceso, esto hace que sea más sensible a errores en la generación de la penumbra.

Finalmente, el Capítulo 7 presenta una nueva técnica de filtrado de sombras. La principal diferencia entre las técnicas de este tipo y las de "soft

shadows" es que el objetivo de las primeras es la eliminación del dentado, producido por la utilización de técnicas derivadas de los mapas de sombra, mientras que el objetivo de las últimas es la generación de la penumbra realista que depende de factores como las posiciones relativas entre los objetos y el tamaño, posición y forma de la fuente de luz. La técnica propuesta en este capítulo presenta una nueva formulación para la evaluación de los mapas de sombra, con el objetivo de reducir la sensibilidad de éstas en cuando a producción de errores en la representación de las sombras antidentadas. En la sección de resultados se puede comprobar cómo se consigue reducir este tipo de fallos sin aumentar los costes computacionales y de memoria.

Futuras líneas de investigación

En esta tesis se presentan técnicas de diversa índole para mejorar la representación de escenas naturales en tiempo real para aplicaciones interactivas. Sin embargo, aunque las técnicas propuestas presentan mejoras sobre métodos existentes, también tienen desventajas que dependen de varios factores o presentan oportunidades de mejora que deben ser estudiados en el futuro. Esta sección ofrece una serie de líneas de trabajo que se desprenden de esta tesis y que pueden servir para mejorar estas técnicas, completarlas o hacerlas más robustas.

En primer lugar, como extensión del modelo multirresolución para especies vegetales se propone estudiar las ventajas de una implementación en OpenCL en lugar de CUDA, ya que esta API permite compartir recursos entre distintos contextos, lo cual sería útil para compartir información con OpenGL. Otro aspecto interesante sería la inclusión de técnicas de simulación de efectos naturales como viento o lluvia y ver cómo se debería modificar el modelo multirresolución para soportar dichos efectos.

Sobre la técnica de manejo de escenas multirresolución, se propone el estudio de cómo afectaría la inclusión de grandes cantidades de objetos animados en la escena y cómo habría que modificar el gestor presentado para soportar esos cambios, asumiendo que la animación influye en el funcionamiento del modelo multirresolución subyacente. También se propone la extensión el gestor para no sólo "cachear" niveles de detalle sino también animaciones, de forma que el funcionamiento de éste deje de ser exclusivo para manejar niveles de detalle y también sirva para manejar grandes cantidades de animaciones, optimizando el rendimiento en este tipo de escenas.

En el Capítulo 5 se propone una técnica para la iluminación de vegetación en tiempo real, capturando el volumen de los árboles teniendo en cuenta una dirección preferente. Como trabajo futuro se propone el estudio de cómo extender este método para tener en cuenta más direcciones para estimar el volumen del árbol y así ofrecer una mejor aproximación de la

iluminación en este tipo de objetos. Además es conveniente aumentar el estudio con árboles con distintos tipos de hoja para extender la técnica en caso de que no sirva para todos.

En futuras líneas de investigación en torno a los artículos de sombras presentados en esta tesis se proponen posibles mejoras de distinto tipo. Primero, la calidad de técnica de generación de penumbras realistas presentada en el Capítulo 6 es muy dependiente del objeto sobre el que se arrojan las sombras y de la complejidad de la escena. En el peor de los casos esta técnica podría no funcionar bien y generar penumbras incorrectas o con errores. Se propone el estudio de los casos en los que la técnica es más susceptible a fallar con el fin de extenderla para hacerla más robusta en este aspecto. En cuanto a la técnica de filtrado de sombras presentada en el Capítulo 7, las líneas de trabajo futuras pasan por estudiar variantes de la técnica que permitan aumentar la robustez de la misma para escenas complejas en las que no es posible aplicar correctamente el método de antidentado.

Preface

Abstract

Interactive rendering of natural scenes tipically faces performance problems when being implemented for real-time applications, due to the amount of information to be processed each frame. A single forest is tipically composed by a high number of vegetal species (thousands of trees and plants) and each single one is composed of a high amount of elements, such as leaves and branches. A precise description of this kind of scenes involves the processing of a massive amount of information that makes it impracticable for real-time applications, such as videogames. Several works have appeared during the last twenty years with the aim of solving this problem. Each one of these works provides both advantages and disadvantages. Sometimes those disadvantages are so strict that highly reduce effectifity. The main aim of this thesis dissertation is to present a set of techniques for enhancing efficiency in real-time applications when rendering natural scenes. Taking this into account, a number of novel approaches are proposed in this work which provide improvements on different areas in the field of computer graphics. On the one hand, a new multiresolution model for efficient rendering of vegetal species is presented. This proposal is accompained with a novel technique for managing massively propulated multiresolution scenes for avoiding performance penalties due to the uncontrolled of continuous level of detail approaches. On the other hand, a new illumination model for the foliage are proposed which aims to improve illumination of forests in real-time. Finally, two new approaches for improving shadow mapping efficiency are presented, one for shadow filtering and another one for efficient soft shadows generation.

Keywords: Real-Time Rendering, GPU, multiresolution, Level-of-Detail, illumination, shadows.

Funding

This work has been supported by the Spanish Ministry of Science and Technology (grants TSI-2004-02940, TIN2007-68066-C04-01, TIN2007-68066-C04-02, TIN2010-21089-C03-01 and TIN2010-21089-C03-03), be the Jaume I University (PREDOC-2006-54) and by Fundació Bancaixa (P1 1B2010-08).

Acknowledgements

This thesis is the result of years of work in the field of Computer Graphics and has been possible thanks to the help and support of many people.

First of all, I would like to thank Miguel Chover, who has been my supervisor since the beginning of my Ph.D. studies, for tutoring me and for allowing me to work on the best job one can do: Computer Graphics!

I would also like to thank to my colleagues at the University Jaume I: Francis, Anna, Cristina, Arturo, Pascual, Joaquín and rest of the members of the Computer Graphics Department. Special thanks go to Carlos, for his guidance in getting through all the process of the thesis, to Inma, for taking so much time and effort for correcting this text, to Ricardo, for the beautiful cover of this book, and to Óscar for being a cool office-mate.

At this point, I would like to mention the people from Budapest: Gergely, Milán, Zsófi, László Szécsi, Istvan and of course to László Szirmay, from the Technical University of Budapest. Also thanks to the people from Girona: Mateu, Nico and Xavi, and to the members of the experts comitee: Anton Penzov, Barnabás Takács and Miquel Feixas.

Last but not least my special appreciation goes to my parents, my whole family and friends for their support and encouragement.

Finally, I would like to give the biggest "Thank you!" to Sandra for years of support, friendship and endless love.

Thank you very much to all of you :)

Index

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Computer graphics is the field of research that studies the theory and techniques for the generation of synthetic images on the computer. In 3D computer graphics the process of generating images from a set of data which approximates a scene is called *rendering*. During rendering, the scene is processed by the graphics pipeline and converted into pixels on the screen. The purpose of this process, which is called *rasterization*, is to generate a two-dimensional representation of the scene: a synthetic image.

Synthetic scenes in computer graphics can be represented in several ways: by using a polygonal representation, by using implicit surface modelling, volumetric modelling (voxels) or fractal representation for procedural modelling of scenes. Nevertheless, the most common representation used in 3D computer graphics is approximating objects and scenes by polygonal surfaces (triangles or quadrilateral meshes) because of their flexibility and their rendering efficiency. However, depending on the distance to the observer, polygonal models start to become inefficient as the size of the projected input geometry approaches the size of a pixel. To solve this problem, Level of Detail algorithms (LoD) adjust the polygonal density of meshes by reducing it according to the viewing distance and hence improving performance.

Sometimes, only a fraction of the scene is projected on the screen and thus processing the whole scene is not efficient. Visibility methods take into account this problem and provide acceleration structures for optimizing the graphics pipeline by minimizing the amount of calculations. Some examples of acceleration structures are octrees and portals, which are extensively used in real-time game engines.

Realism is a key aspect in computer graphics. Although the amount of

geometry improves the realism of the scenes, illumination takes a key role in improving the overall quality of synthetic images. Highly realistic illumination can be achieved by simulating the physics of light and its interaction with the environment. Depending on the trade-off between performance and visual quality several methods exist for performing lighting calculations.

The highest objective in computer graphics is the generation of photorealistic synthetic images as fast as possible. In real-time computer graphics, this means that generating one image should not take more than 66 milliseconds, which allows for generating 15 images per second (15 Hz). The faster the generation of the images, the smoother the animation and the sense of interactivity. From about 75 Hz the human eye is not able to detect differences in performance.

In the last decade, real-time computer graphics have quickly evolved, mainly due on the one hand to the video-game industry, which demands better graphics and faster graphics processors, and on the other hand due to the amount of scientific research that has been done on this area for more than 30 years.

Moreover, the appearance of the first programmable graphics processors introduced unprecedented flexibility for computer graphics programmers for implementing algorithms that were not possible to port to the graphics processors, due to their initial limitations. For this reason, many algorithms that were only used in off-line rendering (such us movies and post-production) to be able to be used for real-time graphics.

The techniques described in this thesis propose improvements to the sub-fields of level of detail, multiresolution management, illumination, soft shadowing and shadow filtering. These filtering techniques make extensive use of the advanced capabilities of the graphics processors for proposing techniques that improve performance of real-time applications in various aspects of the rendering pipeline.

1.1. Objectives of the research

1.1.1. Modelling of vegetation

Efficient rendering of natural scenes has always posed a problem for real-time computer graphics. More specifically, we consider that one of the major problems when rendering outdoor scenes is the efficient visualization of vegetal species, such as plants and trees, due to the massive amount of information involved in rendering. Foliage density, and hence the number of leaves of a tree, can vary depending on the species from several thousands to hundred thousands. Depending on the shape of the leaves, they can be

Figure 1.1: Real forest composed of hundreds of thousands of trees.

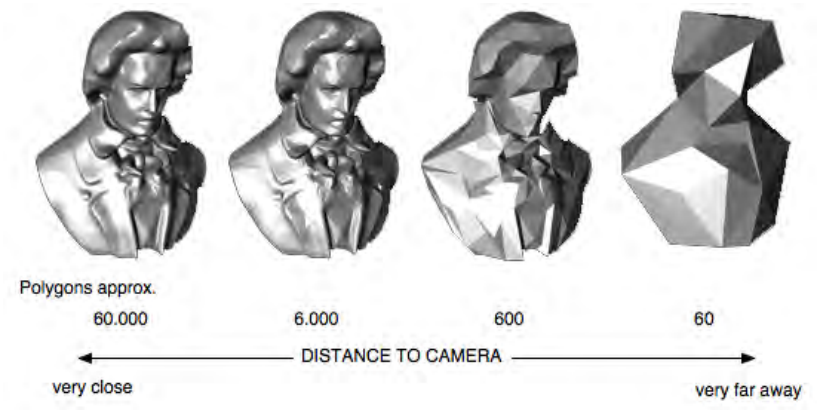


Figure 1.2: The LoD factor of a mesh associated to the viewing distance

modelled using a single quadrilateral or by using some dozens of polygons for complex elements. In addition, as the amount of leaves increase, the number of branches and ramifications also grow with it, amounting some hundred thousand polygons utilized for modelling the trunk and branches. Taking into account that many thousands of trees are possibly used in forest scenes and that they need to be rendered at real-time speeds (more than 15 Hz), billions of polygons should be necessary for rendering large forest (Figure 1.1). This massive amount of information also presents issues for efficient scene management and visibility determination as well as in the area of realistic illumination of vegetation.

This thesis proposes techniques for improving visualization of natural

Figure 1.3: Example of a natural scene with a massive amount of plants and trees in Crysis (Crytek ©2007).

scenes in the main aspects described before: plant modelling, visualization and illumination.

Most of the solutions adopted for games and other real-time applications in order to manage trees and forests rely on the use of billboards and impostors for replacing massive amounts of leaves by rendered polygons that offer good visualization at far distances, but suffer from clustering and parallax artifacts at intermediate distances. Figure 1.3 shows an example of a natural scene rendering using the CryEngine in Crysis by Crytek.

With the motivation of providing a solution to this problem we propose a view-dependent multiresolution LoD model specially designed for efficient rendering of highly detailed trees. This LoD model will allow to smoothly decreasing the geometry complexity of trees depending on the distance to the observer and its orientation.

Techniques based on level of detail approaches allow to reduce the amount of geometry used to render the object while maintaining visual appearance. This solution is commonly used when rendering objects at a certain distance, at which small details are not visible or correctly distinguishable. The LoD factor determines the percentage of geometry to be rendered at a given level of detail and is usually associated to the viewing distance for eliminating unnecessary detail (see Figure 1.2).

1.1.2. Natural Scene Management

The usage of LoD objects allows for minimizing the computational rendering costs. However, multiresolution modelling often involves a step in which the LoD algorithms are executed in order to prepare the mesh for be-

ing rendered. This stage is responsible for processing the data structures to change the level of detail of the mesh. The time needed to perform this step is known as LoD extraction time. This time is a performance penalty that must be taken into account in order to work with multiresolution models because it can degrade performance on highly populated scenes. This cost is a key factor in the performance of multiresolution models and it is object of research in the literature.

Therefore, using an efficient LoD model does not guarantee good performance on massively populated scenes such as large forests and can even stall the rendering system due to the amount of LoD changes per second. For that reason, in order to avoid performing unnecessary calculations that affect negatively to performance, scene management systems control the LoD operations performed over the whole scene. Visibility determination algorithms are also a key feature in this kind of scenes as they provide mechanisms for optimizing rendering, preventing potentially invisible objects from being processed by the graphics hardware. This is an important part of an interactive application, especially when rendering massively populated scenes like forests.

In order to solve this problem we propose a new model for multiresolution scene management based on controlling the frequency at which LoD models perform the extraction process per second. Our technique is based on the concept of reusing previous LoD results, which uses a metric for deciding whether to calculate the desired level of detail or reusing an existing LoD from another object instance which already performed the extraction process. By using this technique we are able to minimize the amount of LoD calculations, maximizing performance on massively populated LoD scenes. This thesis also proposes techniques for exploiting the graphics hardware to improve these aspects when working with LoD models as well as for LoD management of large LoD scenes.

1.1.3. Illumination

Although LoD systems provide an efficient platform for the efficient rendering of geometry, realism is becoming more and more important every day as the computational power of the graphics hardware improves from generation to generation, allowing programmers to implement more realistic effects and illumination algorithms.

One of the most important aspects in computer graphics is how light affects the visualization of the rendered models on screen. While direct illumination can be easily computed, indirect illumination is a current research topic in real-time computer graphics because it is directly responsible of the realism of synthetic scenes. However, global illumination is very hard

to compute because it has to take into account complex light interactions inside the scene. Many approaches have been proposed in order to approximate global illumination in real-time, however these approaches are usually subject to strong restrictions in order to allow interactivity.

In order to provide a solution to this problem, this thesis proposes a new illumination model for the foliage that takes into account the sparse nature of leaves in order to provide better illuminated trees. We exploit the observation that, due to how light interact with surfaces, the inner leaves in a tree will potentially receive less light that the outermost leaves. Our solution uses two depth maps that approximate the shape of the tree and, assuming that leaves are uniformly distributed across the foliage, we are able to approximate the illumination of each single leaf.

1.1.4. Real-time shadows

Shadows are a key feature in computer graphics because they provide a lot of essential visual information on synthetic scenes. Although shadows are naturally generated by the absence of light and thus they are related to the illumination, in real-time computer graphics they are artificially generated due to performance reasons.

Two major techniques are used in real-time computer graphics for shadow rendering: shadow volumes and shadow mapping. Shadow volumes are a geometry-based solution that is based on generating a shadow volume taking into account the position of light and the shadow caster. This is used to determine which pixels in the shadow receiver intersect with the shadow volume and thus are in shadow. Shadow mapping is a two-pass image-based method that stores the depth of the shadow casters in light-space and then uses this information in eye space for determining whether each pixel is lit or not. Although both methods have their own drawbacks and benefits, shadow mapping is preferred over shadow volumes because they are easier to implement and the implementation is more efficiently executed by the graphics hardware.

The two shadowing techniques previously described are able to generate hard-edged shadows that look great for infinitely small light sources. However, these techniques generate aliased shadows and they are not able to generate soft-edges penumbrae, like those produced by area sources in real life, requiring further improvements in this area. This issue has given rise to a number of works in the literature and has become an important topic, due to its importance in realistic representation of illumination.

This thesis dissertation presents improvements to the state of the art of two apparently similar but different fields: shadow map antialiasing and soft shadow mapping.

Area light sources are volumetric objects that emit light from all the points over their surface. In fact, area light sources can be simulated by distributing a high number of point light sources over the surface. Although this approach would produce physically correct soft shadows, it would be very inefficient because a lot of rendering operations should be performed each frame. Taking this into account a number of works proposed techniques for efficiently approximating soft shadow penumbrae.

For solving this issue, this thesis proposes a new algorithm for efficiently computing soft shadows in screen space by using an anisotropic Gaussian blur of variable size. This technique takes into account the position and size of the light source (assuming a planar light source parallel to the shadow receiver) for deciding the size of the penumbrae for each pixel. This technique fits well on deferred shading approaches because it can be applied as another screen space operation.

The field of shadow filtering includes those techniques designed to eliminate aliasing artifacts on hard-edged aliased shadows. The difference between this approach and soft shadowing techniques is that the former ones do not take into account the light source producing uniformly sized penumbrae, even though there exist soft shadowing techniques that extend shadow filtering approaches to perform variable size penumbrae based on the position and size of the light source.

Filtering shadow maps is not trivial as they can not be filtered like regular textures, because they contain depth values, not colors, and averaging them would produce invalid depth values. Several works have appeared in the literature in the last years to deal with efficiently shadow map filtering using the graphics hardware. However, these approaches have limitations and produce artifacts, depending on how they perform the filtering. We introduce a new shadow map filtering algorithm approach that is able to reduce or even eliminate these artifacts for moderately complex scenes.

1.2. Contributions and Overview

The following list shows the organization used in this Ph.D. thesis, presenting the contributions proposed in this dissertation:

- Chapter 2: **Previous Work**
  This chapter presents the state-of-the-art on the different fields discussed in this dissertation. Therefore, we start by considering the work previously carried out on the field of level of detail for foliage rendering reviewing both image-based and geometry-based methods. Next, scene management methods for massively populated level of detail scenes are

reviewed. These methods are important for efficiently handling populated environments with hundreds and even thousands of LoD objects. Then, a review of foliage illumination methods is presented, including the Precomputed Radiance Transfer method, obscurances and ambient occlusion as well as methods based on sub-surface scattering for realistically illuminating the leaves.

Finally, a review of the state of the art on real-time shadows is provided. We focus on relevant work about shadow mapping as our methods are based on this approach. We review the related work about real-time soft shadows and shadow map filtering, respectively.

- Chapter 3: **View-dependent pruning for real-time rendering of trees**
  The novelty of our algorithm relies on the fact that it is a completely GPU-based view-dependent multiresolution model for the foliage. In this method, the data storage structures, the LoD extraction process and rendering algorithms have been designed to be performed on the GPU. This approach has some direct advantages. First, it removes any traffic between the CPU and the graphics processor, avoiding the PCIe bottleneck.

  Secondly, the multiresolution models to date have been designed for single threaded systems or do not specify how their algorithms are executed on parallel architectures. This chapter provides the basics for building view-dependent multiresolution models on highly parallel environments such as the GPU.

  Finally, we propose a LoD management system that allows for efficiently managing the level of detail of thousands of tree instances. This system, which is also entirely executed on the GPU, prevents the system to collapse the computing resources and maximizes performance.

- Chapter 4: **LoD Manager**
  A framework for efficiently rendering massive multiresolution scenes in real-time applications is introduced in this chapter. This approach uses the concept of frame rate feedback to automatically adapt the level of detail of the scene to achieve a target user-defined frame rate. This approach offers more interesting results rather than static heuristics because it allows for dynamic LOD adaptation. Even though it is less accurate compared to predictive heuristics, it also is considerably less expensive compared to predictive methods, which is the main aim of this chapter: minimize the CPU work as much as possible.

- Chapter 5: **Real-time illumination of foliage using depth maps**
  This chapter presents a new method for foliage illumination which

takes into account direct, indirect illumination and self-shadowing. Both indirect illumination and self-shadowing are approximated by means of a novel technique using depth maps. In addition, a new shadow casting algorithm is developed to render shadows produced by the foliage onto regular surfaces which enhances the appearance of this kind of shadows compared to traditional shadow mapping techniques.

- Chapter 6: **Screen Space Soft Shadows**
  A new technique for the real-time rendering of shadows with penumbrae based on shadow mapping is presented in this chapter. The method uses a screen-aligned texture which contains the distance between the shadow and its potential occluder. This information is used to set up the size of an anisotropic Gaussian filter kernel applied in screen space which smoothens the standard shadows to create the penumbra. Given that a Gaussian filter is separable, the number of samples required to create the penumbra is much lower than in other soft shadowing approaches. In consequence, higher performance is obtained while also allowing perceptually correct penumbrae to be represented.

- Chapter 7: **Improving Shadow Map Filtering with Statistical Analysis**
  This chapter presents our shadow map filtering method that makes use of statistical filtering for approximating the probability that the shaded point passes the depth test. Our approach is capable of highly reducing "light bleeding" artifacts, or even eliminating it for moderately complex scenes, with no penalty of performance or storage costs in the Gaussian case. Moreover, for very complex scenes, it can be converted to a layered approach (in the same way as layered variance shadow maps) for completely eliminating these artifacts. Very few layers are needed in this case, still outperforming existing techniques both in performance and storage costs.

CHAPTER 2

State of the Art

In the last years, a lot of works have appeared in the literature covering the topics involved in real-time rendering of natural scenes. The increase of the computational power of the graphics hardware allowed for creating a great variety of new techniques for improving several aspects in the field of natural phenomena, like rain, fire, ocean and terrain rendering.

This Chapter presents the state of the art focused on the topics related to this thesis: vegetation modeling for real-time applications, level of detail scene management, global illumination techniques oriented to the foliage as well as on shadows for real-time applications.

2.1. Related work for plant modeling

Extensive research has been carried out offering real-time visualization of detailed plant species. To solve this problem, many approaches have been proposed that can be classified in two different groups: image-based and geometry-based algorithms. At a glance, image-based methods use less geometry and provide good results at a far-medium distance, while geometry-based methods are able to offer the best results at shorter distances.

2.1.1. Image-based rendering

This is one of the most common methods of representing trees because of its simplicity. Impostors are the most popular example of image-based rendering. In this method the geometry of the object is replaced with an

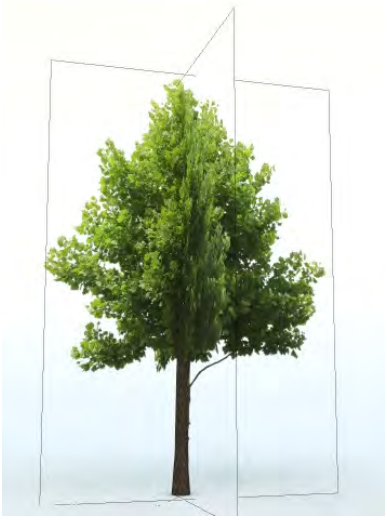


Figure 2.1: Tree approximated with two crossing billboards.

image of it textured on a polygon within the scene. See Figure 2.1 for an example of a billboarded tree.

This technique has been used in different works presented by several authors up until now. Max [Max96] extends this technique by adding depth information to the precalculated images. He precalculates multiple z-layers from a set of viewpoints using multiple orthogonal projections and stores colours, normals and depths for each pixel of each layer. That information is used later for reprojecting pixels, taking into account their location and depth, interpolating viewing directions for reconstructing the tree at any viewpoint. According to the author, popping artifacts arise when changing levels of detail. Although this technique performed properly on software renderers, it does not fit well on current GPUs due to the reprojections.

Later, Max et al. [MDK99] extend their previous hierarchical reprojection approach adding texture hardware support. They use the OpenGL color matrix for transforming the precomputed normals. In order to take advantage of the texture mapping hardware, they switch from using sorted lists of depths at each pixel to just storing a set of slices of the foliage.

In this context, Shade et al. [SSHS98] and Chang et al. [CBL99] introduce layered depth images (LDI) to render objects from pre-computed pixel-based representations with depth from different viewpoints. This information allows them to recalculate different views from the stored images of the scene. The advantage of this representation is that they are able to calculate intermediate representations of smooth surfaces without the gaps found in other methods. In a preprocess they store multiple pixels along each line of sight. This means a linear storage cost with the depth complexity of

Figure 2.2: Image-based forest rendering from [FMU05].

the scene, which is not efficient for forest scenes because of the high amount of depth layers in such scenes.

Other authors present solutions to this problem using the concept of volumetric textures. Volumetric textures are an extension of traditional two-dimensional textures that not only capture information of a model surface but are used to capture information of a whole volume or volumetric object. This techniques are often used to realistically render materials like marble or ice. However, this concept is not limited to that and can be extended to deal with smoke rendering and even forest rendering, as shown below.

Meyer et al. [MN98] present a technique for interactively rendering complex natural scenes such as landscapes. They convert complex natural objects into mipmapped volumetric textures before they are raytraced by slicing blocks of geometry. These slices are stored later into a series of thin layers that are used at rendering time as textures. They make use of transparent textures for representing complex objects like plants, trees and fur. Although this technique allows to efficiently render very complex objects and scenarios in real-time it presents some limitations. First, the volumetric approximation needed for representing dense and large forests would only allow to use this technique for rendering at far distances. Second, although the authors present a way to animate volumetric objects, it only works for simple animations that affect to large areas. However, for physical-based tree animations it is important to provide localized animations.

The works presented in 2001 by Meyer et al. [MNP01] and in 2004 by Reche et al. [RMD04] obtain 2D images from volumetric textures and combine them depending on the position of the camera. The former propose a

Figure 2.3: Forest created with SpeedTree (c) from Gothic 3 (c).

technique for rendering trees over a landscape with shading and shadows. They use sets of six-dimensional light fields that store the color of a leaf (or pack of leaves) given a viewing and lighting direction. This information is stored into a hierarchy that is traversed at rendering time for rendering the adequate node, depending on the viewing distance. The latter presents a method for reconstructing image-based representations of trees from real photographs of trees. Then they use a set of billboards at run-time for representing the tree. However, this results in low quality images for close-ups. [LRDM06] extend this work and estimate opacity in a volume to generate and visualize view-dependent textures attached to cells of that volume.

One of the main problems of these methods is that suffers from parallax artifacts that are visible at medium-close distances. Moreover, they are limited to the resolution of the textures used for date storage.

García et al. [GSSK05] [GP08] solve the parallax problem by using impostors that group sets of leaves and using indirect texturing to drastically increase the detail of the leaves without incrementing the memory footprint. They distribute a set of quadrilaterals (impostors) that approximate the volume of the foliage. Then each leaf is stored in the closest quadrilateral taking into account their position and coplanarity. The advantage of this method is that they use an indirect rendering mechanism that allows them to store leaves as single texels, which color indicates the orientation of the leaf. This tecnique allows for better utilization texture memory allowing for high quality leaf reconstructions, even at close distances. Later in [GP08] they improve their work with a layered approach that solves the problem of overlapping leaves.

Other works based on billboard clouds can be found in the literature.

These works are based on substituting the geometric representation of trees in a preprocess by billboards which always are rendered facing the viewing direction.

Décoret et al. [DDSD03] and Fuhrmann et al. [FMU05] present a technique for extreme simplification of models with billboard clouds (see Figure 2.2). Their algorithm distributes a set of quadrilaterals over the object volume and precalculates the viewing information from the highly detailed geometry. That way they are able to store the visual information of models with several thousands of trees by using only a few hundred textured polygons. Moreover, they are able to perform real-time illumination calculations by storing the normal maps along with the diffuse color into the billboard textures. An important drawback of this method is that models can not be animated because they are precalculated. Later they apply this technique to the rendering of trees [FMU05], allowing to render extremely large and dense forests interactively, presenting a new billboard cloud creation metric that takes into account the sparse nature of leaves.

Dylan et al. [LEST06] use a similar approach but use a stochastic distribution of billboards over the foliage. Although authors say that this algorithm is not suitable representing for continuous surfaces it perform well for sparse objects (like the foliage) allowing to represent detailed trees with less than 100 quadrilaterals.

Mantler et al. [MJW07] use the concept of billboard clouds and extend this technique by using boxes instead of quadrilaterals. They store depth information obtained using raycasting methods in a preprocess and use this information later performing displacement mapping for high quality reconstruction of complex models with just some dozens of triangles. Therefore, this technique provides better visual results than the standard billboard cloud approximation.

Finally, one of the most successful approaches in image-based representation of vegetation is developed by SpeedTree [Spe10], which was able to provide a full framework for tree and forest representation oriented to real-time applications such as games (see Figure 2.3). Nowadays this software is one of the most popular in the field of tree visualization in real-time. Dozens of commercial videogames have used this technology successfully because it greatly balances image quality, flexibility and performance. It uses a viewpoint oriented billboard representation with several levels of details depending on the viewing distance.

Summarizing, image-based methods are useful and efficient approaches for rendering forests at medium-far distances but present some disadvantages: they usually exhibit clustering and parallax effects at close distances and do not allow fot physically based animation of vegetation.

Figure 2.4: Trees generated with the XFrog editor. [XFr10].

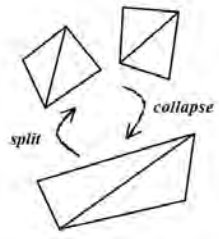


Figure 2.5: Leaf collapse operation introduced by [RCB+02].

2.1.2. Geometry-based rendering

Plant and tree rendering approaches based on geometry representations provide better visual quality at all rendering distances, compared to image-based methods, and do not lose realism as the viewer moving towards the object. However, the amount of primitives that form the trees makes it necessary to use certain techniques to obtain interactive visualization of large forests. Figure 2.4 shows some geometry-based trees modelled with the XFrog commercial application, rendered using an offline renderer photorealistic.

Due to the sparse nature of the foliage, which is composed of a set of disconnected leaves and not a continuous smooth surface, traditional simplification operations that are usually applied to polygonal surfaces can not be used for foliage simplification. Therefore, one of the main problems in geometry-based representation of trees is to develop appropriate simplification procedures that allow for removing detail from the foliage while maintaining perceptual visual appearance.

Figure 2.6: Geometric simplification of the folaige by [RCB+02].

For solving this problem Remolar et al. [RCB+02] introduce a new simplification operation specially designed for the foliage that allows for removing detail while minimizing the visual impact of the operation. They introduce the concept of leaf collapse as an adequate simplification operation for the foliage. A leaf collapse (see Figures 2.5 and 2.6) takes as input two leaves and substitute them with a newer bigger leaf that potentially fills the gap created by eliminating the former leaves.

This approach allows for effectively decreasing the amount of rendered geometry while maximizing the overall shape of the tree. Authors use this technique for constructing a multiresolution LoD model for the foliage for decreasing the amount of detail depending on the distance to the tree.

Weber and Penn [WP95] present an algorithm for the procedurally creation of trees, and although they do not use accurate botanical principles for modeling the tree they are able to construct high quality trees of several species. They also take count physical properties like wind sway for tree movement. Although they provide image degradation at range for increasing rendering speed at far distances, their rendering algorithm is outdated and does not take full benefit of current GPUs.

Some authors propose to use rendering methods based on points and lines so that, when the tree is rendered at a certain, it can not be distinguishable from using triangles for representing geometry. Following this line of research Stamminger et al. [SD01] propose to use a combination of points and lines instead of triangles for representing the foliage. They key advantage of using points instead of geometry is that it is faster to draw because less vertices need to be processed (in the case of complex geometry) and that it allows for a natural popping-free way of implementing level of detail depending on the viewing distance.

Following the idea of using points and lines for representing the foliage, Deussen et al. [DCSD02] present a method for rendering large natural scenes in real-time. They store a representation of the models using a hierarchical data structure that allows them to smoothly reduce the geometrical repre-

Figure 2.7: Representing plants with lines at far distances on [DCSD02].

sentation. Moreover, data reduction is driven by visual importance of vegetal species, allowing for maintaining the visual fidelity of the representation even drastically reducing the amount of geometry.

Gilet et al. [GMN05] propose to use a combination of triangles and points for efficiently rendering vegetal species. When the tree or block of plants are close to the observer, they are rendered at full level of detail using triangles to represent leaves and branches. However, as the models move away from the viewer triangles are progressively replaces by points which size increase with the viewing distance. As the distance increases the amount of points needed for rendering decreases but their size is increased for maintaining visual appearance. The advantage of using points is that it is easy to decrease the level of detail just by removing points from the soup.

Cook et al. [CH05] present Stochastic Pruning as a powerful operation for extreme simplifications of sparse geometry such as the foliage. Later, the same authors revised their work [CHPR07] by extending the idea to the simplification of aggregate detail, which uses the same approach of stochastic pruning to non-vegetal species, such as crowds. The idea behind this works is to progressively eliminate primitives from a complex polygonal representation using a stochastic metric. According to the authors, that approach enables them to rapidly and easily discard potential unseen geometry after performing visual modifications to the remaining geometry.

In recent years, several papers based on multiresolution LoD models have appeared. Some of them work with multiresolution models of images, such as the work presented by Meyer et al. [MNP01] and Lluch et al. [LCV03]. Most of LoD models are based on geometry, as the work presented by Remolar et al. [RCRB03][RRCR04]. In that work authors present a multiresolution representation of a tree based exclusively on isolated polygons. They can

represent different resolutions in a same tree following a view-dependent approaching. Even this algorithm is able of effectively reducing foliage complexity, they perform the LoD extraction process on the CPU, which is a great penalty when using massive forest scenes.

In 2006, Rebollo et al. [RRCR06] improved this representation by adapting data structures to the graphics hardware. The next year, the same authors present an improved version [RRC+07] of this work which uses a GPU-oriented storage for multiresolution data so that it can be efficiently rendered. In that work, they divide the foliage into clusters and treat them like independent LoD models with their own buffers on the GPU. The advantage of this approach is that performing simple changes in the LoD do not force to update the vertex buffer of the whole tree but just a part of it. They also propose to use a multiresolution model for the trunk based on triangle strips [RC04]. However, although this work uses an efficient rendering approach, the LoD extraction step is still performed on the CPU.

Rebollo et al. [RGR+07] propose a new approach for fast foliage simplification on the fly and negligible extraction cost. The article is based on generating simplified leaves selecting different vertices based on a *vertex-skipping* approach. Although this technique performs the LoD extraction step really fast at constant time, it is not able to preserve foliage appearance on high compression ratios.

Finally, [DZYJ10] presents a new multiresolution model for the foliage that allows for high compression ratios. Their approach is based on recursively collapsing pairs of leaves and finally replacing them by other primitives, as lines or points. These collapses are precalculated in a preprocess by subdividing the foliage using a binary tree scheme for rapidly finding pairs to simplify.

2.2. Related work for Scene Management

The growth in computing capabilities that the graphics hardware experienced in the last decade allow developers to build more and more complex scenes. Real-time rendering applications usually make use of LoD techniques for improving performance in such scenarios, introducing the idea of scene manager which takes control of individual renderable models and takes decisions about their rendering behaviour.

Funkhouser and Séquin [FS93] demonstrated that it is necessary to use a predictive selection scheme, based mainly on the complexity of the current frame, rather than a reactive framework, based on the feedback obtained. They formulated this problem as an optimization task which is equivalent to a constrained version of the Multiple Choice Knapsack Problem. Even

though this problem is NP-complete, some authors like [FS93] or [MB01] obtained several techniques that could only guarantee a solution that is at least half as good as the optimum one. [WS98] reconsidered this problem for the special case of continuous multiresolution models, obtaining a non-iterative closed form solution which was cheap to evaluate every frame.

This way, the problem of the time-critical multiresolution rendering can be presented as an optimization problem for finding the LoD that maximizes the scene quality under timing constraints. Funkhouser and Séquin [FS93] developed a generalization of the predictive approach, using approximate heuristics of the cost and the quality obtained that were efficient and accurate enough to obtain the best image possible within the target frame time. The work in [GB99] extended the use of predictive techniques with more precise heuristics for the cost and the benefit of the resolution of the objects. It also considers but not includes temporal coherence to minimize sudden changes. These optimizations are very accurate but costly, and as they assign one variable for each object, rendering scenes with a large number of objects tends to be a slow solution.

All the previous works have applied static, feedback of predictive heuristics. But, in all cases, a criterion to select the most adequate level of detail must be used. This way, it is possible to use the size, the speed, the position in the scene, etc. Many authors have addressed the necessity of investigating how the human perception system works. [Red94] considers the necessity of including an analysis of the human visual system to understand how it works and to offer more adequate results, extending his results in his subsequents publications. In this sense, several authors have included biometrics into their heuristics, considering spatiotemporal sensitivity [YPG01] or developing frameworks with eye tracking as the basis [DDGM00].

Other authors have addressed this problem from different points of view. The authors of [RL00] use a multiresolution hierarchy based on bounding spheres with a rendering system based on points specially designed for 3D scanned models with a great geometric complexity. They perform the LoD selection based on the projected size in the screen, and adjust the threshold from frame to frame. They also gradually refine the model when the viewpoint is not moved for a period of time. The most novel aspect of [IAN02] is the use of a distributed rendering architecture to obtain a stable frame-rate. A group of researchers have presented the concept of interruptible rendering [WLWD03] to find a rational compromise between spatial and temporal detail, producing a complete image on the back buffer almost immediately and then incrementally refining it so that the refinement can be interrupted at any time. Zach [Zac02] presents a solution based on geomorphing where the LoD management is achieved by distributing the LoD selection and caluclation between several frames, reusing the old resolution until the new one is

ready. As the new LoDs will appear in future frames, they need a path prediction process to obtain future viewpoints as directions. They also use cost and benefits computation, but include some feedback strategy to compensate for some assumptions they make. These authors extended their work in [ZMK02], presenting an approach for discrete and continuous models where the time spent for LoD selection is amortized over several frames.

2.3. Related work for Illumination of plants

Realistic illumination is an important factor that provides visual quality to synthetic images. Although realistic illumination can be achieved by simulating (or at least highly approximating) the physics of light transfer and its interactions with the environment, that would be very inefficient even for offline rendering. Therefore, several authors have proposed uncountable solutions to this problem and how to efficiently compute global illumination for realistic visualization. More specifically, there are some methods to simulate global illumination in real time, such as Precomputed Radiance Transfer [SKS02]. This method uses spherical harmonics to capture low frequency illumination scenarios (including soft shadows and interreflections of objects). On the other hand [SSBD03] uses a geometry instantiation system and precise phase functions for hierarchical radiosity in botanical environments.

Mendez et al. [MSC03] introduce Obscurances as a method to simulate diffuse illumination by considering neighbor light contributions instead of the global ones. Ambient Occlusion [PG04] enhances the illumination of an object by determining the light visibility of each part of the object in a way that the most occluded is an object point the lesser light it will receive from the exterior. Other authors [Bun05] adapted [PG04] to the GPU so that the ambient occlusion is computed directly in the fragment shader. Another approach for real-time illumination of trees is [HPAD06]. In this work, ellipsoidal occluders that describe the shape of the tree are evaluated at run time.

Reeves and Blau [RB85] present a tree rendering algorithm which also takes into account lighting and shadowing. The method is based on particle systems. The relative position of each particle inside the tree is used to approximate the illumination and shadowing at a given point.

Jensen et al. [JMLH01] present a new model for subsurface light transport which is useful on translucent objects such as leaves. Franzke et al. [FFD03] introduce an accurate plant rendering algorithm using [JMLH01] as a leaf illumination method and improving it for leaf rendering.

Finally, [LBO07] presents an expressive illumination technique for fo-

liage. It calculates implicit surfaces that approximate the general shape of the foliage. The implicit surfaces are used both for estimating the global illumination coefficient at a given point and for realigning leaves normals to calculate the diffuse reflection.

2.4. Related work for Shadowing

Although shadows are created by the lack of light and then their representation can be realistically calculated using physics of light, in computer graphics they are usually treated as a separate element in the scene for the sake of rendering efficiency. Realistic shadow generation is a 'hot' topic in real-time graphics because it highly contributes to the visual quality of synthetic images and they have been under research since the dawn of computer graphics.

A very straightforward way of calculating shadows is by using ray casting to test whether light rays hit a point or not. However, due to its computational cost this method is not well suited for real-time computer graphics and then alternative methods have appeared.

Williams introduced shadow mapping for general meshes in 1978 [Wil78]. Although this method is highly suitable for the graphics hardware, its main drawback is aliasing and its memory consumption. Thus lots of authors have suggested their own approaches to solve this problem.

Adaptive Shadow Maps (ASM) [FFBG01] reduces aliasing by storing the shadow map as a hierarchical grid. This allows us for huge memory savings, but it is not graphics hardware friendly, because of its hierarchical structure. Arvo [Arv04] proposes to use a tiled grid data structure to tessellate the light's viewport, as a simplified version of ASM. Each cell in this grid contains a sampling density depending on a heuristical analysis.

There are some perspective parametrizations to maximize the area occupied by shadow casters if they are near the observer. This allows for rendering high quality shadows near the camera at the cost of loosing detail, but not quality, on points that are far away from the observer. The most representative shadowing methods that use this scheme are [SDD03][WSP04][MT04]. Parallel Split Shadow Maps [ZSXL06] use a similar approach, but it treats the continuous depth range as multiple depth layers. This allows us to utilize better the shadow map resolution.

As seen, there are no few methods around this topic, but there are no specialized shadow casting methods for foliage that takes into account the leaves structure and its spread nature.

A number of authors have developed techniques for the real-time genera-

Figure 2.8: Importance of perceptually correct shadows: shadows with no penumbra (left), with uniform penumbra (middle) and with variable penumbra rendered with our method (right). Notice how the penumbra becomes sharper as the shadow approaches the occluder and the quality of self-shadows compared to uniform penumbra methods.

tion of penumbrae. These methods can be classified into two groups: uniform and variable-sized soft shadows. Uniform soft shadows are faster to compute but variable-sized penumbrae methods are able to generate more realistic shadows (see Figure 2.8) as they require to perform costly operations.

In 1987, Reeves [RSC87] presented a technique called percentage-closer filtering (PCF) which makes it possible to reduce the aliasing and to simulate an artificial penumbra of uniform size. Haines [Hai01] provides a method to render realistic penumbrae with the limitation that the shadow receiver must be a plane. Chan et al. [CD03] and Akenine-Möller et al. [AMA02] are able to render a realistic penumbra. However, as they use the geometric silhouette of the shadow caster, they have similar limitations to those found in shadow volume approaches. Hasenfratz et al. [HLHS03] presented a survey of techniques for the rendering of the penumbrae, which was updated in 2008 by other authors [Bav08].

Some authors approximate the contents of the shadow map in such a way that it can be filtered like a regular texture, reducing the aliasing. To do so, Donnelly et al. [DL06][LM08] store the mean and the squared mean of the distribution of depth to calculate the variance in real time. Annen et al. [AMB+07] approximate the depth function using a 1D Fourier expansion which provides a good filtering of the shadow but introduces some ringing. Furthermore, Annen et al. [Sal07] introduced a new technique for filtering the shadow map by approximating the step function of depths with an exponential function.

These methods can be used to reduce aliasing and to generate a fixed-sized penumbra. However, they are not able to produce realistic variable-sized penumbrae.

Brabec et al. [BS01] uses a hardware compliant version of [PSS98] which is able to recreate visually correct penumbrae, although it is not physically correct. Schwarz et a. [SS07] is able to represent quality physically-based soft shadows in real time, at the expense of speed.

F. Randima [Fer05] introduces Percentage-closer soft shadows (PCSS) which is able to represent shadows with variable-sized penumbrae. This technique is usually combined with a filtering technique like [DL06][LM08] or [Sal07] to reduce the noise. Guennebaud et al. [DF94] are also able to generate accurate high-quality penumbrae of variable size, however the complexity of the process involved makes this method much slower than the others. They interpret the shadow map as a 3D representation of the scene and back-projects its texels to determine the amount of light which is visible at a given point.

Finally, Anne et al. [ADM$^+$08] present a realistic shadowing approach based on convolution shadow maps [AMB$^+$07], which filters the shadow map in light space by using convolution operations.

Shadow mapping is a widely used hardware friendly method for computing shadows in real-time scenes. However, although it is a very efficient method that scales well, it produces aliasing due to the texture-based nature of the algorithm. Aliasing can be reduced by two orthogonal approaches: projection optimization and shadow filtering [SWP10]. The former deals with how shadow caster objects are projected over the shadow map in order to optimize texture space for important parts of the scene [WSP04][MT04][SDD03]. The latter deals with how the shadow map is filtered in order to reduce aliasing. An important difference between them is that projection optimization is used during the shadow map creation time, while the shadow filtering is applied when rendering the scene with the already created shadow map by deciding how to interpret the data. Both lines of research are orthogonal but complementary as techniques of both fields can be used together to provide efficient anti-aliasing.

Our method can be classified in the field of shadow filtering approaches. There are several works in the literature for dealing with this problem. One of the first methods introduced to alleviate aliasing in shadow mapping is *percentage-closer filtering* (PCF)[RSC87] which is able to filter the shadow map by averaging the outcomes of depth comparisons against the shadow map, instead of the depth values themselves. In fact, this is a "post-filtering" method as it applies averaging after the non-linear depth test. However, this means that filtering can be executed only when the distance of the shaded point is available, so it should be repeated for every shaded point, which makes PCF expensive when large filter sizes are used. The cost of PCF can be reduced by moving the filtering operation before the depth comparison. Such pre-filtering methods belong to two main branches, those that apply

depth transformation, and those that are based on statistical analysis.

Convolution shadow maps [AMB+07] apply depth transformation and approximately express the visibility function in a product form $\epsilon(z_o - z_r) \approx \sum_i g_i(z_r) \cdot h_i(z_o)$ and then linearly filter the $h_i(z_o)$ factors. This method allows for anti-aliasing by representing the depth distribution with respect to a basis which allows for linear filtering. Convolution shadow maps do not scale well when the number of coefficients increases as all of the basis coefficients need to be sampled for a given texel of the shadow map. *Exponential shadow maps* (ESM) [Sal07] approximate the shadow test using a single exponential function. This technique allows for both shadow map pre-filtering and hardware accelerated filtering, although it still presents artifacts, for example, its shadows are unrealistically light when the occluder is far from the shadow receiver.

The most important representative of statistics based shadow filtering is the method of *variance shadow maps* (VSM) [DL06]. The variance shadow map method stores the depth and the squared depth of the shadow casters. These values are filtered as a regular texture. The filtered values are used at rendering time to calculate the first two moments M_1 and M_2 of the depth values over the shadow filter region. Then, the one-tailed version of the Chebyshev's Inequality allows us to approximate an upper bound of the probability of shadowing if receiver depth z_r is greater than the *mean depth* $\tilde{z}_o$:

$$P(z_o \geq z_r) \leq \frac{\sigma^2}{\sigma^2 + (z_r - \tilde{z}_o)^2}, \tag{2.1}$$

where $\tilde{z}_o = M_1$ is the average of the depth values and $\sigma^2 = M_2 - M_1^2$ is their variance. If receiver depth z_r is greater than the *mean depth* $\tilde{z}_o$, then the variance shadow map method approximates the visibility function by this upper bound of probability $P(z_o \geq z_r)$:

$$v(z_r) = \frac{\sigma^2}{\sigma^2 + (z_r - \tilde{z}_o)^2}. \tag{2.2}$$

If the receiver depth is smaller than the mean depth, we assume that the surface is fully lit and thus $v(z_r) = 1$.

Variance shadow maps are an efficient hardware friendly method whose performance scales well with the screen resolution. However, artifacts appear when the scene is at least moderately complex from the light's point of view. A typical problem occurs on parts of objects that are completely occluded but some amount of light still leaks inside the shadows (see Figure 2.9).

Actually, light bleeding artifacts are introduced by the fact that there is not enough information to disambiguate all the possible cases correctly.

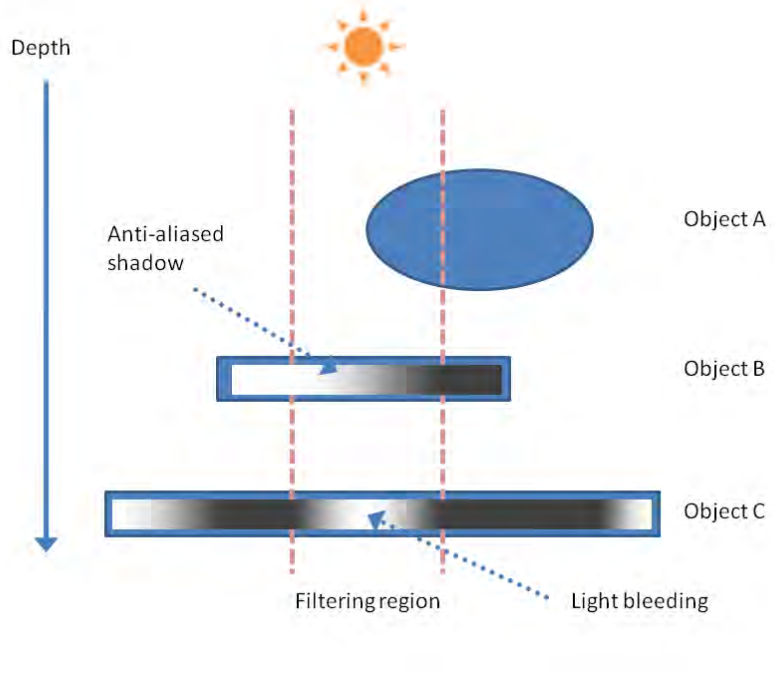


Figure 2.9: A typical case where light bleeding artifacts appear. Although object C is completely occluded by object B, it still presents some penumbra from object A, which is "bleeding" incorrectly.

However, as stated by [LM08], storing more moments in the shadow map for evaluating the visibility function would not solve the problem, because higher-order moments are numerically unstable.

Layered variance shadow maps (LVSM) [LM08] is an evolution of VSM developed for solving the "light bleeding" artifacts. LVSM divides the light's depth space into multiple layers which allows for a correct filtering of the shadow map. By using this technique we can obtain different upper bounds for $P(z_o \geq z_r)$, some tighter than others. When rendering the shadows, this allows for selecting the appropriate warp in which the light bleeding artifacts are less visible or even eliminated. This technique introduces the problems of selecting the number and the optimal placement of the warps. These layers are distributed by using an automatic method based on the Lloyd relaxation algorithm. Although performance decreases and the storage cost increases as more layers are used, it still can perform the shadow filtering with just a single access to the shadow map.

Our algorithm is a statistical method and allows for eliminating (or at least highly reducing) light bleeding artifacts. Simply put, our algorithm replaces the Chebyshev's Inequality by a Gaussian or a power function ap-

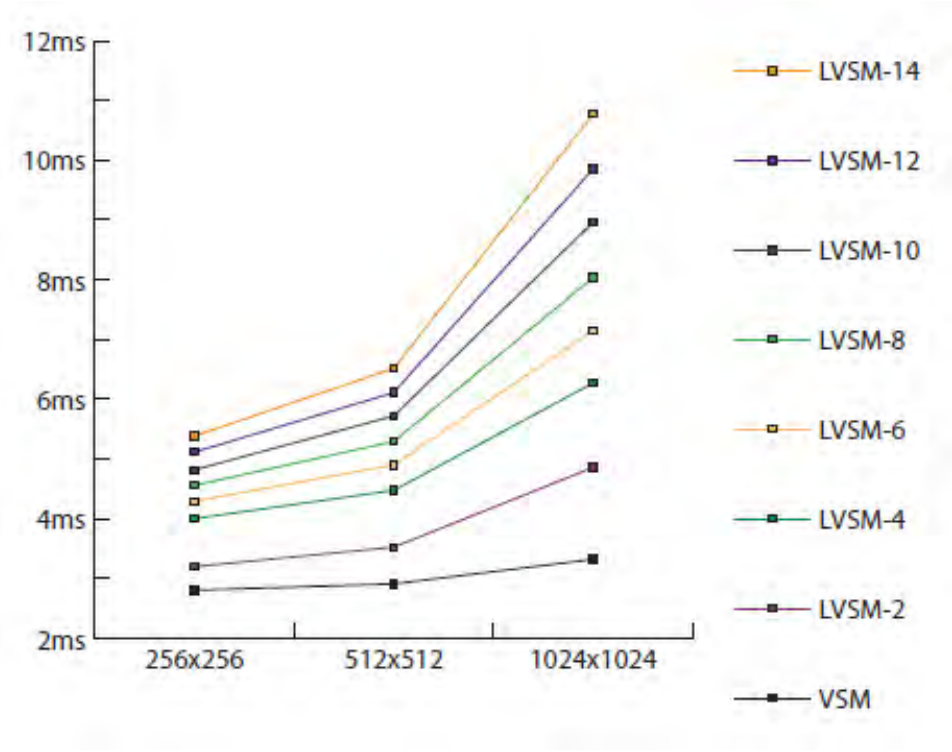


Figure 2.10: Performance cost of using layers.

proximation which are able to approximate the depth distribution more accurately. While our algorithm highly reduces the "light bleeding" artifacts, it cannot completely eliminate them for complex scenes. In these cases, we can use a layered approach (just like LVSM) to completely eliminate artifacts with a lower cost as much fewer layers need to be used.

CHAPTER 3

View-dependent pruning for real-time rendering of trees

The main problem in the real-time rendering of vegetation is the massive amount of primitives to be rendered. These primitives are needed to fully describe the geometry of the plants. However, some of them are not visible depending on the location of the viewer. This dissertation chapter focuses on this fact to interactively reduce the amount of geometry needed to represent the foliage through a view-dependent multiresolution scheme. Following a camera-dependent criterion, the less visible parts of the foliage are detected in real time, and rendered with a decreased level of detail for improving efficiency. This fact considerably reduces the extraction and visualization time of the geometry that represents the foliage. The novelty of the presented method is that its design is oriented to being efficient on massively parallel architectures, such as the graphics processing unit.

3.1. Introduction

Efficient rendering of vegetal species is a key feature for enhancing the realism of outdoor scenes. However, realistic visualization of plants in a natural environment has always posed a challenging problem, due to the massive amount of geometry needed to represent a plant. This is especially true in dense forests, where the massive amount of primitives can easily overwhelm the most advanced rendering system available.

Multiresolution level of detail -LoD- models [RLB+02] are well known

methods for altering the polygonal complexity of objects in order to improve performance in highly detailed meshes. The basic idea behind these methods is that highly detailed models do not always need to be represented at full detail. Multiresolution models alleviate this problem by diminishing the amount of triangles in a progressive way, so that the viewer does not perceive great changes while decreasing the level of detail.

Every LoD scheme is based on a simplification method in order to construct the data structure. The appropriate geometry, depending of some criteria, is extracted in real time. Usually, LoD schemes are designed for continuous surfaces. This is the reason they do not work properly with the representation of the foliage, because of the characteristics of their geometry [DCSD02]. This part of the plants, which is composed of sparse non-connected geometry, cannot be optimally simplified using standard simplification methods, as edge collapses. The multiresolution models specifically designed for trees usually introduce their own simplification scheme adapted to the nature of the foliage, such as those based on leaf-collapses [RCB+02] or others based on pruning [CH05].

This chapter presents a new view-dependent multiresolution model for the foliage of the trees that takes advantage of the graphics hardware. In order to build the LoD scheme, a stochastic pruning method [CH05] [CHPR07] is applied in a preprocess, which is proved to deal correctly with sparse meshes, such as the foliage. This fact allows us to interactively remove unneeded data. Moreover, the presented multiresolution scheme provides a view-dependent solution. In real-time, the less visible parts of the foliage are detected and rendered with a coarser approximation in order to improve efficiency. The appropriate resolution of the foliage is calculated taking into account both the distance of the tree to the observer as well as the visibility of the leaves. Our algorithm is designed to run efficiently on highly parallel architectures, such as the graphic process unit (GPU). Due to the design of the scheme, the level of detail can be calculated in parallel, so that the extraction time is considerably reduced.

The rest of the chapter is organized as follows. Next, we present an overview of the method in Section 3.2. In Section 3.3 the processes involved in the construction of the presented LoD scheme are described and run-time processes are detailed in Section 3.4. Section 3.5 deals with the implementation details and the level of detail manager for forest rendering are explained in Section 3.6. Next, we discuss the differences between our approach and other existing techniques in Section 3.7. Then, we show and discuss the results in Section 3.8. Finally, conclusions are presented in Section 3.9.

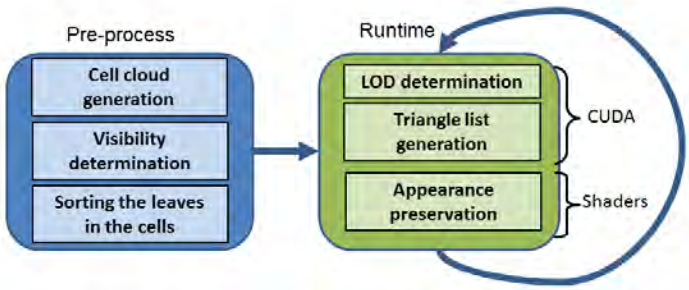


Figure 3.1: Scheme of the presented method.

3.2. Method Overview

This dissertation chapter proposes a new view-dependent geometry-based multiresolution algorithm for foliage rendering specially designed for highly parallel architectures. The algorithm takes into account the distance of the foliage to the camera, as well as its relative position, in order to provide a view-dependent level of detail solution. The presented scheme is oriented to trees which leaves are represented by two triangles in a quad.

The algorithm is composed of two different stages as it is illustrated in Figure 3.1. Firstly, there is a preprocess step which prepares the input data (a mesh representing the foliage) for constructing the view-dependent approach. This pre-process performs the following operations. First, foliage is divided into a cloud of cells, represented as oriented bounding boxes (OBBs). Next, the visibility of each cell is computed from a set of external viewpoints surrounding the foliage (Figure 3.2). Finally, leaves inside each cell are stochastically sorted [CH05] in order to perform the simplification operation at run-time.

Secondly, there is a run-time stage which includes the algorithms needed to interactively alter the level of detail. First, each cell evaluates in real-time the position of the camera in order to decide the LoD factor associated to it. LoD factors are used as a percentage value for deciding the amount of leaves needed to represent the contents of a cell. Next, based on this factor, a list of triangles that constitute the visible leaves is generated for rendering, selecting them from the original set. Finally, the size and the color of the remaining geometry are altered in order to preserve the visual appearance of the original mesh.

From an implementation point of view, performing all these operations on different independent cells means the LoD extraction algorithm is parallelizable, and thus GPU-friendly. The direct benefits of implementing it on

Figure 3.2: Example of a distribution of cameras surrounding the foliage.

the graphics hardware are that the traffic between the CPU and the GPU is minimized and the time needed to perform the needed calculations is drastically decreased, because of the parallel nature of the domain. Therefore, this method is able of taking advantage of both processing power of the GPU and the wide bandwidth of on-board graphics memory.

3.3. Pre-process

In order to build the data structure, the first step is to generate a cloud of 3D cells over the foliage, so that every leaf in the foliage is clustered in a cell. Next, the visibility of each cell is tested from different angles from the exterior of the foliage. Thus, each cell is bound to a value that determines the visibility of the leaves it contains from a set of viewpoints located around the foliage (see Figure 3.2). Finally, a stochastic number is assigned to every leaf in the cell and the leaves are re-organized using this number, following the work presented in [CH05].

3.3.1. Cell cloud generation

The first step is to generate a cloud of cells around the foliage. In this process, it is important to generate the cells taking into account the shape of the foliage and the distribution of the leaves. The objective is to maximize the number of leaves contained in each cell while minimizing the size of the cells needed to be spread over the foliage. For that purpose, we use the method introduced by Gottschalk et al. [GLM96] which generates a cloud of

oriented bounding boxes that are fitted as tightly as possible to the shape of the object. The number of the cells in the foliage will determine the number of processes performed in parallel in the GPU.

For the cell cloud generation, we implement an OBBTree structure which allows us to generate a tree of tighly packed cells distributed over the foliage. This OBBTree allows us to obtain a cell cloud of a variable number of cells by simply selecting all the cells of a given maximum depth.

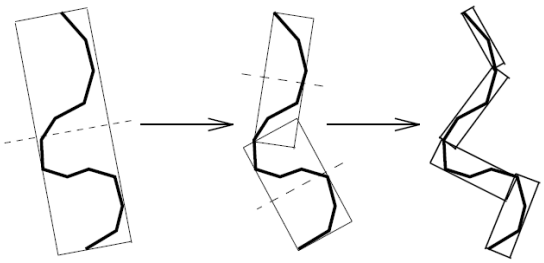


Figure 3.3: Procedure for building the OBBTree by recursively partitioning the bounded polygons.

For constructing the OBBTree, first the bounding volume of the whole foliage is computed. Then, a process recursively partitions the bounding volume, using the major axis criterion, and builds a tree of a given maximum depth. Figure 3.3 shows this recursive process. When the tree is computed, the resulting cell cloud is selected by traversing the tree and selecting the leaf nodes. Figure 3.4 shows an example where the cell cloud is computed from an input tree by using an OBBTree.

The number of cells generated in the foliage is the criterion used for parallelizing our LoD algorithm. In the GPU, each kernel execution processes the level of detail of each single cell in parallel. In our scheme, the amount of cells determines the softness of the view-dependent approximation because having more cells means a finer granularity. Therefore, the more threads in parallel, the better visual results obtained. In this scheme, we decided to setup our kernels to use 512 threads per block, the maximum amount of threads per block in CUDA. Then, in this preprocess, OBBTree is generated which has 512 cells as leaves of the tree data structure. All the leaves are included in one of these cells distributed around the foliage.

3.3.2. Visibility determination

After performing the previous step, the visibility of each cell is computed in order to provide a visibility factor that determines the visibility of the leaves it contains from a set of external cameras. These cameras or view-

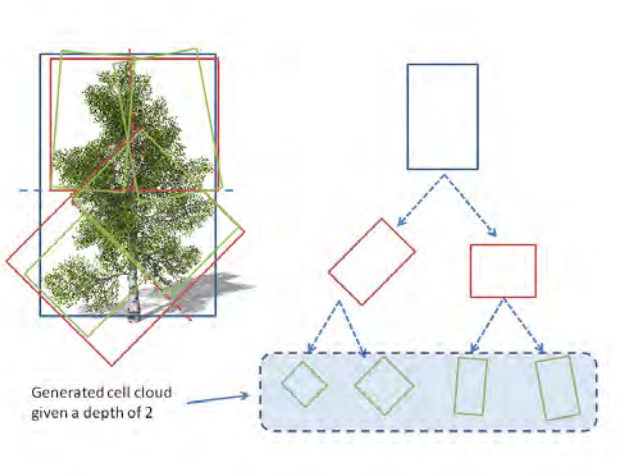


Figure 3.4: Example for illustrating the 3D-cell cloud generation process using an OBBTree.

points are uniformly distributed around the foliage in order to capture the shape of the foliage from a finite number of directions, as shown in Figure 3.2. In order to decide the amount of viewpoints and their distribution around the foliage, we took into consideration the following works from Lindstrom et al. [LT00] and Castelló et at. [CSCF08]. The good results obtained in their works using uniformly distributed viewpoints led us to the decision to keep this criterion for distributing the cameras around the foliage. Moreover, they suggest in their work that using more than twenty cameras to perform a image-driven simplification does not provide more accurate information. In our tests, we have found that usually the number of necessary pre-cached cameras is a small value. After various experiments, we tested that 16 cameras were enough to capture the general shape of the foliage in order to provide good estimations.

The visibility factor is a function of the type $vis(cell, camID)$ which associates each cell/camera pair with a floating-point value in the range $[0, 1]$. This factor determines how much of the foliage it contains is visible from a given viewpoint. Cell visibility is computed, for every cell/camera pair, by taking into account the number of pixels of the leaves in this cell that are not occluded by the rest of the foliage. The cost of this pre-process is $O(n_{cells} \times n_{cameras})$, with n_{cells} being the number of cells in the cloud and $n_{cameras}$ the number of cameras located around the tree.

Our implementation uses hardware occlusion queries to obtain the number of pixels that pass the z-buffer test. This part of the process is done on the GPU. Algorithm 1 clarifies the process involved in calculating the visibility values.

$$vis(cell, camID) = \frac{visPixels}{totalPixels} \in [0, 1] \tag{3.1}$$

Basically, for each viewpoint, $camID$, the contents of each cell are rendered four times. The first and second passes are used to obtain the number of pixels visible from the current viewpoint without occlusion from other cells: the first pass is used to set up the depth buffer and the second one is used to obtain the number of visible pixels ($totalPixels$). The other two passes are used to calculate the number of non-occluded pixels visible from the current viewpoint: the third pass renders all cells to fill the depth buffer and the last pass renders the current cell again to count the number of pixels that pass the depth test and thus the number of pixels which are visible from the current viewpoint ($visPixels$). Therefore, the visibility factor for each cell-camera pair is calculated using Equation 3.1.

Algorithm 1 Cell visibility determination on the GPU

```
cells ← ListofCells
leaves ← ListofLeaves
cameras ← ListofCameras
for all ce ∈ cells do
  for all ca ∈ cameras do
    clearBuffers(color, depth)
    renderLeavesCell(ce, ca)    ♯firstpass
    resetQuery()
    renderLeavesCell(ce, ca)    ♯secondpass
    totalPixels ← queryRendered()
    renderLeavesAll(ca)         ♯thirdpass
    resetQuery()
    renderLeavesCell(ce, ca)    ♯fourthpass
    visPixels ← queryRendered()
    visCellView[ce, ca] ← visPixels / totalPixels
  end for
end for
```

3.3.3. Cell-based stochastic sorting of leaves

Once the leaves are clustered in cells, they are sorted following the criterion introduced by Cook and Halstead [CH05]. A stochastic criterion is used to assign a random number to each leaf, which determines the order of simplification of each leaf in the cell. This process makes it possible to optimize the rendering of models made up of a large amount of disconnected geometry, such as plants and trees.

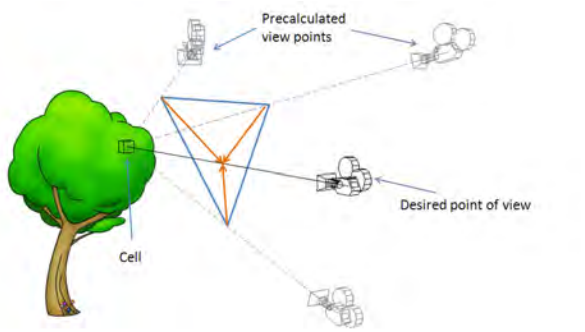


Figure 3.5: The visibility factor of the current viewpoint is obtained in real time by interpolating the visibility factors of the three nearest stored points of view.

In order to make this simplification process easier, leaves are stored in the GPU taking into account this random number. Each leaf is represented by two triangles, whose indices are finally stored in the graphics hardware. This fact allows for optimal performance performing memory accesses.

3.4. Run-time

This section describes the process performed at runtime on the GPU in order to generate and visualize the appropriate level of detail. Three stages are clearly differentiated: LoD determination, triangles list generation and appearance preservation.

3.4.1. LoD determination

In order to prevent unneeded geometry from being rendered, a level of detail factor is calculated for each cell (LoD_{cell_factor}) to determine in real time the appropriate level of detail of the geometry contained in every cell. In order to obtain a view-dependent approach, we have implemented in this scheme a function of the type shown in Equation 3.2. It depends both on the visibility of the cell in the current situation of the camera and on the distance of the object to it. However, this function can be easily adapted to different requirements of the scene.

$$LoD_{cell\_factor}(viewLoD(cell), dist(near, far)) \in [0, 1] \tag{3.2}$$

Firstly, the visualization of one cell, $viewLoD(cell)$, is interactively determined by using the pre-calculated LoD factors of the cell in the pre-process. For every cell, the visibilities of the three closest viewpoint directions are

linearly combined and weighted, to approximate an estimation of the current visibility (Figure 3.5).

$$viewLoD(cell) = \sum_{k=1}^{3} camWeights_k \cdot \ vis(cell, camIDs_k) \tag{3.3}$$

where $camWeights_k$ and $camIDs_k$ represent the importance and the unique identifier of the selected camera k, respectively (see Algorithm 2 for details).

Secondly, the function $dist(near, far)$ maps the relative positions of the tree ($treePos$) and the observer ($camPos$) into a value in the range $[0, 1]$. It takes into account two user-defined values which represent the distances for minimum and maximum LoD ($near$ and far planes). This function is defined in Equation 3.4.

$$\begin{aligned} |camPos - treePos| < near \rightarrow \ & dist(near, far) = 1 \\ |camPos - treePos| > far \rightarrow \ & dist(near, far) = 0 \\ & else \\ dist(near, far) = 1 - & \frac{|camPos - treePos| - near}{far - near} \end{aligned} \tag{3.4}$$

Finally, the appropriate LoD (LoD_{cell_factor}) is determined by using both the visibility of the cell and the distance to the camera and is calculated as follows:

$$LoD_{cell\_factor} = dist \cdot \ viewLoD \tag{3.5}$$

These parameters cause the system to behave as a variable multiresolution model because the level of detail depends on both the position and the orientation of the observer. Notice that setting $viewLoD(cell)$ to 1 for each pair $(cell, camera)$ causes the system to behave as a uniform multiresolution system which only depends on the distance to the viewer.

At the end of the process, one LoD_{cell_factor} is obtained for each cell around the foliage. These values are calculated in parallel on the GPU and stored in a buffer located in video memory. The storage cost of this buffer is $O(n_{cells})$, with n_{cells} being the total number of cells distributed around the foliage.

3.4.2. Triangles list generation

The objective of this process is to obtain the leaves needed to visualize the foliage at a given level of detail. All the cells are processed taking into account the LoD_{cell_factor} previously calculated. As result of this process, a list of triangle indices that represent the leaves in the current approximation is generated.

As leaves have been previously ordered using the stochastic number in the pre-process, outputting a certain level of detail is accomplished by just copying into the render buffer the first $(n_{leavesCell} \cdot \ LoD_{cell\_factor})$ leaves contained in each cell, where $n_{leavesCell}$ is the amount of leaves in that cell that represent the best approximation.

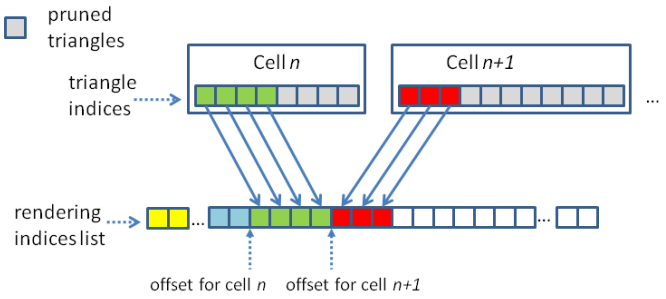


Figure 3.6: Unpruned leaves of each cell are determined and stored into a sequential list used for visualization. Offsets are needed in order to avoid collisions.

Taking advantage of the graphics hardware, the resulting indices list is generated for all cells simultaneously. For this reason, it is necessary to determine, for each one of them, the offset position in the resulting index buffer to start writing indices to. This fact avoids collisions writing to the buffer, as it is shown in Figure 3.6. Let o_n be the offset position for cell n and u_n the amount of unpruned triangles of cell n, o_{n+1} is calculated as follows:

$$o_{n+1} = o_n + u_n \tag{3.6}$$

It is important to notice that calculating a valid offset for a cell requires the sum of the offsets of previous cells. Although this problem seems to be inherently sequential, there exist some works in the literature that deal with this problem and how it can be efficiently implemented in parallel systems using the *all-prefix-sums* operation described in [Ble90]. More information about the algorithm used in this chapter can be found in [Har07] and [HSO07] for a detailed description of the algorithm and its efficient CUDA implementation.

Once cell offsets are calculated, the system is ready to start generating indices. Finally, for each cell a global variable is incremented in video memory to indicate the total amount of indices generated. This information is necessary in order to know the amount of geometry finally generated in the LoD extraction process for rendering purposes.

3.4.3. Appearance preservation

The runtime modifies the area of the rendered leaves in order to reduce the visual impact of pruning. Thus, the visual appearance remains the closest possible to the original unpruned model. In this LoD scheme, the technique presented in [CH05] has been applied. The total area of all leaves of the object can be expressed as Equation 3.7, where a is the average area of each single leaf and n is the number of leaves in the most detailed representation.

$$area_{total} = na \tag{3.7}$$

Let u be the parameter in the interval $[0, 1]$ that quantifies the amount of geometry that remains after the pruning process is applied to the foliage. This parameter takes into account the distance of the object to the observer (z). The function is defined in Equation 3.8. Let h be the parameter that controls the aggressiveness of the pruning function.

$$u = z^{-log_h 2} \tag{3.8}$$

Therefore, nu is the number of leaves in the current level of detail and nua is the total area of the unpruned foliage in this approximation. As pruning decreases the number of leaves, the area of the foliage also decreases. It must be compensated in order to maintain the visual concordance between levels of detail. Then, rendered elements are scaled by the factor s to compensate the pruning of primitives,

$$area_{total} = (nu)(as), s = 1/u \tag{3.9}$$

In practice, this step is performed in the vertex shader and has an almost negligible rendering cost.

When the tree is rendered at a medium distance it happens that leaves are so small that the texture on them is no longer distinguishable. As it has been previously said, leaves are represented by a quad textured by an image with alpha channel for opacity. Due to the texture sampling performed by the graphics hardware on this kind of textures, when they are so small, lots of pixels receive an incorrect averaged alpha value which causes the

foliage to visually lose leaf density as it moves away from the observer. This is solved by performing a color correction adjustment in the pixel shader as follows. We compare the size of the pixel with the size of the leaf for gradually disabling the alpha values of the leaf texture till no alpha channel is used and the whole leaf is rendered with a single color. This technique is similar to the method used by Cook et al. [CH05], which also apply a color preservation algorithm.

3.5. Run-time implementation details

The implementation of the runtime stage is based on CUDA, the programming API that takes advantage of the unified multiprocessor architecture of the GPU. It enables efficient management of all the resources of the GPU without the limitations of the pipeline, offering an interface for the rendering API such as OpenGL or Direct3D to share data and resources. Our method takes advantage of this architecture to access and write geometry in the on-board memory of the graphics device.

The process performed at run time by our implementation is detailed in Algorithm 2. Every kernel invocation in the algorithm is acompanied by the symbols $<$ and $>$ just behind it. These symbols contain a number that represents the number of threads the kernel executes for completing the task.

The algorithm uses the following data structures:

- *LoDCells* is a buffer allocated in video memory. Its length is equal to the number of cells. It is used to store the appropriate LoD of each cell according to the current situation of the camera.
- *cameras* is a buffer allocated in host memory. Its length is equal to the number of viewpoints. It is used to store the position of each viewpoint surrounding the tree.
- *cells* is a buffer in video memory storing the indices to the leaves contained in each cell.
- *offsets* is a buffer in video memory to store the offsets for each cell in the resulting index buffer of leaves.
- *lodIndices* is a buffer in video memory used to store the indices of the triangles representing the foliage in the appropriated LoD.

Algorithm 2 works as follows. First, function $FindCameras$ seeks for the three nearest viewpoints given the current camera location, $vpoint$. This function outputs three camera identifiers along, $camIDs$, with three weights

Algorithm 2 Algorithm for processes performed at run time.

$nC \leftarrow NumberofCells$
$FindCameras([in]vpoint, [in]cameras, [out]camIDs, [out]camWgs)$
$AssignLoD < nC > ([out]LoDCells, [in]vpoint, [in]camIDs, [in]camWgs)$
$CalcOffsets < nC > ([out]offsets, [in]LoDCells)$
$MapOGLBufferToCUDA(lodIndices)$
$KernelDoLoD < nC > ([in]cells, [in]offsets, [out]lodIndices)$
$UnMapOGLBuffer(lodIndices)$

defining the influence of each camera in the current situation, $camWgs$. As it was said above, 16 cameras were enough to capture the general shape of the foliage in order to provide good estimations.

Next, the function $AssignLoD$ calculates a single LoD_{cell_factor} for each cell (in the range $[0, 1]$) on the GPU. Results are stored in the buffer $LoDCells$. The function takes into account the distance of the object to the camera, $vpoint$ and combines it with the view-dependent cell factors associated to the three nearest precached cameras weighted according to their influence, $camWgs$. This LoD_{cell_factor} is calculated in parallel for every cell.

Then, function $CalcOffsets$ calculates the offsets where each cell must start writing data to the buffer $lodIndices$ in order to avoid collisions. This way, each cell is assigned a space in the final buffer and data can be written in a parallel way on the GPU.

Finally, the OpenGL buffer is mapped in order to make it accessible from CUDA, function $MapOGLBufferToCUDA$. Indices data are written in the buffer through the invocation of the function $KernelDoLoD$ which copies the resulting indices to the index buffer sequentially.

After this process finishes, the buffer $lodIndices$ is unmapped and the tree is ready to be rendered at the current level of detail.

As it was said in a previous section, the criterion used for parallelizing our LoD algorithm is based on the cells. Each kernel execution processes the level of detail of each single cell in parallel. The official CUDA documentation recommends around 192 or 256 minimum threads per block. However, given that the amount of registers used by our kernels is less than 16, we decided to setup our kernels to use 512 threads per block, the maximum amount of threads per block in CUDA. In this way, we are able to maximize occupancy of the CUDA resources as well as enabling the algorithm for scaling well on future graphics hardware.

In order to achieve high memory bandwidth in CUDA, memory is divided into equally-sized memory banks, which can be accessed simultaneously. Bank conflicts arise when various kernels attempt to access to the same mem-

ory bank at the same time, which causes bad performance. In our scheme, each single thread represents a single cell of the cloud and manages its level of detail. Each cell contains its own set of leaves, which are mutually exclusive. This way, neither calculating the LoD_{cell_factor} nor generating the final indices list access to the same memory location and thus, avoiding the bank conflict problem.

Finally, the size of the leaves is altered at run-time using a vertex shader. In a preprocess step, each of the four vertices of a leaf is assigned a precomputed vector $\vec{e}$ in the following way:

$$\forall i \in \{0,1,2,3\}\, \vec{e_i} = \left\| \vec{v_i} - 0{,}25 \sum_{j=0}^{3} \vec{v_j} \right\| \tag{3.10}$$

The resulting $\vec{e}$ vector is passed to the vertex shader and used for determining the direction on which each vertex must be moved in order to alter the size of the leaf.

3.6. Forest rendering

This chapter have introduced a multiresolution model for foliage rendering that is completely oriented to the GPU. However, efficiently handling a forest is not trivial and some extra work needs to be done in order to avoid costly calculations. Every multiresolution model has associated an extraction time for providing a certain level of detail. Even though our algorithm is executed on the GPU and it provides better extraction times than other CPU oriented algorithms (like [RRCR04][RCB+02][RRCR06]), it would be inefficient to be extracting the appropriate level of detail of hundreds or even thousands of trees every frame.

To solve that problem, we provide a level of detail management system which reduces the amount of LoD extraction operations in the whole scene by delaying them to next frames. This LoD system runs entirely on the GPU in order to maximize performance. When the viewpoint changes, some LoDs of the tree instances have to be updated. Let LoD_{err} be the difference between the current (LoD_{stored}) and the desired ($LoD_{desired}$) level of detail for a tree instance in the present view. This term is defined as follows:

$$LoD_{err} = \mid LoD_{desired} - LoD_{stored} \mid \tag{3.11}$$

Where $LoD_{desired}$ is defined as the sum of LoD_{cell_factor} for all tree cells at a given time:

$$LoD_{desired} = \sum_{i=1}^{ncells} (LoD_{cell\_factor})_i$$

At run time, a set of tree instances are selected every frame for updating their level of detail based on different factors: their individual distance to the observer ($dist$) and their LoD_{err}.

The LoD manager computes the factors $dist$ and LoD_{err} on all tree instances in parallel on the GPU, and obtains their *LoD urgency* (LoD_u) as shown in Equation 3.12.

$$LoD_u = \frac{LoD_{err} \cdot old}{dist} \tag{3.12}$$

The term old in Equation 3.12 denotes the amount of time the LoD factor of a single tree remains unchanged. In practice, this is implemented as a CUDA array, and it is incremented every time the LoD management is executed and it is reset when a tree instance is selected for changing its LoD.

This formulation ensures that tree instances that are closer to the observer's position will update their level of detail more frequently because, as they are potentially affecting more pixels than further trees, they are considered more important in the scene. The old term in equation ensures that all trees are going to be recalculated once in a while, preventing further trees of being never updated.

Furthermore, our level of detail management system is able to efficiently determine the amount of tree instances that intersect the viewing frustum and therefore are selected for being rendered. This is important for large forest scenes where thousands or even tens of thousands of trees are used.

Regarding the implementation, the LoD manager runs as follows. First, it runs a CUDA kernel for every tree instance of the forest, calculates LoD_{err} and updates old values for every one of them. In the same step, viewpoint visibility is also calculated by intersecting each tree bounding volume against the frustum. These operations are efficiently performed due to their simplicity and the massive parallel computing power of current GPUs. A second CUDA kernel is used for performing an ordering over each tree marked as visible in the previous step. The sorting criterion used is $LoD_{urgency}$. Our sorting algorithm is based on previous work for efficient parallel sorting found in the literature [Ion97][SA08].Then, a new user defined parameter is involved in the process. This new parameter defines the time for performing LoD extraction tasks before rendering each frame. During this time, the LoD extraction process is executed for the first trees in the array, until the time defined by the user is over.

Figure 3.7: Visual results on a close up with 50 % reduction on the level of detail over the original tree shown in Figure 3.12.

3.7. Discussion

This section discusses the differences and advantages of our method against other geometry-based LoD approaches for the foliage.

Our approach has been designed from scratch as a completely GPU-based view-dependent multiresolution model for the foliage. This means that the data storage, the LoD extraction and rendering algorithms have been designed to be stored and executed on the GPU. This approach has some direct advantages. Firstly, compared to [RCB+02] and [RRC+07] it removes any traffic between the CPU and the graphics processor, avoiding the PCIe bottleneck of these methods due to their necessity of uploading to GPU memory all vertex indices each time the LoD extraction process is performed.

In addition, it provides a level of detail management system that is also designed to be executed in parallel on the GPU and prevents the GPU to perform unnecessary LoD operations on large forest scenes. Rebollo et al. [RRC+07] use a foliage subdivision system for preventing uploading the whole foliage to the GPU each time the LoD changes. However, without a LoD manager both [RCB+02] and [RRC+07] provide especially bad performance, because the system is easily stalled with data transfers.

The work presented by Deng et al. [DZYJ10] uses a technique similar to [DVS03] for efficiently rendering LoD meshes on the graphics hardware. Although this approach is very efficient and flexible and does not require bus traffic for extracting the LoD, it has three major drawbacks. Firstly, as the rendering is based on sequential point trees [DVS03], it uses a large number

of rendering calls to visualize each tree. This is especially inefficient when a large number of trees are being rendered, such as a forest. Moreover, they use a combination of triangles and lines for rendering the foliage, increasing the amount of needed drawing calls. Secondly, as the authors say in their paper, their approach is designed to view trees at moderate distances, not being suitable for close-ups. In contrast, Figure 3.7 shows the realism achieved with our method even for close-ups. Lastly, their LoD approach is not view-dependent and does not allow for decreasing the level of detail in hidden parts of the foliage, which is important for better preserving the appearance of the tree while reducing compression ratios.

3.8. Results

This section demonstrates our technique with practical tests for measuring performance and visual quality. We have configured our test bed framework in the following way. All tests have been performed on an Athlon64 3500+ with 3GB of RAM and a GeForce 8800GT graphics card. The trees used in the experiments are geometrically described in Table 3.1. This section is divided in two parts for separating the tests performed on a tree level and those tests performed on a forest level.

Tree	Leaves	Triangles
Olea europaea	16,107	32,214
Fraxinus ornus	28,645	57,290
Quercus cerris	41,155	82,310
Cedrus atlantica	131,042	262,084

Table 3.1: Description of the foliage of the trees used in our experiment.

Pre-process	
Tree	Time (milliseconds)
Olea europaea	6,499
Fraxinus ornus	9,380
Quercus cerris	14,217
Cedrus atlantica	12,720

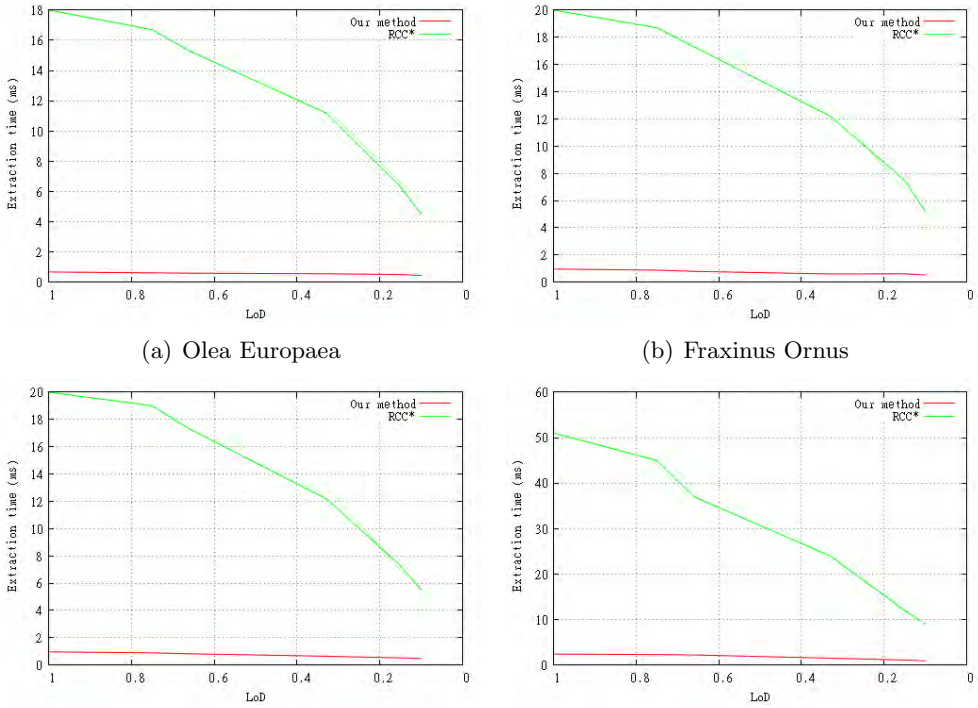
Table 3.2: Time employed in constructing the data structure of the multiresolution model.

Table 3.2 shows the time employed in the preprocess stage. The processes involved in this step are not performed in real time. After they are executed,

the data structure is prepared to interactively extract the appropriate level of detal.

3.8.1. Single tree analysis

For our single tree analysis we configured a rotating camera around each tree in order to provide a good estimation for the visibility dependent factor of our method. Moreover, for considering the distance factor of the foliage to the observer, the tree is moved away from the camera incrementally from the near to the far planes in order to test the whole range of active distances.



(a) Olea Europaea (b) Fraxinus Ornus

(c) Quercus Cerris (d) Cedrus Atlantica

Figure 3.8: Performance comparison charts for extraction times between our method and RCC*. LoD 1 means 100 % and LoD 0 means 0 %. 0.1 % used as minimum LoD.

In our experiments, we have checked the time (in milliseconds) employed in reducing the level of detail to 66 %, 33 % and 15 % as well as the time needed for pre-processing each foliage using 16 uniformly distributed cameras and a cell cloud of 512 boxes. Each measurement involves the time of extracting the geometry corresponding to the LoD, i.e., the time of mapping the hardware vertex buffers to CUDA, the execution of all the necessary processes in the CUDA kernels and the time of un-mapping the buffers to be rendered.

Table 3.3 shows the level of detail extraction times for these four different types of trees. For each value on the table, a number of independent measurements (20 for each value in our case) were taken and averaged in order to provide a good estimation of the LoD extraction times. Moreover, an additional comparison with our method running on a CPU is shown. It can be clearly seen that due to the parallel hardware-accelerated nature of our method it outperforms the extraction times obtained by the CPU-based method. Notice that, since our method has been designed to run efficiently on a highly parallel architecture, it cannot perform well on a single-core CPU. Figure 3.8 shows a quantitative detailed analysis comparing the performance of our method with RCC*. It can be seen how performing the LoD extraction process in parallel on the GPU and avoiding any traffic between the CPU and the GPU enables for a huge performance boost compared to a only GPU-oriented approach.

Tree	66 %	33 %	15 %
Olea europaea	0.6	0.56	0.51
Fraxinus ornus	0.90	0.72	0.66
Quercus cerris	0.92	0.70	0.65
Cedrus atlantica	2.3	1.63	1.23

Table 3.3: LoD extraction times in milliseconds for different tree models.

In order to demonstrate the benefits of moving the level of detail extraction process from the CPU to the GPU, the graphs in Figure 3.7 are provided. This figure compares the extraction times of reducing a tree to 10 % of its original geometry with both a CPU-based LoD method RCC* and the method presented in this chapter, which runs completely on the GPU.

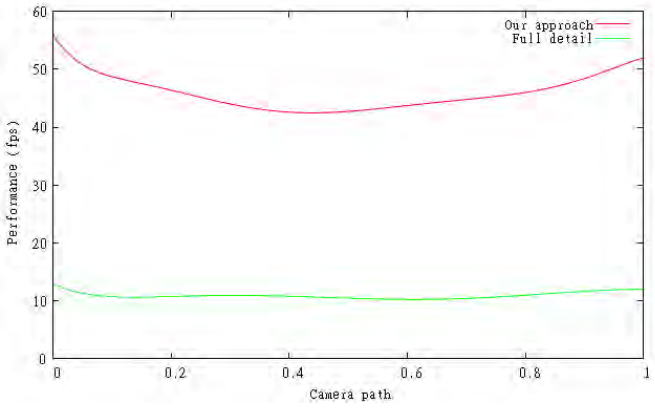
Taking advantage of the view-dependent nature of our pruning algorithm we are able to greatly reduce the geometrical complexity of the resulting tree while still achieving good visual results, shown in Figure 3.13. This figure shows a comparative study of the pruning quality of our method, comparing the quality of the resulting unpruned geometry against the original tree at the same distance to the camera and with the same orientations. Notice how the appearance of the trees is preserved even when the complexity of the foliage is reduced to 10 % of the original complexity.

Table 3.4 shows a comparative study of storage costs with two trees of different polygonal complexity using our method and the methods presented by Deng et al. [DZYJ10] and Rebollo et al. [RRC+07]. It can be seen that, as our method is completely implemented on the GPU, our CPU storage costs are null. Moreover, the use of stochastic pruning allows our method to require less memory to be stored overall than its competitors.

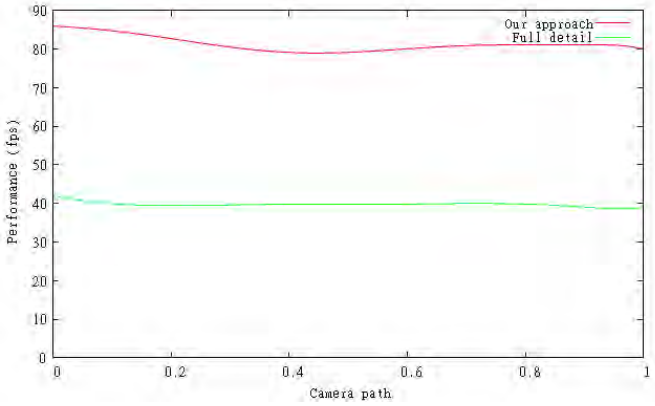
	Olea europaea			Cedrus Atlantica		
	CPU	GPU	Total	CPU	GPU	Total
Our method	0.00	1.29	1.29	0.00	9.21	9.21
[DZYJ10]	2.89	8.27	11.16	9.08	11.33	20.41
[RRC+07]	0.63	1.27	1.9	4.51	9.19	13.7

Table 3.4: Comparison of our storage cost with two existing techniques measured in Megabytes.

3.8.2. Forest analysis



(a) Dense forest



(b) Sparse forest

Figure 3.9: Performance comparison of forest scenes with our approach and with full LoD.

For the forest results analysis we set up two kind of scenes: dense and sparse forests. For the dense forest we populated the scene with 6,450 trees while only 210 trees were used for the sparse forest scene. Both scene types

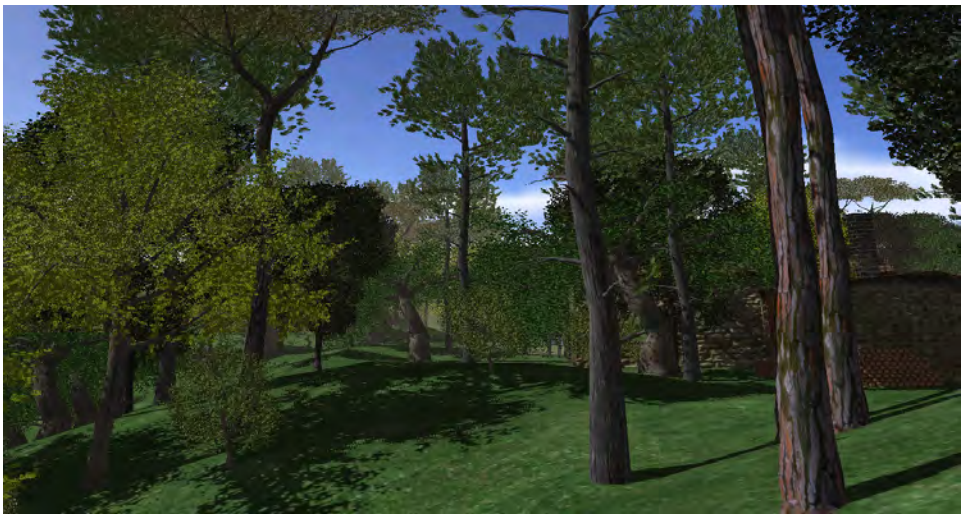


Figure 3.10: Visual results of our technique for a forest with a LoD reduced to 10 %.

are populated with the following types of trees: Fraxinus Ornus, Quercus Cerris, Olea Europaea and Cedrus Atlantica (see Table 3.1 and Figure 3.10 for details). Shadow map resolution in our scenes is 4096x4096 pixels, however, it is not recalculated every time, but only on the first frame, assuming that neither the trees nor the light source (the sun) change. Table 3.5 shows detailed information rendering scenes in Figures 3.10 and 3.11. These scenes are selected because they represent examples of two different types of scenarios: sparse (Figure 3.11) and dense forests (Figure 3.10).

Figure 3.10 shows a visual comparison of our technique managing the level of detail of our dense forest scene. At the camera location used to render the scene showed in Figure 3.10, 508 tree instances are detected to lie inside of the frustum and thus marked as visible. The LoD manager of the forest uses around 12 milliseconds each frame to perform the tasks described in Section 3.4. These tasks include the tree visibility determination, urgency calculations and tree ordering for selecting the most urgent trees to be updated. The LoD manager selects an average of 8 trees per frame for LoD recalculation, an average of 440 LoD extractions per second at 50 frames per second. It can be seen in Figure 3.10 how the view-dependent nature of our technique is able to remove potentially invisible detail from the most hidden parts of the trees so that the impact on the final image is minimized.

Performance improvements obtained using our approach are shown in Figure 3.9 for the two forests scenes: dense and sparse. It can be seen that our technique is especially useful when used for large forests scenes.

	Figure 3.10	Figure 3.11
Trees in scene	6,450	210
Trees in view	508	24
LoD extraction/management	12 ms	10 ms
Average compression ratio	10 %	30 %
Rendering frequency (LoD)	56 fps	86 fps
Rendering frequency (no LoD)	13 fps	42 fps

Table 3.5: Performance results rendering scenes on Figures 3.10 and 3.11.

Figure 3.11: Snowy scene showing our pruning algorithm in a sparse forest environment with 10 % LoD.

3.9. Conclusions

This chapter presents a view-dependent multiresolution level of detail method for real-time rendering of the foliage designed for highly parallel systems. The method is based on stochastically pruning unneeded leaves for a given LoD, depending on the distance to the viewer and its relative position with the foliage. In this way, high amounts of leaves can be rapidly discarded while preserving the general shape of the tree.

The design of this LoD scheme is based on the GPU. All the processes involved in obtaining the geometry that represents the appropriate resolution of the foliage run in this graphics hardware. Bandwidth traffic between the

CPU and the GPU, typically found in multiresolution models, is completely removed since CUDA allows for gathering and scattering operations from any direction of the video memory. It also allows for freeing the CPU from extracting the level of detail by translating this task to the GPU. Thus, it can be performed in parallel spreading the task among the whole amount of multiprocessors of the graphics hardware. As the technique runs completely on the GPU, it becomes more scalable, taking into account that the velocity at which the GPU increases its performance is much higher compared to the CPU.

Finally, we introduce a level of detail management for the forest in order to avoid stalling the system when dealing with dense forests composed of many hundreds of trees. This management system is designed to take advantage of the parallel nature of the GPU for maximizing performance and scalability.

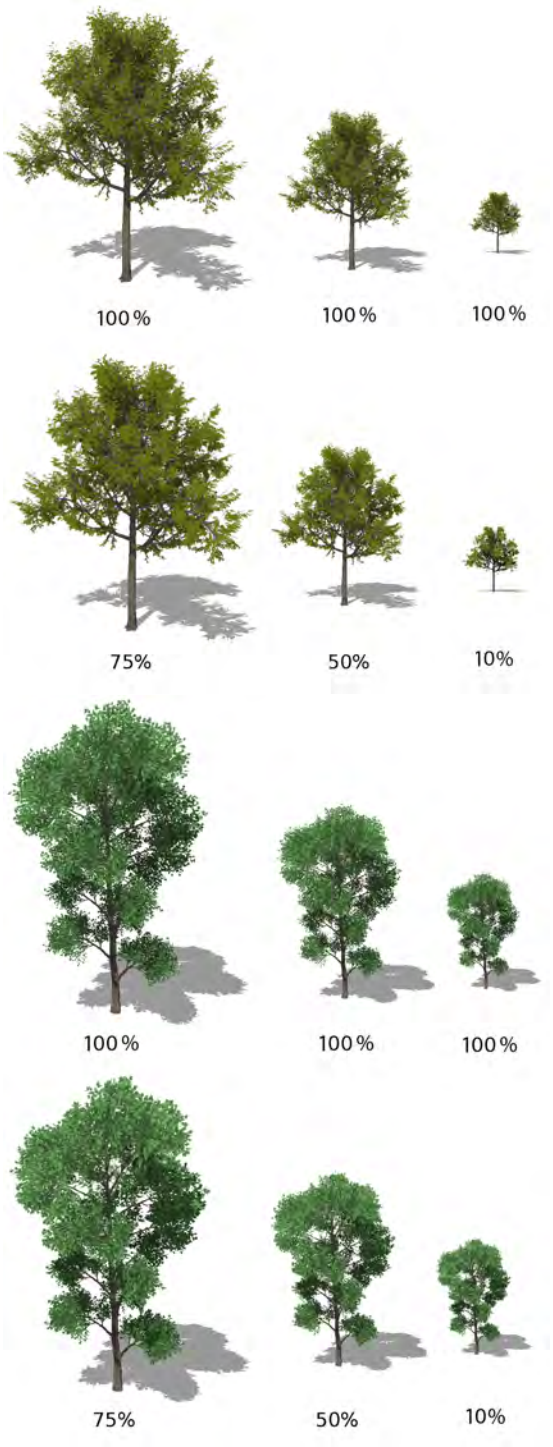


Figure 3.12: The upper rows show the tree rendered at full geometrical complexity. The lower rows show the same tree at the same distances at the following reduction factors: 75 %, 50 % and 10 %.

CHAPTER 4

LoD Manager: a framework for rendering multiresolution models in real-time applications

Multiresolution modeling has proven to be a good solution for continuously adapting the level of detail of 3D meshes, diminishing the amount of geometry to render for improving performance. However, this solution is not widely used because it presents inefficient level of detail update routines which reduce the overall performance. This chapter introduces a set of techniques that allow for adapting the level of detail while adjusting to time constraints and maintaining image quality. We call this level of detail management system the LoD Manager. In order to fulfill the requirements of current game engines, the LoD Manager exploits the graphics hardware and reuses previously calculated levels of detail which avoids saturating the application when massively populated scenes are used. Finally, we show the integration of our LoD Manager into a game engine and we demonstrate the validity of our solution with an interactive application as well as with a GPU implementation of our algorithm.

4.1. Introduction

In recent years, computer graphics have experienced an intense evolution as new graphics hardware offers a final image quality that was totally impossible to imagine a few years before. This allowed interactive graphics

applications, such as computer games, virtual reality environments or CAD applications, to include more complex scenes and detailed environments.

The necessity of highly realistic scenarios often involves including many polygonal meshes made up of a high number of triangles, which poses a problem for maintaining interactivity. In these applications, it is important to guarantee a stable frame rate while reducing the perceived lag [Wlo95]. The lag, the delay between performing an action and seeing the result on the screen, is as much important as the frame rate in order to perceive interactivity.

One of the possible solutions to this problem is the use of level-of-detail techniques to maintain a balance between image quality and rendering speed. Nowadays, multiresolution modeling can be considered as a compulsory feature of libraries and game engines. In this sense, graphics libraries such as OpenInventor or OSG, and game engines such as Director, Torque or CryEngine, introduce multiresolution models to easily alleviate the amount of geometry that must be rendered in a scene, and thus improving performance. Most of them use static heuristics, like the distance of the object to the observer or the size of its projected area in screen-space, as the metric to select the suitable level of detail. Other works like [ASVNB00] add a criterion based on the occlusion information to obtain a tighter estimation of the contribution of each object to the scene. These heuristics, despite improving frame rates, are usually not enough. They cannot guarantee stable frame rates as they are not adaptive and cannot correctly work in scenarios where objects are moving in and out of the scene or where the objects change their level of detail quickly.

In order to improve the results of the static heuristics, some authors have introduced the use of feedback algorithms, which take into account the past rendering times. These algorithms, even though are more adapted to the rendering conditions, also suffer from oscillation and "unavoidable overshoot" when rendering discontinuous environments. These techniques present a good alternative for scenarios like flight simulators where there's a large amount of coherence between frames. This is the case of the solution presented in [Hop98], which provides temporal coherence through the runtime creation of geomorphs to control the level of detail.

Motivation

A method for preventing the system to collapse due to massive amount of simultaneous LoD extraction operations is presented in Chapter 3. The method presented for multiresolution forest rendering is based on delaying LoD changes to other frames based on the LoD urgency of each tree. Although this method allows for more stable frame rates, in certain situations

it causes that the LoD changes of some objects in the scene are delayed more than an acceptable time. This chapter introduces a new LoD management system for multiresolution scenes that is not based on delaying LoD changes but on reusing previous LoD calculations. This system allows to minimize the amount of actual LoD operations, allowing for a more stable frame rate.

Other solutions have been proposed in the literature to deal with the LoD scene management problem. All the previous work shown in Chapter 2 have in common that, to optimize the GPU usage, they have developed complex heuristics [RL00][FS93], that have a certain CPU penalty. Changes in the current level of detail have associated a CPU consumption time, needed to calculate and update the object to its new rendering state. This issue is specially problematic when dealing with scenes with lots (some hundreds or even thousands) of LoD objects. In this case, changing the level of detail of the objects without any control could cause the application interactivity to drop.

Many of the articles detailed in the state of the art were written in the early days of the GPUs (or even in earlier times [FS93]) when it was viable to spend some CPU processing time to optimize the GPU rendering process. Nowadays, due to the great scalability of the graphics cards, we must revise all that related work to provide an updated and practical viewpoint of that situation: overloading the CPU is a delicate task that in most cases will cause it to be a bottleneck for the graphics hardware.

Therefore, the aim of this method is to develop a level of detail manager with very low CPU requirements, freeing the CPU by minimizing the number of real changes in levels of detail. Nowadays, the GPUs have experienced a great evolution in computational power. That provokes that real-time applications tend to be CPU bounded, i.e. the CPU limits the GPU. Thus, developing heuristics that involve high CPU processing times can be counterproductive. Therefore, the objective of this chapter is to provide a simple yet effective method that lowers the CPU usage in order to keep the bottleneck on the GPU.

Our approach also uses the concept of frame rate feedback to automatically adapt the LoD of the scene to achieve a target user-defined frame rate. This approach offers more interesting results compared to static heuristics because it allows for dynamic LoD adaptation. Even though it is less accurate compared to predictive heuristics, it is also considerably more inexpensive compared to predictive methods, which is the main aim of this technique: to minimize the CPU work as much as possible.

Figure 4.1: Top: A LoD scene composed by 3000 LoD objects. bottom: A LoD forest populated with 150 highly detailed trees.

4.2. Method Overview

In some situations, scenes may be formed by hundreds or even thousands of objects. The use of LoD techniques is a solution to maintain a balance between image quality and rendering speed. However, if objects are represented by LoD models, every time the relative position between each object and the observer changes, the appropriate resolution of every LoD object potentially needs to be recalculated. One of the main problems of multiresolution models is the time employed in the extraction of the appropriate LoD. Changing the level of detail of the objects without any control could cause the application interactivity to drop.

4.2.1. Desired level of detail of objects in the scene

The appropriate level of detail of one object in a scene depends on certain heuristics. In this chapter, we use an heuristic that takes as input the distance of the object to the camera and the current application frame rate. On the one hand, the distance of the object to the camera defines a linear function that is mapped to the range [0,1], shown in Equation 4.2. Two distances are established by the user, $range_{near}$ and $range_{far}$. Objects will be represented by the most detailed approximation if they are situated closer than $range_{near}$, and by the worst approximation if they are further than $range_{far}$. Equation 4.2 adjusts the LoD of the objects situated between these two distances.

$$\begin{array}{lcl} dist_{cam} < range_{near} & \rightarrow & dist = range_{near} \\ dist_{cam} > range_{far} & \rightarrow & dist = range_{far} \\ range_{near} < dist_{cam} < range_{far} & \rightarrow & dist = dist_{cam} \end{array} \tag{4.1}$$

where $dist_{cam}$ is the distance from the observer to the object being rendered.

$$lod(dist) = \frac{range_{far} - dist}{range_{far} - range_{near}} \tag{4.2}$$

On the other hand, the heuristic gets the frame rate as a feedback parameter to alter the linearity of the LoD function. Users can establish a desired frame rate fr for the visualization. If the application is running under this desired frame rate, this can be controlled making that more objects reduce their level of detail. This action will reduce the number of polygons in the scene, so the frame rate will increase. Otherwise, if we are running above, more objects can increase their LoD. This is controlled by the variable n.

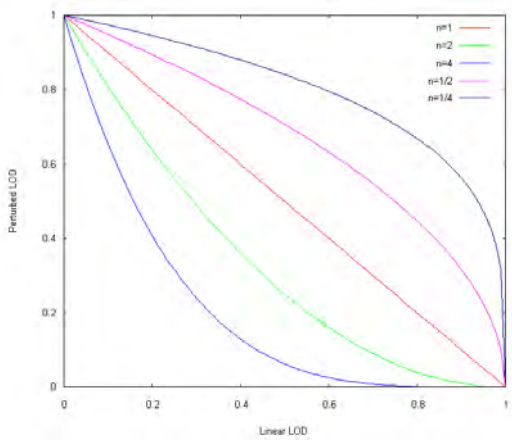


Figure 4.2: Perturbation function to calculate the desired LoD factor to adapt it to the current frame rate.

Let *desired LoD*, *d_lod*, be the appropriate level of detail of the object depending on the heuristic and the fr determined by the user (Equation 4.3). An illustrative chart is shown in Figure 4.2. When $n = 1$, the LoD of an object changes in a linear way. If we need to increase the frame rate, n will take values greater than 1. Otherwise, if the frame rate has to be reduced, n will take values less than 1.

$$d\_lod = lod(dist)^n \tag{4.3}$$

4.2.2. Sharing precalculated LoDs

In order to optimize the scene rendering process, the LoD manager aims to minimize the changes of resolution for all LoD instances, performing only the necessary ones. In massively populated scenes, it is usual to find objects that share the same geometry, i.e. they are of the same type t. Based on this fact, there is a certain possibility that two or more objects of the same type t can share a similar level of detail. This is the main idea of the presented scene manager: when two instances of the same type are similar enough and both have a similar desired LoD, the LoD manager assigns them a precalculated level of detail. Thus, an object can hold its own level of detail or a *borrowed* one.

The data organization of the LoD manager is shown in Figure 4.3. Let O be an array where the LoD manager stores references to all the visible objects in the scene. Let D_t be an array of a user defined length N_t associated to a type of LoD objects t. Each position of D_t represents a level of detail that is being visualized in the scene. If we define a LoD factor, $lodf \in [0, 1]$,

where 1 is the maximum level of detail and 0 is the worst approximation, any intermediate values in the range represent intermediate levels of detail. A position i in the array D_t contains a discrete LoD associated to the levels of detail in the following range.

$$[i/N_t, (i+1)/N_t) \in [0,1]$$

For example, assuming an array D_t of length 3, the first position in the array would represent a discrete approximation for the range [0,0.33), the second position one for [0.33, 0.66) and the last one for the range [0.66,1].

Then, when a LoD factor of type t ($lodf^t$) is determined for an object, the index i where is stored the discretized representation of its range, is obtained as follows:

$$i = trunc(lodf^t * N_t) \tag{4.4}$$

Two representations i and j of the same type t, $lodf_i^t$ and $lodf_j^t$, are similar enough when they are in the same range, so both are associated to the same position of the array D_t. Let S_t be the similarity factor defined as Equation 4.5 shows.

$$S_t(lodf_i^t, lodf_j^t) \leftrightarrow trunc[lodf_i^t \cdot N_t] = trunc[lodf_j^t \cdot N_t] \tag{4.5}$$

The main idea of the LoD manager is to avoid extracting the level of detail of a visible object if there is one stored in the D_t array which is similar enough to the desired LoD, *d_lod*. Then we say that the object will hold a *borrowed* level of detail. However, it is important for each to store a reference to its own index buffer. That's because when an object changes its level of detail, it must update its own index buffer, not the borrowed one.

4.2.3. Rendering algorithm for LoD scenes

Initially, visible objects are classified following their species or types t. Then, the array D_t is created for each different type of objects in the scene, and each position of the arrays is initialized to 0. Following the user defined frequency fr, each visible object O_i in the scene is checked in order to adequate the rendered level of detail *lodf* to the current conditions of the scene, and its desired LoD factor, $d\_lod(T_i)$, is calculated. Then, for every O_i the similarity of its target LoD $d\_lod(T_i)$ is compared with the discretized LoD stored in the array D_t. Algorithm 3 shows these steps.

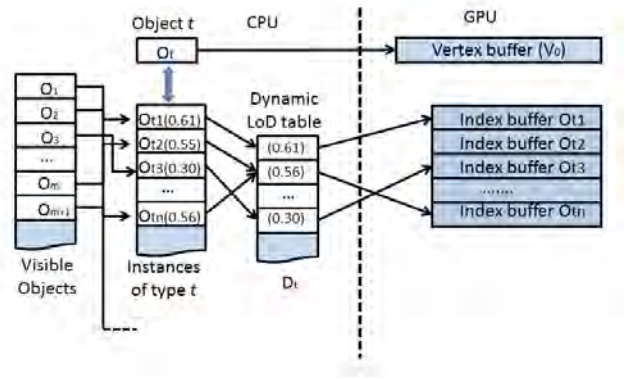


Figure 4.3: Data organization of the LoD manager.

Algorithm 3 Algorithm of LoD rendering manager.

for all $O \in VisibleObjs$ **do**
  $t = type(O)$
  $Calculate\_d\_lof(O)$
  {Calculate the associated position in the array}
  $i = trunc[d\_lod(O) \cdot N_t]$
  if NOT $S(lodf(O), d\_lod(O))$ **then**
    {the rendered LoD is not similar to the desired one. This has to be changed}
    if NOT $S_t(D_t[i], d\_lod(O))$ **then**
      {the stored LoD is not similar enough}
      $Perform\_Extraction\_dlod(O)$
      $lodf(O) = d\_lod(O)$
      {update the stored LoD in D[i]}
      $D_t[i] = d\_lod(O)$
    else
      {the stored LoD is similar enough}
      $lodf(O) = D_t[i]$
    end if
  end if
end for

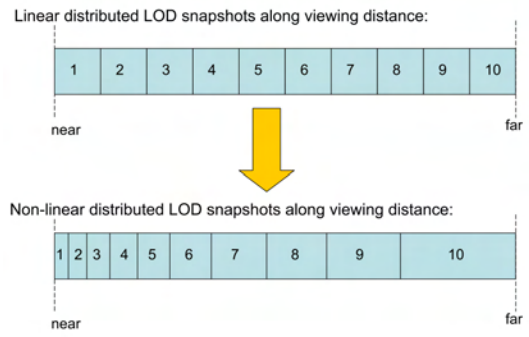


Figure 4.4: Linear vs non-linear LoD snapshots distribution.

4.2.4. Non-linear precalculated LoD intervals

As it is said before, the vector D_t stores snapshots of previously calculated levels of detail following a linear distribution. However, more real applications will prefer to use a non-linear distribution to allow perform much finer LoD changes for closer models and much coarser LoD changes for objects that are far away from the viewer.

This distribution function can be customizable by the user so that it can be used in very different client applications and situations. An illustration can be seen in Figure 4.4.

This optimization will reduce popping effects because queries for closer models will be classified with less granularity. It is important to note that Equation 4.5 should be adapted to the new snapshot distribution.

4.2.5. The minimum and maximum LoD special case

The maximum and minimum levels of detail are important. That's because in closeups, each object must be represented at its full level of detail, at 1 not at 0.97 for example. To solve this, there are two special entries in the array D_t for the minimum and maximum level of detail.

4.3. Implementation and results

This section explains how the technique described in this chapter is implemented into an existing Game Engine.

4.3.1. Library usage

We have implemented this method as a library which is independent to the underlying multiresolution model used to represent the objects. There are only some requirements that the multiresolution model must fulfill. These requirements are:

- The objects must provide an interface to change their level of detail. This interface must be implemented using the range $[0, 1]$ as the active LoD range.
- The objects must be able to implement a *fast LoD switching* functionality. In practice, this can be done by *borrowing* index buffers from other objects while keeping the original index buffer for further LoD calculations.

Our implementation provide a `LoDObject` class interface which provides some virtual functions the multiresolution models must implement. Thus, is really simple to handle several types of different multiresolution models inside the same scene.

4.3.2. LoD Models

In our implementation we have used two different multiresolution models: the one presented in Chapter 3 for representing plants and trees and another one for general meshes called LoDStrips [RC04].

LoDStrips is a multiresolution model based on triangle strips. It efficiently defines a continuous sequence of level of detail changes from a base mesh. It is a index-based mutltiresolution model, i.e. it calculates the current index set for a defined level of detail, without affecting to the vertex list.

Both models require a certain amount of time to change the level of detail (extraction time), depending on how much changes must be accomplished, and they can easily implement the *fast LoD switching* functionality described in section 4.2. Therefore, they are valid multiresolution models to demonstrate the usefulness of our manager heuristics.

4.3.3. GPU Implementation

The work presented in this chapter is susceptible of being implemented on the GPU for optimizing performance. The LoD manager can be implemented on the GPU as follows. In this case the array V is stored on graphics memory for making it accessible to the CUDA kernels. For efficiently porting this algorithm to the GPU some aspects of it must be altered, due to the

massive parallel architecture of the GPU. Firstly, objects can not borrow LoD instances calculated on the current frame, as they do on the CPU implementation, because CUDA kernels are executed in parallel. Therefore, we change this approach by reusing LoD calculations that happened in previous frames.

In order to avoid conflicts, we use two V buffers for the visible objects: one for reading the results of the previous frame and one for writing the results for the current frame, which will be used for the next frame for LoD borrowing. As in the first iteration of the algorithm there are no results of the previous frame, then use choose to initialize V by performing the first iteration of the algorithm on the CPU. Another advantage of using the GPU is that, taking into account the tremendous parallel power of the current graphics hardware, the visible objects list can be processed on the GPU with logarithmic cost, instead of being linear as it happens when using the CPU approach.

Next, in a similar way as it is done in Section 3.6 in the previous chapter, our algorithm runs a CUDA kernel for every LoD instance of the scene, calculates the error and updates *old* values for every one of them and, finally, calculates $Scene_{err}$. In the same step, viewpoint visibility is also calculated by intersecting each object bounding volume against the frustum. These operations are efficiently performed due to their simplicity and the massive parallel computing power of current GPUs. If the $Scene_{err}$ exceeds the threshold established by the user, a second CUDA kernel is used for performing an ordering over each LoD object marked as visible in the previous step. Our sorting algorithm is based on previous work for efficient parallel sorting found in the literature [Ion97][SA08].

Finally, a new user defined parameter is involved in the process. This new parameter defines the time of performing LoD extraction tasks before rendering each frame. During this time, the LoD extraction process is executed for the first object in the array, until the time defined by the user is finally consumed.

4.4. Results

In our tests, we have used two different multiresolution models: one for general meshes [RC04] and the model presented in Chapter 3.6, which is specially designed for handling vegetal species. The test machine is an Athlon 64 3500+ CPU with 1 Gb RAM and a GeForce 6800 Ultra video card. The Ogre mesh features 1960 triangles and its minimum level of detail reduces the triangle count to the 10 %. It implements the LoDStrips algorithm (briefly described in section 4.3.2). The Tree mesh represents an *Olea europaea* with

97133 triangles at full level of detail, it uses the LoDTree algorithm to reduce its triangle count to the 10 % at its minimum level of detail.

Two different tests are proposed: a performance test which measures the performance boost when using the LoD Manager, and a visual quality test that will prove the visual acceptability of the method.

Performance test

We have used two different test scenes. The first scene was populated with 3000 independent LoD objects of the Ogre mesh, shown in Figure 4.1. The second scene adds 300 highly detailed tree LoD objects to the previous one to show how the algorithm can deal with heterogeneous scenes.

All the test scenes move the camera through a predefined path. Figures 4.6 and 4.7 show the performance comparison enabling and disabling the LoD Manager in two different scenes. Thus, the improvements in performance offered by the LoD Manager can be easly measured. In addition, this figure offers the number of triangles rendered during the walk-through. These graphs are a good help to understand the frame rate obtained, and also proves how the number of triangles rendered with and without the LoD Manager is nearly the same, proving that the LoD Manager offers higher frame rates while maintaining a similar visual quality.

Figures 4.6 and 4.7 show how the LoD Manager efficiently manages level of detail changes and minimizes the CPU consumption due to the LoD management. In fact, when dealing with scenes with a high count of independent LoD objects (like in the scene of the 3000 ogres), the CPU consumption dedicated to LoD changes becomes the bottle neck of the application reducing the performance to make it unsuitable for interactive content.

Visual quality test

We have provided some performance tests where our LoD Manager proves its usefulness in LoD scenes populated with lots of independent LoD object. Now we will demonstrate that our heuristics does not affect the visual quality of the models in a significant manner. The topmost image in Figure 4.5 shows a scene populated with ogres using our technique managing the level of detail of the whole scene. The middle image shows the same scene without any LoD management approach active, i.e. each independent object treats its own level of detail independently. The differences caused by our method are shown in the third image of Figure 4.5, where a red pixel shows a difference between the two images, and a black one indicates that the pixel is the same. We can see that the visual differences are almost imperceptible and that they only can seen comparing images to per-pixel level, as we have

done in this subsection.

4.5. Conclusions

We have introduced a new technique to minimize the number of level of detail changes of a scene populated with a high count of LoD objects. This technique allows us to reuse LoD calculations to minimize the CPU computation time. In Chapter 2 we have analysed some methods which use more complicated heuristics than ours, and thus, they require more computation time. Our algorithm also features a feedback heuristic that is able to globaly reduce or increase the LoD of the scene to achieve a user defined frame rate.

Nowadays, the great scalability of the graphics processor units has contributed to make them more powerful than the CPUs for parallelizable algorithms. Thus, real world applications tend to be CPU bound and the GPU becomes limited by the CPU power, it's more useful a technique that saves CPU time as well as providing a real world acceptable LoD management, rather than more sophisticated techniques that consume CPU to save GPU cycles. This is specially true when dealing with scenes with a high number of LoD objects, where predictive methods tend to be completely unsuitable for real time applications.

Even though our technique has been designed to be much simpler than predictive heuristics, it has proved to be quite simple to implement as well as effective to minimize CPU consumption, to manage heterogeneous LoD scenes and to help maintain target user-defined frame rates.

Figure 4.5: Top: screenshot of a scene using the LoD Manager. Middle: screenshot of the scene with the LoD Manager disabled. Bottom: per-pixel differences between the other two pictures.

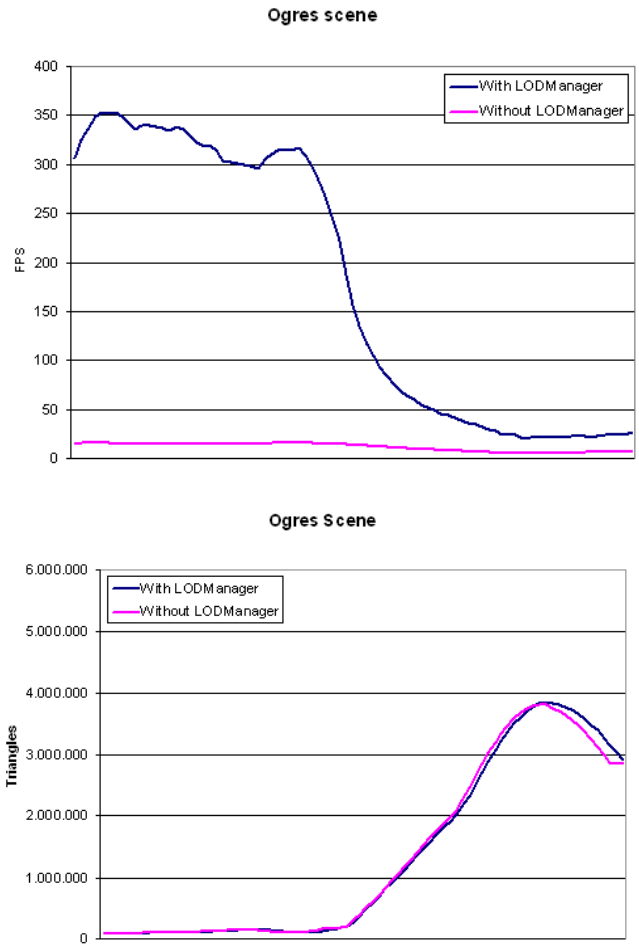


Figure 4.6: Performance comparison with and without LoD Manager in the Ogres scene (topmost figure). The horizontal axis represent the current time of the simulation. The figure on the bottom shows the amount of triangles sent to the renderer.

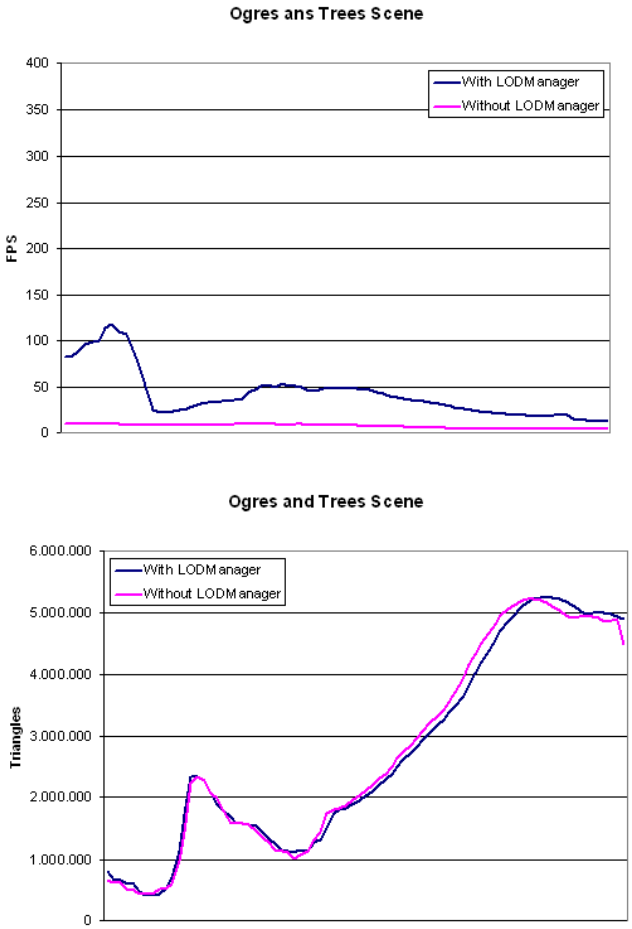


Figure 4.7: Performance comparison with and without LoD Manager in the forest scene (topmost figure). The horizontal axis represent the current time of the simulation. The figure on the bottom shows the amount of triangles sent to the renderer.

CHAPTER 5

Real-time illumination of foliage using depth maps

Illumination is a critical part in the process of image synthesis creation in order to achieve realistic results. This area is so important that even really simple scenes look realistic if they are rendered using high-quality illumination techniques. Global illumination methods provide approaches to simulate the physics of real-world lighting and its interaction with the objects in the scene, achieving realistic illumination effects. Nevertheless, global illumination techniques are computationally expensive and, commonly, not suited for real-time applications. This chapter proposes a novel technique for enhancing illumination effects for plants and trees in real-time applications.

5.1. Introduction

Due to the scattered nature of the foliage, the standard Phong model is unable of realistically illuminating this kind of tree models. In order to correctly perceive the density of the foliage, the interaction of the light inside of the foliage has to be taken into account. To solve this problem there exist in the literature several methods for simulating global illumination effects, such as radiosity, path tracing of photon mapping, which are suitable for foliage rendering but they are very expensive to be used in real time applications. Therefore, special methods for foliage lighting and shading are needed (such as [LBO07]).

This chapter presents a new method for foliage lighting, shading and shadowing that provides good quality illumination in real time while keep-

ing acceptable frame rates. More over, this chapter also provides a shadow casting algorithm for the foliage which uses leaves density estimation criteria for determining the amount of shadows intensity at a given point.

Our approach is based on the rendering equation introduced by Kajiya [Kaj86] described as follows:

$$L_o(x, \vec{w}) = L_e(x, \vec{w}) + \int_\Omega f_r(x, \vec{w}', \vec{w}) L_i(x, \vec{w}')(\vec{w}' \cdot \vec{n}) d\vec{w}' \qquad (5.1)$$

where the amount of light irradiating from an object $L_o(x, \vec{w})$ at a given point x and direction $\vec{w}$ depends on the light the object emanates $L_e(x, \vec{w})$, which we can ignore because leaves do not emit light, and on the incoming light $L_i(x, \vec{w}')$. Light incoming from all directions is modulated with the angle of incidence of the light onto the surface $\vec{w}' \cdot \vec{n}$ and the bidirectional reflectance distribution function (BRDF) $f_r(x, \vec{w}', \vec{w})$ that describes the reflectance function, which depends on the properties of the material.

This chapter discusses how to implement each part of the Equation 5.1 to provide a realistic illumination for the foliage in real-time.

Finally, a new shadow casting algorithm is introduced. This algorithm takes into account leaf density information in a given light direction in order to render shadows produced by the foliage over other surfaces, such as the ground or the trunk.

5.2. Method Overview

Our method is based on the observation that the illumination over real-life trees is greatly affected by the illumination that comes from the scene and how that global illumination interacts with the complex structure of leaves. Figure 5.1 shows a schematic view of how light interacts with the foliage in terms of direct lighting, which comes directly from the sun and through the leaves, and indirect lighting, which is the light that reaches the foliage after being reflected by the clouds and the ground.

Figure 5.1 shows how, due to this lighting interaction, leaves which are inside of the foliage have a greater probability of being less affected by the light than external leaves, because they are more exposed to environmental lighting. Figure 5.2 represents an example in a real-life photograph of a tree that shows this behavior.

In order to capture the overall volume of the foliage, our method uses two depth maps. The first depth map stores the nearest depth values and the second one stores the furthest depth values when rendering the foliage. Figure 5.3 illustrates this process. The difference of the two depth maps

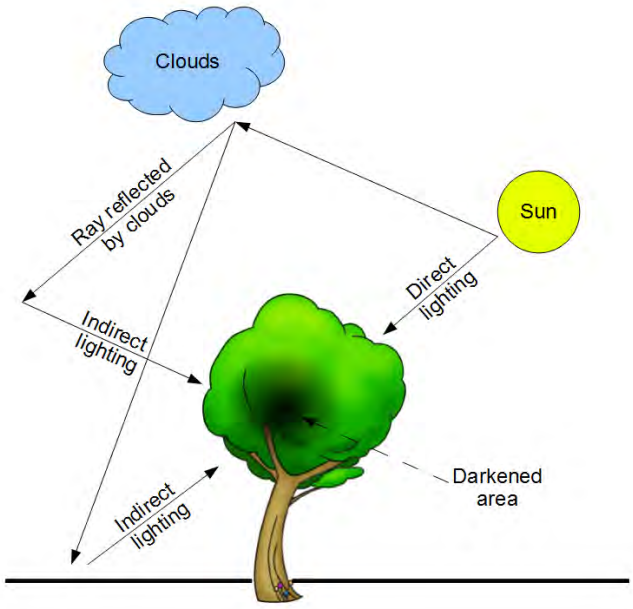


Figure 5.1: Schematic representation of different light interactions with the foliage.

gives us an estimation of the overall depth for each light direction. Thus, assuming that the leaves are uniformly distributed over the foliage, we are able to calculate the relative position of each single leaf inside the foliage at rendering time.

Although Equation 5.1 takes into account the light incoming from all directions, evaluating all light directions would be very expensive. Thus, we apply our BRDF calculations only over the light direction which comes directly from the light source, separating the direct from the indirect light contributions. Therefore, we obtain the following formula for light irradiance at a given point x and direction $\vec{w}$, where the direct light contribution is separated from the indirect lighting ($A_\Omega(x)$):

$$L_o(x, \vec{w}) = A_\Omega(x) + f_r(x, \vec{w'}, \vec{w}) L_i(x, \vec{w'})(\vec{w'} \cdot \vec{n}) \tag{5.2}$$

The method proposed replaces the A_Ω term in Equation 5.2 by a new indirect lighting algorithm specifically designed for the foliage.

Our illumination method has been developed having in mind the nature of the leaves in order to simulate the complex interaction of the light inside the foliage. A visual analysis of the interaction of light with the foliage provides a simple conclusion about this issue: both inner leaves as well as those leaves in the opposite side from the light source receive less light and

Figure 5.2: Leaves which are inside of the foliage have a greater probability of being less affected by the light than external leaves, because they are more exposed to environmental lighting.

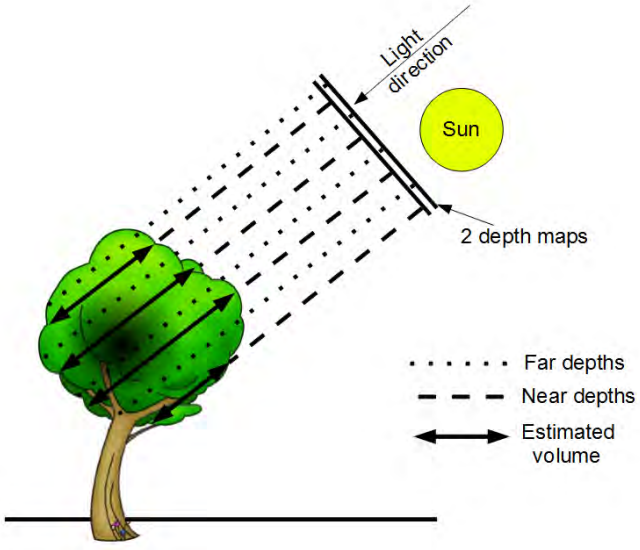


Figure 5.3: Our method captures the volume of the foliage by using two different depth maps.

(a) Tree foliage without illumination. (b) Indirect lighting affecting the foliage.

Figure 5.4: Results of our indirect lighting approach.

Figure 5.5: Different views of a tree with different ambient light contributions. From left to right: white, reddish and yellow light.

thus, are darker. This is caused due to the auto-occlusion of the leaves, which prevents the light from reaching those leaves and makes them receive less light.

Our approach is based on approximating the shape of the foliage from the light source and illuminating each leaf depending on its position inside the foliage volume in order to approximate self-shadowing effects of the leaves. We use two depth maps capturing the nearer and further parts of the foliage from the light source which we call D_n and D_f. The main idea is that the nearer a leaf is to D_f respective to D_n the darker it should be rendered. In addition, another texture C is used to determine the amount of leaf intersections per pixel, from the light source. This value will be used to determine the leaf density in a given direction.

When rendering a leaf at run time, the relative positions of the leaf and the light source are used to calculate the adequate texture coordinates for accessing the depth maps, in the same way like in traditional shadow

Figure 5.6: Left: foliage without illumination. Middle: ambient lighting only. Right: the complete illumination system, including direct and indirect lighting, self-shadowing and shadows casted over the trunk and branches.

mapping. In practice, this is done on the graphics hardware in the fragment shading step. The pixel shader compares the depth of each leaf in light space with the minimum and the maximum depths at that point to determine its proximity to those values. This factor is weighted with the value contained in the texture C which determines the amount of leaves at that light direction. Equation 5.3 shows the formula used to calculate the self-shadowing factor (S) of the leaf depending on the light source i.

This shadowing factor replaces the $L_i(x, \vec{w}')$ term in Equation 5.2:

$$L_i(x, \vec{w}') = \alpha N_c \left(1 - \frac{Z_x - Z_n}{Z_f - Z_n}\right) \tag{5.3}$$

where α is the transparency level of the leaf, N_c is the number of leaf collisions at a certain light direction given by the texture C, Z_x is the depth of the current leaf fragment in light space, and Z_n and Z_f are the minimum and maximum depths in that light direction given by textures D_n and D_f respectively.

Therefore Equation 5.3 provides the darkening factor for each pixel depending on the light source direction and the general shape of the foliage volume. The results of this equation matches to the light intensity function of Equation 5.2. Figure 5.6 shows an example of our illumination approach for the foliage.

The direct lighting contribution of the foliage is calculated in the following way. Due to the translucent nature of leaves, a subsurface scattering based BRDF is needed to correctly simulate the illumination on the leaves. Jensen et al. [JMLH01] propose an efficient method for subsurface scattering

which separates the scattering process in a single scattering term $L^{(1)}$ and a diffusion approximation term L_d, as shown in Equation 5.4.

$$L(x, \vec{w}) = L^{(1)}(x, \vec{w}) + L_d(x, \vec{w}) \tag{5.4}$$

Frankze et al. [FFD03] show how Equation 5.4 can be approximated as shown in Equation 5.5 due to the minimum thickness of the leaves providing a method easier to evaluate in real-time:

$$f_r(x, \vec{w}', \vec{w}) = L^{(1)} + L_d = (1 + e^{-s_i} e^{-s_o}) L_i(x_i, \vec{w}') \cdot (\vec{N} \cdot \vec{w}') \tag{5.5}$$

Where s_i is the leaf thickness and s_o is a random outgoing distance inside the material from the actual sample position. This approximation matches the $f_r(x, \vec{w}', \vec{w})$ component in Equation 5.2 and describes the BRDF associated to the direct illumination.

Direct illumination is evaluated in the pixel shader fetching some parameters from textures such as leaf thickness and normal information.

5.3. High-frequency illumination

The process explained in previous section allows us to calculate low frequency global illumination, taking into account direct and indirect components, for the foliage. However, for further improving the quality of our illumination solution we also need to provide accurate high frequency lighting interactions inside the foliage. For that purpose, we developed two different new algorithms that are used along with the solution for further enhancing the quality of our illumination.

The first of these techniques is an accurate visibility estimation per leaf done on the GPU. The second one is a screen-space ambient occlusion technique. The former is useful for accurate high frequency illumination assuming that leaves to preserve their relative position, as it is calculated in a preprocess, in scenes that there are no wind effects. The latter is entirely calculated in real-time and is useful for scenes where the foliage is being moved by strong winds, however it is slower to calculate than the former one. Therefore, the user must choose between these two solutions depending on the parameters of the scene.

5.3.1. Precalculating visibility

This section calculates the indirect illumination contribution as a preprocess step. The amount of light a leaf receives from the scene is calculated

as the visibility of the leaf from the exterior of the foliage. Figure 5.4 shows the results of applying our indirect light algorithm.

The ambient light received from a leaf depends on the visibility of each leaf from the exterior of the foliage. As all the leaves on the tree are of the same size, the visibility value for each leaf from outside of the foliage is calculated using an orthogonal projection of the leaf over a known virtual viewport.

This has been implemented by rendering each leaf with a unique colour with a depth buffer activated. Thus, the visibility of each leaf from outside the tree is given by the number of pixels of the same colour on the six faces of a cubemap surrounding the foliage. To represent the colour of the ambient light affecting each leaf, a texture read operation is performed over a downsampled cubemap that contains the environment of the tree. The normals of both faces of each leaf are used as texture coordinates to fetch this data from the cubemap. Figure 5.5 shows a tree illuminated using only indirect illumination with different scene light ambient absorption.

These visibility calculations are performed for both sides of each leaf, because each face of a single leaf can receive different amounts of light with a different colour, depending on the scene and the depending on the direction the leaf is facing. However, light tends to spread across and through the leaf, depending on its translucency. Thus, the transparency level of the leaf is used to add light reception values from both sides of the leaf. Thus, the light scattering property of the leaves is taken into account to calculate the ambient occlusion term.

Finally, the ambient occlusion colour A_Ω for a given face of each leaf i is calculated as shown in Equation 5.6.

$$A_\Omega = [C_i I_i (V_i/V)^n + \alpha[C'_i I'_i (V'_i/V)^n] \tag{5.6}$$

where α is the transparency of the leaf, I_i is the colour the current face of the leaf i absorves from the scene, I'_i is the colour the opposite face of the leaf absorves from the scene, C_i and C'_i represent the colour of the front and opposite faces of the leaf i, V_i and V'_i are the number of pixels generated by the current and opposite faces of the triangle i respectively and V is the number of pixels generated by projecting the leaf without occlusion (the rest of the foliage). The parameter n is always positive and controls how rapidly the darkening for the ambient term occurs in the foliage. Values of $n > 1$ will result in a more rapid darkening and values of $n < 1$ will cause the darkening to slow down from outer to inner parts of the foliage.

The results of this process are stored per vertex so that it can be applied per leaf at run time.

5.3.2. Screen-space ambient occlusion

This method consists of computing an obscurance value for each visible point in the scene. To do this, only the information of the visible parts of the scene is used. We use a deferred shading pipeline to obtain the positions and the normals, in eye space, for every pixel in the screen. in a subsequent step, these images are used to generate obscurance values corresponding to each part of the scene visible and, thus, increasing the realism of the rendered image.

The algorithm is divided in two parts. First of all, the scene is visualized from the observer's point of view, storing the positions and the normals in eye space as textures, as well as the diffuse colors for every pixel. All these textures are generated simultaneously, processing the geometry just once, by using a technique called Multiple Render Targets (MRT), which is available on modern graphics hardware.

In a second step, the previously generated information is processed to calculate the obscurance values using our algorithm. We render a full screen quad so that every pixel in the screen is evaluated. Thus, the pixel shader which implements our algorithm, is executed over every pixel in the screen and decides the level of obscurancy and color of each pixel.

The explanation below describes how to calculate the level of obscurancy for every pixel on the screen. The algorithm is only executed over every pixel which does not belong to the background, given a fragment P, with position P_p and a normal P_n in eye space.

The values generated for each valid sample are composed of four channels of information. The first three channels contain color information and the fourth channel contains the amount of valid samples taken into account.

Sampling function

Sampling of possible occluders for the point P is performed in image-space. This means that the sampling distance is measured in pixels. This decision simplifies calculations and increases the performance.

The main idea is taking samples inside a circle centered in the point P. This allows us to use less samples to obtain the desired quality. To do that, two random values are computed inside the range [-1.0,1.0] and they are multiplied by a third random value to define the radius, which lies inside the range [MinimumRadius,MaximumRadius]. To choose the random values we use the Van der Corput's method of generating pseudo-random numbers.

To enhance the relation quality/performance even more, these radii are multiplied by a factor which depends on the distance to observer. This way,

further points will take samples with a minor radius in image-space, while for nearer points, the radius will be greater. To determine this factor, we divide the distance to the camera by a user defined value which can be interpreted as the scene radius. The scene radius defines the actual size of the scene in the real world. This way, the greater the value is, the less area will be taken into account to perform the screen space sampling. Thus, the area affected by the obscurances is user determined by using this parameter.

Transfer function

Once an occluder is determined to be visible, its distance to $p_{occluded}$ is used. If this distance is greater than a d_{max} distance, then there is no occlusion at all and the algorithm returns a (1,1,1,1) vector for that sample (Figure 6). Otherwise, the three channel obscurance values correspond to the following formula:

$$reflectivity \cdot \sqrt{dist/d_{max}}$$

where $dist$ is the distance between the occluder and the occluded point, d_{max} is te maximum distance for accepting occluders and $reflectivity$ corresponds to the diffuse color at point B. Observe that color bleeding can be disabled by setting the $reflectivity$ factor to a value of 1.

5.4. Shadow casting over other surfaces

While we have covered the lighting interaction of the leaves, how the light penetrates across the foliage and reaches another surfaces is also important. The foliage can be seen as a set of multiple layers of translucent leaves. Therefore, the amount of shadowing other objects receive from the foliage depends on how many leaves intersect a light direction and their amount of transparency. To simulate this, we use the texture C (see Section 5.2) which stores the amount of leaf intersections at a given direction weighted with the leaf transparency at each point.

This texture can be calculated in a single pass and updated along with the others shadow maps. We use this information to render more realistic shadows over surfaces, where the depth of the foliage is taken into account to visualize more convincing foliage shadows.

Figure 5.7 shows the final appearance of our shadow casting algorithm. Notice how the shadow map shows the depth of the foliage, being more opaque where there is more leaf density throught the light direction, and being more transparent where there is lower leaf density.

Figure 5.7: A detailed view of our shadow mapping approach for the foliage. Notice how the depth of the foliage is captured in the shadows.

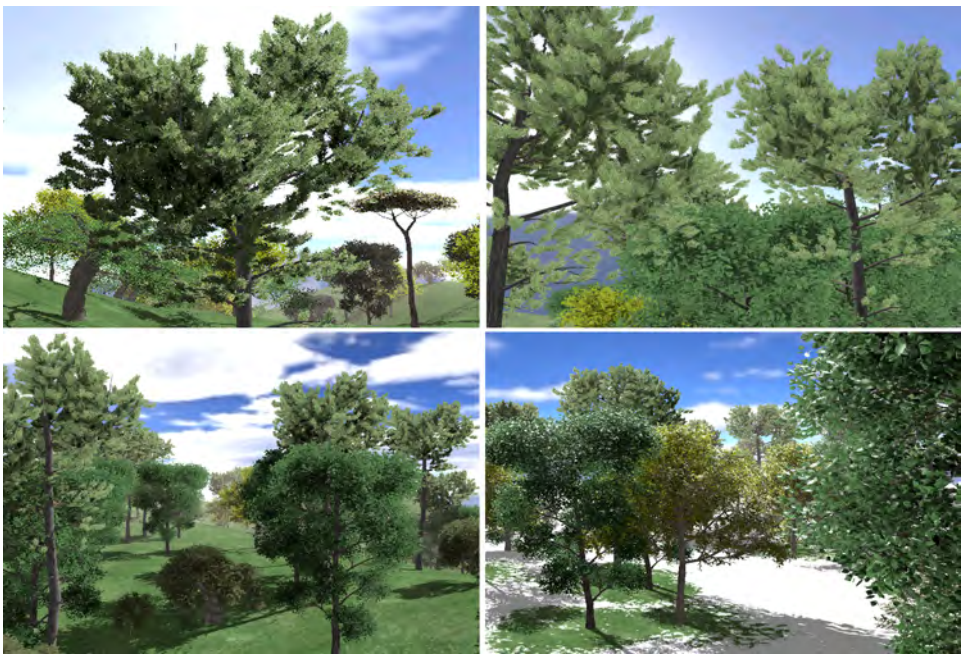


Figure 5.8: Forest scenes with our illumination and shadowing approach.

5.5. Results

In our tests, we have used a geometry-based continuous level of detail algorithm for the foliage. This allows for performance optimizations when rendering the forest scene with such amount of trees.

Figure 5.8 shows the results of our illumination solution. Notice how the illumination captures general shape of the foliage, darkening those parts that are difficult to reach for the light.

This method for foliage illumination requires to access to three different

depth maps per pixel to evaluate the illumination equation. However, this is optimized to require just one texture read operation by packing all textures in a single three channel floating point texture. Thus the overhead of applying this method is just one texture read operation and a few arithmetical operations in the pixel shader. Therefore, applying our method adds a little overhead in these cases, being the main drawback to store three values per texel instead of just one. However, the visual quality of this method for foliage illumination justifies this storage overhead.

The cost of casting foliage shadows over the ground or any other surface (as the trunk) is negligible compared to a standard shadow mapping algorithm, as the only difference is what the shadow map contains and a couple of arithmetical operations in the pixel shader.

5.6. Conclusions

This dissertation chapter presents an approach for foliage illumination and an expressive shadow casting algorithm for leaves which can be used in real time applications. The algorithm is based on depth maps. This means that in scenarios where shadow mapping is being used to simulate the shadows of the foliage as well as the auto-occlusion of the leaves, this method will improve the visual quality of the scene at the expense of little computational cost.

We propose two methods for calculating the ambient occlusion factor of the leaves. First, for static scenes our method uses the GPU to precalculate the ambient occlusion term, this allows to speed-up the rendering because it is easier to evaluate at run-time. Trees are always located in the same place in the space. Therefore we take this into account to accelerate the ambient occlusion calculation by preprocessing it and storing as per-vertex attributes. Thus, the cost of applying the ambient occlusion is negligible. Second, for dynamic scenes we propose to use a screen space-based ambient occlusion technique, which allows for localized high frequency illumination effects in real-time at a reasonable computational cost. This is useful when dealing with windy scenes.

As it is said before, this method uses depth maps as the base tool to infer the illumination and to render the shadows, such as trapezoidal, perspective, light-space perspective shadow maps. This method is built on top on existing shadow mapping algorithms, dealing with the meaning of the information contained at each texel in the shadow map, so it does not compete with other shadow mapping methods, but extends them.

Although we have used geometry-based trees in this article, the algorithm is also applicable to image-based or point-based trees because the

information needed to calculate the illumination is stored in a separate map and is not attached to geometry (like in [LBO07]) which is a restriction in real-time rendering.

CHAPTER 6

Screen Space Soft Shadows

Shadows are a very important element in synthetic scenes because they greatly contribute to enhance the realism of the rendered images. Nowadays, shadow mapping is the most used technique in real-time applications because it can be implemented efficiently on the graphics hardware and its performance scales very well. It is also one of the most active areas of research in the last years.

The most common shadowing methods for real-time applications can be grouped in two categories: shadow mapping and shadow volumes. However, the usage of shadow mapping has grown in the last years due to its performance scalability, because it is very easy to implement in the graphics hardware and to produce smooth shadows.

6.1. Introduction

Shadow mapping is a very efficient technique to calculate shadows for point light sources. However, lights in the real world have a volume which generates penumbrae. Figure 6.1 shows an example of real-world penumbrae, or soft shadows. A point in a surface is considered to be in penumbrae when it is not completely visible by the the light source (Figure 6.2).

Unfortunately, the traditional shadow mapping algorithm is unable to generate shadows with penumbrae, as it can not handle area light sources (Figure 6.2).

In order to generate physically correct penumbrae, we need to determine the amount of light visible from the point being shaded, which is proportional

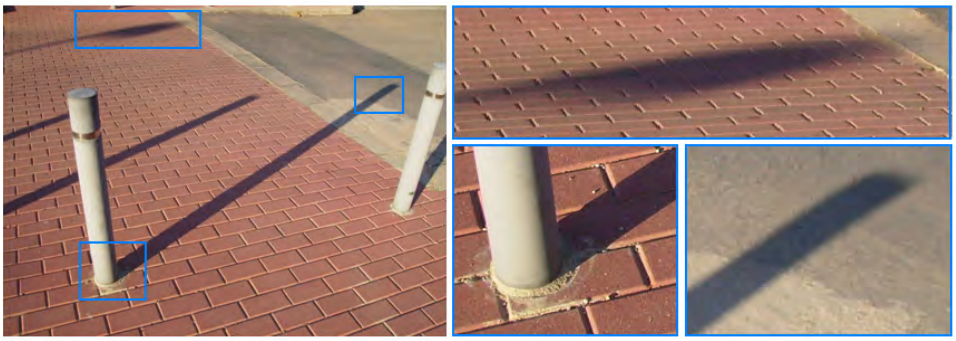


Figure 6.1: Example of real world penumbrae. Shadows become sharper as they approach the occluder.

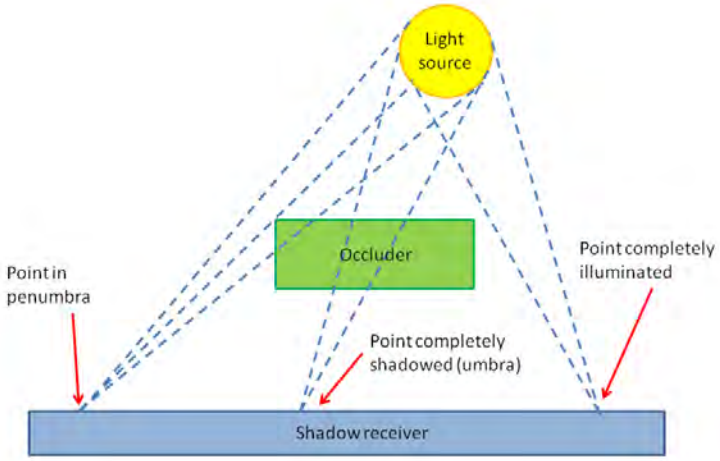


Figure 6.2: The size of the penumbra is determined by the amount of light rays reaching the point being rendered.

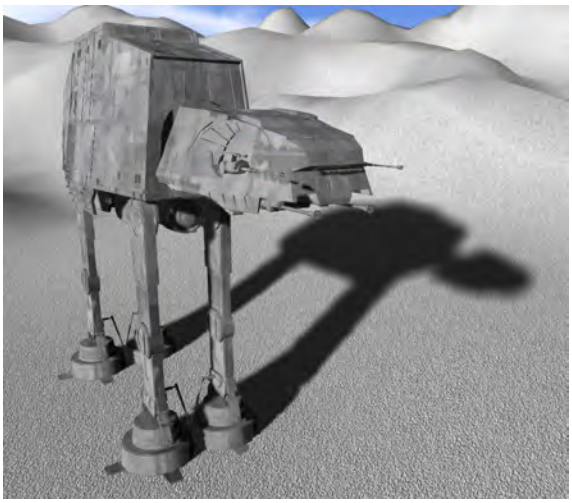


Figure 6.3: Scene rendered with our method using a 11x11 Gaussian anisotropic kernel in screen space. The image shows how the soft shadow becomes sharper as it approaches to the occluder.

to the size of the penumbra.

A common idea used for representing shadows with penumbrae is to approximate area lights by a set of point light sources, and then to combine the contributions of each single shadow. With this method, the softness of the penumbra is proportional to the number of virtual light sources used. However, in practice this method is very expensive, because the shadow casters need to be rendered many times, introducing a huge overhead in geometry-limited scenes. Therefore, more practical solutions are needed in order to be used in real-time applications.

6.2. Method Overview

The aim of this dissertation chapter is to introduce a new soft shadow mapping algorithm for generating variable-sized penumbrae that minimizes texture look-ups in order to maximize performance. Our technique generates shadows with penumbrae using an anisotropic Gaussian blur filter in screen space with variable size. The idea behind this approach is simple: a Gaussian filter is separable and then it requires far fewer texture accesses than other kernel-based sampling approaches, thus improving performance.

This chapter proposes a new method for calculating soft shadows with variable penumbrae in real time. The method is based on blurring the shadows from the observer's point of view by using an anisotropic Gaussian filter of variable size. The aspect ratio of the anisotropic Gaussian filter is deter-

mined by using the normal at the point being rendered. The size of the area affected by the filter, which generates softer or sharper penumbrae, varies per pixel and depends on the amount of light potentially received from the area light source. This factor is determined by the visibility of the area light from the point being rendered. The formula used to estimate how much light is received was proposed by [Fer05] (Equation 6.1).

$$w_{penumbra} = \frac{(d_{receiver} - d_{blocker}) \cdot \ w_{light}}{d_{blocker}} \tag{6.1}$$

where $w_{penumbra}$ is the final width of the penumbra, $d_{receiver}$ and $d_{blocker}$ are the distances of the receiver and the blocker to the light and w_{light} is the size of the area light.

Observation reveals that shadows produced by area lights (including the penumbra) are larger than shadows produced by point lights, because the area affected by the penumbra increases with the size of the light source. Therefore, our method generates a "dilated" version of the shadow map in order to evaluate the Gaussian filter for those pixels potentially belonging to the area affected by the penumbra. This process is detailed in Section 6.2.1. Without this "dilated" shadow map, we only would be able to render the so called inner penumbrae.

As a result, this method is able to generate soft shadows with perceptually correct penumbrae, depending on the distance between the shadow casters, the shadow receivers and the size and position of the light source (Figure 6.3).

The following steps describe the process performed, for each light source, to generate soft shadows with our method.

1. Calculate the standard shadow map (S_{map}) and a "dilated" version (S'_{map}) of the shadow map.

2. Render the scene from the observer's point of view and calculate the following elements in the same rendering pass: the shadows without penumbrae (or hard-shadows), the depth buffer, a normal buffer and the shading of the scene (without shadows). The distances map is also calculated in the same rendering pass. This map contains the distance from the point being evaluated (P) to the first light occluder, as well as the linear distance to the observer.

3. Deferred shadowing: render a full screen quad with our custom anisotropic Gaussian blur filter to blur the hard-edged shadows in screen space and to combine them with the shaded scene. The per-pixel size of the area affected by the blurring kernel is calculated using the data in the distances map.

Figure 6.4: Different intermediate steps of our algorithm. From left to right: the model with hard shadows, the standard shadow map, the dilated shadow map and the final result of blurring the shadows with the anisotropic Gaussian filter.

The configuration of the multiple render targets (MRT) used to calculate all needed buffers in one rendering pass (step 2) is described as follows:

- MRT0. Diffuse color without shadows.
- MRT1. Normal-depth buffer (RGB: normal's XYZ. Depth is stored in the alpha channel).
- MRT2. Shadow buffer.
- MRT3. Distances map, which contains the following information in the first three channels.
  - R: distance of the shadow caster to the point being rendered (D).
  - G: distance of the observer to the point being rendered (Z).
  - B: mask value determining whether the point is inside the penumbra or not.

6.2.1. Calculating the shadow maps

First of all, the standard shadow map is calculated from the light source. It is important to note that this information is insufficient to directly determine the distance to the occluder in order to represent the outter penumbrae.

To solve this problem, we create a coarser version of the shadow map by preprocessing it in the following way. Each pixel of the coarse shadow map will approximate a block of pixels of the standard shadow map. The criterion used for this approximation is the minimum value (closest to the light). The contents of the coarse shadow map are used as a depth estimation to calculate the distance map, not to generate the shadow itself. We use this criterion because, performing the average of samples of the shadow map

without taking into account the shadow receiver, would compute incorrect z-values, and thus incorrect penumbrae.

The dilation is performed in light space, by applying an isotropic min-filter to the original shadow map, after it is computed. Given that this filter is separable, it is computed efficiently as two one-dimensional filters.

The amount of dilation is proportional to the size of the area light source, because the size of the penumbra is also proportional to the size of the light source. This is implemented by increasing the radius of the "minimum-value" filter kernel. However, as we are performing the dilation of the shadow map in image space, the shadow receiver can not be taken into account to calculate the size of the penumbra. As a consequence, the user has to apply a constant factor to the amount of dilation, because the size of the penumbra is also proportional to the distance between the shadow caster and the shadow receiver. This factor is interpreted as the maximum distance to the occluder in the scene. If this parameter is too small, penumbrae will not be completely smooth. However, if the parameter is too large the resulting penumbrae will be less accurate. In practice, it is not difficult to visually set up this value for a given scene.

Once calculated, the filtered shadow map will allow us to calculate the distances map for every point of the penumbrae in screen space, including the outer penumbrae.

6.2.2. Calculating the distances map

The distances map is a screen-aligned texture that contains, per pixel, the distance of the shadow to its potential occluder and its distance to the observer. This is computed by rendering a full screen quad so that every pixel in the screen is evaluated. Distances to the occluder are computed by transforming the point being evaluated to the light space. This way, its depth value can be compared directly with the depth of the coarse occluder.

For optimization purposes, the distances map also stores a mask determining which pixels will never receive neither a shadow nor a penumbra. The shadow mask is useful to reduce texture look-ups and improve performance.

6.2.3. Applying the Gaussian filter

Determining the size of the penumbrae

This step generates the penumbra by applying an anisotropic Gaussian blur filter in screen space. The size of the region affected by the kernel varies per pixel depending on:

- The distance of the shadow to the occluder.
- The distance of the light source to the occluder.
- The size of the light source.

To take these factors into account, F. Randima [Fer05] introduced a formula (Equation 6.1) which estimates the size of the penumbra by using the parallel planes approach. This equation assumes that the occluder, shadow receiver and light sources are parallel. However, in practice it works very well and provides a formula which is not expensive to evaluate.

We derive Equation 6.1 by adding the distance of the pixel to the observer to the computations, because our filter is applied in screen space and the area affected by the filter diminishes as its distance from the observer increases. Equation 6.2 shows how the previously calculated buffers are now combined in order to determine the size of the area affected by the filter in screen space.

$$w_{penumbra} = \frac{(d_{receiver} - d_{blocker}) \cdot \ w_{light}}{d_{blocker} \cdot \ d_{observer}} \tag{6.2}$$

In Equation 6.2, the size of the penumbra ($w_{penumbra}$) depends on the following members. The term ($d_{receiver} - d_{blocker}$) represents the distance between the shadow receiver and the shadow caster. $d_{observer}$ is the distance to the observer. These parameters are stored in the distances map. w_{light} is the size of the light source. Finally, $d_{blocker}$ represents the contents of the "coarse" shadow map and stores the distance to the blocker in light space.

Anisotropic filtering

The anisotropic Gaussian filter is a separable filter, and then, one two-dimensional blurring can be performed in two sequential one-dimensional blurring passes: one horizontal and one vertical. This is the key to our method, because applying a Gaussian filter to create the penumbra requires far fewer texture accesses compared to the PCSS approach, which is not separable, allowing the cost of our method to be $O(n+n)$ instead of $O(n^2)$.

For each sample accessed to perform the Gaussian filter, their distance to the observer is taken into account to discard samples whose distance to the current pixel is greater than a certain threshold. This is used to prevent the filter kernel from taking into account parts of the scene which are close in eye space but are far away in world space. It also avoids having to filter the shadows with the contents of the background.

The number of samples taken by the Gaussian filter determines both the quality of the shadows and the performance. Therefore, this trade-off

(a) Percentage-closer soft shadows.

(b) Our algorithm. Kernel(23x23)

(c) Our algorithm. Kernel(11x11)

(d) Our algorithm. Kernel(5x5)

(e) Reference raytraced shadow.

Figure 6.5: Visual quality comparison between our algorithm (with three different kernel sizes) and other approaches: a raytraced shadow (e) and an implementation of PCSS (a).

decision is left to the user as a customizable parameter. An interesting optimization, in order to reduce the number of texture accesses, is to decrease the number of samples as the area affected by the blurring kernel decreases.

To determine the shape of the anisotropic filtering, the normal of the current pixel is fetched from the normal buffer (generated previously using Multiple Render Targets). Using this normal, the local tangent space is calculated and used to determine the local X, Y and Z axes. Projecting these axes to eye space allows us to determine the shape and orientation of the ellipse which defines the anisotropic filter.

To perform the anisotropic filtering in an efficient way, we use the method presented by Geusebroek et al. [GS01]. This work derives the anisotropic Gauss filtering and proposes to apply it as a separable kernel, which can be evaluated efficiently.

Finally, after the vertical blurring pass is performed, the penumbra has already been calculated and the pixel shader combines it with the unshadowed scene C to create the final image with complete shadows with penumbrae.

6.2.4. Using average instead of minimum depth

Our technique provides a simplification which allows us to rapidly generate penumbrae minimizing the number of texture accesses per pixel, as based on the minimum depth texture approach. However, while this technique is able to generate plausible soft shadows in most scenes, it may not be completely accurate for some scenes with very complex shadow casters and receivers.

Fernando R. [Fer05] shows how the average depth of the potential occluders is a valid measure to determine the size of the penumbra at a given point. This process, called the *blocker search* step, is accomplished by performing a number of samples over the shadow map in order to determine the average distance of potential occluders. The size of the sampling area is proportional to the size of the light source. On typical scenes, performing 5x5 samples over the shadow map is sufficient to provide accurate results. However, as the size of the light source increases, more samples may be needed for detailed objects to avoid artifacts due to the spacing of the samples in texture space.

Moreover, Annen T. et al [AMB+07] introduce an optimization to the blocker search step by performing it as a convolution filter. This way, this step can be done efficiently on the graphics hardware.

Therefore, if desired, a traditional approach based on the blocker search can be implemented to use the average depth instead of the minimum depth,

as used in percentage-closer soft shadows, while still being able to use our screen-based anisotropic Gaussian filtering to generate the penumbrae.

Obviously, the minimum depth texture is not longer necessary when computing the size of the penumbra. Therefore, the step of generating that texture can be safely skipped.

6.3. Implementation details

As the penumbra is generated using a screen space Gaussian filter, there are some aspects that must be taken into account when applying this filter because otherwise artifacts would be introduced. For example, Figure 6.6 shows the results of applying the screen space Gaussian filter without taking into account the underlying geometry.

These artifacts are caused by the fact that the Gaussian filter is based on filtering the hard-edged shadows that are *visible* from the screen. This makes not possible for the Gaussian filter to take into account those parts in the shadows that are occluded from the camera's point of view. A good example of that problem can be seen in Figure 6.6, where the planar object is completely occluding it's own shadow from the user's point of view. That causes that the Gaussian blur is not capable of blurring that shadow and generating a penumbra around it which in this case should be visible (see Figure 6.6.b). We present two different solutions for solving this problem.

6.3.1. Multi-layered shadows

As stated before, the problem of blurring occluded shadows is that they are not visible from de user's point of view, i.e. they are now shown in the first layer stored which is stored on the frame buffer. A natural way of solving this problem could be calculating the hard-edged shadows for the first N visible layers from the user's point of view and storing in different textures so that they could be selected by the filter as complementary information when needed. Figure 6.7 shows the first two shadow layers taking as input the scene shows in Figure 6.6.

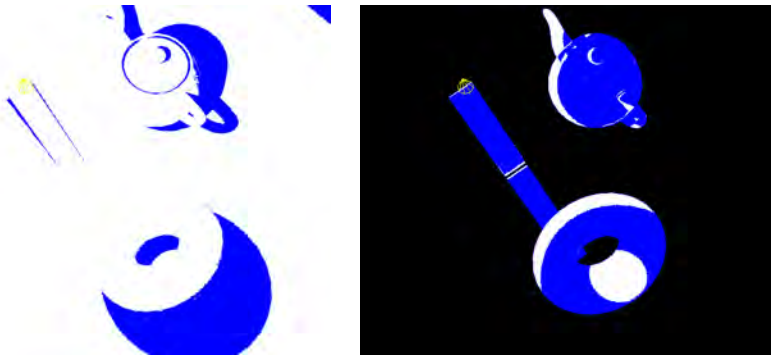
Perceptualy, we found that the number of layers needed for reducing artifacts does not increase linearly with the complexity of the scene, but logarithmically. That means that computational and storage costs of our algorithm would increase by $O(log(n))$, being n a value linearly proportional to scene's complexity. In practice, the number of layers is a user-defined parameter and determines the amount of artifacts avoided during filtering.

For improving efficiency when using this technique, the *streaming out* capabilities of the graphics hardware can be used for avoiding transforming

(a)

(b)

Figure 6.6: Figure (a) shows some artifacts which are the result of applying the Gaussian filter to the shadow buffer without taking into account the underlying geometry. Figure (b) solves this problem by using our new *lazy shadowing* technique.



(a) (b)

Figure 6.7: The figure on the left shows the hard-edged shadows seen from the user's point of view. Figure on the right shows the hard-edged shadows that are NOT seen from the camera and belong to the second shadow layer. Black pixels on the second image show the parts of the scene stored on the first layer.

geometry primitives multiple times. This means that, after rendering the first layer, transformed vertices are stored in a GPU-allocated vertex buffer that can be used in subsequent layers, increasing performance for secondary layers. Moreover, the amount of pixels that need to be evaluated in subsequent layers decrease in every layer, because those present in computed layers are not necessary.

Algorithm 4 shows the algorithm performed by the screen space filter to blur the hard edged shadows and how the algorithm chooses between two different layers when there is no sufficient information in the first layer.

6.3.2. Lazy shadowing determination

Using multi-layered shadows with sufficient layers can alleviate the problem of shadowing artifacts. However, the fact that it needs multiple layers can reduce performance in complex scenes. We propose a lazy shadowing approach for avoiding the need of multiple shadow layers to be generated. This approach is based on calculating the shadows on demand during the application of the Gaussian filter, which can be calculated by using the light's projection matrix used for generating the shadow map.

The main problem of this technique is that, for those points that are not visible from the user's point of view, we can not determine whether a point is in shadow or because we don't know the world position of that point. For example, consider Figure 6.8. The blue square represents the filtering area for a given point located in the ground near the shadow caster itself. This is a conflictive area since the blur filter is not able to filter the parts of

Algorithm 4 Pseudo-code illustrating the different steps performed by the algorithm.

```
nSamples ← Number Of Filtering Samples
errDepth ← User − Defined Depth Error
sampleRef ← FetchSample(center) {fetch nSamples}
for i = 1 to nSamples do
  shadowSample ← FetchSample(layer2, offset(i)) {compare depths}
  if |shadowSample.z − sampleRef.z| ≤ errDepth then
    accumWithWeight(layer1)
  else
    shadowSample ← FetchSample(layer2, offset(i))
    if |shadowSample.z − sampleRef.z| ≤ errDepth then
      accumWithWeight(layer1)
    else
      More Layers Would Be Needed
      Do Not Take Sample Into Account
    end if
  end if
end for
```

the shadow that are not visible from the user's point of view, because are occluded be the shader caster itself. This case would introduce light ghosting around the object's silhouette, as shown in Figure 6.6.

Lazy shadowing determination is able to solve this problem by extrapolating the world position of those points taking into account the current view/projection matrix and the local tangent and normal vectors at that point. This is possible to calculate because we assume that the surface is locally planar in the direction on the tangent vectors.

This technique proves to be better than the multiple layered approach because it is able to greatly remove the light leaking artifacts without the need of storing extra information (multiple layers) and with the extra benefit of only fetching the shadow map for points behind objects when needed.

6.4. Results

This section presents performance and quality tests performed using our method with different scene configurations. All tests were run on an Athlon +3500 processor with 3GB of RAM memory and a GeForce 8800GT graphics card. In order to better showing the quality of the shadows, quality comparative images were rendered using a black ambient light over untextured surface. This way, shadows can be studied easily.

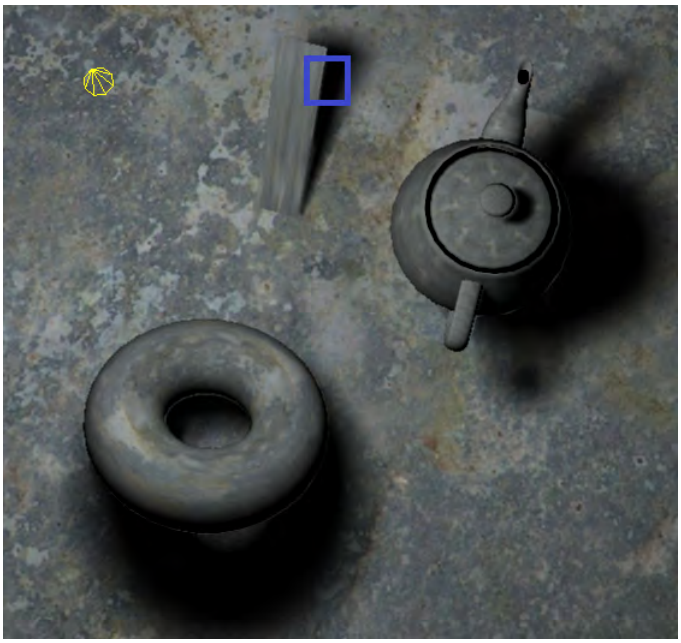
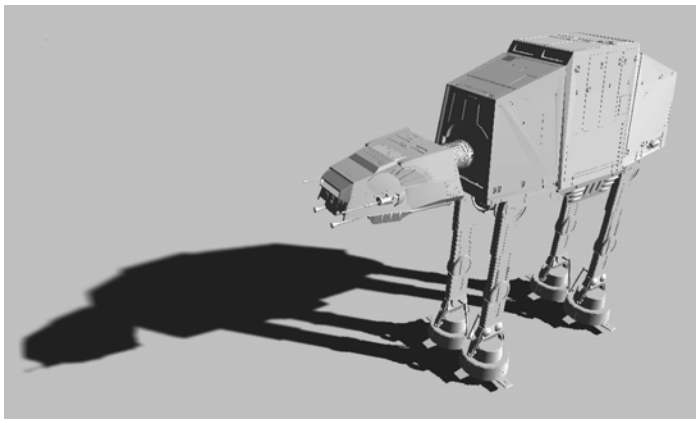


Figure 6.8: The blue square represents the filtering kernel on a conflictive area for a given point located in the ground near the shadow caster itself.

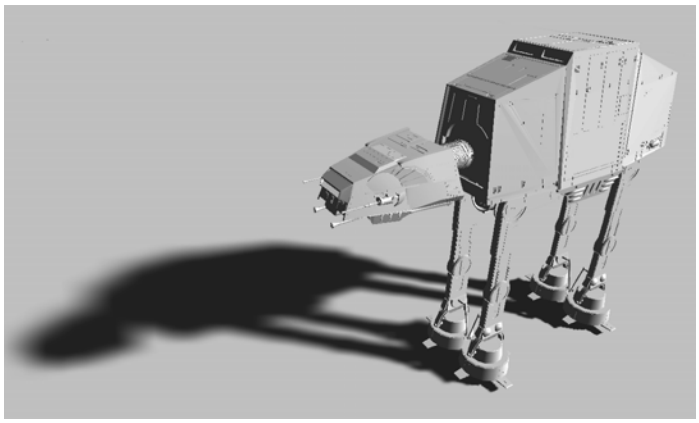
6.4.1. Quality tests

The number of samples used by the Gaussian kernel greatly affects the final quality of the penumbra, especially when large light sources are used and large penumbrae must be generated. Figure 6.5 shows a set of shadows generated with different kernel sizes in order to show penumbrae quality with different configurations. Three kernel sizes were used: 5x5, 11x11 and 23x23. The image shows how the small 5x5 kernel produces some discretization artifacts in the penumbra. The 11x11 kernel is useful for the majority of cases, but it can be insufficient when the camera comes close to the penumbra. In these cases a 23x23 kernel is more than enough for obtaining good quality.

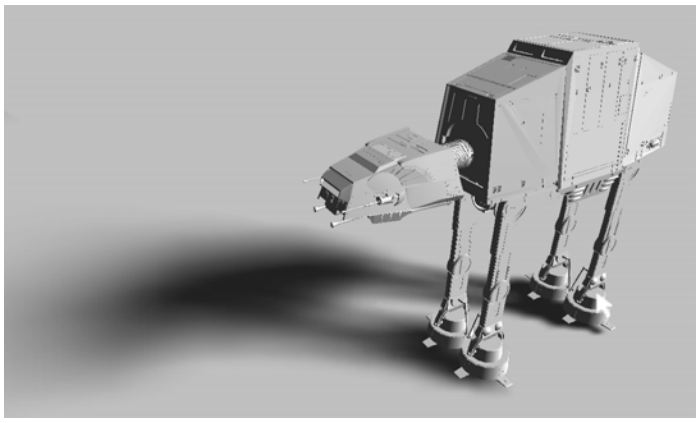
Figure 6.9 shows the effects of increasing the size of the area light. This figure shows how the size of the penumbra grows proportionally to the size of the light source. Figure 6.9(c) shows that, even with a huge area light, the algorithm is able to represent visually pleasant shadows with perceptually correct penumbrae.



(a) Small light source.



(b) Medium light source.



(c) Large light source.

Figure 6.9: Effect of changing the size of the light source. The size of the penumbra is proportional to the size of the light source.

Figure 6.10: Example of penumbrae with different light sizes and different light colors.

6.4.2. Performance tests

Table 6.1 compares performance in the AT-AT scene (shown in figure 6.9), using different configurations and techniques. The first and second columns indicate the methods and configurations used. The rest of the columns show performance (measured in frames per second) for each configuration in both scenes. Standard shadow mapping is used to provide the time needed to calculate shadows without penumbra. In the second row, the time needed to calculate a uniform-sized penumbra is provided. This penumbra is calculated using a percentage-closer filter combined with a screen space blur filter which removes artifacts and softens penumbrae. Next, many timings are taken using our method with some different kernel configurations under different screen resolutions. It can be seen how our method performs very well, being its costs similar to the uniform-sized penumbrae cost. Finally, percentage-closer soft shadows are used in order to provide performance measurements for comparing our technique with a well known soft shadowing method.

As it is shown in the table, our technique is able to perform very well even at high screen resolutions, outperforming percentage-closer soft shadows (PCSS) with similar kernel sizes and screen resolutions. Moreover, it can be seen how performance drops when using PCSS while incrementing the screen resolution and kernel sizes, while performance remains more stable with our algorithm.

6.5. Conclusions

This chapter presents a new approach for calculating soft shadows with variable-sized penumbrae in real-time. To optimize this task, we introduce the concept of distance map, which stores the distance from a pixel potential-

Method	Kernel setup	800x600	1280x1024	1600x1200
standard	*None*	942 fps	595 fps	245 fps
PCF	$PCF(3x3)$	462 fps	230 fps	175 fps
SSSS	$K(5x5)$	553 fps	278 fps	213 fps
SSSS	$K(11x11)$	513 fps	256 fps	181 fps
SSSS	$K(23x23)$	441 fps	221 fps	155 fps
PCSS	$B_s(5x5) + PCF(5x5)$	504 fps	239 fps	183 fps
PCSS	$B_s(5x5) + PCF(11x11)$	251 fps	106 fps	78 fps
PCSS	$B_s(5x5) + PCF(23x23)$	122 fps	49 fps	37 fps

Table 6.1: Performance results measured in frames per second (FPS) on the AT-AT scene (200K triangles). SSSS stands for our method (Screen Space Soft Shadows). K refer to the kernel sizes used with each technique. B_s stands for *blocker search*, used in the PCF algorithm.

ly affected by the penumbra to the occluder that produces that shadow. This distance is used to generate penumbrae in screen space using an anisotropic Gaussian blurring kernel.

The bottleneck of the PCSS approach is the number of texture accesses required to achieve smooth penumbrae. First, it has to perform a blocker search to determine the overall distance of the shadow to the occluder. Although this step requires at least 3x3 texture reads, it is advisable to use at least 5x5 or even 7x7 to completely avoid artifacts on complex shadow casters. Our method performs the blocker search by just accessing the distances map, which can be generated from a low-resolution coarse shadow map. In addition, PCSS needs to take multiple samples of the shadow map in order to generate the penumbrae. In practice, 13x13 is a good kernel size to achieve smooth shadows with PCSS. Thus, the number of samples required to generate the penumbra with this method is: 5x5 (blocker search) + 13x13 (PCF) = 194 texture reads.

Besides, since our algorithm uses a separable filter, the cost of computing the penumbra is $O(n + n)$ instead of $O(n^2)$, as in the PCSS approach. As an example, using an 11x11 kernel with PCSS would require 11x11=121 texture accesses, while by using a separable Gaussian blur it can be performed with only 11+11=22 texture look-ups. This method also proves to be very scalable because increasing the kernel to 17x17 requires only 34 samples with our method and 289 samples with PCSS. This means that even using a massive 50x50 Gaussian filter (50+50 = 100 texture look-ups), our method would offer better performance compared to PCSS while generating extremely smooth penumbrae.

Moreover, the use of an anisotropic filtering allows our method to take into account the orientation of the surface being shadowed. This way, the

screen space filtering is able to deliver precise penumbrae even at grazing angles.

Another advantage of using our method is that it is compliant with the concept of deferred shading. This shading scheme, which is commonly used in films and post-production, is getting popular in the field of real-time graphics. Deferred lighting (see [Eng08]) uses a similar approach for rendering efficiently a high amount of lights in real-time. Our technique is easily integrable in a deferred shading pipeline, performing all the calculations in screen space, taking as input the same buffers used in the deferred shading (except for the distances map). The direct benefits of our approach are that no superfluous calculations are wasted on invisible pixels, as it is applied in screen space over the computed shadow buffer.

Nevertheless, this technique presents some limitations. The first limitation is that we are simplifying the blocker search by using a minimum depth filter, which selects the minimum depth from the light source instead of an average depth of the blockers. Another issue is that the coarse shadow map can not take into account the distance of the shadow to the receiver in order to dilate the shadow map, which forces the user to set a fixed safe distance by hand.

However, despite its limitations, our technique is able to deliver perceptually correct penumbrae on controlled scenes, with a performance boost compared with PCSS, almost multiplying by 3 the performance obtained with large kernels.

CHAPTER 7

Improving Shadow Map Filtering with Statistical Analysis

Shadow maps are widely used in real-time applications. However, they cannot be filtered linearly as regular textures, which leads to severe aliasing. This problem has been attacked by methods that transform the depth values to allow approximate linear filtering and to approaches based on statistical analysis. Statistical methods, including *variance shadow maps* suffer from "light bleeding" artifacts. In this dissertation chapter we propose a new statistical filtering method for shadow maps, which approximates the cumulative distribution function (CDF) of depths with a power function or alternatively by a Gaussian CDF instead of bounding it with Chebyshev Inequality. This approximation significantly reduces "light bleeding" artifacts, keeping the same performance and spatial cost as the original variance shadow maps. The proposed method can also benefit from a layered approach, and can suppress the residual bleeding artifacts in complex scenes with few layers, needing only a fraction of the processing and storage cost of classical layered variance shadow maps. Like the original variance shadow maps, the algorithm is easy to implement on the graphics hardware and is fairly scalable.

7.1. Introduction

Shadow mapping is a popular and effective way of solving the shadowing problem. The depth buffer represents the occluder geometry in a discretized form. From this information, we need to reconstruct the distance from the

light source and the occluder in the direction of the shaded point in order to decide whether or not occlusion happens. The shadow test can also be imagined as a step like *visibility function* $v(z_r) = \epsilon(z_o - z_r)$ that is 0 if the occluder's distance from the light source, z_o, is smaller than the receiver distance z_r, and 1 otherwise. Unfortunately, the occluder distance is known only in the centers of the lexels, so this function must be reconstructed in other points. As the occluder geometry may involve high frequency variations, the occluder distance function can only be approximately reconstructed, and high frequency components may distort the reconstructed signal even at low frequencies, which leads to the well known shadow aliasing. Linear signal theory has a solution for the aliasing problem, which low-pass filters the signal to eliminate frequencies above the Nyquist limit. However, from signal processing point of view, shadow mapping is a non-linear operation as it contains a comparison operation represented by the *step function*. Thus filtering the depth values before the comparison does not work. A proper solution is postponing the filtering after the comparison, which is the basic idea of *percentage closer filtering*. However, this means that filtering can be executed only when the distance of the shaded point is available, so it should be repeated for every shaded point.

Techniques aiming at pre-filtering the shadow map belong to two main categories, those that transform the problem into a domain where linear signal processing becomes feasible, and those that use non-linear filtering operations based on statistical analysis.

Our method belongs to the category of statistical filtering and approximates the probability that the shaded point passes the depth test. This probability is obtained from the approximation of the cumulative distribution function of depths with a power function or by a Gaussian CDF instead of bounding it with the Chebyshev Inequality. The two moments of the depth's distribution are used to construct the Gaussian CDF, while the mean, minimum, and the maximum of the distribution are needed to fit the power function. Our approach is capable of highly reducing "light bleeding" artifacts, or even eliminating it for moderately complex scenes, with no penalty of performance or storage costs in the Gaussian case. Moreover, for very complex scenes, it can be converted to a layered approach (in the same way as layered variance shadow maps) for completely eliminating these artifacts. Very few layers are needed in this case, still outperforming LVSM both in performance and storage.

7.2. Method Overview

Instead of introducing layers (like LVSM) in the shadowing method, we propose an orthogonal research direction based on improving the evalua-

tion of the stored information for solving the "light bleeding" artifacts. Our method introduces two new different formulations of the VSM rather than using Chebyshev's Inequality for performing the filtering of the shadow map. These reformulations are based on 1) a Gaussian approximation and 2) a reconstruction of the cumulative distribution function. Reconstruction of CDF was also proposed in [Gru08] and for volumetric ambient occlusion [RSKU+10].

The application of probability theory techniques in shadow mapping is made possible by the following observation. We can make a few fundamental assumptions on the unknown visibility function:

- At $z = 0$, that is when the shaded point is at the light source, the visibility function is 1.
- At $z = \infty$, that is when the shaded point is very far from the light source, the visibility function is 0.
- The visibility function is monotonically decreasing, i.e. if it is father away, then there are more occluders between the light source and the receiver point, so the effect of the light source is possibly smaller, but definitely not larger.

From the point of view of statistics, *uncertainty* is involved in the depth buffer, since the depth values are known only in lexel centers. The goal is to guess the visibility function at arbitrary point with minimizing this inherent uncertainty. Unlike in signal processing, we are not constrained by linear operations thus by the selection of a proper estimation, the depth testing can be made more robust.

In order to propose a practically useful statistical approach, we need to consider two additional requirements.

- Unoccluded planar objects are expected to be fully lit, so the visibility function must give value 1 for the average depth value. This is similar to *depth biasing* in classical shadow mapping and means that statistical probability of occlusion must be corrected to avoid self-occlusions.
- As the variables needed for the estimation will be obtained with filtering the depth buffer, we have to define them in a way which allows separable filtering since separable filtering has better complexity than general non-separable filtering.

In order to compute the visibility function, we start with the probability of no occlusion $P(z_o \geq z_r)$ and bias it to avoid self occlusions. Self occlusions are eliminated for planar surfaces if average depth $\tilde{z}_o$ is associated with visibility 1, thus our proposed visibility approximation is:

$$v(z_r) = \frac{P(z_o \geq z_r)}{P(z_o \geq \tilde{z}_o)}, \quad \text{if } z_r > \tilde{z}_o \text{ and 1 otherwise.} \tag{7.1}$$

The probabilities are computed from the cumulative probability distribution $F(z) = P(z_o < z)$ of random variable z_o:

$$v(z_r) = \frac{1 - F(z_r)}{1 - F(\tilde{z}_o)}, \quad \text{if } z_r > \tilde{z}_o \text{ and 1 otherwise.} \tag{7.2}$$

7.2.1. Gaussian cumulative distribution

Let $F(z; \tilde{z}_o, \sigma^2)$ be the cumulative depth distribution function and let $\tilde{z}_o$ and σ^2 be the mean and the variance, respectively. In contrast to Chebysev's Inequality, our method will approximate the PDF with a Gaussian distribution function:

$$F(z; \tilde{z}_o, \sigma^2) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{z - \tilde{z}_o}{\sigma\sqrt{2}}\right)\right], \tag{7.3}$$

where $\operatorname{erf}(x)$ is the *error function*, which can be approximated by the following expression [Win08]:

$$\operatorname{erf}(x) \approx \sqrt{1 - \exp\left(-x^2 \frac{4/\pi + ax^2}{1 + ax^2}\right)}$$

where

$$a = \frac{8(\pi - 3)}{3\pi(4 - \pi)} \approx 0{,}140012$$

Substituting the Gaussian CDF into our biased visibility formula (equation 7.2), we get:

$$v(z_r) = 1 - \operatorname{erf}\left(\frac{z_r - \tilde{z}_o}{\sigma\sqrt{2}}\right), \quad \text{if } z_r > \tilde{z}_o \text{ and 1 otherwise.} \tag{7.4}$$

7.2.2. Reconstruction of the cumulative distribution with the Power function

Our second approach is based on reconstructing the cumulative distribution from three different values that are obtained by filtering an area of the depth buffer. The larger the area of interest, the more blurred the shadows. This filtering operation is configured as a separable kernel that calculates the minimum value $z_{\text{mín}}$, the maximum value $z_{\text{máx}}$, and the mean $\tilde{z}_o$ of the depth values.

Our cumulative distribution $F(z)$ must be zero if $z \leq z_{\text{mín}}$, equal to 1 if $z \geq z_{\text{máx}}$, and non-decreasing in between. The following normalized depth parameter is introduced for notational simplicity:

$$t = \frac{z - z_{\text{mín}}}{z_{\text{máx}} - z_{\text{mín}}} = \frac{z - z_{min}}{\Delta z},$$

where $\Delta z = z_{\text{máx}} - z_{\text{mín}}$. Using the normalized parameter, the cumulative distribution function must be zero if $t \leq 0$, and 1 if $t \geq 1$, and non-decreasing in the $[0, 1]$ interval. Taking into account the distribution of all possible shadow map values, the cumulative distribution function may be a step function at $t_{\text{mín}}$ and $t_{\text{máx}}$ at the two extremes, respectively. Therefore, our goal is to find a function that increases from 0 to 1 and has the flexibility to adapt to the two extreme cases. Considering these, we propose to use the function t^β where β is the parameter of data fitting (see Figure 7.1). Thus, the cumulative distribution is

$$F(z) = t^\beta \quad \text{where} \quad z(t) = t\Delta z + z_{\text{mín}}. \tag{7.5}$$

Let us consider the constraint on the mean:

$$\tilde{z}_o = \int_{z_{\text{mín}}}^{z_{\text{máx}}} z \mathrm{d}F = \int_0^1 z(t) \frac{\mathrm{d}F}{\mathrm{d}t} \mathrm{d}t =$$

$$\int_0^1 (t\Delta z + z_{\text{mín}}) \beta t^{\beta-1} \mathrm{d}t = \frac{\beta \Delta z}{\beta + 1} + z_{\text{mín}}$$

Solving this equation for β, we get:

$$\beta = \frac{\tilde{z}_o - z_{\text{mín}}}{z_{\text{máx}} - \tilde{z}_o}. \tag{7.6}$$

The visibility formula needs the computation of the cumulative distribution for the expected depth, which corresponds to

$$\tilde{t} = \frac{\tilde{z}_o - z_{\text{mín}}}{\Delta z} = \frac{\beta}{\beta + 1}.$$

The visibility function is obtained from equation 7.2:

$$v(t) = \frac{1 - t^\beta}{1 - \left(\frac{\beta}{\beta+1}\right)^\beta}, \quad \text{if } t > \tfrac{\beta}{\beta+1} \text{ and 1 otherwise.} \tag{7.7}$$

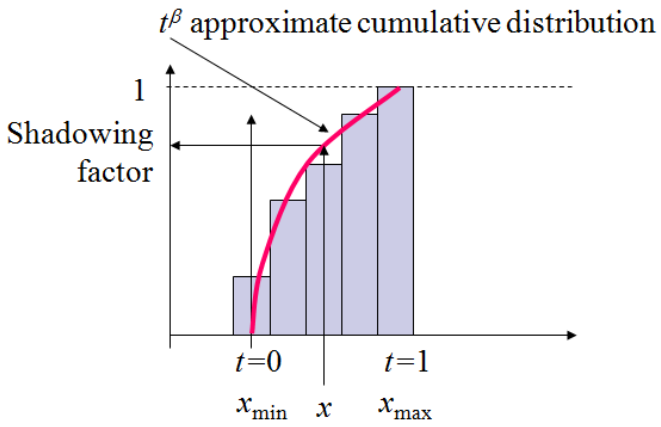


Figure 7.1: Approximation of the cumulative distribution function with t^β.

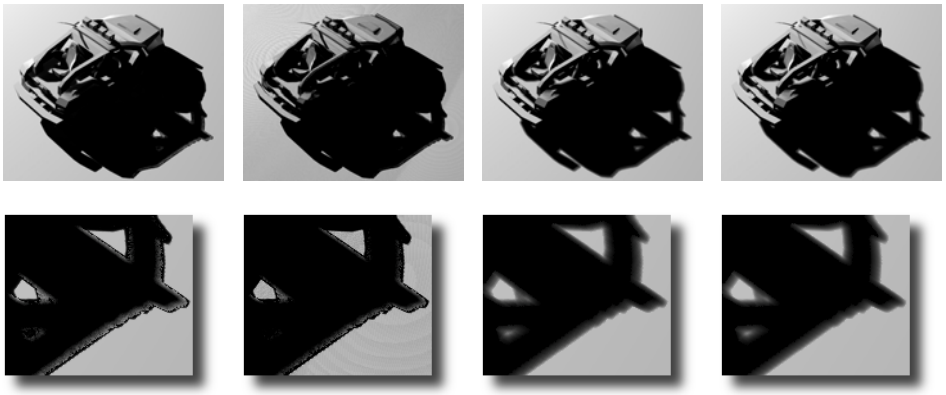
7.3. Results

This section presents visual quality and performance tests to compare our two new filtering approaches with the Chebyshev's Inequality approximation used in the traditional variance shadow maps. All quality and performance tests were generated on an Intel Core2 Quad Q9550 CPU @ 2.83Ghz with a NVIDIA GeForce 280GTX using Direct3D 10. Performance data were measured on three very different test scenes: the Car scene, Spheres scene and the Chairs scene.

Figures 7.3, 7.4 and 7.6 show a visual comparison of our approaches and existing methods such as VSM and ESM. It can be seen how our approaches are able of highly reducing, or even eliminating, these artifacts. In general terms, the Gaussian CDF approach is more robust than the Power CDF, which is still presenting artifacts in some cases.

Floating point precision of the shadow map is a determinant factor for both performance and visual quality. VSM and our Gaussian approximation use a shadow map with two floating point channels, while the proposed Power CDF method needs at least three channels for storing the mean, minimum and maximum values. As shown in Figure 7.2, VSM and our Gaussian approach are very sensitive to floating point precision. However, the Power CDF reconstruction is able to render artifact-free soft shadows even with 16 bits per channel. This enables for improving performance, as shown in Table 7.1.

Table 7.1 compares performance obtained with our approaches and with the Chebyshev's Inequality approximation. All times were measured using

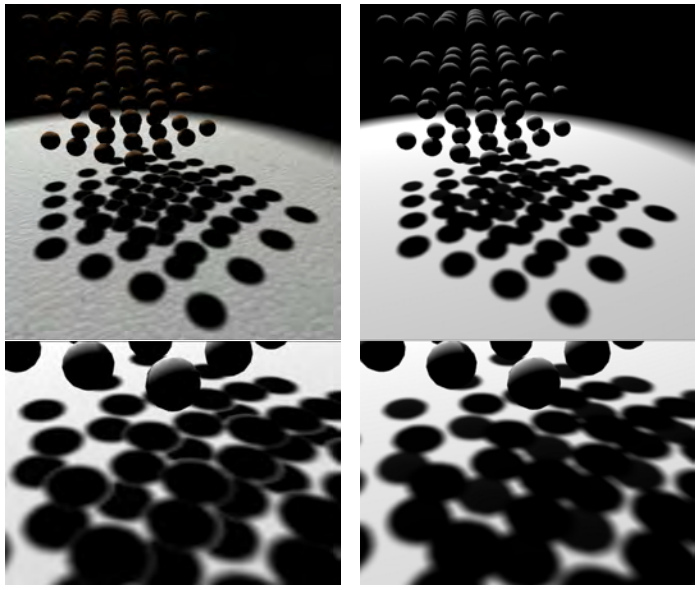


(e) 16 bit Chebyshev's Inequality. (f) 16 bit Gaussian CDF. (g) 16 bit CDF reconstruction. (h) 32 bit CDF reconstruction.

Figure 7.2: Visual quality comparison using different shadow map floating point precisions. The bottom row presents close-ups of the upper shadows. Both VSM and our Gaussian CDF approximation introduce artifacts when using 16 bits per channel. However, our Power CDF reconstruction technique is able of properly rendering anti-aliased shadows in both cases.

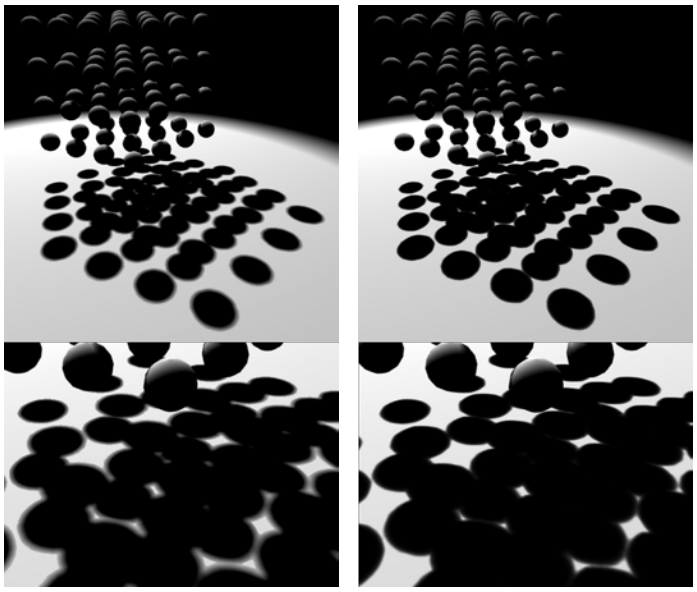
	512x512	1024x1024	2048x2048
Chebyshev	481 fps	342 fps	142 fps
Exponential	495 fps	392 fps	165 fps
Gaussian CDF	456 fps	331 fps	142 fps
CDF recons. 32bits	420 fps	243 fps	82 fps
CDF recons. 16bits	468 fps	381 fps	160 fps

Table 7.1: Performance table of our two new approaches compared to the Chebyshev's Inequality at different shadow map resolutions. The Car scene was rendered at fullHD (1920×1080). Chebyshev's and the Gaussian CDF approaches used 32 bits per channel in the shadow maps.



(a) Standard VSM using Chebyshev's Inequality.

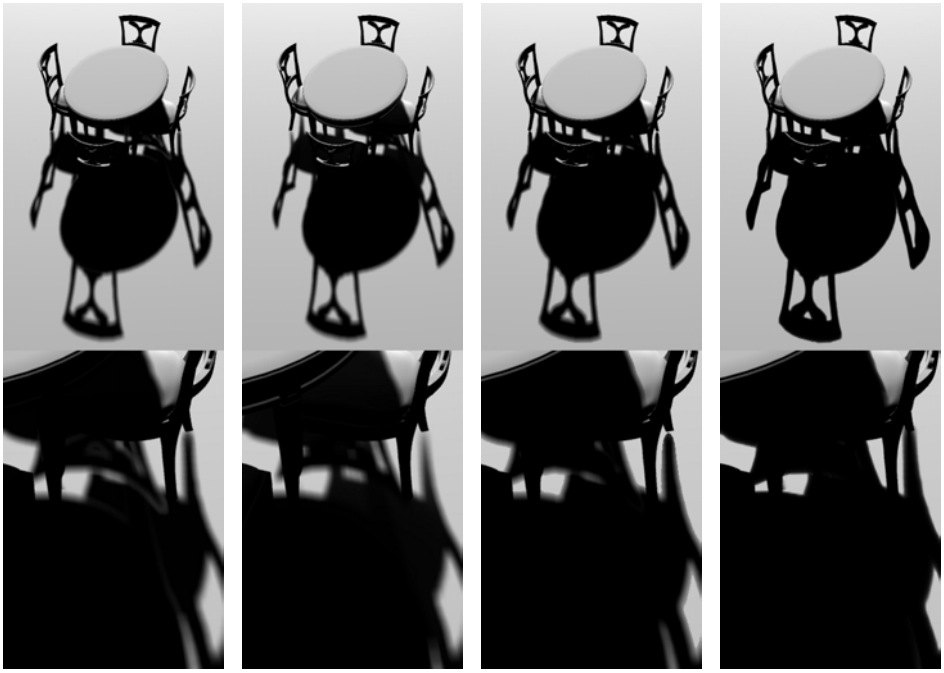
(b) Exponential shadow maps.



(c) Gaussian CDF approximation.

(d) Power CDF approximation.

Figure 7.3: Visual quality comparison on the spheres scene.

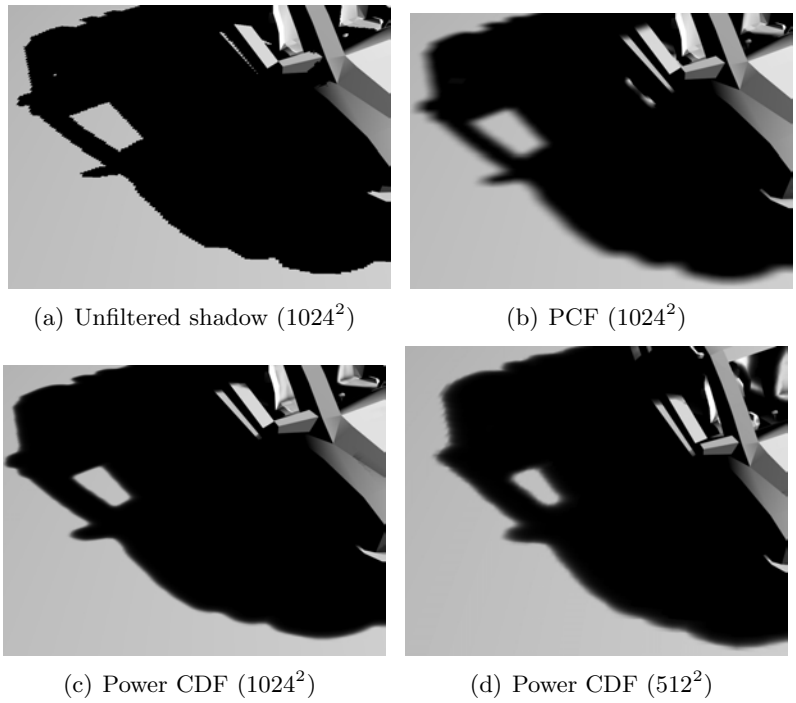


(a) VSM. (b) Exponential shadow maps. (c) Gaussian CDF approximation. (d) Power CDF approximation.

Figure 7.4: Visual quality comparison on the chairs scene. The white arrow in subfigure (a) shows the light bleeding artifacts caused by the Chebyshev's Inequality.

the Car scene at a screen resolution of 1920×1080. The reason of using a single scene for all tests is that the performance of these methods is independent of the geometry complexity, but depend on just the screen and shadow map resolutions. It can be seen that our Gaussian CDF approach is as fast as the traditional VSM approximation. For our Power CDF method, we used two different shadow map precisions. At higher resolution this method is slower than the other two, because this technique needs to store more information in the shadow map (mean, minimum and maximum values), which has an impact on graphics memory bandwidth when performing the filtering.

It is important to note that, as it can be seen in Figures 7.3, 7.4 and 7.6, our Power CDF method is able of performing sharper antialiased shadows compared to other methods (like VSM, ESM or even to our Gaussian CDF approximation) using the same amount of texture samples and the same filtering kernel size. Figure 7.3 shows an analysis of the shadow sharpness of our Power CDF function compared to percentage closer filtering. This property of our method allows us to use lower resolution shadow maps while providing similar antialiasing sharpness (see Figure 7.3 for an example).



(a) Unfiltered shadow (1024^2) (b) PCF (1024^2)

(c) Power CDF (1024^2) (d) Power CDF (512^2)

Figure 7.5: Visual quality comparison of shadows sharpness comparing PCF with the Power CDF with the same number of samples and filtering kernel size. Figures (a), (b) and (c) uses a 1024x1024 shadow map. Figure (d) uses a 512x512 shadow map.

As shown in the results compared to the exponential shadow maps, our algorithm is not sensitive the light leaking artifacts introduced by the multiple distant occluders. For the sake of clarity, In Figure 8 the light leaking artifacts introduced by ESM is highlighted with a red circle.

7.4. Conclusion

We have developed two new shadow filtering approaches based on variance shadow maps. These methods are based on replacing the Chebyshev's Inequality for finding an upper-bound for the depth distribution of the shadow map by two new approximations: a power function or alternatively a Gaussian CDF. This allows for enhancing the visual results of shadows by reducing "light bleeding" artifacts. In the case of the Gaussian approximation, no penalty in performance or storage costs is introduced. For the power CDF, performance is even improved by using a 16 bit floating point precision shadow map.

The two proposed techniques have both advantages and disadvantages.

(a) Standard VSM using Chebyshev's Inequality.

(b) Exponential shadow maps.

(c) VSM using our Gaussian CDF approximation.

(d) VSM using our approach for reconstructing the CDF.

Figure 7.6: Visual quality comparison on the car scene. Light leaking artifact introduced by ESM is highlighted with a red circle in Figure (b). Our techniques are able to remove light leaking artifacts on both VSM and ESM.

On the one hand, the Gaussian CDF approach is able to filter shadows while highly reducing light leaking artifacts of the classical VSM. On the other hand, the Power CDF reconstruction requires more texture storage but is better in reducing the light leaking artifacts. Moreover, the Power CDF reconstruction generates sharper anti-aliased shadows (as shown in Figure 7.5), which allows the use of lower resolution shadow maps.

In conclusion, the new technique presented in this chapter produces good quality anti-aliased shadows at high performance, and are very scalable for complex scenes.

CHAPTER 8

Conclusions and future work

This thesis presents a number of techniques and methods for improving performance and visual quality when rendering interactive environments. The conclusions of the research done in this thesis and the future research directions are presented in the section.

Throughout the different contributions presented, the aim of this work has been to enhance the interactive visualization of natural scenes in different fields such as: level of detail, scene management, illumination and shadowing.

This dissertation started by presenting our view-dependent multiresolution model for the foliage designed for exploiting massively parallel architectures, which runs entirely on the GPU. Our results demonstrate that this approach outperforms previous approaches that are not designed to run completely on the graphics hardware. Moreover, we have proposed a method for efficiently managing massively populated level of detail scenes in Chapter 4. We demonstrate that this approach is important for large multiresolution scenes to prevent bottlenecks when extracting the level of detail for hundreds or even thousands of objects. In Chapter 5, we propose a new real-time illumination method for the foliage, based on a dual depth map approach, designed for taking into account the sparse nature of the leaves. This method is able to increase realism of trees by taking into account the position of the leaves inside of the foliage for approximating their illumination. Finally, two techniques built on top of shadow mapping are proposed. Firstly, Chapter 6 introduces a new soft shadow mapping method which is able of calculating visually plausible soft shadows in screen space. The advantages of this approach are that it naturally fits on deferred shading renderers and that they are faster to calculate than existing methods.

Secondly, Chapter 7 introduces two new methods for improving shadow map filtering based on statistical information. We demonstrate that these algorithms are capable of rendering antialiased shadows and that they are less prone to "light leaking" artifacts compared to existing methods.

This chapter is organized as follows. Firstly, we conclude on the contributions offered by the different proposals. Then, we outline ideas for future work. Finally, we provide a list of publications related to this Ph.D. dissertation and the research projects that have enabled the development of this thesis work.

8.1. Conclusions

First, a state of the art covering the different areas related with the themes treated in this thesis is presented in Chapter 2. These areas include multiresolution level of detail, LoD scene management, real-time illumination of trees and shadow mapping techniques.

Chapter 3 presents a technique for adaptively changing the level of detail of the foliage in an efficient way, using the GPU. In order to minimize popping artifacts, we use a view-dependent technique to select the less visible parts of the foliage from the observer's point of view. This technique allows us to decrease the geometrical complexity of the less visible parts of the foliage.

In order to optimally perform this operations, the multiresolution model has been designed as a parallelizable algorithm which runs efficiently on the GPU. The benefits of executing our algorithm on the graphics hardware are:

- The cost of the algorithm is reduced by one order of magnitude and is executed more efficiently.
- Unlike other solutions, no data is transferred through the PCIe bus, which would become a bottleneck.

As shown in Chapter 3 in Figure 3.8, our algorithm outperforms other solutions based on the CPU while conserving the visual aspect of the foliage even when drastically reducing the amount of polygonal complexity, as shown in Figure 8.1.

Chapter 4 introduces our technique for efficiently managing large groups of level of detail objects. Our algorithm exploits the observation that, in massive sets of LoD objects, it is highly probable that lots of them share a similar level of detail. Moreover, the further the objects are from the observer, the more difficult is to distinguish the visual differences between similar levels of detail.

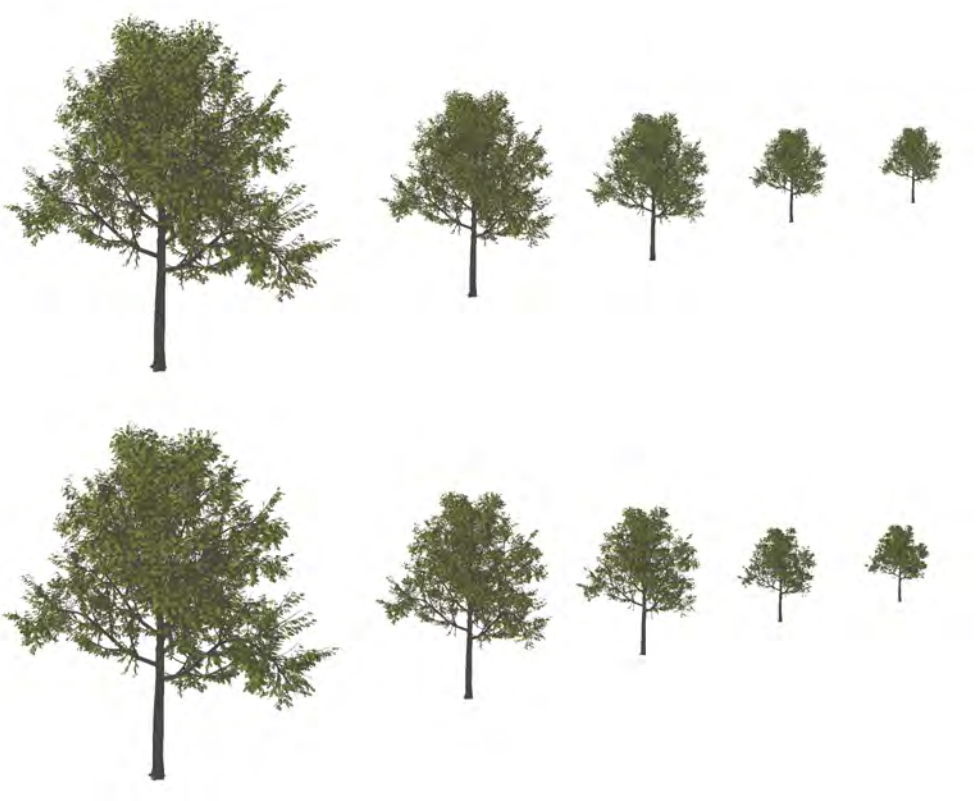


Figure 8.1: The upper row shows a tree at different distances, rendered without level of detail. The lower row shows the same tree at the same distances rendered with our multiresolution method, using the following reduction factors: 61 %, 48 %, 38 %, 26 % and 16 % respectively.

This is important for performance reasons because the process of extracting the level of detail has a computational cost associated to it. Therefore, changing the level of detail of massive sets of LoD objects can collapse the visualization process and become the bottleneck of the application.

For alleviating this problem, our method "caches" previously calculated LoDs and assigns them to other objects that need a similar level of detail. This algorithm drastically reduces the amount of times the LoD extraction process takes place in the scene. As a consequence, our technique improves performance and prevents the LoD extraction process to become the bottleneck of the graphics application.

As demonstrated in Chapter 4 the LoD Manager efficiently manages level of detail changes and minimizes CPU consumption. In fact, when dealing with scenes with a high count of LoD objects (Figure 4.1), the computational cost caused by the LoD process becomes the bottleneck of the application, reducing the performance and making it unsuitable for interactive applications. Our method solves this problem and keeps the bottleneck of the application in the rendering stage, instead of in the extraction process.

Foliage illumination is a difficult problem due to the complex interaction of light inside the foliage. Moreover, light coming from all directions also need to be taken into account. Having this into mind, Chapter 5 presents a technique for representing the direct and indirect contributions of light to enhance the quality of the lighting approximation for the foliage.

This technique is a novel approach based on capturing the shape of the foliage with two depth maps. Then, the amount of illumination received at a certain point inside of the foliage is approximated by taking into account its relative position inside the volume defined by the two depth maps.

This solution provides a low frequency illumination of the foliage in real-time. In addition, we rely on two different techniques for calculating the high-frequency illumination term. The first one is based on a preprocess that calculates the ambient occlusion term computing the visibility of the leaves inside the foliage. The second one is based on a screen space technique for locally calculating the occlusion of each leaf.

As shown in Figure 8.2, this technique is able to represent high quality illumination for the foliage in real-time.

Chapter 6 introduces a novel technique for calculating soft shadows in real-time. The originality of this technique relies on the fact that the soft shadows are calculated in screen space. The technique is based on applying an anisotropic Gaussian blur of variable size, depending on the amount of penumbrae needed for each pixel on the screen. The size of the penumbra is calculated by taking into account the distance of the shadow caster and the shadow receiver, as well as the size of the light source.

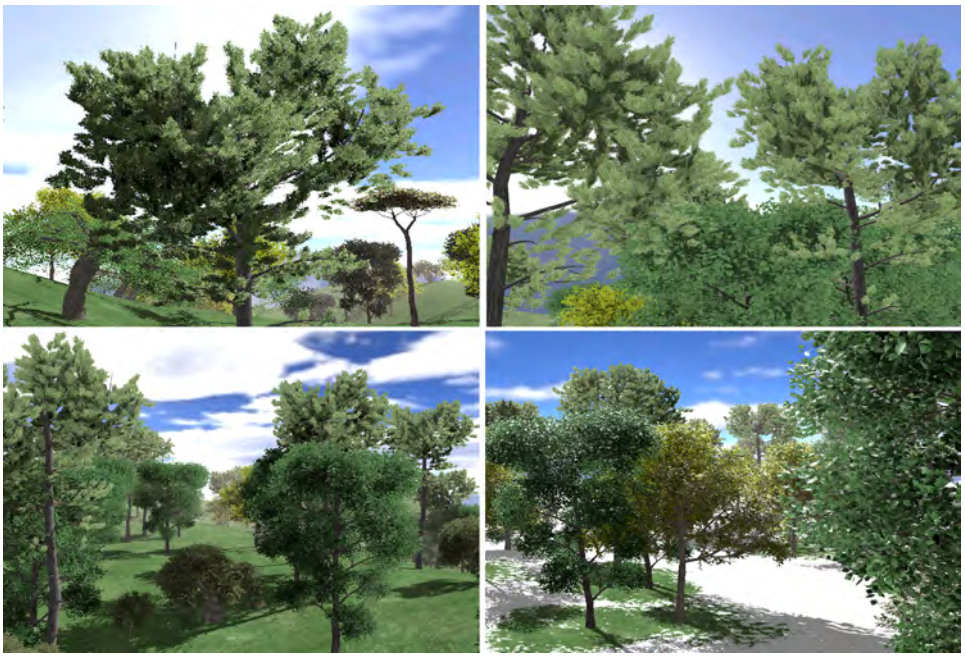


Figure 8.2: Forest scenes with our foliage illumination approach.

Figure 8.3: Real-time soft shadows generated with our algorithm in screen-space.

This technique is able to render highly smoothed soft shadows in real-time and performs better than existing techniques, such as back-projection or percentage-closer soft shadows, due to the fact that we use a separable Gaussian filter, with a computational complexity of $O(n + n)$ instead of $O(n^2)$. Figure 8.3 shows the visual quality results of our technique.

Chapter 7 introduces a new shadow filtering algorithm for shadow mapping. The technique is based on exploring the capabilities of using CDF approximation in the shadow filtering estimation. The results of this ap-

proach show that light bleeding artifacts are highly reduced compared to other techniques, such as VSM, and performance is still very good because it consists of just evaluating one formula.

Two different techniques are presented in this section. In essence, they replace the Chebyshev's inequality estimation used in techniques like VSM or LVSM by more precise approximations. The first approximation is based on the Gaussian cumulative distribution while the second one is based on reconstructing the cumulative distribution using a power function.

As seen in Figure 8.4 these approximations result on a reliable shadow filtering technique that is able to drastically reduce light leaking artifacts while keeping the high performance associated to these kind of shadow filtering techniques.

8.2. Future work

We have various lines of research for future work. For our view-dependent LoD approach for the foliage, we propose the use of OpenCL rather than CUDA for the GPU implementation. This API allows for direct resource sharing between the compute and rendering contexts, which increases performance. Another future line of research would be to include in the approach the physical effects of the environment, such as the effects of the wind, into the multiresolution scheme, and the possibility of calculating the physics on the GPU using CUDA or OpenCL.

Regarding the LoD management method presented in Chapter 4, we propose to explore better heuristics to allow for minimizing the amount of LoD changes and thus, maximize performance. Moreover, it would be also interesting to extend the LoD Manager in order to deal with animated meshes. Calculating the animation matrices for a massive amount of animated characters is time consuming and can easily become the bottleneck of the application. To solve this problem, the same principles that the LoD Manager uses for reusing LoD calculations could be applied for reusing calculations related to the animation system.

Chapter 5 presents a technique for real-time foliage illumination using two shadow maps for approximating the volume the leaves in order to determine the occlusion of a leaf inside the foliage. This approximation is estimated by using two depth maps. As future work, it would interesting to take more directions into account and study how the new approach affects to performance as well as the benefits obtained. Moreover, we propose to study how our technique performs on trees with other types of leaves and how the algorithm should be changed in the case that the technique is not directly applicable to them.

(a) Standard VSM using Chebyshev's Inequality.

(b) Exponential shadow maps.

(c) VSM using our Gaussian CDF approximation.

(d) VSM using our approach for reconstructing the CDF.

Figure 8.4: Visual quality comparison on the car scene. Light leaking artifact introduced by ESM is highlighted with a red circle in Figure (b). Our techniques are able to remove light leaking artifacts on both VSM and ESM.

Regarding the screen-space soft shadowing technique presented in Chapter 6, the main idea of this method is to apply a Gaussian filter of variable size to the hard-edged shadows in screen space. Therefore, the best case of our approach is when the shadow receiver is a plane and the observer is looking to it in a perpendicular direction. However, although the technique can perform well over non-planar geometry it could be difficult to create artifact-free penumbrae with very complex shadow receivers. Thus, as a future line of research we propose to enhance the technique in order to make it more robust when projecting shadows over complex/irregular geometry. Moreover, as shown in Chapter 6, the algorithm is designed as a multi-layered method in order to handle hidden shadows. An interesting future line of research would be to study how to modify the algorithm in order to eliminate this requirement and how could it be converted into a single layered approach.

Finally, Chapter 7 presents a technique for shadow map filtering which considerably reduces the amount of light leaking artifacts present in existing techniques, such as variance shadow maps or exponential shadow maps. While our technique is able to highly decrease the amount of leaking artifacts, it's still not able to completely remove them for very complex shadow casters. Therefore the future work proposed for this topic is to study new formulations to completely remove artifacts in all situations. Finally, a common requirement of shadow filtering techniques which use pre-filtering to generate antialiased shadows is that both the shadow casters and the shadow receivers must be stored in the shadow map. This requirement might affect performance and relaxing it would be an interesting line of research.

8.3. Publications

For assessing the different works presented in throughout this Ph.D. dissertation, this section lists the different publications obtained while developing this thesis.

8.3.1. Book Chapters

- Jesús Gumbau Portalés, Miguel Chover Sellés, Mateu Sbert. *Screen Space Soft Shadows.* GPU Pro. Chapter 4. Part VII. AK Peters 2010. ISBN: 978-1-56881-472-8. Editor Wolfgang Engel. [GCS10]

8.3.2. Journals

- Jesús Gumbau Portalés, Miguel Chover Sellés, Inmaculada Remolar Quintana, Cristina Rebollo Santamaría. COMPUTER AND GRAPH-

ICS. *View-dependent pruning for real-time rendering of trees.* (Accepted, in press).

- Carlos González Ballester, Jesús Gumbau Portalés, Miguel Chover Sellés, Jose Francisco Ramos Romero, Ricardo Javier Quirós Bauset. COMPUTER-AIDED DESIGN. *User-assisted simplification method for triangle meshes preserving boundaries.* Num. 41. pp. 1095-1106. 2009. [GGC+09]

- Cristina Rebollo Santamaría, Inmaculada Remolar Quintana, Miguel Chover Sellés, Jesús Gumbau Portalés, Oscar Enrique Ripollés Mateu. JOURNAL OF COMPUTERS. *A Clustering Framework for Real-Time.* Num. 4, vol 2. pp. 57-67. 2007. [RRC+07]

- Oscar Enrique Ripollés Mateu, Jose Francisco Ramos Romero, Miguel Chover Sellés, Jesús Gumbau Portalés, Ricardo Javier Quirós Bauset. WSEAS TRANSACTIONS ON COMPUTERS. *A Tool for the Creation and management of level-of.detail models 3D applications.* Num. Issue 7, vol 7. pp. 1020-1029. 2008. [RRC+08]

- Oscar Enrique Ripollés Mateu, Miguel Chover Sellés, Jesús Gumbau Portalés, Jose Francisco Ramos Romero, Anna Puig Centelles. GRAPHICAL MODELS. *Rendering continuous level-of-detail meshes by Masking Strips.* Num. 71. pp. 184-195. 2009. [RCG+09]

8.3.3. International Conferences

- Jesús Gumbau Portalés, Oscar Enrique Ripollés Mateu, Miguel Chover Sellés. *LODManager:a framework for rendering multiresolution models in real-time appplications.* WSCG'2007. Czech Republica: 29-01-2007. 2007 University of West Bohemia. ISBN: 978-80-86943-02-2. [GRC07]

- Jesús Gumbau Portalés, Miguel Chover Sellés, Cristina Rebollo Santamaría, Inmaculada Remolar Quintana. LECTURE NOTES IN COMPUTER SCIENCE. *Real-Time Illumination of Foliage Using Depth Maps.* Num. 5102. pp. 136-145. 2008. [GCRR08]

- Cristina Rebollo Santamaría, Inmaculada Remolar Quintana, Miguel Chover Sellés, Jesús Gumbau Portalés. LECTURE NOTES IN COMPUTER SCIENCE. *Hardware-Oriented Visualisation of Trees.* Num. 4263. pp. 374-383. 2006. [RRCG06]

- Oscar Enrique Ripollés Mateu, Jesús Gumbau Portalés, Miguel Chover Sellés, Jose Francisco Ramos Romero, Anna Puig Centelles. *View-Dependent Multiresolution Modeling on the GPU.* WSCG'2009. Plzen:

02-02-2009. 2009 University of West Bohemia. ISBN: 978-80-86943-94-7. [RGC+09]

- Jose Francisco Ramos Romero, Miguel Chover Sellés, Jesús Gumbau Portalés, Ricardo Javier Quirós Bauset. *GEOTOOL: A Tool to Manage Polygonal Meshes and Build Multiresolution Models*. Recent Advances in Systems, Communications Computers. Hangzhou, China: 06-04-2008. 2008 WSEAS. ISBN: 978-960-6766-61-9. [RCGQ08]

- Cristina Rebollo Santamaría, Jesús Gumbau Portalés, Oscar Enrique Ripollés Mateu, Miguel Chover Sellés, Inmaculada Remolar Quintana. *Fast Rendering of Leaves*. Computer Graphics and Imaging. Innsbruck, Austria: 13-02-2007. 2007 Acta Press. ISBN: 978-0-88986-644-7. [RGR+07]

8.3.4. National Conferences

- Jesús Gumbau Portalés, Miguel Chover Sellés, Inmaculada Remolar Quintana. *Simplificación interactiva de vegetación usando el Hardware Gráfico*. CEIG 09 Congreso Español de Informática Grafica. San Sebastian: 09-09-2009. 2009 Eurographics Association. ISBN: 978-3-905673-72-2. [GCR09]

- Jesús Gumbau Portalés, Miguel Chover Sellés. *Representación de sombras suaves en tiempo real usando mapas de distancia*. CEIG 2008. Barcelona: 03-09-2008. 2008 Universitat Politècnica de Catalunya. ISBN: 978-3-905673-69-2. [GC09]

- Nicolau Sunyer Ferrer, Jesús Gumbau Portalés, Miguel Chover Sellés, Mateu Sbert Casasayas. *Screen Space Obscurances*. CEIG 09. San Sebastian: 09-09-2009. 2009 Eurographics Association. ISBN: 978-3-905673-72-2. [SGCS09]

- Carlos González Ballester, Jesús Gumbau Portalés, Miguel Chover Sellés, Pascual Castelló Boscá. *Simplifiación de mallas para juegos*. XVII Congreso Español d Informática Gráfica. Zaragoza: 11-09-2007. 2007 Thomson. ISBN: 978-84-9732-595-0. [GGCC07]

Bibliography

[ADM+08] Thomas Annen, Zhao Dong, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Real-time, all-frequency shadows in dynamic scenes. *ACM Trans. Graph.*, 27(3):1–8, 2008.

[AMA02] Tomas Akenine-Möller and Ulf Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 297–306, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[AMB+07] Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Convolution shadow maps. In *Rendering Techniques 2007: Eurographics Symposium on Rendering*, pages 51–60, Grenoble, France, 2007. Eurographics.

[Arv04] J. Arvo. Tiled shadow maps. In *Proceedings of Computer Graphics International 2004*, pages 240–247, 2004.

[ASVNB00] Carlos Andújar, Carlos Saona-Vázquez, Isabel Navazo, and Pere Brunet. Integrating occlusion culling with levels of detail through hardly-visible sets. *Computer Graphics Forum (Proceedings of Eurographics '00)*, 3:499–506, 2000.

[Bav08] Louis Bavoil. Advanced soft shadow mapping techniques. *Game Developers Conference*, 2008.

[Ble90] Guy E. Blelloch. Prefix sums and their applications. *Synthesis of Parallel Algorithms*, 1990.

[BS01] Stefan Brabec and Hans-Peter Seidel. Single sample soft shadows using depth maps. In *Proceedings of Graphics Interface. pp. 219–228.*, 2001.

[Bun05] M. Bunnell. *Dynamic Ambient Occlusion And Indirect Lighting. GPU Gems 2.* 2005.

[CBL99] Chun-Fa Chang, Gary Bishop, and Anselmo Lastra. Ldi tree: a hierarchical representation for image-based rendering. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 291–298, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[CD03] Eric Chan and Frédo Durand. Rendering fake soft shadows with smoothies. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 208–218, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[CH05] R. L. Cook and J. Halstead. Stochastic pruning. *Proceedings of Eurographics Workshop on Natural Phenomena*, 2005.

[CHPR07] Robert L. Cook, John Halstead, Maxwell Planck, and David Ryu. Stochastic simplification of aggregate detail. *ACM Trans. Graph.*, 26, July 2007.

[CSCF08] P. Castelló, M. Sbert, M. Chover, and M. Feixas. Viewpoint-based simplification using f-divergences. *Information Sciences*, 11(178):2375–2388, 2008.

[DCSD02] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 219–226. IEEE Computer Society, 2002.

[DDGM00] Robert Danforth, Andrew Duchowski, Robert Geist, and Elizabeth Mcaliley. A platform for gaze-contingent virtual environments. In *In Smart Graphics (2000 AAAI Spring Symposium, Technical Report SS-00-04), (Menlo Park, CA, 2000), AAAI*, pages 66–70, 2000.

[DDSD03] X. Décoret, F. Durand, F. Sillion, and J. Dorsey. Billboard clouds for extreme model simplification. In *Proceedings of the ACM Siggraph*. ACM Press, 2003.

[DF94] George Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using backprojection. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 223–230, New York, NY, USA, 1994. ACM.

[DL06] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 161–165, New York, NY, USA, 2006. ACM.

[DVS03] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. In *ACM Transactions on Graphics*, pages 657–662, 2003.

[DZYJ10] Qingqiong Deng, Xiaopeng Zhang, Gang Yang, and Marc Jaeger. Multiresolution foliage for forest rendering. *Comput. Animat. Virtual Worlds*, 21(1):1–23, 2010.

[Eng08] Wolfgang Engel. Designing a renderer for multiple lights - the light pre-pass renderer. $ShaderX^7$: *Advanced rendering techniques*, 2008.

[Fer05] Randima Fernando. Percentage-closer soft shadows. In *In SIGGRAPH 05: ACM SIGGRAPH 2005 Sketches, ACM*, 2005.

[FFBG01] R. Fernando, S. Fernandez, K. Bala, and D.P. Greenberg. Adaptive shadow maps. In *SIGGRAPH '01*, pages 387–390, New York, USA, 2001.

[FFD03] Leaves Oliver Franzke, Oliver Franzke, and Oliver Deussen. Accurate graphical representation of plant. In *In Plant Modelling and Applications (2003), de Reffye P.* Ed.), Springer-Verlag, 2003.

[FMU05] A Fuhrmann, S Mantler, and E Umlauf. Extreme model simplification for forest rendering. *in Proceedings of EGWNP - Eurographics Workshop on Natural Phenomena*, 2005.

[FS93] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics*, 27(Annual Conference Series):247–254, 1993.

[GB99] Enrico Gobbetti and Eric Bouvier. Time-critical multiresolution scene rendering. In *Proceedings IEEE Visualization*, pages 123–130, Conference held in San Francisco, CA, USA, October 1999. IEEE Computer Society Press.

[GC09] Jesús Gumbau and Miguel Chover. Representación de sombras suaves en tiempo real usando mapas de distancia. In *CEIG 2008. Barcelona: 03-09-2008. 2008 Universitat Politècnica de Catalunya.*, 2009.

[GCR09] Jesús Gumbau, Miguel Chover, and Inmaculada Remolar. Simplificación interactiva de vegetación usando el hardware gráfico. In *CEIG 09 Congreso Español de Informática Grafica. San Sebastian. Eurographics Association.*, 2009.

[GCRR08] Jesús Gumbau, Miguel Chover, Cristina Rebollo, and Inmaculada Remolar. Real-time illumination of foliage using depth maps. In *Proceedings of the 8th international conference on Computational Science, Part II*, ICCS '08, pages 136–145, Berlin, Heidelberg, 2008. Springer-Verlag.

[GCS10] Jesús Gumbau, Miguel Chover, and Mateu Sbert. Screen pace soft shadows. In *GPU Pro*, pages 477–491, 2010.

[GGC+09] C. González, J. Gumbau, M. Chover, F. Ramos, and R. Quirós. User-assisted simplification method for triangle meshes preserving boundaries. *Comput. Aided Des.*, 41:1095–1106, December 2009.

[GGCC07] Carlos González, Jesús Gumbau, Miguel Chover, and Pascual Castelló. Simplifiación de mallas para juegos. In *XVII Congreso Español d Informática Gráfica. Zaragoza.*, 2007.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.

[GMN05] G. Gilet, A. Meyer, and F. Neyret. Point-based rendering of trees. In P. Poulin E. Galin, editor, *Eurographics Workshop on Natural Phenomena*, 2005.

[GP08] I. García and G. Patow. Igt: inverse geometric textures. *ACM Transaction Graphics*, 27(5):1–9, 2008.

[GRC07] Jesús Gumbau, Oscar Ripollés, and Miguel Chover. Lodmanager: A framework for rendering multiresolution models in real-time appplications. In *WSCG'2007*, 2007.

[Gru08] H. Grun. Approximate cumulative distribution function shadow mapping. In Wolfgang Engel, editor, *Shader X 7*. Course Technology, 2008.

[GS01] Jan M. Geusebroek and Arnold W. M. Smeulders. Fast anisotropic gauss filtering. *IEEE Transactions on Image Processing*, 12, 2001.

[GSSK05] I. García, M. Sbert, and L. Szirmay-Kalos. Leaf cluster impostors for tree rendering with parallax. In *Proc. Eurographics 2005 (Short Presentations)*. Eurographics, 2005.

[Hai01] Eric Haines. Soft planar shadows using plateaus. *J. Graph. Tools*, 6(1):19–27, 2001.

[Har07] Mark Harris. Parallel prefix sum (scan) with cuda. nvidia. http://developer.download.nvidia.com, 2007.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, dec 2003.

[Hop98] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 35–42, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[HPAD06] K. Hegeman, S. Premoze, M. Ashikhmin, and G. Drettakis. Approximate ambient occlusion for trees. In C. Sequin and M. Olano, editors, *SIGGRAPH '06*. ACM SIGGRAPH, March 2006.

[HSO07] Mark Harris, Shubhabrata Sengupta, and John D. Owens. Parallel prefix sum (scan) with cuda. *GPU Gems 3 (2007). Chapter 39.*, 2007.

[IAN02] J. Edward Swan II, Jesus Arango, and Bala Nakshatrala. Interactive, distributed, hardware-accelerated lod-sprite terrain rendering with stable frame rates. In R. Erbacher, P. Chen, M. Gröhn, J. Roberts, and C. Wittenbrink, editors, *Visualization and Data Analysis 2002, Proceedings of SPIE Volume 4665*, pages 177–188. January 2002.

[Ion97] Mihai F Ionescu. Optimizing parallel bitonic sort. Technical report, Santa Barbara, CA, USA, 1997.

[JMLH01] H.W. Jensen, S.R. Marschner, M. Levoy, and P. Hanrahan. A practical model for subsurface light transport. In Eugene Fiume, editor, *SIGGRAPH*, pages 511–518, 2001.

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

[LBO07] T. Luft, M. Balzer, and Deussen O. Expressive illumination of foliage based on implicit surfaces. In *Natural Phenomena 2007*, September 2007.

[LCV03] J. Lluch, E. Camahort, and R. Vivo. Procedural multiresolution for plant and tree rendering. In *AFRIGRAPH '03: Proceedings of the 2nd international conference on Computer graphics,*

virtual Reality, visualisation and interaction in Africa, pages 31–38. ACM Press, 2003.

[LEST06] J. Dylan Lacewell, D. Edwards, P. Shirley, and William B. Thompson. Stochastic billboard clouds for interactive foliage rendering. *Journal of graphics tools*, 11(1):1–12, 2006.

[LM08] Andrew Lauritzen and Michael McCool. Layered variance shadow maps. In *GI '08: Proceedings of graphics interface 2008*, pages 139–146, Toronto, Ont., Canada, Canada, 2008. Canadian Information Processing Society.

[LRDM06] C. Linz, A. Reche, G. Drettakis, and M. Magnor. Effective multi-resolution rendering and texture compression for captured volumetric trees. In Eurographics, editor, *Proceedings of the Eurographics Workshop on Natural Phenomena*, 2006.

[LT00] P. Lindstrom and G. Turk. Image-driven simplification. *ACM TOG*, 3(19):204–241, 2000.

[Max96] N. Max. Hierarchical rendering of trees from precomputed multi-layer z-buffers. In Xavier Pueyo and Peter Schröder, editors, *Rendering Techniques '96, Proceedings of the Eurographics Workshop*, pages 165–174. Eurographics, Springer-Verlag, 1996.

[MB01] Ashton E. W. Mason and Edwin H. Blake. A graphical representation of the state spaces of hierarchical level-of-detail scene descriptions. *IEEE Transactions on Visualization and Computer Graphics*, 7(1):70–75, 2001.

[MDK99] N. Max, O. Deussen, and B. Keating. Hierarchical image-based rendering using texture mapping hardware. In Dani Lischinski and Gregory Ward Larson, editors, *Rendering Techniques '99, Proceedings of the Eurographics Workshop*, pages 57–62, 1999.

[MJW07] S. Mantler, S. Jeschke, and M. Wimmer. Displacement mapped billboard clouds. In *Proceedings of Symposium on Interactive 3D Graphics and Games*, 2007.

[MN98] A. Meyer and F. Neyret. Interactive volumetric textures. In George Drettakis and Nelson Max, editors, *Eurographics Rendering Workshop 1998*, pages 157–168. Springer Wein, 1998.

[MNP01] A. Meyer, F. Neyret, and P. Poulin. Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering*. Springer-Verlag, 2001.

[MSC03] A. Méndez, M. Sbert, and J. Catá. Real-time obscurances with color bleeding. In *SCCG '03*, pages 171–176, New York, USA, 2003.

[MT04] T. Martin and T.S. Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Rendering Techniques*, pages 153–160, 2004.

[PG04] M. Pharr and S. Green. Ambient occlusion. In *GPU Gems 2. nNidia.*, pages 279–292, 2004.

[PSS98] Steven Parker Peter, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical report, University of Utah, 1998.

[RB85] W. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *SIGGRAPH '85*, pages 313–322, New York, USA, 1985.

[RC04] J. F. Ramos and M. Chover. Lodstrips: Level of detail strips. In *International Conference on Computational Science*, pages 107–114, 2004.

[RCB+02] I. Remolar, M. Chover, O. Belmonte, J. Ribelles, and C. Rebollo. Geometric simplification of foliage. In I. Navazo and Ph. Slusallek, editors, *Proc. Eurographics 2002 (Short Presentations)*. Eurographics, 2002.

[RCG+09] Oscar Ripollés, Miguel Chover, Jesús Gumbau, Francisco Ramos, and Anna Puig-Centelles. Rendering continuous level-of-detail meshes by masking strips. *Graph. Models*, 71:184–195, September 2009.

[RCGQ08] Jose Francisco Ramos, Miguel Chover, Jesús Gumbau, and Ricardo Javier Quirós. Geotool: A tool to manage polygonal meshes and build multiresolution models. In *Recent Advances in Systems, Communications Computers*, 2008.

[RCRB03] I. Remolar, M. Chover, J. Ribelles, and O. Belmonte. View-dependent multiresolution model for foliage. *Journal of WSCG'03*, 11(2):370–378, 2003.

[Red94] Martin Reddy. Reducing lags in virtual reality systems using motion-sensitive level of detail. In *Proceedings of the second UK VR-SIG Conference*, 1994.

[RGC+09] Oscar Enrique Ripollés, Jesús Gumbau, Miguel Chover, Jose Francisco Ramos, and Anna Puig-Centelles. View-dependent multiresolution modeling on the gpu. In *WSCG'2009*, 2009.

[RGR+07] C. Rebollo, J. Gumbau, O. Ripollés, M. Chover, and I. Remolar. Fast rendering of leaves. In *Proceedings of the Ninth IASTED International Conference on Computer Graphics and Imaging*, CGIM '07, pages 46–53, Anaheim, CA, USA, 2007. ACTA Press.

[RL00] Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 343–352. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[RLB+02] J. Ribelles, A. López, O. Belmonte, I. Remolar, and M. Chover. Multiresolution modeling of arbitrary polygonal surfaces: a characterization. *Computers & Graphics*, 26(3):449–462, 2002.

[RMD04] A. Reche, I. Martin, and G. Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, 23(3):720–727, 2004.

[RRC+07] C. Rebollo, I. Remolar, M. Chover, J. Gumbau, and O. Ripollés. A clustering framework for real-time rendering of tree foliage. *Journal of Computers*, 2007.

[RRC+08] Oscar Ripollés, Francisco Ramos, Miguel Chover, Jesús Gumbau, and Ricardo Quirós. A tool for the creation and management of level-of-detail models for 3d applications. *W. Trans. on Comp.*, 7:1020–1029, July 2008.

[RRCG06] C. Rebollo, I. Remolar, M. Chover, and J. Gumbau. Hardware-oriented visualisation of trees. In *Lecture Notes in Computer Science vol. 4263/2006*, pages 374–383, 2006.

[RRCR04] I. Remolar, C. Rebollo, M. Chover, and J. Ribelles. Real time tree rendering. In *Lecture Notes in Computational Science 3039*, pages 173–180, 2004.

[RRCR06] C. Rebollo, I. Remolar, M. Chover, and O. Ripollés. An efficient continuous level of detail model for foliage. In *Proc. of 14-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2006)*, pages 335–342. UNION agency, 2006.

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. *SIGGRAPH Comput. Graph.*, 21(4):283–291, 1987.

[RSKU+10] Marc Ruiz, Laszlo Szirmay-Kalos, Tamas Umenhoffer, Imma Boada, Miquel Feixas, and Mateu Sbert. Volumetric ambient occlusion for volumetric models. In *Proceedings of CGI*, 2010.

[SA08] Erik Sintorn and Ulf Assarsson. Fast parallel gpu-sorting using a hybrid algorithm. *J. Parallel Distrib. Comput.*, 68(10):1381–1388, 2008.

[Sal07] Marco Salvi. Rendering filtered shadows with exponential shadow maps. *ShaderX*^{6} : *Advanced rendering techniques*, 2007.

[SD01] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In S.Gortler and C.Myszkowski, editors, *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 151–162. Springer-Verlag, 2001.

[SDD03] M. Stamminger, G. Drettakis, and C. Dachsbacher. Perspective shadow maps. In *Game Programming Gems IV*. 2003.

[SGCS09] Nicolau Sunyer, Jesús Gumbau, Miguel Chover, and Mateu Sbert. Screen space obscurances. In *CEIG 09. San Sebastian: 09-09-2009. 2009 Eurographics Association.*, 2009.

[SKS02] P.P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02*, pages 527–536, New York, USA, 2002.

[Spe10] Speedtree, interactive data visualization inc. `http://www.idvinc.com/speedtree/`, 2010.

[SS07] Michael Schwarz and Marc Stamminger. Bitmask soft shadows. *Computer Graphics Forum*, pages 515–524, 2007.

[SSBD03] C. Soler, F. Sillion, F. Blaise, and P. Dereffye. An efficient instantiation algorithm for simulating radiant energy transfer in plant models. *ACM Trans. Graph.*, pages 204–233, 2003.

[SSHS98] J. Shade, S.Gortler, L. He, and R. Szeliski. Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242. ACM Press, 1998.

[SWP10] Daniel Scherzer, Michael Wimmer, and Werner Purgathofer. A survey of real-time hard shadow mapping methods. In *State of the Art Reports Eurographics*, 2010.

[Wil78] L. Williams. Casting curved shadows on curved surfaces. In *SIGGRAPH '78*, pages 270–274, New York, USA, 1978.

[Win08] S. Winitzki. A handy approximation for the error function and its inverse. http://homepages.physik.uni-muenchen.de/∼winitzki/erf-approx.pdf. In *Proceedings of CGI*, 2008.

[Wlo95] M. Wloka. Lag in multiprocessor virtual reality. *Presence*, 4(1):50–63, 1995.

[WLWD03] Cliff Woolley, David Luebke, Benjamin Watson, and Abhinav Dayal. Interruptible rendering. In *SI3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 143–151, New York, NY, USA, 2003. ACM Press.

[WP95] J. Weber and J. Penn. Creation and rendering of realistic trees. In Robert Cook, editor, *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119–128. ACM Press, 1995.

[WS98] Michael Wimmer and Dieter Schmalstieg. Load balancing for smooth lods. Technical Report TR-186-2-98-31, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 1998. human contact: technical-report@cg.tuwien.ac.at.

[WSP04] M. Wimmer, D. Scherzer, and W. Purgathofer. Light space perspective shadow maps, 2004.

[XFr10] XFrog. Greenworks: Organic software. `http://www.greenworks.de/`, 2010.

[YPG01] Hector Yee, Sumanita Pattanaik, and Donald P. Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. In *ACM Transactions on Graphics*, pages 39–65. ACM Press, 2001.

[Zac02] Christopher Zach. Integration of geomorphing into level of detail management for realtime rendering. In *SCCG '02: Proceedings of the 18th spring conference on Computer graphics*, pages 115–122, New York, NY, USA, 2002. ACM Press.

[ZMK02] Christopher Zach, Stephan Mantler, and Konrad Karner. Time-critical rendering of discrete and continuous levels of detail. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 1–8, New York, NY, USA, 2002. ACM Press.

[ZSXL06] F. Zhang, H. Sun, L. Xu, and L.K. Lun. Parallel-split shadow maps for large-scale virtual environments. In *VRCIA '06*, pages 311–318, 2006.